

# Iris Detection Authenticator

Nathan D. Tang and Bryan K. Chau

November 2022

## **Abstract**

The development of iris biometric identification recognition is presented. Iris recognition differs from other methods because data acquisition is non-physical and is more accessible. It has been proven that the iris does not change as an individual ages and is well protected from external damages due to the eyelid and cornea, acting as a shield to the iris. In addition, the iris is almost impossible to forge due to its complex patterns and the current limitations in technology. Using Canny Edge Detection, Hough Transform, rubber-sheet normalization, Histogram of Gradient feature extraction, and the MultiMedia University iris database as our subjects, we design a more reliable iris recognition software.

# Contents

<b>1</b>	<b>Introduction and Background</b>	<b>3</b>
<b>2</b>	<b>Related Work</b>	<b>3</b>
<b>3</b>	<b>Scope of Work</b>	<b>3</b>
<b>4</b>	<b>Design and Implementation</b>	<b>3</b>
4.1	Overview . . . . .	3
4.2	Edge Map . . . . .	4
4.3	Pupil and Iris Detection . . . . .	6
4.4	Iris Extraction . . . . .	6
4.5	Image Enhancement: Removal of Extreme Pixels and Increasing Contrast . . . . .	8
4.6	Iris Normalization . . . . .	9
4.7	Further Image Enhancement: Further Removal of Noisy Pixels . . . . .	9
4.8	Feature Extraction . . . . .	11
4.9	Feature Matching . . . . .	12
<b>5</b>	<b>Testing</b>	<b>13</b>
<b>6</b>	<b>Analysis and Evaluation of Results</b>	<b>15</b>
<b>7</b>	<b>Conclusion</b>	<b>16</b>

# 1 Introduction and Background

Biometric authentication is an ever-advancing technology that has seen great improvements in recent years. A popular form of this is iris recognition which has several implementations in various technological sectors around the globe. The iris is a fascinating subject of biometric security. Because each person has a unique iris pattern, it is widely used for secure authentication over other biometrics such as facial recognition and fingerprint sensing. By the end of this project, our team will design a more reliable biometric identification software in the form of iris recognition, utilizing advanced computer vision techniques.

In the images we use, we have to assume that the iris is in full view with little to zero occlusion from eyelashes and eyelids. We also assume minimal glare on the irises that create bright spots and hide critical information. The image of the iris also has to be taken straight on, meaning the iris has to be circular, because we are using Circular Hough Transform to detect the iris.

For this project, we are using a set of captured iris images. The data set is from the MultiMedia University (MMU) Iris database for Biometric Attendance system. It is a public database consisting of consistent, closeup eye images intended for machine learning training model use. The database consists of 440 total eye images of 44 individuals with 10 eye images each (5 left, 5 right). Each bitmap image is grayscale and of size 320x240 [1].

## 2 Related Work

There are numerous articles and research on iris recognition techniques and algorithms. Some researchers proposed different theories for approaching the iris recognition problem. Daugman, for example, proposed the use of an integro-differential operator for localizing iris regions [2]. Others have proposed the use of Canny Edge Detection with a Circular Hough Transform for removing eyelash and eyelid noise and obtaining the iris regions. Another technique developed for filtering unwanted noise is called an adaptive fuzzy switching noise reduction (AFSNR) filter [3]. Our work is related to these and many other projects using various algorithms for iris recognition. Jiménez L. *et al.* [4] propose the use of Hough Transform to find a circumference in the eye image to describe the iris and pupil contours, and Chawla *et al.* [2] introduce Daugman's rubber sheet model for normalization. We also look at a literature review of the most prominent algorithms in each stage of iris recognition. In their review, Ng *et al.* [5] describe various methods for image preprocessing, feature extraction, and template matching. Of these methods, some notable ones are integro-differential operator, Hough transform, homogeneous rubber sheet model, Hamming distance, and weighted Euclidean distance. In another work, separate from iris detection, Sultana *et al.* [6] use HOG feature descriptors for object detection. Many of the techniques and algorithms described above are adopted within this work on iris recognition.

## 3 Scope of Work

In this paper, we develop an algorithm for iris detection and authentication. We use the aforementioned data set as the basis for training and testing our algorithm. The development of this algorithm is separated into multiple sections. The first section is image preprocessing. This involves identifying and separating the iris from the eye images using Canny Edge Detection and Hough Transform. Then, we normalize the image. To do this, we apply the MATLAB function, `ImToPolar`, to convert from Cartesian coordinates to polar coordinates. Once the image is normalized, we extract features using HOG. Finally, we use Euclidean distance and SVM for feature matching, and we test our database of eye images. At the end of this paper, we analyze the results of our findings, discuss their implications, and consider future versions or adaptations.

## 4 Design and Implementation

### 4.1 Overview

We first create an 'eyes' struct to hold the original grayscale eye images ('orig'), edge maps ('edge'), extracted pupil binary images ('pupil'), and iris extractions ('iris') that will be developed in the following sections.

```
1 % struct to hold grayscale eye images, edge maps, pupil extraction, and
2 % iris extraction
3 eyes = struct();
```

The first step in our approach was to develop an edge map using the Canny Edge Detector on the grayscale eye image. Then, we used a pixel intensity threshold to extract a binary map that allowed us to use the Circular Hough Transform to find the edge of the pupil in the image. After finding the pupil center and radius, we search radially outward for the iris edge and then extract only the iris portion from the rest of the image. The next step after extraction is enhancing the image. Here we remove unwanted noise, eyelids, eyelashes, and as much glare from the image as possible by thresholding it. Once we have preprocessed the image of the iris, we then normalize it using the Rubber Sheet Normalization algorithm, remapping the image to polar coordinates. Finally, after normalizing the image, we extract features from the normalized images using different feature extraction methods such as HOG and a 5x5 average filter to get feature descriptors. We then use these feature descriptors in conjunction with different feature-matching techniques such as Euclidean distance and SVM. This will ultimately decide whether the iris image is authenticated or deemed an imposter. These steps can be summarized in Figure 1. We will discuss each step in detail in the following sections.

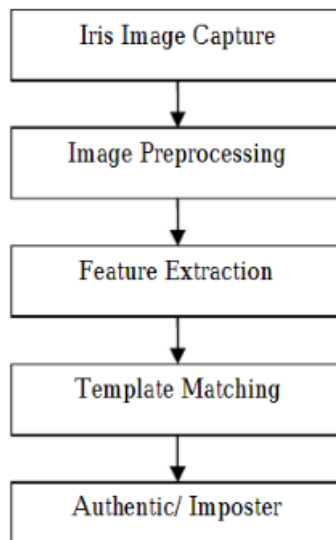


Figure 1: General overview of the Iris Detection Authenticator steps to preprocess, extract, and match iris’ to authenticated users. We will use Canny Edge Detection, Circular Hough Transform, and rubber sheet normalization for preprocessing. Then 5x5 average filter and HOG for feature extraction. Finally, we will use Euclidean distance and SVM to template match/feature match the iris images [7].

## 4.2 Edge Map

Using the Canny Edge Detector, we produced the edge maps that will later be used to detect the iris edge.

Canny Edge Detector has three parameters: 1) High threshold: captures strong edges but misses weaker edges 2) Low threshold: captures weak edges but has false edges/noise 3) Sigma ( $\sigma$ ): corresponds to the Gaussian filter mask size, or how much smoothing the image undergoes

We chose the Canny Edge Detector because it features hysteresis thresholding where only the low threshold edges connected to the high threshold edges are kept (good continuation between adjacent pixels) and also produces single-pixel-wide edge maps. Alternatively, simpler gradient-based or Laplacian-based edge detectors are very susceptible to noise and produce multiple-pixel-wide edge maps, which may interfere with our iris edge detection. Figures 2 and 3 show our trials for optimizing the Canny Edge Detector.

```

1 %Edge Detection: Canny Edge Detector
2 eyes(img).edge = edge(eyes(img).orig, 'canny', 0.15, 3);
  
```

The Canny Edge Detector function in MATLAB defaults so that if only one threshold is specified, that threshold is the high threshold value. The low threshold value is automatically assigned to be 40% of the high threshold.

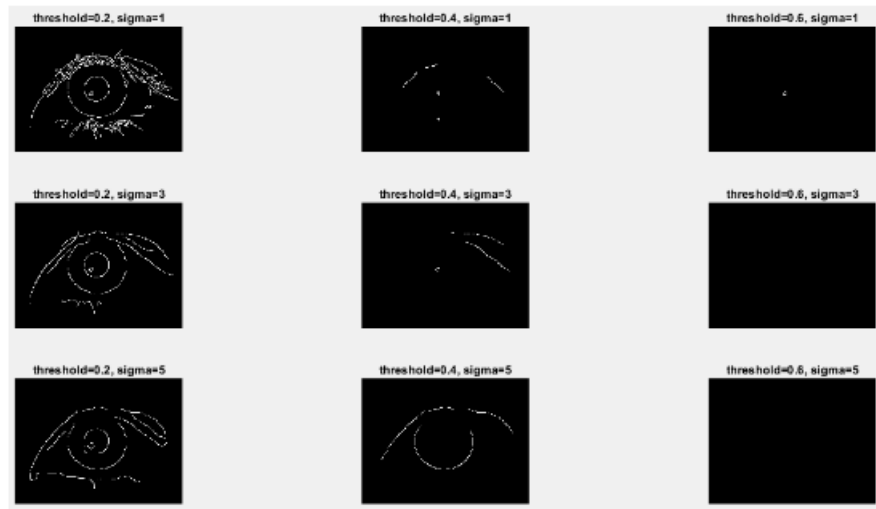


Figure 2: First trial of various thresholds and sigma values for Canny Edge Detector. From the results, it is clear that the threshold should remain under 0.4, with threshold=0.2 giving the best results. Within the threshold=0.2 column,  $\sigma=1$  includes too much detail from the eyelashes, while  $\sigma=3$  and 5 have very good results in terms of keeping the iris edges intact while filtering out the eyelashes. Because  $\sigma=3$  includes a more complete iris shape, we will assume threshold=0.2 and  $\sigma=3$  gives the best results for the Canny Edge detection step for our given images.

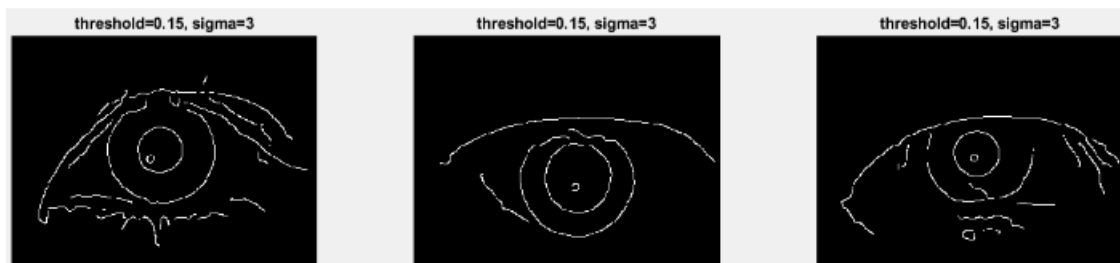


Figure 3: Second trial for optimizing Canny Edge Detector. The previously found threshold=0.2 was found to be too strong for some of the test images, so it was lowered to 0.15, with the sigma value staying the same. The threshold=0.15 and  $\sigma=3$  parameters were tested on all twelve sample images in our data set, with the first three results shown in the figure.

### 4.3 Pupil and Iris Detection

Next, in Figure 4 we used the original grayscale images and threshold the pixel intensities to make a binary image where everything except for the darkest part of the eye (the pupil) is filtered out.

