Eye Tracking
SIGB Spring 2014

Marcus Gregersen
mabg@itu.dk

Martin Faartoft
mlfa@itu.dk

Mads Westi
mwek@itu.dk

March 26th 2014
IT University of Copenhagen

1 Introduction

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

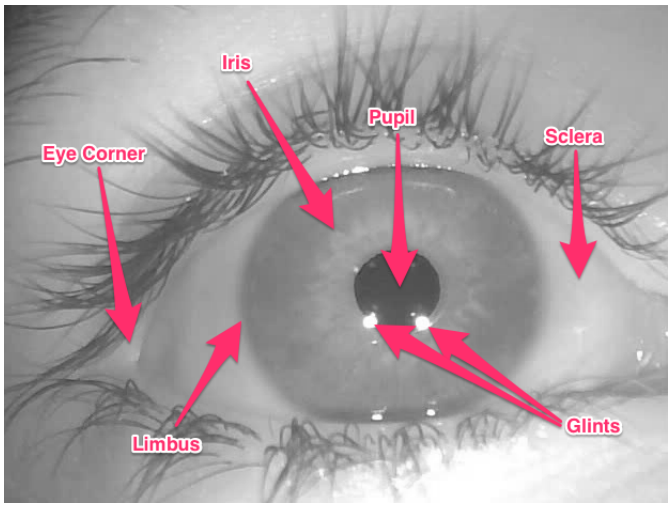


Figure 1: Names and positions of major eye features

2 Pupil Detection

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

2.1 Thresholding

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

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

The next step, is to do connected component analysis, and fit an ellipsis through the blobs. As seen in Figure 3, this succesfully 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.

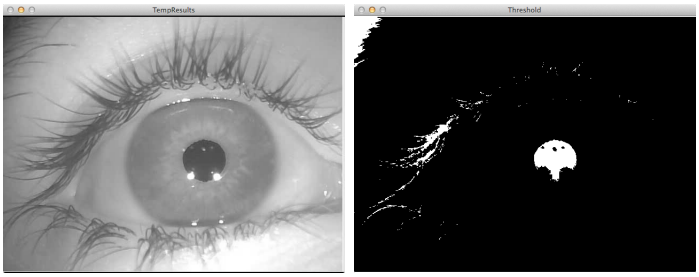


Figure 2: Thresholding eye1.avi

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.

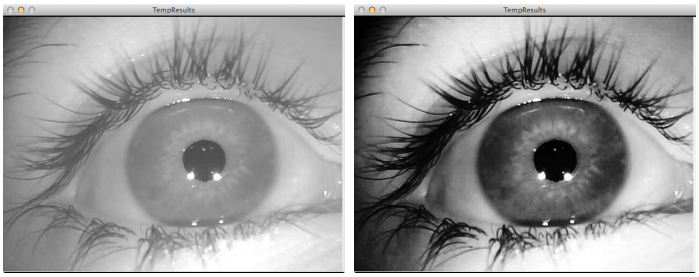


Figure 4: Eye1 before and after Histogram Equalization

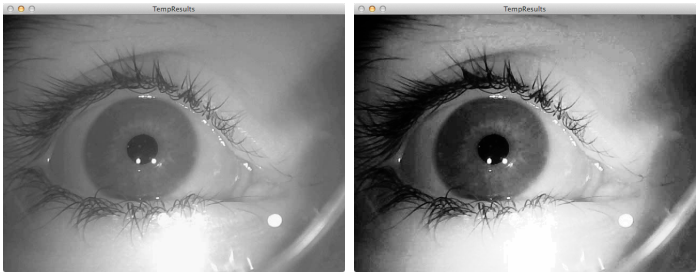


Figure 5: Eye3 before and after Histogram Equalization

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.

