THE OFF-LINE PROGRAMMING OF A PC BASED INDUSTRIAL ROBOT WITH SENSORY FEED-BACK

Volume II of II

Hugh Jonas Christian Andersen
Johannesburg, March 1996

Volume II of II

TABLE OF CONTENTS

APPENDIX 1

Paper Submitted for Publication

BASIC ASSEMBLY USING AN INDUSTRIAL ROBOT WITH SENSORY FEEDBACK

20 March 1996
Edward Fielding and Hugh Andersen
University of the Witwatersrand, Johannesburg, South Africa

1. ABSTRACT

The work described in this paper concerns the development of an off-line robot programming package incorporating sensory feedback to allow interaction with the real world. The package will ultimately be used for basic assembly operations. The package consists of three systems: an off-line programming package, a choice of suitable sensors and a PC based robot control hardware and software.

The package is demonstrated by grasping and inserting a cube into a square hole using the sensors to adjust the position of the end effector as required.

2. INTRODUCTION

The need for sensor-based automatic motion planning and control of industrial robots in an unstructured environment is extensive. For example in-factory transportation, household chores, military applications, chemical, radioactive, and other applications dangerous to humans.

Significant research is concentrated on building systems capable of generating purposeful motion in highly uncertain, complex environment, using on-line information from robot sensors. An example of such a task would be moving a mobile robot or a manipulator arm from its starting position to a goal position in a scene with unknown arbitrarily shaped obstacles. Carrying out such tasks requires, first, sensors and related hardware and software for on-line data acquisition and processing, second, certain intelligent strategies, and third, integration of sensing capabilities into the task planning function.

Task planning can be divided into two categories: on-line and off-line programming. On-line programming is the development of robot programmes using the robot to perform the actions and storing the commands and positions as teachpoints as the robot is "walked" through the task. In contrast off-line programming is the development of robot programmes without using the robot and is beneficial primarily due to reduced lost production hours.

In contrast, off-line programming will allow the programmes to be developed and stored at an earlier stage and down loaded to the robot when required so reducing the amount of down time.

3. DEVELOPMENT OF THE SYSTEM

3.1 *PC Based Robot Control Files*

The PC based robot control hardware and software has already been developed by Mr. E Illos[1].

3.2 *Choice of a Suitable Robot Simulation Package*

Of the many robot simulation packages are commercially available, Workspace Version 3, (WS3), supplied by Robot Simulations Ltd., England, was chosen as the platform for developing off-line robot programmes. WS3 allows robots and their surroundings to be modeled in the package using the package's graphical features. The package also allows static and dynamic analysis of the robot during simulations.

The fact that the package is PC based is its as this allows great flexibility which was important in this project.

3.3 *Choice of Suitable Sensors*

Vision and ultrasonic ranging are the most common sensors available for determination of an object's position in Cartesian co-ordinates. However there are fundamental problems associated with using solely machine vision or ultrasound, and therefore a combination of the two techniques was used. These problems are discussed briefly below.

3.3.1 Machine Vision

Two cameras are often used to obtain three dimensional Cartesian position via stereo disparity. This technique involves viewing a target with two cameras positioned a known distance apart. The two images are then matched to determine the same object in each image and the position of the object is determined by triangulation.

Borgfores[2] notes that the matching of two images is problematic due to possible occlusion, lighting and orientation affects, the results of which often hinder or prevent the accurate positioning of an object. These problems which would require complex analysis made the technique unattractive for our application.

3.3.2 3D Positioning Using Ultrasonic Ranging

The Cartesian position of an object can also be determined using a bank of ultrasonic sensors and neural networks as discussed by Watanabe[3]. The disadvantages of the technique, for our application, is the physical size of the bank of receivers which cannot be mounted onto a robot end effector.

3.3.3 Choice of Sensor(s)

Because of the problems discussed above, it was decided to combine the best attributes of machine vision and ultrasound. As a result the CCD was used to determine the two dimensional position of the target in the plane

perpendicular to the tool roll axis and the ultrasonic sensor to determine the distance between the target and the tool.

4. BASIC ASSEMBLY - GRASPING AND INSERTING A CUBE

The package developed was demonstrated by locating a cube, grasping it, and inserting it into a square hole. The operations to align the tool with the cube or hole are similar and the cube and hole will be referred to as the "target" henceforth.

In order to grasp or insert the cube, the tool center and orientation must both be aligned with the "target" center and orientation. These operations are explained below.

4.1 Locating the Position of the Target

The center of the target is found by examining a run length encoded (RLE) image. By using scan lines as in Figure 4.1 the edges of the cube can be found using Sobel's edge detection algorithm, which is the best edge detection algorithm according to Gouws[4]. Once the edges of the cube have been found the center of the cube and orientation in the camera coordinate reference frame can be determined.

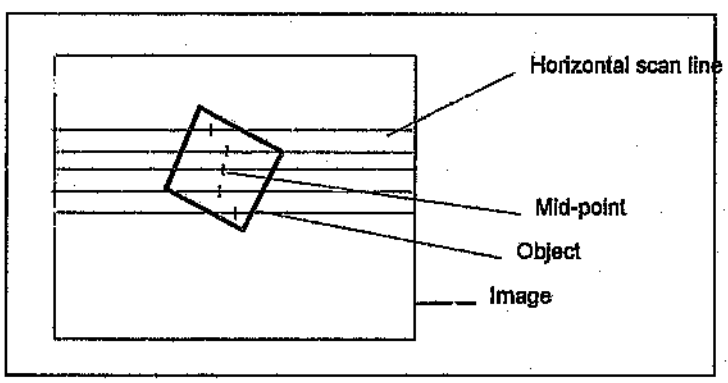


Figure 4.1 : The Use of Scan Lines

Once the camera has been calibrated the position of the object in the camera's reference frame in millimeters can be determined. A homogenous transform matrix is used to convert the position from the camera reference frame to the robot base reference frame. The ultrasonic sensor measures the distance between the camera and the target.

4.1.1 Inverse Kinematics-Converting from Tool to Joint Space

The technique of finding the joint angles in order to place the tool at the grasping vector is known as inverse kinematics. Numerous inverse kinematics algorithms exist, however the algorithm presented by Schilling[5] was used as it offers a fast, closed form solution.

The algorithm solves the inverse kinematics problem by de-coupling and solving the homogenous transform matrix between the tool and robot base reference frames. Using the Denavit-Hartenburg method of representing a robot, this matrix is shown below where each term is a function of the joint angles vector, $\underline{Q}$:

$\underline{Q} = \{q_1, q_2, q_3, q_4, q_5\}$.

$$T_{base}^{tool}(q) = \begin{bmatrix} c_1c_{234}c_5 + s_1s_5 & -c_1c_{234}s_5 + s_1c_5 & -c_1s_{234} & c_1\left(a_2c_2 + a_3c_{23} + a_4c_{234} - d_5s_{234}\right) \\ s_1c_{234}c_5 - c_1s_5 & -s_1c_{234}s_5 - c_1c_5 & -s_1s_{234} & s_1\left(a_2c_2 + a_3c_{23} + a_4c_{234} - d_5s_{234}\right) \\ -s_{234}c_5 & s_{234}s_5 & -c_{234} & d_1 - a_2s_2 - a_3s_{23} - a_4s_{234} - d_5c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

c_1	$= \cos(q_1)$	s_1	$= \sin(q_1)$
c_2	$= \cos(q_2)$	s_2	$= \sin(q_2)$
c_3	$= \cos(q_3)$	s_3	$= \sin(q_3)$
c_5	$= \cos(q_5)$	s_5	$= \sin(q_5)$
c_{23}	$= \cos(q_2 + q_3)$	s_{23}	$= \sin(q_2 + q_3)$
c_{234}	$= \cos(q_2 + q_3 + q_4)$	s_{234}	$= \sin(q_2 + q_3 + q_4)$

The input to the algorithm is the tool configuration vector, $\underline{W}(q)$, which consists of the Cartesian coordinates of the tool in the robot base frame and the tool roll, pitch and yaw angles. Where:

$\underline{W}(q) = (X, Y, Z, Yaw, Pitch, Roll)_B$

which, when using the transform matrix, equates to

$$W(q) = \begin{bmatrix} c1(a2c2 + a3c23 + a4c234 - d5s234) \\ s1(a2c2 + a3c23 + a4c234 - d5s234) \\ d1 - a2s2 - a3s23 - a4s234 - d5c234 \\ -[\exp(q_5/\pi)]c1s234 \\ -[\exp(q_5/\pi)]c1s234 \\ -[\exp(q_5/\pi)]c234 \end{bmatrix}$$

Because the approach vector specifies the tool orientation and not the tool angle, a method to couple the tool roll angle with the approach vector is required. The tool roll angle could simply be added as another input parameter into the inverse kinematics algorithm, but it is possible to couple the roll angle with the approach vector which results in fewer input parameters. The approach vector is a unit vector specifying the direction only and can therefore be scaled by a positive quantity without changing the specified direction.

To recover the tool roll angle from a scaled approach vector an invertable function of the roll angle q_5 is used to scale the length of r^3. The scaling function used is:

$$f(q_5) \equiv \exp\left(\frac{q_5}{\pi}\right)$$

Other scaling functions can also be used but the function is sufficient because $f(q_5)>0$ for all q_5 and $f^{-1}(q_n)$ is well defined. By scaling the approach vector by $f(q_5)$ only three values need to be input into the inverse kinematics algorithm instead of four which makes for a simpler algorithm.

Solution To The Inverse Kinematics Problem By Decoupling The Arm Equation

Since a five degree of freedom (dof) robot does not have yaw motion, q_1, the robot base rotation, can be found by dividing w_2 by w_1 which leaves s_1/c_1 as the factor before each cancels out. Taking the inverse tan of the dividend will result in q_1 being found. i.e.

$$q_1 = a\tan 2(w_2, w_1)$$

atan2 is used as this function is defined over the entire range $[-\pi,\pi]$.

The elbow angle q_3 is the most difficult joint variable to extract as it is strongly coupled with the shoulder and tool pitch angles. Firstly an intermediate variable q_{234} the global pitch angle is isolated. Where:

$$q_{234} = q_2 + q_3 + q_4$$

is the tool pitch angle measured relative to the work surface $x_o y_o$ plane. Inspection of the last three components of W(q) reveals that

$$\frac{-(c_1 w_4 + s_1 w_5)}{-w_6} = \frac{s_{234}}{c_{234}}$$

since the base angle is already known the global tool pitch angle can be computed using:

$$q_{234} = a\tan 2\left[-(c_1 w_4 + s_1 w_5), -w_6\right]$$

Once the shoulder angle q_2 and the elbow angle q_3 are known the tool pitch angle q_4 can be computed from the global pitch angle q_{234}. To isolate the shoulder and elbow angles two more intermediate variables are defined:

$$b_1 = c_1 w_1 + s_1 w_2 - a_4 c_{234} + d_5 s_{234}$$
$$b_2 = d_1 - a_4 s_{234} - d_5 c_{234} - w_3$$

where b_1 and b_2 are both known because q_1 and q_{234} have been determined. Taking the expressions and substituting them into the above equations yields:

$$b_1 = a_2 c_2 + a_3 c_{23}$$
$$b_2 = a_2 s_2 + a_3 s_{23}$$

These two equations involving the shoulder and elbow angles, are independent of one another and the coupling with the tool pitch angle has been removed. The elbow angle can be found by computing $\|b\|^2$ and after using trigonometric identities and simplification we have:

$$\|b\|^2 = a_2^2 + 2a_2 a_3 c_3 + a_3^2$$

thus the cosine of the elbow angle has been determined. Solving this equation results in two possible solutions, elbow up and elbow down.

$$q_3 = \pm a\cos\left[\frac{\|b\|^2 - a_2^2 - a_3^2}{2a_2 a_3}\right]$$

Thus the solution for the inverse kinematics for a 5 dof robot is not unique although it is possible to decide whether elbow up or elbow down is required.

The shoulder joint, q_2 , can be isolated by expanding c_{23} and s_{23} in b_1 and b_2. After rearranging we get:

$$b_1 = (a_2 + a_3 c_3)c_2 - (a_3 s_3)s_2$$
$$b_2 = (a_2 + a_3 c_3)s_2 + (a_3 s_3)c_2$$

Since q_3 is known the above two equations constitute a system of two simultaneous equations in the unknowns c_2 and s_2. After using row operations to solve the simulations equations we get:

$$c_2 = \frac{(a_2 + a_3 c_3)b_1 + a_3 s_3 b_2}{\|b\|^2}$$

$$s_2 = \frac{(a_2 + a_3 c_3)b_2 + a_3 s_3 b_1}{\|b\|^2}$$

The above two equations are expressions for q_2 in sine and cosine from which q_2 can be can be recovered over the whole range $[\pi, -\pi]$ using the *atan2* function.

$$q_2 = a\tan 2\left[(a_2 + a_3c_3)b_2 - a_3s_3b_1\ ,\ (a_2 + a_3c_3)b_1 + a_3s_3b_2\right]$$

The tool pitch angle can be now found since the both the shoulder and elbow angles are known. Recall that

$$q_{234} = q_2 + q_3 + q_4$$

therefore

$$q_4 = q_{234} - q_2 - q_3$$

The tool roll angle, q_5, can be determined in two ways. The first way is to use the last three terms of the tool vector (recall that these terms are scaled by the roll angle)

$$q_5 = \pi\ \ln\left(w_4^2 + w_5^2 + w_6^2\right)^{1/2}$$

Alternatively the roll angle can be found by examining the tool rotation matrix $\underline{R}$, which is derived from the tool to base transform matrix and is a function of the robot joint angles, shown below.

$$\underline{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} = \begin{bmatrix} c_1c_{234}c_5 + s_1s_5 & -c_1c_{234}s_5 & -c_1s_{234} \\ s_1c_{234}c_5 - c_1s_5 & -s_1c_{234}s_5 - c_1c_5 & -s_1s_{234} \\ -s_{234}c_5 & s_{234}s_5 & -c_{234} \end{bmatrix}$$

By multiplying the first component of the normal vector, R_{11}, by s_1 and subtracting c_1 times the second component the result is:

$$s_5 = s_1R_{11} - c_1R_{21}$$

This will allow the roll angle to be computed over the range $\left[-\pi/2\ ,\ \pi/2\right]$ however to determine the roll angle over the entire range $\left[\pi\ ,-\pi\ \right]$ c_5 must also be determined. By multiplying the first component of the sliding vector by s_1 and subtracting c_1 times the second component the result is:

$$c_5 = s_1R_{12} - c_1R_{22}$$

Given the tool roll angle as a function of cosine and sine q_5 can be calculated over the entire range using the *atan2* function as below.

$$q_5 = a\tan 2\left[s_1R_{11} - c_1R_{21}\ ,\ s_1R_{12} - c_1R_{22}\right]$$

However R_{11} ,R_{21} , R_{12} and R_{22} , the tool rotation matrix, still need to be found before q_5 can be evaluated. Since we know what the tool yaw, pitch and roll angles must be, $\underline{R}$ can be determined using the composite rotation matrix YPR(ϕ).

YPR(ϕ), is a composite rotation matrix formed by rotating a mobile coordinate frame about a fixed coordinate frame a yaw angle,(ϕ_1), about the Z axis, a pitch angle, ,(ϕ_2), about the Y axis and a roll angle, ,(ϕ_3), about the X axis. The resulting matrix is:

$$YPR(\phi) = \begin{bmatrix} c_2c_3 & s_1s_2c_3 - c_1s_3 & c_1s_2c_3 + s_1s_3 \\ c_2s_3 & s_1s_2s_3 + c_1c_3 & c_1s_2s_3 - s_1c_3 \\ -s_2 & s_1c_2 & c_1c_2 \end{bmatrix}$$

Hence by inserting the tool yaw, pitch and roll angles into YPR(ϕ) $\underline{R}$ can be found.

Hence the joint angles in order to place the tool in the required position and orientation h ve been determined. Figure 3.2 illustrates this algorithm graphically.

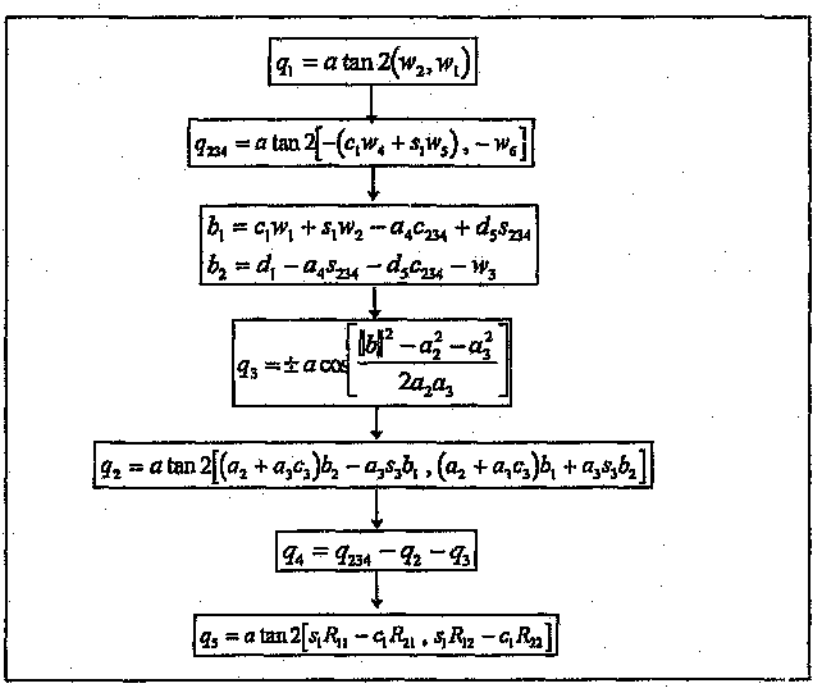


Figure 4.2: Illustration of the Inverse Kinematics Algorithm

4.2 *Determining the Orientation of the Target*

Once the edges and the center of area of the cube have been found the information can be exploited to find the orientation of the cube with respect to the camera axes.

The distance from each edge to the center of area can be found and stored in an array using Pythagoras' theorem as shown in Figure 4.3. This array is then sorted to find the maximum four distances which correspond to the four corners of the cube. The angle between adjacent corners can now be found using simple trigonometry and averaged to find the orientation of the cube with respect to the camera reference frame.

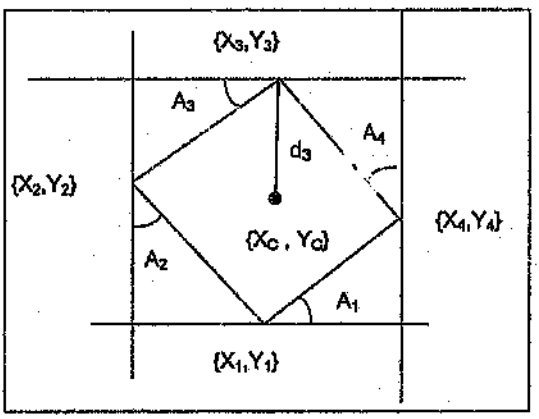


Figure 4.3: Calculating the Target Orientation

5. EXPERIMENTAL RESULTS

WS3 was used successfully to model the robot and the real world surrounding it. The model was then successfully used to simulate the grasping of a cube and the insertion of it into a square hole.

The camera and ultrasonic sensors to allow the robot to interact with the real world were accurate and took on average 6 seconds to determine the position and orientation of the target. They were able to position the tool above the center of the cube or hole to within a radius of 2mm and the alignment to within 2 degrees. This allowed a 50x50x40 mm cube to be inserted into a 55x55x50 mm hole.

The robot control files are able to control the motion of all joints of the robot to a tolerance within 30 seconds 90% of the time.

6. CONCLUSION

The package successfully completed the basic assembly task although there are certain problems associated with the technique.

WS3 proved itself as an excellent PC based robot simulation package for the development of off-line programmes. However the package has weak features for interaction with other software and DOS which prevent it from being used as the platform to control the robot and sensors. Work is being done by Chen[6] on a package to incorporate these features.

The choice of sensors in this project was a good one as they were fast and produced good results. However the sensors did have two problems associated with them. Firstly the ultrasonic sensor had a range limitation of 150mm which could pose a problem for some assembly tasks. Secondly the camera was inclined at an angle to the tool roll axis to prevent the robot from being "blinded" once the cube had been grasped. This causes fore shortening of the image and although it was not a limitation for this demonstration it could cause a problem in later assembly tasks.

7. REFERENCES

1. Illos E.
"Control Of A Robot Arm"
Fourth year project
University of the Witwatersrand, 1991

2. Borgefors G.
"Heirarchical Chamfer Matching : A Parametric Edge Matching Algorithm"
IEEE Transactions on Pattern and Machine Intelligence.
vol 10, no. 6, Nov. 1988, pp.849-864.

3. Watanabe S and Yoneyama M.
An Ultrasonic Visual Sensor For Three-Dimensional Object Recognition Using Neural Networks.
IEEE Transactions on Robotics and Automation.
vol.8, no.2 Apr 1992, pp.240-249

4. Gouws J.
"The Development Of A Machine Vision Based Milking Robot."
PhD thesis, Wageningen Agricultural University, The Netherlands 1993

5. Schilling RJ.
"Fundamentals of Robotics - Analysis and Control"
Prentice Hall Int.ed.1990

6. Chen C et al,
"Simulation And Animation Of Sensor-Driven Robots"
IEEE Transactions on Robotics and Automation,
vol.10, no.5 Oct 1994, pp.684-688

7. Hor M and Huang Y.
"Grasping A Polyhedron Using An Eye-In-Hand Robot"
The International Conference on Intelligent Manufacturing 1995

APPENDIX 2

Listing of The Source Code

A complete listing of the source code follows. The software was written in Microsoft C/C++ Version 7. The software is written as a project which is statically linked at the time of compiling the project.

The following files make up the project :

Source code file	Main procedures
align.c	• aligns the tool with the target.
camera.c	• controls the camera and • finds the edges, centre of area, corners.
consts.c	• contains all the robot constants such as link length, limits etc.
htms.c	• contains the homogeneous transform matrices.
init_rbt.c	• initializes the Lm 628 chip, camera and ultrasonic sensor.
inv_kin.c	• calculates the joint space given the cartesian space.
lm_cmnd.c	• controls the motion of the joints.
master.c	• the "master" file which starts the package and calls the procedures.
movehome.c	• moves the robot to the home position.
ultrasnd.c	• initializes the a to d card and • obtains the ultrasonic sensor's voltage and • converts the ultrasonic sensor's voltage into a measurement.
utils.c	• menus and • saving the current joint space.
w_offlin.c	• translates the WS3 track file into a file which lm_cmnd.c can use to control the robot.
ws3toc.c	• converts the WS3 track file from WS3 format into a format that the source code can understand.