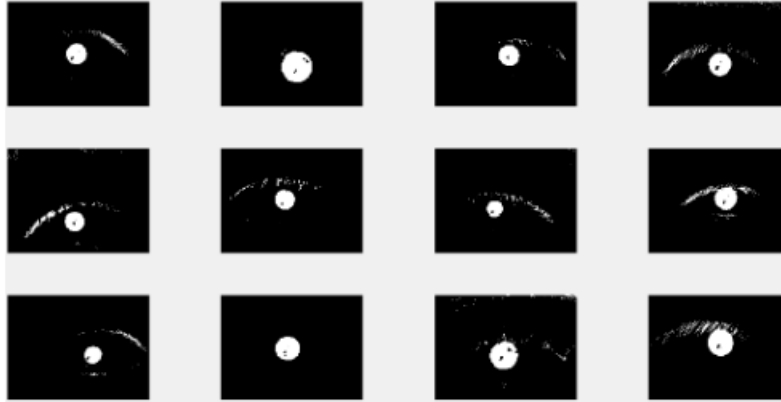


Figure 4: Pupil detection using a threshold mask on the original grayscale image to produce a binary image. The threshold value for pixel intensity that we determined to work reasonably well after some trial and error was 50.

```
1 %Pupil detection threshold (empirically determined)
2 eyes(img).pupil = eyes(img).orig < 50;
```

Then, we used the Circular Hough Transform on each binary image, shown in Figure 5. Hough Transform is used for feature extraction or line detection from binary images. Although we have our edge map from the Canny Edge Detector, the Circular Hough Transform was not able to detect the iris edge from the edge map. Therefore, we used the Circular Hough Transform on the binary image of the pupil to find the center and radius of the pupil. Since the iris is just a larger concentric circle around the pupil, we then traversed radially outward from the edge of the pupil in the edge map that we found earlier until another edge was detected. To ensure the next detected edge was not an eyelid or eyelash, we traversed diagonally downward to the right from the pupil edge.

```
1 %Circular Hough Transform
2 [centers, radii] = imfindcircles(eyes(img).pupil,[6 100]);
3
4 % retain the strongest circle (pupil edge)
5 coords = centers(1,:); pupil_radius = radii(1);
6
7 % start from lower right of pupil edge
8 x = round(coords(1))+round(pupil_radius);
9 y = round(coords(2))+round(pupil_radius);
10 while 1 % traverse radially outward (rightward/downward)
11     if eyes(img).edge(y,x) == 1 % iris edge detected
12         % calculate radial distance from pupil to iris edge
13         iris_radius = ((x-coords(1))^2 + (y-coords(2))^2)^0.5;
14         break
15     end
16     x = x+1; y = y+1;
17     if x == 240 || y == 320 % if no iris edge was detected
18         iris_radius = pupil_radius*2; % set iris radius to 2x pupil radius
19         break
20     end
21 end
```

### 4.4 Iris Extraction

Now that we have detected the inner and outer edges of the iris, we can extract the iris portion, our region of interest (ROI), from the original grayscale image, shown in Figure 6. We use the circle center coordinates (coords(1), coords(2)) and the pupil/iris radii found previously to determine our region of interest.

```
1 % extract iris
2 % return 2-D grid of coordinates for grayscale eye image
3 [xgrid, ygrid] = meshgrid(1:size(eyes(img).orig,2), 1:size(eyes(img).orig,1));
```

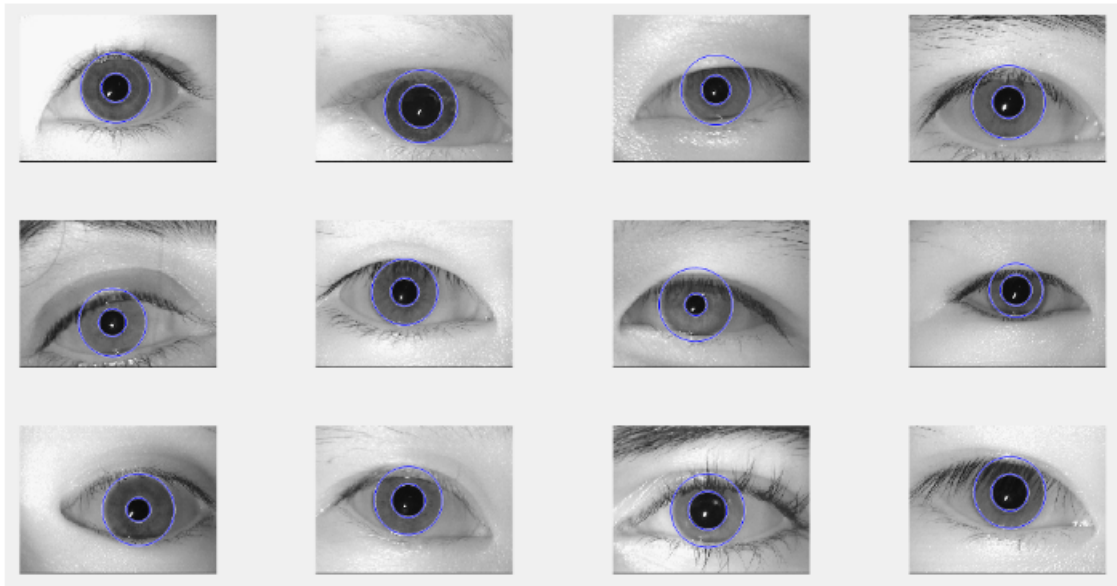


Figure 5: Circular Hough Transform results for all twelve sample images. We based the iris extraction mask on the location of the pupil and traversed radially outward from the edge of the pupil to find the iris edge from the edge map. All pupils are perfectly detected and outlined, whereas the iris outlines are reasonably accurate. The fourth image in the second row features an iris outline that is slightly within the manually distinguishable iris, but the rest of the results are as expected.

```

4 % filter out anything outside iris and anything inside pupil
5 % pixel value = 1 for region of interest, 0 for non-ROI
6 mask = ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) <= iris_radius.^2 &...
7         ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) >= pupil_radius.^2;
8 % apply mask to original grayscale image to get extracted iris
9 eyes(img).iris = eyes(img).orig .* uint8(mask);
10
11 imshow(eyes(img).iris);

```

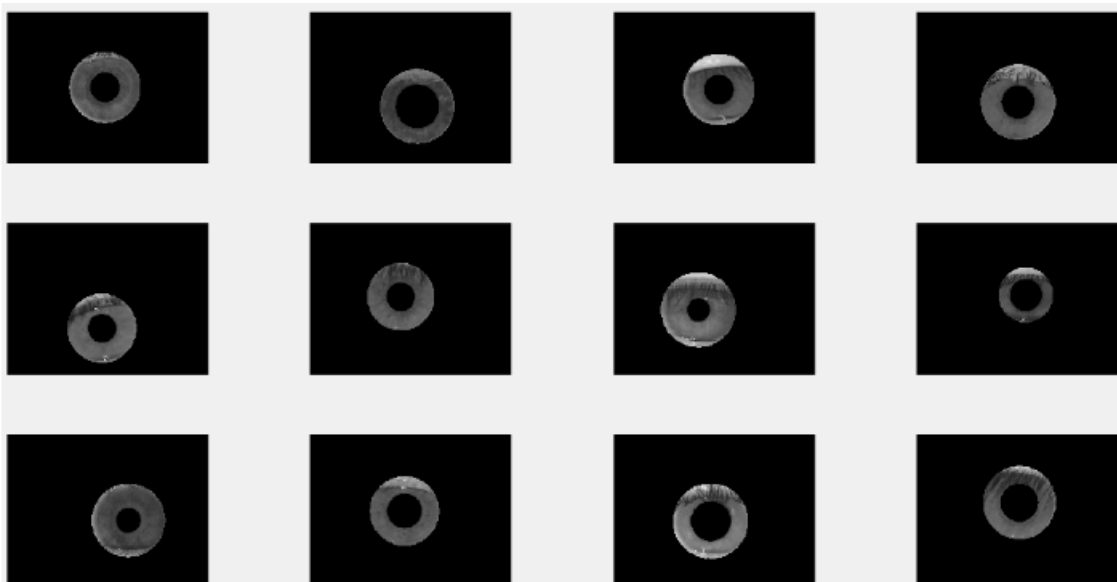


Figure 6: Extracted irises for all twelve sample images. The region between the two concentric circles (iris outer edge and pupil edge/iris inner edge) is kept, while everything else is filtered out.

## 4.5 Image Enhancement: Removal of Extreme Pixels and Increasing Contrast

With the iris extracted, we can apply thresholding to remove unwanted eyelashes, eyelids, and reflections. These are commonly found in our data set and could present unwanted errors when extracting features from the iris. In Figure 7, we resize the image to only include the iris and threshold the pixel intensities to make a binary image where everything except for the iris is filtered out.

```
1 %Image enhancement
2 %round estimate values for radius and center point
3 rounded_coords = round(coords);
4 round_iris_r = round(iris_radius);
5 round_pupil_r = round(pupil_radius);
6
7 %Resize image to only include iris pixels
8 eyes(img).iris_only = eyes(img).iris(rounded_coords(2)- ...
9     round_iris_r:rounded_coords(2)+ round_iris_r, rounded_coords(1)- ...
10    round_iris_r:rounded_coords(1)+round_iris_r);
11
12 %Remove extreme pixels (eyelid, eyelash, reflection on eye)
13 eyes(img).iris_only = uint8(eyes(img).iris_only > 50) .* ...
14 uint8(eyes(img).iris_only < 130) .* eyes(img).iris_only;
```

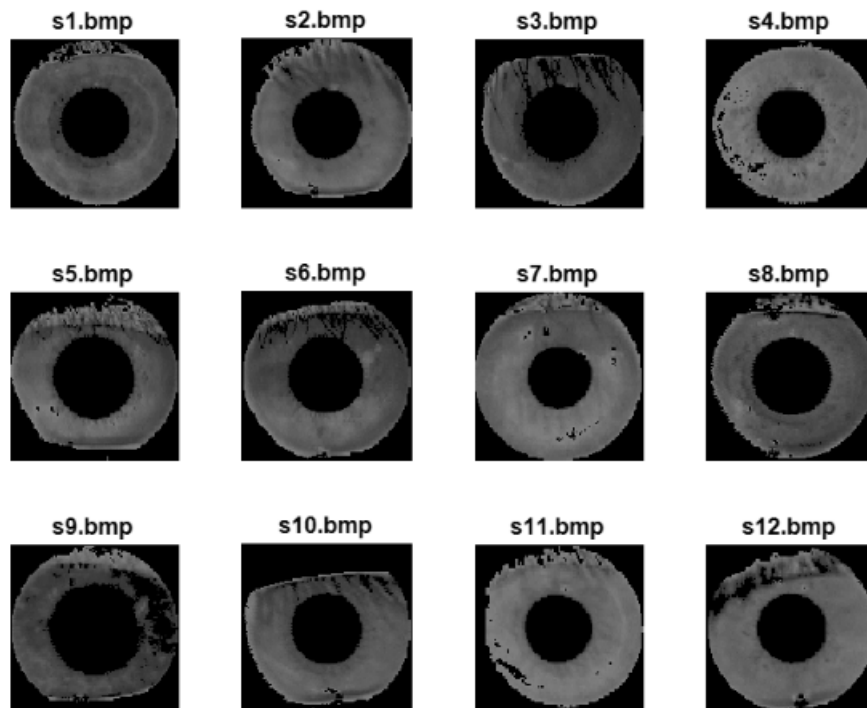


Figure 7: Removal of non-iris pixels using a threshold mask on the resized grayscale image of the iris. The threshold value for pixel intensity that we determined to work reasonably well after some trial and error was pixels  $> 50$  for eyelash removal and pixels  $> 130$  for glare/skin removal.

In addition to removing unwanted pixels, we can further enhance the image by increasing the contrast of the iris by remapping the image intensity values to the full display range of the intensity values. Since we removed some pixels through thresholding, this will greatly increase the contrast present in the iris. This will help with feature detection as there will be a larger spread in intensity values. This increase in contrast can easily be seen in comparison from Figure 7 to Figure 8.

```
1 %Increase contrast
2 [M,N] = size(eyes(img).iris_only);
3
4 for x=1:M
5     for y=1:N
6         %ignore thresholded pixels
7         if eyes(img).iris_only(y,x) == 0
```



```

8     continue
9     end
10    %remap [51,149] intensities to [0,255]
11    eyes(img).iris_only(y,x) = ((double(eyes(img).iris_only(y,x))-50)/(130-50))*255;
12    end
13    end

```

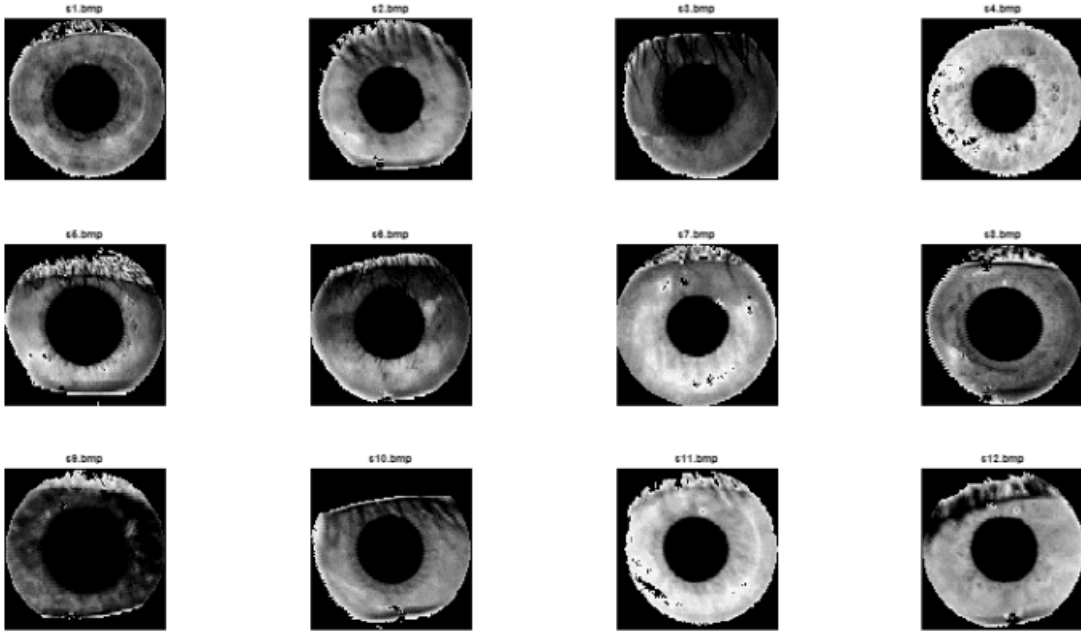


Figure 8: Increasing contrast of iris by remapping intensity values to full scale range of intensity values. Since we removed pixels with intensity values  $> 50$  and intensity values  $> 150$ , we were able to remap the remaining pixels to the full scale. In our case, we were left with pixel intensities from  $[51, 149]$  and were able to extend this range to  $[0, 255]$ , giving a higher contrast.

