AN IMPLEMENTATION OF OBJECT RECOGNITION USING BINOCULAR VISION

By

Xi Luo

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2011

Nashville, Tennessee

Approved by:

Professor Kazuhiko Kawamura

Professor Mitchell Wilkes

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge the full support and guidance rendered by my advisor Dr. Kazuhiko Kawamura in my pursuit of knowledge in Vanderbilt University as well as in my research and compilation of this thesis. Also, I would like to thank Dr. Mitch Wilkes for his helping me better understand the tasks. In addition, I would like to express my hearty thanks to my fellow graduate students in the CIS and all my friends in Vanderbilt University for helping me all along the way of my life and study here. I also would like to express my deepest gratitude for the financial and academic support so kindly provided by the School of Engineering and the Graduate School of Vanderbilt University and all the faculty members.

My special thanks also go to my family for their advice, encouragement and love extended all through my entire life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ISAC	Intelligent SoftArm Control
CRL	Cognitive Robotics Laboratory
HSV	Hue Saturation Value
BGR	Blue Green Red
CMY	Cyan Magenta Yellow
OpenCV	Open-source Computer Vision
IPP	Intel's integrated Performance Primitives
IPL	Intel Processing Library
GUI	Graphical User Interface
MLL	Machine Learning Library
PTU	Pan-Tilt Unit
PCI	Peripheral Component Interconnect
DAC	Digital to Analog Converter
ACD	Analog to Digital Converter
CPP	C-Plus-Plus source code file
H	C-Plus-Plus header code file
ROI	Region of Interest

Chapter I

INTRODUCTION

I.1. Problem Description

According to the evidence found in biology and psychology, the process of human visual perception is related to a “fixation-move-fixation” rhythm that has been named as active vision by David Marr [1][2]. In the recent past, the scientists in electrical engineering and computer science adopted this concept and expected to manipulate the viewpoint of the cameras in order to investigate the environment and then get better information from it. Increasingly, many robotic platforms are implementing active vision to conduct tasks.

ISAC is the humanoid robot developed by Cognitive Robotics Laboratory (CRL) in Vanderbilt University [3]. ISAC is equipped with a stereo vision system and two McKibben Artificial Muscles [4] manipulators.

In this thesis, the stereo pan-tilt cameras mounted at the head of ISAC are used to implement the process of “fixation-move-fixation” rhythm in vision and two sets of pan-tilt units serve as the moving mechanism for active vision. Some objects are placed on the table in front of ISAC. These objects are toys with the combination in different colors and different shapes. ISAC uses cameras to scan the environment randomly in horizontal direction until it finds the target object, which has been manually chosen. Segmentation in HSV image [5] is used first to identify the color and then connected components in binary image are used for identifying the shape. Database is served as an infusion memory to incorporate different colors and shapes. Then based on the

position of the object, the visual system maps the location of the object in relation to the body of ISAC, and uses the arm to point out the corresponding object. Since the cameras are moving along horizontal direction with pan units, there is a linear mapping relationship according to the different combinations of angles of two sets of pan-tilt units. If the object is placed within the workspace of ISAC's right arm, use its arm to point out the object; otherwise, a warning message will be shown on the screen saying, "The location is out of reach of ISAC".

I.2. Active Vision in Psychology

Before active vision was proposed, passive vision had been broadly accepted in visual science for a long time. The term *passive vision* was theorized by David Marr [6] which means keeping the gaze steady and continually watching a specific scene. The passive approach has its wide applicability because many psychologists believe that the purpose of vision is to form a mental representation and passive vision can offer us a wide range of information immediately without paying any specific attention. There is an underlying assumption based on passive vision, that is, visual projection on retina is an incidental response.

However, more and more evidences showed incompatible results with those obtained by passive vision. First contradiction was when the eyes are moving the visual representation does not maintain constantly. The second contradiction occurs in the binding problem [7][8] which involves integration and combination of different kinds of features rather than simply adding those features together. For example, generally speaking, visual mechanism always perceives the local particular features such as

yellow and red color or a triangle or a square shape. But in point of view of binding problem, the perception is the combination of these four features instead of yellow, red, triangle and square. Solution of the binding problem was found to be visual attention which implies preferring the more important features to others. Selection on attention of a visual region can be happened in two ways. When facing in one direction and looking at one object, we can say, “Something catches our eyes”. We can also look at one thing but give attention to another at the same time.

Since 1980, active vision has been put forward since lots of evidences are found that humans obey a “fixation-move-fixation” rhythm in visual sampling process [1][2]. Active vision includes these crucial problems that will be discussed Chapter IV:

a) How to make decision when to terminate one fixation and where to move the gaze?

b) What information is taken during a fixation?

I.3. Active Vision in Robotics

Compatible to the principal methodology for obtaining visual images in the retinal area provided by biology and psychology, a number of scientists and engineers in robotics are dedicating to implement reproducible active vision experiments and operations on the specific physical platforms. Although they have a similar philosophy on vision, unlike psychology, active vision in robotics places more emphasis on practical applications [9].

Mathematical models provide many kinds of methods to model the image formed on the retinal stimulation. For example, with discrete Fourier analysis all images can be

reconstructed by a series of discrete sinusoidal waves. Based on this model, computer vision scientists developed machine architectures in order to simulate human visual processes. Also grounded on the static image as a starting point, they intend to process the image with many kinds of mathematical based algorithms.

Generally speaking, active vision can be achieved by moving cameras and off-the-shelf image processing tools. If the image does not include the desirable features, the cameras will move in certain patterns, such as going along with a specific trajectory or just move randomly. Although color, which is sensitive to the lighting conditions, can not be reliable under some circumstances, color is still a staple feature that is often used during the process of fixation. Besides color, contour and shape are other available choices. With only one feature, the system may find more than one candidate object. Obviously, the binding of two or more features can limit the number of candidate objects and then provide much more precise results. However, this may require different types of devices and fast speed of processing.

Chapter II

OVERVIEW OF APPROACH

The research described in this thesis integrates a number of hardware and software components. Some of them were programmed by the former students. This chapter discusses the hardware platform and software modules in detail.

II.1. Methodology

Since the focus of this thesis is to detect objects in different colors and shapes in an active vision circumstance, many commonly used computer vision algorithms are included and some of them are developed in several previous thesis and projects in the CRL.

Extraction and Labeling of Connected Components

In computer vision, there are several pre-existing color models for describing the specification of the colors such as RGB, CMY and HSV. This thesis uses HSV-based (hue, saturation, value) color model since it greatly decreases the size of color and grey-scale information of an image. A set of isolated points are clustered into objects by color extraction. Hue filtering is used to segment the specified color. When the value of hue is set, a mask is applied to the image [10]. In a binary image, when the value of pixels satisfy a specified criterion, such as hue, the value transformed by masking function is set to zero which shows in white color; otherwise, it is shown in black [11]. All the pixels in the appointed hue range are marked as foreground which are shown in white and other pixels are marked as background are shown in black. In brief, a

connected component is an entity of foreground pixels that are neighbor to each other. If there is a gap between the foreground pixels separated by background, then these set of pixels are considered to be disconnected components which means that they are two different objects.

In this thesis, the objects are assumed to be in three different kinds of shapes: triangle, square and round. The criterion for detecting triangle object is to calculate the extrema points and the preset threshold difference between the maximum axis length and minimum axis length. Ideally, a triangle has only three extrema points. However, due to the noise in binarization and the limitation of the image resolution, there might be some noise points locating around the real extrema points. In order to solve this problem, erosion and dilation methods are used to cluster the extrema points along with noise points and then round those points. Similar methods [12] are applied to detect square objects. Figures 1 to 3 show an example of the result of detecting a blue triangle object. As shown in Figure 3, the system has successfully extracted the triangle, labeled as “1”, from other objects in the same color.

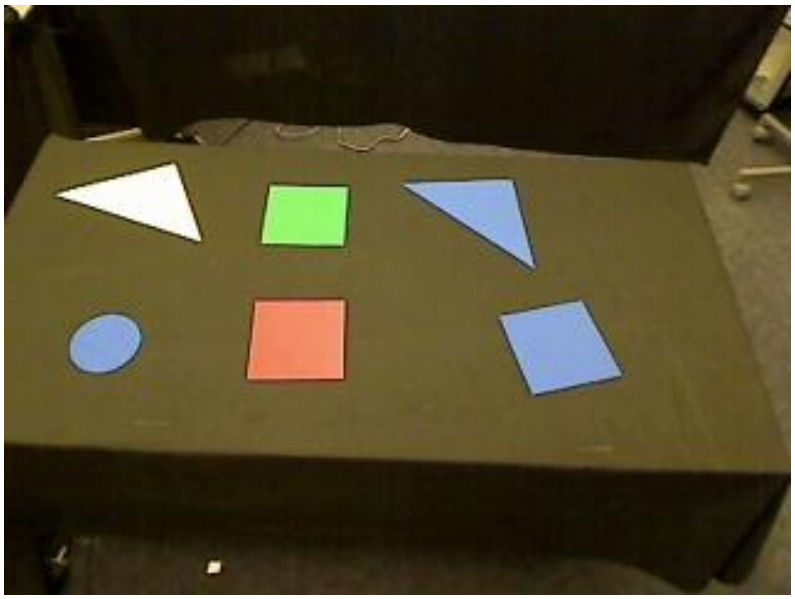

Figure 1: Original Image Grabbed by Camera

Figure 2: Image processed by hue filtering

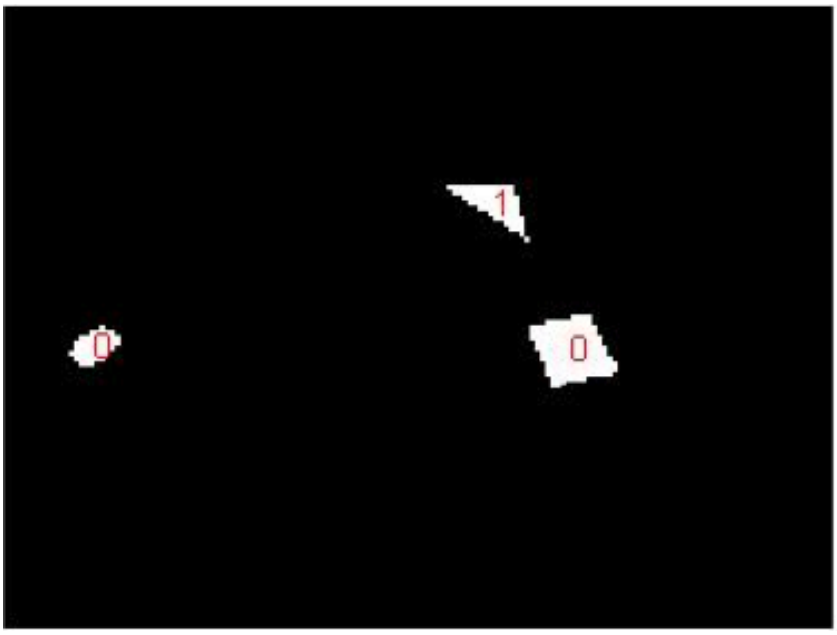

Figure 3: Image Points Out the Triangle Shape

These images have also dealt with morphological erosion and dilation methods. Further discussion about these functions is described in Chapter IV.

Hand Reaching

Once the predetermined object has been recognized and located, ISAC tries to reach it using its right arm. The original software interfaced with arm control was written by Huan Tan in 2009. The arm control code works as follows: give the 3-D coordinates with respect to the origin of coordinate system and then command the designated arm to reach that coordinates. The origin of coordinate system is the center of the shoulder of the right arm. Corresponding coordinates-relation mapping between different coordinate systems is also made in this thesis. Arm control for ISAC will be further described in detail in section 2 of Chapter III.

II.2. Approach

Since ISAC needs to use stereo-cameras to detect and locate the objects in different colors and shapes, this process requires a plentiful of computer vision source codes. Some of the codes are built on OpenCV, and others are built on Matlab grounded on fast processing speed.

OpenCV

Much of the color detection in the system is based on Intel's OpenCV (Open-source Computer Vision) Library [13]. OpenCV originally came from one project which aims at CPU-intensive advancement by Intel Research. Not only the functions are optimized algorithm, but also the efficiency of OpenCV is grounded on Intel's integrated Performance Primitives (IPP) library which operates multimedia processing and multi-core functions in assembly language. The first version of OpenCV released to the public is alpha version at the IEEE Conference on Computer Vision and Pattern Recognition in 2000. The current version is 2.2 which was released in Dec. 2010. In this version, OpenCV has made changes to C++ interface for easier operation, new functions and better performance on the implementation.

OpenCV contains over 500 programming functions for real-time computer vision image processing that covered many fields in vision, including factory product inspection, security systems, medical imaging stereo vision and robotics. All the functions in OpenCV are assigned to implement on IpIImage structure. The IpIImage structure was designed for Intel Processing Library (IPL) which includes many parameters that can satisfy a variety need of image processing, shown in Figure 4.

```
typedef struct _IplImage
{
    int  nSize;             /* sizeof(IplImage) */
    int  ID;                /* version (=0)*/
    int  nChannels;         /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int  alphaChannel;      /* ignored by OpenCV */
    int  depth;             /* pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                               IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported */
    char colorModel[4]; /* ignored by OpenCV */
    char channelSeq[4]; /* ditto */
    int  dataOrder;         /* 0 - interleaved color channels, 1 - separate color channels.
                               cvCreateImage can only create interleaved images */
    int  origin;            /* 0 - top-left origin,
                               1 - bottom-left origin (Windows bitmaps style) */
    int  align;             /* Alignment of image rows (4 or 8).
                               OpenCV ignores it and uses widthStep instead */
    int  width;             /* image width in pixels */
    int  height;            /* image height in pixels */
    struct _IplROI *roi;/* image ROI. when it is not NULL, this specifies image region to process */
    struct _IplImage *maskROI; /* must be NULL in OpenCV */
    void  *imageId;         /* ditto */
    struct _IplTileInfo *tileInfo; /* ditto */
    int  imageSize;         /* image data size in bytes
                               (=image->height*image->widthStep
                               in case of interleaved data)*/
    char *imageData;    /* pointer to aligned image data */
    int  widthStep;     /* size of aligned image row in bytes */
    int  BorderMode[4]; /* border completion mode, ignored by OpenCV */
    int  BorderConst[4]; /* ditto */
    char *imageDataOrigin; /* pointer to a very origin of image data
                              (not necessarily aligned) -
                              it is needed for correct image deallocation */
}
IplImage;
```

Figure 4: IplImage Structure

OpenCV also contains practical functions with HighGUI for users to develop simple GUI (Graphical User Interface) and MLL (Machine Learning). The usage of OpenCV library is discussed in later sections.

Platform

The vision system of this research resides on the computer in the Cognitive Robotics Lab named Sally, which runs Windows XP. The arm control system is operated on the computer named Octavia, which runs Windows 2000. Below are the typical command sequences to enter to initiate the vision system shown in Figure 5.

```
//variables for pan/tilt units
CameraHead hd;                  //head object (pan/tilts)
hd.Initialize();
hd.Home();                      //ensure the pan/tilts are at the home position

cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_WIDTH, x);
cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_HEIGHT, y);

cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_WIDTH, x);
cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_HEIGHT, y);

cvNamedWindow("Left", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Left", 100, 400);

cvNamedWindow("Right", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Right", 440, 400);

IplImage* frame1 = cvQueryFrame(capture2);
IplImage* out1 = 0;
IplImage* frame2 = cvQueryFrame(capture1);
IplImage* out2 = 0;

cvShowImage( "Left", frame1);
cvShowImage( "Right",frame2);
```

Figure 5: Command Sequence for Initiating Vision System

Figure 6 shows the arm control panel. Left two columns show the current position and joints angles of left and right arm. Right two columns show the desired position or desired joints angles of two arms. Once filled in the desired position with respect to ISAC of left arm and right arm or the desired angles of joints, press “Start” button to move the single or both arms to desired position or desired joints angles. Press ”End” button when the movement of reaching is completed.

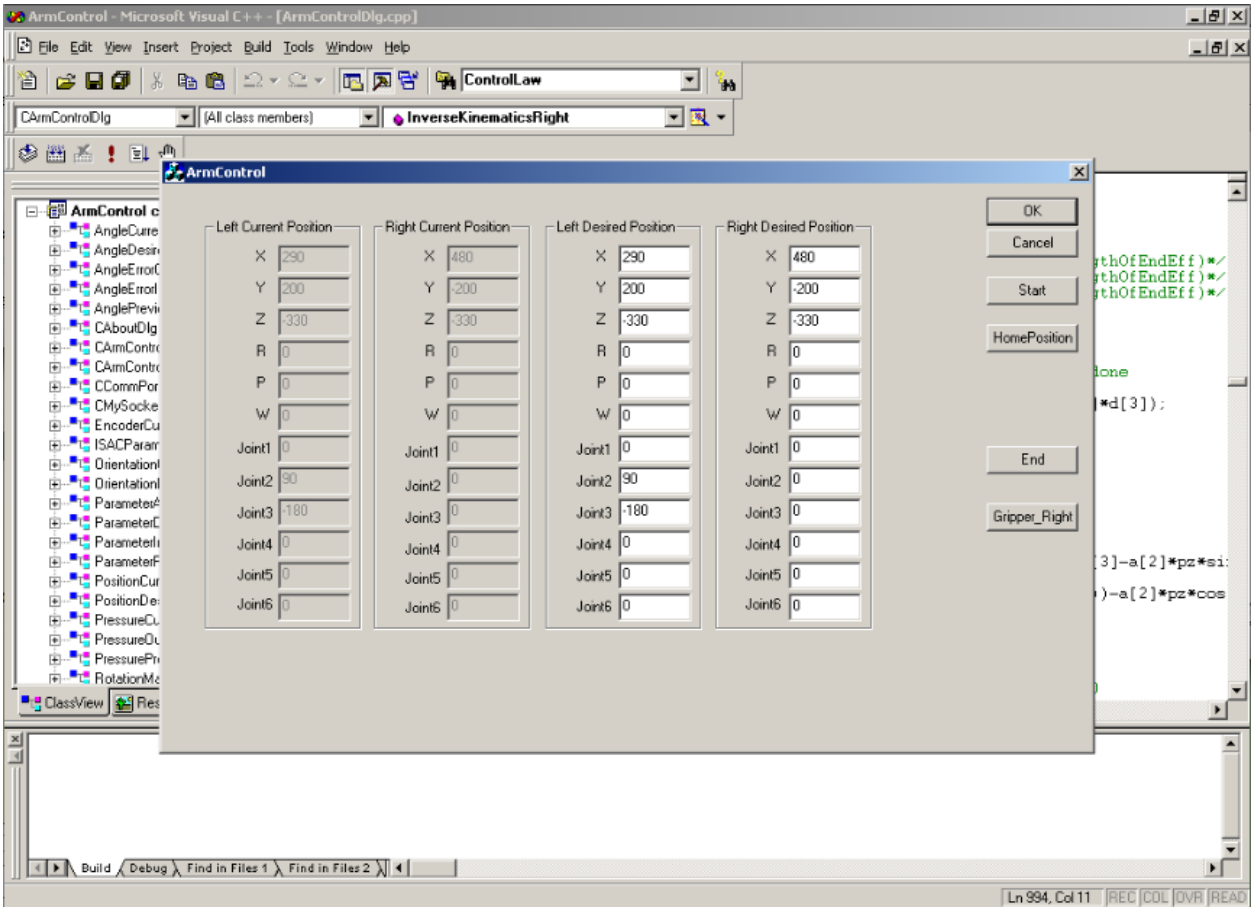

Figure 6: Arm Control Panel

Chapter III

SYSTEM SETUP

This Chapter discusses in detail the hardware that has been used in the experiments. The experiments have been performed on the humanoid robot ISAC (Intelligent Soft Arm Control) which is located in the Cognitive Robotics Laboratory at Vanderbilt University [14]. The overall picture of ISAC is shown in Figure 7.

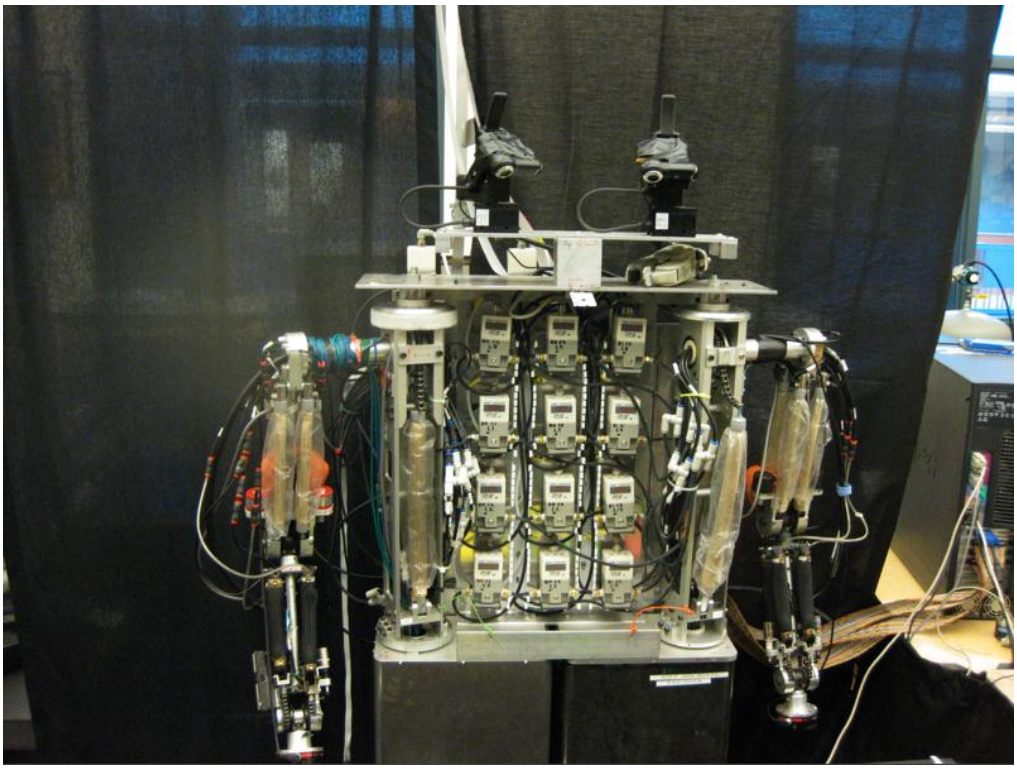

Figure 7: ISAC Humanoid Robot

III.1. Active Vision Hardware

The vision system operates on the computer in the Cognitive Robotics Lab named Sally. The hardware includes two Logitech cameras and two sets of Directed Perception pan-tilt units. Figure 8 shows the ISAC vision system.

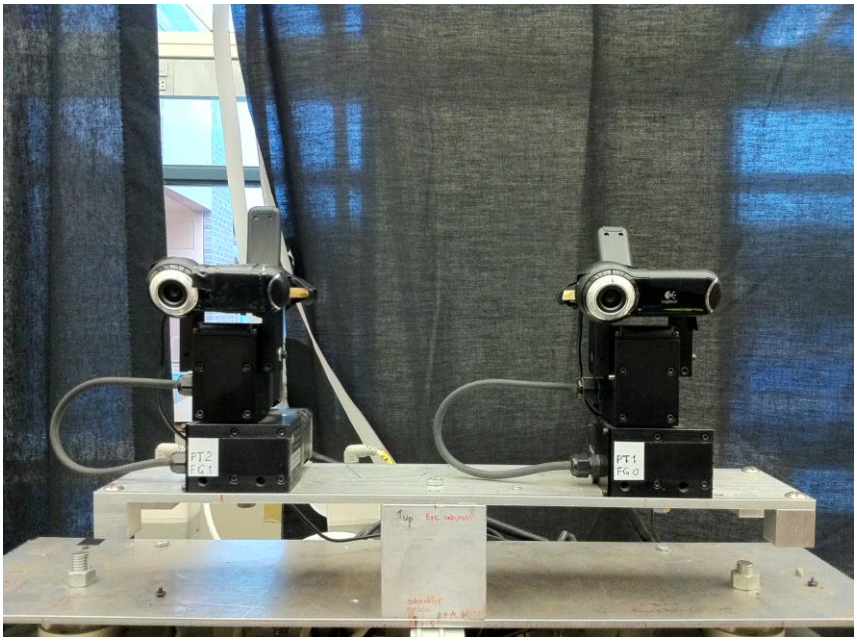

Figure 8: ISAC Vision System

III.1.1. Cameras

ISAC uses two Logitech Webcam Pro 9000 cameras [15], which is shown in Figure 9.

Figure 9: Logitech Webcam Pro 9000 Cameras

The resolution of these cameras is 1600×1200. However, in order to satisfy the speed of image processing of active vision, the effective resolution has been set to 320×240. Standing at the base of cameras, there is a universal clip which lets the glass lens move vertically. But during the experiments, pan-tilt units control the direction of focus, thus the lens are kept stationary and kept straight forward.

III.1.2. Directed Perception Pan-Tilt Units

The Logitech cameras are connected to the two sets of Directed Perception PTU-D46-17.5 pan-tilt units [16] separately, shown in Figure 10. Along with these pan-tilt units, the cameras can move horizontally and vertically respectively. This means both webcams to possess two degree-of-freedom instead of six rectus muscle in human eyes. This model of PTU has a position resolution of 0.05143(1/2 step) and 0.0123 (1/8 step) in the pan and tilt directions. They can move at the velocity up to 300° per second. In the experiments, once the cameras tracked target object, the positions of PTU were fixed.

Figure 10: Pan-tilt Unit

III.2. Arm

Arm control system includes several components for the movement of two arms consisting of McKibben artificial muscles [17], air pressure valves, rotary encoders, and control boards for the valves and the encoders. The detail of each component is described below.

McKibben Artificial Muscles

The arms are six-degree-of-freedom serial robot manipulators which are actuated by McKibben artificial muscles. McKibben artificial muscles are pneumatic artificial muscles which usually work in pairs: one muscle is agonist and another is antagonist [18]. The skeleton of muscle is a rubber tube which is wrapped with stretch fabric mesh as shown in Figure 11. The muscles are imbued with compressed air at a constant pressure value. The muscles expand when they are pressurized with air. However,

similar to human, the muscle can not be expanded excessively since the flexibility of the muscles is restricted by the sum strength of all the single fibers.

Figure 11: Skeleton of the Mckibben Artificial Muscles

A McKibben artificial muscle has many advantages in robot applications. First advantage is its lightweight which can reduce the impulsive force during collision [19]. Secondly, the muscle is compliant and it would not continue to extend when it reaches its physics limit no matter how much power have been given on force [20]. This flexibility in mechanism would also have high safety factor when it has been applied to working robot within human environment [21].

A McKibben artificial muscle also has some disadvantages. Not only pressure but also the inflated state of the muscle would affect the force acted on the muscles. Based on this non-linearity mathematical models of inflation and control of the muscle has less accuracy between input force and output extension than other kinds of actuators such as motors. However, seeing from a biological point of view, this limitation in force-extension relationship on body is analogous to the characteristic of humans and may be overcome by cleverly designing the experiments. Other disadvantages include hysteresis and delay in pressure control since it needs some time for compressing and transporting air in long tubes and inflation in muscles.

Each of ISAC arms has 12 McKibben artificial muscles working in antagonistic pairs with 6 degrees of freedom. Figure 12 shows the ISAC's right arm.

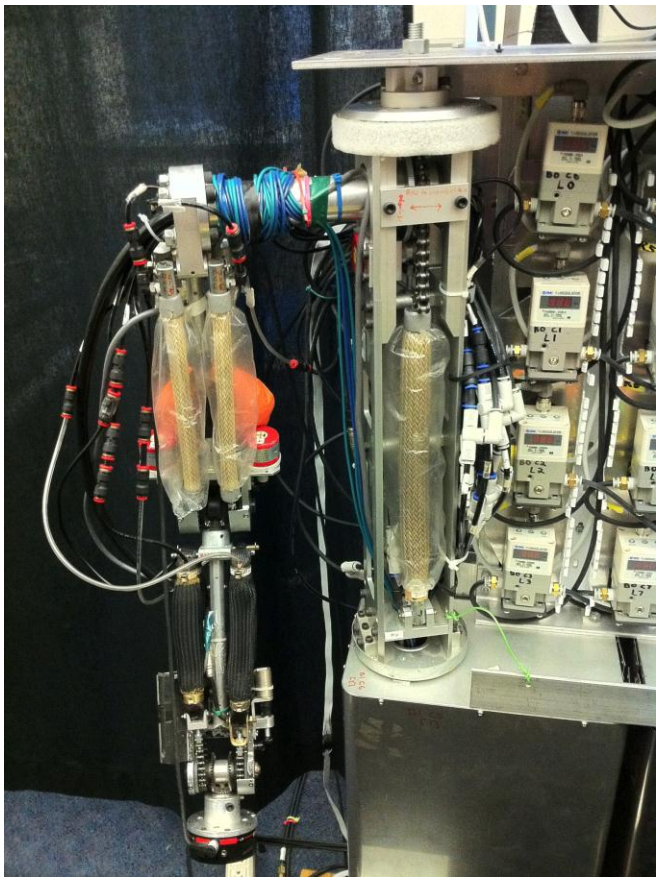

Figure 12: Right Arm of ISAC

The older muscles in white are made by Bridgestone Corporation [22] and the newer black ones are made by Shadow Robotics [23].

The arm joints are assigned from 0 to 5. As demonstrated in Figure 13, coordinate frame 0 of joint 0 is the base of each arm and frame 1 corresponds to the rotation of the base of arm. Frame 2 of joint 2 refers to the rotation of the shoulder along with Y2 axis and frame 3 of joint 3 controls the rotation of the elbow along with Z3 axis. Frame 4 and 5 of joint 4 and 5 refer to the pitch and roll of the wrist, and frame 6 to the rotation of the end-effector.

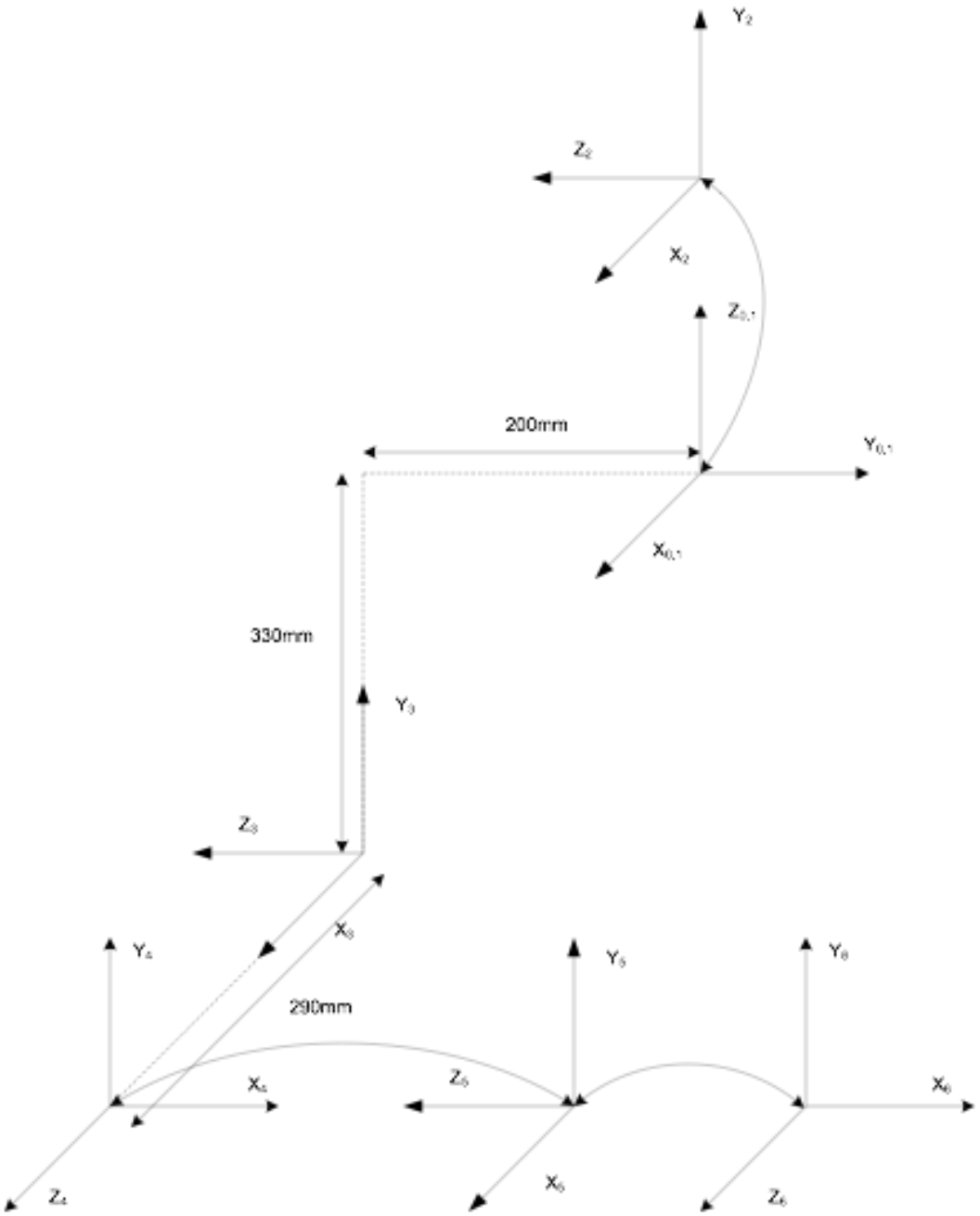

Figure 13: Design of Joints of Right Arm

The muscles used between joints 1,2 and 2,3 are one pair of pneumatic muscles while the muscles between joint 2,3 and joint 3,4 are a set of 4 air muscles, as shown in Figure 14.

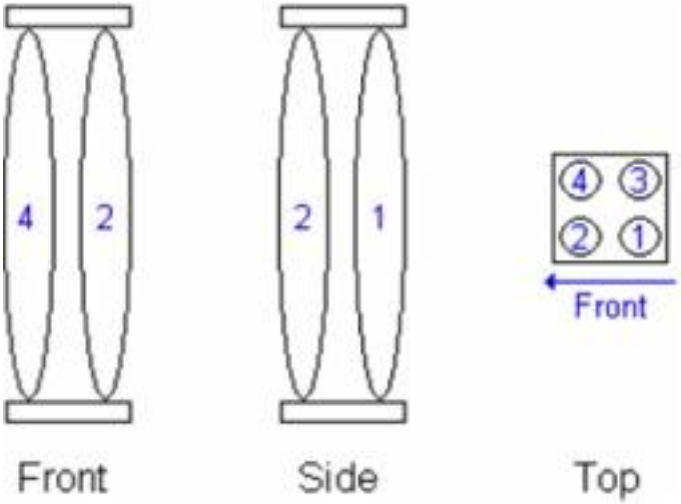

Figure 14: Two Pair of Muscles Configuration

The elbow joint, which connects upper arm and forearm and the wrist joint, which connects forearm and the end-effector, can both bend and rotate. These two joints apply universal joints as shown in Figure 15.

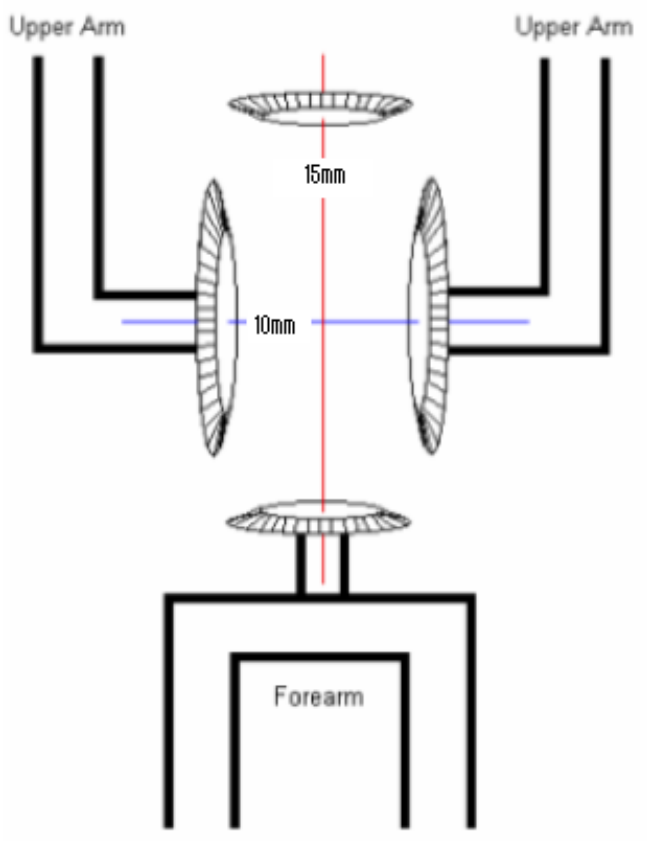

Figure 15: Universal Joints of Elbow and Wrist

With these configurations of muscles and universal joints, ISAC can complete many actions. For example, action of lifting its forearms can be achieved by contracting muscles 2 and 4 and loosening muscles 1 and 3 of upper arms. Action of anti-clockwise rotating the forearms can be done by contracting muscles 1 and 4 and loosening muscles 2 and 3.

SMC Electro-Pneumatic Regulator Valves

As mentioned in the last section, the McKibben artificial muscles are powered by compressed air under a constant pressure in the tank. Since the variances in the air may harm the muscles, such as impurity and inapposite temperature, air from the storage tank first passes through the air dryer and the cooler, then goes to twelve SMC

ITV 2050-312CN4 Electro-Pneumatic Regulator valve arrays [24] (shown in Figure 16) for each muscle of the left arm, the right arm and the pneumatic end effector. These valves regulate the amount of air passed through to the muscles.

Figure 16: SMC ITV 2050 series Electro-Pneumatic Regulator Valve

By applying 0 to 10 volts of input voltage, these valves regulate the air pressure for each muscle between 0.005 and 0.9 MPa [24]. Current air pressure relieves through an exhaust port on the valve as input voltage decreases. One array valves locate on ISAC's chest which regulates the right arm and another array on the back which in charge of the left arm. The valves are controlled by three MOTENC-Lite PCI board [25] which installed in the computer. This controller board will be discussed in next section.

Encoders

In order to improve the accuracy of movement of the arm, arm control adopts closed-loop control. The feedback variables are the current angles of the joints which can be obtained by rotary encoders located at every joint [26]. Each encoder provides the angular offset of single axis of rotation on every joint. The universal joints on elbow and wrist correspond to two encoders since those kinds of joints can rotate in two

directions. Unfortunately right now SUMTAK and Epson-Seiko can not provide the data any more because of the end of production. However, all rotary encodes operate based on common principles: the light source located in encoder projects light onto a rotating code disk. Through slits on the disk, the light goes onto a stationary phase plate. Compared the phases of lights projected on two different disks, the distance that encoder has rotated can be obtained by the same MOTENC-Lite controller cards that control the valve arrays.

Controller Cards

As mentioned in the previous sections, controller cards are in charge of controlling the volt input signals of the electro-pneumatic regulators and identifying values of the encoders. These cards are Vital Systems MOTENC-Lite PCI Boards shown in Figure 17.

Figure 17: MOTENC-Lite: PCI Board

There are three PCI boards installed in the PC “Octavia” in CRL. Each board controls ISAC’s 24 muscles and provides the feedback of 12 encoders. They are connected by ribbon cables to the DAC/ACD/Encoder Termination Boards which work as a bridge between the PCI boards, valves and encoders.

Figure 18: DAC/ADC/Encoder Termination Board

As shown in Figure 18, the black terminal blocks located on the short edges are DAC (digital to analog converter) and ADC (analog to digital converter) blocks which deal with the analog voltage inputs from the valves and provide feedback to them. The 8-connection blue terminal blocks located on the long edges are assigned to deal with the encoders. Since one board supports four encoders, three cards can satisfy the needs of 12 encoders in all.

Chapter IV

IMPLEMENTATION OF ACTIVE VISION SYSTEM

The goal of this research is to build up a system that a robot can find the target object with a specific color and a shape in a considerably dynamic environment. The development involves two subsystems: one is an active vision subsystem and another is an arm control subsystem. Each subsystem includes several smaller components that must be developed. The active vision system is operated on a computer named Sally. The system also embedded two databases for storing the coordinates of candidate objects that means the objects are in correct color but wrong shape. Once the object is both in correct color and shape, the program will record its position to a database and transfer that coordinate to the Octivia for ISAC's arm to reach. Figure 19 illustrates the decision-making steps for the entire active vision system.

IV.1. Active Vision System

In active vision system implementation, there are two key challenges: a) The system requires to operate in real-time which means image processing time should be short enough in the continuous searching loop, and b) color and shape detection should be fast and precise which requires a fault-tolerance mechanism to handle unstable image inputs.

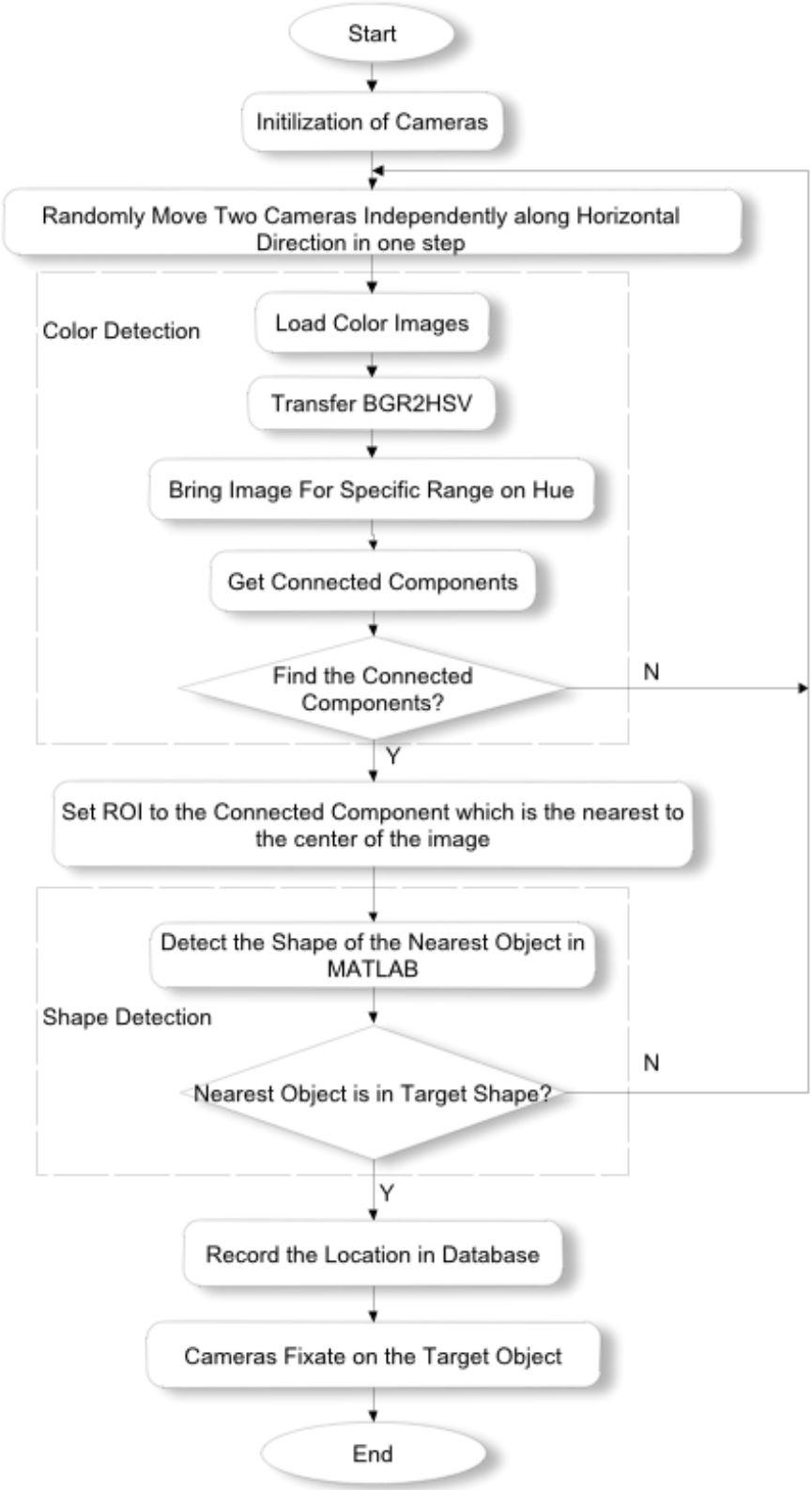

Figure 19: Flowchart of the active vision system

IV.1.1. Active Vision System Setup

The experiments were performed on one humanoid robot ISAC that is equipped with stereo cameras. All the objects are placed on the table in front of ISAC. The table is covered with the black cloth in order to diminish the noise from detecting colors and shapes. Figure 20 shows the experimental set up.

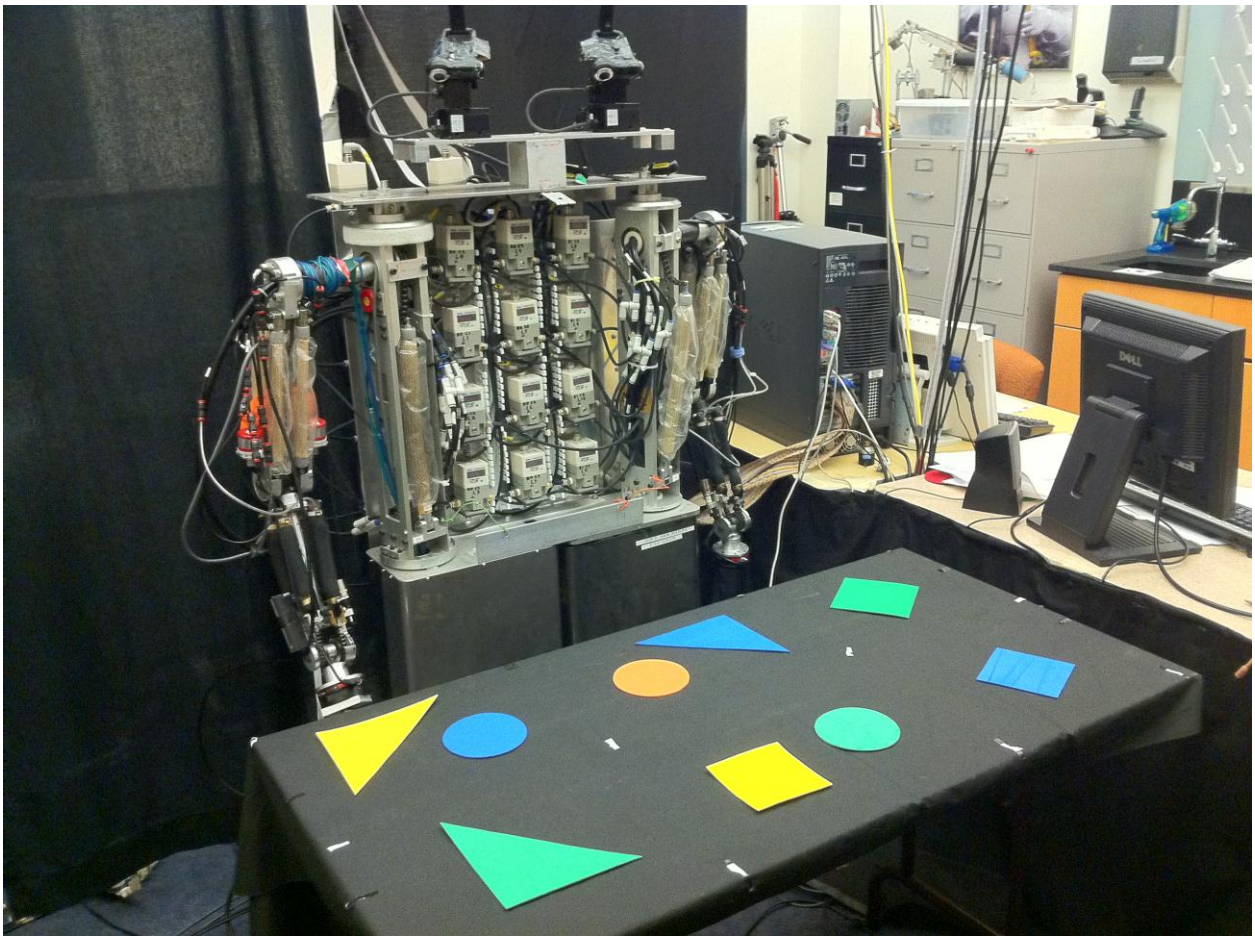

Figure 20: Experiment Set Up

The active vision system consist of two sets of Directed Perception pan-tilt units (hardware) and C++ Source Code (CameraHead.cpp) and C++ header files (CameraHead.h) (software) that were written by former students in our lab. Although

those codes include many kinds of movements of pan-tilt units, only four commands were used in this thesis. The control function is shown in Figure 21.

```
//variables for pan/tilt units
CameraHead hd;              //head object (pan/tilts)
hd.Initialize();    //initialize the units
hd.Home();                  //ensure the pan/tilts are at the home position
MoveHead();                 //move the pan/tilt units to the specific angles
```

Figure 21: Pan-Tilt control code

The function *CameraHead.hd* was used to declare a public class which was already programmed in the CameraHead.cpp file. The *hd.Initialize()* function was used to initialize the pan-tilt units and pre-set their moving parameters. The function *hd.Home()* ensures that the pan-tilt units are at the home position. In the experiments, the home position was set to 4 degrees along the pan axis and -35 degree along the tilt axis. Figure 22 shows how pan-tilt units can randomly move along the pan axis to generate the one-dimensional active vision.

```
double Number_rand;
double headAngles0[4];

srand(GetTickCount());

//Scan at random angles
//Pan Unit of the left eye
Number_rand=(double)rand()/((double)RAND_MAX + 1);
headAngles0[0] = -45+Number_rand*90;;
headAngles0[1] = -35;

//Pan Unit of the left eye
Number_rand=(double)rand()/((double)RAND_MAX + 1);
headAngles0[2] = -45+Number_rand*90;
headAngles0[3] = -35;

hd.MoveHead(headAngles0);
```

Figure 22: code to randomly move pan-tilt units

The function *srand(GetTickCount())* is used to generate a pseudorandom number whose random seed is the time elapsed on the operating system. The function *(double)rand()/((double)RAND_MAX + 1)* and *headAngles0[] = -45+Number_rand*90* would result in a random float number from 0.0 to 1.0 and then set the angles of pan units to the range of -45 to 45 degree. Finally, *hd.MoveHead(headAngles0)* function moves the units to the specific angles.