ALIGN.C

```
/*
   ALIGN.C  aligns the robot with the target based on the offset distances
   and angle obtained from  CAMERA.C functions
*/

#include <math.h>
#include"c:\masters\proj\ext_vars.h"
#include"c:\masters\proj\fnproto.h"

float DistThresh=2;

/* ****************
AlignTool(int Object)

USAGE
-- aligns the tool in the Z plane, compensating for the rotation of axes
1 and 5

INPUTS
whether we are looking for the cube or hole centre of area

OUTPUTS
the X and Y differences between the tool centre and cube/hole centre in
robot base coords which is then passed on to inverse kinematics

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

AlignTool(int Object)
{
float c1,c2,c3,c23,c234,c5,Cb;
float s1,s2,s3,s23,s234,s5,Sb;
float Cy,Cp,Cr,Sy,Sp,Sr;
float R11, R12, R13, R14, R21, R22, R23, R24;
float R31, R32, R33, R34, R41, R42, R43, R44;
float T11, T12, T13, T14, T21, T22, T23, T24;
float T31, T32, T33, T34, T41, T42, T43, T44;
float Temp[4][4];
int OffsetThresh=2, i, Flag;
//all dimensions in mm !!!
float Xcam, Ycam, Zcam, Xtool, Ytool, Ztool, Xbase, Ybase, Zbase;
float ToolTol=2, AngTol=2;        //millimeters and degrees
float  PixPermmX, PixPermmY, Dist2, IntX, IntY;
float Pos[4], Rot[4][4], IntHt,temp[8], D;
extern int Objdepth;
extern float Xo, Yo, Zo, Beta, Zus;

SetCursor(ACTIVITY,2);
printf("Aligning the tool with the target");

Xbase=Ybase=ToolTol + 10;
```

```
OffsetAngle=AngTol+10;

while((fabs(Xbase)>ToolTol)||(fabs(Ybase)>ToolTol)||(fabs(OffsetAngle)>AngTol))
{
InputHandler(TARGET);
UltraSndSensor();

if(Dist>800 || Dist<5)
 {Errors(1);  break; }

  //Zcam == dist from camera to cube/hole
if(Object==1)                    //align with cube
  D = (Dist - Zus) + Zo - ObjDepth;
if(Object==0)                    //align with hole
  D = (Dist - Zus) + Zo ;

Zcam = (float)D/sin((90-fabs(Beta))/R_TO_D);

PixPermmX = ax*pow(Zcam,4) + bx*pow(Zcam,3) + cx*pow(Zcam,2)
          + dx*Zcam + ex;
PixPermmY = ay*pow(Zcam,4) + by*pow(Zcam,3) + cy*pow(Zcam,2)
          + dy*Zcam + ey;

//convert from pix to mm in camera coords
Xcam = OffsetX/PixPermmX;
Ycam = OffsetY/PixPermmY;

//HTM from camera to tool coords
Cb =   cos(Beta/R_TO_D);
Sb =   sin(Beta/R_TO_D);

T11 = 1;        T12 = 0;        T13 = 0;        T14 = Xo;
T21 = 0;        T22 = Cb;       T23 = -Sb;      T24 = Yo;
T31 = 0;        T32 = Sb;       T33 = Cb;       T34 = Zo;
T41 = 0;        T42 = 0;        T43=0;          T44=1;

Xtool = T11*Xcam + T12*Ycam + T13*Zcam + Xo;
Ytool = T21*Xcam + T22*Ycam + T23*Zcam + Yo;
Ztool = Dist - 50;               //can also use above HTM but using US
                                 //directly will give better results

//HTM from tool to robot base coords

for(i=1;i<=5;i++)
 temp[i]= AbsAng[i];

c1     = cos(temp[1] / R_TO_D);
c5     = cos(temp[5] / R_TO_D);
c23    = cos((temp[2]+temp[3])/R_TO_D);
c234 = cos((temp[2]+temp[3]+temp[4])/R_TO_D);
s1     = sin(temp[1] / R_TO_D);
```

```
s5      = sin(temp[5] / R_TO_D);
s23     = sin((temp[2]+temp[3])/R_TO_D);
s234 = sin((temp[2]+temp[3]+temp[4])/R_TO_D);

R11=c1*c234*c5+s1*s5; R12=-c1*c234*s5+s1*c5;  R13=-c1*s234;
R14=c1*(a[2]*c2+a[3]*c23-d[5]*s234);
R21=s1*c234*c5-c1*s5; R22=-s1*c234*s5-c1*c5;  R23=-s1*s234;
R24=s1*(a[2]*c2+a[3]*c23-d[5]*s234);
R31=-s234*c5;  R32=s234*s5;             R33=-c234;
R34=d[1]-a[2]*s2-a[3]*s23-d[5]*c234;
R41=0;          R42=0;                  R43=0;
R44=1;


Xbase=R11*Xtool + R12*Ytool + R13*Ztool;  /* + R       ,*/
Ybase=R21*Xtool + R22*Ytool + R23*Ztool;  /* + R24*1;*/
Zbase=R31*Xtool + R32*Ytool + R33*Ztool;  /* + R34*1;*/

if((fabs(Xbase)>ToolTol)||(fabs(Ybase)>ToolTol)||(fabs(OffsetAngle)>AngTol) )
 {
 Pos[1]=AbsPos[0] + Xbase;
 Pos[2]=AbsPos[1] + Ybase;
 Pos[3]=AbsPos[2];
 Yaw = 180;
 Pitch = 0;
 Roll -= OffsetAngle;

 Cy=cos(Yaw/R_TO_D);        Cp=cos(Pitch/R_TO_D);        Cr=cos(Roll/R_TO_D);
 Sy=sin(Yaw/R_TO_D);Sp=sin(Pitch/R_TO_D); Sr=sin(Roll/R_TO_D);

 Temp[1][1]=Cp*Cr;    Temp[1][2]=Sy*Sp*Cr - Cy*Sr;  Temp[1][3]=Cy*Sp*Cr + Sy*Sr;
 Temp[2][1]=Cp*Sr;    Temp[2][2]=Sy*Sp*Sr + Cy*Cr;  Temp[2][3]=Cy*Sp*Sr - Sy*Cr;
 Temp[3][1]=-Sp;      Temp[3][2]=Sy*Cp;             Temp[3][3]=Cy*Cp;

 InvKinematics(Pos,Temp,ZERO4);
 Update();
 }
}

SetCursor(ACTIVITY,2);
printf("                              ");

}

/* ********************************
IntersectPos(int Object2)

USAGE
positions the tool a certain height above the cube/hole so that the
cube/hole is in the CCD's field of view
```

```
INPUTS
whether we are at the cube or hole

OUTPUTS
the difference between the present height and the height the CCD ought to
be above the cube/hole passed on to inverse kinematics

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

IntersectPos(int Object2)
{
float P[7], D1, IntHt;
extern int Objdepth;

SetCursor(ACTIVITY,2);
printf("Moving tool along Z axis");

/*  move the tool down until the camera c_line & tool c_line
  are on top of the target  */

if(Object2==1)                                  //align with cube
 IntHt=IntersectHeight;
if(Object2==0)                                  //align with hole
 IntHt=IntersectHeight-ObjDepth;

UltraSndSensor();

while(fabs(Dist-IntHt)>DistThresh )
  {
  UltraSndSensor();
  If( Dist>800 || Dist<5)
        { Errors(1); break; }
  else
        {
        D1=Dist-IntHt;
        MoveZAxis(-D1);
        }
  Update();
 }
}


/* ***************************
void GraspObject(void)

USAGE
moves the tool down the robot Z axis until the gripper fingers will
close on the sides of the cube

INPUTS
the present height of the tool above the cube
```

```
OUTPUTS
the amount the tool should move down the robot Z axis which is
then passed on to inverse kinematics

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

void GraspObject(void)
{
float PickHeight;
float P[7], D1, D2, FstPt, SecPt;
int DistThresh=3, Flag=FALSE, Flag2=TRUE, count=0;
extern float Zus;

SetCursor(ACTIVITY,2);
printf("Moving Down to Grasp the Target");

FstPt = ObjDepth + fabs(Zus) + 60;      //FstPt is the first pick point
SecPt = ObjDepth/2 + fabs(Zus) - 5;     //SecPt is the second pick point


/*  move the tool down until the camera c_line & tool c_line
  are on top of the target  */

while(fabs(Dist-FstPt) > DistThresh)
 {
 UltraSndSensor();
 if( Dist>800 || Dist<5)
        { Errors(1); break;  }
 else
        {
        D1=Dist-FstPt;          //these dists are the US dists above the bench
        MoveZAxis(-D1);
        Update();
        }
}

while(fabs(Dist-SecPt) > DistThresh && count<=1)
 {
 UltraSndSensor();
 if( Dist>800 || Dist<5)
        { Errors(1); break; }
 else
        {
         count++;
        D1=Dist-SecPt;
        MoveZAxis(-D1);
        Update();
        }
}
```

```
SetCursor(ACTIVITY,2);
printf("                              ");

}

/* ****************************
void InsertObject(void)

USAGE
moves the tool down the robot Z axis until the gripper fingers will
insert the cube into the hole

INPUTS
the present height of the tool above the cube

OUTPUTS
the amount the tool should move down the robot Z axis which is
then passed on to inverse kinematics

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

void InsertObject(void)
{
float InsertHeight;
float P[7], D1, D2, FstPt, SecPt;
int DistThresh=3, Flag=FALSE, Flag2=TRUE, count=0;
extern float Zus;


SetCursor(ACTIVITY,2);
printf("Moving Down to Insert the Object");

FstPt = ObjDepth + fabs(Zus) + 60;      //FstPt is the first pick point
SecPt = ObjDepth + fabs(Zus) - 30;      //SecPt is the second pick point

SetCursor(ACTIVITY,2);
printf("Moving tool along Z axis");

/*move the tool down until the camera c_line & tool c_line
are on top of the target */

while(fabs(Dist-FstPt) > DistThresh)
      {
      UltraSndSensor();
      if( Dist>800 || Dist<5)
        { Errors(1); break; }
      else
        {
        D1=Dist-FstPt;          //these dists are the US dists above the bench
        MoveZAxis(-D1);
```

```
        Update();
        }
      }

while(fabs(Dist-SecPt) > DistThresh && count<=1)
      {
      UltraSndSensor();
      If( Dist>800 || Dist<5)
        { Errors(1); break; }

        count++;
        D1=Dist-SecPt;
        MoveZAxis(-D1);
        Update();
      }

OpenHand();

SetCursor(ACTIVITY,2);
printf("                              ");

}
```

CAMERA.C

```
/*-------------------------------------------------------------

  TITLE   :     CAMERA.C              27 Apr 1995

  DESCRIPTION:
    A live video image is displayed on the PC monitor
    and can be calibrated to measure the width of an object.
    Once the programme has been calibrated it can be used to find the
    dimensions and orientation of an object.

 -------------------------------------------------------------*/

#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include "c:\masters\proj\ext_vars.h"
 #include"c:\masters\proj\fnproto.h"

typedef unsigned char  BYTE ;
typedef unsigned int   WORD ;
typedef int            BOOL ;


#define VIDEO_INT  0x10

#define DRIVER_SIG     "VBLAST"
#define SIG_OFFSET     0x103
#define VB_GETVERSION  0x00
#define VB_INITIALIZE 0x01
#define VB_TERMINATE        0x02
#define VB_GETIOPORT        0x03
#define VB_GETVIDADDX       0x04
#define VB_GETINTRNUM       0x06
#define VB_GETFREEZE        0x0A
#define VB_GETDISPMODE      0x12
#define VB_READFRMBUF  0x1C
#define VB_GETPHYCOOR  0x28
#define VB_SETLOGICEXT 0x105
#define VB_SETFITMODE  0x108
#define VB_SETSCALE    0x10F
#define VB_SETFREEZE   0x10A
#define VB_SETDISPMODE     0x112
#define VB_SETPOS      0x115
#define VB_SETSIZE     0x116
#define VB_SETCOLKEY        0x117
#define VB_WRITEFRMBUF 0x11C
```

```
#define WIN_X    0
#define WIN_Y    2
#define WIN_WIDTH 40
#define WIN_HEIGHT 14

#define VID_XORG   5
#define VID_YORG   100
#define VID_HEIGHT 500
#define VID_WIDTH  460

#define WHITE        0x70
#define BLANK                0x00

#define INCREASE     1
#define DECREASE     0

#define READ         0
#define WRITE        1

#define GREY_LEVELS          256
#define CRET             0x0D
#define ESCAPE               0x1B
#define SPACE            0x20
#define UP          0x48
#define DOWN     0x50


unsigned       far *vidPtr = (unsigned far *)0xB8000000L;
int            wVBIntrNum;
union REGS     regs;
struct SREGS   segregs;
typedef  struct tagWINREC{
            int wX ;
            int wY ;
            int wW ;
            int wH ;
            } WINREC ;
typedef WINREC far* LPWINREC ;
WINREC  winrec ;

unsigned int  wXpos , wYpos , wWidth , wHeight ;
char     huge* ptrbuffer   ;
char     huge* ptr         ;
long   dwMemorySize    ;
int CameraConst;

unsigned long WIDTH , HEIGHT;
int LeftX[75], RightX[75], Y[75], CntrHor=0;
int TopY[75], BtmY[75], X[75], CntrVer=0;
int LimX, LimY;
```

```
int CentroidX, CentroidY;
int VideoScale;
float XScale, YScale;
int Xdist[75], Ydist[75];
int  *Index;

/* ***************
 int FindVbistdrv(char *szDriverSig)
 USAGE
  Checks whether FS200DRV is installed and find the interrupt vector
  to call FS200DRV API functions.
INPUTS
  szDriverSig - string containing the driver signature
 OUTPUTS
  Interrupt Vector Number
  0 if FS200DRV not installed

**************** */
int FindVbistdrv(char *szDriverSig)
{
        void    far *lpIntrVect;
        int     i, found = 0, wSigLen = strlen(szDriverSig);

   /* check from interrupt vector 80h to BFh */
   for (i = 0x80; i < 0xC0 && !found; i++)
   {
    /* get pointer to driver */
    lpIntrVect = (void far *)_dos_getvect(i);

    /* set signature offset */
    FP_OFF(lpIntrVect) = SIG_OFFSET;

    /* check for driver signature */
    if (!_fstrncmp(lpIntrVect, (char far *)szDriverSig, wSigLen))
     {
          found = i;
          break;
     }
   }

   return(found);
}

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

  int CallVbistdrv(REGS *regs, SREGS *segregs)

  USAGE
   Calls FS200DRV with registers set with appropriate value for the API
   function to be performed.
```

```
  INPUTS
    regs    * registers set with appropriate value for the API call
    segregs - segment registers set with appropriate value for the API call

  OUTPUTS
    TRUE    - API call successful
    FALSE   - API call failed
    regs    - registers contain the return values
    segregs - segment registers contain the return values

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

int CallVblstdrv(union REGS *regs, struct SREGS *segregs)
{
        static char *szErrMessage[] = {
        "Video Blaster FS200 driver is busy",
        "VIDEOBLST_FS200 environment variable is not present or "
                        "incorrectly set",
        "Video Blaster FS200 is not found at the specified IO "
                        "address",
                        "Interrupt not detected using the specified interrupt "
                        "number",
                        "Function specified is invalid",
                        "Parameter passed to the function is out of range",
                        "Function is invalid in a particular Fit-window mode",
        "Unable to open CTFS200.CFG file",
        "Not enough data in CTFS200.CFG file",
        "CTFS200.CFG file is not a valid Video Blaster FS200 "
        "configuration file" ,
        "FS200DRV driver has not been initialized. "
                };
        int         fRetVal = TRUE;

  int86x(wVBIntrNum, regs, regs, segregs);

  /* check for error */
  if (regs->x.cflag)
  {
  printf("\nERROR: %s\n", szErrMessage[regs->x.ax-1]);
  fRetVal = FALSE;
  }

  return (fRetVal);
}

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

  void DisableDriver(void)

  USAGE
    Disables video driver and exit.
```

```
  INPUTS        OUTPUTS
    None          None

 ***************************** */
void DisableDriver(void)
{
   /* Terminate driver */
   rogs.x.bx = VB_TERMINATE;
   CallVblstdrv(&regs, &segregs);
   SetCursor(ACTIVITY,2);
   printf("Driver terminated");

}

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

  void DrawWindow(int win_x, int win_y, int win_width, int win_height,
                  int color )

  USAGE
    Function to draw window

  INPUTS                                              OUTPUTS
    win_x    : x coordinate of window                 None
    win_y    : y coordinate of window
    win_width : window's width
    win_height: window's height
    color    : color that use to fill up the window

 ***************************** */
void DrawWindow(int win_x ,int win_y ,int win_width ,int win_height ,int color )
{
  int     x , y ;
  int     tmpx , tmpy ;
  unsigned long videodata ;

  tmpy = win_y+win_height ;
  tmpx = win_x+win_width  ;

  videodata = color ;
  videodata <<=8 ;
  videodata |=0x20 ;  // a blank character

   /* Draw window */
   for (y = win_y; y < tmpy ; y++)
    for (x = win_x; x < tmpx ; x++)
      *(vidPtr + y * 80 + x) = videodata;

}
```

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

  void SetupVideoWindow(void)

  USAGE
    Setup initial video window

  INPUTS          OUTPUTS
   None             None

 ****************************** */
void SetupVideoWindow(void)
{

   /* Set Logical Window Extent to 1000,1000       */
   /* Note : This is also the default value if you */
   /*          do not set it explicitly            */

   regs.x.bx = VB_SETLOGICEXT ;
   regs.x.ax = 1000 ;
   regs.x.dx = 1000 ;
   if (!CallVbistdrv(&regs, &segregs))
     DisableDriver();


   /* Set color key */
   regs.x.bx = VB_SETCOLKEY;
   regs.x.ax = 7;      // colorkey == white
   if (!CallVbistdrv(&regs, &segregs))
     DisableDriver();

   /* Set video position */
   regs.x.bx = VB_SETPOS;
   regs.x.ax = VID_XORG;
   regs.x.dx = VID_YORG;
   if (!CallVbistdrv(&regs, &segregs))
   DisableDriver();

   /* Set video width and height */
    regs.x.bx = VB_SETSIZE;
    regs.x.ax = VID_WIDTH;
    regs.x.dx = VID_HEIGHT;
    if (!CallVbistdrv(&regs, &segregs))
    DisableDriver();

   /* Set fit mode to 4 */
   regs.x.bx = VB_SETFITMODE ;
   regs.x.ax = 4 ;
   if (!CallVbistdrv(&regs, &segregs))
      DisableDriver();
```

```
    /* Set video display on */
    regs.x.bx = VB_SETDISPMODE;
    regs.x.ax = 1;
    if (!CallVbistdrv(&regs, &segregs))
          DisableDriver();

    /* Translate the video logical coordinate to physical coordinate */
    regs.x.bx = VB_GETPHYCOOR ;
    regs.x.ax = VID_XORG ;
    regs.x.dx = VID_YORG ;
    if (!CallVbistdrv(&regs, &segregs))
       DisableDriver();
    wXpos = regs.x.ax ;
    wYpos = regs.x.dx ;

    regs.x.bx = VB_GETPHYCOOR ;
    regs.x.ax = VID_WIDTH  ;
    regs.x.dx = VID_HEIGHT ;
    if (!CallVbistdrv(&regs, &segregs))
       DisableDriver();
    wWidth  = regs.x.ax ;
    wHeight = regs.x.dx ;

    /* round width to 4 pixels multiple */
    if(wWidth & 3)
       wWidth=(wWidth &~3) +4 ;

    winrec.wX    = wXpos ;
    winrec.wY    = wYpos ;
    winrec.wW    = wWidth ;
    winrec.wH    = wHeight ;
}

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

  void AllocateMemory(void)

  USAGE
    Allocate memory to store image

  INPUTS              OUTPUTS
    None                 None

  **************************** */
void AllocateMemory(void)
{
   dwMemorySize = wWidth*3 ;
   dwMemorySize *=wHeight  ;

   WIDTH=3*wWidth; HEIGHT=dwMemorySize/WIDTH;
   VideoScale = WIDTH/HEIGHT;
```

```
    if( (ptr = (char huge *) halloc(dwMemorySize, sizeof(char)))==NULL)
      {
      DisableDriver() ;
      printf("Memory Allocation Failed!") ;

      }
}

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

  void InputHandler(void)

  USAGE
    Handles user input from keyboard.

  INPUTS                OUTPUTS
    None                  None

  ****************************** */
void InputHandler(int key)
{

 /* get a key input */
switch (key)
   {
   case 1       :   /* Toggle colorkey on/off */
          regs.x.bx = VB_GETFREEZE;
             if (!CallVbistdrv(&regs, &segregs))
                                  DisableDriver();
             regs.x.ax ^= 1;
             regs.x.bx = VB_SETFREEZE;
             if (!CallVbistdrv(&regs, &segregs))
                  DisableDriver();
             break;

   case 2       :  /*find corners */
             Freeze();
             FindEdges();
             Centroid();
             MaxDist();
             UnFreeze();
             FreeMem();
             break;

   case 3       :  /*markedges */
             Freeze();
             FindEdges();
             PrintEdges('h');
             PrintEdges('v');
             UnFreeze();
```

```
            break;

    case 4          :   /* Toggle video on/off */
            regs.x.bx = VB_GETDISPMODE;
            if (!CallVbIstdrv(&regs, &segregs))
                    DisableDriver();

            regs.x.ax ^= 1;
            regs.x.bx = VB_SETDISPMODE;
            if (!CallVbIstdrv(&regs, &segregs))
                    DisableDriver();
               break;

    case 5          :  /*find target centroid */
               Freeze();
               FindEdges();
               Centroid();
               UnFreeze();
               break;
    }
}

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

   void ReadORWriteBuffer(int action)

   USAGE
     Perform Read/Write from/to framebuffer

   INPUTS                                          OUTPUTS
     action   : 0 to read ; 1 for write.          None

  ****************************** */
void ReadORWriteBuffer(BOOL bAction)
{
 WINREC far* lpWinRec = (WINREC far*) &winrec   ;

 regs.x.ax  = 1                  ; // RGB24
 segregs.es = FP_SEG((char far*) ptr) ;
 regs.x.di  = FP_OFF((char far*) ptr) ;

 segregs.ds = FP_SEG(lpWinRec) ;
 regs.x.si  = FP_OFF(lpWinRec) ;

  if (bAction == READ)
    regs.x.bx  = VB_READFRMBUF  ;
  else if (bAction == WRITE )
    regs.x.bx  = VB_WRITEFRMBUF  ;
  if (!CallVbIstdrv(&regs, &segregs))
   DisableDriver();
```

```
}

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

  void SetCursor(int row, int column)

  USAGE
   Set cursor position

  INPUTS                              OUTPUTS
   row   : x coordinate       None
   column : y coordinate

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

void SetCursor(int row, int column)
{
   /* Set cursor position*/

   regs.h.ah = 2     ; // BIOS Video Service Function 2
   regs.h.bh = 0     ; // page 0
   regs.h.dh = row   ; // row
   regs.h.dl = column ; // column
   int86(VIDEO_INT , &regs, &regs) ;
   printf("                                                    ");

   regs.h.ah = 2   ; // BIOS Video Service Function 2
   regs.h.bh = 0     ; // page 0
   regs.h.dh = row   ; // row
   regs.h.dl = column ; // column
   int86(VIDEO_INT , &regs, &regs) ;
}

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

  void Clear()

  USAGE
   Clears the line sbeneath the video image

  INPUTS                              OUTPUTS
   row   : x coordinate       None
   column : y coordinate

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

void Clear(void)
{
int i;

for(i=18;i<=23;i++)
```

```
    printf("                                                    ");
}

/* ************************
  void DisplayStartupScreen(void)

  USAGE
  Draw the Startup Screen

  INPUTS          OUTPUTS
   None           None

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

void DisplayStartupScreen(void)
{
   /*      Display Title */
   printf("\tVIEW OF CAMERA MOUNTED ON TOOL\n" ) ;
   printf("\t=========================== " ) ;

   /* BIOS Video Service : Hide cursor */
  /* regs.h.ah = 1          ;
   regs.h.ch = 0x20   ;
   regs.h.cl = 0      ;
   int86(VIDEO_INT , &regs, &regs) ;*/

   /* Draw a main window and fill WHITE */
   DrawWindow(WIN_X, WIN_Y, WIN_WIDTH ,WIN_HEIGHT ,WHITE ) ;

}


/* *****************************
void CrossHair(int Number, int Colour)

USAGE
draws a cross hair on the screen at the position and colour specified

INPUTS
the camera carteslan position and colour

OUTPUTS
a crosshair !

