Amirkabir University of Technology
(Tehran Polytechnic)

Assignment 2
When Cameras Fail: Image Enhancement in Spatial Domain

Homeworks Guidelines and Policies

- ***What you must hand in.*** It is expected that the students submit an assignment report (HW2_[student_id].pdf) as well as required source codes (.m or .py) into an archive file (HW2_[student_id].zip).
- ***Pay attention to problem types.*** Some problems are required to be solved *by hand* (shown by the icon), and some need to be implemented (shown by the icon).
  Please don't use implementation tools when it is asked to solve the problem by hand, otherwise you'll be penalized and lose some points.
- ***Don't bother typing!*** You are free to solve by-hand problems on a paper and include picture of them in your report. Here, cleanness and readability are of high importance. Images should also have appropriate quality.
- ***Reports are critical.*** Your work will be evaluated mostly by the quality of your report. Don't forget to explain what you have done, and provide enough discussions when it's needed.
- ***Appearance matters!*** In each homework, 5 points (out of a possible 100) belongs to compactness, expressiveness and neatness of your report and codes.
- ***Python is also allowable.*** By default, we assume you implement your codes in MATLAB. If you're using Python, you have to use equivalent functions when it is asked to use specific MATLAB functions.
- ***Be neat and tidy!*** Your codes must be separated for each question, and for each part. For example, you have to create a separate .m file for part b. of question 3. Please name it like p3b.m.
- ***Use bonus points to improve your score.*** Problems with bonus points are marked by the icon. These problems usually include uncovered related topics or those that are only mentioned briefly in the class.
- ***Moodle access is essential.*** Make sure you have access to Moodle because that's where all assignments as well as course announcements are posted on. Homework submissions are also done through Moodle.

- ***Assignment Deadline.*** Please submit your work **before the end of May 8th**.
- ***Delay policy.*** During the semester, students are given 7 free late days which they can use them in their own ways. Afterwards there will be a 25% penalty for every late day, and no more than three late days will be accepted.
- ***Collaboration policy.*** We encourage students to work together, share their findings and utilize all the resources available. However you are not allowed to share codes/answers or use works from the past semesters. Violators will receive a zero for that particular problem.
- ***Any questions?*** If there is any question, please don't hesitate to contact me through **ali.the.special@gmail.com**.

1. Mathematics of Image Point Processing Operations (12 Pts.)

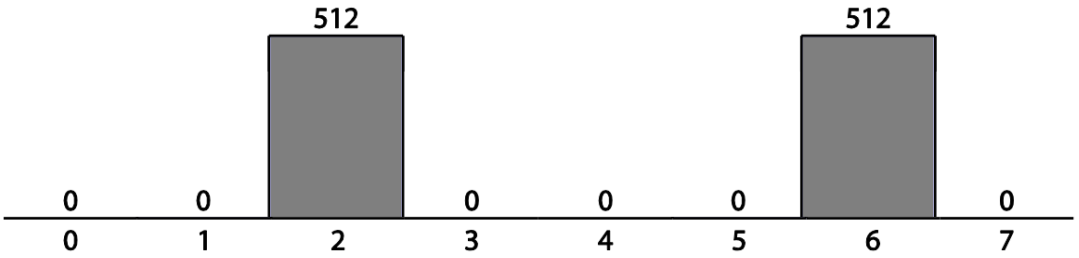
Keywords: *Image Point Processing, Image Histogram, Histogram Equalization, Histogram Matching (Specification), Image Negative, Bit-plane Slicing, Image Thresholding, Contrast Stretching*

The first task contains several problems in the topic of **Point Processing**. Point processing operations are an important group of **Image Enhancement** techniques in **Spatial Domain**, which focus on enhancing images solely with manipulating pixel intensity values. The problems here are aimed to make sure you've theoretically mastered those operations.

First, consider the following 3-bit image of the size 8 × 8:

5	6	5	6	5	2	1	1
4	4	5	4	5	3	2	2
5	5	3	5	6	3	2	2
4	5	3	4	3	5	3	3
4	4	2	3	6	6	5	3
5	4	2	6	6	6	6	3
5	5	4	4	5	4	4	3
2	2	3	3	2	1	2	1

a. Find the digital negative of the image.
b. Perform bit-plane slicing and write down all bit-planes representations of the image.
c. Compute the image histogram.
d. Determine a "valley" in the histogram, and threshold the image at this value. Write down the resultant image matrix.
e. Apply linear scaling for stretching the gray levels of the image to the range of 0-255. Write down the result matrix.
f. Apply histogram equalization to the image. Write the resultant matrix as well as the equalized histogram.
g. Suppose you want to shape the histogram of the above image to match that of a second image, given below:



Use histogram specification to devise a gray-level transformation that shapes the histogram of the original image to look like the second. Note that you have to adopt a strategy to handle certain cases, including when two input values map to the same output value, and no input values map onto a particular output value.

