Project 1
Image Filtering and Hough Transform

Due date: 23:59 Sunday September 22th (2019)

In this project, you will implement some basic image processing algorithms and putting them together to build a Hough Transform based line detector. Your code will be able to find the start and end points of straight line segments in images. We have included a number of images for you to test your line detector code. Like most vision algorithms, the Hough Transform uses a number of parameters whose optimal values are data dependent: The optimal set of parameter values are different for different input images. By running your code on the test images you will learn what these parameters do and how changing their values effects performance.

Many of the algorithms you will be implementing as part of this assignment are functions in the Matlab image processing toolbox. You are not allowed to use calls to functions in this assignment. You may however compare your output to the output generated by the image processing toolboxes to make sure you are on the right track.

1. Instructions

These instructions are also true to all the remaining projects unless indicated. Please read them carefully.

1. Students are encouraged to discuss projects. However, each student needs to write code and a report all by him/herself. Code should NOT be shared or copied. Do NOT use external code unless permitted.
2. Post questions to Piazza so that everybody can share, unless the questions are private. Please look at Piazza first if similar questions have been posted.
3. Generate a zip or tgz package, and upload to coursys (https://coursys.sfu.ca/2019fa-cmpt-412-d1/). The package must contain the following in this layout (they will be different for the other projects but will be similar):
    - {SFUID}
        - {SFUID}.pdf (your write-up, the main document for us to look and grade)
        - matlab
            - myImageFilter.m
            - myEdgeFilter.m
            - myHoughTransform.m
            - myHoughLines.m
            - Any helper functions you need

        - ec
            - myHoughLineSegments.m (you may not use it)
            - ec.m (you may not use it)
            - your own images
            - your own results
4. File paths: Make sure that any file paths that you use are relative and not absolute so that we can easily run code on our end. For instance, you cannot write “imread(‘/some/absolute/path/data/abc.jpg’)”. Write “imread(‘../data/abc.jpg’)” instead.

2. Tasks

We have included a main script named houghScript.m that takes care of reading in images from a directory, making function calls to the various steps of the Hough transform (the functions that you will be implementing) and generates images showing the output and some of the intermediate steps. You are free to modify the script as you want, but note that your script houghScript.m should be executable for our evaluation. Please make sure your code runs correctly with the original script and generates the required output images.

Every script and function you write in this section should be included in the matlab/ directory. Please include resulting images in your write-up.

2.1. Convolution (3 pts)

Write a function that convolves an image with a given convolution filter

function [img1] = myImageFilter(img0, h)

As input, the function takes a greyscale image (img0) and a convolution filter stored in matrix h. The output of the function should be an image img1 of the same size as img0 which results from convolving img0 with h. You can assume that the filter h is odd sized along both dimensions. You will need to handle boundary cases on the edges of the image. For example, when you place a convolution mask on the top left corner of the image, most of the filter mask will lie outside the image. One solution is to output a zero value at all these locations, the better thing to do is to pad the image such that pixels lying outside the image boundary have the same intensity value as the nearest pixel that lies inside the image.

You can call Matlab’s function to pad array. However, your code can not call Matlab’s imfilter, conv2, convn, filter2 functions, or any other similar functions. You may compare your output to these functions for comparison and debugging. This function should be vectorized. Examples and meaning of vectorization can be found here. Specifically, try to reduce the number of for loops that you use in the function as much as possible. “Somehow” visualize and show the results of at least 2 convolution results while changing the kernel. An image can be the same or different. Note that convolution may produce negative intensity or an intensity beyond the normal maximum value (255). “Somehow” visualize results effectively.

2.2. Edge detection (3 pts)

Write a function that finds edge intensity and orientation in an image. Show the output of your function for at least one of the given images in your write-up.

```
function [img1] = myEdgeFilter(img0, sigma)
```

The function will input a greyscale image (img0) and scalar (sigma). Sigma is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. The function will output img1, the edge magnitude image.