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

void CrossHair(int Number, int Colour)
{
 long i, j, centre[30];
 WINREC far* lpWinRec = (WINREC far*) &winrec        ;
 int k;
```

```
ptrbuffer=ptr ;
/*middle=dwMemorySize/2 + 1.5*wWidth; */
/*centre = y * WIDTH + x; */                    //centre of crosshair

for(k=1;k<=Number;k++)
{
centre[k] = (long)CsHairY[k] * WIDTH + (long)CsHairX[k];

 for(i=centre[k]-10;i<centre[k]+10;i+=2)                    //horizontal bar
   ptrbuffer[i]=Colour;

 for(i=centre[k]-2*WIDTH;i<centre[k]+2*WIDTH;i+=WIDTH)    //vertical bar
   {
   ptrbuffer[i]=Colour;
   ptrbuffer[i+1]=Colour;
   ptrbuffer[i+2]=Colour;
   }
 }

 ReadORWriteBuffer(WRITE);
 }


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


USAGE
s
  1 and 5

INPUTS


OUTPUTS



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

void Histogram()
{
unsigned long i;

ReadORWriteBuffer(READ) ;
/*
for(i=0;i<=dwMemorysize;i++)
  {

  }
*/
```

```
}

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

USAGE
s
  1 and 5

INPUTS


OUTPUTS


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

void Freeze(void)
{
regs.x.ax = 1;                          //freezes image
regs.x.bx = VB_SETFREEZE;
if (!CallVbIstdrv(&regs, &segregs))
  DisableDriver();
FzeFlag=FROZEN;
}

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

USAGE
s
  1 and 5

INPUTS


OUTPUTS


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

void UnFreeze(void)
{
regs.x.ax = 0;                          //unfreezes image
regs.x.bx = VB_SETFREEZE;
if (!CallVbIstdrv(&regs, &segregs))
```

```
  DisableDriver();
  eFlag=LIVE;


/* ******************************
void FindEdges(void)

USAGE
finds the edges of the cube or hole using the Sobel edge detector

INPUTS
the image

OUTPUTS
the position of the edges found stored in camera X and Y coordinates

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

void FindEdges(void)
 {
 int StepY = 4;
 int StepX = 25;
 int MAXX=400, MINX=-400;
 int MAXY=400, MINY=-400;

 int n=0, EdgeStrX=0, EdgeStrY=0;
 int MaxX, MinX, MinY, MaxY;
 int a,b,c,d,e,f,g,h,i,j,k,l;
 int Xave=0, Yave=0;
 unsigned long index;
 unsigned long x, y, x1=0, x2=0, y1=0, y2=0,

 WINREC far* lpWinRec = (WINREC far*) &winrec      ;

 SetCursor(ACTIVITY,2);
 printf("Finding Edges");

 ptrbuffer=ptr ;
 ReadORWriteBuffer(READ) ;

CntrHor=0;  CntrVer=0;

 /*  *******FIND HORIZONTAL EDGES*******  */

 MaxX=MAXX; MinX=MINX;

 for(y=5;y<HEIGHT-5;y+=StepY)
  {
  for(x=50;x<WIDTH-50;x++)
       {
```

```
        Index=y*WIDTH+x;
        a=ptrbuffer[Index+1-WIDTH];
        b=ptrbuffer[Index+1];
        c=ptrbuffer[Index+1+WIDTH];
        d=ptrbuffer[Index-1-WIDTH];
        e=ptrbuffer[Index-1];
        f=ptrbuffer[Index-1+WIDTH];

        EdgeStrX = d+2*e+f - (a+2*b+c);
        if(EdgeStrX<MinX       )              //for black fore, white back
             { MinX=EdgeStrX;  x1=x;   }      //left edge

        if(EdgeStrX>MaxX       )              //for black fore, white back
             { MaxX=EdgeStrX; x2=x;    }      //right edge

        }  /*end of row*/

    if( x1!=0 && x2!=0 && x2>x1+50)
     { LeftX[CntrHor]=x1;
          RightX[CntrHor]=x2;
          Y[CntrHor]=y; CntrHor++;
     }

    MaxX=MAXX;  MinX=MINX; x1=x2=0;

    } /*end of col*/

/* *******FIND VERTICAL EDGES******** */

for(x=50;x<WIDTH-50;x+=StepX)
 {
 for(y=10;y<HEIGHT;y++)
       {
       Index=y*WIDTH+x;
       g=ptrbuffer[Index-1+WIDTH];
       h=ptrbuffer[Index+WIDTH];
       i=ptrbuffer[Index+1+WIDTH];
       j=ptrbuffer[Index-1-WIDTH];
       k=ptrbuffer[Index-WIDTH];
       l=ptrbuffer[Index+1-WIDTH];

       EdgeStrY = j+2*k+l - (g+2*h+i);
       if(EdgeStrY>MaxY       )          //for black fore, white back
        {      MaxY=EdgeStrY;       y2=y; }   //bottom edge
       if(EdgeStrY<MinY )                //for black fore, white back
        {      MinY=EdgeStrY;       y1=y; }   //top edge

       }

       if(y1!=0 && y2!=0 && y2>y1+5)
        { TopY[CntrVer]=y1;
```

```
            BtmY[CntrVer]=y2;
            X[CntrVer]=x;
            CntrVer++;
          }

    MaxY=MAXY;          MinY=MINY;   y1=y2=0;
  }

SetCursor(ACTIVITY,2);
printf("                ");

}


/* ****************************
int Centroid()

USAGE
finds the cube or hole centre of area

INPUTS
the position of the image coordinates

OUTPUTS
the centre of area of the cube or hole in camera coordinates
**************************** */

Int Centroid()
 {
 int Xave=0, Yave=0;
 int TotalXave=0;
 int TotalYave=0;
 int i;
 int Wd, Ht, Wdave=0, Htave=0;

 SetCursor(ACTIVITY,2);
 printf("Finding Centroid");

 for(i=0;i<CntrHor;i++)
  {
  Xave = ( LeftX[i] + RightX[i] )/2 ;
  TotalXave += Xave;
  Wd =  RightX[i] - LeftX[i];
  Wdave += Wd;
  }

 for(i=0;i<CntrVer;i++)
  {
  Yave = ( TopY[i] + BtmY[i] )/2 ;
  TotalYave += Yave;
  Ht =  BtmY[i] - TopY[i];
```

```
  Htave += Ht;
  }

if(CntrHor!=0 && CntrVer!=0)
  {
  CentroidX = TotalXave/CntrHor;
  CentroidY = TotalYave/CntrVer;
  ObjWd =  Wdave/CntrHor;
  ObjHt =  Htave/CntrVer;
  }
else
  Errors(1);

LimX = ObjWd/2;
LimY = (ObjHt/2) * VideoScale;

OffsetX =  (float)CentroidX - (float)WIDTH/2 ;
OffsetY =  (float)CentroidY - (float)HEIGHT/2;

OffsetY = -1*OffsetY;      //the CCD orign is at the
OffsetX = -1*OffsetX;                        //top left and not bottom right

CsHairX[1]=WIDTH/2;
CsHairX[2]=CentroidX;
CsHairY[1]=HEIGHT/2;
CsHairY[2]=CentroidY;

CrossHair(2,White);

SetCursor(ACTIVITY,2);
printf("                    ");
}


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


USAGE
s
  1 and 5

INPUTS


OUTPUTS


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

```
MaxDist()
{
float dx, dy, PixDist[100];
float Max=0;
int i, j=0, Mkr, k=0, Flag=0, NoVals, Flag2=0;
int XScale=5, YScale=1, Num;

SetCursor(ACTIVITY,2);
printf("Locating Corners");

/*Num = CntrHor+CntrVer;
PixDist = (float *) calloc((int)Num, sizeof(float) );*/

  for(i=0;i<CntrHor;i++)
  {
        dx = abs ( (CentroidX - LeftX[i]) );
        dy = abs ( (CentroidY - Y[i]) * VideoScale );
        PixDist[j] = sqrt( pow((float)dx,2) + pow((float)dy,2) );
        Xdist[j] = LeftX[i];
        Ydist[j] = Y[i];
        j++;
  }

  for(i=0;i<CntrHor;i++)
  {
        dx = abs ( (CentroidX - RightX[i]) );
        dy = abs ( (CentroidY - Y[i]) * VideoScale );
        PixDist[j] = sqrt( pow((float)dx,2) + pow((float)dy,2) );
        Xdist[j] = RightX[i];
        Ydist[j] = Y[i];
        j++;
  }


SetCursor(ACTIVITY,2);
printf(" end of maxdist  ");

Sort(PixDist,j);
}


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


USAGE
s
  1 and 5

INPUTS
```

```
OUTPUTS


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

Sort(float list[],int size)
{
   int out,in, tp,i;
   float temp;

   Index = (int *) calloc((int)size, sizeof(int) );

   for(i=0;i<size;i++)
         Index[i]=i;

   for(out=0;out<size-1;out++)
         for(in=out+1;in<size;in++)
            {
                  if(list[out] > list[in] )
                  {
                     tp=Index[in];
                     Index[in]=Index[out];
                     Index[out]=tp;
                     temp=list[in];
                     list[in]=list[out];
                     list[out]=temp;
                  }
            }

Corners(list,Index,size-1);

}


/* ******************************
Corners(float Dist[],int Index[],int size)

USAGE
finds the corners from the arrays holding the edge positions

INPUTS
the edge positions

OUTPUTS
the positions of the 4 corners

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

```
Corners(float Dist[],int Index[],int size)
{
   int Cnr[6],CnrNo[6], start=size, Mkr[6], Flag, NoCnrs=4;
   int wait, k, DeltaX[5], DeltaY[5];
   int i;
   int dx1,dx2,dx3,dx4,dy1,dy2,dy3,dy4;
   int Ang[5];

  Cnr[0]=Index[start];
  Mkr[0]=size;

   for(k=1;k<=NoCnrs-1;k++)
   {
        for(i=start-1;i>start-11;i--)
        {
        Flag=0;
        switch (k)
        {
        case 1: dx1 = abs(Xdist[Index[i]]-Xdist[Cnr[k-1]]);
                dy1 = abs(Ydist[Index[i]]-Ydist[Cnr[k-1]]) * VideoScale;
                if( dx1>LimX || dy1>LimY )
                {
                   Cnr[k]=Index[i];
                   Mkr[k]=i;
                   Flag=1;
                }
                break;
        case 2: dx1 = abs(Xdist[Index[i]]-Xdist[Cnr[k-1]]);
                dy1 = abs(Ydist[Index[i]]-Ydist[Cnr[k-1]])* VideoScale;
                dx2 = abs(Xdist[Index[i]]-Xdist[Cnr[k-2]]);
                dy2 = abs(Ydist[Index[i]]-Ydist[Cnr[k-2]])* VideoScale;
                if( (dx1>LimX || dy1>LimY )  && ( dx2>LimX || dy2>LimY ) )
                   {
                   Cnr[k]=Index[i];
                   Mkr[k]=i;
                   Flag=1;
                   }
                break;

        case 3: dx1 = abs(Xdist[Index[i]]-Xdist[Cnr[k-1]]);
                dy1 = abs(Ydist[Index[i]]-Ydist[Cnr[k-1]])* VideoScale;
                dx2 = abs(Xdist[Index[i]]-Xdist[Cnr[k-2]]);
                dy2 = abs(Ydist[Index[i]]-Ydist[Cnr[k-2]])* VideoScale;
                dx3 = abs(Xdist[Index[i]]-Xdist[Cnr[k-3]]);
                dy3 = abs(Ydist[Index[i]]-Ydist[Cnr[k-3]])* VideoScale;
                if( (dx1>LimX||dy1>LimY) && (dx2>LimX||dy2>LimY) && (dx3>LimX ||
dy3>LimY) )
                {
                   Cnr[k]=Index[i];
                   Mkr[k]=i;
                   Flag=1;
```

```
                }
                break;
        }
        if(Flag==1)
          break;
        }
  start--;
  }

TargetAngle( Cnr );

SetCursor(ACTIVITY,2);
printf("                    ");

}


/* ****************************
TargetAngle(int Cnr2[])

USAGE
calculate the rotation of the cube hole from the corners found

INPUTS
the position of the corners

OUTPUTS
the rotation of the cube or hole wrt the camera's x,y coordinates

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

TargetAngle(int Cnr2[])
{
int i, out, in, tp, CnrNo[6];
long temp, PixNo[6];
float Ang1, Ang2;
int dx1, dy1, dx2, dy2;


SetCursor(ACTIVITY,2);
printf("Calculating Target Orientation");

for(i=0;i<=3;i++)
  PixNo[i] = Xdist[Cnr2[i]] + Ydist[Cnr2[i]] * WIDTH;

for(i=0;i<4;i++)
        CnrNo[i]=Cnr2[i];


  for(out=0;out<4-1;out++)
```

```
        for(in=out+1;in<4;in++)
          {
              if(PixNo[out] > PixNo[in] )
              {
                 tp=CnrNo[in];
                 CnrNo[in]=CnrNo[out];
                 CnrNo[out]=tp;
                 temp=PixNo[in];
                 PixNo[in]=PixNo[out];
                 PixNo[out]=temp;
              }
          }

    CsHairX[1]=Xdist[CnrNo[0]];
    CsHairX[2]=Xdist[CnrNo[1]];
    CsHairX[3]=Xdist[CnrNo[2]];
    CsHairX[4]=Xdist[CnrNo[3]];

    CsHairY[1]=Ydist[CnrNo[0]];
    CsHairY[2]=Ydist[CnrNo[1]];
    CsHairY[3]=Ydist[CnrNo[2]];
    CsHairY[4]=Ydist[CnrNo[3]];

    CrossHair(4,White);


dx1 = Xdist[CnrNo[0]] - Xdist[CnrNo[2]];
dy1 = ( Ydist[CnrNo[2]] - Ydist[CnrNo[0]] ) * VideoScale;
Ang1 = atan( (float)dx1/(float)dy1 ) * R_TO_D  ;
dx2 = Xdist[CnrNo[1]] - Xdist[CnrNo[3]];
dy2 = ( Ydist[CnrNo[3]] - Ydist[CnrNo[1]] ) * VideoScale;
Ang2 = atan( (float)dx2/(float)dy2 ) * R_TO_D ;

OffsetAngle = (int)ceil( (Ang1 + Ang2)/2 );

SetCursor(ACTIVITY,2);
printf("                                  ");
SetCursor(ACTIVITY,2);
printf("Showing Corners and Centroid Found");

Sleep(1000);

SetCursor(ACTIVITY,2);
printf("                                  ");

}


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

```
USAGE
s
  1 and 5

INPUTS


OUTPUTS


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

void FreeMem(void)
{
free(Index);

}


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


USAGE
s
  1 and 5

INPUTS


OUTPUTS


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

int PrintEdges(char Direction)
{
int count;

if(Direction=='h' || Direction=='H')
  {
  for(count=0;count<CntrHor;count+=2)
    {
    CsHairX[count+1]=LeftX[count];
    CsHairY[count+1]=Y[count];
    CrossHair(count,White);
    }
```

```
  for(count=0;count<CntrHor;count+=2)
   {
   CsHairX[count+1]=RightX[count];
   CsHairY[count+1]=Y[count];
   CrossHair(count,White);
   }
  }

 if(Direction=='v' || Direction=='V')
  {
  for(count=0;count<CntrVer;count+=2)
   {
   CsHairX[count+1]=X[count];
   CsHairY[count+1]=TopY[count];
   CrossHair(count,White);
   }
  for(count=0;count<CntrVer;count+=2)
   {
   CsHairX[count+1]=X[count];
   CsHairY[count+1]=BtmY[count];
   CrossHair(count,White);
   }
  }

}


ClearScreen()
{
/* clear screen */
 regs.x.ax = 3;
 int86(VIDEO_INT, &regs, &regs);
}


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

  Main Program

  USAGE
    Main program entry point.

  INPUTS              OUTPUTS
    None                None

 ****************/
void TurnOnCamera(void)
{

 ClearScreen();
```

```
/* Check for the driver's signature string. If failed , terminate program */
if (wVBIntrNum = FindVbistdrv(DRIVER_SIG))
 {
  /* Initialise driver */
  regs.x.bx = VB_INITIALIZE;
  if (!CallVbistdrv(&regs, &segregs))
        DisableDriver();

  DisplayStartupScreen() ;
  SetupVideoWindow();
  AllocateMemory() ;
  CameraFlag=1;
 }
  else
  {
  Errors(8);
  }
}
```

CONSTS.C

```
/*
   these are the robots constants :
   Tool   =      tool length
   a[6]   =      the robot's link lengths
   d[6]   =      the robot's base height and tool length
   a,b,c...      =  the constants before the x^4 etc terms of the polynomial
                    decribing the pixel/mm wrt distance of the camera
                    ax... for the horizontal axis
                    ay... for the vertical axis
   Limit[6]      =  the joint limits in degrees
   Intersect     =  the distance from the intersection of the camera cl
                    to the tool grippers
   Beta =        the angle the camera centre line makes with the tool
                 centre line                              U
   ZeroPos[6]    =  the current zero position              |
   HomePos[6]    =  the current home position              o
   Alpha[6]      =  the robot's twist angles         |
                                                     o
zero        o--o--o--o                    home       |
            |     |                                  o
            |     M                                  |
            |                                        o
            -                                        |
                                                     -
*/


/*robot initialization values*/
#define TOOL    280
#define OBJDEPTH  42

/*robot defined values*/
int AxisSwitch[6]={ 0 , -1 , 1 , -1 , -1 , -1 };          /*to get the right rotations*/
int Pulses[6]={0, 850 , 860 , 870 , 533 , 352 };
int     Tool = TOOL;
int a[6]    = {0, 0 , 350 , 350 , 0 , 0};
int d[6]    = {0, 775 , 0 , 0 , 0 , TOOL};
float JntLmt[7] = {0 , 100, 110 , 110, 110, 270, -180}; //last two values are
float Zero[6]    = {0, 0 , 0 , 0 , 0 , 0 };             //for the sensor boundries
float Home[6]    = {0, 0 , -90 , 0 , -90 , 180 };
int Alpha[6] = {0 , -90 , 0 , 0 , -90 , 0 };

/*robot joint angles and tool position and orientation*/
float AbsAng[6];    //Q(1,2,3,4,5)
float AbsPos[3];    //(X,Y,Z)
float Yaw, Pitch, Roll;
char LimitFlag;
```

```
/*output initialization values*/

int ABSANG  =15;
int ABSPOS  =16;
int ORIENT  = 17;
int RESULTS1 =        18;
int RESULTS2 =        19;
int RESULTS3 =        20;
int ACTIVITY = 22;
int CHOICE  = 23;

/*camera initialization values*/
int CameraFlag=0;
float ax=1.70443e-9, bx=-2.21865e-6, cx=0.00112057, dx=-0.277211, ex=34.2303;
float ay=2.03885e-10, by=-3.64075e-7, cy=0.00023185, dy=-0.068:675, ey=9.57066;

float OffsetX, OffsetY, OffsetAngle;
int ObjWd, ObjHt;
int ObjDepth=OBJDEPTH;
int FzeFlag=1;

/*user variables*/
char choice[6];


/*Inputhandler choices*/
int ESCAPE  =          0x1B;
int FREEZE  =  1   ;       /*freezes video image*/
int TARGET  =          2   ;       /*finds centroid, cnrs & orientation*/
int EDGES   =  3   ;       /*marks the edges of the target*/
int VIDEO   =  4   ;       /*toggles video on / off*/
int CENTROID =  5  ;       //finds the centroid only


/*camera and ultrasound mounting on robot details
Xo, Yo, Zo are the camera displacements from the tool centre
Beta is the camera rotation of the camera about the tool normal vector*/

float Ma=81, Mb=165, Mc=143, Md=90, Me=32;
float IntersectHeight=306+OBJDEPTH;
float Zus = -70;
float Yo=100.0, Xo=0.0, Zo=-58.00;
float Beta=19.00, Sigma=180;


/*ultrasound global variables*/
float Dist;

int CsHairX[30], CsHairY[30];

/*camera calibration values
```

distance	camera x axis		pix/mm camera x axis	camera y axis
179	288	91	9.6	3.033
146	350	107	11.67	3.567
200	250	75	8.333	2.5
215	237	74	7.9	2.467
230	230	68	7.67	2.267
245	216	63	7.2	2.1
260	202	63	6.73	2.1
275	180	60	6.0	2.0
290	193	59	6.433	1.967
300	177	55	5.9	1.833
140	356	108	11.867	3.6
120	417	129	13.9	4.3
114	418	125	13.93	4.167
350	147	48	4.9	1.6
325	154	52	5.133	1.733
370	140	44	4.67	1.467
402	128	41	4.267	1.367
414	125	41	4.167	1.367

polynomial fit

x axis : pix/mm = ax*D^4 + bx*D^3 + cx*D^2 + dx*D + ex
y axis : pix/mm = ay*D^4 + by*D^3 + cy*D^2 + dy*D + ey

x axis	y axis
ax=1.70443e-9	ay=2.06885e-10
bx=-2.21865e-6	by=-3.64075e-7
cx=0.00112057	cy=0.00023185
dx=-0.277211	dy=-0.0681675
ex=34.2303	ey=9.57060

*/

HTMS.C

```
/*******************
USAGE
this file contains all the HTMs namely :

CtoT        camera to tool
TtoB        tool to base
YPR         the tool to base rotation matrix using
            the yaw, pitch and roll of the tool as the input

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


#include <math.h>
#include "c:\masters\proj\ext_vars.h"

#define R_TO_D                  57.29578

void TtoB();
void CtoT();
void YPR(float Yaw, float Pitch, float Roll);


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

USAGE

INPUTS                                                  OUTPUTS

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

void CtoT()
{
float c1,c2,c3,c23,c234,c5,Cb;
float Cb,Sb;
float T11, T12, T13, T14, T21, T22, T23, T24;
float T31, T32, T33, T34, T41, T42, T43, T44;
extern float Beta, Xo,Yo,Zo;

Cb =    cos(Beta/R_TO_D);
Sb =    sin(Beta/R_TO_D);

T11 = 1;        T12 = 0;        T13 = 0;        T14 = Xo;
T21 = 0;        T22 = Cb;       T23 = -Sb;      T24 = Yo;
T31 = 0;        T32 = Sb;       T33 = Cb;       T34 = Zo;
T41 = 0;        T42 = 0;        T43=0;          T44=1;

Xtool = T11*Xcam + T12*Ycam + T13*Zcam + Xo;
Ytool = T21*Xcam + T22*Ycam + T23*Zcam + Yo;
```

```
Ztool = Dist - 50;                    //can also use above HTM but using US
                                      //directly will give better results

}

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

USAGE

INPUTS                                                    OUTPUTS

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

void TtoB()
{
float c1,c2,c3,c23,c234,c5;
float s1,s2,s3,s23,s234,s5;
float R11, R12, R13, R14, R21, R22, R23, R24;
float R31, R32, R33, R34, R41, R42, R43, R44;


c1      = cos(AbsAng[1] / R_TO_D);
c5      = cos(AbsAng[5] / R_TO_D);
c23     = cos((AbsAng[2]+AbsAng[3])/R_TO_D);
c234    = cos((AbsAng[2]+AbsAng[3]+AbsAng[4])/R_TO_D);
s1      = sin(AbsAng[1] / R_TO_D);
s5      = sin(AbsAng[5] / R_TO_D);
s23     = sin((AbsAng[2]+AbsAng[3])/R_TO_D);
s234    = sin((AbsAng[2]+AbsAng[3]+AbsAng[4])/R_TO_D);

R11=c1*c234*c5+s1*s5; R12=-c1*c234*s5+s1*c5;        R13=-c1*s234;
R14=c1*(a[2]*c2+a[3]*c23-d[5]*s234);
R21=s1*c234*c5-c1*s5; R22=-s1*c234*s5-c1*c5;        R23=-s1*s234;
R24=s1*(a[2]*c2+a[3]*c23-d[5]*s234);
R31=-s234*c5;  R32=s234*s5;            R33=-c234;
R34=d[1]-a[2]*s2-a[3]*s23-d[5]*c234;
R41=0;            R42=0;               R43=0;
R44=1;


Xbase=R11*Xtool + R12*Ytool + R13*Ztool;  /* + R14*1;*/
Ybase=R21*Xtool + R22*Ytool + R23*Ztool;  /* + R24*1;*/
Zbase=R31*Xtool + R32*Ytool + R33*Ztool;  /* + R34*1;*/

}

/*******************
void YPR(float Yaw, float Pitch, float Roll)
```

```
USAGE
calculates the tool rotation matrix from the orientation of the tool
in yaw, pitch and roll angles

INPUTS                                                    OUTPUTS
the tool yaw, pitch and roll angles          the rotational matrix

**********/

void YPR(float Yaw, float Pitch, float Roll)
{
float Cy,Cp,Cr,Sy,Sp,Sr, RotYPR[4][4];

Cy=cos(Yaw/R_TO_D); Cp=cos(Pitch/R_TO_D);          Cr=cos(Roll/R_TO_D);
Sy=sin(Yaw/R_TO_D);  Sp=sin(Pitch/R_TO_D); Sr=sin(Roll/R_TO_D);

RotYPR[1][1]=Cp*Cr;   RotYPR[1][2]=Sy*Sp*Cr - Cy*Sr; RotYPR[1][3]=Cy*Sp*Cr + Sy*Sr;
RotYPR[2][1]=Cp*Sr;   RotYPR[2][2]=Sy*Sp*Sr + Cy*Cr;RotYPR[2][3]=Cy*Sp*Sr - Sy*Cr;
RotYPR[3][1]=-Sp;     RotYPR[3][2]=Sy*Cp;                 RotYPR[3][3]=Cy*Cp;

}
```

