

# Eye Tracking

## SIGB Spring 2014

Marcus Gregersen  
[mabg@itu.dk](mailto:mabg@itu.dk)

Martin Faartoft  
[mlfa@itu.dk](mailto:mlfa@itu.dk)

Mads Westi  
[mwek@itu.dk](mailto: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.

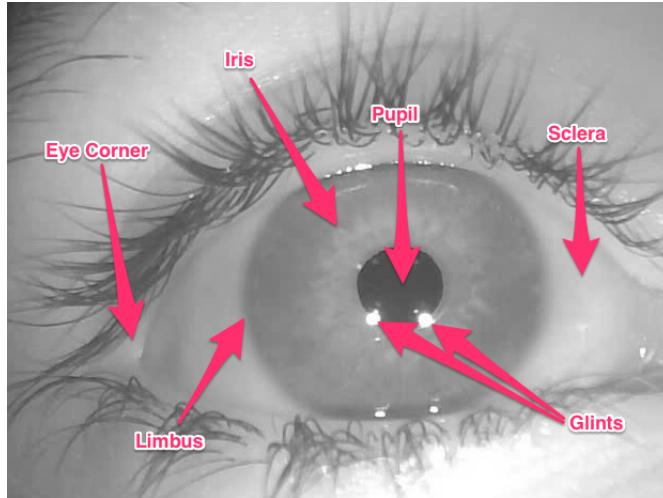


Figure 1: Names and positions of major eye features

# 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.

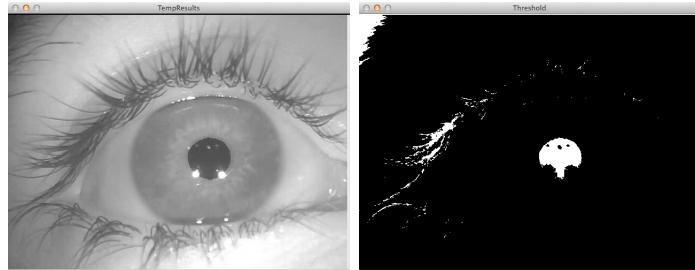


Figure 2: Thresholding eye1.avi

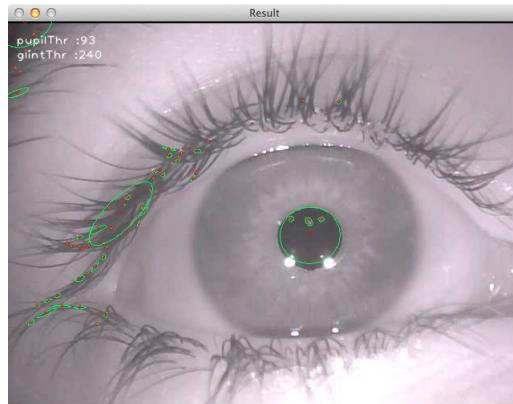


Figure 3: Fitting ellipses on blobs from eye1.avi (green figures are ellipses fitted through blobs, red dots are the centerpoint of each blob)

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.

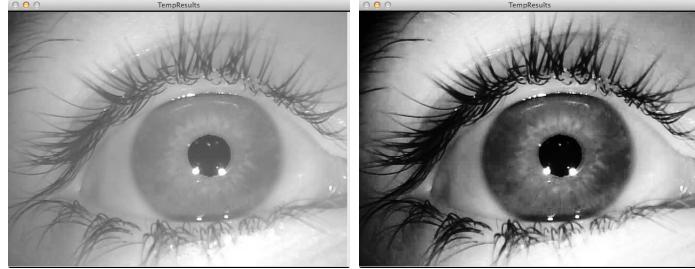


Figure 4: Eye1 before and after Histogram Equalization

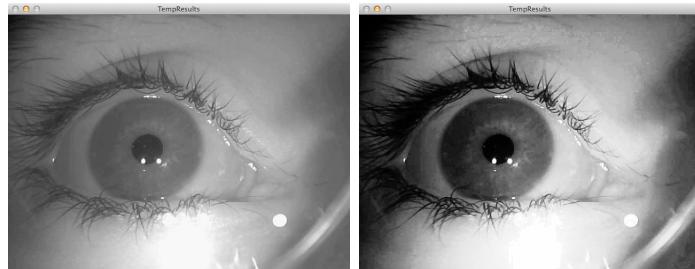


Figure 5: Eye3 before and after Histogram Equalization

**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.

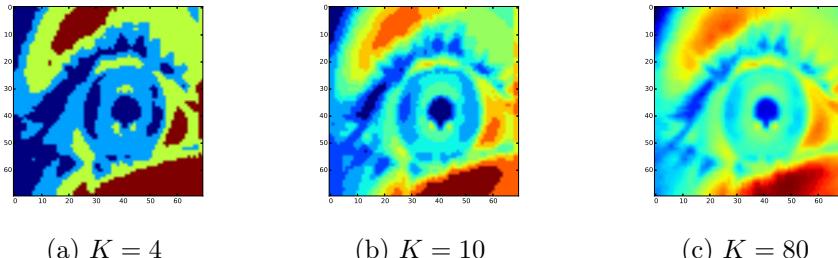


Figure 7: K-means for different values of  $K$

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.

