Computer Vision – CSYS
Ass#1: Image Enhancement

Contents

Instructions

1. Six problems (**one per member**)
2. Carefully check the **requirements** of each problem and its **evaluation** criteria
3. **Test cases** and **expected outputs** of each problem can be found in the assignment materials
4. **Delivery:** on campus during **Lab times** and/or **TA office hours**
5. **DUE TO: 5th** week [WED 11 MAR] isA

Question#1: Fingers Extraction & Alignment

Description

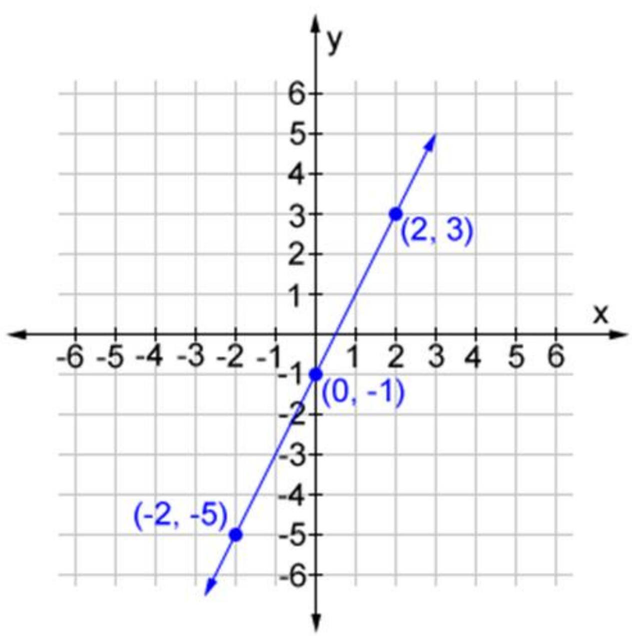
As a preprocessing step for hand-based biometrics (e.g. knuckles, finger vein…), the fingers need to be extracted in separate images. Then, the rotation angle of each finger needs to be calculated and fixed.

Generally, the rotation angle can be calculated via the slope of a line between any two points at the finger side, as follows:

$$Slope = \frac{\Delta Y}{\Delta X} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$Angle = \tan^{-1}(Slope)$$

Following is an illustrated example:



Slope = (3 – (-1)) / (2 – 0) = 4/2 = 2

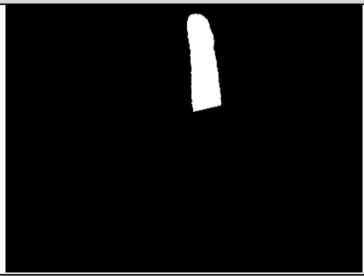
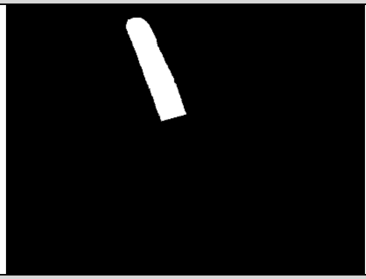
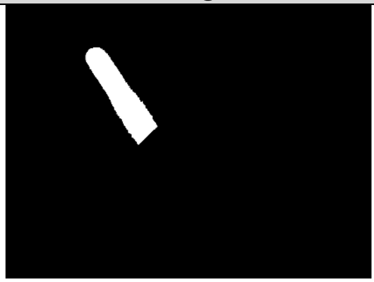
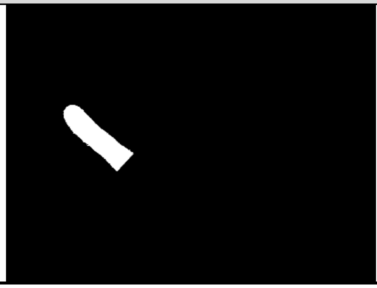
Angle = $\tan^{-1}$ (2) = 63.4°

Thus, given the original hand image and the mask binary image of each finger, it's required to

1. Extract each of the four fingers (all except thumb) in separate images,
2. Calculate its rotation angle,
3. Fix the rotation.

Example

Input:

Original Image	**Masked Images** (scaled down for preview purpose only)	
	**Index**	**Middle**
		
	Ring	**Little**
		

Output:

Index			**Middle**			**Ring**			**Little**		
Extracted Finger	**Angle from X**	**Fixed Rotation**	**Extracted Finger**	**Angle from X**	**Fixed Rotation**	**Extracted Finger**	**Angle from X**	**Fixed Rotation**	**Extracted Finger**	**Angle from X**	**Fixed Rotation**
	96°			112°			123°			138°	

Requirement

1. Write <u>three</u> MATLAB functions with the following signatures:

1) `fingerImg = ExtractFinger(handImg, fingerMsk)`

Where:

- `handImg`: hand original colored image
- `fingerMsk`: mask image of a finger
- `fingerImg`: extracted colored finger image

2) `angle = FindFingerRotAngle(fingerImg)`

Where:

- `fingerImg`: extracted colored finger image
- `angle`: rotation angle from +ve x-axis (in degree)

3) `fixedFingerImg = FixFingerRotation(fingerImg, angle)`

Where:

- `fingerImg`: extracted colored finger image
- `angle`: rotation angle from +ve x-axis (in degree)
- `fixedFingerImg`: colored finger image after fixing its rotation to be vertically aligned