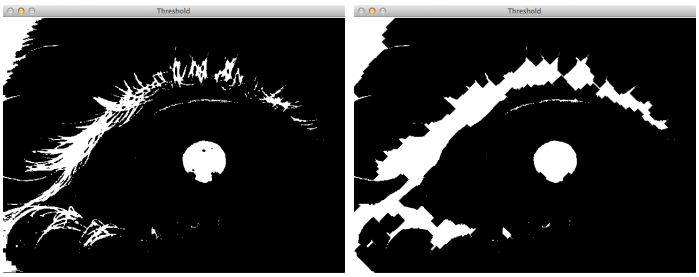


Figure 6: Eye1 before and after Closing (5 iterations, 5x5 CROSS structuring element)

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

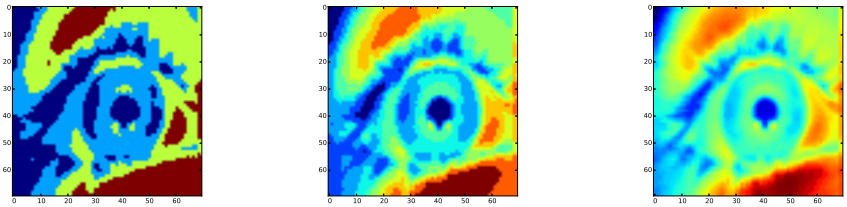
2. One or more pupils were found within the distance, return those

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.



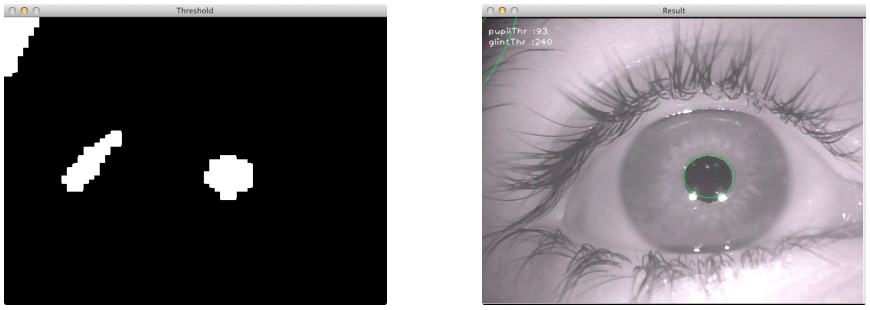
(a) $K = 4$ (b) $K = 10$ (c) $K = 80$

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 unfortunate because, if the pupil could be found as a single cluster, the amount of evaluation on the BLOB could be reduced and give better scalability on the position of the eye relative to the camera.

2.3 Pupil Detection using Gradient Magnitude

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

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

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

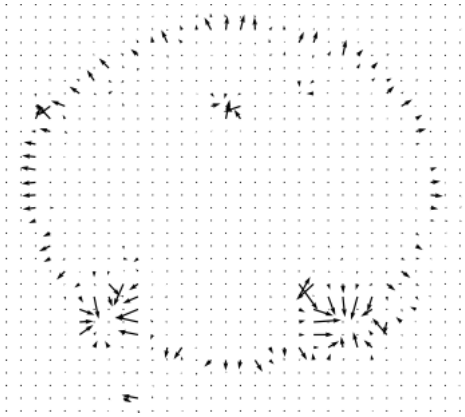


Figure 9: Quiver plot of Eye1 gradients (zoomed on pupil area)

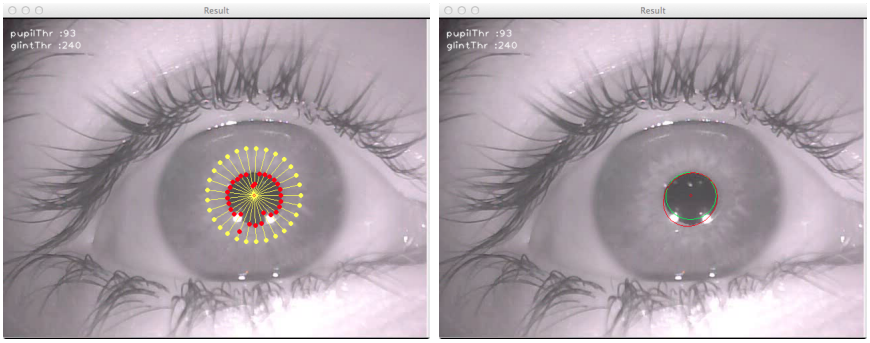


Figure 10: Left: Eye1 showing line-segments for gradient magnitude maximization (yellow) and maximum gradient values along the lines (red). Right: Eye1 showing pupil approximation (green), and new pupil detection (red)

