```cpp
//attempt to make an eyetracker
//Till Handel
//march 2013, GPLv3


//declare included headers
#include <cv.h>
#include <highgui.h>
#include <math.h>
#include <stdio.h>

using namespace cv;
using namespace std;

//global variables
vector<int> mousestatus(4);
//0=nothing,1=lbuttondown,2=lbuttonhold+move,3=lbuttonup - {l1,r1,l2,r2}
vector<bool> onmbutton(5); //{l1,r1,l2,r2,l3}
vector<Point2f> mousepos(8);  //{l1 d,l1 u,r1 d,r1 u,l2 d,l2 u,r2 d,r2 u,l3 d,l3 u}
vector<Point2f> calibmouse(4); //last 3 calibration points clicked in image 3
vector<Point2f> calibleye(4); //last 3 calibration left eye positions
vector<Point2f> calibreye(4); //last 3 calibration right eye positions
Matx33f lmap, rmap; //matrices for coordinate-transformation
vector<Point2f> eyepos(2); //{l,r} - current eye position


//secondary functions

static void onMouse1( int event, int x, int y, int, void* ) {
    switch (event){
        case CV_EVENT_LBUTTONDOWN : {
            mousepos[0] = Point(x,y);
            mousestatus[0] = 1;
            onmbutton[0] = true;
            } break;
        case CV_EVENT_MOUSEMOVE : {
            if(onmbutton[0]) {
                mousepos[1] = Point(x,y);
                mousestatus[0] = 2;
                }
            if(onmbutton[1]) {
                mousepos[3] = Point(x,y);
                mousestatus[1] = 2;
                }} break;
        case CV_EVENT_LBUTTONUP : {
            mousepos[1] = Point(x,y);
            mousestatus[0] = 3;
            onmbutton[0] = false;
            } break;
        case CV_EVENT_RBUTTONDOWN : {
            mousepos[2] = Point(x,y);
            mousestatus[1] = 1;
            onmbutton[1] = true;
            } break;
        case CV_EVENT_RBUTTONUP : {
            mousepos[3] = Point(x,y);
            mousestatus[1] = 3;
            onmbutton[1] = false;
            }
        default : {
            mousestatus[0] = 0;
            mousestatus[1] = 0;
            return;
            }
    }
}


static void onMouse2( int event, int x, int y, int, void* ) {
    switch (event){
        case CV_EVENT_LBUTTONDOWN : {
            mousepos[4] = Point(x,y);
            mousestatus[2] = 1;
            onmbutton[2] = true;
```

```
            } break;
        case CV_EVENT_MOUSEMOVE : {
            if(onmbutton[2]) {
                mousepos[5] = Point(x,y);
                mousestatus[2] = 2;
                }
            if(onmbutton[3]) {
                mousepos[7] = Point(x,y);
                mousestatus[3] = 2;
                }} break;
        case CV_EVENT_LBUTTONUP : {
            mousepos[5] = Point(x,y);
            mousestatus[2] = 3;
            onmbutton[2] = false;
            } break;
        case CV_EVENT_RBUTTONDOWN : {
            mousepos[6] = Point(x,y);
            mousestatus[3] = 1;
            onmbutton[3] = true;
            } break;
        case CV_EVENT_RBUTTONUP : {
            mousepos[7] = Point(x,y);
            mousestatus[3] = 3;
            onmbutton[3] = false;
            }
        default : {
            mousestatus[2] = 0;
            mousestatus[3] = 0;
            return;
            }
    }
}



static void onMouse3( int event, int x, int y, int, void* ) {
    switch (event){
        case CV_EVENT_LBUTTONDOWN : {
            calibmouse[0] = calibmouse[1];
            calibmouse[1] = calibmouse[2];
            calibmouse[2] = calibmouse[3];
            calibleye[0] = calibleye[1];
            calibleye[1] = calibleye[2];
            calibleye[2] = calibleye[3];
            calibreye[0] = calibreye[1];
            calibreye[1] = calibreye[2];
            calibreye[2] = calibreye[3];
            onmbutton[4] = true;
            } break;
        case CV_EVENT_MOUSEMOVE : {
            if(onmbutton[4]) {
                calibmouse[3] = Point(x,y);
                onmbutton[4] = true;
            }} break;
        case CV_EVENT_LBUTTONUP : {
            calibmouse[3] = Point(x,y);
            calibleye[3] = eyepos[0];
            calibreye[3] = eyepos[1];
            onmbutton[4] = false;

            //calculate coordinate transformation
            lmap = getPerspectiveTransform(calibleye,calibmouse);  //maybe use
            'findHomography'
            rmap = getPerspectiveTransform(calibreye,calibmouse);
            } break;
            default : {
                onmbutton[4] = false;
                return;
            }
    }
}

Mat rotateImage(const Mat& source, double angle)
{
```

```
    Point2f src_center(source.cols/2.0F, source.rows/2.0F);
    Mat rot_mat = getRotationMatrix2D(src_center, angle, 1.0);
    Mat dst;
    warpAffine(source, dst, rot_mat, source.size());
    return dst;
}



//main function
int main(int, char**){

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

    //flags

    //what cameras are used??