IV.1.2. Learning Stage

Learning stage presets the hue range of colors and the number of extrema points of different shapes that has already been completed before the program starts. Based on the model of the cameras used in this thesis, the exposure and gain level can be set manually with the software provided by Logitech Company [27]. Select the appropriate value for exposure and gain and then preset the hue range as Table 1 shows. The range chosen in Table 1 would better describe the color based on experiment result.

Colors	Low	High
Orange	14	23
Green	40	60
Blue	80	130
Yellow	30	32

Table 1: Hue Range for Color Detection

For learning the shape, if the number of extrema points of one object is 3, then the object can be determined as a triangle and if it is 4, the object can be considered as a square. Also a fault-tolerance mechanism had been used in detecting square and

triangle that will be discussed later. Determining the round shape is based on three characteristics that will discuss in Section IV.1.3.2.

IV.1.3. Tracking Stage

Once the color and the shape of the target object were selected, the program starts to track that object. Accordingly to the learning stage, there are two detection parts: color and shape. Color detection is all operated in Visual Studio 2008. Shape detection is completed in MATLAB. In this case, there is also an integration part for color and shape detection. Since the image processing handles the image as a matrix, the reason for using MATLAB is that MATLAB deals with matrix operation much better and more accurate than c++. Although MATLAB is not real-time software, in these experiments the processing time of MATLAB was fast enough.

IV.1.3.1. Color Detection

Color detection is performed on single camera at one time. That is to say, in order for the system to acquire binocular vision, the color detection algorithm needs to run twice. Figure 23 shows the codes for color detection with the left camera.

```
//Image color detection
IplImage* hsv1 = cvCreateImage(cvGetSize(frame1), 8, 3);
IplImage* hue1 = cvCreateImage(cvGetSize(frame1), 8, 1);
IplImage* sat1 = cvCreateImage(cvGetSize(frame1), 8, 1);
IplImage* val1 = cvCreateImage(cvGetSize(frame1), 8, 1);
IplImage* FrameLeftProcessed = cvCreateImage(cvGetSize(frame1), 8, 1);
IplConvKernel * selem = cvCreateStructuringElementEx(3,3,1,1,CV_SHAPE_RECT);          //kernel for use with
erode/dilate

//Extract Hue/Sat/Val from BGR Image
cvCvtColor(frame1, hsv1, CV_BGR2HSV);                                  //convert from BGR to HSV
cvSplit(hsv1, hue1, sat1, val1, 0);                                //extract hue/sat/val channels

//Filter by Hue
cvInRangeS(hue1, cvScalar(colors[0]), cvScalar(colors[1]), FrameLeftProcessed);          //filter by Hue
cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);//Erode/Dilate maskH to eliminate noise
cvDilate(FrameLeftProcessed, FrameLeftProcessed, selem, 3);
```

```
cvNamedWindow("Left_Processed", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Left_Processed", 100, 600);

cvShowImage( "Left_Processed", FrameLeftProcessed);


cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working
now/pics/left_processed.jpg",FrameLeftProcessed);

//Find all the positive objects in left eye
getConnectedCompsleft(FrameLeftProcessed,Comps_left);                    //retrieve a list of all connected
components in filtered image

m=0;
if (comptot_left != 0)
{
        for (int j = 0; j < comptot_left; j++)
        {
                //get center
                positionl[j][0] = Comps_left[j]->rect.x + Comps_left[j]->rect.width/2;
                positionl[j][1] = Comps_left[j]->rect.y + Comps_left[j]->rect.height/2;

                //get radius
                if (Comps_left[j]->rect.width > Comps_left[j]->rect.height)
                {
                        positionl[j][2] = Comps_left[j]->rect.width/2;
                }
                else
                {
                        positionl[j][2] = Comps_left[j]->rect.height/2;
                }

                if(positionl[j][2]>7)
                {
                        double q;
                        position_l[m][0]=positionl[j][0];
                        position_l[m][1]=positionl[j][1];
                        position_l[m][2]=positionl[j][2];
                        //calculate the distance from each object to the center of right camera
                        q= (double)(position_l[m][0]-160)*(position_l[m][0]-160)+(position_l[m][1]-
120)*(position_l[m][1]-120);
                        position_l[m][3]=sqrt(q);
                        m++;

                }
        }

}

for(b=0;b<m;b++)
{
        position_sl[b][0]=position_l[b][0];
        position_sl[b][1]=position_l[b][1];
        position_sl[b][2]=position_l[b][2];
        position_sl[b][3]=position_l[b][3];
}
```

```
 //sort position according to the distance
for(k=0;k<m;k++)
{
        for(s=0;s<m-1;s++)
        {
                if(position_sl[s][3]>position_sl[s+1][3])
                {
                        double t,tx,ty,tr;
                        tx=position_sl[s][0];
                        ty=position_sl[s][1];
                        tr=position_sl[s][2];
                        t=position_sl[s][3];
                        position_sl[s][0]=position_sl[s+1][0];
                        position_sl[s][1]=position_sl[s+1][1];
                        position_sl[s][2]=position_sl[s+1][2];
                        position_sl[s][3]=position_sl[s+1][3];
                        position_sl[s+1][0]=tx;
                        position_sl[s+1][1]=ty;
                        position_sl[s+1][2]=tr;
                        position_sl[s+1][3]=t;

                }
        }
}
```

Figure 23: Color detection code

These color detection process first converts the BGR (Blue-Green-Red) image, which is a 32-bit per pixel representation provided by the Logitech web cameras, into a three-band 32-bit HSV(Hue Saturation Value) represented image. Then the HSV image is split into three 8-bit one-band images. These single bands are called: hue, saturation and value. This research only used the hue band. Then the hue image is segmented for a specific range of hue that has been determined in the learning stage and selected for the target color. If the hue value of the pixel is located in the specified hue range, then the corresponding pixel in the mask image is colored white. Otherwise it is colored black. The mask image is processed by erosion and dilatation with a 3*3 square structuring element [28]. The erosion can help diminish the noise by eliminating the small blobs of foreground(shown in white) in the mask. Dilation function expands the eroded foreground back to its original size. The mask *FrameLeftProcessed* is analyzed by the

getConnectedComps function (cf. section II.2.1) which returns a list of all connected foreground components which are the objects in one particular color. The center coordinates and the radius of all the components are stored in *positionl* variable. Based on experimental test, the radius of the minimum of the objects is 7. So if the radius of the detected connected components is bigger than 7, the detected components are considered as the possible target object. Then the distance between the center of object and the center of the camera is calculated and stored in the third element of *position_l* variable. Processed with a bubble sort, the object with the smallest distance, whose position is stored in *position_sl* variable, is the candidate object that will be processed by shape detection. The image of mask *FrameLeftProcessed* is saved for shape detection.

IV.1.3.2. Shape detection

Shape recognition is done using Matlab. There are three candidate shapes in the experiments: triangle, square and round. All objects in other shapes would be determined as “unknown” shapes. Figure 24 shows the code processed in MATLAB.

```
function [shape]=ShapeRecognition_right(action,varargin)
    shape = zeros(1,1);
    S2 = imread(lower(action));
    S2 = im2bw(S2);
    figure(4);
    imshow(S2);

    S3 = bwlabel(S2,8);
    figure(5);
    imshow(S3);
    S4 =
regionprops(S3,'MinorAxisLength','MajorAxisLength','Area','Perimeter','Extrema','
Centroid');
    se = strel('disk',5);

    q = S4.MajorAxisLength-S4.MinorAxisLength;
    temp = round(S4.Extrema);
    temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
    temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
    mask = zeros(max(temp(:,1)),max(temp(:,2)));
    for cnt = 1:8
        mask(temp(cnt,1),temp(cnt,2))=1;
    end
    mask2 = imdilate(mask,se);
    [labeled,numObjects] = bwlabel(mask2,8);

    score = numObjects;
    score1 = abs(1 - q/max([S4.MajorAxisLength,S4.MinorAxisLength]));
    score2 = abs(1 - abs(pi*((mean([S4.MajorAxisLength,S4.MinorAxisLength]))/2)^2-
S4.Area)/S4.Area);
    score3 = abs(1 - abs(pi*(max([S4.MajorAxisLength,S4.MinorAxisLength]))-
S4.Perimeter)/S4.Perimeter);
    scoreall=mean([score1;score2;score3]);
    if (score==4)&&(q<=14)
       shape=4;
    end
    if (((score==4)&&(q>14))||(score==3))
       shape=3;
    end
    if (score~=4)&&(score~=3)
        if(scoreall>0.7)
            shape=5;
        else
            shape=0;
        end
    end
    if (scoreall>0.94)&&(score==4)
       shape=5;
    end

    figure(6);
    imshow(S2);
    text(S4.Centroid(1),S4.Centroid(2),num2str(shape),'color','red');

    disp(shape);
end
```

Figure 24: Shape Detection in MATLAB

Shape detection first measures a set of shape properties of the object in the binary image I2 which was the one saved in last step of color detection by *regionprops* function

[29]. The properties used in the experiments are *MinorAxisLength*, *MajorAxisLength*, *Area*, *Perimeter*, *Extrema* and *Centroid*. *Extrema* specifies eight extrema points at the top-left, top-right, right-top, right-bottom, bottom-right, bottom-left, left-bottom and left-top of the region shown in Figure 25.

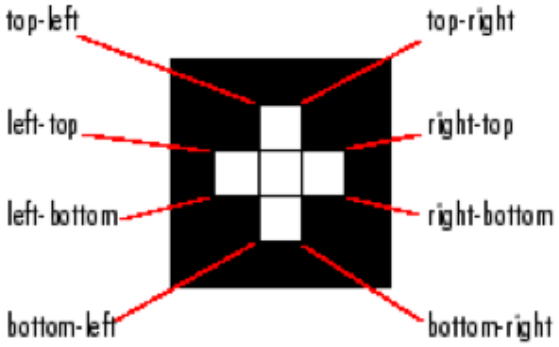

Figure 25: *Extrema* Points of *Regionprops*

Extrema property is used to calculate the number of extrema points for the candidate object. Then remap the coordinates of the objects to a new image which origin is the minimum value of the extrema points. Based on the deficient size of image and noise received from binary image, a few adjacent points might sit around the real extrema points. *imdilate* function is used to group the close-by points together by taking the convolution with a structuring element generated from *strel* function. The structuring element used here is a flat disk-shaped morphological element with radius of 5. If the number of extrema points is 4 and the distance between major and minor axis length is smaller than 14, the shape of the object is determined as square. There are two situations to determine the triangle shape, one is the number of extrema points is 3 and other one is when the number of extrema points is still equal to 4 but its difference between major and minor axis length is bigger than 14 by using *MinorAxisLength, MajorAxisLength* properties. These two conditions would help separate the triangle and

square shape in experiments. Since the cameras are moving and would look at the table horizontally, the round object always looks like the ellipse. *MinorAxisLength, MajorAxisLength, Area* and *Perimeter* properties are used to determine the shape of round. If the object fulfills the three standards presented by variable *score1, score2* and *score3,* the program would consider it as a round object. The variable *score1* measures the longest and shortest length of the ellipse. *score2* measures the calculated area of the object by pi*(diameter/2)^2 with the area directly obtained *Area* property by *regionprops* function. *score3* measures the calculated perimeter by pi*diameter with the obtained *Perimeter* property by *regionprops* function. The equations are listed below:

$$score1 = abs(1- (r_l-r_s)/max(r_l,r_s)) \quad (1)$$

$$score2 = abs(1-abs(pi*mean(r_l,r_s)/2)^2-A)/A \quad (2)$$

$$score3 = abs(1-abs(pi*max(r_l,r_s))-P/P) \quad (3)$$

If the shape of the object is real round, the values of *score1, score2* and *score3* should be 1. The *mean* function calculates the mean of *score1, score2* and *score3*. With the tolerance obtained by experiment results, if the mean score is bigger than 0.7 and the extrema points are neither equal to 4 nor 3, the object is determined as round. Besides, there is another possible situation: if the variable *scoreall*, the mean of *score1, 2* and *3* is bigger than 0.94 and the extrema points are equal to 4, the system would still determine the shape as round. This situation happens when the object was placed in Zone 2 and 4, the shape in grabbed image looks quite like a square based on current considerably low resolution. If the candidate object can not satisfy all the requirements shown above, then the object is considered as unknown which illustrates ISAC have not learned about this object.

IV.1.3.3. Integration with Color and Shape Characteristics

According to the protocol of this thesis shown in Figure 19, filtered by color, only the image containing the nearest object (nearest to the center of each camera) would be transferred to shape detection in MATLAB. In this case, setting ROI (Region of Interest) is a powerful method which always segments the target region for further processing. As shown in Figure 26, ROI is a rectangular subimage of a larger-size image that locates close to top left of the original image.

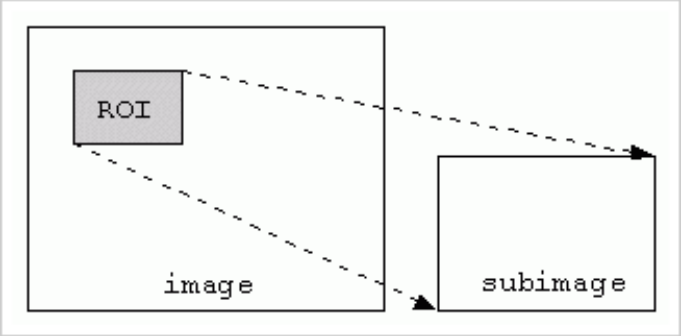

Figure 26: Illustration of Region of Interest

In this case, once set ROI, the program would operate only on the subset of the image provided by the ROI instead of acting on the entire image. Figure 27 shows the integration code processed in Visual Studio 2008.

```
/*********************************
/*Image shape detection in left eye
/*********************************/
can_il=(int)position_sl[0][0];
can_jl=(int)position_sl[0][1];
can_rl=(int)position_sl[0][2];

//detect the shape
IplImage *left_detect = cvLoadImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working now/pics/left_processed.jpg",1);

Sleep(100);

cvSetImageROI(left_detect, cvRect(can_il-can_rl-7, can_jl-can_rl-7, 2*can_rl+10, 2*can_rl+10));

//create destination image: cvGetSize will return the width and the height of ROI
IplImage *candidate_left = cvCreateImage(cvGetSize(left_detect), left_detect->depth, left_detect->nChannels);

//copy subimage
cvCopy(left_detect, candidate_left);
cvResetImageROI(left_detect);

cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/My Documents/MATLAB/candidate_left.jpg",candidate_left);

Sleep(100);

cvNamedWindow("Left Object", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Left Object", 900, 100);
cvShowImage("Left Object",candidate_left);
```

Figure 27: Shape detection in Visual Studio Part

Program first loads the mask image saved in color detection. Function *cvSetImageROI()* sets the rectangle as the ROI whose center is the center of the candidate object and length of side is the diameter of the candidate object with a small tolerance(in this experiment, tolerance was set to 10). *cvCopy* function copies the ROI image to a new image and *cvSaveImage* saves the ROI rectangle for processing in

MATLAB which describe in section IV.1.2.2. *cvResetImageROI* function resets the ROI back to the original image for later new object searching.

IV.1.3.4. Object Locating

Once the candidate object satisfies the conditions of the color and shape, recording the 2D coordinates of the object and mapping that coordinates to the real environment are the following steps. Since active vision requires the visual capture when the cameras are moving, the locations of the target object in the grabbed images are changing all the time. According to this situation, regional mapping method is used. There are twelve markers labeled on the table and the area of table is separated into six subregions. Figure 28 shows the locations of markers.

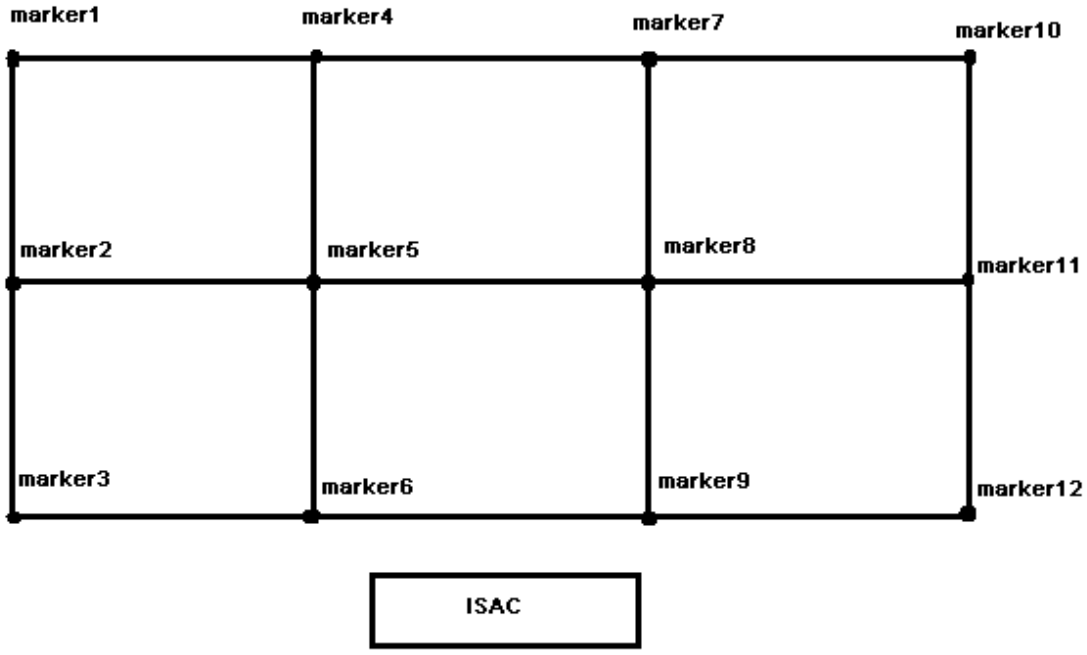

Figure 28: Location of markers.

For each camera, record the 2-D coordinates of these twelve markers in every five degrees along the pan axis. Appendix A shows the recorded 2D coordinates of left and right camera respectively. As shown in Figure 29, seeing the environment from different angles, the field of view of cameras is distorted. Perspective transform method is used to solve this distortion. The transform part is completed in MATLAB.

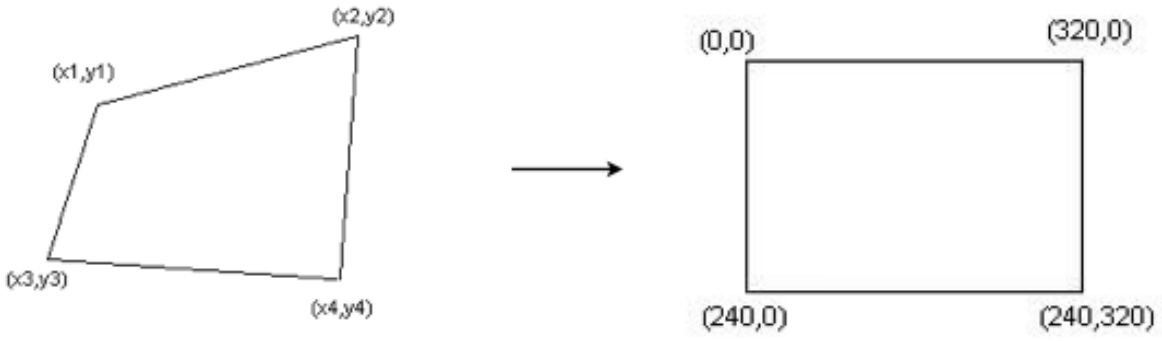

Figure 29: Perspective Transform the coordinates

The locating steps for each camera show in Figure 30.

1. Get the current angle of the pan unit.
2. Look up two 2D coordinates tables with close-by two angles in Table 2 and Table 3.
3. Find out which region the object is located in.
4. Perspective transforms the coordinates to a set of linear changing coordinates.
5. Calculate the 3D coordinates in real environment.

Figure 30: code for object locating

2D Coordinates Mapping

As shown in Figure 31, locating the object in one of six subregions is based on the relationships between the line and dot. It has three different situations: (1) the dot locates on the line; (2) the dot locates above the line; (3) the dot locates under the line.

As shown in Figure 31, locating the object in one of six subregions is based on the relationships between the line and dot. It has three different situations: (1) the dot locates on the line; (2) the dot locates above the line; (3) the dot locates under the line.

```
bool mk=0;
double k1,k2,k3,k4,b1,b2,b3,b4; //k1,k2,k3,k4 are the slopes of the four lines around the rectangle
bool p1,p2,p3,p4;

for(mi=1;mi<19;mi++)
{
	mk=((headAngles0[0]<label_left[mi][0][0])&&(headAngles0[0]>=label_left[mi+1][0][0]));
	if(mk!=0)
	{
		break;
	}
}

int num_angle=0;
num_angle=mi;
mk=0;
bool flag=0;

for(mj=1;mj<9;mj++)
{
	if(((label_left[num_angle][mj][0]!=0)||(label_left[num_angle][mj][1]!=0))&&((label_left[num_angle][mj+1]
[0]!=0)||(label_left[num_angle][mj+1][1]!=0))&&((label_left[num_angle][mj+3][0]!=0)||(label_left[num_angle][mj+
3][1]!=0))&&((label_left[num_angle][mj+4][0]!=0)||(label_left[num_angle][mj+4][1]!=0)!=0))
	{
		k1=(label_left[num_angle][mj+3][1]-
	label_left[num_angle][mj][1])/(label_left[num_angle][mj+3][0]-label_left[num_angle][mj][0]);
		b1=label_left[num_angle][mj][1]-k1*label_left[num_angle][mj][0];
		p1=(position_sl[0][1]>k1*position_sl[0][0]+b1);
		k2=(label_left[num_angle][mj+4][1]-
label_left[num_angle][mj+1][1])/(label_left[num_angle][mj+4][0]-label_left[num_angle][mj+1][0]);
		b2=label_left[num_angle][mj+1][1]-k2*label_left[num_angle][mj+1][0];
		p2=(position_sl[0][1]<=k2*position_sl[0][0]+b2);
		k3=(label_left[num_angle][mj][1]-
label_left[num_angle][mj+1][1])/(label_left[num_angle][mj][0]-label_left[num_angle][mj+1][0]);
		b3=label_left[num_angle][mj][1]-k3*label_left[num_angle][mj][0];
		k4=(label_left[num_angle][mj+3][1]-
label_left[num_angle][mj+4][1])/(label_left[num_angle][mj+3][0]-label_left[num_angle][mj+4][0]);
		b4=label_left[num_angle][mj+3][1]-k4*label_left[num_angle][mj+3][0];

		if((k3>0)&&(k4>0))
		{
			p3=(position_sl[0][1]<k3*position_sl[0][0]+b3);
			p4=(position_sl[0][1]>k4*position_sl[0][0]+b4);
		}
		else if((k3<0)&&(k4>0))
		{
```

```
                    p3=(position_sl[0][1]>k3*position_sl[0][0]+b3);
                    p4=(position_sl[0][1]>k4*position_sl[0][0]+b4);
                }
                else if((k3>0)&&(k4<0))
                {
                    p3=(position_sl[0][1]<k3*position_sl[0][0]+b3);
                    p4=(position_sl[0][1]<k4*position_sl[0][0]+b4);
                }
                else
                {
                    p3=(position_sl[0][1]>k3*position_sl[0][0]+b3);
                    p4=(position_sl[0][1]<k4*position_sl[0][0]+b4);
                }

                mk=p1&&p2&&p3&&p4;

                if(mk!=0)
                {
                    flag++;
                    break;
                }
        }
}

                if(flag!=0)
                {
                    break;
                }
```

Figure 31: Code for locating the subregion

The variables k1, k2, k3, k4 are the slopes of four lines around one region. If the coordinates of the objects are located under the upper line(k1), above the lower line(k2) and also located on the left of right line(k3) and on the right of the left line(k4), the program determines this dot located in this region. Since the direction of y axis in Visual Studio is opposite of the direction in Cartesian coordinate system, the coordinates of the objects should be smaller than the line with slope k2 and bigger than the line with slope k1.

Once the subregion is determined, the program transfers the coordinates of four endpoints of the region along with the coordinates of the object to MATLAB for perspective transform. Figure 32 shows the code for processing in MATLAB.

```
function posi = perspectivetrans(x1,x2,x3,x4,y1,y2,y3,y4,x11,
x12,x13,x14,y11,y12,y13,y14,angle1,angle2,angle,p1,p2)

A1=[-x1 -y1 -1  0      0   0 0 0 0
    0  0   0   -x1    -y1 -1  0 0 0
    -x2 -y2 -1 0 0 0 x2*320 y2*320 320
    0 0 0 -x2 -y2 -1 0 0 0
    -x3 -y3 -1 0 0 0 0 0 0
    0 0 0 -x3 -y3 -1 x3*240 y3*240 240
    -x4 y4 -1 0 0 0 x4*320 y4*320 320
    0 0 0 -x4 -y4 -1 x4*240 y4*240 240];
 [u,s,v1]=svd(A1);
 hr1=v1(:,9)';
 h1=reshape(hr1,3,3)';
 a1=inv(h1)*[p1;p2;1];
 posi1(1)=a1(1)/a1(3);
 posi1(2)=a1(2)/a1(3);

 A2=[-x11 -y11 -1  0      0   0 0 0 0
    0  0   0   -x11    -y11 -1  0 0 0
    -x12 -y12 -1 0 0 0 x12*320 y12*320 320
    0 0 0 -x12 -y12 -1 0 0 0
    -x13 -y13 -1 0 0 0 0 0 0
    0 0 0 -x13 -y13 -1 x13*240 y13*240 240
    -x14 -y14 -1 0 0 0 x14*320 y14*320 320
    0 0 0 -x14 -y14 -1 x14*240 y14*240 240];

 [u,s,v2]=svd(A2);
 hr2=v2(:,9)';
 h2=reshape(hr2,3,3)';
 a2=inv(h2)*[p1;p2;1];
 posi2(1)=a2(1)/a2(3);
 posi2(2)=a2(2)/a2(3);

 posi(1) = posi2(1)+(posi1(1)-posi2(1))/(angle1-angle2)*(angle-angle2);
 posi(2) = posi2(2)+(posi1(2)-posi2(2))/(angle1-angle2)*(angle-angle2);

end
```

Figure 32: Perspective transform in MATLAB

The variables *(x1,y1)*, *(x2,y2)*, *(x3,y3)*, *(x4,y4)* and *(x11,y11)*, *(x12,y12)*, *(x13,y13)*, *(x14,y14)* are the coordinates of the endpoints of the subregions with two close-by angles. 320 and 240 is the size of the original image. *svd()* function calculates the singular value decomposition of A1 and A2 which includes the mapping coordinates. *h1*

and *h2* are the 3*3 homogeneous matrix. Then calculate the inverse of the homogeneous matrix by *inv()* function. Multiplying the coordinates of the object would yield the transformed coordinates. The reason that coordinates shown in Figure 31 are all 3D is that the 3rd dimension enables the system to model more general projections other than the affine transform with 2D coordinates. Then linearly map the coordinates at current angle between two close-by recorded angles if the coordinates of the current angles have not been recorded.

3D Coordinates Calculation

Each marker on the table has its coordinates in real environment with respect to the body of ISAC. The position of object in real environment can be achieved based on the positions of the markers. Figure 33 shows the coordinate system of vision system and the one of ISAC.

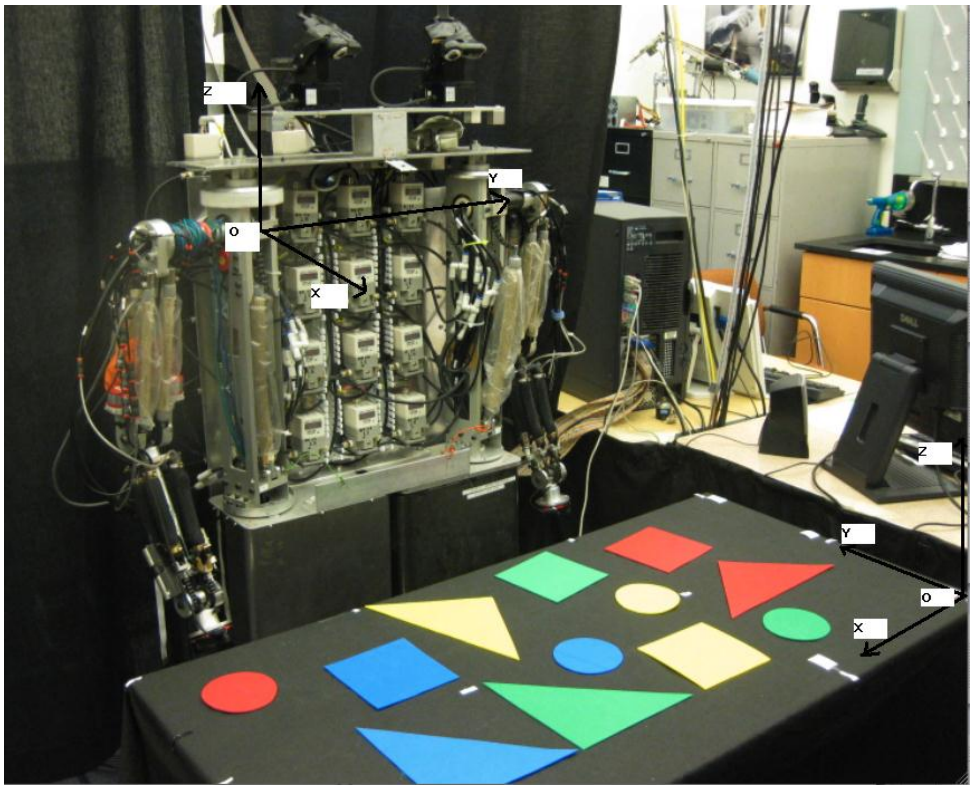

Figure 33: Origin of the coordinate system of ISAC

The transform equations between two coordinate systems are shown below:

$$X_{\text{wrt ISAC}} = X_{\text{upper-right corner}} + Y_{\text{wrt vision system}} / 240 * (X_{\text{upper-left corner}} - X_{\text{upper-right corner}}) \quad (4)$$

$$Y_{\text{wrt ISAC}} = Y_{\text{lower-right corner}} - X_{\text{wrt vision system}} / 320 * (Y_{\text{lower-right corner}} - Y_{\text{upper-right corner}}) \quad (5)$$

$$Z_{\text{wrt ISAC}} = -300 \quad (6)$$

Corners are the four endpoints of the six subregions. The coordinates of 12 markers with respect to ISAC were listed in Appendix B.

Fixation of Cameras

Once the object has been located, last step in vision subsystem is to fixate on the target object with two cameras. The process of fixation still uses linear mapping. Here are the steps: Before the experiments, in every five degrees along horizontal direction,

fixate at the center with each camera in grabbed image; Calculated the coordinates of the center point in real world using vision subsystem; Record the coordinates to Look-Up Table (attached in Appendix C). During the experiments, based on the coordinates calculated by vision subsystem, find the neighboring angles of pan units in previous Look-Up Table $Angle_{n+1}$ and $Angle_n$; Calculate the target angles of pan units when the target object is located at the center point by equation (7).

$$\text{Target Angle} = Angle_{n+1} + 5 * (X_{calcated} - X_{n+1}) / (X_n - X_{n+1}) \quad (7)$$

X_n and X_{n+1} are the recorded coordinates in Look-Up Table associated with $Angle_n$ and $Angle_{n+1}$. Finally, complete the fixation by moving the pan units to the target angles with two cameras.

IV.2. Arm Control System

While vision system is operated on Sally, the control system is processed on Octavia. Once the location of the object is calculated and determined within the workspace of ISAC's right arm, then type the coordinates in Octavia for ISAC's arm to reach by determining appropriate pressure values for each McKibben artificial muscle. The code for these functions is written by Huan,Tan [30]. In this thesis, only ISAC's right arm is used.

The initialization process includes two steps: (1) Initialize the home position of ISAC's right arm which specifies the upper arm holding straight down with the elbow bent forward at 90 degree; (2) Initialize all the air-pressure inside the muscles for activating the proportional controller to direct the movement of arm.

The process of implementation works in this way. The controller first takes in a set of joint angles from home position and calculate desired joint angles by inverse kinematics. Then proportional controller actuates the arms to desired position by adding a proportion of the error signal between the joint angles and the current joint angles. In order to reduce the overshoot of arm velocity, these proportion numbers are quite small. By trying to diminish the error all the time, eventually the arm would reach the target position.

Chapter V

EXPERIMENTS

V.1. Goals and Procedures

The goal of this research was to develop a system so that ISAC can actively scan the environment by moving cameras. The performance of determining how successful this system works lies in whether ISAC can recognize and locate the target object in specific color and shape using its dual cameras, and then use its arm to point it out among many other interferential objects.

Owing to the fact that pan movement of cameras during the active vision process is random within the range from -45 degree to 45 degree while the movement of tilt units were set to -35 degree, there are many kinds of uncertainties that might affect the results, such as the accuracy of color identification and shape detection, the accuracy of mapping coordinates with respect to ISAC and the movement of its arm, and so on. Therefore three categories of experiments were implemented to test the performance of the system according to the different situations that might influence the accuracy of the results. All three categories of experiments were executed after the target object and interferential objects had been placed on the table in front of ISAC. Some of the interferential objects are in the same color with the target object, but in different shapes. Some of them have the same shape but are in different color from the target object. The triangle object used in the experiment is a right-angled triangle whose length of the side is 160mm. The diameter of the round is 110mm and the length of side of square is also

110mm. These sizes of objects were the smallest sizes that camearas can detect determined by the distance from cameras to the table.

Experiment One

Since the images would be distorted when camera focuses on them from different angles, Experiment One was designed to test the performance of the color identification and shape detection subsystems when the target object was placed in different coordinates with respect to camera.

The region on the table where the objects had been placed was separated into six zones based on the locations of markers as shown in Figure 34. Placing the object in different zones allows testing how ISAC can deal with shape recognition from different aspects. Obviously, the images when object was placed in Zone 4 were not as distorted as placed in other zones.

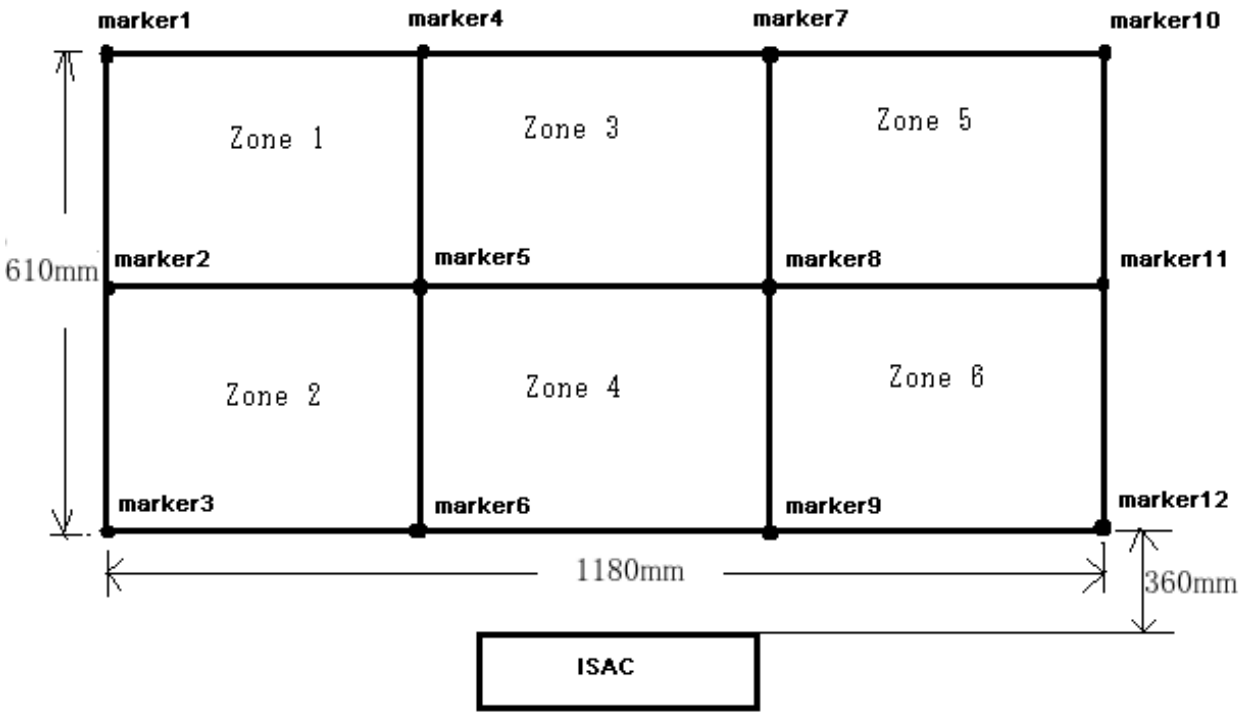

Figure 34: Placement of Zones

In this category of experiment, set the target object with each shape and test the accuracy for each shape in each color (i.e. green, orange, blue and yellow). Since the protocol of shape detection was designed to always detect the nearest object to the center of cameras, when the cameras were moving, moved the target object in different zones and waited for the time that object was located nearest to the center of cameras. Ten trials were taken for detecting each color and each shape when the target object was placed in one zone. Record the times that system successfully recognized the color and shape and calculate the ratio of success. Below is the example when target object was in green color and with a round shape.

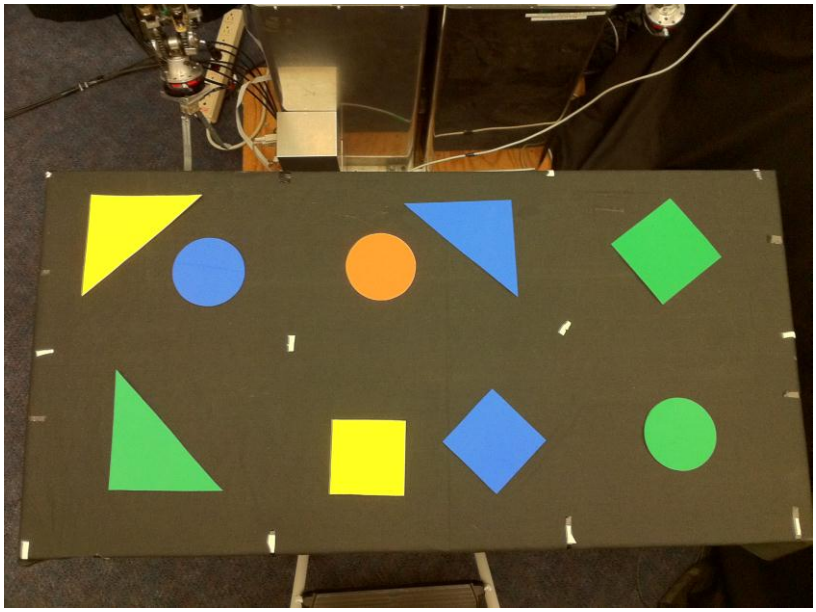

Figure 35: Placement of Objects(Target Object was Green Round in Zone 1)

Figure 35 shows the placement of objects. The target object was placed in Zone 1 while a green square object was in Zone 2 and a green triangle object was in Zone 5. ISAC first scanned and detected the target object(green-round) ten times. Then the target object(green-round) was moved to Zone 3 as shown in Figure 36.

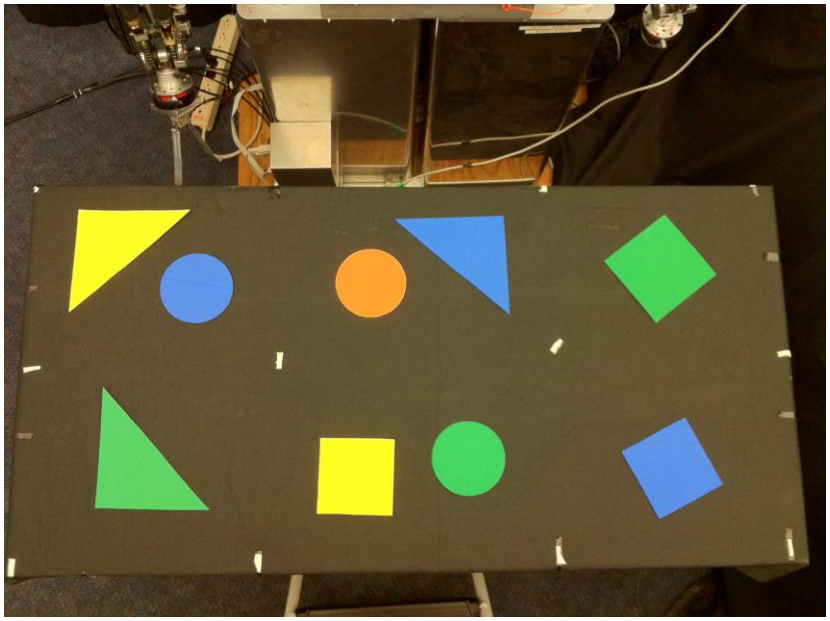

Figure 36: Placement of Objects(Target Object(Green, Round) was placed in Zone 3)

These steps repeated until ISAC scanned and detected the target object (green-round) in all six zones. Then the target object was set to green-triangle, green-square respectively and the steps discussed above were repeated. Since there were four colors, the trials discussed below executed four times accordingly. The posterior probability of each zone was also calculated to better evaluate the performance of vision system.

Experiment Two

Besides the error generated by vision system, reaching movement by arm also bring in some error that would influence the final reaching goal. Therefore the goal of Experiment Two was to test the accuracy of the arm control system. Some interferential objects along with target object were placed in front of ISAC. Since the workspace of ISAC's arm overlays part of Zone 4 and 6, the target object was placed in those zones ten times. If the system successfully recognized the target object, record the coordinates generated by vision system and the positions of the end-effector of ISAC's right arm. Then calculate the error resulted from vision and arm control system separately.

Experiment Three

The goal of Experiment Three was to test the accuracy of the entire system. In this category, the object was randomly placed on the table. The program first recognized the target object, and then mapped the coordinates with respect to ISAC's body, at last fixate the target object with two cameras. If the object is located within the workspace of ISAC, use ISAC' arm to point it out. Otherwise, ISAC remains motionless and there is a

message shown on the screen “The location is out of reach of ISAC”. The distances between the position of the target object in the real world and the end position achieved by the program were also manually measured. Ten trials were executed in this category.

V.2. Results and Analysis

V.2.1. Experiment One

The summaries for Experiment One can be seen in the tables and figures that follow in this section.

V.2.1.1. Analysis for Color Detection

Table 2 to 5 show the results when the target color was set to different colors. The number in “Zones”(specified in Figure 30) represents where the target object was placed. The “Shape” represents the shape of target object. “Correct times” represents the times of the trials that system successfully recognizes the target object within 10 times. The percentage in the table shows the accuracy of shape detection. The full records for each trial are attached in Appendix D. The ratio of accuracy was calculated by the sum of correct times divided by the number of total trials executed.

Zones	**Shape**		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	10	6
Zone 2	**Round**	**Triangle**	**Square**
Correct times	9	10	10
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	10	10
Zone 4	**Round**	**Triangle**	**Square**
Correct times	7	10	9

Zones	Shape		
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	6
Zone 6	**Round**	**Triangle**	**Square**
Correct times	8	10	8
Ratio of Accuracy	83.1%	95.2%	94.2%

Table 2: Result when Target Color was Green

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	8	5
Zone 2	**Round**	**Triangle**	**Square**
Correct times	8	10	6
Zone 3	**Round**	**Triangle**	**Square**
Correct times	8	10	5
Zone 4	**Round**	**Triangle**	**Square**
Correct times	8	10	7
Zone 5	**Round**	**Triangle**	**Square**
Correct times	7	9	6
Zone 6	**Round**	**Triangle**	**Square**
Correct times	10	10	7
Ratio of Accuracy	75%	83.8%	81.8%

Table 3: Result when Target Color was Orange

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	10	7
Zone 2	**Round**	**Triangle**	**Square**
Correct times	9	10	9
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	9	10

Zones	Shape		
Zone 4	**Round**	**Triangle**	**Square**
Correct times	6	10	9
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	6
Zone 6	**Round**	**Triangle**	**Square**
Correct times	10	10	9
Ratio of Accuracy	88.7%	93.7%	90.9%

Table 4: Result when Target Color was Yellow

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	9	7
Zone 2	**Round**	**Triangle**	**Square**
Correct times	7	10	9
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	9	7
Zone 4	**Round**	**Triangle**	**Square**
Correct times	6	10	10
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	7
Zone 6	**Round**	**Triangle**	**Square**
Correct times	8	10	8
Ratio of Accuracy	80%	93.5%	88.9%

Table 5: Result when Target Color was Blue

From the results shown in Table 2 to 5, it can be seen that when the color of target object was set to blue or orange, the ratio of accuracy of shape detection is slightly lower than the one was set to yellow and green. This is due to the fact that shape detection is based on the result image from color detection under certain illumination condition. The selected hue range for colors would influence the erosion

and dilation result of connected components and therefore influence the result image for shape detection. Figure 37 show two binary image samples resulted by blue color and orange color. The left image of Figure 37 shows the binary image when the hue range was set for blue color and the right one shows the result generated by orange color.

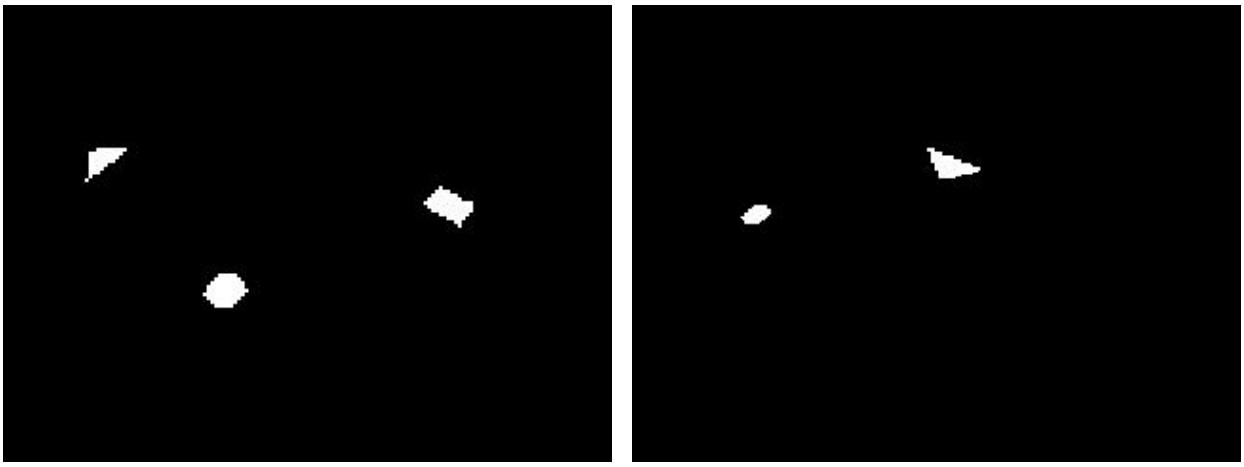

Figure 37: Generated Binary Images (Left:Blue Color, Right:Orange Color)

In the binary images, we can see that the edges around the objects contain some noises that might influence the result of shape detection. In the left image, the candidate object for shape detection on the left is the round one that can not be recognized correctly but in the right image the system successfully detect the candidate object as a triangle. The analysis for specific shape detection will be presented later.

V.2.1.2. Analysis for Shape Detection

Table 6 represents the ratio of accuracy and the posterior probability of color and shape detection when the target object was placed in different zones.

Zones	Shape								
	Round			Triangle			Square		
	Trial Details	Ratio of Accuracy	Posterior Probability	Trial Details	Ratio of Accuracy	Posterior Probability	Trial Details	Ratio of Accuracy	Posterior Probability
Zone 1	40/40	100%	74.1%	37/40	92.5%	94.9%	25/40	62.5%	92.6%

				Square (2) Round (1)			Round (13) Triangle (2)		
Zone 2	33/40 Square (5) Triangle (2)	82.5%	91.7%	40/40	100%	88.9%	34/40 Round (3) Triangle (3)	85%	87.2%
Zone 3	38/40 Square (2)	95%	82.6%	39/40 Square(1)	97.5%	100%	32/40 Round (8)	80%	91.4%
Zone 4	27/40 Square (8) Triangle (5)	67.5%	87.1%	40/40	100%	86.9%	35/40 Round (4) Triangle (1)	87.5%	81.4%
Zone 5	37/40 Square (1) Triangle (2)	92.5%	72.5%	39/40 Square (1)	97.5%	92.9%	25/40 Round (14) Triangle (1)	62.5%	92.6%
Zone 6	36/40 Square (1) Triangle (3)	90%	87.8%	40/40	100%	86.9%	32/40 Round (5) Triangle (3)	80%	96.9%

Table 6: The ratio of accuracy when Target Object was placed in different zones

The data shown in posterior probability columns were calculated by equation (7):

A: event that system decided that found object is the target object.

