CSE 491 Introduction to Computer Vision: Project 1
Hough Transformation

Instructor: Vishnu Boddeti
Due Date: Thu Feb 13, 2018 23:59:5$\bar{9}$

In this assignment you will be implementing 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 on. Like most vision algorithms, the Hough Transform uses a number of parameters whose optimal values are (unfortunately) data dependent (i.e., a set of parameter values that works really well on one image might not be best for another image). By running your code on the test images you will learn about 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 OpenCV, MATLAB etc. 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

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. **Start early!** Especially those not familiar with Python.

3. If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

4. **Write-up:** Your write-up should mainly consist of three parts, your answers to theory questions, the resulting images of each step (i.e. the output of `houghScript.py`), and the discussions for experiments. Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.

5. **Submission:** Your submission for this assignment should be in two parts (1) PDF of your write-up `<msu-id.pdf>`, and (2) a zip file, `<msu-id.zip>`, composed of your

Python implementations (including helper functions), and your implementations, results for extra credit (optional). Please make sure to remove the `data/ and result/` folders, `houghScript.py, drawLine.py, houghlines.py` and any other temporary files you've generated.
Your final upload should have the files arranged in this layout:

- <MSUNetID>
  - python
    - myImageFilter.py
    - myEdgeFilter.py
    - myHoughTransform.py
    - myHoughLines.py
    - yourHelperFunctions.py *(optional)*
  - ec *(optional for extra credit)*
    - myImageFilterX.py
    - ec.py
    - your own images
    - your own results

For your convenience, we have provided you with a simple script `checkSubmission.py` to check whether your submissions are in correct formats or not. The only thing you need to do is place the script at the directory where your submission, `<your-msu-id>.zip`, lies and then execute it. **Please make sure you passed the test before uploading your pdf and zip file to Google Classroom**. Assignments that do not follow this submission rule will be **penalized 10% of the total score**.

2 Theory Questions

Type down your answers for the following questions in your write-up. Each question should only take a couple of lines. In particular, the "proofs" do not require any lengthy calculations. If you are lost in many lines of complicated algebra you are doing something much too complicated (or wrong).

1. Show that if you use the line equation $\rho = x\cos\theta + y\sin\theta$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y).

2. Assuming that the image points (x, y) are in an image of width W and height H (i.e., $x \in [1, W], y \in [1, H]$), what is the maximum absolute value of ρ and what is the range for θ?

3. For point (10, 10) in the image, plot the corresponding sinusoid wave in Hough space. Please use Matplotlib to plot the curve and report the result in your write-up.

3 Programming

We have included a wrapper script named `houghScript.py` 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 we will run the original `houghScript.py` while grading. Please make sure your code runs correctly with the original script and generates the required output images.

Every script/function you wrote in this section should be included in the `python/` directory. As for the result images, please include them in your write-up.

Q3.1 Convolution (15 points)

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

```
img1 = myImageFilter(img0, h)
```

The function will input a greyscale image (`img0`) and a convolution filter stored in matrix `h`. The function will output 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 Numpy's function to pad array. However, your code can not call on any Numpy/Scipy functions or other Python libraries to compute the convolution. You may compare your output to these functions for comparision and debugging.

Q3.1x Implement Convolution with One *for* Loop (extra: 10 points)

Please write a function that does convolution with only one for loop and save it to `ec/` directory. (If you already do so in `Q2.1`, good job, just copy one and save it to `ec/`.) Also, briefly describe how you implement it in your write-up. Illustrations helpful for understanding is highly encouraged.

```
img1 = myImageFilterX(img0, h)
```

Vectorization is an important, must-learn technique while writing code for computer vision using Numpy. Compared to a for loop, vectorization dramatically increases the speed, especially when dealing with big data. Therefore, it would be a good habit to avoid for loop and use vectorization as much as possible. In `Q2.1`, it's straightforward to implement the convolution with **two** for loops. However, through vectorization, it is possible to use only one for loop. This may be quite difficult, especially for those who just start learning Python/Numpy, but we still encourage you to give it a try. In the end, it is worth it! :D

Q3.2 Edge Detection (20 points)

Write a function that finds edge intensity and orientation in an image.

```
img1 = myEdgeFilter(img, sigma)
```

The function will input a greyscale image (`img`) and `sigma` (scalar). `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. Write your own function to create the Gaussian filter or use the one provided within `myEdgeFilter.py`. 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 in the x direction `Ix`, convolve the smoothed image with the x oriented Sobel filter. Similarly, find image gradient in the y direction `Iy` by convolving the smoothed image with the y oriented Sobel filter. You can also output `Ix` and `Iy` if needed.

