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.

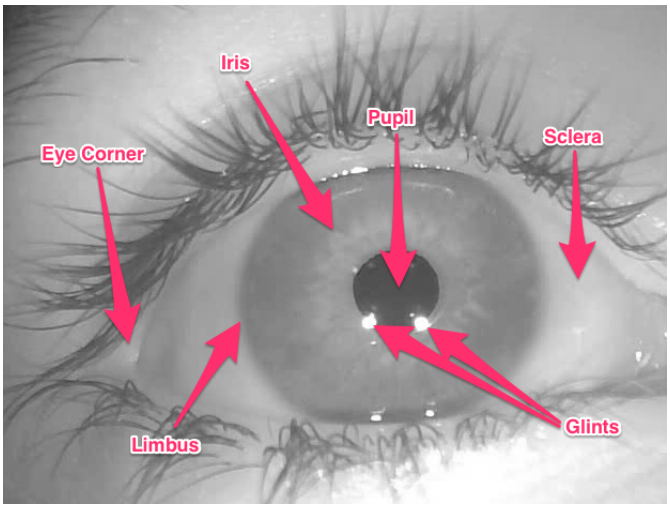


Figure 1: Names and positions of major eye features

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.

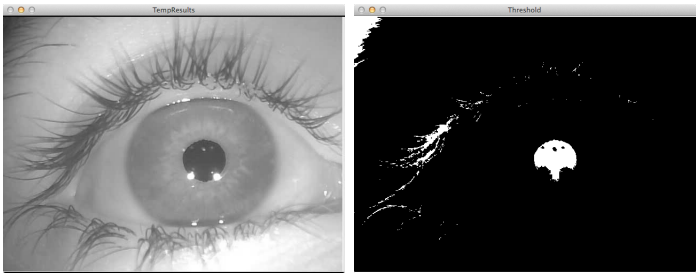


Figure 2: Thresholding eye1.avi

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

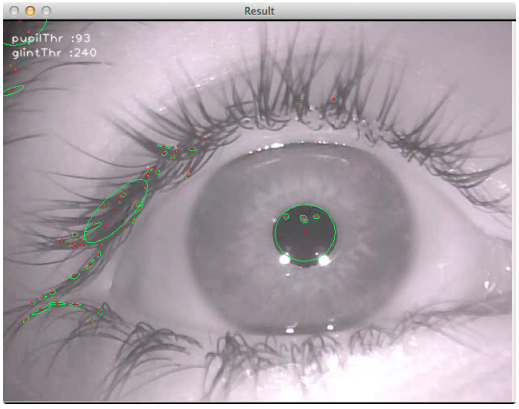


Figure 3: Fitting ellipses on blobs from eye1.avi (green figures are ellipses fitted through blobs, red dots are the centerpoint of each blob)

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.

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 does not generalize to 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.

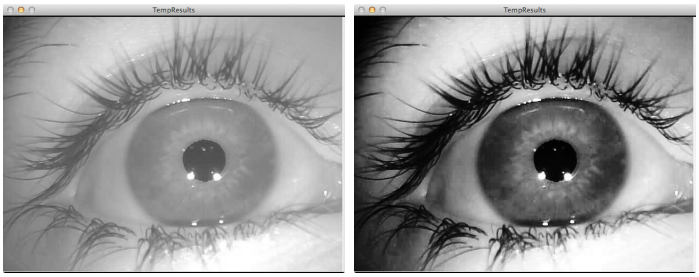


Figure 4: Eye1 before and after Histogram Equalization

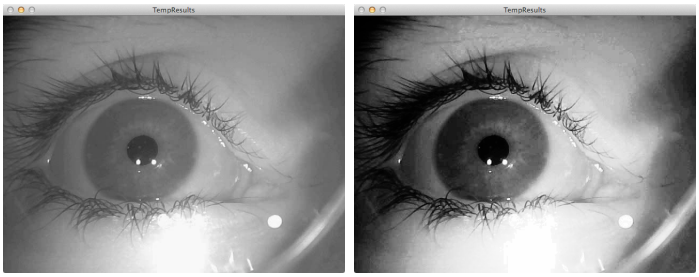


Figure 5: Eye3 before and after Histogram Equalization

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 into something resembling a circle, thereby giving a false positive.

