Real-Time Camera Tracking by Using Fuzzy Control

Beifang Yi, Rong Hu, and Zhiwei Zhu

May 15, 2001

Contents

1 Introduction **3**

2 Background **4**

3 Methodology **4**

3.1 Two Steps 4
3.2 Flowchart for the Project 4
3.3 Fuzzy Rules for Camera Tracking 5
3.4 FSMFs for camera tracking 7

4 Computer Runs **8**

5 Results **9**

5.1 Results in the Simulation of Camera Tracking 9
5.2 Results in the Real-time Camera Tracking 15

6 Analysis and Conclusions **20**

7 Gaze Average Vector Calculation by Using Fuzzy Average Method in Chapter 4 **20**

8 Distribution of the Tasks in This Project **21**

A Camera Tracking Source Codes **23**

B Data of Camera Tracking (Simulation) **42**

B.1 The Recorded Images' Data (Simulation–20 frames) 42
B.2 Data of Camera Tracking (Simulation–upto 133 frames) 43

C Data of Camera Tracking (in Real Time) **46**

C.1 The Recorded Images' Data (17 frames) 46
C.2 The Camera Real Time Tracking Data (upto 185 frames) 47

D Calculating Gaze Vector by Using Fuzzy Average Method **52**

D.1 Codes for Calculating Gaze Fuzzy Average Vector 52
D.2 Input and Result Data of the Gaze Vectors 55

1 Introduction

Fuzzy control is one of the most interesting fields where fuzzy logic theory can be effectively applied in real time control systems. Recently some practical applications of fuzzy control have been reported ([3],[4]). In this project, we will focus on applying fuzzy logic-based controls to camera tracking, which is a subject of computer vision and image processing. The camera captures an image of a size of 640x480 pixels from its *FoV* (Field of View), the center of which is marked with a *cross hair*. The camera can rotate up-and-down and/or side-to-side. If an object (target) appears in the FoV and is caught by the camera, the camera tracking system at first starts up the image processing subsystem to make the target stand out against the background, then locate the target, and finally send commands to the camera so that the cross hair can coincide with the CoM (Center of Mass) of the target. Fig. 1 shows this process.

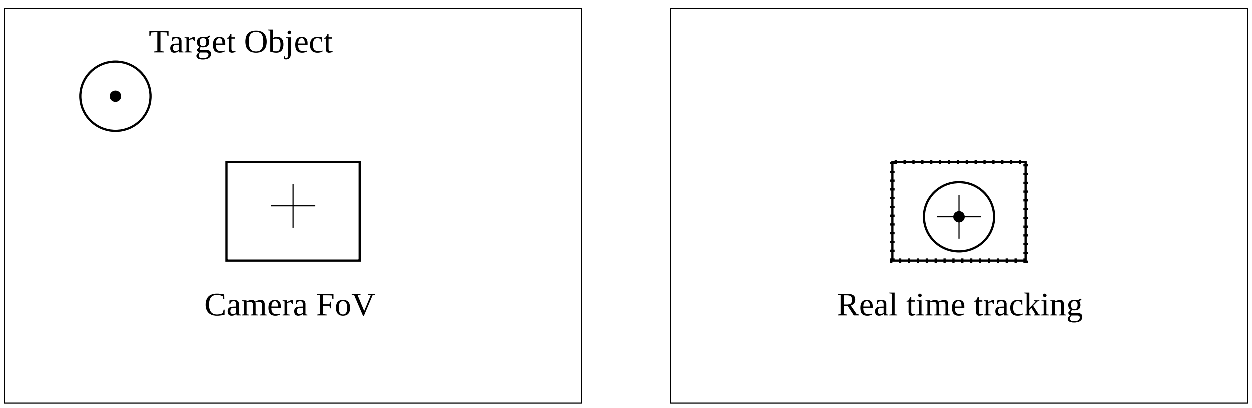
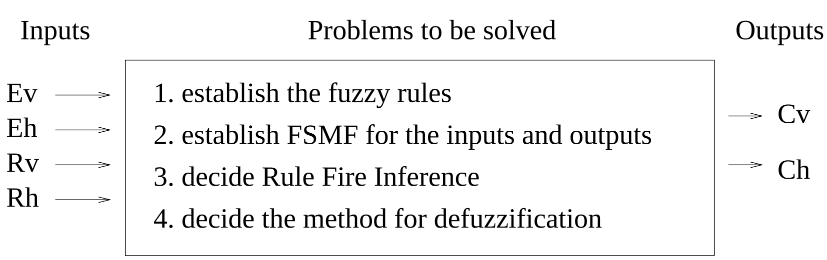


Figure 1. Camera tracking process

Suppose that the target is moving randomly, and the problem in the tracking process is how to rotate the camera by issuing corresponding commands so that the center of the cross hair can follow as quickly as possible and coincide as accurately as possible with the CoM. We can find detailed information about this process in chapter 3 in [2]. Our concentration here is not with the image processing of how to recognize the target, less on the specifications of controlling the camera, but *more on how to set up a set of fuzzy rules and apply those rules to* camera controlling process. Fig. 2 describes our main concerns over this problem.



Ev, Eh −− error between target CoM and FoV cross−hairs along vertical and horizontal axes, respectively.

Rv, Rh −− rate of error change in vertical and horizontal directions, respectively.

Cv, Ch −− counts to be moved in vertical and horizontal dimension, respectively.

Figure 2. The problems to be focused on

2 Background

The fuzzy system has been applied to many practical controlling systems, among which there may have been camera tracking process. But we have not found the specifications about such a topic and applications. Our work on camera tracking can be said, in our department at UNR, to be a starting point toward the field of camera tracking. Our work on this project, although with the emphasis on design and implementation of FSMFs in the camera controlling process, will have the following facilities as the prerequisites to the project:

- Because of the complexity in the recognition of object, we choose a bright white round plate on dark ground as the object (target) in the tracking process.
- Hardware: a Sun workstation and a Sony EVI-D30/31 Pan/Tilt camera.
- Software: Solaris ,Sun C/C++ compiler, Sun Xil library, and the command list/VISCA/RS-232 control protocol delivered with the camera.

In addition, before implementing the fuzzy control system, we have to:

- connect the workstation with the camera, run the test software *SunVideoPlus*, and control the camera through the remote controller;
- write a short program to rotate camera through RS-232 port;
- design and implement the algorithm and program of locating the target (the bright white round plate), that is, to calculate the postion (coordinates) of the target;
- given a defuzzy value, transform it into the commands that can rotate the camera to a certain direction.

We will concentrate on setting up and implementing FSMFs in the camera tracking in the remaining section.

3 Methodology

3.1 Two Steps

Firstly, we will **simulate** the camera tracking problem, assuming that the target object is a white round plate on the dark brown background. Moreover, we use a square frame as a simulated Field of View (FoV) of a camera. A white crosshair on the FoV represents the center of the simulated FoV. And we will employ fuzzy logic based rules to control the movement of the square; making the square frame follow the object and put the cross-hair to this object's CoM (see Figure 3).

On the second step, we will proceed to the camera tracking in real time, as shown in Fig. 1 on page 3.

3.2 Flowchart for the Project

A video camera (Sony EVI-D30/31) is connected to a Sun workstation and it catches a target object in its FOV. While the target is moving, our simulating system will calculate the real CoM of the the target, compute the coordinates of the camera's cross hair, and then move the cross hair to the target's CoM according to the a set of defuzzied values given by our defined fuzzy functions. The overall flowchart for this camera tracking is shown in Fig. 4.

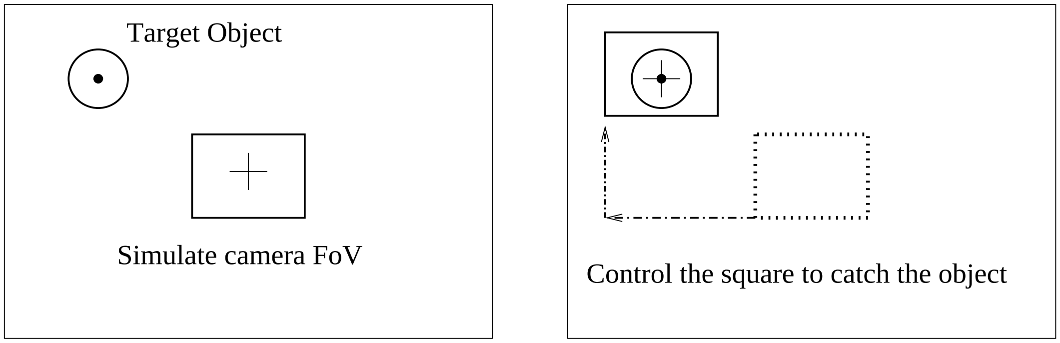


Figure 3. Simulation of camera tracking

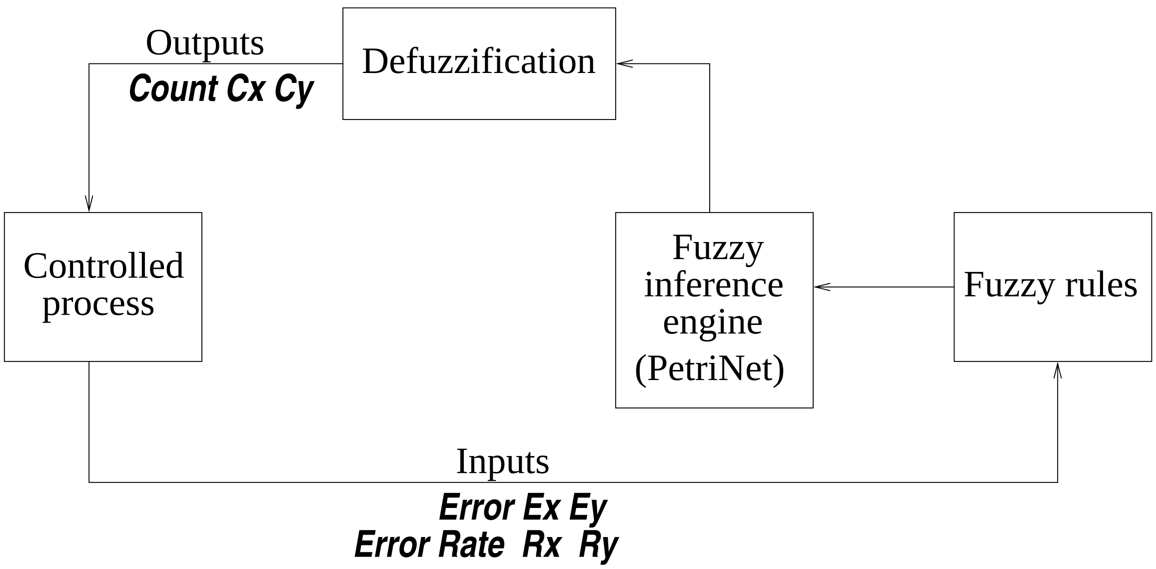


Figure 4. Flowchart of the camera tracking based on fuzzy rule controler

3.3 Fuzzy Rules for Camera Tracking

Before we establish the rules for the camera tracking system, we need to specify the ranges for the various inputs and outputs. The following table is the defined ranges.

Variables	Range				
$Error_h$	FarLeft	NearLeft	Center	NearRight	FarRight
$Error_v$	FarDown	NearDown	Center	NearUp	FarUp
$Rate_h$	MidNegative		Zero	MidPositive	
$Rate_v$	MidNegative		Zero	MidPositive	
$Count_h$	FarLeft	NearLeft	Center	NearRight	FarRight
$Count_v$	FarDown	NearDown	Center	NearUp	FarUp

Now we start to specify what outputs are to be given for the various combinations of inputs. This can be done by a table. If we consider all possible combinations in the table, there will be 30 rules for this tracking problem. It turns out from experiences that only a small fraction of these rules are sufficient for achieving high performance of the resulting fuzzy controller [1]. So we will only use some of these 30 rules.

A simple set of tables are given below for the situation where we control a camera for tracking. Table 1 is the control table for the rules that depend upon the horizontal error and horizontal error rate. The value inside the cell corresponds to the horizontal counts. Similarly, Table 2 is the control table for the rules that depend upon the vertical error and vertical error rate. The value inside the cell corresponds to the vertical counts. Note that the vertical and horizontal motions are independent.