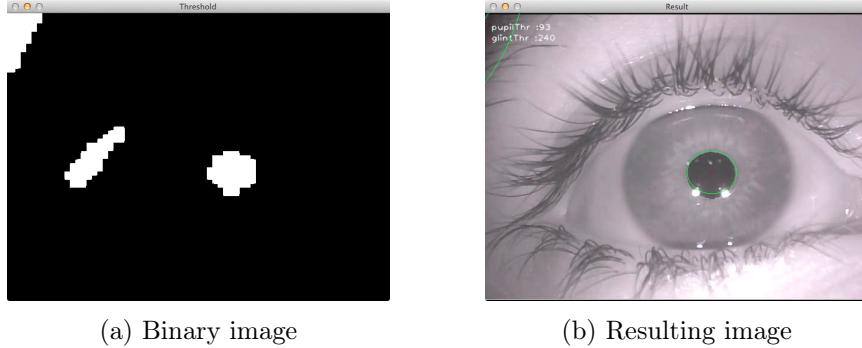


Figure 8: Pupil detection using k-means

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

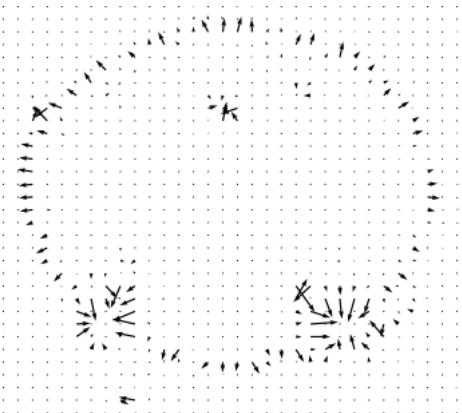


Figure 9: Quiver plot of Eye1 gradients (zoomed on pupil area)

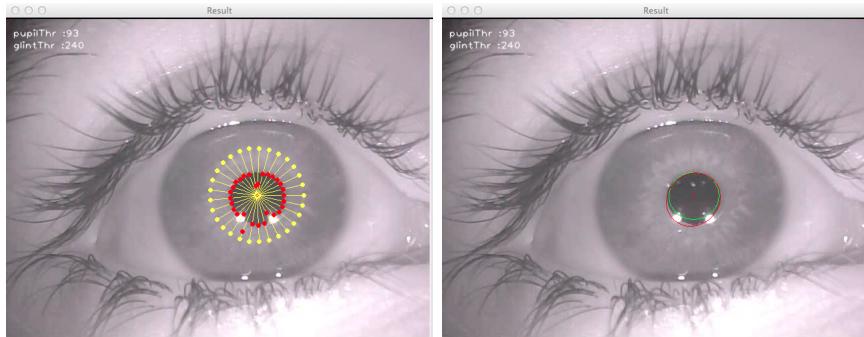


Figure 10: Left: Eye1 showing line-segments for gradient magnitude maximization (yellow) and maximum gradient values along the lines (red). Right: Eye1 showing pupil approximation (green), and new pupil detection (red)

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.

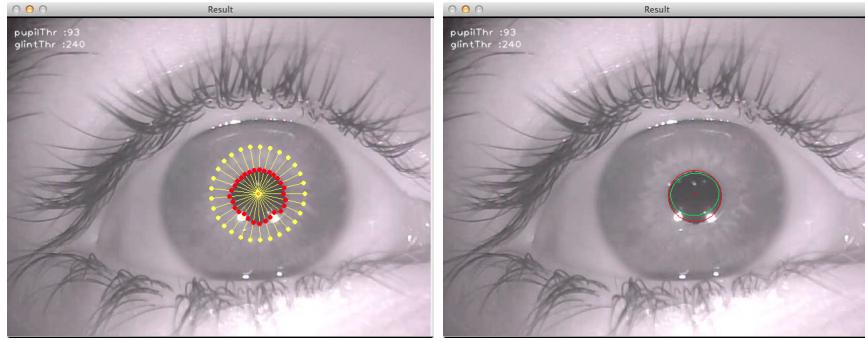


Figure 11: as Figure 10, but pre-processed with a 9x9 Gaussian Blur

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

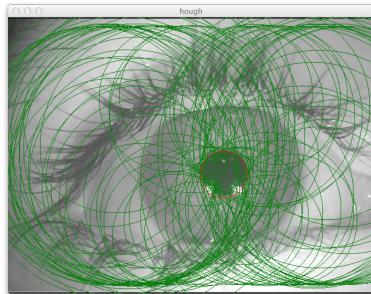
**Gauss kernel size** The size of the gaussian kernel that is applied to the image before the Hough transformation

**$\sigma$  - value** the standard deviation value used to construct the gaussian kernel.

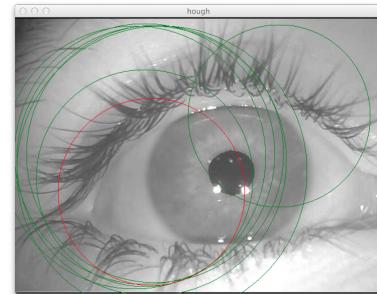
**Accumulator threshold** The minimum cumulative vote threshold in which some parameters for a circle are considered.