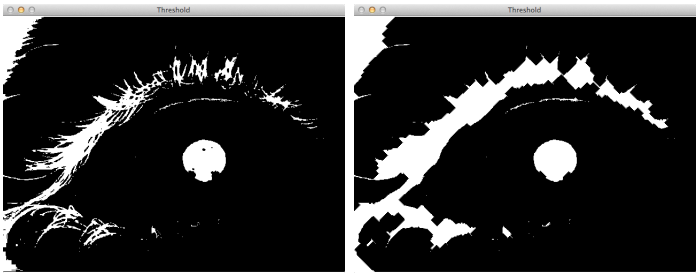


Figure 6: Eye1 before and after Closing (5 iterations, 5x5 CROSS structuring element)

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

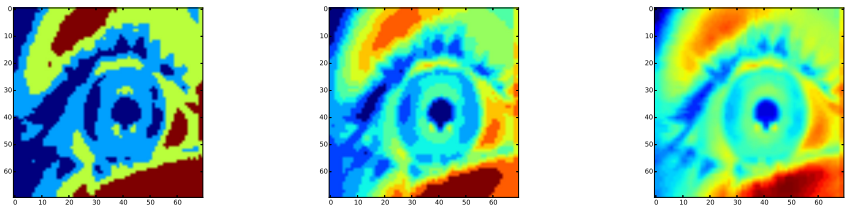
2. One or more pupils were found within the distance, return those

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 the cluster. The value of K is arbitrarily chosen, so that for a sufficiently large number, 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.

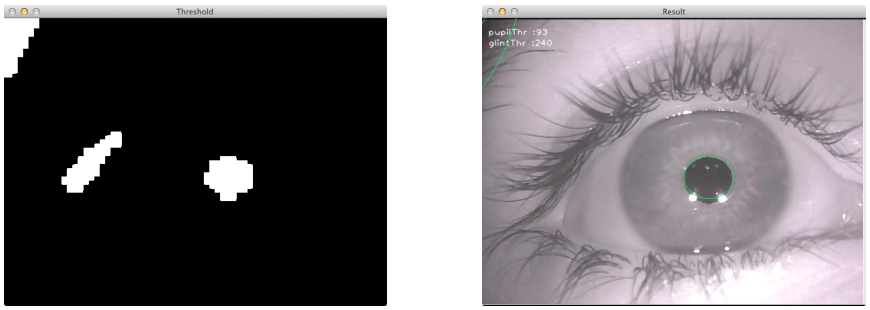


(a) $K = 4$ (b) $K = 10$ (c) $K = 80$

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.



(a) Binary image (b) Resulting image

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 un-

fortunate 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.

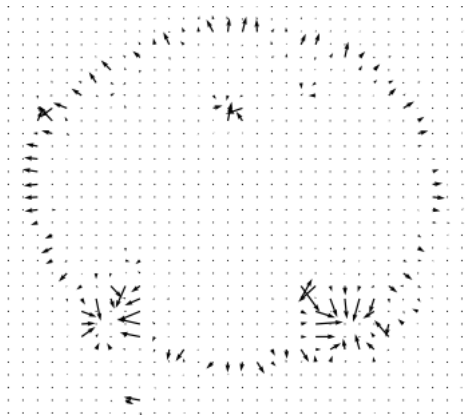


Figure 9: Quiver plot of Eye1 gradients (zoomed on pupil area)

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.

