16720-B Fall 2020 Hough Transform

Instructor: Kris Kitani

TAs: Alireza Golestaneh (Lead), Nadine Chang, Cormac O'Meadhra, Zhengyi Luo, Anand Bhoraskar

Due Sep 21, 2020 11:59 PM

Total Points: 110

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 different Python image processing and computer vision toolbox.
You are **not** allowed to use such functions in this assignment. You may however compare your output to the output generated by the pre-existing functions 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 for those not familiar with Python. For each assignments you may need to install a couple of python libraries, so if you are not familiar with installing different libraries make sure to start early. We use Python 3.x (**not python 2.x**). For this assignment you need to have numpy, opencv, and glob libraries installed. We use opencv version 3.4.1 or higher. See appendix if you dont have python installed on your system.

3. If you have any question, please look at **Piazza** first. Other students may have encountered the same problem, and it may be solved already. If not, post your question in the specified folder. TAs 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 (for example, 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. **Instructions for gradescope** Log onto `http://gradescope.com` with your Andrew ID and you should be registered for the course. Submit the assignment by the specified due date. Each answer in the write-up should be on a different page. In gradescope, you need to mark the pages in the PDF which correspond to each question.

6. **Submission:** Your submission for this assignment should be a zip file, `<andrew-id.zip>`, composed of your write-up (pdf), your python implementations (including helper functions), and your implementations and results for extra credit (optional). **Please make sure to remove the** `data/`, `result/` **folders, and any other temporary files that you've generated.**
   Your final upload should have files arranged in this layout:

- <AndrewID>
  - <AndrewID>.pdf
  - python
    * houghScript.py
    * myImageFilter.py
    * myImageFileterX.py
    * myEdgeFilter.py
    * myHoughTransform.py
    * myHoughLines.py
    * myHoughLinePrune.py
    * yourHelperFunction1.py*(optional)*
    * yourHelperFunction2.py*(optional)*

  Assignments that do not follow this submission rule will be **penalized 10% of the total score**.

7. Please make sure that the file paths that you use are relative and not absolute. That is, it shouldn't be `imread('/name/Documents/subdirectory/hw1/data/abc.jpg')` but `imread('../data/abc.jpg')`.

2 Theory Questions (25 points)

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 perhaps wrong). Each question from Q2.1 to Q2.5 are worth 5 points each.

Q2.1 Show that if you use the line equation $x\cos\theta + y\sin\theta - \rho = 0$, 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).

Q2.2 Why do we parameterize the line in terms of ρ, θ instead of slope and intercept (m, c)? Express the slope and intercept in terms of ρ and θ.

Q2.3 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 of θ?

Q2.4 For points $(10, 10)$, $(15, 15)$ and $(30, 30)$ in the image, plot the corresponding sinusoid waves in Hough space (ρ, θ) and visualize how their intersection point defines the line (what is (m, c) for this line?). Please use Python to plot the curves and report the result in your write-up.

Q2.5 How does the dimension of parameter space affects Hough Transform method? What would you do when the parameter space is high , i.e., 3D or 4D instead of 2D? Briefly explain your method in the write-up.

3 Implementation (75 points)

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 TAs 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 write in this section should be included in the `python/` directory. As for the resulting images, please include them in your write-up

3.1 Convolution (10 points)

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

```
def myImageFilter(img_gray, h):

    return Img1
```

As input, the function takes a greyscale image (img_{gray}) and a convolution filter stored in matrix `h`. The output of the function should be an image `Im1` of the same size as img_{gray} which results from convolving img_{gray} 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 possible solution 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.

Your code **cannot** use pre-exising convolution functions in python such as `numpy.convolve, scipy.signal.convolve2d, imfilter, conv2, convn, filter2, etc`, or other similar functions. You need to write your functions from scratch. You may compare your output to these functions for comparison and debugging.

3.2 Convolution with One *for* Loop (10 points)

Write a function that does convolution with **only one for loop**. (If you already do so in `Q3.1`, good job, just copy it.) Also, briefly describe how you implement it in your write-up. Illustrations helpful for understanding is highly encouraged.

```
def myImageFilterX(img_gray, h):

    return Im1
```

Vectorization is an important, must-learn technique while writing python. Comparing to for loop, vectorization dramatically increases the speed, especially when dealing with big data. Therefore, it would be a good habit to avoid loops and use vectorization as much as possible. In `Q3.1`, 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, but we still encourage you to give it a try.

3.3 Edge Detection (10 points)

Write a function that finds edge intensity and orientation in an image. Include the output of your function for one of the given images in your writeup.

```
def myEdgeFilter(img_gray, sigma):

    return ImEdge,Io,Ix,Iy
```

The function will input a greyscale image (img_{gray}) and `sigma` (scalar). `sigma` is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. The function will output `ImEdge`, the edge magnitude image; `Io` the edge orientation image and `Ix` and `Iy` which are the edge filter responses in the x and y directions respectively.