## 4.6 Iris Normalization

Since the iris may be captured in different sizes due to varying distances from the camera or pupils may be dilated, we can remap each pixel in the iris from Cartesian to polar coordinates. We normalize this mapping, such that it is a fixed-size rectangular block, regardless of pupil dilation, and different radii sizes of irises, as mentioned before [8]. We achieve this normalization by selecting a set number of thetas (angular resolution) to sample and then sampling a set number of pixels radially outward (radial resolution) at those thetas [4]. Figure 9 shows a visual representation of this. Depending on what number of pixels we set for the radial resolution, interpolation will occur if not enough pixels are present in the image at the theta. To do this we use `ImToPolar`, a MATLAB function [9].

```

1 %Iris normalization
2 eyes(img).polar = ImToPolar(eyes(img).iris_only, ...
3     round_pupil_r/round_iris_r, round_iris_r/round_iris_r, 40, 320);

```

## 4.7 Further Image Enhancement: Further Removal of Noisy Pixels

As can be seen in Figure 10, there is a lot of noise in the middle of the normalized iris and on the borders of the image due to our removal of eyelids and eyelashes from the previous steps. This can introduce unwanted features during our feature extraction stage and ideally should be excluded from the image to increase the accuracy and performance of the authenticator. In order to achieve this, we will remove the entire portion of the image that includes the eyelids and eyelashes, to create an image consisting of only the iris, shown in Figure 11.

To test if this image enhancement is viable, we will compare feature extractions/feature matching with and without removing the eyelids and eyelashes.

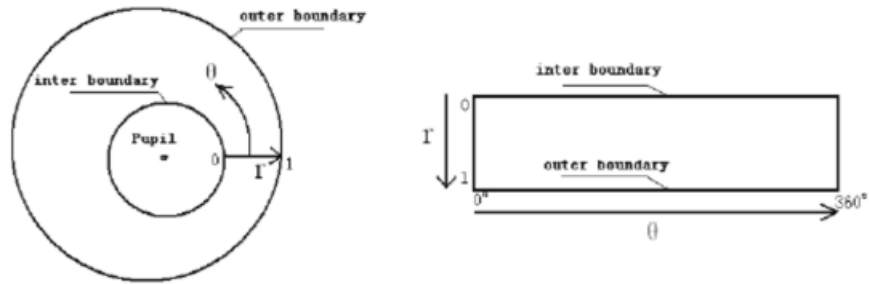


Figure 9: Visualization of iris normalization based on Rubber Sheet Normalization [5]. Theta is increased while pixels are sampled radially outward towards radius,  $r$ . Since the image will be normalized to a sheet of  $M \times N$  pixels, if there are not enough pixels present at the current theta, then pixels will be interpolated to fit the sheet at that theta.

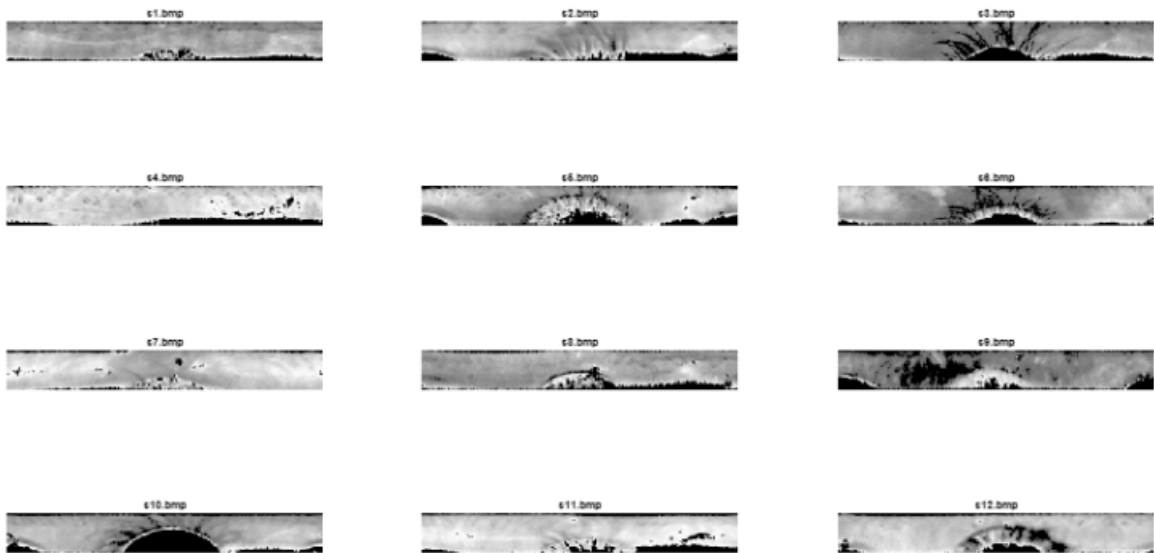


Figure 10: Iris normalization based on Rubber Sheet Normalization. Here we can see the iris being normalized to a sheet. In the middle of the sheets you can see the eyelids and eyelashes which demonstrates stretching out the iris onto the sheet. Through trial error, we chose to map the radii of the irises to 40 pixels and the thetas to 320 pixels. This produced reasonable results on the sheets of irises to be used for feature extraction.

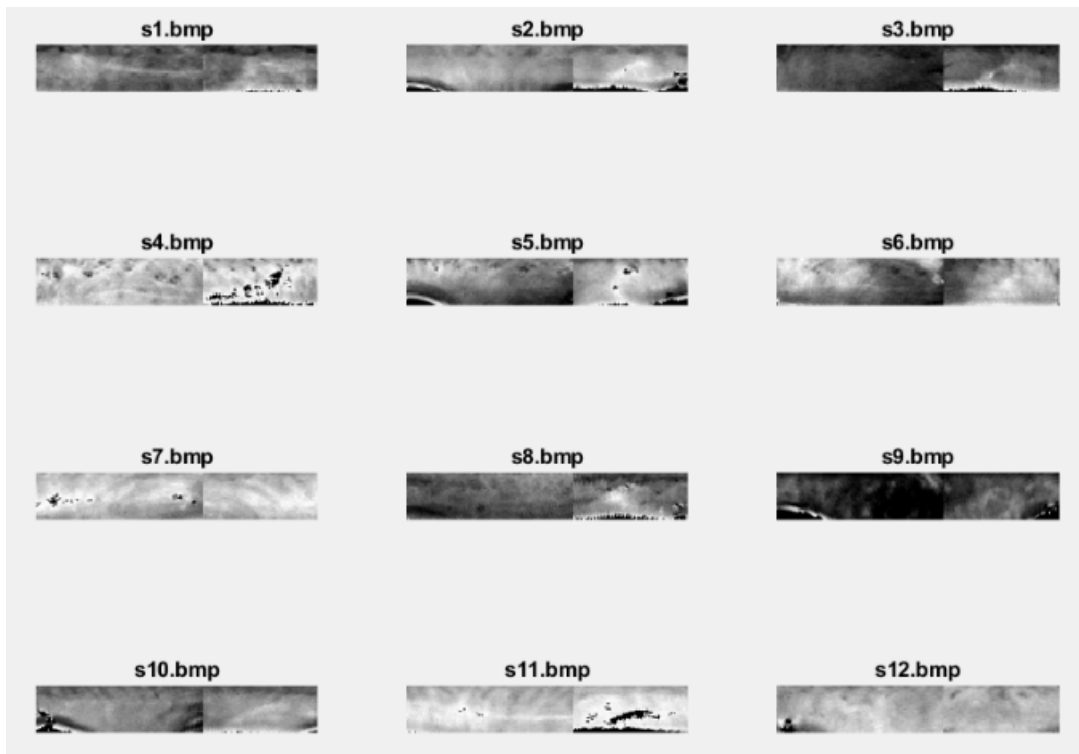


Figure 11: Removal of unwanted noise from eyelashes and eyelids from normalized iris images. Although not all of the noise is removed from the images, most of it is removed as compared to Figure 10, allowing us to get a more accurate iris image.

## 4.8 Feature Extraction

Once we have all the normalized images of the various training images, we then extract features from them. For our project, we first tried using SIFT for feature extraction. However, this proved to be ineffective for our iris images due to its inclination to choose feature points at any noise that was present, since they created strong corners. Instead, we used histogram of gradients, otherwise known as HOG, as it generally provides a good description of the structure of images rather than corners, as seen in SIFT [6]. This algorithm worked better with the images and code we used. The code and Figure 12 show our implementation of HOG and the resulting image.

```

1 %HOG feature extraction
2 cellSize = [2 2]; %used for HOG cell size, smaller = more small details, larger = more large
  details
3
4 [eyes(img).featureVector,hogVisualization] = extractHOGFeatures(uint8(eyes(img).polar), 'CellSize'
  , cellSize);
5
6 figure(9); subplot(4,3,img); imshow(uint8(eyes(img).polar));
7 title(strcat('s',string(img),'.bmp'));
8 hold on;
9 plot(hogVisualization);
10 hold off;

```

In order to pick up smaller details in the iris we will also reduce the cell size of the HOG feature to 2x2 pixels. This reduction in cell size increases the HOG's ability to capture small details in the image, which is useful in our case. In turn, this increases our computation time. However, we do not see a noticeable difference in computation time in our testing, most likely due to our images having small pixel densities [10].

In addition to using HOG, we also used a 5x5 block of average pixel intensities as a simple feature extraction method. This essentially applied a 5x5 average filter across the entire image, giving a vector that can be used to match against input irises. The code is shown below.

```

1 average_kernal = fspecial('average', AVERAGE_FILTER_SIZE); %create 5x5 filter
2 eyes(img).average_featureVector = imfilter(uint8(eyes(img).polar), average_kernal); %apply filter
  over entire polar image

```

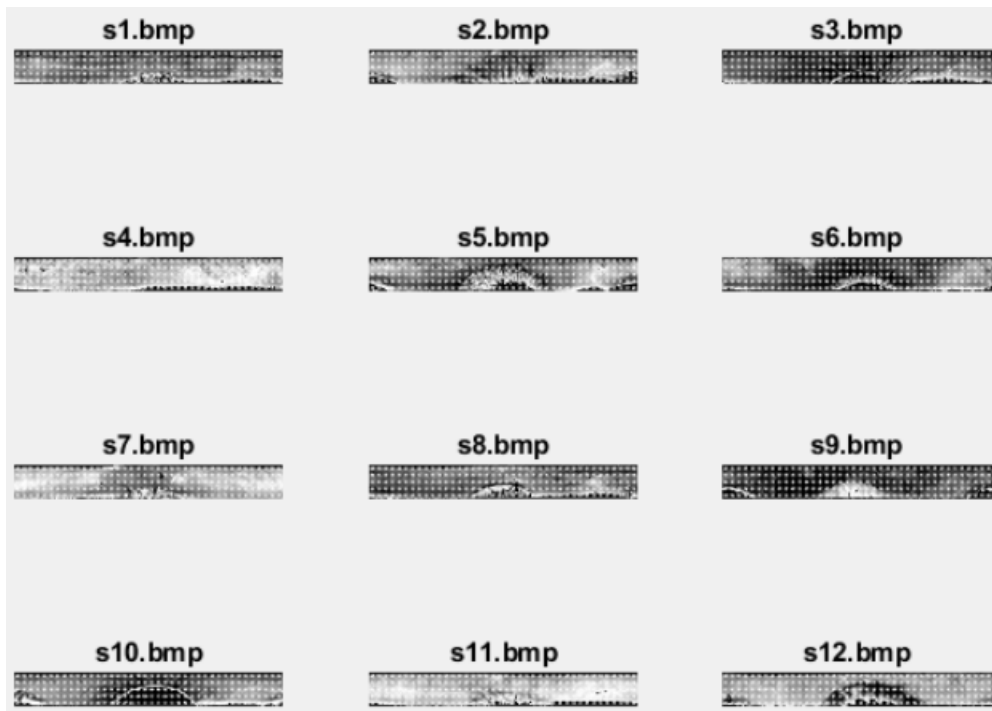


Figure 12: Feature extraction using HOG on normalized iris images. Each image has its own set of features that are different from the others. This allows us to do the same feature extraction on the test images and match them together.

