

Eye Tracking

SIGB Spring 2014

Marcus Gregersen
mabg@itu.dk

Martin Faartoft
mlfa@itu.dk

Mads Westi
mwek@itu.dk

March 26th 2014
IT University of Copenhagen

1 Introduction

In the following, we will experiment with, and discuss different approaches to detecting major eye features. Figure 1 gives the names of the eye features that are used throughout this report.

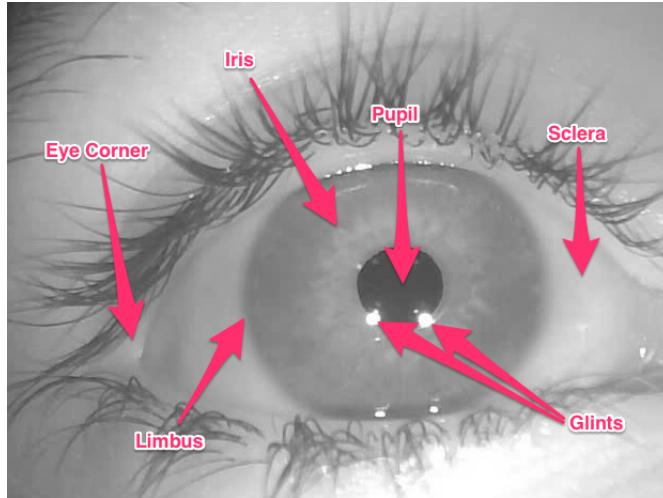


Figure 1: Names and positions of major eye features

2 Pupil Detection

In this section, we will investigate and compare different techniques for pupil detection.

2.1 Thresholding

An obvious first choice of technique, is using a simple threshold to find the pupil, then do connected component (blob) analysis, and finally fit an ellipse on the most promising blobs.

Figure 2 shows an example of an image from the 'eye1.avi' sequence and the binary image produced by, using a threshold that blacks out all pixels with intensities above 93. This manages to separate the pupil nicely from the iris.

The next step, is to do connected component analysis, and fit an ellipsis through the blobs. As seen in Figure 3, this successfully detects the pupil, but is extremely prone to false positives.

By experimenting, we find that requiring that the area of the blob lies in the interval [1000 : 10000], and the extent between [0.4 : 1.0], we eliminate most false positives on the entire eye1 sequence, while still keeping the true positive.

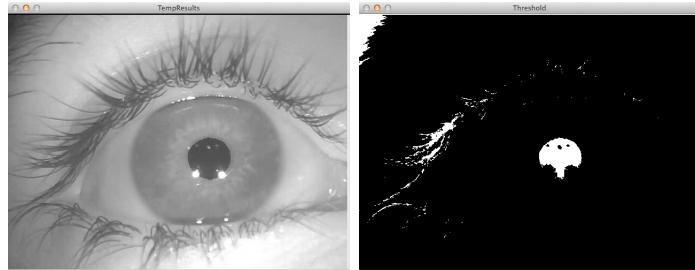


Figure 2: Thresholding eye1.avi

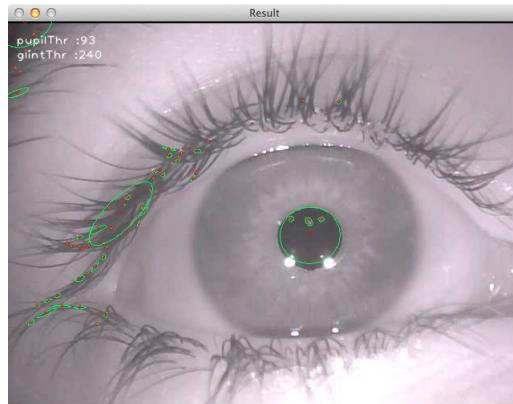


Figure 3: Fitting ellipses on blobs from eye1.avi (green figures are ellipses fitted through blobs, red dots are the centerpoint of each blob)

This approach has several problems, however. Note how the true positive on Figure 3 fails to follow the bottom of pupil correctly. This is due to the glints obscuring part of the boundary between pupil and iris. It also makes some sweeping assumptions:

The pupil has size at least size 1000 If the person on the sequence leans back slightly, the pupil will shrink and we will fail to detect it.

A threshold of 93 will cleanly separate pupil from iris This is true for eye1.avi, but generalizes extremely poorly to the other sequences. If this approach is to be used across multiple sequences recorded in different lighting conditions, the threshold will have to be adjusted by hand for each one.

This problem can be mitigated somewhat with Histogram Equalization. A threshold of 25 on Histogram Equalized images, fares considerably better across several sequences. Note that this will still fail, if parts of the image are significantly darker than the pupil, thereby messing up the equalization.

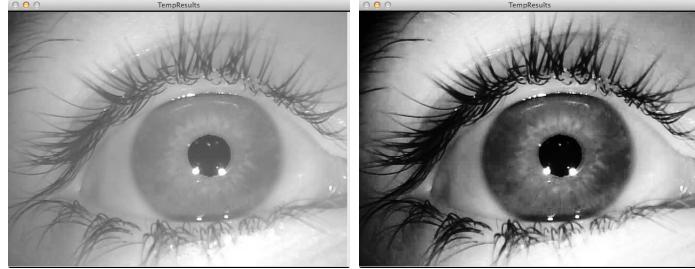


Figure 4: Eye1 before and after Histogram Equalization

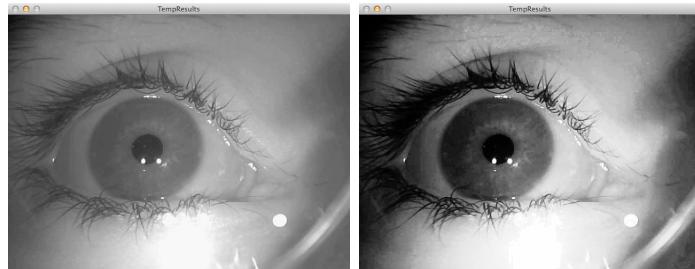


Figure 5: Eye3 before and after Histogram Equalization

Morphology Using Morphology, we can improve the detected pupil. The problem with the glints obscuring part of the boundary can be mitigated with the 'closing' operator - used to fill in holes in binary images. Figure 6 shows binary images before and after applying the closing operator. Notice how the noise inside the pupil is completely removed, and the glints are mostly removed. A downside to using the closing operation, is that adjacent, sparse structures may merge and resemble circles.



Figure 6: Eye1 before and after Closing (5 iterations, 5x5 CROSS structuring element)

Tracking The pupil tracker can be further improved, by using information about the pupil positions from the previous frame. We do it as follows:

1. Search within some threshold distance from each pupil in previous frame

2. One or more pupils were found within the distance, return those
3. No pupils were found within the distance, search the entire image

Because of the fallback clause in '3', it is very unlikely that the true positive is not detected in each frame. The only case where this approach fails, is if the pupil is obscured for a frame (subject blinking for example), while a false positive is still detected. In that case, the pupil will be improperly detected for as long as the false positive continues to be present.

2.2 Pupil Detection using k-means

A method to enhance the BLOB detection of pupil detection is k-means clustering. The method separates the picture in K clusters. Each cluster is a set of pixels, which have values closer to the cluster center, than to other cluster centres - a cluster center corresponds to mean value of the pixels in cluster. The value of K is arbitrarily chosen, so that for a sufficiently large number of K the pupil is evaluated as a single separate cluster. If the pupil is a single cluster a binary image can easily be created and BLOB detection would only need to look at the one object. The following figure illustrates how different values of K impacts the segmentation.