Now, assume an image with the following gray-level histogram:

$$p_r = \frac{e^r - 2}{e - 1}, \quad r \in [0,1]$$

h. Find a gray-level transformation $s = T(r)$ to make the transformed histogram P_s uniform, i.e.:

$$p_s = 1 \quad s \in [0,1]$$

Note: Make sure to show all intermediate steps in the calculations.

2. Who's Moving: A Simple Football Player Detection System (12 Pts.)

Keywords: *Motion Detection, Bit-plane Slicing, Image Difference*

If you think **Bit-plane Slicing** of an image is an old-fashioned image processing operation with no use, think again. Because it not only comes to use in applications like **Image Compression**, but can even handle some high-level machine vision tasks, namely **Motion Detection**. Let's see how.

Imagine two consecutive frames are extracted from a video sequence, and the goal is to detect the moving object(s) inside these frames. One may quickly think of **Image Difference**, but there are more efficient ways available. Assume we first convert the frames into 8-bit grayscale images and then slice grayscale pixel values into eight constituting bits using:

$$a_k = \left\lfloor \frac{I}{2^k} \right\rfloor \bmod 2$$

where I is a grayscale pixel value, k is the bit number and a_k is the bit value of the corresponding bit number (Figure 1).

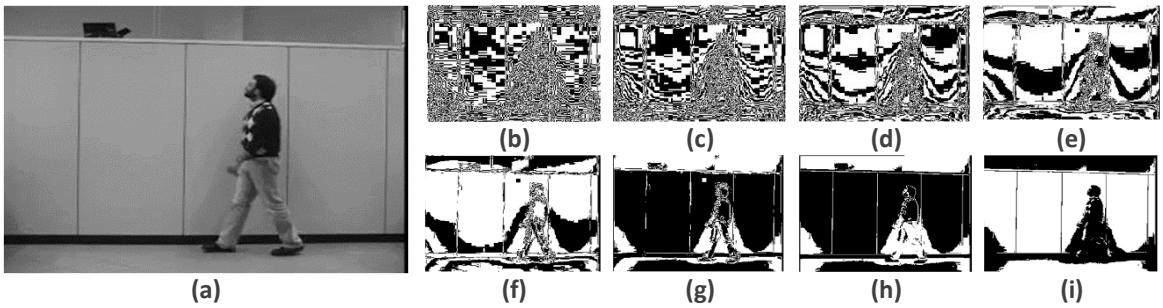


Figure 1 Slicing a frame into bit-plane representations. (a) Original image. (b) to (i) Bit-plane representations from least to most significant bits.

Then, it would be easily possible to compare consecutive frames using XOR function of the corresponding bit-planes, i.e.:

$$c_k = a_k \oplus b_k$$

where a_k, b_k and c_k are bit values of frame 1 pixel, frame 2 pixel and resultant bit value respectively (Figure 2).

