ELEC4630
Assignment 2

Kuang Sheng

45752720

Q1. MRI Image Segmentation

1. Introduction

In the field of digital image processing, only certain objects are interested which usually correspond to regions with specific properties in the image. Image Segmentation is implemented to segment an image into different regions with specific properties and extract these regions for further feature extraction.

In the application of MRI Image sequence of a beating heart, two algorithms have been developed to determine the cross-sectional area of the heart over time, thus the efficacy of the beating heart could be determined by a professional cardiologist as important diagnostic information. The provided video contains 183 image frames in total; thus a general algorithm should be developed to process all the frames without modification or fine tuning.

2. Methodology

There are various boundary segmentation algorithms available such as thresholding, Viterbi, Snake, Morphology and Region Growth etc. The MRI images involve weak edges and regions of low contrast with surroundings.

Before designing appropriate algorithms, the difficulty of this problem should be sorted out. In this section, thresholding based on bimodal grayscale histogram first be evaluated, though this method failed, it helped to select appropriate algorithms and methods. Then morphology and snake algorithms were implemented and evaluated to locate the inner wall.

2.1 Thresholding based on Bimodal Grayscale Histogram (Challenges in processing)

The performance of thresholding approach based on bimodal histogram has been evaluated to figure out the trickiest point. By determining the best setting for the thresholds, the inner wall of the left ventricle could probably be segmented out.

Initially, the MRI images were resized to extract more features by cropping the images and leaving the left ventricle which was interested most.

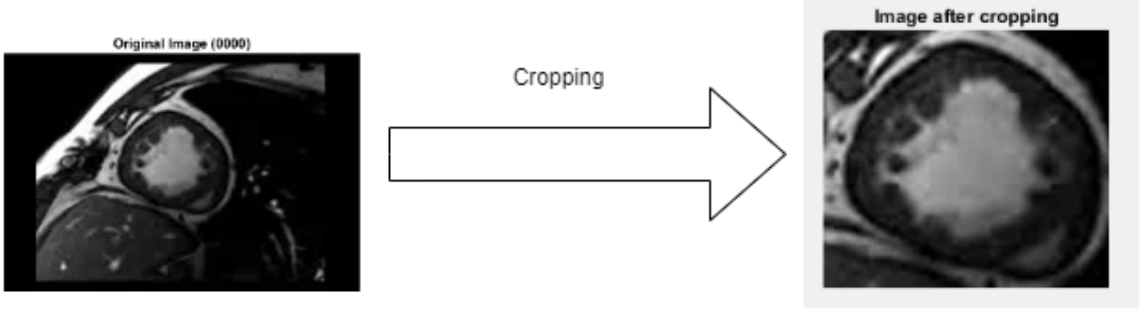


Figure 2.1: Cropping the MRI Image to preserve the left ventricle.

```
%% Threshholding based on Bimodal Histogram
clear, clc, close all;
mriImage = rgb2gray(imread('Cardiac Images/Cardiac MRI Left ventricle0000.jpg'));
x = 170;
y = 85;
width = 200;
height = 180;
Image = imcrop(mriImage, [x y width height]);
figure, imshow(mriImage), title('Original Image (0000)');
figure, imshow(Image), title('Image after cropping');
```

The cropping method would be implemented for the later algorithms as all the information needed is preserved.

The gray scale histogram could be produced as following:

```
figure, imhist(Image), title('Orignal grayscale histogram');
xlabel('Pixel Intensity');
ylabel('Piexel Count');
```

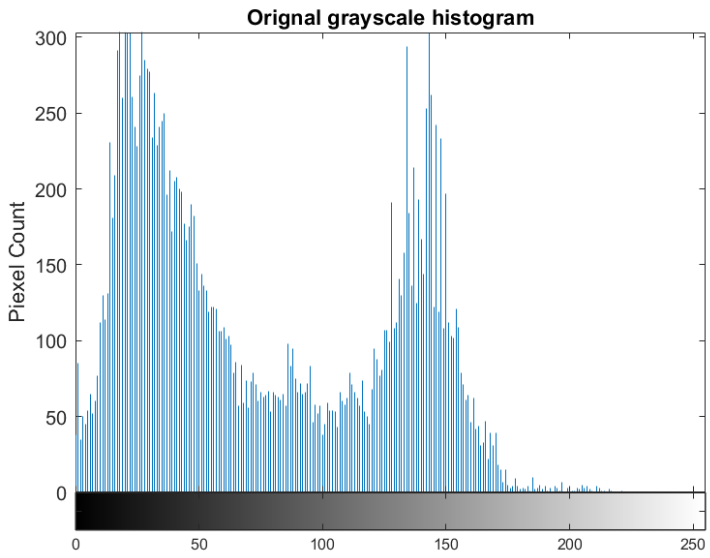


Figure 2.2: Original grayscale histogram

Clearly it is a bimodal histogram, indicating that the image could roughly be divided into two parts, the grayscale is the value corresponding to the vicinity of the two peaks of the grayscale distribution respectively. Selecting the threshold value as the gray value corresponding to the trough point between the two peaks to split the image into two parts, which are ideally the foreground (left ventricle) and background respectively.

To smooth the histogram, an average frequency value of three adjacent gray values was calculated and iterated until the histogram becomes bimodal.

```
hist1 = imhist(Image);
hist2 = hist1;  %Make a copy of histogram
iter = 0;   %Number of iterations

while 1
    %Determine if it is bimodal histogram. If so, find the peak
    [is,peak] = Bimodal(hist1);
    if is == 0                      %If not, smooth the histogram
        hist2(1) = (hist1(1)*2+hist1(2))/3;
        for j = 2:255
            %Find the average point of three adjacent points
            hist2(j) = (hist1(j-1)+hist1(j)+hist1(j+1))/3;
        end
        hist2(256) = (hist1(255)+hist1(256)*2)/3;
        hist1 = hist2;
        iter = iter+1;
        if iter > 500
            break;
        end
    else
        break;
    end
end
```

After smoothing and pointing out the best threshold value,

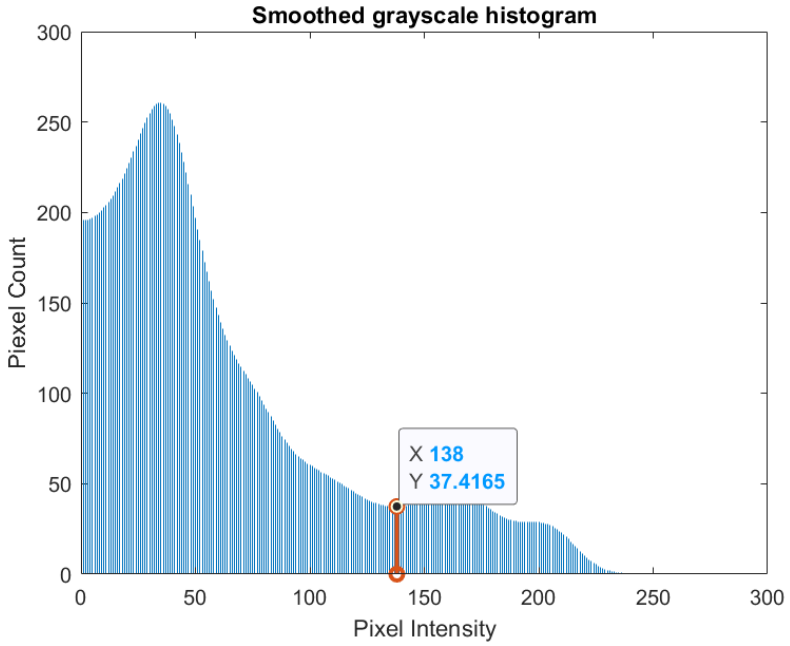


Figure 2.3: Smoothed grayscale histogram

A threshold value of 138 was selected in the first MRI image based on the Bimodal histogram. The image could be binarized based on this threshold value. That is, pixel intensity better than 138 would be set to 1 while pixel intensity lower than 138 would be set to 0.

```
[trough, pos] = min(hist1(peak(1):peak(2))); %Find the trough of histogram
thresh = pos + peak(1);  %Find the threshold

figure, stem(1:256,hist1,'Marker','none');
title('Smoothed grayscale histogram');
xlabel('Pixel Intensity');
ylabel('Piexel Count');
hold on
stem([thresh,thresh],[0,trough],'Linewidth',2);
hold off
result = zeros(size(Image));
result(Image>thresh) = 1;  %Binarize based on Threshold
figure, imshow(result), title('Image after thresholding');
```

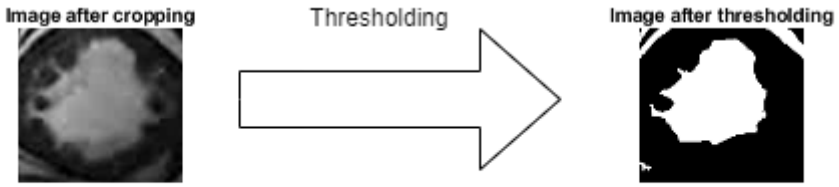


Figure 2.4: Image0000

However, the performance of this method in MRI application is not ideal as some data was lost. In fact, the method performed much worse in later images. Take image 156 as an example:

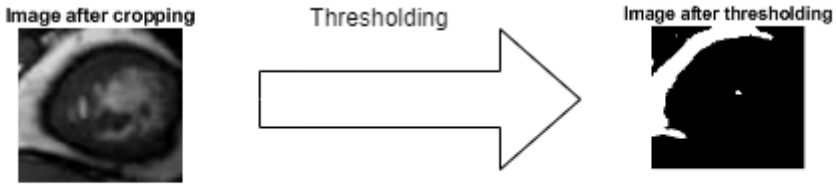


Figure 2.5: Image0156

Since the number of pixels in the foreground is much smaller than the number of pixels in the background, the grayscale histogram becomes unreliable. This method failed in testing because the size of the left ventricle was reducing with the heart beating, meaning that the noise was increasing resulting in a lower SNR (Signal to Noise Ratio). The gray scale value of the components located outside the outer wall of the ventricle is even higher than the components located inside the inner wall. Consequently, the desired inner wall components were filtered out using this thresholding method. Therefore, other methods and algorithms could be considered to solve the puzzle.

2.2 Morphology Approach

Mathematical morphology describes the target image and interests region based on set theory, lattice theory and topology. A morphological filter called structural elements is employed to do morphology operations including erosion, dilation, opening and closing. To segment the inner wall of the left ventricle, morphological opening operation has been performed.

Opening operation is defined as:

$$X \circ S = (X \ominus S) \oplus S$$

In other words, it erodes an image and then dilates the eroded image using the same structural element for both operations. It is useful for removing small objects and thin lines from an image while preserving the shape and size of larger objects in the image.

The approach has the following steps after cropping described at Figure 2.1.

2.2.1 Pre-processing

After cropping the image into appropriate size, the image should be pre-processed to reduce noise for better feature extraction. The contrast of the image could be enhanced using histogram equalization. By spreading out the most frequent intensity pixel values, the grayscale histogram of the image is approximately flat. Moreover, the intensity values of the image could be adjusted by mapping original values to new values. Since the contrast has been enhanced further, the feature of the inner wall of the left ventricle became clearer. Take Image0000 as an example,

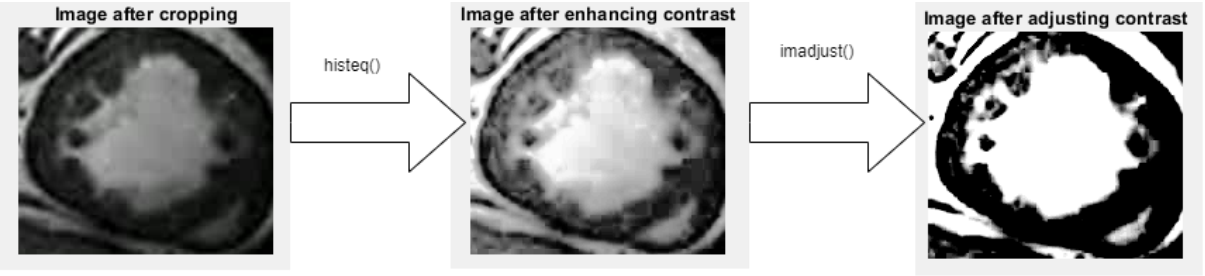


Figure 2.2.1: Image0000

Then a median filter was applied in which each output sample was computed as the median value of the input samples to remove noise from the image and improve the effects of later processing. Apparently the small 'salt & pepper' noise has been filtered out. Therefore, most interested part of the image has been preserved.

Figure 2.2.2: Image0000

```
29
30 -    %% Pre-process data
31 -    I_gray = rgb2gray(histeq(hImage));
32 -    I_gray = imadjust(I_gray,[0.4 0.6],[0 1]);
    I_filt = medfilt2(I_gray, [6 6]);
```

2.2.2 Morphological Operation

A structural element with a disk shape has been created for morphological opening operations. After filling up the holes and removing small objects, the largest connected area was supposed to be the desired area of the inner wall. Thus, we could focus on the largest connected region only.

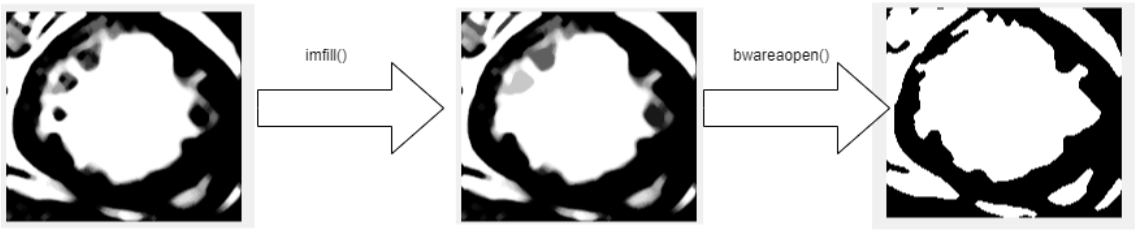


Figure 2.2.3: Image0000

While testing, the inner wall wasn't always the largest connected area as the left ventricle trends to shrink in the later frames and the presence of cardiac muscle, consequently, take Image 0156 as an example,

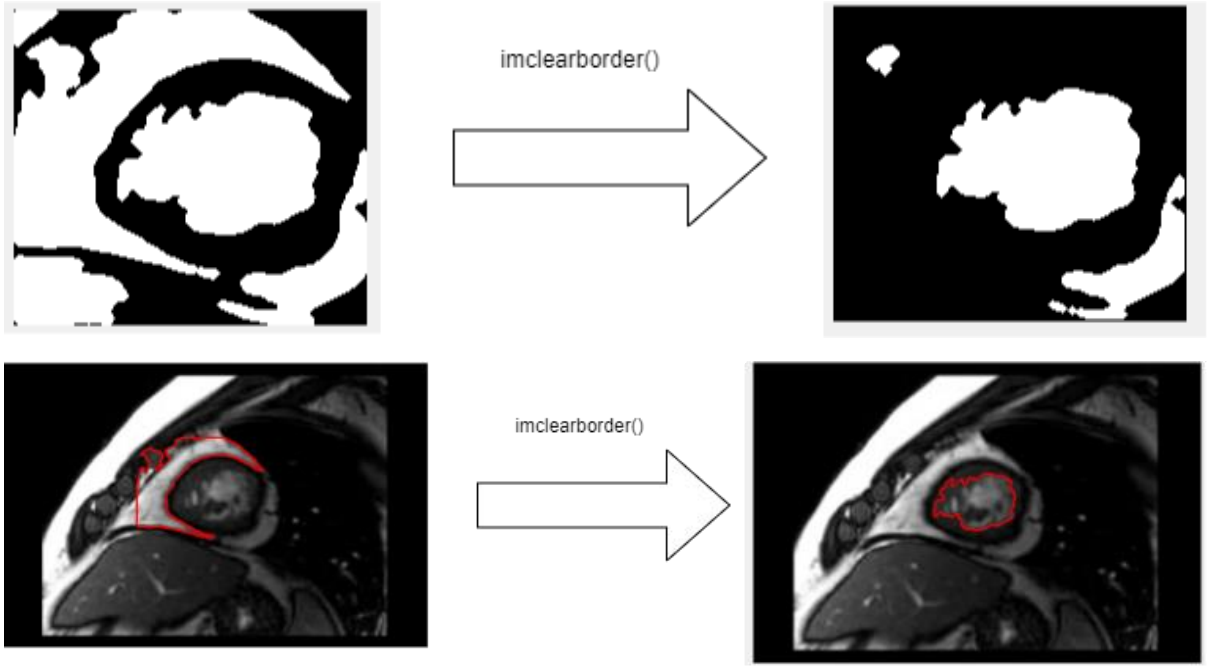


Figure 2.2.4: Image0156

By suppressing light structures connected to image border, the left ventricle should become the largest connected area after processing for all the frames.

```
%% Morphological Operation
SE = strel('disk', 3);
%SE = strel('line', 5,90);

J = imopen(I_filt, SE);
J = imfill(J,'holes');
J = imclearborder(J);
J = bwareaopen(J,100);
```

2.2.3 Determining the internal area (Morphology).

Since the largest connected region was assumed to be the left ventricle after border clearing, the area could be measured from the properties of the image regions after labelling connected components.