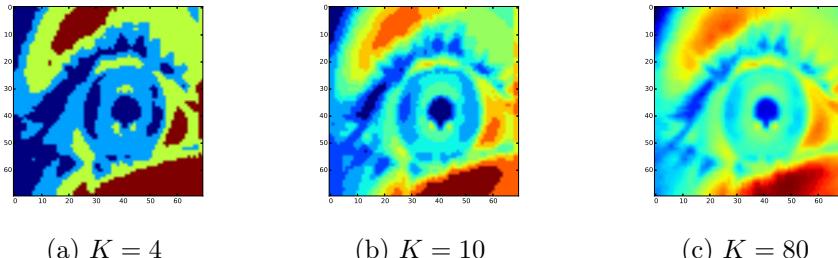


Figure 7: K-means for different values of K

For performance reasons, the k-means procedure can not be calculated from the original image, the clustering is therefore done on a resized 70x70 pixel image. To reduce the impact of noise, the resized image is filtered with a gaussian filter. It is clear in figure 7 that even for a very high K, the pupil is not a separate cluster, Experiments has revealed that at a K value in the range of 10 to 20 gives a reasonably clustered image, while not having a massive impact on performance.

Each pixel in the reduced picture is assigned to a cluster(label), selecting the pixels in the label with the lowest mean value, we can create a binary image, which now can be resized to the original image size. Because the cluster also contains pixels from other regions than the pupil area, the resulting

binary image, on which we can do BLOB detection, is not optimal. Figure 8 shows this.

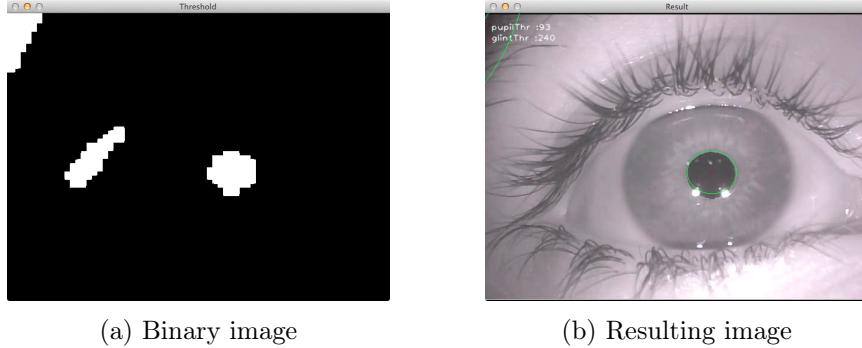


Figure 8: Pupil detection using k-means

Eyelashes and shadows become a part of the pupil cluster, and much like the issues with morphology, they often become connected components, which makes BLOB detection very difficult.

In conclusion the use of k-means did not yield a better result than ordinary thresholding, in many cases the result was actually worse. This is unfortunate because, if the pupil could be found as a single cluster, the amount of evaluation on the BLOB could be reduced and give better scalability on the position of the eye relative to the camera.

2.3 Pupil Detection using Gradient Magnitude

So far, we have been looking at the intensity values of the image. This has yielded reasonable approximate results, but is not as robust as we would like. In the following, we investigate what happens if we look at the change in intensity (the image gradient / first derivative), instead of the absolute intensity value at a given point. The gradients in the X and Y directions, are easily calculated with a Sobel filter. And from these, we can calculate the Gradient Magnitude as: $\sqrt{x^2 + y^2}$ (the Euclidean length of the vector $x + y$), and the orientation as: $\arctan2(y, x)$. Figure 9 shows a subsampled cutout of the Gradient image of Eye1, featuring the pupil and glints.

Note that the pupil boundary is clearly visible on Figure 9. We will attempt to use this information as follows: given an approximate centerpoint and radius for the pupil, scan in a number of directions, d from the centerpoint, find the location of the maximum gradient magnitudes along the line-segments that are described by the centerpoint, a direction from d and the radius. Use this set of points to fit an ellipse, and use that as improved pupil detection.

Figure 10(left) shows the lines considered, and the max gradient points

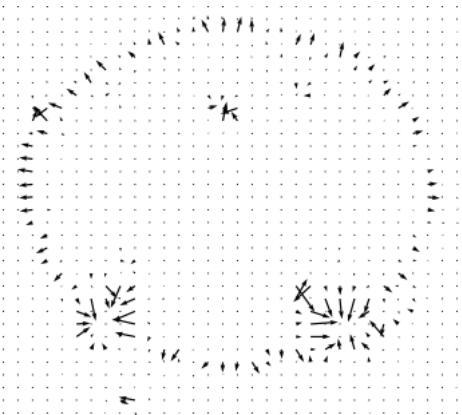


Figure 9: Quiver plot of Eye1 gradients (zoomed on pupil area)

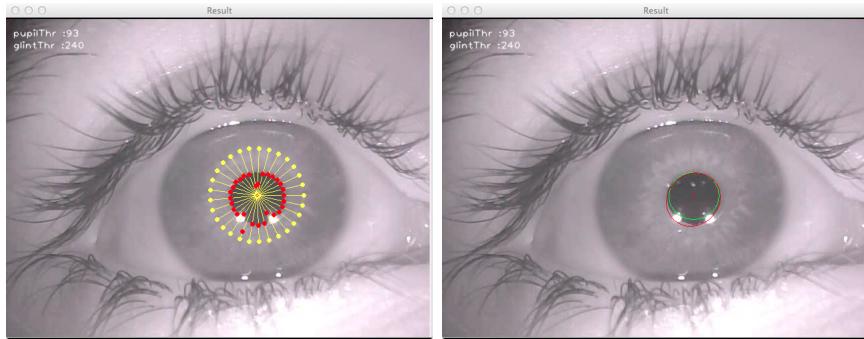


Figure 10: Left: Eye1 showing line-segments for gradient magnitude maximization (yellow) and maximum gradient values along the lines (red). Right: Eye1 showing pupil approximation (green), and new pupil detection (red)

found. Figure 10(right) shows the old and new pupil detections. This approach suffers the same problem as earlier. When the glints obscure part of the boundary, the pupil detection fails to follow the lower boundary. On top of that, there are also issues with noise, notice the red dots inside the top part of the pupil on Figure 10, these are caused by non-system light reflecting off the pupil.

The noise issues can be drastically reduced with proper pre-processing. Figure 11 shows much improved results, when blurring the image beforehand.

We experimented with ignoring gradient points where the orientation was too far from the orientation of the circle normal, but did not see any improvements to the pupil detection.

2.4 Pupil Detection by circular Hough transformation

In an attempt to make our pupil detection more robust we now investigate the result of applying a circular Hough transformation on the eye images.

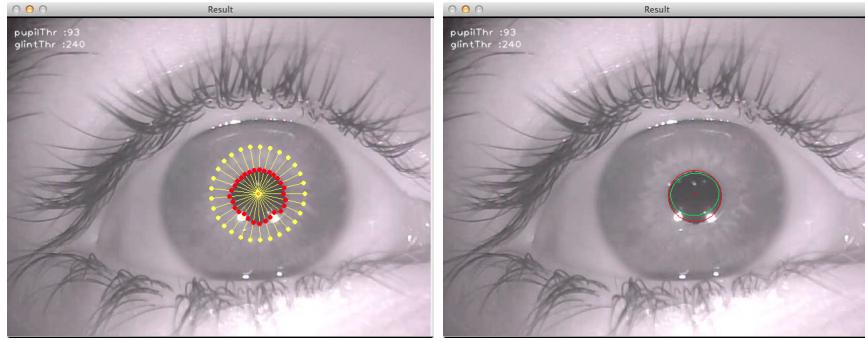


Figure 11: as Figure 10, but pre-processed with a 9x9 Gaussian Blur

The main challenge is finding the correct parameters for the process. We consider the following parameters:

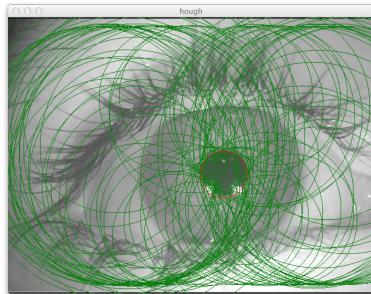
Gauss kernel size The size of the gaussian kernel that is applied to the image before the Hough transformation

σ - value the standard deviation value used to construct the gaussian kernel.

