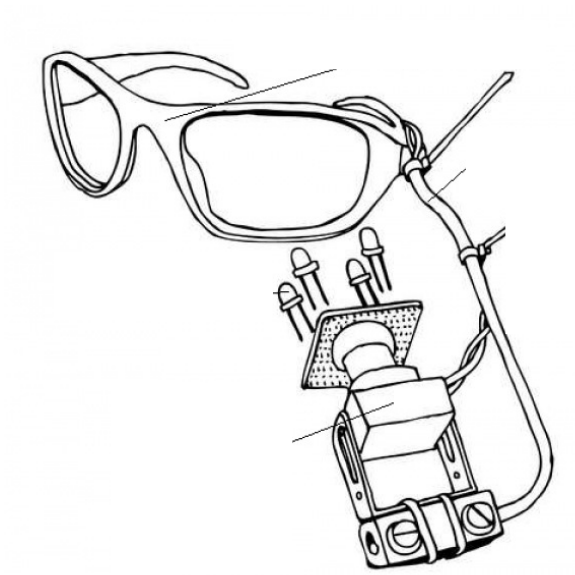
Building an Eye Tracker

SE367 Cognitive Science



Apurva Gupta

Abstract

Gaze tracker is a device which informs about the gaze location of subject in real time. It does so, by tracking the subject's eye movement with respect to head and normalizing head location in space. They are used in experiments on Psychology, Cognitive Science and visual capabilities. Also an emerging use is as an eye-mouse where a standard hand mouse is replaced with head-eye movement controlled interface device. As most popular medium of interaction are keyboard and mouse, modern software interfaces are designed to work with hands. Thus, these interfaces are rendered unusable to people with hand disabilities. In this report we begin by describing various capabilities required in a good tracker and technical problems faced while making one. Then we describe different available designs and their pros and cons. Then we design our own tracker and present results obtained from it.

Brief description of algorithm

A commonly used algorithm used by low cost intrusive eyetrackers is as follows :

1. A head gear with IR Leds and a camera is mounted by the subject
2. In IR light iris appear darker than rest of face as shown in figure below[2]
3. Using image processing iris is separated from rest of face
4. Head location relative to surface in front is determined during calibration procedure
5. Head locations and relative pupil positions are recorded in the calibration session and fed to a training module which approximates a map from eye location to gaze location

The algorithm we have used in **our tracker differs in steps 4 and 5**. Instead of determining head location, we mount another camera on head which captures the scene(S) as is been seen by the user. Then using location of both Iris as obtained in step 3, we mark the gaze location on image S itself. To determine the Iris location to gaze point map , we have used two methods. One relies on training a Neural Network while calibration procedure and other is simple linear interpolation map from eyes space to gaze space.

Previous Work

1. **ITU**

Many open source eye trackers are available with different specialities. For example ITU gaze tracker is an offline system which packs all processing equipment into a backpack allowing the user to carry system anywhere. This system can be utilized to identify gaze patterns in scenarios such as driving, buying in market or painting. Hence it can be used for researchers aiming to classify changes in patterns with learning.

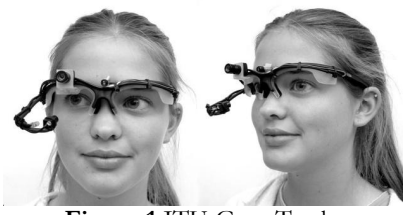


Figure 1 ITU Gaze Tracker

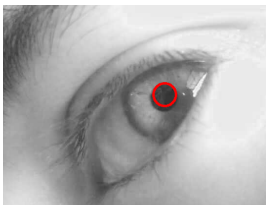


Figure 2 Eyes illuminated with IR light after Iris detection using our own tracker

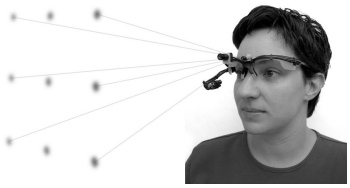


Figure 3 Laser projection on front wall to determine relative head position in ITU's tracker

This intrusive system consists of a head mounted webcam which looks towards the persons eyes(Figure 1) which are illuminated with Infrared Light(Figure 2). A laser pointer projects laser towards screen in front (Figure 3). Another camera captures the reflection of these lasers from front wall. As distortions in these projections are proportional to head angle with respect to surface in front, head location can be determined in space by measuring width, height of each projection.