2. Write an `ApplyAll` function that should:

`ApplyAll(inputPaths_File, outputPath)`

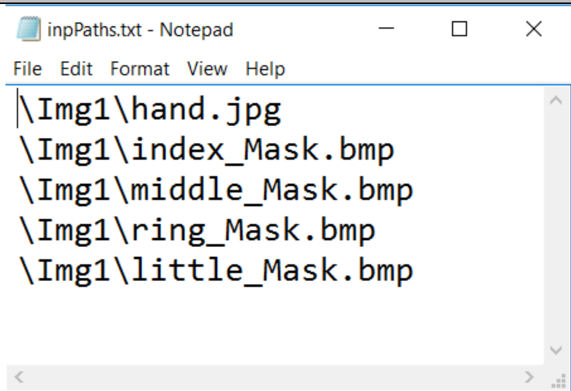
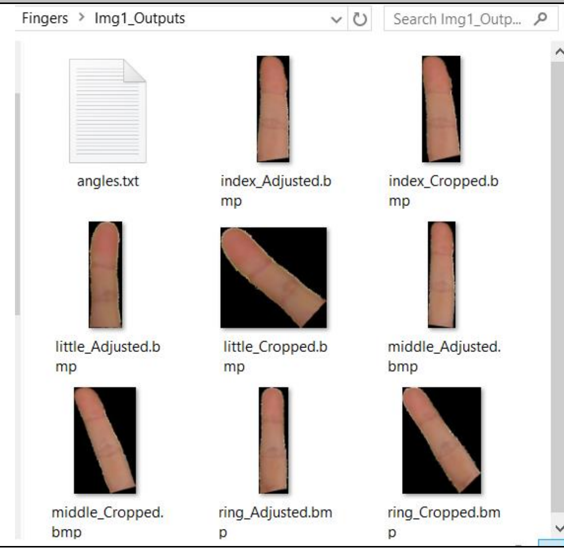
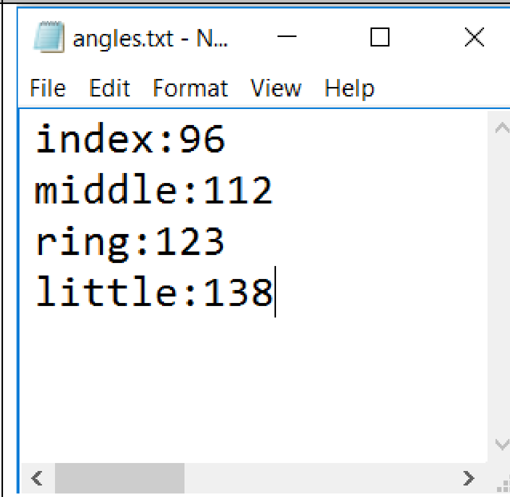
a. Takes a **text file** from `inputPaths_File` containing the path of the five images (hand + 4 masks),

b. Apply the **3 functions** on **each** finger,

c. Save the **three O/Ps** of **each** finger in the destination folder at the given `outputPath`

- Three O/Ps of each finger are: `fingerImg` & `fixedFingerImg` as images + `angle` in text file.

The following figure shows the expected input and output from this function:

Input (text file)	Output Folder	Output 'angles.txt' file
		

Helper

- Dealing with text files: `fopen, fprintf, fscanf, fclose, num2str…`
- Find coordinates that satisfy a specific condition:

```
>> A = [1 5 7; 3 9 2; 7 7 7; 1 1 1]
A =   1     5     7
      3     9     2
      7     7     7
      1     1     1
>> [x y] = find(A == 7)
x =   3     y =  1
      3          2
      1          3
      3          3
```

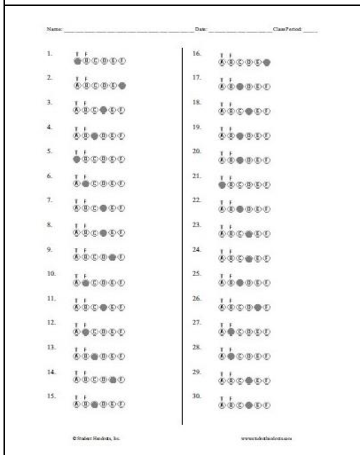
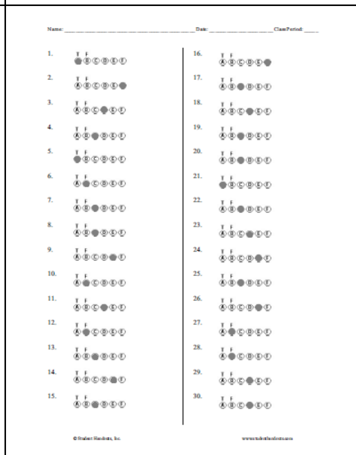
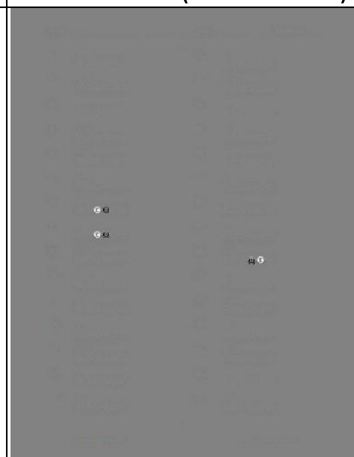
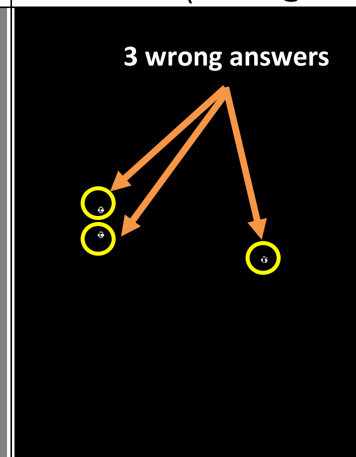
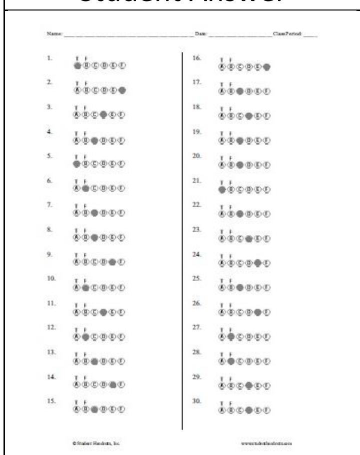
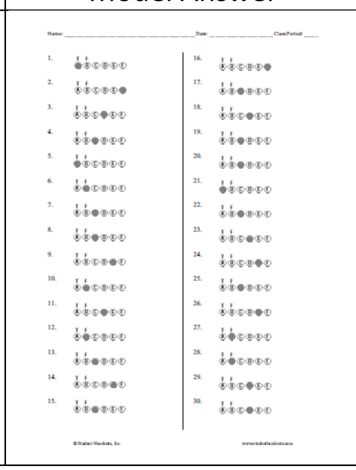
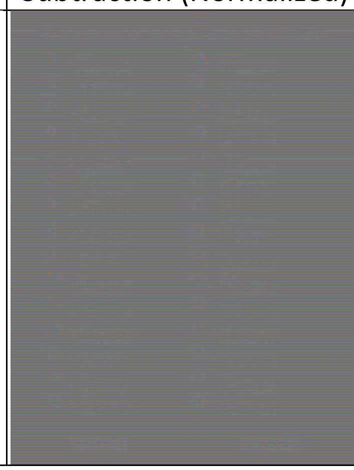
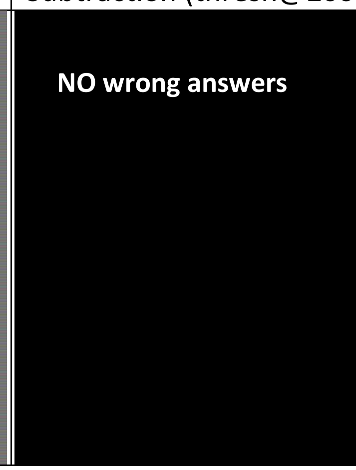
Evaluation