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

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

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

2.4 Pupil Detection by circular Hough transformation

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

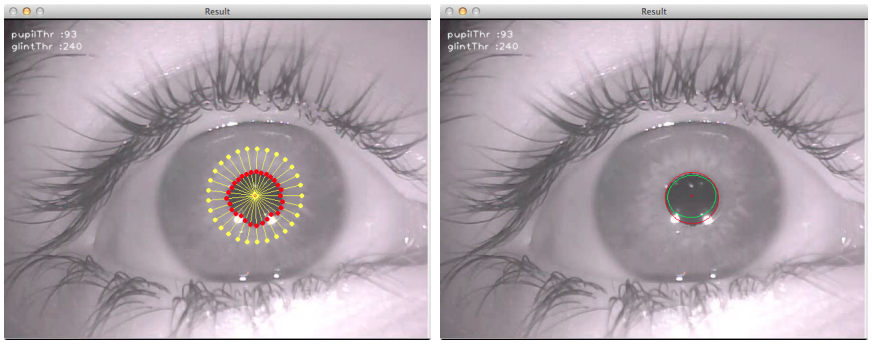


Figure 11: as Figure 10, but pre-processed with a 9x9 Gaussian Blur

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

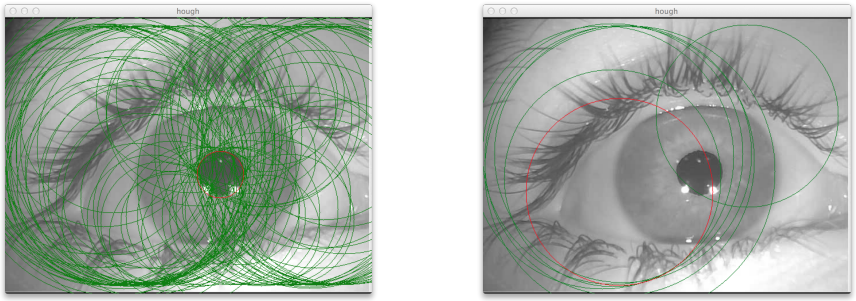
Gauss kernel size The size of the gaussian kernel that is applied to the image before the Hough transformation

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

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

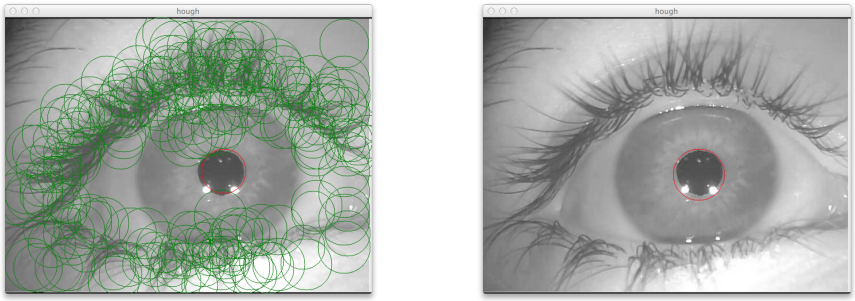
Minimum and Maxiumum Radius The minumum and maximum radius of circles to consider.

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 smooth away the pupil in some other frame if the had subject moved closer or futher away from the camera.

3 Glint Detection

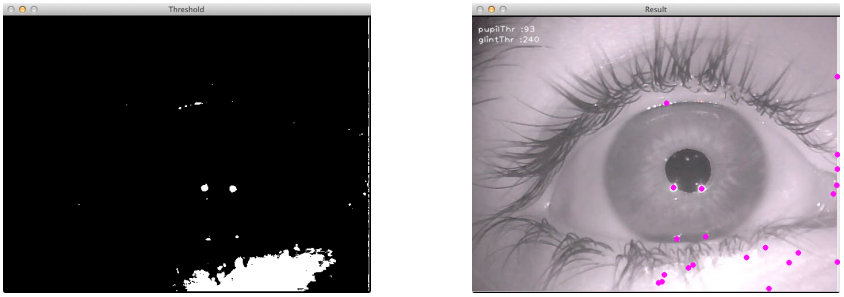
In this section, we will investigate and discuss glint detection.