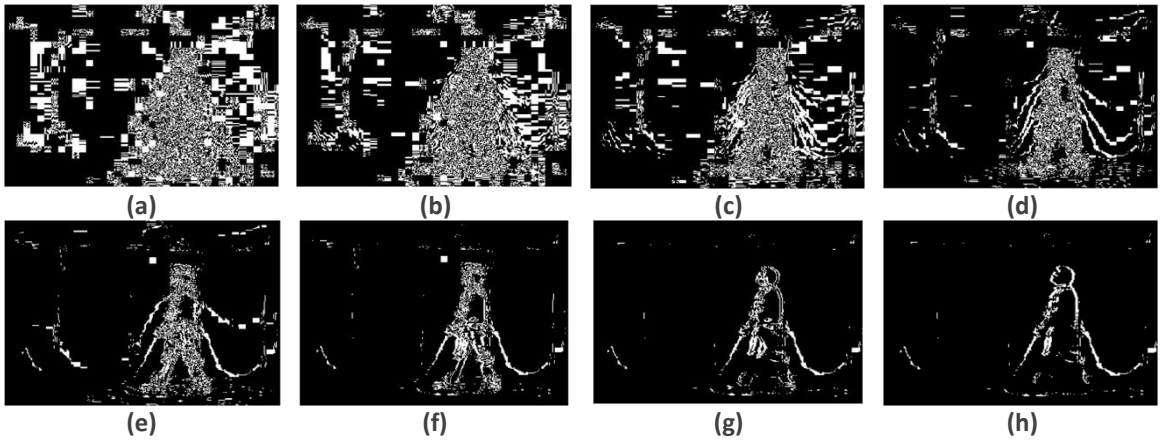
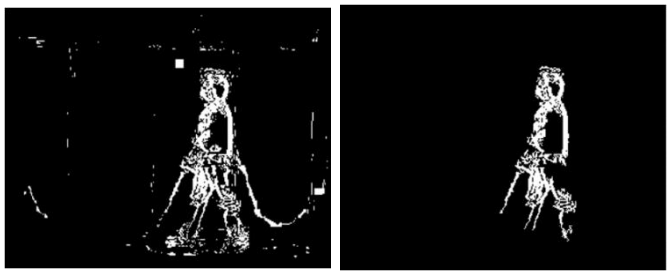


Figure 2 The results of performing XOR function on the corresponding bit-plane representations of the two frames. (a) Least significant bit. (b) to (g) intermediate bit-planes. (h) Most significant bit.

Finally, a grayscale image can be obtained by merging higher bit-planes of the resultant of the XOR operations:

$$Y = \sum_{k=4}^{7} 2^k \times c_k$$

This image is expected to highlight moving regions of the two frames. Some post-processing operations may also be needed to get a more clear result (Figure 3).



(a) **(b)**

Figure 3 Reconstructed image using the four highest bit-planes. (a) Before filtering. (b) After filtering.

In this problem, you are asked to perform the same procedure on three sets of consecutive frames taken from a football match. The final results are expected to highlight players' locations as clear as possible.

a. Implement a function `bitplane_slice()` which takes an input image, converts it into grayscale and then slices it into its bitplanes. Perform bit-plane slicing on the given two frames using your function, and display the results.
b. Perform XOR function on the corresponding bit-planes of the two frames, and display the results.
c. Use the 4 highest bit-planes to obtain moving parts of the two frames.
d. Apply proper image enhancement techniques to obtain a final clean image.

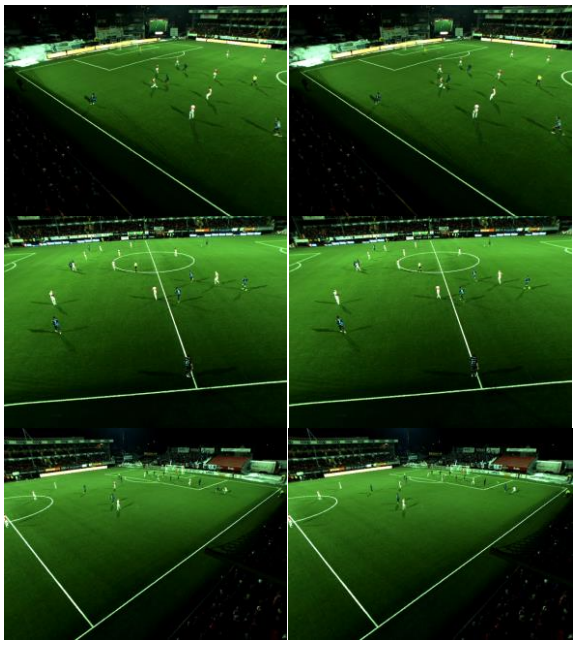


Figure 4 Three sets of consecutive frames of a football match which are used as the inputs of this part.

3. Implementing A Basic Pedestrian Detection/Counting Algorithm (18 Pts.)

Keywords: *Background Subtraction, Foreground Detection, Object Detection, Object Counting, Image Averaging, Image Thresholding, Image Spatial Filtering*

Foreground Detection, also known as **Background Subtraction**, is a machine vision task in which the goal is to detect foreground objects in a given image. These techniques usually analyse video frames recorded with a stationary camera, and distinguish foreground from background based on the changes taking place in the foreground. They are widely used in various applications, such as traffic monitoring, object tracking, human action recognition, and so on.