1. Three MATLAB functions that works correctly 75% [Seen Test: 45%, Unseen Test: 30%]
2. `ApplyAll` function 25% [Seen Test: 15%, Unseen Test: 10%]

Question#2: Automatic Correction of Bubble Sheets

Description

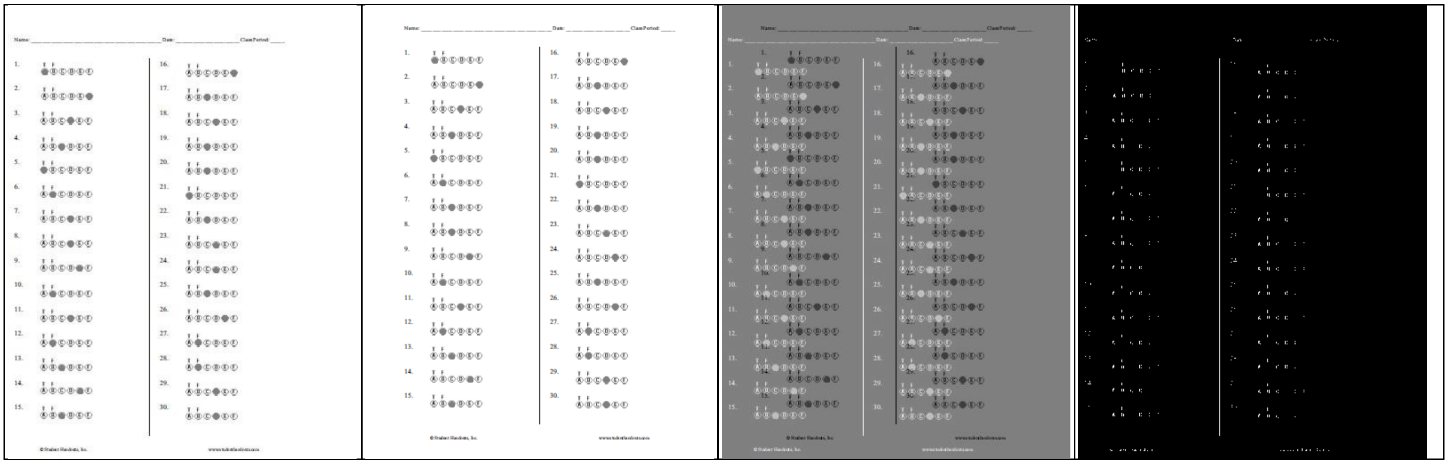
In a typical bubble-sheet MCQs exam, each answer sheet is compared with the model answer sheet for automatic correction. This can simply done by **subtracting** the two sheets to detect the wrong answers (if any),

Student Answer	Model Answer	Subtraction (Normalized)	Subtraction (thresh@200)
			
Student Answer	Model Answer	Subtraction (Normalized)	Subtraction (thresh@200)
			

Sounds good ☺…

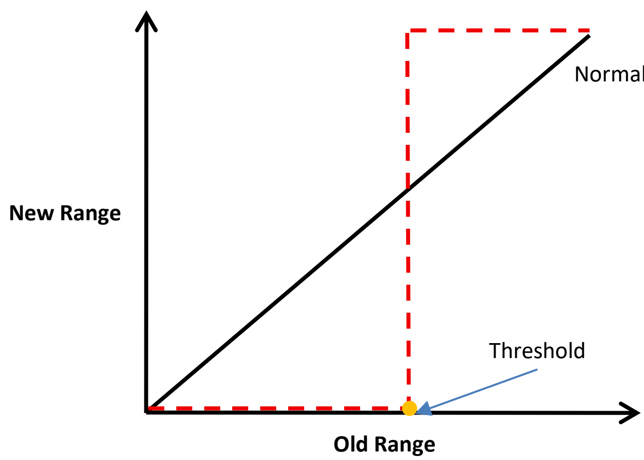
However, in order to work correctly, the two sheets should be PERFECTLY ALIGNED. The model-answer sheet is aligned for sure. On the other hand, the student answer sheet can be TRANSLATED during its acquisition, as shown in the following FIGURE.

