COMP3032 Assignment: Image Contour Extraction

Dionisio Perez-Mavrogenis
School of Electronics and Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

December 6, 2012

Contents

1 Introduction . . . 2
  1.1 Equations used . . . 2
2 Approach . . . 2
  2.1 Constructing the search space . . . 2
  2.2 Construction of Energy and Position matrices . . . 3
3 Evaluation of the algorithm . . . 3
  3.1 Robustness . . . 3
  3.2 Performance . . . 5
  3.3 Application to other images . . . 5
  3.4 Comparison with a different algorithm . . . 5
4 Conclusion . . . 6

A Code listings **9**
  0.1 Code for the 3D matrix algorithm . . . 9
  0.2 Energy extraction function of 2D matrix. . . . 12

B Some of the images generated **13**
  0.3 Images of the search space generated . . . 13
  0.4 Traced contours for different values of λ . . . 13

1 Introduction

1.1 Equations used

The equations used for extracting the optimal contour from the provided image were the following :

$$E(v) = \sum_{i=1}^{N-2} E_i(v_{i-1}, v_i, v_{i+1}) \tag{1}$$

$$E_i(v_{i-1}, v_i, v_{i+1}) = \lambda \frac{|v_{i+1} - 2v_i + v_{i+1}|^2}{|v_{i+1} - v_{i-1}|^2} + (1-\lambda)I(v_i) \tag{2}$$

$$S_i(v_{i+1}, v_i) = \min_{v_{i-1}}[S_{i-1}(v_i, v_{i-1}) + E_i(v_{i-1}, v_i, v_{i+1})] \tag{3}$$

$$E(v_i) = \lambda|v_i - v_{i-1}| + (1-\lambda)I(v_i) \tag{4}$$

$$S_i(v_i) = \min_{v_{i-1}}[S_{i-1}(v_{i-1}) + E_i(v_{i-1}, v_i)] \tag{5}$$

where $\lambda \in [0,1]$ is the regularization parameter and $S_0(v_1, v_0) = 0$.

2 Approach

The coursework was coded in Matlab. Although the approach I have chosen is not exactly the same as the one in the specification(with differences highlighted as encountered), my implementation works fairly well. The image used was the file "tongue.png" provided with the specification, but the images displayed in the report are zoomed-in in the area enclosed be the contours because it is easier to spot details.

2.1 Constructing the search space

The construction of the search space is done by the function `get_search_space`. Given a number M, the search space created will have $M * N$ points, where M includes both points on the starting contours(only $M-2$ points are created).

The search space is constructed in the following manner : for each point in the first contour, calculate the x and y difference with the corresponding point in the second contour ($xdif_i = abs[v_{i,1}(x) - v_{i,2}(x)]/(M-1), ydif_i = abs[v_{i,1}(y) - v_{i,2}(y)]/(M-1)$) and then to get each new point $v'(i)$ between points $v_{i,1}, v_{i,2}$ do :

$$v'_i = [xdif_i * (J-1) + v_{i,1}(x), ydif_i * (J-1) + v_{i,1}(y)], \forall J \in [1, M]$$

where $v_{i,1}$ is point i in contour 1 and dif_i is the absolute difference between points $(v_{i,1}, v_{i,2})$(with x and y denoting the corresponding coordinate).I add $dif * (J-1)$ to $v_{i,1}$ because contour one is assumed to be "bellow" contour two, that is, it has $x_1 \leq x_2$ and $y_1 \leq y_2$. The `get_search_space` function is given as inputs an M, the vector of contour1, the vector of contour2 and the matrix of intensities of the image and returns the intensity matrix for the search space points and matrices X and Y, which contain the corresponding x and y coordinates of the search space points. Because the results of calculating the search space point coordinates might not always yield an integer, Matlab's inbuilt function `round` function was used.

The code for `get_search_space` is given in Appendix A 0.1, Listing A.2, and samples of the generated search space are shown in Appendix B 0.3, Figure B.1 .