INIT_RBT.C

```
#include"c:\masters\proj\ext_vars.h"
//  #include"c:\masters\proj\fnproto.h"

/*function prototypes*/
Init_Robot();


Init_Robot()
{
int AxisNo, i;

SetCursor(ACTIVITY,2);
printf("Initializing Robot and AtoD Board");

BrakesInit();
for( AxisNo=1;AxisNo<=5;AxisNo++)
   {
   SfwReset(AxisNo);
   Sbpa(AxisNo,0x7FFF);
   Rsti(AxisNo,0x10);
   Lfil(AxisNo,15,1,10,1000,0x080F);
   Udf(AxisNo);
   }

StartUp();
}


StartUp()
{
/*change to the working directory*/
_chdir("c:\\masters\\proj");

SetCursor(CHOICE,2);
printf("Read joint angles from file ?");
choice[0]=getche();
if(choice[0]=='y' || choice[0]=='Y')
  Read();

if(choice[0]=='h' || choice[0]=='H')
  {
  AbsAng[1] = Home[1]; AbsAng[2] = Home[2]; AbsAng[3] = Home[3];
  AbsAng[4] = Home[4];AbsAng[5] = Home[5];
  AbsPos[0] = 0; AbsPos[1]=0; AbsPos[2]=1775;
  }
```

```
If(choice[0]!='h' && choice[0]!='H' && choice[0]!='y' && choice[0]!='Y')
  {
  SetCursor(RESULTS1,2);
  printf("enter joint angles <q1,q2,q3,q4,q5> :");
  scanf("%f %f %f %f %f",AbsAng[1],AbsAng[2],AbsAng[3],AbsAng[4],AbsAng[5]);
  }


InitADBoard();
UltraSndSensor();

If(Dist<5)
  Errors(6);
}
```

INV_KIN.C

```
/*SPECIFIC CASE -- TOOL ALWAYS POINTS TOWARDS GROUND => q234=0
*/

#include <math.h>
#include "c:\masters\proj\ext_vars.h"
#include"c:\masters\proj\fnproto.h"

/*function prototype*/
Envelope(float Pos[], float Q[]);

int WorkEnvelope;

/*********************
InvKinematics(float Pos[], float Rot[4][4], int Filter)

USAGE
calculates the joint angles in order to place the tool at the
position and orientation specified as an input

INPUTS
the required tool position and orientation and if any axis are to
be ignored

OUTPUTS
the necessary joint angles

*********************/
InvKinematics(float Pos[], float Rot[4][4], int Filter)
{
float Q[6], Diff[6], Q234;
float Bee, b1, b2;
unsigned long den1;
float num1, num2, den2;
int j, l, Motion, EnFlag=FALSE, J3above00=FALSE, TrkFlag;


Q[1]= atan2( Pos[2] , Pos[1] );

Q234 = atan2( -( cos(Q[1])*Rot[1][3]+sin(Q[1])*Rot[2][3] ),-Rot[3][3] );

b1 = cos(Q[1])*Pos[1] + sin(Q[1])*Pos[2] + d[5]*sin(Q234);
b2 = d[1] - a[4]*sin(Q234) - d[5]*cos(Q234) - Pos[3];
Bee = pow(b1,2) + pow(b2,2) ;
num1 = Bee - pow(a[2],2) - pow(a[3],2) ;
den1 = 2 * (unsigned long)a[2] * (unsigned long)a[3];

if( fabs(num1/den1)<=1 )
  Q[3] = acos ( num1 / den1 ) ;
if( fabs(num1/den1)>=1 )
```

```
  {
  num1=num1-den1;
  Q[3] = acos ( num1 / den1 ) ;
  J3above90=TRUE;
  }

num2 = ( (float)a[2] + (float)a[3]*cos(Q[3]) ) * b2 - (float)a[3]*sin(Q[3])*b1 ;
den2 = ( (float)a[2] + (float)a[3]*cos(Q[3]) ) * b1 + a[3]*sin(Q[3])*b2 ;
Q[2] = atan2 ( num2 , den2 ) ;

Q[5]=atan2(sin(Q[1])*Rot[1][1] - cos(Q[1])*Rot[2][1],
          sin(Q[1])*Rot[1][2] - cos(Q[1])*Rot[2][2] );

If(ErrFlag==FALSE)
  {
  If(J3above90==TRUE)
    Q[3]=(180/R_TO_D) - Q[3];

  Q[4]=Q234-Q[2]-Q[3];

  for(i=1;i<=5;i++)
    Diff[i]=Q[i]*R_TO_D-AbsAng[i];

/*no point in Q5 moving more than +-180 degrees*/

  if(Diff[5]>180)
    Diff[5]=360-Diff[5];

  if(Diff[5]<-180)
    Diff[5]=360+Diff[5];

  /*
  SetCursor(RESULTS1,2);
  printf("after q1=%.3f  q2=%.3f  q3=%.3f q4=%.3f  q5=%.3f",
    Q[1]*R_TO_D,Q[2]*R_TO_D,Q[3]*R_TO_D,Q[4]*R_TO_D,Q[5]*R_TO_D);

  SetCursor(RESULTS2,2);
  printf("Diffs q1=%.3f  q2=%.3f  q3=%.3f q4=%.3f  q5=%.3f",
    Diff[1],Diff[2],Diff[3],Diff[4],Diff[5]);
  getche();
  */

  if(Filter==ZERO4)                    //le don't move axis 4
    {
    TrkFlag=NOTRK4;
    AbsAng[4]+=Diff[4];    Diff[4]=0;
    }
  if(Filter==TRK4)                     //le move axis 4
    TrkFlag=TRK4;

   Envelope(Pos,Q);
```

```
    If(WorkEnvelope==TRUE)
        MoveRobot(Diff,TrkFlag);
    else
        Errors(9);

  }
}


/*******************
Envelope(int Position)

USAGE
determines whether the requested position of the tool tip is within the
robot's outer envelope or within the robots inner envelope

INPUTS
the required tool position

OUTPUTS
if the position is allowable or not

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

Envelope(float Pos[], float Q[])
{
float u1,u2, p1,p2,p3;
float c234,s234, tester,tested,dummy1,dummy2;
int OuterEnvelope, InnerEnvelope;

c234 = cos(Q[2]+Q[3]+Q[4]);
s234 = sin(Q[2]+Q[3]+Q[4]);

u1 = a[4]*c234 - d[5]*s234;
u2 = d[1] - a[4]*s234 - d[5]*c234;

/*outer envelope*/
//test limit on p1
tester = a[2]+a[3]+u1;
tested = fabs(Pos[1]);
If(tested<=tester)
  OuterEnvelope=TRUE;
else
  OuterEnvelope=FALSE;

//test limit on p2
if(OuterEnvelope==TRUE)
{
tester = pow((a[2]+a[3]+u1),2) - pow(Pos[1],2);
```

```
tested = fabs(pow(Pos[2],2));
if(tested<=test  )
  OuterEnvelope=TRUE;
else
  OuterEnvelope=FALSE;
}

//test limit on p3
if(OuterEnvelope==TRUE)
{
dummy1 = sqrt( pow(Pos[1],2) + pow(Pos[2],2) ) - u1;
dummy2 = pow( (a[2]+a[3]),2 );
tester = dummy2 - dummy1;
tested = fabs(pow((Pos[3]-u2),2));
if(tested<=tester)
  OuterEnvelope=TRUE;
else
  OuterEnvelope=FALSE;
}

/*Inner work envelope not yet developed*/
innerEnvelope=TRUE;

if(OuterEnvelope==TRUE && innerEnvelope==TRUE)
  WorkEnvelope=TRUE;
else
  WorkEnvelope=FALSE;

}


/*****************
ToolPos(float DumbAng[])

USAGE
calculates the tool position

INPUTS
the joint angles

OUTPUTS
the tool position in robot base coordinates

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

ToolPos(float DumbAng[])
{
float c1,c2,c3,c5,c23,c234;
float s1,s2,s3,s5,s23,s234,num;
int i;
float R11, R12, R13, R14, R21, R22, R23, R24;
```

```
float R31, R32, R33, R34, R41, R42, R43, R44;

c1  = cos( DumbAng[1]/R_TO_D)                ;
c2  = cos( DumbAng[2]/R_TO_D)                ;
c3  = cos( DumbAng[3]/R_TO_D)                ;
c5  = cos( DumbAng[5]/R_TO_D)                ;
c23 = cos( (DumbAng[2]+DumbAng[3])/R_TO_D )          ;
c234 = cos( (DumbAng[2]+DumbAng[3]+DumbAng[4])/R_TO_D )      ;
s1  = sin( DumbAng[1]/R_TO_D)                ;
s2  = sin( DumbAng[2]/R_TO_D)                ;
s3  = sin( DumbAng[3]/R_TO_D)                ;
s5      = sin( DumbAng[5]/R_TO_D)            ;
s23 = sin( (DumbAng[2]+DumbAng[3])/R_TO_D )          ;
s234 = sin( (DumbAng[2]+DumbAng[3]+DumbAng[4])/R_TO_D )      ;


R11=c1*c234*c5+s1*s5; R12=-c1*c234*s5+s1*c5;  R13=-c1*s234;
R14=c1*(a[2]*c2+a[3]*c23-d[5]*s234);
R21=s1*c234*c5-c1*s5; R22=-s1*c234*s5-c1*c5;        R23=-s1*s234;
R24=s1*(a[2]*c2+a[3]*c23-d[5]*s234);
R31=-s234*c5;   R32=s234*s5;            R33=-c234;
R34=d[1]-a[2]*s2-a[3]*s23-d[5]*c234;
R41=0;          R42=0;                  R43=0;
R44=1;

AbsPos[0]=R14;
AbsPos[1]=R24;
AbsPos[2]=R34;

//calculate the roll, yaw and pitch angles
Roll  = atan2(R12,R11);
Yaw   = atan2(R23,R33);
Pitch = atan2(-R13,R11*cos(Roll) + R12*sin(Roll));

Roll*=R_TO_D;  Yaw*=R_TO_D;  Pitch*=R_TO_D;

}
```

LM_CMND.C

```
/* lm_cmmd.c  the robot control file written in C */

#include "c:\masters\proj\ext_vars.h"
#include"c:\masters\proj\fnproto.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

#define Brake      641
#define IO_Card    0x210
#define BUSY       0
#define IDLE       1
#define DUMMY          0

long fclk = 7000000;
long res=200;
long  Signal_Register, Index_Pos1, Index_Pos2, Dpos1,
                Dpos2, Dvel1, Dvel2, Rvel, Sum;
long Rpos1, Rpos2, Data_Word;
int Hi_Byte, Lo_Byte, SelectMaskHi, BrakeMask;
unsigned char AxisNo;
int Accel, Speed, Status_Array[8], Chip;
long DesPos2;


/* ********** ******
LimitCheck(float Angs[])

USAGE
checks that the a joint will limit will not be exceeded by performing
the required move

INPUTS
the current and required joint angles

OUTPUTS
whether a joint will be exceeded and which one

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

LimitCheck(float Angs[])
{
int i;
extern float JntLmt[7];
```

```
extern char LimitFlag;
float Limit[6];

Limit[1]=JntLmt[1]-Home[1];
Limit[2]=JntLmt[2]-Home[2];
Limit[3]=JntLmt[3]+AbsAng[2]-Home[2]-Home[3];
Limit[4]=JntLmt[4]+AbsAng[2]+AbsAng[3]-Home[2]-Home[3]-Home[4];

for(i=1;i<=4;i++)
  {
  if( fabs(AbsAng[i]+Angs[i]) >= Limit[i] )
     { Errors(i+9);       LimitFlag=TRUE;       break;  }
  }

If( fabs(AbsAng[5]+Angs[5]) >= JntLmt[5] )
     { Errors(5+9);       LimitFlag=TRUE;       }

if( fabs(AbsAng[5]+Angs[5]) <= JntLmt[6] )
     { Errors(5+9);       LimitFlag=TRUE;       }

}

/* ********************
MoveRobot(float Angles[], int TrkAxis4)

USAGE
Loads parameters to the chip to move and then to check position and stop.
Also keeps track of the robots tool position and joint angles.
ToolOrient is a flag telling MoveRobot whether to let the tool
remain in the current orientation or move as Axis 2 & 3 move.

INPUTS
the angle each joint must move and whether joint 4 should move or not

OUTPUTS
the encoder pulses which are passed on to LtrjPLS()


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

MoveRobot(float Angles[], int TrkAxis4)
{
long Degs[6];
float factor=1.02;
extern char LimitFlag;

/*check to see if any angle's limit will be exceeded*/
LimitCheck(Angles);

if(LimitFlag==FALSE)
  {
```

```
  /*update record of all joint angles*/
    for(AxisNo=1;AxisNo<=5;AxisNo++)
         AbsAng[AxisNo] += Angles[AxisNo];

  /*set filter parameters according to the motion of each axis*/
  InitMove(Angles);

  /*make axis 4 change yaw, pitch and roll as 3 or 2 rotate*/
  if(TrkAxis4==TRK4)
    Angles[4]=Angles[2]+Angles[3]+Angles[4];

  for(AxisNo=1;AxisNo<=5;AxisNo++)
    Degs[AxisNo] = (long)( Angles[AxisNo] * (float)AxisSwitch[AxisNo] *
                        (float)Pulses[AxisNo] );

  /*pass the number of degrees the joints must move to Ltrj()*/
  for(AxisNo=1;AxisNo<=5;AxisNo++)
         {
         if(Degs[AxisNo]!=0)
           { LoadPrmsPLS(AxisNo, Degs[AxisNo]); Sleep(0); }
         }

  Sleep(800);
  NewStop(Angles);

  /*update position of tool*/
  Update();

   }
}

/* ********************
InitMove(float Angle[])

USAGE
loads the PID filter coefficients, the velocity and acceleration
for each joint

INPUTS
the angle each joint must move

OUTPUTS
none

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

InitMove(float Angle[])
{
int P,I,D, AxisNo;
unsigned int PosErr=4000;    /*about 5 degrees*/
```

```
for(AxisNo=1;AxisNo<=5;AxisNo++)
  {
  if(Angle[AxisNo]!=0)
    {
    SfwReset(AxisNo);
    Sbpa(AxisNo,0x7FFF);
    Rsti(AxisNo,0x10);
    Lpei(AxisNo,PosErr);
    Dfh(AxisNo);

    if(AxisNo==1)
          { Accel=10; P=500; I=1000; D=1500; Speed=250; }
    if(AxisNo==4)
          { Accel=10; P=500; I=1000; D=1500; Speed=450; }
    if(AxisNo==2 || AxisNo==3 )
          {
          if(Angle[AxisNo]<0 )
            { Accel=200; P=500; I=1000; D=1500; Speed=450; }
          if(Angle[AxisNo]>0 )
            { Accel=10;  P=500; I=1000; D=1500; Speed=450; }
          }
    if( AxisNo==5 )
          { Accel=200; P=500; I=100; D=1500; Speed=300; }

    Lfil(AxisNo,P,I,D,1000,0x080F);
    Udf(AxisNo);
    }
  }
}

/* ***********************
LoadPrmsPLS(int AxisNo, long Degrees)

USAGE
loads the speed, angle  and axisno to move onto the LM628 chip

INPUTS
the joint number and the number of degress it must rotate

OUTPUTS
none

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

LoadPrmsPLS(int AxisNo, long Degrees)
{
int Traj=0x03f;

LtrjPLS( AxisNo , Degrees , Speed , Accel , Traj );

Stt(AxisNo);
```

```
Sleep(2);
ReleaseBrake(AxisNo);

SetCursor(ACTIVITY,2);
printf("Loading Parameters for axisno %d ", AxisNo);
SetCursor(ACTIVITY,2);
printf("                                        ");

}

/* ***************
NewStop(float Angs[])

USAGE
applys the brake and resets the chip when the motion is complete
checks for motion complete by noticing when the encoder value is no longer
changing

INPUTS
the amount each joint must rotate

OUTPUTS
none

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

NewStop(float Angs[])
{
long Pos1, Pos2, Diff;
int AxisNo;

for(AxisNo=1;AxisNo<=5;AxisNo++)
  {
  if(Angs[AxisNo]!=0)
    {
    do
        {
        RdStat(AxisNo);
        Rdrp(AxisNo);     Pos1=Rpos2;
        Sleep(10);
        Rdrp(AxisNo);     Pos2=Rpos2;
        Diff=labs(Pos1-Pos2);
        SetCursor(RESULTS1,2);
        printf("Moving axisno  %d into position",AxisNo);
        }
    while(Diff>0);
    ApplyBrake(AxisNo);          Sleep(5);
    SfwReset(AxisNo);
    }
  }
```

```
for(AxisNo=1;AxisNo<=5;AxisNo++)
  {
  ApplyBrake(AxisNo);    Sleep(5);
  SfwReset(AxisNo);
  }

}


/* ***************
Sleep(int wait)

USAGE
Pauses for a specified number of milliseconds

INPUTS                                          OUTPUTS
the number of milliseconds                      none

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

Sleep(int wait)
{
clock_t goal;

goal =   (clock_t)wait + clock() ;
 while( goal > clock() );

}

/* ***************
Binary(int number)

USAGE
converts a BYTE into an array of binary. LSB is first.

INPUTS                                          OUTPUTS
the decimal number                  the binary number

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

Binary(int number)
 {
 int j;
 int Pattern[8];
 unsigned int mask;

    mask=0x80;
    for(j=0;j<=7;j++)
    {
        Status_Array[7-j]=(mask & number) ? 1 : 0;
        mask>>=1;
```

```
    }

 if(Status_Array[0]==0)
   Chip = BUSY;
 else
   Chip = IDLE;
 }

/* ***************
RdStat(int AxisNo)

USAGE
updates an array corresponding to the 8-bits of the status byte.
The first entry of the array corresponds to the LSB.

 Bit Number      Function

    7      Motor off
    6      Breakpoint reached (Interrupt)
    5      Excessive position error (Interrupt)
    4      Wraparound occurred (Interrupt)
    3      Index pulse observed (I terrupt)
    2      Trajectory complete (Interrupt)
    1      Command error (Interrupt)
    0      Busy bit

INPUTS                                  OUTPUTS
the joint number                    the joints status

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

RdStat(int AxisNo)
{
 int Status=0;

 outp(IO_Card+3,0x90);              /*A=Input;B,C=Output*/

 switch (AxisNo)
 {
   case 1:      SelectMaskHi=0x03; break;     /*0000,0011*/
   case 2:      SelectMaskHi=0x05; break;     /*0000,0101*/
   case 3:      SelectMaskHi=0x09; break;     /*0000,1001*/
   case 4:      SelectMaskHi=0x11; break;     /*0001,0001*/
   case 5:      SelectMaskHi=0x21; break;     /*0010,0001*/
   case 6:      SelectMaskHi=0x41; break;     /*0100,0001*/
 }

 outp(IO_Card+1,0x81);          Sleep(0);
 outp(IO_Card+2,0x80);          Sleep(0);
 outp(IO_Card+1,SelectMaskHi);  Sleep(0);
 outp(IO_Card+2,0x00);          Sleep(0);
```

```
  Status=inp(IO_Card+0);          Sleep(0);
  outp(IO_Card+2,0x80);           Sleep(0);
  outp(IO_Card+1,0x81);           Sleep(0);

  Binary(Status);

}

/* ******************
*                          READ/WRITE COMMANDS                    *
********************* */

/* ***************
Read_Data(int AxisNo)

USAGE
reads a 16-bit data word from the LM628.
The data word is converted to an array of binary format. The first
entry of the array corresponds to the LSB.

INPUTS
the joint number

OUTPUTS


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

Read_Data(int AxisNo)
{

/*
   do
     RdStat(AxisNo);
   while(Chip=BUSY);
*/

  switch (AxisNo)
  {
    case 1:  SelectMaskHi=0x83;  break;  /*1000,0011*/
    case 2:  SelectMaskHi=0x85;  break;  /*1000,0101*/
    case 3:  SelectMaskHi=0x89;  break;  /*1000,1001*/
    case 4:  SelectMaskHi=0x91;  break;  /*1001,0001*/
    case 5:  SelectMaskHi=0xA1;  break;  /*1010,0001*/
    case 6:  SelectMaskHi=0xC1;  break;  /*1100,0001*/
  }


  outp(IO_Card+3,0x90);              Sleep(0);
  outp(IO_Card+1,0x01);                    Sleep(0);
```

```
 outp(IO_Card+2,0x80);              Sleep(0);
 outp(IO_Card+1,SelectMaskHi);      Sleep(0);
 outp(IO_Card+2,0x00);              Sleep(0);
 Hi_Byte=inp(IO_Card+0);            Sleep(0);
 outp(IO_Card+2,0x80);              Sleep(0);
 outp(IO_Card+1,0x01);                    Sleep(0);
 outp(IO_Card+1,SelectMaskHi);      Sleep(0);
 outp(IO_Card+2,0x00);              Sleep(0);
 Lo_Byte=inp(IO_Card+0);            Sleep(0);
 outp(IO_Card+2,0x80);              Sleep(0);
 outp(IO_Card+1,0x01);                    Sleep(0);


 Data_Word = (long)Hi_Byte * 256 + (long)Lo_Byte;

}

/* **************
Write_Command(int AxisNo, unsigned char Command)

USAGE
writes an 8-bit   command to the LM628.

INPUTS                                          OUTPUTS
the joint number and the command                none

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

Write_Command(int AxisNo, unsigned char Command)
{

 do
  {
  if (AxisNo!=5)
   {
   RdStat(AxisNo);
   }
  }
 while(Chip=BUSY);

 switch (AxisNo)
 {
  case 1:  SelectMaskHi=0x03;  break;  /*0000,0011*/
  case 2:  SelectMaskHi=0x05;  break;  /*0000,0101*/
  case 3:  SelectMaskHi=0x09;  break;  /*0000,1001*/
  case 4:  SelectMaskHi=0x11;  break;  /*0001,0001*/
  case 5:  SelectMaskHi=0x21;  break;  /*0010,0001*/
  case 6:  SelectMaskHi=0x41;  break;  /*0100,0001*/
 }

 outp(IO_Card+3,0x80);                    Sleep(0);
```

```
 outp(IO_Card+1,0x81);                    Sleep(0);
 outp(IO_Card+2,0x80);                    Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
 outp(IO_Card+1,(SelectMaskHi & 0xFE) );   Sleep(0);
 outp(IO_Card+0,Command);                 Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
 outp(IO_Card+1,0x81);                    Sleep(0);


}

/* ********************
Write_Data(int AxisNo,long  Data_Word)

USAGE
writes a 16-bit data word to the LM628. The high byte is written first.

INPUTS                                   OUTPUTS
the joint number and the data word       none

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

Write_Data(int AxisNo,long  Data_Word)
{
 unsigned int hi, lo;

 switch (AxisNo)
 {
  case 1:  SelectMaskHi=0x83;  break; /*1000,0011*/
  case 2:  SelectMaskHi=0x85;  break; /*1000,0101*/
  case 3:  SelectMaskHi=0x89;  break; /*1000,1001*/
  case 4:  SelectMaskHi=0x91;  break; /*1001,0001*/
  case 5:  SelectMaskHi=0xA1;  break;/*1010,0001*/
  case 6:  SelectMaskHi=0xC1;  break;        /*1100,0001*/
 }

 outp(IO_Card+3,0x80);                    Sleep(0);
 outp(IO_Card+1,0x81);                    Sleep(0);
 outp(IO_Card+2,0x80);                    Sleep(0);
 hi=Data_Word>>8;
 hi=hi & 0x00ff;
 outp(IO_Card+0,hi);                      Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
 outp(IO_Card+1,SelectMaskHi & 0xFE);          Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
 outp(IO_Card+1,0x81);                    Sleep(0);
 lo=Data_Word & 0x00ff;
 outp(IO_Card+0,lo);                      Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
 outp(IO_Card+1,SelectMaskHi & 0xFE);          Sleep(0);
 outp(IO_Card+1,SelectMaskHi);            Sleep(0);
```

```
 outp(IO_Card+1,0x81);                    Sleep(0);

}

/* ******************
*                  BRAKE AND RELAY CARD COMMANDS                  *
************** */

/* ***************
BrakesInit()

USAGE
sets all the brakes to ON, (Default Value)BrakeMask is used
to maintain some brakes ON, or OFF while at the same time
switching other brakes or relays ON or OFF.

INPUTS                          OUTPUTS
none                    none

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

BrakesInit()
{
int i;

 outp(Brake,0);
 BrakeMask=0;

}


/* ***************
ApplyBrake(int AxisNo)

USAGE
ums ON the brake corresponding to AxisNo
axis no 7 corresponds to opening/closing the gripper

INPUTS                                  OUTPUTS
the joint number                  none

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

ApplyBrake(int AxisNo)
{

 switch (AxisNo)
 {
  case 0: BrakeMask=0;                    break;
  case 1: BrakeMask=BrakeMask & 0xFE;   break;    /*1111,1110*/
  case 2: BrakeMask=BrakeMask & 0xdf;   break;    /*1111,1101*/
```

```
   case 3: BrakeMask=BrakeMask & 0xfb;   break;      /*1111,1011*/
   case 4: BrakeMask=BrakeMask & 0xf7;   break;      /*1111,0111*/
   case 5: BrakeMask=BrakeMask & 0xEF;   break;      /*1110,1111*/
/*  case 6:      BrakeMask=BrakeMask & 0xDF;  break; */      /*1101,1111*/
   case 7: BrakeMask=BrakeMask & 0xBF;   break;      /*1011,1111*/
   case 8: BrakeMask=BrakeMask & 0x7F;   break;      /*0111,1111*/
  }

  outp(Brake,BrakeMask);

}


/* ********************
ReleaseBrake(int AxisNo)

USAGE
turns OFF the brake corresponding to AxisNo
axis no 7 corresponds to opening/closing the gripper

INPUTS                              OUTPUTS
the joint number              none

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

ReleaseBrake(int AxisNo)
{

 switch (AxisNo)
 {
  case 0:/*      BrakeMask=0xff;                  break;   */
  case 1: BrakeMask=BrakeMask | 0x01;   break;      /*0000,0001*/
  case 2: BrakeMask=BrakeMask | 0x20;   break;      /*0000,0010*/
  case 3: BrakeMask=BrakeMask | 0x04;   break;      /*0000,0100*/
  case 4: BrakeMask=BrakeMask | 0x08;   break;      /*0000,1000*/
  case 5: BrakeMask=BrakeMask | 0x10;   break;      /*0001,0000*/
/*  case 6: BrakeMask=BrakeMask | 0x20;  break; */  /*0010,0000*/
  case 7: BrakeMask=BrakeMask | 0x40;   break;      /*0100,0000*/
  case 8: BrakeMask=BrakeMask | 0x80;   break;      /*1000,0000*/
  }

 outp(Brake,BrakeMask);

}

/* ********************
*                    HARDWARE RESET COMMAND                    *
********************* */


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