## 4.9 Feature Matching

The final step in authenticating different irises in our system is matching the extracted features to different iris images. To achieve this we deployed two different methods, one without machine learning and one with it. Our first approach utilized Euclidean distance as a simple way to calculate the mathematical similarity between one training image and an input image. Our second approach used a supervised machine learning algorithm, in our case, Support Vector Machine (SVM), to train from different iris images, allowing it to classify multiple irises more accurately than normal matching algorithms [11].

To implement the Euclidean distance, we simply used the norm function in MATLAB which calculates the distances accordingly between two vectors, which in our case is the HOG feature vector. This result would give us the mathematical distance between two different vectors with a lower number meaning a closer distance. Using this value we can empirically find a threshold value to find feature vectors that are close in value and we will be able to authenticate whether the iris is the same or not.

```

1 % Eucl_distance = norm(eyes(1).featureVector - eyes_test(1).featureVector);
2 % For HOG feature
3 % 12 sample images, so loop 12 times
4 fprintf("Begin HOG feature test-----\n")
5 for img = 1:NUM_OF_REG_EYES
6     for img_test = 1:NUM_OF_REG_EYES
7         Eucl_distance = norm(eyes(img).featureVector - eyes_test(img_test).featureVector);
8         if Eucl_distance < 2.7 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
9             Authenticated(img, img_test) = 1;
10            if img == img_test
11                fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", img, img_test);
12            else
13                fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", img, img_test);
14            end
15        else
16            Authenticated(img, img_test) = 0;
17        end
18    end
19 end

```

On the other hand, for the SVM, we used a MATLAB function called `fitcecoc` which enables SVM to be used with Error-Correcting Output Codes classified, ECOC classifier for short. This classifier consists of multiple binary learners, like SVM, to do multiclass classifying [12]. SVM in simple terms finds a hyperplane that creates a boundary between the data that it is given. For example, in a simple 2D space this hyperplane would be a simple line separating two different data. However, this can be extended to an N-dimensional plane, allowing for high complexity and separation of data that is non-linear [11]. However, traditional SVM only allows for the classification of binary problems, whether the data is X or if it is Y. To solve this problem and to be more useful for our issue, we can use SVM in conjunction with the ECOC classifier to solve multiclass problems. In essence, the ECOC classifier combines multiple SVM hyperplane binary classifiers into one to create multiple boundaries for different classes. This concept can be further explained in Figure 13. Once this SVM with ECOC classifier is trained, it can predict the class of the input iris image. The code is shown below for the training of the SVM with the ECOC classifier.

```

1 %% Feature matching/Classifier for SVM ECOC...
2 % SVM binary classifier
3 % For HOG feature
4 % 12 sample images, so loop 12 times
5 fprintf("Begin SVM ECOC feature test-----\n")
6 features = [];
7
8 %create Nx1 matrix with each row containing a feature vector for image
9 for i = 1:NUM_OF_REG_EYES*NUM_OF_TRAIN_IMAGES_PER_EYE
10     feature = eyes_train(i).featureVector; %feature used to train
11     features = cat(1,features,feature); %put all futures into one vector
12 end
13
14 %create class labels for each feature vector
15 Z = []; %holds the class labels for the feature vector
16 for i = 1:NUM_OF_TRAIN_EYES %create a class array that goes with features
17     Y = ones(NUM_OF_TRAIN_IMAGES_PER_EYE,1) * i;
18     Z = cat(1,Z,Y);
19 end
20
21 %SVM model for binary classification
22 t = templateSVM('Standardize',true,'KernelFunction','rbf');
23
24 %ECOC classifier multiclassification with one v all
25 SVMmodels = fitcecoc(features,categorical(Z),'Learners',t, Coding='onevsall');

```

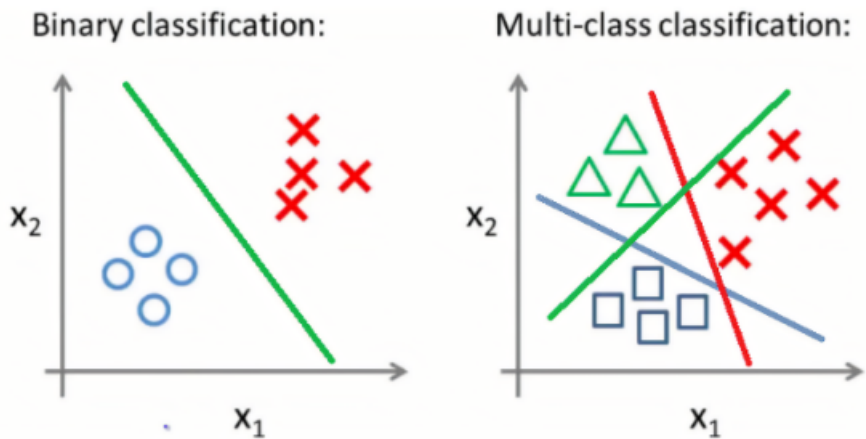


Figure 13: The main difference between binary classification and multiclass classification is the number of hyperplanes that separates the data [13]. This increase in the number of hyperplanes allows for multiclass classification in a one vs all algorithm.

## 5 Testing

With the feature extraction and feature matching providing suitable results, we can now test different iris images in the iris detection authenticator (IDA). To mimic a real-world situation, we chose to register 12 subjects with 4

training images and 1 test image per subject. We also had 32 subjects with 5 test images per subject who were not registered into the system. The images for training included various translations, rotations, light intensity, and occlusions, such as glasses. This allowed the training to be more thorough and give a better understanding of the capabilities of the IDA. The results for testing are listed in Table 1 and Table 2. We tried different permutations of feature extraction and feature matching pairs and have listed the code for testing and results below.

```

1 %% Feature matching/Classifier for Euclidian Distance...
2
3 % Eucl_distance = norm(eyes(1).featureVector - eyes_test(1).featureVector);
4 % For HOG feature
5 % 12 sample images, so loop 12 times
6 fprintf("Begin HOG feature test-----\n")
7 fail_count = 0;
8 for img = 1:NUM_OF_REG_EYES
9     for img_test = 1:NUM_OF_REG_EYES
10        Eucl_distance = norm(eyes(img).featureVector - eyes_test(img_test).featureVector);
11        if Eucl_distance < 32.3 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
12            Authenticated_HOG(img, img_test) = 1;
13            if img == img_test
14                fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", img, img_test);
15            else
16                fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", img, img_test);
17            end
18        else
19            Authenticated_HOG(img, img_test) = 0;
20        end
21    end
22    for img_test = 1:NUM_OF_UNREG_EYES*NUM_OF_UNREG_IMAGES_PER_EYE
23        Eucl_distance = norm(eyes(img).featureVector - eyes_unreg(img_test).featureVector);
24        if Eucl_distance < 32.3 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
25            fail_count = fail_count + 1;
26            fprintf("Match found for eye(%d) with eye_unreg(%d), [FAIL] \n", img, round(img_test
27                /4));
28        end
29    end
30 end
31 for img = 1:NUM_OF_REG_EYES
32     if sum(Authenticated_HOG(img, :))
33     else
34         fprintf("Match NOT found for eye(%d) \n", img)
35     end
36 end
37
38 fprintf("Number of False Postivies = %d [FAIL]\n", fail_count);

```

HOG/Euclidean	Registered Iris	Unregistered Iris
Predicted Registered Iris	TP = 9	FP = 9
Predicted Unregistered Iris	FN = 3	TN = 151

Table 1: Confusion matrix for HOG features with Euclidean distance matching. A total of 12 registered subjects with 4 training and 1 test image per subject. In addition, a total of 32 subjects with 5 test images per subject who were not registered.  $Total\ error = \frac{9+3}{12+160} = 7.0\%$ . TP = True Positive, FP = False Positive, FN = False Negative, and TN = True Negative.

5x5/Euclidean	Registered Iris	Unregistered Iris
Predicted Registered Iris	TP = 5	FP = 94
Predicted Unregistered Iris	FN = 7	TN = 66

Table 2: Confusion matrix for 5x5 Average filter features with Euclidean distance matching. A total of 12 registered subjects with 4 training and 1 test image per subject. In addition, a total of 32 subjects with 5 test images per subject who were not registered.  $Total\ error = \frac{94+7}{12+160} = 58.7\%$

As mentioned before, we also tested removing noisy pixels, as seen in Figure 11, and we found that this did not hinder the testing results and only marginally improved test results. This resulted in findings seen in Table 1 and

Table 2. We decided to keep this image enhancement step as it seemed to help with the algorithm, even if only a little.

We also tried testing SVM with HOG/5x5 but had very inaccurate results. When we tested SVM, we found that it was unable to match any of the registered eyes to its respective persons and found that its ability to differentiate between different irises was poor. Although we are unsure of the exact reason why it did not perform well, we think it has to do with the relatively small amount of training that SVM had to train from. Many SVM-based systems have at least ten times more training data than their testing data and are usually in the range of hundreds or thousands of images [14]. Our training only had 4 images and our testing only consisted of 1 test image, due to the small data set we received. Considering how complex the iris pattern is, this is the conclusion we can come to in the high error rate of our SVM system.

## 6 Analysis and Evaluation of Results

As far as our algorithm goes, the iris was successfully detected through a combination of canny edge detection, thresholding, and Circular Hough Transform. The iris was detected with great accuracy for the most part, even with eyelashes and eyelids occluding the iris. In addition to iris detection, we were able to successfully extract the iris and were able to preprocess the image for feature detection. The iris was fully extracted with great accuracy, but the removal of the eyelids and eyelashes have proved to be difficult and causes issues in extraction due to occlusion. We also normalized the iris following Daugman’s Rubber Sheet Normalization method, allowing our irises to be consistent in between feature extraction.

Although our algorithm works on our database, there are some limitations to our design. Firstly, the eye images need to be open enough such that pupils/irises are unobstructed, otherwise it is almost impossible to extract the iris for obvious reasons. Secondly, the resolution of the image needs to be fairly high as iris patterns are very subtle and the more pixels to work with means better accuracy and results. Lastly, since we are using Circular Hough Transform (as opposed to ellipses), the eyes need to be facing the camera otherwise some iris patterns can be lost. We chose to go with the Circular Hough Transform as there are less parameters, making it less computationally expensive.

While we did see success in general with our algorithms we also want to highlight some issues that we encountered. For instance, one of the issues we saw was that the Circular Hough Transform does not detect the inner iris edge. Instead, to work around this problem, we used a pupil binary mask to find the pupil edge and then used the edge map from our Canny Edge Detector by traversing radially outward from the pupil edge until we reached an edge pixel. This effectively found iris rings and worked out well, as can be seen from our iris detection portion. Another issue we saw was with the feature extraction portion. We first attempted to use SIFT and Harris Corner detection, but that did not work for matching the iris features due to sharp corners caused by the eyelashes and eyelids. So instead we switched to HOG to obtain feature descriptors, as it was better at detecting small details across the entire image. This worked better when we matched the images using Euclidean distance, giving a relatively low error rate of 7%. Continuing with this testing method, we can extrapolate that the error rate will only go down as we add more training data and test images. However, as seen in the test results, the 5x5 average filter features did not perform well and this can most likely be attributed to the complex pattern present in the irises. Thus, we can conclude that it is not suitable as a feature extraction method for irises while HOG is. We also saw issues with SVM which was discussed in depth in the testing portion of this report.

Given more time, we would have tried using the Ellipse Hough Transform in case the eyes were not directly frontward facing. In addition, we would have tried more descriptors for the irises since HOG, although a great descriptor for human bodies and shapes, did not give convincing and distinct descriptors for our iris images. We would also try Haar-like features or Fourier descriptors, since those work well with human physical features and seem to be implemented in some iris recognition algorithms. In terms of matching, we would have trained the SVM further as it seemed to provide suitable results when given a good amount of data. We would have also tried template matching pixel by pixel from our trained data to the test data. If we were to start this project over again, we would have spent more time on developing the descriptors, as this needs heavy engineering in comparison to other portions of the algorithms and, is by far, the most important part in reliability and security for biometric systems.