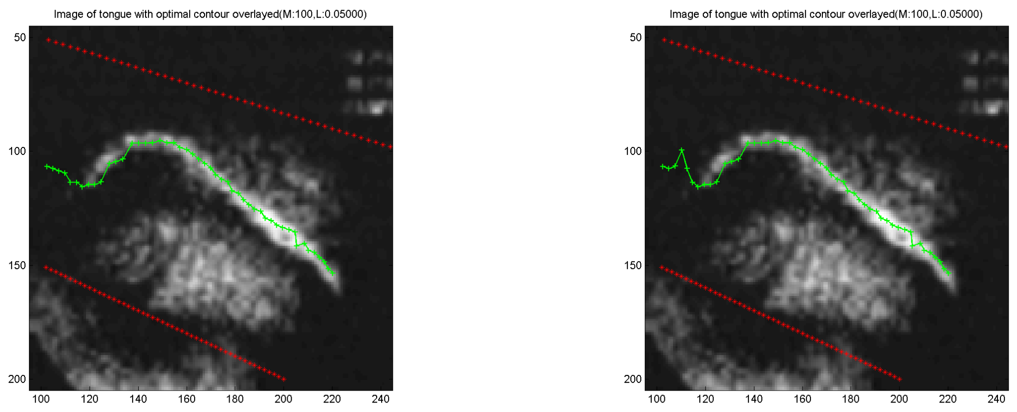


Image of tongue with optimal contour overlayed(M:100,L:0.05000)
Image of tongue with optimal contour overlayed(M:100,L:0.05000)
50
100
150
200
100
120
140
160
180
200
220
240

(a) Initialized to 0.

(b) Initialized to intensity.

Figure 1: Optimal contour for different initialization of the energy matrix(M=100,$\lambda = 0.05$.)

2.2 Construction of Energy and Position matrices

The function `get_matrices` is responsible for calculating the energy and position matrices. It is supplied with arguments : λ (the regularization parameter), the intensity matrix calculated by `get_search_space`, and the X and Y matrices(also calculated by `get_search_space`), and returns the energy matrix (an $M \times N \times M$ matrix) and a vector describing which points belong to the optimal contour (the backtracking is also done in this function), which then is handled by the plotting functions who locate the actual points on the image later on.

For a given point v_i , the entry in the energy matrix is calculated according to equation(3), with resulting value being inserted in $en(y, x, z)$ and z being equal to the y index of v_{i+1}. The energy up to point v_i is taken to be the energy between points v_{i-1} and v_i, as this is assumed to be the optimal between these two and thus the optimal between the triplet.

Simultaneously, the position matrix is being built. At each position, $posm(y, x, z) = z'$, where z' is the y-index of the optimal v_{i-1} point and z represents the y-index of the v_{i+1} point for which this calculation is taking place.

The first and last columns of the position and energy matrices present a special case, since we need points v_0 and v_{N+1} to calculate results for points in columns 1 and N, and such points do not exist. Therefore, column 1 is initialized to 0, and the energy for points in column N is calculated by equations (4) and (5).

Backtracking involved searching through the N-1 column of the energy matrix for the row that contained the minimum value , record the position of the row(y-index) and the position of the minimum value in the row(z-index). The rest of the procedure involves setting $y = z$ and reading $posm(y, x, z)$ into z, then repeating the process for every $x \in [2, N - 1]$.

The code for this section is given in Appendix A, Listing A.3. The plotting functions are called in the invoking script, which is given in Appendix A, Listing A.1.

3 Evaluation of the algorithm

The evaluation following is not general and refers only to my implementation. Also, timings were performed using Matlab's `tic toc` function, on machines in the Zepler labs.

3.1 Robustness

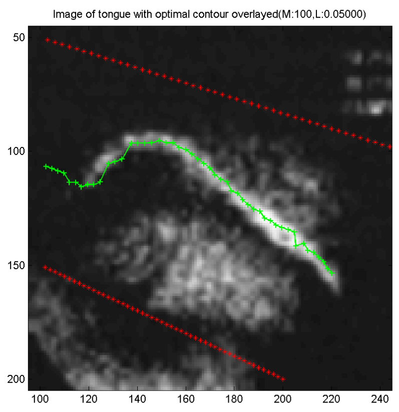


Image of tongue with optimal contour overlayed(M:100,L:0.05000)
50
100
150
200
100
120
140
160
180
200
220
240

(a) $\lambda = 0.05$

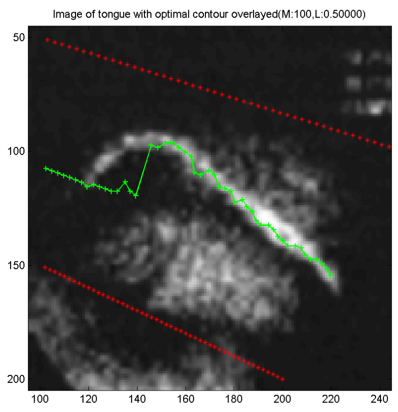


Image of tongue with optimal contour overlayed(M:100,L:0.50000)
50
100
150
200
100
120
140
160
180
200
220
240