Then the exterior boundaries of the inner wall could be traced without searching for the holes. The properties of the image such as connectivity, image size and pixel idx list could be found and counted using the function bwconncomp(), where the biggest pixel idx value would be the internal area in term of pixels of the inner wall. To outline the boundary of the inner wall, the exterior boundaries have been looped through and drawing a tinny line at each boundary coordinate.

```
%% Label and extract info
label = bwlabel(J);
stats = regionprops(label, 'Area','PixelIdxList');

% Locate the largest connected region (left ventricle)
[~, idx] = max([stats.Area]);
bw = ismember(label, idx);

% Find the boundries
boundaries = bwboundaries(bw,'noholes');
```

```
%% Number of Pixels
CC = bwconncomp(bw);
num_pixels = numel(CC.PixelIdxList{1});
area = [area num_pixels];
text = ['Frame: ' num2str(current_f),'    Cross_sectional Area:' num2str(num_pixels,'%0.2f')];
```

```
%% Outline the inner wall

final_im = insertText(mriImage,[5 5],text,'FontSize',12,'TextColor','white');
%figure,imshow(final_im);


for k = 1:length(boundaries)
    boundary = [boundaries{1}];
    final_im = insertShape(final_im, 'Line', [boundary(:,2)+x, boundary(:,1)+y, boundary(:,2)+1+x, boundary(:,1)+1+y], 'LineWidth',
end
```

Therefore, the area of the left ventricle could be determined and displayed as well as the frame number in each output frame. To clearly demonstrate the processed video, one frame was taken from every 20 frames to show the outcome:

```
if (mod(current_f, 20) == 0)
    subplot(3,3,current_f/20);
    imshow(final_im);
    title(sprintf('Frame: %d', current_f));
end
```

The general outcomes for locating inner walls only:

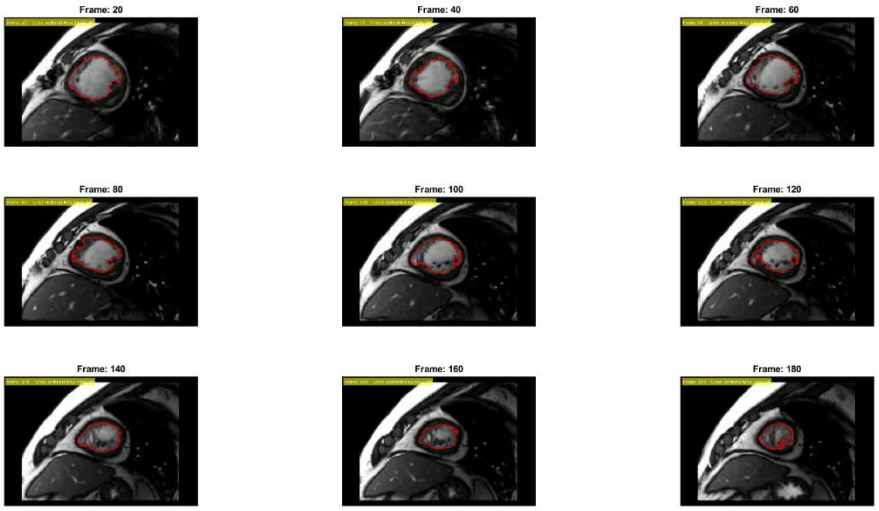


Figure 2.2.5: The general outcomes before boundary smoothing

Moreover, since the inner wall should be fairly circular, the exterior boundary of the inner wall could be smoothed by generating convex hull images from pre-processed left ventricle images by adding a function bwconvhull().

```
% Locate the largest connected region (left ventricle)
[~, idx] = max([stats.Area]);
bw = ismember(label, idx);
```

```
%% Number of Pixels
CC = bwconncomp(bw);
num_pixels = numel(CC.PixelIdxList{1});

% Smooth the boundary
smoothed_boundary = bwconvhull(bw);

 % Find the boundries
boundaries = bwboundaries(smoothed_boundary,'noholes');
```

The effect would be:

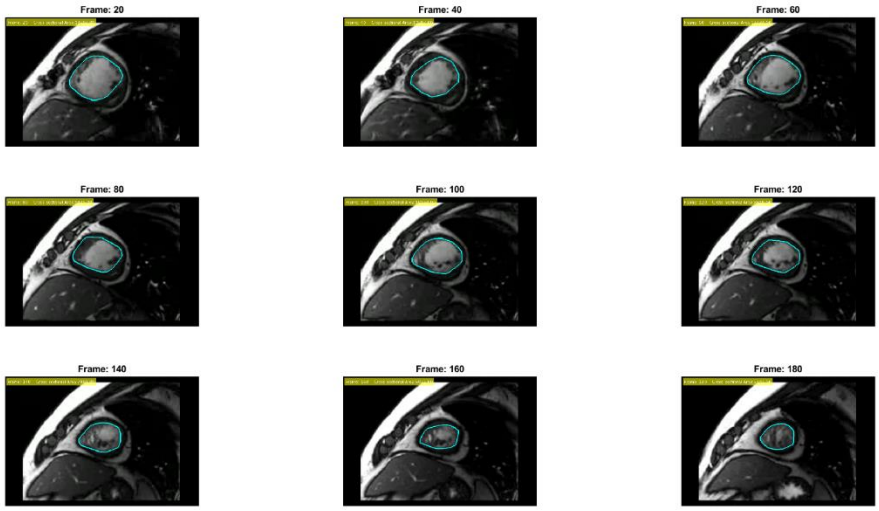


Figure 2.2.6: The general outcomes after boundary smoothing

A graph of the cross-sectional area of the left ventricle versus time has been produced:

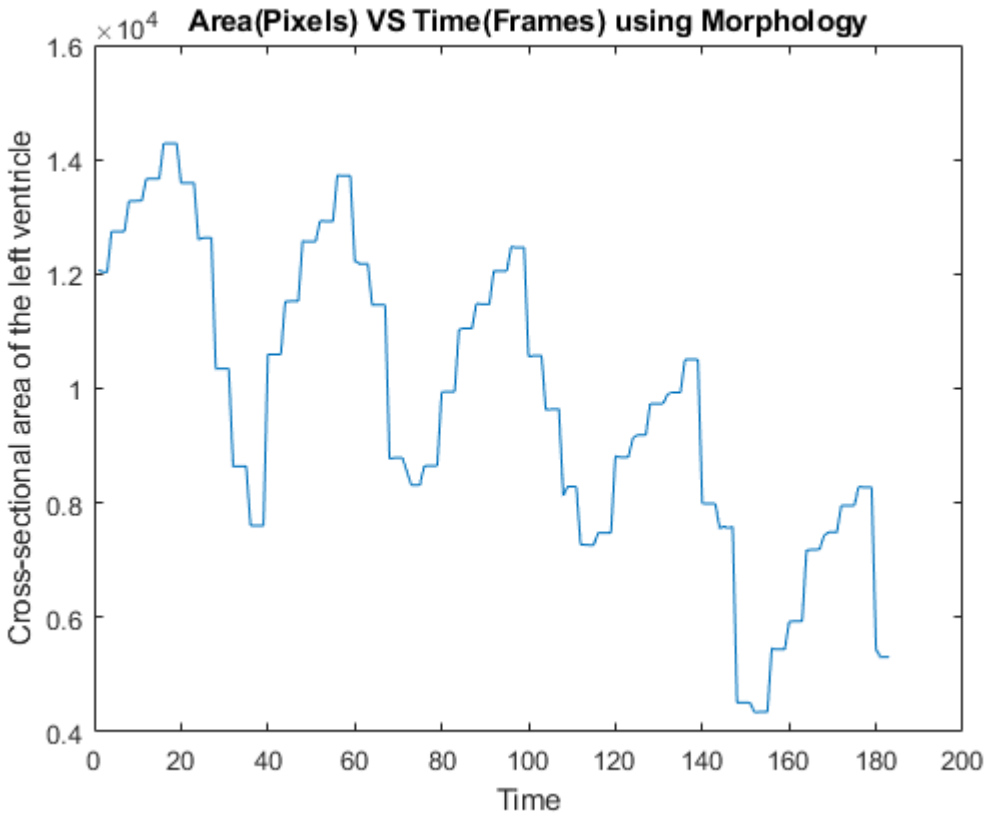


Figure 2.2.6: Area VS Time using Morphology Approach

2.3 Snake (Active Contour Model) Approach

Snake algorithm, also known as active contour algorithm, is one of the popular methods for image segmentation in image processing and computer vision. The algorithm uses a deformable curve ('like a snake') to gradually approach the contours of an object of interest. By computing image features such as gradients and intensity, the shape can be adjusted to closely fit the edges of the object.

The fundamental principle of the Snake algorithm is to minimize an energy function, which consists of internal energy (related to the shape of the snake) and external energy (related to image features). By adjusting the control points of the snake, when the energy function reaches its minimum value, the snake will conform to the edges of the object.

$$E_{total} = E_{internal} + E_{image}$$

The internal energy denotes for the bending energy of the contour,

$$E_{internal} = \alpha Elastic + \beta Smooth$$

Where α, β are weighting parameters:

$$\alpha \rightarrow the\ degree\ of\ the\ resistance\ to\ stretching\ of\ the\ contour$$

$$\beta \rightarrow the\ degree\ of\ the\ resistance\ to\ bending\ of\ the\ contour$$

The external energy is the energy of the image itself, denotes for the sum of gradient magnitude squared.

An initial contour (mask) was placed near the left ventricle. Under the influence of both internal and external energies, the contour trended move towards the edges of the inner wall. This approach has the following steps.

2.3.1 Pre-processing

Similar to the Morphology approach, the very first step was cropping the image and pre-processed the image using the same techniques from the Morphology approach.

```
x = 170;
y = 85;
width = 180;
height = 180;
%Crop the image for better extraction
mriImage = imcrop(mriImage, [x y width height]);
```

```
%% Pre process the image
I_gray = rgb2gray(histeq(mriImage));
I_filt = medfilt2(I_gray, [10 10]);
I_filt = imadjust(I_filt,[0.5 0.7],[0 1]); %test
I_filt = imclearborder(I_filt);

%Create the initial mask covering ventricle
mask = zeros(size(I_filt));
mask(40:end-40,40:end-40) = 1;
```

Since the snake algorithm requires an initial contour surrounding the objects of interest, a mask has been produced. To save computing resources and improve efficiency, the initial contour location should be as close to the inner wall of left ventricle as possible, while it should be large enough to cover all the regions of the objects of interest.

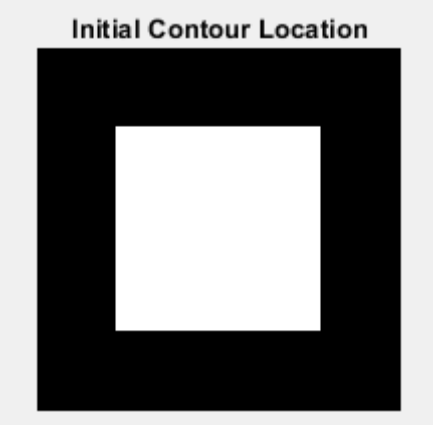


Figure 2.3.1: Initial Contour Location for Snake Algorithm

2.3.2 Iteration

The contour works like an elastic band, once the algorithm started it would iterate to shrink by moving contour points towards the goal. By observation, the gray scale value trended to change faster at the edge of the inner wall. Thus, the gradient magnitude of the image could be utilized to locate the edge of the inner wall and attract the band to stop it. Each point on the contour would be moved to the location with the higher gradient magnitude value within a window. Then the new location of each point was compared with its previous value, if the location not changed, the edge location was found, otherwise the same operation would be iterated until not changing any more or reaching the specified number of loop iterations.

The iteration number should be selected carefully, though the more times the algorithm iterates, the more accurate the result is, it consumes more computational resources at the same time. Take Image0000 as an example,

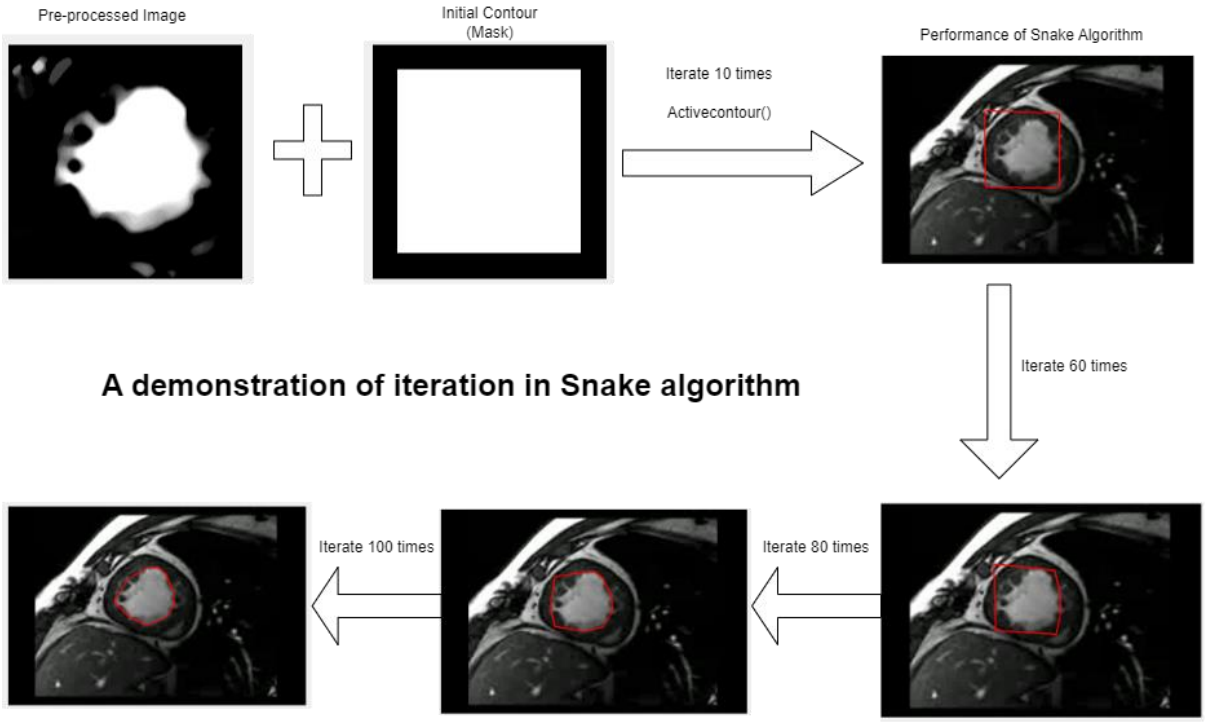


Figure 2.3.2: Image0000

From Figure 2.3.2, 100 iteration times should be enough to locate the inner wall for Image0000. However, as the left ventricle shrink in the later frames, 100 iteration times could not reach the edge of the inner wall accurately. Take Image0156 as an example,

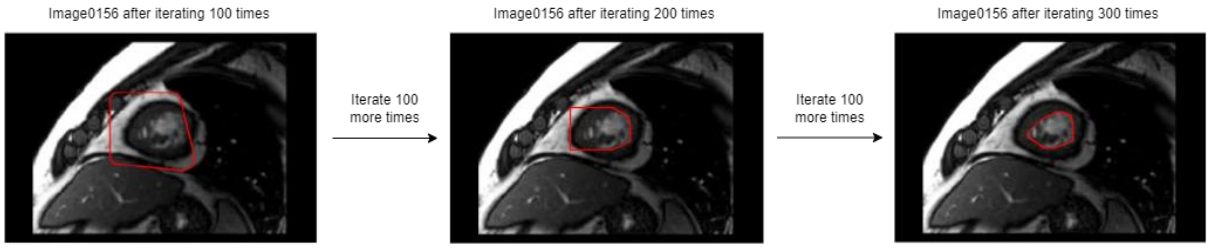


Figure 2.3.3: Image0156

After testing the performance for the entire set of images, a number of 360 has been selected for iterating to obtain a fair result.

```
%Iterate 360 times using snake algorithm
I_snake = activecontour(I_filt,mask,360);
```

2.3.3 Determining the internal area (Snake).

The determination of the internal area was same as the Morphology approach by assuming the largest connected region as the inner wall. Similarly, to clearly demonstrate the processed video, one frame was taken from every 20 frames to show the outcome:

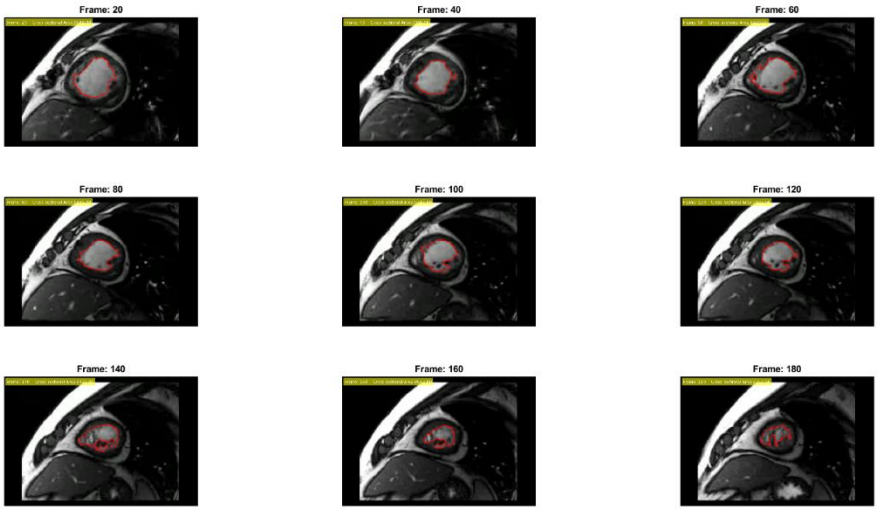


Figure 2.3.4: The general outcomes before boundary smoothing

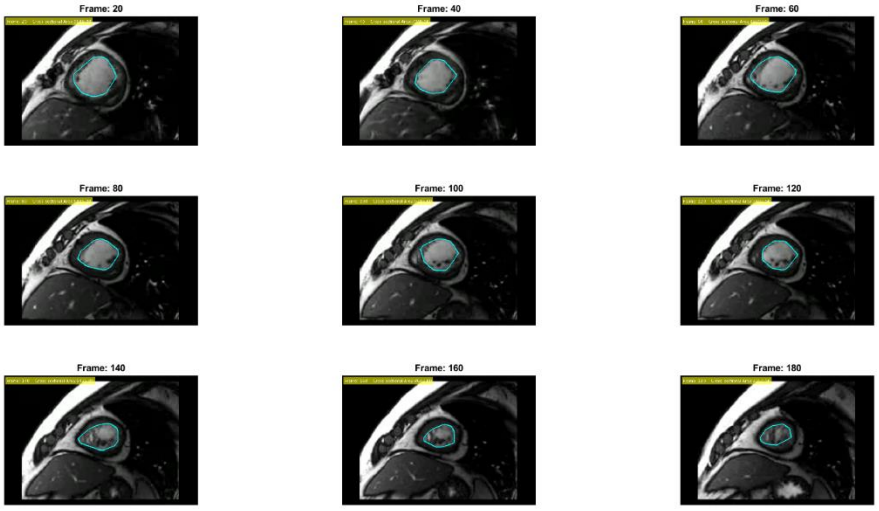


Figure 2.3.5: The general outcomes after boundary smoothing

`

Now that a graph of the cross-sectional area of the left ventricle versus time has been produced:

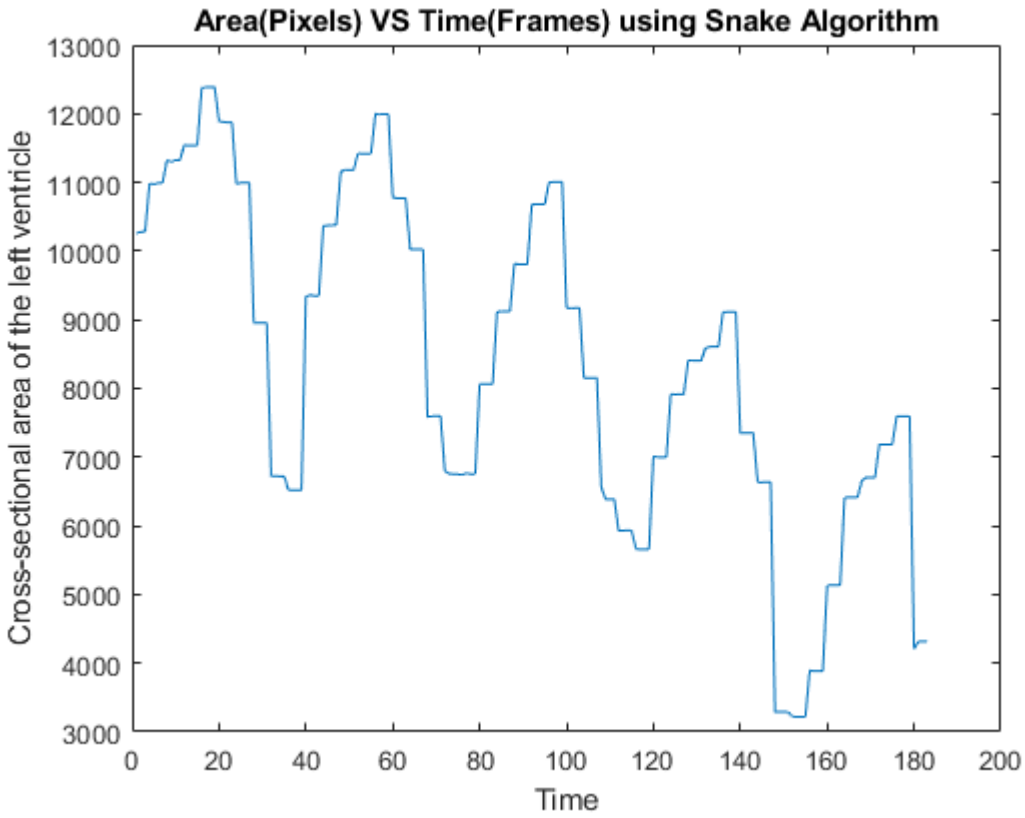


Figure 2.3.6: Area VS Time using Snake Approach

2.4 Comparison between Morphology and Snake Approach

Above all, both Morphology and Snake algorithms could be implemented to do the boundary segmentation operation in the application of MRI beating heart images. Both the two algorithms are sensitive to the selection of parameters, and they performed well in locating the inner wall of the left ventricle provided that the selection of parameters is accurate. However, they each have their advantages and disadvantages as well. To compare them more specifically, the internal area computed from the two algorithms have been plot on the same graph as follows,

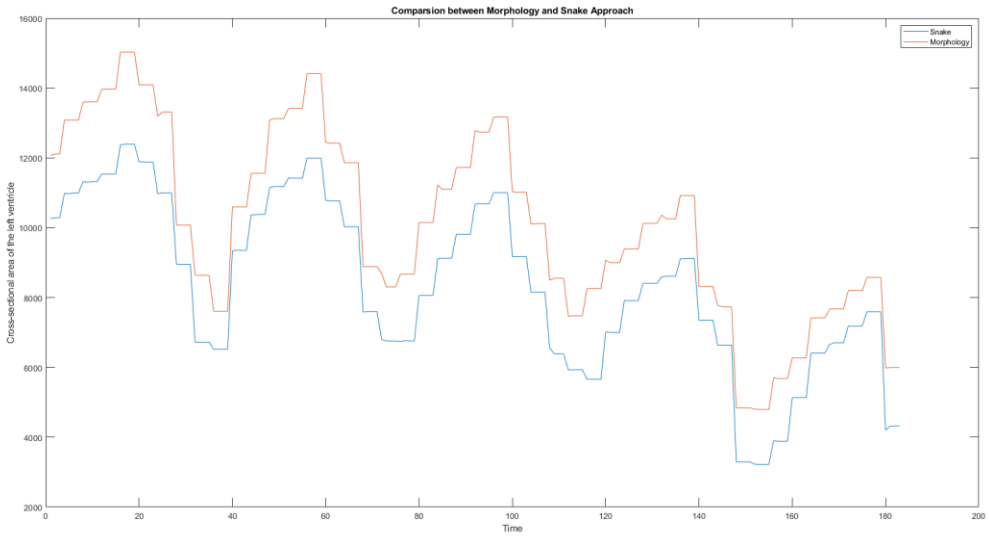


Figure 2.4.1: Comparison between Morphology and Snake Approach

As we can see from Figure 2.4.1, both algorithms correctly described the process of heart beating, i.e., the internal area of the left ventricle keeps expanding and shrinking. The difference is obvious: the area computed from Morphology approach is generally greater than the area computed from Snake approach.

$$Area_{Morphology} > Area_{Snake}$$

Take Image0000 as an example to take a deeper look,

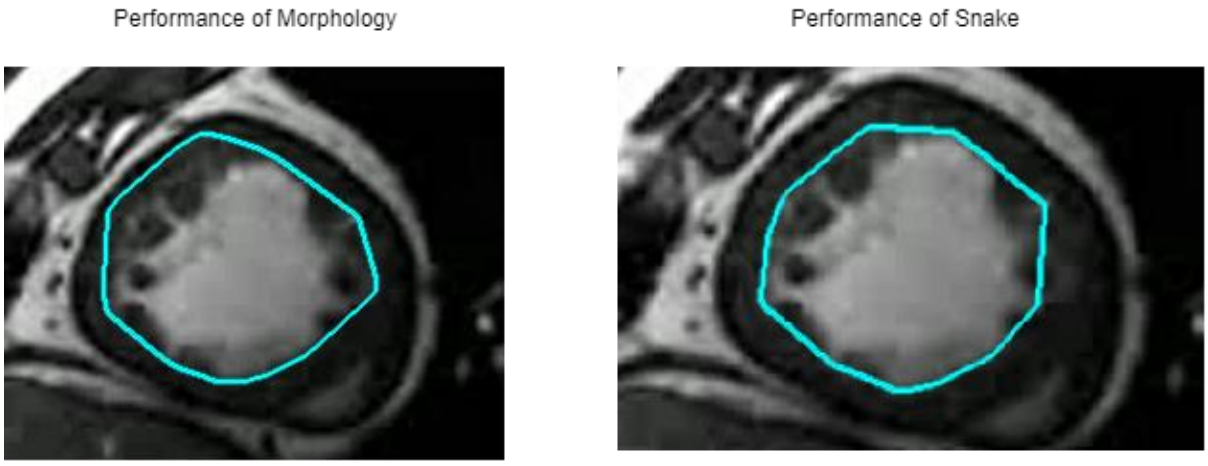


Figure 2.4.2: Image0000

There are some excess black structures being included using Morphology, resulting in a difference between the areas. In contrast, the Snake algorithm outlines the inner wall more rigorous. Take Image0156 as another example when the heart is shrinking,

Figure 2.4.3: Image0156

Apparently, the area using Snake is smaller than the area using Morphology. This is because the snake algorithm detects the boundary of the object based on minimizing its energy function described previously, thus the sum of the gradient magnitude should be as large as possible. By observation we can see that the gradient magnitude value trends to reach the roof as the grayscale value trends to change greatly, that is the edge of the inner wall. Therefore, the Snake algorithm would shrink the contour as much as possible to fit the boundary of the object. In contrast, the Morphology algorithm is less effective for boundary segmentation with varying target shapes and non-convex shapes.

Therefore, the advantage of Snake algorithm is that it can adapt to the segmentation of complex target shapes and non-convex shapes and has a better effect on maintaining the continuity and smoothness of the target contour. However, the disadvantage of Snake algorithm is obvious as well, it requires highly accurate selection of parameters, and is computationally intensive when processing large-scale images. The determination of the initial contour location could be ineffective as the location of the left ventricle is varying, a general initial contour location would result in computational inefficiency, while it is not always realistic to manually determine the initial contour. Moreover, snake cannot easily change topology to segment multiple objects; the contour points

often bunch up or spread out, reducing the stability of the solution; they are difficult to extend to higher dimensions.

In contrast, the primary advantage of Morphology algorithm is that it consumes much less computational resources and easier to design and implement. The low complexity of such algorithm brings advantage of fast computing. While it is faster, the performance could be a bit weaker than Snake algorithm because it could not fit the boundaries perfectly.

Overall, the algorithms each have their own advantages and disadvantages, so the most suitable algorithm needs to be selected on a case-by-case basis. For instance, thresholding probably not a suitable algorithm for this application as the grayscale values of objects and surroundings could be vague to distinguish. In practice, the combination of different algorithms could be implemented to obtain better boundary segmentation results.

2.5 Outer Wall Segmentation

The outer wall composed of thick cardiac muscle tissue; thus the boundary segmentation of outer wall could be more challenging because the outer wall could not form a closed region for most of the frames. Take Image0000, 0155 and 0156 as an example after binarizing and inverting,

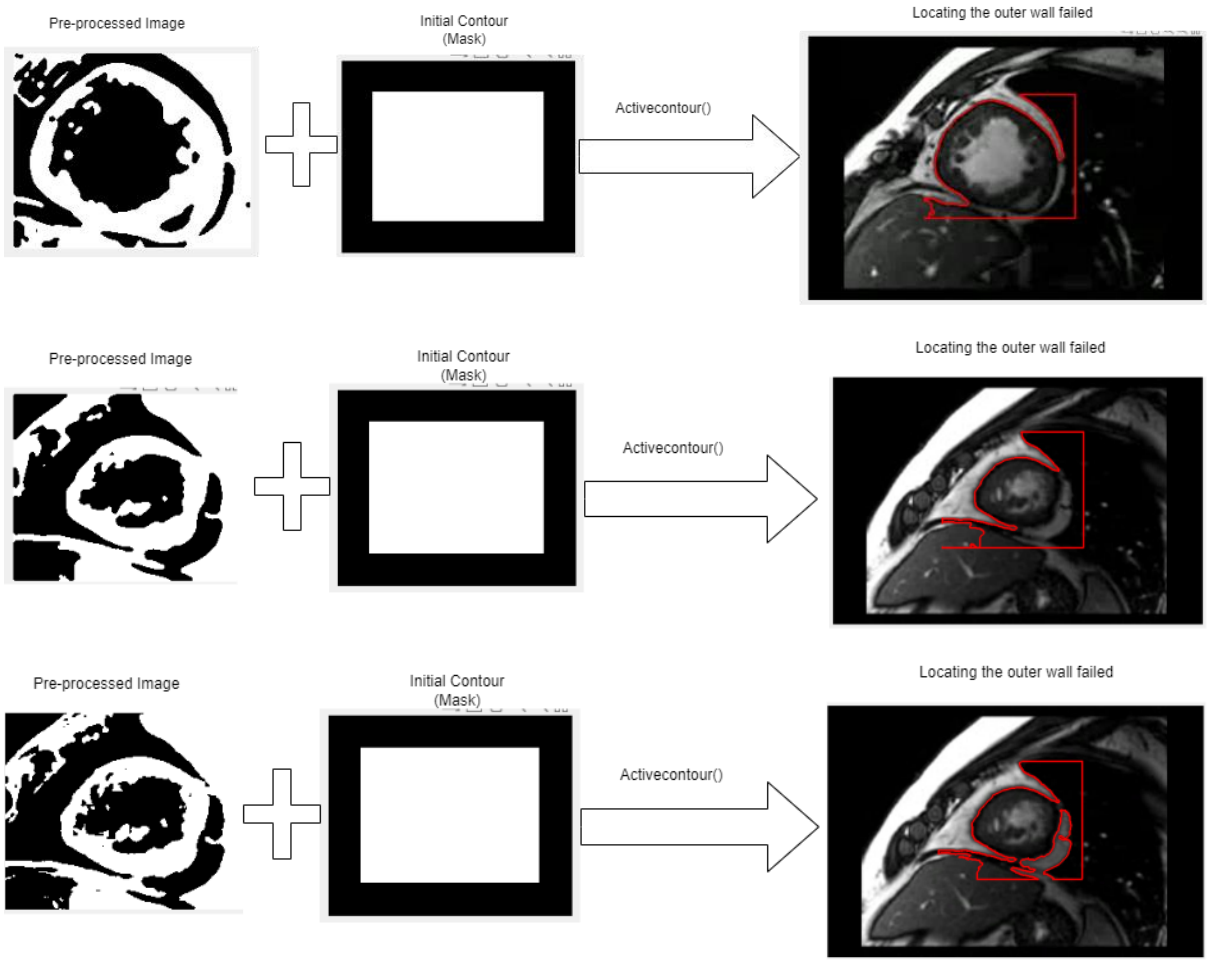


Figure 2.5.1: The difficulty points of locating the outer wall

As we can see from Figure 2.5.1, the closed portion of the outer wall could be located using Snake algorithm, however the open portion of the outer wall could not be successfully located.

However, possible solutions could be suggested to solve the problem. Since the area of the inner wall has been determined from the previous steps, the image could filter out the inner wall based on its area using the function bwareafilt. Now that morphology operations could be implemented without the interference from the inner wall. A combination of morphology and snake algorithms could be employed to locate the outer wall.

Alternatively, since the outer wall is much more circular by observation, once the coordinates of the edge of the outer wall that have been correctly located and determined, the consecutive coordinates missed due to the open area could be estimated based on the properties of circle.

Q2. Volumetric 3D Reconstruction

1. Introduction

Volumetric Modelling as a branch of Dense 3D Reconstruction, is capable of completing 3D geometry from a set of 2D images with various viewpoints and generating highly detailed 3D model accordingly. Unlike sparse reconstruction, volumetric modelling aims to capture the entire geometry of the object including surface textures by representing a 3D object as a set of voxels (volumetric pixels) arranged in a mesh grid.

In this application, shape from silhouette and nested for loops have been implemented to reconstruct 3D model of a toy dinosaur with provided images.

The inputs:

- 36 viewpoints of a toy dinosaur spinning on a turntable (10° per image).
- 36 projection matrices which determine how each spatial point maps onto the image plane.

The output:

- A volumetric model of the dinosaur
- (To be submitted) 4 new viewpoints of the reconstructed dinosaur which do not coincide with any of the original 36 viewpoints. Attempt to texture map the images for an improved solution.

2. Methodology

Shape from Silhouette and Projective Geometry are the critical methods for volumetric modelling the 3D dinosaur with the following steps.