**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

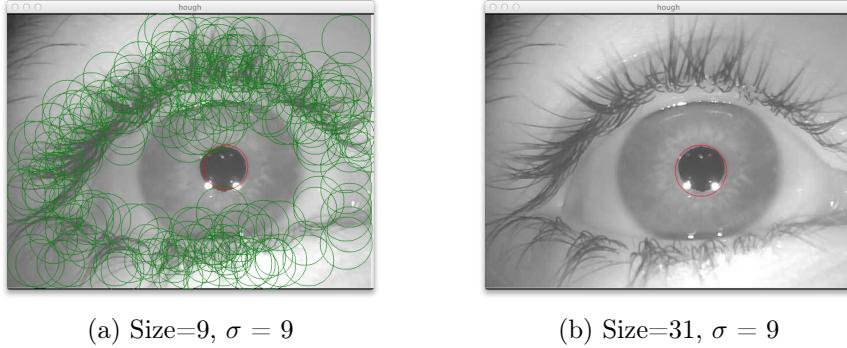


Figure 13: Preprocessing by smoothing with a gaussian kernel

**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.

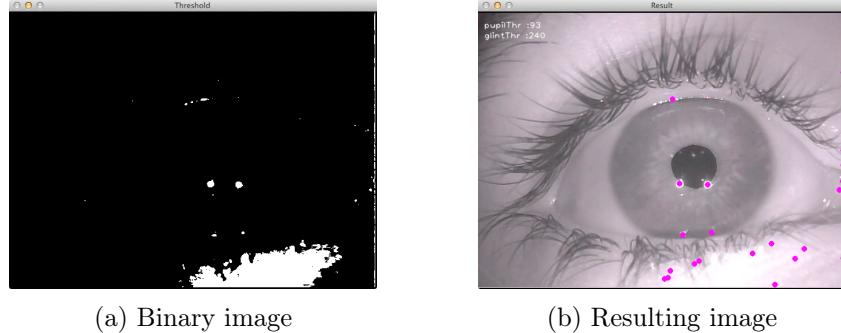


Figure 14: Glint detection with threshold of 240

**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

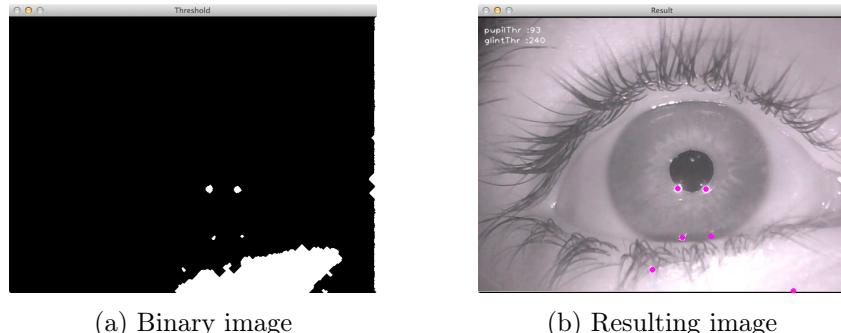


Figure 15: Glint detection with threshold of 240 - using morphology

**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.

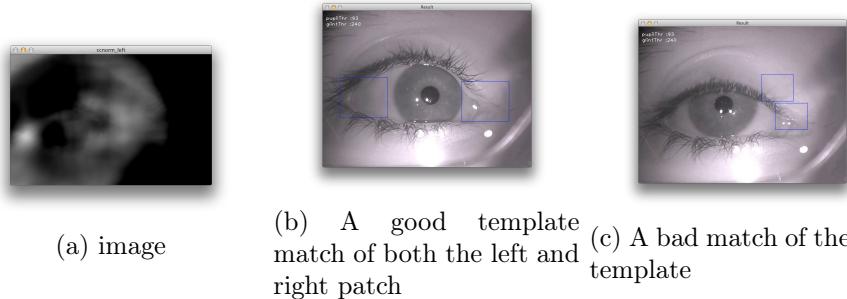


Figure 17: Template Matching

**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.

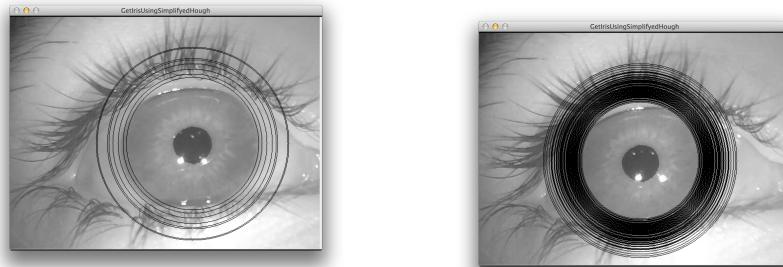
### 5.2 Hough Transformation with position prior

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

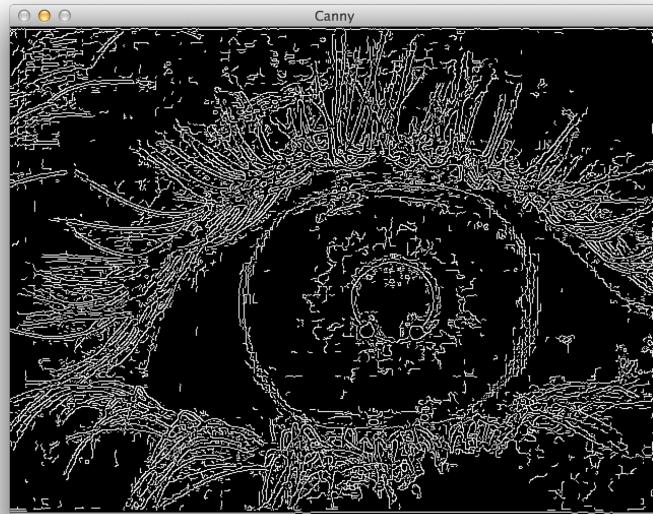


Figure 19: The canny edge image on which the circles in figure (a) are found

**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.

**Performance** We have recorded our own sequence: 'Julie.avi', using an iSlim 321r camera with infrared capabilities. Figure TODO 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.