```
Hdw_Reset(int AxisNo)

USAGE
resets LM628 chip number (AxisNo)

INPUTS                          OUTPUTS
the joint number          none

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

Hdw_Reset(int AxisNo)
{
 unsigned char HdwResetMask;

 outp(IO_Card+3,0x80);

 switch (AxisNo)
 {
  case 0:  HdwResetMask= 0x00;  broak;      /*0000,0000*/
  case 1:  HdwResetMask= 0x7c;  break;      /*0000,0001*/
  case 2:  HdwResetMask= 0x7a;  break;      /*0000,0010*/
  case 3:  HdwResetMask= 0x76;  break;      /*0000,0100*/
  case 4:  HdwResetMask= 0x6e;  break;      /*0000,1000*/
  case 5:  HdwResetMask= 0x5e;  break;      /*0001,0000*/
  case 6:  HdwResetMask= 0x3e;  break;      /*0010,0000*/
  }

  outp(IO_Card+2,0x7e);              Sleep(0);
  outp(IO_Card+2,HdwResetMask);      Sleep(0);
  outp(IO_Card+2,0x7e);
  RdStat(1);

}

/* ********************
*                  DATA REPORTING COMMANDS                  *
********************* */

/* ***************
Rdlp(int AxisNo)

USAGE
is used to read the index position recorded in the index
register using Slp()

INPUTS                          OUTPUTS
the joint number          none

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

Rdlp(int AxisNo)
```

```
{

 Write_Command(AxisNo,0x09);
 Read_Data(AxisNo);
 Index_Pos1=Data_Word;
 Read_Data(AxisNo);
 Index_Pos2=Data_Word;

}

/* ***************
Rddp(int AxisNo)

USAGE
reads the instantaneous desired position output of the profile generator.

INPUTS                          OUTPUTS
the joint number          none

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

Rddp(int AxisNo)
{

 Write_Command(AxisNo,0x08);
 Read_Data(AxisNo);
 Dpos1=Data_Word;
 Read_Data(AxisNo);
 Dpos2=Data_Word;
}

/* ***************
Rdrp(int AxisNo)

USAGE
reads the current actual position of the motor.

INPUTS                          OUTPUTS
the joint number          none

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

Rdrp(int AxisNo)
{
 Write_Command(AxisNo,0x0A);
 Read_Data(AxisNo);
 Rpos1=Data_Word;                Sleep(0);
 Read_Data(AxisNo);
 Rpos2=Data_Word;

}
```

```
/* ***************
Rddv(int AxisNo)

USAGE
reads the integer and fractional portions of the instantaneous desired velocity.

INPUTS                          OUTPUTS
the joint number          none

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

Rddv(int AxisNo)
{

  Write_Command(AxisNo,0x07);
  Read_Data(AxisNo);
  Dvel1=Data_Word;
  Read_Data(AxisNo);
  Dvel2=Data_Word;
}

/* ***************
Rdrv(int AxisNo)

USAGE
reads the integer portion of the instantaneous actual velocity of the motor.

INPUTS                          OUTPUTS
the joint number          none

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

Rdrv(int AxisNo)
{
  Write_Command(AxisNo,0x0B);
  Read_Data(AxisNo);
  Rvel=Data_Word;
}

/* ***************
Rdsum(int AxisNo)

USAGE
reads the value to which the integration sum has accumulated.

INPUTS                          OUTPUTS
the joint number          none

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

```
Rdsum(int AxisNo)
{

  Write_Command(AxisNo,0x0d);
  Read_Data(AxisNo);
  Sum=Data_Word;
}

/* ***************
Rdsigs(int AxisNo)

USAGE
reads the internal signals register of the LM628.

 Bit Number       Function

   15      Host interrupt
   14      Acceleration loaded but not updated
   13      UDF executed but filter not yet updated
   12      Forward direction
   11      Velocity mode
   10      On target
    9      Turn off upon excessive position error
    8      8-bit output mode
    7      Motor off
    6      Breakpoint reached (Interrupt)
    5      Excessive position error (interrupt)
    4      Wraparound occurred (Interrupt)
    3      Index pulse acquired (Interrupt)
    2      Trajectory complete (Interrupt)
    1      Command error (Interrupt)
    0      Acquire next index (SIP executed)


INPUTS                          OUTPUTS
the joint number          none

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

Rdsigs(int AxisNo)
{

  Write_Command(AxisNo,0x0c);
  Read_Data(AxisNo);
  Signal_Register=Data_Word;
}

/* ******************
*                    INITALISATION COMMANDS                    *
*************** */
```

```
/* ***************
SfwReset(int AxisNo)

USAGE
has the same effect as the hardwired RST input. The following data
items are set to zero : filter coefficients and their input buffers,
trajectory parameters and their input buffers, and the motor control
output (80 Hex for 8-bit DAC). All interrupt masks are cleared, except
for Bit 6 (SBPA/SBPR). The output port size is set to 8-bits, and
the current, absolute position is defined as home.

INPUTS                          OUTPUTS
the joint number        none

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

SfwReset(int AxisNo)
{
  Write_Command(AxisNo,0x00);
}

/* ***************
Dfh(int AxisNo)

USAGE
declares the current position as home or absolute zero.

INPUTS                          OUTPUTS
the joint number        none

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

Dfh(int AxisNo)
{
  Write_Command(AxisNo,0x02);
}


/* ******************
*                INTERRUPT CONTROL COMMANDS                    *
************** */

/* ***************
Sip(int AxisNo)

USAGE
is used to record the position corresponding to the occurrence of
the next index pulse.

INPUTS                          OUTPUTS
the joint number        none
```

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

Slp(int AxisNo)
{
  Write_Command(AxisNo,0x03);
}

/* ********************
Lpei(int AxisNo, unsigned int pei)

USAGE
is used to load a threshold for position error detectior
Excessive position error will cause the correspondi        errupt to be set high.

INPUTS                                                    OUTPUTS
the joint value and the position error index              none

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

Lpei(int AxisNo, unsigned int pei)
{
  Write_Command(AxisNo,0x1b);
  Write_Data(AxisNo,pei);
}

/* ***************
Lpei(int AxisNo, unsigned int pei)

USAGE
loads a value of position error which will cause the motor to be turned off.

INPUTS                                                    OUTPUTS
the joint value and the position error index              none

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

Lpes(int AxisNo, unsigned int pes)
{
  Write_Command(AxisNo,0x1a);
  Write_Data(AxisNo,pes);
}


/* ***************
Sbpr(int AxisNo, long bpr)

USAGE
is used to set a breakpoint in terms of relative position.
The sum of the breakpoint value and the target position must
remain within the absolute position range of the system.
```

```
INPUTS                                                    OUTPUTS
the joint value and the break point                       none

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

Sbpr(int AxisNo, long bpr)
{
 int bpr1, bpr2;

 bpr1=bpr>>16;
 bpr=bpr<<16;
 bpr2=bpr>>16;
 Write_Command(AxisNo,0x21);
 Write_Data(AxisNo,bpr1);
 Write_Data(AxisNo,bpr2);
}

/* **************
Sbpa(int AxisNo, long bpa)

USAGE
is used to set a breakpoint in terms of absolute position.

INPUTS                                                    OUTPUTS
the joint value and the break point                       none

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

Sbpa(int AxisNo, long bpa)
{
 int bpa1, bpa2;

 bpa1=bpa>>16;
 bpa=bpa<<16;
 bpa2=bpa>>16;
 Write_Command(AxisNo,0x20);
 Write_Data(AxisNo,bpa1);
 Write_Data(AxisNo,bpa2);
}

/* **************
Mski(int AxisNo, unsigned int Mask)

USAGE
is used to disable specified interrupts to the host processor.
Instead of using the hardwired interrupt signal, this program checks
the status byte to determine interrupt conditions.

Bit Number      Function
```

```
   15    Not used
   14    Not used
   13    Not used
   12    Not used
   11    Not used
   10    Not used
    9    Not used
    8    Not used
    7    Not used
    6    Breakpoint interrupt
    5    Position error interrupt
    4    Wraparound interrupt
    3    Index pulse interrupt
    2    Trajectory complete interrupt
    1    Command error interrupt
    0    Not used

INPUTS                                  OUTPUTS
th joint number and mask          none

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

Mski(int AxisNo, unsigned int Mask)
{
  Write_Command(AxisNo,0x1c);
  Write_Data(AxisNo,Mask);
}

/* **********
Rstl(int AxisNo, unsigned int Reset_Word)

USAGE
is used to reset specified interrupt flags.

INPUTS                                      OUTPUTS
the joint number and reset word          none

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

Rstl(int AxisNo, unsigned int Reset_Word)
{

  Write_Command(AxisNo,0x1d);
  Write_Data(AxisNo,Reset_Word);
}

/* **********
*                  FILTER CONTROL COMMANDS                    *
********** */
```

```
/* **************
Lfil(int AxisNo, int kp, int ki, int kd, int il, int fil)

USAGE
loads the specified filter parameters and filter control word, and
writes them to the LM628.

 Bit Number       Function

   15      Bit 7 of derivative sampling interval
   14      Bit 6 of derivative sampling interval
   13      Bit 5 of derivative sampling interval
   12      Bit 4 of derivative sampling interval
   11      Bit 3 of derivative sampling interval
   10      Bit 2 of derivative sampling interval
    9      Bit 1 of derivative sampling interval
    8      Bit 0 of derivative sampling interval
    7      Not used
    6      Not used
    5      Not used
    4      Not used
    3      Loading kp data
    2      Loading ki data
    1      Loading kd data
    0      Loading il data

INPUTS
the joint number filter parameters and the filter control

OUTPUTS
none

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

Lfil(int AxisNo, int kp, int ki, int kd, int il, int fil)
{
 int kp1, kp2, ki1, ki2, kd1, kd2, il1, il2, fil1, fil2;


 Write_Command(AxisNo,0x1e);
 Write_Data(AxisNo,fil);

 RdStat(1);
  Write_Data(AxisNo,kp);
 RdStat(1);
  Write_Data(AxisNo,ki);
 RdStat(1);
  Write_Data(AxisNo,kd);
 RdStat(1);
  Write_Data(AxisNo,il);
 RdStat(1);
```

```
}

/* **************
Udf(int AxisNo)

USAGE
updates the filter parameters which were loaded with
Lfil(int AxisNo, int kp, int ki, int kd, int il, int fil)

INPUTS                                  OUTPUTS
the joint number                        none

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

Udf(int AxisNo)
{
  Write_Command(AxisNo,0x04);
}

/* ****************
*                    TRAJECTORY CONTROL COMMANDS                    *
***************** */

/* **************
Stt(int AxisNo)

USAGE
is used to execute the trajectory which was loaded with Ltrj(int .....)

INPUTS                                  OUTPUTS
the joint number                        none

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

Stt(int AxisNo)
{
  Write_Command(AxisNo,0x01);
}


/* **************
Ltrj(int AxisNo, float position, int velocity, int acceleration,
    unsigned int traj)

USAGE
loads the specified trajectory parameters and control word, and
writes them to the LM628.

 Bit Number      Function

   15      Not used
```

```
  14    Not used
  13    Not used
  12    Forward direction (Velocity mode only)
  11    Velocity mode
  10    Stop smoothly
   9    Stop abruptly
   8    Turn off motor
   7    Not used
   6    Not used
   5    Acceleration will be loaded
   4    Acceleration data is relative
   3    Velocity will be loaded
   2    Velocity data is relative
   1    Position will be loaded
   0    Position data is relative

INPUTS
the joint number, velocity, acceleration, position and trajectory word

OUTPUTS
none

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

Ltrj(int AxisNo, float position, int velocity, int acceleration, unsigned int traj)

{
 unsigned int a1, a2, v1, v2;
 long   p1, p2;
 long factor=65536;
 double Root;
 double Square;
 long a    l, vel, pos;
 float Fclk2, Factor2, Res2;
 float quot=0.0004096;
 Square=pow(quot,2);
 Res2=(float)res/1000;
 Factor2=(float)factor/1000;
 Fclk2=(float)fclk/1000000;


 pos=(long)(position*(float)res);
 p1=pos>>16;
 p1= p1 & 0x0000ffff;
 p2=pos & 0x0000ffff;

 vel=(long)( velocity*Res2*2048*Factor2/(60*Fclk2) );
 v1=vel >> 16;
 v1=v1 & 0x0000ffff;
 v2=vel & 0x0000ffff;
```

```
accel=(long)( acceleration*res*factor*Square );
a1=accel>>16;
a1= a1 & 0x0000ffff;
a2=accel & 0x0000ffff;

Write_Command(AxisNo,0x1f);
Write_Data(AxisNo,traj);
Write_Data(AxisNo,a1);
Write_Data(AxisNo,a2);
Write_Data(AxisNo,v1);
Write_Data(AxisNo,v2);
Write_Data(AxisNo,p1);
Write_Data(AxisNo,p2);

}
```

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

LtrjPLS(int AxisNo, long position, int velocity, int acceleration,
unsigned int traj)

USAGE
loads the specified trajectory parameters, control words, and writes them to the LM628. HOWEVER it passes the position variables in pulses counts and NOT in revolutions. NB NB nb!

Bit Number	Function	
15	Not used	
14	Not used	
13	Not used	
12	Forward direction (Velocity mode only)	
11	Velocity mode	8
10	Stop smoothly	4
9	Stop abruptly	2
8	Turn off motor	1 —
7	Not used	8
6	Not used	4
5	Acceleration will be loaded	2 or 3
4	Acceleration data is relative	1 -----1-----
3	Velocity will be loaded	8 or f
2	Velocity data is relative	4 or 7
1	Position will be loaded	2 or 3
0	Position data is relative	1 HOOho maybe this

INPUTS
the joint number, velocity, acceleration, positi~ and trajectory word

```
OUTPUTS
none

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

/*older version which works*/

LtrjPLS(int AxisNo, long position, int velocity, int acceleration, unsigned int traj)
{
 unsigned int a1, a2, v1, v2;
 long   p1, p2;
 int trajx;

 long factor=65536;
 double Root;
 double Square;
 long accel, vel, pos;
 float Fclk2, Factor2, Res2;
 float quot=0.0004096;

 Square=pow(quot,2);
 Res2 = (float)res/1000;
 Factor2 = (float)factor/1000;
 Fclk2 = (float)fclk/1000000;


        pos  = position;  /*NOTE THE DIFFERENCE IN THIS LINE!!*/
        p1=pos>>16;
        p1= p1 & 0x0000ffff;
        p2=pos & 0x0000ffff;

        vel=(long)( velocity*Res2*2048*Factor2/(60*Fclk2) );
        v1=vel >> 16;
        v1=v1 & 0x0000ffff;
        v2=vel & 0x0000ffff;

        accel=(long)( acceleration*res*factor*Square );
        a1=accel>>16;
        a1= a1 & 0x0000ffff;
        a2=accel & 0x0000ffff;

        Write_Command(AxisNo,0x1F);/* Load trajectory paramaters*/
        Write_Data(AxisNo,traj);      /* Load trajectory control word*/

        /*The trajectory control word can in some cases specify only,
          one or two paramters to download... thus the case statement*/

          /*
          Write_Data(AxisNo,a1);
          Write_Data(AxisNo,a2);
          Write_Data(AxisNo,v1);
```

```
        Write_Data(AxisNo,v2);
        Write_Data(AxisNo,p1);
        Write_Data(AxisNo,p2);
        */


      trajx = traj;
      trajx = traj & 0x0000ffff;
      if ((trajx | 0x20)==trajx)
        {  Write_Data(AxisNo,a1); Write_Data(AxisNo,a2); }
      if ((trajx | 0x08)==trajx)
        { Write_Data(AxisNo,v1);  Write_Data(AxisNo,v2); }
      if ((trajx | 0x02)==trajx)
        { Write_Data(AxisNo,p1);  Write_Data(AxisNo,p2); }

}
```

MASTER.C

```
/*  this is the master program which controls the actions and
    motion of the robot in a sequential manner. All the necessary
    sub-routines are linked as form of a project.
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graph.h>
#include"c:\masters\proi\ext_vars.h"
/* #include"c:\masters\proj\fnproto.h"*/

void modes();
test();

main()
{

ClearScreen();
Init_Robot();
Menu1();

}
```

MOVEHOME.C

```
# include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "c:\masters\proj\ext_vars.h"
#include"c:\masters\proj\fnproto.h"

#define ON  0
#define OFF 1

/*function prototypes*/
void Microswitches(void);
int Int2Bin(int number, int Array[8]);
void Sample(void);
MoveNeg(int AxisNo);
MovePos(int AxisNo);
Stop(int N);

int AxisNo, i, N;
int IO_addr = 0x300;
int Bin1[8],Bin2[8],Bin3[8];
int SW[15];



/* **************
MveHme();

USAGE
moves the robot to the home position

INPUTS
the current robot position

OUTPUTS
the record of joint angles is set to the home position angles

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

MveHme()
{
char i;
extern char LimitFlag;

SetCursor(ACTIVITY,2);
printf("Moving Robot Home");

RoughPos();
if(LimitFlag==FALSE)
```

```
  {
  Microswitches();
  for(i=1;i<=5;i++)
    AbsAng[i]=rHome[i];
  }

Update();
}

/* **************
RoughPos()

USAGE
moves the robot approximately to the home position

INPUTS                                    OUTPUTS
the current joint angles          none

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

RoughPos()
{
float T[6], Diff[6];
int Speed=250, i, HomeFlag=FALSE;

SetCursor(ACTIVITY,2);
printf("Moving Robot Roughly to Home Position");

for(i=1;i<=5;i++)
  Diff[i]=fabs(AbsAng[i]-Home[i]);

if(Diff[1]<5 && Diff[2]<5 && Diff[3]< ' && Diff[4]<5 && Diff[5]<5 )
    HomeFlag=TRUE;

if(HomeFlag!=TRUE)
  {

  for(i=1;i<=5;i++)
    T[i]=-1*AbsAng[i]+Home[i];

  SetCursor(RESULTS1,2);
  printf("Moving robot home by : %0.1f  %0.1f  %0.1f  %0.1f  %0.1f ?",
    T[1],T[2],T[3],T[4],T[5]);

    MoveRobot(T,TRK4);
  }

SetCursor(ACTIVITY,2);
printf("                                  ");
SetCursor(RESULTS1,2);
printf("                                  ");
```

```
}

/* **************
void Microswitches(void)

USAGE
places the robot in the home position accurately using the robot's
microswitches

INPUTS                  OUTPUTS
none                none

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

void Microswitches(void)
{

/*Initialise the LM628's*/

SetCursor(ACTIVITY,2);
printf("Moving Robot to Home Position using Microswitches");

 for(AxisNo=1;AxisNo<=5;AxisNo++)
   {
   SfwReset(AxisNo);
   Sbpa(AxisNo,0x7FFF);
   Rsti(AxisNo,0x10);
   Lfil(AxisNo,15,1,10,1000,0x080F);
   Udf(AxisNo);
   }

/*=========================
 HOME THETA/ANY AXIS
=========================*/

Sample();

for( N=1;N<=5;N++)      /*N = Axis number, SW = Home, Right, Left M/Switch*/
  {
  if(SW[(N-1)*3+0]==ON)          //home limit switch reached
    {
    MovePos(N);
    do
        {       Sample();  }
    while( SW[(N-1)*3+0]==ON );
    Sleep(1500);
    Stop(N);
    Sleep(100);
    MoveNeg(N);
```

```
    do
        {       Sample();       }
    while( SW[(N-1)*3+0]==OFF );
    Stop(N);
    }

  if( SW[(N-1)*3+0]==OFF && SW[(N-1)*3+1]==OFF && SW[(N-1)*3+2]==OFF )
                /*All switches open, IE-IN BETWEEN LIMITS*/
    {
    MovePos(N);

    do                          //until home or pos limit is reached
    { Sample(); }
    while( SW[(N-1)*3+2]==OFF && SW[(N-1)*3+0]==OFF );