Student Answer	Model Answer	Subtraction (Normalized)	Subtraction (thresh@200)

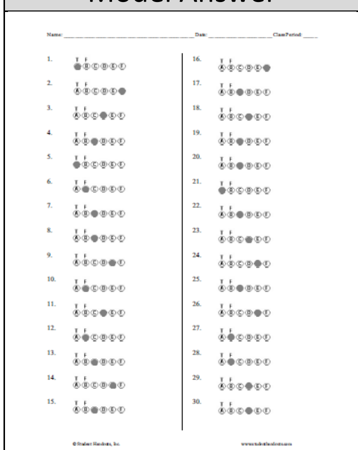
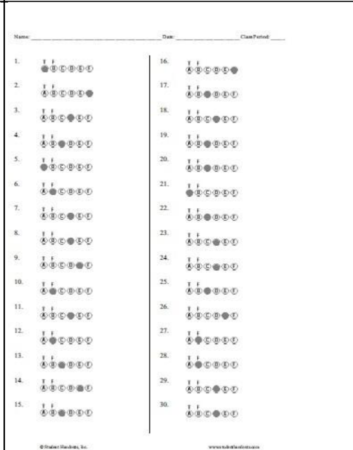
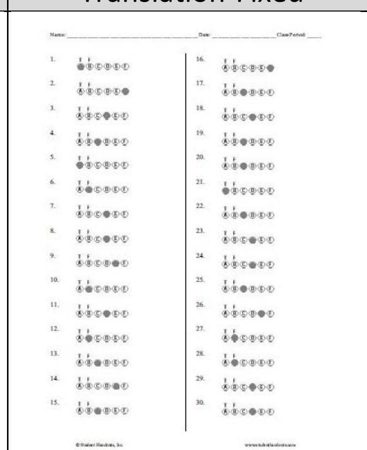
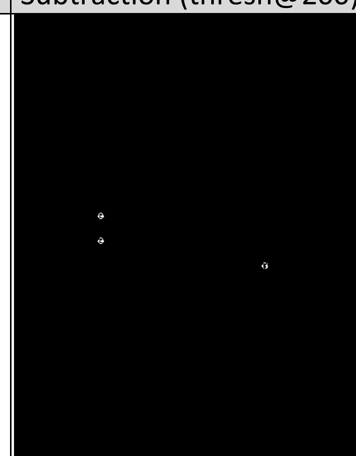


So, given the two sheets (model answer and student answer), it's **<u>REQUIRED</u> to**

1. **Detect and fix the translation** of the student answer sheet.
2. **Highlight wrong answers** (if any) by thresholding the resulting image with the following intensity transformation graph:



Example

INPUT		OUTPUT	
Model Answer	Student Answer	Translation-Fixed	Subtraction (thresh@200)
		 Tx = 37, Ty = -13	

Requirement

Implement <u>two</u> MATLAB functions with the following signatures:

1) `[FixedSheet, xTrans, yTrans]=` **`FixTrans`**`(ModelSheet, TransSheet)`

Where:

- `FixedSheet`: is the `TransSheet` after fixing its translation.
- `xTrans`: detected translation value in x direction.
- `yTrans`: detected translation value in y direction.

2) `ResultImage =` **`Correct`**`(ModelSheet, FixedSheet)`

Where:

- o `ResultImage`: binary image with wrong answers (if any) are shown in white

Helper

- `maketform, imtransform`

Evaluation

1. Find and fix the translation correctly 60% [Seen Test: 40%, Unseen Test: 20%]
2. Extract the result image correctly 40% [Seen Test: 20%, Unseen Test: 20%]

Question#3: Illumination Normalization via Multiscale Retinex gain/offset

Description

refer to the attached presentation for more details

First: Single Scale Retinex (SSR)

The Single-Scale Retinex is given by

$$Ri(x, y) = log\, Ii\,(x, y) - log[\,F(x, y) * Ii(x, y)]$$

Where $I_i(x,y)$ is image distribution in the ith color band, F(x,y) is the Gaussian filter. It's known that image distribution is the product of scenes **reflectance** and **illumination**.

$$I_i(x, y) = S_i(x, y) r_i(x, y)$$

Where Si(x,y) is the spatial distribution of illumination and ri(x,y), the distribution of scene reflectance. The convolution with Gaussian filter works as averaging in the neighborhood. So that:

$$R_i(x, y) = \log \frac{S_i(x, y) r_i(x, y)}{\overline{S_i(x, y)}\,\overline{r_i(x, y)}}$$

Generally, the illumination has slow spatial variation, which means:

$$S_i(x, y) \approx \overline{S_i(x, y)}$$

Then it will be apparent that color constancy (i.e., independence from source illumination spectral and spatial variation) is achieved.

$$R_i(x, y) \approx \log \frac{r_i(x, y)}{\overline{r_i(x, y)}}$$

Refer to next figure for example

Original Image	SSR with sigma = 15
	

Second: Multi Scale Retinex (MSR)

Because of the tradeoff between dynamic range compression and color rendition in Single Scale Retinex SSR (as shown in the figure), Multiscale Retinex MSR is a solution. It's a combination of **weighted** different **scales** of SSR and does not sacrifice either dynamic range compression or color rendition.

