Kim Dang

May 3, 2017

Professor Yong Jae Lee

ECS 174

Problem Set 1

I. Short Answer Problems

1. Since the convolution expression (F * G) * H will have the same result as F * (G * H), with F as the original image, we can compute two filters G and H together, and apply the convolution g * h to the original image F as a single filter. If the image F is large, this can cut down a lot of processing time.

2. The image [1 0 1 0 0 0 1 1] dilated by the structuring element [1 1 1] results in [1 1 1 1 0 1 1 1].

3. The second derivative filter f'' is [0 1/2 -1 1/2 0].

4. Representing image noise as additive Gaussian noise depends on the noise variation along a normal distribution. Additive Gaussian noise not ideal for images where the distribution of noise variation is not normal and is skewed, so smoothing filters specifically for normally distributed noise will not be as effective in these cases.

5. We assume that the part is in an environment where the part itself can be easily distinguished from the conveyer belt and any other surroundings to reduce noise. The camera is fixed above the conveyer belt, and the parts are assumed to be of relatively uniform size and shape under the camera, evenly spaced from each other. We also assume that we have a sufficient set of images or models of what the part should look like. The raw image of each iteration of the part is converted to grayscale, then simplify the image into a more binary image with by thresholding, refined with dilation and erosion. Key components of the part are then identified separately by algorithm of connected components. The distance between these key components are compared to the average or range dictated by the reference model. If the calculated distances are out of the acceptable range, they are reported.

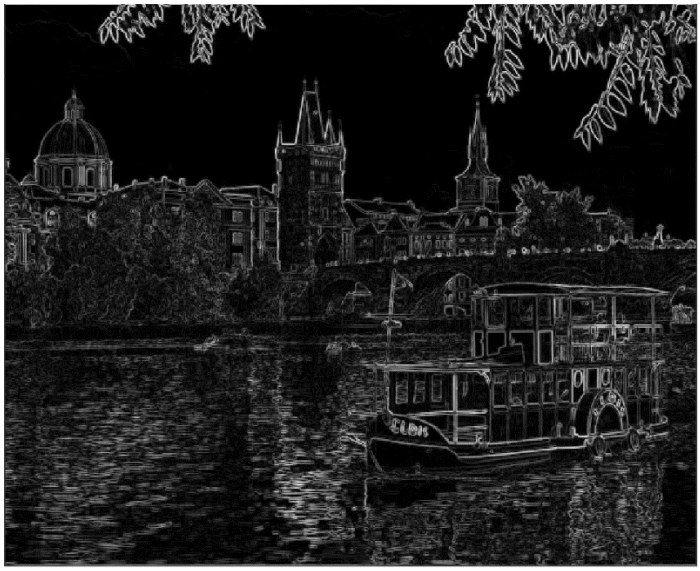
II. Programming Problem

1. The images on the right of the table are the original images, while the images on the left are the images after they underwent seam carving. The images are 100 pixels less in width.

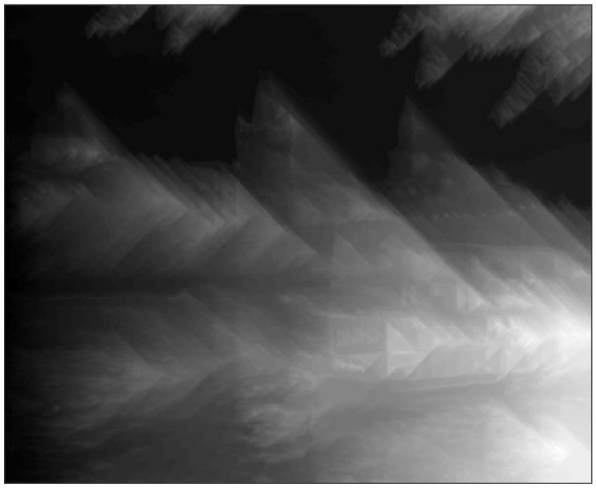
2. The images on the right of the table are the original images, while the images on the right are the images after seam carving. The images are 50 pixels less in height.