In many cases, the high gradient magnitude region along an edge will be quite thick. For finding lines it is best to have edges that are a single pixel wide. Towards this end, make your edge filter implement non-maximum suppression, i.e., for each pixel look at the two neighboring pixels along the gradient direction and if either of those pixels has a larger gradient magnitude then set the edge magnitude at the center pixel to zero. For more details about non-maximum suppression, please refer to the last page of this handout.

Your code cannot call on Numpy/Scipy's functions or other similar existing libraries. A sample result is shown in Fig. 1.

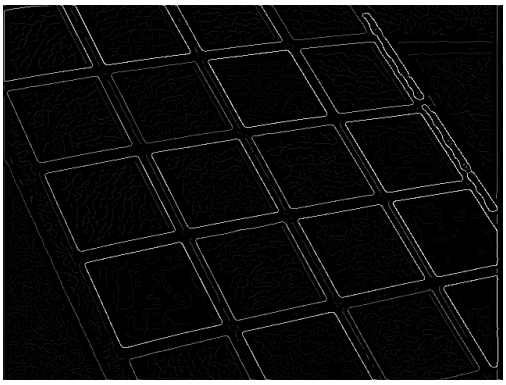

Figure 1: Edge detection result

Q3.3 The Hough Transform (20 points)

Write a function that applies the Hough Transform to an edge magnitude image.

`H, rhoScale, thetaScale = myHoughTransform(Im, threshold, rhoRes, thetaRes)`

`Im` is the edge magnitude image, `threshold` (scalar) is a 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 ρ and θ 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 ρ and θ values over which `myHoughTransform` generates the Hough transform matrix `H`. For example, if `rhoScale(i) =` ρ_i and `thetaScale(j) =` θ_i, then `H(i,j)` contains the votes for $\rho = \rho_i$ and $\theta = \theta_i$.

First, threshold the edge 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 θ and ρ such that $\rho = x\cos\theta + y\sin\theta$, $\theta \in [0, 2\pi]$ and $\rho \in [0, M]$. 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 (ρ, θ) in this range. Therefore, θs corresponding to negative ρ 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 innaccurate. 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 on exiting Python library functions or other similar functions. A sample visualization of `H` is shown in Fig. 2.

Figure 2: Hough transform result

Q3.4 Finding Lines (15 points)

`rhos, thetas = myHoughLines(H, nLines)`

`H` is the Hough transform accumulator; `nLines` is the number of lines to return. Outputs `rhos` and `thetas` are both `nLines`$\times$ 1 vectors that contain the row and column coordinates of peaks in `H` (i.e. 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 highly but shouldn't be selected. These non maximal neighbors can be removed using non maximal suppression. Note that this non maximal suppression step is different to 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/cite the source in your write-up as well as hand in the source in your `python/` directory). Once you have suppressed the non maximal cells in the Hough accumulator, return the coordinates corresponding to the strongest peaks in the accumulator.

Q3.5 Fitting Line Segments for Visualization (5 points)

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.py`. See `houghScript.py` for more details. You can modify the parameters of `houghlines` and see how the visualizations change. As shown in Fig. 3, the result is not perfect, so don't worry if the performance of your implementation is not good. You can still get full credit as long as your implementation make sense.

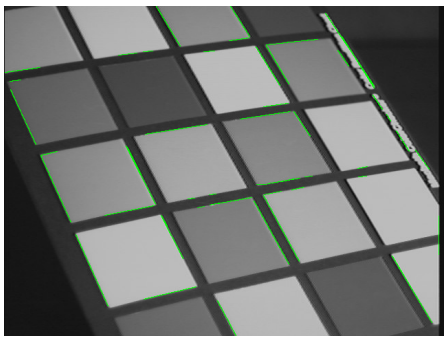

Figure 3: Line segments result

4 Experiments

(15 points)

Use the script included to run your Hough detector on the image set and generate intermediate output images. Did your code work well on all the images 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.

5 Try Your Own Images!

Q5.1x (extra: 10 points)

Take five pictures/images. Write a script `ec.py` to take care of reading in your images, making function calls to the various steps of the Hough transform and generate images showing the output and some of the intermediate steps (like `houghScript.py`). Submit your own images and `ec.py` to `ec/`. As for the resulting images, please include them in your write-up.

6 Non-maximum Suppression

Non-maximum suppression (NMS) is an algorithm used to find local maxima using the property that the value of a local maximum is greater than its neighbors. To implement the NMS in 2D image, one can move a 3x3 (or 7x7, etc) filter over the image. At every pixel, it 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, one 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, one can quantize the gradient direction into 8 groups and compare the center pixel with two of the 8 neighbors in the 3x3 window according to the gradient direction. For example, if the gradient angle of a pixel is 30 degrees, we compare its gradient magnitude with the north east and south west neighbors and suppress its magnitude if it's not greater than the two neighbors.