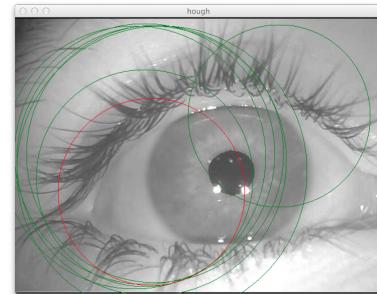
Accumulator threshold The minimum cumulative vote threshold in which some parameters for a circle are considered.

Minimum and Maximum Radius The minumum and maximum radius of circles to consider.

The next step is to experimentally find the parameters that yields the best result over all sequences.



(a) Accumulator threshold at 100



(b) Accumulator threshold at 150

Figure 12: Finding the accumulator threshold values

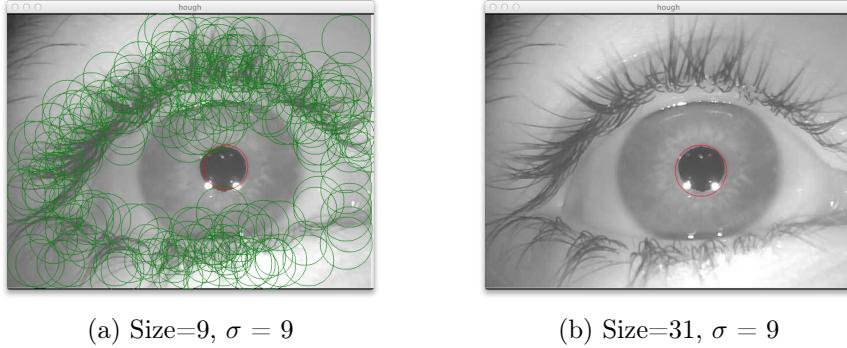


Figure 13: Preprocessing by smoothing with a gaussian kernel

Discussion The circular Hough transformation yields the most robust pupil detection we have been able to produce so far.

The intuition behind this is that in an ideal setting, where the eye for example is not distorted by perspective, the pupil is going to be near-circular, so the process of Hough transforming and then voting for circles is likely to succeed.

In a non ideal setting, for example in a frame were the eye is seen from the side the pupil is not going to yield the same circular properties, and the process is going to fail.

By preprocessing the image by smooting some of the noise is going to be filtered out. The idea is to choose a kernel that is roughly of the same size as the pupil. In this manner smaller features, such as eye lashes will be smoothed away, but still maintaining the pupil feature.

The drawback of setting a constant size for the gaussian kernel is that is is not scale independent. An ideal kernel in some frame may smooth away the pupil in some other frame if the had subject moved closer or futher away from the camera.

3 Glint Detection

In this section, we will investigate and discuss glint detection.

3.1 Thresholding

Like pupil detection thresholding seems like an obvious place to start. The methods are almost identical. By first creating a binary image from a threshold value, and then do a BLOB analysis, resulting in a set of points where glints are present.

Figure 14 shows the glints in ‘eye1.avi’. Because glints are very close to white, a fairly high threshold gives a good result. Furthermore by experi-

menting we found that, by requiring that the BLOB area lies in the interval [10 : 150], we exclude a great deal of unwanted glint detections, it is although clear that there is still a good amount of false positives.

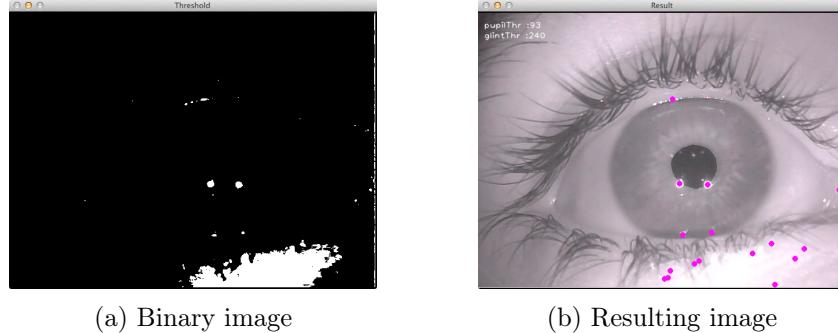


Figure 14: Glint detection with threshold of 240

Morphology To mitigate the false positives, we make use of morphology to remove small “spots” of white by first closing and then opening, we get rid of a lot of unwanted glints, as can be seen in figure 15

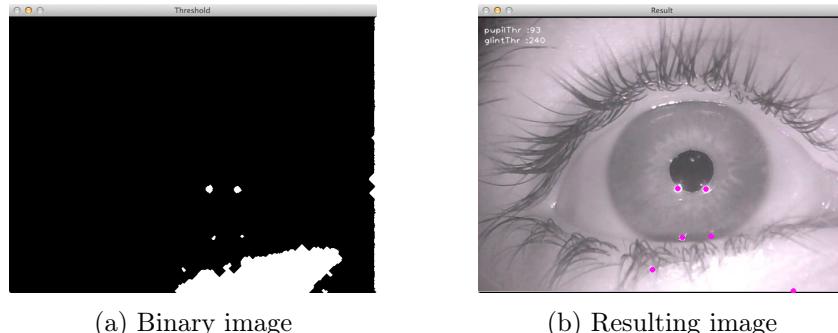


Figure 15: Glint detection with threshold of 240 - using morphology

Filter glints using eye features To further enhance glint detection we have added a function function that excludes glints that have an Euclidian distance from the center of the pupil greater than the largest radius of the ellipse representing the pupil. Figure 16



Figure 16: Glint detection with threshold of 240 - using morphology and filter

Discussion The method works very well, but fail when more than one pupil is detected, in other words the stability of glint detection is dependent the quality of the pupil detection.

4 Eye Corner Detection

We detect eye corners simply by template matching. We use template matching by normalized cross-correlation, which is invariant to lighting and exposure conditions, at the expense of performance. Because of the large changes in intensities in the different sequences, we find this to be the preferred method.

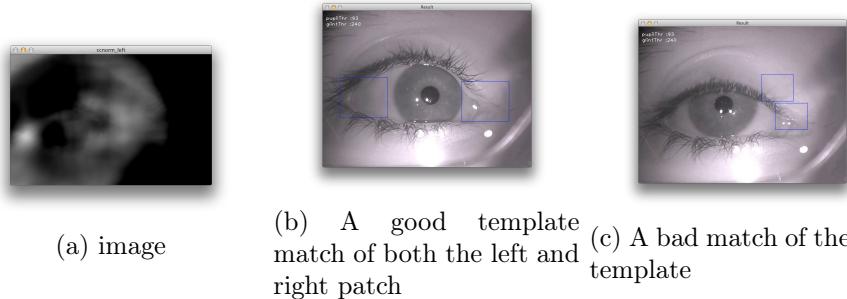


Figure 17: Template Matching

Examples The technique works best in a stable environment. Thus if the test subject moves his/her head on any of the 3 axis, or closer/further away

from the camera, the chosen template will of course produce a lower result at the position of the actual eye corner.