(b) $\lambda = 0.5$

Figure 2: Optimal contour for different values of λ(M=100)

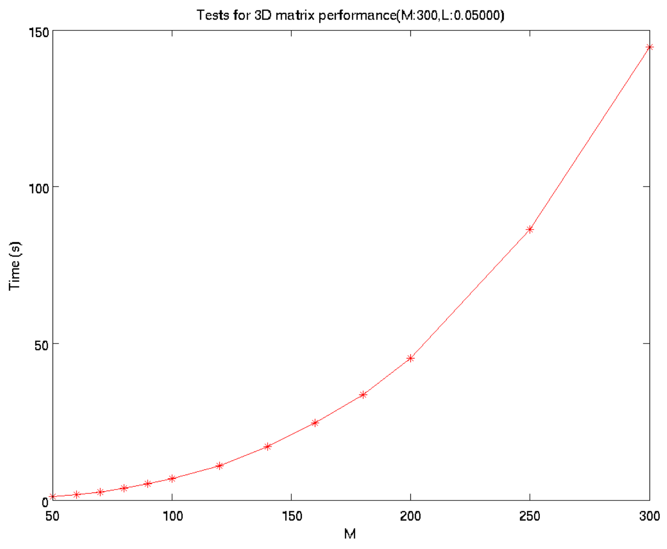


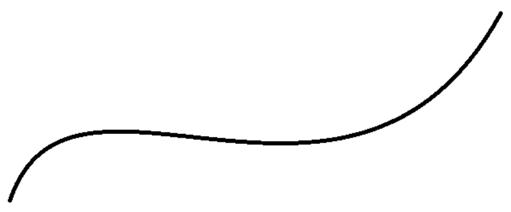
Figure 3: Benchmark of 3D matrix implementation for various values of M.

Robustness to different initialisations Two different initialization methods were tried to test the algorithms robustness. The first method set the energies of the first column to zero, as the coursework suggested, whereas the second set each point's energy to be equal to its intensity.

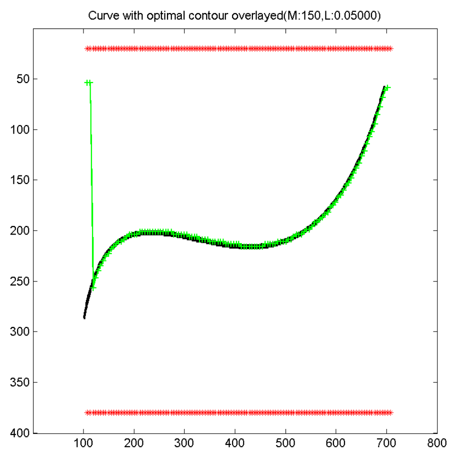
Figure 1 displays the resulting optimal contour for different initializations. The algorithm is robust enough so that the only difference is not on the actual contour being traced, but on the black space to the left of the tongue. Once the algorithm finds the optimal contour, the two contours are effectively identical.

Robustness for varying values of λ With an initial value of $\lambda = 0.5$ the results were not satisfactory,as the result of tracing deviated a lot from the desired result, whereas when varying λ it was found that acceptable results are obtained when $\lambda < 0.2$, and improved as λ got smaller.

Figure 2 shows a comparison for different λ values and Appendix B 0.4 shows resulting traces for for different values of λ



(a) The random curve



(b) The traced contour.

Figure 4: Application to a random curve(M=150)

3.2 Performance

This algorithm performs for $N-1$ columns(where N is the number of points in the initial contours), for M points, $M \times M$ operations and thus a good approximation for the running time would be $N \times M^3$, and one hypothesis suggests that the algorithms has $O(M^3)$ running time, since N could be considered a constant. My implementation uses four `for`-loops to perform the calculations and all calculations have been moved to the outermost loop possible .

Figure 3 is plot of the value of M and the corresponding running times. The shape of the curve suggests an exponential function for the total running time. It has been observed that changing the value of λ has no impact on the running time for any given M.

3.3 Application to other images

The algorithm was applied to the image of a curve that I created, and the results are shown in Figure 4. The result shown in Figure 4(b) is for M=150,λ=0.05 and a zero initialization of the energy matrix's first column. There is a small part in the beginning of the curve that is not tracked (missing points from the initial contours), and the two initial points of the optimal contour that are completely off the curve, as well as the last point of the tracing. These might be attributed to the initialization of the energy matrix's first column and different calculation for the last column's energy. However, when the algorithm "finds" the black curve it tracks the curve correctly, confirming that the initial and last points pose a special case and could perhaps be ignored, or be dealt with in a different fashion.