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. You may use `Gauss2D.py` to get the Gaussian filter. The size of the Gaussian filter should depend on `sigma` (e.g., `hsize=2*ceil(3*sigma)+1`). To find the image gradient in the x direction `Ix`, convolve the smoothed image with the x oriented Sobel filter. Similarly, find `Iy` by convolving the smoothed image with the y oriented Sobel filter.

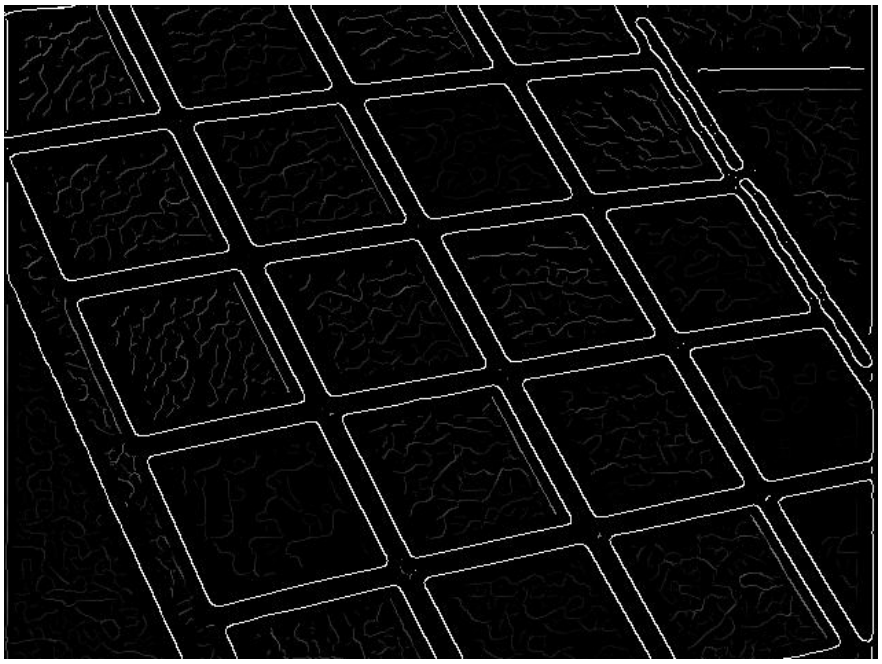

Figure 1: Example of edge intensity image

The edge magnitude image `Im` and the edge orientation image `Io` can be calculated from `Ix` and `Iy`.

In many cases, the high gradient magnitude region along an edge will be quite thick. For finding lines its best to have edges that are a single pixel wide. Towards this end, make your edge filter implement non maximal suppression, that is 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. Map the gradient angle to the closest of 4 cases, where the line is sloped at almost 0°, 45°, 90°, 135°. For eg, 20° would map to 0° and 30° would map to 45°.

For more details on non-maximum suppression, please refer to section 4 of this handout. Your code cannot call pre-existing edge functions from python libraries. An example of edge intensity image can be seen in Figure 1.
We encourage writing vectorized code for efficiency.

3.4 Hough Transform **(15 points)**

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

```
def myHoughTransform(ThrImEdge, threshold, rhoRes, thetaRes):

    return H, rhoScale, thetaScale
```

`ThrImEdge` is the binary image resulted from the edge magnitude image via `threshold` (scalar). `ThrImEdge` pixels with value lower than the threshold are zero and the remaining are one. `rhoRes` (scalar) and `thetaRes` (scalar) are the resolution of the Hough transform accumulator along the ρ and θ axes respectively. For example, if `thetaRes` = 5° and $\theta \in [0, 360°]$, then number of bins along θ axis is $360/5 = 72$. `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_j, then `H(i, j)` contains the votes for $\rho = \rho_i$ and $\theta = \theta_j$.

First, threshold the edge image to get `ThrImEdge`. 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. Parametrize lines in terms of θ and ρ such that $\rho = x\text{cos}\theta + y\text{sin}\theta$. Range of ρ and θ should be such that each line in the image corresponds to a unique pair (ρ, θ).

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 pre-existing python functions for `hough` transform.

3.5 Finding Lines **(15 points)**

```
def myHoughLines(H, nLines):

    return peakRho, peakTheta
```

`H` is the Hough transform accumulator and `nLines` is the number of lines to return. Outputs contain the indicies of the accumulator array H that correspond to a local maxima. `peakRho` and `peakTheta` are both 1d vectors that contain the parameters (ρ and θ respectively) of the lines found in an image.

Figure 2: Hough transform result.

Ideally, you would want this function to return the ρ and θ values 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. Implement your own non maximal suppression. Once you have suppressed the non maximal cells in the Hough accumulator, return the line parameters corresponding to the strongest peaks in the accumulator. If you 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).