The changes in scale can be mitigated by using an image pyramid, i.e. convolving the template with multiple versions of a downsampled or upscaled version of the input frame.

Changes in rotation can be mitigated in a similar manner by rotating the template in a number of steps around its own axis.

Changes in perspective, i.e if the person turns his/her head, could possibly be mitigated by transforming the image by a homography.

The methods can be combined to detect a change in both scale and rotation. A problem with these techniques is that they introduce some computational complexity.

5 Iris / Limbus Detection

5.1 Iris detection using thresholding

We experimented with simple thresholding, to detect the shape and position of the iris. What we found, was that it will work under ideal circumstances, but is extremely brittle with regards to chosen threshold and system lighting. Furthermore, because the iris intensities lies in the middle range (with glints being very bright, and pupil being very dark), many false positives were being picked up in the skin-tone range. This experiment yielded no usable results, and are not discussed further.

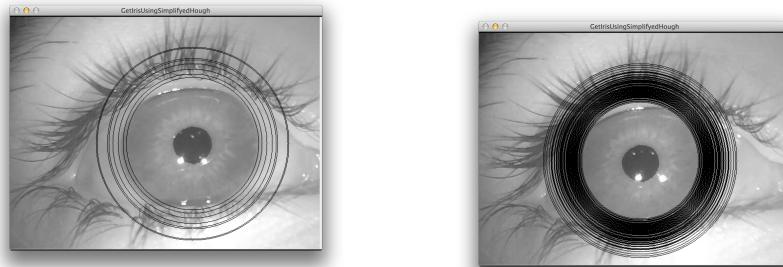
5.2 Hough Transformation with position prior

If we assume that we are somehow able to detect the pupil in a robust way, we can use this position as input to a circular Hough transformation.

In this way the Hough tranformation will be one dimensional since the only free variable is the radius of the circle.

The procedure is then to iterate over the circles in a range between some minimum and maximum radius with some step size. We then discretize the periphery of the circle and iterate over this discretization. We then increment the value of the current radius, in a parametric accumulator table, if the corresponding point in the Canny edge image is larger than zero.

The radius with the highest value in the accumulator table is thus the circle that yields the best fit.



(a) One correct circle and some false positives (b) Multiple true positives

Figure 18: Detecting circles using Hough transformation with position prior

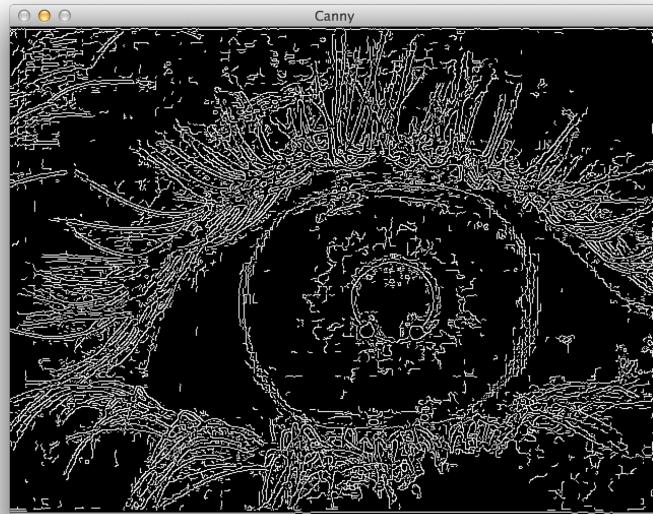


Figure 19: The canny edge image on which the circles in figure (a) are found

Examples In figure (a) the method positively identifies the limbus and some false positives.

In figure (b) the method yields more circles above some given threshold since the perspective distortion exhibits a limbus that appears to be more circular than it actually is.

Discussion The method works but is sensitive on numerous parameters. The threshold should be carefully chosen, and is dependent on the granularity of the Canny edge image it takes as input. A canny edge image with a lot

of edges yields an accumulator table with more votes, and thus more iris candidates.

Looking at Figure 19 we see that one possible improvement could be to smooth the image before producing the Canny edge image. This would remove the noise produced by eye lashes and other non-circular features.

6 Conclusion

After ample experimentation, we find that the following techniques perform best:

Pupil Hough transform gives a robust pupil detection, that works on image gradient, instead of actual intensity. This makes it much more tolerant of changing lighting conditions. We preprocess the input image with a Gaussian blur, to minimize the effect of noise.

Eye Corners We have only experimented with Template Matching for eye corner detection. We have not attempted to use a generic set of templates across multiple sequences, but have settled with having to define templates for each sequence.

Glints Our thresholding approach is extremely good at finding all the highlights of the image, but results in many false positives. Filtering those with blob detection eliminates most of the false positives. Further, requiring that the glints are close to the pupil center, results in a very robust glint detection (given that the pupil is correctly detected)

Iris / Limbus We use our own, simplified Hough transform, that assumes the pupil center has been found correctly, and that the limbus is approximately circular. This is the least robust of our feature detections, but works well enough in practice to be useful.

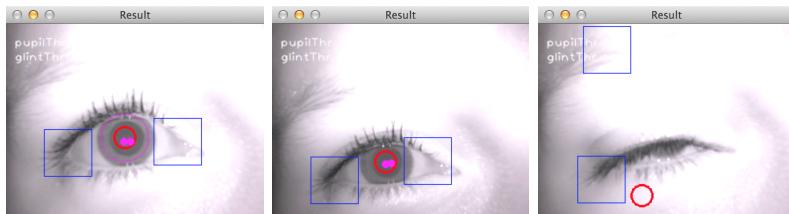


Figure 20: Left: working detection, middle: Iris detection failed, right: Totally failed detection (blink)

Performance We have recorded our own sequence: 'Julie.avi', using an iSlim 321r camera with infrared capabilities. Figure 20 shows an image from 'Julie.avi' where detection is successful, and an image where the detection fails. Enclosed on the CD-ROM is the full sequence 'Julie.avi' (unedited recording), and 'Julie_Detection.avi' that shows the performance of our eye tracker across the entire sequence.

There is certainly room for improvement, especially in the areas of eye corner and iris detection. But overall we are happy with the performance of our eye tracker, especially the various Hough transforms we apply, perform very well.