<table>
<tr><th>Original</th><th>SSR sigma = 6</th><th>SSR sigma = 50</th><th>SSR sigma = 100</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
</table>

$$MSR = \sum_{s=1}^{N} \omega_s \times SSR_s \qquad [1]$$

Where N is the number of the scales, SSR_s is the Single Scale Retinex at the s^{th} scale and ω_s is the contribution weight of each SSR_i in the final image. The obvious question about MSR is the number of scales needed, scale values, and weight values. Experiments showed that three scales are enough for most of the images, and the weights can be equal. Generally fixed **scales of 15,80 and 250** can be used. The following figure shows an example of MSR at three scales:

<table>
<tr><th>Input</th><th>Output</th></tr>
<tr>
<td>
</td>
<td>
</td>
</tr>
<tr><td></td><td>Params: scales = [15, 80, 250]</td></tr>
</table>

MSR is good for gray images. But it could be a problem for the color images because it does not consider the relative intensity of color bands, as shown in next figure. Considering the images "out of gray world", whose average intensity for three color band are far from equal, the output of MSR for three channels will be closer, which make it looks more gray.

Input	Output (MSR)
	
	Params: *scales = [15, 80, 250]*

Third: Multi Scale Retinex with Color Resotration (MSRCR)

The solution to this problem is to introduce **weights for three color** channels **depending** on the **relative intensity** of the three channels in the original images:

$$C_i(x,y) = f\left[I_i^{'}(x,y)\right]$$

Where $I_i^{'}(x,y)$ the relative intensity of three channels, as follows:

$$I_i^{'}(x,y) = \frac{I_i(x,y)}{\sum_{i=1}^{3} I_i(x,y)}$$

The color restoration function f can be defined as follows:

$$\boldsymbol{C_i(x,y) = \beta\, log[\,\alpha I_i^{'}(x,y)]} \qquad \textbf{[2]}$$

Where $\boldsymbol{\alpha}$ & $\boldsymbol{\beta}$ are user-defined params between 0 and 255, **usually set to 125 and 46**, respectively.

The MSR with Color Restoration (MSRCR) can be described as:

$$MSRCR_i(x,y) = C_i(x,y)MSR_i(x,y)$$

Generally, the output after MSRCR processing will be **out of display domain**. It needs to be shifted and compressed to the display domain. Auto **gain/offset** method is usually used, where the gain and offset are based on the specified image statistics. But research showed that the histograms MSR of different images got typical shape and position, which means the gain and offset parameters could be set once and run forever. Following the equation of MSRCR with gain/offset:

$$\boldsymbol{MSRCR_i(x,y) = G[C_i(x,y)MSR_i(x,y) + b]} \qquad \textbf{[3]}$$

Where ***G*** and ***b*** are the gain and offset, respectively, that are used to bring the range to the display domain. Their values are usually determined by experiments. A value of **0.354** is common for the gain ***G*** and of **352** for the offset ***b***.

Because the pixel number with values at high and low regions are very small, **clipping (i.e. cutoff)** the values that exceed the range [0, 255] can improve the contrast of the images and does not sacrifice information.

Example

Params:

- scales = [15, 80, 250]
- $\boldsymbol{\alpha} = 125$ & $\boldsymbol{\beta} = 46$
- $\boldsymbol{G} = 0.354$ & $\boldsymbol{b} = 352$

Input	Output (MSRCR with gain/offset)
	

Requirement

1. Write two MATLAB functions to implement the following:

1) MSR (equ. 1), with the following signature:

```
Res = MSR(Img, Scales, Weights)
```

Where:

- `Img, Res`: are the original image and the enhanced result image, respectively
- `Scales, Weights`: array of Gaussian scales for MSR and their corresponding weights (equ. 1)

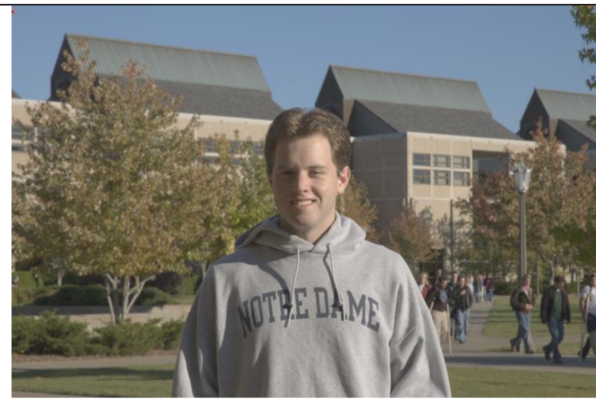
2) MSRCR with gain/offset (equ. 3), with the following signature:

```
Res = MSRCR_go(Img, Scales, Weights, Alpha, Beta, Gain, Offset)
```

Where:

- `Img, Res`: are the original image and the enhanced result image, respectively
- `Scales, Weights`: array of Gaussian scales for MSR and their corresponding weights (equ. 1)
- `Alpha, Beta`: to be used for color restoration (equ. 2)
- `Gain, Offset`: to be used to bring the range to the display domain (equ. 3)

2. Play with the param values of **`MSRCR_go`** to find the best combination for reducing the illumination effects in the following (attached) face images:



Helper

- `fspecial, imfilter`

Evaluation

1. MATLAB function that works correctly
    a. MSR only: 30% [Seen Test: 15%, Unseen Test: 15%]
    b. MSRCR with gain/offset 50% [Seen Test: 25%, Unseen Test: 25%]
2. Find best params for the three images 20% (10% per each)

FINGERPRINT: Problem Description

In fingerprint recognition, one of the most common approaches is the minutiae-based approach. In this approach, three common phases are applied:

1. Preprocessing
2. Feature extraction
3. Matching

The next three questions address the first two steps (preprocessing & feature extraction)

