

Assignment 1

Computer Vision 15–385, Spring 12

Due Date: Tuesday 02/14/2012

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 (ie 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 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 Submitting Your Assignment

Your submission for this assignment will comprise of answers to a set of theoretical questions, the code for your MATLAB implementation and a short writeup describing your observations and improvements you made to your code. The answers to the theory questions in Section 2 should be in a plaintext file or a pdf named `theory.txt` or `theory.pdf`. Each of the MATLAB functions you write as described in Section 3 along with any extra helper functions you wrote should be in a folder named `matlab`, upload only `.m` files to this folder and make sure all the files needed for your code to run (except data) are included. The writeup describing your experiments (refer to Section 4) should be in a plaintext file or a pdf named `experiments.txt` or `experiments.pdf`.

A directory has been created on for uploading all your course related files. The directory can be found at `afs/cs.cmu.edu/academic/class/15385-s12-users/andrewid` (for graduate students, the folder is `15685-s12-users`). Inside your submission directory, create a sub-folder named `p1` for this assignment. Do not add any extra layers of indirection to your directory structure. Your final upload should have the files arranged in this layout:

- `afs/cs.cmu.edu/academic/class/15385-s12-users/andrewid`
 - `p1`
 - `theory.txt` or `theory.pdf`
 - `experiments.txt` or `experiments.pdf`
 - `matlab`
 - `myImageFilter.m`

- myEdgeFilter.m
- myHoughTransform.m
- myHoughLines.m
- myHoughLineSegments.m
- houghScript.m (*already provided, upload your modified copy*)
- drawLine.m (*already provided*)
- yourHelperFunction1.m (*optional*)
- yourHelperFunction2.m (*optional*)

You may need to run `aklog cs.cmu.edu` when you log in to be able read and write from your submission directory. Your files are due by 23:59:59 on the submission day. Please check to make sure you have write permissions to your submission folder ahead of time so that problems (if any) can be fixed. We will be using timestamps to determine submission times and for late day counting, so do not modify your files after the submission deadline unless you wish to use a late day.

2 Theory Questions

Question 1: Composing filters

(10 points)

Consider the following three filters \mathcal{G} , \mathcal{E} and \mathcal{M} . \mathcal{G} is a Gaussian smoothing kernel, \mathcal{E} is one of the linear kernels used by the Sobel edge detector and \mathcal{M} is a median filter. Is applying \mathcal{G} to an image followed by \mathcal{E} equivalent to applying \mathcal{E} to an image followed by \mathcal{G} ? How about if \mathcal{M} is used in place of \mathcal{G} ? In both cases, explain your answer.

Question 2: Decomposing a steerable filter

(10 points)

In the continuous domain, a two dimensional Gaussian kernel \mathcal{G} with standard deviation σ is given by $G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$. Show that convolution with \mathcal{G} is equivalent to convolving with \mathcal{G}_x followed by \mathcal{G}_y where \mathcal{G}_x and \mathcal{G}_y are 1 dimensional Gaussian kernels in the x and y directions respectively with standard deviation σ . From a computational efficiency perspective, explain which is better, convolving with \mathcal{G} in a single step or the two step \mathcal{G}_x -and- \mathcal{G}_y approach.

Question 3: Hough Transform Line Parameterization

(10 points)

Show that if you use the line equation $x\sin\theta - y\cos\theta + \rho = 0$, 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) . Does the period (or frequency) of the sinusoid vary with the image point (x, y) ?

3 Programming

We have included a wrapper 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 use and modify the script as you please, but make sure your final submission contains a version of the script that will run your code on all the test images and generate the required output images.

Convolution

(15 points)

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

```
function [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 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.

In the interests of running time, you might want your function to treat `h` kernels that are just row or column vectors and not full matrices separately, but this is optional.

Your code can not call on MATLAB's `imfilter`, `conv2`, `convn`, `filter2` functions or other similar functions. You may compare your output to these functions for comparison and debugging.

Edge Detection

(15 points)

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

```
function [Im Io Ix Iy] = 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 `Im`, 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. 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.

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.

Your code can not call on MATLAB's `edge` function or other similar functions. You may use `edge` for comparison and debugging.

The Hough Transform

(15 points)

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

```
function [H] = 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.

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 $x \sin \theta - y \cos \theta + \rho = 0$ where θ lies between 0 and π and the range of ρ is large enough to accomodate all lines that could lie in an image. 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 can not call on MATLAB's `hough` function or other similar functions. You may use `hough` for comparison and debugging.

Finding Lines

(10 points)

```
function [lineRho lineTheta] = myHoughLines(H, rhoRes, thetaRes, nLines)
```

`H` is the Hough transform accumulator; `rhoRes` and `thetaRes` are the accumulator resolution parameters and `nLines` is the number of lines to return. Outputs `lineRho` and `lineTheta` are both `nLines` \times 1 vectors that contain the parameters (ρ and θ respectively) of the lines found in an image.

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. You can either implement your own non maximal suppression code or find a suitable function on the Internet (you must acknowledge / cite the source).

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

Your code can not call on MATLAB's `houghpeaks` function or other similar functions. You may use `houghpeaks` for comparison and debugging.

Fitting Line Segments

(10 points)

Implement an algorithm that prunes the detected lines into line segments that do not extend beyond the objects they belong to.

```
function [lines] = myHoughLineSegments(lineRho, lineTheta, Im)
```

Your function should output `lines` is a MATLAB array of structures containing the pixel locations of the start and end points of each line segment in the image. The start location of the i^{th} line segment should be stored as a 2×1 vector `lines(i).start` and the end location as a 2×1 vector in `lines(i).stop`.

Your code can not call on MATLAB's `houghlines` function or other similar functions. You may use `houghlines` for comparison and debugging.

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 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? Include a `experiments.txt` or `experiments.pdf` file with your assignment submission describing how well your code worked on different images, what effect the parameters had and any improvements 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.