2.1 Pre-Process Images

Since there are 36 2D images, they were stored in a cell array. The images acquisition and processing could be done inside a nested for loop.

```
%% Read the images and store to the cell array
for i = 1:iImages
    im_name = sprintf('dino%02d.jpg',i-1);
    images{i} = imread(im_name);

    % imageSizes(i, 1): width of the ith image; imageSizes(i, 2): height of the ith image.
    imageSizes(i, :) = size(images{i}); % Get size of current image
```

By observation, the colour of the dinosaur composed of intensive red and pale blue, while the background composed intensive blue and pale red. Thus, the colour difference between the foreground and the background could be computed for setting the threshold and separate them. A binarized image could be generated and required to remove noise for accurate voxelization and modelling. Now that the shape from silhouette should be clear. Take the first image as an example,

```
 % Compute silhouette using thresholding
colourDiff = images{i}(:,:,1) - images{i}(:,:,3);
silhouette = colourDiff > 0; % Foreground pixels have positive difference

% Remove Noise
silhouette = imclearborder(silhouette);
silhouette = imfill(silhouette, 'holes');

% Store silhouette in a new cell array
Sil_images{i} = silhouette;
```

Figure 2.1: Shape from Silhouette (dino00.jpg)

2.2 Volumetric Modelling

The correspondences between camera coordinates and world coordinates could be established using provided projection matrices. The world coordinates and range of the dinosaur in each image were provided:

```
% World Coordinates for dinosours, resolution = 2
% @Volume.txt
x = -180:2:90;
y = -80:2:70;
z = 20:2:460;

% Projection matrices
% @Dino projection matrices.txt
P0 = [1.134783 1.069317 0.046803 347.735102;
     -0.447199 0.330630 1.035582 -3.804233;
     0.000382 -0.000339 0.000072 1.000000];

P1 = [1.303228 0.856019 0.046803 347.735102;
     -0.382992 0.403262 1.035582 -3.804233;
     0.000318 -0.000400 0.000072 1.000000];
```

```
P_Mat = cat(3, P0, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, ...
    P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33, P34, P35);

% Create 3D meshgrid
[X, Y, Z] = meshgrid(x, y, z);
voxel = ones(numel(X), 1);
```

Therefore, a 3D mesh grid could be created to store voxels in 3D. Given a voxel in real space, to project it onto the image seen by a specific camera, the provided projection matrix was used as follows:

$$P \times \begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

For instance, let the voxel v have world-coordinates (a, b, c), and the corresponding pixel p have image-coordinates (x, y) which we want to derive. Let P be the 3 x 4 projection matrix for that particular camera. Then:

$$v = [a, b, c, 1];$$

$$p = P * v;$$

$$[x, y] = p(1:2)./p(3);$$

Thus, the 3D voxels could be projected onto 2D silhouette images after carving,

```
%Project voxels onto 2D silhouette images
%@Projection.txt
Cam_Z = P_Mat(3,1,i) * X + P_Mat(3,2,i) * Y + P_Mat(3,3,i) * Z + P_Mat(3,4,i);
Cam_X = P_Mat(1,1,i) * X + P_Mat(1,2,i) * Y + P_Mat(1,3,i) * Z + P_Mat(1,4,i);
Cam_Y = P_Mat(2,1,i) * X + P_Mat(2,2,i) * Y + P_Mat(2,3,i) * Z + P_Mat(2,4,i);

Cam_X = round(Cam_X ./ Cam_Z);
Cam_Y = round(Cam_Y ./ Cam_Z);

%A binary mask that identifies the valid pixels within the image dimensions.
bwMask = find((Cam_X >= 1) & (Cam_X <= imageSizes(i, 2)) & ...
                (Cam_Y >= 1) & (Cam_Y <= imageSizes(i, 1)));

%Carve away invalid pixels
Cam_X = Cam_X(bwMask);
Cam_Y = Cam_Y(bwMask);
```

Then V matrix was required to be designed to compute 3D coordinate of each voxel and obtain the region of interest in world coordinates.

```
%% Generating V matrix to compute 3D coordinate of each voxel
% remove repetitions
Xu = unique(X);
Yu = unique(Y);
Zu = unique(Z);

% Create mesh grid for voxels
[voxelX,voxelY,voxelZ] = meshgrid(Xu, Yu, Zu);

% Voxelization: Convert into a mesh grid of voxels
V = zeros(size(voxelX));
N = numel(X);
for j = 1:N
    ix = (Xu == X(j));
    iy = (Yu == Y(j));
    iz = (Zu == Z(j));
    iz = flipud(iz); % Flip array in the up-down direction
    V(iy,ix,iz) = voxel(j); % Final V matrix
end
```

The above code snippet converted data into a voxel mesh grid so that the 3D coordinate of each voxel could be identified. Now that the volumetric modelling has completed.

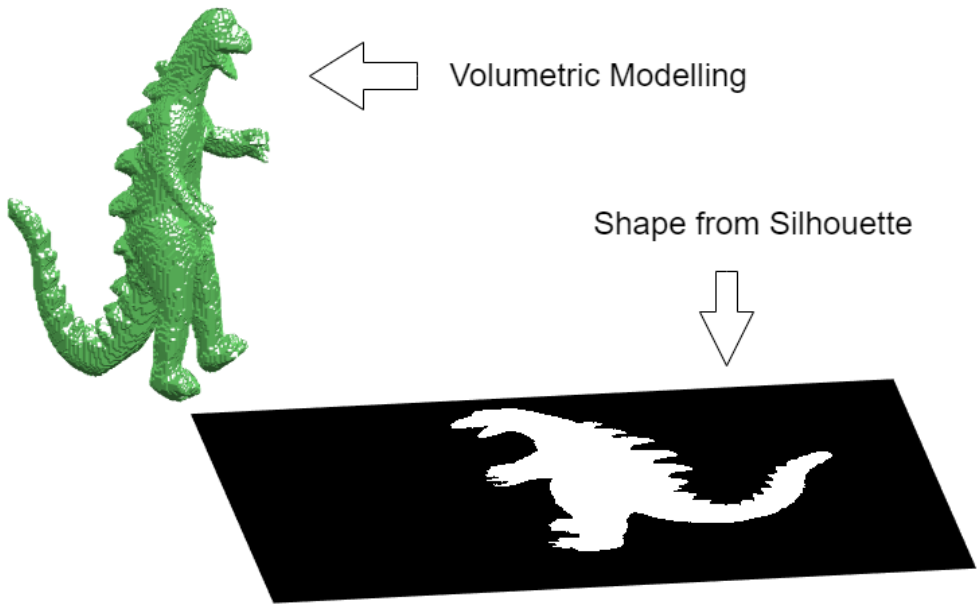


Figure 2.2: Effect of Volumetric Modelling

However, as we can see from Figure2.2, the claws of the dinosaur might be distorted and not exactly the same as the original image. This might because the original colour of the claws is white, which being distorted in the step of pre-processing. This could be improved using texture mapping.

2.3 Texture Mapping

Texture mapping involves mapping texture data from the original2D images onto the reconstructed 3D model. The texture analysis faces multiple challenges. Firstly, the original images are taken from different viewpoints, thus the texture information could lose due to the varying angles. Secondly, distortion occurs when project the voxels onto 2D Silhouette. Thirdly, the process of texture reconstruction could be highly computational inefficient as the dataset is extensive.

To texture map the images for an improved solution, colour of the dinosaur could be mapped first to make the model look more similar to the originals. I firstly tried to adjust the edge colour to dark orange using RGB triplet to approach the original colour. However, though the colour was approaching the original colour, the texture information lost due to the edge effect. The face colour could be set instead to avoid distortion. A dark orange roughly composes of a high portion of red and a relatively low portion of green with no blue being involved. In addition, the background colour can be modified. Therefore, a more realistic 3D model was constructed in the perspective of colour:

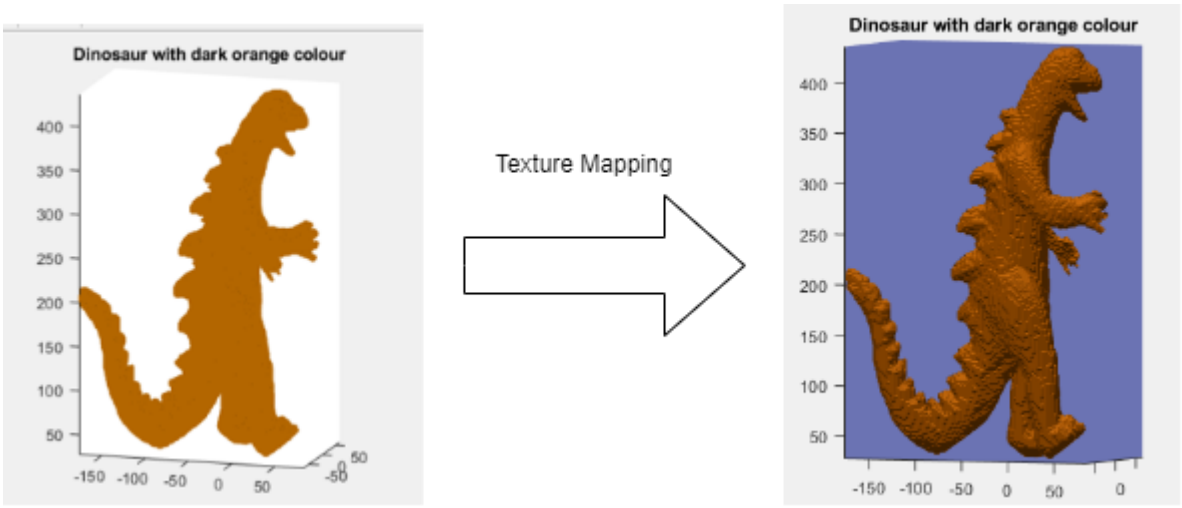


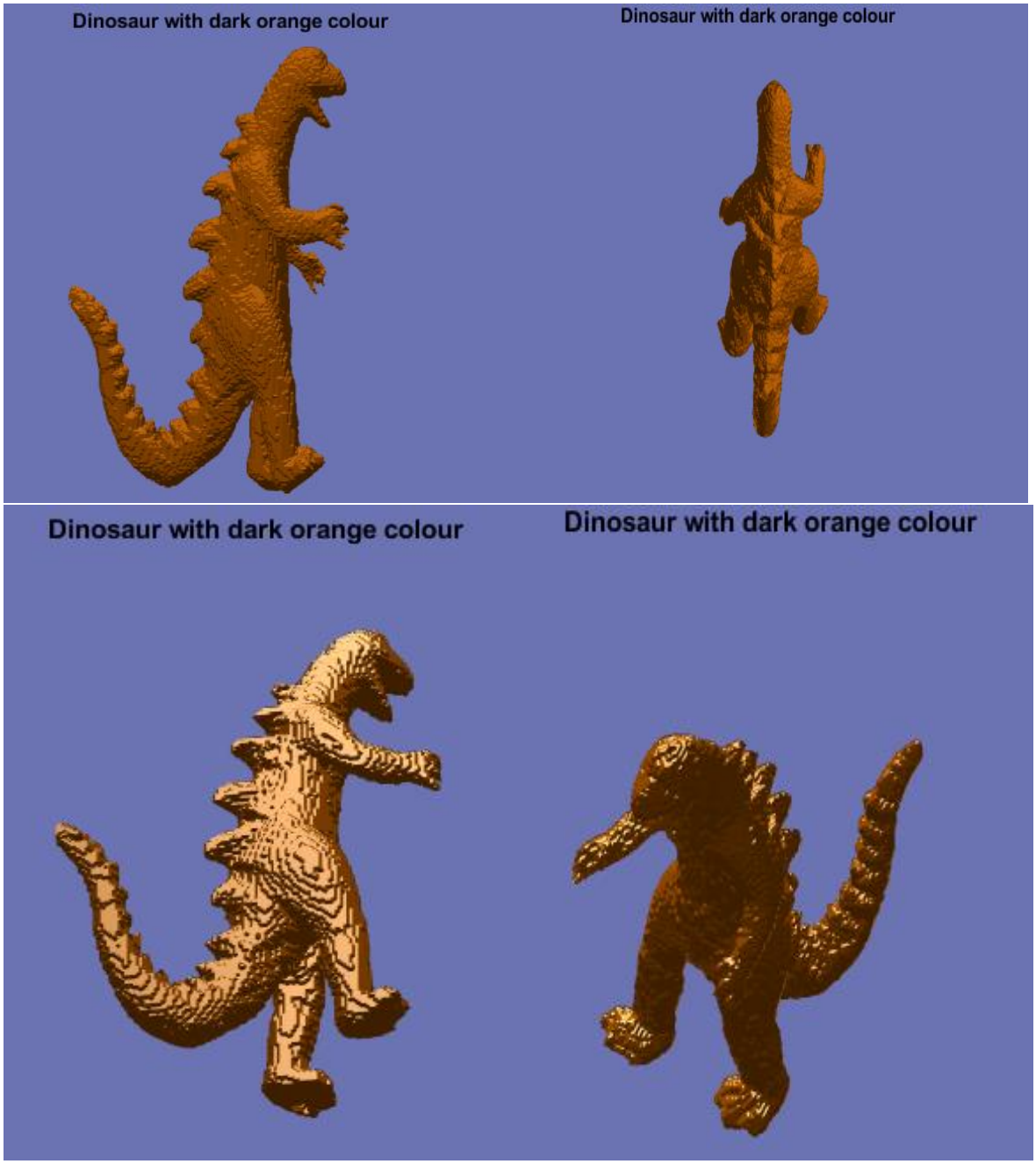
Figure 2.3: Effect of colour restoration

The challenging part is that the 3D model is made up of a set of individual facets, where each of which has its own colour and edge. In contrast, though a general colour of dark orange could be applied, all the facets the model have the same colour. Hence, the hardest part of colour mapping is figuring out the positions of voxels corresponding to the different colour areas of the original dinosaur. For example, since the dinosaur's chest, teeth and claws are white, the positions of corresponding voxels are required to be located and set to white.

Possible solutions could be saving the colour information of each individual image within the nested for loop so that the specific colour of the dinosaur could be obtained for further process. There are 36 cameras in total with different directions which can be vectorized. Once the normal vector of each voxel in the 3D view can be derived, the vectorized camera direction can be matched to each normal vector based on the shortest distance. Thus, the corresponding 2D picture can be loaded into the 3D model. Since each voxel can contain colour information such as RGB values, colormap function may be implemented to import the original colours to the reconstructed 3D model, working in conjunction with the projection mechanism. Therefore, the spital coordinates of each voxel will be determined in correspondence with the colour information, then the varying colour should be restored back to the model.

Once colour has been restored, the texture mapping could be done using texture analysis techniques such as joint probability matrix. Since the grayscale distribution of texture images is normally periodic, joint probability distribution can do statistical investigation on the pixels of an image to evaluate the grayscale distribution. Matrix can be employed to describe the correlation between features, thereby facilitating the transformation of textures. If there are sufficient data samples available, Deep Learning (DL) and Generative Adversarial Networks (GAN) can also be used for texture mapping.

4 new viewpoints of reconstructed dinosaur were illustrated as below:



Q3. Face Recognition

1. Introduction

Despite eigenface technique is obsolete nowadays, it was the innovative method bring face recognition system to a truly usable level. This question aims to build a simple face recognition system based on eigenface technique to recognize faces from the database on the website with about 96% correct recognition and a GUI interface.

2. Methodology

To process face recognition, the images are generally treated as basis vectors. During the process of vectorizing images, each pixel of the image is assigned to one dimension, resulting in a high-dimensional vector for a typical image. Eigenface technique functions by projecting the face images onto a feature space that spans the significant variations among known images. It then characterizes an individual face by a weighted sum of the eigenface features (Matthew 1991).

However, such a high dimensionality poses difficulties for subsequent image calculations, which knowns as 'curse of dimensionality'. Therefore, it is essential to reduce the dimensionality of the image while retaining as much important information as possible. Principal component analysis (PCA) is a dimensionality reduction method that can reduce the image dimensionality without losing important information (keeps principal components only). It basically functions by eliminating the correlation of image data and find a feature vector space in which the data of each category could be appropriately classified. Based on the distance metric of different face images in the feature space, eigenface technique judges their similarity and achieves face recognition.

Step 1

Firstly, a representative training set of face images was obtained from the data and transformed into vector form. The dataset contains frontal photos of 6 people. Each person has multiple photos with various shooting angles, expressions, and lighting conditions, while each face image could be represented as 16384 (128*128 pixels)-dimensional vector $\mathcal{T}_i$. Therefore, a set S contains M=6 images could be obtained:

$$S = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_6\}$$

Figure 2.1: Training dataset

Step 2

After obtaining a set of training face vectors S, the average face Ψ can be calculated. To compute the average face, the following definition of Ψ was used:

$$\Psi = \frac{1}{M}\sum_{n=1}^{M} \mathcal{T}_n$$

```
%% Calculate average face
avg_face = mean(face_vectors,2);
% Reshape the average face vector back to image
avg_face_image = reshape(avg_face, image_size);
figure, imshow(avg_face_image);
title('Average face of the set');
```

Thus, by transforming the average face vector back to image (arrangement of pixels), the average face looks like:

Figure 2.2: The average face

Step 3

The face vectors can be standardized by calculating the difference between each eigenface and the average face. Thus, each face differs from the average by the vector $\Phi_i = \mathcal{T}_i - \Psi$.

```
%% Standardise face vectors
face_diff = face_vectors - avg_face;
```

Step 4

The set of very large vectors was then subjected to PCA. It seeks a set of M orthonormal vectors, u_n, which effectively represents the distribution of Φ_i derived from Step 3. The kth vector, u_k was chosen such that,

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}(u_k^T\Phi_n)^2$$

is a maximum, subject to,

$$u_l^T u_k = \begin{cases} 1, if\ l = k \\ 0, otherwise \end{cases}$$

The vector u_k and scalars λ_k are the eigenvectors and eigenvalues respectively, of the covariance matrix,

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T$$

where the matrix $A = [\Phi_1, \Phi_2, \dots, \Phi_m]$. Therefore, the covariance matrix C can be computed from the standardized face vectors. However, since determine the covariance matrix under 128^2

dimensions are significantly computational ineffective and intractable, a more feasible method was implemented.

Step 5

Since 6 eigenfaces were involved, there would be only 5, rather than 128^2, meaningful eigenvectors as the remaining eigenvectors are zeros. Hence, the 128^2-dimensional eigenvectors could be solved with a 6×6 matrix, which was constructed based on

$L = A^T A$. Keep in mind that the order of A^T *and* A cannot be reversed.

where $L_{mn} = \Phi_n^T \Phi_n$

```
% Perform PCA to obtain the eigenfaces
% covariance matrix
C = face_diff' * face_diff;
```

Then the eigenfaces can be computed,

$$u_l = \sum_{k=1}^{M} v_{lk} \Phi_k, \quad l=1,2,\ldots,6$$

The implementation of PCA significantly reduced the computational complexity from the order of the number of pixels in the images (128^2) to the order of the number of images in the training set (6). The associated eigenvalues prioritize the eigenvectors based on their usefulness. By transforming the eigenfaces back to images (arrangement of pixels), the 6 eigenfaces look like:

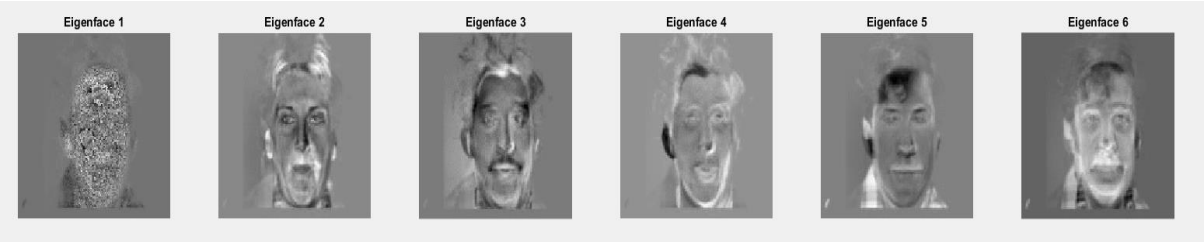


Figure 2.3: The eigenfaces

Step 6

Since the eigenfaces were obtained, face recognition can be performed by projecting a new facial image into the subspace spanned by the eigenfaces ('feature space') and then classifying the face by comparing its position in feature space with the positions of known individuals (Determine the shortest Euclidean distance).