B: event that found object is target object.

P(B | A | each zone) = P(A | each zone) and P(B | each zone) / P(A | each zone) (7)

In “trial details” columns, the listed shape and the numbers in parenthesis represent to how many times the system had incorrectly determined the object to a specific shape. For example, in Zone 2, when the target object is in round shape, “Square (5)” means that the system mistakenly recognized the object to be a square shape five times.

Table 7 represents the results of overall ratio of accuracy and overall posterior probability for each shape.

Times	Round	Triangle	Square
Ratio of Accuracy	87.9%	97.5%	76.3%
Posterior Probability	81.5%	91.4%	89.3%

Table 7: Overall Accuracy and Posterior Probability for Color and Shape Detection

Triangle Object Detection

Since the placement and the orientation of the objects were randomly selected, from the statistics shown in Table 6 and 7, we can conclude that the accuracy of triangle shape detection is better than other shapes. Figure 38 show some sample images of triangle object that have been successfully detected. Left image in Figure 38 is extracted by yellow color and the middle image is by orange color and right one by blue color.

Figure 38: Sample Images of Triangle Objects

(Left:Yellow, Middle:Orange, Right: Blue)

This result can be explained by the fact that based on the illumination condition in the lab condition and the resolution of grabbed images, the extrema points of triangle shape are much more remarkable and easier for MATLAB to detect.

Round and Square Object

For round and square objects, it is noticed that in Zone 1 and 5, the ratio of accuracy for square object are slightly low but its posterior possibility is quite high while the ratio of accuracy for round object are slightly high but with a low posterior possibility. Figure 39 shows some sample images of square objects.

Figure 39: Sample Images of Square Objects

Since the system was designed to detect the shape by two cameras independently and there is no correlation between the two results, the distances between the object and two cameras are kind of huge. When the camera located with bigger distance firstly

found the square object, the size of object in image would influence the detection result. In this case, if the object was placed in Zone 1 and detected by right camera at first or the object was placed in Zone 5 and detected by left camera firstly, the distance between the object and the camera was larger and therefore the size of object in grabbed image was considerably smaller which become hard to detect.

We can also notice that in Zone 4, the ratio of accuracy for round object was considerably lower but its posterior possibility was high. Figure 40 show sample images when the round object was placed in Zone 4.

Figure 40: Sample Image for Round Detection unsuccessfully recognized

(Left: blue, Right: green)

The objects shown in Figure 40 were incorrectly recognized as square objects. From Figure 40, it seems that under the current resolution of grabbed images, the round shape looks quite similar to square shape. With these low resolutions of the images, pixelwise-based algorithm would much easily mistakenly to determine the round object as a square since they share the same amount of extrema points. In other five zones, the images of the round objects are distorted and their shapes look more like the ellipses, which become easier to distinguish from square object.

Besides those deficiencies, from the result in Table 7, we can conclude that the results were still acceptable in image processing field no matter where the target object had been placed.

V.2.2. Experiment Two

Ten trials were performed to test the accuracy of localization in vision and arm control system separately when the target object was placed within the workspace of ISAC's arm. The results were shown in Table 8.

Object	Coordinates in Real World (X,Y)	Coordinates Calculated by Vision System (X,Y)	Distance between the second and third column (mm)	Coordinates of Hand (X,Y)	Distance from Hand to Target Object (mm)	Error of the Entire System
Yellow, Round	490, -130 (Zone 6)	473, -152	27.8029	420, -120	61.9112	70.7107
Green, Triangle	520, 150 (Zone 4)	503, 187	40.7185	545,130	70.8025	32.0156
Orange, Square	560, -200 (Zone 6)	543, -155	48.1041	500, -150	43.2897	78.1025
Blue, Round	510, 170 (Zone 4)	492, 120	53.1413	450, 150	51.6140	63.2456
Green, Square	470, -190 (Zone 6)	468, -152	38.0526	410, -130	62.0322	84.8528
Yellow, Triangle	550, -50 (Zone 6)	524, -34	30.5287	540, -75	44.0114	26.9258
Blue, Square	490, -90 (Zone 6)	474, -78	20	440, -50	44.0454	64.0312
Green, Round	450, -160 (Zone 6)	510, -140	63.2456	500, -100	41.2311	78.1025
Yellow, Triangle	460, 100 (Zone 4)	424, 57	56.0803	440, 120	65	28.2843
Blue, Square	510, 70 (Zone 4)	538, 52	33.2866	540, 105	53.0377	46.0977

Table 8: Performance of Two Subsystems

Once completing the object localization in vision system, assign the generated 3D coordinates in arm control panel and let arm reach the object, and then record the coordinates with respect to ISAC's body in fifth column. The data shown in fourth column were the distances between the coordinates calculated by vision system and the one in the real world. The data shown in sixth column were the distance from the

end-effector of the arm to the target object. Last column lists the error of the entire system including the vision and arm control system.

Achieved from Table 8, the average error in vision system is 41.096 and its standard deviation is 13.7967. In arm control system, the average error is 53.6975 and the standard deviation is 10.6079. The average error of the entire system is 57.2369 and its standard deviation is 22.147. It can be seen that the errors generated by vision system are relatively smaller than the ones caused by arm control system. This can be explained by the fact that the orientations, distances and the speeds of the arm's movements would bring in irregular errors because the muscles of the arm were pneumatic. The manual measurement would generate some error as well. Vision system and arm control system are relatively independent and there are no direct relationships between the errors generated by vision system and arm control system, we can conclude that two subsystems are not coupled. Compared to the horizontal length of table (1180mm), the error percentage in vision subsystem is 3.48%(calculated from 41.096/1180) while the error percentage in entire system is 4.85%(calculated from 57.2369/1180). With these relatively small errors, it can be concluded that two subsystems successfully completed the tasks of localization.

V.2.3. Experiment Three

Table 9 shows the result of testing the performance of entire system.

Object	Recogniza ble?	Coordinate in Real World (X,Y)	Coordinate Calculated by Vision System (X,Y)	Error Generated by Vision System (mm)	Within the workspace of ISAC?	Reach the object by hand?	Coordin-ate of Hand (X,Y)	Error from Hand to Target Object
Yellow, Round	Yes	810, -130 (Zone 5)	784, -165	43.6005	No	N/A	N/A	N/A

Object	Recognizable?	Coordinate in Real World (X,Y)	Coordinate Calculated by Vision System (X,Y)	Error Generated by Vision System (mm)	Within the workspace of ISAC?	Reach the object by hand?	Coordinate of Hand (X,Y)	Error from Hand to Target Object
Blue, Square	Yes	500, -260 (Zone 2)	465, -222	51.6624	No	N/A	N/A	N/A
Green, Triangle	Yes	560,150 (Zone 4)	569,105	45.8912	No	N/A	N/A	N/A
Blue, Triangle	Yes	540, -240 (Zone 6)	504, -232	36.8782	Yes	Yes	480, -200	72.1110
Yellow, Triangle	Yes	810,260 (Zone 3)	809,208	52.0096	No	N/A	N/A	N/A
Orange, Round	No	774, -200 (Zone 5)	N/A	N/A	N/A	N/A	N/A	N/A
Green, Square	Yes	860,130 (Zone 3)	834,100	39.6989	No	N/A	N/A	N/A
Blue, Round	Yes	750,490 (Zone 1)	740,439	51.9711	No	N/A	N/A	N/A
Yellow, Square	Yes	790,630 (Zone 1)	783,630	6	No	N/A	N/A	N/A
Green, Round	Yes	500, -120 (Zone 6)	510, -186	66.7533	Yes	Yes	465, -150	46.0977

Table 9: Performance of Entire System

In Experiment Three, the system had successfully recognized the target objects in nine trials. This result can be matched with the overall ratio of accuracy and posterior probability results shown in Table 7. From the errors measured in distance listed in fifth column of Table 9, the mean error is 42.4923mm and its standard derivation is 16.2672. The error percentage is 3.6%(calculated from 42.4923/1180). Because the errors could be the summation of mechanical errors when reading the degrees of pan-tilt units, errors from coordinates mapping and the manual measurement caused by human, the results can be considered acceptable. Also if the target object is located within the workspace of ISAC, it successfully reaches the object with its right arm. However,

based on the placement of the target object, the hand reaching movement was completed in only two trials.

Chapter VI

CONCLUSION

The research in this thesis combined computer vision and robot arm control systems to achieve the goals for recognizing and localization of the target object. The results and their analysis are discussed further in section VI.1 and potential solutions according to the problems encountered are discussed in section VI.2.

VI.1. Discussion

The results of Experiment One showed that the color and shape detection subsystem performed reasonably well in detecting the objects in different colors and shapes， especially detecting the triangle objects. The results of detecting square and round objects were less satisfactory. The imprecision behinds the color and shape detection may result from the considerably low resolution of the cameras whose size were 320 by 240 and also the imperfection of the algorithm. Since the shape detection calculated the characteristic extrema points that were operated on pixelwise, the resolution of the cameras was closely related to the detection results. Obviously, improving the resolution would generate much more precise results. However, due to the limits in the way of transporting images in OpenCV and the processing speed of the computer platform, the system can not support larger resolution for dual cameras at the same time which also limited the improvement of the shape detection results. Although the fault-tolerance conditions added improve the accuracy of the algorithm, they also brought the restrictions for wider application of the algorithm on other experiments.

From the results of Experiment Two, ISAC's right arm had successfully reached all the target objects and the errors generated from the vision and arm control subsystems were small. Also it is noticed that since two subsystems were independent, the two kinds of generated errors were not coupled.

In Experiment Three, nine trials in ten had successfully recognized the target object and the differences between the coordinates in real world with respect to ISAC's body and the coordinates calculated by the system were small. These errors can be the cumulated results not only from the mistake generated by the mapping module, but also the mechanical error from reading the angles of pan-tilt units and the measurement error by human.

In conclusion, although there were some deficiencies, it is clear that the color and shape-based detection in vision system were moderately successful and correspondingly the system is judged successful in achieving the aim. With the active vision, the field of view of ISAC was enlarged and ISAC is able to recognize and locate the target object in different color and shape.

VI.2. Future Work

The system has considerably completed the aim for simple color and shape recognition. However, there is plenty of modifications could be added to improve the performance of the system. These improvements may include a much more robust algorithm for color and shape detection, a method for reconstructing the distorted grabbed images, a more robust mapping method for object localization and a more robust system platform.

In the color and shape detection subsystem, this thesis only segmented in hue channel of HSV color space since the segmentation results are already satisfied under current illumination condition in the lab. In order to apply the algorithm for a wider circumstance, segmentation in saturation channel can be added to isolate further regions of interest in the hue image as a masking image. Value channel does not contain any color information, so it used less frequently. Based on different illumination conditions, applying to segmenting in more channels might also generate a more reliable result.

Also the algorithm of shape detection is designed aiming at the specific circumstance of the lab, such as the placement of the table and the settings of the pan-tilt units. In this method, the height of the table and the distance between the table and ISAC would influence the accuracy of the detection accuracy. A feasible method might be to use affine transform [31] or perspective transform of the original twisted image and then detect the shape of the transformed images. This method will bring in more robust result and might reduce the shape presetting of the algorithm.

Thirdly, there are two cameras used in this thesis, however, the processes of the object recognition and localization are completed independently. Later system might embed relativity between the detection results from two cameras in order to obtain a more reliable result. For example, if the system already knows the position of one object, the system can be set automatically to detect using the neighboring camera.

At last, there is a restriction of coordinates mapping and the following object localization since the localization process closely related to the placement of the table. Once the arrangement of the table changes, it needs to record the coordinates of

markers again and repeat the mapping process. Also, the area of the table was separated into six zones. Once the coordinates of the target object lie in the regions of any zone, linear mapping method is used to complete the localization. Although the error generated by mapping system is quite small, basically speaking, increasing the number of zones would improve the accuracy of localization, which would also increase the workload of human. At a high level system, this could be designed as an independent localization system such as fuzzy locating system [32].

Appendix A

LOOK-UP TABLE FOR COORDINATES MAPPING

Shown on Page 41: This code is used for coordinates mapping between the coordinates w.r.t vision system and the one w.r.t ISAC's arm system.

Look-Up Table for left cameras:
//first variable is angel of camera,second variable is no.of dot, third variable is the x,y index;x is 0,y is 1.

```
label_left[0][0][0]=45;     label_left[0][1][0]=202;   label_left[0][1][1]=87;    label_left[0][2][0]=156;
label_left[0][2][1]=124;    label_left[0][3][0]=107;   label_left[0][3][1]=176;   label_left[0][4][0]=279;
label_left[0][4][1]=127;    label_left[0][5][0]=239;   label_left[0][5][1]=180;

label_left[1][0][0]=40;     label_left[1][1][0]=184;   label_left[1][1][1]=83;
label_left[1][2][0]=140;    label_left[1][2][1]=123;   label_left[1][3][0]=94;    label_left[1][3][1]=179;
label_left[1][4][0]=260;    label_left[1][4][1]=118;   label_left[1][5][0]=225;   label_left[1][5][1]=172;

label_left[2][0][0]=35;     label_left[2][1][0]=165;   label_left[2][1][1]=80;
label_left[2][2][0]=124;    label_left[2][2][1]=122;   label_left[2][3][0]=82;    label_left[2][3][1]=183;
label_left[2][4][0]=242;    label_left[2][4][1]=111;   label_left[2][5][0]=211;   label_left[2][5][1]=166;
label_left[2][6][0]=177;    label_left[2][6][1]=237;

label_left[3][0][0]=30;
label_left[3][1][0]=147;    label_left[3][1][1]=79;    label_left[3][2][0]=108;   label_left[3][2][1]=124;
label_left[3][3][0]=69;     label_left[3][3][1]=188;   label_left[3][4][0]=223;   label_left[3][4][1]=105;
label_left[3][5][0]=197;    label_left[3][5][1]=161;   label_left[3][6][0]=167;   label_left[3][6][1]=234

label_left[4][0][0]=25;     label_left[4][1][0]=128;   label_left[4][1][1]=79;
label_left[4][2][0]=92;     label_left[4][2][1]=126;   label_left[4][3][0]=56;
label_left[4][3][1]=194;    label_left[4][4][0]=206;   label_left[4][4][1]=99;    label_left[4][5][0]=183;
label_left[4][5][1]=157;    label_left[4][6][0]=183;   label_left[4][6][1]=231;   label_left[4][7][0]=316;
label_left[4][7][1]=131;    label_left[4][8][0]=313;   label_left[4][8][1]=197;

label_left[5][0][0]=20;     label_left[5][1][0]=109;   label_left[5][1][1]=80;
label_left[5][2][0]=76;     label_left[5][2][1]=130;   label_left[5][3][0]=43;
label_left[5][3][1]=201;    label_left[5][4][0]=187;   label_left[5][4][1]=96;    label_left[5][5][0]=169;
label_left[5][5][1]=153;    label_left[5][6][0]=150;   label_left[5][6][1]=229;   label_left[5][7][0]=296;
label_left[5][7][1]=118;    label_left[5][8][0]=297;   label_left[5][8][1]=183;

label_left[6][0][0]=15;     label_left[6][1][0]=91;    label_left[6][1][1]=82;
label_left[6][2][0]=59;     label_left[6][2][1]=134;   label_left[6][3][0]=30;
label_left[6][3][1]=208;    label_left[6][4][0]=170;   label_left[6][4][1]=93;    label_left[6][5][0]=155;
label_left[6][5][1]=151;    label_left[6][6][0]=141;   label_left[6][6][1]=228;   label_left[6][7][0]=276;
label_left[6][7][1]=108;    label_left[6][8][0]=281;   label_left[6][8][1]=171;

label_left[7][0][0]=10;
label_left[7][1][0]=72;     label_left[7][1][1]=85;    label_left[7][2][0]=42;
label_left[7][2][1]=141;    label_left[7][3][0]=18;    label_left[7][3][1]=218;   label_left[7][4][0]=152;
label_left[7][4][1]=92;     label_left[7][5][0]=142;   label_left[7][5][1]=150;   label_left[7][6][0]=132;
label_left[7][6][1]=228;    label_left[7][7][0]=256;   label_left[7][7][1]=100;   label_left[7][8][0]=265;
label_left[7][8][1]=161;

label_left[8][0][0]=5;      label_left[8][1][0]=52;    label_left[8][1][1]=90;
label_left[8][2][0]=26;     label_left[8][2][1]=148;   label_left[8][3][0]=6;
label_left[8][3][1]=228;    label_left[8][4][0]=134;   label_left[8][4][1]=91;    label_left[8][5][0]=127;
```

```
label_left[8][5][1]=151;    label_left[8][6][0]=122;    label_left[8][6][1]=229;    label_left[8][7][0]=237;
label_left[8][7][1]=93;                label_left[8][8][0]=249;    label_left[8][8][1]=152;    label_left[8][9][0]=275;
label_left[8][9][1]=233;    label_left[8][10][0]=319;   label_left[8][10][1]=96;

label_left[9][0][0]=0;                 label_left[9][1][0]=31;                 label_left[9][1][1]=95;
label_left[9][2][0]=9;                 label_left[9][2][1]=157;    label_left[9][4][0]=116;    label_left[9][4][1]=91;
label_left[9][5][0]=113;    label_left[9][5][1]=152;    label_left[9][6][0]=113;    label_left[9][6][1]=229;
label_left[9][7][0]=218;    label_left[9][7][1]=88;                 label_left[9][8][0]=233;
label_left[9][8][1]=144;    label_left[9][9][0]=263;    label_left[9][9][1]=221;    label_left[9][10][0]=296;
label_left[9][10][1]=84;

label_left[10][0][0]=-5;    label_left[10][1][0]=12;    label_left[10][1][1]=103;   label_left[10][4][0]=98;
label_left[10][4][1]=93;    label_left[10][5][0]=98;    label_left[10][5][1]=153;   label_left[10][6][0]=104;
label_left[10][6][1]=231;   label_left[10][7][0]=199;   label_left[10][7][1]=83;    label_left[10][8][0]=218;
label_left[10][8][1]=137;   label_left[10][9][0]=251;               label_left[10][9][1]=211;   label_left[10][10][0]=275;
label_left[10][10][1]=76;   label_left[10][11][0]=306;  label_left[10][11][1]=124;

label_left[11][0][0]=-10;   label_left[11][4][0]=80;    label_left[11][4][1]=96;    label_left[11][5][0]=85;
label_left[11][5][1]=157;   label_left[11][6][0]=95;    label_left[11][6][1]=235;   label_left[11][7][0]=180;
label_left[11][7][1]=80;    label_left[11][8][0]=202;   label_left[11][8][1]=132;   label_left[11][9][0]=239;
label_left[11][9][1]=202;   label_left[11][10][0]=253;              label_left[11][10][1]=68;   label_left[11][11][0]=287;
label_left[11][11][1]=113;

label_left[12][0][0]=-15;   label_left[12][4][0]=61;    label_left[12][4][1]=100;   label_left[12][5][0]=70;\
label_left[12][5][1]=161;   label_left[12][6][0]=86;    label_left[12][6][1]=239;   label_left[12][7][0]=162;
label_left[12][7][1]=78;    label_left[12][8][0]=186;   label_left[12][8][1]=128;   label_left[12][9][0]=226;
label_left[12][9][1]=194;   label_left[12][10][0]=231;              label_left[12][10][1]=62;   label_left[12][11][0]=266;
label_left[12][11][1]=104;  label_left[12][12][0]=315;  label_left[12][12][1]=161;

label_left[13][0][0]=-20;   label_left[13][4][0]=42;    label_left[13][4][1]=105;   label_left[13][5][0]=55;
label_left[13][5][1]=167;   label_left[13][7][0]=143;   label_left[13][7][1]=77;    label_left[13][8][0]=170;
label_left[13][8][1]=125;   label_left[13][9][0]=214;   label_left[13][9][1]=187;   label_left[13][10][0]=210;
label_left[13][10][1]=57;   label_left[13][11][0]=247;              label_left[13][11][1]=96;   label_left[13][12][0]=297;
label_left[13][12][1]=147;

label_left[14][0][0]=-25;   label_left[14][4][0]=23;    label_left[14][4][1]=112;   label_left[14][5][0]=41;
label_left[14][5][1]=174;   label_left[14][7][0]=124;   label_left[14][7][1]=77;    label_left[14][8][0]=154;
label_left[14][8][1]=123;   label_left[14][9][0]=201;   label_left[14][9][1]=181;   label_left[14][10][0]=190;
label_left[14][10][1]=54;   label_left[14][11][0]=227;              label_left[14][11][1]=90;   label_left[14][12][0]=278;
label_left[14][12][1]=137;

label_left[15][0][0]=-30;   label_left[15][4][0]=4;                 label_left[15][4][1]=121;   label_left[15][5][0]=26;
label_left[15][5][1]=182;   label_left[15][7][0]=105;   label_left[15][7][1]=78;    label_left[15][8][0]=139;
label_left[15][8][1]=122;   label_left[15][9][0]=188;   label_left[15][9][1]=176;   label_left[15][10][0]=169;
label_left[15][10][1]=51;   label_left[15][11][0]=208;              label_left[15][11][1]=85;   label_left[15][12][0]=260;
label_left[15][12][1]=128;

label_left[16][0][0]=-35;   label_left[16][5][0]=11;    label_left[16][5][1]=191;   label_left[16][7][0]=86;
label_left[16][7][1]=81;    label_left[16][8][0]=123;   label_left[16][8][1]=122;   label_left[16][9][0]=175;
label_left[16][9][1]=172;   label_left[16][10][0]=149;  label_left[16][10][1]=50;   label_left[16][11][0]=190;
label_left[16][11][1]=81;   label_left[16][12][0]=242;              label_left[16][12][1]=119;

label_left[17][0][0]=-40;   label_left[17][7][0]=66;    label_left[17][7][1]=84;    label_left[17][8][0]=107;
label_left[17][8][1]=125;   label_left[17][9][0]=162;   label_left[17][9][1]=169;   label_left[17][10][0]=129;
label_left[17][10][1]=50;   label_left[17][11][0]=171;  label_left[17][11][1]=78;   label_left[17][12][0]=225;
label_left[17][12][1]=114;

label_left[18][0][0]=-45;   label_left[18][7][0]=46;    label_left[18][7][1]=88;    label_left[18][8][0]=91;
```

label_left[18][8][1]=126; label_left[18][9][0]=150; label_left[18][9][1]=168; label_left[18][10][0]=108;
label_left[18][10][1]=51; label_left[18][11][0]=152; label_left[18][11][1]=77; label_left[18][12][0]=207;
label_left[18][12][1]=108;

Look-Up Table for right cameras:

label_right[0][0][0]=45; label_right[0][1][0]=167; label_right[0][1][1]=42; label_right[0][2][0]=119;
label_right[0][2][1]=73; label_right[0][3][0]=70; label_right[0][3][1]=113; label_right[0][4][0]=230;
label_right[0][4][1]=78; label_right[0][5][0]=185; label_right[0][5][1]=119; label_right[0][6][0]=136;
label_right[0][6][1]=167; label_right[0][8][0]=294; label_right[0][8][1]=192;

label_right[1][0][0]=40; label_right[1][1][0]=147; label_right[1][1][1]=42; label_right[1][2][0]=101;
label_right[1][2][1]=75; label_right[1][3][0]=53; label_right[1][3][1]=119; label_right[1][4][0]=211;
label_right[1][4][1]=74; label_right[1][5][0]=170; label_right[1][5][1]=117; label_right[1][6][0]=125;
label_right[1][6][1]=168; label_right[1][7][0]=312; label_right[1][7][1]=124; label_right[1][8][0]=281;
label_right[1][8][1]=180;

label_right[2][0][0]=35; label_right[2][1][0]=126; label_right[2][1][1]=42; label_right[2][2][0]=82;
label_right[2][2][1]=78; label_right[2][3][0]=36; label_right[2][3][1]=126; label_right[2][4][0]=192;
label_right[2][4][1]=69; label_right[2][5][0]=155; label_right[2][5][1]=115; label_right[2][6][0]=113;
label_right[2][6][1]=170; label_right[2][7][0]=293; label_right[2][7][1]=112; label_right[2][8][0]=267;
label_right[2][8][1]=169;

label_right[3][0][0]=30; label_right[3][1][0]=106; label_right[3][1][1]=43; label_right[3][2][0]=63;
label_right[3][2][1]=82; label_right[3][3][0]=18; label_right[3][3][1]=134; label_right[3][4][0]=174;
label_right[3][4][1]=68; label_right[3][5][0]=139; label_right[3][5][1]=114; label_right[3][6][0]=101;
label_right[3][6][1]=173; label_right[3][7][0]=274; label_right[3][7][1]=102; label_right[3][8][0]=253;
label_right[3][8][1]=160;

label_right[4][0][0]=25; label_right[4][1][0]=86; label_right[4][1][1]=46; label_right[4][2][0]=44;
label_right[4][2][1]=88; label_right[4][3][0]=2; label_right[4][3][1]=144; label_right[4][4][0]=156;
label_right[4][4][1]=66; label_right[4][5][0]=125; label_right[4][5][1]=114; label_right[4][6][0]=90;
label_right[4][6][1]=176; label_right[4][7][0]=256; label_right[4][7][1]=95; label_right[4][8][0]=239;
label_right[4][8][1]=153; label_right[4][9][0]=225; label_right[4][9][1]=233;

label_right[5][0][0]=20; label_right[5][1][0]=64; label_right[5][1][1]=49; label_right[5][2][0]=24;
label_right[5][2][1]=95; label_right[5][4][0]=137; label_right[5][4][1]=64; label_right[5][5][0]=109;
label_right[5][5][1]=117; label_right[5][6][0]=78; label_right[5][6][1]=182; label_right[5][7][0]=237;
label_right[5][7][1]=87; label_right[5][8][0]=225; label_right[5][8][1]=146; label_right[5][9][0]=217;
label_right[5][9][1]=226;

label_right[6][0][0]=15; label_right[6][1][0]=43; label_right[6][1][1]=55; label_right[6][2][0]=5;
label_right[6][2][1]=103; label_right[6][4][0]=118; label_right[6][4][1]=66; label_right[6][5][0]=93;
label_right[6][5][1]=119; label_right[6][6][0]=66; label_right[6][6][1]=187; label_right[6][7][0]=219;
label_right[6][7][1]=83; label_right[6][8][0]=211; label_right[6][8][1]=139; label_right[6][9][0]=208;
label_right[6][9][1]=219; label_right[6][10][0]=306; label_right[6][10][1]=97; label_right[6][11][0]=315;
label_right[6][11][1]=157;

label_right[7][0][0]=10; label_right[7][1][0]=21; label_right[7][1][1]=60; label_right[7][4][0]=100;
label_right[7][4][1]=68; label_right[7][5][0]=78; label_right[7][5][1]=124; label_right[7][6][0]=54;
label_right[7][6][1]=194; label_right[7][7][0]=201; label_right[7][7][1]=79; label_right[7][8][0]=197;
label_right[7][8][1]=135; label_right[7][9][0]=200; label_right[7][9][1]=216; label_right[7][10][0]=286;
label_right[7][10][1]=86; label_right[7][11][0]=298; label_right[7][11][1]=144; label_right[7][12][0]=318;
label_right[7][12][1]=226;

label_right[8][0][0]=5; label_right[8][4][0]=81; label_right[8][4][1]=71; label_right[8][5][0]=62;
label_right[8][5][1]=129; label_right[8][6][0]=43; label_right[8][6][1]=201; label_right[8][7][0]=183;
label_right[8][7][1]=75; label_right[8][8][0]=183; label_right[8][8][1]=132; label_right[8][9][0]=191;

```
label_right[8][9][1]=210;  label_right[8][10][0]=266;label_right[8][10][1]=77;  label_right[8][11][0]=284;
label_right[8][11][1]=133;label_right[8][12][0]=308;label_right[8][12][1]=213;

label_right[9][0][0]=0;          label_right[9][4][0]=62;    label_right[9][4][1]=76;    label_right[9][5][0]=46;
label_right[9][5][1]=136;  label_right[9][6][0]=32;    label_right[9][6][1]=211;  label_right[9][7][0]=166;
label_right[9][7][1]=74;    label_right[9][8][0]=169;  label_right[9][8][1]=129;  label_right[9][9][0]=181;
label_right[9][9][1]=206;  label_right[9][10][0]=246;label_right[9][10][1]=71;  label_right[9][11][0]=265;
label_right[9][11][1]=124;label_right[9][12][0]=295;label_right[9][12][1]=198;

label_right[10][0][0]=-5;   label_right[10][4][0]=42;  label_right[10][4][1]=81;  label_right[10][5][0]=30;
label_right[10][5][1]=143;label_right[10][6][0]=22;  label_right[10][6][1]=220;label_right[10][7][0]=147;
label_right[10][7][1]=72;  label_right[10][8][0]=155;label_right[10][8][1]=128;label_right[10][9][0]=172;
label_right[10][9][1]=203;label_right[10][10][0]=227;          label_right[10][10][1]=66;
label_right[10][11][0]=249;          label_right[10][11][1]=116;          label_right[10][12][0]=281;
label_right[10][12][1]=187;

label_right[11][0][0]=-10; label_right[11][4][0]=22;  label_right[11][4][1]=88;  label_right[11][5][0]=15;
label_right[11][5][1]=152;label_right[11][6][0]=11;  label_right[11][6][1]=232;label_right[11][7][0]=129;
label_right[11][7][1]=73;  label_right[11][8][0]=140;label_right[11][8][1]=127;label_right[11][9][0]=163;
label_right[11][9][1]=201;label_right[11][10][0]=207;          label_right[11][10][1]=61;
label_right[11][11][0]=232;          label_right[11][11][1]=109;          label_right[11][12][0]=268;
label_right[11][12][1]=176;

label_right[12][0][0]=-15; label_right[12][4][0]=2;    label_right[12][4][1]=97;  label_right[12][5][0]=0;
label_right[12][5][1]=162;label_right[12][7][0]=112;label_right[12][7][1]=75;  label_right[12][8][0]=126;
label_right[12][8][1]=128;label_right[12][9][0]=153;label_right[12][9][1]=200;label_right[12][10][0]=188;
label_right[12][10][1]=57;label_right[12][11][0]=216;          label_right[12][11][1]=104;
label_right[12][12][0]=255;          label_right[12][12][1]=166;

label_right[13][0][0]=-20; label_right[13][7][0]=93;  label_right[13][7][1]=76;  label_right[13][8][0]=111;
label_right[13][8][1]=130;label_right[13][9][0]=144;label_right[13][9][1]=199;label_right[13][10][0]=169;
label_right[13][10][1]=56;label_right[13][11][0]=199;          label_right[13][11][1]=99;
label_right[13][12][0]=241;          label_right[13][12][1]=158;

label_right[14][0][0]=-25; label_right[14][7][0]=74;  label_right[14][7][1]=80;  label_right[14][8][0]=97;
label_right[14][8][1]=133;label_right[14][9][0]=134;label_right[14][9][1]=200;label_right[14][10][0]=149;
label_right[14][10][1]=54;label_right[14][11][0]=182;          label_right[14][11][1]=96;
label_right[14][12][0]=227;          label_right[14][12][1]=151;

label_right[15][0][0]=-30; label_right[15][7][0]=55;  label_right[15][7][1]=85;  label_right[15][8][0]=83;
label_right[15][8][1]=137;label_right[15][9][0]=125;label_right[15][9][1]=200;label_right[15][10][0]=130;
label_right[15][10][1]=54;label_right[15][11][0]=165;          label_right[15][11][1]=94;
label_right[15][12][0]=213;          label_right[15][12][1]=145;

label_right[16][0][0]=-35; label_right[16][7][0]=36;  label_right[16][7][1]=91;  label_right[16][8][0]=68;
label_right[16][8][1]=143;label_right[16][9][0]=116;label_right[16][9][1]=203;label_right[16][10][0]=111;
label_right[16][10][1]=56;label_right[16][11][0]=149;          label_right[16][11][1]=93;
label_right[16][12][0]=199;          label_right[16][12][1]=141;

label_right[17][0][0]=-40; label_right[17][7][0]=16;  label_right[17][7][1]=99;  label_right[17][8][0]=53;
label_right[17][8][1]=149;label_right[17][9][0]=106;label_right[17][9][1]=205;label_right[17][10][0]=91;
label_right[17][10][1]=58;label_right[17][11][0]=132;          label_right[17][11][1]=93;
label_right[17][12][0]=186;          label_right[17][12][1]=137;

label_right[18][0][0]=-45; label_right[18][8][0]=38;  label_right[18][8][1]=155;label_right[18][9][0]=97;
label_right[18][9][1]=209;label_right[18][10][0]=71;label_right[18][10][1]=62;label_right[18][11][0]=116;
label_right[18][11][1]=94;label_right[18][12][0]=171;          label_right[18][12][1]=133;
```

Appendix B

COORDINATES OF MARKERS W.R.T ISAC

Shown on Page 46: Specified the coordinates of 12 markers with respect to ISAC in mapping module.

Marker 1: x = 940, y = 810;

Marker 2: x = 640, y = 830;

Marker 3: x = 340, y = 810;

Marker 4: x = 950, y = 450;

Marker 5: x = 640, y = 450;

Marker 6: x = 360, y = 450;

Marker 7: x = 970, y = 0;

Marker 8: x = 650, y = 20;

Marker 9: x = 360, y = 0;

Marker 10: x = 970, y = -380;

Marker 11: x = 656, y = -370;

Marker 12: x = 360, y = -370;

Appendix C

LOOK-UP TABLE OF FIXATION

Shown on Page 47: This code is used for fixation of two cameras when found the target object.

```
//first variable is index for angel of camera,second variable is the x,y index;x is 0,y is 1.
map_left[0][0]=45;          map_left[0][1]=800;
map_left[1][0]=40;          map_left[1][1]=760;
map_left[2][0]=35;          map_left[2][1]=640;
map_left[3][0]=30;          map_left[3][1]=620;
map_left[4][0]=25;          map_left[4][1]=560;
map_left[5][0]=20;          map_left[5][1]=520;
map_left[6][0]=15;          map_left[6][1]=455;
map_left[7][0]=10;          map_left[7][1]=395;
map_left[8][0]=5;           map_left[8][1]=310;
map_left[9][0]=0;           map_left[9][1]=290;
map_left[10][0]=-5;         map_left[10][1]=215;
map_left[11][0]=-10;        map_left[11][1]=160;
map_left[12][0]=-15;        map_left[12][1]=120;
map_left[13][0]=-20;        map_left[13][1]=60;
map_left[14][0]=-25;        map_left[14][1]=0;
map_left[15][0]=-30;        map_left[15][1]=-70;
map_left[16][0]=-35;        map_left[16][1]=-110;
map_left[17][0]=-40;        map_left[17][1]=-145;
map_left[18][0]=-45;        map_left[18][1]=-450;

map_right[0][0]=45;         map_right[0][1]=800;
map_right[1][0]=40;         map_right[1][1]=470;
map_right[2][0]=35;         map_right[2][1]=450;
map_right[3][0]=30;         map_right[3][1]=360;
map_right[4][0]=25;         map_right[4][1]=310;
map_right[5][0]=20;         map_right[5][1]=245;
map_right[6][0]=15;         map_right[6][1]=200;
map_right[7][0]=10;         map_right[7][1]=120;
map_right[8][0]=5;          map_right[8][1]=80;
map_right[9][0]=0;          map_right[9][1]=60;
map_right[10][0]=-5;        map_right[10][1]=-20;
map_right[11][0]=-10;       map_right[11][1]=-50;
map_right[12][0]=-15;       map_right[12][1]=-100;
map_right[13][0]=-20;       map_right[13][1]=-170;
map_right[14][0]=-25;       map_right[14][1]=-200;
map_right[15][0]=-30;       map_right[15][1]=-220;
map_right[16][0]=-35;       map_right[16][1]=-220;
map_right[17][0]=-40;       map_right[17][1]=-220;
map_right[18][0]=-45;       map_right[18][1]=-220;
```

Appendix D

EXHAUSTIVE RESULT FOR COLOR AND SHAPE DETECTION

Shown on Page 54.

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	10	6 (wrongly recognized as round, 4 times)
Zone 2	**Round**	**Triangle**	**Square**
Correct times	9 (wrongly recognized as square, 1 times)	10	10
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	10	10
Zone 4	**Round**	**Triangle**	**Square**
Correct times	7 (wrongly recognized as square, 1 times; wrongly recognized as triangle, 2 times)	10	9 (wrongly recognized as round, 1 times)
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	6 (wrongly recognized as round, 4 times)
Zone 6	**Round**	**Triangle**	**Square**
Correct times	8 (wrongly recognized as square, 1 times; wrongly recognized as triangle, 1 times)	10	8 (wrongly recognized as round, 2 times)

Result when Target Color was Green

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	8 (wrongly recognized as square, 2 times)	5 (wrongly recognized as round, 3 times; wrongly recognized as triangle, 2 times)

Zones	Shape		
Zone 2	**Round**	**Triangle**	**Square**
Correct times	8 (wrongly recognized as square, 1 times; wrongly recognized as triangle, 1 times)	10	6 (wrongly recognized as round, 2 times; wrongly recognized as triangle, 2 times)
Zone 3	**Round**	**Triangle**	**Square**
Correct times	8 (wrongly recognized as square, 2 times)	10	5 (wrongly recognized as round, 5 times)
Zone 4	**Round**	**Triangle**	**Square**
Correct times	8 (wrongly recognized as square, 1 times; wrongly recognized as triangle, 1 times)	10	7 (wrongly recognized as round, 2 times; wrongly recognized as triangle, 1 times)
Zone 5	**Round**	**Triangle**	**Square**
Correct times	7 (wrongly recognized as square, 1 times; wrongly recognized as triangle, 2 times)	9	6 (wrongly recognized as round, 4 times)
Zone 6	**Round**	**Triangle**	**Square**
Correct times	10	10	7 (wrongly recognized as round, 1 times; wrongly recognized as triangle, 2 times)

Result when Target Color was Orange

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	10	7 (wrongly recognized as round, 3 times)
Zone 2	**Round**	**Triangle**	**Square**
Correct times	9 (wrongly recognized as square, 1 times)	10	9 (wrongly recognized as triangle, 1 times)
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	9 (wrongly recognized as square, 1 times)	10

Zones	Shape		
Zone 4	**Round**	**Triangle**	**Square**
Correct times	6 (wrongly recognized as square, 3 times; wrongly recognized as triangle, 1 times)	10	9 (wrongly recognized as round, 1 times)
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	6 (wrongly recognized as round, 3 times; wrongly recognized as triangle, 1 times)
Zone 6	**Round**	**Triangle**	**Square**
Correct times	10	10	9 (wrongly recognized as triangle, 1 times)

Result when Target Color was Yellow

Zones	Shape		
Zone 1	**Round**	**Triangle**	**Square**
Correct times	10	9 (wrongly recognized as round, 1 times)	7 (wrongly recognized as round, 3 times)
Zone 2	**Round**	**Triangle**	**Square**
Correct times	7 (wrongly recognized as square, 2 times; wrongly recognized as triangle, 1 times)	10	9 (wrongly recognized as round, 1 times)
Zone 3	**Round**	**Triangle**	**Square**
Correct times	10	9 (wrongly recognized as square, 1 times)	7 (wrongly recognized as round, 3 times)
Zone 4	**Round**	**Triangle**	**Square**
Correct times	6 (wrongly recognized as square, 3 times; wrongly recognized as triangle, 1 times)	10	10
Zone 5	**Round**	**Triangle**	**Square**
Correct times	10	10	7 (wrongly recognized as round, 3 times)

Zones	Shape		
Zone 6	**Round**	**Triangle**	**Square**
Correct times	8 (wrongly recognized as triangle, 2 times)	10	8 (wrongly recognized as round, 2 times)