Table 1. The counts for horizontal camera movement

	Error				
Rate of Error	FarLeft	NearLeft	Center	NearRight	FarRight
MidNegative	FarLeft	FarLeft	Zero	Zero	FarRight
Zero		NearLeft		NearRight	
MidPositive		Zero		FarRight	

Table 2. The counts for vertical camera movement

	Error				
Rate of Error	FarDown	NearDown	Center	NearUp	FarUp
MidNegative	FarDown	FarDown	Zero	Zero	FarUp
Zero		NearDown		NearUp	
MidPositive		Zero		FarUp	

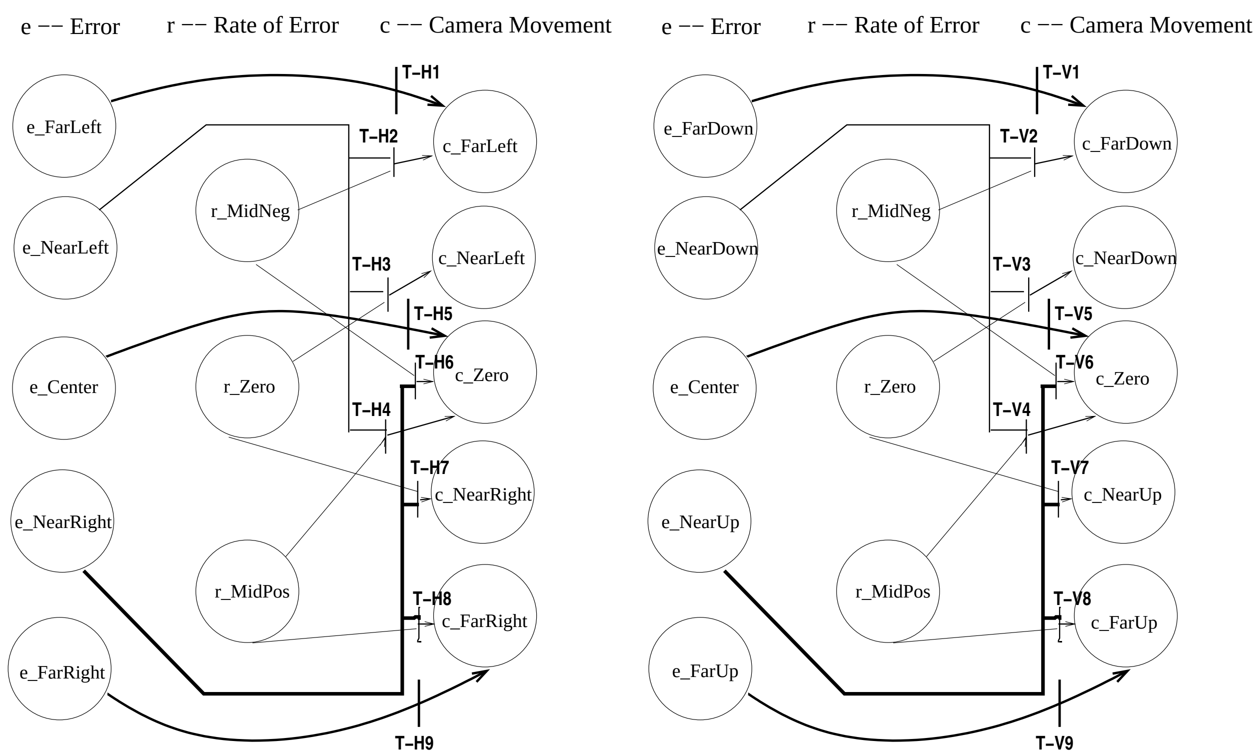
For horizontal movement, the nine rules can be transformed into the following form from Table 1.

No.	RULES		
T-H1	if (e is FarLeft)		then (C is FarLeft)
T-H2	if (e is NearLeft)	and (r is MidNegative)	then (C is FarLeft)
T-H3	if (e is NearLeft)	and (r is Zero)	then (C is NearLeft)
T-H4	if (e is NearLeft)	and (r is MidPositive)	then (C is Zero)
T-H5	if (e is Center)		then (C is Zero)
T-H6	if (e is NearRight)	and (r is MidNegative)	then (C is Zero)
T-H7	if (e is NearRight)	and (r is Zero)	then (C is NearRight)
T-H8	if (e is NearRight)	and (r is MidPositive)	then (C is FarRight)
T-H9	if (e is FarRight)		then (C is FarRight)

For vertical movement, the nine rules can be transformed into the following form from Table 2.

No.	RULES		
T-V1	if (e is FarDown)		then (C is FarDown)
T-V2	if (e is NearDown)	and (r is MidNegative)	then (C is FarDown)
T-V3	if (e is NearDown)	and (r is Zero)	then (C is NearDown)
T-V4	if (e is NearDown)	and (r is MidPositive)	then (C is Zero)
T-V5	if (e is Center)		then (C is Zero)
T-V6	if (e is NearUp)	and (r is MidNegative)	then (C is Zero)
T-V7	if (e is NearUp)	and (r is Zero)	then (C is NearUp)
T-V8	if (e is NearUp)	and (r is MidPositive)	then (C is FarUp)
T-V9	if (e is FarUp)		then (C is FarUp)

The conresponding PetrNnet for these rules is shown in Figure 5, and its implementation in programming can be found in *Appendix* A on page 39.



(*a*) *Petnet in horizontal direction* (*b*) *Petnet in vertical direction*

Figure 5. Petrinets for camera movements

3.4 FSMFs for camera tracking

We have designed six sets of functions, four of which deal with FSMFs of the camera tracking process, and the other two are defuzzy functions:

- **FSMFs for errors in X-Y directions:** Fig. 6 is the diagram for these functions and their implementation in programming can be found in *Appendix* A on page 35;

- **FSMFs for rates of errors in X-Y directions:** Fig. 7 is the diagram for these functions and their implementation in programming can be found in *Appendix* A on page 37;
- **defuzzy functions:** Fig. 8 is the diagram for these functions and their implementation in programming can be found in *Appendix* A on page 38.

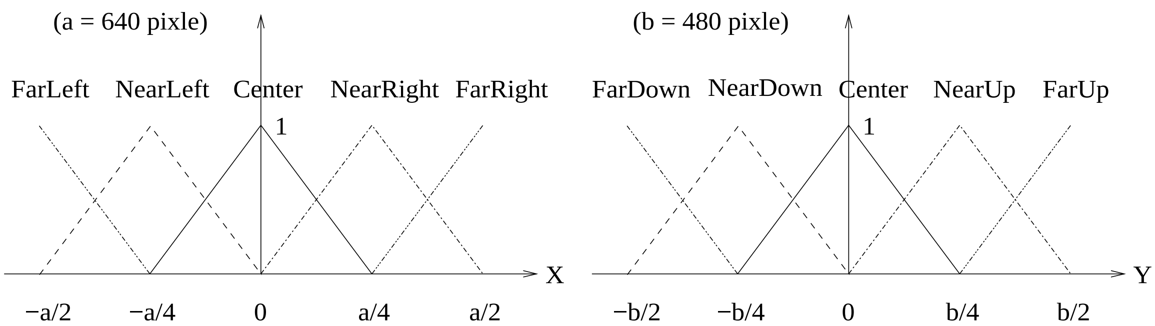


Figure 6. FSMF for the errors in X, Y directions, respectively

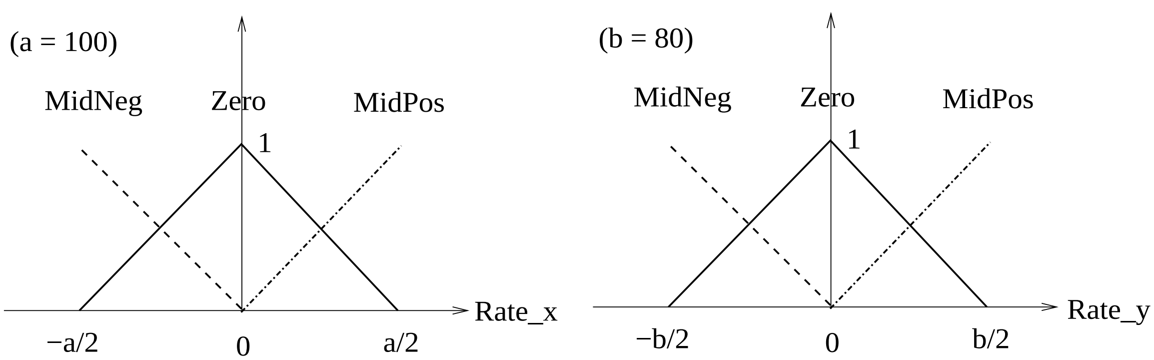


Figure 7. FSMF for the rates of error in X, Y directions, respectively

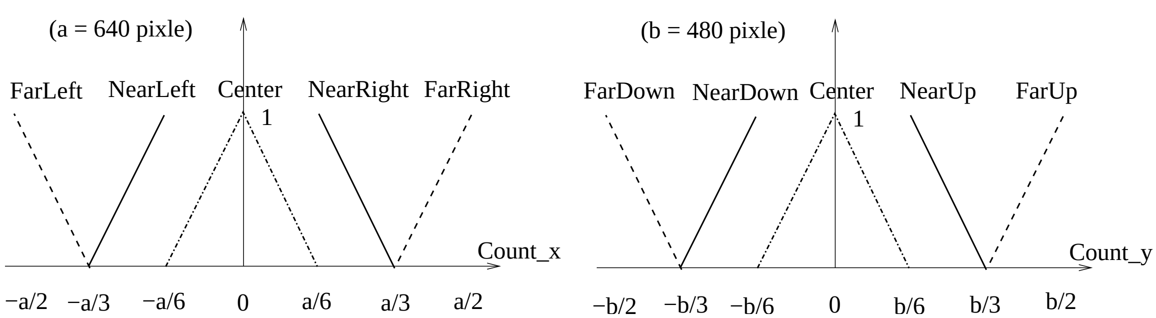


Figure 8. FSMF for the counts the camera will rotate in X, Y directions, respectively

4 Computer Runs

The implementation of camera tracking follows the methodology discussed above. The figure 4 on page 5 gives the main flowchart of programming the camera control process. In addition, progamming in image processing and specifications on how to control the camera (sending commands that the camera can accept) added to the realization of the camera's automatically tracking object in real time. Following is a brief description of the implementation of the tracking process.

- **No input data source needed:** because the camera catches the image (in *pgm* format) in its FoV with the target in it, the system can obtain the necessary data with the result of image processing.
- **Image processing:** the first part of the program (see *Appendix* A on page 23) is used to calculate the **CoM** of the *Target*. Image *thresholding* is involved in this process.
- **FSMFs for CoM's errors in X, Y directions** (corresponding to the Fig. 6 on page 8) are implemented in the program function *errorFuzzy()* (see *Appendix* A on page 35).
- **FSMFs for CoM's rates of errors in X, Y directions** (corresponding to the Fig. 7 on the page 8) are implemented in the program function *rateFuzzy()* (see *appendix* A on page 37).
- **The final fuzzy truth** can be calculated with *PetriNet Method* (corresponding to the Fig. 5 on page 7). The implementation program *PetriNet()* can be found in *Appendix* A on page 39.
- **The defuzzy function** (corresponding to the Fig. 8 on page 8) is implemented in the program function *defuzzy()* (see *Appendix* A on page 38).
- **The camera control** is implemented by transferring the defuzzied values into specific commands that the camera can recognize and take effect. This part of the program is on the *Appendix* A on page 29.

5 Results

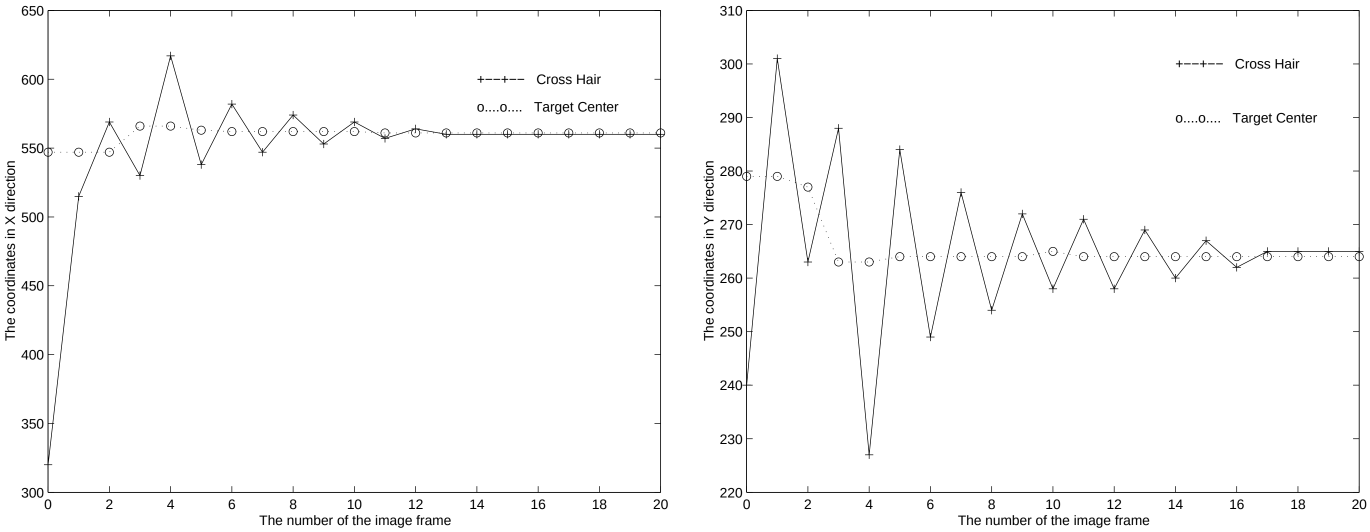
Our camera tracking system was at first aimed to simulate the target tracking process, but during the development we added some project work in the course *Real Time Computing System* so that the tracking system can in real time track the target.

5.1 Results in the Simulation of Camera Tracking

We ran the program twice in simulating the tracking process.

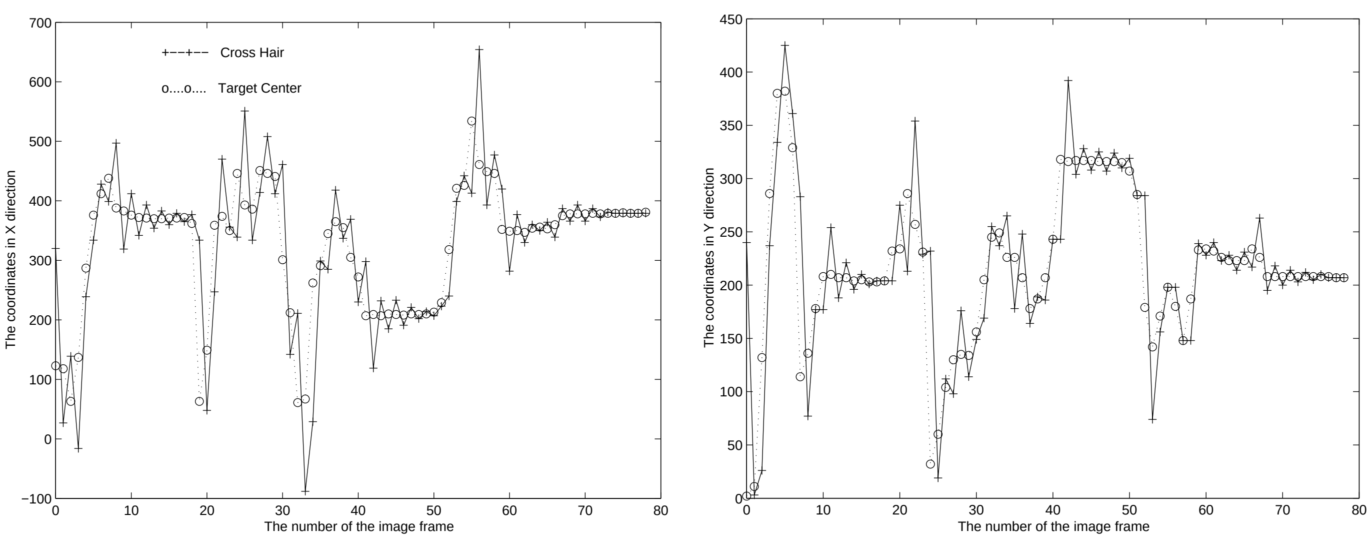
- At first we just allowed the the program to proceed the first 20 frames and recorded these tracking process, the data of which can be found in *Appendix* B.1 on the page 42. We have written down for each frame the target's CoM *coordinate (X, Y)*, cross hair's coordinate (X,Y), and defuzzied value (dX,dY) as an offset to move the cross hair. From the figures Fig. 9 on page 10 we can see how the target moved in X-Y directions and the cross hair followed the target. And the images from Fig. 11 through Fig. 20 are the first 20 frames images the camera recorded. These images can clearly show the tracking process.