Appendix

6.1 Assignment1.py

```
1 import cv2
2 import cv
3 import pylab
4 import math
5 from SIGBTools import RegionProps
6 from SIGBTools import getLineCoordinates
7 from SIGBTools import ROISelector
8 from SIGBTools import getImageSequence
9 from SIGBTools import getCircleSamples
10 import SIGBTools
11 import numpy as np
12 import sys
13 from scipy.cluster.vq import *
14 from scipy.misc import *
15 from matplotlib.pyplot import *

17

19 inputFile = "own_Sequences/julie.avi"
20 outputFile = "eyeTrackerResult.mp4"
21
22 #seems to work okay for eye1.avi
23 default_pupil_threshold = 93

25 #
26 #           Global variable
27 #
28 global imgOrig, leftTemplate, rightTemplate, frameNr
29 imgOrig = [];
30 #These are used for template matching
31 leftTemplate = []
32 rightTemplate = []
33 frameNr =0;

35
36 def GetPupil(gray, thr, min_val, max_val):
37     '''Given a gray level image, gray and threshold value return a
38         list of pupil locations'''
39     #tempResultImg = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR) #used
40         to draw temporary results
41
42     #Threshold image to get a binary image
43     cv2.imshow("TempResults", gray)
44     val, binI =cv2.threshold(gray, thr, 255, cv2.THRESH_BINARY_INV)
45     #print val
46     #Morphology (close image to remove small 'holes' inside the
47         pupil area)
48     st = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
```

```

47     #binI = cv2.morphologyEx(binI, cv2.MORPH_OPEN, st, iterations
48     #=10)
49     #binI = cv2.morphologyEx(binI, cv2.MORPH_CLOSE, st, iterations
50     #=5)
49     cv2.imshow("Threshold",binI)
51     #Calculate blobs, and do edge detection on entire image (
52     #    modifies binI)
51     contours, hierarchy = cv2.findContours(binI, cv2.RETR_LIST,
52     cv2.CHAIN_APPROX_SIMPLE)

53     pupils = []
54     prop_calc = RegionProps()
55     centroids = []
56     for contour in contours:
57         #calculate centroid, area and 'extend' (compactness of
58         #contour)
59         props = prop_calc.CalcContourProperties(contour, ["centroid",
60             "area", "extend"])
60         x, y = props["Centroid"]
61         area = props["Area"]
62         extend = props["Extend"]
63         #filter contours, so that their area lies between min_val
64         #and max_val, and then extend lies between 0.4 and 1.0
65         if (area > min_val and area < max_val and extend > 0.5 and
66         extend < 1.0):
67             pupilEllipse = cv2.fitEllipse(contour)
68             # center, radii, angle = pupilEllipse
69             # max_radius = max(radii)
70             # c_x = int(center[0])
71             # c_y = int(center[1])
72             # cv2.circle(tempResultImg,(c_x,c_y), int(max_radius),
73             #(0,0,255),4) #draw a circle
74             #cv2.ellipse(tempResultImg, pupilEllipse,(0,255,0),1)
75             pupils.append(pupilEllipse)

76     #cv2.imshow("TempResults",tempResultImg)

77     return pupils

77 def GetGlints(gray,thr):
78     min_area = 2
79     max_area = 150
80     ''' Given a gray level image, gray and threshold
81     value return a list of glint locations '''
82     #print thr
83     val, binary_image = cv2.threshold(gray, thr, 255, cv2.
84     THRESH_BINARY)

85     st = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
86     #binary_image = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE
87     , st, iterations=8)
88     #binary_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN,
89     st, iterations=2)

```

```

89

91     #cv2.imshow("Threshold", binary_image)
92     #Calculate blobs, and do edge detection on entire image (
93     #    modifies binI)
93     contours, hierarchy = cv2.findContours(binary_image, cv2.
94         RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

95     glints = [];
96     prop_calc = RegionProps()
97     centroids = []
98     for contour in contours:
99
100         #calculate centroid, area and 'extend' (compactness of
100         #contour)
101         props = prop_calc.CalcContourProperties(contour, ["centroid"
101             , "area", "extend"])
102         x, y = props["Centroid"]
103         area = props["Area"]
104         extend = props["Extend"]
105         print x, y, area, extend

106         #filter contours, so that their area lies between min_val
106         #and max_val, and then extend lies between 0.4 and 1.0
107         if area > min_area and area < max_area: #and extend > 0.4
107             and extend < 1.0):
108             glints.append((x,y))

109             #cv2.circle(tempResultImg,(int(x),int(y)), 2, (0,0,255),4)
110             #draw a circle
111             #cv2.imshow("TempResults",tempResultImg)

112     return glints

113

114     def GetIrisUsingThreshold(gray, thr, min_val, max_val):
115         ''' Given a gray level image, gray and threshold
115         value return a list of iris locations '''
116         val,binary_image = cv2.threshold(gray, thr, 255, cv2.
116             THRESH_BINARY_INV)
117         cv2.imshow("Threshold", binary_image)

118         contours, hierarchy = cv2.findContours(binary_image, cv2.
118             RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

119         irises = [];
120         prop_calc = RegionProps()
121         centroids = []
122         for contour in contours:
123             #calculate centroid, area and 'extend' (compactness of
123             #contour)
124             props = prop_calc.CalcContourProperties(contour, ["centroid"
124                 , "area", "extend"])
125             x, y = props["Centroid"]
126             area = props["Area"]

```

```

    extend = props["Extend"]
133     #filter contours, so that their area lies between min_val
     and max_val, and then extend lies between 0.4 and 1.0
     if area > min_val and area < max_val and extend > 0.5 and
     extend < 1.0:
135         irisEllipse = cv2.fitEllipse(contour)
         # center , radii , angle = pupilEllipse
137         # max_radius = max(radii)
         # c_x = int(center[0])
139         # c_y = int(center[1])
         # cv2.circle(tempResultImg,(c_x,c_y), int(max_radius),
(0,0,255),4) #draw a circle
141         #cv2.ellipse(tempResultImg, pupilEllipse,(0,255,0),1)
         irises.append(irisEllipse)
143     return irises

145 def circularHough(gray):
    ''' Performs a circular hough transform of the image, gray and
        shows the detected circles
147     The circe with most votes is shown in red and the rest in
        green colors '''
    #See help for http://opencv.itseez.com/modules/imgproc/doc/
        feature_detection.html?highlight=houghcircle#cv2.HoughCircles
149     blur = cv2.GaussianBlur(gray, (13,13), 9)

151     dp = 6; minDist = 30
     highThr = 10 #High threshold for canny
153     accThr = 20; #accumulator threshold for the circle centers at
        the detection stage. The smaller it is , the more false
        circles may be detected
     maxRadius = 10;
155     minRadius = 20;
     circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
        minDist, None, highThr, accThr, maxRadius, minRadius)
157     #Make a color image from gray for display purposes
     gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
159     if (circles !=None):
        #print circles
161         all_circles = circles[0]
         M,N = all_circles.shape
163         k=1
         #for c in all_circles:
165             # cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
M),k*128,0))
             # K=k+1
167             c=all_circles[0,:]
             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255),5)
169             #cv2.imshow("hough",gColor)
         return [c]
171

173 def simplifiedHough(edgeImage ,circleCenter ,minR,maxR,N,thr):
     samplePoints = 200
175     accumulator_line = {}

```

```

177     for radius in range(minR, maxR, N):
178         points = getCircleSamples(center=circleCenter, radius=radius
179             , nPoints=samplePoints)
180         accumulator_line[radius] = 0
181         for point in points:
182             try:
183                 if edgeImage[point[0], point[1]] > 0:
184                     accumulator_line[radius] += 1
185             except IndexError:
186                 continue
187         radii = []
188         for radius in accumulator_line:
189             value = accumulator_line[radius]
190             if value > thr:
191                 radii.append(radius)
192             print value
193     return radii
194
195     def GetIrisUsingNormals(gray, pupil, normalLength):
196         ''' Given a gray level image, gray and the length of the
197             normals, normalLength
198             return a list of iris locations '''
199     # YOUR IMPLEMENTATION HERE !!!!
200     pass
201
202     def GetIrisUsingSimplifedHough(gray, pupil):
203         ''' Given a gray level image, gray
204             return a list of iris locations using a simplified Hough
205             transformation '''
206         if pupil != None:
207             edges = cv2.Canny(gray, 30, 20)
208             pupil = pupil
209             pupil_x = int(pupil[0])
210             pupil_y = int(pupil[1])
211             cv2.imshow("edges", edges)
212             radii = simplifiedHough(edges, pupil, 10, 100, 1, 50)
213             irises = []
214             for radius in radii:
215                 #cv2.circle(gray, (pupil_x, pupil_y), radius, (0, 0, 0),
216                 1)
217                 #cv2.circle(gray, pupil, radius, (127,127,127), 2)
218                 #cv2.imshow("GetIrisUsingSimplifedHough", gray)
219                 irises.append((pupil_x, pupil_y, radius))
220     return irises
221
222     def plotVectorField(I):
223         g_x = cv2.Sobel(I, cv.CV_64F, 1, 0, ksize=3)
224         g_y = cv2.Sobel(I, cv.CV_64F, 0, 1, ksize=3) # ksize=3 som ***
225         kwargs