3.4 Comparison with a different algorithm

A second algorithm was implemented in order to extract the contour of the tongue. This is the algorithm presented in the "Dynamic Programming" lecture, and uses Equations (4) and (5) to extract the optimal contour. The algorithm performs a total of $N \times M \times M$ operations; a benchmark of this algorithm's performance is given in Figure 5, and its time complexity could be characterized as $O(M^2)$, if we treat N as a constant term.

Figure 6 shows a comparison of the contours extracted from the two algorithms. Both algorithms extracted the optimal contour fairly well, the 2D implementation being simpler and faster to implement and seems to extract a smoother contour. However, the 2D algorithm has got a sense

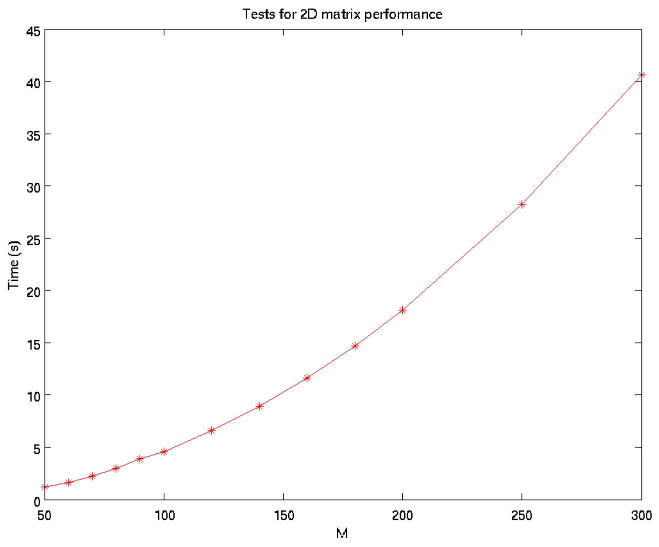


Figure 5: Benchmark of 2D matrix implementation for various values of M.

of continuity that approximates a straight line because it only considers two points, something that is illustrated in Figure 7.

Code for this algorithm is given in Appendix A.2.

4 Conclusion

This implementation of the algorithm might not be the optimal, as I am not as proficient in Matlab, however it illustrates that the algorithm works and allows comparison with other algorithms. Also, even though the background of the image containing the tongue is the original one, the algorithms run on the negated version of that image (so that the contour we extract has the minimum energy) and the result is plotted on the original image.

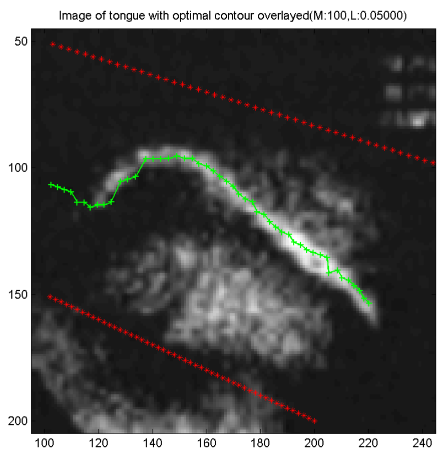


Image of tongue with optimal contour overlayed(M:100,L:0.05000)
50
100
150
200
100
120
140
160
180
200
220
240

(a) 3D matrix, M=100,$\lambda = 0.05$

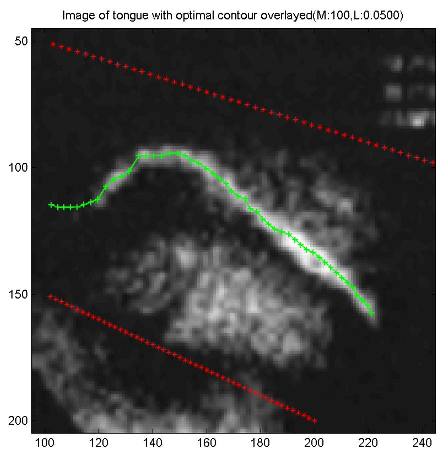
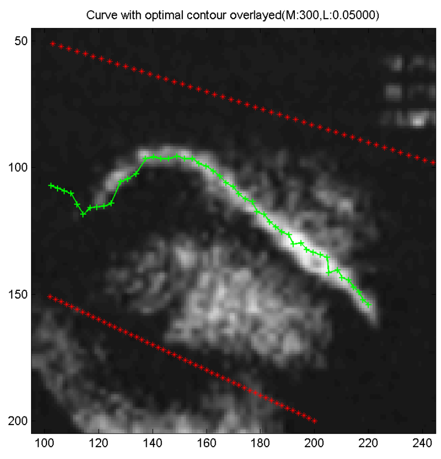