(a) **(b)**

Figure 5 After detecting background image, it's easy to subtract the original image from the obtained background and find foreground image (a) Original image. (b) Background image.

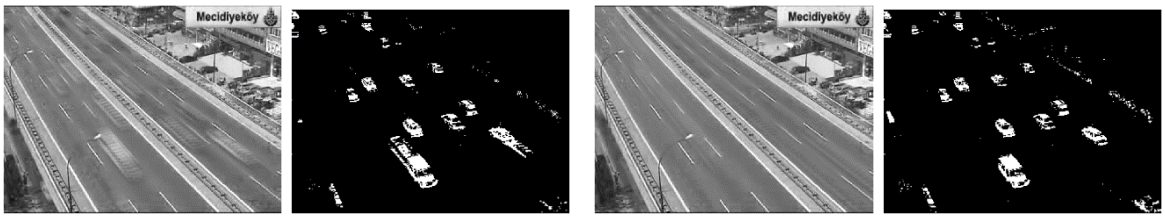
As you learned in the class, **Image Averaging** is a technique used to reduce noises if multiple images of the same scene are available and noise has zero mean and be uncorrelated. In another word, assuming *n* different noisy images I_i, then

$$\bar{I}(x,y) = \frac{1}{n}\sum_{i=1}^{n} I_i(x,y)$$

can be shown to be less noisy.

Now based on this framework and applying **Image Thresholding**, we can introduce a simple background detection algorithm in a video sequence, where background is the mean of the previous *n* frames F_i:

$$B(x,y) = \frac{1}{n}\sum_{i=1}^{n} F_i(x,y)$$



$n = 10$ $n = 50$

Figure 6 The number of images used in the averaging process considerably affects the "cleanness" of the background obtained by the algorithm, and therefore the output foreground image.

Similarly, if the background is more likely to appear in a scene, we can use the median of the previous *n* frames as the background model:

$$B(x,y) = median\left[F_i(x,y)\right]_{i=1}^{n}$$

Your objective is to utilize the same approach to handle a simple pedestrian detection and counting problem. You are provided with 20 frames of a three-second video sequence recorded from a CCTV camera in Oxford town centre, and two test frames of the same scene in a different time. The goal is to detect all pedestrians in the test images by first finding a pedestrian-free image of the area (background) and then subtracting the test images from it, before attempting to count the number of individuals in each of the given test frames.

a. Estimate the background image using the first *n* frames. Set $n = 2, 5, 10, 20$.
b. Subtract the test frame from the background images you obtained in the previous part.
c. Define appropriate thresholds and find foreground masks corresponding to each of the background images.
d. Use the resultant masks to extract individuals in the test frames. Assign black color (0,0,0) to the remaining regions of the image.
e. Repeat this process using median operation, and compare the results.
f. Use the background image obtained in the previous part along with different spatial filters to count the number of pedestrians in the test frames.
   Note: One idea is to reduce the pixels of the connected regions through a loop until they vanish.

Note: Make sure to display and save the intermediate results as well as including them in your report.

(a) **(b)**

Figure 7 Input video sequence. (a) Some extracted frames. (b) Test frames.

4. Image *Obamafication* (!) Using Bilateral Filtering (15 Pts.)

Keywords: *Image Spatial Filtering, Gaussian Filtering, Bilateral Filtering, Image Thresholding*

Barack Obama 2008 presidential campaign was mostly represented by an iconic poster, widely known as "Barack Obama hope poster", with a stencilled face of Obama in solid red, beige and blue (light and dark), and the caption "Hope" below (Figure 8). The poster became one of the most widely recognised symbols of Obama's campaign message, spawning many variations and imitations.

Figure 8 Barack Obama famous "Hope" poster, used in his 2008 presidential campaign.

The poster was created by graphic designer and street artist Shepard Fairey in just one day. In this problem, your task is to create similar effect on two given images. Through this, you'll practice a modified and improved version of **Gaussian Filtering**.

The method we introduce here combines the effects of **Bilateral Filtering** with **Image Thresholding** technique. It first applies bilateral filtering to the image, then converts the result to grayscale and maps the four specific colors to different regions of the images based on user-defined threshold values. Figure 9 depicts this procedure.