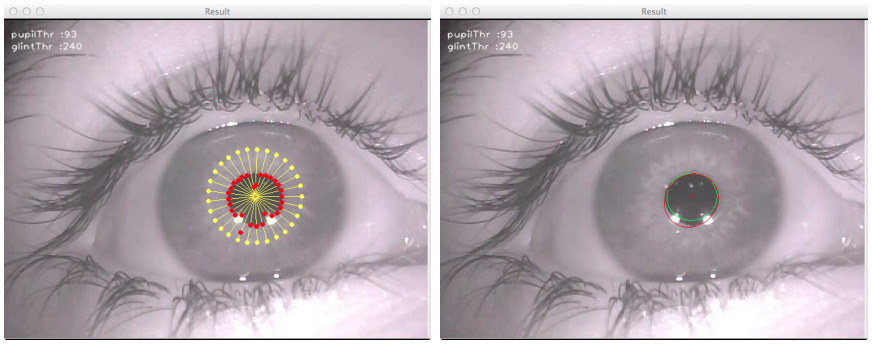


Figure 10: Left: Eye1 showing line-segments for gradient magnitude maximization (yellow) and maximum gradient values along the lines (red). Right: Eye1 showing pupil approximation (green), and new pupil detection (red)

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

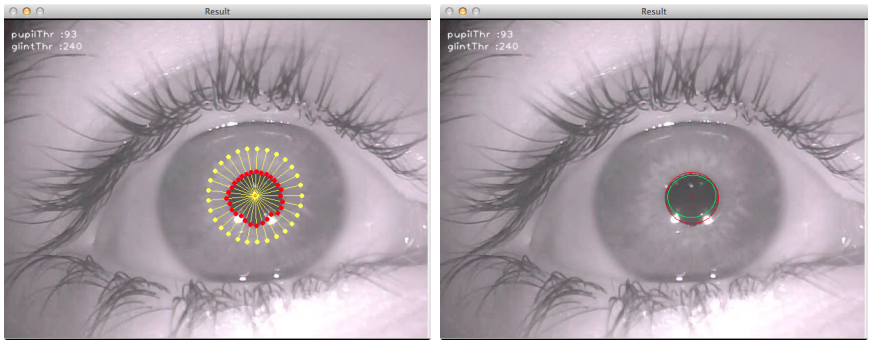


Figure 11: as Figure 10, but pre-processed with a 9x9 Gaussian Blur

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.

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

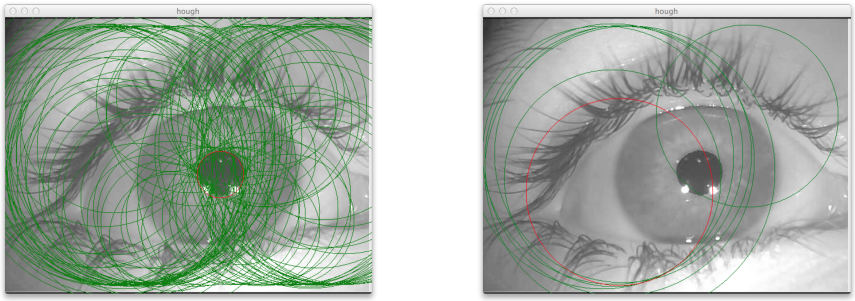
Gauss kernel size The size of the gaussian kernel that is applied to the image before the Hough transformation

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.

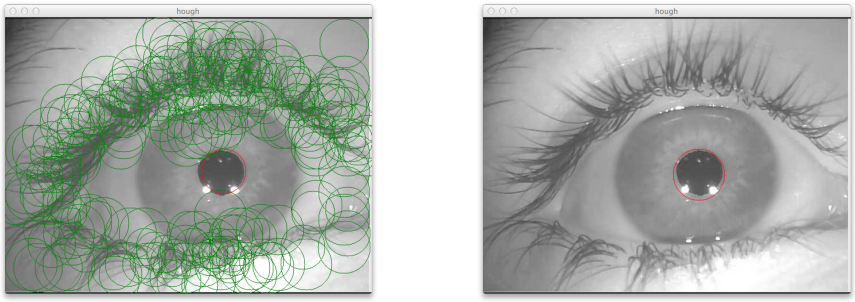
Minimum and Maxiumum Radius The minumum and maximum radius of circles to consider.

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



(a) Size=9, $\sigma = 9$ (b) Size=31, $\sigma = 9$

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 tranforming 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 rougly 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 smoothe away the pupil in some other frame if the had subject moved closer or further away from the camera.

