



PS1

Updated automatically every 5 minutes

CS4495 - Spring 2015 -
OMS Introduction to Computer Vision

Problem Set 1: Edges and Lines

**Due Sunday, February 8th, 2015
at 11:55pm**

Description

This problem set is your first “vision” project where you compute an “answer” – that is some structural or semantic description as to what is in an image. You’ll find edges and objects. And you’ll learn that some methods work well for carefully controlled situations and hardly at all when you relax those constraints.

RULES: You may use Matlab (or image processing functions on other platforms) to find edges, such as Canny or other operators. Don’t forget that those have a variety of parameters and you may need to experiment with them.

BUT: YOU MAY NOT USE ANY HOUGH TOOLS. For example, you need to write your own accumulator array data structures and code for voting and peak finding.

What to submit

Download and unzip a template for your platform (also under: <https://www.udacity.com/wiki/ud810>):

- [ps1_octave_template.zip](#)
- [ps1_matlab_template.zip](#)
- Note: Python template is being developed, will be announced on Piazza



PS1

Updated automatically every 5 minutes

data supplied with the problem set

- ps1-input0.png
- ps1-input0-noise.png
- ps1-input1.png
- ps1-input2.png
- ps1-input3.png

- output/ - directory containing output images and other files your code generates

Note: Output images must be stored with following mandatory naming convention:

```
ps<problem set #>-  
<question #>-<part>-  
<counter>.png
```

Example: ps1-1-a-1.png (first output image for question 1-a)

- ps1.m or ps1.py - code for completing each part, esp. function calls; all functions themselves must be defined in individual function files with filename same as function name, as indicated
- *.m or *.py - Matlab/Octave function files (one function per file), Python modules, any utility code
- ps1_report.pdf - a PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps1-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated).

Zip it as ps1_xxxx.zip, and submit on T-Square.

Questions

1. For this question we will use input/ps1-input0.png:



PS1

Updated automatically every 5 minutes



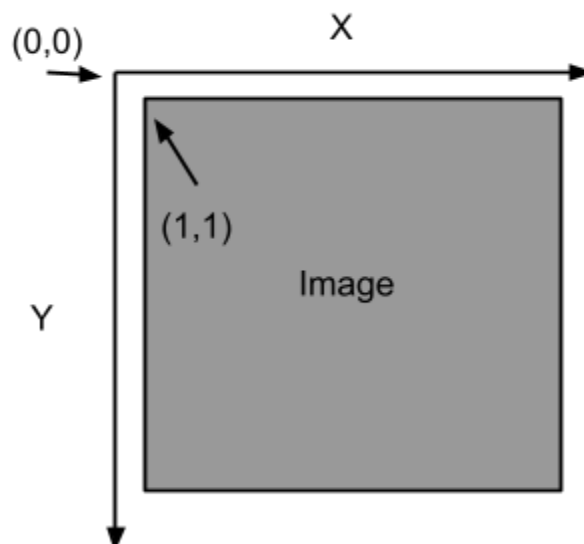
This is a test image for which the answer should be clear, where the “object” boundaries are only lines.

- a. Load the input grayscale image (`input/ps1-input0.png`) as `img` and generate an edge image – which is a binary image with white pixels (1) on the edges and black pixels (0) elsewhere.

For reference, do “doc edge” in Matlab and read about edge operators. Use one operator of your choosing – for this image it probably won’t matter much. If your edge operator uses parameters (like ‘`canny`’) play with those until you get the edges you would expect to see.

Output: Store edge image (`img_edges`) as `ps1-1-a-1.png`

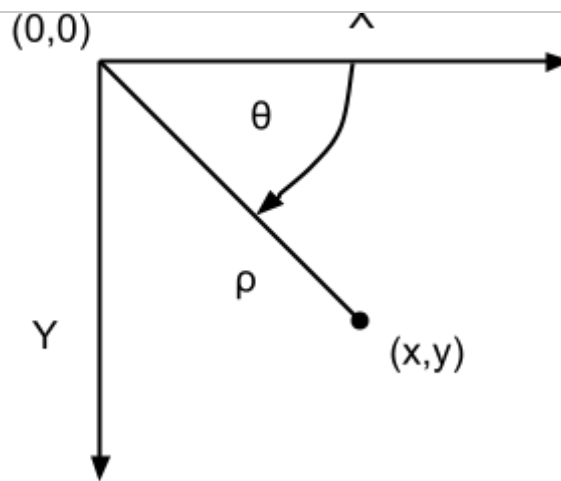
2. Implement a Hough Transform method for finding lines. Note that the coordinate system used is as pictured below with the origin placed one pixel above and to the left of the upper-left pixel of the image and with the Y-axis pointing downwards.





PS1

Updated automatically every 5 minutes



This has the effect of making the positive angular direction clockwise instead of counter-clockwise in the usual convention. Theta (θ) = zero still points in the direction of the positive X-axis.

