ECE-5554 Computer Vision: Problem Set 1

Murat Ambarkutuk
murata@vt.edu

Mechanical Engineering Department,
Virginia Polytechnic Institute and State University

September 16, 2015

Answer Sheet

1 Short Answer Problems

1. The computational complexity of convolution for any arbitrary kernel $F_{0[m\times n]}$ and image $I_{0[M\times N]}$:

$$\mathcal{O}(MNmn)$$

Hence, applying various filters will drastically increase the computational cost of filtering process.
For instance, $F_{0[m\times n]} * (F_{1[m\times n]} * I_{0[M\times N]})$ will be computed with the complexity of $2 \times \mathcal{O}(MNmn)$.
On the other hand, associative property of convolution can be utilizied by convolving the filters first, then applying the acquired filter to image: $(F_{0[m\times n]} * F_{1[m\times n]}) * I_{0[M\times N]}$

$$\mathcal{O}(m^2n^2) + \mathcal{O}(MNmn)$$

Given that in any filtering process the size of filter kernel, by its nature, will be smaller than the image size, the computational complexity will be reduced thanks to the assosicative property of convolution.

$$\mathcal{O}(m^2n^2) + \mathcal{O}(MNmn) \ll 2 \times \mathcal{O}(MNmn)$$

2.

$$\begin{aligned} \vec{I} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}, \vec{f} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \vec{H} = \vec{I} \bigoplus \vec{f} \\ = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{1}$$

3. $\vec{f''} = \begin{bmatrix} \frac{-1}{4} & 0 & \frac{1}{2} & 0 & \frac{-1}{4} \end{bmatrix}$ (After discussing the question with Yazhe Hu and Tian Yi regarding the discussion conducted on the forum, I changed my answer according to Prof. Parikh's prompt.)

4. (a) Gaussian Kernel: Canny edge detection algorithm utilizes Gaussian filter to reduce noise. However, too big kernel will smooth out some of the detail in the input image. In that case, output image would have fewer fine edges.

(b) Wrong Thresholding: Setting too high numbers for high and low threshold would reduce the amount of fine and detailed edges. These values set the overall performance of the algorithm in the steps of double thresholding and hysteresis edge tracking. In the double thresholding step, pixels are marked and categorized according to their gradient magnitude. Setting too high threshold values would miss some of the fine, detailed edges since they might have lower gradient magnitude values than high threshold value. Likewise, too high threshold values would have impact on performance of hysteresis edge tracking step.

5. In real world applications noise occuring in the image may vary depending on many variables, namely, the lightning conditions of the environment in which the camera set up or the type of sensor that are being used. For that reason, modeling the noise with Gaussian noise may not describe the real world for each case of application.

6. A tentative flow of solution can be used for many generic vision problems:

   (a) Pre-processing (Preparation)
      i. Data acquisition
      ii. Data reduction (Downsampling, color space change (e.g. RGB, $\rightarrow$, Grayscale, defining region of interest...) (if needed)
      iii. Noise reduction (Morphological Operations)

   (b) Processing (Analysis)
      i. Detection
         A. Object Detection (Either, or combinations of the steps given below)
            - Background Substraction
            - Feature (Color, geometry, edge, depth ...) based (Connected Components may be utilized with the combination of thresholding)
            - Texture based
         B. Defect detection on the detected object
            - Feature based

  - Color consistency
  - Geometrical Properties (Hough circles and/or lines can be fitted to check shape defects)
  - Depth data also may give hint on the surface imperfections
- Texture based (Imperfections would change the texture statistics)

2 Programming Problem (Seam Carving)

1. Width Reducing

   (a) Prague (Width Resizing)

      i. Originial Image:

Figure 1: Input Image (Prague, 640 × 480)

ii. Resized Image:

Figure 2: Resized Image (Prague, Width, 540 × 480)

iii. Comparison:

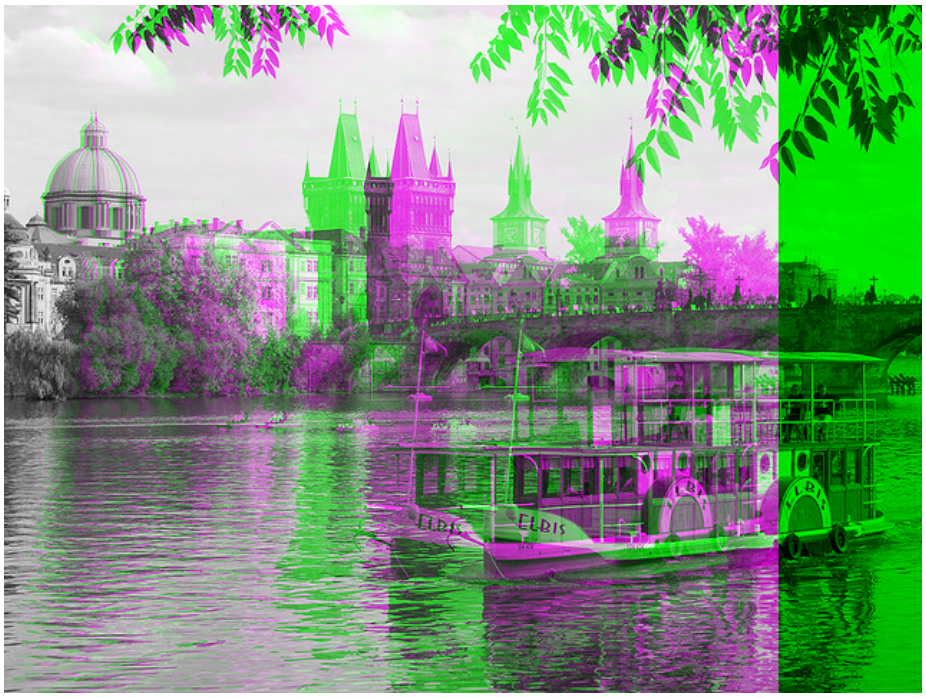

Figure 3: Comparison Image

Legend for comparison image.

- Gray pixels: Common (shared) pixels on both images.
- Green pixels: The pixels that output image introduces.
- Magenta pixels: The pixels that output image lacks.

(b) Mall (Width Resizing)

i. Originial Image:

Figure 4: Input Image (Mall, 775 × 769)

ii. Resized Image:

Figure 5: Resized Image (Mall, Width, 675 $\times$ 769)

iii. Comparison:

Figure 6: Comparison Image

Legend for comparison image.

- Gray pixels: Common (shared) pixels on both images.
- Green pixels: The pixels that output image introduces.
- Magenta pixels: The pixels that output image lacks.

(c) Script: SeamCarvingReduceWidth.m

```
% Clear command window and workspace,
    close all figures
close all, clear all, clc;
profile on;

p = profile('status');
suffix = 'Prague';
% Load the image and show the original
    work
inputFile = ['inputSeamCarving', suffix, '
    .jpg'];
outputFile= ['outputReduceWidth', suffix,
    '.png'];

frame = imread(inputFile);

[h, w, c] = size(frame);
% Compute and acquire the energy map
% TODO - imgradient requires me to pass
    grayscale image!
energyMap = energy_image(frame);

newImage = frame;
newImageGreedy = frame;
newEnergyMap = energyMap;
newEnergyMapGreedy = energyMap;
for k=1:100
          [newImage, newEnergyMap] =
              reduceWidth(newImage,
              newEnergyMap);
     [newImageGreedy, newEnergyMapGreedy] =
         reduceWidthGreedy(newImageGreedy,
        newEnergyMapGreedy);
end

imwrite(newImage, outputFile);
comparisonOriginal = imfuse(frame,newImage
    ,'falsecolor');
```

```
imwrite(comparisonOriginal, ['
    outputReduceWidthInputvsDynamic',
    suffix, '.png']);
comparisonOutput = imfuse(newImage,
    newImageGreedy,'falsecolor');
imwrite(comparisonOutput, ['
    outputReduceWidthComparisonOutputs',
    suffix, '.png']);
figure(1);
subplot(2,2,1), imshow(frame), title('
    Input Image');
subplot(2,2,2), imshow(comparisonOriginal)
    , title('Comparison (Input and Output)'
    );
subplot(2,2,3), imshow(newImage), title('
    Output Image (Dynamic Programming)');
subplot(2,2,4), imshow(newImageGreedy),
    title('Output Image (The Greedy Method)
    ');
saveas(1, ['outputWidth', suffix, '.png'],
    'png');
profile viewer
```