- Then on the second simulation process we recorded the tracking process for longer time, allowing **larger variation in the movement of the target.** The resulting data can be found in *Appendix* B.2 on the page 45. From the figures Fig. 10 we can see the movement of the target and how the target moved in X-Y directions and the cross hair followed the target.



(a) *X direction* (b) *Y direction*

Figure 9. The Movements of Target and the Crosshair for images Fig. 11 to Fig. 20



(a) *X direction* (b) *Y direction*

Figure 10. Simulation: target/crosshair's movements

(a) *frame* 1 (b) *frame* 2

Figure 11. Simulation Process of Camera Tracking

(a) *frame* 3 (b) *frame* 4

Figure 12. Simulation Process of Camera Tracking

(a) *frame* 5 (b) *frame* 6

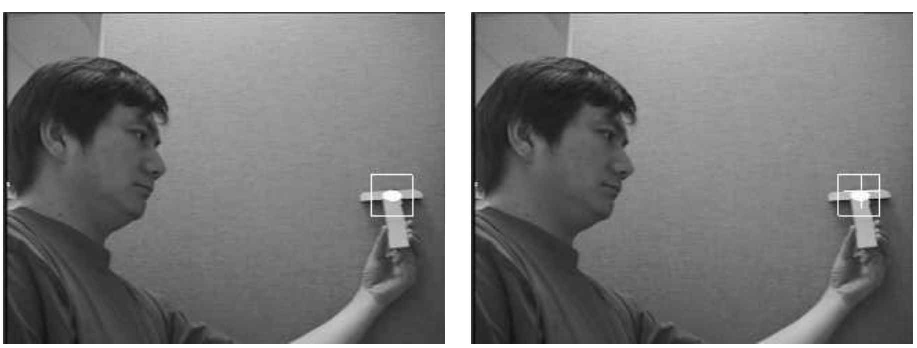
Figure 13. Simulation Process of Camera Tracking

(a) $frame$ 7 (b) $frame$ 8

Figure 14. Simulation Process of Camera Tracking

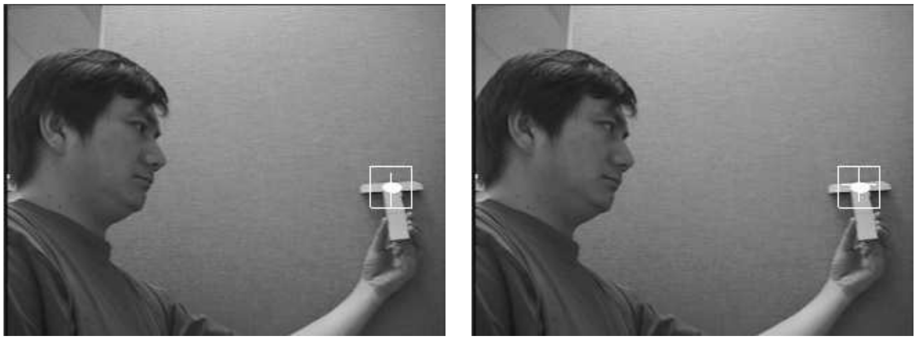
(a) $frame$ 9 (b) $frame$ 10

Figure 15. Simulation Process of Camera Tracking



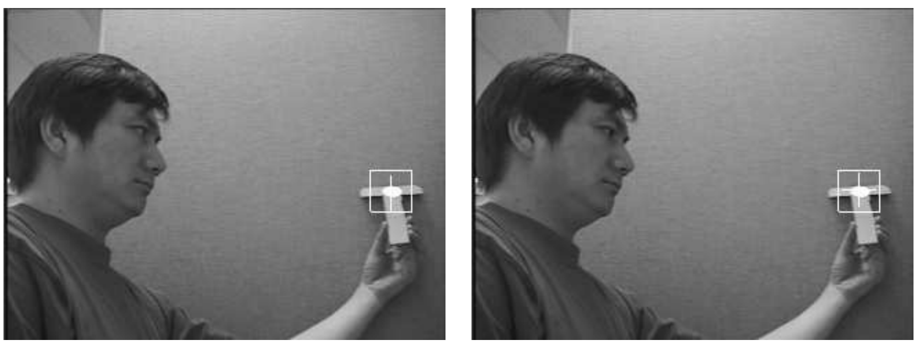
(a) $frame$ 11 (b) $frame$ 12

Figure 16. Simulation Process of Camera Tracking



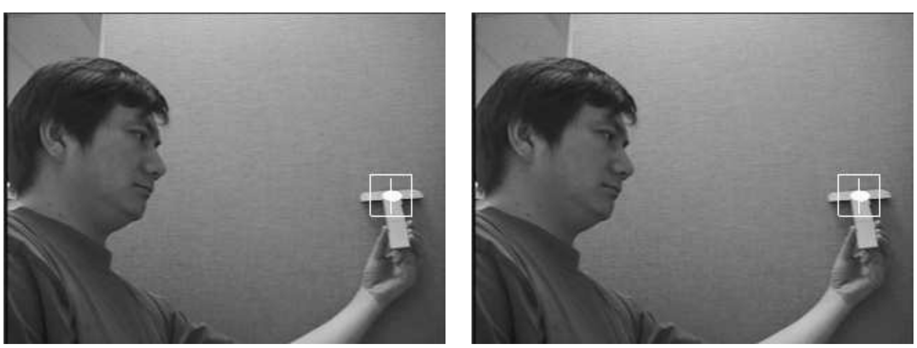
(a) $frame$ 13 (b) $frame$ 14

Figure 17. Simulation Process of Camera Tracking



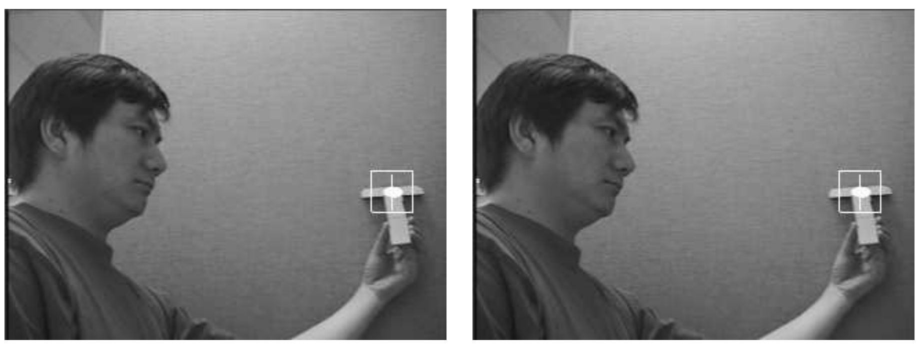
(a) $frame$ 15 (b) $frame$ 16

Figure 18. Simulation Process of Camera Tracking



(a) $frame$ 17 (b) $frame$ 18

Figure 19. Simulation Process of Camera Tracking



(a) $frame$ 19 (b) $frame$ 20

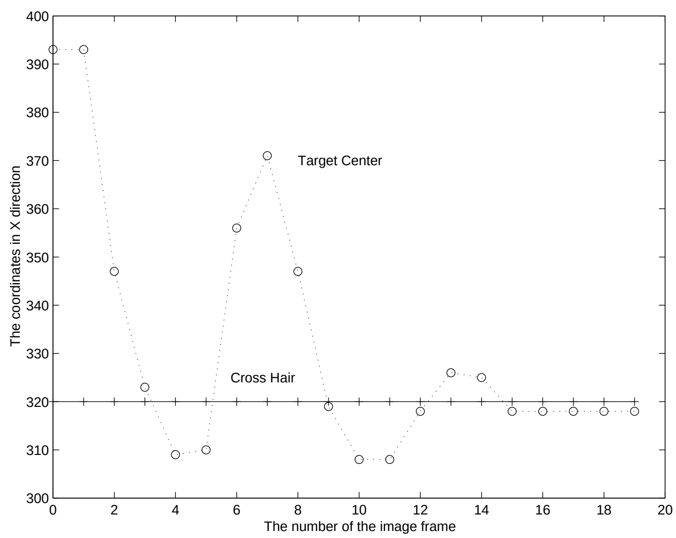
Figure 20. Simulation Process of Camera Tracking

5.2 Results in the Real-time Camera Tracking

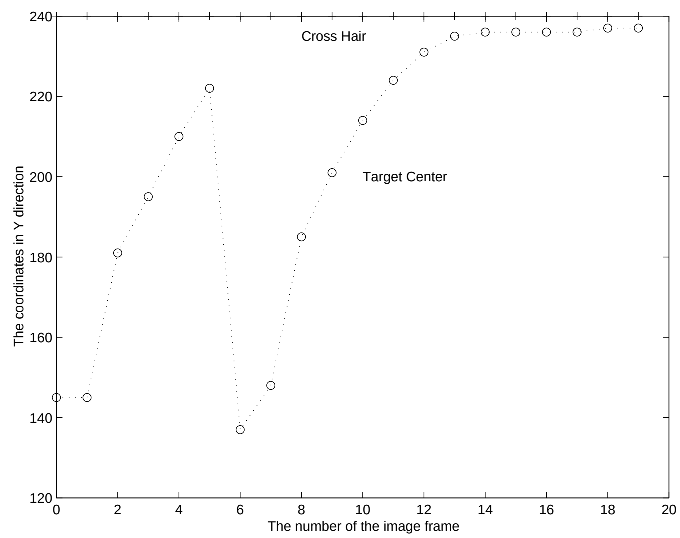
We also ran the program twice in the real-time tracking process.

- Firstly we allowed the the system to proceed the first 20 frames and recorded these tracking process, the data of which can be found in *Appendix* C.1 on the page 46. We have written down for each frame the target's CoM *coordinate (X,Y)*, cross hair's coordinate (X,Y), and defuzzied value (dX,dY) that would be **translated into a set of camera controlling commands to rotate the camera**. From the figures Fig. 21 we can see how the target moved in X-Y directions and the **camera rotated** so that the crosshair could follow the target. And the images from Fig. 23 through Fig. 31 are the first 18 frames of the images the camera recorded. These images can clearly show the tracking process.

- Then on the second real-time tracking process we recorded the process for longer time, allowing **larger variation in the movement of the target.** The resulting data can be found in *Appendix* C.2 on the page 51. From the figures Fig. 22 we can see the movement of the target and how the target moved in X-Y directions and the cross hair followed the target (**because of the camera's rotation**).

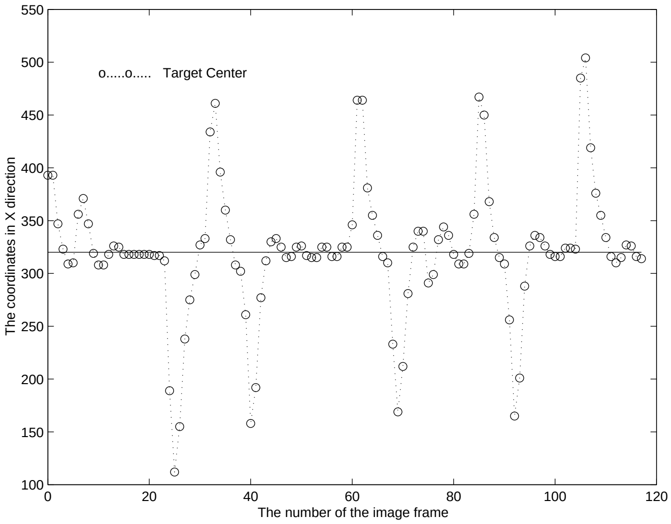


(*a*) *X direction*

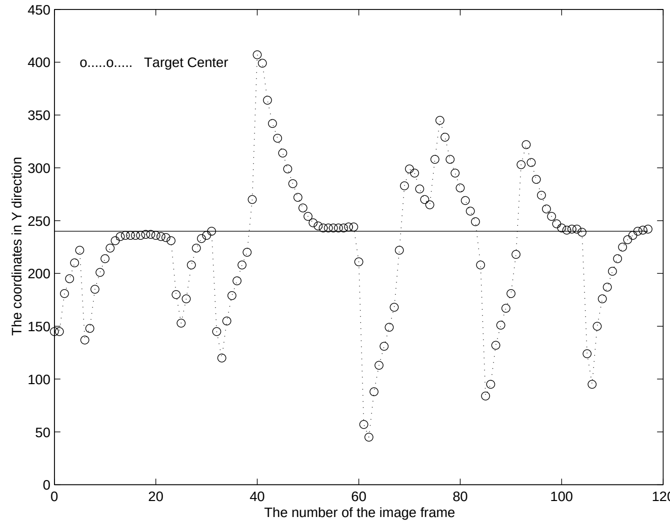


(*b*) *Y direction*

Figure 21. The Movements of target and crosshair for images Fig. 23 to Fig. 31 Please note: there are sudden movements from frame 1 to frame 2 and from frame 5 to frame 6. Because of camera's rotation the cross hair always is at (320,24).