Result when Target Color was Blue

Appendix E

PROGRAM CODE

Code for vision system processed in Visual Studio 2008:

```
// DualCamera.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "highgui.h"
#include "cv.h"
#include "cvaux.h"
#include "CameraHead.h"
#include <math.h>
#include "engine.h"
#include <stdlib.h>
#include "mat.h"
#include "time.h"
#include <iostream>
#include <stdio.h>

//variables for getConnectedComps
int compmax_left=0;
int compmax_right=0;
int compcap=40;                    //should be the same size as the length of Comps below
int comptot_left=0;
int comptot_right=0;
static CvConnectedComp ** Comps_left = new CvConnectedComp*[40];          //holder for Connected Components
static CvConnectedComp ** Comps_right = new CvConnectedComp*[40];         //holder for Connected Components

char coords[] = "C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working now/coords.txt";//objects coordinates
char loadcoords[] = "C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working now/loadcoords.txt";
                    //objects coordinates

void getConnectedCompsleft(IplImage *image, CvConnectedComp ** Comps);
void getConnectedCompsright(IplImage *image, CvConnectedComp ** Comps);

using namespace std;

double headAngles0[4];
double headAngles[4];

double Number_rand;
double position[10][3];
double position_r[5][4];
double position_sr[5][4]; //position by sorted coordinates

double positionl[10][3];
double position_l[5][4];
double position_sl[5][4]; //position by sorted coordinates

double label_left[19][13][2];
double label_right[19][13][2];
double d3mark[12][2];
double d3coor[2];

Engine *ep;
mxArray *traleft=mxCreateDoubleMatrix(1,1, mxREAL);
mxArray *tra_right=mxCreateDoubleMatrix(1,1, mxREAL);

mxArray *POSI=mxCreateDoubleMatrix(1,1, mxREAL);
MATFile *pmat;
BOOL Flagmat;

int main(int argc, _TCHAR* argv[])
```

```
{
    Flagmat = true;
    if(!(ep=engOpen(" \0")))
    {
            fprintf(stderr,"\n Can't start MATLAB engine\n");
            Flagmat = false;
            exit(-1);
    }

    if(Flagmat == false)
    {
            return 0;
    }
    int shape_left;
    int shape_right;

    //determine the colors
    int shapes=0;
    int colors[2]={0};

    char color_input[6]= "";
    cout << "Please enter the name of color:";
    cin >> color_input;

    if(strcmp(color_input,"orange") == 0)
    {
            colors[0]=14;
            colors[1]=23;
    }
    else if(strcmp(color_input,"blue") == 0)
    {
            colors[0]=80;
            colors[1]=130;
    }
    else if(strcmp(color_input,"green") == 0)
    {
            colors[0]=40;
            colors[1]=55;
    }
    else if(strcmp(color_input,"yellow") == 0)
    {
            colors[0]=30;
            colors[1]=32;
    }
    else
    {
            printf("The input color is unknown");
    }

    //determine the shape
    char shape_input[10]= "";
    cout << "Please enter the name of shape:";
    cin >> shape_input;

    if(strcmp(shape_input,"triangle") == 0)
    {
            shapes=3;
    }
    else if(strcmp(shape_input,"round") == 0)
    {
            shapes=5;
    }
    else if(strcmp(shape_input,"square") == 0)
    {
            shapes=4;
    }
    else
    {
            printf("'The input shape is unknown");
```

```
}

cout<<"Searching for the "<<color_input<<" color "<< shape_input<<" object"<<endl;

int i,b,k,s,m;
int can_i=0;
int can_j=0;
int can_r=0;
int can_il=0;
int can_jl=0;
int can_rl=0;

int mi,mj;

//coordinate storage
FILE * coordsfile;                                  //file handle to store coordinates of target object
FILE * loadcoordsfile;                              //file handle to store coordinates of wrong objects

double label_left[19][13][2]={0};
double label_right[19][13][2]={0};
double position[10][3]={0};
double position_r[5][4]={0};
double position_sr[5][4]={0};
double positionl[10][3]={0};
double position_l[5][4]={0};
double position_sl[5][4]={0};

CvCapture* capture1 = cvCaptureFromCAM(0);
CvCapture* capture2 = cvCaptureFromCAM(1);

int x= 320;
int y= 240;

printf("Initializing system...\n");

//variables for pan/tilt units
CameraHead hd;      //head object (pan/tilts)
hd.Initialize();
hd.Home();                          //ensure the pan/tilts are at the home position

cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_WIDTH, x);
cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_HEIGHT, y);

cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_WIDTH, x);
cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_HEIGHT, y);

cvNamedWindow("Left", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Left", 100, 400);

cvNamedWindow("Right", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Right", 440, 400);

IplImage* frame1;
IplImage* out1;
IplImage* frame2;
IplImage* out2;

IplImage* frame3;
IplImage* out3;
IplImage* frame4;
IplImage* out4;

IplImage* FrameLeftProcessed;
IplImage* FrameRightProcessed;

char c;
char d;

while(1)
```

```
{
	srand(GetTickCount());

	Number_rand=(double)rand()/((double)RAND_MAX + 1);
	headAngles0[0] = -45+Number_rand*90;;
	headAngles0[1] = -35;
	Number_rand=(double)rand()/((double)RAND_MAX + 1);
	headAngles0[2] = -45+Number_rand*90;
	headAngles0[3] = -35;

	hd.MoveHead(headAngles0);

	cvWaitKey(500);

	while(1)
	{
		frame1 = cvQueryFrame(capture2);
		out1 = 0;

		frame2 = cvQueryFrame(capture1);
		out2 = 0;

		cvShowImage( "Left", frame1);
		cvShowImage( "Right",frame2);

		if(!frame1)
		{
			break;
		}

		cvWaitKey(500);

		c=getchar();

		if(c=='c')
		{
			frame1 = cvQueryFrame( capture2);
			out1 = 0;

			frame2 = cvQueryFrame( capture1);
			out2 = 0;

			cvShowImage( "Left", frame1);
			cvShowImage( "Right",frame2);

			break;
		}

	}

	//Image color detection
	IplImage* hsv1 = cvCreateImage(cvGetSize(frame1), 8, 3);
	IplImage* hue1 = cvCreateImage(cvGetSize(frame1), 8, 1);
	IplImage* sat1 = cvCreateImage(cvGetSize(frame1), 8, 1);
	IplImage* val1 = cvCreateImage(cvGetSize(frame1), 8, 1);
	IplImage* hsv2 = cvCreateImage(cvGetSize(frame2), 8, 3);
	IplImage* hue2 = cvCreateImage(cvGetSize(frame2), 8, 1);
	IplImage* sat2 = cvCreateImage(cvGetSize(frame2), 8, 1);
	IplImage* val2 = cvCreateImage(cvGetSize(frame2), 8, 1);
	IplImage* FrameLeftProcessed = cvCreateImage(cvGetSize(frame1), 8, 1);
	IplImage* FrameRightProcessed = cvCreateImage(cvGetSize(frame2), 8, 1);
	IplConvKernel * selem = cvCreateStructuringElementEx(3,3,1,1,CV_SHAPE_RECT);	//kernel for use with erode/dilate

	//Extract Hue/Sat/Val from BGR Image
	cvCvtColor(frame1, hsv1, CV_BGR2HSV);                                   //convert from BGR to HSV
	cvSplit(hsv1, hue1, sat1, val1, 0);                                     //extract hue/sat/val channels

	//Filter by Hue
	cvInRangeS(hue1, cvScalar(colors[0]), cvScalar(colors[1]), FrameLeftProcessed);	//filter by Hue
```

```
	cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);				//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameLeftProcessed, FrameLeftProcessed, selem, 3);
	cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);				//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameLeftProcessed, FrameLeftProcessed, selem, 3);
	cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);				//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameLeftProcessed, FrameLeftProcessed, selem, 3);
	cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);				//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameLeftProcessed, FrameLeftProcessed, selem, 3);
	cvErode(FrameLeftProcessed,FrameLeftProcessed,selem,3);				//Erode/Dilate maskH to eliminate noise

	//Extract Hue/Sat/Val from BGR Image
	cvCvtColor(frame2, hsv2, CV_BGR2HSV);						//convert from BGR to HSV
	cvSplit(hsv2, hue2, sat2, val2, 0);						//extract hue/sat/val channels

	//Filter by Hue
	cvInRangeS(hue2, cvScalar(colors[0]), cvScalar(colors[1]), FrameRightProcessed);	//filter by Hue
	cvErode(FrameRightProcessed,FrameRightProcessed,selem,3);			//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameRightProcessed, FrameRightProcessed, selem, 3);
	cvErode(FrameRightProcessed,FrameRightProcessed,selem,3);			//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameRightProcessed, FrameRightProcessed, selem, 3);
	cvErode(FrameRightProcessed,FrameRightProcessed,selem,3);			//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameRightProcessed, FrameRightProcessed, selem, 3);
	cvErode(FrameRightProcessed,FrameRightProcessed,selem,3);			//Erode/Dilate maskH to eliminate noise
	cvDilate(FrameRightProcessed, FrameRightProcessed, selem, 3);
	cvErode(FrameRightProcessed,FrameRightProcessed,selem,3);			//Erode/Dilate maskH to eliminate noise

	cvNamedWindow("Left_Processed", CV_WINDOW_AUTOSIZE);
	cvMoveWindow("Left_Processed", 100, 600);
	cvShowImage( "Left_Processed", FrameLeftProcessed);

	cvNamedWindow("Right_Processed", CV_WINDOW_AUTOSIZE);
	cvMoveWindow("Right_Processed", 440, 600);
	cvShowImage( "Right_Processed",FrameRightProcessed);

	cvWaitKey(100);

	cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working now/pics/left_processed.jpg",
FrameLeftProcessed);
	cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working now/pics/right_processed.jpg",
FrameRightProcessed);

	//Find all the positive objects in left eye
	getConnectedCompsleft(FrameLeftProcessed,Comps_left);		//retrieve a list of all connected components in filtered image

	m=0;
	if (comptot_left != 0)
	{
		for (int j = 0; j < comptot_left; j++)
		{
			//get center
			positionl[j][0] = Comps_left[j]->rect.x + Comps_left[j]->rect.width/2;
			positionl[j][1] = Comps_left[j]->rect.y + Comps_left[j]->rect.height/2;

			//get radius
			if (Comps_left[j]->rect.width > Comps_left[j]->rect.height)
			{
				positionl[j][2] = Comps_left[j]->rect.width/2;
			}
			else
			{
				positionl[j][2] = Comps_left[j]->rect.height/2;
			}

			if(positionl[j][2]>7)
			{

				double q;

				position_l[m][0]=positionl[j][0];
```

```
                                        position_l[m][1]=positionl[j][1];
                                        position_l[m][2]=positionl[j][2];
                                        //calculate the distance from each object to the center of right camera
                                        q= (double)(position_l[m][0]-160)*(position_l[m][0]-160)+(position_l[m][1]-
120)*(position_l[m][1]-120);
                                        position_l[m][3]=sqrt(q);
                                        m++;

                                }
                        }

        }

        for(b=0;b<m;b++)
        {
                position_sl[b][0]=position_l[b][0];
                position_sl[b][1]=position_l[b][1];
                position_sl[b][2]=position_l[b][2];
                position_sl[b][3]=position_l[b][3];
        }

        //sort position according to the distance
        for(k=0;k<m;k++)
        {
                for(s=0;s<m-1;s++)
                {
                        if(position_sl[s][3]>position_sl[s+1][3])
                        {
                                double t,tx,ty,tr;
                                tx=position_sl[s][0];
                                ty=position_sl[s][1];
                                tr=position_sl[s][2];
                                t=position_sl[s][3];
                                position_sl[s][0]=position_sl[s+1][0];
                                position_sl[s][1]=position_sl[s+1][1];
                                position_sl[s][2]=position_sl[s+1][2];
                                position_sl[s][3]=position_sl[s+1][3];
                                position_sl[s+1][0]=tx;
                                position_sl[s+1][1]=ty;
                                position_sl[s+1][2]=tr;
                                position_sl[s+1][3]=t;

                        }
                }
        }


        //Find all the positive objects in right eye
        getConnectedCompsright(FrameRightProcessed,Comps_right); //retrieve a list of all connected components in filtered image

        m=0;
        if (comptot_right != 0)
        {
                for (int j = 0; j < comptot_right; j++)
                {
                        //get center
                        position[j][0] = Comps_right[j]->rect.x + Comps_right[j]->rect.width/2;
                        position[j][1] = Comps_right[j]->rect.y + Comps_right[j]->rect.height/2;

                        //get radius
                        if (Comps_right[j]->rect.width > Comps_right[j]->rect.height)
                        {
                                position[j][2] = Comps_right[j]->rect.width/2;
                        }
                        else
                        {
                                position[j][2] = Comps_right[j]->rect.height/2;
                        }
```

```
                                if(position[j][2]>7)
                                {

                                        double q;

                                        position_r[m][0]=position[j][0];
                                        position_r[m][1]=position[j][1];
                                        position_r[m][2]=position[j][2];
                                        //calculate the distance from each object to the center of right camera
                                        q= (double)(position_r[m][0]-160)*(position_r[m][0]-160)+(position_r[m][1]-
120)*(position_r[m][1]-120);
                                        position_r[m][3]=sqrt(q);
                                        m++;

                                }
                        }

        }

        for(b=0;b<m;b++)
        {
                position_sr[b][0]=position_r[b][0];
                position_sr[b][1]=position_r[b][1];
                position_sr[b][2]=position_r[b][2];
                position_sr[b][3]=position_r[b][3];
        }

        //sort position according to the distance
        for(k=0;k<m;k++)
        {
                for(s=0;s<m-1;s++)
                {

                        if(position_sr[s][3]>position_sr[s+1][3])
                        {
                                double t,tx,ty,tr;
                                tx=position_sr[s][0];
                                ty=position_sr[s][1];
                                tr=position_sr[s][2];
                                t=position_sr[s][3];
                                position_sr[s][0]=position_sr[s+1][0];
                                position_sr[s][1]=position_sr[s+1][1];
                                position_sr[s][2]=position_sr[s+1][2];
                                position_sr[s][3]=position_sr[s+1][3];
                                position_sr[s+1][0]=tx;
                                position_sr[s+1][1]=ty;
                                position_sr[s+1][2]=tr;
                                position_sr[s+1][3]=t;

                        }
                }
        }

        /**********************************
        /*Image shape detection in left eye
        /**********************************/
        can_il=(int)position_sl[0][0];
        can_jl=(int)position_sl[0][1];
        can_rl=(int)position_sl[0][2];

        //detect the shape
        IplImage *left_detect = cvLoadImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working
now/pics/left_processed.jpg",1);

        Sleep(100);

        cvSetImageROI(left_detect, cvRect(can_il-can_rl-7, can_jl-can_rl-7, 2*can_rl+10, 2*can_rl+10));

        //create destination image: cvGetSize will return the width and the height of ROI
```

```
	IplImage *candidate_left = cvCreateImage(cvGetSize(left_detect), left_detect->depth, left_detect->nChannels);

	//copy subimage
	cvCopy(left_detect, candidate_left);
	cvResetImageROI(left_detect);

	cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/My Documents/MATLAB/candidate_left.jpg",candidate_left);

	Sleep(100);

	cvNamedWindow("Left Object", CV_WINDOW_AUTOSIZE);
	cvMoveWindow("Left Object", 900, 100);
	cvShowImage("Left Object",candidate_left);

	engEvalString(ep, "traleft=ShapeRecognition_left('candidate_left.jpg')");

	traleft = engGetVariable(ep,"traleft");
	double *dou2;
	dou2 = mxGetPr(traleft);
	shape_left= *dou2;

	/**********************************
	/*Image shape detection in right eye
	/**********************************/
	can_i=(int)position_sr[0][0];
	can_j=(int)position_sr[0][1];
	can_r=(int)position_sr[0][2];

	//detect the shape
	IplImage *right_detect = cvLoadImage("C:/Documents and Settings/luox2.VANDERBILT/Desktop/can be working
now/pics/right_processed.jpg",1);

	Sleep(100);

	cvSetImageROI(right_detect, cvRect(can_i-can_r-7, can_j-can_r-7, 2*can_r+10, 2*can_r+10));

	//create destination image: cvGetSize will return the width and the height of ROI
	IplImage *candidate_right = cvCreateImage(cvGetSize(right_detect), right_detect->depth, right_detect->nChannels);

	//copy subimage
	cvCopy(right_detect, candidate_right);
	cvResetImageROI(right_detect);

	cvSaveImage("C:/Documents and Settings/luox2.VANDERBILT/My Documents/MATLAB/candidate_right.jpg",
candidate_right);

	Sleep(100);

	cvNamedWindow("Right Object", CV_WINDOW_AUTOSIZE);
	cvMoveWindow("Right Object", 900, 500);
	cvShowImage("Right Object",candidate_right);

	engEvalString(ep, "tra_right=ShapeRecognition_right('candidate_right.jpg')");

	tra_right = engGetVariable(ep,"tra_right");
	double *tou2;
	tou2 = mxGetPr(tra_right);
	shape_right= *tou2;

	/**********************************
	*Object Localization
	**********************************/
	if((shape_left==0)|(shape_right==0))
	{
		printf("The object is unknown\n");

	}
	//Object Localization in left eye
	else if(shapes==shape_left)
```

```
{
	printf("Object has been found with left eye\n");

	bool mk=0;
	double k1,k2,k3,k4,b1,b2,b3,b4;
	bool p1,p2,p3,p4;
	//first number is angel of camera,second number is no.of dot, third number is the x,y index;x is 0,y is 1.
	label_left[0][0][0]=45;		label_left[0][1][0]=202;	label_left[0][1][1]=87;
label_left[0][2][0]=156;	label_left[0][2][1]=124;	label_left[0][3][0]=107;	label_left[0][3][1]=176;
label_left[0][4][0]=279;	label_left[0][4][1]=127;	label_left[0][5][0]=239;	label_left[0][5][1]=180;
	label_left[1][0][0]=40;		label_left[1][1][0]=184;	label_left[1][1][1]=83;
label_left[1][2][0]=140;	label_left[1][2][1]=123;	label_left[1][3][0]=94;
label_left[1][3][1]=179;	label_left[1][4][0]=260;	label_left[1][4][1]=118;	label_left[1][5][0]=225;
label_left[1][5][1]=172;
	label_left[2][0][0]=35;		label_left[2][1][0]=165;	label_left[2][1][1]=80;
label_left[2][2][0]=124;	label_left[2][2][1]=122;	label_left[2][3][0]=82;
label_left[2][3][1]=183;	label_left[2][4][0]=242;	label_left[2][4][1]=111;	label_left[2][5][0]=211;
label_left[2][5][1]=166;	label_left[2][6][0]=177;	label_left[2][6][1]=237;
	label_left[3][0][0]=30;		label_left[3][1][0]=147;	label_left[3][1][1]=79;
label_left[3][2][0]=108;	label_left[3][2][1]=124;	label_left[3][3][0]=69;
label_left[3][3][1]=188;	label_left[3][4][0]=223;	label_left[3][4][1]=105;	label_left[3][5][0]=197;
label_left[3][5][1]=161;	label_left[3][6][0]=167;	label_left[3][6][1]=234;
	label_left[4][0][0]=25;		label_left[4][1][0]=128;	label_left[4][1][1]=79;
label_left[4][2][0]=92;		label_left[4][2][1]=126;	label_left[4][3][0]=56;
label_left[4][3][1]=194;	label_left[4][4][0]=206;	label_left[4][4][1]=99;
label_left[4][5][0]=183;	label_left[4][5][1]=157;	label_left[4][6][0]=183;	label_left[4][6][1]=231;
label_left[4][7][0]=316;	label_left[4][7][1]=131;	label_left[4][8][0]=313;
label_left[4][8][1]=197;
	label_left[5][0][0]=20;		label_left[5][1][0]=109;	label_left[5][1][1]=80;
label_left[5][2][0]=76;		label_left[5][2][1]=130;	label_left[5][3][0]=43;
label_left[5][3][1]=201;	label_left[5][4][0]=187;	label_left[5][4][1]=96;
label_left[5][5][0]=169;	label_left[5][5][1]=153;	label_left[5][6][0]=150;	label_left[5][6][1]=229;
label_left[5][7][0]=296;	label_left[5][7][1]=118;	label_left[5][8][0]=297;
label_left[5][8][1]=183;
	label_left[6][0][0]=15;		label_left[6][1][0]=91;		label_left[6][1][1]=82;
label_left[6][2][0]=59;		label_left[6][2][1]=134;	label_left[6][3][0]=30;
label_left[6][3][1]=208;	label_left[6][4][0]=170;	label_left[6][4][1]=93;
label_left[6][5][0]=155;	label_left[6][5][1]=151;	label_left[6][6][0]=141;	label_left[6][6][1]=228;
label_left[6][7][0]=276;	label_left[6][7][1]=108;	label_left[6][8][0]=281;
label_left[6][8][1]=171;
	label_left[7][0][0]=10;		label_left[7][1][0]=72;		label_left[7][1][1]=85;
label_left[7][2][0]=42;		label_left[7][2][1]=141;	label_left[7][3][0]=18;
label_left[7][3][1]=218;	label_left[7][4][0]=152;	label_left[7][4][1]=92;
label_left[7][5][0]=142;	label_left[7][5][1]=150;	label_left[7][6][0]=132;	label_left[7][6][1]=228;
label_left[7][7][0]=256;	label_left[7][7][1]=100;	label_left[7][8][0]=265;
label_left[7][8][1]=161;
	label_left[8][0][0]=5;		label_left[8][1][0]=52;		label_left[8][1][1]=90;
label_left[8][2][0]=26;		label_left[8][2][1]=148;	label_left[8][3][0]=6;
label_left[8][3][1]=228;	label_left[8][4][0]=134;	label_left[8][4][1]=91;
label_left[8][5][0]=127;	label_left[8][5][1]=151;	label_left[8][6][0]=122;	label_left[8][6][1]=229;
label_left[8][7][0]=237;	label_left[8][7][1]=93;		label_left[8][8][0]=249;
label_left[8][8][1]=152;	label_left[8][9][0]=275;	label_left[8][9][1]=233;	label_left[8][10][0]=319;
label_left[8][10][1]=96;
	label_left[9][0][0]=0;		label_left[9][1][0]=31;		label_left[9][1][1]=95;
label_left[9][2][0]=9;		label_left[9][2][1]=157;	label_left[9][4][0]=116;	label_left[9][4][1]=91;
label_left[9][5][0]=113;	label_left[9][5][1]=152;	label_left[9][6][0]=113;	label_left[9][6][1]=229;
label_left[9][7][0]=218;	label_left[9][7][1]=88;		label_left[9][8][0]=233;
label_left[9][8][1]=144;	label_left[9][9][0]=263;	label_left[9][9][1]=221;	label_left[9][10][0]=296;
label_left[9][10][1]=84;
	label_left[10][0][0]=-5;	label_left[10][1][0]=12;	label_left[10][1][1]=103;
label_left[10][4][0]=98;	label_left[10][4][1]=93;	label_left[10][5][0]=98;	label_left[10][5][1]=153;
label_left[10][6][0]=104;	label_left[10][6][1]=231;	label_left[10][7][0]=199;	label_left[10][7][1]=83;
label_left[10][8][0]=218;	label_left[10][8][1]=137;	label_left[10][9][0]=251;
label_left[10][9][1]=211;	label_left[10][10][0]=275;	label_left[10][10][1]=76;	label_left[10][11][0]=306;
label_left[10][11][1]=124;
	label_left[11][0][0]=-10;	label_left[11][4][0]=80;	label_left[11][4][1]=96;
label_left[11][5][0]=85;	label_left[11][5][1]=157;	label_left[11][6][0]=95;	label_left[11][6][1]=235;
label_left[11][7][0]=180;	label_left[11][7][1]=80;	label_left[11][8][0]=202;	label_left[11][8][1]=132;
label_left[11][9][0]=239;	label_left[11][9][1]=202;	label_left[11][10][0]=253;
```

```
label_left[11][10][1]=68;	label_left[11][11][0]=287;	label_left[11][11][1]=113;
		label_left[12][0][0]=-15;	label_left[12][4][0]=61;	label_left[12][4][1]=100;
label_left[12][5][0]=70;	label_left[12][5][1]=161;	label_left[12][6][0]=86;	label_left[12][6][1]=239;
label_left[12][7][0]=162;	label_left[12][7][1]=78;	label_left[12][8][0]=186;	label_left[12][8][1]=128;
label_left[12][9][0]=226;	label_left[12][9][1]=194;	label_left[12][10][0]=231;
label_left[12][10][1]=62;	label_left[12][11][0]=266;	label_left[12][11][1]=104;	label_left[12][12][0]=315;
label_left[12][12][1]=161;
		label_left[13][0][0]=-20;	label_left[13][4][0]=42;	label_left[13][4][1]=105;
label_left[13][5][0]=55;	label_left[13][5][1]=167;	label_left[13][7][0]=143;	label_left[13][7][1]=77;
label_left[13][8][0]=170;	label_left[13][8][1]=125;	label_left[13][9][0]=214;	label_left[13][9][1]=187;
label_left[13][10][0]=210;	label_left[13][10][1]=57;	label_left[13][11][0]=247;
label_left[13][11][1]=96;	label_left[13][12][0]=297;	label_left[13][12][1]=147;
		label_left[14][0][0]=-25;	label_left[14][4][0]=23;	label_left[14][4][1]=112;
label_left[14][5][0]=41;	label_left[14][5][1]=174;	label_left[14][7][0]=124;	label_left[14][7][1]=77;
label_left[14][8][0]=154;	label_left[14][8][1]=123;	label_left[14][9][0]=201;	label_left[14][9][1]=181;
label_left[14][10][0]=190;	label_left[14][10][1]=54;	label_left[14][11][0]=227;
label_left[14][11][1]=90;	label_left[14][12][0]=278;	label_left[14][12][1]=137;
		label_left[15][0][0]=-30;	label_left[15][4][0]=4;	label_left[15][4][1]=121;
label_left[15][5][0]=26;	label_left[15][5][1]=182;	label_left[15][7][0]=105;	label_left[15][7][1]=78;
label_left[15][8][0]=139;	label_left[15][8][1]=122;	label_left[15][9][0]=188;	label_left[15][9][1]=176;
label_left[15][10][0]=169;	label_left[15][10][1]=51;	label_left[15][11][0]=208;
label_left[15][11][1]=85;	label_left[15][12][0]=260;	label_left[15][12][1]=128;
		label_left[16][0][0]=-35;	label_left[16][5][0]=11;	label_left[16][5][1]=191;
label_left[16][7][0]=86;	label_left[16][7][1]=81;	label_left[16][8][0]=123;	label_left[16][8][1]=122;
label_left[16][9][0]=175;	label_left[16][9][1]=172;	label_left[16][10][0]=149;	label_left[16][10][1]=50;
label_left[16][11][0]=190;	label_left[16][11][1]=81;	label_left[16][12][0]=242;
label_left[16][12][1]=119;
		label_left[17][0][0]=-40;	label_left[17][7][0]=66;	label_left[17][7][1]=84;
label_left[17][8][0]=107;	label_left[17][8][1]=125;	label_left[17][9][0]=162;	label_left[17][9][1]=169;
label_left[17][10][0]=129;	label_left[17][10][1]=50;	label_left[17][11][0]=171;	label_left[17][11][1]=78;
label_left[17][12][0]=225;	label_left[17][12][1]=114;
		label_left[18][0][0]=-45;	label_left[18][7][0]=46;	label_left[18][7][1]=88;
label_left[18][8][0]=91;	label_left[18][8][1]=126;	label_left[18][9][0]=150;	label_left[18][9][1]=168;
label_left[18][10][0]=108;	label_left[18][10][1]=51;	label_left[18][11][0]=152;	label_left[18][11][1]=77;
label_left[18][12][0]=207;	label_left[18][12][1]=108;

		for(mi=1;mi<19;mi++)
		{
			mk=((headAngles0[0]<label_left[mi][0][0])&&(headAngles0[0]>=label_left[mi+1][0][0]));
			if(mk!=0)
			{
				break;
			}
		}

		int num_angle=0;
		num_angle=mi;
		mk=0;
		bool flag=0;

		for(mj=1;mj<9;mj++)
		{

			if(((label_left[num_angle][mj][0]!=0)||(label_left[num_angle][mj][1]!=0))&&((label_left[num_angle][mj+1
			][0]!=0)||(label_left[num_angle][mj+1][1]!=0))&&((label_left[num_angle][mj+3][0]!=0)||(label_left[num_
			angle][mj+3][1]!=0))&&((label_left[num_angle][mj+4][0]!=0)||(label_left[num_angle][mj+4][1]!=0)!=0))
			{

				k1=(label_left[num_angle][mj+3][1]-
label_left[num_angle][mj][1])/(label_left[num_angle][mj+3][0]-label_left[num_angle][mj][0]);
				b1=label_left[num_angle][mj][1]-k1*label_left[num_angle][mj][0];
				p1=(position_sl[0][1]>k1*position_sl[0][0]+b1);
				k2=(label_left[num_angle][mj+4][1]-
label_left[num_angle][mj+1][1])/(label_left[num_angle][mj+4][0]-label_left[num_angle][mj+1][0]);
				b2=label_left[num_angle][mj+1][1]-k2*label_left[num_angle][mj+1][0];
				p2=(position_sl[0][1]<=k2*position_sl[0][0]+b2);
				k3=(label_left[num_angle][mj][1]-
label_left[num_angle][mj+1][1])/(label_left[num_angle][mj][0]-label_left[num_angle][mj+1][0]);
				b3=label_left[num_angle][mj][1]-k3*label_left[num_angle][mj][0];
```

```
                                        k4=(label_left[num_angle][mj+3][1]-
label_left[num_angle][mj+4][1])/(label_left[num_angle][mj+3][0]-label_left[num_angle][mj+4][0]);
                                        b4=label_left[num_angle][mj+3][1]-k4*label_left[num_angle][mj+3][0];

                                        if((k3>0)&&(k4>0))
                                        {
                                                p3=(position_sl[0][1]<k3*position_sl[0][0]+b3);
                                                p4=(position_sl[0][1]>k4*position_sl[0][0]+b4);
                                        }
                                        else if((k3<0)&&(k4>0))
                                        {
                                                p3=(position_sl[0][1]>k3*position_sl[0][0]+b3);
                                                p4=(position_sl[0][1]>k4*position_sl[0][0]+b4);
                                        }
                                        else if((k3>0)&&(k4<0))
                                        {
                                                p3=(position_sl[0][1]<k3*position_sl[0][0]+b3);
                                                p4=(position_sl[0][1]<k4*position_sl[0][0]+b4);
                                        }
                                        else
                                        {
                                                p3=(position_sl[0][1]>k3*position_sl[0][0]+b3);
                                                p4=(position_sl[0][1]<k4*position_sl[0][0]+b4);
                                        }

                                        mk=p1&&p2&&p3&&p4;

                                        if(mk!=0)
                                        {
                                                flag++;
                                                break;
                                        }
                                }

                                if(flag!=0)
                                {
                                        break;
                                }

                                }

                                if(mk==0)
                                {
                                        printf("The object is not of tracking range,please try searching again\n");
                                }

                                if(flag!=0)
                                {
                                        int num_dot;
                                        num_dot=mj;

                                        mxArray *P1=mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(P1), (void *)&position_sl[0][0], sizeof(position_sl[0][0]));
                                        mxArray *P2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(P2), (void *)&position_sl[0][1], sizeof(position_sl[0][1]));

                                        mxArray *Angle = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Angle), (void *)&headAngles0[0], sizeof(headAngles0[0]));

                                        mxArray *Angle1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Angle1), (void *)&label_left[num_angle][0][0],
sizeof(label_left[num_angle][0][0]));
                                        mxArray *X1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X1), (void *)&label_left[num_angle][num_dot][0],
sizeof(label_left[num_angle][num_dot][0]));
                                        mxArray *X2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X2), (void *)&label_left[num_angle][num_dot+3][0],
sizeof(label_left[num_angle][num_dot+3][0]));
                                        mxArray *X3 = mxCreateDoubleMatrix(1, 1, mxREAL);
```

```
                                        memcpy((void *)mxGetPr(X3), (void *)&label_left[num_angle][num_dot+1][0],
sizeof(label_left[num_angle][num_dot+1][0]));
                                        mxArray *X4 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X4), (void *)&label_left[num_angle][num_dot+4][0],
sizeof(label_left[num_angle][num_dot+4][0]));

                                        mxArray *Y1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y1), (void *)&label_left[num_angle][num_dot][1],
sizeof(label_left[num_angle][num_dot][1]));
                                        mxArray *Y2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y2), (void *)&label_left[num_angle][num_dot+3][1],
sizeof(label_left[num_angle][num_dot+3][1]));
                                        mxArray *Y3 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y3), (void *)&label_left[num_angle][num_dot+1][1],
sizeof(label_left[num_angle][num_dot+1][1]));
                                        mxArray *Y4 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y4), (void *)&label_left[num_angle][num_dot+4][1],
sizeof(label_left[num_angle][num_dot+4][1]));

                                        mxArray *Angle2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Angle2), (void *)&label_left[num_angle+1][0][0],
sizeof(label_left[num_angle+1][0][0]));
                                        mxArray *X11 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X11), (void *)&label_left[num_angle+1][num_dot][0],
sizeof(label_left[num_angle+1][num_dot][0]));
                                        mxArray *X12 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X12), (void *)&label_left[num_angle+1][num_dot+3][0],
sizeof(label_left[num_angle+1][num_dot+3][0]));
                                        mxArray *X13 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X13), (void *)&label_left[num_angle+1][num_dot+1][0],
sizeof(label_left[num_angle+1][num_dot+1][0]));
                                        mxArray *X14 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(X14), (void *)&label_left[num_angle+1][num_dot+4][0],
sizeof(label_left[num_angle+1][num_dot+4][0]));

                                        mxArray *Y11 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y11), (void *)&label_left[num_angle+1][num_dot][1],
sizeof(label_left[num_angle+1][num_dot][1]));
                                        mxArray *Y12 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y12), (void *)&label_left[num_angle+1][num_dot+3][1],
sizeof(label_left[num_angle+1][num_dot+3][1]));
                                        mxArray *Y13 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y13), (void *)&label_left[num_angle+1][num_dot+1][1],
sizeof(label_left[num_angle+1][num_dot+1][1]));
                                        mxArray *Y14 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                        memcpy((void *)mxGetPr(Y14), (void *)&label_left[num_angle+1][num_dot+4][1],
sizeof(label_left[num_angle+1][num_dot+4][1]));


                                        //Place the coordinates into the MATLAB workspace
                                        engPutVariable(ep, "P1", P1);
                                        engPutVariable(ep, "P2", P2);
                                        engPutVariable(ep, "Angle", Angle);
                                        engPutVariable(ep, "Angle1", Angle1);
                                        engPutVariable(ep, "X1", X1);
                                        engPutVariable(ep, "X2", X2);
                                        engPutVariable(ep, "X3", X3);
                                        engPutVariable(ep, "X4", X4);
                                        engPutVariable(ep, "Y1", Y1);
                                        engPutVariable(ep, "Y2", Y2);
                                        engPutVariable(ep, "Y3", Y3);
                                        engPutVariable(ep, "Y4", Y4);

                                        engPutVariable(ep, "Angle2", Angle2);
                                        engPutVariable(ep, "X11", X11);
                                        engPutVariable(ep, "X12", X12);
                                        engPutVariable(ep, "X13", X13);
                                        engPutVariable(ep, "X14", X14);
                                        engPutVariable(ep, "Y11", Y11);
```

```
                                        engPutVariable(ep, "Y12", Y12);
                                        engPutVariable(ep, "Y13", Y13);
                                        engPutVariable(ep, "Y14", Y14);

                                        engEvalString(ep,
"POSI=perspectivetrans('mapping',X1,X2,X3,X4,Y1,Y2,Y3,Y4,X11,X12,X13,X14,Y11,Y12,Y13,Y14,Angle 1,Angle2,Angle,P1,P2);");

                                        cvWaitKey(500);

                                        POSI = engGetVariable(ep,"POSI");
                                        double *posi_final;
                                        posi_final = mxGetPr(POSI);

                                        double final_x;
                                        double final_y;
                                        final_x=*posi_final;
                                        final_y=*(posi_final+1);

                                        printf("dot is=%d\n",num_dot);

                                        //3d location for each marker
                                        d3mark[0][0]=940;  d3mark[0][1]=810;  d3mark[1][0]=640;  d3mark[1][1]=830;
        d3mark[2][0]=340;  d3mark[2][1]=810;  d3mark[3][0]=950;  d3mark[3][1]=450;
                                        d3mark[4][0]=640;  d3mark[4][1]=450;  d3mark[5][0]=360;  d3mark[5][1]=450;
        d3mark[6][0]=970;  d3mark[6][1]=0;    d3mark[7][0]=650;  d3mark[7][1]=20;
                                        d3mark[8][0]=360;  d3mark[8][1]=0;    d3mark[9][0]=970;  d3mark[9][1]=-380;
        d3mark[10][0]=656; d3mark[10][1]=-370;  d3mark[11][0]=360; d3mark[11][1]=-370;

                                        d3coor[0]=d3mark[num_dot+3][0]+final_y/240*(d3mark[num_dot+2][0]-
d3mark[num_dot+3][0]);
                                        d3coor[1]=d3mark[num_dot][1]-final_x/320*(d3mark[num_dot][1]-d3mark[num_dot+3][1]);

                                        printf("%f,%f\n",d3coor[0],d3coor[1]);
                                        break;

                                }


                        }
                        //Object Localization in right eye
                        else if(shapes==shape_right)
                        {
                                printf("Object has been found with right eye\n");

                                bool mk=0;
                                double k1,k2,k3,k4,b1,b2,b3,b4;
                                bool p1,p2,p3,p4;

                                //first number is angel of camera,second number is no.of dot, third number is the x,y index;x is 0,y is 1.
                                label_right[0][0][0]=45;        label_right[0][1][0]=167;       label_right[0][1][1]=42;
        label_right[0][2][0]=119;       label_right[0][2][1]=73;        label_right[0][3][0]=70;        label_right[0][3][1]=113;
        label_right[0][4][0]=230;       label_right[0][4][1]=78;        label_right[0][5][0]=185;       label_right[0][5][1]=119;
        label_right[0][6][0]=136;       label_right[0][6][1]=167;       label_right[0][8][0]=294;       label_right[0][8][1]=192;
                                label_right[1][0][0]=40;        label_right[1][1][0]=147;       label_right[1][1][1]=42;
        label_right[1][2][0]=101;       label_right[1][2][1]=75;        label_right[1][3][0]=53;        label_right[1][3][1]=119;
        label_right[1][4][0]=211;       label_right[1][4][1]=74;        label_right[1][5][0]=170;       label_right[1][5][1]=117;
        label_right[1][6][0]=125;       label_right[1][6][1]=168;       label_right[1][7][0]=312;       label_right[1][7][1]=124;
        label_right[1][8][0]=281;       label_right[1][8][1]=180;
                                label_right[2][0][0]=35;        label_right[2][1][0]=126;       label_right[2][1][1]=42;
        label_right[2][2][0]=82;        label_right[2][2][1]=78;        label_right[2][3][0]=36;        label_right[2][3][1]=126;
        label_right[2][4][0]=192;       label_right[2][4][1]=69;        label_right[2][5][0]=155;       label_right[2][5][1]=115;
        label_right[2][6][0]=113;       label_right[2][6][1]=170;       label_right[2][7][0]=293;       label_right[2][7][1]=112;
        label_right[2][8][0]=267;       label_right[2][8][1]=169;
                                label_right[3][0][0]=30;        label_right[3][1][0]=106;       label_right[3][1][1]=43;
        label_right[3][2][0]=63;        label_right[3][2][1]=82;        label_right[3][3][0]=18;        label_right[3][3][1]=134;
        label_right[3][4][0]=174;       label_right[3][4][1]=68;        label_right[3][5][0]=139;       label_right[3][5][1]=114;
        label_right[3][6][0]=101;       label_right[3][6][1]=173;       label_right[3][7][0]=274;       label_right[3][7][1]=102;
        label_right[3][8][0]=253;       label_right[3][8][1]=160;
                                label_right[4][0][0]=25;        label_right[4][1][0]=86;        label_right[4][1][1]=46;
```

```
label_right[4][2][0]=44;	label_right[4][2][1]=88;	label_right[4][3][0]=2;
label_right[4][3][1]=144;	label_right[4][4][0]=156;	label_right[4][4][1]=66;	label_right[4][5][0]=125;
label_right[4][5][1]=114;	label_right[4][6][0]=90;	label_right[4][6][1]=176;	label_right[4][7][0]=256;
label_right[4][7][1]=95;	label_right[4][8][0]=239;	label_right[4][8][1]=153;	label_right[4][9][0]=225;
label_right[4][9][1]=233;
		label_right[5][0][0]=20;	label_right[5][1][0]=64;	label_right[5][1][1]=49;
label_right[5][2][0]=24;	label_right[5][2][1]=95;	label_right[5][4][0]=137;	label_right[5][4][1]=64;
label_right[5][5][0]=109;	label_right[5][5][1]=117;	label_right[5][6][0]=78;	label_right[5][6][1]=182;
label_right[5][7][0]=237;	label_right[5][7][1]=87;	label_right[5][8][0]=225;	label_right[5][8][1]=146;
label_right[5][9][0]=217;	label_right[5][9][1]=226;
		label_right[6][0][0]=15;	label_right[6][1][0]=43;	label_right[6][1][1]=55;
label_right[6][2][0]=5;		label_right[6][2][1]=103;	label_right[6][4][0]=118;
label_right[6][4][1]=66;	label_right[6][5][0]=93;	label_right[6][5][1]=119;	label_right[6][6][0]=66;
label_right[6][6][1]=187;	label_right[6][7][0]=219;	label_right[6][7][1]=83;	label_right[6][8][0]=211;
label_right[6][8][1]=139;	label_right[6][9][0]=208;	label_right[6][9][1]=219;	label_right[6][10][0]=306;
label_right[6][10][1]=97;	label_right[6][11][0]=315;	label_right[6][11][1]=157;
		label_right[7][0][0]=10;	label_right[7][1][0]=21;	label_right[7][1][1]=60;
label_right[7][4][0]=100;	label_right[7][4][1]=68;	label_right[7][5][0]=78;	label_right[7][5][1]=124;
label_right[7][6][0]=54;	label_right[7][6][1]=194;	label_right[7][7][0]=201;	label_right[7][7][1]=79;
label_right[7][8][0]=197;	label_right[7][8][1]=135;	label_right[7][9][0]=200;	label_right[7][9][1]=216;
label_right[7][10][0]=286;	label_right[7][10][1]=86;	label_right[7][11][0]=298;	label_right[7][11][1]=144;
label_right[7][12][0]=318;	label_right[7][12][1]=226;
		label_right[8][0][0]=5;	label_right[8][4][0]=81;	label_right[8][4][1]=71;
label_right[8][5][0]=62;	label_right[8][5][1]=129;	label_right[8][6][0]=43;	label_right[8][6][1]=201;
label_right[8][7][0]=183;	label_right[8][7][1]=75;	label_right[8][8][0]=183;	label_right[8][8][1]=132;
label_right[8][9][0]=191;	label_right[8][9][1]=210;	label_right[8][10][0]=266;	label_right[8][10][1]=77;
label_right[8][11][0]=284;	label_right[8][11][1]=133;	label_right[8][12][0]=308;	label_right[8][12][1]=213;
		label_right[9][0][0]=0;	label_right[9][4][0]=62;	label_right[9][4][1]=76;
label_right[9][5][0]=46;	label_right[9][5][1]=136;	label_right[9][6][0]=32;	label_right[9][6][1]=211;
label_right[9][7][0]=166;	label_right[9][7][1]=74;	label_right[9][8][0]=169;	label_right[9][8][1]=129;
label_right[9][9][0]=181;	label_right[9][9][1]=206;	label_right[9][10][0]=246;	label_right[9][10][1]=71;
label_right[9][11][0]=265;	label_right[9][11][1]=124;	label_right[9][12][0]=295;	label_right[9][12][1]=198;
		label_right[10][0][0]=-5;	label_right[10][4][0]=42;	label_right[10][4][1]=81;
label_right[10][5][0]=30;	label_right[10][5][1]=143;	label_right[10][6][0]=22;	label_right[10][6][1]=220;
label_right[10][7][0]=147;	label_right[10][7][1]=72;	label_right[10][8][0]=155;	label_right[10][8][1]=128;
label_right[10][9][0]=172;	label_right[10][9][1]=203;	label_right[10][10][0]=227;	label_right[10][10][1]=66;
label_right[10][11][0]=249;	label_right[10][11][1]=116;	label_right[10][12][0]=281;	label_right[10][12][1]=187;
		label_right[11][0][0]=-10;	label_right[11][4][0]=22;	label_right[11][4][1]=88;
label_right[11][5][0]=15;	label_right[11][5][1]=152;	label_right[11][6][0]=11;	label_right[11][6][1]=232;
label_right[11][7][0]=129;	label_right[11][7][1]=73;	label_right[11][8][0]=140;	label_right[11][8][1]=127;
label_right[11][9][0]=163;	label_right[11][9][1]=201;	label_right[11][10][0]=207;	label_right[11][10][1]=61;
label_right[11][11][0]=232;	label_right[11][11][1]=109;	label_right[11][12][0]=268;	label_right[11][12][1]=176;
		label_right[12][0][0]=-15;	label_right[12][4][0]=2;	label_right[12][4][1]=97;
label_right[12][5][0]=0;	label_right[12][5][1]=162;	label_right[12][7][0]=112;	label_right[12][7][1]=75;
label_right[12][8][0]=126;	label_right[12][8][1]=128;	label_right[12][9][0]=153;	label_right[12][9][1]=200;
label_right[12][10][0]=188;	label_right[12][10][1]=57;	label_right[12][11][0]=216;	label_right[12][11][1]=104;
label_right[12][12][0]=255;	label_right[12][12][1]=166;
		label_right[13][0][0]=-20;	label_right[13][7][0]=93;	label_right[13][7][1]=76;
label_right[13][8][0]=111;	label_right[13][8][1]=130;	label_right[13][9][0]=144;	label_right[13][9][1]=199;
label_right[13][10][0]=169;	label_right[13][10][1]=56;	label_right[13][11][0]=199;	label_right[13][11][1]=99;
label_right[13][12][0]=241;	label_right[13][12][1]=158;
		label_right[14][0][0]=-25;	label_right[14][7][0]=74;	label_right[14][7][1]=80;
label_right[14][8][0]=97;	label_right[14][8][1]=133;	label_right[14][9][0]=134;	label_right[14][9][1]=200;
label_right[14][10][0]=149;	label_right[14][10][1]=54;	label_right[14][11][0]=182;	label_right[14][11][1]=96;
label_right[14][12][0]=227;	label_right[14][12][1]=151;
		label_right[15][0][0]=-30;	label_right[15][7][0]=55;	label_right[15][7][1]=85;
label_right[15][8][0]=83;	label_right[15][8][1]=137;	label_right[15][9][0]=125;	label_right[15][9][1]=200;
label_right[15][10][0]=130;	label_right[15][10][1]=54;	label_right[15][11][0]=165;	label_right[15][11][1]=94;
label_right[15][12][0]=213;	label_right[15][12][1]=145;
		label_right[16][0][0]=-35;	label_right[16][7][0]=36;	label_right[16][7][1]=91;
label_right[16][8][0]=68;	label_right[16][8][1]=143;	label_right[16][9][0]=116;	label_right[16][9][1]=203;
label_right[16][10][0]=111;	label_right[16][10][1]=56;	label_right[16][11][0]=149;	label_right[16][11][1]=93;
label_right[16][12][0]=199;	label_right[16][12][1]=141;
		label_right[17][0][0]=-40;	label_right[17][7][0]=16;	label_right[17][7][1]=99;
label_right[17][8][0]=53;	label_right[17][8][1]=149;	label_right[17][9][0]=106;	label_right[17][9][1]=205;
label_right[17][10][0]=91;	label_right[17][10][1]=58;	label_right[17][11][0]=132;	label_right[17][11][1]=93;
label_right[17][12][0]=186;	label_right[17][12][1]=137;
		label_right[18][0][0]=-45;	label_right[18][8][0]=38;	label_right[18][8][1]=155;
label_right[18][9][0]=97;	label_right[18][9][1]=209;	label_right[18][10][0]=71;	label_right[18][10][1]=62;
```

```
label_right[18][11][0]=116;	label_right[18][11][1]=94;	label_right[18][12][0]=171;	label_right[18][12][1]=133;


		for(mi=1;mi<19;mi++)
		{
			mk=((headAngles0[2]<label_right[mi][0][0])&&(headAngles0[2]>=label_right[mi+1][0][0]));
			if(mk!=0)
			{
				break;
			}
		}

		int num_angle=0;
		num_angle=mi;
		mk=0;
		bool flag=0;

		for(mj=1;mj<9;mj++)
		{
		if(((label_right[num_angle][mj][0]!=0)||(label_right[num_angle][mj][1]!=0))&&((label_right[num_angle][
	mj+1][0]!=0)||(label_right[num_angle][mj+1][1]!=0))&&((label_right[num_angle][mj+3][0]!=0)||(label_right[num_a
	ngle][mj+3][1]!=0))&&((label_right[num_angle][mj+4][0]!=0)||(label_right[num_angle][mj+4][1]!=0)!=0))
			{

				k1=(label_right[num_angle][mj+3][1]-
label_right[num_angle][mj][1])/(label_right[num_angle][mj+3][0]-label_right[num_angle][mj][0]);
				b1=label_right[num_angle][mj][1]-k1*label_right[num_angle][mj][0];
				p1=(position_sr[0][1]>k1*position_sr[0][0]+b1);
				k2=(label_right[num_angle][mj+4][1]-
label_right[num_angle][mj+1][1])/(label_right[num_angle][mj+4][0]-label_right[num_angle][mj+1][0]);
				b2=label_right[num_angle][mj+1][1]-k2*label_right[num_angle][mj+1][0];
				p2=(position_sr[0][1]<=k2*position_sr[0][0]+b2);
				k3=(label_right[num_angle][mj][1]-
label_right[num_angle][mj+1][1])/(label_right[num_angle][mj][0]-label_right[num_angle][mj+1][0]);
				b3=label_right[num_angle][mj][1]-k3*label_right[num_angle][mj][0];
				k4=(label_right[num_angle][mj+3][1]-
label_right[num_angle][mj+4][1])/(label_right[num_angle][mj+3][0]-label_right[num_angle][mj+4][0]);
				b4=label_right[num_angle][mj+3][1]-k4*label_right[num_angle][mj+3][0];

				if((k3>0)&&(k4>0))
				{
					p3=(position_sr[0][1]<k3*position_sr[0][0]+b3);
					p4=(position_sr[0][1]>k4*position_sr[0][0]+b4);
				}
				else if((k3<0)&&(k4>0))
				{
					p3=(position_sr[0][1]>k3*position_sr[0][0]+b3);
					p4=(position_sr[0][1]>k4*position_sr[0][0]+b4);
				}
				else if((k3>0)&&(k4<0))
				{
					p3=(position_sr[0][1]<k3*position_sr[0][0]+b3);
					p4=(position_sr[0][1]<k4*position_sr[0][0]+b4);
				}
				else
				{
					p3=(position_sr[0][1]>k3*position_sr[0][0]+b3);
					p4=(position_sr[0][1]<k4*position_sr[0][0]+b4);
				}

				mk=p1&&p2&&p3&&p4;

				if(mk!=0)
				{
					flag++;
					break;
				}

		}
```

```
                                        if(flag!=0)
                                        {
                                                break;
                                        }

                                }

                                        if(mk==0)
                                        {
                                                printf("The object is not of tracking range,please try searching again\n");
                                        }

                                        if(flag!=0)
                                        {
                                                int num_dot;
                                                num_dot=mj;

                                                mxArray *P1=mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(P1), (void *)&position_sr[0][0],
sizeof(position_sr[0][0]));
                                                mxArray *P2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(P2), (void *)&position_sr[0][1],
sizeof(position_sr[0][1]));

                                                mxArray *Angle = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Angle), (void *)&headAngles0[2],
sizeof(headAngles0[2]));

                                                mxArray *Angle1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Angle1), (void *)&label_right[num_angle][0][0],
sizeof(label_right[num_angle][0][0]));
                                                mxArray *X1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X1), (void *)&label_right[num_angle][num_dot][0],
sizeof(label_right[num_angle][num_dot][0]));
                                                mxArray *X2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X2), (void *)&label_right[num_angle][num_dot+3][0],
sizeof(label_right[num_angle][num_dot+3][0]));
                                                mxArray *X3 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X3), (void *)&label_right[num_angle][num_dot+1][0],
sizeof(label_right[num_angle][num_dot+1][0]));
                                                mxArray *X4 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X4), (void *)&label_right[num_angle][num_dot+4][0],
sizeof(label_right[num_angle][num_dot+4][0]));

                                                mxArray *Y1 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y1), (void *)&label_right[num_angle][num_dot][1],
sizeof(label_right[num_angle][num_dot][1]));
                                                mxArray *Y2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y2), (void *)&label_right[num_angle][num_dot+3][1],
sizeof(label_right[num_angle][num_dot+3][1]));
                                                mxArray *Y3 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y3), (void *)&label_right[num_angle][num_dot+1][1],
sizeof(label_right[num_angle][num_dot+1][1]));
                                                mxArray *Y4 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y4), (void *)&label_right[num_angle][num_dot+4][1],
sizeof(label_right[num_angle][num_dot+4][1]));

                                                mxArray *Angle2 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Angle2), (void *)&label_right[num_angle+1][0][0],
sizeof(label_right[num_angle+1][0][0]));
                                                mxArray *X11 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X11), (void *)&label_right[num_angle+1][num_dot][0],
sizeof(label_right[num_angle+1][num_dot][0]));
                                                mxArray *X12 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X12), (void *)&label_right[num_angle+1][num_dot+3][0],
sizeof(label_right[num_angle+1][num_dot+3][0]));
                                                mxArray *X13 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X13), (void *)&label_right[num_angle+1][num_dot+1][0],
```

```
sizeof(label_right[num_angle+1][num_dot+1][0]));
                                                mxArray *X14 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(X14), (void *)&label_right[num_angle+1][num_dot+4][0],
sizeof(label_right[num_angle+1][num_dot+4][0]));

                                                mxArray *Y11 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y11), (void *)&label_right[num_angle+1][num_dot][1],
sizeof(label_right[num_angle+1][num_dot][1]));
                                                mxArray *Y12 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y12), (void *)&label_right[num_angle+1][num_dot+3][1],
sizeof(label_right[num_angle+1][num_dot+3][1]));
                                                mxArray *Y13 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y13), (void *)&label_right[num_angle+1][num_dot+1][1],
sizeof(label_right[num_angle+1][num_dot+1][1]));
                                                mxArray *Y14 = mxCreateDoubleMatrix(1, 1, mxREAL);
                                                memcpy((void *)mxGetPr(Y14), (void *)&label_right[num_angle+1][num_dot+4][1],
sizeof(label_right[num_angle+1][num_dot+4][1]));


                                                //Place the coordinates into the MATLAB workspace
                                                engPutVariable(ep, "P1", P1);
                                                engPutVariable(ep, "P2", P2);
                                                engPutVariable(ep, "Angle", Angle);
                                                engPutVariable(ep, "Angle1", Angle1);
                                                engPutVariable(ep, "X1", X1);
                                                engPutVariable(ep, "X2", X2);
                                                engPutVariable(ep, "X3", X3);
                                                engPutVariable(ep, "X4", X4);
                                                engPutVariable(ep, "Y1", Y1);
                                                engPutVariable(ep, "Y2", Y2);
                                                engPutVariable(ep, "Y3", Y3);
                                                engPutVariable(ep, "Y4", Y4);

                                                engPutVariable(ep, "Angle2", Angle2);
                                                engPutVariable(ep, "X11", X11);
                                                engPutVariable(ep, "X12", X12);
                                                engPutVariable(ep, "X13", X13);
                                                engPutVariable(ep, "X14", X14);
                                                engPutVariable(ep, "Y11", Y11);
                                                engPutVariable(ep, "Y12", Y12);
                                                engPutVariable(ep, "Y13", Y13);
                                                engPutVariable(ep, "Y14", Y14);

                                                engEvalString(ep,
"POSI=perspectivetrans('mapping',X1,X2,X3,X4,Y1,Y2,Y3,Y4,X11,X12,X13,X14,Y11,Y12,Y13,Y14,Angle1,Angle2,Angle,P1,P2);");

                                                cvWaitKey(500);

                                                POSI = engGetVariable(ep,"POSI");
                                                double *posi_final;
                                                posi_final = mxGetPr(POSI);

                                                double final_x;
                                                double final_y;
                                                final_x=*posi_final;
                                                final_y=*(posi_final+1);

                                                //3d location for each marker
                                                d3mark[0][0]=940;  d3mark[0][1]=810;  d3mark[1][0]=640;  d3mark[1][1]=830;
        d3mark[2][0]=340;  d3mark[2][1]=810;  d3mark[3][0]=950;  d3mark[3][1]=450;
                                                d3mark[4][0]=640;  d3mark[4][1]=450;  d3mark[5][0]=360;  d3mark[5][1]=450;
        d3mark[6][0]=970;  d3mark[6][1]=0;          d3mark[7][0]=650;  d3mark[7][1]=20;
                                                d3mark[8][0]=360;  d3mark[8][1]=0;          d3mark[9][0]=970;
        d3mark[9][1]=-380; d3mark[10][0]=656; d3mark[10][1]=-370;       d3mark[11][0]=360; d3mark[11][1]=-370;

                                                d3coor[0]=d3mark[num_dot+3][0]+final_y/240*(d3mark[num_dot+2][0]-
d3mark[num_dot+3][0]);
                                                d3coor[1]=d3mark[num_dot][1]-final_x/320*(d3mark[num_dot][1]-
d3mark[num_dot+3][1]);
```

```
                                        printf("%f,%f\n",d3coor[0],d3coor[1]);
                                        break;

                                }
                        }
                        else
                        {
                                        printf("The object has not been found\n");

                                        loadcoordsfile = fopen(loadcoords, "a");

fprintf(loadcoordsfile,"%f,%f,%f,%f\n",position_sl[0][0],position_sl[0][1],position_sr[0][0],position_sr[0][1]);
                                        fclose(loadcoordsfile);


                        }

                }

                coordsfile = fopen(coords, "a");
                fprintf(coordsfile,"%s,%s,%f,%f\n",colors_input,shape_input,d3coor[0],d3coor[1]);
                fclose(coordsfile);

                /**********************************
                * Hand Reaching
                **********************************/
                if((d3coor[0]^2+ d3coor[1]^2+330^2-(330+375)^2+200^2)>0)
                {
                        printf("The location is out of reach of ISAC\n");
                }

                /**********************************
                * Object Tracking
                **********************************/
                double map_left[19][2]={0};
                double map_right[19][2]={0};
                int map_mi;
                int map_mj;
                bool map_mk;
                bool map_mr;
                double map_angle[4];

                map_mk=0;
                map_mr=0;

                map_left[0][0]=45;              map_left[0][1]=800;
                map_left[1][0]=40;              map_left[1][1]=760;
                map_left[2][0]=35;              map_left[2][1]=640;
                map_left[3][0]=30;              map_left[3][1]=620;
                map_left[4][0]=25;              map_left[4][1]=560;
                map_left[5][0]=20;              map_left[5][1]=520;
                map_left[6][0]=15;              map_left[6][1]=455;
                map_left[7][0]=10;              map_left[7][1]=395;
                map_left[8][0]=5;               map_left[8][1]=310;
                map_left[9][0]=0;               map_left[9][1]=290;
                map_left[10][0]=-5;             map_left[10][1]=215;
                map_left[11][0]=-10;map_left[11][1]=160;
                map_left[12][0]=-15;map_left[12][1]=120;
                map_left[13][0]=-20;map_left[13][1]=60;
                map_left[14][0]=-25;map_left[14][1]=0;
                map_left[15][0]=-30;map_left[15][1]=-70;
                map_left[16][0]=-35;map_left[16][1]=-110;
                map_left[17][0]=-40;map_left[17][1]=-145;
                map_left[18][0]=-45;map_left[18][1]=-450;

                map_right[0][0]=45;             map_right[0][1]=800;
                map_right[1][0]=40;             map_right[1][1]=470;
                map_right[2][0]=35;             map_right[2][1]=450;
```

```
map_right[3][0]=30;		map_right[3][1]=360;
map_right[4][0]=25;		map_right[4][1]=310;
map_right[5][0]=20;		map_right[5][1]=245;
map_right[6][0]=15;		map_right[6][1]=200;
map_right[7][0]=10;		map_right[7][1]=120;
map_right[8][0]=5;		map_right[8][1]=80;
map_right[9][0]=0;		map_right[9][1]=60;
map_right[10][0]=-5;		map_right[10][1]=-20;
map_right[11][0]=-10;		map_right[11][1]=-50;
map_right[12][0]=-15;		map_right[12][1]=-100;
map_right[13][0]=-20;		map_right[13][1]=-170;
map_right[14][0]=-25;		map_right[14][1]=-200;
map_right[15][0]=-30;		map_right[15][1]=-220;
map_right[16][0]=-35;		map_right[16][1]=-220;
map_right[17][0]=-40;		map_right[17][1]=-220;
map_right[18][0]=-45;		map_right[18][1]=-220;

for(map_mj=0;map_mj<19;map_mj++)
{
	map_mk=((d3coor[1]<map_left[map_mj][1])&&(d3coor[1]>=map_left[map_mj+1][1]));

	if(map_mk!=0)
	{
		break;
	}
}

for(map_mi=0;map_mi<19;map_mi++)
{
	map_mr=((d3coor[1]<map_right[map_mi][1])&&(d3coor[1]>=map_right[map_mi+1][1]));

	if(map_mr!=0)
	{
		break;
	}
}

map_angle[0]=map_left[map_mj+1][0]+5*(d3coor[1]-map_left[map_mj+1][1])/(map_left[map_mj][1]-
map_left[map_mj+1][1]);
map_angle[1]=-35;
map_angle[2]=map_right[map_mi+1][0]+5*(d3coor[1]-map_right[map_mi+1][1])/(map_right[map_mi][1]-
map_right[map_mi+1][1]);
map_angle[3]=-35;

if(map_mr==0)
{
	map_angle[2]=-44;
}

if(map_mk==0)
{
	map_angle[0]=-44;
}


/*		if(map_angle[0]>45||map_angle[0]<-45||map_angle[2]>45||map_angle[2]<-45)
		{
			printf("ERROR!Angles of motors are out of range!");
			cvWaitKey(0);
		}
*/
		hd.MoveHead(map_angle);

		cvWaitKey(500);

		while(1)
		{

			frame3 = cvQueryFrame(capture2);
```

```
                        out3 = 0;

                        frame4 = cvQueryFrame(capture1);
                        out4 = 0;

                        cvShowImage( "Left_track", frame3);
                        cvShowImage( "Right_track",frame4);

                        if(!frame3)
                        {
                                break;
                        }

                        cvWaitKey(500);

                        d=getchar();

                        if(d=='d')
                        {
                                frame3 = cvQueryFrame(capture2);
                                out3 = 0;

                                frame4 = cvQueryFrame(capture1);
                                out4 = 0;

                                cvShowImage( "Left_track", frame3);
                                cvShowImage( "Right_track",frame4);

                                printf("The object tracking is completed!");
                                cvWaitKey(0);
                        }

                }


                //Clean Up
                cvReleaseCapture(&capture1);
                cvReleaseCapture(&capture2);

        return 0;


}


//*********************************************************************
//Authors: Jack Noble & Tom Billings
//Description: getConnectedComponents is used to distinguish between foreground
//                      and background elements within the image.    It does so by rasterscanning the image
//                      for foreground pixels (pixel == 255) and numbers the areas accordingly.
//                      If adjacent areas have the same numbers and are both foreground or background pixels
//                      they are combined into one connected component.    Only foreground pixels are placed
//                      into Comps.    If more than 20 components are found, a new Comps is
//                      initialized including old values.
//
//Variables:
//                      image is a binary image that has foreground and backgroun elements
//                      Comps holds all of the data for the connected components found in image
//
//Modified: by Sean Begley
//                                              - replaced hard #s for img width/hight with soft values retreived from
//                                                 passed in IplImage - 2/5/2008
//Modified again: by Xi Luo                      - 3/22/2011
//*********************************************************************
void getConnectedCompsleft(IplImage *image, CvConnectedComp ** Comps)
{
        CvScalar pixval;
        comptot_left = 0;
        int CurrentCompVal = 0;
```

```
	// used if need    to reinitialize components array for more capacity
	CvConnectedComp ** NewComp;

	for (int j = 0; j < image->width*image->height; j++)
	{
		pixval = cvGet2D(image,j/image->width,j%image->width);
		// if this pixel has the value of 255, this component has not yet been recorded
		if (pixval.val[0] > CurrentCompVal)
		{
			if (CurrentCompVal >= compmax_left)
				Comps[CurrentCompVal] = new CvConnectedComp;
			// fill this component in with the next lowest value
			cvFloodFill(image,cvPoint(j%image->width,j/image->width),cvScalar(CurrentCompVal +
1),cvScalar(0),cvScalar(0),Comps[CurrentCompVal],4);
			CurrentCompVal++;
			// if at capacity reinitialize array to higher capacity
			if (CurrentCompVal >= compcap)
			{
				NewComp = new CvConnectedComp*[compcap + 20];
				for (int i = 0; i < CurrentCompVal; i++)
				{
					// copy component values into new array
					NewComp[i] = new CvConnectedComp;
					NewComp[i]->area = Comps[i]->area;
					NewComp[i]->value.val[0] = Comps[i]->value.val[0];
					NewComp[i]->rect = Comps[i]->rect;
					// delete old values
					delete Comps[i];
				}
				// update pointers and capacity
				compcap = compcap + 20;
				Comps = NewComp;
			}
		}
	}
	// update total number of components being used
	comptot_left = CurrentCompVal;
	// ensure total number of components ever used is up to date (for deletion)
	if (comptot_left > compmax_left)
		compmax_left = comptot_left;
	return;
}

void getConnectedCompsright(IplImage *image, CvConnectedComp ** Comps)
{
	CvScalar pixval;
	comptot_right = 0;
	int CurrentCompVal = 0;
	// used if need    to reinitialize components array for more capacity
	CvConnectedComp ** NewComp;

	for (int j = 0; j < image->width*image->height; j++)
	{
		pixval = cvGet2D(image,j/image->width,j%image->width);
		// if this pixel has the value of 255, this component has not yet been recorded
		if (pixval.val[0] > CurrentCompVal)
		{
			if (CurrentCompVal >= compmax_right)
				Comps[CurrentCompVal] = new CvConnectedComp;
			// fill this component in with the next lowest value
			cvFloodFill(image,cvPoint(j%image->width,j/image->width),cvScalar(CurrentCompVal +
1),cvScalar(0),cvScalar(0),Comps[CurrentCompVal],4);
			CurrentCompVal++;
			// if at capacity reinitialize array to higher capacity
			if (CurrentCompVal >= compcap)
			{
				NewComp = new CvConnectedComp*[compcap + 20];
				for (int i = 0; i < CurrentCompVal; i++)
				{
```

```
                                        // copy component values into new array
                                        NewComp[i] = new CvConnectedComp;
                                        NewComp[i]->area = Comps[i]->area;
                                        NewComp[i]->value.val[0] = Comps[i]->value.val[0];
                                        NewComp[i]->rect = Comps[i]->rect;
                                        // delete old values
                                        delete Comps[i];
                                }
                                // update pointers and capacity
                                compcap = compcap + 20;
                                Comps = NewComp;
                        }
                }
        }
        // update total number of components being used
        comptot_right = CurrentCompVal;
        // ensure total number of components ever used is up to date (for deletion)
        if (comptot_right > compmax_right)
                compmax_right = comptot_right;
        return;
}
```