Question#4: Fingerprint Preprocessing

First: Alignment

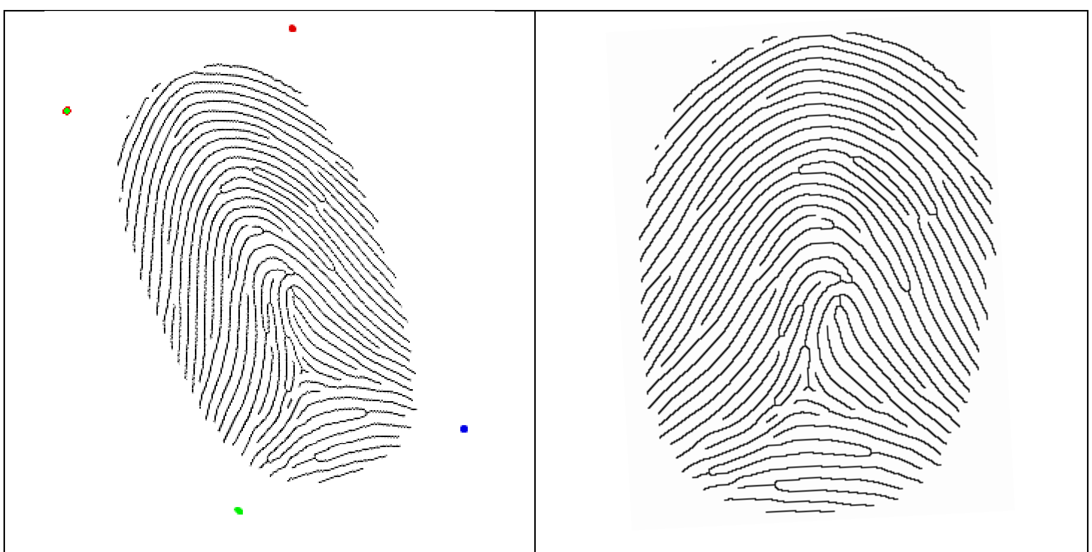
Given an input fingerprint image with four markers on its corners, design and implement and algorithm to:

1. Automatically detect the location of the four markers.
2. Use these markers to correctly align (warp) the image to a given size.

The markers colors are set as follows, as shown in the next Figure (a):

1. Top left: green area surrounded by red.
2. Top right: red.
3. Bottom left: green.
4. Bottom right: blue.

> **NOTES:**
> 1. Black and white pixels are not necessarily having value of 0 and 255, respectively.
> 2. The color of the mark doesn't necessarily require its component to be maximum (i.e. red mark not necessarily means that its red is 255).



a. Fingerprint with four colored markers. **b.** Expected output after alignment (warping).

Second: Enhancement

In a pixel-wise image processing operation the new value of each pixel only depends on its previous value and some global parameters. The normalization approach used by Hong, Wan, and Jain (1998) determines the new intensity value of each pixel in an image as:

$$\mathbf{I}'[x,y] = \begin{cases} m_0 + \sqrt{(\mathbf{I}[x,y]-m)^2 \cdot v_0 / v} & \text{if } \mathbf{I}[x,y] > m \\ m_0 - \sqrt{(\mathbf{I}[x,y]-m)^2 \cdot v_0 / v} & \text{otherwise,} \end{cases}$$

where m and v are the image mean and variance, m_0 and v_0 are the desired mean and variance after the normalization.

Example

Params: m_0 = 100 and v_0 = 100

Input	Output
	

Requirements

1. Write a MATLAB function with the following signature:

```
[AlignedImage, Corners] = Align(InputImage, DW, DH)
```

- `DW`: desired (expected) width of the fingerprint after aligning.
- `DH`: desired (expected) height of the fingerprint after aligning.
- `Corners`: a 2×4 matrix containing the (x, y) locations of the four corners

2. Write a MATLAB function with the following signature:

```
Res = EnhanceFP(Img, m0, v0)
```

Where:

- `Img, Res`: are the original image and the enhanced result image, respectively
- `m0`: desired mean
- `v0`: desired variance

Helper

- **Alignment:** `maketform, imtransform`
- **Enhancement:** `mean2, std2`

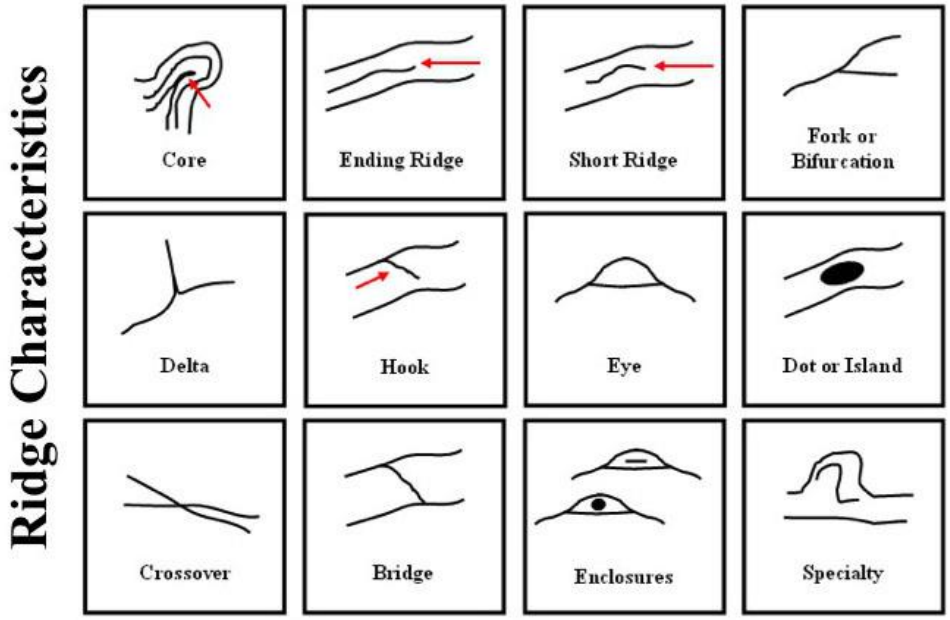
Evaluation

1. Alignment function that works correctly 70% [Seen Test: 30%, Unseen Test: 40%]
2. Enhancement function that works correctly 30%

