Problem Set 1 solutions

Prepared by Ahsan Abdullah and Vivek Dubey

I. Short answer problems

1. A. Linearly Separable Filters: If a 2-D filter can be represented as a convolution of two 1-D filters, then convolving an image with two 1-D filters is computationally more efficient.
   B. Sequential Filters: Due to the associative property of convolution, convolving an image with one filter after another is equivalent to convolving the image with a convolved result of the two filters. Since usually the filter size is smaller than the image size, the latter is computationally more efficient. That is, (I*f)*g will have more computations than I*(f*g).

2. [0 1 1 1 1 1 1]

3. Let I be an image, and f' be the given filter that gives us the first derivative of an image when convolved with the image.
   I'= I * f'
   Therefore, the double derivative will be the same filter applied to the first derivative.
   I'' = (I * f') * f'
   I'' = I * (f' * f')
   I'' = I * f''; where f'' = f' * f'
   f' convolved with f' is equivalent to f' correlated with f' flipped vertically and horizontally.
   Therefore, f'' = [0 0 ¼ 0 -1/2 0 ¼ 0 0]

4. In cases where the noise in an image is not Gaussian and instead is, say Salt and Pepper, Gaussian noise assumption for representation would be incorrect.

5. Since this is a design question, there could be several correct answers to this question. We could use several techniques for detecting flaws in products on the conveyer belt and a few of the possible steps are described below.
   Chamfer Matching:
   - Perform edge detection on an image of the perfect item.
   - Perform binarification on the image
   - Capture the template of the desired product property
   - Scan each image as it is captured upon the item arrival on the conveyer belt for the best match with the template, with Chamfer distance as our matching criterion.

   Hough Transform:

   - Convert the color image to grayscale
   - Apply an edge detection algorithm to the grayscale image
   - Apply a threshold to the edge image to clearly decide for each image point if it's an edge point or not
   - Perform the actual Hough transform by converting the threshold-edge image to the Hough space.
   - Perform a threshold based voting to find the points in Hough space that represent lines
   - Convert the lines detected in hough space back to image space for comparison.

Deformable Contours:

- It will be applied to the gradient magnitude of the image and not the edge points
- Place the deformable contour('snake') near the image contour of interest
- During an iterative process, the snake is attracted towards the target contour by minimizing the energy functions that control the shape and location of the contour within the image.

II. Programming problem: content-aware image resizing

energyimage function

```
function [energyImage] = energy_image(im)
    im = rgb2gray(im);
    im = double(im);

    % Using Gradient function, can be done in other ways.
    [gradX, gradY] = gradient(im);
    energyImage = sqrt((gradX.^2) + (gradY.^2));
end
```

cumulativeEnergyMap function

```
function cumulativeMinMap = cumulative_minimum_energy_map(energyImage,
seamDir)
% Finding Cumulative Energy Map by Dynamic Programming on the input Energy
Image

cumulativeMinMap = ones(size(energyImage));
cumulativeMinMap(:,:) = intmax;

if strcmp(seamDir,'VERTICAL') == 1
    cumulativeMinMap(1,:) = energyImage(1,:);
    for i = 2:size(cumulativeMinMap, 1)
        for j = 1:size(cumulativeMinMap, 2)
            val1 = intmax;val3 = intmax;
            if (j > 1)
                val1 = cumulativeMinMap(i-1,j-1);
            end
            val2 = cumulativeMinMap(i-1,j);
            if (j <  size(cumulativeMinMap, 2))
                val3 = cumulativeMinMap(i-1,j+1);
            end

            cumulativeMinMap(i,j) = energyImage(i,j) +
min(val1,min(val2,val3));
        end
    end
else
    cumulativeMinMap(:,1) = energyImage(:,1);
    for j = 2:size(cumulativeMinMap, 2)
        for i = 1:size(cumulativeMinMap, 1)
            val1 = intmax; val3 = intmax;
            if (i > 1)
                val1 = cumulativeMinMap(i-1,j-1);
```

```
            end
            val2 = cumulativeMinMap(i,j-1);
            if (i <  size(cumulativeMinMap, 1))
                val3 = cumulativeMinMap(i+1,j-1);
            end

            cumulativeMinMap(i,j) = energyImage(i,j) +
min(val1,min(val2,val3));
        end
    end

end
end
```