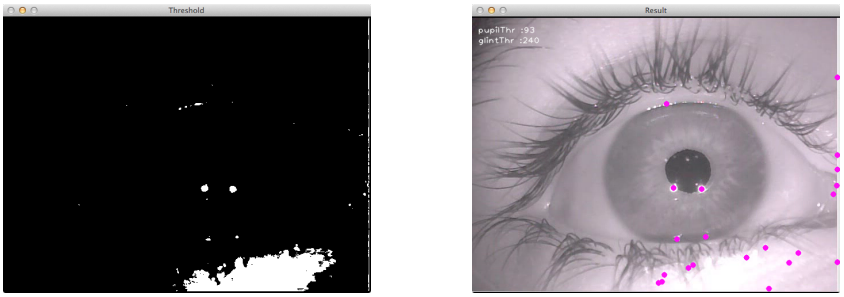
3 Glint Detection

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.

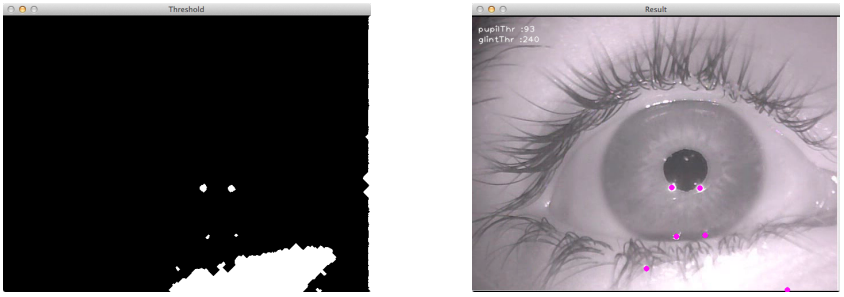
Figure 14 shows the glints in 'eye1.avi'. Because glints are very close to white, a fairly high threshold gives a good result. Furthermore by experimenting we found that, by requiring that the BLOB area lies in the interval $[10 : 150]$, we exclude a great deal of unwanted glint detections, it is although clear that there is still a good amount of false positives.



(a) Binary image (b) Resulting image

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



(a) Binary image (b) Resulting image

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 eclipse representing the pupil. Figure 16

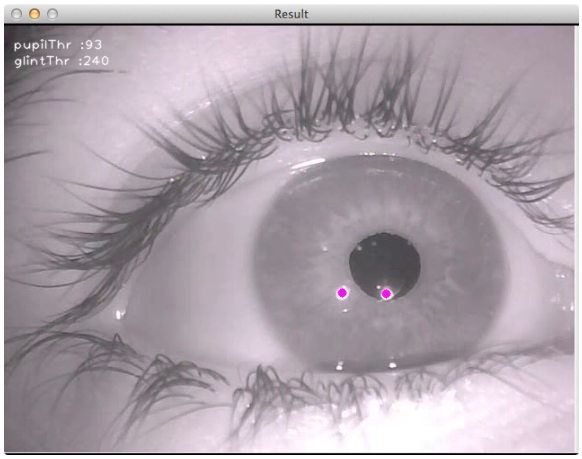
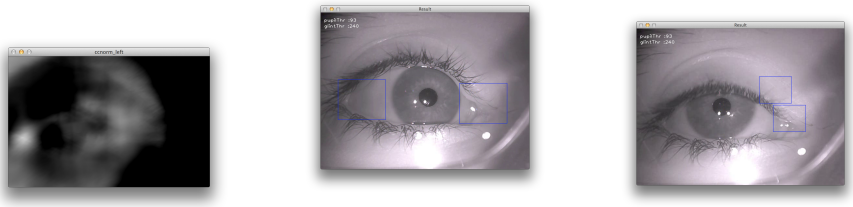


Figure 16: Glint detection with threshold of 240 - using morphology and filter

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.



(a) image

(b) A good template match of both the left and right patch

(c) A bad match of the left template

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.