## 7 Conclusion

As previously mentioned in the analysis, the algorithms up until feature extraction had no issues. Iris detection worked with high accuracy for circular regions. Image enhancement and normalization also worked well for the data set we used. While eyelashes and eyelids posed an issue for feature extraction, we were still able to obtain adequate feature descriptors for feature matching. With the time constraints, we were able to develop a decent algorithm for validating a person's irises with registered iris data, but the results we have are a good indicator that our system can be further improved. In future iterations of this project, we would go back to the Hough Transform and use an elliptic version instead of the circular one we used in this project. We would also include more feature descriptors and compare which ones performed best with irises. Finally, we would also look for other databases that offer more training data with better resolution to get even better results in our feature matching algorithm. Overall, in this project, we successfully found and extracted irises and developed a working algorithm for iris recognition.



## References

- [1] N. Mohammad, *Mmu iris dataset, multimedia university iris database for biometric attendance system*, Jul. 2020.
- [2] S. K. Chawla and A. Oberoi, “A robust algorithm for iris segmentation and normalization using hough transform,” Nov. 2011.
- [3] A. Banitalebi and S. A. R. Abu Bakar, “Adaptive fuzzy switching noise reduction filter for iris pattern recognition,” *Jurnal Teknologi*, vol. 73, pp. 1–2015, Feb. 2015. DOI: [10.11113/jt.v73.3381](https://doi.org/10.11113/jt.v73.3381).
- [4] F. Jiménez L., C. Pardo-Beainy, and O. Mendez, “Biometric iris recognition using hough transform,” Sep. 2013, pp. 1–6, ISBN: 978-1-4799-1121-9. DOI: [10.1109/STSIWA.2013.6644905](https://doi.org/10.1109/STSIWA.2013.6644905).
- [5] R. Ng, Y. H. Tay, and K. Mok, “A review of iris recognition algorithms,” vol. 2, Sep. 2008, pp. 1–7, ISBN: 978-1-4244-2327-9. DOI: [10.1109/ITSIM.2008.4631656](https://doi.org/10.1109/ITSIM.2008.4631656).
- [6] M. Sultana, T. Ahmed, P. Chakraborty, M. Khatun, M. R. Hasan, and M. Uddin, “Object detection using template and hog feature matching,” *International Journal of Advanced Computer Science and Applications*, vol. 11, pp. 233–238, Jul. 2020. DOI: [10.14569/IJACSA.2020.0110730](https://doi.org/10.14569/IJACSA.2020.0110730).
- [7] P. Verma, M. Dubey, S. Basu, and P. K. Verma, “Hough transform method for iris recognition-a biometric approach,” 2012.
- [8] Mohammadi Arvacheh, Ehsan, “A study of segmentation and normalization for iris recognition systems,” 2006. [Online]. Available: <http://hdl.handle.net/10012/2846>.
- [9] P. Manandhar, *Polar to/from rectangular transform of images*, Dec. 2007.
- [10] M. Tyagi, “Hog (histogram of oriented gradients): An overview,” *Towards Data Science*, Jul. 2021.
- [11] alokesh985 and om\_agarwal\_2411, “Introduction to support vector machines (svm),” *GeeksforGeeks*, May 2022.
- [12] *Classificationecoc, multiclass model for support vector machines (svms) and other classifiers*, 2014.
- [13] Utku, “Onevsall classification using logistic regression,” *Utku’s Blog*, Apr. 2022.
- [14] T. Kavzoglu and I. Colkesen, “The effects of training set size for performance of support vector machines and decision trees,” Jul. 2012.

# Appendix

## Appendix A - Matlab Functions

### iris\_detection.m

```
1 %% Clear all/Close all
2 close all
3 clear all
4 clc
5 %% Iris Detection template subset
6
7 FIGURE_ON = 1;
8 NUM_OF_REG_EYES = 12;
9 REMOVE_CENTER_AND_EDGES = 1;
10 AVERAGE_FILTER_SIZE = 5;
11
12 cellSize = [2 2]; %used for HOG cell size, smaller = more small details, larger = more large
    details
13
14 % struct to hold grayscale eye images, edge maps, pupil extraction, and
15 % iris extraction
16 eyes = struct();
17
18
19
20 % N sample images, so loop N times
21 for img = 1:NUM_OF_REG_EYES
22     %load in images (all are grayscale bitmaps of size 320x240)
23     % from MMU Iris database
24     eyes(img).orig = im2gray(imread(strcat('s',string(img),'.bmp')));
25
26     %Edge Detection: Canny Edge Detector
27     eyes(img).edge = edge(eyes(img).orig,'canny',0.15,3);
28
29     %plotting canny edge images
30     %     figure(2); subplot(3,4,img); imshow(eyes(img).edge);
31     %     title(strcat('s',string(img),'.bmp'))
32
33     %Pupil detection threshold (empirically determined)
34     eyes(img).pupil = eyes(img).orig < 50;
35
36     %Circular Hough Transform
37     [centers, radii] = imfindcircles(eyes(img).pupil,[6 100]);
38
39     % retain the strongest circle (pupil edge)
40     coords = centers(1,:);%center = P-by-2 matrix, P = number of circles, x in 1st column, y in 2
    nd column
41     pupil_radius = radii(1);
42
43     % start from lower right of pupil edge
44     x = round(coords(1))+round(pupil_radius);
45     y = round(coords(2))+round(pupil_radius);
46     while 1 % traverse radially outward (rightward/downward)
47         if eyes(img).edge(y,x) == 1 % iris edge detected
48             % calculate radial distance from pupil to iris edge
49             % calculate radial distance from center of pupil iris edge
50             iris_radius = ((x-coords(1))^2 + (y-coords(2))^2)^0.5;
51             break
52         end
53         x = x+1; y = y+1;
54         if x == 240 || y == 320 % if no iris edge was detected
55             iris_radius = pupil_radius*2; % set iris radius to 2x pupil radius
56             break
57         end
58     end
59
60     %plotting original picture with iris and pupil identified
61     figure(3); subplot(3,4,img); imshow(eyes(img).orig);
62     title(strcat('s',string(img),'.bmp'))
```

```

63 % draw pupil circle
64 viscircles(centers(1,:), pupil_radius, 'Color','b','LineWidth',0.5);
65 % draw iris circle
66 viscircles(centers(1,:), iris_radius, 'Color','b','LineWidth',0.5);
67
68 % extract iris
69 % return 2-D grid of coordinates for grayscale eye image
70 [xgrid, ygrid] = meshgrid(1:size(eyes(img).orig,2), 1:size(eyes(img).orig,1));
71 % filter out anything outside iris and anything inside pupil
72 % pixel value = 1 for region of interest, 0 for non-ROI
73 mask = ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) <= iris_radius.^2 &...
74         ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) >= pupil_radius.^2;
75 % apply mask to original grayscale image to get extracted iris
76 eyes(img).iris = eyes(img).orig .* uint8(mask);
77
78 % plotting iris extracted
79 figure(4); subplot(3,4,img); imshow(eyes(img).iris);
80 title(strcat('s',string(img),'_bmp'))
81
82 %Image enhancement
83 %round estimate values for radii and center point
84 rounded_coords = round(coords);
85 round_iris_r = round(iris_radius);
86 round_pupil_r = round(pupil_radius);
87
88 %Resize image to only include iris
89 eyes(img).iris_only = eyes(img).iris(rounded_coords(2)- ...
90     round_iris_r:rounded_coords(2)+ round_iris_r, rounded_coords(1)- ...
91     round_iris_r:rounded_coords(1)+round_iris_r);
92 figure(5); subplot(3,4,img); imshow(eyes(img).iris_only);
93 title(strcat('s',string(img),'_bmp'))
94
95 %Remove extreme pixels(eyelid, eyelash, reflection on eye)
96 eyes(img).iris_only = uint8(eyes(img).iris_only > 50) .* ...
97 uint8(eyes(img).iris_only < 130) .* eyes(img).iris_only;
98
99 figure(6); subplot(3,4,img); imshow(eyes(img).iris_only);
100 title(strcat('s',string(img),'_bmp'))
101
102 %Increase contrast
103 [M,N] = size(eyes(img).iris_only);
104
105 for x=1:M
106     for y=1:N
107         %ignore thresholded pixels
108         if eyes(img).iris_only(y,x) == 0
109             continue
110         end
111         %remap [51,149] intensities to [0,255]
112         eyes(img).iris_only(y,x) = ((double(eyes(img).iris_only(y,x))-50)/(130-50))*255;
113     end
114 end
115
116 figure(7); subplot(3,4,img); imshow(eyes(img).iris_only, []);
117 title(strcat('s',string(img),'_bmp'))
118
119 eyes(img).iris_only = uint8(eyes(img).iris_only);
120
121 %Iris normalization
122 eyes(img).polar = ImToPolar(eyes(img).iris_only, ...
123     round_pupil_r/round_iris_r, round_iris_r/round_iris_r, 40, 320);
124
125 %cut from 110-240 to get rid of eyelashes.... maybe
126 if REMOVE_CENTER_AND_EDGES
127     eyes(img).polar = [eyes(img).polar(5:end-5,1:110) eyes(img).polar(5:end-5,240:end-5)];
128 end
129
130 figure(8); subplot(4,3,img); imshow(eyes(img).polar, []);
131 title(strcat('s',string(img),'_bmp'))
132
133 %Extracting HOG features...

```

```

134 [eyes(img).featureVector,hogVisualization] = extractHOGFeatures(uint8(eyes(img).polar), '
135 CellSize', cellSize);
136
137
138 average_kernal = fspecial('average', AVERAGE_FILTER_SIZE); %create 5x5 filter
139 eyes(img).average_featureVector = imfilter(uint8(eyes(img).polar), average_kernal); %apply
140 filter over entire polar image
141
142 figure(16); subplot(4,3,img); imshow(eyes(img).average_featureVector);
143
144 figure(9); subplot(4,3,img); imshow(uint8(eyes(img).polar));
145 hold on;
146 plot(hogVisualization);
147 hold off;
148
149
150 % eyes(img).features = detectHarrisFeatures(uint8(eyes(img).polar));
151 % im_w_detected = insertMarker(uint8(eyes(img).polar), eyes(img).features, 'circle');
152 % figure(9); subplot(3,4,img); imshow(im_w_detected, []);
153 % title(strcat('s',string(img),'.bmp'))
154
155 end
156
157 %% Registered Test images subset
158
159 NUM_OF_TEST_EYES = 12;
160
161 % struct to hold grayscale eye images, edge maps, pupil extraction, and
162 % iris extraction
163 eyes_test = struct();
164
165 num_of_unregistered_eyes = 12;
166
167 % N sample images, so loop N times
168 for img = 1:num_of_unregistered_eyes
169 %load in images (all are grayscale bitmaps of size 320x240)
170 % from MMU Iris database
171 eyes_test(img).orig = im2gray(imread(strcat('t1_',string(img),'.bmp')));
172
173 %Edge Detection: Canny Edge Detector
174 eyes_test(img).edge = edge(eyes_test(img).orig, 'canny',0.15,3);
175
176 %plotting canny edge images
177 % figure(2); subplot(3,4,img); imshow(eyes(img).edge);
178 % title(strcat('s',string(img),'.bmp'))
179
180 %Pupil detection threshold (empirically determined)
181 eyes_test(img).pupil = eyes_test(img).orig < 50;
182
183 %Circular Hough Transform
184 [centers, radii] = imfindcircles(eyes_test(img).pupil,[6 100]);
185
186 % retain the strongest circle (pupil edge)
187 coords = centers(1,:);%center = P-by-2 matrix, P = number of circles, x in 1st column, y in 2
188 nd column
189 pupil_radius = radii(1);
190
191 % start from lower right of pupil edge
192 x = round(coords(1))+round(pupil_radius);
193 y = round(coords(2))+round(pupil_radius);
194 while 1 % traverse radially outward (rightward/downward)
195     if eyes_test(img).edge(y,x) == 1 % iris edge detected
196         % calculate radial distance from pupil to iris edge
197         % calculate radial distance from center of pupil iris edge
198         iris_radius = ((x-coords(1))^2 + (y-coords(2))^2)^0.5;
199         break
200     end
201     x = x+1; y = y+1;
202     if x == 240 || y == 320 % if no iris edge was detected