## References

## Appendix

### 6.1 Assignment1.py

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

17

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

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

35
def GetPupil(gray, thr, min_val, max_val):
36   '''Given a gray level image, gray and threshold value return a
      list of pupil locations'''
37   #tempResultImg = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR) #used
      to draw temporary results
38
39   #Threshold image to get a binary image
40   #gray = cv2.equalizeHist(gray)
41   cv2.imshow("TempResults", gray)
42   val, binI =cv2.threshold(gray, thr, 255, cv2.THRESH_BINARY_INV)
#print val
```

```

45  #Morphology (close image to remove small 'holes' inside the
46  #    pupil area)
47  st = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
48  #binI = cv2.morphologyEx(binI, cv2.MORPH_OPEN, st, iterations
49  #    =10)
50  #binI = cv2.morphologyEx(binI, cv2.MORPH_CLOSE, st, iterations
51  #    =5)
52  cv2.imshow("Threshold",binI)
53  #Calculate blobs, and do edge detection on entire image (
54  #    modifies binI)
55  contours, hierarchy = cv2.findContours(binI, cv2.RETR_LIST,
56  #    cv2.CHAIN_APPROX_SIMPLE)

57  pupils = [];
58  prop_calc = RegionProps()
59  centroids = []
60  for contour in contours:
61      #calculate centroid, area and 'extend' (compactness of
62      #contour)
63      props = prop_calc.CalcContourProperties(contour, ["centroid",
64      "area", "extend"])
65      x, y = props["Centroid"]
66      area = props["Area"]
67      extend = props["Extend"]
68      #filter contours, so that their area lies between min_val
69      #and max_val, and then extend lies between 0.4 and 1.0
70      if (area > min_val and area < max_val and extend > 0.5 and
71      extend < 1.0):
72          pupilEllipse = cv2.fitEllipse(contour)
73          # center, radii, angle = pupilEllipse
74          # max_radius = max(radii)
75          # c_x = int(center[0])
76          # c_y = int(center[1])
77          # cv2.circle(tempResultImg,(c_x,c_y), int(max_radius),
78          #(0,0,255),4) #draw a circle
79          #cv2.ellipse(tempResultImg, pupilEllipse,(0,255,0),1)
80          pupils.append(pupilEllipse)

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

82  return pupils

83 def GetGlints(gray,thr):
84     min_area = 10
85     max_area = 150
86     ''' Given a gray level image, gray and threshold
87     value return a list of glint locations '''
88     #print thr
89     val, binary_image = cv2.threshold(gray, thr, 255, cv2.
90         THRESH_BINARY)

91
92     st = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
93     binary_image = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE,

```

```

    st , iterations=8)
binary_image = cv2.morphologyEx(binary_image , cv2.MORPH_OPEN,
    st , iterations=2)
89

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

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

105 #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_area and area < max_area: #and extend > 0.4
and extend < 1.0:
107         glints.append((x,y))
        #print x, y, area, extend
109         #cv2.circle(tempResultImg,( int(x) ,int(y)) , 2, (0,0,255) ,4)
        #draw a circle
    #cv2.imshow("TempResults" ,tempResultImg)

111 return glints
113
114 def GetIrisUsingThreshold(gray , thr , min_val , max_val):
115     ''' Given a gray level image , gray and threshold
value return a list of iris locations '''
116     val,binary_image = cv2.threshold(gray , thr , 255 , cv2.
        THRESH_BINARY_INV)
117     cv2.imshow("Threshold" , binary_image)
118
119 contours , hierarchy = cv2.findContours(binary_image , cv2.
    RETR_LIST , cv2.CHAIN_APPROX_SIMPLE)
120
121 irises = [];
122 prop_calc = RegionProps()
123 centroids = []
124 for contour in contours:
#calculate centroid , area and 'extend' (compactness of
contour)
    props = prop_calc.CalcContourProperties(contour , [ "centroid"
        , "area" , "extend"])
    x, y = props[ "Centroid" ]

```

```

129     area = props[ "Area" ]
130     extend = props[ "Extend" ]
131     #filter contours, so that their area lies between min_val
132     #and max_val, and then extend lies between 0.4 and 1.0
133     if area > min_val and area < max_val and extend > 0.5 and
134     extend < 1.0:
135         irisEllipse = cv2.fitEllipse(contour)
136         # center , radii , angle = pupilEllipse
137         # max_radius = max(radii)
138         # c_x = int(center[0])
139         # c_y = int(center[1])
140         # cv2.circle(tempResultImg,(c_x,c_y), int(max_radius),
141         # (0,0,255),4) #draw a circle
142         #cv2.ellipse(tempResultImg, pupilEllipse ,(0,255,0),1)
143         irises.append(irisEllipse)
144
145     return irises
146
147 def circularHough(gray):
148     ''' Performs a circular hough transform of the image, gray and
149     shows the detected circles
150     The circe with most votes is shown in red and the rest in
151     green colors '''
152     #See help for http://opencv.itseez.com/modules/imgproc/doc/
153     # feature_detection.html?highlight=houghcircle#cv2.HoughCircles
154     blur = cv2.GaussianBlur(gray, (31,31), 11)
155
156     dp = 6; minDist = 30
157     highThr = 20 #High threshold for canny
158     accThr = 850; #accumulator threshold for the circle centers at
159     #the detection stage. The smaller it is, the more false
160     #circles may be detected
161     maxRadius = 50;
162     minRadius = 155;
163     circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
164     minDist, None, highThr, accThr, maxRadius, minRadius)
165
166     #Make a color image from gray for display purposes
167     gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
168     if (circles !=None):
169         #print circles
170         all_circles = circles[0]
171         M,N = all_circles.shape
172         k=1
173         for c in all_circles:
174             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
175             M),k*128,0))
176             K=k+1
177             c=all_circles[0,:]
178             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255),5)
179             cv2.imshow("hough",gColor)
180
181 def simplifiedHough(edgeImage, circleCenter, minR, maxR, N, thr):
182     samplePoints = 100
183     accumulator_line = {}

```