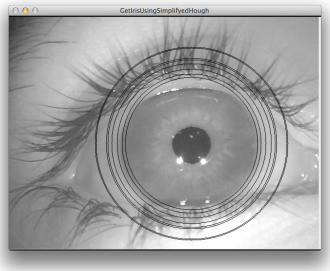
5.2 Hough Transformation with position prior

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 tranformation.

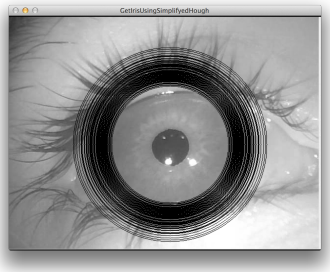
In this way the Hough tranformation will be one dimensional since the only free variable is the radius of the circle.

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

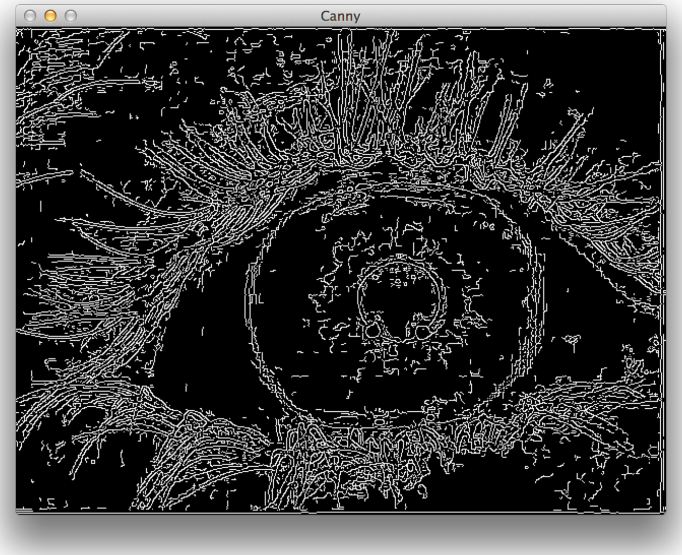


Figure 19: The canny edge image on which the circles in figure (a) are found

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 smoothe 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 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.

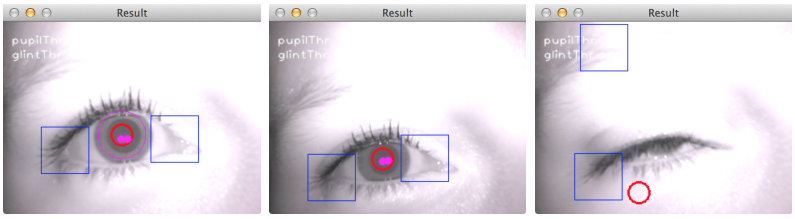


Figure 20: Left: working detection, middle: Iris detection failed, right: Totally failed detection (blink)

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

```
import cv2
import cv
import pylab
import math
from SIGBTools import RegionProps
from SIGBTools import getLineCoordinates
from SIGBTools import ROISelector
from SIGBTools import getImageSequence
from SIGBTools import getCircleSamples
import SIGBTools
import numpy as np
import sys
from scipy.cluster.vq import *
from scipy.misc import *
from matplotlib.pyplot import *


inputFile = "own_Sequences/julie.avi"
outputFile = "eyeTrackerResult.mp4"

#seems to work okay for eye1.avi
default_pupil_threshold = 93

#-------------------------
#          Global variable
#-------------------------
global imgOrig,leftTemplate,rightTemplate,frameNr
imgOrig = [];
#These are used for template matching
leftTemplate = []
rightTemplate = []
frameNr =0;


def GetPupil(gray,thr, min_val, max_val):
  '''Given a gray level image, gray and threshold value return a
     list of pupil locations'''
  #tempResultImg = cv2.cvtColor(gray,cv2.COLOR_GRAY2BGR) #used
    to draw temporary results

  #Threshold image to get a binary image

  cv2.imshow("TempResults", gray)
  val,binI =cv2.threshold(gray, thr, 255, cv2.THRESH_BINARY_INV)
  #print val
  #Morphology (close image to remove small 'holes' inside the
    pupil area)
  st = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
```