Image of tongue with optimal contour overlayed(M:100,L:0.0500)
50
100
150
200
100
120
140
160
180
200
220
240

(b) 2D matrix, M=100,$\lambda = 0.05$



(c) 3D matrix, M=300,$\lambda = 0.05$.

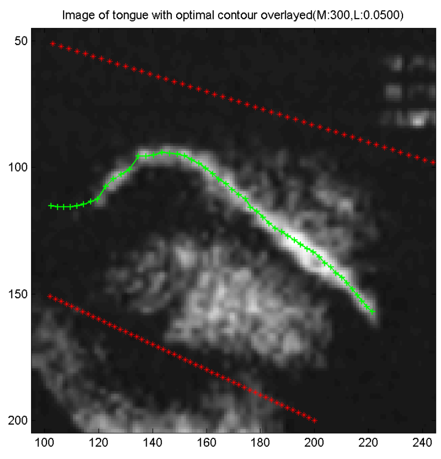


Image of tongue with optimal contour overlayed(M:300,L:0.0500)
50
100
150
200
100
120
140
160
180
200
220
240

(d) 2D matrix, M=300,$\lambda = 0.05$.

Figure 6: Comparison of the two algorithms

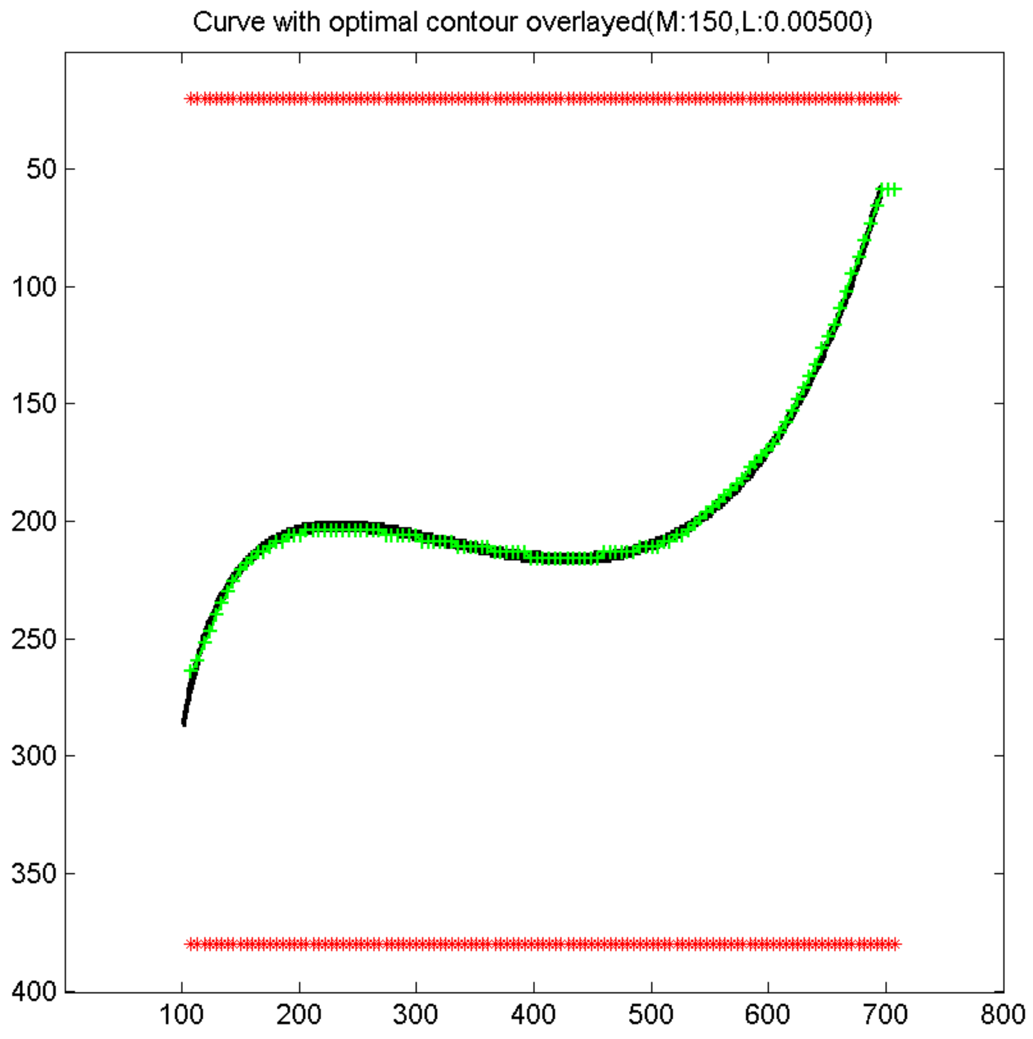


Figure 7: Tracing of curve by 2D algorithm ($M = 150, \lambda = 0.005$), with portions of the contour being completely straight, illustrating this algorithm's linear sense of continuity.

Appendix A

Code listings

0.1 Code for the 3D matrix algorithm

Listing A.1: Code used to initialize, run and plot the results for the 3D algorithm.

```
M = 100 ;
L = 0.05 ;
im = imread('tongue.png');
im = double (im);
immin = min(min(im)) ;
immax= max(max(im));
ctr1 = load('init1.ctr');
ctr2 = load('init2.ctr');
im = (im - immin)/(immax-immin);
im2 = 1 - im ; %this makes the image black and white and allows energy
%minimizations rather than maximization

%intensities : matrix of intensities x,y : matrices of points corresponding
%the above intensities.
[intensities,x,y]=get_search_space(M,ctr1,ctr2,im2);
%en : energy matrix pos :position matrix