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-

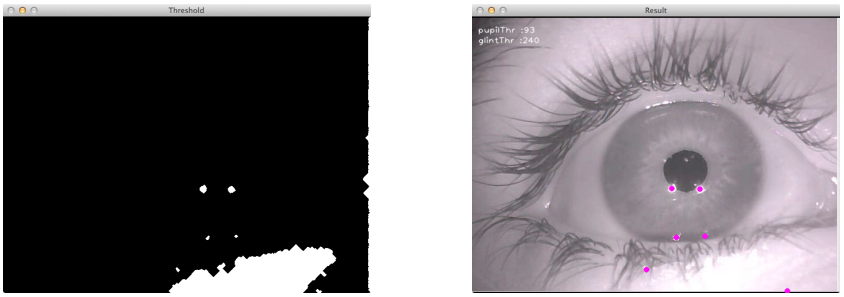
menting we found that, by requiring that the BLOB area lies in the interval [10 : 150], we exclude a great deal of unwanted glint detections, it is although clear that there is still a good amount of false positives.



(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

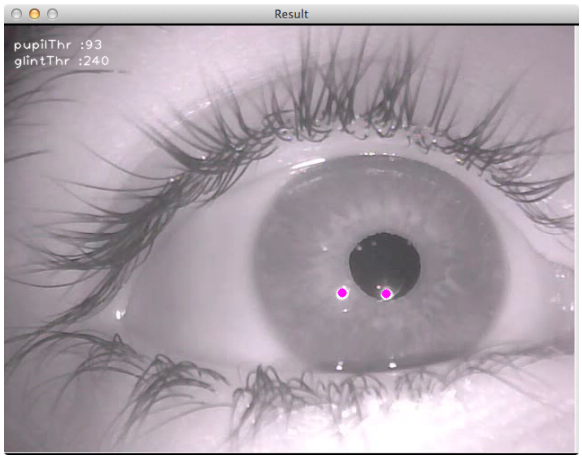


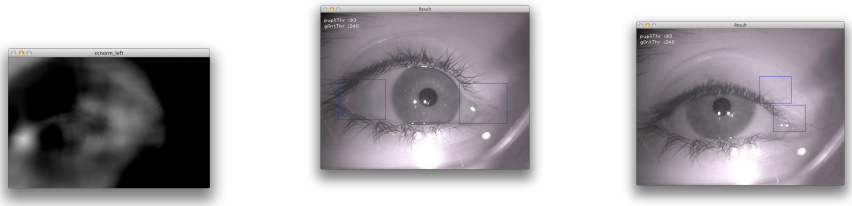
Figure 16: Glint detection with threshold of 240 - using morphology and filter

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.

More intro text



(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.

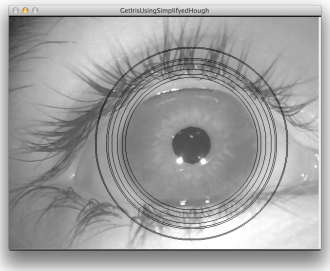
5.2 Hough Transformation with position prior

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.

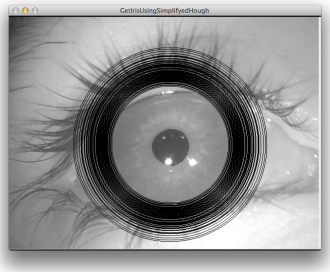
In this way the Hough tranformation will be one dimensional since the only free variable is the radius of the circle.

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

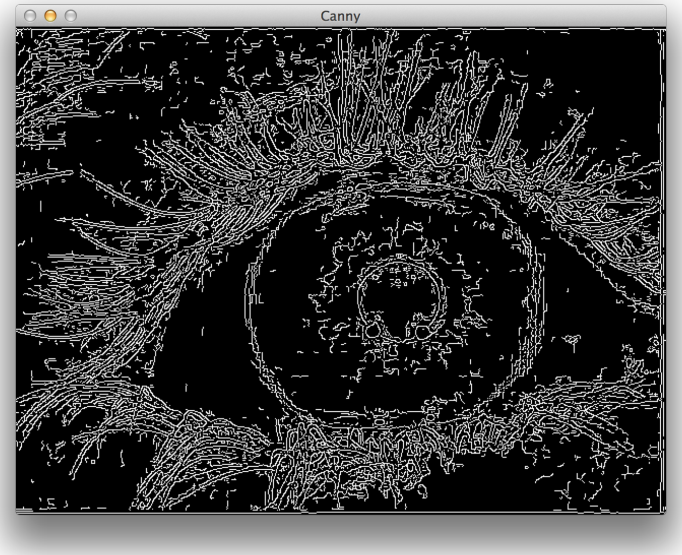


Figure 19: The canny edge image on which the circles in figure (a) are found

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 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