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

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

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

75    return pupils

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


85
      st = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
87    #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 = []
      for contour in contours:
99
        #calculate centroid, area and 'extend' (compactness of
        contour)
101     props = prop_calc.CalcContourProperties(contour, ["centroid"
        , "area", "extend"])
        x, y = props["Centroid"]
103     area = props["Area"]
        extend = props["Extend"]
105     print x, y, area, extend

107     #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):
109       glints.append((x,y))

111       #cv2.circle(tempResultImg,(int(x),int(y)), 2, (0,0,255),4)
        #draw a circle
      #cv2.imshow("TempResults",tempResultImg)
113
      return glints
115
  def GetIrisUsingThreshold(gray, thr, min_val, max_val):
117   ''' Given a gray level image, gray and threshold
      value return a list of iris locations'''
119   val,binary_image = cv2.threshold(gray, thr, 255, cv2.
        THRESH_BINARY_INV)
      cv2.imshow("Threshold", binary_image)
121
      contours, hierarchy = cv2.findContours(binary_image, cv2.
        RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
123
      irises = [];
125   prop_calc = RegionProps()
      centroids = []
127   for contour in contours:
        #calculate centroid, area and 'extend' (compactness of
        contour)
129     props = prop_calc.CalcContourProperties(contour, ["centroid"
        , "area", "extend"])
        x, y = props["Centroid"]
131     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):
        points = getCircleSamples(center=circleCenter, radius=radius
      , nPoints=samplePoints)
179     accumulator_line[radius] = 0
        for point in points:
181       try:
            if edgeImage[point[0], point[1]] > 0:
183           accumulator_line[radius] += 1
          except IndexError:
185         continue
    radii = []
187 for radius in accumulator_line:
      value = accumulator_line[radius]
189   if value > thr:
        radii.append(radius)
191     print value
    return radii
193


195
    def GetIrisUsingNormals(gray,pupil,normalLength):
197   ''' Given a gray level image, gray and the length of the
      normals, normalLength
      return a list of iris locations'''
199   # YOUR IMPLEMENTATION HERE !!!!
      pass
201
    def GetIrisUsingSimplifyedHough(gray,pupil):
203   ''' Given a gray level image, gray
      return a list of iris locations using a simplified Hough
        transformation'''
205   if pupil != None:
        edges = cv2.Canny(gray, 30, 20)
207     pupil = pupil
        pupil_x = int(pupil[0])
209     pupil_y = int(pupil[1])
        cv2.imshow("edges", edges)
211     radii = simplifiedHough(edges, pupil, 10, 100, 1, 50)
        irises = []
213     for radius in radii:
          #cv2.circle(gray, (pupil_x, pupil_y), radius, (0, 0, 0),
        1)
215       #cv2.circle(gray, pupil, radius, (127,127,127), 2)
          #cv2.imshow("GetIrisUsingSimplifyedHough", gray)
217       irises.append((pupil_x,pupil_y,radius))
    return irises
219


221
    def plotVectorField(I):
223   g_x = cv2.Sobel(I, cv.CV_64F, 1,0, ksize=3)
      g_y = cv2.Sobel(I, cv.CV_64F, 0,1, ksize=3) # ksize=3 som **
        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:
      x_mesh_dim]
    sample_g_y = g_y[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
      x_mesh_dim]
231
    quiver(sample_g_x, sample_g_y)
233 show()


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

247 # for x in range(X):
    #   for y in range(Y):
249 #     orientation[x][y] = np.arctan2(g_y[x][y], g_x[x][y]) *
      (180 / math.pi)
    #     magnitude[x][y] = math.sqrt(g_y[x][y] ** 2 + g_x[x][y]
      ** 2)
251


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

263   minVal, maxVal, minLoc, maxloc_left_from = cv2.minMaxLoc(
      ccnorm_left)
      minVal, maxVal, minLoc, maxloc_right_from, = cv2.minMaxLoc(
      ccnorm_right)
265
      l_x,l_y = leftTemplate.shape
267   max_loc_left_from_x = maxloc_left_from[0]
      max_loc_left_from_y = maxloc_left_from[1]
269
```

```
        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
        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
        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,
         maxloc_right_to)