Possible Designs

1. **One camera** on screen in front looks at the subject and determines gaze location. In this method lot of learning mechanisms are required to determine gaze accurately. [Implemented and tested before project]
2. **Three cameras** : Two for looking at eyes and one records scene visible to user. It has high precision and required low learning but weighs a lot due to 3 cameras
3. Thus we go with **two cameras**, one for eyes and other for scene . This method tool will require less learning and will have manageable weight, easy calibration.

Final Design

Finally we have chosen to implement this design as it is lightweight and hence can be used on children. Also it involves very less calibration and is robust in structure.

1. **Requirements**
    1. 2 webcameras
    2. Safety glasses or Spectacles (to mount eye camera)
    3. Head Phone skeleton (to mount head camera)
    4. Cycle Spoke or thick wire (for tying eye camera at a distance from safety glasses)
    5. Adhesive tape
    6. Strong glue

2. **Construction Procedure**

    Making IR camera from a normal webcam

    Sensors of normal cameras are capable of interpreting IR, but a filter prevents IR light from reaching the sensor. So we need to remove the filter. Open the camera shell and disassemble the lens case. Look out for red film like covering:

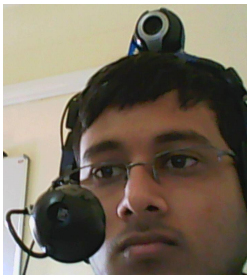
    1. If film is on the lens , scratch it with abrasive sand paper
    2. If film is separate from lens, then break the film

Step 1 : Mounting the head camera

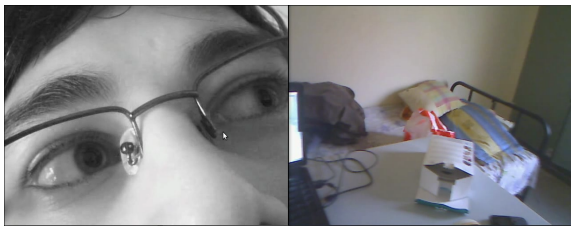
Mount other camera on headphone after removing all wires from the headphone. Keep the camera in a position such that its range of vision is a subset of subject's vision. This camera will look at the scene in front and its image will be tagged with estimated gaze from other camera's images.

Step 2 : Mounting the IR eye camera

Mount the IR capable camera on spectacles as shown in the figure on the right. To keep the camera fixed with respect to spectacles use a thick wire (in our case, we used a cycle spoke as it fits into standard screw threads of Logitech webcams)

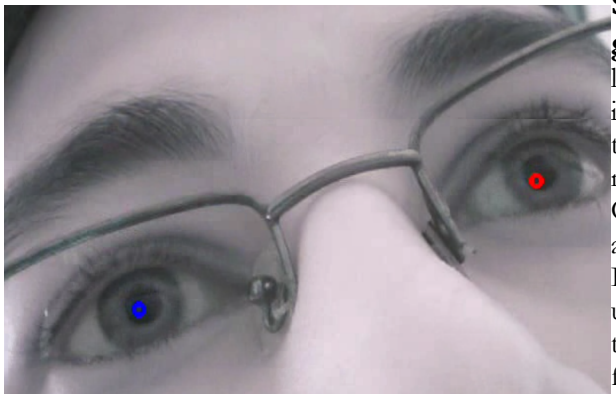


Step 3 : Mounting on head Wear the spectacles and the headphones together. Join both the cameras usb ports to computer. Since we are not using IR lights, the IR camera will work only in daylight, candle light or bulb. It will not work in Flouroscent lights such as CFL and Tubelights. Spectacles can be adjusted such that nose doesnot come into the picture and both eyes are clearly visible. All these adjustmente will be taken care while initial configuration of headgear.



Step 4 : Getting Both camera images Using OpenCV get both the images of camera on screen. As we can see the eye image is tilted. This tilt will be corrected at the time of calibration. As location of eyes is fixed with respect to IR camera, we can crop both the eyes from this image and locate IRIS on it without processing the complete image.