```

```

202         iris_radius = pupil_radius*2; % set iris radius to 2x pupil radius
203         break
204     end
205 end
206
207 %plotting original picture with iris and pupil identified
208 figure(10); subplot(3,4,img); imshow(eyes_test(img).orig);
209 title(strcat('t1-',string(img),'.bmp'))
210 % draw pupil circle
211 viscircles(centers(1,:), pupil_radius, 'Color','b','LineWidth',0.5);
212 % draw iris circle
213 viscircles(centers(1,:), iris_radius, 'Color','b','LineWidth',0.5);
214
215 % extract iris
216 % return 2-D grid of coordinates for grayscale eye image
217 [xgrid, ygrid] = meshgrid(1:size(eyes_test(img).orig,2), 1:size(eyes_test(img).orig,1));
218 % filter out anything outside iris and anything inside pupil
219 % pixel value = 1 for region of interest, 0 for non-ROI
220 mask = ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) <= iris_radius.^2 &...
221         ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) >= pupil_radius.^2;
222 % apply mask to original grayscale image to get extracted iris
223 eyes_test(img).iris = eyes_test(img).orig .* uint8(mask);
224
225 % plotting iris extracted
226 % figure(4); subplot(3,4,img); imshow(eyes(img).iris);
227 % title(strcat('s',string(img),'.bmp'))
228
229 %Image enhancement
230 %round estimate values for radii and center point
231 rounded_coords = round(coords);
232 round_iris_r = round(iris_radius);
233 round_pupil_r = round(pupil_radius);
234
235 %Resize image to only include iris
236 eyes_test(img).iris_only = eyes_test(img).iris(rounded_coords(2)- ...
237         round_iris_r:rounded_coords(2)+ round_iris_r, rounded_coords(1)- ...
238         round_iris_r:rounded_coords(1)+round_iris_r);
239 figure(11); subplot(3,4,img); imshow(eyes_test(img).iris_only);
240 title(strcat('t1-',string(img),'.bmp'))
241
242 %Remove extreme pixels(eyelid, eyelash, reflection on eye)
243 eyes_test(img).iris_only = uint8(eyes_test(img).iris_only > 50) .* ...
244     uint8(eyes_test(img).iris_only < 130) .* eyes_test(img).iris_only;
245
246 figure(12); subplot(3,4,img); imshow(eyes_test(img).iris_only);
247 title(strcat('t1-',string(img),'.bmp'))
248
249 %Increase contrast
250 [M,N] = size(eyes_test(img).iris_only);
251
252 for x=1:M
253     for y=1:N
254         if eyes_test(img).iris_only(y,x) == 0
255             continue
256         end
257         eyes_test(img).iris_only(y,x) = ((double(eyes_test(img).iris_only(y,x))-50)/(130-50))
258 *255;
259     end
260 end
261
262 figure(13); subplot(3,4,img); imshow(eyes_test(img).iris_only, []);
263 title(strcat('t1-',string(img),'.bmp'))
264
265 eyes_test(img).iris_only = uint8(eyes_test(img).iris_only);
266
267 %Iris normalization (rubber sheet algo)
268 eyes_test(img).polar = ImToPolar(eyes_test(img).iris_only, ...
269     round_pupil_r/round_iris_r, round_iris_r/round_iris_r, 40, 320);
270
271 %cut from 110-240 to get rid of eyelashes.... maybe
272 if REMOVE_CENTER_AND_EDGES

```

```

272     eyes_test(img).polar = [eyes_test(img).polar(5:end-5,1:110) eyes_test(img).polar(5:end
-5,240:end-5)];
273     end
274
275     figure(14); subplot(4,3,img); imshow(eyes_test(img).polar, []);
276     title(strcat('t1-',string(img),'.bmp'))
277
278     %Extracting HOG features...
279
280     [eyes_test(img).featureVector,hogVisualization] = extractHOGFeatures(uint8(eyes_test(img).
polar), 'CellSize', cellSize);
281
282     average_kernal = fspecial('average', AVERAGE_FILTER_SIZE);
283     eyes_test(img).average_featureVector = imfilter(uint8(eyes_test(img).polar), average_kernal);
284
285     figure(17); subplot(4,3,img); imshow(eyes_test(img).average_featureVector);
286
287     figure(15); subplot(4,3,img); imshow(uint8(eyes_test(img).polar));
288     hold on;
289     plot(hogVisualization);
290     hold off;
291 end
292
293 %% Unregistered Test images subset
294
295 NUM_OF_UNREG_EYES = 32;
296 NUM_OF_UNREG_IMAGES_PER_EYE = 4;
297
298 % struct to hold grayscale eye images, edge maps, pupil extraction, and
299 % iris extraction
300 eyes_unreg = struct();
301
302 image_folder = pwd;
303 image_folder =strcat(image_folder,'\unregistered_eyes');
304 filenames = dir(fullfile(image_folder, '*.bmp'));
305 total_images = NUM_OF_UNREG_EYES*NUM_OF_UNREG_IMAGES_PER_EYE;
306
307 % N sample images, so loop N times
308 for image_num = 1:total_images
309     f = fullfile(image_folder, filenames(image_num).name);
310     %load in images (all are grayscale bitmaps of size 320x240)
311     % from MMU Iris database
312     eyes_unreg(image_num).orig = im2gray(imread(f));
313
314 %     figure,imshow(f)
315
316 %Edge Detection: Canny Edge Detector
317 eyes_unreg(image_num).edge = edge(eyes_unreg(image_num).orig, 'canny',0.15,3);
318
319 %plotting canny edge images
320 %     figure(2); subplot(3,4,image_num); imshow(eyes_unreg(image_num).edge);
321 %     title(strcat('s',string(image_num),'.bmp'))
322
323 %Pupil detection threshold (empirically determined)
324 eyes_unreg(image_num).pupil = eyes_unreg(image_num).orig < 50;
325
326 %Circular Hough Transform
327 [centers, radii] = imfindcircles(eyes_unreg(image_num).pupil,[6 100]);
328
329 % retain the strongest circle (pupil edge)
330 coords = centers(1,:);%center = P-by-2 matrix, P = number of circles, x in 1st column, y in 2
nd column
331 pupil_radius = radii(1);
332
333 %     viscircles(coords, pupil_radius,'EdgeColor','b');
334
335 % start from lower right of pupil edge
336 x = round(coords(1))+round(pupil_radius);
337 y = round(coords(2))+round(pupil_radius);
338 while 1 % traverse radially outward (rightward/downward)
339     if eyes_unreg(image_num).edge(y,x) == 1 % iris edge detected

```

```

340         % calculate radial distance from pupil to iris edge
341         % calculate radial distance from center of pupil iris edge
342         iris_radius = ((x-coords(1))^2 + (y-coords(2))^2)^0.5;
343         break
344     end
345     x = x+1; y = y+1;
346     if x == 240 || y == 320 % if no iris edge was detected
347         iris_radius = pupil_radius*2; % set iris radius to 2x pupil radius
348         break
349     end
350 end
351
352 %      %plotting original picture with iris and pupil identified
353 %      figure(3); subplot(9,4,image_num); imshow(eyes_unreg(image_num).orig);
354 %      title(strcat('s',string(image_num),'.bmp'))
355 %      % draw pupil circle
356 %      viscircles(centers(1,:), pupil_radius,'Color','b','LineWidth',0.5);
357 %      % draw iris circle
358 %      viscircles(centers(1,:), iris_radius,'Color','b','LineWidth',0.5);
359
360 % extract iris
361 % return 2-D grid of coordinates for grayscale eye image
362 [xgrid, ygrid] = meshgrid(1:size(eyes_unreg(image_num).orig,2), 1:size(eyes_unreg(image_num).
orig,1));
363 % filter out anything outside iris and anything inside pupil
364 % pixel value = 1 for region of interest, 0 for non-ROI
365 mask = ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) <= iris_radius.^2 &...
366         ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) >= pupil_radius.^2;
367 % apply mask to original grayscale image to get extracted iris
368 eyes_unreg(image_num).iris = eyes_unreg(image_num).orig .* uint8(mask);
369
370 % plotting iris extracted
371 %      figure(4); subplot(3,4,image_num); imshow(eyes_unreg(image_num).iris);
372 %      title(strcat('s',string(image_num),'.bmp'))
373
374 %Image enhancement
375 %round estimate values for radii and center point
376 rounded_coords = round(coords);
377 round_iris_r = round(iris_radius);
378 round_pupil_r = round(pupil_radius);
379
380 %Resize image to only include iris
381 eyes_unreg(image_num).iris_only = eyes_unreg(image_num).iris(rounded_coords(2)- ...
382         round_iris_r:rounded_coords(2)+ round_iris_r, rounded_coords(1)- ...
383         round_iris_r:rounded_coords(1)+round_iris_r);
384 %      figure(5); subplot(9,4,image_num); imshow(eyes_unreg(image_num).iris_only);
385 %      title(strcat('s',string(image_num),'.bmp'))
386
387 %Remove extreme pixels(eyelid, eyelash, reflection on eye)
388 eyes_unreg(image_num).iris_only = uint8(eyes_unreg(image_num).iris_only > 50) .* ...
389         uint8(eyes_unreg(image_num).iris_only < 130) .* eyes_unreg(image_num).iris_only;
390
391 %      figure(6); subplot(9,4,image_num); imshow(eyes_unreg(image_num).iris_only);
392 %      title(strcat('s',string(image_num),'.bmp'))
393
394 %Increase contrast
395 [M,N] = size(eyes_unreg(image_num).iris_only);
396
397 for x=1:M
398     for y=1:N
399         %ignore thresholded pixels
400         if eyes_unreg(image_num).iris_only(y,x) == 0
401             continue
402         end
403         %remap [51,149] intensities to [0,255]
404         eyes_unreg(image_num).iris_only(y,x) = ((double(eyes_unreg(image_num).iris_only(y,x))
-50)/(130-50))*255;
405     end
406 end
407
408 %      figure(7); subplot(9,4,image_num); imshow(eyes_unreg(image_num).iris_only, []);

```

```

409 %     title(strcat('s',string(image_num),'.bmp'))
410
411     eyes_unreg(image_num).iris_only = uint8(eyes_unreg(image_num).iris_only);
412
413     %Iris normalization
414     eyes_unreg(image_num).polar = ImToPolar(eyes_unreg(image_num).iris_only, ...
415         round_pupil_r/round_iris_r, round_iris_r/round_iris_r, 40, 320);
416
417     %cut from 110-240 to get rid of eyelashes... maybe
418     if REMOVE_CENTER_AND_EDGES
419         eyes_unreg(image_num).polar = [eyes_unreg(image_num).polar(5:end-5,1:110) eyes_unreg(
420 image_num).polar(5:end-5,240:end-5)];
421     end
422 %     figure(8); subplot(9,4,image_num); imshow(eyes_unreg(image_num).polar, []);
423 %     title(strcat('s',string(image_num),'.bmp'))
424
425     %Extracting HOG features...
426
427     [eyes_unreg(image_num).featureVector,hogVisualization] = extractHOGFeatures(uint8(eyes_unreg(
428 image_num).polar), 'CellSize', cellSize);
429
430     average_kernel = fspecial('average', AVERAGE_FILTER_SIZE);
431     eyes_unreg(image_num).average_featureVector = imfilter(uint8(eyes_unreg(image_num).polar),
432 average_kernel);
433 %     figure(16); subplot(9,4,image_num); imshow(eyes_unreg(image_num).average_featureVector);
434 %
435 %     figure(9); subplot(9,4,image_num); imshow(uint8(eyes_unreg(image_num).polar));
436 %     hold on;
437 %     plot(hogVisualization);
438 %     hold off;
439
440
441
442 %     eyes_unreg(image_num).features = detectHarrisFeatures(uint8(eyes_unreg(image_num).polar));
443 %     im_w_detected = insertMarker(uint8(eyes_unreg(image_num).polar), eyes_unreg(image_num).
444 features, 'circle');
445 %     figure(9); subplot(3,4,image_num); imshow(im_w_detected, []);
446 %     title(strcat('s',string(image_num),'.bmp'))
447 end
448
449 %% Feature matching/Classifier for Euclidian Distance...
450
451 % Eucl_distance = norm(eyes(1).featureVector - eyes_test(1).featureVector);
452 % For HOG feature
453 % 12 sample images, so loop 12 times
454 fprintf("Begin HOG feature test-----\n")
455 fail_count = 0;
456 for img = 1:NUM_OF_REG_EYES
457     for img_test = 1:NUM_OF_REG_EYES
458         Eucl_distance = norm(eyes(img).featureVector - eyes_test(img_test).featureVector);
459         if Eucl_distance < 32.3 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
460             Authenticated_HOG(img, img_test) = 1;
461             if img == img_test
462                 fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", img, img_test);
463             else
464                 fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", img, img_test);
465             end
466         else
467             Authenticated_HOG(img, img_test) = 0;
468         end
469     end
470 for img_test = 1:NUM_OF_UNREG_EYES*NUM_OF_UNREG_IMAGES_PER_EYE
471     Eucl_distance = norm(eyes(img).featureVector - eyes_unreg(img_test).featureVector);
472     if Eucl_distance < 32.3 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
473         fail_count = fail_count + 1;
474         fprintf("Match found for eye(%d) with eye_unreg(%d), [FAIL] \n", img, round(img_test
/4));