Code of Shape detection processed in MATLAB:

```
function posi=perspectivetrans(action,varargin)

x1=varargin{1};
x2=varargin{2};
x3=varargin{3};
x4=varargin{4};
y1=varargin{5};
y2=varargin{6};
y3=varargin{7};
y4=varargin{8};

x11=varargin{9};
x12=varargin{10};
x13=varargin{11};
x14=varargin{12};
y11=varargin{13};
y12=varargin{14};
y13=varargin{15};
y14=varargin{16};
angle1=varargin{17};
angle2=varargin{18};
angle=varargin{19};
p1=varargin{20};
p2=varargin{21};

A1=[-x1 -y1 -1  0      0   0 0 0 0
    0  0   0   -x1    -y1 -1  0 0 0
    -x2 -y2 -1 0 0 0 x2*319 y2*319 319
    0 0 0 -x2 -y2 -1 0 0 0
    -x3 -y3 -1 0 0 0 0 0 0
    0 0 0 -x3 -y3 -1 x3*239 y3*239 239
    -x4 y4 -1 0 0 0 x4*319 y4*319 319
    0 0 0 -x4 -y4 -1 x4*239 y4*239 239];
 [u,s,v1]=svd(A1);
 hr1=v1(:,9)';
 h1=reshape(hr1,3,3)';
 a1=inv(h1)*[p1;p2;1];
 posi1(1)=a1(1)/a1(3);
 posi1(2)=a1(2)/a1(3);

 A2=[-x11 -y11 -1  0      0   0 0 0 0
    0  0   0   -x11    -y11 -1  0 0 0
    -x12 -y12 -1 0 0 0 x12*319 y12*319 319
    0 0 0 -x12 -y12 -1 0 0 0
    -x13 -y13 -1 0 0 0 0 0 0
    0 0 0 -x13 -y13 -1 x13*239 y13*239 239
    -x14 -y14 -1 0 0 0 x14*319 y14*319 319
    0 0 0 -x14 -y14 -1 x14*239 y14*239 239];

 [u,s,v2]=svd(A2);
 hr2=v2(:,9)';
 h2=reshape(hr2,3,3)';
 a2=inv(h2)*[p1;p2;1];
 posi2(1)=a2(1)/a2(3);
 posi2(2)=a2(2)/a2(3);

 posi(1) = posi2(1)+(posi1(1)-posi2(1))/(angle1-angle2)*(angle-angle2);
 posi(2) = posi2(2)+(posi1(2)-posi2(2))/(angle1-angle2)*(angle-angle2);

end
```

```
function [shape]=ShapeRecognition_left(action,varargin)
    shape = zeros(1,1);
    S2 = imread(lower(action));
    S2 = im2bw(S2);
    figure(4);
    imshow(S2);
```

```
    S3 = bwlabel(S2,8);
    figure(5);
    imshow(S3);
    S4 =
regionprops(S3,'MinorAxisLength','MajorAxisLength','Area','Perimeter','Extrema','Centroid');
    se = strel('disk',5);

    q = S4.MajorAxisLength-S4.MinorAxisLength;
    temp = round(S4.Extrema);
    temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
    temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
    mask = zeros(max(temp(:,1)),max(temp(:,2)));
    for cnt = 1:8
        mask(temp(cnt,1),temp(cnt,2))=1;
    end
    mask2 = imdilate(mask,se);
    [labeled,numObjects] = bwlabel(mask2,8);

    score = numObjects;
    score1 = abs(1 - q/max([S4.MajorAxisLength,S4.MinorAxisLength]));
    score2 = abs(1 - abs(pi*((mean([S4.MajorAxisLength,S4.MinorAxisLength]))/2)^2-
S4.Area)/S4.Area);
    score3 = abs(1 - abs(pi*(max([S4.MajorAxisLength,S4.MinorAxisLength]))-
S4.Perimeter)/S4.Perimeter);
    scoreall=mean([score1;score2;score3]);
    if (score==4)&&(q<=14)
       shape=4;
    end
    if (((score==4)&&(q>14))||(score==3))
       shape=3;
    end
    if (score~=4)&&(score~=3)
        if(scoreall>0.7)
            shape=5;
        else
            shape=0;
        end
    end
    if (scoreall>0.94)&&(score==4)
       shape=5;
end

    figure(6);
    imshow(S2);
    text(S4.Centroid(1),S4.Centroid(2),num2str(shape),'color','red');

    disp(shape);
end

function [shape]=ShapeRecognition_right(action,varargin)
    shape = zeros(1,1);
    S2 = imread(lower(action));
    S2 = im2bw(S2);
    figure(4);
    imshow(S2);

    S3 = bwlabel(S2,8);
    figure(5);
    imshow(S3);
    S4 =
regionprops(S3,'MinorAxisLength','MajorAxisLength','Area','Perimeter','Extrema','Centroid');
    se = strel('disk',5);

    q = S4.MajorAxisLength-S4.MinorAxisLength;
    temp = round(S4.Extrema);
    temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
    temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
    mask = zeros(max(temp(:,1)),max(temp(:,2)));
    for cnt = 1:8
        mask(temp(cnt,1),temp(cnt,2))=1;
    end
    mask2 = imdilate(mask,se);
    [labeled,numObjects] = bwlabel(mask2,8);

    score = numObjects;
    score1 = abs(1 - q/max([S4.MajorAxisLength,S4.MinorAxisLength]));
    score2 = abs(1 - abs(pi*((mean([S4.MajorAxisLength,S4.MinorAxisLength]))/2)^2-
S4.Area)/S4.Area);
    score3 = abs(1 - abs(pi*(max([S4.MajorAxisLength,S4.MinorAxisLength]))-
S4.Perimeter)/S4.Perimeter);
    scoreall=mean([score1;score2;score3]);
```

```
    if (score==4)&&(q<=14)
       shape=4;
    end
    if (((score==4)&&(q>14))||(score==3))
       shape=3;
    end
    if (score~=4)&&(score~=3)
        if(scoreall>0.7)
            shape=5;
        else
            shape=0;
        end
    end
    if (scoreall>0.94)&&(score==4)
       shape=5;
end

    figure(6);
    imshow(S2);
    text(S4.Centroid(1),S4.Centroid(2),num2str(shape),'color','red');

    disp(shape);
end
```

Code for arm control system processed in Visual Studio 2008:

```
#include "stdafx.h"

#include "CameraHead.h"
#include <math.h>                                   //for trig functions
#include <time.h>                                   //for delay function

//constants
#ifndef PI
#define PI 3.14159
#define R2D         (180.0/PI)
#define D2R (PI/180.0)
#endif

BOOL CameraHead::Initialize()
{

        USES_CONVERSION;

        if ( pCommLEFT == NULL )
        {
                // COM1 port
                char* pFN1 = OLE2T(m_bsDevFilename1);
                pCommLEFT       = new CCommPort( pFN1, 9600 );          // Capture COM1@9600baud
        }

        if ( pCommRIGHT == NULL )
        {
                // COM2 port.
                char* pFN2 = OLE2T(m_bsDevFilename2);
                pCommRIGHT      = new CCommPort( pFN2, 9600 );          // Capture COM2@9600baud
        }

        for ( int axis_index = 0; axis_index < NUM_AXES; axis_index++ )
                m_psOut[ axis_index ] = 0;

        // re-initialize pan-tilt unit
        char strOut[35];
        BOOL commResult1 = FALSE;
        BOOL commResult2 = FALSE;

        // it won't hurt to set these commands each time this is called.

        // Enable terse feedback
        sprintf( strOut, "%s ", TERSE_FEEDBACK );

        commResult1 = SendToLeft( strOut );
```

```
	commResult2 = SendToRight( strOut );

	if ( commResult1 && commResult2 )
	{
		// Disable DP head limits on startup
		sprintf( strOut, "%s ", DISABLE_LIMITS );
		commResult1 = SendToLeft( strOut );
	    commResult2 = SendToRight( strOut );

		if ( commResult1 && commResult2 )
		{
			// Disable command echo
			sprintf( strOut, "%s ", DISABLE_ECHO );
			commResult1 = SendToLeft( strOut );
			commResult2 = SendToRight( strOut );

			// sample pan/tilt positions
/***			if ( commResult1 && commResult2 )
			{
				commResult1 = DPSample();

				if ( !commResult1 )
					return E_FAIL;
			}
***/
			double val[4];

			for ( int i = 0; i < NUM_AXES; i++ )
			{
				val[i] = 0;
			}

			// set to the home position
			commResult1 = CommandHeadAbsolute(val,NUM_AXES);
		}
		else
		{
//			ATLTRACE("****** CCatch::FinalComponentInit() -- Writing to COMM FAILED!\n" );
//			printf("****CameraHead.Device**** : Writing to comm FAILED!\n");
			return FALSE;
		}
	}
	else
	{
//		ATLTRACE("****** CCatch::FinalComponentInit() -- Writing to COMM FAILED!\n" );
//		printf("****CameraHead.Device**** : Writing to comm FAILED!\n");
		return FALSE;
	}

	//set new speed and accel for pan/tilt
	put_LeftPanAccel((long)8000);
	put_RightPanAccel((long)8000);
	put_LeftPanSpeed((long)2500);
	put_RightPanSpeed((long)2500);
	put_LeftTiltAccel((long)8000);
	put_RightTiltAccel((long)8000);
	put_LeftTiltSpeed((long)2500);
	put_RightTiltSpeed((long)2500);

//	ATLTRACE("****** CCatch::FinalComponentInit -- RETURNED OK!\n" );
//	printf("****CameraHead.Device**** : FinalComponentInitialization done!\n");
	return true;

}

/////////////////////////////////////////////////////////////////////////////

	Name:				CCatch::IndexIsValid()
```

```
        Purpose:        Check parameters given by the user.  Parameters must represent
                                a valid axis of the pan-tilt unit.

        Created By:             RMW - watson@vuse.vanderbilt.edu
        Date Modified:  Sunday, August 17, 1997

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

bool
CameraHead::IndexIsValid
        (
        long Index                                      // in: index to test for validity
        )
{
        return ( ( Index >= 0 ) && ( Index < NUM_AXES ) );
}


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

        Name:                   CCatch::SampleHead()


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

BOOL CameraHead::SampleHead
        (
        double *    Val,                                        // out:     values representing where the pan-tilt motors are
now
                                                                                //              Valid verge range:
[MIN_RANGE, MAX_RANGE]
        long            Length                                  // in: how long is the vector
        )
{

        // Get the new pan and tilt positions.
        BOOL result = FALSE;

        result = DPSample();

        if ( !result )
                return FALSE;

        if ( Val != NULL )
        {
                // Fill up the user's structure
                        Val[ LEFT_PAN    ] = (double) m_plCurrentSample[ LEFT_PAN ];    // Val[0]
                        Val[ RIGHT_PAN ] = (double) m_plCurrentSample[ RIGHT_PAN ];             // Val[2]
                        Val[ LEFT_TILT ] = (double) m_plCurrentSample[ LEFT_TILT ];  // Val[1]
                        Val[ RIGHT_TILT] = (double) m_plCurrentSample[ RIGHT_TILT ];// Val[3]
        }
        else return FALSE;

        return TRUE;
}

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

        Name:                   CCatch::CommandHeadAbsolute()

        Purpose:        Method to tell the pan-tilt unit to go to a certain location.

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

BOOL CameraHead::CommandHeadAbsolute
        (
        double *        Val,                                    // in: values to send to the pan-tilt motors
                                                                                //              Valid verge range:
[MIN_RANGE(0), MAX_RANGE(255)]
```

```
	long			Length					// in: how long is the vector
	)
{

	if ( Val != NULL )
	{
		/** CCatch::CommandHeadAbsolute = %.0f %.0f %.0f %.0f", Val[LEFT_MOTOR], Val[RIGHT_MOTOR],
Val[PAN_MOTOR], Val[TILT_MOTOR] ***/

		if ( ( pCommLEFT != NULL ) && ( pCommRIGHT != NULL ) )
		{
			char strOut[35];
			BOOL result = FALSE;

			// calculate values to send out.
			for ( int i = 0; i < NUM_AXES; i++ )
			{
				m_psOut[i] = Val[i] * m_pdAxisGain[i] + m_pdAxisOffset[i];
				m_plCurrentSample[i] = Val[i];
			}


			// pan
			if ( ExceedsThreshold( m_psOut[ LEFT_PAN ], LEFT_PAN ) )
			{
				sprintf( strOut, "%s%d ", COMMAND_PAN_ABS, m_psOut[LEFT_PAN] );
				SendToLeft( strOut );
			}

			if ( ExceedsThreshold( m_psOut[ RIGHT_PAN ], RIGHT_PAN ) )
			{
				sprintf( strOut, "%s%d ", COMMAND_PAN_ABS, m_psOut[RIGHT_PAN] );
				SendToRight( strOut );
			}

			// tilt
			if ( ExceedsThreshold( m_psOut[ LEFT_TILT ], LEFT_TILT ) )
			{
				sprintf( strOut, "%s%d ", COMMAND_TILT_ABS, m_psOut[LEFT_TILT] );
				SendToLeft( strOut );
			}


			if ( ExceedsThreshold( m_psOut[ RIGHT_TILT ], RIGHT_TILT ) )
			{
				sprintf( strOut, "%s%d ", COMMAND_TILT_ABS, m_psOut[RIGHT_TILT] );
				SendToRight( strOut );
			}
		}// if the comm ports were opened properly //

	}// if pVal is usable //
	else return FALSE;

	return TRUE;
}



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

	Name:				ParseStringToInteger()

	Purpose:		Parse those darn strings returned by the DP head into a
					usable integer.

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

int
CameraHead::ParseStringToInteger
```

```
	(
	char *	pStr					// change: string to convert
	)
{
	char	newStr[35];
	int		result = -1;

	// chop the first two characters
	strcpy( newStr, &(pStr[2]) );

	// find the position of the newline character
	int newline_position = 0;
	char* strJunk = strchr( newStr, '\n' );

	if ( strJunk != NULL )
	{
		newline_position = strJunk - newStr;

		// chop off the end of the string
		newStr[newline_position] = '\0';

		// ...finally, convert the integer.
		result = atoi( newStr );
	}
	else
		result = -1;

	return result;
}


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

	Name:				CCatch::DPSample()

	Purpose:		Get the position of the pan and tilt.

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

BOOL
CameraHead::DPSample
	(
	)
{
	char strOut[35];
	char strIn[35];
	BOOL result = FALSE;


	// get Left pan-tilt unit position
	strcpy( strIn, "" );
	strcpy( strOut, "" );


	if ( pCommLEFT != NULL )
	{
		printf("LEFT LEFT LEFT LEFT\n");
		//get pan position
		sprintf( strOut, "%s ", QUERY_PAN );
		SendToLeft( strOut );

		strcpy( strOut, "" );

		result = pCommLEFT->ReadComm( strIn, 10 );
		printf("PAN1: %d\n", result);
		//do it again
		// Get left pan-tilt unit position
		strcpy( strIn, "" );
		sprintf( strOut, "%s ", QUERY_PAN );
```

```
		SendToLeft( strOut );

		strcpy( strOut, "" );
		result = pCommLEFT->ReadComm( strIn, 10 );
		printf("PAN2: %d\n", result);


		strcpy( strIn, "" );
		sprintf( strOut, "%s ", QUERY_PAN );
		SendToLeft( strOut );

		strcpy( strOut, "" );
		result = pCommLEFT->ReadComm( strIn, 10 );
		printf("PAN3: %d\n", result);

		if ( result == TRUE )
		{
			/** ATLTRACE("** DPSAMPLE PAN = %s\n", strIn ); **/
			// the returned command looks like: "* 0", so parse it properly.
			m_plCurrentSample[ LEFT_PAN ] = ParseStringToInteger( strIn );

			// get tilt position
			strcpy( strIn, "" );
			strcpy( strOut, "" );

			sprintf( strOut, "%s ", QUERY_TILT );
			SendToLeft( strOut );

			result = pCommLEFT->ReadComm( strIn, 10 );

			if ( result == TRUE )
			{
				/** ATLTRACE("** DPSAMPLE TILT = %s\n", strIn ); **/
				m_plCurrentSample[ LEFT_TILT ] = ParseStringToInteger( strIn );
			}
			else
				return FALSE;

		}
		else
			return FALSE;

}
// get right pan-tilt unit position
strcpy( strIn, "" );
strcpy( strOut, "" );

if ( pCommRIGHT != NULL)
{

		// get pan position
		sprintf( strOut, "%s ", QUERY_PAN );
		SendToRight( strOut );

		strcpy( strOut, "" );

		result = pCommRIGHT->ReadComm( strIn, 10 );

		if ( result )
		{
		//	ATLTRACE("** DPSAMPLE RIGHT PAN = %s\n", strIn );
			// the returned command looks like: "* 0", so parse it properly.
			m_plCurrentSample[ RIGHT_PAN] = ParseStringToInteger( strIn );

			// get tilt position
			strcpy( strIn, "" );
			strcpy( strOut, "" );

			sprintf( strOut, "%s ", QUERY_TILT );
```

```
                SendToRight( strOut );

                result = pCommRIGHT->ReadComm( strIn, 10 );

                if ( result )
                {
        //              ATLTRACE("** DPSAMPLE RIGHT TILT = %s\n", strIn );
                        m_plCurrentSample[ RIGHT_TILT] = ParseStringToInteger( strIn );
                }
                else
                        return FALSE;
        }
        else
                return FALSE;
    }

    return TRUE;
}


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

    Name:                   CCatch::get_LeftPanAccel()

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

BOOL CameraHead::get_LeftPanAccel
    (
    long *      pVal                    // out: the current acceleration value
    )
{
    if ( pVal != NULL )
    {
        char strOut[35];
        char strIn[35];
        BOOL result = FALSE;

        // make the query
        sprintf( strOut, "%s ", QUERY_PAN_ACCEL );
        result = SendToLeft( strOut );

        if ( result )
        {
            // retrieve the result
            strcpy( strIn, "" );
            result = pCommLEFT->ReadComm( strIn, 10 );

            if ( result )
            {
                long acceleration = ParseStringToInteger( strIn );

                *pVal = acceleration;
            }
            else
                return FALSE;
        }
        else
            return FALSE;
    }
    else
        return FALSE;

    return TRUE;
}

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

    Name:                   CCatch::get_RightPanAccel()
```

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

BOOL CameraHead::get_RightPanAccel
	(
	long *	pVal					// out: the current acceleration value
	)
{
	if ( pVal != NULL )
	{
		char strOut[35];
		char strIn[35];
		BOOL result = FALSE;

		// make the query
		sprintf( strOut, "%s ", QUERY_PAN_ACCEL );
		result = SendToRight( strOut );

		if ( result )
		{
			// retrieve the result
			strcpy( strIn, "" );
			result = pCommRIGHT->ReadComm( strIn, 10 );

			if ( result )
			{
				long acceleration = ParseStringToInteger( strIn );

				*pVal = acceleration;
			}
			else
				return FALSE;
		}
		else
			return FALSE;
	}
	else
		return FALSE;

	return TRUE;
}


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

	Name:				CCatch::put_LeftPanAccel()

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

BOOL CameraHead::put_LeftPanAccel
	(
	long	newVal					// in: the new acceleration you want.
	)
{
	if ( newVal > 0 )
	{
		char strOut[35];
		BOOL result = FALSE;

		sprintf( strOut, "%s%d ", SET_PAN_ACCEL, newVal );

		result = SendToLeft( strOut );

		if ( !result )
			return FALSE;

		LeftPanAccel = newVal;

	}
	else
```

```
                return FALSE;

        return TRUE;
}

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

        Name:                   CCatch::put_RightPanAccel()

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

BOOL CameraHead::put_RightPanAccel
        (
        long    newVal                          // in: the new acceleration you want.
        )
{
        if ( newVal > 0 )
        {
                char strOut[35];
                BOOL result = FALSE;

                sprintf( strOut, "%s%d ", SET_PAN_ACCEL, newVal );

                result = SendToRight( strOut );

                if ( !result )
                        return FALSE;

                RightPanAccel = newVal;

        }
        else
                return FALSE;

        return TRUE;
}

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

        Name:                   CCatch::get_LeftTiltAccel()


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

BOOL CameraHead::get_LeftTiltAccel
        (
        long *  pVal                            // out: return acceleration value to user.
        )
{
        if ( pVal != NULL )
        {
                char strOut[35];
                char strIn[35];
                BOOL result = FALSE;

                // make the query
                sprintf( strOut, "%s ", QUERY_TILT_ACCEL );
                result = SendToLeft( strOut );

                if ( result )
                {
                        // retrieve the result
                        strcpy( strIn, "" );

                        result = pCommLEFT->ReadComm( strIn, 10 );

                        if ( result )
                        {
                                long acceleration = ParseStringToInteger( strIn );
```

```
                                        *pVal = acceleration;
                                }
                                else
                                        return FALSE;
                        }
                        else
                                return E_FAIL;
                }
                else
                        return FALSE;

                return TRUE;
}

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

        Name:                           CCatch::get_RightTiltAccel()


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

BOOL CameraHead::get_RightTiltAccel
        (
        long *      pVal                                        // out: return acceleration value to user.
        )
{
        if ( pVal != NULL )
        {
                char strOut[35];
                char strIn[35];
                BOOL result = FALSE;

                // make the query
                sprintf( strOut, "%s ", QUERY_TILT_ACCEL );
                result = SendToRight( strOut );

                if ( result )
                {
                        // retrieve the result
                        strcpy( strIn, "" );

                        result = pCommRIGHT->ReadComm( strIn, 10 );

                        if ( result )
                        {
                                long acceleration = ParseStringToInteger( strIn );
                                *pVal = acceleration;
                        }
                        else
                                return FALSE;
                }
                else
                        return E_FAIL;
        }
        else
                return FALSE;

        return TRUE;
}

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

        Name:                           CCatch::put_LeftTiltAccel()

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

BOOL CameraHead::put_LeftTiltAccel
        (
        long        newVal                                  // in: the new acceleration value
```

```
	)
{
	if ( newVal > 0 )
	{
		char strOut[35];
		BOOL result = FALSE;

		sprintf( strOut, "%s%d ", SET_TILT_ACCEL, newVal );

		result = SendToLeft( strOut );

		if ( !result )
			return FALSE;

		LeftTiltAccel = newVal;
	}
	else
		return FALSE;

	return TRUE;
}


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

	Name:				CCatch::put_RightTiltAccel()

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

BOOL CameraHead::put_RightTiltAccel
	(
	long	newVal					// in: the new acceleration value
	)
{
	if ( newVal > 0 )
	{
		char strOut[35];
		BOOL result = FALSE;

		sprintf( strOut, "%s%d ", SET_TILT_ACCEL, newVal );

		result = SendToRight( strOut );

		if ( !result )
			return FALSE;

		RightTiltAccel = newVal;
	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::SendToLeft()

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

BOOL
CameraHead::SendToLeft
	(
	char *	pStr					// in: string to send.
	)
{
	BOOL commResult = FALSE;

	if ( pCommLEFT != NULL )
```

```
	{
		commResult = pCommLEFT->WriteComm( pStr );
/*
		LPVOID lpMsgBuf;

		FormatMessage(
			FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,	NULL,
			GetLastError(),
			MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
			(LPTSTR) &lpMsgBuf,	0,	NULL );// Display the string.

		::MessageBox( NULL, (LPCTSTR) lpMsgBuf, "GetLastError", MB_OK|MB_ICONINFORMATION );

		// Free the buffer.
		LocalFree( lpMsgBuf );
*/
	}
	else
		commResult = FALSE;

	return commResult;
}

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

	Name:				CCatch::SendToRight()

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

BOOL
CameraHead::SendToRight
	(
	char *	pStr					// in: string to send.
	)
{
	BOOL commResult = FALSE;

	if ( pCommRIGHT != NULL )
	{
		commResult = pCommRIGHT->WriteComm( pStr );
/*
		LPVOID lpMsgBuf;

		FormatMessage(
			FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,	NULL,
			GetLastError(),
			MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
			(LPTSTR) &lpMsgBuf,	0,	NULL );// Display the string.

		::MessageBox( NULL, (LPCTSTR) lpMsgBuf, "GetLastError", MB_OK|MB_ICONINFORMATION );

		// Free the buffer.
		LocalFree( lpMsgBuf );
*/
	}
	else
		commResult = FALSE;

	return commResult;
}


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

	Name:				CCatch::get_LeftPanSpeed()

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

```
BOOL CameraHead::get_LeftPanSpeed
	(
	long *	pVal					// out: current speed setting of pan motor.
	)
{
	if ( pVal != NULL )
	{
		char strOut[35];
		char strIn[35];
		BOOL result = FALSE;

		// make the query
		sprintf( strOut, "%s ", QUERY_PAN_SPEED );
		result = SendToLeft( strOut );

		if ( result )
		{
			// retrieve the result
			strcpy( strIn, "" );
			result = pCommLEFT->ReadComm( strIn, 10 );

			if ( result )
			{
				long speed = ParseStringToInteger( strIn );
				*pVal = speed;
			}
			else
				return FALSE;
		}
		else
			return FALSE;
	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::get_RightPanSpeed()

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

BOOL CameraHead::get_RightPanSpeed
	(
	long *	pVal					// out: current speed setting of pan motor.
	)
{
	if ( pVal != NULL )
	{
		char strOut[35];
		char strIn[35];
		BOOL result = FALSE;

		// make the query
		sprintf( strOut, "%s ", QUERY_PAN_SPEED );
		result = SendToRight( strOut );

		if ( result )
		{
			// retrieve the result
			strcpy( strIn, "" );
			result = pCommRIGHT->ReadComm( strIn, 10 );

			if ( result )
			{
				long speed = ParseStringToInteger( strIn );
				*pVal = speed;
```

```
                        }
                        else
                                return FALSE;
                }
                else
                        return FALSE;
        }
        else
                return FALSE;

        return TRUE;
}


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

        Name:                   CCatch::put_LeftPanSpeed()

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

BOOL CameraHead::put_LeftPanSpeed
        (
        long    newVal                                          // in: whatever you want the new speed to be.
        )
{
        if ( newVal > 0 )
        {
                BOOL result = FALSE;
                char strOut[35];

                sprintf( strOut, "%s%d ", SET_PAN_SPEED, newVal );

                result = SendToLeft( strOut );

                if ( !result )
                        return FALSE;

                LeftPanSpeed = newVal;
        }
        else
                return FALSE;

        return TRUE;
}

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

        Name:                   CCatch::put_RightPanSpeed()

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

BOOL CameraHead::put_RightPanSpeed
        (
        long    newVal                                          // in: whatever you want the new speed to be.
        )
{
        if ( newVal > 0 )
        {
                BOOL result = FALSE;
                char strOut[35];

                sprintf( strOut, "%s%d ", SET_PAN_SPEED, newVal );

                result = SendToRight( strOut );

                if ( !result )
                        return FALSE;

                RightPanSpeed = newVal;
```

```
	}
	else
		return FALSE;

	return TRUE;
}
/*---------------------------------------------------------------------------

	Name:				CCatch::get_LeftTiltSpeed()

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

BOOL CameraHead::get_LeftTiltSpeed
	(
	long *	pVal					// out: the current speed of the tilt motor
	)
{
	if ( pVal != NULL )
	{
		char strOut[35];
		char strIn[35];
		BOOL result = FALSE;

		// make the query
		sprintf( strOut, "%s ", QUERY_TILT_SPEED );

		result = SendToLeft( strOut );

		if ( result )
		{
			// retrieve the result
			strcpy( strIn, "" );
			result = pCommLEFT->ReadComm( strIn, 10 );

			if ( result )
			{
				long speed = ParseStringToInteger( strIn );
				*pVal = speed;
			}
			else
				return FALSE;
		}
		else
			return FALSE;
	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::get_RightTiltSpeed()

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

BOOL CameraHead::get_RightTiltSpeed
	(
	long *	pVal					// out: the current speed of the tilt motor
	)
{
	if ( pVal != NULL )
	{
		char strOut[35];
		char strIn[35];
		BOOL result = FALSE;

		// make the query
```

```
		sprintf( strOut, "%s ", QUERY_TILT_SPEED );

		result = SendToRight( strOut );

		if ( result )
		{
			// retrieve the result
			strcpy( strIn, "" );
			result = pCommRIGHT->ReadComm( strIn, 10 );

			if ( result )
			{
				long speed = ParseStringToInteger( strIn );
				*pVal = speed;
			}
			else
				return FALSE;
		}
		else
			return FALSE;
	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::put_LeftTiltSpeed()

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

BOOL CameraHead::put_LeftTiltSpeed
	(
	long	newVal					// in: new speed of tilt motor
	)
{
	if ( newVal > 0 )
	{
		char strOut[35];
		BOOL result = FALSE;

		sprintf( strOut, "%s%d ", SET_TILT_SPEED, newVal );
		result = SendToLeft( strOut );

		if ( !result )
		{
			return FALSE;
		}

		LeftTiltSpeed = newVal;

	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::put_RightTiltSpeed()

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

BOOL CameraHead::put_RightTiltSpeed
	(
	long	newVal					// in: new speed of tilt motor
```

```
	)
{
	if ( newVal > 0 )
	{
		char strOut[35];
		BOOL result = FALSE;

		sprintf( strOut, "%s%d ", SET_TILT_SPEED, newVal );
		result = SendToRight( strOut );

		if ( !result )
		{
			return FALSE;
		}

		RightTiltSpeed = newVal;

	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::Reset()

	Purpose:		Reset the pan-tilt head.

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

BOOL CameraHead::Reset()
{
	char strOut[35];
	BOOL result1 = FALSE;
	BOOL result2 = FALSE;

	// reset pan and tilt
	sprintf( strOut, "%s ", RESET );
	result1 = SendToLeft( strOut );
	result2 = SendToRight( strOut );


	if ( result1 && result2 )
	{
		m_plCurrentSample[ LEFT_PAN    ] = 0;
		m_plCurrentSample[ LEFT_TILT ] = 0;
		m_plCurrentSample[ RIGHT_PAN ] = 0;
		m_plCurrentSample[ RIGHT_TILT] = 0;
	}
	else
		return FALSE;

	return TRUE;
}

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

	Name:				CCatch::Stop()

	Purpose:		Stop any pan-tilt motion in progress.

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

BOOL CameraHead::Stop()
{
	char strOut[35];
	BOOL result1 = FALSE;
```

```
	BOOL result2 = FALSE;

	sprintf( strOut, "%s ", HALT_ALL );
	result1 = SendToLeft( strOut );
	result2 = SendToRight( strOut );

	if ( !result1 || !result2 )
		return FALSE;

	return TRUE;
}


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

	Name:			CCatch::get_AxisThreshold()

	Purpose:		Get the movement threshold for this axis.

	Created By:		RMW - watson@vuse.vanderbilt.edu
	Date Modified:	Friday, August 22, 1997

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

BOOL CameraHead::get_AxisThreshold
	(
	long	Index,				// in: which axis you are interested in
	long *	pVal				// out: the current threshold value
	)
{
	if ( pVal == NULL )
		return FALSE;

	if ( IndexIsValid( Index ) )
		*pVal = m_plAxisThreshold[ Index ];
	else return FALSE;

	return TRUE;
}


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

	Name:			CCatch::put_AxisThreshold()

	Purpose:		Set the movement threshold for this axis.

	Created By:		RMW - watson@vuse.vanderbilt.edu
	Date Modified:	Friday, August 22, 1997

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

BOOL CameraHead::put_AxisThreshold
	(
	long	Index,				// in: the axis you are interested in
	long	newVal				// in: the new threshold value
	)
{
	if ( IndexIsValid( Index ) )
		m_plAxisThreshold[ Index ] = newVal;
	else return FALSE;

	return TRUE;
}


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

	Name:			CCatch::ExceedsThreshold()
```

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

BOOL
CameraHead::ExceedsThreshold
	(
	long	Val,
	long	axis_index
	)
{
		if (abs(Val - m_plCurrentSample[axis_index]) >= m_plAxisThreshold[axis_index])
		return TRUE;
	else
		return FALSE;
}



void CameraHead::MoveHead(double *command)
{
		// COMMAND HEAD
		// Setup output values from input vector
		for(int i=0;i<NUM_AXES;i++)
			m_pdLinkCurrentHeadCommand[i] = command[i];

		// convert angles to positions and sent to the head.
		BOOL hrAngleConversionStatus = FALSE;
		hrAngleConversionStatus = HeadAngles2Positions( m_pdLinkCurrentHeadCommand, NUM_AXES );

		// here's where you SET values.
		if ( SUCCEEDED( hrAngleConversionStatus ) )
			CommandHeadAbsolute( m_pdLinkCurrentHeadCommand, NUM_AXES );
}

void CameraHead::GetHeadPositions(double *val)
{
		// SAMPLE HEAD
		SampleHead( m_pdLinkCurrentHeadSample, NUM_AXES );

		// convert angles to positions and sent to the head.
		BOOL hrPositionConversionStatus = FALSE;
		hrPositionConversionStatus = HeadPositions2Angles( m_pdLinkCurrentHeadSample, NUM_AXES );

	//	m_pdLinkCurrentHeadSample[NUM_AXES]     = m_pdLinkCurrentHeadSample[2] - m_dPreviousPosition[0];
//		m_pdLinkCurrentHeadSample[NUM_AXES+1] = m_pdLinkCurrentHeadSample[3] - m_dPreviousPosition[1];

		if ( SUCCEEDED( hrPositionConversionStatus ) )
		{
			CalculateOverallPanTilt(m_pdLinkCurrentHeadSample);
			for(int i=0;i<NUM_AXES;i++)
				val[i] = m_pdLinkCurrentHeadSample[i];
		}

//		m_dPreviousPosition[0] = m_pdLinkCurrentHeadSample[2];
//		m_dPreviousPosition[1] = m_pdLinkCurrentHeadSample[3];
}

////////////////////////////////////////////////////////////////////////
// Function to obtain verge angles and a single pan-tilt
// angle from two cameras (08/05/03)
// Val = [LP LT RP RT OverallPan Avg Tilt VergeAngles]
// Notes on the mathematical theory can be found in:
// Cis\common\InetPub\wwwroot\IRL\Doc\Head\Head Change\Isac_Vis_Angles
////////////////////////////////////////////////////////////////////////

bool CameraHead::CalculateOverallPanTilt(double* Val)
{
	double dtempL, dtempR, Alpha; //AA, BB;
```

```
	double ThetaLeft, ThetaRight;

	// Collect and Allocate Catch Angles from Input Vector
	dtempL = Val[0];
	dtempR = Val[2];

	// Converting from degrees to radians
	ThetaLeft  = D2R*(90-dtempL);
	ThetaRight		= D2R*(90+dtempR);

	// Calculate overall pan
	// This equation is to find a singular pan from two cameras
	// The document can be found at Cis\common\InetPub\wwwroot\IRL\Doc\Head\Head Change\Finding a Singular Pan

	Alpha = atan (sin(ThetaRight-ThetaLeft)/(2*sin(ThetaLeft)*sin(ThetaRight)));

	// Store this in the 5th element of the array    as an overall pan angle in degrees
	Val[4] = R2D * Alpha;	// clockwise angles are negative

	// Calculate average Tilt
	Val[5] = (Val[1] + Val[3]) / 2;

	// Converting from degrees to radians
	dtempL = D2R*dtempL;
	dtempR = D2R*dtempR;

	// Calculate verge angles
//	AA = sin(dtempR-dtempL);
//	BB = pow(cos(dtempR),2)+ pow(cos(dtempL),2) + 2*cos(dtempR)*sin(dtempL)*cos(dtempR-dtempL);
//	BB = pow(BB,0.5);
//	Val[6] = atan(AA/BB) * R2D;

	// calculate overall pan
	//		Val[4] = atan((tan(dtempL)+tan(dtempR))/2) * R2D;

	return TRUE;

}


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

	Name:				CCatch::HeadAngles2Positions()

	Purpose:		Convert angles that come in into positions understood by the head.