Question#5: Fingerprint Features #1

Description

There are many types of the minutiae (landmark points), as shown in the following figure:



Different types of landmark points

Among these types, we are interested in the **<u>following</u>** landmarks:

1. **Endpoint:** in which a fingerprint point is surrounded by one point only, as shown in Figure 1.
2. **Short ridge**: a short line that starts and ends by two endpoints, as shown in Figure 2.

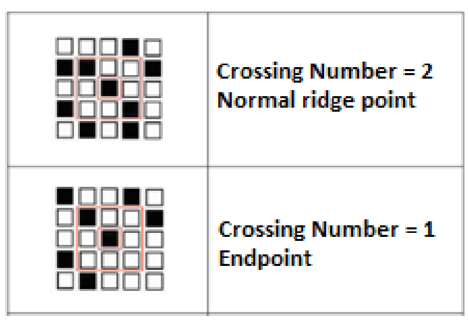


Figure 1: Number of surrounding points for normal point and endpoint.

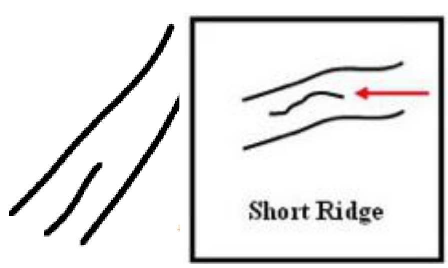
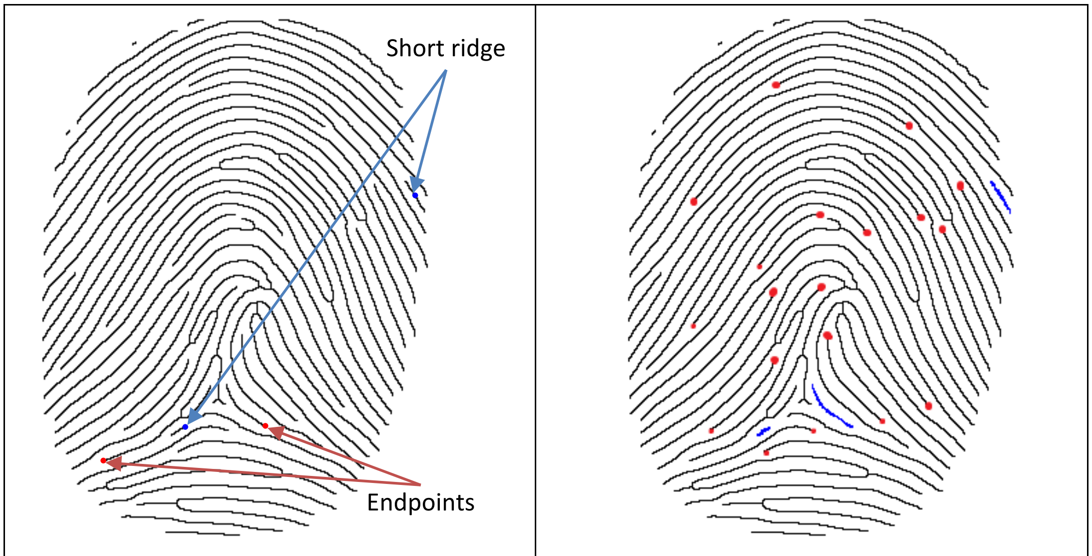


Figure 2: Example of end point & short ridge landmark.

Given a preprocessed and thinned fingerprint image (i.e. lines are one pixel thick), as shown in Figure 3 (a), design and implement an algorithm to extract the locations (x, y) of the **endpoints and short ridges** and mark them, as shown in Figure 3 (b).



a. Preprocessed fingerprint	**b.** Extracted minutiae (endpoints in red and short ridges in blue)

Figure 3: (a) input preprocessed fingerprint, (b) expected output

Requirements

Write a MATLAB function with the following signature:

```
[Endpoints ShortRidges] = ExtractLandmarks(InputImage, RidgeLen)
```

- `RidgeLen`: the maximum length (number of pixels) of a short ridge.
- `Endpoints`: 2×N matrix containing the (x, y) coordinates of the endpoints.
- `ShortRidges`: 4×M matrix containing the (x1, y1) and (x2, y2) coordinates of the two endpoints of each short ridge.