    Stop(N);

    if(SW[(N-1)*3+2]==ON)       //+ve limit switch reached
        {
        MoveNeg(N);
        do
         { Sample();  }
        while( SW[(N-1)*3+0]==OFF );
        Stop(N);
        }
    else
        {
        if(SW[(N-1)*3+0]==ON)           //home limit switch reached
          {
          MovePos(N);
          do
           {    Sample(); }
          while( SW[(N-1)*3+0]==ON );
          Sleep(1500);
          Stop(N);
          Sleep(100);
          MoveNeg(N);

          do
           {    Sample();       }
          while( SW[(N-1)*3+0]==OFF );
          Stop(N);
          }
        }
    }
  }

SetCursor(ACTIVITY,2);
printf("                                    ");

}
```

```
/* **************
MovePos(int AxisNo)

USAGE
moves a link in a positive rotation

INPUTS                    OUTPUTS
 none                   none

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

MovePos(int AxisNo)
{
int traj;

SfwReset(AxisNo);
Sbpa(AxisNo,0x7FFF);
Rstl(AxisNo,0x10);
Lfil(AxisNo,15,1,10,1000,0x080F);
Udf(AxisNo);

if(AxisNo==1 || AxisNo==3 || AxisNo==4)
  traj=0x083C;
if(AxisNo==2 || AxisNo==5)
  traj=0x183C;

Ltrj(AxisNo,0,200,10,traj); ReleaseBrake(AxisNo);  Stt(AxisNo);
}

/* **************
MoveNeg(int AxisNo)

USAGE
moves a link in a negitive rotation

INPUTS                    OUTPUTS
 none                   none

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

MoveNeg(int AxisNo)
{
int traj;

SfwReset(AxisNo);
Sbpa(AxisNo,0x7FFF);
Rstl(AxisNo,0x10);
Lfil(AxisNo,15,1,10,1000,0x080F);
Udf(AxisNo);
```

```
if(AxisNo==1 || AxisNo==3 || AxisNo==4)
  traj=0x183C;
if(AxisNo==2 || AxisNo==5)
  traj=0x083C;

Ltrj(AxisNo,0,200,10,traj); ReleaseBrake(AxisNo);  Stt(AxisNo);
}

/* ******************************
Stop(int N)

USAGE
stops a link from rotating

INPUTS                          OUTPUTS
the link number          none

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

Stop(int N)
{
Ltrj(N,0,77,10,0x0200); Stt(N); ApplyBrake(N);     //stop
SfwReset(N);
}


/* ******************************
int Int2Bin(int number, int Array[])

USAGE
converts a decimal number to a binary number

INPUTS                                  OUTPUTS
the decimal number          the binary number

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

int Int2Bin(int number, int Array[])
{
 int j;
 unsigned int mask;
 char choice;

    mask=0x80;
    for(j=1;j<=8;j++)
    {
        Array[9-j]=(mask & number) ? 1 : 0;
        mask>>=1;
    }
}
```

```
/* ****************************
void Sample(void)

USAGE
samples the status of the microswitches to see which are on and which are
off

INPUTS                    OUTPUTS
none                the status of the microswitches

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

void Sample(void)
{
int Io_Addr=0x300;
int a, b, temp;

outp(Io_Addr+3,0x9b);
outp(Io_Addr+3,0xb9);
outp(Io_Addr+3,0xff);
outp(Io_Addr+3,0x10);
outp(Io_Addr+3,0x9b);

a = inp(Io_Addr);
b = inp(Io_Addr+1);
Int2Bin(a,Bin1);
Int2Bin(b,Bin2);


SW[0]  = Bin1[1];  SW[1]  = Bin1[2];   SW[2]  = Bin1[3];
SW[3]  = Bin1[4];  SW[4]  = Bin1[6];   SW[5]  = Bin1[5];
SW[6]  = Bin2[1];  SW[7]  = Bin1[7];   SW[8]  = Bin1[8];
SW[9]  = Bin2[2];  SW[10] = Bin2[4];   SW[11] = Bin2[3];
SW[12] = Bin2[5];  SW[14] = Bin2[6];   SW[13] = Bin2[7];

 /*Not in sequence due to different wiring order !*/

temp=SW[5];
SW[5]=SW[4];
SW[4]=temp;

temp=SW[7];
SW[7]=SW[8];
SW[8]=temp;

}
```

```
/*

configuration of limit switches..

neg  home  pos
  ----> positive direction of rotation

** where if robot is moving from home in a positive direction the pos
   microswitch will switch on as the robot joint is reached

** where if robot is moving from home in a negetive direction the neg
   microswitch will switch on as the robot joint is reached

SW[0] =    home ms joint 1
SW[1] =    neg ms  joint 1
SW[2] =    pos ms  joint 1

SW[3] =    home ms joint 2
SW[4] =    pos ms  joint 2
SW[5] =    neg ms  joint 2

SW[6] =    home ms joint 3
SW[7] =    pos ms  joint 3
SW[8] =    neg ms  joint 3

SW[9] =    home ms joint 4
SW[10] =   neg ms  joint 4
SW[11] =   pos ms  joint 4

SW[12] =   home ms joint 5
SW[13] =   neg ms  joint 5
SW[14] =   pos ms  joint 5

*/
```

RASND.C

```
/*
 * returns the distance from the ultrasonic sensor to the
 * nearest object in front of it
 */

#include "c:\masters\proj\edr.h"          /* driver functions */
#include "c:\masters\proj\ext_vars.h"
//  #include"c:\masters\proj\fnproto.h"

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

/*function prototypes*/
void InitADBoard();
int UltraSndSensor();
void ReleaseADBoard(void);


int bh;                  /* our board handle */
int boardtype;           /* our board type */



void InitADBoard()
{
int baseaddr=0x700, t;

/* first get a board handle to use */
bh=EDR_AllocBoardHandle();
if (!bh)  /* this should never happen with DOS */
        {
        printf("No free board handles\n");
        exit(1);
        }

/* initialize the board at the specified base address */
t=EDR_InitBoardType(bh,baseaddr,PC26);
if (t) printerror(t);

}


int UltraSndSensor()
{
int baseaddr, err,tp,c,t, Counts, NoCounts=10;
char s[80], choice;
```

```
int Channel=2, r;          /*remember to use the correct channel !!!*/
long uvolts;
float Ratio=8.10;          /*8.10mV/mm*/
float Output, Total=0;

/* display the voltage on channel 2  */

for(Counts=0;Counts<NoCounts;Counts++)
  {
  /*reading ultrasound channel*/
  r=EDR_ADInOneVoltage(bh,Channel,&uvolts);
  if (r<0)
    printerror(r);
  else
    Output = ( (float)uvolts/1000 )/Ratio;

  Total+=Output;
  }

Dist=Total/(float)NoCounts;


}


void ReleaseADBoard(void)
{
/* release the board handle (not necessary for DOS) */
EDR_FreeBoardHandle(bh);
}


printerror(int r)
/* displays error msg and exits */
{
char s[80];
EDR_StrError(r,s);        /* convert error number into a string */
SetCursor(ACTIVITY,2);
printf("%s\n",s);
exit(1);
}
```

UTILS.C

```
/*file utilities*/


#include<stdio.h>
#include "c:\masters\proj\ext_vars.h"
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <conio.h>
#include <graph.h>
#include"c:\masters\proj\fnproto.h"


# define  OPEN  0
# define  CLOSED  1

/*function prototypes
void Write(void);
void Read(void);
void Menu1(void);
void TestWhatever(void);
void PickandPlace(void);
void Pick(void);
void Place(void);
void Pause(void);
void Quit(void);
void CalibrateCamera(void);
void FindDims(void);
void CloseHand(void);
void OpenHand(void);
void HandleInput(void);
void Update(void);
void AlignCamera(void);
Errors(int ErrorNo);
void MoveHome(void);
void Pendant(void);
MoveZAxis(float ZDist);
void ConstVelo(void);
void Step(void);
void IntPosChoice(void);
void AlgnTlCentChoice(void);
void MoveZAxisUser(void);
void Translate(void);
void YPR1(float Yaw, float Pitch, float Roll);
void CallWS3(void);
void ClearText(void);
```

```
*/

int HandFlag=OPEN;
float Rot[4][4];
int NoChoices1=6, NoChoices2=11, left=50, max;

/* ******************************
void Write(void)

USAGE
writes the joint angles to a text file

INPUTS                                    OUTPUTS
none                              the current joint angles

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

void Write(void)
{
FILE *StreamOut;
int i, numread, numwritten;
int Q[6];

SetCursor(ACTIVITY,2);
printf("Writing Robot joint Angles to file");

StreamOut = fopen("c:\\masters\\proj\\J_Angles.out","w+t");
fprintf(StreamOut, "%f %f %f %f %f",
   AbsAng[1],AbsAng[2],AbsAng[3],AbsAng[4],AbsAng[5]);
fclose(StreamOut);

SetCursor(ACTIVITY,2);
printf("                                  ");

}


/* ******************************
void Read(void)

USAGE
reads the joint angles from the text file from the last session

INPUTS                          OUTPUTS
none                      the joint angles

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

void Read(void)
{
FILE *StreamIn;
```

```
int list[6];
int i, numread, numwritten;

SetCursor(ACTIVITY,2);
printf("Reading Robot joint Angles from J_Angles.out");


StreamIn = fopen("c:\\masters\\proj\\J_Angles.out","r+t");
fscanf(StreamIn, "%f %f %f %f %f",
    &AbsAng[1],&AbsAng[2],&AbsAng[3],&AbsAng[4],&AbsAng[5]);
fclose(StreamIn);

}



/* ******************************
void Quit(void)

USAGE
switches off the camera and ultrasound cards

INPUTS                    OUTPUTS
none                  calls the read module

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

void Quit(void)
{
int AxisNo;

Write();
Sleep(500);
if(CameraFlag==1)
  DisableDriver();
ReleaseADBoard();

for(AxisNo=1;AxisNo<=5;AxisNo++)
  SfwReset(AxisNo);
BrakesInit();
ClearScreen();

exit(0);
}


/* ********** ********** ******
void CalibrateCamera(void)

USAGE
```

```
calibrates the camera

INPUTS                                  OUTPUTS
none                                    none

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

void CalibrateCamera(void)
{
char choice[6];


printf("\npress 'y' to turn on camera :");
choice[0] = getche();
if(choice[0]=='y')
   TurnOnCamera();

do
  {
    SetCursor(22,2);
    printf("press 'y' to unfreeze video :");
    choice[1] = getche();
    if(choice[1]=='y')
         InputHandler(FREEZE);

    SetCursor(CHOICE,2);
    printf("Press 'y' to find pix dists :");
    choice[2] = getche();
    if(choice[2]=='y')
         InputHandler(CENTROID);

    PrintEdges('h');
    PrintEdges('v');

    SetCursor(RESULTS1,2);
    printf("OffX=%d pix  OffY=%d pix  ObjWd=%d pix  ObjHt=%d pix",
         OffsetX, OffsetY, ObjWd, ObjHt);

    SetCursor(CHOICE,2);
    printf("Press 'y' to repeat :");
    choice[3] = getche();

    InputHandler(FREEZE);

  }
while(choice[3]=='y');

DisableDriver();
}
```

```
/* ****************************
void FindDims(void)

USAGE
finds the number of mm of an object in the horizontal and
vertical directions in order to check camera calibration

INPUTS
the distance of the object from the camera

OUTPUTS
the dimension of the object

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

void FindDims(void)
{
char choice[6];
float  PixPermmX, PixPermmY;
int dec;

printf("\npress 'y' to turn on camera :");
choice[0] = getche();
if(choice[0]=='y')
   TurnOnCamera();

do
  {
    SetCursor(22,2);
    printf("press 'y' to unfreeze video :");
    choice[1] = getche();
    if(choice[1]=='y')
         InputHandler(FREEZE);

    SetCursor(CHOICE,2);
    printf("enter distance between target and camera  :");
         scanf("%d",&dec);

    SetCursor(CHOICE,2);
    printf("Press 'y' to find dims  :");
    choice[2] = getche();
    if(choice[2]=='y')
         InputHandler(CENTROID);

    SetCursor(RESULTS1,2);
    printf(" ObjWd=%d pix  ObjHt=%d pix", ObjWd, ObjHt);

    PixPermmX = ax*pow(dec,4) + bx*pow(dec,3) + cx*pow(dec,2)
              + dx*dec + ex;
    PixPermmY = ay*pow(dec,4) + by*pow(dec,3) + cy*pow(dec,2)
              + dy*dec + ey;
```

```
  OffsetX = (int)(OffsetX/PixPermmX);
  OffsetY = (int)(OffsetY/PixPermmY);
  ObjWd   = (int)(ObjWd/PixPermmX);
  ObjHt   = (int)(ObjHt/PixPermmY);

    SetCursor(RESULTS2,2);
    printf("ObjWd=%d mm  ObjHt=%d mm",  ObjWd, ObjHt);

    GetCursor(RESULTS3,2);
    printf("dist=%d mm ",dec);

    SetCursor(CHOICE,2);
    printf("Press 'y' to repeat :");
    choice[3] = getche();

    InputHandler(FREEZE);

  }
while(choice[3]=='y');

DisableDriver();
}


/* ******************************
void CloseHand(void)

USAGE
closes the gripper

INPUTS                          OUTPUTS
none                          none

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

void CloseHand(void)
{
ReleaseBrake(7);
HandFlag=CLOSED;
}

/* ******************************
void OpenHand(void)

USAGE
opens the gripper

INPUTS                          OUTPUTS
none                          none
```

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

void OpenHand(void)
{
ApplyBrake(7);
HandFlag=OPEN;
}

/* ******************************
void OpenCloseHnd(void)

USAGE
toggles the gripper open/close condition

INPUTS                                    OUTPUTS
none                                  none

****************************** */
void OpenCloseHnd(void)
{
if(HandFlag==CLOSED)
  OpenHand();
else
  if(HandFlag==OPEN)
    CloseHand();
}

/* ******************************
void Menu2(void)

USAGE
shows the second menu and obtains input

INPUTS                                    OUTPUTS
the user choice              the module called by the user

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

void Menu2(void)
{
int number=1, i;
char *String2[] = {
 " 1: Step();",
 " 2: Quit();",
 " 3: Pendant();",
 " 4: MoveHome();",
 " 5: AlignCamera",
 " 6: UnFreeze Image",
 " 7: Freeze Image",
 " 8: TestWhatever();",
 " 9: MoveZAxis();",
```

```
  " 10: OpenCloseHnd();",
  " 11: Main Menu",
  "  ",
  "  "
                    };

if(NoChoices1>NoChoices2)
   max=NoChoices1;
if(NoChoices1<NoChoices2)
   max=NoChoices2;

for(i=0;i<=max-1;i++)
     {
     SetCursor(i+2,left);
     printf("                ");
     }

while(number!=6 && number!=2)
   {
   SetCursor(1,left);
   printf("UTILITIES MENU");

   for(i=0;i<=NoChoices2-1,i++)
     {
     SetCursor(i+2,left);
     printf("%s",String2[i]);
     }

Update();

SetCursor(CHOICE,2);
printf("enter choice :");
scanf("%d",&number);

switch(number)
   {
   case 1:      Step();          break;
   case 2:      Quit();          break;
   case 3:      Pendant();               break;
   case 4:      MveHme();                break;
   case 5:      AlignCamera();           break;
   case 6:      UnFreeze();              break;
   case 7:      Freeze();                break;
   case 8:      TestWhatever();          break;
   case 9:      MoveZAxisUser();         break;
   case 10:     OpenCloseHnd();          break;
   case 11:     Menu1();                 break;
   }
   }

}
```

```
/* ****************************
void Menu1(void)

USAGE
shows the first menu and obtains input

INPUTS                                          OUTPUTS
the user choice                 the module called by the user

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

void Menu1(void)
{
int number, i;
char *String[] = {
  " 1: TurnOnCamera();",
  " 2: Run WS3 sim. pkg.",
  " 3: translate WS3 to C",
  " 4: run WS3 C file",
  " 5: Quit();",
  " 6: Utilities Menu",
  "                    "
                    };

if(NoChoices1>NoChoices2)
   max=NoChoices1;
if(NoChoices1<NoChoices2)
   max=NoChoices2;

for(i=0;i<=max-1;i++)
     {
     SetCursor(i+2,left);
     printf("                ");
     }

while(number!=5 && number!=6)
   {
   SetCursor(1,left);
   printf("MAIN MENU");

   for(i=0;i<=NoChoices1-1;i++)
     {
     SetCursor(i+2,left);
     printf("%s",String[i]);
     }

   Update();

   SetCursor(CHOICE,2);
```

```
  printf("Enter choice (n) :");
  scanf("%d",&number);

  switch(number)
    {
    case 1:     TurnOnCamera();     break;
    case 2:     CallWS3();          break;
    case 3:     Convert();          break;
    case 4:     RunWS3trk();        break;
    case 5:     Quit();       break;
    case 6:     Menu2();            break;
    }
  }
}


/* ******************************
void TestWhatever(void)

USAGE
tests whatever module the user wants to

INPUTS                              OUTPUTS
none                          none

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

void TestWhatever(void)
{

}


/* ******************************
void AlgnTlCentChoice(void)

USAGE

INPUTS

OUTPUTS

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

void AlgnTlCentChoice(void)
{
int a;

SetCursor(CHOICE,2);
```

```
printf("Align with cube<1> or hole<0>, enter choice :");
scanf("%d",&a);
AlignTool(a);

}

/* ****************************
void IntPosChoice(void)

USAGE

INPUTS

OUTPUTS

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

void IntPosChoice(void)
{
int a;

SetCursor(CHOICE,2);
printf("Intersect pos for cube<1> or hole<0>, enter choice :");
scanf("%d",&a);
IntersectPos(a);

}

/* ****************************
void TestPulses(void)

USAGE
determined the number of pulses per degree for each joint

INPUTS                                  OUTPUTS
none                                    none

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

void TestPulses(void)
{
float Ang, Pulse, A[6]={0,0,0,0,0,0};
int AxisNo, i, Times=10;
long Degs;
int AxisSwitch[6]={ 0 , -1 , 1 , -1 , -1 , -1 };
extern int Status_Array[8];
extern long     Signal_Register, Index_Pos1, Index_Pos2, Dpos1,
                Dpos2,  Dvel1, Dvel2, Rvel, Sum;

SetCursor(CHOICE,2);
printf("enter axis no :  ");
```

```
scanf("%d",&AxisNo);
SetCursor(CHOICE,2);
printf("enter no degrees:  ");
scanf("%f",&Ang);
SetCursor(CHOICE,2);
printf("enter no pulses:  ");
scanf("%f",&Pulse);

A[AxisNo]=Ang;

Degs = (long)( (float)Ang * (float)AxisSwitch[AxisNo] * (float)Pulse );
Degs=Degs/Times;


for(i=1;i<=Times;i++)
  {
  InitMove(A);
  LoadPrmsPLS(AxisNo, Degs);
    do
          { RdStat(AxisNo);  }
    while(Status_Array[2]!=1);
  ApplyBrake(AxisNo);
  Dfh(AxisNo);
  Rsti(AxisNo,0x7e);
  Sleep(1);
  }

BrakesInit();
SfwReset(AxisNo);
choice[2]=getche();

for(i=1;i<=Times;i++)
  {
  InitMove(A);
  LoadPrmsPLS(AxisNo, -Degs);
    do
          { RdStat(AxisNo);  }
    while(Status_Array[2]!=1);
  ApplyBrake(AxisNo);
  Dfh(AxisNo);
  Rsti(AxisNo,0x7e);
  Sleep(1);
  }

BrakesInit();
SfwReset(AxisNo);
}

/* ****************************
void ClearText(void)
```

```
USAGE
clears the text from the screen

OUTPUTS                                INPUTS
none                                   none

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

void ClearText(void)
{

/*clear the screen from all text*/

SetCursor(ABSANG-1,left-7);
printf("                              ");
SetCursor(ABSANG,left-7);
printf("                              ");
SetCursor(ABSANG+1,left-7);
printf("                              ");
SetCursor(ABSANG+2,left-7);
printf("                              ");
SetCursor(17,1);
printf("                              ");
SetCursor(18,1);
printf("                              ");
SetCursor(19,1);
printf("                              ");
SetCursor(20,1);
printf("                              ");
SetCursor(21,1);
printf("                              ");
SetCursor(22,1);
printf("                              ");
SetCursor(23,1);
printf("                              ");

}


/* ******************************
void Update(void)

USAGE
prints the tool orientation and position on the screen

INPUTS
the ultrasound distance and the joint angles

OUTPUTS
the current height above the bench (if above bench), the current tool
position and orientation
```

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

void Update(void)
{
int i, left=50;
float DumbAngles[6];

ClearText();

SetCursor(ABSANG-1,left-7);
printf("Q : (%0.1f,%0.1f,%0.1f,%0.1f,%0.1f)",
   AbsAng[1],AbsAng[2],AbsAng[3],AbsAng[4],AbsAng[5]);

for(i=1;i<=5;i++)
  DumbAngles[i]=AbsAng[i];

ToolPos(DumbAngles);

SetCursor(ABSANG,left-7);
printf("(X,Y,Z) : (%0.1f,%0.1f,%0.1f)", AbsPos[0],AbsPos[1],AbsPos[2]);

SetCursor(ABSANG+1,left-7);
printf("(Y,P,R) : (%0.1f,%0.1f,%0.1f)", Yaw, Pitch, Roll);

UltraSndSensor();
SetCursor(ABSANG+2,left-7);
printf("Distance =  %0.2f mm", Dist);

SetCursor(ABSANG+3,left-7);
if(FzeFlag==FROZEN)
  printf("Frozen Image");
if(FzeFlag==LIVE)
  printf("Live Image");

Write();
}


/* ******************************
void AlignCamera(void)

USAGE
places the tool above the bench, turns on the camera and allows the
camera to be aligned with the tool X and Y axes