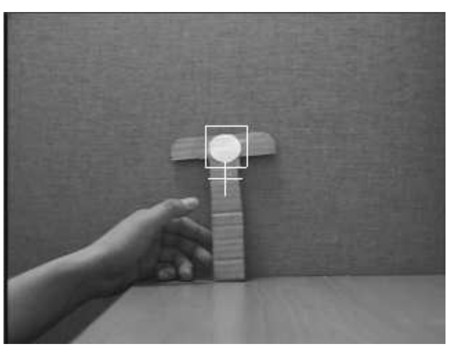


(a) *Y direction*

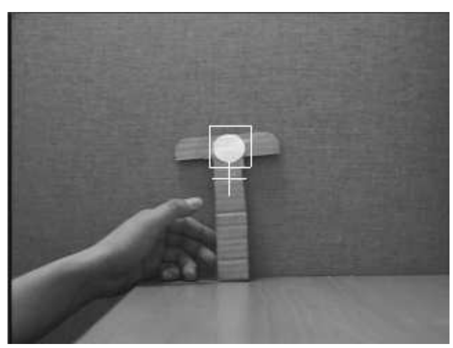


(b) *X direction*

Figure 22. Real time tracking: target/crosshair's movements.Please note: there are sudden target's movements at frames 24, 25, 32, 33, 39, 61, 68, 85.... Because of camera's rotation the cross hair always is at (320,24).

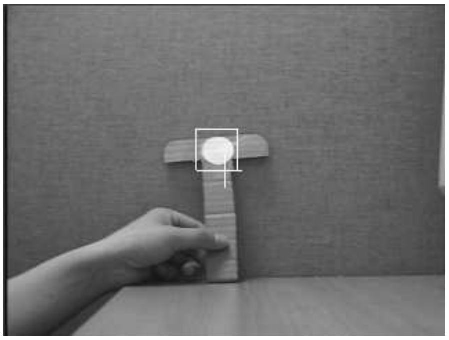


(a) *frame* 3

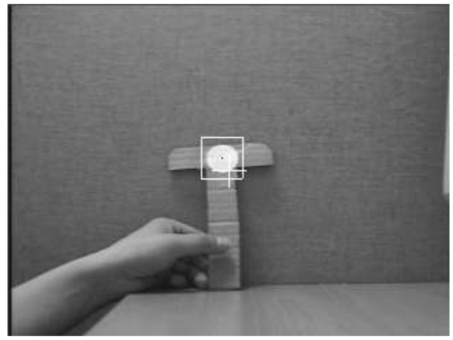


(b) *frame* 4

Figure 23. Real-time Camera Tracking

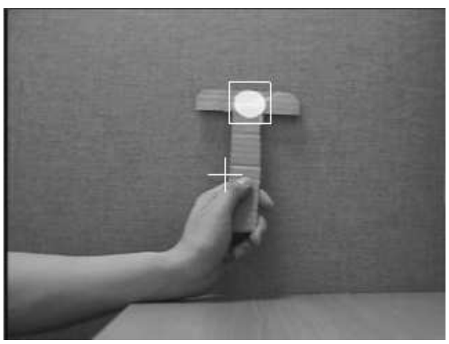


(a) $frame$ 5

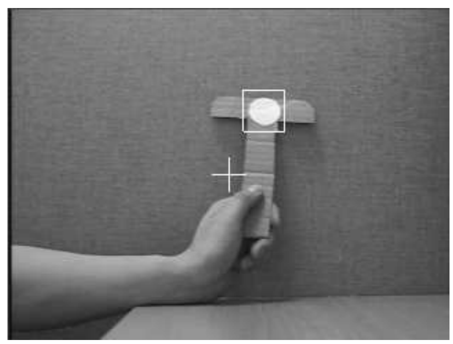


(b) $frame$ 6

Figure 24. Real-time Camera Tracking

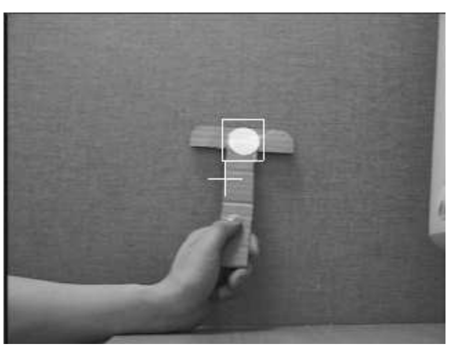


(a) $frame$ 7

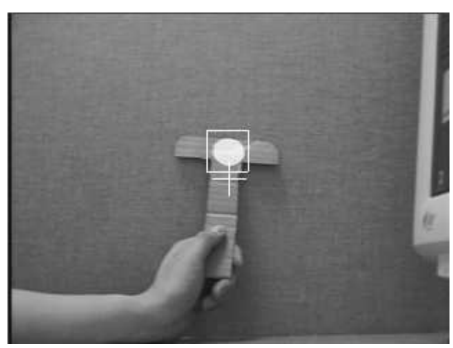


(b) $frame$ 8

Figure 25. Real-time Camera Tracking



(a) $frame$ 9

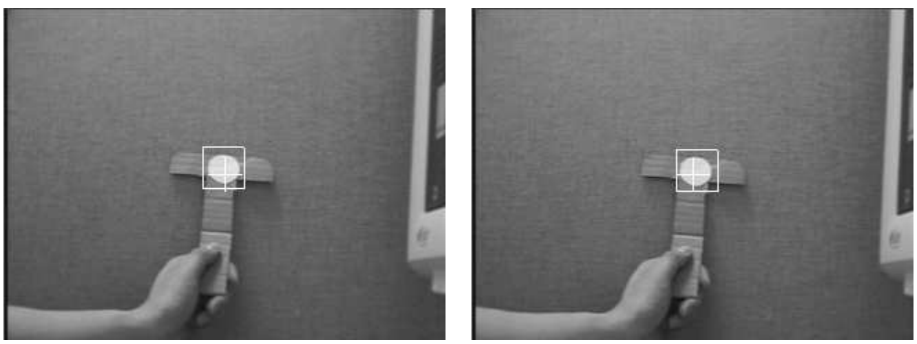


(b) $frame$ 10

Figure 26. Real-time Camera Tracking

(a) $frame$ 11 (b) $frame$ 12

Figure 27. Real-time Camera Tracking



(a) $frame$ 13 (b) $frame$ 14

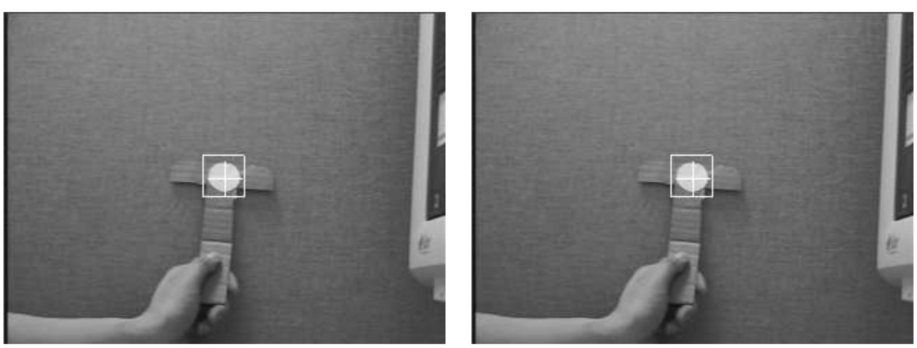
Figure 28. Real-time Camera Tracking

(a) $frame$ 15 (b) $frame$ 16

Figure 29. Real-time Camera Tracking

(a) $frame$ 17 (b) $frame$ 18

Figure 30. Real-time Camera Tracking



(a) $frame$ 19 (b) $frame$ 20

Figure 31. Real-time Camera Tracking

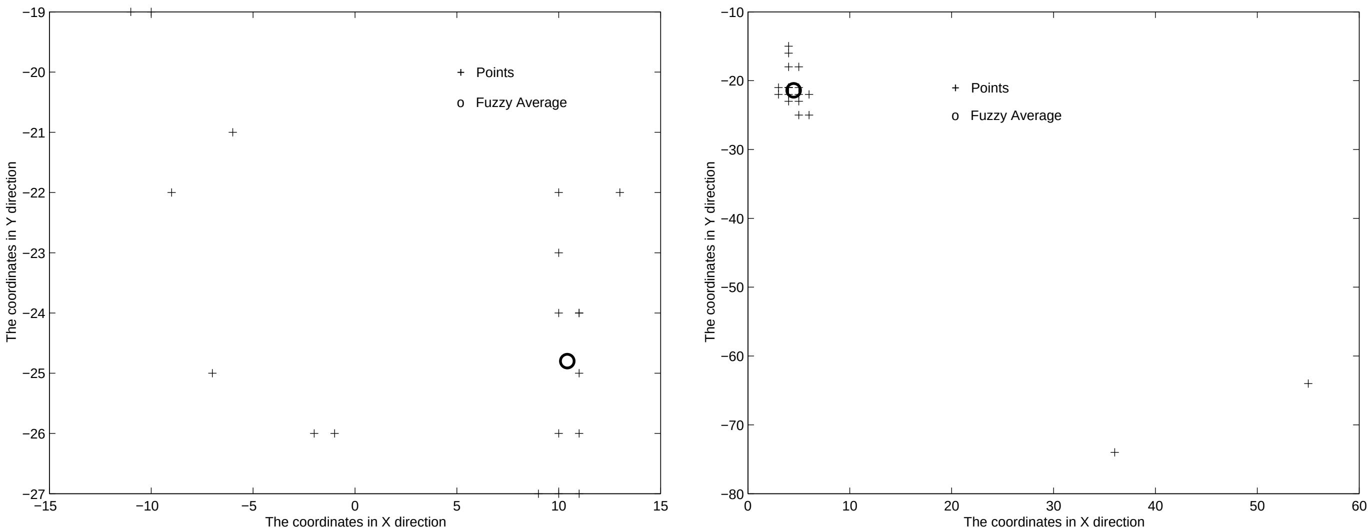
6 Analysis and Conclusions

Our camera tracking system works very well in the lab room under normal lighting conditions. If a strong light appears in the FoV of the camera, the target (a white round plate) can not be recognized and thus not be tracked, but such a problem falls into the image processing field. Our projet's goal is to test whether the designed *fuzzy logic rules and the FSMFs* can work right. The results in the tracking tests have shown that the design philosophy of the camera tracking by using fuzzy logic methods and its implementation can meet the camera tracking scenario both in simulation and real time process. The accuracy in simulation can reach up to 1 pixel: on page 42 in *Appendix* B.1 and page 45 in *Appendix* B.2, when **defuzzy values** (dX,dY) both equal to zero for more than two continuous frames we consider the cross hair center and CoM of the target coincide with each other. While in the real time tracking when the **target center CoM** remains very close (for example, when CoM equals (318,237))to the cross hair center (320,240) for more than two continuous frames, we consider the cross hair center and CoM of the target coincide with each other. From the data on page 46 in *Appendix* C.1 and page 51 in *Appendix* C.2, we can conclude that the accuracy in real time tracking can reach up to 3 pixels. The decrease in the accuracy from simulation to real time tracking lies in the transformation of the defuzzy values to the corresponding camera control commands. A certain number of very small defuzzied values can not take effect in the transformation. Therefore the future improvement on the project can be achieved from two sides:

- **select other kind of FSMFs for the errors, the rates of the errors of target's CoM, and the another defuzzy funtion**, so that when the target's CoM is very close to the cross hair center, the defuzzy value can be relatively large.
- **test the tracking process so that a dynamic transformation factor (from defuzzy value to camera's control commands) can be available in different situations.**

7 Gaze Average Vector Calculation by Using Fuzzy Average Method in Chapter 4

In computer vision research area of eye/gaze tracking, how to calculate the gaze vector is an important issue. Because during a certain period of time (for example 1 - 2 seconds), when a person's eyes look at one spot and we can detect the coordinate of the spot and detect and compute the gaze vectors (the differences between the centers of pupil and glint). Psychological studies have told us that everyone's looks will **unconsciously** drift away even though she/he makes all the efforts to concentrate on one spot. So with the recorded set of gaze vectors, how to get rid of the drift-away looks (outliers or abnormal gaze vectors) is first must-step in the computer vision of gaze-tracking system.
Here we have used the fuzzy average method in chapter 4 ([2]) to solve this problem and obtained very successful results.
Some part of our program, which implemented this fuzzy average application, can be seen in *Appendix* D.1 on page 54, and the input data (9 set of gaze vectors, that is, the data was sampled when each of 9 different spot had been looked at) and the output (**the fuzzy average of each of the 9 sets of gaze vectors**) are show in *Appendix* D.2 on page 60.
Some of the **MWFEVs** (Modified Weighted Fuzzy Expected Value) of the 9 sets of vectors are shown in Fig. 32 below:



(a) for **the 1st gaze vectors** (b) **8th gaze vectors**

Figure 32. Fuzzy Average Calculation for Gaze Vectors (the data is on page 60)

8 Distribution of the Tasks in This Project

Both of us have cooperated throughout the process of the design and implementation of this project. But each of us has put concentration on different tasks:

- **Rong Hu has been mainly responsible for:**
  - designing FSMFs for the camera tracking and the defuzzy function;
  - testing the implementation of FSMFs for the camera tracking and the defuzzy function;
  - writing the section *Methodology* and part of the first 2 sections;
  - drawing all the figures in the first 3 sectons;
  - transforming all the recorded test data into correponding diagrams;
  - programing the fuzzy average function;
  - preparing the references.
- **Beifang Yi has been mainly responsible for:**
  - implementing FSMFs for the camera tracking and the defuzzy function;
  - implementing the fuzzy average function in the gaze vector average calculation;
  - implementing and testing the whole camera tracking process both in simulation and real-time scenario;
  - recording the test data and images and transforming the images into pictures in section *Rusults*;
  - writing the remaining parts of the report and arranging the whole report;
  - editing this report.