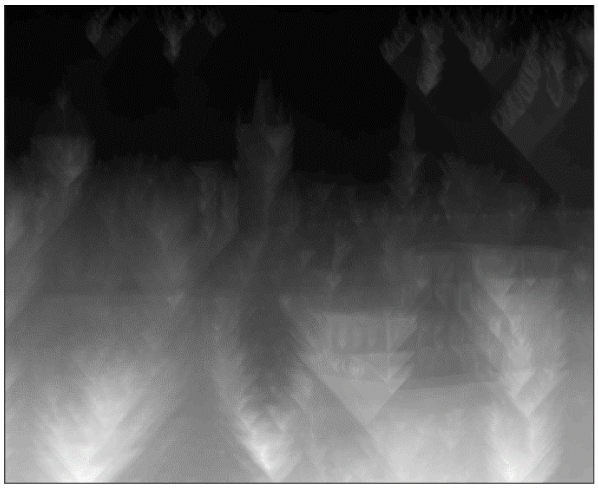
3.

a. This is the energy function output of the image ‘inputSeamCarvingPrague.jpg’.



b. These two images are the cumulative minimum energy maps in the horizontal (left) and vertical (right) directions.





The energy function output is merely the outlines of the edges in the image, delineating key components that our seam carving script should avoid when calculating the minimum cost path. The cumulative minimum energy then shows the key components more clearly. The white areas indicate high energy cost on the right of the horizontal energy map and at the bottom of the vertical energy map. This is because our cumulative energy map function sweeps through the image left to right for horizontal, and top to bottom for vertical.

In the horizontal cumulative energy map, high energy is shown at the large boat, buildings, and bridges, since we shouldn’t start compressing the boat when we carve the image horizontal. The vertical cumulative energy map shows high energy cost at the areas such as the boat and the highly reflective parts of the water. Once again, the boats is a key component to avoid carving into.

4. Figure (4) is the original image of ‘inputSeamCarvingPrague.jpg’. Figure (4a) is the first selected horizontal seam, and figure (4b) shows the first selected vertical seam.

Figure 4

Figure 4a

Figure 4b

The first horizontal seam appears in the sky which has the same color and little contrast, so there is little cost to cut this seam out. The first vertical seam appears right through the edge of between the dark tower and the white building the seam would be a near vertical line. The vertical seam goes through the reflection of the white building because the colors are the same there, indicating low cost.

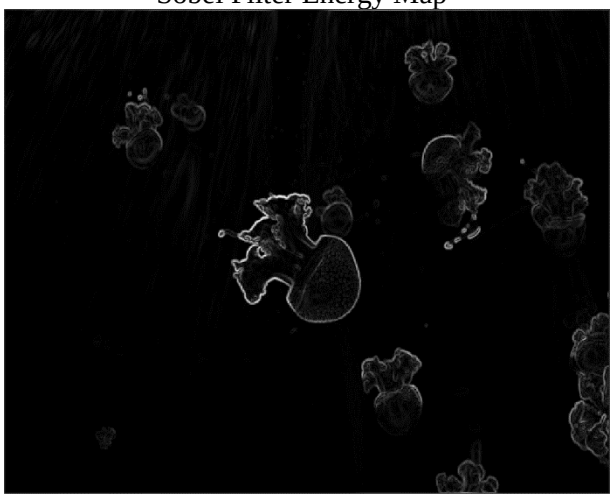
5. The current scripts use a Sobel filter to create the energy map of the image. We will try using the predefined ‘motion’ filter to create the energy map of the image instead. We attempt reduce the width of the image by 100 pixels using the Sobel filter and the ‘motion’ filter. The energy maps and results are presented in the next page.