```

173     for radius in range(minR, maxR, N):
175
176         points = getCircleSamples(center=circleCenter, radius=radius
177             , nPoints=samplePoints)
178         accumulator_line[radius] = 0
179         for point in points:
180             try:
181                 if edgeImage[point[0], point[1]] > 0:
182                     accumulator_line[radius] += 1
183             except IndexError:
184                 continue
185         radii = []
186         for radius in accumulator_line:
187             value = accumulator_line[radius]
188             if value > thr:
189                 radii.append(radius)
190             print value
191         return radii
192
193     def GetIrisUsingNormals(gray, pupil, normalLength):
194         ''' Given a gray level image, gray and the length of the
195             normals, normalLength
196             return a list of iris locations '''
197     # YOUR IMPLEMENTATION HERE !!!!!
198     pass
199
200     def GetIrisUsingSimplifyedHough(gray, pupil):
201         ''' Given a gray level image, gray
202             return a list of iris locations using a simplified Hough
203             transformation '''
204         if pupil:
205             edges = cv2.Canny(gray, 30, 20)
206             pupil = pupil[0]
207             pupil_x = int(pupil[0])
208             pupil_y = int(pupil[1])
209             #cv2.imshow("edges", edges)
210             radii = simplifiedHough(edges, pupil, 120, 200, 1, 15)
211             for radius in radii:
212                 cv2.circle(gray, (pupil_x, pupil_y), radius, (0, 0, 0), 1)
213                 #cv2.circle(gray, pupil, radius, (127,127,127), 2)
214             cv2.imshow("GetIrisUsingSimplifyedHough", gray)
215
216     def plotVectorField(I):
217         g_x = cv2.Sobel(I, cv.CV_64F, 1, 0, ksize=3)
218         g_y = cv2.Sobel(I, cv.CV_64F, 0, 1, ksize=3) # ksize=3 som ***
219         kwargs
220
221         x_orig_dim, y_orig_dim = I.shape
222         x_mesh_dim, y_mesh_dim = (3, 3)

```

```

223     sample_g_x = g_x[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
224         x_mesh_dim]
225     sample_g_y = g_y[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
226         x_mesh_dim]
227     quiver(sample_g_x, sample_g_y)
228     show()
229
231 def getGradientImageInfo(I):
232     g_x = cv2.Sobel(I, cv.CV_64F, 1,0)
233     g_y = cv2.Sobel(I, cv.CV_64F, 0,1) # ksize=3 som **kwargs
234
235     X,Y = I.shape
236     orientation = np.zeros(I.shape)
237     magnitude = np.zeros(I.shape)
238     sq_g_x = cv2.pow(g_x, 2)
239     sq_g_y = cv2.pow(g_y, 2)
240     fast_magnitude = cv2.pow(sq_g_x + sq_g_y, .5)
241
242     # for x in range(X):
243     #     for y in range(Y):
244     #         orientation[x][y] = np.arctan2(g_y[x][y], g_x[x][y]) *
245     #             (180 / math.pi)
246     #         magnitude[x][y] = math.sqrt(g_y[x][y] ** 2 + g_x[x][y]
247     #             ** 2)
248
249     #print fast_magnitude[0]
250     #print magnitude[0]
251
252     return fast_magnitude, orientation
253
254 def GetEyeCorners(orig_img, leftTemplate, rightTemplate,
255     pupilPosition=None):
256     if leftTemplate != [] and rightTemplate != []:
257         ccnorm_left = cv2.matchTemplate(orig_img, leftTemplate, cv2.
258             TM_CCOEFF_NORMED)
259         ccnorm_right = cv2.matchTemplate(orig_img, rightTemplate,
260             cv2.TM_CCOEFF_NORMED)
261
262         minVal, maxVal, minLoc, maxloc_left_from = cv2.minMaxLoc(
263             ccnorm_left)
264         minVal, maxVal, minLoc, maxloc_right_from, = cv2.minMaxLoc(
265             ccnorm_right)
266
266         l_x,l_y = leftTemplate.shape
267         max_loc_left_from_x = maxloc_left_from[0]
268         max_loc_left_from_y = maxloc_left_from[1]
269
270         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

```

```

269     maxloc_left_to = (max_loc_left_to_x, max_loc_left_to_y)
270
271     r_x,r_y = leftTemplate.shape
272     max_loc_right_from_x = maxloc_right_from[0]
273     max_loc_right_from_y = maxloc_right_from[1]
274
275     max_loc_right_to_x = max_loc_right_from_x + r_x
276     max_loc_right_to_y = max_loc_right_from_y + r_y
277     maxloc_right_to = (max_loc_right_to_x, max_loc_right_to_y)
278
279     return (maxloc_left_from, maxloc_left_to, maxloc_right_from,
280             maxloc_right_to)
281
282     bgr_yellow = 0,255,255
283     bgr_blue = 255, 0, 0
284     bgr_red = 0, 0, 255
285     def circleTest(img, center_point):
286         nPts = 20
287         circleRadius = 100
288         P = getCircleSamples(center=center_point, radius=circleRadius,
289                             nPoints=nPts)
290         for (x,y,dx,dy) in P:
291             point_coords = (int(x),int(y))
292             cv2.circle(img, point_coords, 2, bgr_yellow, 2)
293             cv2.line(img, point_coords, center_point, bgr_yellow)
294
295     def findEllipseContour(img, gradient_magnitude,
296                           gradient_orientation, estimatedCenter, estimatedRadius, nPts
297                           =30):
298         center_point_coords = (int(estimatedCenter[0]), int(
299             estimatedCenter[1]))
300         P = getCircleSamples(center = estimatedCenter, radius =
301                             estimatedRadius, nPoints=nPts)
302         for (x,y,dx,dy) in P:
303             point_coords = (int(x),int(y))
304             cv2.circle(img, point_coords, 2, bgr_yellow, 2)
305             cv2.line(img, point_coords, center_point_coords, bgr_yellow)
306
307         newPupil = np.zeros((nPts,1,2)).astype(np.float32)
308         t = 0
309         for (x,y,dx,dy) in P:
310             #< define normalLength as some maximum distance away from
311             # initial circle >
312             #< get the endpoints of the normal -> p1,p2>
313             point_coords = (int(x),int(y))
314             normal_gradient = dx, dy
315             #cv2.circle(img, point_coords, 2, bgr_blue, 2)
316             max_point = findMaxGradientValueOnNormal(gradient_magnitude,
317                                                     gradient_orientation, point_coords, center_point_coords,
318                                                     normal_gradient)
319             cv2.circle(img, tuple(max_point), 2, bgr_red, 2) #locate the
320             # max points
321             #< store maxPoint in newPupil>
322             newPupil[t] = max_point

```