[en, pos] = get_matrices_3D(L,intensities, x,y);
%optimal xs and ys
pts = get_optimals_3D(pos,x,y);
str = sprintf('Curve with optimal contour overlayed(M:%d,L:%.5f)',M,L);
%uncomment for printing to a file
fig = figure('name',str);%,'visible','off');
imagesc(im)
colormap(gray)
axis square
hold on
%plot contours
plot(ctr1(:,1),ctr1(:,2),'r*',ctr2(:,1),ctr2(:,2),'r*','MarkerSize',5)
%plot optimal path
plot(pts(:,1),pts(:,2),'g+-','LineWidth',1,'MarkerSize',5)
%view only% search space
axis([95 245 45 205])
hold off
title(str);
```

```
%uncomment to allow for printing to a file
%fname =sprintf('clcap-%d-%.5f.png',M,L);
%print(fig,'-dpng',fname);
```

Listing A.2: Function generating the search space for the 3D algorithm.

```
function [intensities, x,y] = get_search_space (M, ctr1 , ctr2, im)

intensities = zeros(M, size(ctr1(:,1),1)) ;
x = zeros(M, size(ctr1(:,1),1)) ;
y = zeros(M, size(ctr1(:,1),1)) ;

for index = 1 : size(ctr1(:,1),1)
    xdif = (ctr2(index, 1) - ctr1(index,1))/(M-1) ;
    ydif = (ctr2(index, 2) - ctr1(index,2))/(M-1) ;
    for J = 1 : M
        cx = xdif*(J-1) + ctr1(index,1) ;
        cy = ydif*(J-1) + ctr1(index,2) ;
        intensities(J,index) = im(round(cy),round(cx));
        x(J,index) = cx ;
        y(J,index) = cy ;
    end
end

end
```