References

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 = "Sequences/eye1.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
  #gray = cv2.equalizeHist(gray)
  cv2.imshow("TempResults", gray)
  val,binI =cv2.threshold(gray, thr, 255, cv2.THRESH_BINARY_INV)
  #print val
```

```
45    #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 = 10
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)

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

glints = [];
prop_calc = RegionProps()
centroids = []
for contour in contours:
    #calculate centroid, area and 'extend' (compactness of
    contour)
    props = prop_calc.CalcContourProperties(contour, ["centroid"
    , "area", "extend"])
    x, y = props["Centroid"]
    area = props["Area"]
    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
    if area > min_area and area < max_area: #and extend > 0.4
    and extend < 1.0):
        glints.append((x,y))
        #print x, y, area, extend
        #cv2.circle(tempResultImg,(int(x),int(y)), 2, (0,0,255),4)
    #draw a circle
#cv2.imshow("TempResults",tempResultImg)

return glints

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

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

    irises = [];
    prop_calc = RegionProps()
    centroids = []
    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"]
        extend = props["Extend"]
131     #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:
133       irisEllipse = cv2.fitEllipse(contour)
          # center, radii, angle = pupilEllipse
135       # max_radius = max(radii)
          # c_x = int(center[0])
137       # c_y = int(center[1])
          # cv2.circle(tempResultImg,(c_x,c_y), int(max_radius),
        (0,0,255),4) #draw a circle
139       #cv2.ellipse(tempResultImg, pupilEllipse,(0,255,0),1)
          irises.append(irisEllipse)
141   return irises

143 def circularHough(gray):
    ''' Performs a circular hough transform of the image, gray and
      shows the  detected circles
145   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
147   blur = cv2.GaussianBlur(gray, (31,31), 11)

149   dp = 6; minDist = 30
    highThr = 20 #High threshold for canny
151   accThr = 850; #accumulator threshold for the circle centers at
        the detection stage. The smaller it is, the more false
      circles may be detected
    maxRadius = 50;
153   minRadius = 155;
    circles = cv2.HoughCircles(blur,cv2.cv.CV_HOUGH_GRADIENT, dp,
      minDist, None, highThr,accThr,maxRadius, minRadius)
155
    #Make a color image from gray for display purposes
157   gColor = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
    if (circles !=None):
159   #print circles
      all_circles = circles[0]
161   M,N = all_circles.shape
      k=1
163   for c in all_circles:
        cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (int(k*255/
      M),k*128,0))
165     K=k+1
      c=all_circles[0,:]
167   cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255),5)
      cv2.imshow("hough",gColor)
169
  def simplifiedHough(edgeImage,circleCenter,minR,maxR,N,thr):
171   samplePoints  = 100
    accumulator_line = {}
```

```
173
    for radius in range(minR, maxR, N):
175
      points = getCircleSamples(center=circleCenter, radius=radius
      , nPoints=samplePoints)
177     accumulator_line[radius] = 0
      for point in points:
179       try:
          if edgeImage[point[0], point[1]] > 0:
181         accumulator_line[radius] += 1
        except IndexError:
183       continue
    radii = []
185 for radius in accumulator_line:
      value = accumulator_line[radius]
187   if value > thr:
        radii.append(radius)
189     print value
    return radii
191


193
  def GetIrisUsingNormals(gray,pupil,normalLength):
195   ''' Given a gray level image, gray and the length of the
      normals, normalLength
      return a list of iris locations'''
197   # YOUR IMPLEMENTATION HERE !!!!
    pass
199
  def GetIrisUsingSimplifyedHough(gray,pupil):
201   ''' Given a gray level image, gray
    return a list of iris locations using a simplified Hough
      transformation'''
203   if pupil:
      edges = cv2.Canny(gray, 30, 20)
205   pupil = pupil[0]
      pupil_x = int(pupil[0])
207   pupil_y = int(pupil[1])
      #cv2.imshow("edges", edges)
209   radii = simplifiedHough(edges, pupil, 120, 200, 1, 15)
      for radius in radii:
211     cv2.circle(gray, (pupil_x, pupil_y), radius, (0, 0, 0), 1)
        #cv2.circle(gray, pupil, radius, (127,127,127), 2)
213     cv2.imshow("GetIrisUsingSimplifyedHough", gray)


