

```

1 //attempt to make an eyetracker
2 //Till Handel
3 //march 2013, GPLv3
4
5
6 //declare included headers
7 #include <cv.h>
8 #include <highgui.h>
9 #include <math.h>
10 #include <stdio.h>
11
12 using namespace cv;
13 using namespace std;
14
15 //global variables
16 vector<int> mousestatus(4);
17 //0=nothing,1=lbuttondown,2=lbuttonhold+move,3=lbuttonup - {l1,r1,l2,r2}
18 vector<bool> onmbutton(5); //{l1,r1,l2,r2,l3}
19 vector<Point2f> mousepos(8); //{l1 d,l1 u,r1 d,r1 u,l2 d,l2 u,r2 d,r2 u,l3 d,l3 u}
20 vector<Point2f> calibmouse(4); //last 3 calibration points clicked in image 3
21 vector<Point2f> calibleye(4); //last 3 calibration left eye positions
22 vector<Point2f> calibreye(4); //last 3 calibration right eye positions
23 Matx33f lmap, rmap; //matrices for coordinate-transformation
24 vector<Point2f> eyepos(2); //{l,r} - current eye position
25
26 //secondary functions
27
28 static void onMouse1( int event, int x, int y, int, void* ) {
29     switch (event){
30         case CV_EVENT_LBUTTONDOWN : {
31             mousepos[0] = Point(x,y);
32             mousestatus[0] = 1;
33             onmbutton[0] = true;
34             } break;
35         case CV_EVENT_MOUSEMOVE : {
36             if(onmbutton[0]) {
37                 mousepos[1] = Point(x,y);
38                 mousestatus[0] = 2;
39             }
40             if(onmbutton[1]) {
41                 mousepos[3] = Point(x,y);
42                 mousestatus[1] = 2;
43             } break;
44         case CV_EVENT_LBUTTONUP : {
45             mousepos[1] = Point(x,y);
46             mousestatus[0] = 3;
47             onmbutton[0] = false;
48             } break;
49         case CV_EVENT_RBUTTONDOWN : {
50             mousepos[2] = Point(x,y);
51             mousestatus[1] = 1;
52             onmbutton[1] = true;
53             } break;
54         case CV_EVENT_RBUTTONUP : {
55             mousepos[3] = Point(x,y);
56             mousestatus[1] = 3;
57             onmbutton[1] = false;
58             }
59         default : {
60             mousestatus[0] = 0;
61             mousestatus[1] = 0;
62             return;
63         }
64     }
65 }
66
67
68 static void onMouse2( int event, int x, int y, int, void* ) {
69     switch (event){
70         case CV_EVENT_LBUTTONDOWN : {
71             mousepos[4] = Point(x,y);
72             mousestatus[2] = 1;
73             onmbutton[2] = true;