INPUTS                    OUTPUTS
 none                   none

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

```
void AlignCamera(void)
{
#define  READ            0
#define  WRITE                   1
int TotX=10, TotY=10, i;
float P[4], R[4][4];

extern unsigned long WIDTH , HEIGHT;

SetCursor(CHOICE,2);
printf("move to align position ?");
choice[3]=getche();
if(choice[3]=='y')
  {
  Mve   ne();
  P[1]=513.539001;  P[2]=0;    P[3]=761.368225;
  R[1][1]=-1;    R[1][2]=0;       R[1][3]=0;
  R[2][1]=0;     R[2][2]=1;       R[2][3]=0;
  R[3][1]=0;     R[3][2]=0;       R[3][3]=-1;
  InvKinematics(P,R,TRK4);
  }

do
  {
  SetCursor(CHOICE,2);
  printf("press any key when ready :");
  choice[1] = getche();

  ReadORWriteBuffer(READ);

/*write vertical cross-hairs to screen*/
  for(i=1;i<=TotY;i++)
    {
    CsHairX[i] = WIDTH/2;
    CsHairY[i] = (i*HEIGHT)/(TotY+1);
    }

/*write horizontal cross-hairs to screen*/
  for(i=TotY+1;i<=TotX+TotY;i++)
    {
    CsHairX[i] = ((i-TotY)*WIDTH)/(TotX+1);
    CsHairY[i] = HEIGHT/2;
    }

  CsHairX[TotY+TotX+1] = WIDTH/2;     //put a cross-hair in the middle
  CsHairY[TotY+TotX+1] = HEIGHT/2;

  CrossHair(TotX+TotY+1,Black);

  ReadORWriteBuffer(WRITE);
```

```
  UltraSndSensor();
  SetCursor(RESULTS1,2);
  printf("Tool is %0.3f mm above bench ", Dist);

  SetCursor(CHOICE,2);
  printf("press 'y' to repeat :");
  choice[0] = getche();
  if(choice[0]=='y')
    UnFreeze();
  }
while(choice[0]=='y');

UnFreeze();
}


/* ****************************
Errors(int ErrorNo)

USAGE
prints an error message if the something goes wrong

INPUTS                                  OUTPUTS
an error number                 none

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

Errors(int ErrorNo)
{
char *ErrorMsg[]= {
                   " Problem with finding the centroid",
                   " Problem with Ultrasound",
                   " Domain error - cannot find joint angle 3",
                   " Robot limit reached",
                   " Cannot run WS3.exe",
                   " Cannot run Pendant.exe",
                   " Fatal error - Ultrasonic sensor not on.",
                   " FS200DRV driver not installed ",
                   " Translation from WS3 to C not working",
                   " Work envelope exceeded",
                   " Cannot move robot : joint 1 limit will be exceeded",
                   " Cannot move robot : joint 2 limit will be exceeded",
                   " Cannot move robot : joint 3 limit will be exceeded",
                   " Cannot move robot : joint 4 limit will be exceeded",
                   " Cannot move robot : joint 5 limit will be exceeded"
                   };

SetCursor(ACTIVITY,2);
printf("Error : %s ",ErrorMsg[ErrorNo] );
Beep(3000,150); Sleep(100);
Beep(3000,150); Sleep(100);
```

```
Beep(3000,150); Sleep(1500);
}


/* ******************************
void Pendant(void)

USAGE
calls the teach pendant developed by Ehud Ilios

INPUTS                                  OUTPUTS
none                                    none

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

void Pendant(void)
{
if(system("Pendant.exe")==-1)
   Errors(7);
}

/* ******************************
void CallWS3(void)

USAGE
calls the robot simulation package WS3

INPUTS                                  OUTPUTS
none                                    none

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

void CallWS3(void)
{

/*change to the WS3 directory*/
_chdir("c:\\WS3");

if(system("c:\\ws3\\WS3.exe")==-1)
   Errors(6);

/*change back to the working directory*/
_chdir("c:\\masters\\proj");

}

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

USAGE
```

```
INPUTS

OUTPUTS

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

void MoveZAxisUser(void)
{
float A;

SetCursor(CHOICE,2);
printf("Enter amount to move tool in Z Axis : ");
scanf("%f",&A);

MoveZAxis(A);
}


/* ******************************
void MoveXAxis(void)

USAGE
moves the tool in the robot base X axis keeping the same yaw, pitch and roll
angles

INPUTS                          OUTPUTS
none                        none

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

void MoveXAxis(float XDist)
{
float P[4];

P[1]=AbsPos[0] + XDist;
P[2]=AbsPos[1];
P[3]=AbsPos[2];

YPR1(Yaw,Pitch,Roll);
InvKinematics(P,Rot,NOTRK4);
Update();
}


/* ******************   *********
void MoveYAxis(void)

USAGE
moves the tool in the robot base Y axis keeping the same yaw, pitch and roll
angles
```

```
INPUTS                          OUTPUTS
none                            none

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

void MoveYAxis(float YDist)
{
float P[4];

P[1]=AbsPos[0];
P[2]=AbsPos[1] + YDist;
P[3]=AbsPos[2];

YPR1(Yaw,Pitch,Roll);
InvKinematics(P,Rot,NOTRK4);
Update();
}

/* ****************************
MoveZAxis(float ZDist)

USAGE
moves the tool in the robot base Z axis keeping the same yaw, pitch and roll
angles

INPUTS                          OUTPUTS
none                            none

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

MoveZAxis(float ZDist)
{
float P[4];

P[1]=AbsPos[0];
P[2]=AbsPos[1];
P[3]=AbsPos[2] + ZDist ;

YPR1(Yaw,Pitch,Roll);
InvKinematics(P,Rot,NOTRK4);
Update();

}

/* ****************************
void YPR1(float Yaw, float Pitch, float Roll)

USAGE
calculates the tool rotation matrix from the orientation of the tool
in yaw, pitch and roll angles
```

```
INPUTS                                                    OUTPUTS
the tool yaw, pitch and roll angles          the rotational matrix

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

void YPR1(float Yaw, float Pitch, float Roll)
{
float Cy,Cp,Cr,Sy,Sp,Sr;


/*printf("ypr1   yaw=%f   pitch=%f  roll=%f  ",Yaw,Pitch,Roll);*/

Cy=cos(Yaw/R_TO_D); Cp=cos(Pitch/R_TO_D);          Cr=cos(Roll/R_TO_D);
Sy=sin(Yaw/R_TO_D);  Sp=sin(Pitch/R_TO_D); Sr=sin(Roll/R_TO_D);

Rot[1][1]=Cp*Cr;   Rot[1][2]=Sy*Sp*Cr - Cy*Sr;  Rot[1][3]=Cy*Sp*Cr + Sy*Sr;
Rot[2][1]=Cp*Sr;   Rot[2][2]=Sy*Sp*Sr + Cy*Cr;  Rot[2][3]=Cy*Sp*Sr - Sy*Cr;
Rot[3][1]=-Sp;     Rot[3][2]=Sy*Cp;                Rot[3][3]=Cy*Cp;

}


/* ******************************
void Step(void)

USAGE
moves a joint and amount both specified by the user

INPUTS                                                    OUTPUTS
none                                                 none

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

void Step(void)
{
int AxisNo, Speed=300, Type, Rotation;
float Q[6]={0,0,0,0,0,0};
int Traj=0x03f, Accel, P, I, D;
long Degrees;

do
  {
  SetCursor(CHOICE,2);
  printf("enter axisno ?");
  scanf("%d",&AxisNo);

  SetCursor(CHOICE,2);
  printf("enter angle ?");
  scanf("%f",&Q[AxisNo]);

  MoveRobot(Q, NOTRK4);
```

```
  Q[1]=Q[2]=Q[3]=Q[4]=Q[5]=0;

  /*
  Degrees = (long)( Q[AxisNo] * (float)AxisSwitch[AxisNo] *
                    (float)Pulses[AxisNo] );

  SfwReset(AxisNo);
  Sbpa(AxisNo,0x7FFF);
  Fsti(AxisNo,0x10);
  Dfh(AxisNo);

  Accel=10;      P=500; I=750; D=1500; Speed=150;

  Lfil(AxisNo,P,I,D,1000,0x080F);
  Udf(AxisNo);

  LtrjPLS( AxisNo , Degrees , Speed , Accel , Traj );
  Stt(AxisNo);
  Sleep(2);
  ReleaseBrake(AxisNo);

  Sleep(2500);

  NewStop(Q);

  ApplyBrake(AxisNo);
  Sleep(5);
  SfwReset(AxisNo);
  */


  SetCursor(CHOICE,2);
  printf("repeat ?");
  choice[1]=getche();

  }
while(choice[1]=='y');

}


/* ******************************
void ConstVelo(void)

USAGE
moves a joint at a constant velocity

INPUTS                                    OUTPUTS
none                                  none
```

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

void ConstVelo(void)
{
float P[6]={0,0,0,0,0,0}, Ang;
int AxisNo;
extern int Status_Array[8];

SetCursor(CHOICE,2);
printf("which axis to move ?");
scanf("%d",&AxisNo);
SetCursor(1,2);
printf("axisno=%d ",AxisNo);
SetCursor(CHOICE,2);
printf("how many degrees ?");
scanf("%f",&Ang);

P[AxisNo]=Ang;

SetCursor(2,2);
printf("p1=%f  p2=%f  p3=%f axisno=%d ",P[1],P[2],P[3],AxisNo);
choice[1]=getche();

RdStat(AxisNo);
SetCursor(3,2);
printf(" 1st 4  :  %d  %d  %d  %d ",
  Status_Array[0],Status_Array[1],Status_Array[2],Status_Array[3] );
SetCursor(4,2);
printf(" 2nd 4  :  %d  %d  %d  %d ",
  Status_Array[4],Status_Array[5],Status_Array[6],Status_Array[7] );

SetCursor(CHOICE,2);
printf("ok to move ?");
choice[2]=getche();

if(choice[2]=='y')
   /*MoveRobotCV(P);*/printf("error - file not here!!");

RdStat(AxisNo);
SetCursor(6,2);
printf(" end ...1st 4  :  %d  %d  %d  %d ",
  Status_Array[0],Status_Array[1],Status_Array[2],Status_Array[3] );
SetCursor(7,2);
printf(" 2nd 4  :  %d  %d  %d  %d ",
  Status_Array[4],Status_Array[5],Status_Array[6],Status_Array[7] );
choice[2]=getche();

}

/* ******************************
Beep( int frequency, int duration )
```

```
USAGE
makes the PC speaker beep

INPUTS                                                  OUTPUTS
the frequecy and duration to beep                       none

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

Beep( int frequency, int duration )
{
   int control;

   if( frequency )
   {
         /* 75 is about the shortest reliable duration of a sound. */
         if( duration < 75 )
            duration = 75;

         /* Prepare timer by sending 10111100 to port 43. */
         _outp( 0x43, 0xb6 );

         /* Divide input frequency by timer ticks per second and
          * write (byte by byte) to timer.
          */
         frequency = (unsigned)(1193180L / frequency);
         _outp( 0x42, (char)frequency );
         _outp( 0x42, (char)(frequency >> 8) );

         /* Save speaker control byte. */
         control = inp( 0x61 );

         /* Turn on the speaker (with bits 0 and 1). */
         _outp( 0x61, control | 0x3 );
   }

   Sleep(duration);

   /* Turn speaker back on if necessary. */
   if( frequency )
         _outp( 0x61, control );

}
```

W_OFFLIN.C

```
/*begin robot motion*/

#include<stdio.h>
#include<stdlib.h>
#include "c:\masters\proj\ext_vars.h"
#include "c:\masters\proj\fnproto.h"

/*function prototypes*/
void YPR(float Yaw, float Pitch, float Roll);
void PickPos(void);
void InsertPos(void);
void TestPos(void);
void RunWS3trk(void);

int FlagPick=FALSE, InsertFlag=FALSE, Delay;
float Rot[4][4], Dumb;

/*  ************************
void RunWS3trk(void)

USAGE
runs the WS3 track file written in C

INPUTS                          OUTPUTS
WS3c.trk                  controls the robot !!!

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

void RunWS3trk(void)
{
#define SAME    0
#define NOTSAME  1

int Filter;
char cmnd[30], Action[10];
float Pos[4];
FILE *Fptrin1;

Fptrin1 = fopen("c:\\masters\\proj\\ws3c.trk","r");

while(strncmp(cmnd,"/*end*/",7) != SAME)
  {
  fscanf(Fptrin1, "%s", cmnd);

  if(strncmp(cmnd, "InvKin", 6) == SAME)
    {
    fscanf(Fptrin1, "%f,%f,%f,%f,%f,%f,%s\n",
          &Pos[1],&Pos[2],&Pos[3],&Yaw,&Pitch,&Roll,&Action);
```

```
  if(strncmp(Action, "TRK4", 4) == SAME)
       Filter=TRK4;

  YPR(Yaw,Pitch,Roll);
  InvKinematics(Pos,Rot,Filter);
  }
if(strncmp(cmnd, "AlignToolCube", 13) == SAME)
  AlignTool(CUBE);

if(strncmp(cmnd, "AlignToolHole", 13) == SAME)
  AlignTool(HOLE);

if(strncmp(cmnd, "CloseHand", 9) == SAME)
  CloseHand();

if(strncmp(cmnd, "OpenHand", 8) == SAME)
  OpenHand();

if(strncmp(cmnd, "PosHole", 7) == SAME)
  InsertObject();

if(strncmp(cmnd, "PosCube", 7) == SAME)
  GraspObject();

if(strncmp(cmnd, "IntPosCube", 10) == SAME)
  IntersectPos(CUBE);

if(strncmp(cmnd, "IntPosHole", 10) == SAME)
  IntersectPos(HOLE);

if(strncmp(cmnd, "MoveRel", 7) == SAME)
  {
  fscanf(Fptrin1, "%f,%f,%f",&Pos[1],&Pos[2],&Po
  MoveXAxis(Pos[1]);
  MoveYAxis(Pos[2]);
  MoveZAxis(Pos[3]);
  }

if(strncmp(cmnd, "MoveAway", 8) == SAME)
  {
  fscanf(Fptrin1, "%f",&Dumb);

  Filter=NOTRK4;
  Pos[1]=AbsPos[0];
  Pos[2]=AbsPos[1];
  Pos[3]=AbsPos[2] + Dumb;

  YPR(Yaw,Pitch,Roll);
  InvKinematics(Pos,Rot,Filter);
  }
```

```
  if(strncmp(cmnd, "MoveHome", 8) == SAME)
    MveHme();

  if(strncmp(cmnd, "Pause", 5) == SAME)
    {
    fscanf(Fptrin1, "%f",&Delay);
    Sleep(Delay);
    }

  }

}


/* ******************************
void YPR(float Yaw, float Pitch, float Roll)

USAGE
calculates the tool rotation matrix from the orientation of the tool
in yaw, pitch and roll angles

INPUTS                                          OUTPUTS
the tool yaw, pitch and roll angles        the rotational matrix

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

void YPR(float Yaw, float Pitch, float Roll)
{
float Cy,Cp,Cr,Sy,Sp,Sr;

Cy=cos(Yaw/R_TO_D); Cp=cos(Pitch/R_TO_D);        Cr=cos(Roll/R_TO_D);
Sy=sin(Yaw/R_TO_D);  Sp=sin(Pitch/R_TO_D); Sr=sin(Roll/R_TO_D);

Rot[1][1]=Cp*Cr;   Rot[1][2]=Sy*Sp*Cr - Cy*Sr;  Rot[1][3]=Cy*Sp*Cr + Sy*Sr;
Rot[2][1]=Cp*Sr;   Rot[2][2]=Sy*Sp*Sr + Cy*Cr;  Rot[2][3]=Cy*Sp*Sr - Sy*Cr;
Rot[3][1]=-Sp;     Rot[3][2]=Sy*Cp;             Rot[3][3]=Cy*Cp;

}
```

WS3TOC.C

```
/* converts a WS3 track file into a C file
NB THE TOOL ORIENTATION IN WS3 MUST ALAWYS BE SET ON
YAW, PITCH, ROLL --IN THAT ORDER, FOR THIS PROGRAMME TO
WORK*/

#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>
#include "c:\masters\proj\ext_vars.h"

#define SAME    0
#define NOTSAME  1
#define TRUE    1
#define FALSE   0
#define NumStrings   12
#define R_TO_D                  57.29578

void Header1(void);
void Footer(void);
void FeedBack(void);
void ConvertCmnds(void);
void Modify(void);
void TranslateOne(void);
YPR(float Yaw, float Pitch, float Roll);

void Header2(void);
void WriteFile(void);
void StoreStrings(void);
void FndHndCmnds(void);
void ChngInvKin(int k);

FILE *Fptrin1;
FILE *FptrOut1;
FILE *Fptrin2;
FILE *FptrOut2;

char string[2][NumStrings+2][30];
char text[150][100];
int Totalstrings;
float R[4][4];

Convert()
{
SetCursor(RESULTS1,2);
printf("\t\tTHE TOOL ORIENTATION MUST BE IN THE ORDER (YAW, PITCH, ROLL)");
SetCursor(RESULTS2,2);
printf("\t\tIN WS3 IN ORDER FOR THE TRANSLATION TO BE CORRECT.");
SetCursor(RESULTS3,2);
```

```
printf("\t\tPRESS 'Y' TO CONTINUE 'N' TO EXIT.        : ");
choice[0]=getche();


If(choice[0]=='y' || choice[0]=='Y')
  {
  ConvertCmnds();
  Modify();
  }

}

/* ****************************
void ConvertCmnds(void)

USAGE
converts the WS3 commands into equivalent C commands

INPUTS                                    OUTPUTS
the WS3 track file                  the C track file

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

void ConvertCmnds(void)
{
FptrOut1 = fopen("c:\\masters\\proj\\dummy.txt","w+t");
Fptrin1 = fopen("c:\\ws3\\trk\\Wits.kl","r");

Header1();
TranslateOne();
Footer();

fclose(Fptrin1);
fclose(FptrOut1);

}

void Header1(void)
{
fprintf(FptrOut1, "\n/*the_WS3_dummy_file*/ \n\n");
}

void Footer(void)
{
fprintf(FptrOut1, "\n\n/*end*/");
}


/* ****************************
void TranslateOne(void)
```

```
USAGE
translates the WS3 track file to a dummy text file which is then examined for
open or close hand commands

INPUTS                                  OUTPUTS
the WS3 track file                  dummy.txt file

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

void TranslateOne(void)
{
int axis, d1, d2;
float ang, P[8];
char cmnd1[10], cmnd2[10],cmnd[10],cmnd3[10],arb[10], arb2[10], c1,c2;

while(strncmp(cmnd,"END",3) != SAME)
  {
  fscanf(Fptrin1, "%s", cmnd);

  /*MOVE COMMANDS*/

  if(strncmp(cmnd, "MOVE", 5) == SAME)
    {
    fscanf(Fptrin1, "%s", cmnd2);

    /*MOVE AXIS  Axis(Axisno,?,Degrees)*/
    if(strncmp(cmnd2, "AXIS", 4) == SAME)
        {
        fscanf(Fptrin1, "%d %s %f",&axis,arb,&ang);
        fprintf(FptrOut1, "\n/*Move Axis Command*/\n");
        fprintf(FptrOut1, "MoveRobot  %d %f \n");
        }

    /*MOVE TO  x,y,x,yaw, pitch,roll*/
    if(strncmp(cmnd2, "TO", 2) == SAME)
        {
        fscanf(Fptrin1, "%1s",cmnd3);
        if(strncmp(cmnd3, "P", 1) == SAME)
          {
          fscanf(Fptrin1, "%[^(] \n",arb);
          fscanf(Fptrin1, "%1s",arb);
          fscanf(Fptrin1, "%f,%f,%f,%f,%f,%f",
            &P[1],&P[2],&P[3],&P[4],&P[5],&P[6]);

          fprintf(FptrOut1, "InvKin  %0.3f,%0.3f,%0.3f,%0.3f,%0.3f,%0.3f\n",
                P[1],P[2],P[3],P[4],P[5],P[6]);
          }

        if(strncmp(cmnd3, "$HOME", 1) == SAME)
          fprintf(FptrOut1, "MoveHome\n");
```

```
        }

    /*MOVE NEAR  (x,y,x,yaw,pitch,roll,") by z*/
    if(strncmp(cmnd2, "NEAR", 4) == SAME)
        {
        fscanf(Fptrin1, "%[^(]",arb);
        fscanf(Fptrin1, "%1s",arb);
        fscanf(Fptrin1, "%f,%f,%f,%f,%f,%f",
           &P[1],&P[2],&P[3],&P[4],&P[5],&P[6]);

        fscanf(Fptrin1, ",%s",arb),
        HndElbShld(arb);

        fscanf(Fptrin1, "%s %f",arb,&P[7]);

        fprintf(FptrOut1, "InvKin   %0.3f,%0.3f,%0.3f,%0.3f,%0.3f,%0.3f\n",
                  P[1],P[2],P[3]+P[7],P[4],P[5],P[6]);
        }

    /*MOVE RELATIVE  (x,y,z)*/
    if(strncmp(cmnd2, "RELATIVE", 8) == SAME)
        {
        fscanf(Fptrin1, "%[^(]",arb);
        fscanf(Fptrin1, "(%f,%f,%f)",&P[1],&P[2],&P[3]);

        fprintf(FptrOut1, "MoveRel  %0.3f,%0.3f,%0.3f\n",
                P[1],P[2],P[3]);
        }

    /*MOVE AWAY  n */
    if(strncmp(cmnd2, "AWAY", 4) == SAME)
        {
        fscanf(Fptrin1, "%f",&P[7]);

        fprintf(FptrOut1, "MoveAway %0.3f\n",P[7]);

        }

    } /*end of MOVE commands*/

if(strncmp(cmnd, "CLOSE", 5) == SAME)
    {
    fscanf(Fptrin1, "%s",cmnd2);

    if(strncmp(cmnd2, "HAND", 4) == SAME)
        {
        fprintf(FptrOut1,"AlignToolCube\n");
        fprintf(FptrOut1,"PosCube\n");
        fprintf(FptrOut1,"CloseHand\n");
        }
    }
```

```
  if(strncmp(cmnd, "OPEN", 4) == SAME)
    {
    fscanf(Fptrin1, "%s", cmnd2);
    if(strncmp(cmnd2, "HAND", 4) == SAME)
          {
          fprintf(FptrOut1,"AlignToolHole\n");
          fprintf(FptrOut1,"PosHole\n");
          fprintf(FptrOut1,"OpenHand\n");
          }
    }
  }
}



/* ****************************
redundant function

USAGE
hand,elbow,shoulder condition

INPUTS                                          OUTPUTS

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

HndElbShld(char string[])
{
if(strncmp(string,"ELB",3) == SAME)   /*hand,elbow,shoulder condition*/
  {
  printf("ELB ???\n\n");
  }
}


/* ****************************
void Modify(void)

USAGE
the WS3 track file cannot be translated directly as the tool needs to
be aligned accurately with the cube or hole before the hand opens or
closes

INPUTS                                  OUTPUTS
dummy.txt file          ws3trk.c file which can be run directly

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

void Modify(void)
{
Fptrin2 = fopen("c:\\masters\\proj\\dummy.txt","r");
```

```
FptrOut2 = fopen("c:\\masters\\proj\\ws3c.trk","w+t");

StoreStrings();
FndHndCmnds();
WriteFile();

fclose(FptrIn2);
fclose(FptrOut2);
}


/* ******************************
void StoreStrings(void)

USAGE
copy the strings in dummy.txt into memory

INPUTS                    OUTPUTS
dummy.txt        none

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

void StoreStrings(void)
{
int k=0,i=0,Flag=FALSE;

while(Flag==FALSE)
  {
  fscanf(FptrIn2, "%s", text[i]);
  if(strncmp(text[i],"/*end*/",7) == SAME)
        Flag=TRUE;
  Totalstrings=i;
  i++;
  }

}

/* ******************************
void FndHndCmnds(void)

USAGE
find the commands to open or close the hand in dummy.txt

INPUTS
the strings stored in memory

OUTPUTS
the number of the string containing a command to open or close the
hand

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

```
void FndHndCmnds(void)
{
int i;

for(i=TotalStrings;i>=1;i--)
    {
    if(strncmp(text[i],"OpenHand",8) == SAME)
        ChngInvKin(i);
    if(strncmp(text[i],"CloseHand",9) == SAME)
        ChngInvKin(i);
    }
}

/* ****************************
void ChngInvKin(int k)

USAGE
-- aligns the tool in the Z plane, compensating for the rotation of axes
  1 and 5

INPUTS
whether we are looking for the cube or hole centre of area

OUTPUTS
the X and Y differences between the tool centre and cube/hole centre in
robot base coords which is then passed on to inverse kinematics

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

void ChngInvKin(int k)
{
int   h,j, margin=200;
float  p1,p2,p3,p4,p5,p6;


for(h=k;h>=1;h--)
  {
  if(strncmp(text[h],"InvKin",6) == SAME)
    {
    sscanf(text[h+1], "%f,%f,%f,%f,%f,%f",&p1,&p2,&p3,&p4,&p5,&p6);
    p3+=margin;
    j=sprintf(text[h+1], "%0.3f,%0.3f,%0.3f,",p1,p2,p3);
    j+=sprintf(text[h+1]+j, "%0.3f,%0.3f,%0.3f",p4,p5,p6);
    break;
    }
  if(strncmp(text[h],"MoveRel",7) == SAME)
    {
    sscanf(text[h+1], "%f,%f,%f",&p1,&p2,&p3);
    if(p3!=0)
        {
```

```
            p3+=margin;
            j= sprintf(text[h+1], "%0.3f,%0.3f,%0.3f",p1,p2,p3);
            }
        break;
        }
    }

}

/* ******************************
void WriteFile(void)

USAGE
writes the modified file to a text file

INPUTS                              OUTPUTS
the modified strings          WS3trk.c

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

void WriteFile(void)
{
int i;

for(i=0;i<=Totalstrings;i++)
  {
  if(strncmp(text[i],"InvKin",6) == SAME)
    strcat(text[i+1],",TRK4");

  fprintf(FptrOut2, text[i]);
  fprintf(FptrOut2, "\n");

  if(i==0 || i==Totalstrings-1)
    fprintf(FptrOut2, "\n");
  }
}
```

