ELEC4630 Assessment Task 2

Part 1: Location of the larger straight lines a supplied image

The lines in the image were found using the following code. It's a relatively simple process aided greatly by the hough() function included in MATLAB. The image is converted to grayscale, blurred to avoid destructive noise and then edge detected with a Sobel filter. The hough transform is then found and that analysed to find the 4 highest points of coincidence. These points are then converted back into lines and the lines drawn on the image.

```
% s4262468's Solution to Q1 Assignment2 of ELEC4630
close, clear, clc

i = imread('tajmahal2004.jpg');
ig = rgb2gray(i);

G = fspecial('gaussian',[5 5],2);
igb = imfilter(ig,G,'same');

ic = edge(igb,'canny');
is = edge(igb, 'sobel');

t = imfilter(double(is), G, 'same');

% find hough transform
[H,T,R] = hough(t,'RhoResolution',1,'Theta',-80:0.2:-40);
```

```
17.
18. % Find the highest values in the transform and return their points
19. P = houghpeaks(H,4);
20. % Which is similar to:
21. % Htemp = H; P = [];
22. % for k = 1:4,
23. %     [x,y] = ind2sub(size(H),find(Htemp==max(Htemp(:)))); % find the best val
24. %     P(k,:) = [x y]; radius = round(size(H)/50); % store and make radius
25. %     Htemp(x-radius(1):x+radius(1), y-radius(2):y+radius(2)) = 0; % don't look near here ag
    ain.
26. % end
27.
28. %find the lines that are made from the points in the hough transform
29. lines = houghlines(t,T,R,P, 'FillGap', 30, 'MinLength', 200);
30.
31. figure, imshow(i), hold on
32. max_len = 0;
33. for k = 1:length(lines)
34.     xy = [lines(k).point1; lines(k).point2];
35.    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
36. end
```

Part 2:

2.1 *Viterbi Implementation*

The second part of this assignment was quite difficult as it was attempted using an adapted Viterbi Algorithm to find the outline of targeted part of the heart. This method is loosely based on the literature that was recommended in the assignment brief.

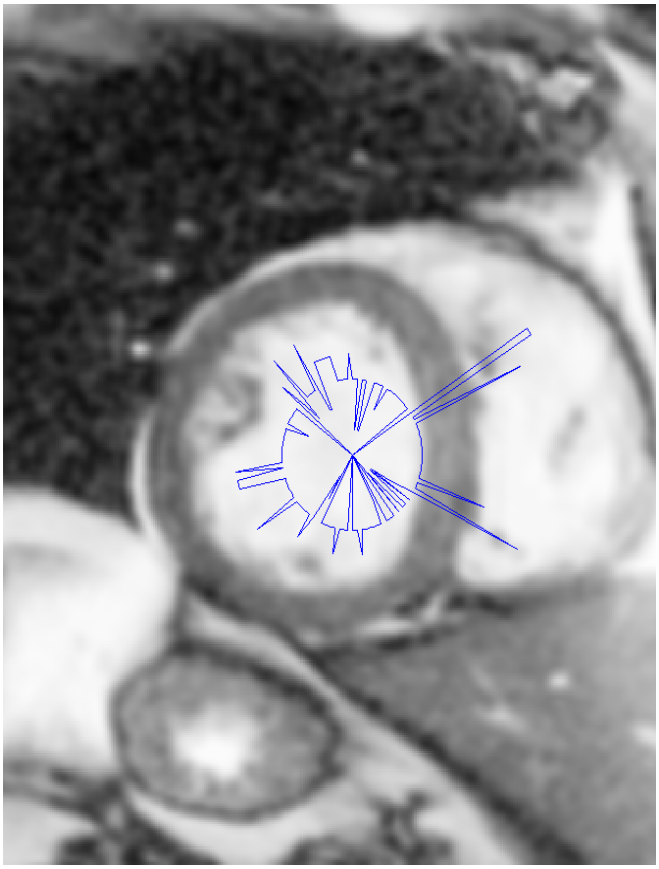
Firstly, the creation and transformation of a selection of the images pixels into a grid or trellis like formation was performed. The variables cirSplit and radPtNum set the radial distance between trellis states and how many points there are in each state respectively. The speed of the code is heavily based on the magnitude of these numbers as increasing either increases the necessary iterations of the algorithm.