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

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

305   newPupil = np.zeros((nPts,1,2)).astype(np.float32)
      t = 0
307   for (x,y,dx,dy) in P:
        #< define normalLength as some maximum distance away from
        initial circle >
309     #< get the endpoints of the normal -> p1,p2>
        point_coords = (int(x),int(y))
311     normal_gradient = dx, dy
        #cv2.circle(img, point_coords, 2, bgr_blue, 2)
313     max_point = findMaxGradientValueOnNormal(gradient_magnitude,
         gradient_orientation, point_coords, center_point_coords,
        normal_gradient)
        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(
      newPupil)>
319   return cv2.fitEllipse(newPupil)

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

351   return filtered_pupils, filtered_glints

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

359 def filterGlintsIris(glints, irises):
    new_glints = []
361   if glints and irises:
      for glint in glints:
```

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


373

375 def update(I):
      '''Calculate the image features and display the result based
        on the slider values'''
377   #global drawImg
      global frameNr,drawImg, gray
379   img = I.copy()
      sliderVals = getSliderVals()
381   gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
      gray = cv2.equalizeHist(gray)
383
      # Do the magic
385   #pupils = GetPupil(gray,sliderVals['pupilThr'], sliderVals['
        minSize'], sliderVals['maxSize'])
      pupils = circularHough(gray)
387   glints = GetGlints(gray,sliderVals['glintThr'])
      pupils, glints = FilterPupilGlint(pupils,glints)
389   #irises = GetIrisUsingThreshold(gray, sliderVals['pupilThr'],
        sliderVals['minSize'], sliderVals['maxSize'])

391   K=10
      d=40
393   #labelIm,centroids = detectPupilKMeans(gray,K=K,distanceWeight
        =d,reSize=(70,70))
      #pupils = get_pupils_from_kmean(labelIm,centroids,gray,
        sliderVals['minSize'],sliderVals['maxSize'])
395
      #magnitude, orientation = getGradientImageInfo(gray)
397   if pupils:
        irises = GetIrisUsingSimplifyedHough(gray, pupils[0])
399
      #plotVectorField(gray)
401   #Do template matching
      global leftTemplate
403   global rightTemplate

405   corners = GetEyeCorners(gray, leftTemplate, rightTemplate)

407   #detectPupilHough(gray, 100)
      #irises = detectIrisHough(gray, 400)
409
      #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
    #Elipse
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:
441     C = int(glint[0]),int(glint[1])
        cv2.circle(img,C, 2,(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])
455   cv2.circle(img, C, radius, (255,0,255), 1)

457 cv2.imshow("Result", img)
```

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

465
  def printUsage():
467   print "Q or ESC: Stop"
    print "SPACE: Pause"
469   print "r: reload video"
    print 'm: Mark region when the video has paused'
471   print 's: toggle video  writing'
    print 'c: close video sequence'
473
  def run(fileName,resultFile='eyeTrackingResults.avi'):
475
    ''' MAIN Method to load the image sequence and handle user
      inputs'''
477   global imgOrig, frameNr,drawImg, leftTemplate, rightTemplate,
      gray
    setupWindowSliders()
479   props = RegionProps();
    cap,imgOrig,sequenceOK = getImageSequence(fileName)
481   videoWriter = 0

483   frameNr =0
    if(sequenceOK):
485     update(imgOrig)
    printUsage()
487   frameNr=0;
    saveFrames = False
489
    while(sequenceOK):
491     sliderVals = getSliderVals();
      frameNr=frameNr+1
493     ch = cv2.waitKey(1)
      #Select regions
495     if(ch==ord('m')):
        if(not sliderVals['Running']):
497         roiSelect=ROISelector(imgOrig)
          pts,regionSelected= roiSelect.SelectArea('Select eye
      corner',(400,200))
499         if(regionSelected):
            if leftTemplate == []:
501           leftTemplate = gray[pts[0][1]:pts[1][1],pts[0][0]:
      pts[1][0]]
            else:
503           rightTemplate = gray[pts[0][1]:pts[1][1],pts[0][0]:
      pts[1][0]]