Listing A.3: Function generating the position and energy matrices and doing the backtracking.

```
function [ en,pos ] = get_matrices(l,intensities, x , y)
rows = size(intensities(:,1),1);
cols = size(intensities(1,:),2);
en = zeros(rows,cols, rows);
pos = zeros(1,cols-1) ;
posm = zeros(rows,cols, rows);
%initialize matrix
for ly = 1:rows
    en(ly,1,:) = 0;
    %en(ly,1,:) = intensities(ly,1); %uncomment to initialize matrix
    %dfferently
end
    for cx = 2:cols-1
        for cy = 1:rows
            curr_int = intensities(cy,cx);
            curx = 2*x(cy,cx) ;
            cury = 2*y(cy,cx) ;
            for ny = 1:rows
                sums = zeros(1,rows);
                tx1  = x(ny,cx+1) - curx;
                ty1  = y(ny,cx+1) - cury;
                nextx = x(ny,cx+1);
                nexty = y(ny,cx+1);
                for py = 1:rows
```

```
                    tx = x(py,cx-1) + tx1;
                    ty = y(py,cx-1) + ty1;
                    enumerator = tx^2 + ty^2;
                    dx = nextx - x(py,cx-1);
                    dy = nexty - y(py,cx-1);
                    denominator = dx^2 + dy^2;
                    proximity = l*(enumerator/denominator) + (1-l)*curr_int ;
                    curr_en = en(py,cx-1,cy) + proximity;
                    sums(1,py)= curr_en ;
                end
                [minv, mini ] = min(sums);
                en(cy,cx,ny) = minv;
                posm(cy,cx, ny) = mini;
            end
        end
    end

    %fill last column
    for cy =1:rows
         sums= zeros(1,rows);
        for py =1:rows
          expre=l*sqrt((x(cy,cols)-x(py,cols))^2+(y(cy,cols)-y(py,cols))^2)
          +(1-l)*intensities(cy,cols);
          sums(1,py)=expre;
        end
        en(cy,cols,:) = min(sums);
    end

         %start backtracking : locate the minimum in the penultimate column
    [~ , miny] = min(en(:,cols -1));
    pos(1,1) = miny;

         %backtrack through the rest of the columns
    for X = fliplr(2:cols -1)
        newz= miny;
        newy = posm(miny,X,newz);
        pos(1,X) = newy;
        miny = newz;
    end
end%end function
```

Listing A.4: Function that gets the optimal points returned from `get_matrices` and retrieves their (x,y) coordinates, and adds them to an $N \times 2$ vector for the plotting functions.

```
function [ opts ] = get_optimals(pos, x ,y)
%Get (x,y) coordinates of points lying in the optimal contour
cols = size(x,2);
opts = zeros(cols -1,2);
for i = 1:cols-1
    opts(i,1) = x(pos(i),i);
    opts(i,2) = y(pos(i),i);
```

```
end
end
```

0.2 Energy extraction function of 2D matrix.

Since the search space extraction and plotting functions are identical to the 3D implementation, only the energy calculation and back-tracking code for the 2D implementations is shown.

Listing A.5: Function that generates the position and energy matrices for the 2D implementation of the algorithm.

```
function [ en , pos] = fill_energy_matrix(l,intensities , x , y)

rows = size(intensities(:,1),1);
cols = size(intensities(1,:),2);
en = zeros(rows,cols);
pos = zeros(rows,cols) ;
sum = zeros(1,rows);
ty=zeros(1,rows);
%%Initialize first column of matrix
for a1 = 1: rows
    en(a1,1) = intensities (a1,1);
end

%fill rest of matrix based on that
for a = 2:cols
    for b = 1:rows
        for j = 1 : rows
          curr_value = en(j,a-1)+
            point_energy(intensities,l,x(b,a),y(b,a),x(j,a-1),y(j,a-1),a,b);
          sum(1,j) = curr_value;
          ty(1,j)=j;%positions of previous box used to calculate energy
        end
        [minv mini] = min(sum);
        en(b,a) = minv;
        pos(b,a) = ty(mini) ;
        ty=zeros(1,rows);
        sum = zeros(1,rows);
    end
end

end
```

Listing A.6: Function that does the calculation for the transition between points v_{i-1}, v_i.

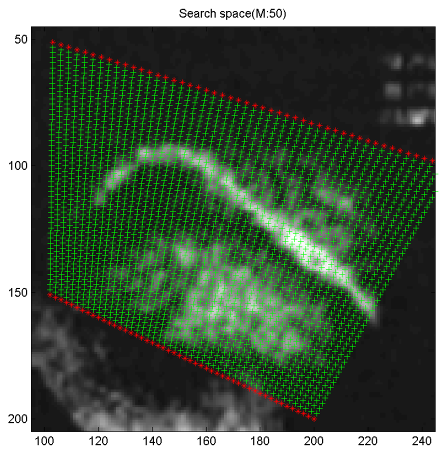
```
%intensity matrix,
%l,
%x-coord/ycoord in actual image(of current sampling point),
%x/y-coord on actual image(of previous point),
%current x,y index in intensity matrix
function [y] = point_energy(im ,l , cx , cy , px , py, cx_i , cy_i)
    y=l*sqrt((cx-px)^2+(cy-py)^2)+(1-l)*im(cy_i,cx_i);
end
```