References

[1] G. J. Klir and B. Yuan. Fuzzy sets and fuzzy logic theory and applications. *Prentice Hall P T R Upper Saddle River, New Jersey 07458*, 1995.

[2] Carl Looney. Fuzzy systems. http://ultima.cs.unr.edu:80/cs791j/syl791j.html, 2001.

[3] H. T. Nguyen and N. R. Prasad. Fuzzy modeling and contral selected works of m. sugeno. *CRC Press*, 1999.

[4] M. J. Patyra and D. M. Mlynek. Fuzzy logic implementation and applications. *Wiley Teubner*, 1996.

A Camera Tracking Source Codes

```
/*
********************************************************************************
*             CS634  Fuzzy Logic Project---Camera Tracking
*
*            INSTRUCTOR: Dr. Carl Looney
*            AUTHORS:    Rong Hu
*                        Beifang Yi
*            DATE:       Spring 2001
********************************************************************************
*/
#include <sys/ddi.h>
#include <stdio.h>
#include <stdlib.h>
#include <xil.h>
#include <_XilDefines.h>
#include <curses.h>
#include <xil/xil.h>
#include <math.h>

#include <errno.h>
#include <sys/types.h>
#include <string.h>

#define devname "/dev/o1k0" /* Camera Address*/

/** for fuzzy rules ****/
#define XConst 640  /* pixels along X-axle */
#define YConst 480  /* pixels along Y-axle */
#define RateXConst 100
#define RateYConst 80
#define TRUE 1
#define FALSE 0
#define tThreshold 0.01      /* thresholds at transition bars */
#define tFuzzyTruth 1        /* fuzzy truths for the transition bars */
#define fuzzyTokens 13                /* number of tokens */
#define fuzzyBars 9                /* number of transition bars */

void errorFuzzy(int,int,float[]);  /*calculate fuzzy truths for errors of X and Y*/
void rateFuzzy(int,int,float[]);   /*calculate fuzzy truths for rates in directions of X and Y*/
int defuzzy(int,float[]);          /*defuzzy the output fuzzy thuths*/
void PetriNet(float[]);

/***** For Image Processing ******/
```

```
typedef struct Point{
      int x;
      int y;
      int count;
      }POINT;

POINT Search(XilImage ,int, int, int, int,XilSystemState,int);
POINT Center(XilImage ,int ,int ,int ,int,int );
void Draw(XilImage ,int ,int ,int ,int);
void write_to_pgm(XilImage image, char filename[]);
void DrawCrossHair(XilImage ,int ,int);

Xil_boolean error_handler(XilError);

Atom closedownAtom;

float averagePixelsAfterT;
int   LENGTH=31;
int   THRU=100;
float EOfactor=1.1;

/*****    Main Program  ***********/
main()
{
  /**** for image processing and X-Window Displaying  ****/
    XilSystemState state;
    XilImage hardImage, backupImage, display,tempImage, transImage, dstImage;
    XilImage rescaled_image, scaled_image;
    unsigned int width, height, nbands,h,w;
    int winx = 0, winy = 0, xyflag = 0;
    XilDataType datatype;
    Display *xdisplay;
    Window xwindow;
    XEvent event;
    int display_depth;
    XilMemoryStorage storage;
    Xil_boolean status;
    int frames,k;
    int port = 1;
    Xil_boolean error_out = FALSE, found=FALSE;
    int i,j,argumentStr,tempStart,tempEnd;
    POINT target,targetCenter;
    float low1[1]={0.0};
    float high1[1]={200.0};
    float map1[1]={0.0};
    float low2[1]={200.0};
    float high2[1]={255.0};
```

```
  float map2[1]={255.0};
  float max[1], min[1];
  float max1[1],max2[1],max0;
  float blackPixel[1]={0.0}, brightPixel[1]={255.0};
  float imageMatrix[480][640];
  float yHist[480],xHist[640];
  int centerX1=-1,centerX2=-1,centerY1=-1,centerY2=-1;

  /** for fuzzy tracking*****/

  int crossHairX,crossHairY,fileCounter;
  int defuzzyX,defuzzyY;
  int CoM1X,CoM1Y,CoM2X,CoM2Y,targetDX,targetDY;
  int errorRX,errorRY;
  float fuzzyTruthX[13],fuzzyTruthY[13];

  FILE* fData;
  char* imageFile[20]={"image1.pgm","image2.pgm","image3.pgm","image4.pgm","image5.pgm",
            "image6.pgm","image7.pgm","image8.pgm","image9.pgm","image10.pgm",
            "image11.pgm","image12.pgm","image13.pgm","image14.pgm","image15.pgm",
            "image16.pgm","image17.pgm","image18.pgm","image19.pgm","image20.pgm"};


for (h=0;h<height;h++)
   yHist[h]=0.0;
for (w=0;w<width;w++)
   xHist[w]=0.0;


/* system initialization */
state = xil_open();
if (state == NULL) {
  fprintf(stderr, "unable to open xil library\n");
  exit(1);}

if (error_out) xil_install_error_handler(state, error_handler);

/*** Create camera device  ****/
{
  XilDevice device;
  if (!(device = xil_device_create(state, "MMACo1k")))
    {
     fprintf(stderr, "Unable to create a device object\n");
     xil_close(state);
     exit(1);
    }
```

```
  xil_device_set_value(device, "DEVICE_NAME", (void *) devname);
  xil_device_set_value(device, "PORT_V", (void *) port);
  if (!(hardImage = xil_create_from_device(state, "MMACo1k", device))){
    fprintf(stderr, "failed to open MMACo1k device\n");
    xil_close(state);
    exit(1);
  }
  xil_device_destroy(device);
}

   /* end of creating device */

xil_get_info(hardImage, &width, &height, &nbands, &datatype);
backupImage = xil_create_child(hardImage, 0, 0, width, height, 0, 1);

 /*start to prepare for X window display*/
{
 XSizeHints hints;
 int port;
 char *format;
 char titlebar[1024];

 /* xlib window creation */
 xdisplay = XOpenDisplay(NULL);
 if (!xdisplay) {
   fprintf(stderr, "Unable to connect to server\n");
   xil_close(state);
    exit(1);
 }

display_depth = DefaultDepth(xdisplay, DefaultScreen(xdisplay));
xwindow = XCreateSimpleWindow(xdisplay, DefaultRootWindow(xdisplay),
                                   winx, winy, width, height, 0, 0, 0);
if (!xwindow) {
  fprintf(stderr, "Unable to create window\n");
  xil_close(state);
  exit(1);
}
if (xyflag) {
  hints.flags = USPosition;
  hints.x = winx;
  hints.y = winy;
}
else {
  hints.flags = 0;
 }
```

```
 xil_get_device_attribute(hardImage, "FORMAT", (void **) &format);
 xil_get_device_attribute(hardImage, "PORT_V", (void **) &port);
 sprintf(titlebar, "%s: Port %d [%s]",
             (devname ? devname : "/dev/o1k0"), port, format);

     /* if the user tries to shut down the window using the menu */
 if (closedownAtom = XInternAtom(xdisplay, "WM_DELETE_WINDOW", False))
   XSetWMProtocols(xdisplay, xwindow, &closedownAtom, 1);

     /* list window events used */
 XSelectInput(xdisplay, xwindow, ExposureMask | ButtonPressMask);

     /* make the window visible */
 XMapWindow(xdisplay, xwindow);

     /* wait for the window to be mapped (an Expose event) */
 do
  XNextEvent(xdisplay, &event);
 while (event.xany.type != Expose);

 display = xil_create_from_window(state, xdisplay, xwindow);
}
   /*end of X window display */

{
 XilLookup grayramp;
 int num_entries = 256;
 int i;

 Xil_unsigned8 *graydata = malloc(3 * num_entries);
 for (i = 0; i < num_entries; i++) {
  graydata[i * 3 + 2] = graydata[i * 3 + 1] = graydata[i * 3] = i;}
  grayramp = xil_lookup_create(state, XIL_BYTE, XIL_BYTE,
                                     3, num_entries, 0,graydata);
  init_cmap(grayramp, xdisplay, xwindow, 0);
}

 transImage=xil_create(state,width,height,1,XIL_BYTE);
 tempImage   = xil_create(state, width, height, 1, XIL_BYTE);
 dstImage    = xil_create(state, width, height, 1, XIL_BYTE);


/*
 ***************************************************************************
 *  Initilization: for image processing and fuzzy rule setting.
 *                Get the first useful image!
 ***************************************************************************
```

```
*/
 if ((fData=fopen("trackingData.dat","w"))==NULL){
   printf("can't open the file to write the tracking data!\n");
    exit(1);
 }
 fclose(fData);

 crossHairX=320;crossHairY=240;
 for (i=0;i<13;i++){
   fuzzyTruthX[i]=0.0;fuzzyTruthY[i]=0.0;}
 targetCenter.x=0;targetCenter.y=0;

 do {
  xil_toss(tempImage);
  xil_toss(dstImage);
  xil_copy(backupImage, transImage);
  xil_threshold(transImage,tempImage,low1,high1,map1);
  xil_threshold(tempImage,dstImage,low2,high2,map2);
  DrawCrossHair(transImage,crossHairX,crossHairY);
  target=Search(dstImage,0,0,width-1,height-1,state,2);
  if(target.count>78) {
    targetCenter=Center(dstImage,target.x-20,target.y-20,80,80,2);
    targetDX=targetCenter.x;targetDY=targetCenter.y;
    printf("Frame:= %i,target=(%i,%i)\n", frames,targetDX,targetDY);
    Draw(transImage,targetCenter.x-50,targetCenter.y-50,target.count,2);
    CoM1X=targetCenter.x;CoM1Y=targetCenter.y;
    xil_set_pixel(transImage,targetCenter.x, targetCenter.y, &blackPixel[0]);
    if ((CoM1X>0) && (CoM1Y>0)) found=TRUE;
   }
 xil_copy(transImage, display);

}while (found==FALSE);
defuzzyX=0.0;defuzzyY=0.0;fileCounter=0;
 /*
 *******************************************************************************
 *    Tracking Process Begins Here!
 *******************************************************************************
 */

   if ((fData=fopen("trackingData.dat","a"))==NULL){
     printf("can't open the file to write the tracking data!\n");
     exit(1);
    }
   fprintf(fData," Frames---TargetCenter(X,Y)---CrossHair(X,Y)---DefuzzyVaue(dX,dY)---\n");
   for (frames = 0; ; frames++) {
     for (i=0;i<299;i++)
     for(j=0;j<9999;j++);
```

```
   found=FALSE;
   do {
   xil_toss(tempImage);
   xil_toss(dstImage);
   xil_copy(backupImage, transImage);
   xil_threshold(transImage,tempImage,low1,high1,map1);
   xil_threshold(tempImage,dstImage,low2,high2,map2);
   target=Search(dstImage,0,0,width-1,height-1,state,2);
   if(target.count>78) {
     targetCenter=Center(dstImage,target.x-20,target.y-20,80,80,2);
   fprintf(fData,"%7i%9i%9i%9i%9i%8i%8i\n",frames,targetCenter.x,
          targetCenter.y,crossHairX,crossHairY,defuzzyX,defuzzyY);
  printf("Frame:= %i,targetCenter(%i,%i) \n", frames,targetCenter.x,targetCenter.y);
  printf("Old targetCenter(%i,%i)\n",CoM1X,CoM1Y);
  printf("OLd Crosshair (%i;%i)\n",crossHairX,crossHairY);

     CoM2X=targetCenter.x;CoM2Y=targetCenter.y;
     if ((CoM2X>0) && (CoM2Y>0)){
        found=TRUE;
        errorRX=CoM2X-CoM1X;errorRY=CoM2Y-CoM1Y;
        targetDX=CoM2X-crossHairX+320;targetDY=CoM2Y-crossHairY+240;
        errorFuzzy(targetDX,XConst,fuzzyTruthX);
        errorFuzzy(targetDY,YConst,fuzzyTruthY);
        rateFuzzy(errorRX,RateXConst,fuzzyTruthX);
        rateFuzzy(errorRY,RateYConst,fuzzyTruthY);
        PetriNet(fuzzyTruthX);PetriNet(fuzzyTruthY);
defuzzyX=defuzzy(XConst,fuzzyTruthX);
defuzzyY=defuzzy(YConst,fuzzyTruthY);
        printf("---errorFuzzyX,Y=(%i,%i)\n",targetDX,targetDY);
        printf("---rateFuzzyX,Y=(%i,%i)\n",errorRX,errorRY);
        printf("---defuzzy(%i,%i)\n",defuzzyX,defuzzyY);
        crossHairX+=defuzzyX;crossHairY+=defuzzyY;
        printf("---Crosshair (%i;%i)\n",crossHairX,crossHairY);
     xil_set_pixel(transImage,targetCenter.x, targetCenter.y, &blackPixel[0]);
     Draw(transImage,targetCenter.x-50,targetCenter.y-50,target.count,2);
     DrawCrossHair(transImage,crossHairX,crossHairY);
     CoM1X=CoM2X;CoM1Y=CoM2Y;

     /*
     *********************************************************
     *    to record the frame images to keep the tracking
     *       process ONLY IF NEEDED because of time consuming!
     *********************************************************
     */

     /*   if (fileCounter<20){
          write_to_pgm(transImage,imageFile[fileCounter]);
```

```
            fileCounter++;
          }
  */
  for (i=0;i<13;i++){
    fuzzyTruthX[i]=0.0;fuzzyTruthY[i]=0.0;}
  targetCenter.x=0;targetCenter.y=0;
}
if(frames==350) fclose(fData);
    }
  xil_copy(transImage, display);

}while (found==FALSE);
    }

 xil_destroy(display);
 XCloseDisplay(xdisplay);

    return 0;
}


Xil_boolean
error_handler(XilError error)
{
    xil_call_next_error_handler(error);
    fprintf(stderr, "\n***ERROR received; example exiting.\n");
    exit(1);
}


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

 FUNCTION    : write_to_pgm
 PURPOSE     : Output a XilImage into a pgm file. For image debug.
 INPUT       : XilImage image
 OUTPUT      : image is converted into PGM image and saved as filename.

*****************************************************************************************/
void write_to_pgm(XilImage image, char filename[])
{
   unsigned int width,
                height,
                nbands;

   XilDataType datatype;
   FILE *fpt;
```

```
   fpt=fopen(filename, "wb");
   if(!fpt)
      {
       printf("File could not be opened\n");
       exit(1);
      }

   xil_get_info(image,&width,&height,&nbands,&datatype);

   if(nbands == 1)  /* grayscale image */
   {
      unsigned char value;
      int r;
      int c;
      float values;
      fprintf(fpt, "%s\n", "P5");
      fprintf(fpt, "%d %d\n", width, height);
      fprintf(fpt,"%d\n", 255);
      printf("%s\n",filename);

    for( r=0; r<height; r++){
         for( c=0; c<width; c++)
           {
            xil_get_pixel(image,c,r, &values);
            value = (unsigned char) values;
            fprintf(fpt,"%c",value);
           }
      }
      fclose(fpt);
   }
   else
   {
      printf("Only 1-Band Images may be written to PGM format");
   }
}


/****************************************************************************************
   PURPOSE  : Draw a square to mark the tracked pupil or glint.
   INPUT    : XilImage img, (s_x,s_y) is the start coordinate to draw. size is