find_optical_vertical_seam function

```
function seam = find_optimal_vertical_seam(cumulativeEnergyMap)

% Traceback of the Dynamic Programming to find the Optimal Seam
% in vertical direction

seam = zeros(size(cumulativeEnergyMap, 1),1);
lastR = cumulativeEnergyMap(end, :);

[~, minEnds] = find(lastR == min(lastR));
seam(end) = minEnds(1);
lastInd = seam(end);

for i = numel(seam)-1:-1:1
    val1 = intmax;val3 = intmax;
    if (lastInd > 1)
        val1 = cumulativeEnergyMap(i, lastInd-1);
    end

    val2 = cumulativeEnergyMap(i, lastInd);

    if (lastInd < size(cumulativeEnergyMap, 2))
        val3 = cumulativeEnergyMap(i, lastInd+1);
    end
    minArr = [val1 val2 val3];
    minVal = min(minArr);

    [~, col] = find(minArr == minVal);
    seam(i) = col(1) + lastInd - 2;
    lastInd = seam(i);
end
end
```

find_vertical_horizontal_seam function

```
function seam = find_optimal_horizontal_seam(cumulativeEnergyMap)

% Traceback of the Dynamic Programming to find the Optimal Seam
% in horizontal direction

seam = zeros(size(cumulativeEnergyMap, 2),1);
lastC = cumulativeEnergyMap(:,end);
```

```
[minEnds, ~ ] = find(lastC == min(lastC));
seam(end) = minEnds(1);
lastInd = seam(end);

for i = numel(seam)-1:-1:1
    val1 = intmax;val3 = intmax;
    if (lastInd > 1)
        val1 = cumulativeEnergyMap(lastInd-1, i);
    end

    val2 = cumulativeEnergyMap(lastInd, i);

    if (lastInd < size(cumulativeEnergyMap, 1))
        val3 = cumulativeEnergyMap(lastInd+1, i);
    end
    minArr = [val1; val2; val3];
    minVal = min(minArr);

    [row, ~] = find(minArr == minVal);
    seam(i) = row(1) + lastInd - 2;
    lastInd = seam(i);
end
end
```

display_seam function

```
function display_seam(im, seam, seamDirection)
figure,
imshow(im);
hold on;
if strcmp(seamDirection, 'VERTICAL') == 1
    plot(seam, 1:numel(seam))
else
    plot(1:numel(seam), seam)
end
end
```

reduce_width function

```
function [reducedColorImage, reducedEnergyImage] = reduce_width(im,
energyImage)

CumMinMap = cumulative_minimum_energy_map(energyImage, 'VERTICAL');
vertSeam = find_optimal_vertical_seam(CumMinMap);
reducedColorImage = zeros(size(im,1), size(im,2)-1, 3);

for i = 1:numel(vertSeam)
    reducedColorImage(i,:,:) = [im(i, 1:vertSeam(i)-1, :) im(i,
vertSeam(i)+1:end, :)];
end
 reducedColorImage = uint8(reducedColorImage);
 reducedEnergyImage = energy_image(reducedColorImage);
%  display_seam(im, vertSeam, 'VERTICAL');
end
```

reduce_height function

```
function [reducedColorImage, reducedEnergyImage] = reduce_height(im,
energyImage)

CumMinMap = cumulative_minimum_energy_map(energyImage, 'HORIZONTAL');
horizSeam = find_optimal_horizontal_seam(CumMinMap);
reducedColorImage = zeros(size(im,1) - 1, size(im,2), 3);

for i = 1:numel(horizSeam)
    reducedColorImage(:,i,:) = [im(1:horizSeam(i)-1, i, :);
im(horizSeam(i)+1:end, i, :)];
end
 reducedColorImage = uint8(reducedColorImage);
 reducedEnergyImage = energy_image(reducedColorImage);
% display_seam(im, horizSeam, 'HORIZONTAL');
end
```