215


217 def plotVectorField(I):
    g_x = cv2.Sobel(I, cv.CV_64F, 1,0, ksize=3)
219 g_y = cv2.Sobel(I, cv.CV_64F, 0,1, ksize=3) # ksize=3 som **
      kwargs

221 x_orig_dim, y_orig_dim = I.shape
    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:
      x_mesh_dim]
225 sample_g_y = g_y[0:x_orig_dim:x_mesh_dim,0:y_orig_dim:
      x_mesh_dim]

227 quiver(sample_g_x, sample_g_y)
    show()
229

231 def getGradientImageInfo(I):
    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

235 X,Y = I.shape
    orientation = np.zeros(I.shape)
237 magnitude = np.zeros(I.shape)
    sq_g_x = cv2.pow(g_x, 2)
239 sq_g_y = cv2.pow(g_y, 2)
    fast_magnitude = cv2.pow(sq_g_x + sq_g_y, .5)
241
    # for x in range(X):
243 #   for y in range(Y):
    #     orientation[x][y] = np.arctan2(g_y[x][y], g_x[x][y]) *
      (180 / math.pi)
245 #     magnitude[x][y] = math.sqrt(g_y[x][y] ** 2 + g_x[x][y]
      ** 2)


247
    #print fast_magnitude[0]
249 #print magnitude[0]

251 return fast_magnitude,orientation

253 def GetEyeCorners(orig_img, leftTemplate, rightTemplate,
      pupilPosition=None):
    if leftTemplate != [] and rightTemplate != []:
255   ccnorm_left = cv2.matchTemplate(orig_img, leftTemplate, cv2.
      TM_CCOEFF_NORMED)
      ccnorm_right = cv2.matchTemplate(orig_img, rightTemplate,
      cv2.TM_CCOEFF_NORMED)
257
      minVal, maxVal, minLoc, maxloc_left_from = cv2.minMaxLoc(
      ccnorm_left)
259   minVal, maxVal, minLoc, maxloc_right_from, = cv2.minMaxLoc(
      ccnorm_right)

261   l_x,l_y = leftTemplate.shape
      max_loc_left_from_x = maxloc_left_from[0]
263   max_loc_left_from_y = maxloc_left_from[1]

265   max_loc_left_to_x = max_loc_left_from_x + l_x
      max_loc_left_to_y = max_loc_left_from_y + l_y
267
```

```
        maxloc_left_to = (max_loc_left_to_x, max_loc_left_to_y)
269
        r_x,r_y = leftTemplate.shape
271     max_loc_right_from_x = maxloc_right_from[0]
        max_loc_right_from_y = maxloc_right_from[1]
273
        max_loc_right_to_x = max_loc_right_from_x + r_x
275     max_loc_right_to_y = max_loc_right_from_y + r_y
        maxloc_right_to = (max_loc_right_to_x, max_loc_right_to_y)
277
        return (maxloc_left_from, maxloc_left_to, maxloc_right_from,
         maxloc_right_to)
279
bgr_yellow = 0,255,255
281 bgr_blue = 255, 0, 0
bgr_red = 0, 0, 255
283 def circleTest(img, center_point):
    nPts = 20
285 circleRadius = 100
    P = getCircleSamples(center=center_point, radius=circleRadius,
       nPoints=nPts)
287 for (x,y,dx,dy) in P:
      point_coords = (int(x),int(y))
289   cv2.circle(img, point_coords, 2, bgr_yellow, 2)
      cv2.line(img, point_coords, center_point, bgr_yellow)
291
def findEllipseContour(img, gradient_magnitude,
    gradient_orientation, estimatedCenter, estimatedRadius, nPts
    =30):
293 center_point_coords = (int(estimatedCenter[0]), int(
    estimatedCenter[1]))
    P = getCircleSamples(center = estimatedCenter, radius =
    estimatedRadius, nPoints=nPts)
295 for (x,y,dx,dy) in P:
      point_coords = (int(x),int(y))
297   cv2.circle(img, point_coords, 2, bgr_yellow, 2)
      cv2.line(img, point_coords, center_point_coords, bgr_yellow)
299
    newPupil = np.zeros((nPts,1,2)).astype(np.float32)
301 t = 0
    for (x,y,dx,dy) in P:
303   #< define normalLength as some maximum distance away from
      initial circle >
      #< get the endpoints of the normal -> p1,p2>
305   point_coords = (int(x),int(y))
      normal_gradient = dx, dy
307   #cv2.circle(img, point_coords, 2, bgr_blue, 2)
      max_point = findMaxGradientValueOnNormal(gradient_magnitude,
       gradient_orientation, point_coords, center_point_coords,
      normal_gradient)
309   cv2.circle(img, tuple(max_point), 2, bgr_red, 2) #locate the
       max points
      #< store maxPoint in newPupil>
311   newPupil[t] = max_point
```