              the tracked pupil size(pixel number), camera 1, 2 select small or
              big camera.
   OUTPUT   : The bright pupils in img are marked
*****************************************************************************************/
void Draw(XilImage img,int s_x,int s_y,int size,int camera)
{
```

```
    int i,j,col,row;
    int width,height;
    float pixel_val[1]={255.0};

    if(camera==1){ width=10+size;height=10+size;}  /*  small camera  */
    if(camera==2) /* big camera */
      {
        width=60;
        height=60;
        if(s_x>20) s_x=s_x+20;
        if(s_y>20) s_y=s_y+20;
     }
      if(s_x<=0) s_x=1; if(s_y<=0) s_y=1;
      if(s_x+width>=640) width=638-s_x;
      if(s_y+height>=480) height=478-s_y;
      for(col=s_x; col<width+s_x; col++)
         for(row=s_y; row<=height+s_y; row+=height)
       {
          for(i=0;i<2;i++)
          xil_set_pixel(img,col-i, row-i, &pixel_val[0]);
       }

      for(row=s_y; row<height+s_y; row++)
         for(col=s_x; col<=width+s_x; col+=width)
      {
          for(i=0;i<2;i++)
          xil_set_pixel(img,col-i, row-i, &pixel_val[0]);
      }
}


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

   PURPOSE     : Compute the center and size of bright pupil from source image.
   DESCRIPTION : src is the source image handle. (s_x, s_y) is the start
                 coordinate to compute, width and height are the area covered.
   OUTPUT      : return the structure of POINT, which includes pupil center(x,y)
                 and size.

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

POINT Center(XilImage src,int s_x,int s_y,int width,int height,int camera)
{
   int i,j,k=0;
   float *a;
   POINT *Array;
   POINT temp;
```

```
   temp.x=0; temp.y=0,temp.count=0;

   if(s_x<0) s_x=0; if(s_y<0) s_y=0;

   a=(float *)malloc(5*sizeof(float));
   if(camera==1)   Array=(POINT *)malloc((500)*sizeof(POINT));
   else  Array=(POINT *)malloc((10000)*sizeof(POINT));

   if(s_x+width>=640) width=640-s_x;
   if(s_y+height>=480) height=480-s_y;

   for(i=s_x;i<s_x+width;i++)
    for(j=s_y;j<s_y+height;j++)
   {
     a[0]=0;
     xil_get_pixel(src,i,j,a);
     if(a[0]==255)
    {
      Array[k].x=i;
      Array[k].y=j;
      temp.x+=Array[k].x;
      temp.y+=Array[k].y;
      k++;
    }
   }

   if(k!=0){ temp.x=temp.x/k;  temp.y=temp.y/k; temp.count=k; }

      free(a);
      free(Array);
      return temp;
}

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

  PURPOSE   : Search pupil in a child image.
  INPUT     : img is a thresholded XilImage. (x,y) is the start coordinate to search.
              width and height is the end point to search, they are not the actual
              width and height of the image.
  OUTPUT    : Output pupil center and pupil size.

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

POINT Search(XilImage img,int x,int y,int width, int height,XilSystemState state,int camera)
{
    int w,max=0;
```

```
 float pixel_val[1];
 XilHistogram hist;
 unsigned int data[2];
 XilHistogram histogram;

 XilImage child;
 unsigned int ix;
 unsigned int jy;
 /* unsigned int jy1;*/
 unsigned int xWidth;
 unsigned int yHeight;
 unsigned int sBand = 0;
 unsigned int nBands = 1;
 float low2[1]={0.0};
 float high2[1]={255.0};
 unsigned int nbins[1]={2};

 POINT P,P1;

 if(x<0) x=0; if(y<0) y=0;
 P.x=x; P.y=y; P.count=0;

 if(width>640) width=640;
 if(height>480) height=480;
 data[0]=0;data[1]=0;
 if(camera==1){
   xWidth =10;
   yHeight =10; w=20; }
 else{
    w=80;
    xWidth =80;
    yHeight =80; }


for(ix=x; ix<=width-xWidth;ix+=xWidth)
  for(jy=y; jy<=height-yHeight;jy+=yHeight){
    if(ix>width-xWidth) xWidth=width-ix;
    if(jy>height-yHeight) yHeight=height-jy;
    child = xil_create_child(img,ix,jy,xWidth,yHeight,sBand,nBands);
    hist=xil_histogram_create(state,1,&nbins[0],&low2[0],&high2[0]);
    xil_histogram(child,hist,1,1);
    xil_histogram_get_values(hist, &data[0]);
    xil_histogram_destroy(hist);
    if(data[1]>max) {
      max=data[1]; P.x=ix;P.y=jy;P.count=max;}
    xil_toss(child);
 }
```

```
  P1=Center(img,P.x,P.y,w,w,camera);
  return P1;
}


/***** Draw CrossHair at the Center of an Image Frame******/

void DrawCrossHair(XilImage img,int crossX,int crossY)
{
  int i,j,startX,startY,endX,endY;
  int length=50;
  float pixel_val[1]={255.0};

  startX=crossX-length/2;startY=crossY-length/2;
  endX=crossX+length/2;endY=crossY+length/2;
  if (startX<1) startX=1;
  if (startY<1) startY=1;
  if (endX>639) endX=639;
  if (endY>479) endY=479;
  for (i=startX;i<=endX;i++)
    xil_set_pixel(img,i, crossY, &pixel_val[0]);
  for (j=startY;j<=endY;j++)
    xil_set_pixel(img,crossX, j, &pixel_val[0]);
}


/*
*******************************************************************************
* 1. to calculate fuzzy truth for error in X direction, just give X-error for erro
*    and XConst (that is, 640) to replace cnst;
* 2. to calculate fuzzy truth for error in Y direction, just give Y-error for error
*    and YConst (that is, 480) to replace cnst;
* 3. because of using only global array fuzzyTruth[] for processing X-Y errors, thus
*    when the fisrt one (e.g. X-error truth) is calculated, it should be kept in other
*    place for later use.
**********************************************************************************
*/

void errorFuzzy(int error,int cnst,float fuzzyTruth[])
{
  int i;
  float temp;
  error=error-cnst/2;
  temp=error*4.0;
  temp=temp/cnst;
  temp=floor(temp);
  i=(int)temp;
```

```
  switch (i){
  case -2:
fuzzyTruth[0]=((-4.0)/cnst)*(error+cnst/4.0);
fuzzyTruth[1]=(4.0/cnst)*(error+cnst/2.0);
fuzzyTruth[2]=0.0;
fuzzyTruth[3]=0.0;
fuzzyTruth[4]=0.0;
break;
  case -1:
fuzzyTruth[0]=0.0;
fuzzyTruth[1]=((-4.0)/cnst)*error;
fuzzyTruth[2]=(4.0/cnst)*(error+cnst/4.0);
fuzzyTruth[3]=0.0;
fuzzyTruth[4]=0.0;
break;
  case 0:
fuzzyTruth[0]=0.0;
fuzzyTruth[1]=0.0;
fuzzyTruth[2]=((-4.0)/cnst)*(error-cnst/4.0);
fuzzyTruth[3]=(4.0/cnst)*error;
fuzzyTruth[4]=0.0;
break;
  case 1:
fuzzyTruth[0]=0.0;
fuzzyTruth[1]=0.0;
fuzzyTruth[2]=0.0;
fuzzyTruth[3]=((-4.0)/cnst)*(error-cnst/2.0);
fuzzyTruth[4]=(4.0/cnst)*(error-cnst/4.0);
break;
  default:
printf("error when calculating errors.\n");
exit(0);
  }
}

/*
*******************************************************************************
* 1. to calculate fuzzy truth for rate in X direction, just give X-rate for rate
*    and XConst (that is, 640) to replace cnst;
* 2. to calculate fuzzy truth for rate in X direction, just give X-rate for rate
*    and YConst (that is, 480) to replace cnst;
* 3. because of using only global array fuzzyTruth[] for processing X-Y rates, thus
*    when the fisrt one (e.g. X-rate truth) is calculated, it should be kept in other
*    place for later use.
*******************************************************************************
*/
```

```
void rateFuzzy(int rate,int cnst,float fuzzyTruth[])
{
  int i;

  if (rate>0) i=1;
   else
    if (rate==0)
      { fuzzyTruth[6]=1.0;
        fuzzyTruth[5]=0.0;
        fuzzyTruth[7]=0.0;
        return;
      }
    else
     i=-1;
  if (rate<=(-cnst)){
    fuzzyTruth[6]=0.0;
    fuzzyTruth[7]=0.0;
    fuzzyTruth[5]=1.0;
    return;}
if (rate>=cnst){
 fuzzyTruth[7]=1.0;
 return;}

  switch (i){
  case 1:
fuzzyTruth[5]=0.0;
fuzzyTruth[6]=((-1.0)/cnst)*(rate-cnst);
fuzzyTruth[7]=(1.0/cnst)*rate;
break;
  case -1:
fuzzyTruth[5]=((-1.0)/cnst)*rate;
fuzzyTruth[6]=(1.0/cnst)*(rate+cnst);
fuzzyTruth[7]=0.0;
break;
  default:
printf("error when calculating rateFuzzyTruth.\n");
exit(0);
  }
}
/*
 *************************************************************************************
 *  1. according to the PetriNet output, which is stored in the last five elements in
 *     the global array fuzzyTruth[], the defuzzy() function returns the X-Y defuzzy
 *     values (only one for each calling of this function.
 *  2. to defuzzy X fuzzy truths, replace XConst (640) for cnst; for Y, YConst (480)
 *     for cnst.
 ***************************************************************************************
```

```
 */

int defuzzy(int cnst,float fuzzyTruth[])
{

 int flag,value;
 int df1,df2,df3;

 flag=0;

 if (fuzzyTruth[10]==1.0) return 0;
 if (fuzzyTruth[11]>0.0 || fuzzyTruth[12]>0.0) flag=1;
  else flag=-1;
 /*  if (fuzzyTruth[8]>0.0 || fuzzyTruth[9]>0.0) flag=-1;
   else {printf("confused about defuzzying!!\n");
         return -1;
   }
   */
 /*printf("\nflag %i \n",flag);*/

 value=0;
 switch (flag){
 case 1:
  df1=(int)(cnst/6-(cnst*fuzzyTruth[10])/6);
  df2=(int)(cnst/3-(cnst*fuzzyTruth[11])/6);
  df3=(int)(cnst/3+(cnst*fuzzyTruth[12])/6);
  /*  printf("d1=%i, d2=%i, d3=%i \n",df1,df2,df3);*/
  value=(int)(df1*fuzzyTruth[10]+df2*fuzzyTruth[11]+df3*fuzzyTruth[12]);
  /* printf("value=%i \n",value);*/
  value=(int)(value/(fuzzyTruth[10]+fuzzyTruth[11]+fuzzyTruth[12]));
  /* printf("value=%i \n",value);*/
  return value;
  break;
 case -1:
  df1=(int)(-cnst/6+(cnst*fuzzyTruth[10])/6);
  df2=(int)(-cnst/3+(cnst*fuzzyTruth[9])/6);
  df3=(int)(-cnst/3-(cnst*fuzzyTruth[8])/6);
  /*  printf("d1=%i ,d2=%i ,d3=%i \n",df1,df2,df3);*/
  value=(int)(df1*fuzzyTruth[10]+df2*fuzzyTruth[9]+df3*fuzzyTruth[8]);
  /*  printf("value=%i \n",value);*/
  value=(int)(value/(fuzzyTruth[10]+fuzzyTruth[9]+fuzzyTruth[8]));
  /*  printf("value=%i \n",value);*/
return value;
  break;
 }/*end of switch*/

 return value;
```

```
}


/*************************************************************************
* File name:   projectPetNet.c
* Description: This source file implement Petri Net for camera tracking
* Input:      (1) fuzzy truths for conditions
*             (2) condition -> bar ARROW information
*             (3) bar -> condition ARROW information: not necessary
* Output:      Fuzzy truth for each condition in Petri Net
* Author:      RONG HU
* Date:        April 11, 2001
*************************************************************************/

void PetriNet(float fuzzyT[] )
{
  FILE *fp1, *fp2, *fp3; /* to get input information  */
  int  k, l, Chg;                    /* Chg==TRUE: cond.'s fuzzy truth changed after processing */
  /* Chg==FALSE: cond.'s fuzzy truth not changed  */
  float temp;
  float f[fuzzyTokens+1], savf[fuzzyTokens+1];          /* current and precious fuzzy truths for
  float A[fuzzyBars+1][fuzzyTokens+1];                 /* an arrow from condition to transition ba
  /* to deal with one condition to more than one transition */
  /* bars, use matrix instead of vector */
  float a[fuzzyTokens+1];                           /* an arrow form transition bar to condition */
  /*float tThreshold;*/                           /* thresholds at transition bars */
  /*float tFuzzyTruth;*/                           /* fuzzy truths for the transition bars */
  float fmin;                        /* transition fuzzy values from antecedents */
  int loop;

  /* =================  prepare the data  ============================= */


    for ( k = 1; k <= fuzzyTokens; k++ ){
      f[k]=fuzzyT[k-1];      /* get fuzzyTruth from previous procedure */
      /* cout << "Condition C[" <<k <<"] has a fuzzy truth " << f[k] <<"\n";*/
    }

    /* to get "condition ==> bar ARROW" information  */
    for ( l = 1; l <=fuzzyBars; l++)
     for ( k = 1; k <= fuzzyTokens; k++ )
        A[l][k] = -1;

   A[1][1]=1;
   A[2][2]=2;  A[2][6]=2;
   A[3][2]=3;  A[3][7]=3;
```

```
   A[4][2]=4;  A[4][8]=4;
   A[5][3]=5;
   A[6][4]=6;  A[6][6]=6;
   A[7][4]=7;  A[7][7]=7;
   A[8][4]=8;  A[8][8]=8;
   A[9][5]=9;

   /* to get "bar ==> condition ARROW" information */
 a[1]=9; a[2]=9; a[3]=10; a[4]=11; a[5]=11; a[6]=11; a[7]=12; a[8]=13; a[9]=13;

 /* save current token fuzzy truths */
    for(k=1; k<=fuzzyTokens; k++){
      savf[k]= f[k];
    }


    /*cout <<"\n\n================ UPDATE TRANSITION BARS ====================\n";*/
    /* keep processing until no change in fuzzy truth for each condition */
    loop=1;
    do {
      /* cout <<"\nThe " << loop << "th loop:\n ";   */
      for(l=1; l<=fuzzyBars; l++){

/* cout <<"\nl=" <<l<<"  "; */
fmin = 1;             /* set initial minimum fuzzy truth */
for(k=1; k<=fuzzyTokens; k++)
  {
    if (A[l][k] == 1) /* check preset */
      {
if(fmin > f[k])  fmin = f[k]; /* get minimum */
      }
  }

       if (fmin > tFuzzyTruth)  fmin = tFuzzyTruth; /*consider fuzzy truth of rule*/

       if (fmin > tThreshold) /*  fuzzy truth greater than threshold of rule */
 /*  then the rule FIRE*/
  {
    for(k=1; k<=fuzzyTokens; k++){
      if (a[l] == k)   /* check poset */
{
  if (f[k]< fmin) {
    f[k]=fmin; /* update fuzzy truth in poset */
    /*       cout <<"  update fuzzy truth f[" << k<<"]=" << f[k];*/
                 }
}
    }
```

```
  }
      }

      Chg=FALSE;       /* check the change in fuzzy truth for conditions*/
      for(k=1; k<=fuzzyTokens; k++){
if (savf[k] !=f[k]){
  Chg=TRUE;
  savf[k]=f[k];
}
  }
     if (Chg==1) {
       /* cout << "\nChange = TRUE\n"; */
     }
     else /* cout <<"\nChange = FALSE\n"; */

    loop++;

    }while(Chg==TRUE);

    /*Print out */

 for ( k = 1; k <= fuzzyTokens; k++ ){
      fuzzyT[k-1]=f[k];      /* update fuzzyTruth for defuzzy */
      /* cout << "Condition C[" <<k-1 <<"] has a fuzzy truth " << fuzzyT[k-1] <<"\n";*/
    }
}
```