```



```

475     end
476     end
477 end
478
479 for img = 1:NUM_OF_REG_EYES
480     if sum(Authenticated_HOG(img, :))
481     else
482         fprintf("Match NOT found for eye(%d) \n", img)
483     end
484 end
485
486 fprintf("Number of False Postivies = %d [FAIL]\n", fail_count);
487
488 fprintf("-----\n")
489
490 % Eucl_distance = norm(eyes(1).featureVector - eyes_test(1).featureVector);
491 % For 5x5 Average filter..
492 % 12 sample images, so loop 12 times
493 fprintf("Begin 5x5 avg feature test-----\n")
494 fail_count = 0;
495 for img = 1:NUM_OF_REG_EYES
496     for img_test = 1:NUM_OF_REG_EYES
497         Eucl_distance = norm(im2double(eyes(img).average_featureVector) - im2double(eyes_test(
498             img_test).average_featureVector));
499         if Eucl_distance < 7 %7 for 5x5
500             Authenticated_AVG(img, img_test) = 1;
501             if img == img_test
502                 fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", img, img_test);
503             else
504                 fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", img, img_test);
505             end
506         else
507             Authenticated_AVG(img, img_test) = 0;
508         end
509     end
510     for img_test = 1:NUM_OF_UNREG_EYES*NUM_OF_UNREG_IMAGES_PER_EYE
511         Eucl_distance = norm(im2double(eyes(img).average_featureVector) - im2double(eyes_unreg(
512             img_test).average_featureVector));
513         if Eucl_distance < 7 %7 for 5x5
514             fail_count = fail_count + 1;
515             fprintf("Match found for eye(%d) with eye_unreg(%d), [FAIL] \n", img, round(img_test
516                 /4));
517         end
518     end
519 end
520
521 for img = 1:NUM_OF_REG_EYES
522     if sum(Authenticated_AVG(img, :))
523     else
524         fprintf("Match NOT found for eye(%d) \n", img)
525     end
526 end
527
528 fprintf("Number of False Postivies = %d [FAIL]\n", fail_count);
529
530 fprintf("-----\n")
531
532 %% Reading and processing training images
533
534 NUM_OF_TRAIN_EYES = 12;
535 NUM_OF_TRAIN_IMAGES_PER_EYE = 4;
536
537 % struct to hold grayscale eye images, edge maps, pupil extraction, and
538 % iris extraction
539 eyes_train = struct();
540
541 image_folder = pwd;
542 image_folder =strcat(image_folder, '\Labeled_data');
543 filenames = dir(fullfile(image_folder, '*.bmp'));
544 % total_images = numel(filenames);

```

```

543 total_images = NUM_OF_TRAIN_EYES*NUM_OF_TRAIN_IMAGES_PER_EYE;
544
545 % N sample images, so loop N times
546 for image_num = 1:total_images
547     f = fullfile(image_folder, filenames(image_num).name);
548     %load in images (all are grayscale bitmaps of size 320x240)
549     % from MMU Iris database
550     eyes_train(image_num).orig = im2gray(imread(f));
551
552     %Edge Detection: Canny Edge Detector
553     eyes_train(image_num).edge = edge(eyes_train(image_num).orig, 'canny',0.15,3);
554
555     %plotting canny edge images
556     %     figure(2); subplot(3,4,image_num); imshow(eyes_train(image_num).edge);
557     %     title(strcat('s',string(image_num),'.bmp'))
558
559     %Pupil detection threshold (empirically determined)
560     eyes_train(image_num).pupil = eyes_train(image_num).orig < 50;
561
562     %Circular Hough Transform
563     [centers, radii] = imfindcircles(eyes_train(image_num).pupil,[6 100]);
564
565     % retain the strongest circle (pupil edge)
566     coords = centers(1,:);%center = P-by-2 matrix, P = number of circles, x in 1st column, y in 2
nd column
567     pupil_radius = radii(1);
568
569     % start from lower right of pupil edge
570     x = round(coords(1))+round(pupil_radius);
571     y = round(coords(2))+round(pupil_radius);
572     while 1 % traverse radially outward (rightward/downward)
573         if eyes_train(image_num).edge(y,x) == 1 % iris edge detected
574             % calculate radial distance from pupil to iris edge
575             % calculate radial distance from center of pupil iris edge
576             iris_radius = ((x-coords(1))^2 + (y-coords(2))^2)^0.5;
577             break
578         end
579         x = x+1; y = y+1;
580         if x == 240 || y == 320 % if no iris edge was detected
581             iris_radius = pupil_radius*2; % set iris radius to 2x pupil radius
582             break
583         end
584     end
585
586     %plotting original picture with iris and pupil identified
587     %     figure(3); subplot(9,4,image_num); imshow(eyes_train(image_num).orig);
588     %     title(strcat('s',string(image_num),'.bmp'))
589     %     % draw pupil circle
590     %     viscircles(centers(1,:), pupil_radius, 'Color','b','LineWidth',0.5);
591     %     % draw iris circle
592     %     viscircles(centers(1,:), iris_radius, 'Color','b','LineWidth',0.5);
593
594     % extract iris
595     % return 2-D grid of coordinates for grayscale eye image
596     [xgrid, ygrid] = meshgrid(1:size(eyes_train(image_num).orig,2), 1:size(eyes_train(image_num).
orig,1));
597     % filter out anything outside iris and anything inside pupil
598     % pixel value = 1 for region of interest, 0 for non-ROI
599     mask = ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) <= iris_radius.^2 &...
600     ((xgrid-coords(1)).^2 + (ygrid-coords(2)).^2) >= pupil_radius.^2;
601     % apply mask to original grayscale image to get extracted iris
602     eyes_train(image_num).iris = eyes_train(image_num).orig .* uint8(mask);
603
604     % plotting iris extracted
605     %     figure(4); subplot(3,4,image_num); imshow(eyes_train(image_num).iris);
606     %     title(strcat('s',string(image_num),'.bmp'))
607
608     %Image enhancement
609     %round estimate values for radii and center point
610     rounded_coords = round(coords);
611     round_iris_r = round(iris_radius);

```

```

612 round_pupil_r = round(pupil_radius);
613
614 %Resize image to only include iris
615 eyes_train(image_num).iris_only = eyes_train(image_num).iris(rounded_coords(2)- ...
616     round_iris_r:rounded_coords(2)+ round_iris_r, rounded_coords(1)- ...
617     round_iris_r:rounded_coords(1)+round_iris_r);
618 % figure(5); subplot(9,4,image_num); imshow(eyes_train(image_num).iris_only);
619 % title(strcat('s',string(image_num),'.bmp'))
620
621 %Remove extreme pixels(eyelid, eyelash, reflection on eye)
622 eyes_train(image_num).iris_only = uint8(eyes_train(image_num).iris_only > 50) .* ...
623 uint8(eyes_train(image_num).iris_only < 130) .* eyes_train(image_num).iris_only;
624
625 % figure(6); subplot(9,4,image_num); imshow(eyes_train(image_num).iris_only);
626 % title(strcat('s',string(image_num),'.bmp'))
627
628 %Increase contrast
629 [M,N] = size(eyes_train(image_num).iris_only);
630
631 for x=1:M
632     for y=1:N
633         %ignore thresholded pixels
634         if eyes_train(image_num).iris_only(y,x) == 0
635             continue
636         end
637         %remap [51,149] intensities to [0,255]
638         eyes_train(image_num).iris_only(y,x) = ((double(eyes_train(image_num).iris_only(y,x)
-50)/(130-50))*255;
639     end
640 end
641
642 % figure(7); subplot(9,4,image_num); imshow(eyes_train(image_num).iris_only, []);
643 % title(strcat('s',string(image_num),'.bmp'))
644
645 eyes_train(image_num).iris_only = uint8(eyes_train(image_num).iris_only);
646
647 %Iris normalization
648 eyes_train(image_num).polar = ImToPolar(eyes_train(image_num).iris_only, ...
649     round_pupil_r/round_iris_r, round_iris_r/round_iris_r, 40, 320);
650
651 %cut from 110-240 to get rid of eyelashes.... maybe
652 if REMOVE_CENTER_AND_EDGES
653     eyes_train(image_num).polar = [eyes_train(image_num).polar(5:end-5,1:110) eyes_train(
image_num).polar(5:end-5,240:end-5)];
654 end
655
656 % figure(8); subplot(9,4,image_num); imshow(eyes_train(image_num).polar, []);
657 % title(strcat('s',string(image_num),'.bmp'))
658
659 %Extracting HOG features...
660
661 [eyes_train(image_num).featureVector,hogVisualization] = extractHOGFeatures(uint8(eyes_train(
image_num).polar), 'CellSize', cellSize);
662
663
664 average_kernel = fspecial('average', AVERAGE_FILTER_SIZE);
665 eyes_train(image_num).average_featureVector = imfilter(uint8(eyes_train(image_num).polar),
average_kernel);
666
667 % figure(16); subplot(9,4,image_num); imshow(eyes_train(image_num).average_featureVector);
668 %
669 % figure(9); subplot(9,4,image_num); imshow(uint8(eyes_train(image_num).polar));
670 % hold on;
671 % plot(hogVisualization);
672 % hold off;
673
674
675
676 % eyes_train(image_num).features = detectHarrisFeatures(uint8(eyes_train(image_num).polar));
677 % im_w_detected = insertMarker(uint8(eyes_train(image_num).polar), eyes_train(image_num).
features, 'circle');

```

```

678 %     figure(9); subplot(3,4,image_num); imshow(im_w_detected, []);
679 %     title(strcat('s',string(image_num),'.bmp'))
680
681 end
682
683 % eye_train_but_avg = struct();
684 %
685 % count = 1;
686 % for image_num = 1:NUM_OF_TRAIN_EYES
687 %     eye_train_but_avg(image_num).featureVector = eyes_train(count).featureVector + ...
688 %     eyes_train(count+1).featureVector + eyes_train(count+2).featureVector + eyes_train(count+3).
        featureVector;
689 %     eye_train_but_avg(image_num).featureVector = eye_train_but_avg(image_num).featureVector/4;
690 %     count = count + 1;
691 % end
692 %
693 %% Feature matching/Classifier for Euclidian Distance...
694 %
695 % Eucl_distance = norm(eyes(1).featureVector - eyes_test(1).featureVector);
696 % For HOG feature
697 % 12 sample images, so loop 12 times
698 % fprintf("Begin HOG AVG feature test-----\n")
699 % for img = 1:NUM_OF_REG_EYES
700 %     for img_test = 1:NUM_OF_REG_EYES
701 %         Eucl_distance = norm(eyes(img).featureVector - eye_train_but_avg(img_test).featureVector
        );
702 %         if Eucl_distance < 26 % 2.7 for 8x8 cell, 10.5 for 4x4, 32.3 for 2x2
703 %             Authenticated_HOG_AVG(img, img_test) = 1;
704 %             if img == img_test
705 %                 fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", img, img_test);
706 %             else
707 %                 fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", img, img_test);
708 %             end
709 %         else
710 %             Authenticated_HOG_AVG(img, img_test) = 0;
711 %         end
712 %     end
713 % end
714 %
715 % for img = 1:NUM_OF_REG_EYES
716 %     if sum(Authenticated_HOG_AVG(img, :))
717 %     else
718 %         fprintf("Match NOT found for eye(%d) \n", img)
719 %     end
720 % end
721 % fprintf("-----\n")
722
723 %% Feature matching/Classifier for SVM ECOC...
724 % SVM binary classifier
725 % For HOG feature
726 % 12 sample images, so loop 12 times
727 fprintf("Begin SVM ECOC feature test-----\n")
728 features = [];
729
730 %create Nx1 matrix with each row containing a feature vector for image
731 for i = 1:NUM_OF_REG_EYES*NUM_OF_TRAIN_IMAGES_PER_EYE
732     feature = eyes_train(i).featureVector; %feature used to train
733     features = cat(1,features,feature);
734 end
735
736 %create class labels for each feature vector
737 Z = [];
738 for i = 1:NUM_OF_TRAIN_EYES %create a class array that goes with features
739     Y = ones(NUM_OF_TRAIN_IMAGES_PER_EYE,1) * i;
740     Z = cat(1,Z,Y);
741 end
742
743 t = templateSVM('Standardize',true,'KernelFunction','rbf'); %SVM model for binary classification
744
745