2. Height Resizing

   (a) Prague (Height Resizing)

      i. Originial Image: Please see Figure-1

      ii. Resized Image:

Figure 7: Resized Image (Prague, Height, 640 × 380)

iii. Comparison:

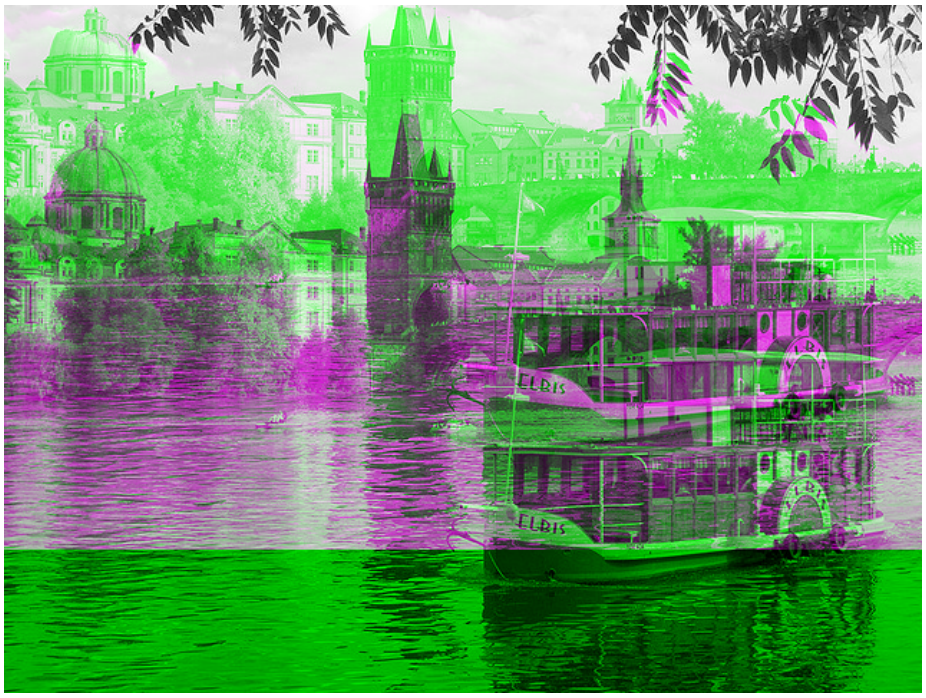

Figure 8: Comparison of Input Image and Output of Dynamic Programming Implementation

Legend for comparison image.

- Gray pixels: Common (shared) pixels on both images.
- Green pixels: The pixels that output image introduces.
- Magenta pixels: The pixels that output image lacks.

(b) Mall (Height Resizing)

i. Originial Image: Please see Figure-4

ii. Resized Image:

Figure 9: Resized Image (Mall, Height, 775 $\times$ 669)

iii. Comparison:

Figure 10: Comparison of Input Image and Output of Dynamic Programming Implementation

Legend for comparison image.

- Gray pixels: Common (shared) pixels on both images.
- Green pixels: The pixels that output image introduces.
- Magenta pixels: The pixels that output image lacks.

(c) Script: SeamCarvingReduceHeight.m

```
% Clear command window and workspace,
    close all figures
close all, clear all, clc;
profile on;

p = profile('status');
suffix = 'Mall';
% Load the image and show the original
    work
inputFile = ['inputSeamCarving', suffix, '
    .jpg'];
outputFile= ['outputReduceHeight', suffix,
     '.png'];

frame = imread(inputFile);

[h, w, c] = size(frame);
% Compute and acquire the energy map
% TODO - imgradient requires me to pass
    grayscale image!
energyMap = energy_image(frame);

newImage = frame;
newImageGreedy = frame;
newEnergyMap = energyMap;
newEnergyMapGreedy = energyMap;
for k=1:100
     [newImage, newEnergyMap] =
         reduceHeight(newImage,newEnergyMap)
         ;
     [newImageGreedy, newEnergyMapGreedy] =
          reduceHeightGreedy(newImageGreedy,
         newEnergyMapGreedy);
end

imwrite(newImage, outputFile);
comparisonOriginal = imfuse(frame,newImage
    ,'falsecolor');
```

```
imwrite(comparisonOriginal, ['
    outputReduceHeightInputvsDynamic',
    suffix, '.png']);
comparisonOutput = imfuse(newImage,
    newImageGreedy,'falsecolor');
imwrite(comparisonOutput, ['
    outputReduceHeightComparisonOutputs',
    suffix, '.png']);
figure(1);
subplot(2,2,1), imshow(frame), title('
    Input Image');
subplot(2,2,2), imshowpair(newImage,
    newImageGreedy), title('Comparison');
subplot(2,2,3), imshow(newImage), title('
    Output Image (Dynamic Programming)');
subplot(2,2,4), imshow(newImageGreedy),
    title('Output Image (The Greedy Method)
    ');
saveas(1, ['outputHeight', suffix, '.png'
    ],'png');
profile viewer
```

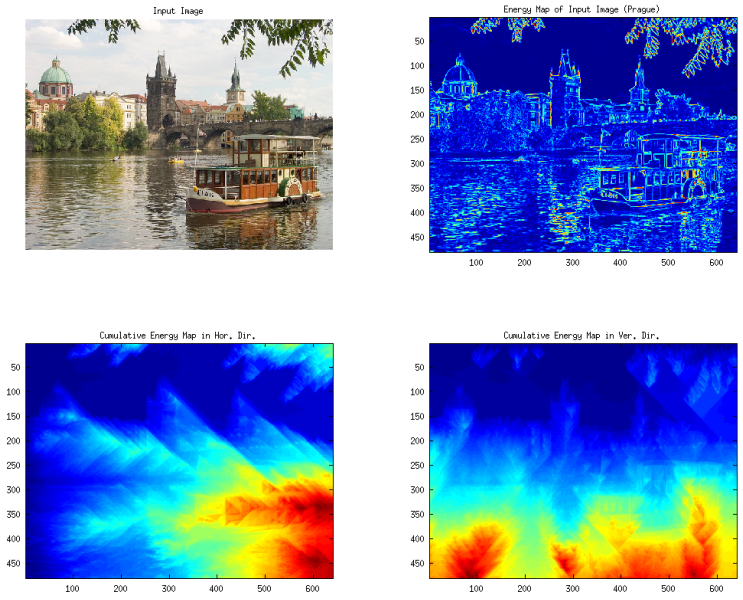

Figure 11: Input Image, Energy Map and Cumulative Energy Functions

3. As seen on Figure-11, the elements of horizontal cumulative energy map is increasing through west to east direction since it is the same direction which the algorithm adds up minimum 8-connected element computed in the previous step.
   Likewise, the elements in vertical cumulative energy map is increasing in north to south direction.
   Due to the fact that energy level of each pixel is not still, the final distribution of accummulated energy levels vary. The horizontal cumulative energy map adds up the bottom region since the ripples of the water surface and the reflection caused by these ripples creates high differences among neighboring pixels. Along with that, the lowest region of the same map is where all the pixels related to sky were add up.