3.6 Fitting Line Segments for Visualization (15 points)

```
def myHoughLineSegments(img_rgb,ThrImEdge,peakRho,peakThetas,rhoScale,thetaScale):

    return OutputImage
```

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. Here you need to write your `myHoughLineSegments`, this function get img_{rgb}, `ThrImEdge`, `peakRho`, `peakThetas`, `rhoScale`, `thetaScale` and return `OutputImage`. `OutputImage` demonstrate the input image and the hough transform results (see Figure 3). In `myHoughLineSegments` you need to prune the detected lines into line segments that do not extend beyond the objects they belong to. Then, you need to draw them on the img_{rgb} and return the results.

As shown in Fig. 3, the result is not perfect, so don't worry if the performance of your implementation is not that great.

Run the `houghScript.py` file and include the aforementioned visual results (`Im1`, `ImEdge`, `ThrImEdge`, `H`, `OutputImage`) in your write-up.

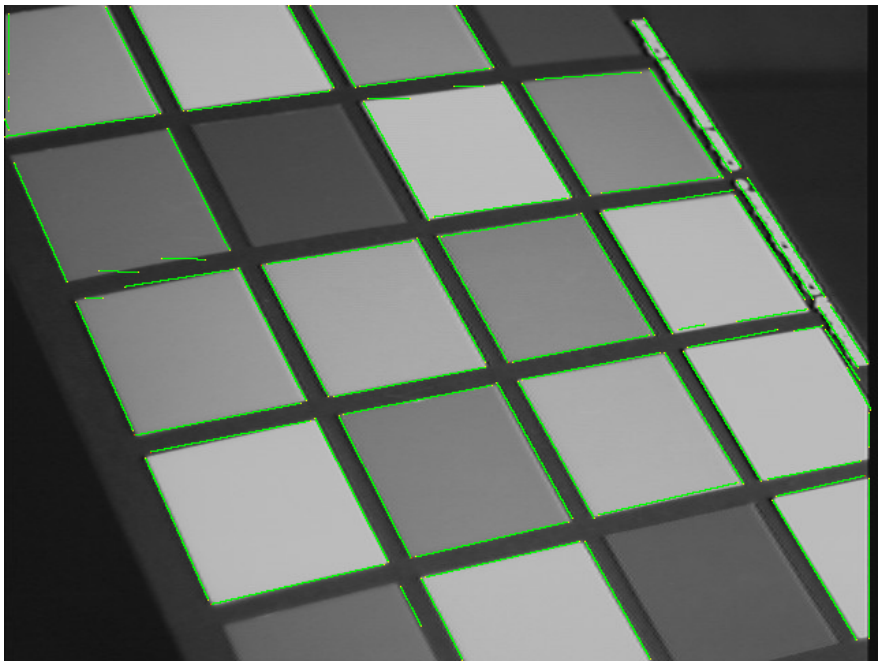

Figure 3: Line segments result

4 Experiments (10 points)

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?
Tabulate the different experiments that you have performed with their results in your write-up. Also describe how well your code worked on different images, what effect the parameters had on any improvements that you made to your code to make it work better. If you made changes to your code that required changes to the results generation script, include your updated version in your submission.

Appendix

Installing Python 3

If you don't have python installed the easiest way is use Anaconda (https://docs.anaconda.com/anaconda/install/) to install python. Anaconda also let you install other packages and libraries easily.

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 (see Figure 4). To implement the NMS in 2D image, one can move a 3×3 (or 7×7) 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 neighbours. 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 3×3 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.

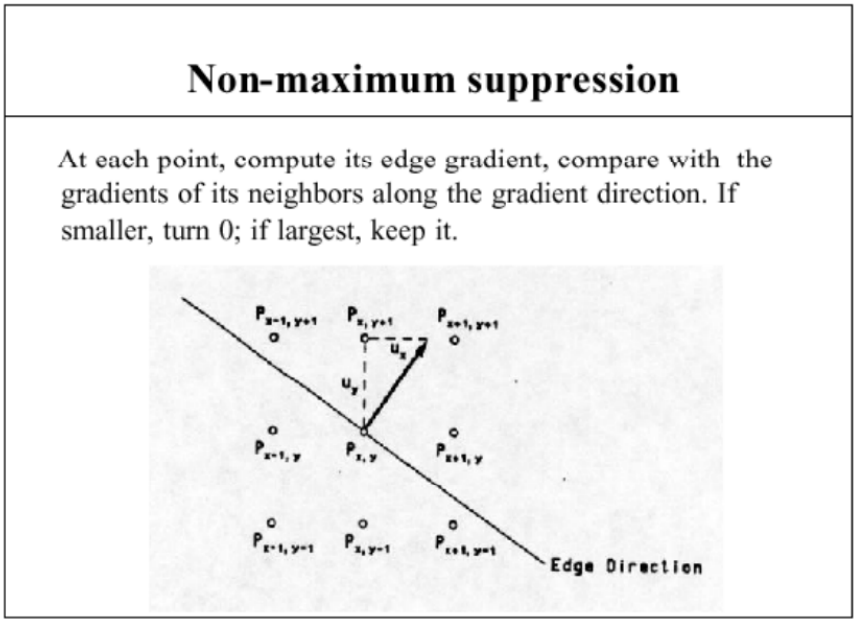

Figure 4: NMS for 2D