```

```

74     } break;
75     case CV_EVENT_MOUSEMOVE : {
76         if(onmbutton[2]) {
77             mousepos[5] = Point(x,y);
78             mousestatus[2] = 2;
79         }
80         if(onmbutton[3]) {
81             mousepos[7] = Point(x,y);
82             mousestatus[3] = 2;
83         }} break;
84     case CV_EVENT_LBUTTONDOWN : {
85         mousepos[5] = Point(x,y);
86         mousestatus[2] = 3;
87         onmbutton[2] = false;
88     } break;
89     case CV_EVENT_RBUTTONDOWN : {
90         mousepos[6] = Point(x,y);
91         mousestatus[3] = 1;
92         onmbutton[3] = true;
93     } break;
94     case CV_EVENT_RBUTTONUP : {
95         mousepos[7] = Point(x,y);
96         mousestatus[3] = 3;
97         onmbutton[3] = false;
98     }
99     default : {
100        mousestatus[2] = 0;
101        mousestatus[3] = 0;
102        return;
103    }
104 }
105 }
106
107
108
109 static void onMouse3( int event, int x, int y, int, void* ) {
110     switch (event){
111         case CV_EVENT_LBUTTONDOWN : {
112             calibmouse[0] = calibmouse[1];
113             calibmouse[1] = calibmouse[2];
114             calibmouse[2] = calibmouse[3];
115             calibleye[0] = calibleye[1];
116             calibleye[1] = calibleye[2];
117             calibleye[2] = calibleye[3];
118             calibreye[0] = calibreye[1];
119             calibreye[1] = calibreye[2];
120             calibreye[2] = calibreye[3];
121             onmbutton[4] = true;
122         } break;
123         case CV_EVENT_MOUSEMOVE : {
124             if(onmbutton[4]) {
125                 calibmouse[3] = Point(x,y);
126                 onmbutton[4] = true;
127             }} break;
128         case CV_EVENT_LBUTTONUP : {
129             calibmouse[3] = Point(x,y);
130             calibleye[3] = eyepos[0];
131             calibreye[3] = eyepos[1];
132             onmbutton[4] = false;
133
134             //calculate coordinate transformation
135             lmap = getPerspectiveTransform(calibleye,calibmouse); //maybe use
136             'findHomography'
137             rmap = getPerspectiveTransform(calibreye,calibmouse);
138         } break;
139         default : {
140             onmbutton[4] = false;
141             return;
142         }
143     }
144 }
145 Mat rotateImage(const Mat& source, double angle)
146 {

```

```

147     Point2f src center(source.cols/2.0F, source.rows/2.0F);
148     Mat rot mat = getRotationMatrix2D(src center, angle, 1.0);
149     Mat dst;
150     warpAffine(source, dst, rot mat, source.size());
151     return dst;
152 }
153
154
155 //main function
156 int main(int, char**){
157
158
159 //*****
160
161     //flags
162
163     //what cameras are used??
164     bool qcaml = true; //left
165     bool qcaml2 = false; //right
166     bool qcaml3 = true; //middle
167
168     //USB-id's of the cameras
169     int l cam = 1;
170     int r cam = 0;
171     int m cam = 2;
172
173     //rotate right image 180 deg.?
174     bool qrot = true;
175
176 //*****
177
178
179
180
181     //declare variables & constants
182     Mat im1, im2, im3;
183     vector<Vec3f> circles(1);
184     vector<double> radius(2); //measured iris radius
185
186
187
188
189     //video capture code
190     VideoCapture cap1(l cam); // open camera 1
191     if(!cap1.isOpened()) // check if it succeeded
192         return -1;
193
194     VideoCapture cap2(r cam); // open camera 2
195     if(!cap2.isOpened()) // check if it succeeded
196         return -1;
197
198     VideoCapture cap3(m cam); // open camera 3
199     if(!cap3.isOpened()) // check if it succeeded
200         return -1;
201
202
203     if (qcaml) {
204         namedWindow("left",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
205         setMouseCallback("left",onMouse1,0);
206     }
207     if (qcaml2) {
208         namedWindow("right",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
209         setMouseCallback("right",onMouse2,0);
210     }
211     if (qcaml3) {
212         namedWindow("middle",CV_WINDOW_NORMAL|CV_GUI_EXPANDED);
213         setMouseCallback("middle",onMouse3,0);
214     }
215
216
217
218     for(;;){
219
220