A new facial image can be transformed into its eigenface vector (project into feature space) by,

$$\omega_k = u_k^T (\mathcal{T} - \Psi)$$

The projection operation characterizes an individual face by a weighted sum of the eigenface features. The weights form a vector $\Omega^T = [\omega_1, \omega_2, \ldots, \omega_M]$. Face recognition can be achieved by comparing these weights to those of known individuals.

To examine the success rate by recognizing 33 facial images from the database, all images were extracted to the same folder for conveniency. The first character of each test image file corresponds to the correct known individual, for example, '3b.bmp' means the facial image corresponds to the third person. In addition, the recognized person was matched by finding the shortest Euclidean distance of the weights.

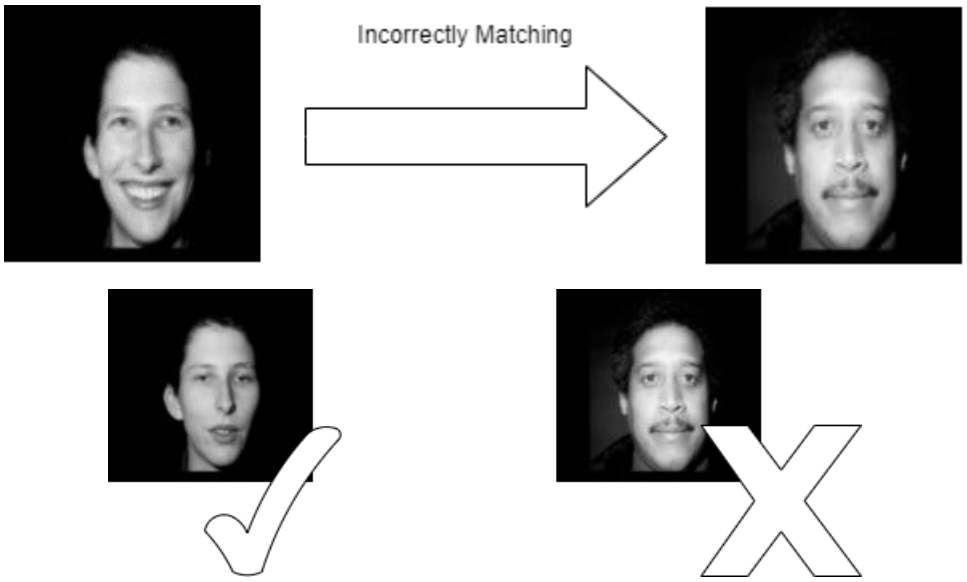
```
for num = 1:num_images
    inputName = allInputs{1,num};
    [input, map1] = imread(inputName);
    %% Reshape to gray image with 16384 Dimensions
    test_image_gray = rgb2gray(ind2rgb(input, map1));
    test_image_vector = test_image_gray(:);

    % Standardise image vectors
    test_image_diff = double(test_image_vector) - avg_face;

    % Project test image onto feature space
    test_coeffs = eigenface_basis' * test_image_diff;

    % Euclidean distance between the projection coefficients of the test image and each of the training images.
    distances  = pdist2(test_coeffs', projection_coeffs', 'euclidean');
    % Find the matched person from minimum Euclidean distance
    [~, matched_person] = min(distances);

    %% Determine the success rate
    fprintf('Input image %s matched to person %d\n', allInputs{1,num}, matched_person);
    if str2num(allInputs{1,num}(1)) == matched_person
       success = success+1;
    end

end

fprintf('Successful Rate is %.2f%%\n', success*100/num_images);
```

Therefore, the face recognition can be performed, and success rate was evaluated.

```
Command Window
Input image 1a.bmp matched to person 1
Input image 1b.bmp matched to person 1
Input image 1c.bmp matched to person 1
Input image 1d.bmp matched to person 1
Input image 1e.bmp matched to person 1
Input image 1f.bmp matched to person 1
Input image 1g.bmp matched to person 1
Input image 2a.bmp matched to person 2
Input image 2b.bmp matched to person 2
Input image 2c.bmp matched to person 2
Input image 2d.bmp matched to person 2
Input image 3a.bmp matched to person 3
Input image 3b.bmp matched to person 3
Input image 3c.bmp matched to person 5
Input image 3d.bmp matched to person 3
Input image 3e.bmp matched to person 3
Input image 3f.bmp matched to person 3
Input image 3g.bmp matched to person 3
Input image 3h.bmp matched to person 3
Input image 3i.bmp matched to person 3
Input image 3j.bmp matched to person 3
Input image 4a.bmp matched to person 4
Input image 4b.bmp matched to person 4
Input image 4c.bmp matched to person 4
Input image 4d.bmp matched to person 4
Input image 5a.bmp matched to person 5
Input image 5b.bmp matched to person 5
Input image 5c.bmp matched to person 5
Input image 5d.bmp matched to person 5
Input image 6a.bmp matched to person 6
Input image 6b.bmp matched to person 6
Input image 6c.bmp matched to person 6
Input image 6d.bmp matched to person 6
Successful Rate is 96.97%
```

Figure 2.4: The face recognition results.

From Figure 2.4, we can see that 32 out of 33 images were successfully recognized, achieving a success rate of 96.97%, while the input image '3c.bmp' incorrectly matched to person 5.



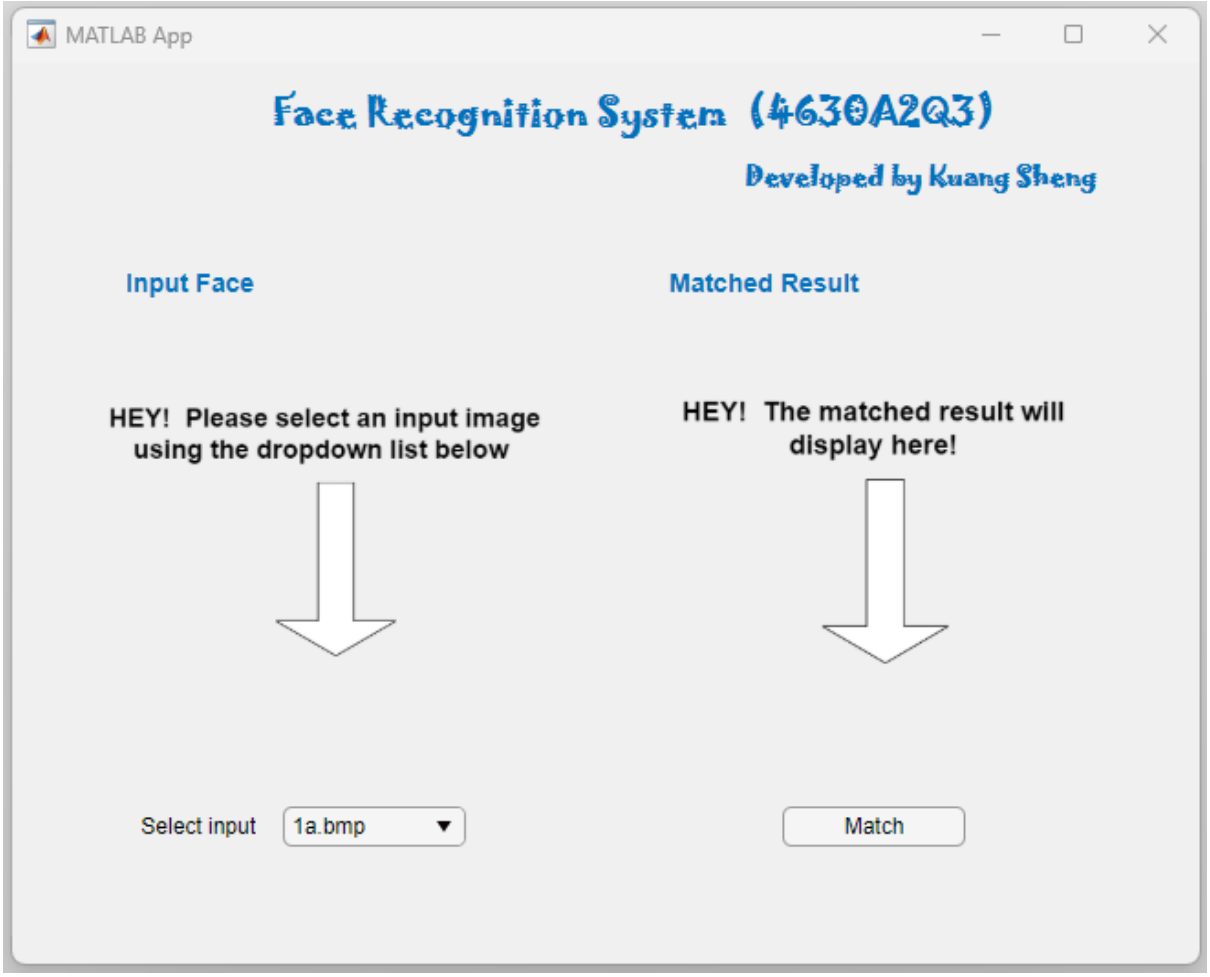
This could because eigenface technique is sensitive to lighting conditions. The eigenface technique often assumes that images have similar lighting conditions, while it is often not the case in practise. By observation, the matched image has similar lighting conditions as the input image, while the correct image is darker. The variation of the lighting conditions can affect the accuracy of the algorithm.

3. GUI interface

The GUI user interface was designed and developed based on 'AppDesigner' in MATLAB. The GUI has the following properties:

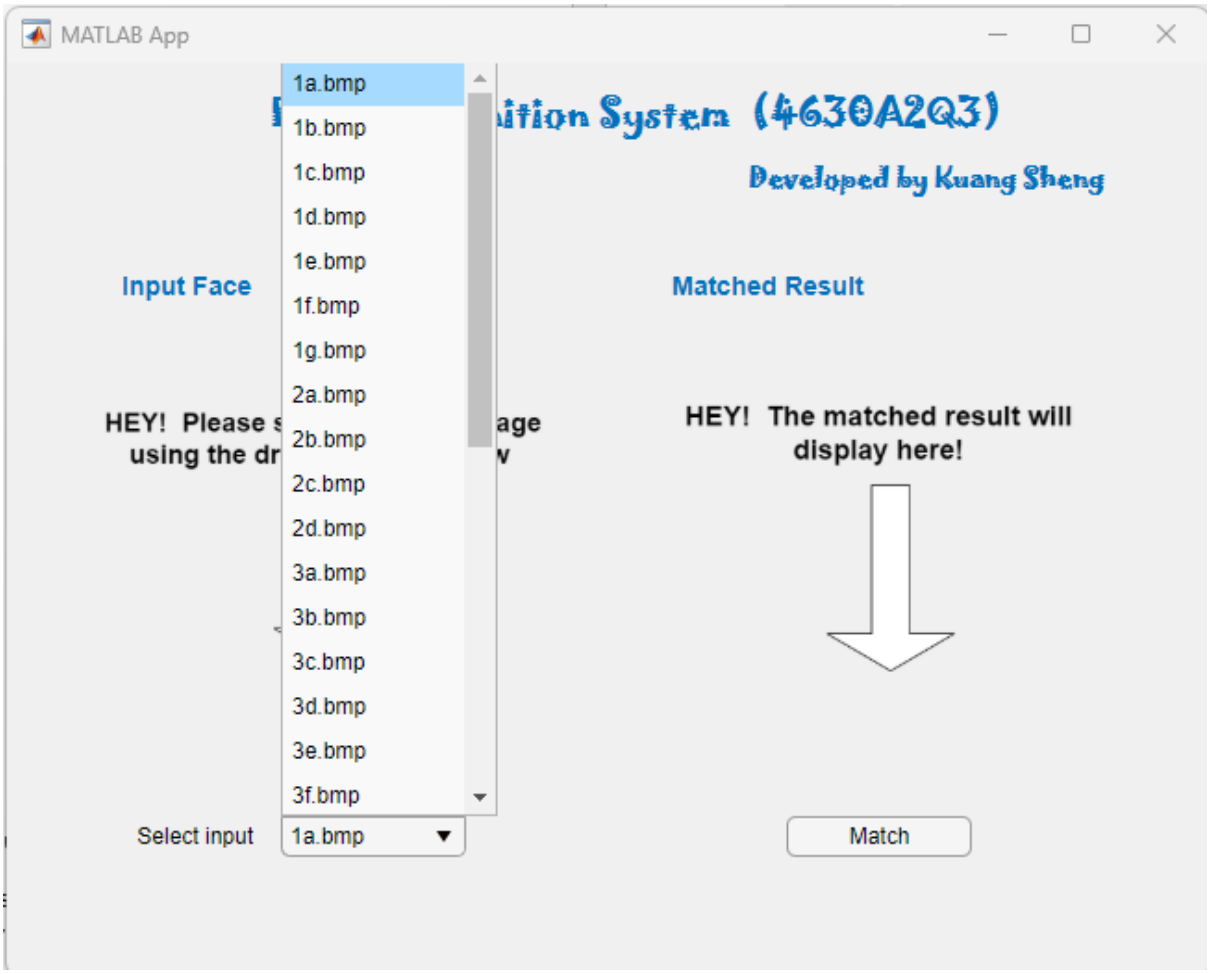
```
% Properties that correspond to app components
properties (Access = public)
    UIFigure                    matlab.ui.Figure
    SelectinputDropDownLabel    matlab.ui.control.Label
    SelectinputDropDown         matlab.ui.control.DropDown
    MatchButton                 matlab.ui.control.Button
    Image                       matlab.ui.control.Image
    Image2                      matlab.ui.control.Image
    InputFaceLabel              matlab.ui.control.Label
    MatchedResultLabel          matlab.ui.control.Label
    FaceRecognitionSystem4630A2Q3Label  matlab.ui.control.Label
    DevelopedbyKuangShengLabel  matlab.ui.control.Label
end
```

The initial canvas:



The dropdown list contains all 33 facial images for inputting.

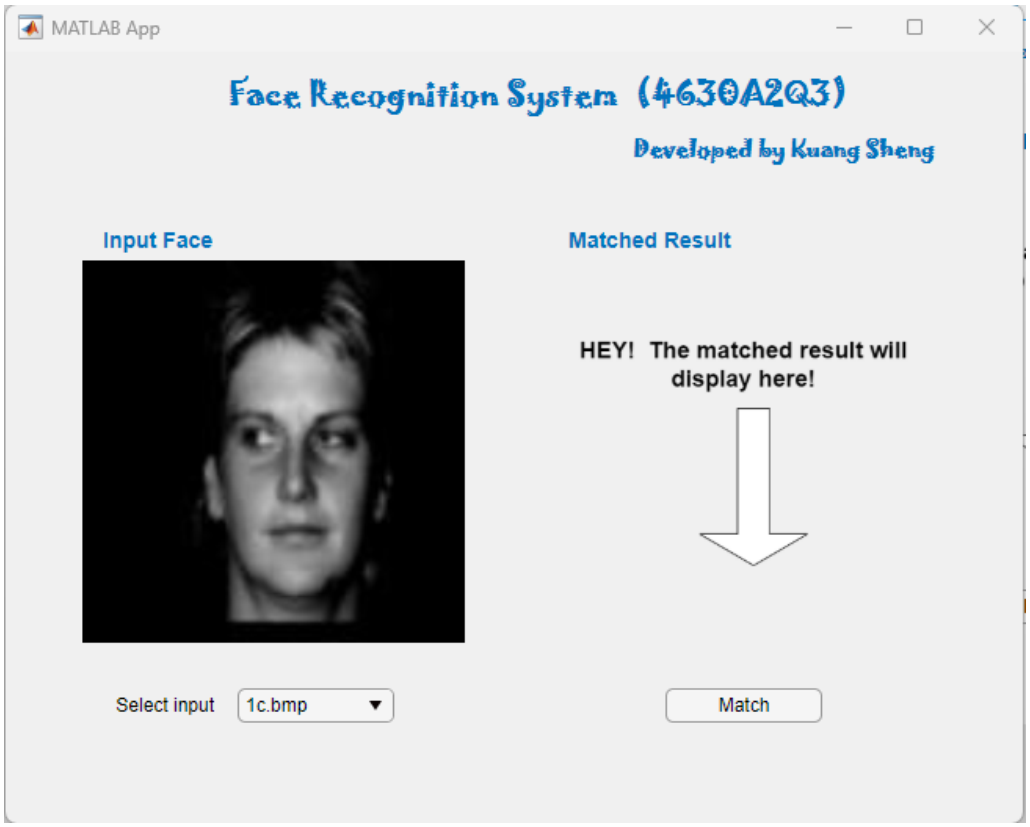
```
% Create SelectinputDropDown
app.SelectinputDropDown = uidropdown(app.UIFigure);
app.SelectinputDropDown.Items = {'1a.bmp', '1b.bmp', '1c.bmp', '1d.bmp', '1e.bmp', '1f.bmp', '1g.bmp', '2a.bmp', '2b.bmp', '2c.bmp', '2d.bmp',
app.SelectinputDropDown.ValueChangedFcn = createCallbackFcn(app, @SelectinputDropDownValueChanged, true);
app.SelectinputDropDown.Position = [149 65 100 22];
app.SelectinputDropDown.Value = '1a.bmp';
```