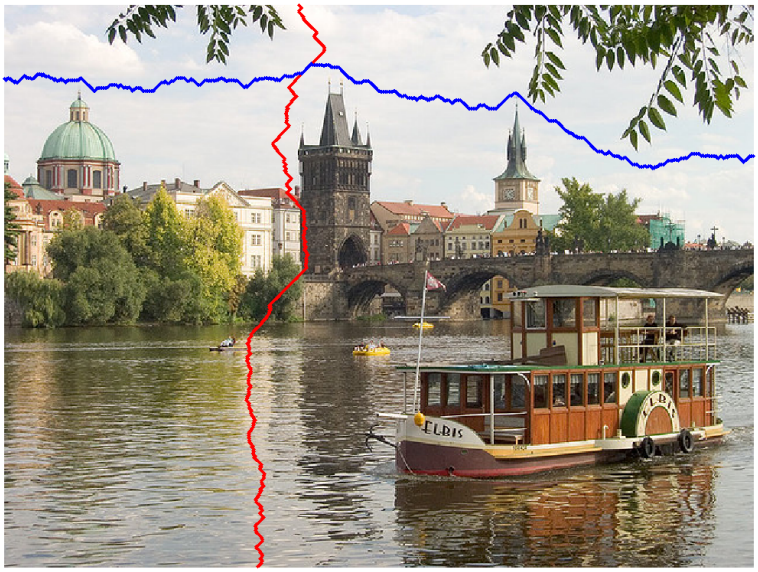

Figure 12: Red Pixels: Optimal Vertical Seam, Blue Pixels: Optimal Horizontal Seam

4. As the cumulative energy maps show in Figure-11, eastern part of the horizontal cumulative energy map and southern part of the vertical cumulative energy map display the highest energy accumulation, where dynamic programming algorithm accumulates the energy differences of neighboring pixels. Starting from minima the last row or column of cumulative energy map, the algorithm finds the lowest energy accumulation within 8-connected neighbors. This way, the algorithm tries to find the seam which connects the lowest energy pixels without leaving the 8-connected region.
   In other words, the algorithm chooses the lowest energy elements by finding from most bottom row or from most right column, for vertical or horizontal resizing, respectively. The algorithm then proceeds choosing one row above or one column left with the same procedure.
   Hence, the algorithm finally results finding the seam which con-

necting lowest energy elements without choosing too far away elements. (Please see: Figure-12)

5. New energy function used is Laplacian of Gaussian;

```
function energyMap = energy_image(im)
%%% New Energy Function Laplacian of
    Gaussian (LoG)
frameGray = rgb2gray(im);
LoG = conv2(fspecial('laplacian'),
    fspecial('gaussian'));
energyMap = imfilter(frameGray, LoG);
end
```

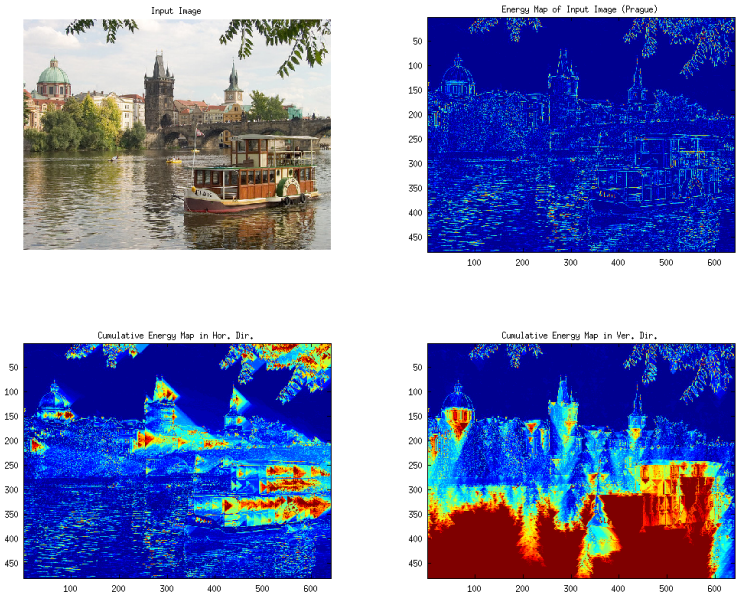

Figure 13: New Cumulative Energy Map with LoG

As Figure-13 illustrates; applying LoG filter rather than Prewitt makes algorithm less aware of the changes since, Gaussian has smoothing effect. Thus, the position of first seams drastically changed. To see the change please compare 12 and 14.

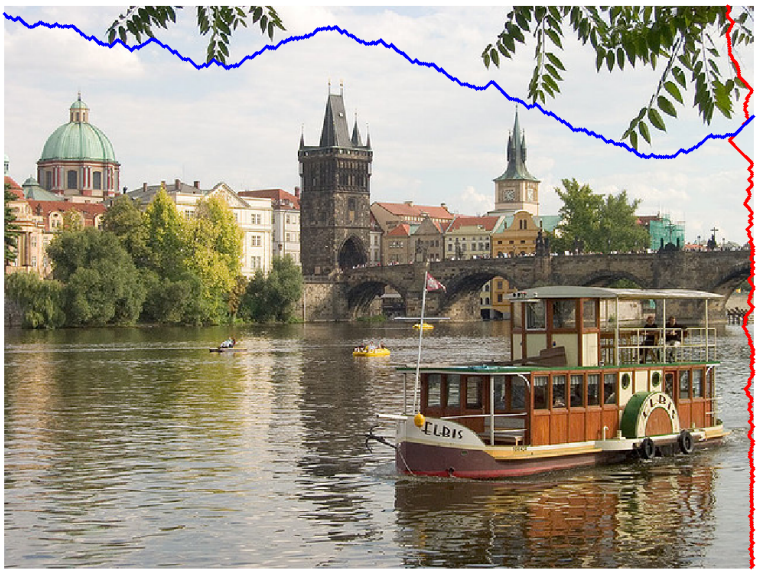

Figure 14: The first seams with LoG, Red and Blue Pixels depict Optimal and Horizontal Vertical Seams, respectively.

6. Qualitative Results

   (a) N. Korea Shot Taken From ISS (**Partially Failed!**) Source
   Because the glare on the eastern side of Earth is relatively smooth (see Figure-15), the algorithm carved through that area to protect to content of the image while reducing the width (see Figure-16). While this behavior of the algorithm is expected to be seen to some extent, 100 pixel width reduction created a artifact over that atmospheric region.
   On the other hand, resizing the height of the image succeeds by eleminating the dark pixels of space. After this operation, the view gains more cinematic view thanks to the new aspect ratio and enables viewers to visulize the vast-depth of space. (see Figure-18)
   For both outputs, the pixels belonging to North Korea are subject to erosion (besides Pyongyang, the capital), since the gradient of that area is not high as the neighboring

regions.
In conclusion, Seam Carving displayed better performance while reducing the height than Matlab did (see Figure-20).

- Input:

Figure 15: Input Image, 532 × 354

- Output:

Figure 16: Seam Carving Result Image, 432 × 354

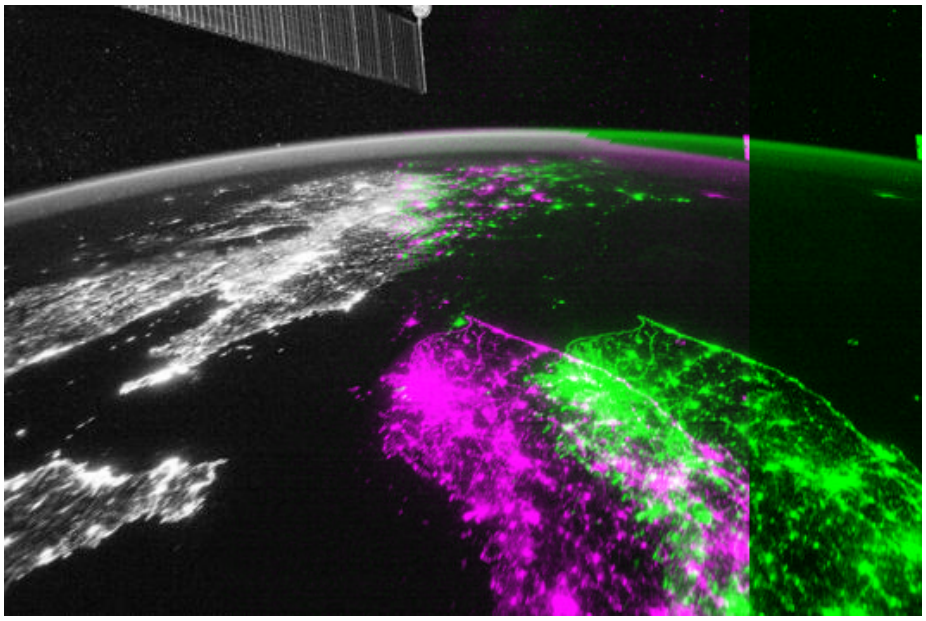

Figure 17: Input Image, Seam Carving Result Image

Figure 18: Seam Carving Result Image, 532 × 254

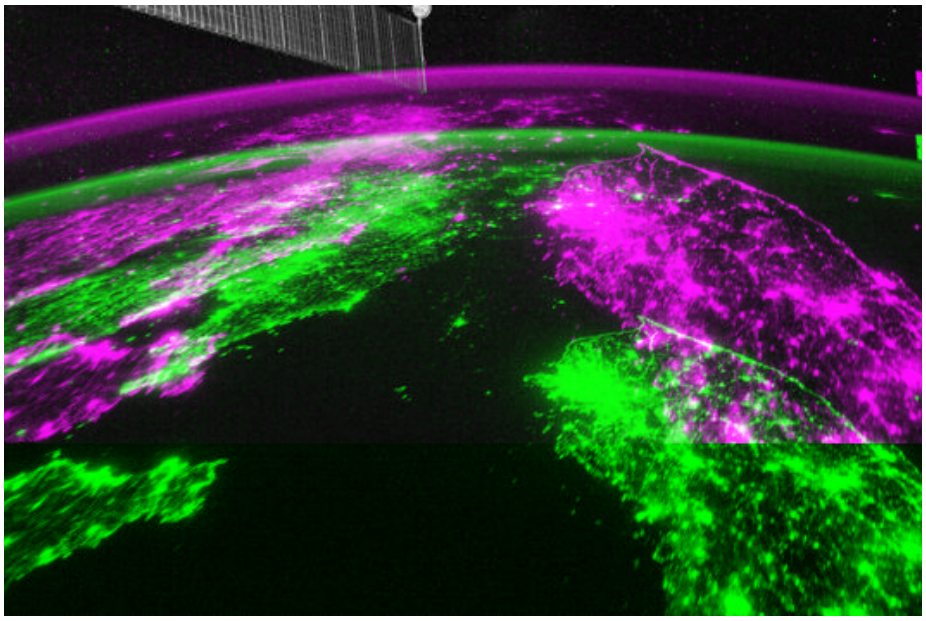

Figure 19: Input Image, Seam Carving Result Image

- Matlab (imresize):

Figure 20: imresize() Result, 432 × 354

(b) Central Park, Manhattan Source
This image was chosen intentionally since the content has the lowest gradient region, while background has the highest. Therefore it is expected to see that the algorithm would try to keep background (in this case the blocks of Manhattan) while deleting seams mostly regions where Central Park can be seen. The result, on the other hand, confirmed the theory to some extent. Width resizing operation was suprisingly accomplished (Figure-22 and Figure-23). As opposed to successful utilization of width resizing, the algorithm could not perform the same qualitative result in the height reducing for Central Park image (Figure-24 and Figure-25). The main reason that make the algorithm fail may be the aspect ratio of the input image. Since the height of source image is 360 pixels, removing 100 pixels

force algorithm to remove seams passing through Central Park. To eleminate that kind of failure scenarios, a qualitative parameter could be introduced to impede the algorithm from deleting seams passing through main object.

- Input:

Figure 21: Input Image, 640 × 360

- Output:

Figure 22: Seam Carving Result Image, 540 × 360

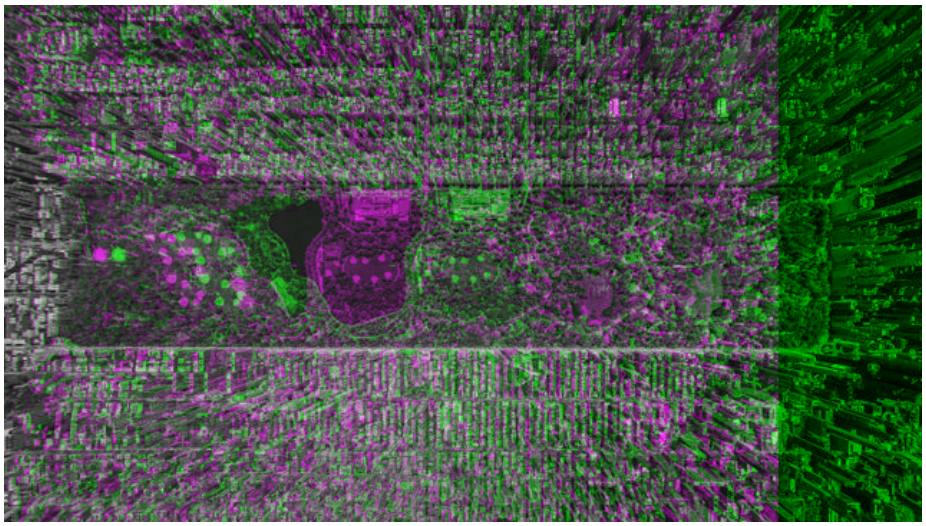

Figure 23: Input Image, Seam Carving Result Image

Figure 24: Seam Carving Result Image, 640 × 260

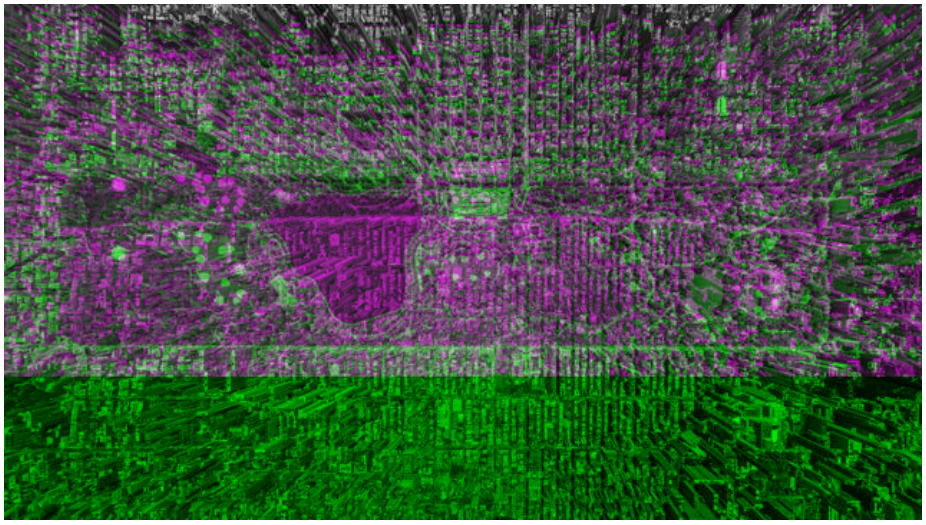

Figure 25: Input Image, Seam Carving Result Image

- Matlab (imresize):

Figure 26: imresize() Result, 540 × 360

(c) Penn Station, New York City Source
This image is used to analyze the performance of the algorithm where the content provides many edges creating big variances in the energy image. Considering that these variances will be added up in the cumulative energy map to find optimal seams, the algorithm is expected to have diffuculties for choosing the seam connecting the lowest energy regions.
As it can be seen in output image depicted in Figure-28 and the comparison image depicted in Figure-29, the algorithm exploited the fact that the picture is old and folded. The content has many details created by difference in the illumination which resulted in high contrast and detailed edges in the roof regions, while the floor and walls are depicted relatively smooth the picture is faded in these areas. Finally, the seam carving algorithm could successively resize the input image, which can be seen in the Figures-28 and 30.