APPENDIX 3

The Denavit-Hartenburg Principle For Mathematically Representing A Robot

1. Number the joints from 1 to n starting with the base and ending with the tool yaw, pitch and roll in that order;

2. Assign a right-handed orthonormal co-ordinate frame L_o to the robot base making sure that z^o aligns with the axis of joint 1. Set k=1;

3. Align z^k with the axis of joint k+1;

4. Locate the origin of L_k at the intersection of the z^k and z^{k-1} axes. If they do not intersect, use the intersection of z^k with a common normal between z^k and z^{k-1};

5. Select x^k to be orthogonal to both z^k and z^{k-1}. If z^k and z^{k-1} are parallel, point x^k away from z^{k-1};

6. Select y^k to form a right handed orthonormal co-ordinate frame L_k;

7. Set k=k+1. If k<n go to step 2; else continue;

8. Set the origin of L_n at the tool tip. Align z^n with the approach vector, y^n, with the sliding vector and x^n with the normal vector of the tool. Set k=1;

9. Locate point b^k at the intersection point of the x^k and z^{k-1} axes. If they do not intersect, use the intersection of x^k with a common normal between x^k and z^{k-1};

10. Compute ϕ_k as the angle of rotation from x^{k-1} to x^k measured about z^{k-1};

11. Compute d_k as the distance from the origin of frame L_{k-1} to point b_k measured along z^{k-1};

12. Compute a_k as the distance from point b^k to the origin of frame L_k measured x^k;

13. Compute α_k as the angle of rotation from z^{k-1} to z^k measured about x^k;

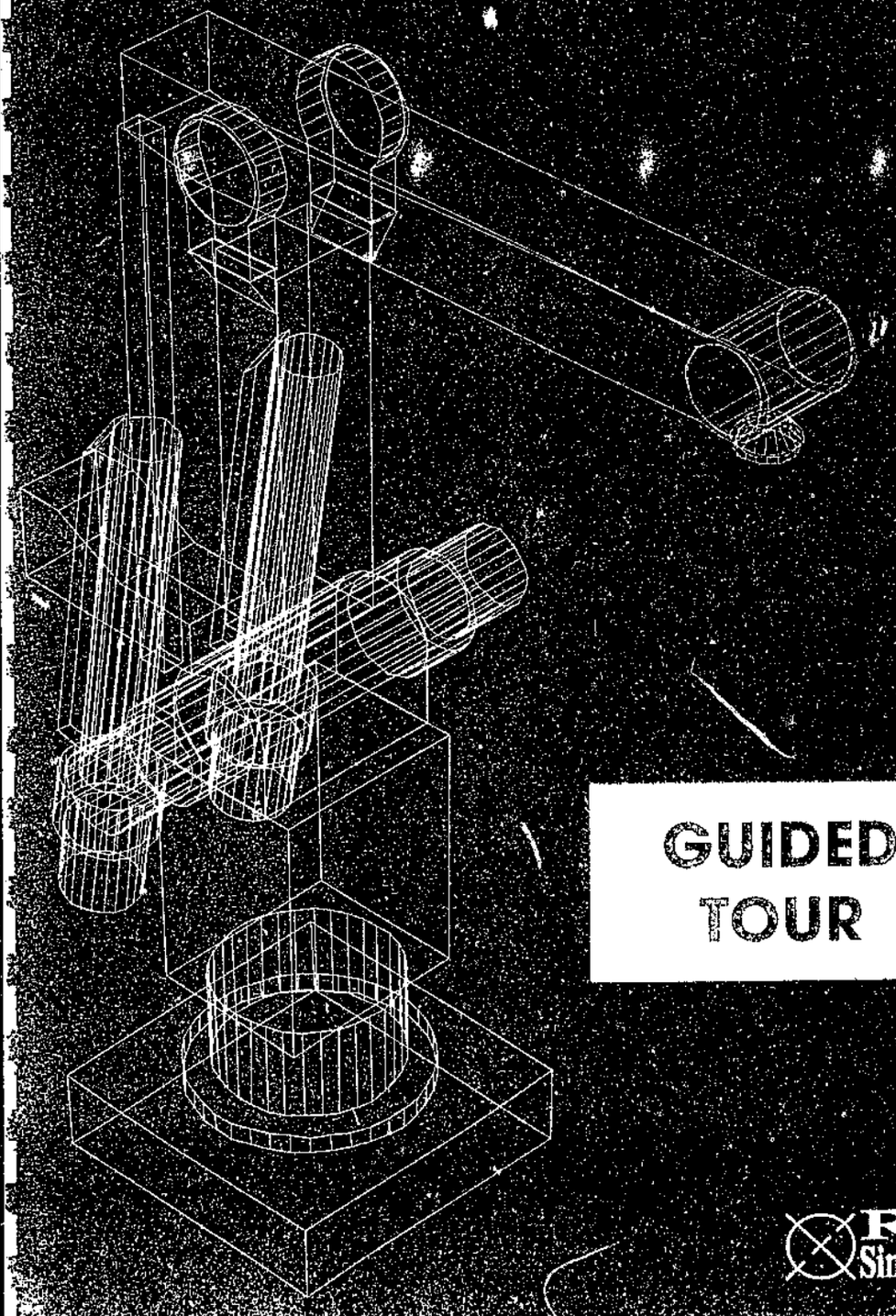
14. Set k=k+1. If k≤n, go to step 8; else stop.

APPENDIX 4

List of Hardware And Manufacturer's Details

1. A 386SX PC with 8Mb RAM and a 420Mb Hard Drive;
2. An Eagle Technologies PC 30 AtoD card;
3. An Eagle Technologies PC 26 relay card;
4. A custom built motor control card with five off LM628 motor control chips;
5. Workspace Version 3, Robot Simulation Package.
6. A Video Blaster frame grabbing card (manufacturer's details follow);
7. A "Chipper" CPT8950P CCD with 542H x 582V pixels (manufacturer's details follow) and
8. A Honeywell Ultrasonic Distance Sensor Model 940–A4V–2E–1C0 (manufacturer's details follow).

WORKSPACE



GUIDED TOUR

Robot Simulations Ltd.

INTRODUCTION

WORKSPACE 3.0 is a complex and comprehensive modelling and simulation tool. For the beginner this complexity can be a daunting obstacle on the road leading to the first tentative learning experiences. This ' HOW TO .....' manual is designed to enable the first-time user to get to grips with WORKSPACE 3.0 with minimum effort and maximum enjoyment. A series of simple exercises guide the user through the wide ranging capabilities of WORKSPACE 3.0. Each exercise illustrates one or more of the features of the package. Explanations of both the terminology used and technical features are included as appropriate. Exercises often utilise the 'top down' approach whereby global features are examined before details are considered. An advantage of this method is that when the more complex or obscure features are encountered they fit easily into context.

It is recommended that the beginner follows the 'HOW TO .... ' exercises in sequential order, i.e. working from exercise 1 to exercise 10. After gaining some experience individual exercises may be chosen to brush up on a particular topic as and when required.

An accompanying REFERENCE MANUAL gives precise details of all WORKSPACE 3.0 features.

SYSTEM REQUIREMENTS

WORKSPACE 3.0 will run on any IBM compatible 286, 386 or 486 PC. However machines with less than 1MB of RAM must use the the special WS3UNPRO.EXE in place of the WS3.EXE file. Purchasers should inform Robot Simulations Ltd. if they intend to run WORKSPACE 3.0 on 286 machines. For this manual it is assumed that the user has a minimum machine configuration of a 386 SX with 2MB of RAM and a Microsoft compatible mouse. If you do not have a mouse the arrow and return keys may be used to execute commands.

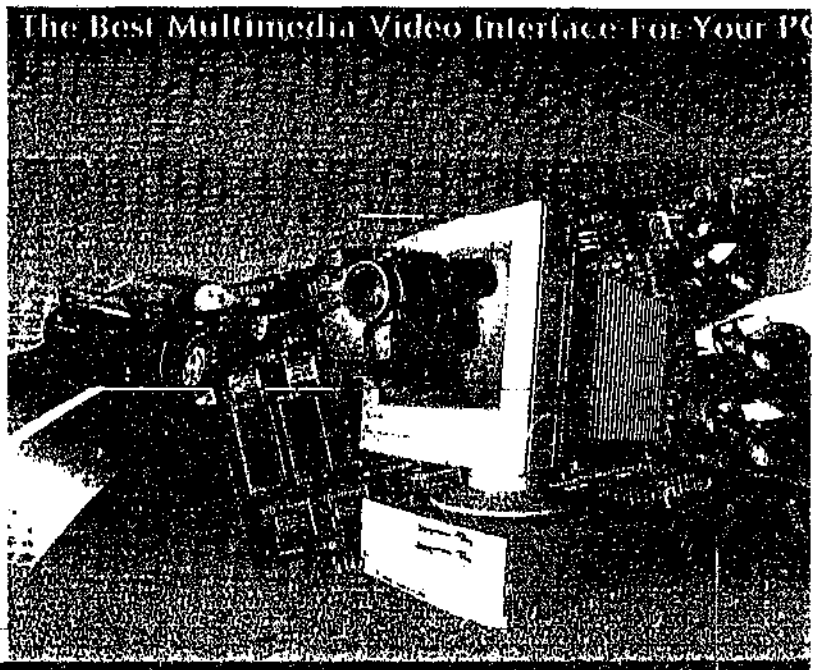
Please note that WORKSPACE 3.0 will not run without a DONGLE. (A dongle is a small electronic key which plugs into the parallel printer port on the back of the computer.) An appropriate dongle is supplied with each purchased licence of WORKSPACE 3.0.

WORKSPACE 3.0 has been extensively tested and 'debugged' prior to release. However it is inevitable that a package of this complexity will have minor bugs which have not come to light during the rigorous testing. Users who experience any difficulties should contact:

Robot Simulations Ltd.,
Lynnwood Business Centre
Lynnwood Terrace,
Newcastle Upon Tyne,
NE4 6UL
ENGLAND
Tel: +44 (0)91 272 3673
Fax: +44 (0)91 272 0121

Video

The Best Multimedia Video Interface For Your PC



BLASTER™ SE100

VIEW VIDEO IN A WINDOW
VIDEO CAPTURE
GRAPHICS OVERLAY

- Display full-motion digital video in a movable, resizable window
- Scale images from full screen to icon size
- Video control of hue, saturation, contrast and brightness
- Capture, manipulate and export video images
- Chroma keying, text overlay and other special effects
- Attractive bundled software

CREATIVE LABS

Video Blaster SE100

Video Blaster SE100 is designed to be a video overlay and capture board for users. It integrates video images from a wide range of sources - video cameras, VCRs, laser disc - with your PC applications. Video Blaster SE100 allows a user to overlay live video on the PC. The overlay window can be re-sized to any size up to full screen. It allows user to create interactive and stunning multimedia presentation.

VIDEO FEATURES

Video Acquisition
- Software selectable video source from 2 composite or 1 S-Video input
- Supports NTSC, NTSC-M, PAL, PAL-B and PAL-G

Video Overlay
- Overlay live video on VGA in re-sizable windows
- Chroma keying on selected color in YUV domain

Frame Grabbing Function
- Capture high quality single frames
- Supports BMP, TIFF, GIF, MMP, PCX, TGA and JPEG

Image manipulation
- Zooming and scaling of live video
- Freeze, save and load images
- Resize, flip and crop live video
- Special effects like Flipping, Strobing, Magnifying and Source switching
- Hue, saturation, contrast and brightness control

Masking
- Chroma keying to superimpose live video on VGA display
- Color keying to export live video

Video Capture
- Capture video and save into hard disk as AVI files

HARDWARE SPECIFICATIONS

Input
- 2 female RCA jacks for 2 composite video signals (75 ohms)
- 1 S-Video connector for S-Video input
- 15-pin "D" shaped connector from VGA card
- Feature connector port to VGA card

Output
- 15-pin "D" shaped connector to VGA monitor

I/O and Memory Addressing
- Selectable I/O addresses
- Software selectable frame buffer address on expanded and extended memory address space
- Selectable Interrupt 5,10 (default), 11, 12 and 15

Additional Features
- The VB SE100 has no 16 MB RAM limitation. It also allows both Memory or I/O mapping.
- Print directly to printer in color or B/W
- Twain driver support

VGA Compatibility
Video Blaster SE100 is designed to support up to 800x600 256 colors SVGA display standard which incorporates a feature connector. Check with your respective dealers to ensure compatibility between your system and Video Blaster SE100

** Subject to VGA card used.*

System Requirements
- IBM compatible 386SX and above
- 4 MB RAM (minimum)
- 2 MB Hard Disk space (minimum)
- VGA card with VESA compliant feature connector
- 16 bit ISA expansion slot
- DOS version 3.1 and above
- Windows version 3.1 or later

Package Includes :
- Video Blaster SE100 card
- Video Blaster SE100 connecting cables
- Video Blaster SE Kit application software

Manufactured by Creative Technology Ltd
67 Ayer Rajah Crescent #03-18 Singapore 0513
Tel +65 773 0233 Fax +65 773 0353

Americas Inquiries Creative Labs, Inc
1901 McCarthy Boulevard Milpitas CA 95035
Tel +1 (408) 428 6600 Fax +1 (408) 428 6611
Sales Hotline for USA/Canada +1 (800) 998 5227
Technical Support for USA/Canada +1 (405) 742 6622

European Inquiries
UK Creative Labs, UK
Tel +44 (734) 344322 Fax +44 (734) 320100
France Creative Labs, France
Tel +33 (1) 39208400 Fax +33 (1) 39208606
Germany Creative Labs, Germany
Tel +49 (89) 9928710 Fax +49 (89) 99287122
Benelux Creative Labs, Benelux
Tel +32 (3) 281 3570 Fax +32 (3) 281 3580

ALDUS® PHOTOSTYLER® SE

A special edition of this popular photo editing software turns your PC into a full design and production machine. Provides access to photo CDs and allows you to retouch, [illegible] effects and embellish images. Includes sample photo CDs.

HSC DIGITAL MORPH™

A powerful graphics tool providing morphing, warping, animation and special effects of 8 through 24-bit graphics files. Let your own [illegible] explore the endless capabilities of this popular graphics program.

ASYMETRIX DIGITAL VIDEO PRODUCER™

Create your very own video presentation of movie-quality through this powerful video capture/video editing tool. Incorporate filter [illegible], audio cross-fading and other special effects for an all new and original multimedia product.

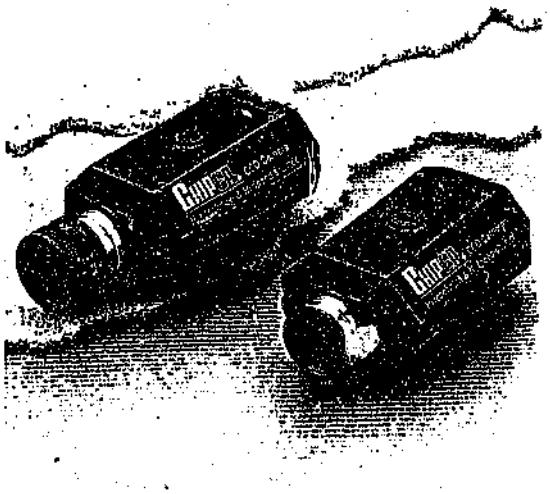
CREATIVE
CREATIVE LABS

Printed in Singapore

Shuttle Type Auto ESC Solid State CCD Cameras

New Product Introduction

Model: CPT8950 Series
CPT8950P Series



- 1/3 Inch Format CCD Image Sensor
  CPT8950: 542H x 492V Active Pixels
  CPT8950P: 542H x 582V Active Pixels
- Full Performance and Resolution
- Excellent Sensitivity
- Minimal Blooming and Transfer Smear
- Signal-to-Noise : 46 dB Typical
- No Geometric Distortion
- Built-in Auto. ESC.
- Line-Lock with V. Phase Adjustment on AC Models (Switching Power)
- 12 VDC and 24 VAC Type
- System
  CPT8950 : EIA RS-170
  CPT8950P : CCIR

Specifications

Imager CCD Image Sensor: 1/3-inch image format.
CPT-8950: 542H x 492V active picture elements.
CPT-8950P: 542H X 582V active picture elements.

Sensitivity (2856 K) Scenes 0.01 fc (0.1 Lux).

Horizontal Resolution 420 TVL.

Signal-to-Noise 46 dB Typical.

Gray Scale At least 10 steps.

Video Output 1.0 V p-p, 75 ohm.

Electronic Shutter
Automatic Electronic Shutter Control(ESC).
Range : 1/60 -- 1/30,000

Gamma Correction 0.45.

Geometric Distortion None.

Lens Mount "CS" mount; "C" mount with adapter.

Camera Mounts 1/4"-20, top and bottom.

Synchronization
Line-Lock: Synchronizes camera to power line zero crossing for roll-free vertical interval switching. Line phase can be externally adjusted (continuously) to allow vertical synchronization in multiphase power installation.

Rear Connectors
Video out: BNC
Power 24 VAC Type : Screw terminals.
12 VDC Type : DC jacket.

Controls
Flange Focus.
Phase Adjustment.

Dimensions and Weights
125 x 65 x 48 mm - 310g

Temperature
Operating: -20 to +55 °C.
Storage: -30 to +70 °C.

Humidity 0 to 95% relative, non-condensing.

Vibration 3g swept sine wave, 15 to 2000Hz.

Shock 50g, 11ms, 1/2 sine.

Type

Model	P.S.	Line-Lock
CPT8950	12 VDC	without
CPT8950/L24	24 VAC	with
CPT8950P	12 VDC	without
CPT8950P/L24	24 VAC	with

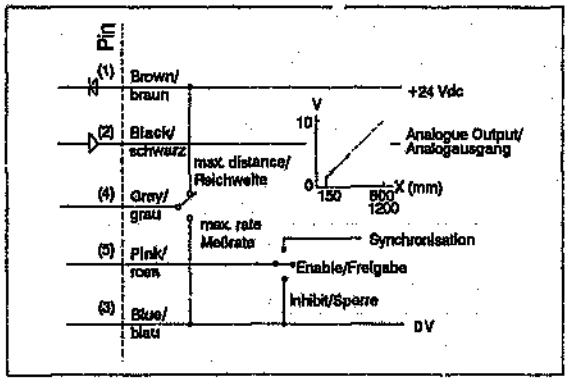
Note :
We reserve the right to change the design and specification without notice.

Ultrasonic Distance Sensor
Ultraschall Abstandssensor

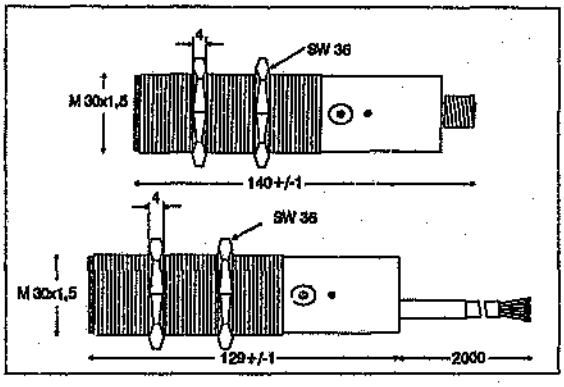
940-A4V-2E-1C0
940-A4Y-2E-1C0

Technical Data / Technische Daten

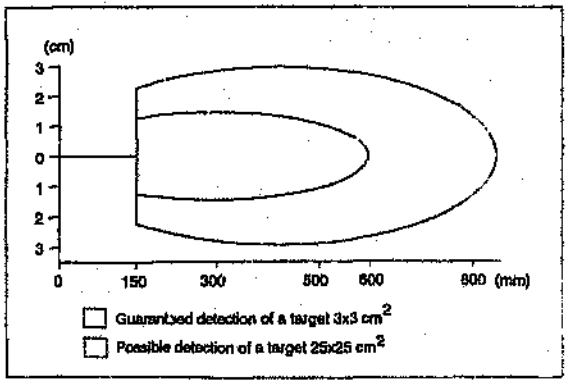
Mode/Betriebsart	Max. sensing Distance Max. Reichweite	Max. measuring rate Max. Meßrate
Sensing distance/Reichweite	150 ... 1200 mm	150 ... 600 mm
Response time/Ansprechzeit 90% final value/Endwert	50 ms	50 ms
Beam angle/Schallkeule	8°	6°
Mode selection/Wahl Betriebsart Connection/Anschluß Pin 4 (grey/grau) to/an	+24 V	0 V
Repeatability/Wiederholgenauigkeit	+/- 1mm	
Linearity Error/Linearitätsfehler %	<0,2	
Temperature compensation/ Temperaturkompensation	0 ... 70°C measurement error/Meßfehler +/-1%, 0... 50°C	
Operating voltage/Betriebsspannung	19...50 Vdc	
Current consumption/Stromaufnahme	<25 mA without load/ohne Last	
Output/Ausgang	0...10Vdc(10mA), sensitivity/Steilheit 8 ...17mV/mm	
Adjustment aid/Ausrichthilfe	LED intensity/Intensität LED	
Inhibit/Sperre	Connection to 0 V interrupts transmission/ Anschluß an 0V unterbricht Sendebetrieb	
Synchronisation	Connection with synchronisationunit/ Verbinden mit Synchronisiereinheit	
Housing/Gehäuse	Stainless steel/Edelstahl	
Sealing/Schutzart	IP65	
Termination/Elektrischer Anschluß	5 pin connector/5 poliger Stecker angle/Winkel No: 66195045-001; corresp./entspr. Binder 713-09-436-00-05 straight/gerade No: 66195044-001; corresp./entspr. Binder 713-09-436-10-05	



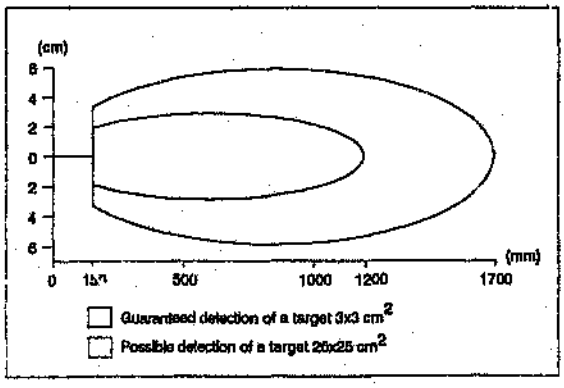
Connections / Anschluß



Dimensions/Abmessungen in mm



Sensing Range/Erfassungsbereich
max. measuring rate/Meßrate



Sensing Range/Erfassungsbereich
max. distance/Reichweite

MB-1093

APPENDIX 5

Finding The Target Centroid And Orientation

Finding The Centroid

1. Find the top, bottom, left and right edges;
2. Calculate the position of the X centre of the target for each of the n rows which have left and right edges. Call the result X_i;
3. Add all the X_i for i=1 to i=n call the result X_{total};
4. Divide X_{total} by n , call the result X;
5. X is the position of the target's centroid in CCD co-ordinates;
6. Repeat for the top and bottom edges to find Y and
7. The target's X and Y position of the centroid has now been found in CCD co-ordinates in pixels.

Finding The Orientation

1. Find the top, bottom, left and right edges;
2. Find the target's centroid;
3. Calculate the distance between each edge and the centroid using Pythagorous' theorem and store the results in an array called Dist[n]. Dist[n] keeps a record of where on the boundry the edge was found;
4. Sort array Dist[n] in order of distance, greatest distance to least;
5. The cube and square hole both have 4 sides and therefore the distance between the corners will be at the top of the array Dist[n] . Therefore select the first 4 entries of Dist[n] as the corners with the condition that the distance between them must be greater than half the width or height of the cube to prevent error;
6. Using simple trigonometry calculate the angle between the 4 corners with respect to the CCD y axis and average the results in order to find the angle the target makes with the CCD y axis;

7. Multiply this result by the CCD's aspect ratio and
8. Rotate the tool by this value.