505     if ch == 27:
```

```
        break
    if (ch==ord('s')):
        if((saveFrames)):
            videoWriter.release()
            saveFrames=False
            print "End recording"
        else:
            imSize = np.shape(imgOrig)
            videoWriter = cv2.VideoWriter(resultFile, cv.CV_FOURCC('D','I','V','3'), 15.0,(imSize[1],imSize[0]),True) #Make a video writer
            saveFrames = True
            print "Recording..."


    if(ch==ord('q')):
        break
    if(ch==32): #Spacebar
        sliderVals = getSliderVals()
        cv2.setTrackbarPos('Stop/Start','Controls',not sliderVals['Running'])
    if(ch==ord('r')):
        frameNr =0
        sequenceOK=False
        cap,imgOrig,sequenceOK = getImageSequence(fileName)
        update(imgOrig)
        sequenceOK=True

    sliderVals=getSliderVals()
    if(sliderVals['Running']):
        sequenceOK, imgOrig = cap.read()
        if(sequenceOK): #if there is an image
            update(imgOrig)
        if(saveFrames):
            videoWriter.write(drawImg)
if(videoWriter!=0):
    videoWriter.release()
        print "Closing videofile..."
#----------------------

def detectPupilKMeans(gray,K=2,distanceWeight=2,reSize=(40,40)):
    ''' Detects the pupil in the image, gray, using k-means
        gray            : grays scale image
        K               : Number of clusters
        distanceWeight  : Defines the weight of the position parameters
        reSize          : the size of the image to do k-means on
    '''
    #Resize for faster performance
    smallI = cv2.resize(gray, reSize)
    smallI = cv2.GaussianBlur(smallI,(3,3),20)
    M,N = smallI.shape
    #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)
593   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;
611 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:
623 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;
639 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):
        #print circles
647     all_circles = circles[0]
        M,N = all_circles.shape
649     k=1
        for c in all_circles:
651       irises.append(c)
          cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
        M),k*128,0))
653       K=k+1
          #Circle with max votes
655     c=all_circles[0,:]
        cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255))
657   cv2.imshow("hough",gColor)
      return irises
659 #----------------------
    #          UI related
661 #----------------------

663 def setText(dst, (x, y), s):
      cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0,
        (0, 0, 0), thickness = 2, lineType=cv2.CV_AA)
665   cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0, (255,
         255, 255), lineType=cv2.CV_AA)

667 vertical_window_size = 523
    horizontal_window_size = 640
669
    def setupWindowSliders():
671   ''' Define windows for displaying the results and create
        trackbars'''
      cv2.namedWindow("Result")
673   cv2.moveWindow("Result", 0, 0)
      cv2.namedWindow('Threshold')
675   cv2.moveWindow("Threshold", 0, vertical_window_size)
      cv2.namedWindow('Controls')
677   cv2.moveWindow("Controls", horizontal_window_size, 0)
      cv2.resizeWindow('Controls', horizontal_window_size, 0)
679   cv2.namedWindow("TempResults")
      cv2.moveWindow("TempResults", horizontal_window_size,
        vertical_window_size)
681   #Threshold value for the pupil intensity
      cv2.createTrackbar('pupilThr','Controls',
        default_pupil_threshold, 255, onSlidersChange)
683   #Threshold value for the glint intensities
      cv2.createTrackbar('glintThr','Controls', 240, 255,
        onSlidersChange)
685   #define the minimum and maximum areas of the pupil
      cv2.createTrackbar('minSize','Controls', 20, 2000,
        onSlidersChange)
687   cv2.createTrackbar('maxSize','Controls', 2000,2000,
        onSlidersChange)
      #Value to indicate whether to run or pause the video
689   cv2.createTrackbar('Stop/Start','Controls', 0,1,
```

```
    onSlidersChange)

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

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

def euclidianDistance(a,b):
  a_x, a_y = a
  b_x, b_y = b
  return math.sqrt((a_x - b_x) ** 2 + (a_y - b_y) **2)

#-------------------------
#          main
#-------------------------
run(inputFile)
```