- Input:

Figure 27: Input Image, 550 × 449

- Output:

Figure 28: Seam Carving Result Image, 450 × 449

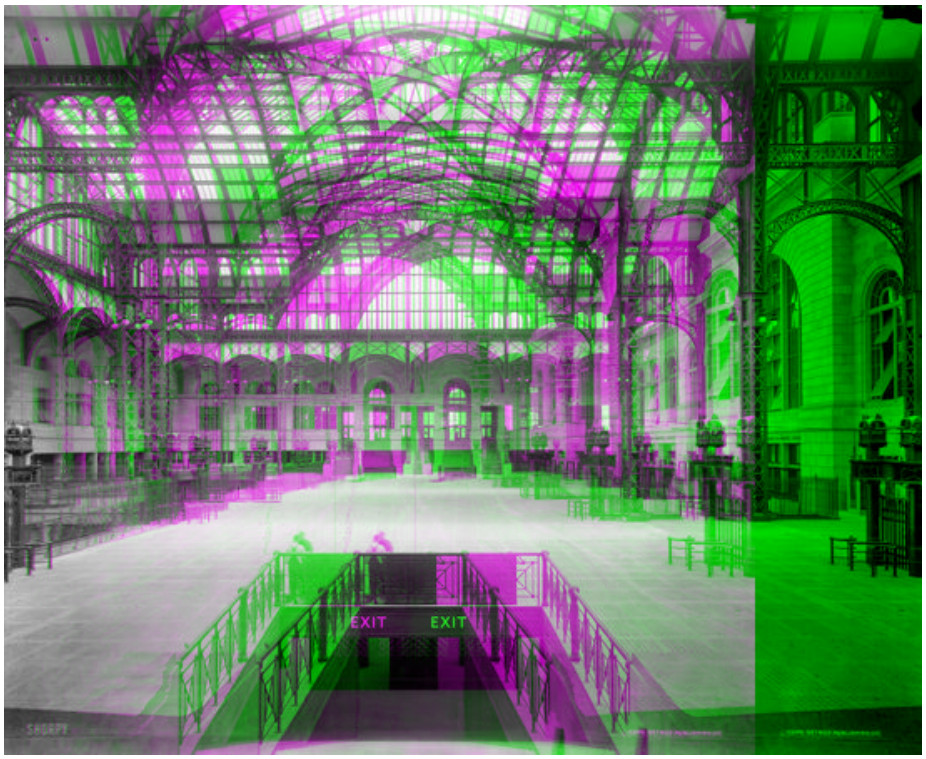

Figure 29: Input Image, Seam Carving Result Image

Figure 30: Seam Carving Result Image, 550 × 349

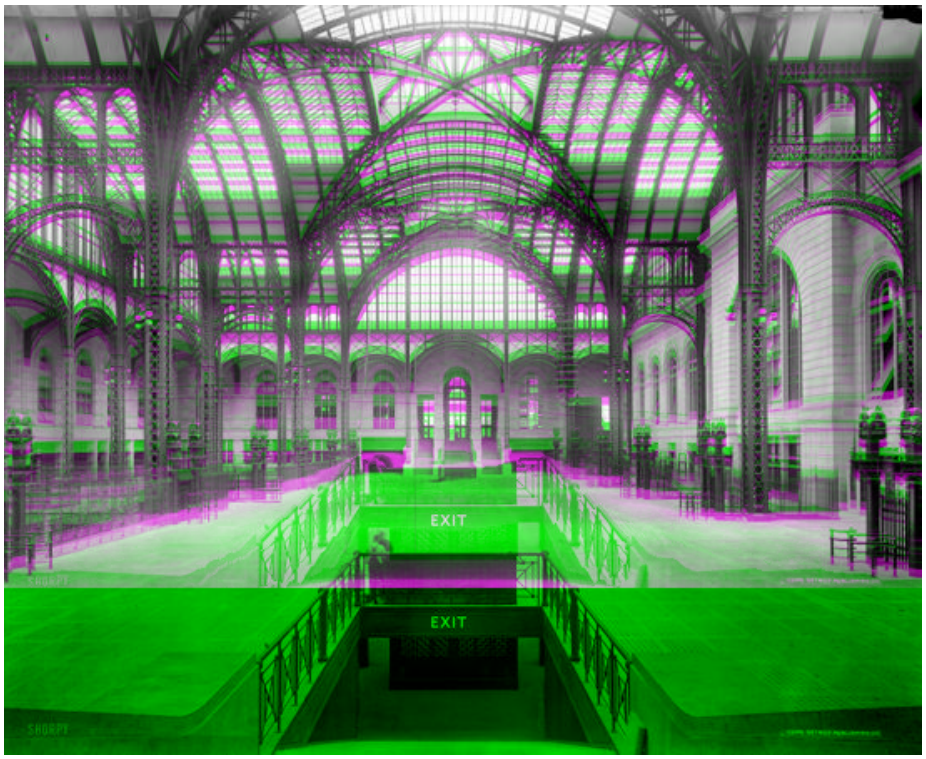

Figure 31: Input Image, Seam Carving Result Image

- Matlab (imresize):

Figure 32: imresize() Result, 540 × 360

(d) Burruss Hall Webcam, Virginia Tech Source
This snapshot is taken from a webcam placed on top of Burruss Hall. The main objective for utilizing the seam carving algorithm is that to protect Drillfield and the students on it.
For width resizing the algorithm has showed siginificant success, since there is no deformation or defect can be seen on the output image (Figure-34).
As for the height reduction, which can be seen in Figure-36, the algoritm has processed the image as expected by choosing the seams from the region where sky is depicted.

- Input:

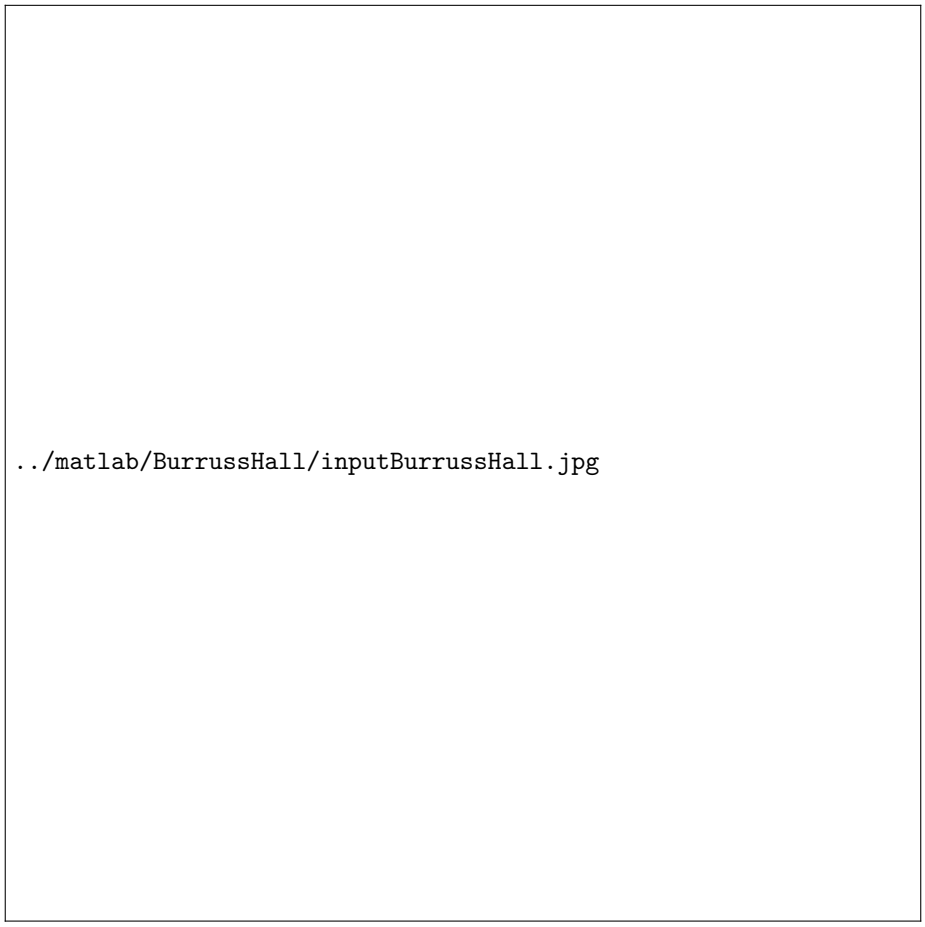

Figure 33: Input Image, 640 × 480

- Output:

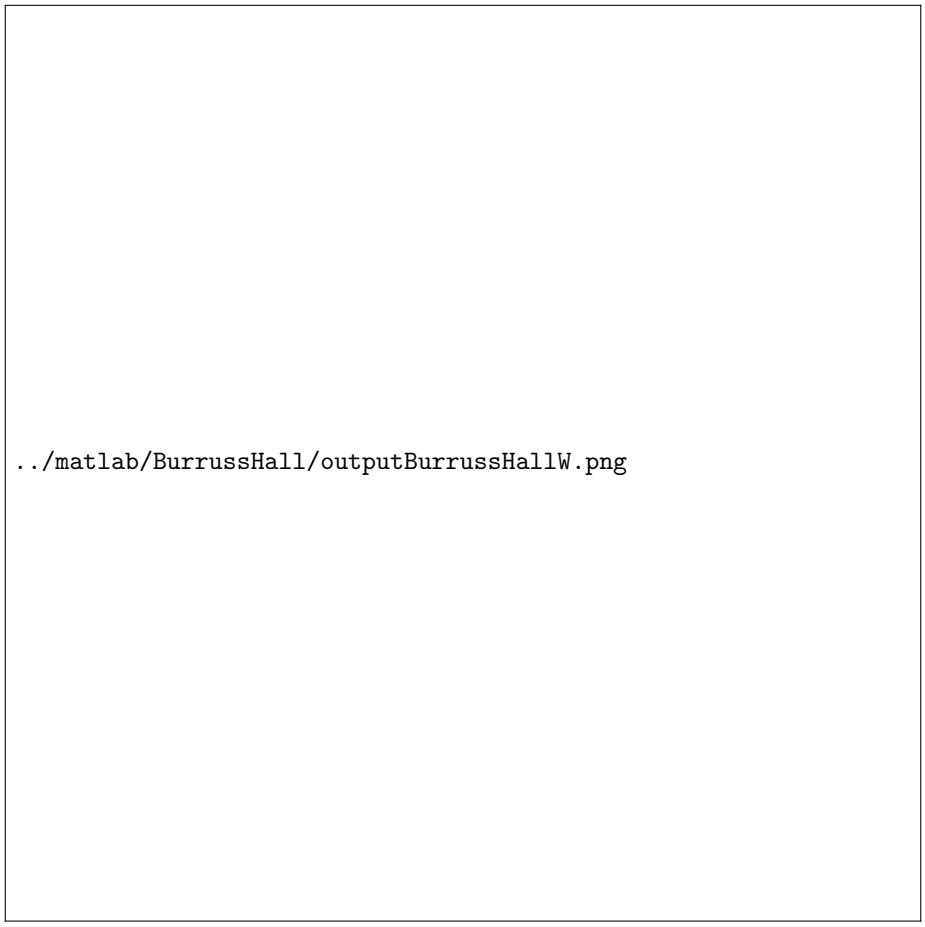

Figure 34: Seam Carving Result Image, 540 × 480

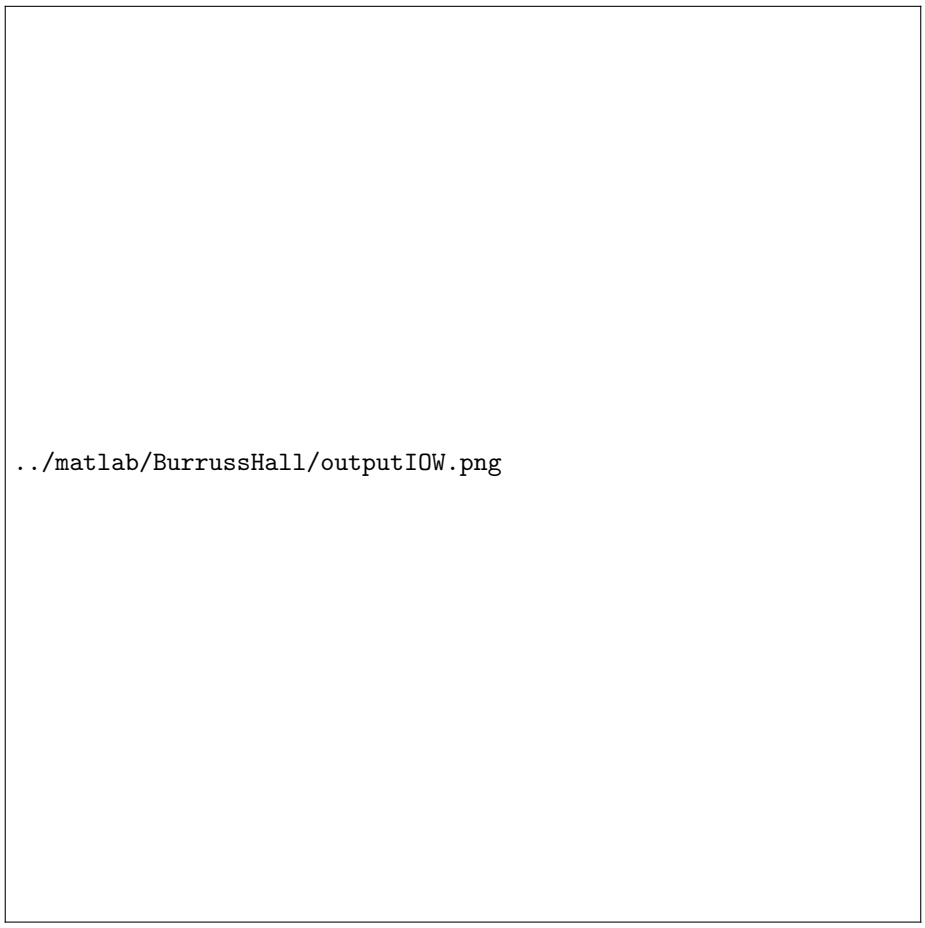

Figure 35: Input Image, Seam Carving Result Image

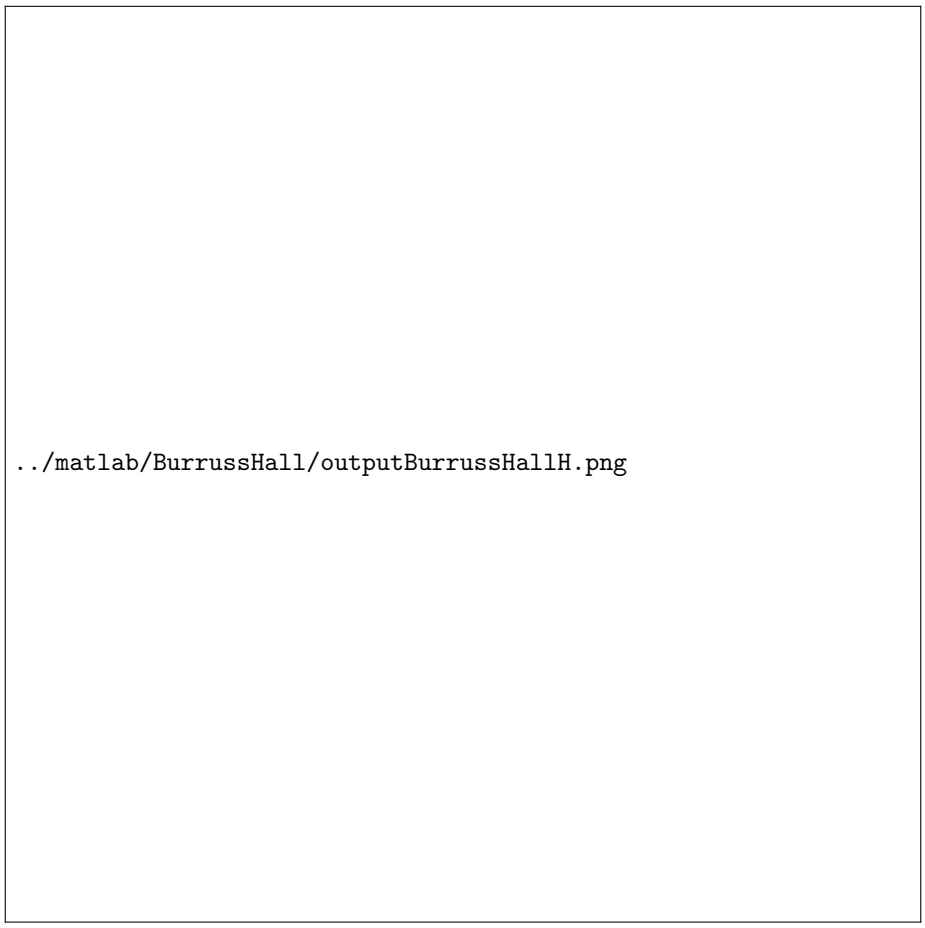

Figure 36: Seam Carving Result Image, 640 × 380

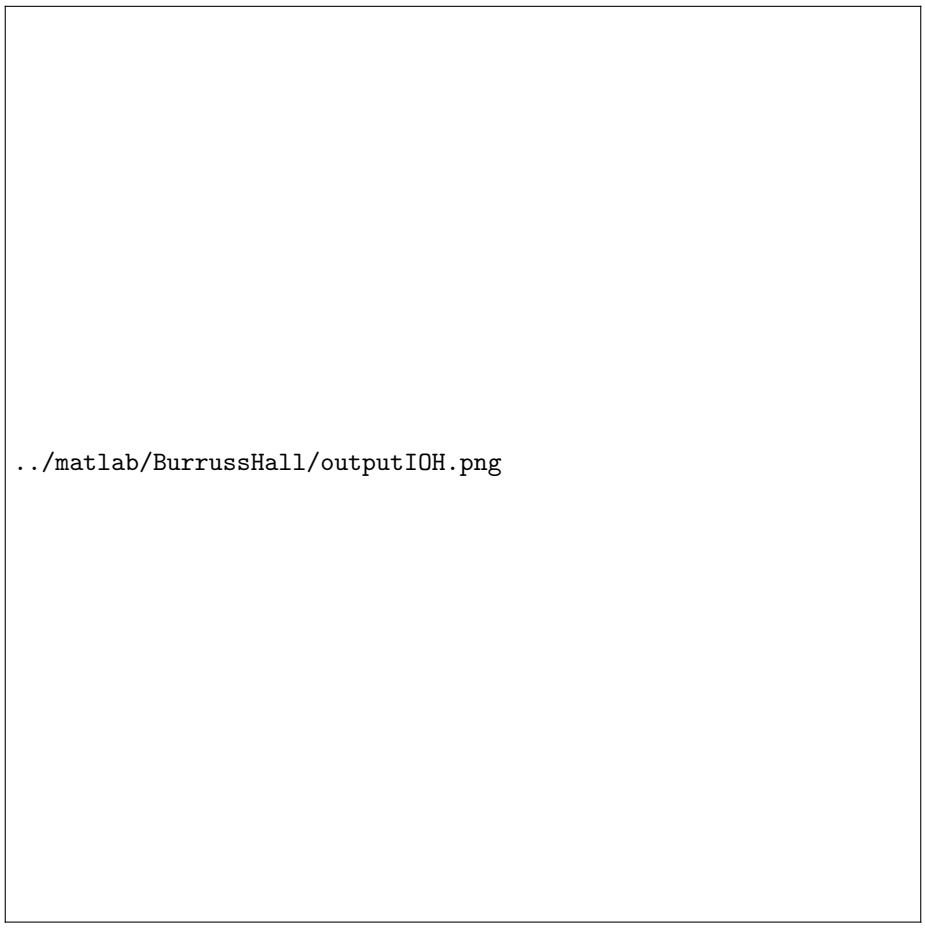

Figure 37: Input Image, Seam Carving Result Image

- Matlab (imresize):

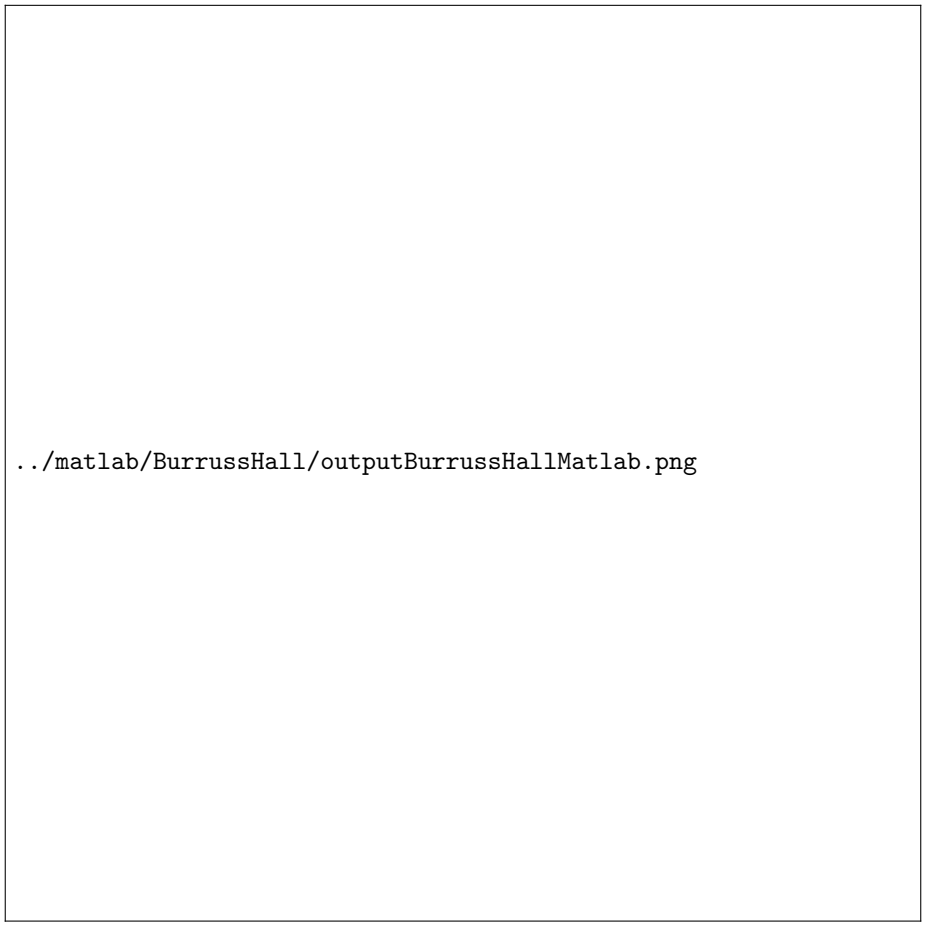

Figure 38: imresize() Result, 540 × 480

3 Extra Credit

4. **Width Increasing by Seam Carving**
The code below (Function-4) utilizes the general steps of Seam Carving algorithm except adding pixels where the seam passed through by averaging the east and weast neighbors of the individual pixel.

$$\forall \vec{I[i]} = [I[i][1:j], mean(I[i][j-1], I[i][j+1]), I[i][j+1:end]]$$

where i is the ith row of the image and j is the optimal seam position for that row. The input image is depicted in Figure-1, while output can be seen in Figure-39.

```
% TODO Add comments for the code and the
    function
function [newImage, newEnergyMap] =
    increaseWidthGreedy(image, energyMap)
     if(nargin~=2)
          errorMessage = 'Missing argument';
          error(errorMessage);
     end
     [h, w, c] = size(image);
     newImage = zeros(h, w+1,c, 'uint8');
     newEnergyMap = zeros(h, w+1);
     % Find the optimal vertical seam
     optimalVerticalSeam =
        find_optimal_vertical_seam(energyMap);