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

BOOL
CameraHead::HeadAngles2Positions
	(
	double *	pVal,					// in/out: angles->positions
	long		length					// in: how big
	)
{
	// Angle 2 Position conversion
	if ( pVal != NULL )
	{
		pVal[ LEFT_PAN    ] =    pVal[ LEFT_PAN    ] / PULSE_TO_ANG_PAN_LEFT;
		pVal[ LEFT_TILT ] =    pVal[ LEFT_TILT ] / PULSE_TO_ANG_TILT;
		pVal[ RIGHT_PAN ] =    pVal[ RIGHT_PAN ] / PULSE_TO_ANG_PAN_RIGHT;
		pVal[ RIGHT_TILT] =    pVal[ RIGHT_TILT] / PULSE_TO_ANG_TILT;
	}
	else
		return FALSE;

	return TRUE;
}
```

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

        Name:                   CCatch::HeadPositions2Angles()

        Purpose:        Convert positions that come in into angles understood by everyone.

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

BOOL
CameraHead::HeadPositions2Angles
        (
        double *    pVal,                                   // in/out: positions->angles
        long            length                                  // in: how big
        )
{
        // Position 2 Angle conversion
        if ( pVal != NULL )
        {
                pVal[ LEFT_PAN    ] = pVal[ LEFT_PAN    ] * PULSE_TO_ANG_PAN_LEFT;
                pVal[ LEFT_TILT ] = pVal[ LEFT_TILT ] * PULSE_TO_ANG_TILT;
                pVal[ RIGHT_PAN ] = pVal[ RIGHT_PAN ] * PULSE_TO_ANG_PAN_RIGHT;
                pVal[ RIGHT_TILT] = pVal[ RIGHT_TILT] * PULSE_TO_ANG_TILT;
        }
        else
                return FALSE;

        return TRUE;
}

/*************************************************************
* Author:           Xi Luo 02/14/2010
* Name:               CameraHead::Home
* Description:    Define the home position as random angles
*************************************************************/
void CameraHead::Home()
{
        double buff[4];
        buff[0] = 4;
        buff[1] = -35.0;
        buff[2] = 4;
        buff[3] = -35.0;

        MoveHead(buff);
}
```

```
/*** I have to encapsulate this motor controlling ***/
/*** why? 'cuz IMA sucks!!! ***/
//#include "stdafx.h"
#include "windows.h"
#include "atlbase.h"
#include "commport.h"
#include "string.h"
#include "stdio.h"
//#include "iostream.h"
#include <iostream>
#include "conio.h"
#include <math.h>
#include "cv.h"                    //image stuff
////////////////////////////////////////////////////////////////////////////
// CCatch
class CameraHead
{
public:

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

        Name:                           CCatch Component Constructor

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

        CameraHead() :
                        // *** Note: If you want to change the command packet layout, then
                        // you need to change these next 4 constants!

                                LEFT_PAN(0),                    // index of the left pan motor in the arrays
                                LEFT_TILT(1),                   // index of the left tilt motor in the arrays
                                RIGHT_PAN(2),                   // index of the right pan motor in the arrays
                                RIGHT_TILT(3),                  // index of the right tilt motor in the arrays

                                NUM_AXES(6),                    // # axes on the left/right pan-tilt units currently.
//                              NUM_AXES_ALL(6),                // # axes on the Left/right pan-tilt units and overall
pan-tilt.

                                COMMAND_PAN_ABS("pp"),  // string which indicates you are sending a command to the
pan motor.
                                COMMAND_TILT_ABS("tp"), // string which indicates you are sending a command to the tilt
motor.
                                COMMAND_PAN_REL("po"),  // string which commands pan relative
                                COMMAND_TILT_REL("to"), // string which commands tilt relative

                                QUERY_PAN("pp"),                // get the pan position
                                QUERY_TILT("tp"),               // get the tilt position
                                QUERY_MIN_PAN("pn"),            // get min pan limit
                                QUERY_MAX_PAN("px"),            // get max pan limit
                                QUERY_MIN_TILT("tn"),           // get min tilt limit
                                QUERY_MAX_TILT("tx"),           // get max tilt limit
                                QUERY_PAN_SPEED("ps"),          // get pan speed
                                QUERY_TILT_SPEED("ts"),         // get tilt speed
                                QUERY_PAN_ACCEL("pa"),          // get pan acceleration
                                QUERY_TILT_ACCEL("ta"),         // get tilt acceleration

                                SET_PAN_ACCEL("pa"),            // set pan acceleration
                                SET_TILT_ACCEL("ta"),           // set tilt acceleration
                                SET_PAN_SPEED("ps"),            // set pan speed
                                SET_TILT_SPEED("ts"),           // set tilt speed

                                HALT_ALL("h"),                          // stop all pan-tilt unit
                                RESET("r"),                                     // reset the pan-tilt unit
                                RESTORE_DEFAULTS("df"),         // restore factory default settings
                                DISABLE_LIMITS("ld"),           // disable limits on DP head.

                                SYNC_DP("a"),                           // allow last DP command to finish
                                TERSE_FEEDBACK("ft"),           // no wordy responses, Newman.
                                DISABLE_ECHO("ed"),                     // don't echo commands we send.
```

```
                                        // DP PAN
                                        MOTOR_RANGE_ANGLES_PAN(318.0f),     // pan angle range
                                        MOTOR_RANGE_PULSES_PAN_LEFT(6184.0),    // pulses in left servo's range
                                        MOTOR_RANGE_PULSES_PAN_RIGHT(6178.0), // pulses in right servo's range
                                        PULSE_TO_ANG_PAN_LEFT((double)
(MOTOR_RANGE_ANGLES_PAN/MOTOR_RANGE_PULSES_PAN_LEFT)),
                                        PULSE_TO_ANG_PAN_RIGHT((double)
(MOTOR_RANGE_ANGLES_PAN/MOTOR_RANGE_PULSES_PAN_RIGHT)),

                                // DP Tilt
                                        MOTOR_RANGE_ANGLES_TILT(106.0f),    // tilt angle range
                                        MOTOR_RANGE_PULSES_TILT(2061.0f),   // pulses in servo's range
                                        PULSE_TO_ANG_TILT((double)
(MOTOR_RANGE_ANGLES_TILT/MOTOR_RANGE_PULSES_TILT))


        {

                // TODO: Initialize all variables
                m_pdAxisGain = NULL;
                m_pdAxisOffset = NULL;
                m_plAxisThreshold = NULL;

                m_plCurrentSample = NULL;
                m_pdLinkCurrentHeadCommand = NULL;
                m_pdLinkCurrentHeadSample = NULL;
        //      m_pdLinkOldHeadCommand = NULL;
//              m_pdLinkOldHeadSample = NULL;


                pCommLEFT = NULL;
            pCommRIGHT = NULL;
                m_psOut = NULL;

                m_pdBuffer = NULL;
                m_bFirstCue = false;

                m_iFlagCue = 0;

                m_pdBuff1 = new double[4];
                m_pdBuff2 = new double[4];
                m_pdBuff3 = new double[4];

                for (int i=0; i<4; i++)
                {
                        m_pdBuff1[i] = 0;
                        m_pdBuff2[i] = 1;
                        m_pdBuff3[i] = 2;
                }


                // setup class variables to appropriate sizes
                if ( NUM_AXES > 0 )
                {
                        m_psOut = new long[NUM_AXES];

                        m_pdAxisGain =          new double[ NUM_AXES ];
                        m_pdAxisOffset = new double[ NUM_AXES ];
                        m_plAxisThreshold = new long[ NUM_AXES ];

                        m_pdLinkCurrentHeadCommand = new double[ NUM_AXES ];

                        // new head will display 2 more values: representing a
                        // overall pan/tilt value for both cameras
                        m_pdLinkCurrentHeadSample = new double[ NUM_AXES];
                        m_plCurrentSample = new long[ NUM_AXES ];

                        // Two missing pointers were noticed not having been allocated
```

```
                        // It is done here:
//                      m_pdLinkOldHeadCommand = new double[ NUM_AXES];
//                      m_pdLinkOldHeadSample = new double[ NUM_AXES];
                }


                // initialize axis gains and offsets

                for ( int j = 0; j < NUM_AXES; j++ )
                {
                        m_pdAxisGain[j] = 1.0;
                        m_pdAxisOffset[j] = 0.0;
                        m_plAxisThreshold[j] = 0;
                }

                // setup default values for motors and communications parameters
                m_plCurrentSample[ LEFT_PAN    ] = 0;
                m_plCurrentSample[ LEFT_TILT ] = 0;
                m_plCurrentSample[ RIGHT_PAN ] = 0;
                m_plCurrentSample[ RIGHT_TILT] = 0;

                m_bsDevFilename1 = "\\COM1";
                m_bsDevFilename2 = "\\COM2";
                m_bsCParam1 = "";
                m_bsCParam2 = "";
                m_lCTimeout1 = 0;
                m_lCTimeout2 = 0;

        }


        ~CameraHead()
        {
                /********** Beginning CCatch::~CCatch() **********/
                        if ( m_psOut != NULL )
                        delete[] m_psOut;

                m_psOut = NULL;


                ATLTRACE("*** Deleting Commports ....");
                if ( pCommLEFT != NULL )
                        delete pCommLEFT;
                pCommLEFT = NULL;
                ATLTRACE(" ... PT Left destroyed,");

                if ( pCommRIGHT != NULL )
                        delete pCommRIGHT;
                pCommRIGHT = NULL;
                ATLTRACE(" PT Right destroyed ...\n");

                if ( m_pdAxisGain != NULL )
                        delete[] m_pdAxisGain;
                m_pdAxisGain = NULL;

                if ( m_pdAxisOffset != NULL )
                        delete[] m_pdAxisOffset;
                m_pdAxisOffset = NULL;

                if ( m_plAxisThreshold != NULL )
                        delete[] m_plAxisThreshold;
                m_plAxisThreshold = NULL;

                if ( m_plCurrentSample != NULL )
                        delete[] m_plCurrentSample;
                m_plCurrentSample = NULL;

                /********** CCatch::~CCatch() Deleted Current Sample**********/
```

```
		if ( m_pdLinkCurrentHeadCommand != NULL )
			delete[] m_pdLinkCurrentHeadCommand;
		m_pdLinkCurrentHeadCommand = NULL;

		/********** CCatch::~CCatch() Deleted Link Head Command**********/


//		ATLTRACE("********** CCatch::~CCatch() Deleted Link Head Command**********\n");


		if ( m_pdLinkCurrentHeadSample != NULL )
			delete[] m_pdLinkCurrentHeadSample;
		m_pdLinkCurrentHeadSample = NULL;


		/********** Finishing CCatch::~CCatch() **********/
//		ATLTRACE("********** Finishing CCatch::~CCatch() **********\n");


	}


public:
//	double m_dPreviousPosition[2];		// for pan and tilt velocity
	long LeftPanSpeed;
	long RightPanSpeed;
	long LeftPanAccel;
	long RightPanAccel;
	long LeftTiltSpeed;
	long RightTiltSpeed;
	long LeftTiltAccel;
	long RightTiltAccel;
//	long VergeLimit( long Val );

public:
	/** conversion function ***/
	BOOL HeadAngles2Positions( double* pVal, long length );
	BOOL HeadPositions2Angles( double* pVal, long Length );
	int ParseStringToInteger( char* pStr );

	/** internal command **/
	BOOL CommandHeadAbsolute(double* Val, long Length);
	BOOL SampleHead(double* Val, long Length);
//	BOOL CommandHeadRelative(double* Val, long Length);
	bool CalculateOverallPanTilt(double* Val);
	BOOL DPSample();
	BOOL ExceedsThreshold( long, long );
	BOOL SendToLeft( char* pStr );
	BOOL SendToRight( char* pStr );

	/** auxillary command **/
	bool IndexIsValid( long Index );

	/** properties **/
	BOOL get_AxisThreshold(long Index, long* pVal);
	BOOL put_AxisThreshold(long Index, long newVal);
	BOOL get_LeftTiltSpeed(long *pVal);
	BOOL get_RightTiltSpeed(long *pVal);
	BOOL put_LeftTiltSpeed(long newVal);
	BOOL put_RightTiltSpeed(long newVal);
	BOOL get_LeftPanSpeed(long *pVal);
	BOOL get_RightPanSpeed(long *pVal);
	BOOL put_LeftPanSpeed(long newVal);
	BOOL put_RightPanSpeed(long newVal);
	BOOL get_LeftTiltAccel(long *pVal);
	BOOL get_RightTiltAccel(long *pVal);
	BOOL put_LeftTiltAccel(long newVal);
	BOOL put_RightTiltAccel(long newVal);
```

```
        BOOL get_LeftPanAccel(long *pVal);
        BOOL get_RightPanAccel(long *pVal);
        BOOL put_LeftPanAccel(long newVal);
        BOOL put_RightPanAccel(long newVal);
//      BOOL get_TiltLimit(long Length, long *pVal);
//      BOOL get_PanLimit(long Length, long *pVal);
//      BOOL get_CommReadTimeout2(long *pVal);
//      BOOL put_CommReadTimeout2(long newVal);
//      BOOL get_CommReadTimeout1(long *pVal);
//      BOOL put_CommReadTimeout1(long newVal);
//      BOOL get_AxisGain(long Index, double *pVal);
//      BOOL put_AxisGain(long Index, double newVal);
//      BOOL get_AxisOffset(long Index, double *pVal);
//      BOOL put_AxisOffset(long Index, double newVal);
//      BOOL get_CommParams2(char *pVal);
//      BOOL put_CommParams2(char newVal);
//      BOOL get_CommParams1(char *pVal);
//      BOOL put_CommParams1(char newVal);
//      BOOL get_DeviceFilename2(char *pVal);
//      BOOL put_DeviceFilename2(char newVal);
//      BOOL get_DeviceFilename1(char *pVal);
//      BOOL put_DeviceFilename1(char newVal);

public:
        /** general interface command **/
        BOOL Reset();
        BOOL Stop();
        BOOL Initialize();
        void MoveHead(double *command);
        void GetHeadPositions(double *val);
        void Home();


//private:

        // constants useful for this class. (OK--kat)
        // DP Pan
        const double MOTOR_RANGE_ANGLES_PAN;            // pan angle range
        const double MOTOR_RANGE_PULSES_PAN_LEFT;   // pulses in left servo's range
        const double MOTOR_RANGE_PULSES_PAN_RIGHT;  // pulses in right servo's range
        const double PULSE_TO_ANG_PAN_LEFT;
        const double PULSE_TO_ANG_PAN_RIGHT;

        // DP Tilt
        const double MOTOR_RANGE_ANGLES_TILT;       // tilt angle range
        const double MOTOR_RANGE_PULSES_TILT;       // pulses in servo's range
        const double PULSE_TO_ANG_TILT;

        const long      NUM_AXES;                               // how many axes does this pantilt have?

        const char*             COMMAND_TILT_REL;               // command for relative tilt motion
        const char*             COMMAND_PAN_REL;                // command for relative pan motion
        const char*             COMMAND_TILT_ABS;               // command for absolute tilt motion
        const char*             COMMAND_PAN_ABS;                // command for absolute pan motion

        const long      LEFT_PAN;
        const long      LEFT_TILT;
        const long      RIGHT_PAN;
        const long      RIGHT_TILT;

        const char*             QUERY_TILT;                             // get the tilt position
        const char*             QUERY_PAN;                              // get the pan position

        const char*             QUERY_MIN_PAN;                  // get min pan limit
        const char*             QUERY_MAX_PAN;                  // get max pan limit
        const char*             QUERY_MIN_TILT;                 // get min tilt limit
        const char*             QUERY_MAX_TILT;                 // get max tilt limit
        const char*             QUERY_PAN_SPEED;        // get pan speed
        const char*             QUERY_PAN_ACCEL;        // get pan acceleration
```

```
        const char*             QUERY_TILT_SPEED;       // get tilt speed
        const char*             QUERY_TILT_ACCEL;       // get tilt acceleration

        const char*             SET_PAN_ACCEL;                  // set pan acceleration
        const char*             SET_PAN_SPEED;                  // set pan speed
        const char*             SET_TILT_ACCEL;                 // set tilt acceleration
        const char*             SET_TILT_SPEED;                 // set tilt speed

        const char*             HALT_ALL;                               // stop the pan-tilt unit

        const char*             RESET;                                  // reset the pan-tilt unit
        const char*             RESTORE_DEFAULTS;               // restore factory default settings

        const char*             DISABLE_LIMITS;                 // disable limits on DP head.
        const char*             DISABLE_ECHO;                   // don't echo commands we send.

        const char*             SYNC_DP;                        // allow last DP command to finish

        const char*             TERSE_FEEDBACK;                 // no wordy responses, Newman.

        // COMMPORT parameters
        CComBSTR m_bsDevFilename2;                              // Device filenames for 1 & 2
        CComBSTR m_bsDevFilename1;
        CComBSTR m_bsCParam2;                                   // Device parameters for 1 & 2
        CComBSTR m_bsCParam1;

        long    m_lCTimeout1;                                   // Read timeouts for 1 & 2
        long    m_lCTimeout2;

        double* m_pdAxisGain;                                   // Axis gains
        double* m_pdAxisOffset;                                 // Axis offsets

        double dTime;                                                   // Time flag

        double* m_pdLinkCurrentHeadCommand;     // most recent commands you got from vec sig (for new head)
        double* m_pdLinkCurrentHeadSample;      // most recent sample you sent to vec sig (for new head)
        long*   m_plAxisThreshold;                      // each axis' threshold value. if
                                                                                                        // new_command -
old_command <
                                                                                                        // threshold, the
new_command will not be sent to that axis
  // for the old signal vector 09/10/03
//      double* m_pdLinkOldHeadCommand;         // most recent commands for the old head
//      double* m_pdLinkOldHeadSample;          // most recent sample for the old head


        long*                   m_plCurrentSample;
        long*                   m_psOut;                        // temp value

        CCommPort*              pCommLEFT;                              // COM1 for Left pan/tilt control
        CCommPort*              pCommRIGHT;                             // COM2 for right pan/tilt control

        bool m_bFirstCue;               // Will be used as a flag to avg. sound Cues
        double* m_pdBuffer;             // Stores data to be averaged later

        int m_iFlagCue;                         // Flag for median calculation
        double* m_pdBuff1;              // Buffer to calculate median from three cues.
        double* m_pdBuff2;
        double* m_pdBuff3;


};
```

```
/*** I have to encapsulate this motor controlling ***/
/*** why? 'cuz IMA sucks!!! ***/
//#include "stdafx.h"
#include "windows.h"
#include "atlbase.h"
#include "commport.h"
#include "string.h"
#include "stdio.h"
//#include "iostream.h"
#include <iostream>
#include "conio.h"
#include <math.h>
#include "cv.h"                    //image stuff
////////////////////////////////////////////////////////////////////////////
// CCatch
class CameraHead
{
public:

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

        Name:                           CCatch Component Constructor

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

        CameraHead() :
                        // *** Note: If you want to change the command packet layout, then
                        // you need to change these next 4 constants!

                                LEFT_PAN(0),                    // index of the left pan motor in the arrays
                                LEFT_TILT(1),                   // index of the left tilt motor in the arrays
                                RIGHT_PAN(2),                   // index of the right pan motor in the arrays
                                RIGHT_TILT(3),                  // index of the right tilt motor in the arrays

                                NUM_AXES(6),                    // # axes on the left/right pan-tilt units currently.
//                              NUM_AXES_ALL(6),                // # axes on the Left/right pan-tilt units and overall
pan-tilt.

                                COMMAND_PAN_ABS("pp"),  // string which indicates you are sending a command to the
pan motor.
                                COMMAND_TILT_ABS("tp"), // string which indicates you are sending a command to the tilt
motor.
                                COMMAND_PAN_REL("po"),  // string which commands pan relative
                                COMMAND_TILT_REL("to"), // string which commands tilt relative

                                QUERY_PAN("pp"),                // get the pan position
                                QUERY_TILT("tp"),               // get the tilt position
                                QUERY_MIN_PAN("pn"),            // get min pan limit
                                QUERY_MAX_PAN("px"),            // get max pan limit
                                QUERY_MIN_TILT("tn"),           // get min tilt limit
                                QUERY_MAX_TILT("tx"),           // get max tilt limit
                                QUERY_PAN_SPEED("ps"),          // get pan speed
                                QUERY_TILT_SPEED("ts"),         // get tilt speed
                                QUERY_PAN_ACCEL("pa"),          // get pan acceleration
                                QUERY_TILT_ACCEL("ta"),         // get tilt acceleration

                                SET_PAN_ACCEL("pa"),            // set pan acceleration
                                SET_TILT_ACCEL("ta"),           // set tilt acceleration
                                SET_PAN_SPEED("ps"),            // set pan speed
                                SET_TILT_SPEED("ts"),           // set tilt speed

                                HALT_ALL("h"),                          // stop all pan-tilt unit
                                RESET("r"),                                     // reset the pan-tilt unit
                                RESTORE_DEFAULTS("df"),         // restore factory default settings
                                DISABLE_LIMITS("ld"),           // disable limits on DP head.

                                SYNC_DP("a"),                           // allow last DP command to finish
                                TERSE_FEEDBACK("ft"),           // no wordy responses, Newman.
                                DISABLE_ECHO("ed"),                     // don't echo commands we send.
```

```
                                        // DP PAN
                                        MOTOR_RANGE_ANGLES_PAN(318.0f),     // pan angle range
                                        MOTOR_RANGE_PULSES_PAN_LEFT(6184.0),    // pulses in left servo's range
                                        MOTOR_RANGE_PULSES_PAN_RIGHT(6178.0), // pulses in right servo's range
                                        PULSE_TO_ANG_PAN_LEFT((double)
(MOTOR_RANGE_ANGLES_PAN/MOTOR_RANGE_PULSES_PAN_LEFT)),
                                        PULSE_TO_ANG_PAN_RIGHT((double)
(MOTOR_RANGE_ANGLES_PAN/MOTOR_RANGE_PULSES_PAN_RIGHT)),

                                // DP Tilt
                                        MOTOR_RANGE_ANGLES_TILT(106.0f),    // tilt angle range
                                        MOTOR_RANGE_PULSES_TILT(2061.0f),   // pulses in servo's range
                                        PULSE_TO_ANG_TILT((double)
(MOTOR_RANGE_ANGLES_TILT/MOTOR_RANGE_PULSES_TILT))


        {

                // TODO: Initialize all variables
                m_pdAxisGain = NULL;
                m_pdAxisOffset = NULL;
                m_plAxisThreshold = NULL;

                m_plCurrentSample = NULL;
                m_pdLinkCurrentHeadCommand = NULL;
                m_pdLinkCurrentHeadSample = NULL;
        //      m_pdLinkOldHeadCommand = NULL;
//              m_pdLinkOldHeadSample = NULL;


                pCommLEFT = NULL;
            pCommRIGHT = NULL;
                m_psOut = NULL;

                m_pdBuffer = NULL;
                m_bFirstCue = false;

                m_iFlagCue = 0;

                m_pdBuff1 = new double[4];
                m_pdBuff2 = new double[4];
                m_pdBuff3 = new double[4];

                for (int i=0; i<4; i++)
                {
                        m_pdBuff1[i] = 0;
                        m_pdBuff2[i] = 1;
                        m_pdBuff3[i] = 2;
                }


                // setup class variables to appropriate sizes
                if ( NUM_AXES > 0 )
                {
                        m_psOut = new long[NUM_AXES];

                        m_pdAxisGain =          new double[ NUM_AXES ];
                        m_pdAxisOffset = new double[ NUM_AXES ];
                        m_plAxisThreshold = new long[ NUM_AXES ];

                        m_pdLinkCurrentHeadCommand = new double[ NUM_AXES ];

                        // new head will display 2 more values: representing a
                        // overall pan/tilt value for both cameras
                        m_pdLinkCurrentHeadSample = new double[ NUM_AXES];
                        m_plCurrentSample = new long[ NUM_AXES ];

                        // Two missing pointers were noticed not having been allocated
```

```
                        // It is done here:
//                      m_pdLinkOldHeadCommand = new double[ NUM_AXES];
//                      m_pdLinkOldHeadSample = new double[ NUM_AXES];
                }


                // initialize axis gains and offsets

                for ( int j = 0; j < NUM_AXES; j++ )
                {
                        m_pdAxisGain[j] = 1.0;
                        m_pdAxisOffset[j] = 0.0;
                        m_plAxisThreshold[j] = 0;
                }

                // setup default values for motors and communications parameters
                m_plCurrentSample[ LEFT_PAN    ] = 0;
                m_plCurrentSample[ LEFT_TILT ] = 0;
                m_plCurrentSample[ RIGHT_PAN ] = 0;
                m_plCurrentSample[ RIGHT_TILT] = 0;

                m_bsDevFilename1 = "\\COM1";
                m_bsDevFilename2 = "\\COM2";
                m_bsCParam1 = "";
                m_bsCParam2 = "";
                m_lCTimeout1 = 0;
                m_lCTimeout2 = 0;

        }


        ~CameraHead()
        {
                /********** Beginning CCatch::~CCatch() **********/
                        if ( m_psOut != NULL )
                        delete[] m_psOut;

                m_psOut = NULL;


                ATLTRACE("*** Deleting Commports ....");
                if ( pCommLEFT != NULL )
                        delete pCommLEFT;
                pCommLEFT = NULL;
                ATLTRACE(" ... PT Left destroyed,");

                if ( pCommRIGHT != NULL )
                        delete pCommRIGHT;
                pCommRIGHT = NULL;
                ATLTRACE(" PT Right destroyed ...\n");

                if ( m_pdAxisGain != NULL )
                        delete[] m_pdAxisGain;
                m_pdAxisGain = NULL;

                if ( m_pdAxisOffset != NULL )
                        delete[] m_pdAxisOffset;
                m_pdAxisOffset = NULL;

                if ( m_plAxisThreshold != NULL )
                        delete[] m_plAxisThreshold;
                m_plAxisThreshold = NULL;

                if ( m_plCurrentSample != NULL )
                        delete[] m_plCurrentSample;
                m_plCurrentSample = NULL;

                /********** CCatch::~CCatch() Deleted Current Sample**********/
```

```
		if ( m_pdLinkCurrentHeadCommand != NULL )
			delete[] m_pdLinkCurrentHeadCommand;
		m_pdLinkCurrentHeadCommand = NULL;

		/********** CCatch::~CCatch() Deleted Link Head Command**********/


//		ATLTRACE("********** CCatch::~CCatch() Deleted Link Head Command**********\n");


		if ( m_pdLinkCurrentHeadSample != NULL )
			delete[] m_pdLinkCurrentHeadSample;
		m_pdLinkCurrentHeadSample = NULL;


		/********** Finishing CCatch::~CCatch() **********/
//		ATLTRACE("********** Finishing CCatch::~CCatch() **********\n");


	}


public:
//	double m_dPreviousPosition[2];		// for pan and tilt velocity
	long LeftPanSpeed;
	long RightPanSpeed;
	long LeftPanAccel;
	long RightPanAccel;
	long LeftTiltSpeed;
	long RightTiltSpeed;
	long LeftTiltAccel;
	long RightTiltAccel;
//	long VergeLimit( long Val );

public:
	/** conversion function ***/
	BOOL HeadAngles2Positions( double* pVal, long length );
	BOOL HeadPositions2Angles( double* pVal, long Length );
	int ParseStringToInteger( char* pStr );

	/** internal command **/
	BOOL CommandHeadAbsolute(double* Val, long Length);
	BOOL SampleHead(double* Val, long Length);
//	BOOL CommandHeadRelative(double* Val, long Length);
	bool CalculateOverallPanTilt(double* Val);
	BOOL DPSample();
	BOOL ExceedsThreshold( long, long );
	BOOL SendToLeft( char* pStr );
	BOOL SendToRight( char* pStr );

	/** auxillary command **/
	bool IndexIsValid( long Index );

	/** properties **/
	BOOL get_AxisThreshold(long Index, long* pVal);
	BOOL put_AxisThreshold(long Index, long newVal);
	BOOL get_LeftTiltSpeed(long *pVal);
	BOOL get_RightTiltSpeed(long *pVal);
	BOOL put_LeftTiltSpeed(long newVal);
	BOOL put_RightTiltSpeed(long newVal);
	BOOL get_LeftPanSpeed(long *pVal);
	BOOL get_RightPanSpeed(long *pVal);
	BOOL put_LeftPanSpeed(long newVal);
	BOOL put_RightPanSpeed(long newVal);
	BOOL get_LeftTiltAccel(long *pVal);
	BOOL get_RightTiltAccel(long *pVal);
	BOOL put_LeftTiltAccel(long newVal);
	BOOL put_RightTiltAccel(long newVal);
```

```
        BOOL get_LeftPanAccel(long *pVal);
        BOOL get_RightPanAccel(long *pVal);
        BOOL put_LeftPanAccel(long newVal);
        BOOL put_RightPanAccel(long newVal);
//      BOOL get_TiltLimit(long Length, long *pVal);
//      BOOL get_PanLimit(long Length, long *pVal);
//      BOOL get_CommReadTimeout2(long *pVal);
//      BOOL put_CommReadTimeout2(long newVal);
//      BOOL get_CommReadTimeout1(long *pVal);
//      BOOL put_CommReadTimeout1(long newVal);
//      BOOL get_AxisGain(long Index, double *pVal);
//      BOOL put_AxisGain(long Index, double newVal);
//      BOOL get_AxisOffset(long Index, double *pVal);
//      BOOL put_AxisOffset(long Index, double newVal);
//      BOOL get_CommParams2(char *pVal);
//      BOOL put_CommParams2(char newVal);
//      BOOL get_CommParams1(char *pVal);
//      BOOL put_CommParams1(char newVal);
//      BOOL get_DeviceFilename2(char *pVal);
//      BOOL put_DeviceFilename2(char newVal);
//      BOOL get_DeviceFilename1(char *pVal);
//      BOOL put_DeviceFilename1(char newVal);

public:
        /** general interface command **/
        BOOL Reset();
        BOOL Stop();
        BOOL Initialize();
        void MoveHead(double *command);
        void GetHeadPositions(double *val);
        void Home();


//private:

        // constants useful for this class. (OK--kat)
        // DP Pan
        const double MOTOR_RANGE_ANGLES_PAN;        // pan angle range
        const double MOTOR_RANGE_PULSES_PAN_LEFT;   // pulses in left servo's range
        const double MOTOR_RANGE_PULSES_PAN_RIGHT;  // pulses in right servo's range
        const double PULSE_TO_ANG_PAN_LEFT;
        const double PULSE_TO_ANG_PAN_RIGHT;

        // DP Tilt
        const double MOTOR_RANGE_ANGLES_TILT;       // tilt angle range
        const double MOTOR_RANGE_PULSES_TILT;       // pulses in servo's range
        const double PULSE_TO_ANG_TILT;

        const long      NUM_AXES;                   // how many axes does this pantilt have?

        const char*             COMMAND_TILT_REL;   // command for relative tilt motion
        const char*             COMMAND_PAN_REL;    // command for relative pan motion
        const char*             COMMAND_TILT_ABS;   // command for absolute tilt motion
        const char*             COMMAND_PAN_ABS;    // command for absolute pan motion

        const long      LEFT_PAN;
        const long      LEFT_TILT;
        const long      RIGHT_PAN;
        const long      RIGHT_TILT;

        const char*             QUERY_TILT;             // get the tilt position
        const char*             QUERY_PAN;              // get the pan position

        const char*             QUERY_MIN_PAN;          // get min pan limit
        const char*             QUERY_MAX_PAN;          // get max pan limit
        const char*             QUERY_MIN_TILT;         // get min tilt limit
        const char*             QUERY_MAX_TILT;         // get max tilt limit
        const char*             QUERY_PAN_SPEED;    // get pan speed
        const char*             QUERY_PAN_ACCEL;    // get pan acceleration
```

```
        const char*             QUERY_TILT_SPEED;       // get tilt speed
        const char*             QUERY_TILT_ACCEL;       // get tilt acceleration

        const char*             SET_PAN_ACCEL;                  // set pan acceleration
        const char*             SET_PAN_SPEED;                  // set pan speed
        const char*             SET_TILT_ACCEL;                 // set tilt acceleration
        const char*             SET_TILT_SPEED;                 // set tilt speed

        const char*             HALT_ALL;                               // stop the pan-tilt unit

        const char*             RESET;                                  // reset the pan-tilt unit
        const char*             RESTORE_DEFAULTS;               // restore factory default settings

        const char*             DISABLE_LIMITS;                 // disable limits on DP head.
        const char*             DISABLE_ECHO;                   // don't echo commands we send.

        const char*             SYNC_DP;                        // allow last DP command to finish

        const char*             TERSE_FEEDBACK;                 // no wordy responses, Newman.

        // COMMPORT parameters
        CComBSTR m_bsDevFilename2;                              // Device filenames for 1 & 2
        CComBSTR m_bsDevFilename1;
        CComBSTR m_bsCParam2;                                   // Device parameters for 1 & 2
        CComBSTR m_bsCParam1;

        long    m_lCTimeout1;                                   // Read timeouts for 1 & 2
        long    m_lCTimeout2;

        double* m_pdAxisGain;                                   // Axis gains
        double* m_pdAxisOffset;                                 // Axis offsets

        double dTime;                                                   // Time flag

        double* m_pdLinkCurrentHeadCommand;     // most recent commands you got from vec sig (for new head)
        double* m_pdLinkCurrentHeadSample;      // most recent sample you sent to vec sig (for new head)
        long*   m_plAxisThreshold;                      // each axis' threshold value. if
                                                                                                        // new_command -
old_command <
                                                                                                        // threshold, the
new_command will not be sent to that axis
  // for the old signal vector 09/10/03
//      double* m_pdLinkOldHeadCommand;         // most recent commands for the old head
//      double* m_pdLinkOldHeadSample;          // most recent sample for the old head


        long*                   m_plCurrentSample;
        long*                   m_psOut;                        // temp value

        CCommPort*              pCommLEFT;                              // COM1 for Left pan/tilt control
        CCommPort*              pCommRIGHT;                             // COM2 for right pan/tilt control

        bool m_bFirstCue;               // Will be used as a flag to avg. sound Cues
        double* m_pdBuffer;             // Stores data to be averaged later

        int m_iFlagCue;                         // Flag for median calculation
        double* m_pdBuff1;              // Buffer to calculate median from three cues.
        double* m_pdBuff2;
        double* m_pdBuff3;


};

// C++ container classes for obtaining exclusive access
//    to the comm port for simple use such as reading/writing
//    ASCII characters. Handles all setup so the user
//    simply calls the WriteComm or ReadComm functions to read
//    and write to comm port specified in the constructor.
//************
```

```
// Summer 1996
// Joe Christopher
// MFC parts taken out by Rick Watson 8/5/1997
//************

//#include "stdafx.h"

// Standard library
#include "windows.h"
#include   <stdlib.h>

class CCommPort
{
public:
        CCommPort(char* Port, ULONG Baud);
        //CCommPort(char* Port="\\COM2" , ULONG    Baud=9600 );  // Constructor with default COM2, 9600 baud
        ~CCommPort();
        BOOL WriteComm( char* Output);
        BOOL WriteChar( char Output );
        BOOL ReadComm( char* Input, ULONG Bytes );

protected:
        // Vars. used in setting up and obtaining COMM port
        DWORD m_dwError;
        DWORD m_dwNumBytes;
        COMMTIMEOUTS m_ctTimeouts;

        BOOL    m_bGetComm;         // Booleans for evaluating successful setup
        BOOL    m_bSetComm;         // of COMM port parameters such as baud, parity
        BOOL    m_bGetTime;         // stop bits, timeouts, byte size, etc.
        BOOL    m_bSetTime;         //

        DCB                 m_dDcb;             // device control block for the port
        HANDLE  m_hCom;             // handle for the comm port

 };
```

```
// ArmControlDlg.cpp : implementation file
//

#include "stdafx.h"
#include "ArmControl.h"
#include "ArmControlDlg.h"
#include "math.h"

#include "Headers\winmotenc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define PI 3.1415926535897931

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CArmControlDlg dialog

CArmControlDlg::CArmControlDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CArmControlDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CArmControlDlg)
```

```
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CArmControlDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CArmControlDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CArmControlDlg, CDialog)
	//{{AFX_MSG_MAP(CArmControlDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_Start, OnStart)
	ON_BN_CLICKED(IDC_End, OnEnd)
	ON_BN_CLICKED(IDC_HomePosition, OnHomePosition)
	ON_BN_CLICKED(IDC_Gripper_Right, OnGripperRight)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CArmControlDlg message handlers

BOOL CArmControlDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.	The framework does this automatically
	//	when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	AngleCurrentLeft.SetParent(this);
	AngleCurrentRight.SetParent(this);
	AngleDesiredLeft.SetParent(this);
	AngleDesiredRight.SetParent(this);
	AngleErrorCurrentLeft.SetParent(this);
	AngleErrorCurrentRight.SetParent(this);
	AnglePreviousLeft.SetParent(this);
	AnglePreviousRight.SetParent(this);
	BaseToWristRotationMatrixLeft.SetParent(this);
	BaseToEndRotationMatrixLeft.SetParent(this);
	BaseToWristRotationMatrixRight.SetParent(this);
	BaseToEndRotationMatrixRight.SetParent(this);
```

```
	ConnectSocket.SetParent(this);
	EncoderCurrentLeft.SetParent(this);
	EncoderCurrentRight.SetParent(this);
	ISACParameterLeft.SetParent(this);
	ISACParameterRight.SetParent(this);
	OrientationCurrentLeft.SetParent(this);
	OrientationCurrentRight.SetParent(this);
	OrientationDesiredLeft.SetParent(this);
	OrientationDesiredRight.SetParent(this);
	ParameterDifferentiationLeft.SetParent(this);
	ParameterDifferentiationRight.SetParent(this);
	ParameterIntegrationLeft.SetParent(this);
	ParameterIntegrationRight.SetParent(this);
	ParameterProportionLeft.SetParent(this);
	ParameterProportionRight.SetParent(this);
	PositionCurrentLeft.SetParent(this);
	PositionCurrentRight.SetParent(this);
	PositionDesiredLeft.SetParent(this);
	PositionDesiredRight.SetParent(this);
	PressureLeft.SetParent(this);
	PressureCurrentLeft.SetParent(this);
	PressureCurrentRight.SetParent(this);
	PressurePreviousLeft.SetParent(this);
	PressurePreviousRight.SetParent(this);
	ServerSocket.SetParent(this);
	ConnectSocket.SetParent(this);
	WristToEndRotationMatrixLeft.SetParent(this);
	WristToEndRotationMatrixRight.SetParent(this);
	Initialization();
	ServerSocket.Create(4000);
	ServerSocket.Listen();
	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CArmControlDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CArmControlDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
```

```
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//	the minimized window.
HCURSOR CArmControlDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CArmControlDlg::Initialization()
{
	int i=0;
	int j=0;
	int k=0;
	int numb =0;
	//Initializing the Cards
	if( numb=vitalInit() )
		printf("%d board(s) detected\n",numb);
	else
		printf( "Error initializing WinMotenc library\n" );

	double BufferVoltage=0.1;
	ISACParameterLeft.D3=200;
	ISACParameterLeft.D4=290;
	ISACParameterLeft.A2=330;
	ISACParameterLeft.A3=0;
	ISACParameterLeft.Alpha0=0;
	ISACParameterLeft.Alpha1=-PI/2;
	ISACParameterLeft.Alpha2=0;
	ISACParameterLeft.Alpha3=-PI/2;
	ISACParameterLeft.Alpha4=PI/2;
	ISACParameterLeft.Alpha5=-PI/2;

	PositionDesiredLeft.X=PositionCurrentLeft.X=ISACParameterLeft.D4;
	PositionDesiredLeft.Y=PositionCurrentLeft.Y=ISACParameterLeft.D3;
	PositionDesiredLeft.Z=PositionCurrentLeft.Z=-ISACParameterLeft.A2;
	OrientationDesiredLeft.R=OrientationCurrentLeft.R=0;
	OrientationDesiredLeft.P=OrientationCurrentLeft.P=0;
	OrientationDesiredLeft.W=OrientationCurrentLeft.W=0;

	ParameterProportionLeft.Muscle0=0.0009;
	ParameterProportionLeft.Muscle1=0.0009;
	ParameterProportionLeft.Muscle2=0.0013;
	ParameterProportionLeft.Muscle3=0.0013;
	ParameterProportionLeft.Muscle4=0.0003;
	ParameterProportionLeft.Muscle5=0.0015;
	ParameterProportionLeft.Muscle6=0.0015;
	ParameterProportionLeft.Muscle7=0.0003;
	ParameterProportionLeft.Muscle8=0.0015;
	ParameterProportionLeft.Muscle9=0.0015;
	ParameterProportionLeft.Muscle10=0.0003;
	ParameterProportionLeft.Muscle11=0.0003;

	ISACParameterRight.D3=200;
	ISACParameterRight.D4=290;
	ISACParameterRight.A2=330;
	ISACParameterRight.A3=0;
	ISACParameterRight.Alpha0=0;
	ISACParameterRight.Alpha1=-PI/2;
	ISACParameterRight.Alpha2=0;
	ISACParameterRight.Alpha3=-PI/2;
	ISACParameterRight.Alpha4=PI/2;
	ISACParameterRight.Alpha5=-PI/2;
```

```
	PositionDesiredRight.X=PositionCurrentRight.X=ISACParameterRight.D4;
	PositionDesiredRight.Y=PositionCurrentRight.Y=-ISACParameterRight.D3;
	PositionDesiredRight.Z=PositionCurrentRight.Z=-ISACParameterRight.A2;
	OrientationDesiredLeft.R=OrientationCurrentRight.R=0;
	OrientationDesiredLeft.P=OrientationCurrentRight.P=0;
	OrientationDesiredLeft.W=OrientationCurrentRight.W=0;

	ParameterProportionRight.Muscle0=0.0009*3;
	ParameterProportionRight.Muscle1=0.0009*3;
	ParameterProportionRight.Muscle2=0.0013*3;
	ParameterProportionRight.Muscle3=0.0013*3;
	ParameterProportionRight.Muscle4=0.0003*3;
	ParameterProportionRight.Muscle5=0.0015*3;
	ParameterProportionRight.Muscle6=0.0015*3;
	ParameterProportionRight.Muscle7=0.0003*3;
	ParameterProportionRight.Muscle8=0.0015*3;
	ParameterProportionRight.Muscle9=0.0015*3;
	ParameterProportionRight.Muscle10=0.0015*3;
	ParameterProportionRight.Muscle11=0.0015*3;

	PressurePreviousLeft.Muscle0=PressureLeft.Muscle0=PressureCurrentLeft.Muscle0=2.3;
	PressurePreviousLeft.Muscle1=PressureLeft.Muscle1=PressureCurrentLeft.Muscle1=1.7;
	PressurePreviousLeft.Muscle2=PressureLeft.Muscle2=PressureCurrentLeft.Muscle2=2.0;
	PressurePreviousLeft.Muscle3=PressureLeft.Muscle3=PressureCurrentLeft.Muscle3=2.0;
	PressurePreviousLeft.Muscle4=PressureLeft.Muscle4=PressureCurrentLeft.Muscle4=0;
	PressurePreviousLeft.Muscle5=PressureLeft.Muscle5=PressureCurrentLeft.Muscle5=2.2;
	PressurePreviousLeft.Muscle6=PressureLeft.Muscle6=PressureCurrentLeft.Muscle6=2.4;
	PressurePreviousLeft.Muscle7=PressureLeft.Muscle7=PressureCurrentLeft.Muscle7=0;
	PressurePreviousLeft.Muscle8=PressureLeft.Muscle8=PressureCurrentLeft.Muscle8=2.3;
	PressurePreviousLeft.Muscle9=PressureLeft.Muscle9=PressureCurrentLeft.Muscle9=1.7;
	PressurePreviousLeft.Muscle10=PressureLeft.Muscle10=PressureCurrentLeft.Muscle10=1.7;
	PressurePreviousLeft.Muscle11=PressureLeft.Muscle11=PressureCurrentLeft.Muscle11=2.3;

	PressurePreviousRight.Muscle0=PressureRight.Muscle0=PressureCurrentRight.Muscle0=2.1;
	PressurePreviousRight.Muscle1=PressureRight.Muscle1=PressureCurrentRight.Muscle1=1.9;
	PressurePreviousRight.Muscle2=PressureRight.Muscle2=PressureCurrentRight.Muscle2=2.0;
	PressurePreviousRight.Muscle3=PressureRight.Muscle3=PressureCurrentRight.Muscle3=2.0;
	PressurePreviousRight.Muscle4=PressureRight.Muscle4=PressureCurrentRight.Muscle4=0;
	PressurePreviousRight.Muscle5=PressureRight.Muscle5=PressureCurrentRight.Muscle5=2.2;
	PressurePreviousRight.Muscle6=PressureRight.Muscle6=PressureCurrentRight.Muscle6=2.4;
	PressurePreviousRight.Muscle7=PressureRight.Muscle7=PressureCurrentRight.Muscle7=0;
	PressurePreviousRight.Muscle8=PressureRight.Muscle8=PressureCurrentRight.Muscle8=3.6;
	PressurePreviousRight.Muscle9=PressureRight.Muscle9=PressureCurrentRight.Muscle9=0.4;
	PressurePreviousRight.Muscle10=PressureRight.Muscle10=PressureCurrentRight.Muscle10=0.4;
	PressurePreviousRight.Muscle11=PressureRight.Muscle11=PressureCurrentRight.Muscle11=3.6;
	for(i=0;i<25;i++) // 25x100 ms = 2.5 sec makes the muscles blow up
	{
/*		vitalSelectBoard(0);
		if(BufferVoltage <= PressureCurrentLeft.Muscle1 + 0.05)
		{
			vitalDacWrite(1, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle2 + 0.05)
		{
			vitalDacWrite(2, BufferVoltage);
		}
*/		BufferVoltage +=0.1; // Increasing the ref voltage slowly not to harm
		Sleep(100); // Wait 100 ms at each step
		vitalSelectBoard(1);
		if(BufferVoltage <= PressureCurrentRight.Muscle0 + 0.05)
		{
			vitalDacWrite(4, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentRight.Muscle2 + 0.05)
		{
			vitalDacWrite(6, BufferVoltage);
		}
		BufferVoltage +=0.1; // Increasing the ref voltage slowly not to harm
		Sleep(100); // Wait 100 ms at each step
```

```
	}
	Sleep(200);

	BufferVoltage =0.1;
	for(i=0;i<25;i++) // 25x100 ms = 2.5 sec makes the muscles blow up
	{
/*		vitalSelectBoard(0);
		if(BufferVoltage <= PressureCurrentLeft.Muscle0 + 0.05)
		{
			vitalDacWrite(0, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle3 + 0.05)
		{
			vitalDacWrite(3, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle4 + 0.05)
		{
			vitalDacWrite(4, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle5 + 0.05)
		{
			vitalDacWrite(5, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle6 + 0.05)
		{
			vitalDacWrite(6, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle7 + 0.05)
		{
			vitalDacWrite(7, BufferVoltage);
		}
		vitalSelectBoard(1);
		if(BufferVoltage <= PressureCurrentLeft.Muscle8 + 0.05)
		{
			vitalDacWrite(0, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle9 + 0.05)
		{
			vitalDacWrite(1, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle10 + 0.05)
		{
			vitalDacWrite(2, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentLeft.Muscle11 + 0.05)
		{
			vitalDacWrite(3, BufferVoltage);
		}
*/		// Right Arm
		vitalSelectBoard(1);
		if(BufferVoltage <= PressureCurrentRight.Muscle1 + 0.05)
		{
			vitalDacWrite(5, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentRight.Muscle3 + 0.05)
		{
			vitalDacWrite(7, BufferVoltage);
		}
		vitalSelectBoard(2);

			vitalDacWrite(0, 0);

		if(BufferVoltage <= PressureCurrentRight.Muscle5 + 0.05)
		{
			vitalDacWrite(1, BufferVoltage);
		}
		if(BufferVoltage <= PressureCurrentRight.Muscle6 + 0.05)
		{
			vitalDacWrite(2, BufferVoltage);
```

```
	}

		vitalDacWrite(3, 0);

	if(BufferVoltage <= PressureCurrentRight.Muscle8 + 0.05)
	{
		vitalDacWrite(4, BufferVoltage);
	}
	if(BufferVoltage <= PressureCurrentRight.Muscle9 + 0.05)
	{
		vitalDacWrite(5, BufferVoltage);
	}
	if(BufferVoltage <= PressureCurrentRight.Muscle10 + 0.05)
	{
		vitalDacWrite(6, BufferVoltage);
	}
	if(BufferVoltage <= PressureCurrentRight.Muscle11 + 0.05)
	{
		vitalDacWrite(7, BufferVoltage);
	}
	BufferVoltage +=0.1; // Increasing the ref voltage slowly not to harm
	Sleep(100); // Wait 100 ms at each step
}

Gripper_Pressure_Right=0;
vitalSelectBoard(1);
vitalEncoderResetIndex(2);
vitalEncoderResetIndex(3);
vitalEncoderSetIndexModel(EXT_ENCODER_INDEX_MODEL_AUTO);
vitalSelectBoard(2);
vitalEncoderResetIndex(0);
vitalEncoderResetIndex(1);
vitalEncoderSetIndexModel(EXT_ENCODER_INDEX_MODEL_AUTO);
Sleep(200);
ResetLeftEncoders();
ResetRightEncoders();
ReadLeftEncoders();
ReadRightEncoders();
ComputeLeftAngles();
ComputeRightAngles();
AngleDesiredLeft.Joint1=AnglePreviousLeft.Joint1=AngleCurrentLeft.Joint1;
AngleDesiredLeft.Joint2=AnglePreviousLeft.Joint2=AngleCurrentLeft.Joint2;
AngleDesiredLeft.Joint3=AnglePreviousLeft.Joint3=AngleCurrentLeft.Joint3;
AngleDesiredLeft.Joint4=AnglePreviousLeft.Joint4=AngleCurrentLeft.Joint4;
AngleDesiredLeft.Joint5=AnglePreviousLeft.Joint5=AngleCurrentLeft.Joint5;
AngleDesiredLeft.Joint6=AnglePreviousLeft.Joint6=AngleCurrentLeft.Joint6;

AngleDesiredRight.Joint1=AnglePreviousRight.Joint1=AngleCurrentRight.Joint1;
AngleDesiredRight.Joint2=AnglePreviousRight.Joint2=AngleCurrentRight.Joint2;
AngleDesiredRight.Joint3=AnglePreviousRight.Joint3=AngleCurrentRight.Joint3;
AngleDesiredRight.Joint4=AnglePreviousRight.Joint4=AngleCurrentRight.Joint4;
AngleDesiredRight.Joint5=AnglePreviousRight.Joint5=AngleCurrentRight.Joint5;
AngleDesiredRight.Joint6=AnglePreviousRight.Joint6=AngleCurrentRight.Joint6;

SetDlgItemInt(IDC_LeftCurrentJoint1,(int)AngleCurrentLeft.Joint1);
SetDlgItemInt(IDC_LeftCurrentJoint2,(int)AngleCurrentLeft.Joint2);
SetDlgItemInt(IDC_LeftCurrentJoint3,(int)AngleCurrentLeft.Joint3);
SetDlgItemInt(IDC_LeftCurrentJoint4,(int)AngleCurrentLeft.Joint4);
SetDlgItemInt(IDC_LeftCurrentJoint5,(int)AngleCurrentLeft.Joint5);
SetDlgItemInt(IDC_LeftCurrentJoint6,(int)AngleCurrentLeft.Joint6);

SetDlgItemInt(IDC_RightCurrentJoint1,(int)AngleCurrentRight.Joint1);
SetDlgItemInt(IDC_RightCurrentJoint2,(int)AngleCurrentRight.Joint2);
SetDlgItemInt(IDC_RightCurrentJoint3,(int)AngleCurrentRight.Joint3);
SetDlgItemInt(IDC_RightCurrentJoint4,(int)AngleCurrentRight.Joint4);
SetDlgItemInt(IDC_RightCurrentJoint5,(int)AngleCurrentRight.Joint5);
SetDlgItemInt(IDC_RightCurrentJoint6,(int)AngleCurrentRight.Joint6);

SetDlgItemInt(IDC_LeftCurrentX,(int)PositionCurrentLeft.X);
```

```
	SetDlgItemInt(IDC_LeftCurrentY,(int)PositionCurrentLeft.Y);
	SetDlgItemInt(IDC_LeftCurrentZ,(int)PositionCurrentLeft.Z);
	SetDlgItemInt(IDC_LeftCurrentR,(int)OrientationCurrentLeft.R);
	SetDlgItemInt(IDC_LeftCurrentP,(int)OrientationCurrentLeft.P);
	SetDlgItemInt(IDC_LeftCurrentW,(int)OrientationCurrentLeft.W);

	SetDlgItemInt(IDC_RightCurrentX,(int)PositionCurrentRight.X);
	SetDlgItemInt(IDC_RightCurrentY,(int)PositionCurrentRight.Y);
	SetDlgItemInt(IDC_RightCurrentZ,(int)PositionCurrentRight.Z);
	SetDlgItemInt(IDC_RightCurrentR,(int)OrientationCurrentRight.R);
	SetDlgItemInt(IDC_RightCurrentP,(int)OrientationCurrentRight.P);
	SetDlgItemInt(IDC_RightCurrentW,(int)OrientationCurrentRight.W);

	SetDlgItemInt(IDC_LeftDesiredJoint1,(int)AngleDesiredLeft.Joint1);
	SetDlgItemInt(IDC_LeftDesiredJoint2,(int)AngleDesiredLeft.Joint2);
	SetDlgItemInt(IDC_LeftDesiredJoint3,(int)AngleDesiredLeft.Joint3);
	SetDlgItemInt(IDC_LeftDesiredJoint4,(int)AngleDesiredLeft.Joint4);
	SetDlgItemInt(IDC_LeftDesiredJoint5,(int)AngleDesiredLeft.Joint5);
	SetDlgItemInt(IDC_LeftDesiredJoint6,(int)AngleDesiredLeft.Joint6);

	SetDlgItemInt(IDC_RightDesiredJoint1,(int)AngleDesiredRight.Joint1);
	SetDlgItemInt(IDC_RightDesiredJoint2,(int)AngleDesiredRight.Joint2);
	SetDlgItemInt(IDC_RightDesiredJoint3,(int)AngleDesiredRight.Joint3);
	SetDlgItemInt(IDC_RightDesiredJoint4,(int)AngleDesiredRight.Joint4);
	SetDlgItemInt(IDC_RightDesiredJoint5,(int)AngleDesiredRight.Joint5);
	SetDlgItemInt(IDC_RightDesiredJoint6,(int)AngleDesiredRight.Joint6);

	SetDlgItemInt(IDC_LeftDesiredX,(int)PositionDesiredLeft.X);
	SetDlgItemInt(IDC_LeftDesiredY,(int)PositionDesiredLeft.Y);
	SetDlgItemInt(IDC_LeftDesiredZ,(int)PositionDesiredLeft.Z);
	SetDlgItemInt(IDC_LeftDesiredR,(int)OrientationDesiredLeft.R);
	SetDlgItemInt(IDC_LeftDesiredP,(int)OrientationDesiredLeft.P);
	SetDlgItemInt(IDC_LeftDesiredW,(int)OrientationDesiredLeft.W);

	SetDlgItemInt(IDC_RightDesiredX,(int)PositionDesiredRight.X);
	SetDlgItemInt(IDC_RightDesiredY,(int)PositionDesiredRight.Y);
	SetDlgItemInt(IDC_RightDesiredZ,(int)PositionDesiredRight.Z);
	SetDlgItemInt(IDC_RightDesiredR,(int)OrientationDesiredRight.R);
	SetDlgItemInt(IDC_RightDesiredP,(int)OrientationDesiredRight.P);
	SetDlgItemInt(IDC_RightDesiredW,(int)OrientationDesiredRight.W);
}

void CArmControlDlg::ResetLeftEncoders()
{
	vitalSelectBoard(0);
	for(int channel=0;channel<4;channel++)
	{
		vitalResetCounter(channel);
	}
	vitalSelectBoard(1);
	for(channel=0;channel<2;channel++)
	{
		vitalResetCounter(channel);
	}
}

void CArmControlDlg::ResetRightEncoders()
{
	vitalSelectBoard(1);
	for(int channel=2;channel<4;channel++)
	{
		vitalResetCounter(channel);
	}
	vitalSelectBoard(2);
	for(channel=0;channel<4;channel++)
	{
		vitalResetCounter(channel);
	}
}
```

```
void CArmControlDlg::ReadLeftEncoders()
{
	//Read Encoder Values
	long EncoderBuffer;

	vitalSelectBoard(0);
	vitalEncoderRead(0, &EncoderBuffer);
	EncoderCurrentLeft.Encoder1=EncoderBuffer;
	vitalEncoderRead(1, &EncoderBuffer);
	EncoderCurrentLeft.Encoder2=EncoderBuffer;
	vitalEncoderRead(2, &EncoderBuffer);
	EncoderCurrentLeft.Encoder3=EncoderBuffer;
	vitalEncoderRead(3, &EncoderBuffer);
	EncoderCurrentLeft.Encoder4=EncoderBuffer;

	vitalSelectBoard(1);
	vitalEncoderRead(0, &EncoderBuffer);
	EncoderCurrentLeft.Encoder5=EncoderBuffer;
	vitalEncoderRead(1, &EncoderBuffer);
	EncoderCurrentLeft.Encoder6=EncoderBuffer;

	EncoderCurrentLeft.Encoder3*=-1;
	EncoderCurrentLeft.Encoder6*=-1;
}

void CArmControlDlg::ReadRightEncoders()
{
	long EncoderBuffer;
	vitalSelectBoard(1);
	//!!!First two encoders are flipped!!! (The readings show that)
	vitalEncoderRead(2, &EncoderBuffer);
	EncoderCurrentRight.Encoder2=EncoderBuffer;
	vitalEncoderRead(3, &EncoderBuffer);
	EncoderCurrentRight.Encoder1=EncoderBuffer;

	vitalSelectBoard(2);
	vitalEncoderRead(0, &EncoderBuffer);
	EncoderCurrentRight.Encoder3=EncoderBuffer;
	vitalEncoderRead(1, &EncoderBuffer);
	EncoderCurrentRight.Encoder4=EncoderBuffer;
	vitalEncoderRead(2, &EncoderBuffer);
	EncoderCurrentRight.Encoder5=EncoderBuffer;
	vitalEncoderRead(3, &EncoderBuffer);
	EncoderCurrentRight.Encoder6=EncoderBuffer;

	EncoderCurrentRight.Encoder2*=-1;
	EncoderCurrentRight.Encoder3*=-1;
}

void CArmControlDlg::ComputeLeftAngles()
{
	AngleCurrentLeft.Joint1 =	(double) EncoderCurrentLeft.Encoder1 / -5092.0 /6.28 * 360;
	AngleCurrentLeft.Joint2 =	(double) -EncoderCurrentLeft.Encoder2 / 5092.0 /6.28 * 360 + 90;
	AngleCurrentLeft.Joint3 =	((double) -EncoderCurrentLeft.Encoder4 / 4244.0 /2 /6.28 * 360 - (double)
EncoderCurrentLeft.Encoder3 / 4244 /2 /6.28 * 360) -180 ;
	AngleCurrentLeft.Joint4 =	((double) EncoderCurrentLeft.Encoder4 / 4244.0 /6.28 * 360 - (double)
EncoderCurrentLeft.Encoder3 / 4244 /6.28 * 360 ) ;
	AngleCurrentLeft.Joint5 =	((double) -EncoderCurrentLeft.Encoder5 / 636.6 /2 /6.28 * 360 - (double)
EncoderCurrentLeft.Encoder6 / 636.6 /2 /6.28 * 360);
	AngleCurrentLeft.Joint6 =	((double) -EncoderCurrentLeft.Encoder5 / 636.6 /6.28 * 360 + (double)
EncoderCurrentLeft.Encoder6 / 636.6 /6.28 * 360);
}

void CArmControlDlg::ComputeRightAngles()
{
	AngleCurrentRight.Joint1 =	(double) EncoderCurrentRight.Encoder1 / -5092.0 /6.28 * 360;
```

```
	AngleCurrentRight.Joint2 =	(double) (EncoderCurrentRight.Encoder2 / 5092.0 /6.28 * 360);
	AngleCurrentRight.Joint3 =	((double) -EncoderCurrentRight.Encoder4 / 4244.0 /2 /6.28 * 360 - (double)
EncoderCurrentRight.Encoder3 / 4244 /2 /6.28 * 360)	;
	AngleCurrentRight.Joint4 =	((double) EncoderCurrentRight.Encoder4 / 4244.0 /6.28 * 360 - (double)
EncoderCurrentRight.Encoder3 /	4244 /6.28 * 360 ) ;
	AngleCurrentRight.Joint6=	((double) (-EncoderCurrentRight.Encoder5 / 636.6 /6.28 * 360) - (double)
EncoderCurrentRight.Encoder6 / 636.6	/6.28 * 360);
	// Opposite of Right Arm 4-5 changed and 4 multiplied by -1
	AngleCurrentRight.Joint5=	((double) (-EncoderCurrentRight.Encoder5 / 636.6/2 /6.28 * 360) + (double)
EncoderCurrentRight.Encoder6 / 636.6/2 /6.28 * 360);
}

void CArmControlDlg::CreateRotationMatrixBaseToWrist_left(double gamma, double beta, double alpha)
{
	BaseToWristRotationMatrixLeft.Element[0][0] = cos(alpha)*cos(beta);
	BaseToWristRotationMatrixLeft.Element[0][1] = cos(alpha)*sin(beta)*sin(gamma)-sin(alpha)*cos(gamma);
	BaseToWristRotationMatrixLeft.Element[0][2] = cos(alpha)*sin(beta)*cos(gamma)+sin(alpha)*sin(gamma);


	BaseToWristRotationMatrixLeft.Element[1][0] = sin(alpha)*cos(beta);
	BaseToWristRotationMatrixLeft.Element[1][1] = sin(alpha)*sin(beta)*sin(gamma)+cos(alpha)*cos(gamma);
	BaseToWristRotationMatrixLeft.Element[1][2] = sin(alpha)*sin(beta)*cos(gamma)-cos(alpha)*sin(gamma);


	BaseToWristRotationMatrixLeft.Element[2][0] = -sin(beta);
	BaseToWristRotationMatrixLeft.Element[2][1] = cos(beta)*sin(gamma);
	BaseToWristRotationMatrixLeft.Element[2][2] = cos(beta)*cos(gamma);
}

void CArmControlDlg::CreateRotationMatrixBaseToWrist_right(double gamma, double beta, double alpha)
{
	BaseToWristRotationMatrixRight.Element[0][0] = cos(alpha)*cos(beta);
	BaseToWristRotationMatrixRight.Element[0][1] = cos(alpha)*sin(beta)*sin(gamma)-sin(alpha)*cos(gamma);
	BaseToWristRotationMatrixRight.Element[0][2] = cos(alpha)*sin(beta)*cos(gamma)+sin(alpha)*sin(gamma);


	BaseToWristRotationMatrixRight.Element[1][0] = sin(alpha)*cos(beta);
	BaseToWristRotationMatrixRight.Element[1][1] = sin(alpha)*sin(beta)*sin(gamma)+cos(alpha)*cos(gamma);
	BaseToWristRotationMatrixRight.Element[1][2] = sin(alpha)*sin(beta)*cos(gamma)-cos(alpha)*sin(gamma);


	BaseToWristRotationMatrixRight.Element[2][0] = -sin(beta);
	BaseToWristRotationMatrixRight.Element[2][1] = cos(beta)*sin(gamma);
	BaseToWristRotationMatrixRight.Element[2][2] = cos(beta)*cos(gamma);
}



void CArmControlDlg::CreateRotationMatrixWristToEnd_left(double alpha, double beta, double gamma)
{
	double result1[3][3];
	double result2[3][3];
	double result3[3][3];
	result1[0][0] = 1;
	result1[0][1] = 0;
	result1[0][2] = 0;

	result1[1][0] = 0;
	result1[1][1] = cos(alpha);
	result1[1][2] = -sin(alpha);