```
      t += 1
313   #<fitPoints to model using least squares- cv2.fitellipse(
        newPupil)>
      return cv2.fitEllipse(newPupil)
315
    def findMaxGradientValueOnNormal(gradient_magnitude,
        gradient_orientation, p1, p2, normal_orientation):
317     #Get integer coordinates on the straight line between p1 and
         p2
      pts = SIGBTools.getLineCoordinates(p1, p2)
319   values = gradient_magnitude[pts[:,1],pts[:,0]]
      #orientations = gradient_orientation[pts[:,1],pts[:,0]]
321   #normal_angle = np.arctan2(normal_orientation[1],
        normal_orientation[0]) * (180 / math.pi)

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

337 def FilterPupilGlint(pupils,glints):
      ''' Given a list of pupil candidates and glint candidates
        returns a list of pupil and glints'''
339   filtered_glints = []
      filtered_pupils = pupils
341   for glint in glints:
        for pupil in pupils:
343       if (is_glint_close_to_pupil(glint, pupil)):
            filtered_glints.append(glint)
345
      return filtered_pupils, filtered_glints
347
    def is_glint_close_to_pupil(glint, pupil):
349   center, radii, angle = pupil
      max_radius = max(radii)
351   distance = euclidianDistance(center, glint)
      return (distance < max_radius)
353
    def filterGlintsIris(glints, irises):
355   new_glints = []
      if glints and irises:
357     for glint in glints:
          for iris in irises:
359         iris_x, irix_y, iris_radius = iris
            print glint
```

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


369
   def update(I):
371   '''Calculate the image features and display the result based
        on the slider values'''
      #global drawImg
373   global frameNr,drawImg, gray
      img = I.copy()
375   sliderVals = getSliderVals()
      gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
377   #gray = cv2.GaussianBlur(gray, (9,9),9)

379   # Do the magic
      pupils = GetPupil(gray,sliderVals['pupilThr'], sliderVals['
        minSize'], sliderVals['maxSize'])
381   glints = GetGlints(gray,sliderVals['glintThr'])
      #pupils, glints = FilterPupilGlint(pupils,glints)
383   #irises = GetIrisUsingThreshold(gray, sliderVals['pupilThr'],
        sliderVals['minSize'], sliderVals['maxSize'])

385   K=10
      d=40
387   #labelIm,centroids = detectPupilKMeans(gray,K=K,distanceWeight
        =d,reSize=(70,70))
      #pupils = get_pupils_from_kmean(labelIm,centroids,gray,
        sliderVals['minSize'],sliderVals['maxSize'])
389
      magnitude, orientation = getGradientImageInfo(gray)
391   if pupils:
        GetIrisUsingSimplifyedHough(gray, pupils[0])
393
      #plotVectorField(gray)
395   #Do template matching
      global leftTemplate
397   global rightTemplate

399   #corners = GetEyeCorners(gray, leftTemplate, rightTemplate)

401   #detectPupilHough(gray, 100)
      #irises = detectIrisHough(gray, 400)
403
      #glints = filterGlintsIris(glints,irises)
405
      #Display results
407   global frameNr,drawImg
      x,y = 10,10
409   #setText(img,(x,y),"Frame:%d" %frameNr)
```

```
    sliderVals = getSliderVals()
411
    # for non-windows machines we print the values of the
      threshold in the original image
413 if sys.platform != 'win32':
      step=18
415   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)
417 cv2.imshow('Result',img)

419 #Uncomment these lines as your methods start to work to
      display the result in the
    #original image
421
    for pupil in pupils:
423   #cv2.ellipse(img,pupil,(0,255,0),1)
      C = int(pupil[0][0]),int(pupil[0][1])
425
      contour = findEllipseContour(img, magnitude, orientation, C,
       70)
427   cv2.ellipse(img, contour, bgr_red, 1)
      cv2.circle(img,C, 2, (0,0,255),1)
429   #circleTest(img, C)
    for glint in glints:
431     C = int(glint[0]),int(glint[1])
        #cv2.circle(img,C, 2,(255,0,255),5)
433


435
    # if corners:
437 #   left_from, left_to, right_from, right_to = corners
    #   cv2.rectangle(img, left_from , left_to, 255)
439 #   cv2.rectangle(img, right_from , right_to, 255)

441 # for iris in irises:
    #   cv2.ellipse(img,iris,(0,255,0),1)
443 #   C = int(iris[0][0]),int(iris[0][1])
    #   cv2.circle(img,C, 2, (0,0,255),4)
445
    cv2.imshow("Result", img)
447
      #For Iris detection - Week 2
449   #circularHough(gray)

451 #copy the image so that the result image (img) can be saved in
      the movie
    drawImg = img.copy()
453

455 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]]
491         else:
              rightTemplate = gray[pts[0][1]:pts[1][1],pts[0][0]:
    pts[1][0]]
493
      if ch == 27:
495     break
      if (ch==ord('s')):
497     if((saveFrames)):
          videoWriter.release()
499       saveFrames=False
          print "End recording"
501     else:
          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')):
            break
511       if(ch==32): #Spacebar
            sliderVals = getSliderVals()
513         cv2.setTrackbarPos('Stop/Start','Controls',not sliderVals[
      'Running'])
          if(ch==ord('r')):
515         frameNr =0
            sequenceOK=False
517         cap,imgOrig,sequenceOK = getImageSequence(fileName)
            update(imgOrig)
519         sequenceOK=True

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

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

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

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

    pupils = [];
    prop_calc = RegionProps()
    for contour in contours:
      #calculate centroid, area and 'extend' (compactness of
      contour)
       props = prop_calc.CalcContourProperties(contour, ["centroid"
       , "area", "extend"])
       x, y = props["Centroid"]
       area = props["Area"]
       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
       if (area > min_val and area < max_val and extend > 0.4 and
       extend < 1.0):
          pupilEllipse = cv2.fitEllipse(contour)
          pupils.append(pupilEllipse)
    return pupils

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

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