```

        t += 1
313  #<fitPoints to model using least squares- cv2.fitellipse(
            newPupil)>
      return cv2.fitEllipse(newPupil)

315
316  def findMaxGradientValueOnNormal(gradient_magnitude,
317      gradient_orientation, p1, p2, normal_orientation):
318      #Get integer coordinates on the straight line between p1 and
319      p2
320      pts = SIGBTools.getLineCoordinates(p1, p2)
321      values = gradient_magnitude[pts[:,1],pts[:,0]]
322      #orientations = gradient_orientation[pts[:,1],pts[:,0]]
323      #normal_angle = np.arctan2(normal_orientation[1],
324          normal_orientation[0]) * (180 / math.pi)

325      # orientation_difference = abs(orientations - normal_angle)
326      # print orientation_difference[0:10]
327      # max_index = 0 #np.argmax(values)
328      # max_value = 0
329      # for index in range(len(values)):
330          # if orientation_difference[index] < 20:
331              # if values[index] > max_value:
332                  max_index = index
333                  max_value = values[index]
334      #print orientations[max_index], normal_angle
335      max_index = np.argmax(values)
336      return pts[max_index]
337      #return coordinate of max value in image coordinates

338  def FilterPupilGlint(pupils, glints):
339      ''' Given a list of pupil candidates and glint candidates
340          returns a list of pupil and glints '''
341      filtered_glints = []
342      filtered_pupils = pupils
343      for glint in glints:
344          for pupil in pupils:
345              if (is_glint_close_to_pupil(glint, pupil)):
346                  filtered_glints.append(glint)
347
348      return filtered_pupils, filtered_glints

349  def is_glint_close_to_pupil(glint, pupil):
350      center, radii, angle = pupil
351      max_radius = max(radii)
352      distance = euclidianDistance(center, glint)
353      return (distance < max_radius)

354  def filterGlintsIris(glints, irises):
355      new_glints = []
356      if glints and irises:
357          for glint in glints:
358              for iris in irises:
359                  iris_x, iris_y, iris_radius = iris
360                  print glint

```

```

361         iris_vector = np.array([iris_x, irix_y])
362         distance = np.linalg.norm(glint - iris_vector)
363         if distance < iris_radius:
364             new_glints.append(glint)
365             #print iris
366             return new_glints
367
368
369     def update(I):
370         '''Calculate the image features and display the result based
371             on the slider values'''
372         #global drawImg
373         global frameNr,drawImg, gray
374         img = I.copy()
375         sliderVals = getSliderVals()
376         gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
377         #gray = cv2.GaussianBlur(gray, (9,9),9)
378
379         # Do the magic
380         pupils = GetPupil(gray,sliderVals['pupilThr'], sliderVals['
381             minSize'], sliderVals['maxSize'])
382         glints = GetGlints(gray,sliderVals['glintThr'])
383         #pupils, glints = FilterPupilGlint(pupils,glints)
384         #irises = GetIrisUsingThreshold(gray, sliderVals['pupilThr'],
385             sliderVals['minSize'], sliderVals['maxSize'])
386
387         K=10
388         d=40
389         #labelIm, centroids = detectPupilKMeans(gray,K=K,distanceWeight
390             =d,reSize=(70,70))
391         #pupils = get_pupils_from_kmean(labelIm,centroids,gray,
392             sliderVals['minSize'],sliderVals['maxSize'])
393
394         magnitude, orientation = getGradientImageInfo(gray)
395         if pupils:
396             GetIrisUsingSimplifedHough(gray, pupils[0])
397
398         #plotVectorField(gray)
399         #Do template matching
400         global leftTemplate
401         global rightTemplate
402
403         #corners = GetEyeCorners(gray, leftTemplate, rightTemplate)
404
405         #detectPupilHough(gray, 100)
406         #irises = detectIrisHough(gray, 400)
407
408         #glints = filterGlintsIris(glints,irises)
409
410         #Display results
411         global frameNr,drawImg
412         x,y = 10,10
413         #setText(img,(x,y),"Frame:%d" %frameNr)

```

```

sliderVals = getSliderVals()

# for non-windows machines we print the values of the
# threshold in the original image
if sys.platform != 'win32':
    step=18
    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)
cv2.imshow('Result', img)

#Uncomment these lines as your methods start to work to
#display the result in the
#original image

for pupil in pupils:
    #cv2.ellipse(img, pupil, (0,255,0), 1)
    C = int(pupil[0][0]), int(pupil[0][1])
    contour = findEllipseContour(img, magnitude, orientation, C,
        70)
    cv2.ellipse(img, contour, bgr_red, 1)
    cv2.circle(img, C, 2, (0,0,255), 1)
    #circleTest(img, C)
    for glint in glints:
        C = int(glint[0]), int(glint[1])
        #cv2.circle(img, C, 2, (255,0,255), 5)

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

# for iris in irises:
#     cv2.ellipse(img, iris, (0,255,0), 1)
#     C = int(iris[0][0]), int(iris[0][1])
#     cv2.circle(img, C, 2, (0,0,255), 4)

cv2.imshow("Result", img)