	result1[2][0] = 0;
	result1[2][1] = sin(alpha);
	result1[2][2] = cos(alpha);


	result2[0][0] = cos(beta);
	result2[0][1] = 0;
	result2[0][2] = -sin(beta);
```

```
	result2[1][0] = 0;
	result2[1][1] = 1;
	result2[1][2] = 0;

	result2[2][0] = sin(beta);
	result2[2][1] = 0;
	result2[2][2] = cos(beta);


	result3[0][0] = cos(gamma);
	result3[0][1] = -sin(gamma);
	result3[0][2] = 0;

	result3[1][0] = sin(gamma);
	result3[1][1] = cos(gamma);
	result3[1][2] = 0;

	result3[2][0] = 0;
	result3[2][1] = 0;
	result3[2][2] = 1;
	int i;
	for( i = 0; i < 3; i++)
	{
		for( int j = 0; j < 3; j++ ){
			result1[i][j] = result1[i][0]*result2[0][j] + result1[i][1]*result2[1][j] + result1[i][2]*result2[2][j];
		}
	}
	for( i = 0; i < 3; i++)
	{
		for( int j = 0; j < 3; j++ ){
			WristToEndRotationMatrixLeft.Element[i][j] = result1[i][0]*result3[0][j] + result1[i][1]*result3[1][j] +
result1[i][2]*result3[2][j];
		}
	}
}

void CArmControlDlg::CreateRotationMatrixWristToEnd_right(double alpha, double beta, double gamma)
{
	double result1[3][3];
	double result2[3][3];
	double result3[3][3];
	result1[0][0] = 1;
	result1[0][1] = 0;
	result1[0][2] = 0;

	result1[1][0] = 0;
	result1[1][1] = cos(alpha);
	result1[1][2] = -sin(alpha);

	result1[2][0] = 0;
	result1[2][1] = sin(alpha);
	result1[2][2] = cos(alpha);


	result2[0][0] = cos(beta);
	result2[0][1] = 0;
	result2[0][2] = -sin(beta);

	result2[1][0] = 0;
	result2[1][1] = 1;
	result2[1][2] = 0;

	result2[2][0] = sin(beta);
	result2[2][1] = 0;
	result2[2][2] = cos(beta);


	result3[0][0] = cos(gamma);
```

```
	result3[0][1] = -sin(gamma);
	result3[0][2] = 0;

	result3[1][0] = sin(gamma);
	result3[1][1] = cos(gamma);
	result3[1][2] = 0;

	result3[2][0] = 0;
	result3[2][1] = 0;
	result3[2][2] = 1;
	int i;
	for( i = 0; i < 3; i++)
	{
		for( int j = 0; j < 3; j++ ){
			result1[i][j] = result1[i][0]*result2[0][j] + result1[i][1]*result2[1][j] + result1[i][2]*result2[2][j];
		}
	}
	for( i = 0; i < 3; i++)
	{
		for( int j = 0; j < 3; j++ ){
			WristToEndRotationMatrixRight.Element[i][j] = result1[i][0]*result3[0][j] + result1[i][1]*result3[1][j] +
result1[i][2]*result3[2][j];
		}
	}
}

void CArmControlDlg::InverseKinematicsLeft()
{
	double theta[6];
	double EndEffectorVect[3];
	int lengthOfEndEff = 270;
	int i=0;
	int j=0;

	double a[6];
	double d[6];
	a[0]=ISACParameterLeft.A0;
	a[1]=ISACParameterLeft.A1;
	a[2]=ISACParameterLeft.A2;
	a[3]=ISACParameterLeft.A3;
	a[4]=ISACParameterLeft.A4;
	a[5]=ISACParameterLeft.A5;

	d[0]=ISACParameterLeft.D1;
	d[1]=ISACParameterLeft.D2;
	d[2]=ISACParameterLeft.D3;
	d[3]=ISACParameterLeft.D4;
	d[4]=ISACParameterLeft.D5;
	d[5]=ISACParameterLeft.D6;

	AnglePreviousLeft.Joint1=AngleCurrentLeft.Joint1*PI/180;
	AnglePreviousLeft.Joint2=AngleCurrentLeft.Joint2*PI/180;
	AnglePreviousLeft.Joint3=AngleCurrentLeft.Joint3*PI/180;
	AnglePreviousLeft.Joint4=AngleCurrentLeft.Joint4*PI/180;
	AnglePreviousLeft.Joint5=AngleCurrentLeft.Joint5*PI/180;
	AnglePreviousLeft.Joint6=AngleCurrentLeft.Joint6*PI/180;

	CreateRotationMatrixBaseToWrist_left(PI,-PI/2,0);
	CreateRotationMatrixWristToEnd_left(OrientationDesiredLeft.R,OrientationDesiredLeft.P,OrientationDesiredLeft.W);
	for( i = 0; i < 3; i++)
	{
		for( j = 0; j < 3; j++ ){
			BaseToEndRotationMatrixLeft.Element[i][j] =
BaseToWristRotationMatrixLeft.Element[i][0]*WristToEndRotationMatrixLeft.Element[0][j] +
BaseToWristRotationMatrixLeft.Element[i][1]*WristToEndRotationMatrixLeft.Element[1][j] +
BaseToWristRotationMatrixLeft.Element[i][2]*WristToEndRotationMatrixLeft.Element[2][j];
		}
	}
	BaseToEndRotationMatrixLeft.Element[0][0]=0;
```

```
BaseToEndRotationMatrixLeft.Element[0][1]=0;
BaseToEndRotationMatrixLeft.Element[0][2]=1;
BaseToEndRotationMatrixLeft.Element[1][0]=0;
BaseToEndRotationMatrixLeft.Element[1][1]=1;
BaseToEndRotationMatrixLeft.Element[1][2]=0;
BaseToEndRotationMatrixLeft.Element[2][0]=-1;
BaseToEndRotationMatrixLeft.Element[2][1]=0;
BaseToEndRotationMatrixLeft.Element[2][2]=0;
EndEffectorVect[0] = BaseToEndRotationMatrixLeft.Element[0][2];
EndEffectorVect[1] = BaseToEndRotationMatrixLeft.Element[1][2];
EndEffectorVect[2] = BaseToEndRotationMatrixLeft.Element[2][2];

double px = PositionDesiredLeft.X*1.0 /*- (endEffectorVect[0]*lengthOfEndEff)*/;
double py = PositionDesiredLeft.Y*1.0 /*- (endEffectorVect[1]*lengthOfEndEff)*/;
double pz = PositionDesiredLeft.Z*1.0 /*- (endEffectorVect[2]*lengthOfEndEff)*/;
double dummyAngle;
double dummy1, dummy2; //dummy1;y dummy2:x
double K;

theta[0]=atan2(py,px)+atan2(d[2],-sqrt(px*px+py*py-d[2]*d[2]));


dummy1= (-px*px-py*py-pz*pz+a[2]*a[2]+a[3]*a[3]+d[2]*d[2]+d[3]*d[3])/(2.0*a[2]*d[3]);

dummy2=-sqrt(1-dummy1*dummy1);

theta[2]=atan2(dummy1,dummy2);


if(theta[2]>(PI/2))
{
        theta[2]=-2*PI+theta[2];
}


dummy1=a[2]*(px*cos(theta[0])+py*sin(theta[0]))*cos(theta[2])+pz*d[3]-a[2]*pz*sin(theta[2]);

dummy2=(px*cos(theta[0])+py*sin(theta[0]))*(d[3]-a[2]*sin(theta[2]))-a[2]*pz*cos(theta[2]);
dummyAngle = atan2( dummy1, dummy2 );


theta[1] = dummyAngle - theta[2];

if (theta[1] < (-PI/2.0))           // Warning - this should be under 0
{                                                                       // Fixup reflected solutions.
        theta[1] = 2.0*PI + theta[1];
}


//If s5 != 0 we can solve theta3 as below

dummy1                                                                  =                                                               -
BaseToEndRotationMatrixLeft.Element[0][2]*sin(theta[0])+BaseToEndRotationMatrixLeft.Element[1][2]*cos(theta[0]);
dummy2                  =               -BaseToEndRotationMatrixLeft.Element[0][2]*cos(theta[0])*cos(theta[1]+theta[2])-
BaseToEndRotationMatrixLeft.Element[1][2]*sin(theta[0])*cos(theta[1]+theta[2])+BaseToEndRotationMatrixLeft.Element[2][2]*sin(th
eta[1]+theta[2]);
//if else is for singularity check
if((fabs(dummy1) < 0.001) && (fabs(dummy2) < 0.001))
{
        theta[3] = AnglePreviousLeft.Joint4;
}
else
        theta[3] = atan2(dummy1,dummy2);

if (theta[3] >= (PI/2.0 - 0.0002))
{
  theta[3] = theta[3] - PI;
}
if (theta[3] <= (-PI/2.0 + 0.0002))
```

```
	{
	  theta[3] = theta[3] + PI;
	}


	//So we have theta01, theta11, theta21, theta31 (pair 0)
	//So we have theta01, theta12, theta22, theta32 (pair 1)
	//So we have theta02, theta13, theta21, theta33 (pair 2)
	//So we have theta02, theta14, theta22, theta34 (pair 3)
	//pairs
	dummy1 = -1*( BaseToEndRotationMatrixLeft.Element[0][2]*( cos(theta[0])*cos(theta[1]+theta[2])*cos(theta[3]) + sin(theta[0])*sin(theta[3]) )
		+ BaseToEndRotationMatrixLeft.Element[1][2]*( sin(theta[0])*cos(theta[1]+theta[2])*cos(theta[3]) - cos(theta[0])*sin(theta[3]) )
		- BaseToEndRotationMatrixLeft.Element[2][2]*( sin(theta[1]+theta[2]))*cos(theta[3]) );

	dummy2 = BaseToEndRotationMatrixLeft.Element[0][2]*( -cos(theta[0])*sin(theta[1]+theta[2]) )
		+ BaseToEndRotationMatrixLeft.Element[1][2]*( -sin(theta[0])*sin(theta[1]+theta[2]) )
		+ BaseToEndRotationMatrixLeft.Element[2][2]*( -cos(theta[1]+theta[2]) );

	theta[4] = atan2(dummy1,dummy2);



	dummy1 = -BaseToEndRotationMatrixLeft.Element[0][0]*( cos(theta[0])*cos(theta[1]+theta[2])*sin(theta[3]) - sin(theta[0])*cos(theta[3]) )
		-BaseToEndRotationMatrixLeft.Element[1][0]*( sin(theta[0])*cos(theta[1]+theta[2])*sin(theta[3]) + cos(theta[0])*cos(theta[3]) )
		+BaseToEndRotationMatrixLeft.Element[2][0]*( sin(theta[1]+theta[2])*sin(theta[3]) );

	dummy2 = BaseToEndRotationMatrixLeft.Element[0][0]*( ( cos(theta[0])*cos(theta[1]+theta[2])*cos(theta[3]) + sin(theta[0])*sin(theta[3]) )*cos(theta[4]) - cos(theta[0])*sin(theta[1]+theta[2])*sin(theta[4]) )
		+ BaseToEndRotationMatrixLeft.Element[1][0]*( ( sin(theta[0])*cos(theta[1]+theta[2])*cos(theta[3]) - cos(theta[0])*sin(theta[3]) )*cos(theta[4]) - sin(theta[0])*sin(theta[1]+theta[2])*sin(theta[4]) )
		-BaseToEndRotationMatrixLeft.Element[2][0]*( sin(theta[1]+theta[2])*cos(theta[3])*cos(theta[4]) + cos(theta[1]+theta[2])*sin(theta[4]) );
	theta[5] = atan2(dummy1,dummy2);

	if (theta[5] >= (PI/2.0 - 0.0002))
	{
		theta[5] = theta[5] - PI;
	}
	if (theta[5] <= (-PI/2.0 + 0.0002))
	{
		theta[5] = theta[5] + PI;
	}
	theta[3]=theta[3]+theta[5];
	theta[5]=0;

	AngleDesiredLeft.Joint1=theta[0]*180.0/PI;
	AngleDesiredLeft.Joint2=theta[1]*180.0/PI;
	AngleDesiredLeft.Joint3=theta[2]*180.0/PI;
	AngleDesiredLeft.Joint4=theta[3]*180.0/PI;
	AngleDesiredLeft.Joint5=theta[4]*180.0/PI;
	AngleDesiredLeft.Joint6=theta[5]*180.0/PI;
}

  void CArmControlDlg::InverseKinematicsRight()
{
	double theta[6];
	double EndEffectorVect[3];
	int lengthOfEndEff = 270;
	int i;
	int j=0;

	double a[6];
	double d[6];
	a[0]=ISACParameterRight.A0;
	a[1]=ISACParameterRight.A1;
```

```
	a[2]=ISACParameterRight.A2;
	a[3]=ISACParameterRight.A3;
	a[4]=ISACParameterRight.A4;
	a[5]=ISACParameterRight.A5;

	d[0]=ISACParameterRight.D1;
	d[1]=ISACParameterRight.D2;
	d[2]=ISACParameterRight.D3;
	d[3]=ISACParameterRight.D4;
	d[4]=ISACParameterRight.D5;
	d[5]=ISACParameterRight.D6;

	AnglePreviousRight.Joint1=AngleCurrentRight.Joint1*PI/180;
	AnglePreviousRight.Joint2=AngleCurrentRight.Joint2*PI/180;
	AnglePreviousRight.Joint3=AngleCurrentRight.Joint3*PI/180;
	AnglePreviousRight.Joint4=AngleCurrentRight.Joint4*PI/180;
	AnglePreviousRight.Joint5=AngleCurrentRight.Joint5*PI/180;
	AnglePreviousRight.Joint6=AngleCurrentRight.Joint6*PI/180;


	CreateRotationMatrixBaseToWrist_right(PI,-PI/2,0);
	CreateRotationMatrixWristToEnd_right(OrientationDesiredRight.R,OrientationDesiredRight.P,OrientationDesiredRight.W);
	for( i = 0; i < 3; i++)
	{
		for( j = 0; j < 3; j++ ){
			BaseToEndRotationMatrixRight.Element[i][j]                                                                    =
BaseToWristRotationMatrixRight.Element[i][0]*WristToEndRotationMatrixRight.Element[0][j]                                  +
BaseToWristRotationMatrixRight.Element[i][1]*WristToEndRotationMatrixRight.Element[1][j]                                  +
BaseToWristRotationMatrixRight.Element[i][2]*WristToEndRotationMatrixRight.Element[2][j];
		}
	}
	BaseToEndRotationMatrixLeft.Element[0][0]=0;
	BaseToEndRotationMatrixLeft.Element[0][1]=0;
	BaseToEndRotationMatrixLeft.Element[0][2]=1;
	BaseToEndRotationMatrixLeft.Element[1][0]=1;
	BaseToEndRotationMatrixLeft.Element[1][1]=0;
	BaseToEndRotationMatrixLeft.Element[1][2]=0;
	BaseToEndRotationMatrixLeft.Element[2][0]=0;
	BaseToEndRotationMatrixLeft.Element[2][1]=1;
	BaseToEndRotationMatrixLeft.Element[2][2]=0;

	EndEffectorVect[0] = BaseToEndRotationMatrixRight.Element[0][2];
	EndEffectorVect[1] = BaseToEndRotationMatrixRight.Element[1][2];
	EndEffectorVect[2] = BaseToEndRotationMatrixRight.Element[2][2];

	double px = PositionDesiredRight.X*1.0 /*- (endEffectorVect[0]*lengthOfEndEff)*/;
	double py = PositionDesiredRight.Y*1.0 /*- (endEffectorVect[1]*lengthOfEndEff)*/;
	double pz = PositionDesiredRight.Z*1.0 /*- (endEffectorVect[2]*lengthOfEndEff)*/;
	double dummyAngle;
	double dummy1, dummy2; //dummy1 sin;dymmy2 cos
	double K;

	theta[0]=atan2(py,px)-atan2(-d[2],sqrt(px*px+py*py-d[2]*d[2])); //done

	dummy2= (a[2]*a[2]+d[2]*d[2]+d[3]*d[3]-px*px-py*py-pz*pz)/(2.0*a[2]*d[3]);

	theta[2]=asin(dummy2);

	if(theta[2]>(PI/2))
	{
		theta[2]=-2*PI+theta[2];
	}


	dummy1=a[2]*(px*cos(theta[0])+py*sin(theta[0]))*cos(theta[2])+pz*d[3]-a[2]*pz*sin(theta[2]);

	dummy2=(px*cos(theta[0])+py*sin(theta[0]))*(d[3]-a[2]*sin(theta[2]))-a[2]*pz*cos(theta[2]);
	dummyAngle = atan2( dummy1, dummy2 );
```

```
	theta[1] = dummyAngle - theta[2];

	if (theta[1] < (-PI/2.0))		// Warning - this should be under 0
	{							// Fixup reflected solutions.
		theta[1] = 2.0*PI + theta[1];
	}


	//If s5 != 0 we can solve theta3 as below

	dummy1 = BaseToEndRotationMatrixRight.Element[0][2]*sin(theta[0])-
BaseToEndRotationMatrixRight.Element[1][2]*cos(theta[0]);
	dummy2 = -BaseToEndRotationMatrixRight.Element[0][2]*cos(theta[0])*sin(theta[1]+theta[2])-
BaseToEndRotationMatrixRight.Element[1][2]*sin(theta[0])*sin(theta[1]+theta[2])+BaseToEndRotationMatrixRight.Element[2][2]*cos(
theta[1]+theta[2]);
	//if else is for singularity check
	if((fabs(dummy1) < 0.001) && (fabs(dummy2) < 0.001))
	{
		theta[3] = AnglePreviousRight.Joint4;
	}
	else
		theta[3] = atan2(dummy1,dummy2);

	if (theta[3] >= (PI/2.0 - 0.0002))
	{
	  theta[3] = theta[3] - PI;
	}
	if (theta[3] <= (-PI/2.0 + 0.0002))
	{
	  theta[3] = theta[3] + PI;
	}


	//So we have theta01, theta11, theta21, theta31 (pair 0)
	//So we have theta01, theta12, theta22, theta32 (pair 1)
	//So we have theta02, theta13, theta21, theta33 (pair 2)
	//So we have theta02, theta14, theta22, theta34 (pair 3)
	//pairs
	dummy1 = -1*BaseToEndRotationMatrixRight.Element[0][2]* cos(theta[0])*sin(theta[1]+theta[2])*cos(theta[3]) +
sin(theta[0])*sin(theta[3])*BaseToEndRotationMatrixRight.Element[0][2]
		-1*BaseToEndRotationMatrixRight.Element[1][2]* sin(theta[0])*sin(theta[1]+theta[2])*cos(theta[3]) -
cos(theta[0])*sin(theta[3])*BaseToEndRotationMatrixRight.Element[1][2]
		+ BaseToEndRotationMatrixRight.Element[2][2]* cos(theta[1]+theta[2])*cos(theta[3]);

	dummy2 = BaseToEndRotationMatrixRight.Element[0][2]*cos(theta[0])*cos(theta[1]+theta[2])
		+ BaseToEndRotationMatrixRight.Element[1][2]*sin(theta[0])*cos(theta[1]+theta[2])
		+ BaseToEndRotationMatrixRight.Element[2][2]*sin(theta[1]+theta[2]);

	theta[4] = atan2(dummy1,dummy2);



	dummy1 = BaseToEndRotationMatrixRight.Element[2][0]*cos(theta[0])*sin(theta[1]+theta[2])*sin(theta[3])
		+BaseToEndRotationMatrixRight.Element[1][0]*sin(theta[0])*sin(theta[1]+theta[2])*sin(theta[3])
		-BaseToEndRotationMatrixRight.Element[2][0]*cos(theta[1]+theta[2])*sin(theta[3])
		-BaseToEndRotationMatrixRight.Element[1][0]*cos(theta[0])*cos(theta[3])
		+BaseToEndRotationMatrixRight.Element[0][0]*sin(theta[0])*cos(theta[3]);

	dummy2 = BaseToEndRotationMatrixRight.Element[0][1]*cos(theta[0])*sin(theta[1]+theta[2])*sin(theta[3])
		+BaseToEndRotationMatrixRight.Element[1][1]*sin(theta[0])*sin(theta[1]+theta[2])*sin(theta[3])
		-BaseToEndRotationMatrixRight.Element[2][1]*cos(theta[1]+theta[2])*sin(theta[3])
		-BaseToEndRotationMatrixRight.Element[1][1]*cos(theta[0])*cos(theta[3])
		+BaseToEndRotationMatrixRight.Element[0][1]*sin(theta[0])*cos(theta[3]);
	theta[5] = atan2(dummy1,-dummy2);

	if (theta[5] >= (PI/2.0 - 0.0002))
	{
		theta[5] = theta[5] - 2*PI;
```

```
	}
	if (theta[5] <= (-PI/2.0 + 0.0002))
	{
		theta[5] = theta[5] + 2*PI;
	}
	theta[3]=theta[3]+theta[5];
	theta[5]=0;

	AngleDesiredRight.Joint1=theta[0]*180.0/PI;
	AngleDesiredRight.Joint2=theta[1]*180.0/PI;
	AngleDesiredRight.Joint3=theta[2]*180.0/PI;
	AngleDesiredRight.Joint4=theta[3]*180.0/PI;
	AngleDesiredRight.Joint5=theta[4]*180.0/PI;
	AngleDesiredRight.Joint6=theta[5]*180.0/PI;

}

void CArmControlDlg::LeftPID()
{
	ReadLeftEncoders();
	ComputeLeftAngles();
	AngleErrorCurrentLeft.Joint1 = AngleDesiredLeft.Joint1-AngleCurrentLeft.Joint1;
	AngleErrorCurrentLeft.Joint2 = AngleDesiredLeft.Joint2-AngleCurrentLeft.Joint2;
	AngleErrorCurrentLeft.Joint3 = AngleDesiredLeft.Joint3-AngleCurrentLeft.Joint3;
	AngleErrorCurrentLeft.Joint4 = AngleDesiredLeft.Joint4-AngleCurrentLeft.Joint4;
	AngleErrorCurrentLeft.Joint5 = AngleDesiredLeft.Joint5-AngleCurrentLeft.Joint5;
	AngleErrorCurrentLeft.Joint6 = AngleDesiredLeft.Joint6-AngleCurrentLeft.Joint6;

	PressureCurrentLeft.Muscle0=PressureLeft.Muscle0-(ParameterProportionLeft.Muscle0*AngleErrorCurrentLeft.Joint1);
	PressureCurrentLeft.Muscle1=PressureLeft.Muscle1+(ParameterProportionLeft.Muscle1*AngleErrorCurrentLeft.Joint1);

	PressureCurrentLeft.Muscle2=PressureLeft.Muscle2+(ParameterProportionLeft.Muscle2*AngleErrorCurrentLeft.Joint2);
	PressureCurrentLeft.Muscle3=PressureLeft.Muscle3-(ParameterProportionLeft.Muscle3*AngleErrorCurrentLeft.Joint2);

	PressureCurrentLeft.Muscle4=0;
	PressureCurrentLeft.Muscle5=PressureLeft.Muscle5-(ParameterProportionLeft.Muscle5*AngleErrorCurrentLeft.Joint3);
	PressureCurrentLeft.Muscle6=PressureLeft.Muscle6-(ParameterProportionLeft.Muscle6*AngleErrorCurrentLeft.Joint3);
	PressureCurrentLeft.Muscle7=0;

	PressureCurrentLeft.Muscle4=0;
	PressureCurrentLeft.Muscle5=PressureLeft.Muscle5+(ParameterProportionLeft.Muscle5*AngleErrorCurrentLeft.Joint4);
	PressureCurrentLeft.Muscle6=PressureLeft.Muscle6-(ParameterProportionLeft.Muscle6*AngleErrorCurrentLeft.Joint4);
	PressureCurrentLeft.Muscle7=0;

	PressureCurrentLeft.Muscle8=PressureLeft.Muscle8+(ParameterProportionLeft.Muscle8*AngleErrorCurrentLeft.Joint5);
	PressureCurrentLeft.Muscle9=PressureLeft.Muscle9-(ParameterProportionLeft.Muscle9*AngleErrorCurrentLeft.Joint5);
	PressureCurrentLeft.Muscle10=PressureLeft.Muscle10+(ParameterProportionLeft.Muscle10*AngleErrorCurrentLeft.Joint5);
	PressureCurrentLeft.Muscle11=PressureLeft.Muscle11-(ParameterProportionLeft.Muscle11*AngleErrorCurrentLeft.Joint5);

	PressureCurrentLeft.Muscle8=PressureLeft.Muscle8-(ParameterProportionLeft.Muscle8*AngleErrorCurrentLeft.Joint6);
	PressureCurrentLeft.Muscle9=PressureLeft.Muscle9-(ParameterProportionLeft.Muscle9*AngleErrorCurrentLeft.Joint6);
	PressureCurrentLeft.Muscle10=PressureLeft.Muscle10+(ParameterProportionLeft.Muscle10*AngleErrorCurrentLeft.Joint6);
	PressureCurrentLeft.Muscle11=PressureLeft.Muscle11+(ParameterProportionLeft.Muscle11*AngleErrorCurrentLeft.Joint6);

	if(PressureCurrentLeft.Muscle0<0)
	{
		PressureCurrentLeft.Muscle0=0;
	}
	if(PressureCurrentLeft.Muscle0>3.6)
	{
		PressureCurrentLeft.Muscle0=3.6;
	}
	if(PressureCurrentLeft.Muscle1<0)
	{
		PressureCurrentLeft.Muscle1=0;
	}
	if(PressureCurrentLeft.Muscle1>3.6)
	{
		PressureCurrentLeft.Muscle1=3.6;
```

```
}
if(PressureCurrentLeft.Muscle2<0)
{
	PressureCurrentLeft.Muscle2=0;
}
if(PressureCurrentLeft.Muscle2>3.6)
{
	PressureCurrentLeft.Muscle2=3.6;
}
if(PressureCurrentLeft.Muscle3<0)
{
	PressureCurrentLeft.Muscle3=0;
}
if(PressureCurrentLeft.Muscle3>3.6)
{
	PressureCurrentLeft.Muscle3=3.6;
}
if(PressureCurrentLeft.Muscle4<0)
{
	PressureCurrentLeft.Muscle4=0;
}
if(PressureCurrentLeft.Muscle4>3.6)
{
	PressureCurrentLeft.Muscle4=3.6;
}
if(PressureCurrentLeft.Muscle5<0)
{
	PressureCurrentLeft.Muscle5=0;
}
if(PressureCurrentLeft.Muscle5>3.6)
{
	PressureCurrentLeft.Muscle5=3.6;
}
if(PressureCurrentLeft.Muscle6<0)
{
	PressureCurrentLeft.Muscle6=0;
}
if(PressureCurrentLeft.Muscle6>3.6)
{
	PressureCurrentLeft.Muscle6=3.6;
}
if(PressureCurrentLeft.Muscle7<0)
{
	PressureCurrentLeft.Muscle7=0;
}
if(PressureCurrentLeft.Muscle7>3.6)
{
	PressureCurrentLeft.Muscle7=3.6;
}
if(PressureCurrentLeft.Muscle8<0)
{
	PressureCurrentLeft.Muscle8=0;
}
if(PressureCurrentLeft.Muscle8>3.6)
{
	PressureCurrentLeft.Muscle8=3.6;
}
if(PressureCurrentLeft.Muscle9<0)
{
	PressureCurrentLeft.Muscle9=0;
}
if(PressureCurrentLeft.Muscle9>3.6)
{
	PressureCurrentLeft.Muscle9=3.6;
}
if(PressureCurrentLeft.Muscle10<0)
{
	PressureCurrentLeft.Muscle10=0;
}
```

```
	if(PressureCurrentLeft.Muscle10>3.6)
	{
		PressureCurrentLeft.Muscle10=3.6;
	}
	if(PressureCurrentLeft.Muscle11<0)
	{
		PressureCurrentLeft.Muscle11=0;
	}
	if(PressureCurrentLeft.Muscle11>3.6)
	{
		PressureCurrentLeft.Muscle11=3.6;
	}
}

void CArmControlDlg::RightPID()
{
	ReadRightEncoders();
	ComputeRightAngles();
	AngleErrorCurrentRight.Joint1 = AngleDesiredRight.Joint1-AngleCurrentRight.Joint1;
	AngleErrorCurrentRight.Joint2 = AngleDesiredRight.Joint2-AngleCurrentRight.Joint2;
	AngleErrorCurrentRight.Joint3 = AngleDesiredRight.Joint3-AngleCurrentRight.Joint3;
	AngleErrorCurrentRight.Joint4 = AngleDesiredRight.Joint4-AngleCurrentRight.Joint4;
	AngleErrorCurrentRight.Joint5 = AngleDesiredRight.Joint5-AngleCurrentRight.Joint5;
	AngleErrorCurrentRight.Joint6 = AngleDesiredRight.Joint6-AngleCurrentRight.Joint6;

	PressureCurrentRight.Muscle0=PressureCurrentRight.Muscle0-
(ParameterProportionRight.Muscle0*AngleErrorCurrentRight.Joint1);
	PressureCurrentRight.Muscle1=PressureCurrentRight.Muscle1+(ParameterProportionRight.Muscle1*AngleErrorCurrentRight.Jo
int1);

	PressureCurrentRight.Muscle2=PressureCurrentRight.Muscle2-
(ParameterProportionRight.Muscle2*AngleErrorCurrentRight.Joint2);
	PressureCurrentRight.Muscle3=PressureCurrentRight.Muscle3+(ParameterProportionRight.Muscle3*AngleErrorCurrentRight.Jo
int2);

	PressureCurrentRight.Muscle4=0;
	PressureCurrentRight.Muscle5=PressureCurrentRight.Muscle5+(ParameterProportionRight.Muscle5*AngleErrorCurrentRight.Jo
int3);
	PressureCurrentRight.Muscle6=PressureCurrentRight.Muscle6+(ParameterProportionRight.Muscle6*AngleErrorCurrentRight.Jo
int3);
	PressureCurrentRight.Muscle7=0;

/*	PressureCurrentRight.Muscle4=0;
	PressureCurrentRight.Muscle5=PressureCurrentRight.Muscle5+(ParameterProportionRight.Muscle5*AngleErrorCurrentRight.Jo
int4);
	PressureCurrentRight.Muscle6=PressureCurrentRight.Muscle6-
(ParameterProportionRight.Muscle6*AngleErrorCurrentRight.Joint4);
	PressureCurrentRight.Muscle7=0;
*/
/*
	PressureCurrentRight.Muscle8=PressureCurrentRight.Muscle8+(ParameterProportionRight.Muscle8*AngleErrorCurrentRight.Jo
int5);
	PressureCurrentRight.Muscle9=PressureCurrentRight.Muscle9-
(ParameterProportionRight.Muscle9*AngleErrorCurrentRight.Joint5);
	PressureCurrentRight.Muscle10=PressureCurrentRight.Muscle10+(ParameterProportionRight.Muscle10*AngleErrorCurrentRig
ht.Joint5);
	PressureCurrentRight.Muscle11=PressureCurrentRight.Muscle11-
(ParameterProportionRight.Muscle11*AngleErrorCurrentRight.Joint5);

	PressureCurrentRight.Muscle8=PressureCurrentRight.Muscle8-
(ParameterProportionRight.Muscle8*AngleErrorCurrentRight.Joint6);
	PressureCurrentRight.Muscle9=PressureCurrentRight.Muscle9-
(ParameterProportionRight.Muscle9*AngleErrorCurrentRight.Joint6);
	PressureCurrentRight.Muscle10=PressureCurrentRight.Muscle10+(ParameterProportionRight.Muscle10*AngleErrorCurrentRig
ht.Joint6);
	PressureCurrentRight.Muscle11=PressureCurrentRight.Muscle11+(ParameterProportionRight.Muscle11*AngleErrorCurrentRig
ht.Joint6);
*/
	if(PressureCurrentRight.Muscle0<0)
```

```
	{
		PressureCurrentRight.Muscle0=0;
	}
	if(PressureCurrentRight.Muscle0>3.6)
	{
		PressureCurrentRight.Muscle0=3.6;
	}

	if(PressureCurrentRight.Muscle1<0)
	{
		PressureCurrentRight.Muscle1=0;
	}
	if(PressureCurrentRight.Muscle1>3.6)
	{
		PressureCurrentRight.Muscle1=3.6;
	}

	if(PressureCurrentRight.Muscle2<0)
	{
		PressureCurrentRight.Muscle2=0;
	}
	if(PressureCurrentRight.Muscle2>3.6)
	{
		PressureCurrentRight.Muscle2=3.6;
	}

	if(PressureCurrentRight.Muscle3<0)
	{
		PressureCurrentRight.Muscle3=0;
	}
	if(PressureCurrentRight.Muscle3>3.6)
	{
		PressureCurrentRight.Muscle3=3.6;
	}

//	if(PressureCurrentRight.Muscle4<0)
//	{
//		PressureCurrentRight.Muscle4=0;
//	}
//	if(PressureCurrentRight.Muscle4>3.6)
//	{
//		PressureCurrentRight.Muscle4=3.6;
//	}
	if(PressureCurrentRight.Muscle5<0)
	{
		PressureCurrentRight.Muscle5=0;
	}
	if(PressureCurrentRight.Muscle5>3.6)
	{
		PressureCurrentRight.Muscle5=3.6;
	}

	if(PressureCurrentRight.Muscle6<0)
	{
		PressureCurrentRight.Muscle6=0;
	}
	if(PressureCurrentRight.Muscle6>3.6)
	{
		PressureCurrentRight.Muscle6=3.6;
	}

//	if(PressureCurrentRight.Muscle7<0)
//	{
//		PressureCurrentRight.Muscle7=0;
//	}
//	if(PressureCurrentRight.Muscle7>3.6)
//	{
//		PressureCurrentRight.Muscle7=3.6;
//	}
```

```
/*	if(PressureCurrentRight.Muscle8<0)
	{
		PressureCurrentRight.Muscle8=0;
	}
	if(PressureCurrentRight.Muscle8>3.6)
	{
		PressureCurrentRight.Muscle8=3.6;
	}
	if(PressureCurrentRight.Muscle9<0)
	{
		PressureCurrentRight.Muscle9=0;
	}
	if(PressureCurrentRight.Muscle9>3.6)
	{
		PressureCurrentRight.Muscle9=3.6;
	}
	if(PressureCurrentRight.Muscle10<0)
	{
		PressureCurrentRight.Muscle10=0;
	}
	if(PressureCurrentRight.Muscle10>3.6)
	{
		PressureCurrentRight.Muscle10=3.6;
	}
	if(PressureCurrentRight.Muscle11<0)
	{
		PressureCurrentRight.Muscle11=0;
	}
	if(PressureCurrentRight.Muscle11>3.6)
	{
		PressureCurrentRight.Muscle11=3.6;
	}
*/
}

void CArmControlDlg::PressureOutputLeft()
{
	PressureLeft.Muscle0=PressureCurrentLeft.Muscle0;
	PressureLeft.Muscle1=PressureCurrentLeft.Muscle1;
	PressureLeft.Muscle2=PressureCurrentLeft.Muscle2;
	PressureLeft.Muscle3=PressureCurrentLeft.Muscle3;
	PressureLeft.Muscle4=PressureCurrentLeft.Muscle4;
	PressureLeft.Muscle5=PressureCurrentLeft.Muscle5;
	PressureLeft.Muscle6=PressureCurrentLeft.Muscle6;
	PressureLeft.Muscle7=PressureCurrentLeft.Muscle7;
	PressureLeft.Muscle8=PressureCurrentLeft.Muscle8;
	PressureLeft.Muscle9=PressureCurrentLeft.Muscle9;
	PressureLeft.Muscle10=PressureCurrentLeft.Muscle10;
	PressureLeft.Muscle11=PressureCurrentLeft.Muscle11;

	if(PressureLeft.Muscle0<0)
	{
		PressureLeft.Muscle0=0;
	}
	if(PressureLeft.Muscle0>3.6)
	{
		PressureLeft.Muscle0=3.6;
	}
	if(PressureLeft.Muscle1<0)
	{
		PressureLeft.Muscle1=0;
	}
	if(PressureLeft.Muscle1>3.6)
	{
		PressureLeft.Muscle1=3.6;
	}
	if(PressureLeft.Muscle2<0)
	{
		PressureLeft.Muscle2=0;
```

```
}
if(PressureLeft.Muscle2>3.6)
{
	PressureLeft.Muscle2=3.6;
}
if(PressureLeft.Muscle3<0)
{
	PressureLeft.Muscle3=0;
}
if(PressureLeft.Muscle3>3.6)
{
	PressureLeft.Muscle3=3.6;
}
if(PressureLeft.Muscle4<0)
{
	PressureLeft.Muscle4=0;
}
if(PressureLeft.Muscle4>3.6)
{
	PressureLeft.Muscle4=3.6;
}
if(PressureLeft.Muscle5<0)
{
	PressureLeft.Muscle5=0;
}
if(PressureLeft.Muscle5>3.6)
{
	PressureLeft.Muscle5=3.6;
}
if(PressureLeft.Muscle6<0)
{
	PressureLeft.Muscle6=0;
}
if(PressureLeft.Muscle6>3.6)
{
	PressureLeft.Muscle6=3.6;
}
if(PressureLeft.Muscle7<0)
{
	PressureLeft.Muscle7=0;
}
if(PressureLeft.Muscle7>3.6)
{
	PressureLeft.Muscle7=3.6;
}
if(PressureLeft.Muscle8<0)
{
	PressureLeft.Muscle8=0;
}
if(PressureLeft.Muscle8>3.6)
{
	PressureLeft.Muscle8=3.6;
}
if(PressureLeft.Muscle9<0)
{
	PressureLeft.Muscle9=0;
}
if(PressureLeft.Muscle9>3.6)
{
	PressureLeft.Muscle9=3.6;
}
if(PressureLeft.Muscle10<0)
{
	PressureLeft.Muscle10=0;
}
if(PressureLeft.Muscle10>3.6)
{
	PressureLeft.Muscle10=3.6;
}
```

```
	if(PressureLeft.Muscle11<0)
	{
		PressureLeft.Muscle11=0;
	}
	if(PressureLeft.Muscle11>3.6)
	{
		PressureLeft.Muscle11=3.6;
	}
	vitalSelectBoard(0);
	vitalDacWrite( 0, PressureLeft.Muscle0);
	vitalDacWrite( 1, PressureLeft.Muscle1);
	vitalDacWrite( 2, PressureLeft.Muscle2);
	vitalDacWrite( 3, PressureLeft.Muscle3);
	vitalDacWrite( 4, PressureLeft.Muscle4);
	vitalDacWrite( 5, PressureLeft.Muscle5);
	vitalDacWrite( 6, PressureLeft.Muscle6);
	vitalDacWrite( 7, PressureLeft.Muscle7);
	vitalSelectBoard(1);
	vitalDacWrite( 8, PressureLeft.Muscle8);
	vitalDacWrite( 9, PressureLeft.Muscle9);
	vitalDacWrite( 10, PressureLeft.Muscle10);
	vitalDacWrite( 11, PressureLeft.Muscle11);
}

void CArmControlDlg::PressureOutputRight()
{
	PressureRight.Muscle0=PressureCurrentRight.Muscle0;
	PressureRight.Muscle1=PressureCurrentRight.Muscle1;
	PressureRight.Muscle2=PressureCurrentRight.Muscle2;
	PressureRight.Muscle3=PressureCurrentRight.Muscle3;
	PressureRight.Muscle4=PressureCurrentRight.Muscle4;
	PressureRight.Muscle5=PressureCurrentRight.Muscle5;
	PressureRight.Muscle6=PressureCurrentRight.Muscle6;
//	PressureRight.Muscle7=PressureCurrentRight.Muscle7;
	PressureRight.Muscle8=PressureCurrentRight.Muscle8;
	PressureRight.Muscle9=PressureCurrentRight.Muscle9;
	PressureRight.Muscle10=PressureCurrentRight.Muscle10;
	PressureRight.Muscle11=PressureCurrentRight.Muscle11;

	if(PressureRight.Muscle0<0)
	{
		PressureRight.Muscle0=0;
	}
	if(PressureRight.Muscle0>3.6)
	{
		PressureRight.Muscle0=3.6;
	}
	if(PressureRight.Muscle1<0)
	{
		PressureRight.Muscle1=0;
	}
	if(PressureRight.Muscle1>3.6)
	{
		PressureRight.Muscle1=3.6;
	}
	if(PressureRight.Muscle2<0)
	{
		PressureRight.Muscle2=0;
	}
	if(PressureRight.Muscle2>3.6)
	{
		PressureRight.Muscle2=3.6;
	}
	if(PressureRight.Muscle3<0)
	{
		PressureRight.Muscle3=0;
	}
	if(PressureRight.Muscle3>3.6)
	{
```

```
                PressureRight.Muscle3=3.6;
        }
//      if(PressureRight.Muscle4<0)
//      {
//              PressureRight.Muscle4=0;
//      }
//      if(PressureRight.Muscle4>3.6)
//      {
//              PressureRight.Muscle4=3.6;
//      }
        if(PressureRight.Muscle5<0)
        {
                PressureRight.Muscle5=0;
        }
        if(PressureRight.Muscle5>3.6)
        {
                PressureRight.Muscle5=3.6;
        }
        if(PressureRight.Muscle6<0)
        {
                PressureRight.Muscle6=0;
        }
        if(PressureRight.Muscle6>3.6)
        {
                PressureRight.Muscle6=3.6;
        }
//      if(PressureRight.Muscle7<0)
//      {
//              PressureRight.Muscle7=0;
//      }
//      if(PressureRight.Muscle7>3.6)
//      {
//              PressureRight.Muscle7=3.6;
//      }
/*      if(PressureRight.Muscle8<0)
        {
                PressureRight.Muscle8=0;
        }
        if(PressureRight.Muscle8>3.6)
        {
                PressureRight.Muscle8=3.6;
        }
        if(PressureRight.Muscle9<0)
        {
                PressureRight.Muscle9=0;
        }
        if(PressureRight.Muscle9>3.6)
        {
                PressureRight.Muscle9=3.6;
        }
        if(PressureRight.Muscle10<0)
        {
                PressureRight.Muscle10=0;
        }
        if(PressureRight.Muscle10>3.6)
        {
                PressureRight.Muscle10=3.6;
        }
        if(PressureRight.Muscle11<0)
        {
                PressureRight.Muscle11=0;
        }
        if(PressureRight.Muscle11>3.6)
        {
                PressureRight.Muscle11=3.6;
        }*/
        vitalSelectBoard(1);
        vitalDacWrite( 4, PressureRight.Muscle0);
        vitalDacWrite( 5, PressureRight.Muscle1);
```

```
	vitalDacWrite( 6, PressureRight.Muscle2);
	vitalDacWrite( 7, PressureRight.Muscle3);
	vitalSelectBoard(2);
	vitalDacWrite( 0, 0);
	vitalDacWrite( 1, PressureRight.Muscle5);
	vitalDacWrite( 2, PressureRight.Muscle6);
//	vitalDacWrite( 3, 0);
	vitalDacWrite( 4, PressureRight.Muscle8);
	vitalDacWrite( 5, PressureRight.Muscle9);
	vitalDacWrite( 6, PressureRight.Muscle10);
	vitalDacWrite( 7, PressureRight.Muscle11);
}

void CArmControlDlg::OnStart()
{
	// TODO: Add your control notification handler code here
	CString tb;

	GetDlgItemText(IDC_LeftDesiredX,tb);
	PositionDesiredLeft.X=atoi(tb);
	GetDlgItemText(IDC_LeftDesiredY,tb);
	PositionDesiredLeft.Y=atoi(tb);
	GetDlgItemText(IDC_LeftDesiredZ,tb);
	PositionDesiredLeft.Z=atoi(tb);

	GetDlgItemText(IDC_LeftDesiredR,tb);
	OrientationDesiredLeft.R=atoi(tb);
	GetDlgItemText(IDC_LeftDesiredP,tb);
	OrientationDesiredLeft.P=atoi(tb);
	GetDlgItemText(IDC_LeftDesiredW,tb);
	OrientationDesiredLeft.W=atoi(tb);

	GetDlgItemText(IDC_RightDesiredX,tb);
	PositionDesiredRight.X=atoi(tb);
	GetDlgItemText(IDC_RightDesiredY,tb);
	PositionDesiredRight.Y=atoi(tb);
	GetDlgItemText(IDC_RightDesiredZ,tb);
	PositionDesiredRight.Z=atoi(tb);

	GetDlgItemText(IDC_RightDesiredR,tb);
	OrientationDesiredRight.R=atoi(tb);
	GetDlgItemText(IDC_RightDesiredP,tb);
	OrientationDesiredRight.P=atoi(tb);
	GetDlgItemText(IDC_RightDesiredW,tb);
	OrientationDesiredRight.W=atoi(tb);

	InverseKinematicsRight();
/*	SetDlgItemInt(IDC_LeftDesiredJoint1,(int)AngleDesiredLeft.Joint1);
	SetDlgItemInt(IDC_LeftDesiredJoint2,(int)AngleDesiredLeft.Joint2);
	SetDlgItemInt(IDC_LeftDesiredJoint3,(int)AngleDesiredLeft.Joint3);
	SetDlgItemInt(IDC_LeftDesiredJoint4,(int)AngleDesiredLeft.Joint4);
	SetDlgItemInt(IDC_LeftDesiredJoint5,(int)AngleDesiredLeft.Joint5);
	SetDlgItemInt(IDC_LeftDesiredJoint6,(int)AngleDesiredLeft.Joint6);*/
	SetDlgItemInt(IDC_RightDesiredJoint1,(int)AngleDesiredRight.Joint1);
	SetDlgItemInt(IDC_RightDesiredJoint2,(int)AngleDesiredRight.Joint2);
	SetDlgItemInt(IDC_RightDesiredJoint3,(int)AngleDesiredRight.Joint3);
	SetDlgItemInt(IDC_RightDesiredJoint4,(int)AngleDesiredRight.Joint4);
	SetDlgItemInt(IDC_RightDesiredJoint5,(int)AngleDesiredRight.Joint5);
	SetDlgItemInt(IDC_RightDesiredJoint6,(int)AngleDesiredRight.Joint6);
	int step=0;
	for(step=0;step<75;step++)
	{
/*		LeftPID();*/
		RightPID();
/*		PressureOutputLeft();*/
		PressureOutputRight();
		Sleep(50);
		vitalSelectBoard(1);
		vitalEncoderResetIndex(2);
```

```
			vitalEncoderResetIndex(3);
			vitalSelectBoard(2);
			vitalEncoderResetIndex(1);
			Sleep(50);
		}
}


void CArmControlDlg::OnAccept()
{
		ServerSocket.Accept(ConnectSocket);
}

void CArmControlDlg::OnReceive()
{
		char pBuf[512];
		char command;
		char sendbuf[512];
		int iBufSize=1024;
		int iRcvd;
		int i;
		short angle;
		short position_received[3];
		int step=0;
		ConnectSocket.Receive(pBuf,512);
		char* pBufp;
		pBufp=pBuf;

		memcpy(&command,pBufp,sizeof(char));
		pBufp+=sizeof(char);
		switch(command)
		{
		case 2:
			memcpy(&position_received[0],pBufp,2*sizeof(char));
			pBufp+=sizeof(unsigned short);

			memcpy(&position_received[1],pBufp,2*sizeof(char));
			pBufp+=sizeof(unsigned short);

			memcpy(&position_received[2],pBufp,2*sizeof(char));

			PositionDesiredRight.X=position_received[0];
			PositionDesiredRight.Y=position_received[1];
			PositionDesiredRight.Z=position_received[2];
			SetDlgItemInt(IDC_RightDesiredX,PositionDesiredRight.X);
			SetDlgItemInt(IDC_RightDesiredY,PositionDesiredRight.Y);
			SetDlgItemInt(IDC_RightDesiredZ,PositionDesiredRight.Z);
			SetDlgItemInt(IDC_RightCurrentX,PositionDesiredRight.X);
			SetDlgItemInt(IDC_RightCurrentY,PositionDesiredRight.Y);
			SetDlgItemInt(IDC_RightCurrentZ,PositionDesiredRight.Z);

			InverseKinematicsRight();
			Sleep(500);
		/*	SetDlgItemInt(IDC_LeftDesiredJoint1,(int)AngleDesiredLeft.Joint1);
			SetDlgItemInt(IDC_LeftDesiredJoint2,(int)AngleDesiredLeft.Joint2);
			SetDlgItemInt(IDC_LeftDesiredJoint3,(int)AngleDesiredLeft.Joint3);
			SetDlgItemInt(IDC_LeftDesiredJoint4,(int)AngleDesiredLeft.Joint4);
			SetDlgItemInt(IDC_LeftDesiredJoint5,(int)AngleDesiredLeft.Joint5);
			SetDlgItemInt(IDC_LeftDesiredJoint6,(int)AngleDesiredLeft.Joint6);*/
			SetDlgItemInt(IDC_RightDesiredJoint1,(int)AngleDesiredRight.Joint1);
			SetDlgItemInt(IDC_RightDesiredJoint2,(int)AngleDesiredRight.Joint2);
			SetDlgItemInt(IDC_RightDesiredJoint3,(int)AngleDesiredRight.Joint3);
			SetDlgItemInt(IDC_RightDesiredJoint4,(int)AngleDesiredRight.Joint4);
			SetDlgItemInt(IDC_RightDesiredJoint5,(int)AngleDesiredRight.Joint5);
			SetDlgItemInt(IDC_RightDesiredJoint6,(int)AngleDesiredRight.Joint6);
			Sleep(500);
			for(step=0;step<75;step++)
			{
		//		LeftPID();
```

```
                        RightPID();
        //              PressureOutputLeft();
                        PressureOutputRight();
                        Sleep(50);
                        vitalSelectBoard(1);
                        vitalEncoderResetIndex(2);
                        vitalEncoderResetIndex(3);
                        vitalSelectBoard(2);
                        vitalEncoderResetIndex(1);
                        Sleep(50);
                }
                break;
        case 3:
                memcpy(&angle,pBufp,2*sizeof(char));
                PressureRight.Muscle8=1.8-(1.8)*angle/90+1.8;//3.6
                PressureRight.Muscle9=1.8+(1.8)*angle/90-1.4;//0.4
                PressureRight.Muscle10=1.8-(1.8)*angle/90-1.4;//0.4
                PressureRight.Muscle11=1.8+(1.8)*angle/90+1.8;//0.4

                if(PressureRight.Muscle8<0)
                {
                        PressureRight.Muscle8=0;
                }
                if(PressureRight.Muscle8>3.6)
                {
                        PressureRight.Muscle8=3.6;
                }
                if(PressureRight.Muscle9<0)
                {
                        PressureRight.Muscle9=0;
                }
                if(PressureRight.Muscle9>3.6)
                {
                        PressureRight.Muscle9=3.6;
                }
                if(PressureRight.Muscle10<0)
                {
                        PressureRight.Muscle10=0;
                }
                if(PressureRight.Muscle10>3.6)
                {
                        PressureRight.Muscle10=3.6;
                }
                if(PressureRight.Muscle11<0)
                {
                        PressureRight.Muscle11=0;
                }
                if(PressureRight.Muscle11>3.6)
                {
                        PressureRight.Muscle11=3.6;
                }
                vitalSelectBoard(2);
                vitalDacWrite( 4, PressureRight.Muscle8);
                vitalDacWrite( 5, PressureRight.Muscle9);
                vitalDacWrite( 6, PressureRight.Muscle10);
                vitalDacWrite( 7, PressureRight.Muscle11);
                Sleep(50);
                break;
        case 4:
                OnGripperRight();
                Sleep(50);
                break;
        default:
                break;
        }
}

void CArmControlDlg::OnEnd()
{
```

```
	// TODO: Add your control notification handler code here
	int i=0;
	int j=0;
	double BufferVoltage;
	BufferVoltage=0.1;

	for(i=0;i<20;i++) // 36x100 ms = 3.6 sec makes the muscles blow up
	{
/*		vitalSelectBoard(0);
		PressureLeft.Muscle0 = PressureLeft.Muscle0-BufferVoltage;
		vitalDacWrite( 0, PressureLeft.Muscle0);
		Sleep(10);
*/		vitalSelectBoard(1);
		PressureRight.Muscle1 = PressureRight.Muscle1-BufferVoltage;
		vitalDacWrite( 5, PressureRight.Muscle1);
		BufferVoltage += 0.05;
		Sleep(200);
	}
	BufferVoltage=0.1;
	for(i=0;i<72;i++)
	{
/*		vitalSelectBoard(0);
		if( PressureLeft.Muscle0-BufferVoltage >0 )
		{
			vitalDacWrite( 0, PressureLeft.Muscle0-BufferVoltage);
		}
		if( PressureLeft.Muscle1-BufferVoltage >0 )
		{
			vitalDacWrite( 1, PressureLeft.Muscle1-BufferVoltage);
		}
		if( PressureLeft.Muscle2-BufferVoltage >0 )
		{
			vitalDacWrite( 2, PressureLeft.Muscle2-BufferVoltage);
		}
		if( PressureLeft.Muscle3-BufferVoltage >0 )
		{
			vitalDacWrite( 3, PressureLeft.Muscle3-BufferVoltage);
		}
		if( PressureLeft.Muscle4-BufferVoltage >0 )
		{
			vitalDacWrite( 4, PressureLeft.Muscle4-BufferVoltage);
		}
		if( PressureLeft.Muscle5-BufferVoltage >0 )
		{
			vitalDacWrite( 5, PressureLeft.Muscle5-BufferVoltage);
		}
		if( PressureLeft.Muscle6-BufferVoltage >0 )
		{
			vitalDacWrite( 6, PressureLeft.Muscle6-BufferVoltage);
		}
		if( PressureLeft.Muscle7-BufferVoltage >0 )
		{
			vitalDacWrite( 7, PressureLeft.Muscle7-BufferVoltage);
		}

		vitalSelectBoard(1);
		if( PressureLeft.Muscle8-BufferVoltage >0 )
		{
			vitalDacWrite( 0, PressureLeft.Muscle8-BufferVoltage);
		}
		if( PressureLeft.Muscle9-BufferVoltage >0 )
		{
			vitalDacWrite( 1, PressureLeft.Muscle9-BufferVoltage);
		}
		if( PressureLeft.Muscle10-BufferVoltage >0 )
		{
			vitalDacWrite( 2, PressureLeft.Muscle10-BufferVoltage);
		}
		if( PressureLeft.Muscle11-BufferVoltage >0 )
```

```
                {
                        vitalDacWrite( 3, PressureLeft.Muscle11-BufferVoltage);
                }
*/
                if( PressureRight.Muscle0-BufferVoltage >0 )
                {
                        vitalDacWrite( 4, PressureRight.Muscle0-BufferVoltage);
                }
                if( PressureRight.Muscle1-BufferVoltage >0 )
                {
                        vitalDacWrite( 5, PressureRight.Muscle1-BufferVoltage);
                }
                if( PressureRight.Muscle2-BufferVoltage >0 )
                {
                        vitalDacWrite( 6, PressureRight.Muscle2-BufferVoltage);
                }
                if( PressureRight.Muscle3-BufferVoltage >0 )
                {
                        vitalDacWrite( 7, PressureRight.Muscle3-BufferVoltage);
                }
                vitalSelectBoard(2);
//              if( PressureRight.Muscle4-BufferVoltage >0 )
//              {
//                      vitalDacWrite( 0, PressureRight.Muscle4-BufferVoltage);
//              }
                if( PressureRight.Muscle5-BufferVoltage >0 )
                {
                        vitalDacWrite( 1, PressureRight.Muscle5-BufferVoltage);
                }
                if( PressureRight.Muscle6-BufferVoltage >0 )
                {
                        vitalDacWrite( 2, PressureRight.Muscle6-BufferVoltage);
                }
//              if( PressureRight.Muscle7-BufferVoltage >0 )
//              {
//                      vitalDacWrite( 3, PressureRight.Muscle7-BufferVoltage);
//              }
                if( PressureRight.Muscle8-BufferVoltage >0 )
                {
                        vitalDacWrite( 4, PressureRight.Muscle8-BufferVoltage);
                }
                if( PressureRight.Muscle9-BufferVoltage >0 )
                {
                        vitalDacWrite( 5, PressureRight.Muscle9-BufferVoltage);
                }
                if( PressureRight.Muscle10-BufferVoltage >0 )
                {
                        vitalDacWrite( 6, PressureRight.Muscle10-BufferVoltage);
                }
                if( PressureRight.Muscle11-BufferVoltage >0 )
                {
                        vitalDacWrite( 7, PressureRight.Muscle11-BufferVoltage);
                }
                Sleep(200);
                BufferVoltage += 0.05;
        }
        Gripper_Pressure_Right=0;
        vitalSelectBoard(2);
        vitalDacWrite(3, Gripper_Pressure_Right);

        PositionDesiredLeft.X=PositionCurrentLeft.X=0;
        PositionDesiredLeft.Y=PositionCurrentLeft.Y=0;
        PositionDesiredLeft.Z=PositionCurrentLeft.Z=0;
        OrientationDesiredLeft.R=OrientationCurrentLeft.R=0;
        OrientationDesiredLeft.P=OrientationCurrentLeft.P=0;
        OrientationDesiredLeft.W=OrientationCurrentLeft.W=0;

        PositionDesiredRight.X=PositionCurrentRight.X=0;
        PositionDesiredRight.Y=PositionCurrentRight.Y=0;
```

```
PositionDesiredRight.Z=PositionCurrentRight.Z=0;
OrientationDesiredLeft.R=OrientationCurrentRight.R=0;
OrientationDesiredLeft.P=OrientationCurrentRight.P=0;
OrientationDesiredLeft.W=OrientationCurrentRight.W=0;


PressurePreviousLeft.Muscle0=PressureLeft.Muscle0=PressureCurrentLeft.Muscle0=0;
PressurePreviousLeft.Muscle1=PressureLeft.Muscle1=PressureCurrentLeft.Muscle1=0;
PressurePreviousLeft.Muscle2=PressureLeft.Muscle2=PressureCurrentLeft.Muscle2=0;
PressurePreviousLeft.Muscle3=PressureLeft.Muscle3=PressureCurrentLeft.Muscle3=0;
PressurePreviousLeft.Muscle4=PressureLeft.Muscle4=PressureCurrentLeft.Muscle4=0;
PressurePreviousLeft.Muscle5=PressureLeft.Muscle5=PressureCurrentLeft.Muscle5=0;
PressurePreviousLeft.Muscle6=PressureLeft.Muscle6=PressureCurrentLeft.Muscle6=0;
PressurePreviousLeft.Muscle7=PressureLeft.Muscle7=PressureCurrentLeft.Muscle7=0;
PressurePreviousLeft.Muscle8=PressureLeft.Muscle8=PressureCurrentLeft.Muscle8=0;
PressurePreviousLeft.Muscle9=PressureLeft.Muscle9=PressureCurrentLeft.Muscle9=0;
PressurePreviousLeft.Muscle10=PressureLeft.Muscle10=PressureCurrentLeft.Muscle10=0;
PressurePreviousLeft.Muscle11=PressureLeft.Muscle11=PressureCurrentLeft.Muscle11=0;

PressurePreviousRight.Muscle0=PressureRight.Muscle0=PressureCurrentRight.Muscle0=0;
PressurePreviousRight.Muscle1=PressureRight.Muscle1=PressureCurrentRight.Muscle1=0;
PressurePreviousRight.Muscle2=PressureRight.Muscle2=PressureCurrentRight.Muscle2=0;
PressurePreviousRight.Muscle3=PressureRight.Muscle3=PressureCurrentRight.Muscle3=0;
PressurePreviousRight.Muscle4=PressureRight.Muscle4=PressureCurrentRight.Muscle4=0;
PressurePreviousRight.Muscle5=PressureRight.Muscle5=PressureCurrentRight.Muscle5=0;
PressurePreviousRight.Muscle6=PressureRight.Muscle6=PressureCurrentRight.Muscle6=0;
PressurePreviousRight.Muscle7=PressureRight.Muscle7=PressureCurrentRight.Muscle7=0;
PressurePreviousRight.Muscle8=PressureRight.Muscle8=PressureCurrentRight.Muscle8=0;
PressurePreviousRight.Muscle9=PressureRight.Muscle9=PressureCurrentRight.Muscle9=0;
PressurePreviousRight.Muscle10=PressureRight.Muscle10=PressureCurrentRight.Muscle10=0;
PressurePreviousRight.Muscle11=PressureRight.Muscle11=PressureCurrentRight.Muscle11=0;
ResetLeftEncoders();
ResetRightEncoders();
ReadLeftEncoders();
ReadRightEncoders();
ComputeLeftAngles();
ComputeRightAngles();
AngleDesiredLeft.Joint1=AnglePreviousLeft.Joint1=AngleCurrentLeft.Joint1;
AngleDesiredLeft.Joint2=AnglePreviousLeft.Joint2=AngleCurrentLeft.Joint2;
AngleDesiredLeft.Joint3=AnglePreviousLeft.Joint3=AngleCurrentLeft.Joint3;
AngleDesiredLeft.Joint4=AnglePreviousLeft.Joint4=AngleCurrentLeft.Joint4;
AngleDesiredLeft.Joint5=AnglePreviousLeft.Joint5=AngleCurrentLeft.Joint5;
AngleDesiredLeft.Joint6=AnglePreviousLeft.Joint6=AngleCurrentLeft.Joint6;

AngleDesiredRight.Joint1=AnglePreviousRight.Joint1=AngleCurrentRight.Joint1;
AngleDesiredRight.Joint2=AnglePreviousRight.Joint2=AngleCurrentRight.Joint2;
AngleDesiredRight.Joint3=AnglePreviousRight.Joint3=AngleCurrentRight.Joint3;
AngleDesiredRight.Joint4=AnglePreviousRight.Joint4=AngleCurrentRight.Joint4;
AngleDesiredRight.Joint5=AnglePreviousRight.Joint5=AngleCurrentRight.Joint5;
AngleDesiredRight.Joint6=AnglePreviousRight.Joint6=AngleCurrentRight.Joint6;

SetDlgItemInt(IDC_LeftCurrentJoint1,(int)AngleCurrentLeft.Joint1);
SetDlgItemInt(IDC_LeftCurrentJoint2,(int)AngleCurrentLeft.Joint2);
SetDlgItemInt(IDC_LeftCurrentJoint3,(int)AngleCurrentLeft.Joint3);
SetDlgItemInt(IDC_LeftCurrentJoint4,(int)AngleCurrentLeft.Joint4);
SetDlgItemInt(IDC_LeftCurrentJoint5,(int)AngleCurrentLeft.Joint5);
SetDlgItemInt(IDC_LeftCurrentJoint6,(int)AngleCurrentLeft.Joint6);

SetDlgItemInt(IDC_RightCurrentJoint1,(int)AngleCurrentRight.Joint1);
SetDlgItemInt(IDC_RightCurrentJoint2,(int)AngleCurrentRight.Joint2);
SetDlgItemInt(IDC_RightCurrentJoint3,(int)AngleCurrentRight.Joint3);
SetDlgItemInt(IDC_RightCurrentJoint4,(int)AngleCurrentRight.Joint4);
SetDlgItemInt(IDC_RightCurrentJoint5,(int)AngleCurrentRight.Joint5);
SetDlgItemInt(IDC_RightCurrentJoint6,(int)AngleCurrentRight.Joint6);

SetDlgItemInt(IDC_LeftCurrentX,(int)PositionCurrentLeft.X);
SetDlgItemInt(IDC_LeftCurrentY,(int)PositionCurrentLeft.Y);
SetDlgItemInt(IDC_LeftCurrentZ,(int)PositionCurrentLeft.Z);
SetDlgItemInt(IDC_LeftCurrentR,(int)OrientationCurrentLeft.R);
```

```
	SetDlgItemInt(IDC_LeftCurrentP,(int)OrientationCurrentLeft.P);
	SetDlgItemInt(IDC_LeftCurrentW,(int)OrientationCurrentLeft.W);

	SetDlgItemInt(IDC_RightCurrentX,(int)PositionCurrentRight.X);
	SetDlgItemInt(IDC_RightCurrentY,(int)PositionCurrentRight.Y);
	SetDlgItemInt(IDC_RightCurrentZ,(int)PositionCurrentRight.Z);
	SetDlgItemInt(IDC_RightCurrentR,(int)OrientationCurrentRight.R);
	SetDlgItemInt(IDC_RightCurrentP,(int)OrientationCurrentRight.P);
	SetDlgItemInt(IDC_RightCurrentW,(int)OrientationCurrentRight.W);

	SetDlgItemInt(IDC_LeftDesiredJoint1,(int)AngleDesiredLeft.Joint1);
	SetDlgItemInt(IDC_LeftDesiredJoint2,(int)AngleDesiredLeft.Joint2);
	SetDlgItemInt(IDC_LeftDesiredJoint3,(int)AngleDesiredLeft.Joint3);
	SetDlgItemInt(IDC_LeftDesiredJoint4,(int)AngleDesiredLeft.Joint4);
	SetDlgItemInt(IDC_LeftDesiredJoint5,(int)AngleDesiredLeft.Joint5);
	SetDlgItemInt(IDC_LeftDesiredJoint6,(int)AngleDesiredLeft.Joint6);

	SetDlgItemInt(IDC_RightDesiredJoint1,(int)AngleDesiredRight.Joint1);
	SetDlgItemInt(IDC_RightDesiredJoint2,(int)AngleDesiredRight.Joint2);
	SetDlgItemInt(IDC_RightDesiredJoint3,(int)AngleDesiredRight.Joint3);
	SetDlgItemInt(IDC_RightDesiredJoint4,(int)AngleDesiredRight.Joint4);
	SetDlgItemInt(IDC_RightDesiredJoint5,(int)AngleDesiredRight.Joint5);
	SetDlgItemInt(IDC_RightDesiredJoint6,(int)AngleDesiredRight.Joint6);

	SetDlgItemInt(IDC_LeftDesiredX,(int)PositionDesiredLeft.X);
	SetDlgItemInt(IDC_LeftDesiredY,(int)PositionDesiredLeft.Y);
	SetDlgItemInt(IDC_LeftDesiredZ,(int)PositionDesiredLeft.Z);
	SetDlgItemInt(IDC_LeftDesiredR,(int)OrientationDesiredLeft.R);
	SetDlgItemInt(IDC_LeftDesiredP,(int)OrientationDesiredLeft.P);
	SetDlgItemInt(IDC_LeftDesiredW,(int)OrientationDesiredLeft.W);

	SetDlgItemInt(IDC_RightDesiredX,(int)PositionDesiredRight.X);
	SetDlgItemInt(IDC_RightDesiredY,(int)PositionDesiredRight.Y);
	SetDlgItemInt(IDC_RightDesiredZ,(int)PositionDesiredRight.Z);
	SetDlgItemInt(IDC_RightDesiredR,(int)OrientationDesiredRight.R);
	SetDlgItemInt(IDC_RightDesiredP,(int)OrientationDesiredRight.P);
	SetDlgItemInt(IDC_RightDesiredW,(int)OrientationDesiredRight.W);
}

void CArmControlDlg::OnHomePosition()
{
	// TODO: Add your control notification handler code here
	LeftHomePosition();
	RightHomePosition();

	PositionDesiredLeft.X=PositionCurrentLeft.X=ISACParameterLeft.D4;
	PositionDesiredLeft.Y=PositionCurrentLeft.Y=ISACParameterLeft.D3;
	PositionDesiredLeft.Z=PositionCurrentLeft.Z=-ISACParameterLeft.A2;
	OrientationDesiredLeft.R=OrientationCurrentLeft.R=0;
	OrientationDesiredLeft.P=OrientationCurrentLeft.P=0;
	OrientationDesiredLeft.W=OrientationCurrentLeft.W=0;

	PositionDesiredRight.X=PositionCurrentRight.X=ISACParameterRight.D4;
	PositionDesiredRight.Y=PositionCurrentRight.Y=ISACParameterRight.D3;
	PositionDesiredRight.Z=PositionCurrentRight.Z=-ISACParameterRight.A2;
	OrientationDesiredLeft.R=OrientationCurrentRight.R=0;
	OrientationDesiredLeft.P=OrientationCurrentRight.P=0;
	OrientationDesiredLeft.W=OrientationCurrentRight.W=0;

	PressurePreviousLeft.Muscle0=PressureLeft.Muscle0=PressureCurrentLeft.Muscle0=2.3;
	PressurePreviousLeft.Muscle1=PressureLeft.Muscle1=PressureCurrentLeft.Muscle1=1.7;
	PressurePreviousLeft.Muscle2=PressureLeft.Muscle2=PressureCurrentLeft.Muscle2=2.0;
	PressurePreviousLeft.Muscle3=PressureLeft.Muscle3=PressureCurrentLeft.Muscle3=2.0;
	PressurePreviousLeft.Muscle4=PressureLeft.Muscle4=PressureCurrentLeft.Muscle4=0;
	PressurePreviousLeft.Muscle5=PressureLeft.Muscle5=PressureCurrentLeft.Muscle5=2.2;
	PressurePreviousLeft.Muscle6=PressureLeft.Muscle6=PressureCurrentLeft.Muscle6=2.4;
	PressurePreviousLeft.Muscle7=PressureLeft.Muscle7=PressureCurrentLeft.Muscle7=0;
	PressurePreviousLeft.Muscle8=PressureLeft.Muscle8=PressureCurrentLeft.Muscle8=2.3;
	PressurePreviousLeft.Muscle9=PressureLeft.Muscle9=PressureCurrentLeft.Muscle9=1.7;
```

```
PressurePreviousLeft.Muscle10=PressureLeft.Muscle10=PressureCurrentLeft.Muscle10=1.7;
PressurePreviousLeft.Muscle11=PressureLeft.Muscle11=PressureCurrentLeft.Muscle11=2.3;

PressurePreviousRight.Muscle0=PressureRight.Muscle0=PressureCurrentRight.Muscle0=2.3;
PressurePreviousRight.Muscle1=PressureRight.Muscle1=PressureCurrentRight.Muscle1=1.7;
PressurePreviousRight.Muscle2=PressureRight.Muscle2=PressureCurrentRight.Muscle2=2.0;
PressurePreviousRight.Muscle3=PressureRight.Muscle3=PressureCurrentRight.Muscle3=2.0;
PressurePreviousRight.Muscle4=PressureRight.Muscle4=PressureCurrentRight.Muscle4=0;
PressurePreviousRight.Muscle5=PressureRight.Muscle5=PressureCurrentRight.Muscle5=2.2;
PressurePreviousRight.Muscle6=PressureRight.Muscle6=PressureCurrentRight.Muscle6=2.4;
PressurePreviousRight.Muscle7=PressureRight.Muscle7=PressureCurrentRight.Muscle7=0;
PressurePreviousRight.Muscle8=PressureRight.Muscle8=PressureCurrentRight.Muscle8=2.3;
PressurePreviousRight.Muscle9=PressureRight.Muscle9=PressureCurrentRight.Muscle9=1.7;
PressurePreviousRight.Muscle10=PressureRight.Muscle10=PressureCurrentRight.Muscle10=1.7;
PressurePreviousRight.Muscle11=PressureRight.Muscle11=PressureCurrentRight.Muscle11=2.3;

ResetLeftEncoders();
ResetRightEncoders();
ReadLeftEncoders();
ReadRightEncoders();
ComputeLeftAngles();
ComputeRightAngles();
AngleDesiredLeft.Joint1=AnglePreviousLeft.Joint1=AngleCurrentLeft.Joint1;
AngleDesiredLeft.Joint2=AnglePreviousLeft.Joint2=AngleCurrentLeft.Joint2;
AngleDesiredLeft.Joint3=AnglePreviousLeft.Joint3=AngleCurrentLeft.Joint3;
AngleDesiredLeft.Joint4=AnglePreviousLeft.Joint4=AngleCurrentLeft.Joint4;
AngleDesiredLeft.Joint5=AnglePreviousLeft.Joint5=AngleCurrentLeft.Joint5;
AngleDesiredLeft.Joint6=AnglePreviousLeft.Joint6=AngleCurrentLeft.Joint6;

AngleDesiredRight.Joint1=AnglePreviousRight.Joint1=AngleCurrentRight.Joint1;
AngleDesiredRight.Joint2=AnglePreviousRight.Joint2=AngleCurrentRight.Joint2;
AngleDesiredRight.Joint3=AnglePreviousRight.Joint3=AngleCurrentRight.Joint3;
AngleDesiredRight.Joint4=AnglePreviousRight.Joint4=AngleCurrentRight.Joint4;
AngleDesiredRight.Joint5=AnglePreviousRight.Joint5=AngleCurrentRight.Joint5;
AngleDesiredRight.Joint6=AnglePreviousRight.Joint6=AngleCurrentRight.Joint6;

SetDlgItemInt(IDC_LeftCurrentJoint1,(int)AngleCurrentLeft.Joint1);
SetDlgItemInt(IDC_LeftCurrentJoint2,(int)AngleCurrentLeft.Joint2);
SetDlgItemInt(IDC_LeftCurrentJoint3,(int)AngleCurrentLeft.Joint3);
SetDlgItemInt(IDC_LeftCurrentJoint4,(int)AngleCurrentLeft.Joint4);
SetDlgItemInt(IDC_LeftCurrentJoint5,(int)AngleCurrentLeft.Joint5);
SetDlgItemInt(IDC_LeftCurrentJoint6,(int)AngleCurrentLeft.Joint6);

SetDlgItemInt(IDC_RightCurrentJoint1,(int)AngleCurrentRight.Joint1);
SetDlgItemInt(IDC_RightCurrentJoint2,(int)AngleCurrentRight.Joint2);
SetDlgItemInt(IDC_RightCurrentJoint3,(int)AngleCurrentRight.Joint3);
SetDlgItemInt(IDC_RightCurrentJoint4,(int)AngleCurrentRight.Joint4);
SetDlgItemInt(IDC_RightCurrentJoint5,(int)AngleCurrentRight.Joint5);
SetDlgItemInt(IDC_RightCurrentJoint6,(int)AngleCurrentRight.Joint6);

SetDlgItemInt(IDC_LeftCurrentX,(int)PositionCurrentLeft.X);
SetDlgItemInt(IDC_LeftCurrentY,(int)PositionCurrentLeft.Y);
SetDlgItemInt(IDC_LeftCurrentZ,(int)PositionCurrentLeft.Z);
SetDlgItemInt(IDC_LeftCurrentR,(int)OrientationCurrentLeft.R);
SetDlgItemInt(IDC_LeftCurrentP,(int)OrientationCurrentLeft.P);
SetDlgItemInt(IDC_LeftCurrentW,(int)OrientationCurrentLeft.W);

SetDlgItemInt(IDC_RightCurrentX,(int)PositionCurrentRight.X);
SetDlgItemInt(IDC_RightCurrentY,(int)PositionCurrentRight.Y);
SetDlgItemInt(IDC_RightCurrentZ,(int)PositionCurrentRight.Z);
SetDlgItemInt(IDC_RightCurrentR,(int)OrientationCurrentRight.R);
SetDlgItemInt(IDC_RightCurrentP,(int)OrientationCurrentRight.P);
SetDlgItemInt(IDC_RightCurrentW,(int)OrientationCurrentRight.W);

SetDlgItemInt(IDC_LeftDesiredJoint1,(int)AngleDesiredLeft.Joint1);
SetDlgItemInt(IDC_LeftDesiredJoint2,(int)AngleDesiredLeft.Joint2);
SetDlgItemInt(IDC_LeftDesiredJoint3,(int)AngleDesiredLeft.Joint3);
SetDlgItemInt(IDC_LeftDesiredJoint4,(int)AngleDesiredLeft.Joint4);
SetDlgItemInt(IDC_LeftDesiredJoint5,(int)AngleDesiredLeft.Joint5);
```

```
	SetDlgItemInt(IDC_LeftDesiredJoint6,(int)AngleDesiredLeft.Joint6);

	SetDlgItemInt(IDC_RightDesiredJoint1,(int)AngleDesiredRight.Joint1);
	SetDlgItemInt(IDC_RightDesiredJoint2,(int)AngleDesiredRight.Joint2);
	SetDlgItemInt(IDC_RightDesiredJoint3,(int)AngleDesiredRight.Joint3);
	SetDlgItemInt(IDC_RightDesiredJoint4,(int)AngleDesiredRight.Joint4);
	SetDlgItemInt(IDC_RightDesiredJoint5,(int)AngleDesiredRight.Joint5);
	SetDlgItemInt(IDC_RightDesiredJoint6,(int)AngleDesiredRight.Joint6);

	SetDlgItemInt(IDC_LeftDesiredX,(int)PositionDesiredLeft.X);
	SetDlgItemInt(IDC_LeftDesiredY,(int)PositionDesiredLeft.Y);
	SetDlgItemInt(IDC_LeftDesiredZ,(int)PositionDesiredLeft.Z);
	SetDlgItemInt(IDC_LeftDesiredR,(int)OrientationDesiredLeft.R);
	SetDlgItemInt(IDC_LeftDesiredP,(int)OrientationDesiredLeft.P);
	SetDlgItemInt(IDC_LeftDesiredW,(int)OrientationDesiredLeft.W);

	SetDlgItemInt(IDC_RightDesiredX,(int)PositionDesiredRight.X);
	SetDlgItemInt(IDC_RightDesiredY,(int)PositionDesiredRight.Y);
	SetDlgItemInt(IDC_RightDesiredZ,(int)PositionDesiredRight.Z);
	SetDlgItemInt(IDC_RightDesiredR,(int)OrientationDesiredRight.R);
	SetDlgItemInt(IDC_RightDesiredP,(int)OrientationDesiredRight.P);
	SetDlgItemInt(IDC_RightDesiredW,(int)OrientationDesiredRight.W);

}

void CArmControlDlg::LeftHomePosition()
{
	int step=20;
	int i;
	double BufferVoltage[12];
	BufferVoltage[0]=(2.3-PressureLeft.Muscle0)/step;
	BufferVoltage[1]=(1.7-PressureLeft.Muscle1)/step;
	BufferVoltage[2]=(2.0-PressureLeft.Muscle2)/step;
	BufferVoltage[3]=(2.0-PressureLeft.Muscle3)/step;
	BufferVoltage[4]=(0-PressureLeft.Muscle4)/step;
	BufferVoltage[5]=(2.2-PressureLeft.Muscle5)/step;
	BufferVoltage[6]=(2.4-PressureLeft.Muscle6)/step;
	BufferVoltage[7]=(0-PressureLeft.Muscle7)/step;
	BufferVoltage[8]=(2.3-PressureLeft.Muscle8)/step;
	BufferVoltage[9]=(1.7-PressureLeft.Muscle9)/step;
	BufferVoltage[10]=(1.7-PressureLeft.Muscle10)/step;
	BufferVoltage[11]=(2.3-PressureLeft.Muscle11)/step;
	for(i=0;i<step;i++)
	{
		vitalSelectBoard(0);
		PressureLeft.Muscle0=PressureLeft.Muscle0+BufferVoltage[0];
		vitalDacWrite(0, PressureLeft.Muscle0);
		PressureLeft.Muscle1=PressureLeft.Muscle1+BufferVoltage[1];
		vitalDacWrite(1, PressureLeft.Muscle1);
		PressureLeft.Muscle2=PressureLeft.Muscle2+BufferVoltage[2];
		vitalDacWrite(2, PressureLeft.Muscle2);
		PressureLeft.Muscle3=PressureLeft.Muscle3+BufferVoltage[3];
		vitalDacWrite(3, PressureLeft.Muscle3);
		PressureLeft.Muscle4=PressureLeft.Muscle4+BufferVoltage[4];
		vitalDacWrite(4, PressureLeft.Muscle4);
		PressureLeft.Muscle5=PressureLeft.Muscle5+BufferVoltage[5];
		vitalDacWrite(5, PressureLeft.Muscle5);
		PressureLeft.Muscle6=PressureLeft.Muscle6+BufferVoltage[6];
		vitalDacWrite(6, PressureLeft.Muscle6);
		PressureLeft.Muscle7=PressureLeft.Muscle7+BufferVoltage[7];
		vitalDacWrite(7, PressureLeft.Muscle7);
		vitalSelectBoard(1);
		PressureLeft.Muscle8=PressureLeft.Muscle8+BufferVoltage[8];
		vitalDacWrite(0, PressureLeft.Muscle8);
		PressureLeft.Muscle9=PressureLeft.Muscle9+BufferVoltage[9];
		vitalDacWrite(1, PressureLeft.Muscle9);
		PressureLeft.Muscle10=PressureLeft.Muscle10+BufferVoltage[10];
		vitalDacWrite(2, PressureLeft.Muscle10);
		PressureLeft.Muscle11=PressureLeft.Muscle11+BufferVoltage[11];
```

```
			vitalDacWrite(3, PressureLeft.Muscle11);
		}

}

void CArmControlDlg::RightHomePosition()
{
		int step=20;
		int i;
		double BufferVoltage[12];
		BufferVoltage[0]=(2.3-PressureRight.Muscle0)/step;
		BufferVoltage[1]=(1.7-PressureRight.Muscle1)/step;
		BufferVoltage[2]=(2.0-PressureRight.Muscle2)/step;
		BufferVoltage[3]=(2.0-PressureRight.Muscle3)/step;
		BufferVoltage[4]=(0-PressureRight.Muscle4)/step;
		BufferVoltage[5]=(2.2-PressureRight.Muscle5)/step;
		BufferVoltage[6]=(2.4-PressureRight.Muscle6)/step;
		BufferVoltage[7]=(0-PressureRight.Muscle7)/step;
		BufferVoltage[8]=(2.3-PressureRight.Muscle8)/step;
		BufferVoltage[9]=(1.7-PressureRight.Muscle9)/step;
		BufferVoltage[10]=(1.7-PressureRight.Muscle10)/step;
		BufferVoltage[11]=(2.3-PressureRight.Muscle11)/step;
		for(i=0;i<20;i++)
		{
			vitalSelectBoard(1);
			PressureRight.Muscle0=PressureRight.Muscle0+BufferVoltage[0];
			vitalDacWrite(4, PressureRight.Muscle0);
			PressureRight.Muscle1=PressureRight.Muscle1+BufferVoltage[1];
			vitalDacWrite(5, PressureRight.Muscle1);
			PressureRight.Muscle2=PressureRight.Muscle2+BufferVoltage[2];
			vitalDacWrite(6, PressureRight.Muscle2);
			PressureRight.Muscle3=PressureRight.Muscle3+BufferVoltage[3];
			vitalDacWrite(7, PressureRight.Muscle3);
			vitalSelectBoard(2);
			PressureRight.Muscle0=PressureRight.Muscle4+BufferVoltage[4];
			vitalDacWrite(0, PressureRight.Muscle4);
			PressureRight.Muscle1=PressureRight.Muscle5+BufferVoltage[5];
			vitalDacWrite(1, PressureRight.Muscle5);
			PressureRight.Muscle2=PressureRight.Muscle6+BufferVoltage[6];
			vitalDacWrite(2, PressureRight.Muscle6);
			PressureRight.Muscle3=PressureRight.Muscle7+BufferVoltage[7];
			vitalDacWrite(3, PressureRight.Muscle7);
			PressureRight.Muscle4=PressureRight.Muscle8+BufferVoltage[8];
			vitalDacWrite(4, PressureRight.Muscle8);
			PressureRight.Muscle5=PressureRight.Muscle9+BufferVoltage[9];
			vitalDacWrite(5, PressureRight.Muscle9);
			PressureRight.Muscle6=PressureRight.Muscle10+BufferVoltage[10];
			vitalDacWrite(6, PressureRight.Muscle10);
			PressureRight.Muscle7=PressureRight.Muscle11+BufferVoltage[11];
			vitalDacWrite(7, PressureRight.Muscle11);

		}



}

void CArmControlDlg::OnGripperRight()
{
		// TODO: Add your control notification handler code here
		if(Gripper_Pressure_Right==0)
		{
			Gripper_Pressure_Right=3.6;
			vitalSelectBoard(2);
			vitalDacWrite(3, Gripper_Pressure_Right);
		}
		else
		{
			Gripper_Pressure_Right=0;
```

```
                vitalSelectBoard(2);
                vitalDacWrite(3, Gripper_Pressure_Right);
        }
}
```