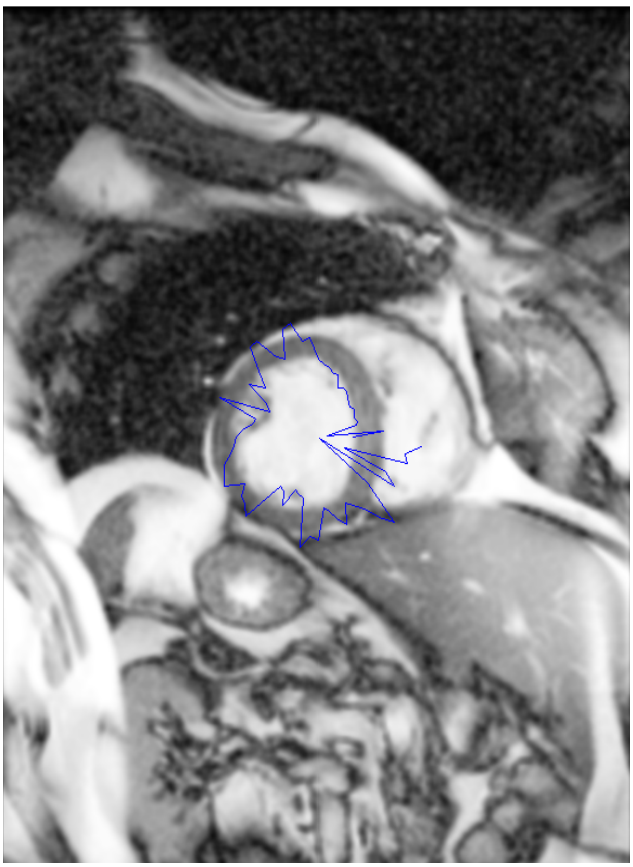
Once the trellise is made the pathfinding starts with the creation of the weightings for both the beginning and the end of the trellis. To ensure that the path found through the trellis is enclosed the trellis is iterated over radPtNum times. Each time the path is forced to start and end at the same depth, starting at 1 and going upwards. Once the best path for each of these situations has been found the best from each of these is taken out and used as the best path through the trellis.

The vcost() function is what was used to determine the cost of a move between 2 points. This was probably the failing point of my implementation, although it did produce not terrible results. Shown believe are the outputs of the script using 2 different cost functions. The top and bottom left images show the same cost function in both trellis and final forms while the bottom right picture shows one of my later and improved cost functions.

The cost between 2 points starts at 0 and increased in value by undesirable properties. These properties included being close to the top of the trellis, having a large gap between the points, having a large change in value between the points and having a large change in gradient between the line made by the current point and it's precursor and the line made by the current 2 points of the function. Additionally, if the gradient

between the 2 lines is greater than a quarter of depth of the trellis, the cost is made unreasonable in order to discount that point entirely.