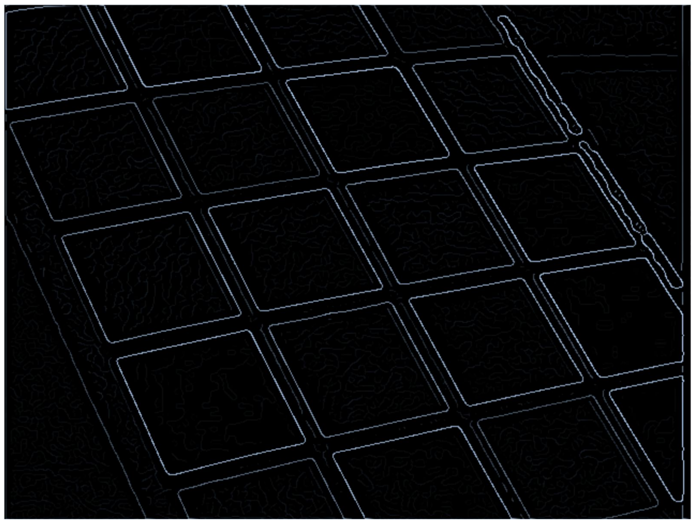
First, use your convolution function to smooth out the image with the specified Gaussian kernel. This helps reduce noise and spurious fine edges in the image. Use fspecial to get the kernel for the Gaussian filter. The size of the Gaussian filter should depend on sigma (e.g., hsize = 2 * ceil(3 * sigma) + 1).

The edge magnitude image img1 can be calculated from image gradients in the x direction and y direction. To find the image gradient *imgx* in the x direction, convolve the smoothed image with the x-oriented Sobel filter. Similarly, find image gradient *imgy* in the y direction by convolving the smoothed image with the y-oriented Sobel filter. You can also output *imgx* and *imgy* if needed.

In many cases, the high gradient magnitude region along an edge will be quite thick. It is best to extract thin edges that have a single pixel width. Towards this end, make your edge filter perform “non-maximum suppression”, that is, for each pixel look at the two neighboring pixels along the gradient direction. If either of those pixels has a larger gradient magnitude then set the edge magnitude at the center pixel to zero. Map the gradient angle to the closest of 4 cases, where the line is sloped at almost 0°, 45°, 90°, and 135°. For example, 30° would map to 45°.

For more details about non-maximum suppression, please refer to the last page of this handout.

Your code cannot call on Matlab’s edge function, or any other similar functions. You may use edge for comparison and debugging. A sample result is shown below.



2.3 Hough Transform (3 pts)

Write a function that applies the Hough Transform to an edge magnitude image. Show the output for at least one of the given images in your write-up.

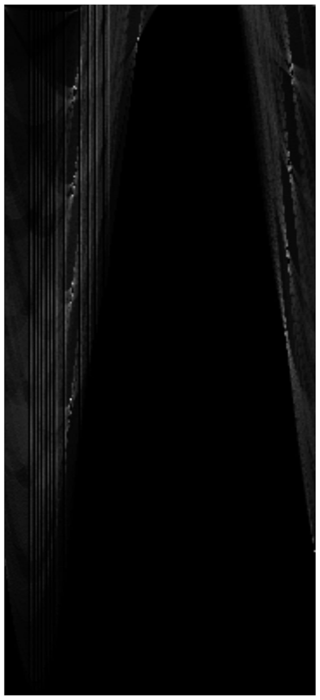
function [H, rhoScale, thetaScale] = myHoughTransform(Im, threshold, rhoRes, thetaRes)

Im is the edge magnitude image, threshold (scalar) is an edge strength threshold used to ignore pixels with a low edge filter response. rhoRes (scalar) and thetaRes (scalar) are the resolution of the Hough Transform accumulator along the rho and theta axes respectively. H is the Hough transform accumulator that contains the number of "votes" for all the possible lines passing through the image. rhoScale and thetaScale are the arrays of rho and theta values over which myHoughTransform generates the Hough transform matrix H. For example, if rhoScale(i) = rho_i and thetaScale(j) = theta_j, then H(i, j) contains the votes for rho=rho_i and theta=theta_j.

First, threshold the edge magnitude image. Each pixel (x, y) above the threshold is a possible point on a line and votes in the hough transform for all the lines it could be a part of. Parameterize lines in terms of theta and rho such that rho = x cos(theta) + y sin(theta), theta $\in [0,\ 2\pi]$ and rho $\in [0, M]$. (Or you can use the negative rho to define your Hough space which gives you nice curves. So, your Hough space can be theta $\in [0,\ \pi]$ and rho $\in [-M, M]$.) M should be large enough to accomodate all lines that could lie in an image. Each line in the image corresponds to a unique pair (rho, theta) in this range. Therefore, theta values corresponding to negative rho values are invalid, and you should not count those votes.

The accumulator resolution needs to be selected carefully. If the resolution is set too low, the estimated line parameters might be inaccurate. If resolution is too high, run time will increase and votes for one line might get split into multiple cells in the array.

Your code cannot call Matlab's hough function, or any other similar functions. You may use hough for comparison and debugging. A sample visualization of H is shown below.