#For Iris detection - Week 2
#circularHough(gray)

#copy the image so that the result image (img) can be saved in
#the movie
drawImg = img.copy()

def printUsage():

```

```

        print "Q or ESC: Stop"
457     print "SPACE: Pause"
        print "r: reload video"
459     print 'm: Mark region when the video has paused'
        print 's: toggle video writing'
461     print 'c: close video sequence'

463 def run(fileName ,resultFile='eyeTrackingResults.avi'):

465     ''' MAIN Method to load the image sequence and handle user
        inputs '''
        global imgOrig , frameNr ,drawImg , leftTemplate , rightTemplate ,
        gray
467     setupWindowSliders()
        props = RegionProps()
469     cap ,imgOrig ,sequenceOK = getImageSequence(fileName)
        videoWriter = 0

471     frameNr =0
473     if(sequenceOK):
        update(imgOrig)
475     printUsage()
        frameNr=0;
477     saveFrames = False

479     while(sequenceOK):
        sliderVals = getSliderVals();
481     frameNr=frameNr+1
        ch = cv2.waitKey(1)
483     #Select regions
        if(ch==ord('m')):
485         if(not sliderVals['Running']):
            roiSelect=ROISelector(imgOrig)
487         pts ,regionSelected= roiSelect.SelectArea('Select eye
corner',(400,200))
            if(regionSelected):
489             if leftTemplate == []:
                leftTemplate = gray[pts [0][1]:pts [1][1] ,pts [0][0]:
                pts [1][0]]
            else:
                rightTemplate = gray[pts [0][1]:pts [1][1] ,pts [0][0]:
                pts [1][0]]
491
493     if ch == 27:
495         break
496     if (ch==ord('s')):
497         if((saveFrames)):
498             videoWriter.release()
499             saveFrames=False
500             print "End recording"
501         else:
502             imSize = np.shape(imgOrig)
503             videoWriter = cv2.VideoWriter(resultFile , cv.CV_FOURCC(
'D','I','V','3'), 15.0 ,(imSize [1] ,imSize [0]) ,True) #Make a

```

```

        video writer
            saveFrames = True
505        print "Recording..."
507

509    if(ch==ord('q')):
510        break
511    if(ch==32): #Spacebar
512        sliderVals = getSliderVals()
513        cv2.setTrackbarPos('Stop/Start','Controls',not sliderVals[
514            'Running'])
515    if(ch==ord('r')):
516        frameNr =0
517        sequenceOK=False
518        cap,imgOrig,sequenceOK = getImageSequence(fileName)
519        update(imgOrig)
520        sequenceOK=True

521    sliderVals=getSliderVals()
522    if(sliderVals['Running']):
523        sequenceOK, imgOrig = cap.read()
524        if(sequenceOK): #if there is an image
525            update(imgOrig)
526        if(saveFrames):
527            videoWriter.write(drawImg)
528        if(videoWriter!=0):
529            videoWriter.release()
530            print "Closing videofile..."
531 #
532

533 def detectPupilKMeans(gray,K=2,distanceWeight=2,reSize=(40,40)):
534     ''' Detects the pupil in the image, gray, using k-means
535         gray           : grays scale image
536         K              : Number of clusters
537         distanceWeight : Defines the weight of the position
538             parameters
539         reSize          : the size of the image to do k-means on
540     '''
541     #Resize for faster performance
542     smallI = cv2.resize(gray, reSize)
543     smallI = cv2.GaussianBlur(smallI,(3,3),20)
544     M,N = smallI.shape
545     #Generate coordinates in a matrix
546     X,Y = np.meshgrid(range(M),range(N))
547     #Make coordinates and intensity into one vectors
548     z = smallI.flatten()
549     x = X.flatten()
550     y = Y.flatten()
551     O = len(x)
552     #make a feature vectors containing (x,y,intensity)
553     features = np.zeros((O,3))
554     features[:,0] = z;
555     features[:,1] = y/distanceWeight; #Divide so that the distance

```

```

    of position weighs less than intensity
555 features [:,2] = x/distanceWeight;
556 features = np.array(features , 'f')
557 # cluster data
558 centroids , variance = kmeans(features ,K)
559 #use the found clusters to map
560 label , distance = vq(features , centroids)
561 # re-create image from
562 labelIm = np.array(np.reshape(label ,(M,N)))
563 return labelIm , centroids

565 def get_pupils_from_kmean(labelIm , centroids , gray , min_val ,
566 max_val):
567 result = np.zeros((labelIm .shape))
568 label = np.argmin(centroids [:,0])
569 result [labelIm == label] = [255]
570 y,x=gray .shape
571 result = cv2.resize(result ,(x,y))
572 semi_binI = np.array(result , dtype='uint8 ')
573 #remove gray elements created from the linear interpolation
574 val,binI =cv2.threshold(semi_binI , 0, 255 , cv2.THRESH_BINARY)
575 cv2.imshow("Threshold",binI)
576 #Calculate blobs , and do edge detection on entire image (
577 # modifies binI)
578 contours , hierarchy = cv2.findContours(binI , cv2.RETR_LIST,
579 cv2.CHAIN_APPROX_SIMPLE)

580 pupils = [];
581 prop_calc = RegionProps()
582 for contour in contours:
583 #calculate centroid , area and 'extend' (compactness of
584 #contour)
585 props = prop_calc.CalcContourProperties(contour , ["centroid"
586 , "area" , "extend"])
587 x , y = props["Centroid"]
588 area = props["Area"]
589 extend = props["Extend"]
590 #filter contours , so that their area lies between min_val
591 and max_val , and then extend lies between 0.4 and 1.0
592 if (area > min_val and area < max_val and extend > 0.4 and
593 extend < 1.0):
594     pupilEllipse = cv2.fitEllipse(contour)
595     pupils.append(pupilEllipse)
596 return pupils