```

```

225     x_orig_dim, y_orig_dim = I.shape
227     x_mesh_dim, y_mesh_dim = (3, 3)

229     sample_g_x = g_x[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
230         x_mesh_dim]
231     sample_g_y = g_y[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
232         x_mesh_dim]

233     quiver(sample_g_x, sample_g_y)
234     show()

235 def getGradientImageInfo(I):
236     g_x = cv2.Sobel(I, cv.CV_64F, 1,0)
237     g_y = cv2.Sobel(I, cv.CV_64F, 0,1) # ksize=3 som **kwargs
238
239     X,Y = I.shape
240     orientation = np.zeros(I.shape)
241     magnitude = np.zeros(I.shape)
242     sq_g_x = cv2.pow(g_x, 2)
243     sq_g_y = cv2.pow(g_y, 2)
244     fast_magnitude = cv2.pow(sq_g_x + sq_g_y, .5)

245     # for x in range(X):
246     #     for y in range(Y):
247     #         orientation[x][y] = np.arctan2(g_y[x][y], g_x[x][y]) *
248     #             (180 / math.pi)
249     #         magnitude[x][y] = math.sqrt(g_y[x][y] ** 2 + g_x[x][y]
250     #             ** 2)

251
252     #print fast_magnitude[0]
253     #print magnitude[0]
254
255     return fast_magnitude, orientation
256
257 def GetEyeCorners(orig_img, leftTemplate, rightTemplate,
258     pupilPosition=None):
259     if leftTemplate != [] and rightTemplate != []:
260         ccnorm_left = cv2.matchTemplate(orig_img, leftTemplate, cv2.
261             TM_CCOEFF_NORMED)
262         ccnorm_right = cv2.matchTemplate(orig_img, rightTemplate,
263             cv2.TM_CCOEFF_NORMED)

264         minVal, maxVal, minLoc, maxloc_left_from = cv2.minMaxLoc(
265             ccnorm_left)
266         minVal, maxVal, minLoc, maxloc_right_from, = cv2.minMaxLoc(
267             ccnorm_right)

268         l_x,l_y = leftTemplate.shape
269         max_loc_left_from_x = maxloc_left_from[0]
270         max_loc_left_from_y = maxloc_left_from[1]

```

```

271     max_loc_left_to_x = max_loc_left_from_x + l_x
271     max_loc_left_to_y = max_loc_left_from_y + l_y

273     maxloc_left_to = (max_loc_left_to_x, max_loc_left_to_y)

275     r_x,r_y = leftTemplate.shape
276     max_loc_right_from_x = maxloc_right_from[0]
277     max_loc_right_from_y = maxloc_right_from[1]

279     max_loc_right_to_x = max_loc_right_from_x + r_x
280     max_loc_right_to_y = max_loc_right_from_y + r_y
281     maxloc_right_to = (max_loc_right_to_x, max_loc_right_to_y)

283     return (maxloc_left_from, maxloc_left_to, maxloc_right_from,
284             maxloc_right_to)

285 bgr_yellow = 0,255,255
286 bgr_blue = 255, 0, 0
287 bgr_red = 0, 0, 255
288 def circleTest(img, center_point):
289     nPts = 20
290     circleRadius = 100
291     P = getCircleSamples(center=center_point, radius=circleRadius,
292                          nPoints=nPts)
293     for (x,y,dx,dy) in P:
294         point_coords = (int(x),int(y))
295         cv2.circle(img, point_coords, 2, bgr_yellow, 2)
296         cv2.line(img, point_coords, center_point, bgr_yellow)

297 def findEllipseContour(img, gradient_magnitude,
298                       gradient_orientation, estimatedCenter, estimatedRadius, nPts
299                       =30):
300     center_point_coords = (int(estimatedCenter[0]), int(
301                             estimatedCenter[1]))
302     P = getCircleSamples(center = estimatedCenter, radius =
303                           estimatedRadius, nPoints=nPts)
304     for (x,y,dx,dy) in P:
305         point_coords = (int(x),int(y))
306         cv2.circle(img, point_coords, 2, bgr_yellow, 2)
307         cv2.line(img, point_coords, center_point_coords, bgr_yellow)

308 newPupil = np.zeros((nPts,1,2)).astype(np.float32)
309 t = 0
310 for (x,y,dx,dy) in P:
311     #< define normalLength as some maximum distance away from
312     # initial circle >
313     #< get the endpoints of the normal -> p1,p2>
314     point_coords = (int(x),int(y))
315     normal_gradient = dx, dy
316     #cv2.circle(img, point_coords, 2, bgr_blue, 2)
317     max_point = findMaxGradientValueOnNormal(gradient_magnitude,
318                                              gradient_orientation, point_coords, center_point_coords,
319                                              normal_gradient)
320     cv2.circle(img, tuple(max_point), 2, bgr_red, 2) #locate the

```

```

    max_points
315     #< store maxPoint in newPupil>
      newPupil[t] = max_point
317     t += 1
#<fitPoints to model using least squares- cv2.fitellipse(
318     newPupil)>
319     return cv2.fitEllipse(newPupil)

321 def findMaxGradientValueOnNormal(gradient_magnitude,
322     gradient_orientation, p1, p2, normal_orientation):
323     #Get integer coordinates on the straight line between p1 and
324     p2
325     pts = SIGBTools.getLineCoordinates(p1, p2)
326     values = gradient_magnitude[pts[:,1], pts[:,0]]
327     #orientations = gradient_orientation[pts[:,1], pts[:,0]]
328     #normal_angle = np.arctan2(normal_orientation[1],
329     normal_orientation[0]) * (180 / math.pi)
330
331     # orientation_difference = abs(orientations - normal_angle)
332     # print orientation_difference[0:10]
333     # max_index = 0 #np.argmax(values)
334     # max_value = 0
335     # for index in range(len(values)):
336     #     if orientation_difference[index] < 20:
337     #         if values[index] > max_value:
338     #             max_index = index
339     #             max_value = values[index]
340     #print orientations[max_index], normal_angle
341     max_index = np.argmax(values)
342     return pts[max_index]
343     #return coordinate of max value in image coordinates
344
345 def FilterPupilGlint(pupils, glints):
346     ''' Given a list of pupil candidates and glint candidates
347         returns a list of pupil and glints '''
348     filtered_glints = []
349     filtered_pupils = pupils
350     for glint in glints:
351         for pupil in pupils:
352             if (is_glint_close_to_pupil(glint, pupil)):
353                 filtered_glints.append(glint)

354     return filtered_pupils, filtered_glints

355 def is_glint_close_to_pupil(glint, pupil):
356     x, y, radius = pupil
357     center = (x,y)
358     distance = euclidianDistance(center, glint)
359     return (distance < radius* 1.5)

360 def filterGlintsIris(glints, irises):
361     new_glints = []
362     if glints and irises:
363         for glint in glints:

```

```

363     for iris in irises:
364         iris_x, irix_y, iris_radius = iris
365         print glint
366         iris_vector = np.array([iris_x, irix_y])
367         distance = np.linalg.norm(glint - iris_vector)
368         if distance < iris_radius:
369             new_glints.append(glint)
370             #print iris
371     return new_glints

373

375 def update(I):
376     '''Calculate the image features and display the result based
377     on the slider values'''
378     #global drawImg
379     global frameNr, drawImg, gray
380     img = I.copy()
381     sliderVals = getSliderVals()
382     gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
383     gray = cv2.equalizeHist(gray)

384     # Do the magic
385     #pupils = GetPupil(gray, sliderVals[ 'pupilThr '], sliderVals[ 'minSize '], sliderVals[ 'maxSize '])
386     pupils = circularHough(gray)
387     glints = GetGlints(gray, sliderVals[ 'glintThr '])
388     pupils, glints = FilterPupilGlint(pupils, glints)
389     #irises = GetIrisUsingThreshold(gray, sliderVals[ 'pupilThr '],
390                                     sliderVals[ 'minSize '], sliderVals[ 'maxSize '])

391     K=10
392     d=40
393     #labelIm , centroids = detectPupilKMeans(gray ,K=K, distanceWeight
394         =d, reSize=(70,70))
395     #pupils = get_pupils_from_kmean(labelIm , centroids ,gray ,
396                                     sliderVals[ 'minSize '], sliderVals[ 'maxSize '])

397     #magnitude, orientation = getGradientImageInfo(gray)
398     if pupils:
399         irises = GetIrisUsingSimplifedHough(gray , pupils [0])

400     #plotVectorField(gray)
401     #Do template matching
402     global leftTemplate
403     global rightTemplate

404     corners = GetEyeCorners(gray , leftTemplate , rightTemplate)

405     #detectPupilHough(gray , 100)
406     #irises = detectIrisHough(gray , 400)

407     #glints = filterGlintsIris(glints , irises)

411

```

```

#Display results
413 global frameNr,drawImg
x,y = 10,10
415 #setText(img,(x,y),"Frame:%d" %frameNr)
sliderVals = getSliderVals()
417
# for non-windows machines we print the values of the
# threshold in the original image
419 if sys.platform != 'win32':
    step=18
421 cv2.putText(img, "pupilThr :" +str(sliderVals['pupilThr']), (
    x, y+step), cv2.FONT_HERSHEY_PLAIN, 1.0, (255, 255, 255),
    lineType=cv2.CV_AA)
    cv2.putText(img, "glintThr :" +str(sliderVals['glintThr']), (
    x, y+2*step), cv2.FONT_HERSHEY_PLAIN, 1.0, (255, 255, 255),
    lineType=cv2.CV_AA)
423 cv2.imshow('Result',img)

425 #Uncomment these lines as your methods start to work to
#       display the result in the
#original image
427
#Ellipse
429 # for pupil in pupils:
#    #cv2.ellipse(img,pupil,(0,255,0),1)
431 #    C = int(pupil[0][0]),int(pupil[0][1])
#Circle
433 for pupil in pupils:
    cv2.circle(img, (int(pupil[0]),int(pupil[1])),pupil[2],
(0,0,255),2)
435
#    contour = findEllipseContour(img, magnitude, orientation,
C, 70)
437 #    cv2.ellipse(img, contour, bgr_red, 1)
#    cv2.circle(img,C, 2, (0,0,255),1)
439 #circleTest(img, C)
for glint in glints:
    C = int(glint[0]),int(glint[1])
    cv2.circle(img,C,(255,0,255),5)
443

445 if corners:
447     left_from, left_to, right_from, right_to = corners
        cv2.rectangle(img, left_from, left_to, 255)
449     cv2.rectangle(img, right_from, right_to, 255)

451 for iris in irises:
#cv2.ellipse(img,iris,(0,255,0),1)
453     C = int(iris[0]),int(iris[1])
        radius = int(iris[2])
        cv2.circle(img, C, radius, (255,0,255), 1)

455 cv2.imshow("Result", img)

```

```

459      #For Iris detection - Week 2
460      #
461      #copy the image so that the result image (img) can be saved in
462          the movie
463      drawImg = img.copy()

465
466      def printUsage():
467          print "Q or ESC: Stop"
468          print "SPACE: Pause"
469          print "r: reload video"
470          print 'm: Mark region when the video has paused'
471          print 's: toggle video writing'
472          print 'c: close video sequence'

473      def run(fileName ,resultFile='eyeTrackingResults.avi'):
474
475          ''' MAIN Method to load the image sequence and handle user
476              inputs '''
477          global imgOrig , frameNr,drawImg , leftTemplate , rightTemplate ,
478              gray
479          setupWindowSliders()
480          props = RegionProps();
481          cap ,imgOrig ,sequenceOK = getImageSequence(fileName)
482          videoWriter = 0

483          frameNr =0
484          if(sequenceOK):
485              update(imgOrig)
486              printUsage()
487          frameNr=0;
488          saveFrames = False

489          while(sequenceOK):
490              sliderVals = getSliderVals();
491              frameNr=frameNr+1
492              ch = cv2.waitKey(1)
493              #Select regions
494              if(ch==ord('m')):
495                  if(not sliderVals[ 'Running']):
496                      roiSelect=ROISelector(imgOrig)
497                      pts ,regionSelected= roiSelect.SelectArea('Select eye
498                          corner',(400,200))
499                      if(regionSelected):
500                          if leftTemplate == []:
501                              leftTemplate = gray[pts [0][1]:pts [1][1] ,pts [0][0]:
502                                  pts [1][0]]
503                          else:
504                              rightTemplate = gray[pts [0][1]:pts [1][1] ,pts [0][0]:
505                                  pts [1][0]]

506          if ch == 27:

```

```

        break
507     if (ch==ord('s')):
508         if((saveFrames)):
509             videoWriter.release()
510             saveFrames=False
511             print "End recording"
512         else:
513             imSize = np.shape(imgOrig)
514             videoWriter = cv2.VideoWriter(resultFile, cv.CV_FOURCC('D','I','V','3'), 15.0,(imSize[1],imSize[0]),True) #Make a
515             video writer
516             saveFrames = True
517             print "Recording..."
518
519
520         if(ch==ord('q')):
521             break
522         if(ch==32): #Spacebar
523             sliderVals = getSliderVals()
524             cv2.setTrackbarPos('Stop/Start','Controls',not sliderVals['Running'])
525         if(ch==ord('r')):
526             frameNr =0
527             sequenceOK=False
528             cap,imgOrig,sequenceOK = getImageSequence(fileName)
529             update(imgOrig)
530             sequenceOK=True
531
532         sliderVals=getSliderVals()
533         if(sliderVals['Running']):
534             sequenceOK, imgOrig = cap.read()
535             if(sequenceOK): #if there is an image
536                 update(imgOrig)
537             if(saveFrames):
538                 videoWriter.write(drawImg)
539         if(videoWriter!=0):
540             videoWriter.release()
541             print "Closing videofile..."
542 #
543
544     def detectPupilKMeans(gray,K=2,distanceWeight=2,reSize=(40,40)):
545         ''' Detects the pupil in the image, gray, using k-means
546             gray : grays scale image
547             K : Number of clusters
548             distanceWeight : Defines the weight of the position
549             parameters
550             reSize : the size of the image to do k-means on
551             '''
552         #Resize for faster performance
553         smallI = cv2.resize(gray, reSize)
554         smallI = cv2.GaussianBlur(smallI,(3,3),20)
555         M,N = smallI.shape
556         #Generate coordinates in a matrix