```

```

221 //analysis code - cam1
222 if (qcam1){
223     cap1 >> im1; //get a new frame from camera 1
224
225     if (mousepos[0] != mousepos[1]) {
226         cvtColor(im1, im1, CV_BGR2GRAY); //convert to b/w
227
228         // threshold(im1,im1,50,255,THRESH_BINARY);
229         // adaptiveThreshold(im1,im1,150,ADAPTIVE_THRESH_MEAN_C,THRESH_BINARY,21,0);
230
231         HoughCircles(
232             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
233             circles, //circlepos & radius
234             CV_HOUGH_GRADIENT, //method
235             1, //dp - inverse ratio of resolution
236             abs(mousepos[1].y-mousepos[0].y), //min dist betw circle centers
237             100, //Canny high threshold
238             10, //circle detection threshold
239             radius[0]*0.85, //min circle radius
240             min(radius[0]*1.15,abs(mousepos[1].y-mousepos[0].y)*0.5)); //max circle radius
241         if (circles.size() >= 1){
242             Point center(cvRound(circles[0][0])+min(mousepos[0].x,mousepos[1].x), cvRound(circles[0][1])+min(mousepos[0].y,mousepos[1].y));
243             cvtColor(im1, im1, CV_GRAY2BGR); //convert to color
244             circle( im1, center, cvRound(circles[0][2]), Scalar(0,255,255), 1, 8, 0 ); // draw the circle outline
245             eyepos[0] = center;
246         }
247
248
249
250         circle(im1,eyepos[0],3,Scalar(0,0,255),-1,8,0);
251
252         rectangle(im1,mousepos[0],mousepos[1],Scalar(255,0,0),1,8,0);
253         if(mousestatus[1]==2){
254             radius[0] = min(abs(mousepos[3].x-mousepos[2].x),abs(mousepos[3].y-mousepos[2].y))/2;
255             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);
256         }
257     }
258     imshow("left",im1);
259 }
260
261
262 //analysis code - cam2
263 if (qcam2) {
264     cap2 >> im2; //get a new frame from camera 2
265     if (qrot) {
266         im2 = rotateImage(im2,180);
267     }
268
269     if (mousepos[4] != mousepos[5]) {
270
271         cvtColor(im2, im2, CV_BGR2GRAY); //convert to b/w
272
273
274         HoughCircles(
275             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
276             circles, //circlepos & radius
277             CV_HOUGH_GRADIENT, //method
278             1, //dp - inverse ratio of resolution
279             abs(mousepos[5].y-mousepos[4].y), //min dist betw circle centers
280             100, //Canny high threshold
281             10, //circle detection threshold
282             radius[1]*0.85, //min circle radius
283             min(radius[1]*1.15,abs(mousepos[5].y-mousepos[4].y)*0.5)); //max

```

```

284     circle radius
285     if (circles.size() >= 1){
286         Point center(cvRound(circles[0][0])+min(mousepos[4].x,mousepos[
287             5].x), cvRound(circles[0][1])+min(mousepos[4].y,mousepos[5].y));
288         cvtColor(im2, im2, CV_GRAY2BGR); //convert to color
289         circle( im2, center, cvRound(circles[0][2]), Scalar(0,255,255),
290             1, 8, 0 ); // draw the circle outline
291         eyepos[1] = center;
292     }
293
294     circle(im2,eyepos[1],3,Scalar(0,0,255),-1,8,0);
295
296     rectangle(im2,mousepos[4],mousepos[5],Scalar(255,0,0),1,8,0);
297     if(mousestatus[3]==2){
298         radius[1] = min(abs(mousepos[7].x-mousepos[6].x),abs(mousepos[7
299             ].y-mousepos[6].y))/2;
300         circle(im2,Point(min(mousepos[6].x,mousepos[7].x)+radius[1],min
301             (mousepos[6].y,mousepos[7].y)+radius[1]),radius[1],Scalar(0,0,
302             255),1,8,0);
303     }
304 }
305 imshow("right", im2);
306 }
307
308 //findContours(frame2,frame2,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE,P
309 oint(0,0));
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
321         circle(im3,Point(fixp1(0,0)/fixp1(2,0),fixp1(1,0)/fixp1(2,0)),
322             10,Scalar(255,255,0),-1,8,0);
323     }
324     if(qcam2) {
325         Matx31f fixp2 = rmap*Matx31f(eyepos[1].x,eyepos[1].y,1);
326
327         circle(im3,Point(fixp2(0,0)/fixp2(2,0),fixp2(1,0)/fixp2(2,0)),
328             10,Scalar(0,255,255),-1,8,0);
329     }
330
331     imshow("middle", im3);
332 }
333
334 //termination code////////////////////////////////////
335 if(waitKey(10) >= 0) break;
336 }
337 // the camera will be deinitialized automatically in VideoCapture destructor
338 return 0;
339 }

```