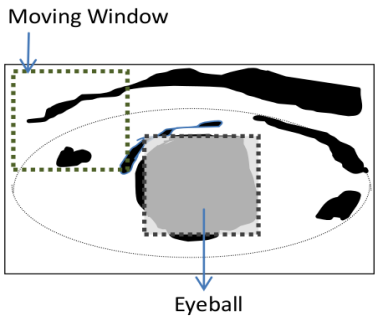
Step 5 : Adaptive Thresholding Crop the eye images from the face image. Apply adaptive thresholding on both eye images to obtain binary image of Iris. Section below explains adaptive thresholding dependent on object size



Step 6 : Marking Iris and calculating gaze Finally when we obtain both the IRIS locations and the image from head camera, next step is to determine gaze location. To do this we can use two different methods. One is to train a neural network to determine the map between iris locations. Other is to make a geometric model of gaze and approximately calculate the gaze location based on it. In our tracker we began by training a neural network using Flood3 libraries. But it resulted in speeds less than realtime due to time lag caused by calling of flood classes many times. Hence we have switched to a geometric head model(explained in section below).

Adaptive Thresholding to obtain Iris

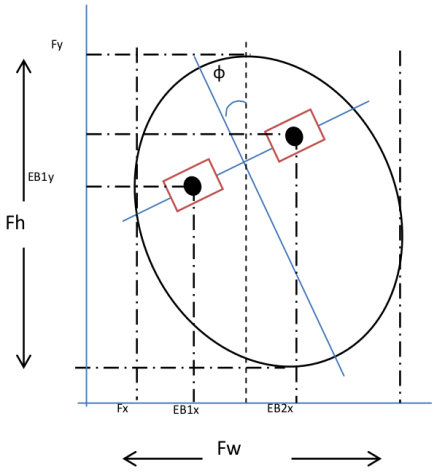
After cropping both eyes, their images $\mathbf{I_E}$ are processed to obtain location of Iris in following steps : Calculate histogram of $\mathbf{I_E}$. Apply threshold on $\mathbf{I_E}$ such that 3% of pixels are below threshold(approximate size of Iris in pixels). Binarize $\mathbf{I_E}$ about this threshold to get $\mathbf{BI_E}$. As Iris is darkest part it always comes below threshold and remains black. In $\mathbf{BI_E}$, some other dark object might come, such as boundary of spectacles. Assuming that such an object will not be of round shape, Iris can be detected using a moving window of size approximately equal to Iris. We use an O(kn) dynamic algorithm for moving window. Construct a black moving window($\mathbf{W}$) of size k placed at (x,y) on $\mathbf{BI_E}$ where k is estimated diameter. Calculate correlation of $\mathbf{W}$ with image pixels of $\mathbf{BI_E}$ as $\mathbf{W}$ is zero everywhere. Store correlation for $(x,y)^{th}$ pixel in another image. Move window from (x,y) to (x+1,y). The correlation change can be calculated by looking at x^{th} column of previous and $x+k+1^{th}$ column of recent window. Determine position with highest correlation. This will be the location of Iris.



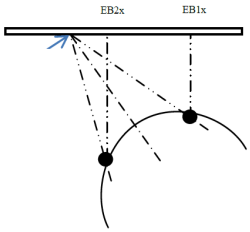
Geometric model for gaze

Let E1 be the location of Iris 1 and E2 be the location of Iris 2. These two should remain parallel to ground when eyes are lookinig forward. While calibrating, we calculate the angle of tilt (w) caused by spectacles and correct all further frames to obtain correct location. Angle of tilt can be obtained as:

$$\omega = \tan^{-1}\frac{(E1.y - E2.y)}{(E1.x - E2.x)}$$

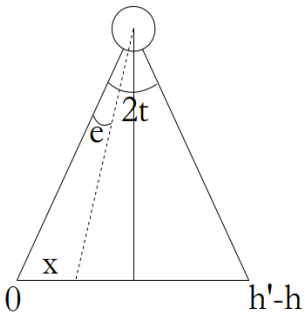


After correcting tilt and obtaining correct Iris locations we interpolate the gaze location. As shown in the following figure as eyeballs rotate in their sockets {(-h,-v),(h,v)}they cover a full horizontal and vertical sweep of image captured from head camera $\mathbf{I}_H$.