          displaySeam(image,
             optimalVerticalSeam, 'v');
%    add the seam
%%  optimal solution, yet does not work
%         newImage = reshape(image, [h*(w),3]);
%    murat = [(1:h); optimalVerticalSeam];
%    indexes = sub2ind([h,w],murat(1),murat(2)
    );
%         newImage(indexes) = [];
%         newImage = reshape(newImage, h, c-1,
    3);
%%  rookie solution
     for row=1:h
```

```
            lower = max(optimalVerticalSeam(row)
                -1,1);
            higher = min(optimalVerticalSeam(row)
                +1, w);
            roiR = image(row,:,1);
            roiG = image(row,:,2);
            roiB = image(row,:,3);
            roiR1 = [roiR(1:optimalVerticalSeam(
                row)), mean(roiR(lower):roiR(
                higher)), roiR(optimalVerticalSeam
                (row)+1:end)];
            roiG1 = [roiG(1:optimalVerticalSeam(
                row)), mean(roiG(lower):roiG(
                higher)), roiG(optimalVerticalSeam
                (row)+1:end)];
            roiB1 = [roiB(1:optimalVerticalSeam(
                row)), mean(roiB(lower):roiB(
                higher)), roiB(optimalVerticalSeam
                (row)+1:end)];
            newImage(row,:,1) = roiR1;
            newImage(row,:,2) = roiG1;
            newImage(row,:,3) = roiB1;
        end
        % re-create energy map
        newEnergyMap = energy_image(newImage);
    end
```