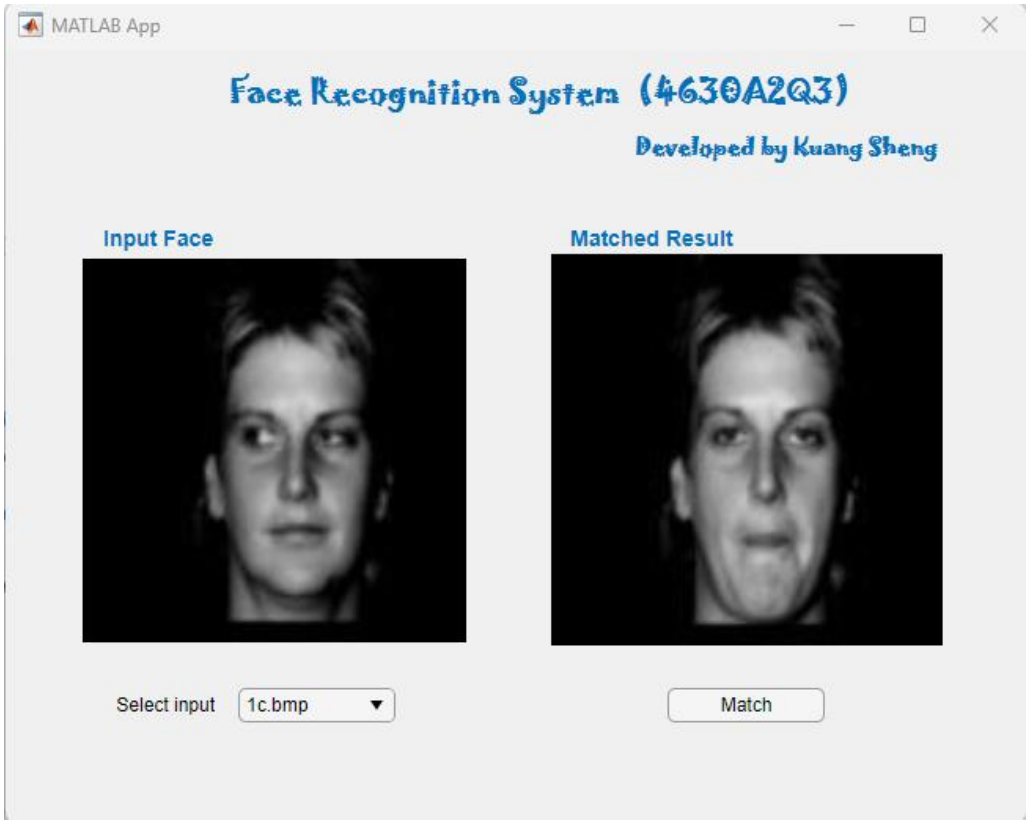
Once selected an input image, the 'ui.control.Image' component will have access to the path of the image and display it after converting.

```
% Value changed function: SelectinputDropDown
function SelectinputDropDownValueChanged(app, event)

     % Get the selected value from the DropDown
    value = app.SelectinputDropDown.Value;

    % Load the image from the 'images' folder
    currentFolder = fileparts(mfilename('fullpath'));
    imagePath = fullfile(currentFolder, value);

    % Read the image and update the displayed image
    [inp, map] = imread(imagePath);
    app.Image.ImageSource = ind2rgb(inp,map); % Convert grayscale image back to rgb
end
```

For example, selecting 1c.bmp as the input image:



Click 'Match' Button:



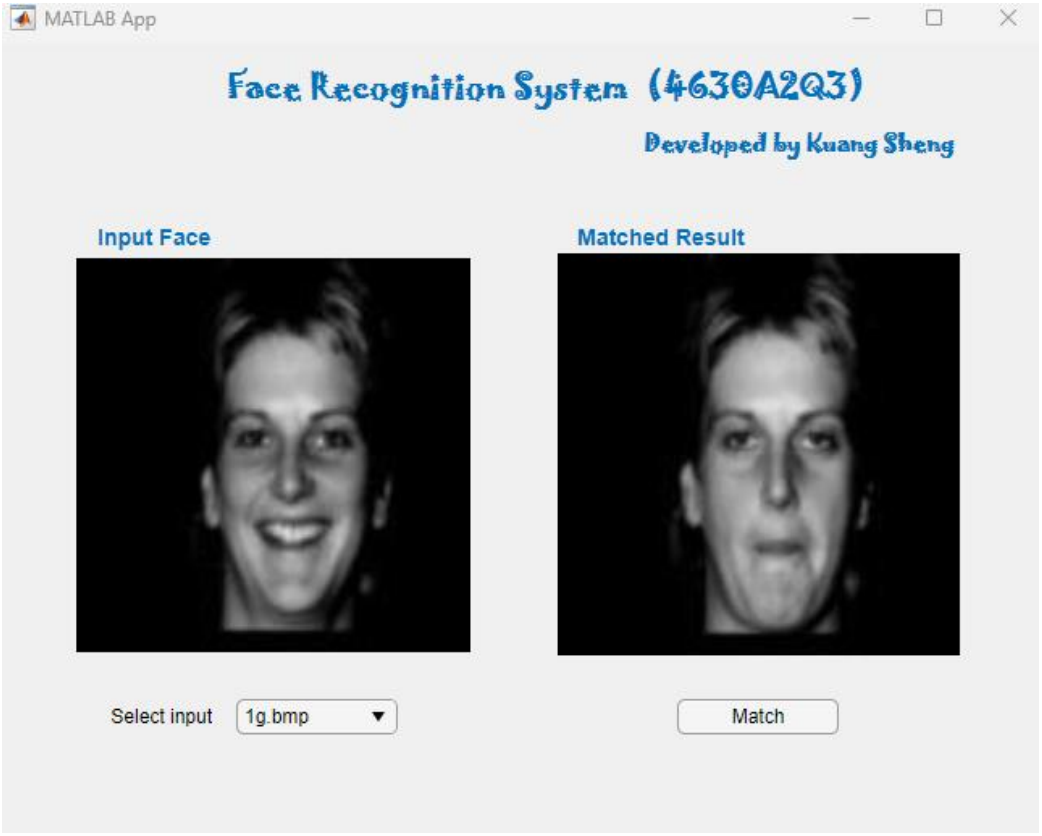
Since same approach, the call back function of the ‘Match’ button is pretty similar to the previous MATLAB Code (See both Appendix and *2. Methodology*).

After matching the person, the corresponding facial image will display on the ‘ui.control.Image2’ component.

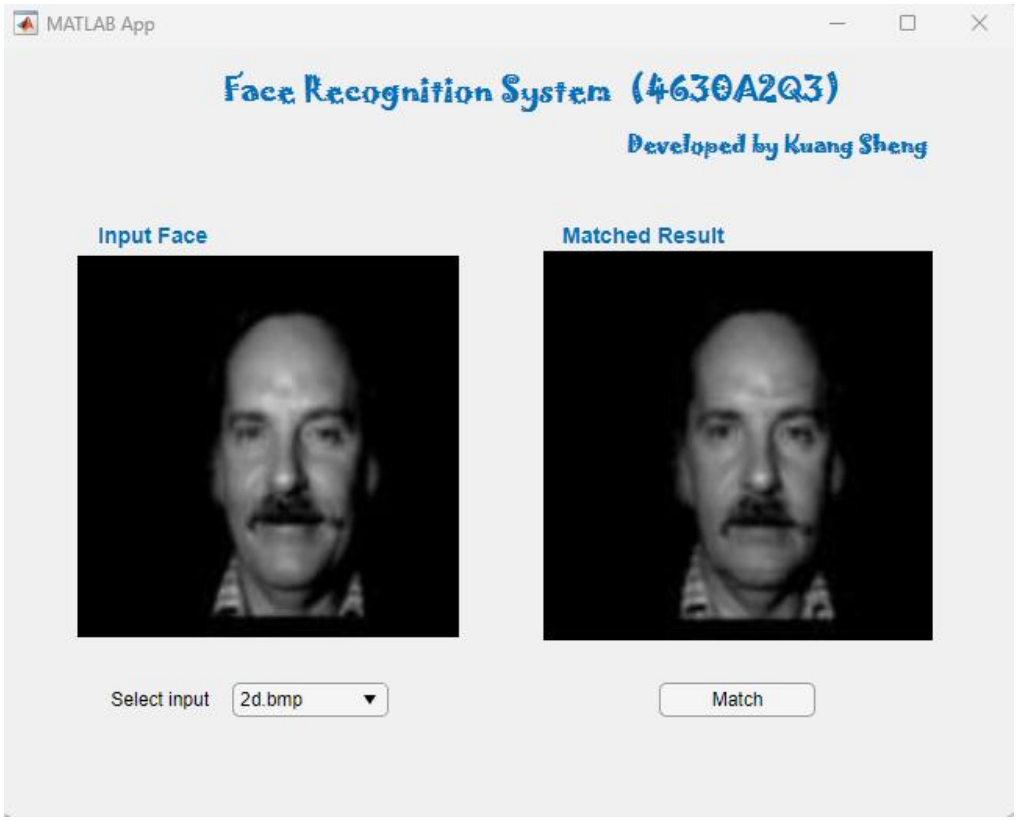
```
% Find the matched person from minimum Euclidean distance
[~, matched_person] = min(distances);
switch matched_person
    case 1
        [inp, map] = imread('eig/1a.bmp');
    case 2
        [inp, map] = imread('eig/2a.bmp');
    case 3
        [inp, map] = imread('eig/3A.BMP');
    case 4
        [inp, map] = imread('eig/4A.BMP');
    case 5
        [inp, map] = imread('eig/5A.BMP');
    case 6
        [inp, map] = imread('eig/6A.BMP');

end

%imagePath = fullfile('eig/%da.bmp',matched_person);

app.Image2.ImageSource = ind2rgb(inp,map);
```

All trials have been tested using the GUI interface. Here are some correct recognition examples using the last image of each person:

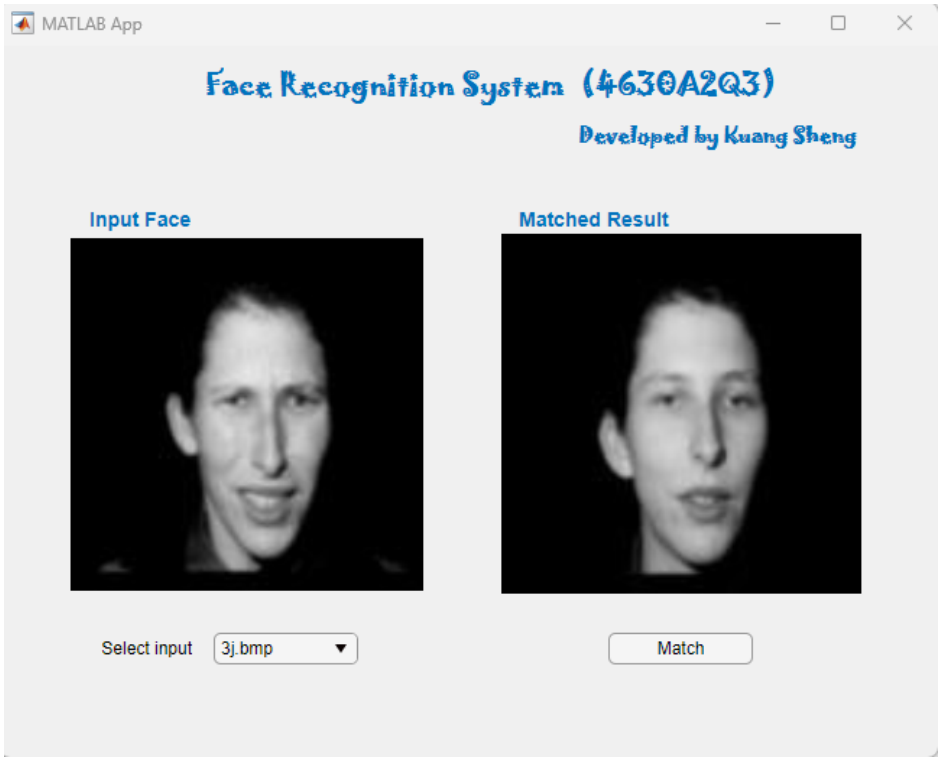
1g.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
1g.bmp
Match

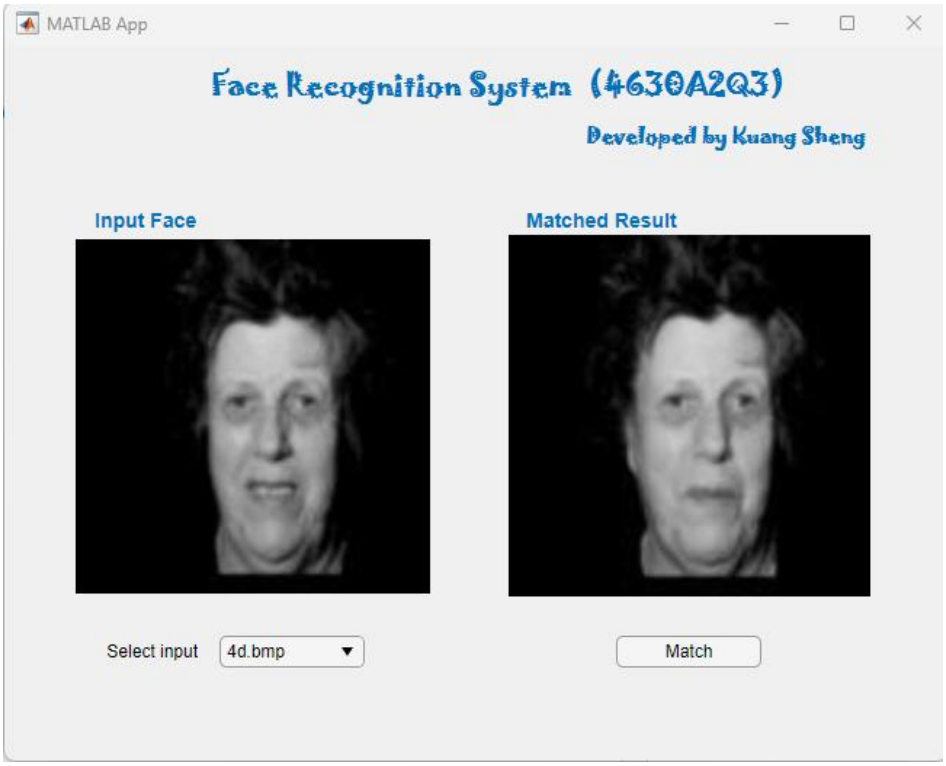
2d.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
2d.bmp
Match

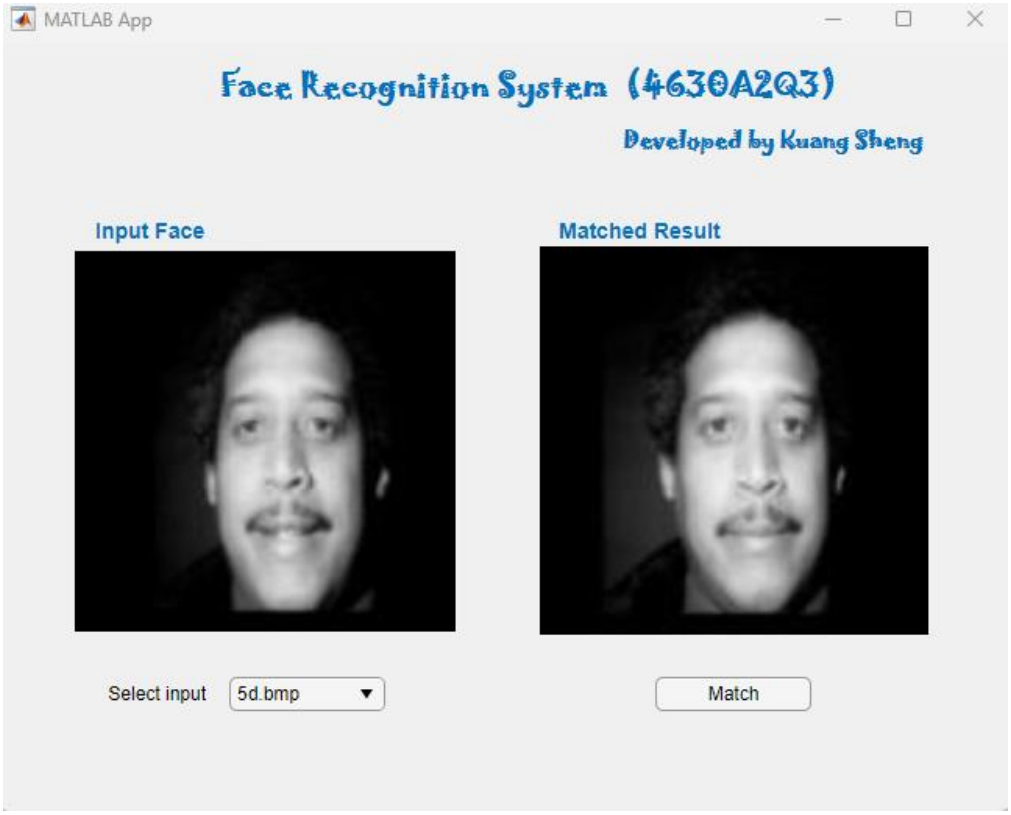
3j.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
3j.bmp
Match