2.2 *Code used for part 2.1*

A2p2.m - The main script for this attempted solution

```
% s4262468's solution to Q2 of ELEC4620's Assignment2
close, clear, clc

% import all them images
mri = {};
mri{1} = imread('MRI1_01.png');
mri{2} = imread('MRI1_02.png');
mri{3} = imread('MRI1_03.png');
mri{4} = imread('MRI1_04.png');
mri{5} = imread('MRI1_05.png');
mri{6} = imread('MRI1_06.png');
mri{7} = imread('MRI1_07.png');
mri{8} = imread('MRI1_08.png');
mri{9} = imread('MRI1_09.png');
mri{10} = imread('MRI1_10.png');
mri{11} = imread('MRI1_11.png');
mri{12} = imread('MRI1_12.png');
mri{13} = imread('MRI1_13.png');
```

```
mri{14} = imread('MRI1_14.png');
mri{15} = imread('MRI1_15.png');
mri{16} = imread('MRI1_16.png');

%% Make the Trellis
h = {};
for j = 1:16,
    h{j} = histeq(mri{j});
    G = fspecial('gaussian',[7 7],5);
    h{j} = imfilter(h{j},G,'same');
end

mid = round(size(h{1})/2);
mid = [mid(2), mid(1)];
cirSize = 100;
% Make the co-ordinates around the circle
cirx = [];
ciry = [];
for deg = 0:15:360,
    cirx = [cirx; (mid(1)+cirSize*cosd(deg))];
    ciry = [ciry; (mid(2)+cirSize*sind(deg))];
end

% Make the inside co-ords
fillx = [];
filly = [];
numpoints = 30;
for j = 1:size(cirx),
    tempx = []; tempy = [];
    [tempx,tempy] = fillline(mid, [cirx(j), ciry(j)], numpoints);
    fillx = [fillx;tempx];
    filly = [filly;tempy];
end

% Lets show the trellis
img = h{1};
trel = []; % A collection of all the data :D

for i = 1:size(cirx),
    for j = 1:size(fillx,2),
        c = fillx(i,j);
        r = filly(i,j);
        b = img(floor(r),floor(c));
        trel(j,i) = b;
    end
end

trel = struct('d',trel,'l',size(trel,1),'w',size(trel,2));
trel.e = edge(uint8(trel.d),'canny');
% imshow(imresize(uint8(trel.d),4))

%% Find our way through the Trellis
paths = [];
vals = [];
```

```
for arb = 1:trel.l,
    % Make the start and end costs first
    startv = []; endv = [];
    for i = 1:trel.l,
        t = vcost(trel.d, [arb,1], [i,2]);
        startv(i,1) = t;

        t = vcost(trel.d, [i,trel.w], [arb,1]);
        endv(i,1) = t;
    end

    % find the best solution for the last row back to arb
    startp(1:trel.l,1) = arb;
    endp(1:trel.l,1) = 0;
    endp(arb,1) = find(endv==min(endv),1,'last');

    % Traverse the trellis and assign values and parents
    trel.p = [startp]; trel.v = [startv];
    for col = 3:trel.w,
        s = trel.d(:,col); % our current column
        for curr = 1:trel.l, % for each point in our column
            minVal = vcost(trel.d, [1 col-1], [curr col], trel.p);
            minPath = 1;
            for prev = 2:trel.l, % look at each point in the last
                val = vcost(trel.d, [prev col-1], [curr col], trel.p);
                if val <= minVal, minVal = val; minPath = prev; end
            end
                trel.p(curr,col-1) = minPath;
                trel.v(curr,col-1) = minVal;
            end
        end
        trel.p = [trel.p endp]; % throw the final values to return home on the end
        trel.v = [trel.v endv];

        % Find the total cost
        totalv = 0;
        col = trel.w;
        curr = arb;
        while col >= 1, % work your way back through trellis
            paths(arb, col) = trel.p(curr,col);
            totalv = totalv + trel.v(curr,col);
            curr = trel.p(curr,col); col=col-1;
        end
        vals(arb) = totalv;
    end

    % find the best path by looking through totalv
    best = find(vals==min(vals),1,'first');

    %% Show the things we found
    figure(1), imshow(uint8(trel.d)), hold on;
    for i = 1:trel.w,
        plot(i,paths(best,i), 'xr');
    end
```

```
127.    hold off;
128.
129.    % draw on the original image
130.    figure(2), imshow(img), hold on;
131.    for i = 2:trel.w,
132.        pt1 = [fillx(i-1, paths(best,i-1)) filly(i-1, paths(best,i-1))];
133.        pt2 = [fillx(i, paths(best,i)) filly(i, paths(best,i))];
134.        x = [pt1(1) pt2(1)];
135.        y = [pt1(2) pt2(2)];
136.        plot(x(1), y(1), '*b'); % plot the point
137.        line(x,y); % plot a line between the points
138.    end
139.    plot(x(2), y(2), '*b'); % print last star
140.    plot(mid(1), mid(2), '*k'); % mark the center
141.
142.    for j = 1:trel.w, plot(cirx(j), ciry(j),'xr'); end % print circum
143.    hold off;
```