Let the horizontal and vertical range covered by eyes be (h,h') and (v,v') respectively in the image and (-t,t) and (-p,p) in rotation angles. And the points of gaze corresponding to these location of eyes cover (g,g') horizontally and (r,r') vertically. Thus, map from eye-angle to eye-pixel space will be related by :

$$H = \frac{(h-h')}{2}, \frac{(H-x)}{H} = \frac{\tan(t-e)}{\tan(t)}$$



Similar formula is for vertical angle too. And point of gaze would be :

$$G = \frac{(g-g')}{2}, \frac{(G - Gaze_x + g)}{G} = \frac{\tan(t-e)}{\tan(t)}$$

Thus, we can determine both the gaze points and mark it on $\mathbf{I}_H$.

Codes for these algorithms can be found attached alongwith appendix. Iris detection code without IR light can be found at **http://home.iitk.ac.in/~apurva/se367/project/codes/eyeball5.c.** This uses one camera and can be used without IR light but has low precision.

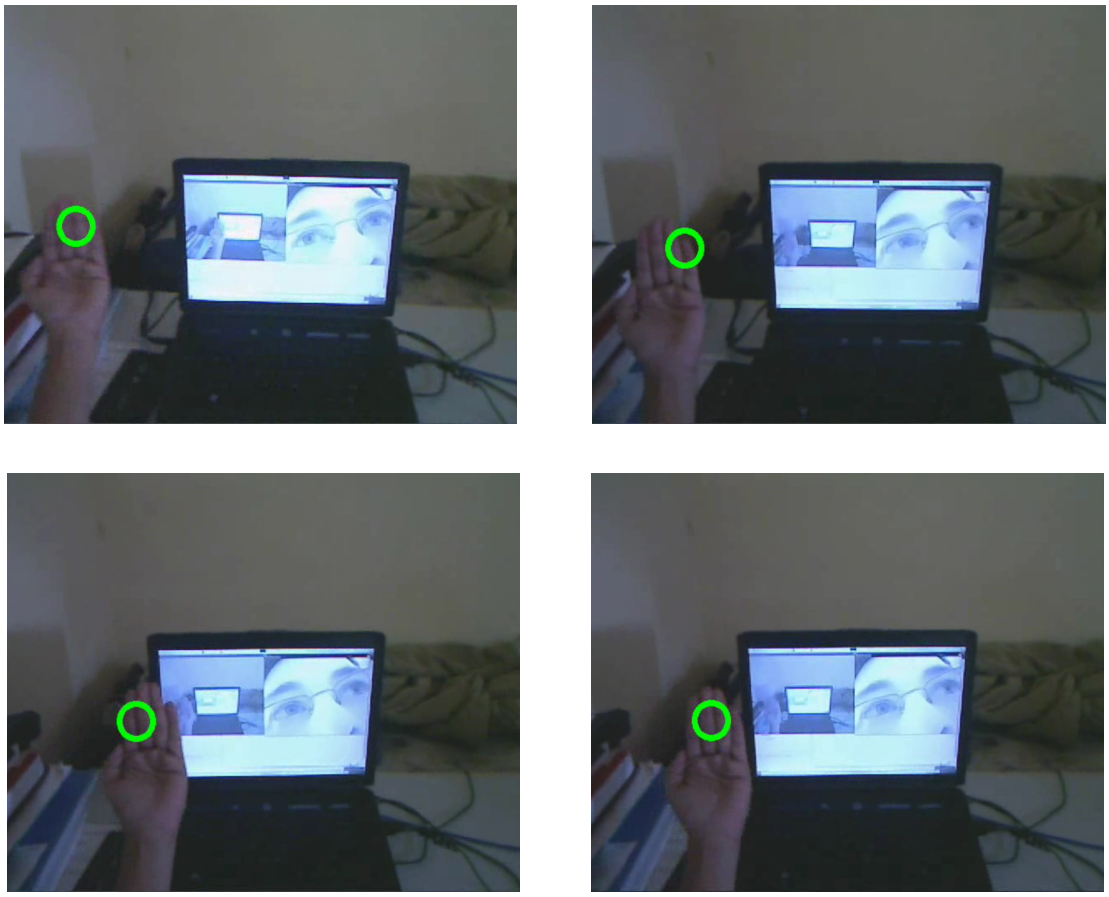
Different considerations

The tracker will work under following conditions:

1. **Offline and Online tracking :** Tracker is capable of working both online(with active cameras) and offline(with recorded videos). Thus, after recording the video, calibration can be improved to obtain better results.
2. **Varying skin tones :** Celtic, European (Light , Dark),Mediterranean,Brown,Black
3. **Different IRIS color :** Amber,Blue,Brown ,Hazel,Gray,Green
4. **Different Light conditions:** Daylight, Dark (with IR only) Ambient light conditions have been tested. As long as a strong IR source is present, ambient light has no effect.
5. **Movement invariance**
    1. No head movement
    2. Vertical head movement
    3. Horizontal head movement
    4. Free head movement

Results

Attached below is a result of tracking while moving a hand done in daylight conditions conducted indoor , with head and eye movement both present on a person with brown eyes.



Result for gaze detection with a moving hand

In the above figure subject looks at the hand while it was moving, and his gaze is marked in the subsequent frames using above described construction and algorithms.

Codes

http://home.iitk.ac.in/~apurva/se367/project/codes/iris_detect.c
http://home.iitk.ac.in/~apurva/se367/project/codes/eyeball5.c

Results

http://home.iitk.ac.in/~apurva/se367/project/videos/head_marked.avi
http://home.iitk.ac.in/~apurva/se367/project/videos/eye1.avi
http://home.iitk.ac.in/~apurva/se367/project/videos/eye2.avi
http://home.iitk.ac.in/~apurva/se367/project/videos/head.avi

References

[1] Babcock, J. S., And Pelz, J. B. 2004. Building a lightweight eyetracking headgear. In Proceedings of the 2004 symposium on Eye tracking research & applications, ACM, San Antonio, Texas, 109-114.

[2] Javier San Agustin, Henrik Skovsgaard , Maria Barret, Martin Tall, Dan Witzner, Evaluation of a Low-Cost Open-Source Gaze Tracker ,Proceedings of the 2010 symposium on Eye tracking research & applications, ACM , Austin TX

[3] Craig Hennessey, Andrew T. Duchowskiy, Expanding the Adoption of Eye-gaze in Everyday Applications, Proceedings of the 2010 symposium on Eye tracking research & applications, ACM , Austin TX

[4] Cohen, A. S. (1983). Informationsaufnahme beim Befahren von Kurven, Psychologie fÃƒÂ¼r die Praxis 2/83, Bulletin der Schweizerischen Stiftung fÃƒÂ¼r Angewandte Psychologie

[5] Marc Eaddy, Gábor Blaskó, Jason Babcock, Steven Feiner, My Own Private Kiosk: Privacy-Preserving Public Displays. 3rd International Semantic Web Conference (ISWC2004)

[6] Arne John Galenstrup, "How is eye-gaze interface control different",
http://www.diku.dk/hjemmesider/ansatte/panic/

[7] David B¨ack , "Neural Network Gaze Tracking using Web Camera" , LiTH-IMT/MI20-EX--05/414—SE, Linköpings universitet

[8] Paul Viola, Michael Jones, Robust Real-time Object Detection, In 2nd international workshop on Statistical and Computational Theories of Vision (2004)

[9] G.R. Bradski, Computer video face tracking for use in a perceptual user interface, Intel Technology Journal, Q2 1998

Image on the title page is courtesy of
http://fastcache.gawkerassets.com/assets/images/4/2009/08/eyewriterv1cartoon.jpg