2.4 Finding lines (3 pts)

Write a function that uses the Hough transform output to detect lines,

```
function [rhos, thetas] = myHoughLines(H, nLines)
```

where H is the Hough transform accumulator, and nLines is the number of lines to return. Outputs (rhos and thetas) are both nLines x 1 vectors that contain the row and column coordinates of peaks in H, that is, the lines found in the image.

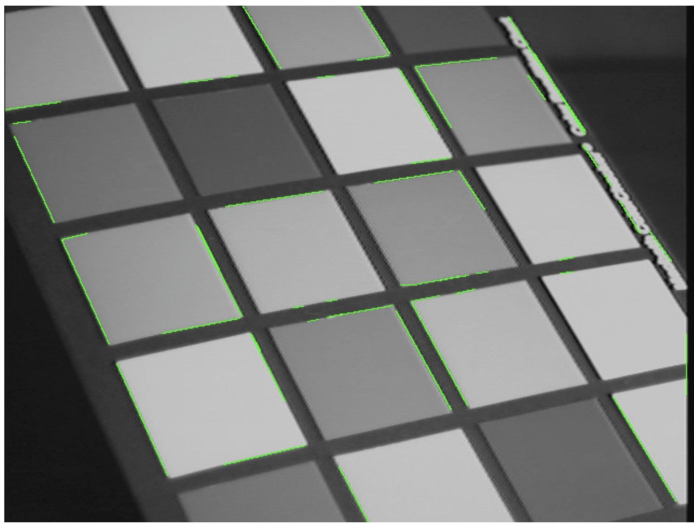
Ideally, you would want this function to return the ρ and θ coordinates for the nLines highest scoring cells in the Hough accumulator. But for every cell in the accumulator corresponding to a real line (likely to be a locally maximal value), there will probably be a number of cells in the neighborhood that also scored high but should not be selected. These non maximal neighbors can be removed using non maximal suppression. Note that this non maximal suppression step is different from the one performed earlier. Here you will consider all neighbors of a pixel, not just the pixels lying along the gradient direction. You can either implement your own non maximal suppression code or find a suitable function on the Internet (you must acknowledge and cite the source in your write-up, as well as hand in the source in your matlab/ directory). Another option is to use Matlab function imerode.

Once you have suppressed the non maximal cells in the Hough accumulator, return the coordinates corresponding to the strongest peaks in the accumulator.

Your code cannot call on Matlab's houghpeaks function or other similar functions. You may use houghpeaks for comparison and debugging. We write this instruction in the first hand-out just in case. We may not repeat these things in future hand-outs, as they are trivial. Please use your best judgement. If we ask you to do X, it is probably wrong to call an existing Matlab function which does X… In at least one Hough image, visualize the results of peak finding (e.g., use plot function to visualize peak locations on top of Hough images).

2.5 Fitting line segments for visualization (2 pts)

Now you have the parameters ρ and θ for each line in an image. However, this is not enough for visualization. We still need to prune the detected lines into line segments that do not extend beyond the objects they belong to. This is done by houghlines and drawLines.m. See the script houghScript.m for more details. You can modify the parameters of houghlines and see how the visualizations change. As shown in Figure 3, the result is not perfect, so do not worry if the performance of your implementation is not good. You can still get full credit as long as your implementation makes sense. Show line segment extraction results for at least 1 image.



2.6 Experiments (2 pts)

Use the script included to run your Hough detector on the image set and generate intermediate output images. Include the set of intermediate outputs for one image in your write-up. Did your code work well on all the image with a single set of parameters? How did the optimal set of parameters vary with images? Which step of the algorithm causes the most problems? Did you find any changes you could make to your code or algorithm that improved performance? In your write-up, you should describe how well your code worked on different images, what effect do the parameters have and any improvements you made to your code to make it work better.

Lastly, download or take at least 2 images by yourself, and run your code. Include the images and results under "ec" directory, and also show them in your write-up.

3. Non-maximum suppression

NMS in 2D image, you can move a 3 × 3 (or 7 × 7, etc.) filter over the image. At every pixel, the filter suppresses the value of the center pixel (by setting its value to 0) if its value is not greater than the value of the neighbors. To use NMS for edge thinning, you should compare the gradient magnitude of the center pixel with the neighbors along the gradient direction instead of all the neighbors. To simplify the implementation, you can quantize the gradient direction into 8 groups and compare the center pixel with two of the 8 neighbors in the 3 × 3 window, according to the gradient direction. For example, if the gradient angle of a pixel is 30◦, we compare its gradient magnitude with the north-east and south-west neighbors and suppress its magnitude if it's not greater than these two neighbors.

4. Grading

Project 1 has 6 components, whose corresponding points are described as above. They sum up to 16 points, as each project accounts for 16 percent of the final grade.