vcost.m

```
1.  function [ cost ] = vcost(trel, cur, point, path)
2.  % Calculates the cost to move from cur to point in trel
3.
4.      skip = 0;
5.      if nargin < 4, skip = 1; else path = trel.p; end
6.
7.      cost = 0;
8.
9.      % points close to the center have a higher weight
10.     cost = cost + (point(1)-size(trel,1))*10;
11.
12.     % if there is a large gap between points it is undesirable
13.     gapweight = 5;
14.     cost = cost + gapweight * abs(cost(1) - point(1));
15.
16.     % if there is a change in value then it is undesirable
17.     valweight = 10;
18.     cost = cost + valweight * abs(trel(cur(1), cur(2)) - trel(point(1), point(2)))^2;
19.
20.     %% Gradiant Cost Calc
21.     % smaller change in gradiant means smaller cost.
22.     % if deltagrad is too large make the cost obscene
23.     if skip ~= 1,
24.         gradweight = 2;
25.         prev = [path(cur(1),cur(2)-1) cur(2)-1]; %first cur(2)-1 is due to path's offset
26.         grad1 = (cur(1)-prev(1))/(cur(2)-prev(2));
27.         grad2 = (point(1)-cur(1))/(point(2)-cur(2));
28.         deltaGrad = grad2 - grad1;
29.         if deltaGrad ~= 0,
30.             cost = cost +  gradweight * abs(deltaGrad/grad1)*100;
31.         end
32.         if abs(deltaGrad) > size(trel,1)/4, % we're jumping 1/4 of the trellis :(
33.             cost = 100000000000000;
34.         end
35.     end
```

```
36.
37. end
```

fillline.m - Sourced from the matlab knowledge base. This function returns an x and y matrix with N points and equal spacing between the points used as it's first 2 arguments