In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values:

$$I_F(i,j) = \frac{\sum_{k,l} I(k,l)\omega(i,j,k,l)}{\sum_{k,l} \omega(i,j,k,l)}$$

where (i, j) and (k, l) are the position of the current pixel and neighboring pixel respectively, I_F is the filtered version of the input image I, and ω is the **Bilateral Weight Function** defined as below:

$$\omega(i,j,k,l)=\exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}-\frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)$$

where σ_d and σ_r are **Smoothing Parameters**. Increasing spatial parameter σ_d smooths larger features, while by increasing range parameter σ_r the filter becomes closer to the Gaussian filter.

(a) **(b)** **(c)** **(d)**

Figure 9 Procedure of the proposed method. (a) Original image. (b) The output of performing bilateral filter on the input image. (c) Conversion the filtered image into grayscale. (d) Using image thresholding to color the image.

Now let's roll! Load the input images and use them in the following steps.

a. Implement the function `bilateral_filter()` based on the definition above.
b. Play around with the parameters and display four distinct results under arbitrary values set for these parameters.
c. Convert the image to grayscale and map the colors in table 1 to pixels based on intensity, using appropriate threshold values. You are free to color each region with your desired color.
d. Use the provided masks to separate background and foreground. Color half of the background with the color "desaturated cyan" and the other half with "lava".
e. Argue on the importance of the threshold value.

Table 1 Hope poster colors and their details

Color	Name	RGB Values
	Prussian Blue	(0, 48, 80)
	Desaturated Cyan	(112, 150, 160)
	Peach-Yellow	(250, 227, 173)
	Lava	(218, 20, 21)

(a) **(b)**

Figure 10 Input images and their masks, which are needed to separate background and foreground. (a) Input 1 and its corresponding mask. (b) Input 2 and its corresponding mask.

5. Object Removal Through A Cool Trick (28 Pts.)

Keywords: *Image Filtering, Image Gradient, Image Resizing, Object Removal, Seam Carving, Dynamic Programming*

Have you ever experienced the dilemma of fitting an image into a report? On one hand, you have to reduce the image size as much as possible, on the other hand you prefer to keep it large enough to be visualized clearly. This issue stems from the fact that traditional image resizing methods ignore the details inside an image. In other words, they regard different parts of an image equally, while there are regions which we prefer to be displayed more vividly. The idea of cropping images is also not quite satisfactory, as it totally eradicates parts of an image which may contain valuable details.

Another simple, yet surprisingly smart image resizing technique was introduced in 2007, which enables us to resize images in a way that not only the image size is reduced, but also the main objects are preserved exactly the same. The intuition behind this method, known as **Seam Carving**, is that informative contents inside an image are indicated by higher energy values, which can be calculated through different **Image Filtering** methods. By using an *energy map*, one can simply detect horizontal or vertical *seams* (connected path of low-energy pixels in an image) with lowest energy, and remove them in order to obtain a final resized image.

(a) (b) (c) (d)

Figure 11 Different resizing strategies applied to an image of Broadway Tower in England. (a) Original image. (b) Normal image scaling which has led the castle to be distorted. (c) The result of cropping, which has caused part of the castle to be deleted. (d) Seam carving technique, which has not only reduced image width, but also kept the castle and the man intact.

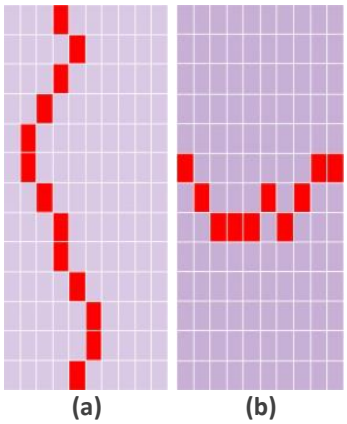
More precisely, Let I be an nxm image and a vertical seam is an 8-connected path in the image from the top to the bottom containing one pixel per row:

$$\mathbf{s}^{\mathbf{x}} = \left\{ s_i^x \right\}_{i=1}^{n} = \left\{ \left(x(i), i \right) \right\}_{i=1}^{n}, \quad \text{s.t.} \quad \forall i, \ \left| x(i) - x(i-1) \right| \le 1$$

Similarly, the horizontal seam is defined as:

$$\mathbf{s}^{\mathbf{y}} = \left\{ s_i^y \right\}_{j=1}^{m} = \left\{ \left(j, y(j) \right) \right\}_{j=1}^{m}, \quad \text{s.t.} \quad \forall j, \ \left| y(j) - y(j-1) \right| \le 1$$

Considering the above definitions, the algorithm can be written in the three main steps:



(a) (b)

Figure 12 Seams are connected path of pixels in the image. (a) Vertical seam. (b) Horizontal seam.