   Rather than depict the edges of the edges, the motion filter creates blobs of the key components of the image already, but with more blurred lines instead of more defined lines like with the Sobel filter. This may cause the cumulative energy map to have have less clarity and defined areas of interest to avoid when carving. As a result, the ‘motion’ filter energy map has a noticeable rigid cut in the center of the image around the central jellyfish, and the central jellyfish is shifted a little more to the left compared to the result of the Sobel filter, which makes more seamless cuts into the image and keeps the subject of the image in the center. The rest of the image is well carved with the ‘motion’ filter. However, we can conclude that the ‘motion’ filter isn’t quite the ideal filter to use for seam carving.

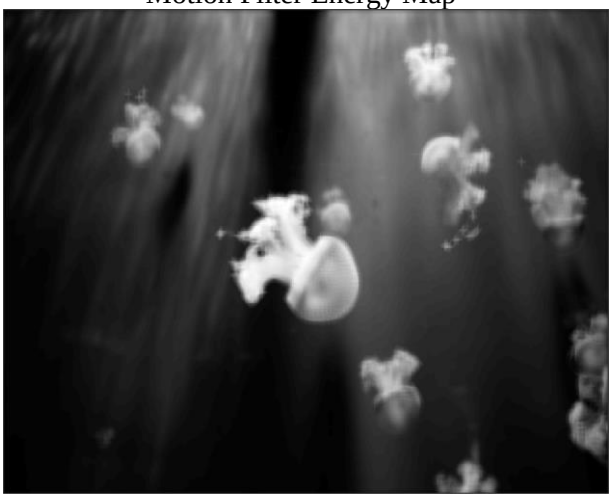
jellyfish.jpg (source: https://unsplash.com/photos/HakjRbaoaQE.)