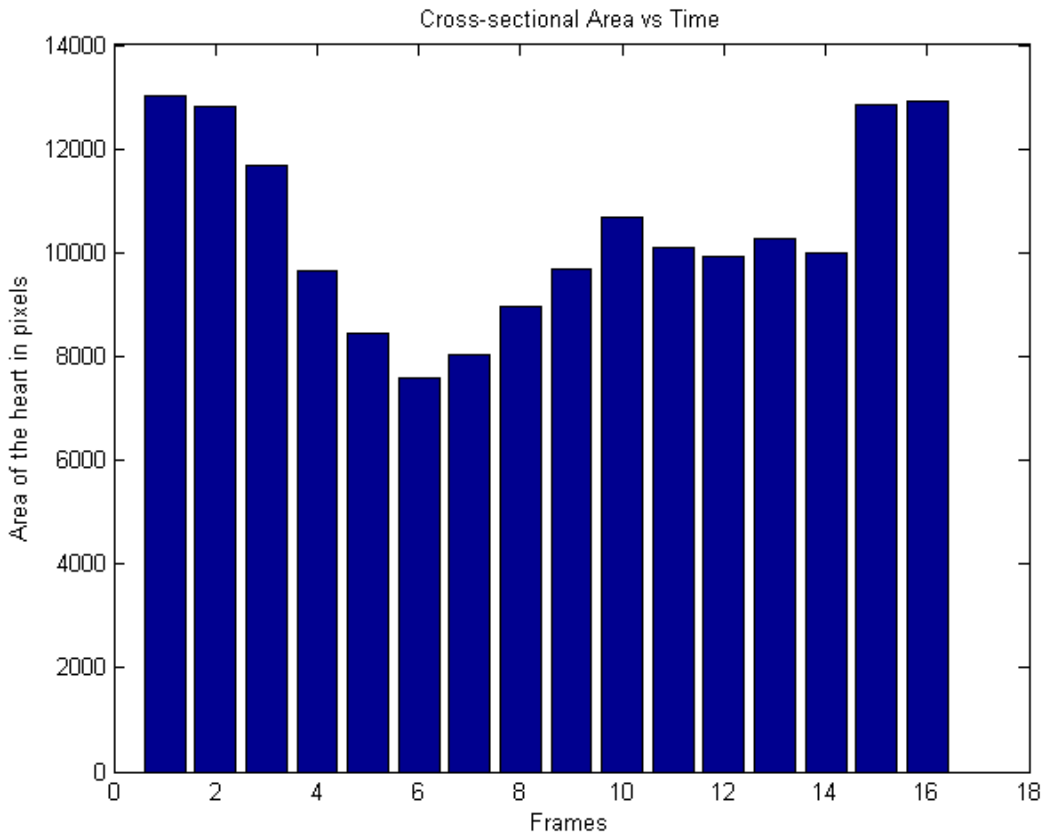
2.3 *Morphological Approach*

Unfortunately I could not use the Viterbi approach to produce results any better than those shown above. As the main portion of what we are doing here is basic segmentation of a similarly coloured area I chose to investigate how I might solve the problem using morphological techniques.

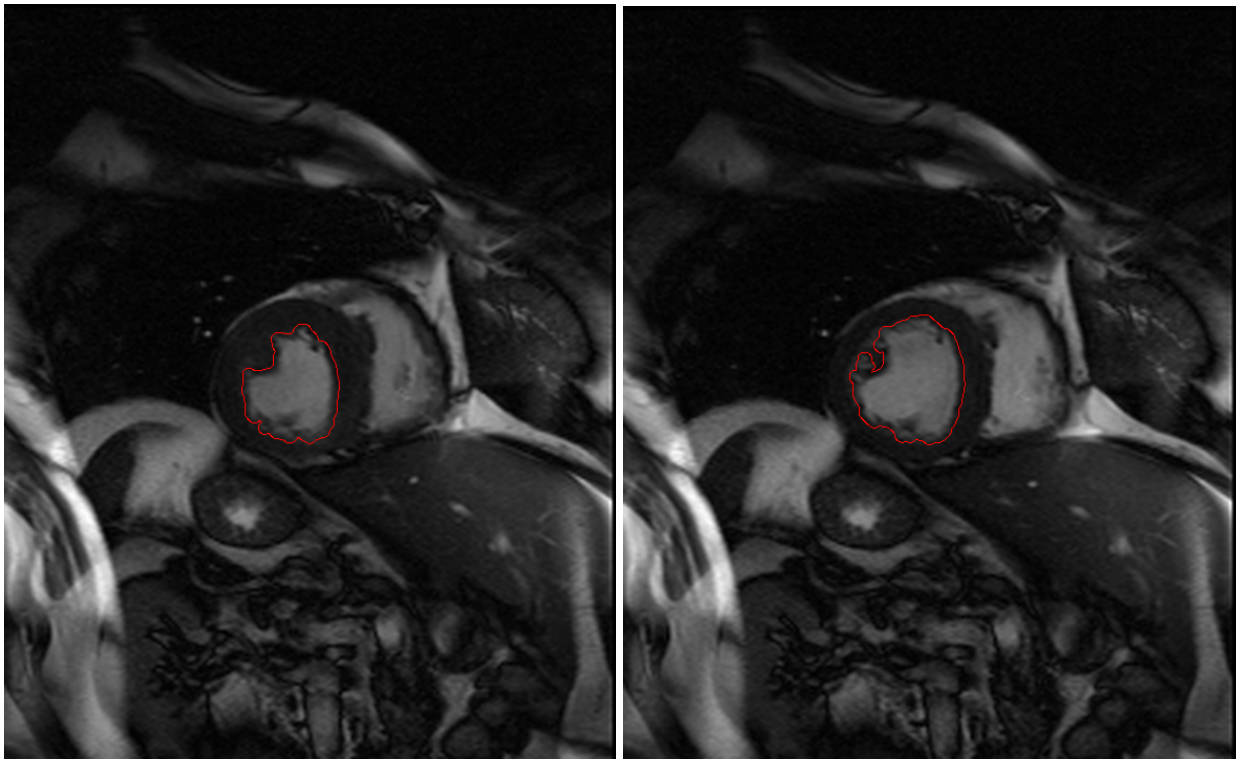
Starting with the assumption that the middle of the image would be inside the interested region I took its value. I then used the the values both above and below the one from the centre to create a black and white thresholded image that contained the interested region as well as others. From here it's not difficult to use the bwlabel() function to determine which region is the one I was looking for. The region is then cleaned up slightly by using imfill() and imclose() with a ball shaped element that provides a softer closing than a simple disc element.