Helper

- `hold on/off, plot`

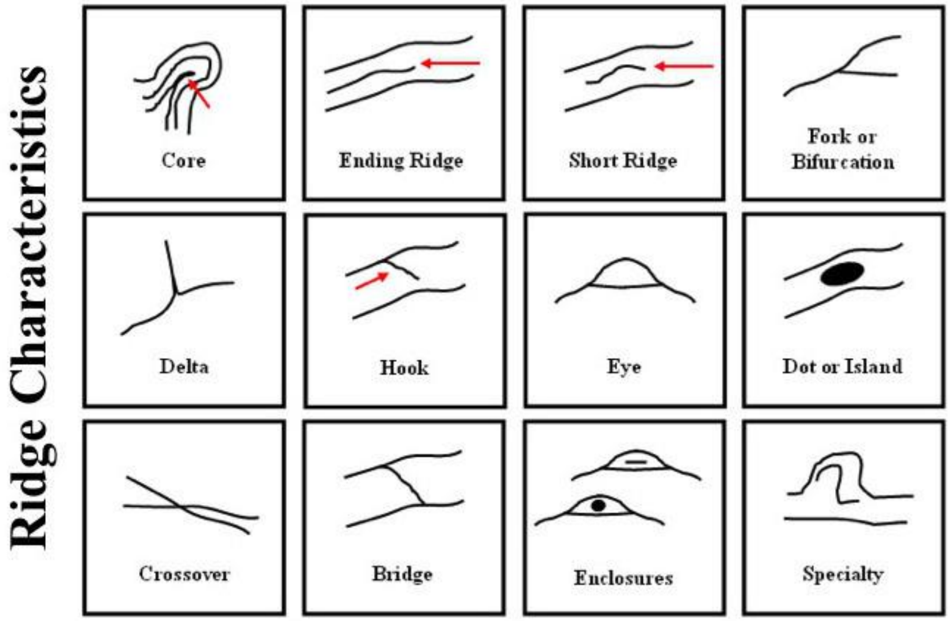
Evaluation

1. Correctly extract the endpoints. 50% [Seen Test: 30%, Unseen Test: 20%]
2. Correctly extract the short ridges 50% [Seen Test: 30%, Unseen Test: 20%]

Question#6: Fingerprint Features #2

Description

There are many types of the minutiae (landmark points), as shown in the following figure:



Different types of landmark points

Among these types, we are interested in the **following** landmarks:

1. **Bifurcation point:** in which a fingerprint point is surrounded by three other points, as shown in **Figure 4**
2. **Eye:** two short enclosed lines that start and end by two bifurcation, as shown in **Figure 5**.

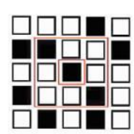
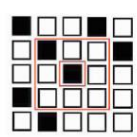
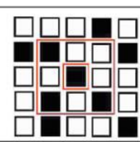
	Crossing Number =2. Normal ridge pixel.
	Crossing Number =1. Termination point.
	Crossing Number =3. Bifurcation point.

Figure 4: Number of surrounding points for normal point, endpoint and bifurcation point.

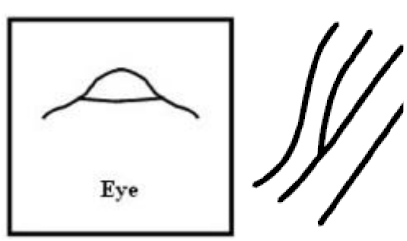
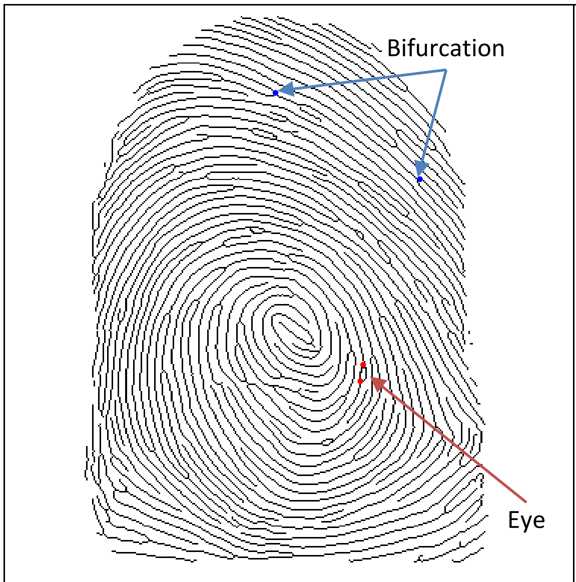
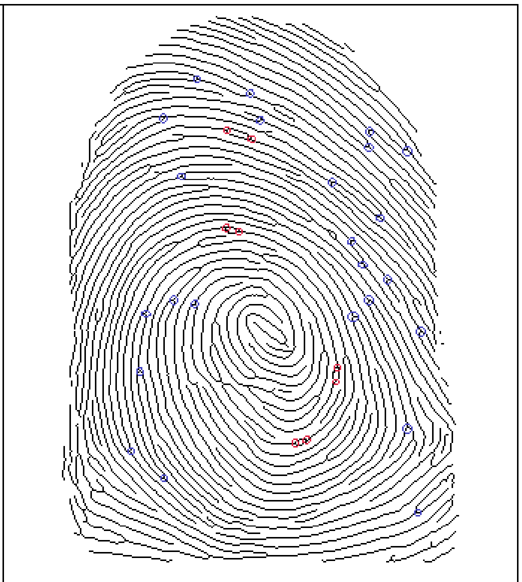


Figure 5: Example of eye and bifurcation landmarks.

Given a preprocessed and thinned fingerprint image (i.e. lines are one pixel thick), as shown in the next figure (a), design and implement an algorithm to extract the locations (x, y) of the **bifurcations and eyes** and mark them, as shown in (b).



a. Preprocessed fingerprint



b. Extracted minutiae (eyes in red and bifurcations in blue)

Figure 6: (a) input preprocessed fingerprint, (b) expected output

Requirements

Write a MATLAB function with the following signature:

```
[Bifurcations Eyes] = ExtractLandmarks(InputImage)
```

- `Bifurcations`: 2×N matrix containing the (x, y) coordinates of each bifurcation.
- `Eyes`: 4×M matrix containing the (x1, y1) and (x2, y2) coordinates of the two endpoints of each eye landmark.

Helper

- `hold on/off, plot`

Evaluation

1. Correctly extract the bifurcations 50% [Seen Test: 30%, Unseen Test: 20%]
2. Correctly extract the eyes 50% [Seen Test: 30%, Unseen Test: 20%]

Wishing you all the best isA 😊