Figure 39: Width Increased Image

Figure 40: Comparison between Input and Output

5. **The Greedy Approach**
   The code below (Function-5) facilitates the process by reducing the processed data by eleminating computing the cumulative energy map. Instead of computing that vast amount of data, the greedy approach starts finding minimum values for each row at the first row. The algorithm then proceeds finding minimum 8-connected neighbors of that individual pixel until reaches to the last row. Albeit faster (see Figure-41) than the dynamic programming solution, it is prone to local minimas while finding seams. This probable proneness of error has resulted in low-quality resizing cases. Such cases can be seen in Figure-44 in the form of bent tower.
   Along with these qualitative results, Figure-41 clearly shows that the latter algorithm spends four times more time while creating cumulative energy maps, while the greedy algorithms skips that part.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
increaseWidth	100	167.097 s	5.600 s	
cumulative_energy_map	100	125.461 s	125.461 s	
displaySeam	200	58.974 s	56.195 s	
increaseWidthGreedy	100	41.592 s	3.575 s	
energy_image	201	8.725 s	0.017 s	
imgradient	201	8.102 s	6.172 s	
mean	288000	4.986 s	4.986 s	
imshow	204	2.597 s	0.260 s	
imgradientxy	201	1.878 s	0.113 s	
imfilter	402	1.653 s	0.035 s	
newplot	708	1.472 s	0.104 s	
newplot>ObserveAxesNextPlot	708	1.324 s	0.061 s	
cla	508	1.264 s	0.017 s	
graphics/private/clo	508	1.246 s	0.502 s	
find_optimal_vertical_seam	200	1.004 s	1.004 s	
imuitools/private/basicImageDisplay	204	0.909 s	0.268 s	
padarray	408	0.736 s	0.476 s	

Figure 41: Matlab Profiling Results

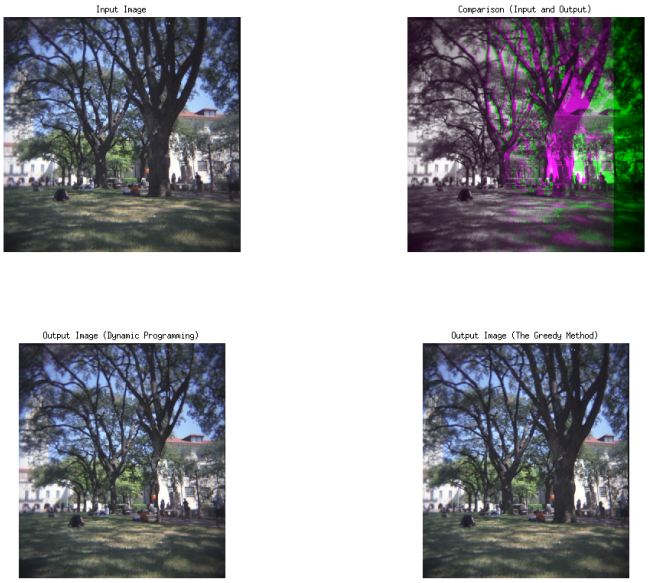

Figure 42: Input Image Mall

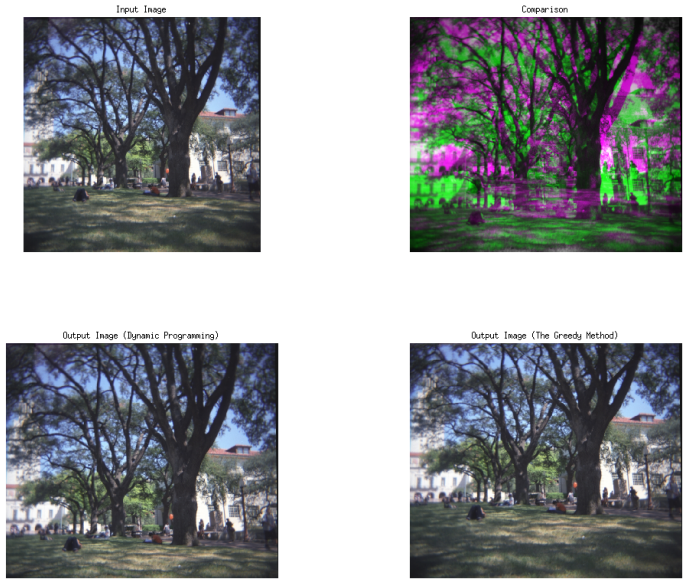

Figure 43: Input Image Mall

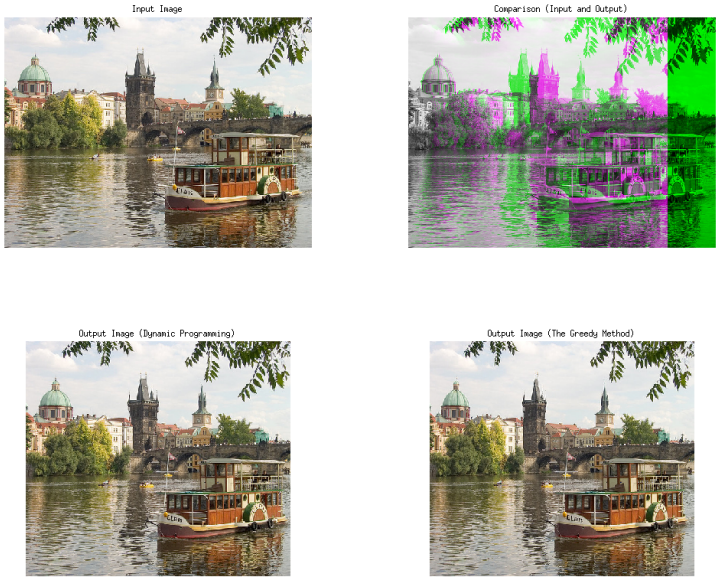

Figure 44: Input Image Prague

Figure 45: Input Image Prague

```
% Clear command window and workspace, close
    all figures
close all, clear all, clc;
profile on;

p = profile('status');
suffix = 'Prague';
% Load the image and show the original work
inputFile = ['inputSeamCarving', suffix, '.
    jpg'];
outputFile= ['outputIncreaseWidth', suffix, '
    .png'];

frame = imread(inputFile);

[h, w, c] = size(frame);
% Compute and acquire the energy map
```

```
% TODO - imgradient requires me to pass
    grayscale image!
energyMap = energy_image(frame);

newImage = frame;
newImageGreedy = frame;
newEnergyMap = energyMap;
newEnergyMapGreedy = energyMap;
for k=1:100
        [newImage, newEnergyMap] =
            increaseWidth(newImage,
            newEnergyMap);
    [newImageGreedy, newEnergyMapGreedy] =
        increaseWidthGreedy(newImageGreedy,
        newEnergyMapGreedy);
end

imwrite(newImage, outputFile);
comparisonOriginal = imfuse(frame,newImage,'
    falsecolor');
imwrite(comparisonOriginal, ['
    outputIncreaseWidthInputvsDynamic', suffix
    , '.png']);
comparisonOutput = imfuse(newImage,
    newImageGreedy,'falsecolor');
imwrite(comparisonOutput, ['
    outputIncreaseWidthComparisonOutputs',
    suffix, '.png']);
figure(1);
subplot(2,2,1), imshow(frame), title('Input
    Image');
subplot(2,2,2), imshowpair(newImage,
    newImageGreedy), title('Comparison');
subplot(2,2,3), imshow(newImage), title('
    Output Image (Dynamic Programming)');
subplot(2,2,4), imshow(newImageGreedy), title
    ('Output Image (Dynamic Programming)');
saveas(1, ['outputHeightIncrease', suffix, '.
    png'],'png');
```

profile viewer