```

```

X,Y = np.meshgrid(range(M),range(N))
557 #Make coordinates and intensity into one vectors
z = smallI.flatten()
559 x = X.flatten()
y = Y.flatten()
561 O = len(x)
#make a feature vectors containing (x,y,intensity)
563 features = np.zeros((O,3))
features[:,0] = z;
565 features[:,1] = y/distanceWeight; #Divide so that the distance
      of position weighs less than intensity
features[:,2] = x/distanceWeight;
567 features = np.array(features,'f')
# cluster data
569 centroids, variance = kmeans(features,K)
#use the found clusters to map
571 label, distance = vq(features,centroids)
# re-create image from
573 labelIm = np.array(np.reshape(label,(M,N)))
return labelIm, centroids
575
def get_pupils_from_kmean(labelIm, centroids, gray, min_val,
    max_val):
577 result = np.zeros((labelIm.shape))
label = np.argmin(centroids[:,0])
579 result[labelIm == label] = [255]
y,x=gray.shape
581 result = cv2.resize(result,(x,y))
semi_binI = np.array(result, dtype='uint8')
583 #remove gray elements created from the linear interpolation
val,binI=cv2.threshold(semi_binI, 0, 255, cv2.THRESH_BINARY)
585 cv2.imshow("Threshold",binI)
#Calculate blobs, and do edge detection on entire image (
    modifies binI)
587 contours, hierarchy = cv2.findContours(binI, cv2.RETR_LIST,
    cv2.CHAIN_APPROX_SIMPLE)
589 pupils = [];
prop_calc = RegionProps()
591 for contour in contours:
    #calculate centroid, area and 'extend' (compactness of
    contour)
    props = prop_calc.CalcContourProperties(contour, ["centroid",
        "area", "extend"])
    x, y = props["Centroid"]
595 area = props["Area"]
extend = props["Extend"]
597 #filter contours, so that their area lies between min_val
and max_val, and then extend lies between 0.4 and 1.0
    if (area > min_val and area < max_val and extend > 0.4 and
extend < 1.0):
599     pupilEllipse = cv2.fitEllipse(contour)
        pupils.append(pupilEllipse)
601 return pupils

```

```

603 def detectPupilHough(gray, accThr=600):
#Using the Hough transform to detect ellipses
605 blur = cv2.GaussianBlur(gray, (9,9),9)
##Pupil parameters
607 dp = 6; minDist = 10
highThr = 30 #High threshold for canny
609 #accThr = 600; #accumulator threshold for the circle centers
    at the detection stage. The smaller it is, the more false
    circles may be detected
maxRadius = 50;
minRadius = 30;
#See help for http://opencv.itseez.com/modules/imgproc/doc/
    feature_detection.html?highlight=houghcircle#cv2.
    HoughCirclesIn thus
613 circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
    minDist, None, highThr, accThr, minRadius, maxRadius)
#Print the circles
615 gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
pupils = list(circles)
617 if (circles !=None):
    #print circles
619     all_circles = circles [0]
M,N = all_circles.shape
621 k=1
    for c in all_circles:
        cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
M),k*128,0))
        K=k+1
625     #Circle with max votes
    c=all_circles [0,:]
627     cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255))
cv2.imshow("hough",gColor)
629 return pupils

631 def detectIrisHough(gray, accThr=600):
#Using the Hough transform to detect ellipses
633 blur = cv2.GaussianBlur(gray, (11,11),9)
##Pupil parameters
635 dp = 6; minDist = 10
highThr = 30 #High threshold for canny
637 #accThr = 600; #accumulator threshold for the circle centers
    at the detection stage. The smaller it is, the more false
    circles may be detected
maxRadius = 150;
minRadius = 100;
#See help for http://opencv.itseez.com/modules/imgproc/doc/
    feature_detection.html?highlight=houghcircle#cv2.
    HoughCirclesIn thus
641 circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
    minDist, None, highThr, accThr, minRadius, maxRadius)
#Print the circles
643 gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
irises = []

```

```

645 if (circles !=None):
646     #print circles
647     all_circles = circles [0]
648     M,N = all_circles.shape
649     k=1
650     for c in all_circles:
651         irises.append(c)
652         cv2.circle(gColor , (int(c[0]) ,int(c[1])) ,c[2] , ( int(k*255/
653             M) ,k*128 ,0))
654         K=k+1
655         #Circle with max votes
656         c=all_circles [0 ,:]
657         cv2.circle(gColor , (int(c[0]) ,int(c[1])) ,c[2] , (0 ,0 ,255))
658         cv2.imshow("hough",gColor)
659         return irises
660 #
661 #
662 #
663 def setText(dst , (x, y), s):
664     cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0 ,
665                 (0, 0, 0), thickness = 2, lineType=cv2.CV_AA)
666     cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0 , (255,
667                 255, 255), lineType=cv2.CV_AA)
668
669 vertical_window_size = 523
670 horizontal_window_size = 640
671
672 def setupWindowSliders():
673     ''' Define windows for displaying the results and create
674         trackbars '''
675     cv2.namedWindow("Result")
676     cv2.moveWindow("Result", 0, 0)
677     cv2.namedWindow('Threshold')
678     cv2.moveWindow("Threshold", 0, vertical_window_size)
679     cv2.namedWindow('Controls')
680     cv2.moveWindow("Controls", horizontal_window_size, 0)
681     cv2.resizeWindow('Controls', horizontal_window_size, 0)
682     cv2.namedWindow("TempResults")
683     cv2.moveWindow("TempResults", horizontal_window_size,
684                     vertical_window_size)
685     #Threshold value for the pupil intensity
686     cv2.createTrackbar('pupilThr','Controls',
687                         default_pupil_threshold, 255, onSlidersChange)
688     #Threshold value for the glint intensities
689     cv2.createTrackbar('glintThr','Controls', 240, 255,
690                         onSlidersChange)
691     #define the minimum and maximum areas of the pupil
692     cv2.createTrackbar('minSize','Controls', 20, 2000,
693                         onSlidersChange)
694     cv2.createTrackbar('maxSize','Controls', 2000,2000,
695                         onSlidersChange)
696     #Value to indicate whether to run or pause the video
697     cv2.createTrackbar('Stop/Start','Controls', 0,1,

```

```

    onSlidersChange)

691 def getSliderVals():
    '''Extract the values of the sliders and return these in a
    dictionary'''
693 sliderVals={}
    sliderVals['pupilThr'] = cv2.getTrackbarPos('pupilThr', 'Controls')
695 sliderVals['glintThr'] = cv2.getTrackbarPos('glintThr', 'Controls')
    sliderVals['minSize'] = 50*cv2.getTrackbarPos('minSize', 'Controls')
697 sliderVals['maxSize'] = 50*cv2.getTrackbarPos('maxSize', 'Controls')
    sliderVals['Running'] = 1==cv2.getTrackbarPos('Stop/Start', 'Controls')
699 return sliderVals

701 def onSlidersChange(dummy=None):
    ''' Handle updates when slides have changed.
703 This function only updates the display when the video is put
    on pause'''
    global imgOrig;
705 sv=getSliderVals()
    if(not sv['Running']): # if pause
        update(imgOrig)

709 def euclidianDistance(a,b):
    a_x, a_y = a
711    b_x, b_y = b
    return math.sqrt((a_x - b_x) ** 2 + (a_y - b_y) **2)
713 #
#_____
715 #         main
#
#_____
717 run(inputFile)

```