```

```

746 SVMmodels = fitcecoc(features,categorical(Z),'Learners',t, Coding='onevsall'); %ECOC
      multiclassifcation with one v all
747
748 for j = 1:NUM_OF_REG_EYES
749     predicted_class = predict(SVMmodels,eyes_test(j).featureVector);
750     fprintf("Match found for eye(%d) with eye_test(%d)\n", j, char(predicted_class))
751     if j == double(predicted_class)
752         fprintf("Match found for eye(%d) with eye_test(%d), [PASS]\n", j, double(predicted_class))
753     ;
754     else
755         fprintf("Match found for eye(%d) with eye_test(%d), [FAIL]\n", j, double(predicted_class))
756     ;
757     end
758 end
759
760 fprintf("End SVM ECOC feature test-----\n")
761
762 %% Feature matching/Classifier for SVM using RBF kernal...
763 % SVM binary classifier
764 % For HOG feature
765 % 12 sample images, so loop 12 times
766 fprintf("Begin SVM feature test-----\n")
767 features = [];
768
769 %create Nx1 matrix with each row containing a feature vector for image
770 for i = 1:NUM_OF_REG_EYES*NUM_OF_TRAIN_IMAGES_PER_EYE
771     feature = eyes_train(i).featureVector; %feature used to train
772     features = cat(1,features,feature);
773 end
774
775 %create class labels for each feature vector
776 Z = [];
777 for i = 1:NUM_OF_TRAIN_EYES %create a class array that goes with features
778     Y = ones(NUM_OF_TRAIN_IMAGES_PER_EYE,1) * i;
779     Z = cat(1,Z,Y);
780 end
781
782 SVMmodels = cell(NUM_OF_REG_EYES,1); %holds the SVM for registered eyes
783 for j = 1:NUM_OF_REG_EYES
784     class_label = Z == j;
785     SVMmodels{j} = fitcsvm(features,class_label,'ClassNames',[false true],'KernelFunction','rbf');
786 end
787
788 % d = 0.005;
789 [x1Grid,x2Grid] = meshgrid(min(features(:,1)):d:max(features(:,1)),...
790     min(features(:,2)):d:max(features(:,2)));
791 % xGrid = [x1Grid(:),x2Grid(:)];
792 % N = size(xGrid,1);
793 % Scores = zeros(N,NUM_OF_REG_EYES);
794
795 % d = 0.005;
796 [x1Grid,x2Grid] = meshgrid(min(features(:,1)):d:max(features(:,1)),...
797     min(features(:,2)):d:max(features(:,2)));
798 % xGrid = [x1Grid(:),x2Grid(:)];
799 % N = size(xGrid,1);
800 % Scores = zeros(N,NUM_OF_REG_EYES);
801
802 eyes_test(img_test).featureVector;
803
804 for j = 1:NUM_OF_REG_EYES
805     [~,score] = predict(SVMmodels{j},xGrid);
806     Scores(:,j) = score(:,2); % Second column contains positive-class scores
807 end
808
809 [~,maxScore]=max(Scores,[],2);
810 figure
811 gscatter(x1(:),x2(:),maxScore,'cym');
812 hold on;
813 gscatter(features(:,1),features(:,2),features,'rgb','.',30);
814 title('\bf Iris Classification Regions');
815 xlabel('Petal Length (cm)');

```

```

814 ylabel('Petal Width (cm)');
815 axis tight
816 hold off

```

## filter\_image.m

```

1 function filtered_image = filter_image(input_image, filter);
2 [r,c] = size(input_image);
3 [r_f,c_f] = size(filter);
4 r_start_index = 1+(r_f-1)/2;
5 c_start_index = 1+(c_f-1)/2;
6 r_end_index   = (r_f-1)/2;
7 c_end_index   = (c_f-1)/2;
8 filtered_image = zeros (r,c);
9
10 for i=r_start_index:r_end_index
11     for j=c_start_index:c_end_index
12         dummy_mtrx = input_image(i-((r_f-1)/2):i+((r_f-1)/2),j-((c_f-1)/2):j+((c_f-1)/2));
13         filtered_block = double(dummy_mtrx) .* filter;
14         filtered_image(i, j) = sum(filtered_block(:));
15     end
16 end
17
18 filtered_image = uint8(filtered_image);

```

## PolarToIm.m

```

1 function imR = PolarToIm (imP, rMin, rMax, Mr, Nr)
2 % POLARTOIM converts polar image to rectangular image.
3 %
4 % V0.1 16 Dec, 2007 (Created) Prakash Manandhar, pmanandhar@umassd.edu
5 %
6 % This is the inverse of ImToPolar. imP is the polar image with M rows and
7 % N columns of data (double data between 0 and 1). M is the number of
8 % samples along the radius from rMin to rMax (which are between 0 and 1 and
9 % rMax > rMin). Mr and Nr are the number of pixels in the rectangular
10 % domain. The center of the image is assumed to be the origin for the polar
11 % co-ordinates, and half the width of the image corresponds to r = 1.
12 % Bilinear interpolation is performed for points not in the imP image and
13 % points not between rMin and rMax are rendered as zero. The output is a Mr
14 % x Nr grayscale image (with double values between 0.0 and 1.0).
15
16
17 imR = zeros(Mr, Nr);
18 Om = (Mr+1)/2; % co-ordinates of the center of the image
19 On = (Nr+1)/2;
20 sx = (Mr-1)/2; % scale factors
21 sy = (Nr-1)/2;
22
23 [M N] = size(imP);
24
25 delR = (rMax - rMin)/(M-1);
26 delT = 2*pi/N;
27
28 for xi = 1:Mr
29     for yi = 1:Nr
30         x = (xi - Om)/sx;
31         y = (yi - On)/sy;
32         r = sqrt(x*x + y*y);
33         if r >= rMin & r <= rMax
34             t = atan2(y, x);
35             if t < 0
36                 t = t + 2*pi;
37             end
38             imR (xi, yi) = interpolate (imP, r, t, rMin, rMax, M, N, delR, delT);
39         end
40     end
41 end

```

```

42
43 function v = interpolate (imP, r, t, rMin, rMax, M, N, delR, delT)
44     ri = 1 + (r - rMin)/delR;
45     ti = 1 + t/delT;
46     rf = floor(ri);
47     rc = ceil(ri);
48     tf = floor(ti);
49     tc = ceil(ti);
50     if tc > N
51         tc = tf;
52     end
53     if rf == rc & tc == tf
54         v = imP (rc, tc);
55     elseif rf == rc
56         v = imP (rf, tf) + (ti - tf)*(imP (rf, tc) - imP (rf, tf));
57     elseif tf == tc
58         v = imP (rf, tf) + (ri - rf)*(imP (rc, tf) - imP (rf, tf));
59     else
60         A = [ rf tf rf*tf 1
61              rf tc rf*tc 1
62              rc tf rc*tf 1
63              rc tc rc*tc 1 ];
64         z = [ imP(rf, tf)
65              imP(rf, tc)
66              imP(rc, tf)
67              imP(rc, tc) ];
68         a = A\double(z);
69         w = [ri ti ri*ti 1];
70         v = w*a;
71     end

```

## ImToPolar.m

```

1 function imP = ImToPolar (imR, rMin, rMax, M, N)
2 % IMTOPOLAR converts rectangular image to polar form. The output image is
3 % an MxN image with M points along the r axis and N points along the theta
4 % axis. The origin of the image is assumed to be at the center of the given
5 % image. The image is assumed to be grayscale.
6 % Bilinear interpolation is used to interpolate between points not exactly
7 % in the image.
8 %
9 % rMin and rMax should be between 0 and 1 and rMin < rMax. r = 0 is the
10 % center of the image and r = 1 is half the width or height of the image.
11 %
12 % V0.1 7 Dec 2007 (Created), Prakash Manandhar pmanandhar@umassd.edu
13
14 [Mr Nr] = size(imR); % size of rectangular image
15 Om = (Mr+1)/2; % co-ordinates of the center of the image
16 On = (Nr+1)/2;
17 sx = (Mr-1)/2; % scale factors
18 sy = (Nr-1)/2;
19
20 imP = zeros(M, N);
21
22 delR = (rMax - rMin)/(M-1);
23 delT = 2*pi/N;
24
25 % loop in radius and
26 for ri = 1:M
27     for ti = 1:N
28         r = rMin + (ri - 1)*delR;
29         t = (ti - 1)*delT;
30         x = r*cos(t);
31         y = r*sin(t);
32         xR = x*sx + Om;
33         yR = y*sy + On;
34         imP (ri, ti) = interpolate (imR, xR, yR);
35     end
36 end
37

```

```

38 function v = interpolate (imR, xR, yR)
39     xf = floor(xR);
40     xc = ceil(xR);
41     yf = floor(yR);
42     yc = ceil(yR);
43     if xf == xc & yc == yf
44         v = imR (xc, yc);
45     elseif xf == xc
46         v = imR (xf, yf) + (yR - yf)*(imR (xf, yc) - imR (xf, yf));
47     elseif yf == yc
48         v = imR (xf, yf) + (xR - xf)*(imR (xc, yf) - imR (xf, yf));
49     else
50         A = [ xf yf xf*yf 1
51              xf yc xf*yc 1
52              xc yf xc*yf 1
53              xc yc xc*yc 1 ];
54         r = [ imR(xf, yf)
55              imR(xf, yc)
56              imR(xc, yf)
57              imR(xc, yc) ];
58         a = A\double(r);
59         w = [xR yR xR*yR 1];
60         v = w*a;
61     end

```

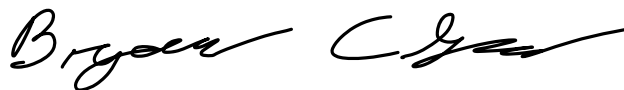


## Appendix B - Analysis of Senior Project Design

**Project Title:** Iris Detection Authenticator

**Students' Names:** Nathan D. Tang and Bryan K. Chau

**Students' Signatures:**



**Advisor's Name:** Jane Zhang

**Advisor's Initials:**

**Date:** 12/07/2022

### Summary of Functional Requirements

The Iris Detection Authenticator is a program utilizing an algorithm that takes a set of eye images, extracts the irises from the images, normalizes the iris images, extracts features from the normalized images, and stores that information. Other eye images can then be tested against the registered data to identify whether they match the registered irises.

### Primary Constraints

The most significant challenge in our project and implementation was obtaining high accuracy in our testing. We used a large data set of 450 eye images, so checking that our iris processing, feature extraction, and feature matching stages worked was a time consuming process. Creating this algorithm to be efficient as well was another difficulty along with its accuracy. Research was another part of our project that proved to be a challenge. There were a number of papers related to our project, but implementing their ideas and techniques took time and effort. We used MATLAB to code our project, and many of the papers did not. Some of the papers and software we gathered were also outdated, so there was some difficulty converting and updating code to become relevant for our work.

### Economic

Not much human capital was needed for the development of this project. Our group consisted of two people, and we were able to complete this project over the course of this year. Financially, there were no extra costs for our project, and the tools we needed were our computers, internet, and MATLAB. As students, we have access to all of these. Since the project is mostly software, the only natural capital is power and energy. The costs for this project is all in the early stages, and benefits will accrue after our algorithm is used in real devices which is out of the scope of our study. For our project, we used an online database of eye images that was free to use. We paid for our own computers, internet, and student fees which pays for our MATLAB student licenses. The development time for our project went as expected. We completed our research, development, testing, and analysis throughout the year.

### If manufactured on a commercial basis

Our project is the program we developed, so that is the only "device" that is manufactured or sold. There are no other costs involved in manufacturing our device. We estimate the profit from licensing our product is \$250 per system, and that results in an estimated \$15,000 a year. The electricity cost for powering this program we estimate is <\$5.

### Environmental

The environmental impacts occur during usage of the program. It requires a small amount of power to run. Since this is software, it has to run on some physical device, so fuel and mineral resources are impacted by manufacturing computers. Metals and energy are needed for the device our project runs on. This also affects the wildlife in the areas where those metals and fuels are mined for.

## **Manufacturability**

Our project itself can be easily manufactured or replicated with our code. There are no issues of manufacturability in regards to the software development

## **Sustainability**

This is very sustainable because our project is heavily software-sided. There are no issues maintaining our software on modern devices. As long as MATLAB has no major updates that affect our code, our product will still be functional for a long time. Since our software runs on a single device, the sustainability impact on resources can be measured by the number of devices our software is on. Upgrading our design requires new research on feature extraction and matching. It is limited by the ability to implement these and any MATLAB software in our code.

## **Ethical**

Our project deals with inherently private data, so there are major ethical risks. A valid concern is that in real applications, our code can be hacked and a person's iris data can be stolen. Encryption and other security measures are needed to mitigate this issue. Furthermore, inaccuracy in our algorithm can lead to false negatives, or worse, false positives.

## **Health and Safety**

Our product hopes to increase hygiene and safety of the user by reducing contact between the user and the device our software is used. Our project inherently reduces contact between the user and the device because it uses images instead of fingerprints as its data. We also hope that our software is secure so data cannot be stolen by any malicious attacks.

## **Social and Political**

Given that this product generally deals with very sensitive data, people may be worried about the data mining that occurs from this device. While it may be true that this device stores data on the user, it is inevitable for the operation of the device. In terms of accessibility, the product will start off at a high price to support the development of the device but will eventually drop in price as cameras get cheaper and the algorithm matures. Everyone will be able to use this device, unless they are physically missing their eyes. It will require no previous knowledge to utilize, making it very accessible.

## **Development**

Throughout the development of this project, we learned many new computer vision techniques. We utilized many of MATLAB's own features as well as implemented new methods such as Hough transform, HOG feature extraction, rubber-sheet normalization, and SVM classification.