REFERENCES

[1] Land, M.F., The functions of eye movements in animals remote from man. In *Eye movement mechanisms, processes and applications*. (eds. J.M.Findlay, R.Walker and R.W.Kentridge) pp.63-76, Elsevier, Amsterdam, 1995

[2] Land, M.F. and Nilsson, D-E. *Animal Eyes*. Oxford University Press, Oxford, 2002

[3] K. Kawamura, et al, “ISAC: Foundations in Human-Humanoid Interaction,” *IEEE Intelligent Systems & their application*, pp.38-45, July/August 2000

[4] B. Tondu and P.Lopez, “Modeling and Control of Mckibben Artificial Muscle Robot Actuators,” *IEEE Control Systems Magazine*, pp.15-36, April 2000

[5] Wikipedia, http://en.wikipedia.org/wiki/HSV

[6] D.Marr, *Vision*, W.H. Freeman, San Francisco, Calif., 1982

[7] Feldman, J., Connectionist models and parallelism in high level vision. *Computer Vision, Graphics, and Image Processing*, 31, 178– 200, 1985

[8] Treisman, A.; Gelade, G., "A feature integration theory of attention.", *Cognitive Psychology* 12 (1): 97–136, 1980

[9] R. Bajcsy, “Active Perception,” *Proceedings of IEEE*, Vol. 76, No.8, pp. 996-1005, August 1988

[10] Shapiro, L., and Stockman, G., *Computer Vision,* Prentice Hall, pp. 69–73, 2002

[11] Rafael C.Gonazalez, *Digital Image Processing*, Prentice Hall, 2010

[12] Edward R. Dougherty, *An Introduction to Morphological Image Processing*, 1992

[13] Willow Garage, http://opencv.willowgarage.com/wiki/

[14] Center of Intelligent System (CIS), Vanderbilt University, http://eecs.vuse.vanderbilt.edu/CIS/cishome.shtml

[15] Logitech Company, http://www.logitech.com/en-gb/webcam-communications/webcams/devices/5867

[16] FLIR Systems Inc., http://www.flir.com/mcs/products/ptu-d46/index.cfm

[17] J. Schröder and D. Erol, “Dynamic Pneumatic Actuator Model for a Model-Based Torque Controller”, *International Symposium on Computational Intelligence and Automation*, pp.342-347, July 2003

[18] Wikipedia, http://en.wikipedia.org/wiki/Pneumatic_artificial_muscles

[19] T. Tsuji, S. Miyata, T. Hashimoto, and H. Kobayashi, “Controller Design for Robot with Pneumatic Artificial Muscles,” in *SICE-ICASE* 2006, pp. 5419-5422, 18-21, Oct. 2006.

[20] Northrup, Steven Gerold., “Biologically-inspired control of a humanoid robot with non-linear actuators”, Ph.D Thesis, Vanderbilt University, 2001

[21] B. Ulutas, E. Erdemir, and K. Kawamura, “Application of a Hybrid Controller with Non-Contact Impedance to a Humanoid Robot,” in *Proc. IEEE International* Workshop on Variable Structure Systems 2008, pp. 378-383, 8-10 June 2008.

[22] Bridgestone Corporation, http://www.bridgestone.com/

[23] Shadow Robot Company, “Shadow Robot Company: Air Muscles Overview,” 6 Feb. 2009, http://www.shadowrobot.com/airmuscles.

[24] SMC Corporation of America, http://www.smcetech.com/

[25] Vital System, Inc., “Motion Control PCI Card,” 7 Feb. 2009, http://www.vitalsystem.com/web/motion/motionLite.php.

[26] Danaher Precision Systems ,“Encoder Basics,” 19 Feb. 2009, http://www.neat.com/products/electronics/pdf/EncoderBasics.pdf.

[27] Logitech Company, http://www.logitech.com/en-gb/support-downloads/downloads

[28] J.Serra, *Image Analysis and Mathematical Morphology*, New York: Academic Press, 1983

[29] The Mathworks, Inc., http://www.mathworks.com/help/toolbox/images/ref/regionprops.html

[30] Huan Tan, Introduction of the initialization steps for ISAC arm control.doc, 2009, CIS unpublished report

[31] Podhajeckyand G, Glasa, J, “On planar affine transform determination”, in *Computers and artificial intelligence*, Vol.16(3), p.259-274, 1997

[32] Wikipedia, http://en.wikipedia.org/wiki/Fuzzy_locating