Appendix 1

Iris detection code in presence of IR light

```
#ifdef _CH_
#pragma package <opencv>
#endif

#include <stdio.h>
#include <stdlib.h>
#include "cv.h"
#include "cvaux.h"
#include "cxcore.h"
#include "highgui.h"
#include "cxmisc.h"
#include "ml.h"
#include <math.h>


int main(int argc, char ** argv)
{
          int threshold_1 = 46.0;
          int threshold_2 = 46.0;
          int end;  end = 0; int key;
          CvCapture * eye_camera = NULL;
          CvCapture * head_camera = NULL;
          IplImage * eye_frame = NULL;
          IplImage * eye_frame_clone = NULL;
          IplImage * eye_frame_clone2 = NULL;
          IplImage * head_frame = NULL;
          IplImage * eye_image1 = NULL;
          IplImage * eye_image2 = NULL;
          IplImage * eye_thresh_1= NULL;
          IplImage * eye_thresh_2= NULL;
          CvRect eye_rect_1;
          CvRect eye_rect_2;
          eye_camera = cvCreateFileCapture("/home/apurva/Output.mpeg");
          head_camera = cvCreateFileCapture("/home/apurva/Output2.mpeg");
```

```
	int frame = 0;
	int eye_min_1x = 19;
	int eye_min_1y = 39;
	int eye_min_2x = 3;
	int eye_min_2y = 6;
	int eye_max_1x = 154;
	int eye_max_1y = 117;
	int eye_max_2x = 136;
	int eye_max_2y = 110;
	CvVideoWriter *writer_h = 0;
	CvVideoWriter *writer_e1 = 0;
	CvVideoWriter *writer_e2 = 0;
	CvVideoWriter *writer_hm = 0;

	writer_h=cvCreateVideoWriter("vid_head.avi",CV_FOURCC_DEFAULT,20,cvSize(640,480),1);
	writer_e1=cvCreateVideoWriter("vid_e1.avi",CV_FOURCC_DEFAULT,20,cvSize(220,120),1);
	writer_e2=cvCreateVideoWriter("vid_e2.avi",CV_FOURCC_DEFAULT,20,cvSize(220,120),1);
	writer_hm=cvCreateVideoWriter("vid_head_marked.avi",CV_FOURCC_DEFAULT,20,cvSize(640,480),1);

	while(!end)
	 {
		//Capture images from camera or from video
		cvGrabFrame (head_camera);
		head_frame = cvRetrieveFrame (head_camera);
		cvGrabFrame (eye_camera);
		eye_frame_clone2 = cvRetrieveFrame (eye_camera);
		eye_frame = cvCloneImage(eye_frame_clone2);
		eye_frame_clone = cvCloneImage(eye_frame_clone2);

		//Create crop rectangles for both eyes
		eye_rect_1 = cvRect(420, 140, 220, 120);
		eye_rect_2 = cvRect(60, 260, 220, 120);

		if(eye_frame)
		{
			printf("%d\n",frame);
			frame = frame+1;
			//Crop eye 1 from eye_camera image
			eye_image1 = cvCreateImage(cvSize(eye_rect_1.width,eye_rect_1.height), eye_frame->depth,eye_frame-
>nChannels);
			cvSetImageROI(eye_frame,eye_rect_1);
			cvCopyImage(eye_frame,eye_image1);

			//Crop eye 2 from eye_camera image
			eye_image2 = cvCreateImage(cvSize(eye_rect_2.width,eye_rect_2.height), eye_frame_clone-
>depth,eye_frame_clone->nChannels);
			cvSetImageROI(eye_frame_clone,eye_rect_2);
			cvCopyImage(eye_frame_clone,eye_image2);

			//Apply threshold on both eye images
			eye_thresh_1 = cvCreateImage(cvSize(eye_image1->width,eye_image1->height),eye_image1->depth,eye_image1-
>nChannels);
			eye_thresh_2 = cvCreateImage(cvSize(eye_image2->width,eye_image2->height),eye_image2->depth,eye_image2-
>nChannels);
			//cvThreshold(eye_image1,eye_thresh_1,threshold_1,255,CV_THRESH_BINARY);
			//cvThreshold(eye_image2,eye_thresh_2,threshold_2,255,CV_THRESH_BINARY);


			// Processing first eye to get iris location
			double eye_1x = 0 ;
			double eye_1y = 0 ;
			int num_thresh = 0;

			for(int i=0;i<eye_image1->height;i++)
			{
				for(int j=0;j<eye_image1->width;j++)
				{
					uchar* ptr1 = &CV_IMAGE_ELEM(eye_image1,uchar,i,j*3);
					  ptr1[0]=ptr1[2];ptr1[1]=ptr1[2];

					  if(ptr1[0]<threshold_1){
						eye_1x = eye_1x+j;
						eye_1y = eye_1y+i;
						num_thresh = num_thresh + 1;
					  }