I. Finding the energy map

Energy function is defined as a kind of function that is related with the content in an image. One intuitive way is to use the **Image Gradient**, since the important contents in an image usually have strong local structures whose gradients are high, while contents that are less important often lie in plain area in the image whose gradients are small. Hence, the energy function is defined as:

$$e_1(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

II. Finding the seam with least energy

Given the energy function, the seam cost is defined as:

$$E(\mathbf{s}) = E(\mathbf{I}_\mathbf{s}) = \sum_{i=1}^{n} e\left(\mathbf{I}(s_i)\right)$$

in which $I(S_i)$ is one pixel of the seam. The goal is to find the optimal seam which minimizes seam cost:

$$s^* = \min_{\mathbf{s}} E(\mathbf{s}) = \min_{\mathbf{s}} \sum_{i=1}^{n} e\left(\mathbf{I}(s_i)\right)$$

The optimal seam can be found by dynamic programming. Dynamic programming is a programming method which stores the result of sub-calculations in order to simplify calculating a more complex result. In finding the optimal vertical seam, we first traverse the image:

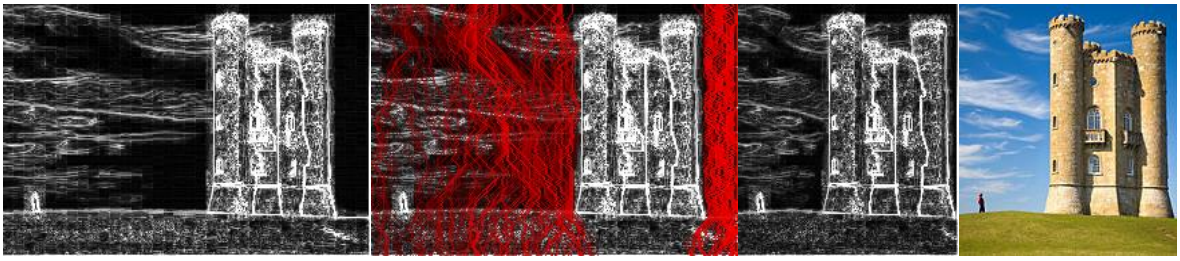
$$M(i, j) = e(i, j) + \min\left(M(i-1, j-1), M(i-1, j), M(i-1, j+1)\right)$$

then we trace back the image from the minimal entry in the last row of the map. The operations for horizontal seam is also similar.

III. Image resizing

So far, a collection of seams, sorted from smallest to highest total energy values across their pixels, have been computed. Assume the goal is to change the size of an image from nxm to nxm', where $m - m' = c$. This can be easily achieved by successively removing c vertical seams from the original image.

The more general situation in which the goal is to resize image from nxm to n'xm' requires additional calculations and will be skipped here.



(a) (b) (c) (d)

Figure 13 The process of seam carving technique. (a) Gradient magnitude of the input image is calculated and used to determine the energy of pixels. (b) Using the energy map, a list of seams are obtained and ranked by their energy. Lower energy seams are of lesser importance to the content of the image. (c) Removing low-energy seams based on their total energy values across their pixels. (d) Final resized image.

Now, let's get our hands dirty. Here, your goal is to implement a seam carving algorithm with the following functions:

- `energy_map = find_energy_map(img)` : It takes an RGB image `img`, converts it to grayscale before computing its energy image `energy_map`, which is a double matrix of the same size as `img` and indicates the energy value at each pixel.

- `seams_list = find_seams_list(energy_map, seam_dir, seam_num)`: It takes a matrix `energy_map` as well as a string `seam_dir` which takes values `'horizontal'` and `'vertical'` which specifies seams direction, and a number `seam_num` which determines the number of seams required. The function must return a `seam_num x img_width` matrix (in case `seam_dir` is `'horizontal'`) with seams ordered by their pixel energy values in its rows, or a `img_height x seam_num` matrix (in case `seam_dir` is `'vertical'`) with seams ordered by their pixel energy values in its columns.
  Hint: Seams must be stored as `1 x img_width` (horizontal) or `img_height x 1` (vertical) arrays where, for example, if their 87th element is 136, it means that in the 87th row (or column) the pixel in row (or column) 136 is to be removed.
- `img_resized = remove_seams(img, seams_list, seam_dir)`: It takes color image `img`, matrix `seams_list` and string `seam_dir`, and attempts to remove pixels from the image according to the ordered list of seams and the direction given, to return a final color image `img_resized` which is the smaller version of the input image.