I found the area of the region by checking the size of the return of find(). Alternatively I could have used regionprops() but I wanted to show that it was just as possible using less complex MATLAB functions. To make the outline, edge detection was performed on the region and each of the points in the detected edge image was then coloured red in a colour conversion of the original picture.

A gif of the resulting images can be found at: imgur.com/7zIt423 and the graph below maps the cross sectional area of the heart in pixel values over time.



Images generated using the following code:



2.4 *Code used for part 2.3*

```
% s4262468's 'Easymode' solution for Part2
clear, close all, clc

% import all them images
mri = {};
mri{1} = imread('MRI1_01.png');
mri{2} = imread('MRI1_02.png');
mri{3} = imread('MRI1_03.png');
mri{4} = imread('MRI1_04.png');
mri{5} = imread('MRI1_05.png');
mri{6} = imread('MRI1_06.png');
mri{7} = imread('MRI1_07.png');
mri{8} = imread('MRI1_08.png');
mri{9} = imread('MRI1_09.png');
mri{10} = imread('MRI1_10.png');
mri{11} = imread('MRI1_11.png');
mri{12} = imread('MRI1_12.png');
mri{13} = imread('MRI1_13.png');
mri{14} = imread('MRI1_14.png');
mri{15} = imread('MRI1_15.png');
mri{16} = imread('MRI1_16.png');

areas = [];
for j = 1:16,
    i = mri{j};

    % get value of center of image.
    mid = round(size(mri{1})/2);
    midval = i(mid(1), mid(2));
```

```
30.
31.     offset = 55; % the offset of the values to look at.
32.
33.     imin = i<=midval-offset;
34.     imax = i<=midval+offset;
35.     isel = imax - imin;
36.
37.     ifil = imfill(isel, 'holes'); % Fill in holes
38.     G = strel('ball',6,2,4);
39.     iclo = imclose(ifil, G); % use closing to in hooks
40.     iclo = imfill(iclo, 'holes'); % fill in any holes made by closed hooks
41.
42.     % Show area overlaid on original image
43. %     figure, imshow(uint8(iclo)*50 + i);
44.     F = bwlabel(iclo); % Label the regions
45.     region = F(mid(1), mid(2)); % found our region
46.     areas(j,1) = size(find(F==region),1); % Find the area of our region
47.
48.     edg = edge(uint8(F==region),'sobel'); % Find the outline of our region
49.
50.     % Find the points that make up the outline
51.     v = []; [v(:,1), v(:,2)] = ind2sub(size(edg), find(edg));
52.
53.     t(:,:,1) = i; t(:,:,2) = i; t(:,:,3) = i; % Make a colour copy
54.
55.     % Make the outline points in the colour copy red
56.     for k = 1:size(v,1), t(v(k,1), v(k,2), :) = [255 0 0]; end
57.
58.     figure, imshow(t) % Show or save the images :D
59. %     imwrite(t,['im' int2str(j) '.png']);
60. end
```