				}
			}

			if(num_thresh > 0){
				eye_1x = eye_1x/num_thresh;
				eye_1y = eye_1y/num_thresh;
				printf("%d,%d\n",(int)eye_1x,(int)eye_1y);
				cvCircle(eye_image1,cvPoint((int)eye_1x,(int)eye_1y) ,5, CV_RGB(255,0,0), 3, 8, 0);
				cvCircle(eye_frame_clone2,cvPoint((int)eye_1x+eye_rect_1.x,(int)eye_1y+eye_rect_1.y) ,5,
CV_RGB(255,0,0), 3, 8, 0);
				//eye_min_1x = (eye_min_1x>(eye_1x))?eye_min_1x:eye_1x;
				//eye_min_1y = (eye_min_1y>(eye_1y))?eye_min_1y:eye_1y;
			}


			//Processing second eyeball to get iris location

			double eye_2x = 0 ;
			double eye_2y = 0 ;
			int num_thresh2 = 0;

			for(int i=0;i<eye_image2->height;i++)
			{
				for(int j=0;j<eye_image2->width;j++)
				{
					uchar* ptr2 = &CV_IMAGE_ELEM(eye_image2,uchar,i,j*3);
					  ptr2[0]=ptr2[2];ptr2[1]=ptr2[2];
```

```
                                        if(ptr2[0]<threshold_2){
                                                  eye_2x = eye_2x+j;
                                                  eye_2y = eye_2y+i;
                                                  num_thresh2 = num_thresh2 + 1;
                                        }

                                }
                        }

                        if(num_thresh2 > 0){
                                  eye_2x = eye_2x/num_thresh2;
                                  eye_2y = eye_2y/num_thresh2;
                                  printf("%d,%d\n",(int)eye_2x,(int)eye_2y);
                                  cvCircle(eye_image2,cvPoint((int)eye_2x,(int)eye_2y) ,5, CV_RGB(0,0,255), 3, 8, 0);
                                  cvCircle(eye_frame_clone2,cvPoint((int)eye_2x+eye_rect_2.x,(int)eye_2y+eye_rect_2.y) ,5,
CV_RGB(0,0,255), 3, 8, 0);
                                  //eye_min_2x = (eye_min_2x>(eye_2x))?eye_min_2x:eye_2x;
                                  //eye_min_2y = (eye_min_2y>(eye_2y))?eye_min_2y:eye_2y;
                        }

                        int gaze_x1  =  eye_1x*(640/(eye_max_1x-eye_min_1x));
                        int gaze_y1  =  eye_1y*(480/(eye_max_1y-eye_min_1y));

                        int gaze_x2  =  eye_2x*(640/(eye_max_2x-eye_min_2x));
                        int gaze_y2  =  eye_2y*(480/(eye_max_2y-eye_min_2y));

                        int gaze_x = 640-(gaze_x1+gaze_x2)/2;
                        int gaze_y = (gaze_y1+gaze_y2)/2;

                        cvCircle(head_frame,cvPoint(gaze_x,gaze_y) ,20, CV_RGB(0,255,0),5, 8, 0);
                }
                //printf("For eye 1 : %d,%d\n",eye_min_1x,eye_min_1y);
                //printf("For eye 2 : %d,%d\n",eye_min_2x,eye_min_2y);

                cvNamedWindow("eye1",CV_WINDOW_AUTOSIZE );
                cvShowImage("eye1",eye_image1);

                cvNamedWindow("eye2",CV_WINDOW_AUTOSIZE );
                cvShowImage("eye2",eye_image2);

                cvNamedWindow("head",CV_WINDOW_AUTOSIZE );
                cvShowImage("head",head_frame);

                cvNamedWindow("headmark",CV_WINDOW_AUTOSIZE );
                cvShowImage("headmark",eye_frame_clone2);
                 cvWriteFrame(writer_e1, eye_image1);
                 cvWriteFrame(writer_e2, eye_image2);
                 cvWriteFrame(writer_h, head_frame);
                  cvWriteFrame(writer_hm, eye_frame_clone2);
                key = cvWaitKey (5);
                if(key != -1)
                end = 1;
                          //deallocation block
}
cvReleaseVideoWriter(&writer_h);
cvReleaseVideoWriter(&writer_e1);
cvReleaseVideoWriter(&writer_e2);
cvReleaseVideoWriter(&writer_hm);
return 0;
 } //closing main()
```