IV. Object removal

Having a binary mask corresponding to the desired object, similar process can be used to perform the task of object removal. First, the masked region's area is measured to decide whether horizontal or vertical seam carving would be more efficient. Next, areas under the masked region are weighted with a large negative value when generating the image's energy map. This guarantees that the minimum seem will be routed through the masked region. The algorithm is then performed as usual, only when removing the minimum seam from the image, the same minimum seam must also be removed from the mask so that the next step's energy calculation is accurate. After the masked region is completely removed, seem insertion is used to return the image back to its initial dimensions.

If you feel that you have to make additional assumptions (for example, regarding the overlapping seams), please do it by stating them clearly in your report. Except for the functions mentioned above, you are free to implement the remaining parts the way you want.

a. Reduce the width of the image in part (a) in Figure 14 by 10%, 25% and 50%. As well as displaying the final result, display the intermediate results including energy maps and selected seams (similar to Figure 13).
b. Reduce the height of the image in part (b) in Figure 14 by 10%, 25% and 50%. As well as displaying the final result, display the intermediate results including energy maps and selected seams (similar to Figure 13).
c. Find an input image of your own on the web in which this technique obtains interesting successful result (either horizontally or vertically). Display the image as well as the result, and describe why the algorithm seems to work well.
d. Find an input image of your own on the web in which this technique obtains interesting poor result (either horizontally or vertically). Display the image as well as the result, and describe why the algorithm seems to work poorly.

(a) **(b)**

Figure 14 Inputs provided for the resizing part. These images contain regions that are of more importance than the others.

e. Use a similar technique to increase the size of an image. You need to implement a function similar to `remove_seams()`, in which instead of removing seams, they are added. Test your algorithm on all the given images, and increase their size by 10%, 25% and 50%.
f. Use the provided masks associated with the images in Figure 15, and remove the designated individuals from the images using the above-mentioned technique.

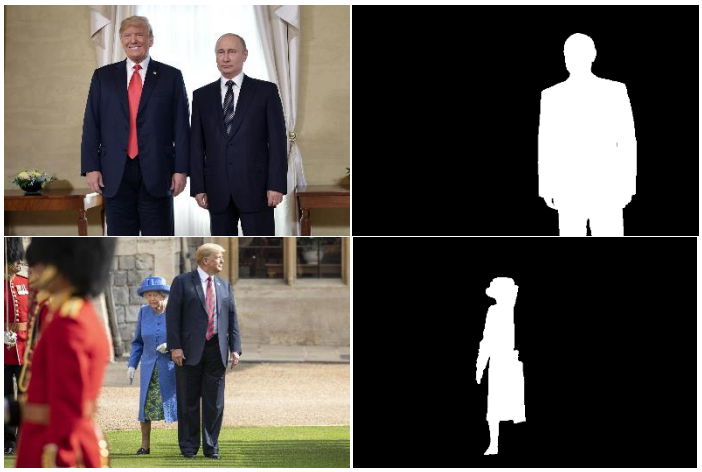


Figure 15 Inputs provided for the object removal part. These images contain individuals that are required to be removed using the given masks.

Recommended MATLAB Functions: `rgb2gray()`, `gradient()`, `fspecial()`, `imfilter()`

6. Some Explanatory Questions (10 Pts.)

Please answer the following questions as clear as possible:

a. Is it technically possible to convolve an image with a spatial filter larger than the image size? Justify your answer.
b. Does repeatedly applying a 3 × 3 Laplacian of Gaussian filter to an image always converge to a certain state? Explain.
c. Is it possible to perform binary template matching using image filtering? If yes, design a 3x3 filter to find all plus signs (+) in an image. If no, explain why.
d. Which one is more efficient: filtering an image with two 1D filters, or filtering it with one 2D filter. Justify your answer.
e. How does image averaging (problem 3) help reducing noise? Explain.

Good Luck!
Ali Abbasi