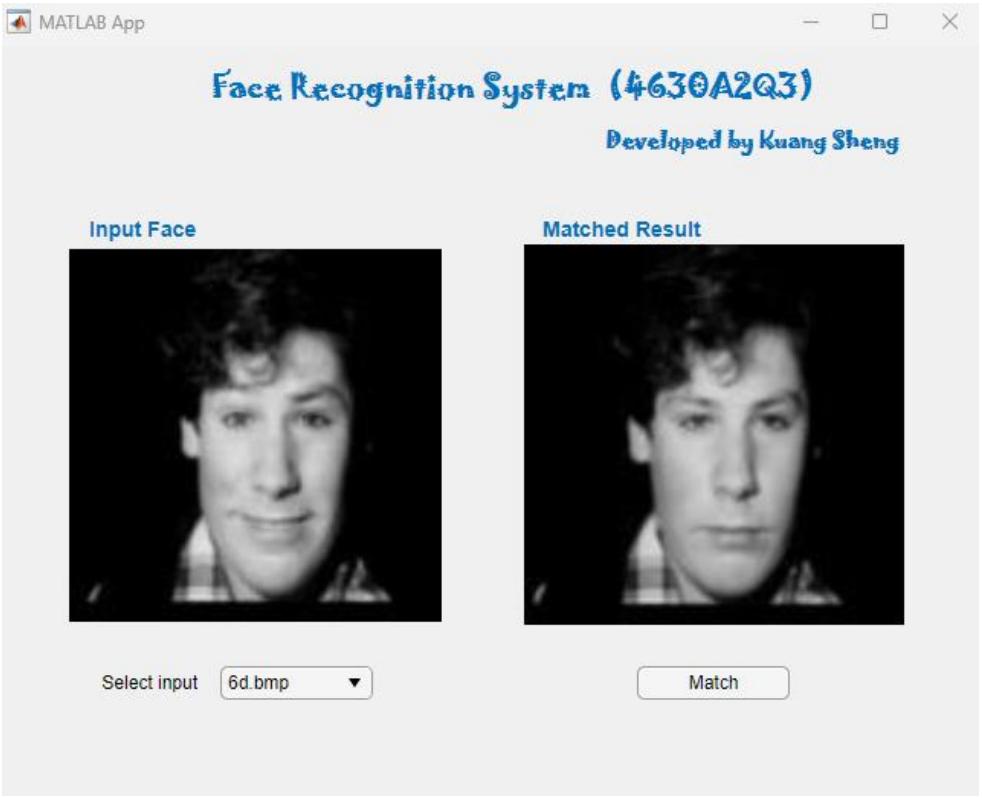
4d.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
4d.bmp
Match

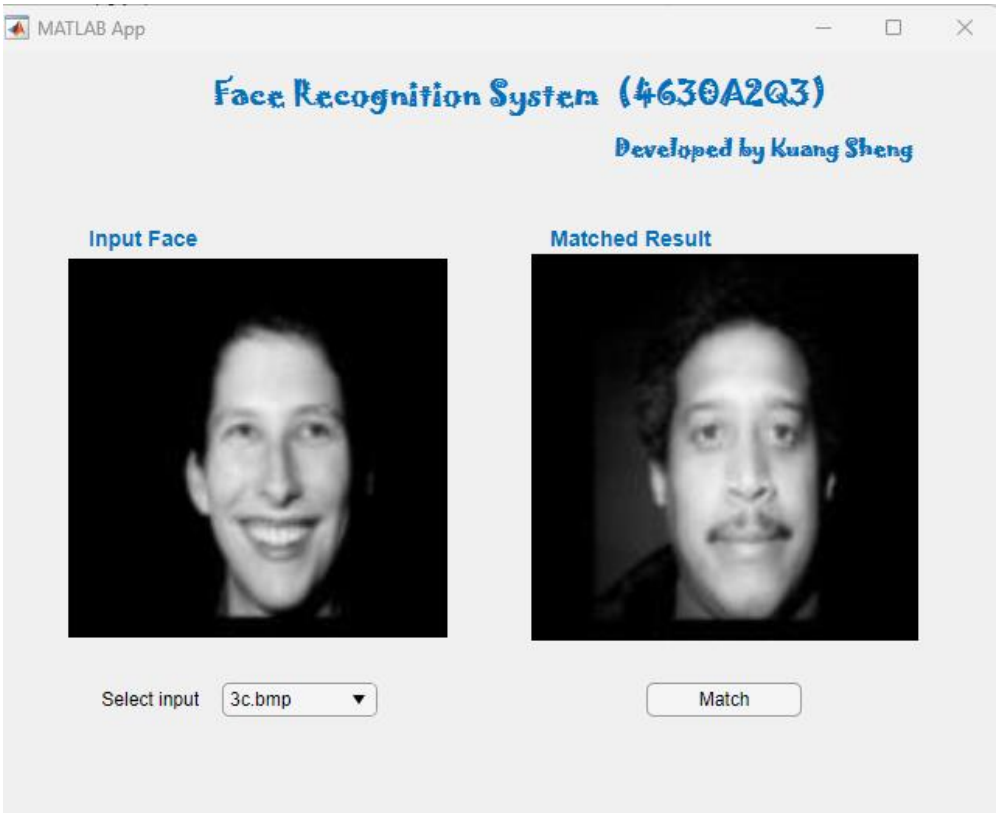
5d.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
5d.bmp
Match

6d.bmp:


MATLAB App
Face Recognition System (4630A2Q3)
Developed by Kuang Sheng
Input Face
Matched Result
Select input
6d.bmp
Match

Finally, the only incorrect case, 3c.bmp:



Citations:

[1] Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. Journal of cognitive neuroscience, 3(1), 71-86. Massachusetts Institute of Technology, Vision and Modeling Group.

[2] Lovell, B. C., & Chen, S. (2003). Robust face recognition for data mining. Intelligent Real-Time Imaging and Sensing Group EMI, School of ITEE, The University of Queensland.

Appendix

MATLAB Code

Q1. Thresholding based on Bimodal Grayscale Histogram

```
%% Threshholding based on Bimodal Histogram
clear, clc, close all;
mriImage = rgb2gray(imread('Cardiac Images/Cardiac MRI Left
ventricle0156.jpg'));
x = 200;
y = 110;
width = 150;
height = 140;
Image = imcrop(mriImage, [x y width height]);
figure, imshow(mriImage), title('Original Image (0000)');
figure, imshow(Image), title('Image after cropping');

figure, imhist(Image), title('Orignal grayscale histogram');
xlabel('Pixel Intensity');
ylabel('Piexel Count');


hist1 = imhist(Image);
hist2 = hist1;  %Make a copy of histogram
iter = 0;   %Number of iterations

while 1
    %Determine if it is bimodal histogram. If so, find the peak
    [is,peak] = Bimodal(hist1);
    if is == 0                      %If not, smooth the histogram
        hist2(1) = (hist1(1)*2+hist1(2))/3;
        for j = 2:255
            %Find the average point of three adjacent points
            hist2(j) = (hist1(j-1)+hist1(j)+hist1(j+1))/3;
        end
        hist2(256) = (hist1(255)+hist1(256)*2)/3;
        hist1 = hist2;
        iter = iter+1;
        if iter > 500
            break;
        end
    else
        break;
    end
end

[trough, pos] = min(hist1(peak(1):peak(2))); %Find the trough of
histogram
thresh = pos + peak(1);  %Find the threshold -100??

figure, stem(1:256,hist1,'Marker','none');
title('Smoothed grayscale histogram');
xlabel('Pixel Intensity');
ylabel('Piexel Count');
hold on
```

```
stem([thresh,thresh],[0,trough],'Linewidth',2);
hold off
result = zeros(size(Image));
result(Image>thresh) = 1;  %Binarize based on Threshold
% SE = strel('disk', 2);
% result = imopen(result, SE);
% result = imfill(result,'holes');
figure, imshow(result), title('Image after thresholding');

%% Determine if the histogram is bimodal, return peak value
function [is,peak]=Bimodal(histgram)
    count = 0;
    for j=2:255
        if histgram(j-1) < histgram(j) && histgram(j+1) <
histgram(j)
            count = count + 1;
            peak(count) = j;  %Record the location of peak
            if count > 2
                is = 0;
                return;
            end
        end
    end
    if count == 2
        is = 1;
    else
        is = 0;
    end
end
```

Q1. Morphology Approach

```
clear, clc, close all;
matlab.video.read.UseHardwareAcceleration('off');

vidObj = VideoReader('Cardiac MRI Left ventricle.mp4');


%% Analyze the frames
init_f = 1;  %Initial frame
last_f = 183;  %Last frame

vidframes = read(vidObj,[init_f last_f]);  %Read current frame

video = VideoWriter('./video.mp4', 'MPEG-4');
open(video);


% x = 180;
% y = 100;
% width = 180;
% height = 160;

x = 170;
y = 85;
width = 200;
height = 180;
area = [];

for i = init_f:last_f

    current_f = i + 1 - init_f;  %Current frame
    mriImage = vidframes(:,:,:,current_f);

    hImage = imcrop(mriImage, [x y width height]);  %???? 150 80 230
200area = [];

    %% Pre-process data
    I_gray = rgb2gray(histeq(hImage));
    I_gray = imadjust(I_gray,[0.4 0.6],[0 1]);
    I_filt = medfilt2(I_gray, [6 6]);
    %I_bw = imbinarize(I_filt);
    %I_bw = I_filt;

    %% Morphological Operation
    SE = strel('disk', 3);
    %SE = strel('line', 5,90);

    J = imopen(I_filt, SE);
    J = imfill(J,'holes');
    J = imclearborder(J);
    J = bwareaopen(J,100);

    %% Label and extract info
    label = bwlabel(J);
    stats = regionprops(label, 'Area','PixelIdxList');
```

```
    % Locate the largest connected region (left ventricle)
    [~, idx] = max([stats.Area]);
    bw = ismember(label, idx);



    %% Number of Pixels
    CC = bwconncomp(bw);
    num_pixels = numel(CC.PixelIdxList{1});

    % Smooth the boundary
    smoothed_boundary = bwconvhull(bw);

     % Find the boundries
    boundaries = bwboundaries(smoothed_boundary,'noholes');

    area = [area num_pixels];
    text = ['Frame: ' num2str(current_f),'    Cross_sectional Area:'
num2str(num_pixels,'%0.2f')];

    %% Outline the inner wall

    final_im = insertText(mriImage,[5
5],text,'FontSize',12,'TextColor','white');

    for k = 1:length(boundaries)
        boundary = [boundaries{1}];
        final_im = insertShape(final_im, 'Line', [boundary(:,2)+x,
boundary(:,1)+y, boundary(:,2)+1+x, boundary(:,1)+1+y], 'LineWidth',
5, 'Color', 'cyan');
    end

    writeVideo(video, final_im);

    if (mod(current_f, 20) == 0)
        subplot(3,3,current_f/20);
        imshow(final_im);
        title(sprintf('Frame: %d', current_f));
    end


end
close(video);
figure,plot(area);
xlabel('Time');
ylabel('Cross-sectional area of the left ventricle');
title('Area(Pixels) VS Time(Frames) using Morphology');
```

Q1. Snake Approach

```
%% Snake Algorithm
clear, clc, close all;

%Open Video
matlab.video.read.UseHardwareAcceleration('off');
vidObj = VideoReader('Cardiac MRI Left ventricle.mp4');

%% Analyze the frames
init_f = 1;  %Initial frame
last_f = 183;  %Last frame

vidframes = read(vidObj,[init_f last_f]);  %Read current frame

video = VideoWriter('./video.mp4', 'MPEG-4');
open(video);
area = [];
for i = init_f:last_f

    current_f = i + 1 - init_f;  %Current frame
    mriImage = vidframes(:,:,:,current_f);

    final_image = mriImage;

    x = 180;
    y = 100;
    width = 180;
    height = 160;
    %Crop the image for better extraction
    mriImage = imcrop(mriImage, [x y width height]);

    %% Pre process the image
    I_gray = rgb2gray(histeq(mriImage));
    I_filt = medfilt2(I_gray, [10 10]);
    I_filt = imadjust(I_filt,[0.4 0.6],[0 1]); %test
    I_filt = imclearborder(I_filt);
    %Create the initial mask covering ventricle
    mask = zeros(size(I_filt));
    mask(15:end-15,15:end-15) = 1;

    %Iterate 360 times using snake algorithm
    I_snake = activecontour(I_filt,mask,360);

    % Connected Region Analysis
    label = bwlabel(I_snake); %Label the regions
    stats = regionprops(label, 'Area',
'BoundingBox','PixelIdxList');

    % Find the largest connected region
    [~, idx] = max([stats.Area]);
    bw = ismember(label, idx);

    % Segmentation

    smoothed_boundary = bwconvhull(bw);
```

```
    boundaries = bwboundaries(smoothed_boundary,'noholes');
    %% Number of Pixels
    CC = bwconncomp(bw);
    num_pixels = numel(CC.PixelIdxList{1});
    area = [area num_pixels];
    text = ['Frame: ' num2str(current_f), '    Cross_sectional
Area:' num2str(num_pixels,'%0.2f')];

    %% Layout the boundries

    final_im = insertText(final_image,[5
5],text,'FontSize',12,'TextColor','white');

    for k = 1:length(boundaries)
        boundary = [boundaries{1}];
        final_im = insertShape(final_im, 'Line', [boundary(:,2)+x,
boundary(:,1)+y, boundary(:,2)+1+x, boundary(:,1)+1+y], 'LineWidth',
5, 'Color', 'red');
    end

    writeVideo(video, final_im);

    if (mod(current_f, 20) == 0)
        subplot(3,3,current_f/20);
        imshow(final_im);
        title(sprintf('Frame: %d', current_f));
    end

end

close(video);
figure,plot(area);
xlabel('Time');
ylabel('Cross-sectional area of the left ventricle');
title('Area(Pixels) VS Time(Frames) using Snake Algorithm');
```