    bool qcam1 = true; //left
    bool qcam2 = false; //right
    bool qcam3 = true; //middle

    //USB-id's of the cameras
    int l_cam = 1;
    int r_cam = 0;
    int m_cam = 2;

    //rotate right image 180 deg.?
    bool qrot = true;

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



    //declare variables & constants
    Mat im1, im2, im3;
    vector<Vec3f> circles(1);
    vector<double> radius(2);  //measured iris radius




    //video capture code
    VideoCapture cap1(l_cam); // open camera 1
    if(!cap1.isOpened())  // check if it succeeded
        return -1;

    VideoCapture cap2(r_cam); // open camera 2
    if(!cap2.isOpened())  // check if it succeeded
        return -1;

    VideoCapture cap3(m_cam); // open camera 3
    if(!cap3.isOpened())  // check if it succeeded
        return -1;


    if (qcam1) {
        namedWindow("left",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
        setMouseCallback("left",onMouse1,0);
    }
    if (qcam2) {
        namedWindow("right",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
        setMouseCallback("right",onMouse2,0);
    }
    if (qcam3) {
        namedWindow("middle",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
        setMouseCallback("middle",onMouse3,0);
    }



    for(;;){


```

```
        //analysis code - cam1
        if (qcam1){
            cap1 >> im1; //get a new frame from camera 1

            if (mousepos[0] != mousepos[1]) {

                cvtColor(im1, im1, CV_BGR2GRAY);  //convert to b/w

//              threshold(im1,im1,50,255,THRESH_BINARY);
//              adaptiveThreshold(im1,im1,150,ADAPTIVE_THRESH_MEAN_C,THRESH_BINARY,21,0);

                HoughCircles(
                im1.operator()(cvRect(min(mousepos[0].x,mousepos[1].x),min(mousepos[0].y,mousepos[1].y),abs(mousepos[1].x-mousepos[0].x),abs(mousepos[1].y-mousepos[0].y))),  //source image
                circles,  //circlepos & radius
                CV_HOUGH_GRADIENT,  //method
                1,  //dp - inverse ratio of resolution
                abs(mousepos[1].y-mousepos[0].y),  //min dist betw circle centers
                100,  //Canny high threshold
                10,  //circle detection threshold
                radius[0]*0.85,  //min circle radius
                min(radius[0]*1.15,abs(mousepos[1].y-mousepos[0].y)*0.5));  //max circle radius
                if (circles.size() >= 1){
                    Point center(cvRound(circles[0][0])+min(mousepos[0].x,mousepos[1].x), cvRound(circles[0][1])+min(mousepos[0].y,mousepos[1].y));
                    cvtColor(im1, im1, CV_GRAY2BGR);  //convert to color
                    circle( im1, center, cvRound(circles[0][2]), Scalar(0,255,255), 1, 8, 0 );  // draw the circle outline
                    eyepos[0] = center;
                    }


                circle(im1,eyepos[0],3,Scalar(0,0,255),-1,8,0);

                rectangle(im1,mousepos[0],mousepos[1],Scalar(255,0,0),1,8,0);
                if(mousestatus[1]==2){
                    radius[0] = min(abs(mousepos[3].x-mousepos[2].x),abs(mousepos[3].y-mousepos[2].y))/2;
                    circle(im1,Point(min(mousepos[2].x,mousepos[3].x)+radius[0],min(mousepos[2].y,mousepos[3].y)+radius[0]),radius[0],Scalar(0,0,255),1,8,0);
                    }