Original Image

Sobel Filter Energy Map



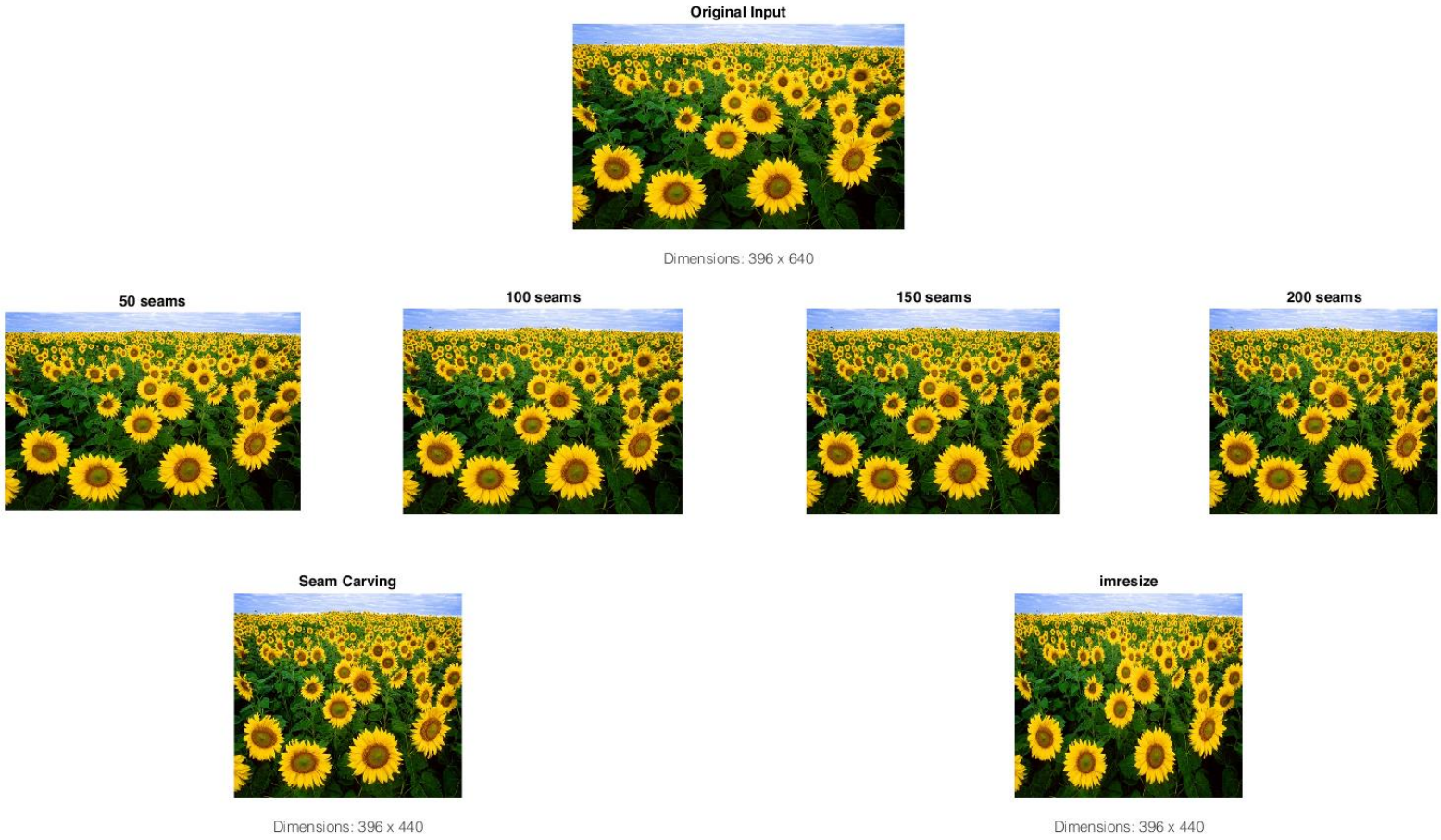
Motion Filter Energy Map



Sobel Filter Result

Motion Filter Result

6. We experiment with four images and display the following.



`sunflowers.m` (source: https://pixabay.com/en/sunflower-sunflower-field-flora-11574/)

We seek to reduce the width of the original images by 200 pixels, meaning we must remove 200 vertical seams. The sunflowers stand out very well from the leaves so, we expect our system will use the seams that go around the flowers rather than through them. Matlab's imresize does take advantage of these dark areas, too, but not as well as our seam carving system does. The seam carving mostly preserves the circular shapes of the sunflowers, while the imresize version does compress some of the flowers vertically.

`ornaments.m` (source: https://unsplash.com/photos/eBE3pEIZjbc.)

This is a very cluttered and complex picture. We expect the ornaments to be preserved because cumulative energy will be greater around these areas, and if the image is seam carved, the bottom and the left side will be cut out to avoid disturbing the complex parts. This image demonstrates a key problem with our current system. The seam carving does not create a distinction between subjects with strict proportions and those without, which our assumption did not take into consideration. While the ornaments are untouched in the seam carving result, the man in the picture is subject to change, and the horizontal seams cut the man in two and takes parts of his face. imresize squashes the man but does not squash him significantly, which is one good thing this function does. However, it does not preserve the ornaments themselves, which have been squashed vertically compared to the original and seam carved results.

`night.m` (source: https://www.pexels.com/photo/architecture-clouds-dawn-dome-380789/)

We cut the height by 100 pixels and width by 100 pixels. We predicted that the image will preserve the center tower, the clouds on the left. We also expect that the shore at the bottom will be cut out because that area is around the same color, and the seam finding function will avoid crossing the edge of the shore. The tower is centered in the picture, its size preserved. The clouds on the right are cut down but not to a significant effect to look out of place. However, the shore is forced to stretch downward, and a significant horizontal cut on the reflection of the pink clouds on the left side of the picture. This is likely a result of many colors in an image.

Dimensions: 426 x 640

Dimensions: 306 x 640

Dimensions: 306 x 640

`soccer.m` ((https://www.pexels.com/photo/action-ball-field-game-274506/)

We cut the image's height by 100px in the hopes that only the lawn is affected. While the soccer ball is compressed using Matlab's imresize, the image carving does preserve the ball and instead reduces the image by taking seams around the ball thanks to the cumulative energy maps we make every time we find a new seam.