597 def detectPupilHough(gray , accThr=600):
598 #Using the Hough transform to detect ellipses
599 blur = cv2.GaussianBlur(gray , (9,9),9)
600 ##Pupil parameters
601 dp = 6; minDist = 10
602 highThr = 30 #High threshold for canny
603 #accThr = 600; #accumulator threshold for the circle centers
604 # at the detection stage. The smaller it is , the more false
605 # circles may be detected

```

```

599     maxRadius = 50;
600     minRadius = 30;
601 #See help for http://opencv.itseez.com/modules/imgproc/doc/
602     feature_detection.html?highlight=houghcircle#cv2.
603     HoughCirclesIn thus
604     circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
605         minDist, None, highThr, accThr, minRadius, maxRadius)
606 #Print the circles
607     gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
608     pupils = list(circles)
609     if (circles !=None):
610         #print circles
611         all_circles = circles[0]
612         M,N = all_circles.shape
613         k=1
614         for c in all_circles:
615             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
616             M),k*128,0))
617             K=k+1
618             #Circle with max votes
619             c=all_circles[0,:]
620             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255))
621             cv2.imshow("hough",gColor)
622             return pupils
623
624 def detectIrisHough(gray, accThr=600):
625     #Using the Hough transform to detect ellipses
626     blur = cv2.GaussianBlur(gray, (11,11),9)
627     ##Pupil parameters
628     dp = 6; minDist = 10
629     highThr = 30 #High threshold for canny
630     #accThr = 600; #accumulator threshold for the circle centers
631     # at the detection stage. The smaller it is, the more false
632     # circles may be detected
633     maxRadius = 150;
634     minRadius = 100;
635     #See help for http://opencv.itseez.com/modules/imgproc/doc/
636     feature_detection.html?highlight=houghcircle#cv2.
637     HoughCirclesIn thus
638     circles = cv2.HoughCircles(blur, cv2.cv.CV_HOUGH_GRADIENT, dp,
639         minDist, None, highThr, accThr, minRadius, maxRadius)
640 #Print the circles
641     gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
642     irises = []
643     if (circles !=None):
644         #print circles
645         all_circles = circles[0]
646         M,N = all_circles.shape
647         k=1
648         for c in all_circles:
649             irises.append(c)
650             cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
651             M),k*128,0))
652             K=k+1

```

```

643     #Circle with max votes
644     c=all_circles[0,:]
645     cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255))
646     cv2.imshow("hough",gColor)
647     return irises
648 #
649 #           UI related
650 #
651
652 def setText(dst, (x, y), s):
653     cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0,
654                 (0, 0, 0), thickness = 2, lineType=cv2.CV_AA)
655     cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0, (255,
656                 255, 255), lineType=cv2.CV_AA)
657
658 vertical_window_size = 523
659 horizontal_window_size = 640
660
661 def setupWindowSliders():
662     ''' Define windows for displaying the results and create
663         trackbars '''
664     cv2.namedWindow("Result")
665     cv2.moveWindow("Result", 0, 0)
666     cv2.namedWindow('Threshold')
667     cv2.moveWindow("Threshold", 0, vertical_window_size)
668     cv2.namedWindow('Controls')
669     cv2.moveWindow("Controls", horizontal_window_size, 0)
670     cv2.resizeWindow('Controls', horizontal_window_size, 0)
671     cv2.namedWindow("TempResults")
672     cv2.moveWindow("TempResults", horizontal_window_size,
673                   vertical_window_size)
674     #Threshold value for the pupil intensity
675     cv2.createTrackbar('pupilThr', 'Controls',
676                        default_pupil_threshold, 255, onSlidersChange)
677     #Threshold value for the glint intensities
678     cv2.createTrackbar('glintThr', 'Controls', 240, 255,
679                        onSlidersChange)
680     #define the minimum and maximum areas of the pupil
681     cv2.createTrackbar('minSize', 'Controls', 20, 2000,
682                        onSlidersChange)
683     cv2.createTrackbar('maxSize', 'Controls', 2000, 2000,
684                        onSlidersChange)
685     #Value to indicate whether to run or pause the video
686     cv2.createTrackbar('Stop/Start', 'Controls', 0,1,
687                        onSlidersChange)
688
689 def getSliderVals():
690     '''Extract the values of the sliders and return these in a
691         dictionary '''
692     sliderVals={}
693     sliderVals['pupilThr'] = cv2.getTrackbarPos('pupilThr', 'Controls')
694     sliderVals['glintThr'] = cv2.getTrackbarPos('glintThr', 'Controls')

```

```

685     sliderVals[ 'minSize' ] = 50*cv2.getTrackbarPos( 'minSize' , 'Controls')
686     sliderVals[ 'maxSize' ] = 50*cv2.getTrackbarPos( 'maxSize' , 'Controls')
687     sliderVals[ 'Running' ] = 1==cv2.getTrackbarPos( 'Stop/Start' , 'Controls')
688     return sliderVals
689
690     def onSlidersChange(dummy=None):
691         ''' Handle updates when slides have changed.
692             This function only updates the display when the video is put
693             on pause '''
694         global imgOrig;
695         sv=getSliderVals()
696         if(not sv[ 'Running' ]): # if pause
697             update(imgOrig)
698
699         def euclidianDistance(a,b):
700             a_x, a_y = a
701             b_x, b_y = b
702             return math.sqrt((a_x - b_x) ** 2 + (a_y - b_y) **2)
703
704     #-----#
705     #-----#
706     run(inputFile)

```