B Data of Camera Tracking (Simulation)

B.1 The Recorded Images' Data (Simulation–20 frames)

Frames---TargetCenter(X,Y)---CrossHair(X,Y)---DefuzzyVaue(dX,dY)---

0	547	279	320	240	0	0
1	547	279	515	301	195	61
2	547	277	569	263	54	-38
3	566	263	530	288	-39	25
4	566	263	617	227	87	-61
5	563	264	538	284	-79	57
6	562	264	582	249	44	-35
7	562	264	547	276	-35	27
8	562	264	574	254	27	-22
9	562	264	553	272	-21	18
10	562	265	569	258	16	-14
11	561	264	557	271	-12	13
12	561	264	564	258	7	-13
13	561	264	560	269	-4	11
14	561	264	560	260	0	-9
15	561	264	560	267	0	7
16	561	264	560	262	0	-5
17	561	264	560	265	0	3
18	561	264	560	265	0	0
19	561	264	560	265	0	0
20	561	264	560	265	0	0

B.2 Data of Camera Tracking (Simulation–upto 133 frames)

```
Frames---TargetCenter(X,Y)---CrossHair(X,Y)---DefuzzyVaue(dX,dY)---
```

0	123	2	320	240	0	0
1	118	11	27	3	-293	-237
2	63	132	139	26	112	23
3	137	286	-16	237	-155	211
4	287	380	239	334	255	97
5	376	382	334	425	95	91
6	412	329	428	361	94	-64
7	438	114	399	283	-29	-78
8	388	136	497	77	98	-206
9	383	178	319	177	-178	100
10	376	208	412	177	93	0
11	372	210	342	254	-70	77
12	371	207	393	188	51	-66
13	370	207	354	221	-39	33
14	370	204	383	196	29	-25
15	371	205	360	210	-23	14
16	371	203	379	201	19	-9
17	372	203	365	204	-14	3
18	362	204	377	204	12	0
19	63	232	334	204	-43	0
20	149	234	48	275	-286	71
21	359	286	247	213	199	-62
22	374	257	470	354	223	141
23	350	231	356	229	-114	-125
24	446	32	339	232	-17	3
25	393	60	551	19	212	-213
26	386	104	334	112	-217	93
27	451	130	414	98	80	-14
28	446	135	508	176	94	78
29	441	134	412	114	-96	-62
30	301	156	461	149	49	35
31	212	205	142	169	-319	20
32	61	245	211	255	69	86
33	67	249	-88	237	-299	-18
34	262	226	29	265	117	28
35	291	226	299	178	270	-87
36	345	207	285	248	-14	70
37	365	178	418	164	133	-84
38	355	187	337	189	-81	25
39	305	207	369	186	32	-3
40	272	243	230	243	-139	57
41	207	318	298	243	68	0

42	209	316	119	392	-179	149
43	207	317	232	304	113	-88
44	210	317	185	328	-47	24
45	209	317	233	308	48	-20
46	208	316	191	325	-42	17
47	210	316	221	307	30	-18
48	209	316	202	324	-19	17
49	210	315	214	310	12	-14
50	213	307	207	319	-7	9
51	229	285	223	284	16	-35
52	318	179	240	284	17	0
53	421	142	399	74	159	-210
54	426	171	442	156	43	82
55	534	198	413	198	-29	42
56	461	180	654	198	241	0
57	449	148	393	148	-261	-50
58	446	187	477	148	84	0
59	352	233	420	239	-57	91
60	349	234	282	228	-138	-11
61	350	232	377	240	95	12
62	347	226	330	223	-47	-17
63	354	223	360	228	30	5
64	356	223	350	214	-10	-14
65	353	223	364	231	14	17
66	360	234	339	217	-25	-14
67	375	226	387	263	48	46
68	378	208	366	195	-21	-68
69	378	208	393	218	27	23
70	378	208	366	200	-27	-18
71	379	208	387	214	21	14
72	378	208	373	203	-14	-11
73	379	208	381	212	8	9
74	379	208	379	205	-2	-7
75	380	208	379	210	0	5
76	379	208	379	207	0	-3
77	379	207	379	207	0	0
78	381	207	379	207	0	0
79	381	207	383	207	4	0
80	382	206	381	207	-2	0
81	382	207	381	207	0	0
82	381	207	381	207	0	0
83	381	207	381	207	0	0
84	383	206	381	207	0	0
85	382	206	385	207	4	0
86	383	205	381	207	-4	0
87	383	206	383	203	2	-4
88	385	206	383	209	0	6

89	385	206	387	204	4	-5
90	384	206	385	207	-2	3
91	385	206	385	207	0	0
92	384	205	385	207	0	0
93	384	206	385	203	0	-4
94	385	205	385	209	0	6
95	382	208	385	201	0	-8
96	314	221	378	218	-7	17
97	288	236	239	226	-139	8
98	287	235	316	255	77	29
99	289	235	267	219	-49	-36
100	289	235	309	247	42	28
101	289	235	274	225	-35	-22
102	288	236	301	243	27	18
103	289	236	278	231	-23	-12
104	289	235	297	240	19	9
105	289	235	283	230	-14	-10
106	289	235	293	239	10	9
107	289	236	286	232	-7	-7
108	289	237	290	240	4	8
109	290	237	290	235	0	-5
110	289	237	290	238	0	3
111	290	237	290	238	0	0
112	290	236	290	238	0	0
113	290	236	290	234	0	-4
114	290	237	290	237	0	3
115	287	243	290	237	0	0
116	286	243	283	255	-7	18
117	285	244	287	233	4	-22
118	286	244	285	254	-2	21
119	285	244	285	236	0	-18
120	286	244	285	250	0	14
121	286	244	285	239	0	-11
122	287	243	285	248	0	9
123	287	244	287	238	2	-10
124	286	244	287	250	0	12
125	286	244	287	239	0	-11
126	286	244	287	248	0	9
127	284	245	287	241	0	-7
128	285	245	280	249	-7	8
129	285	245	288	242	8	-7
130	286	245	284	247	-4	5
131	286	245	286	244	2	-3
132	287	245	286	244	0	0
133	285	246	286	244	0	0

C Data of Camera Tracking (in Real Time)

C.1 The Recorded Images' Data (17 frames)

```
Frames---TargetCenter(X,Y)---CrossHair(X,Y)---DefuzzyVaue(dX,dY)---

      4      309      210      320      240        4     -67
      5      310      222      320      240      -31     -50
      6      356      137      320      240      -18     -32
      7      371      148      320      240       92    -205
      8      347      185      320      240       98     -89
      9      319      201      320      240       47     -75
     10      308      214      320      240        0     -61
     11      308      224      320      240      -35     -44
     12      318      231      320      240      -21     -28
     13      326      235      320      240       -2     -17
     14      325      236      320      240       17      -9
     15      318      236      320      240        8      -7
     16      318      236      320      240       -4      -7
     17      318      236      320      240       -2      -7
     18      318      237      320      240       -2      -7
     19      318      237      320      240       -2      -5
```

C.2 The Camera Real Time Tracking Data (upto 185 frames)

Frames---TargetCenter(X,Y)---CrossHair(X,Y)---DefuzzyVaue(dX,dY)---

0	393	145	320	240	0	0
1	393	145	320	240	151	-189
2	347	181	320	240	100	-89
3	323	195	320	240	47	-78
4	309	210	320	240	4	-67
5	310	222	320	240	-31	-50
6	356	137	320	240	-18	-32
7	371	148	320	240	92	-205
8	347	185	320	240	98	-89
9	319	201	320	240	47	-75
10	308	214	320	240	0	-61
11	308	224	320	240	-35	-44
12	318	231	320	240	-21	-28
13	326	235	320	240	-2	-17
14	325	236	320	240	17	-9
15	318	236	320	240	8	-7
16	318	236	320	240	-4	-7
17	318	236	320	240	-2	-7
18	318	237	320	240	-2	-7
19	318	237	320	240	-2	-5
20	318	236	320	240	-2	-5
21	317	235	320	240	-2	-8
22	317	234	320	240	-4	-10
23	312	231	320	240	-4	-12
24	189	180	320	240	-23	-22
25	112	153	320	240	-261	-122
26	155	176	320	240	-262	-116
27	238	208	320	240	-115	-82
28	275	224	320	240	-86	-52
29	299	233	320	240	-72	-28
30	327	236	320	240	-37	-12
31	333	240	320	240	21	-7
32	434	145	320	240	34	0
33	461	120	320	240	227	-189
34	396	155	320	240	155	-130
35	360	179	320	240	99	-84
36	332	193	320	240	66	-80
37	308	208	320	240	21	-68
38	302	220	320	240	-35	-52
39	261	270	320	240	-42	-35
40	158	407	320	240	-131	76

41	192	399	320	240	-317	207
42	277	364	320	240	-117	126
43	312	342	320	240	-56	86
44	330	328	320	240	-14	89
45	333	314	320	240	30	89
46	325	299	320	240	29	86
47	315	285	320	240	8	78
48	316	272	320	240	-14	67
49	325	262	320	240	-7	52
50	326	254	320	240	14	38
51	317	248	320	240	10	25
52	315	245	320	240	-7	14
53	315	243	320	240	-11	9
54	325	243	320	240	-8	5
55	325	243	320	240	14	5
56	316	243	320	240	8	5
57	316	243	320	240	-11	5
58	325	244	320	240	-7	5
59	325	244	320	240	14	8
60	346	211	320	240	8	7
61	464	57	320	240	70	-73
62	464	45	320	240	287	-201
63	381	88	320	240	114	-180
64	355	113	320	240	72	-98
65	336	131	320	240	58	-92
66	316	149	320	240	29	-89
67	310	168	320	240	-11	-89
68	233	222	320	240	-29	-86
69	169	283	320	240	-175	-32
70	212	299	320	240	-234	95
71	281	295	320	240	-112	95
72	325	280	320	240	-64	75
73	340	270	320	240	14	61
74	340	265	320	240	56	50
75	291	308	320	240	35	42
76	299	345	320	240	-77	129
77	332	329	320	240	-37	144
78	344	308	320	240	35	89
79	336	295	320	240	61	84
80	318	281	320	240	29	75
81	309	269	320	240	-4	62
82	309	259	320	240	-31	48
83	319	249	320	240	-19	33
84	356	208	320	240	0	17
85	467	84	320	240	92	-78
86	450	95	320	240	293	-215
87	368	132	320	240	118	-107