```
643         #Circle with max votes
            c=all_circles[0,:]
645         cv2.circle(gColor, (int(c[0]),int(c[1])),c[2], (0,0,255))
        cv2.imshow("hough",gColor)
647     return irises
    #----------------------------
649 #           UI related
    #----------------------------
651
    def setText(dst, (x, y), s):
653     cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0,
          (0, 0, 0), thickness = 2, lineType=cv2.CV_AA)
        cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0, (255,
           255, 255), lineType=cv2.CV_AA)
655
    vertical_window_size = 523
657 horizontal_window_size = 640

659 def setupWindowSliders():
        ''' Define windows for displaying the results and create
          trackbars'''
661     cv2.namedWindow("Result")
        cv2.moveWindow("Result", 0, 0)
663     cv2.namedWindow('Threshold')
        cv2.moveWindow("Threshold", 0, vertical_window_size)
665     cv2.namedWindow('Controls')
        cv2.moveWindow("Controls", horizontal_window_size, 0)
667     cv2.resizeWindow('Controls', horizontal_window_size, 0)
        cv2.namedWindow("TempResults")
669     cv2.moveWindow("TempResults", horizontal_window_size,
          vertical_window_size)
        #Threshold value for the pupil intensity
671     cv2.createTrackbar('pupilThr','Controls',
          default_pupil_threshold, 255, onSlidersChange)
        #Threshold value for the glint intensities
673     cv2.createTrackbar('glintThr','Controls', 240, 255,
          onSlidersChange)
        #define the minimum and maximum areas of the pupil
675     cv2.createTrackbar('minSize','Controls', 20, 2000,
          onSlidersChange)
        cv2.createTrackbar('maxSize','Controls', 2000,2000,
          onSlidersChange)
677     #Value to indicate whether to run or pause the video
        cv2.createTrackbar('Stop/Start','Controls', 0,1,
          onSlidersChange)
679
    def getSliderVals():
681     '''Extract the values of the sliders and return these in a
          dictionary'''
        sliderVals={}
683     sliderVals['pupilThr'] = cv2.getTrackbarPos('pupilThr', '
          Controls')
        sliderVals['glintThr'] = cv2.getTrackbarPos('glintThr', '
          Controls')
```

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

703 #--------------------------
  #           main
705 #--------------------------
  run(inputFile)
```