Q2.

```
%% 3D Reconstruction CAN WORK
clear, clc, close all;

DinoImages = dir('*.jpg');
iImages = length(DinoImages);  %36
images = cell(iImages, 1);  %Create cell array to store images

% World Coordinates for dinosours, resolution = 2
% @Volume.txt
x = -180:2:90;
y = -80:2:70;
z = 20:2:460;

% Projection matrices
% @Dino projection matrices.txt
P0 = [1.134783 1.069317 0.046803 347.735102;
     -0.447199 0.330630 1.035582 -3.804233;
     0.000382 -0.000339 0.000072 1.000000];

P1 = [1.303228 0.856019 0.046803 347.735102;
     -0.382992 0.403262 1.035582 -3.804233;
     0.000318 -0.000400 0.000072 1.000000];

P2 = [1.432075 0.616711 0.046803 347.735102;
     -0.307148 0.463642 1.035582 -3.804233;
     0.000243 -0.000449 0.000072 1.000000];

P3 = [1.517410 0.358664 0.046803 347.735102;
     -0.221971 0.509934 1.035582 -3.804233;
     0.000162 -0.000484 0.000072 1.000000];

P4 = [1.556638 0.089720 0.046803 347.735102;
     -0.130050 0.540731 1.035582 -3.804233;
     0.000075 -0.000505 0.000072 1.000000];

P5 = [1.548569 -0.181950 0.046803 347.735102;
     -0.034177 0.555099 1.035582 -3.804233;
     -0.000014 -0.000511 0.000072 1.000000];

P6 = [1.493447 -0.448092 0.046803 347.735102;
     0.062734 0.552601 1.035582 -3.804233;
     -0.000102 -0.000500 0.000072 1.000000];

P7 = [1.392948 -0.700619 0.046803 347.735102;
     0.157739 0.533312 1.035582 -3.804233
     -0.000187 -0.000475 0.000072 1.000000];

P8 = [1.250125 -0.931858 0.046803 347.735102;
     0.247952 0.497819 1.035582 -3.804233;
     -0.000267 -0.000435 0.000072 1.000000];

P9 = [1.069317 -1.134783 0.046803 347.735102;
     0.330630 0.447199 1.035582 -3.804233;
     -0.000339 -0.000382 0.000072 1.000000];
```

```
P10 = [0.856019 -1.303228 0.046803 347.735102;
      0.403262 0.382992 1.035582 -3.804233;
      -0.000400 -0.000318 0.000072 1.000000];

P11 = [0.616711 -1.432075 0.046803 347.735102;
      0.463642 0.307148 1.035582 -3.804233;
      -0.000449 -0.000243 0.000072 1.000000];

P12 = [0.358664 -1.517410 0.046803 347.735102;
      0.509934 0.221971 1.035582 -3.804233;
      -0.000484 -0.000162 0.000072 1.000000];

P13 = [0.089720 -1.556638 0.046803 347.735102;
      0.540731 0.130050 1.035582 -3.804233;
      -0.000505 -0.000075 0.000072 1.000000];

P14 = [-0.181950 -1.548569 0.046803 347.735102;
      0.555099 0.034177 1.035582 -3.804233;
      -0.000511 0.000014 0.000072 1.000000];

P15 = [-0.448092 -1.493447 0.046803 347.735102;
      0.552601 -0.062734 1.035582 -3.804233;
      -0.000500 0.000102 0.000072 1.000000];

P16 = [-0.700619 -1.392948 0.046803 347.735102;
      0.533312 -0.157739 1.035582 -3.804233;
      -0.000475 0.000187 0.000072 1.000000];

P17 = [-0.931858 -1.250125 0.046803 347.735102;
      0.497819 -0.247952 1.035582 -3.804233;
      -0.000435 0.000267 0.000072 1.000000];

P18 = [-1.134783 -1.069317 0.046803 347.735102;
      0.447199 -0.330630 1.035582 -3.804233;
      -0.000382 0.000339 0.000072 1.000000];

P19 = [-1.303228 -0.856019 0.046803 347.735102;
      0.382992 -0.403262 1.035582 -3.804233;
      -0.000318 0.000400 0.000072 1.000000];

P20 = [-1.432075 -0.616711 0.046803 347.735102;
      0.307148 -0.463642 1.035582 -3.804233;
      -0.000243 0.000449 0.000072 1.000000];

P21 = [-1.517410 -0.358664 0.046803 347.735102;
      0.221971 -0.509934 1.035582 -3.804233;
      -0.000162 0.000484 0.000072 1.000000];

P22 = [-1.556638 -0.089720 0.046803 347.735102;
      0.130050 -0.540731 1.035582 -3.804233;
      -0.000075 0.000505 0.000072 1.000000];

P23 = [-1.548569 0.181950 0.046803 347.735102;
      0.034177 -0.555099 1.035582 -3.804233;
      0.000014 0.000511 0.000072 1.000000];
```

```
P24 = [-1.493447 0.448092 0.046803 347.735102;
      -0.062734 -0.552601 1.035582 -3.804233;
      0.000102 0.000500 0.000072 1.000000];

P25 = [-1.392948 0.700619 0.046803 347.735102;
      -0.157739 -0.533312 1.035582 -3.804233;
      0.000187 0.000475 0.000072 1.000000];

P26 = [-1.250125 0.931858 0.046803 347.735102;
      -0.247952 -0.497819 1.035582 -3.804233;
      0.000267 0.000435 0.000072 1.000000];

P27 = [-1.069317 1.134783 0.046803 347.735102;
      -0.330630 -0.447199 1.035582 -3.804233;
      0.000339 0.000382 0.000072 1.000000];

P28 = [-0.856019 1.303228 0.046803 347.735102;
      -0.403262 -0.382992 1.035582 -3.804233;
      0.000400 0.000318 0.000072 1.000000];

P29 = [-0.616711 1.432075 0.046803 347.735102;
      -0.463642 -0.307148 1.035582 -3.804233;
      0.000449 0.000243 0.000072 1.000000];

P30 = [-0.358664 1.517410 0.046803 347.735102;
      -0.509934 -0.221971 1.035582 -3.804233;
      0.000484 0.000162 0.000072 1.000000];

P31 = [-0.089720 1.556638 0.046803 347.735102;
      -0.540731 -0.130050 1.035582 -3.804233;
      0.000505 0.000075 0.000072 1.000000];

P32 = [0.181950 1.548569 0.046803 347.735102;
      -0.555099 -0.034177 1.035582 -3.804233;
      0.000511 -0.000014 0.000072 1.000000];

P33 = [0.448092 1.493447 0.046803 347.735102;
      -0.552601 0.062734 1.035582 -3.804233;
      0.000500 -0.000102 0.000072 1.000000];

P34 = [0.700619 1.392948 0.046803 347.735102;
      -0.533312 0.157739 1.035582 -3.804233;
      0.000475 -0.000187 0.000072 1.000000];

P35 = [0.931858 1.250125 0.046803 347.735102;
      -0.497819 0.247952 1.035582 -3.804233;
      0.000435 -0.000267 0.000072 1.000000];

P_Mat = cat(3, P0, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11,
P12, P13, P14, P15, P16, P17, P18, ...
     P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30,
P31, P32, P33, P34, P35);

% Create 3D meshgrid
[X, Y, Z] = meshgrid(x, y, z);
```

```
voxel = ones(numel(X), 1);

%% Read the images and store to the cell array
textureCoords = cell(iImages, 1);
for i = 1:iImages
    im_name = sprintf('dino%02d.jpg',i-1);
    images{i} = imread(im_name);

    % imageSizes(i, 1): width of the ith image; imageSizes(i, 2):
height of the ith image.
    imageSizes(i, :) = size(images{i}); % Get size of current image

     % Compute silhouette using thresholding
    colourDiff = images{i}(:,:,1) - images{i}(:,:,3);
    silhouette = colourDiff > 0; % Foreground pixels have positive
difference

    % Remove Noise
    silhouette = imclearborder(silhouette);
    silhouette = imfill(silhouette, 'holes');

    % Store silhouette in a new cell array
    Sil_images{i} = silhouette;

    %Project voxels onto 2D silhouette images
    %@Projection.txt
    Cam_Z = P_Mat(3,1,i) * X + P_Mat(3,2,i) * Y + P_Mat(3,3,i) * Z +
P_Mat(3,4,i);
    Cam_X = P_Mat(1,1,i) * X + P_Mat(1,2,i) * Y + P_Mat(1,3,i) * Z +
P_Mat(1,4,i);
    Cam_Y = P_Mat(2,1,i) * X + P_Mat(2,2,i) * Y + P_Mat(2,3,i) * Z +
P_Mat(2,4,i);

    Cam_X = round(Cam_X ./ Cam_Z);
    Cam_Y = round(Cam_Y ./ Cam_Z);

    %A binary mask that identifies the valid pixels within the image
dimensions.
    bwMask = find((Cam_X >= 1) & (Cam_X <= imageSizes(i, 2)) & ...
                  (Cam_Y >= 1) & (Cam_Y <= imageSizes(i, 1)));

    %Carve away invalid pixels
    Cam_X = Cam_X(bwMask);
    Cam_Y = Cam_Y(bwMask);

    % Create silhouette mask
    Indices = sub2ind([imageSizes(i, 1), imageSizes(i, 2)],
round(Cam_Y), round(Cam_X));
    Sil_mask = bwMask(silhouette(Indices) >= 1);

    % Store texture coordinates for valid points
    textureCoords{i} = [Cam_X(:), Cam_Y(:)];
    % Carve silhouette into voxels
    X = X(Sil_mask);
    Y = Y(Sil_mask);
    Z = Z(Sil_mask);
```

```
    voxel = voxel(Sil_mask);
end
%figure, imshow(Sil_images{20});


%% Generating V matrix to compute 3D coordinate of each voxel
% remove repetitions
Xu = unique(X);
Yu = unique(Y);
Zu = unique(Z);

% Create mesh grid for voxels
[voxelX,voxelY,voxelZ] = meshgrid(Xu, Yu, Zu);

% Voxelization: Convert into a mesh grid of voxels
V = zeros(size(voxelX));
N = numel(X);
for j = 1:N
    ix = (Xu == X(j));
    iy = (Yu == Y(j));
    iz = (Zu == Z(j));
    iz = flipud(iz); % Flip array in the up-down direction
    V(iy,ix,iz) = voxel(j); % Final V matrix
end

%% Create 3D figure from V matrix
map = jet(256); % 256 colours
figure;
%dino = patch(isosurface(voxelX,voxelY,voxelZ,V, 1e-4)); %Crate
patch obj
s = isosurface(voxelX,voxelY,voxelZ,V, 1e-4);
V1 = s.vertices;
V2 = V1;
F1 = s.faces;
F2 = flipud(F1); %reverse order
frontDino = patch('Vertices', V1, 'Faces', F1, 'FaceColor', [0.8 0.4
0], 'EdgeColor', 'none');
backDino = patch('Vertices', V2, 'Faces', F2, 'FaceColor', [0.8 0.4
0], 'EdgeColor', 'none');

% isonormals(voxelX,voxelY,voxelZ,V, dino); %Set the normals
isonormals(voxelX,voxelY,voxelZ,V, frontDino); %Set the normals
isonormals(voxelX,voxelY,voxelZ,V, backDino); %Set the normals


% Set colormap
colormap(map);

% set(dino,'EdgeColor','none');
title('Dinosaur with dark orange colour');
set(gcf,'Color',[0.42 0.45 0.7]);
set(gca,'Color',[0.42 0.45 0.7]);
axis off;
daspect([1,1,1]),axis tight;
view(3);
camlight('right')
```

```
% Set the viewpoint
% view(45,30); % adjust the viewpoint as needed
% % Save the figure as a PNG image
% print('my_dinosaur.png', '-dpng', '-r300');
```

Q3. Script

```
%% Test for face
clear, clc, close all;

% Define constants
success = 0;
num_persons = 6;
num_images = 33;
N = 16384; %Dimensions = 128*128
image_size = size(imread('1a.bmp'));
allFaces = struct2cell(dir('eig/*.bmp'));
allInputs = struct2cell(dir('./*.bmp'));
face_vectors = zeros(N, num_persons);

%% Vectorise the 6 training images
for num = 1:num_persons
    faceName = allFaces{1,num};
    [face, map] = imread(faceName);
    faceVector = reshape(face, [], size(face, 3)); % Face vector
    rgb = ind2rgb(face, map);
    gray = rgb2gray(rgb);  % Gray Images
    subplot(2,3,num);
    imshow(gray);
    face_vectors(:, num) = gray(:);
end

%% Calculate average face
avg_face = mean(face_vectors,2);
% Reshape the average face vector back to image
avg_face_image = reshape(avg_face, image_size);
figure, imshow(avg_face_image);
title('Average face of the set');

%% Standardise face vectors
face_diff = face_vectors - avg_face;

% Perform PCA to obtain the eigenfaces
% covariance matrix
C = face_diff' * face_diff;

% eigenvectors of covariance
[eVectors, eValues] = eig(C);

eigenvalues_vec = diag(eValues);

%% Calculate the eigenfaces
K = 6;
eigenface_basis = face_diff * eVectors(:, 1:6);
figure;
for k = 1:K
    eigenface = eigenface_basis(:, k);
    eigenface_2d = reshape(eigenface, image_size);
    subplot(1, K, k);
    imshow(eigenface_2d, []);
    title(['Eigenface ' num2str(k)]);
```

```
end


% Compute projection coefficients for each training image
projection_coeffs = eigenface_basis' * face_diff;

for num = 1:num_images
    inputName = allInputs{1,num};
    [input, map1] = imread(inputName);
    %% Reshape to gray image with 16384 Dimensions
    test_image_gray = rgb2gray(ind2rgb(input, map1));
    test_image_vector = test_image_gray(:);

    % Standardise image vectors
    test_image_diff = double(test_image_vector) - avg_face;

    % Project test image onto eigenface feature space
    test_coeffs = eigenface_basis' * test_image_diff;

    % Euclidean distance between the projection coefficients of the
test image and each of the training images.
    distances  = pdist2(test_coeffs', projection_coeffs',
'euclidean');
    % Find the matched person from minimum Euclidean distance
    [~, matched_person] = min(distances);

    %% Determine the success rate
    fprintf('Input image %s matched to person %d\n',
allInputs{1,num}, matched_person);
    if str2num(allInputs{1,num}(1)) == matched_person
       success = success+1;
    end

end

 fprintf('Successful Rate is %.2f%%\n', success*100/num_images);
```

Q3. GUI -> Code View

```
classdef Q3GUI < matlab.apps.AppBase


% Properties that correspond to app components
properties (Access = public)
UIFigure matlab.ui.Figure
SelectinputDropDownLabel matlab.ui.control.Label
SelectinputDropDown matlab.ui.control.DropDown
MatchButton matlab.ui.control.Button
Image matlab.ui.control.Image
Image2 matlab.ui.control.Image
InputFaceLabel matlab.ui.control.Label
MatchedResultLabel matlab.ui.control.Label
FaceRecognitionSystem4630A2Q3Label matlab.ui.control.Label
DevelopedbyKuangShengLabel matlab.ui.control.Label
end




% Callbacks that handle component events
methods (Access = private)


% Value changed function: SelectinputDropDown
function SelectinputDropDownValueChanged(app, event)
% Get the selected value from the DropDown
value = app.SelectinputDropDown.Value;
% Load the image from the 'images' folder
currentFolder = fileparts(mfilename('fullpath'));
imagePath = fullfile(currentFolder, value);
% Read the image and update the displayed image
[inp, map] = imread(imagePath);
app.Image.ImageSource = ind2rgb(inp,map); % Convert grayscale image back
to rgb
end


% Button pushed function: MatchButton
function MatchButtonPushed(app, event)
% Define constants
num_persons = 6;
N = 16384; %Dimensions = 128*128
image_size = size(imread('1a.bmp'));
allFaces = struct2cell(dir('eig/*.bmp'));
face_vectors = zeros(N, num_persons);
%% Vectorise the 6 training images
for num = 1:6
faceName = allFaces{1,num};
[face, map] = imread(faceName);
```

```
rgb = ind2rgb(face, map);
gray = rgb2gray(rgb); % Gray Images
face_vectors(:, num) = gray(:);
end
avg_face = mean(face_vectors,2);
avg_face_image = reshape(avg_face, image_size);
face_diff = face_vectors - avg_face;
% covariance matrix
C = face_diff' * face_diff;
% eigenvectors of covariance
[eVectors, eValues] = eig(C);
eigenvalues_vec = diag(eValues);
%% Calculate the eigenfaces
eigenface_basis = face_diff * eVectors(:, 1:6);
projection_coeffs = eigenface_basis' * face_diff;
test_image_gray = rgb2gray(app.Image.ImageSource);
test_image_vector = test_image_gray(:);
% Standardise image vectors
test_image_diff = double(test_image_vector) - avg_face;
% Project test image onto eigenface basis
test_coeffs = eigenface_basis' * test_image_diff;
% Euclidean distance between the projection coefficients of the test image
and each of the training images.
distances = pdist2(test_coeffs', projection_coeffs', 'euclidean');
% Find the matched person from minimum Euclidean distance
[~, matched_person] = min(distances);
switch matched_person
case 1
[inp, map] = imread('eig/1a.bmp');
case 2
[inp, map] = imread('eig/2a.bmp');
case 3
[inp, map] = imread('eig/3A.BMP');
case 4
[inp, map] = imread('eig/4A.BMP');
case 5
[inp, map] = imread('eig/5A.BMP');
case 6
[inp, map] = imread('eig/6A.BMP');
end
%imagePath = fullfile('eig/%da.bmp',matched_person);


app.Image2.ImageSource = ind2rgb(inp,map);


end
end


% Component initialization
methods (Access = private)
```

```
% Create UIFigure and components
function createComponents(app)


% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Position = [100 100 649 492];
app.UIFigure.Name = 'MATLAB App';


% Create SelectinputDropDownLabel
app.SelectinputDropDownLabel = uilabel(app.UIFigure);
app.SelectinputDropDownLabel.HorizontalAlignment = 'right';
            app.SelectinputDropDownLabel.Position = [66 65 68 22];
            app.SelectinputDropDownLabel.Text = 'Select input';


            % Create SelectinputDropDown
            app.SelectinputDropDown = uidropdown(app.UIFigure);
            app.SelectinputDropDown.Items = {'1a.bmp', '1b.bmp', '1c.bmp',
'1d.bmp', '1e.bmp', '1f.bmp', '1g.bmp', '2a.bmp', '2b.bmp', '2c.bmp',
'2d.bmp', '3a.bmp', '3b.bmp', '3c.bmp', '3d.bmp', '3e.bmp', '3f.bmp',
'3g.bmp', '3h.bmp', '3i.bmp', '3j.bmp', '4a.bmp', '4b.bmp', '4c.bmp',
'4d.bmp', '5a.bmp', '5b.bmp', '5c.bmp', '5d.bmp', '6a.bmp', '6b.bmp',
'6c.bmp', '6d.bmp'};
            app.SelectinputDropDown.ValueChangedFcn =
createCallbackFcn(app, @SelectinputDropDownValueChanged, true);
            app.SelectinputDropDown.Position = [149 65 100 22];
            app.SelectinputDropDown.Value = '1a.bmp';


            % Create MatchButton
            app.MatchButton = uibutton(app.UIFigure, 'push');
            app.MatchButton.ButtonPushedFcn = createCallbackFcn(app,
@MatchButtonPushed, true);
            app.MatchButton.Position = [422 65 100 22];
            app.MatchButton.Text = {'Match'; ''};


            % Create Image
            app.Image = uiimage(app.UIFigure);
            app.Image.Position = [50 114 244 249];
            app.Image.ImageSource = 'input1.png';


            % Create Image2
            app.Image2 = uiimage(app.UIFigure);
            app.Image2.Position = [344 114 256 249];
            app.Image2.ImageSource = 'output1.png';


            % Create InputFaceLabel
            app.InputFaceLabel = uilabel(app.UIFigure);
            app.InputFaceLabel.FontSize = 14;
```

```
            app.InputFaceLabel.FontWeight = 'bold';
            app.InputFaceLabel.FontColor = [0 0.4471 0.7412];
            app.InputFaceLabel.Position = [63 362 87 22];
            app.InputFaceLabel.Text = {'Input Face'; ''};


            % Create MatchedResultLabel
            app.MatchedResultLabel = uilabel(app.UIFigure);
            app.MatchedResultLabel.FontSize = 14;
            app.MatchedResultLabel.FontWeight = 'bold';
            app.MatchedResultLabel.FontColor = [0 0.4471 0.7412];
            app.MatchedResultLabel.Position = [360 362 109 22];
            app.MatchedResultLabel.Text = {'Matched Result'; ''};


            % Create FaceRecognitionSystem4630A2Q3Label
            app.FaceRecognitionSystem4630A2Q3Label =
uilabel(app.UIFigure);
            app.FaceRecognitionSystem4630A2Q3Label.FontName = 'Jokerman';
            app.FaceRecognitionSystem4630A2Q3Label.FontSize = 20;
            app.FaceRecognitionSystem4630A2Q3Label.FontWeight = 'bold';
            app.FaceRecognitionSystem4630A2Q3Label.FontColor = [0 0.4471
0.7412];
            app.FaceRecognitionSystem4630A2Q3Label.Position = [143 452 413
33];
            app.FaceRecognitionSystem4630A2Q3Label.Text = 'Face
Recognition System  (4630A2Q3) ';


            % Create DevelopedbyKuangShengLabel
            app.DevelopedbyKuangShengLabel = uilabel(app.UIFigure);
            app.DevelopedbyKuangShengLabel.FontName = 'Jokerman';
            app.DevelopedbyKuangShengLabel.FontSize = 14;
            app.DevelopedbyKuangShengLabel.FontWeight = 'bold';
            app.DevelopedbyKuangShengLabel.FontColor = [0 0.4471 0.7412];
            app.DevelopedbyKuangShengLabel.Position = [401 421 199 23];
            app.DevelopedbyKuangShengLabel.Text = 'Developed by Kuang
Sheng';


            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end


    % App creation and deletion
    methods (Access = public)


        % Construct app
        function app = Q3GUI
```

```
            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```