Appendix B

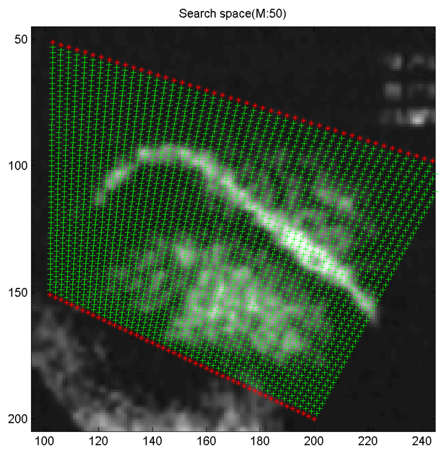
Some of the images generated

0.3 Images of the search space generated

These are two sample images generated by the `get_search_space` function. The search points are not plotted together with the optimal contour because the resulting image would get too cluttered to draw any results.



(a) Search space generated for M=50



(b) Search space generated for M=100

Figure B.1: Plotting of two sample search spaces.

0.4 Traced contours for different values of λ

These tracings are generated by the 3D energy matrix algorithm and have $M = 100$.

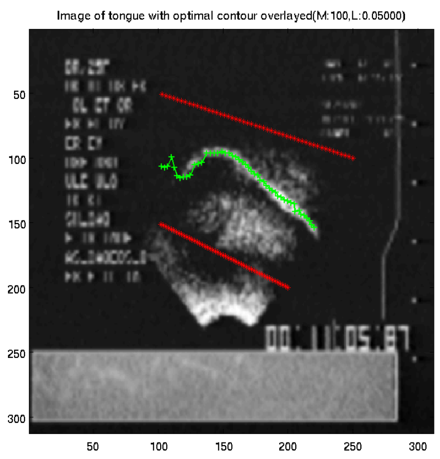


Image of tongue with optimal contour overlayed(M:100,L:0.05000)
50
100
150
200
250
300
50
100
150
200
250
300

(a) $\lambda = 0.05$

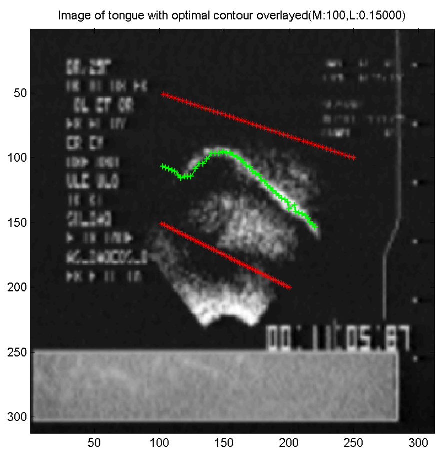


Image of tongue with optimal contour overlayed(M:100,L:0.15000)
50
100
150
200
250
300
50
100
150
200
250
300

(b) $\lambda = 0.15$

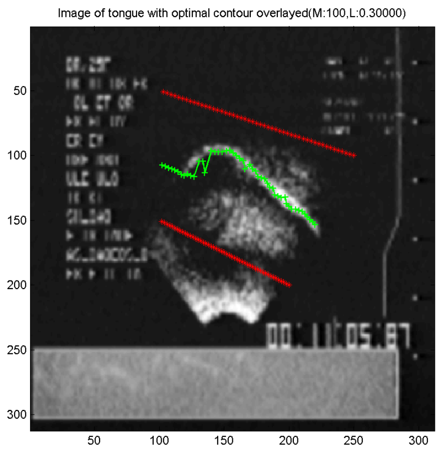


Image of tongue with optimal contour overlayed(M:100,L:0.30000)
50
100
150
200
250
300
50
100
150
200
250
300

(c) $\lambda = 0.3$

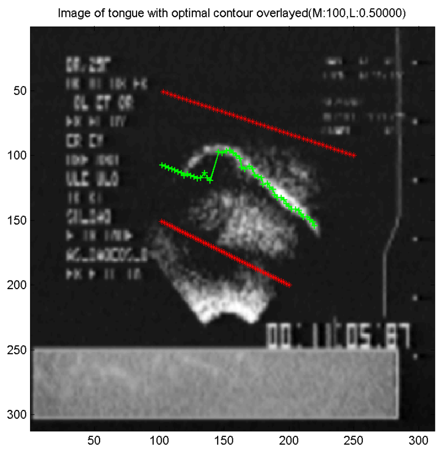


Image of tongue with optimal contour overlayed(M:100,L:0.50000)
50
100
150
200
250
300
50
100
150
200
250
300

(d) $\lambda = 0.5$

Figure B.2: Plotting of two sample search spaces, with M=100.