88	334	151	320	240	63	-82
89	315	167	320	240	25	-89
90	309	181	320	240	-14	-86
91	256	218	320	240	-30	-78
92	165	303	320	240	-139	-38
93	201	322	320	240	-293	125
94	288	305	320	240	-116	105
95	326	289	320	240	-44	82
96	336	274	320	240	17	70
97	334	261	320	240	46	54
98	326	254	320	240	25	37
99	318	247	320	240	10	25
100	316	243	320	240	-4	12
101	316	241	320	240	-10	5
102	324	242	320	240	-7	0
103	324	242	320	240	11	4
104	323	239	320	240	7	3
105	485	124	320	240	4	0
106	504	95	320	240	315	-231
107	419	150	320	240	146	-138
108	376	176	320	240	70	-59
109	355	187	320	240	84	-82
110	334	202	320	240	58	-74
111	316	214	320	240	25	-59
112	310	225	320	240	-11	-44
113	315	232	320	240	-29	-27
114	327	236	320	240	-8	-14
115	326	240	320	240	21	-7
116	316	241	320	240	10	0
117	314	242	320	240	-11	0
118	318	242	320	240	-14	4
119	237	215	320	240	-2	3
120	50	195	320	240	-168	-65
121	94	211	320	240	-285	-89
122	177	229	320	240	-153	-48
123	218	238	320	240	-47	-20
124	233	241	320	240	-113	-3
125	270	241	320	240	-110	0
126	302	242	320	240	-77	0
127	327	241	320	240	-32	4
128	335	242	320	240	21	0
129	329	242	320	240	40	4
130	317	242	320	240	16	3
131	319	244	320	240	-7	3
132	449	264	320	240	0	10
133	504	276	320	240	257	63
134	437	273	320	240	223	72

135	376	263	320	240	81	53
136	352	256	320	240	84	39
137	332	251	320	240	54	28
138	317	248	320	240	21	20
139	311	247	320	240	-7	14
140	311	245	320	240	-26	12
141	319	242	320	240	-16	9
142	324	243	320	240	0	3
143	324	243	320	240	11	6
144	317	243	320	240	7	5
145	317	243	320	240	-7	5
146	317	242	320	240	-4	5
147	317	242	320	240	-4	3
148	317	243	320	240	-4	3
149	317	243	320	240	-4	6
150	318	242	320	240	-4	5
151	371	139	320	240	-2	3
152	389	115	320	240	119	-201
153	357	149	320	240	117	-128
154	328	172	320	240	61	-85
155	308	186	320	240	14	-84
156	305	201	320	240	-35	-75
157	314	214	320	240	-32	-61
158	323	225	320	240	-10	-44
159	326	231	320	240	7	-27
160	326	236	320	240	16	-17
161	318	238	320	240	10	-7
162	315	239	320	240	-4	-3
163	314	240	320	240	-13	0
164	318	240	320	240	-10	0
165	324	241	320	240	-2	0
166	316	243	320	240	11	0
167	83	315	320	240	-11	8
168	68	325	320	240	-267	150
169	144	302	320	240	-228	97
170	207	289	320	240	-77	80
171	232	275	320	240	-87	70
172	257	264	320	240	-110	55
173	289	255	320	240	-92	41
174	313	249	320	240	-53	27
175	329	245	320	240	-12	17
176	332	243	320	240	26	9
177	327	244	320	240	27	5
178	323	243	320	240	12	8
179	317	243	320	240	4	5
180	317	243	320	240	-7	5
181	317	243	320	240	-4	5

182	317	243	320	240	-4	5
183	318	243	320	240	-4	5
184	318	243	320	240	-2	5
185	318	243	320	240	-2	5

D Calculating Gaze Vector by Using Fuzzy Average Method

D.1 Codes for Calculating Gaze Fuzzy Average Vector

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Function name: gazeFuzzyAverage(const float x[],const float y[],int N,float sigma)
* Description:   calculate the gaze average vector by using MWDEV (Modified Weighted
*                Fuzzy Expected Value----Fuzzy Average Method.
* Input:         x[],y[]----a set of gaze coordinates sampling during gaze vector
*                           calibration;
*                N---- the number of the gaze coordinates;
*                sigma----gaze vector error allowance: to control the iteration
*                         depth.
* Output:        a structure defined as:
*                   typedef struct FPoint{
*                         float x;
*                         float y;
*                         int count;
*                        } FPOINT;
*                    (x,y)---the fuzzy average of the gaze vector;
*                     count---the number of the iteration.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/


FPOINT gazeFuzzyAverage(const float x[],const float y[],int N,float sigma)
{
     MAT* point;
     VEC* mean, *oldMean, *weight, *eachVar;
     float Var, downSum, up; /* Var=SUM {(x-mx)^2 + (y-my)^2}*/
     int i, iterNum;
     FPOINT xy;
     xy.x=0.0;xy.y=0.0;xy.count=0;
     /*allocate memory*/
     point= m_get(N, 2);
     mean=v_get(2);
     weight=v_get(N);       /*one for each point  */
     eachVar=v_get(N);      /*(x-mx)^2 + (y-my)^2*/
     oldMean=v_get(2);

     for (i=0;i<N;i++){
      point->me[i][0]=x[i];
      point->me[i][1]=y[i];
     }

     for(i=0; i<N;i++){      /*initial mean  ..../N*/
```

```
         mean->ve[0] +=  point->me[i][0]/N;
         mean->ve[1] +=  point->me[i][1]/N;
     }

     printf("Initail mean with outliers (MEANx MEANy)=======> ");
     v_output(mean);

     Var=0;
     for(i=0; i<N;i++){     /* initial variance    ../(N-1)*/
        eachVar->ve[i] = (pow(point->me[i][0]- mean->ve[0],2) + pow(point->me[i][1]- mean->ve[1],2
        Var += eachVar->ve[i];
     }
     printf("Initail variance with outliers = %f ", Var);

      iterNum=0;
      /*==========================================  Iterative*/
     do{
       iterNum++;
       printf("\n\niterNum=  %i\n",iterNum);
       /*  (r) ----  weight*/
        downSum=0;
        for(i=0; i<N;i++)
        downSum += exp( -eachVar->ve[i]/(2*Var) );

for(i=0; i<N;i++){
   up=exp(-eachVar->ve[i]/(2*Var));
   weight->ve[i] = (up/downSum);
}
        printf("weight   ");
        v_output(weight);

oldMean->ve[0] = mean->ve[0];
        oldMean->ve[1] = mean->ve[1];

         mean->ve[0]=0;  mean->ve[1]=0;    /*(r+1)-----  update  mean */
        for(i=0; i<N;i++){
         mean->ve[0] += weight->ve[i]* point->me[i][0];
         mean->ve[1] += weight->ve[i]* point->me[i][1];
         }

          printf("The update Mean");
          printf("%f \n",mean->ve[0]);
          printf("%f \n",mean->ve[1]);

         Var=0;
        for(i=0; i<N;i++){       /*(r+1)----- update Var   NOTE: use  oldMean*/
    eachVar->ve[i] = (pow(point->me[i][0]- oldMean->ve[0],2) + pow(point->me[i][1]- oldMean->ve[1]
```

```
            Var += weight->ve[i]* eachVar->ve[i];
        }

          printf("The update Variance");
          printf("%f \n",Var);
    }while (fabs(mean->ve[0] - oldMean->ve[0])>sigma);
    /* NOTE: should fix the mean and variance  after several iteratives     */

    /*dellocate memory*/
     m_free(point);
     v_free(mean);
     v_free(weight);
     v_free(eachVar);
     v_free(oldMean);
     xy.x=mean->ve[0];
     xy.y=mean->ve[1];
     xy.count=iterNum;
     return xy;
}
```

D.2 Input and Result Data of the Gaze Vectors

```
Sigma=0.05

The Raw Coordinates (X,Y) of the Gaze Vector 1

11.000000    -27.000000
9.000000    -27.000000
10.000000    -27.000000
11.000000    -24.000000
10.000000    -22.000000
10.000000    -23.000000
11.000000    -24.000000
11.000000    -25.000000
10.000000    -24.000000
10.000000    -24.000000
10.000000    -26.000000
11.000000    -26.000000
13.000000    -22.000000
-1.000000    -26.000000
-2.000000    -26.000000
-7.000000    -25.000000
-9.000000    -22.000000
-10.000000    -19.000000
-6.000000    -21.000000
-11.000000    -19.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 1 is:
10.418807    -24.794008
The Number of Iteration for Calculating the Average Point is: 7


The Raw Coordinates (X,Y) of the Gaze Vector 2

5.000000    -27.000000
4.000000    -26.000000
3.000000    -26.000000
3.000000    -25.000000
4.000000    -24.000000
4.000000    -22.000000
4.000000    -26.000000
3.000000    -26.000000
3.000000    -26.000000
2.000000    -26.000000
4.000000    -26.000000
392.000000    143.000000
```

392.000000   143.000000
24.000000   -27.000000
413.000000   153.000000
413.000000   152.000000
413.000000   152.000000
419.000000   162.000000
419.000000   162.000000
420.000000   162.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 2 is:
3.439593   -25.886011
The Number of Iteration for Calculating the Average Point is: 13

The Raw Coordinates (X,Y) of the Gaze Vector 3

-6.000000   -25.000000
-9.000000   -24.000000
-6.000000   -25.000000
-15.000000   -24.000000
-10.000000   -23.000000
-12.000000   -24.000000
-12.000000   -24.000000
-5.000000   -25.000000
-12.000000   -24.000000
-12.000000   -23.000000
-16.000000   -24.000000
-6.000000   -25.000000
322.000000   124.000000
326.000000   121.000000
327.000000   119.000000
327.000000   120.000000
328.000000   119.000000
329.000000   119.000000
326.000000   122.000000
-16.000000   -22.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 3 is:
-11.720153   -23.702183
The Number of Iteration for Calculating the Average Point is: 21

The Raw Coordinates (X,Y) of the Gaze Vector 4

-12.000000   -19.000000
-11.000000   -19.000000
-16.000000   -18.000000

```
-6.000000    -21.000000
-17.000000    -19.000000
-17.000000    -20.000000
-7.000000    -20.000000
-10.000000    -20.000000
-6.000000    -22.000000
-6.000000    -21.000000
-17.000000    -19.000000
-6.000000    -22.000000
-12.000000    -20.000000
-6.000000    -22.000000
-5.000000    -22.000000
-15.000000    -20.000000
-17.000000    -20.000000
-4.000000    -22.000000
-12.000000    -21.000000
-5.000000    -22.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 4 is:
-10.326320    -20.450439
The Number of Iteration for Calculating the Average Point is: 1


The Raw Coordinates (X,Y) of the Gaze Vector 5

5.000000    -24.000000
5.000000    -24.000000
3.000000    -24.000000
4.000000    -25.000000
3.000000    -25.000000
5.000000    -25.000000
4.000000    -25.000000
5.000000    -25.000000
3.000000    -25.000000
5.000000    -25.000000
5.000000    -25.000000
5.000000    -25.000000
3.000000    -23.000000
4.000000    -23.000000
4.000000    -24.000000
5.000000    -25.000000
4.000000    -23.000000
5.000000    -25.000000
5.000000    -25.000000
4.000000    -24.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 5 is:
```

4.309462   -24.457941
The Number of Iteration for Calculating the Average Point is: 1

The Raw Coordinates (X,Y) of the Gaze Vector 6

13.000000   -23.000000
13.000000   -22.000000
14.000000   -23.000000
13.000000   -23.000000
13.000000   -23.000000
13.000000   -23.000000
13.000000   -17.000000
13.000000   -17.000000
13.000000   -16.000000
15.000000   -17.000000
15.000000   -17.000000
15.000000   -17.000000
15.000000   -18.000000
14.000000   -17.000000
14.000000   -17.000000
15.000000   -16.000000
13.000000   -15.000000
13.000000   -16.000000
15.000000   -10.000000
12.000000   -21.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 6 is:
13.698411   -18.412394
The Number of Iteration for Calculating the Average Point is: 1

The Raw Coordinates (X,Y) of the Gaze Vector 7

15.000000   -18.000000
15.000000   -18.000000
15.000000   -18.000000
12.000000   -18.000000
14.000000   -18.000000
2.000000   -20.000000
-15.000000   -16.000000
-16.000000   -16.000000
-5.000000   -17.000000
-16.000000   -16.000000
-14.000000   -14.000000
-9.000000   -16.000000
-5.000000   -17.000000

337.000000   157.000000
-5.000000   -18.000000
-8.000000   -18.000000
1.000000   -19.000000
0.000000   -19.000000
2.000000   -19.000000
10.000000   -19.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 7 is:
-4.958762   -17.521927
The Number of Iteration for Calculating the Average Point is: 43

The Raw Coordinates (X,Y) of the Gaze Vector 8

4.000000   -16.000000
4.000000   -15.000000
4.000000   -18.000000
5.000000   -23.000000
6.000000   -22.000000
4.000000   -21.000000
4.000000   -23.000000
4.000000   -23.000000
3.000000   -22.000000
3.000000   -21.000000
4.000000   -21.000000
4.000000   -22.000000
5.000000   -22.000000
5.000000   -25.000000
36.000000   -74.000000
55.000000   -64.000000
6.000000   -25.000000
4.000000   -21.000000
5.000000   -21.000000
5.000000   -18.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 8 is:
4.498616   -21.398779
The Number of Iteration for Calculating the Average Point is: 5

The Raw Coordinates (X,Y) of the Gaze Vector 9

-17.000000   -18.000000
-14.000000   -18.000000
-18.000000   -17.000000
-17.000000   -18.000000

```
-8.000000    -19.000000
-16.000000    -17.000000
-17.000000    -18.000000
-17.000000    -17.000000
-17.000000    -18.000000
-17.000000    -18.000000
-17.000000    -17.000000
-4.000000    -19.000000
-2.000000    -20.000000
-9.000000    -22.000000
-11.000000    -24.000000
-10.000000    -23.000000
1.000000    -25.000000
1.000000    -25.000000
-3.000000    -25.000000
-8.000000    -24.000000

The FUZZY AVERAGE Coodinate (X,Y) for the Gaze Vector 9 is:
-16.959009    -17.711718
The Number of Iteration for Calculating the Average Point is: 14
```