III. Extra Credit

4. We attempt a greedy approach instead of dynamic programming to find the optimal seams when we reduce the width or height on an image. Rather than use the `cumulative_energy_map`.m function at all, we instead create a seam directly from the energy map itself. The only difference here is deleting a line of code. For comparison is the snippet of code that is changed:

Greedy approach:

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

    reducedColorImage = uint8(zeros(size(im,1)-1, size(im,2), 3));

    reducedEnergyImage = zeros(size(im,1)-1, size(im,2));

    sv = find_optimal_horizontal_seam(energyImage);

    red = im(:,:,1);
```

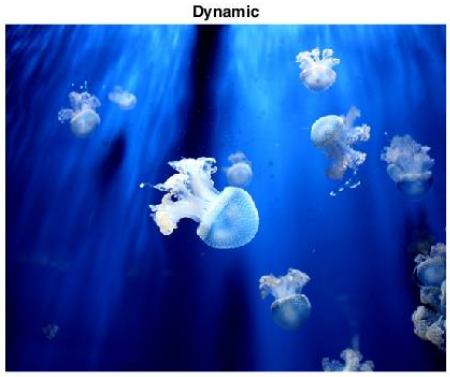
Dynamic Programming approach:

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

    reducedColorImage = uint8(zeros(size(im,1)-1, size(im,2), 3));

    reducedEnergyImage = zeros(size(im,1)-1, size(im,2));

    pv = cumulative_energy_map(energyImage, 'HORIZONTAL');

    sv = find_optimal_horizontal_seam(pv);

    red = im(:,:,1);
```

In order, we tested the greedy approach on several images below.

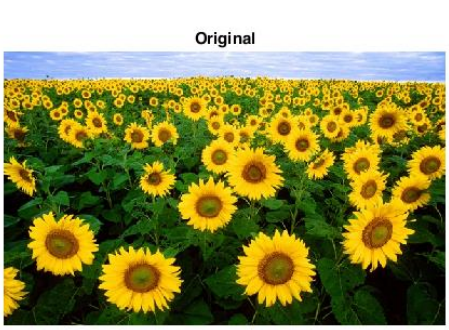
soccer.jpeg (https://www.pexels.com/photo/action-ball-field-game-274506/)

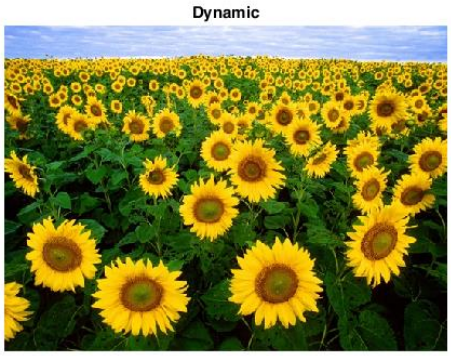
We reduce the image's height by 100 pixels. The greedy approach cuts the lower half of the soccer ball out of the picture and bends the white boundary of the field, while the dynamic programming approach does not affect these key components of the image. Without a cumulative energy map, the system cannot look ahead to avoid the curvature of the ball or the boundary of the field.



jellyfish.jpeg (https://www.pexels.com/photo/close-up-of-jellyfish-swimming-in-sea-336623/)

We attempt to reduce the width of the picture by 100 pixels. The greedy approach appears to compress the center jellyfish more and creates an irregularity in the final product because most of the seams that would be taken out are most likely to be in the center of the image where there is a huge black gap at the bottom of the picture. The dynamic programming approach preserves the shape and size of the center jellyfish.





sunflowers.jpeg (https://www.pexels.com/photo/sky-field-flowers-blue-87056/)

There aren't very many noticeable differences between the greedy and the dynamic programming approach save for one sunflower near the center of the image. The head is made smaller in the greedy approach, and the head is made bigger in the dynamic approach.

drink.jpeg (https://www.pexels.com/photo/alcoholic-bar-beverage-blur-338713/)

We attempt to reduce the height of the picture using horizontal seams. For the most part, the beverage itself is largely untouched, but we can see a noticeable difference in how both algorithms treat the leaves at the top of the glass. In the greedy approach, the middle of the leaves is cut out, while the leaves' size is preserved and are instead cut out in the result of the dynamic programming approach.