                }
            imshow("left",im1);
            }


        //analysis code - cam2
        if (qcam2) {
            cap2 >> im2; //get a new frame from camera 2
            if (qrot) {
                im2 = rotateImage(im2,180);
            }

            if (mousepos[4] != mousepos[5]) {

                cvtColor(im2, im2, CV_BGR2GRAY);  //convert to b/w


                HoughCircles(
                im2.operator()(cvRect(min(mousepos[4].x,mousepos[5].x),min(mousepos[4].y,mousepos[5].y),abs(mousepos[5].x-mousepos[4].x),abs(mousepos[5].y-mousepos[4].y))),  //source image
                circles,  //circlepos & radius
                CV_HOUGH_GRADIENT,  //method
                1,  //dp - inverse ratio of resolution
                abs(mousepos[5].y-mousepos[4].y),  //min dist betw circle centers
                100,  //Canny high threshold
                10,  //circle detection threshold
                radius[1]*0.85,  //min circle radius
                min(radius[1]*1.15,abs(mousepos[5].y-mousepos[4].y)*0.5));  //max
```

```
                circle radius
284             if (circles.size() >= 1){
285                 Point center(cvRound(circles[0][0])+min(mousepos[4].x,mousepos[5].x), cvRound(circles[0][1])+min(mousepos[4].y,mousepos[5].y));
286                 cvtColor(im2, im2, CV_GRAY2BGR);  //convert to color
287                 circle( im2, center, cvRound(circles[0][2]), Scalar(0,255,255), 1, 8, 0 );  // draw the circle outline
288                 eyepos[1] = center;
289                 }
290
291
292             circle(im2,eyepos[1],3,Scalar(0,0,255),-1,8,0);
293
294             rectangle(im2,mousepos[4],mousepos[5],Scalar(255,0,0),1,8,0);
295             if(mousestatus[3]==2){
296                 radius[1] = min(abs(mousepos[7].x-mousepos[6].x),abs(mousepos[7].y-mousepos[6].y))/2;
297                 circle(im2,Point(min(mousepos[6].x,mousepos[7].x)+radius[1],min(mousepos[6].y,mousepos[7].y)+radius[1]),radius[1],Scalar(0,0,255),1,8,0);
298                 }
299             }
300         imshow("right", im2);
301         }
302
303
304
305
306
307                                                                                 
        //findContours(frame2,frame2,hierarchy,CV RETR TREE,CV CHAIN APPROX SIMPLE,Point(0,0));
308
309
310
311         if (qcam3) {
312             cap3 >> im3; //get a new frame from camera 3
313
314             if (onmbutton[4]) {
315                 circle(im3,calibmouse[2],10,Scalar(0,255,0),-1,8,0);
316             }
317
318             if(qcam1) {
319                 Matx31f fixp1 = lmap*Matx31f(eyepos[0].x,eyepos[0].y,1);
320                 circle(im3,Point(fixp1(0,0)/fixp1(2,0),fixp1(1,0)/fixp1(2,0)),10,Scalar(255,255,0),-1,8,0);
321             }
322             if(qcam2) {
323                 Matx31f fixp2 = rmap*Matx31f(eyepos[1].x,eyepos[1].y,1);
324                 circle(im3,Point(fixp2(0,0)/fixp2(2,0),fixp2(1,0)/fixp2(2,0)),10,Scalar(0,255,255),-1,8,0);
325             }
326
327             imshow("middle", im3);
328         }
329
330
331
332
333         //termination code////////////////////////////////////
334         if(waitKey(10) >= 0) break;
335         }
336     // the camera will be deinitialized automatically in VideoCapture destructor
337     return 0;
338     }
339
```