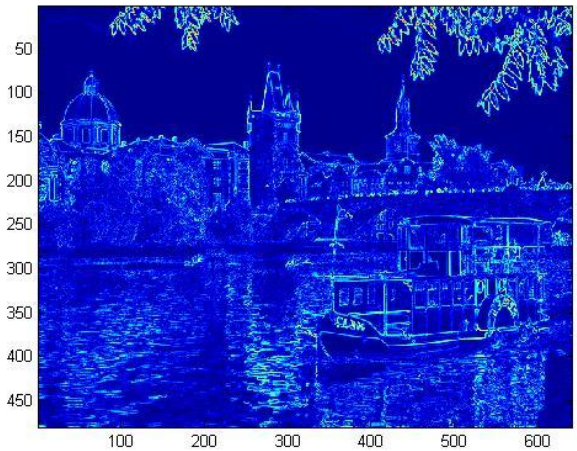
1. C. outputReduceWidthPrague.png

D. outputReduceWidthMall.png

2. outputReduceHeightPrague.png

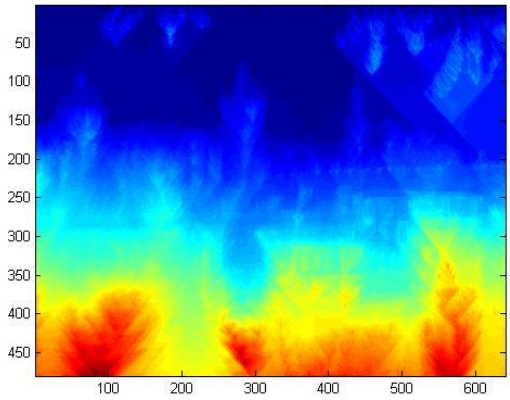
outputReduceWidthMall.png

3. EnergyMap



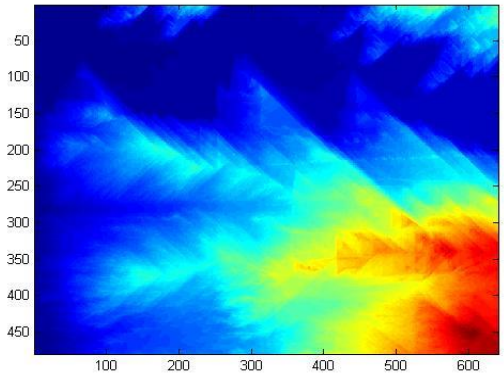
The Energy Map looks like the above because when we calculate the gradient we are essentially capturing the edge-information only. And since we are taking the gradients in both the x and y directions, we see edges in all orientations. So, the bright areas in the above image tell us where the high-energy (edges) areas are, while the dark regions point to the smooth areas in the original image.

CumulativeEnergyMap Vertical



Since we are summing up the energy value from top-down, the lower elements of the above display are red (higher value) and the upper regions are blue. Even here, we can see which are the high energy areas, or intuitively, the areas which our vertical seam should avoid.

CumulativeEnergyMap Horizontal

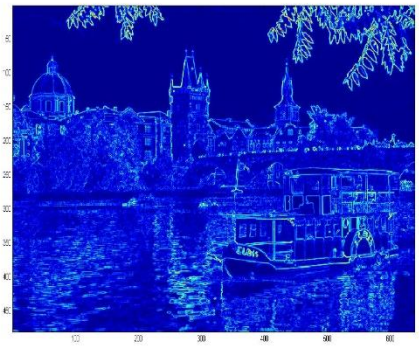


Similar to the vertical CumulativeEnergyMap, the red areas capture the high energy areas, and we can say that the summation has happened from left part of the image to the right. One thing to observe here is that the sky area in the original image corresponds to a lot of blue (low-energy) area.

4. If we observe the horizontal seam, we see that it passes through the sky all the way, avoiding the buildings and leaves. That is where we would observe the change the least since there are no edges in that area. Correspondingly, we see that the vertical seam passes through the least 'noticeable' vertical path too.

5. We used Canny edge detection to generate the energy image and as the first seam comparison, we got the exact same seam. That is to say that the edges that we got through Canny edge detection were mostly the same as when we did our energy calculations.

   Following are how our first seam and the energy image looks.



6. Input example:

Batman !

Reduce width by 100 px and height by 50

Original Image Size: 391x660
Reduced Image Size: 341x560
imresize result: resized to 341x560

As we can see, the areas from the left and right of batman have been removed for vertical seam removal. Also, the length of Batman's 'ears' and forehead have been reduced too. This is because all other areas for horizontal seams have pretty high energy values. The most smooth area in comparison was the one which is removed.