- a. Write a function `hough_lines_acc` that computes the Hough Transform for lines and produces an accumulator array. Your code should conform to the specifications of the Matlab function `hough`: <http://www.mathworks.com/help/images/ref/hough.html>

Note that it has two optional parameters `RhoResolution` and `Theta`, and returns three values - the hough accumulator array `H`, theta (θ) values that correspond to columns of `H` and rho (ρ) values that correspond to rows of `H`.

Apply it to the edge image (`img_edges`) from question 1:

```
[H, theta, rho] =
hough_lines_acc(img_edges);
Or, with one optional parameter
specified ( $\theta$  = integers -90 to 89,
i.e. 180 values including 0):
[H, theta, rho] =
hough_lines_acc(img_edges,
'Theta', -90:89);
```



PS1

Updated automatically every 5 minutes

a-1.png (note: write a normalized uint8 version of the array so that the minimum value is mapped to 0 and maximum to 255).

- b. Write a function `hough_peaks` that finds indices of the accumulator array (here line parameters) that correspond to local maxima. Your code should conform to the specifications of the Matlab function `houghpeaks`: <http://www.mathworks.com/help/images/ref/houghpeaks.html>
Note that you need to return a Qx2 matrix with row indices (here rho) in column 1, and column indices (here theta) in column 2. (This could be used for other peak finding purposes as well.)

Call your function with the accumulator from the step above to find up to 10 strongest lines:

```
peaks = hough_peaks(H,  
10);
```

Function file: `hough_peaks.m`

Output: `ps1-2-b-1.png` - like above, with peaks highlighted (you can use drawing functions).

- c. Write a function `hough_lines_draw` to draw color lines that correspond to peaks found in the accumulator array. This means you need to look up rho, theta values using the peak indices, and then convert them (back) to line parameters in cartesian coordinates (you can then use regular line-drawing functions).

Use this to draw lines on the original grayscale (not edge) image. The lines should extend to the edges of the image (aka infinite lines):

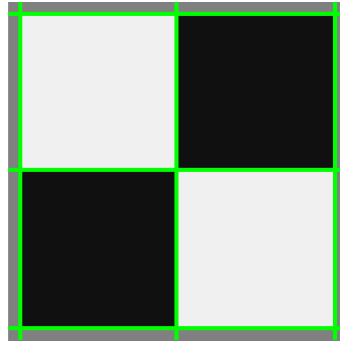
```
hough_lines_draw(img,  
'ps1-2-c-1.png', peaks,
```



PS1

Updated automatically every 5 minutes

It should look something like this:



You might get lines at the boundary of the image too depending upon the edge operator you selected (but those really shouldn't be there).

- d. What parameters did you use for finding lines in this image?

Output: Text response describing your accumulator bin sizes, threshold and neighborhood size parameters for finding peaks, and why/how you picked those.

3. Now we're going to add some noise.

- a. Use `ps1-input0-noise.png` - same image as before, but with noise. Compute a modestly smoothed version of this image by using a Gaussian filter. Make σ at least a few pixels big.

Output: Smoothed image: `ps1-3-a-1.png`

- b. Using an edge operator of your choosing, create a binary edge image for both the original image (`ps1-input0-noise.png`) and the smoothed version above.

Output: Two edge images: `ps1-3-b-1.png` (from original), `ps1-3-b-2.png` (from smoothed)

- c. Now apply your Hough method to the smoothed version of the edge image. Your goal is to adjust the filtering, edge finding, and Hough algorithms to find the lines as best you can in this test case.

Output:



PS1

Updated automatically every 5 minutes

had to do to get the best result you could.

4. For this question use: ps1-input1.png
 - a. This image has objects in it whose boundaries are circles (coins) or lines (pens). For this question you're still finding lines. Load/create a monochrome version of the image (you can pick a single color channel or use a built-in color to grayscale conversion function), and compute a modestly smoothed version of this image by using a Gaussian filter. Make σ at least a few pixels big.
Output: Smoothed monochrome image: ps1-4-a-1.png
 - b. Create an edge image for the smoothed version above.
Output: Edge image: ps1-4-b-1.png
 - c. Apply your Hough algorithm to the edge image to find lines along the pens. Draw the lines in color on the original monochrome (not edge) image. The lines can extend to the edges of the image.
Output:
 - Hough accumulator array image with peaks highlighted: ps1-4-c-1.png
 - Original monochrome image with lines drawn on it: ps1-4-c-2.png
 - Text response: Describe what you had to do to get the best result you could.
5. Now write a circle finding version of the Hough transform. You can implement either the single point method or the point plus gradient method. **WARNING: This part may be hard!!! Leave extra time!**

If you find your arrays getting too big (hint, hint) you might try make the range of radii very small to start with and see if you can find one size circle. Then maybe try the different sizes.

 - a. Implement



PS1

Updated automatically every 5 minutes

from 4-b above), and try calling your function with radius = 20:

```
H =  
hough_circles_acc(img_edges,  
20);
```

This should return an accumulator H of the same size as the supplied image. Each *pixel* value of the accumulator array should be proportional to the likelihood of a circle of the given radius being present (centered) at that location. Find circle centers by using the same peak finding function:

```
centers =  
hough_peaks(H, 10);
```

Function file:

hough_circles_acc.m (hough_peaks.m should already be there)

Output:

- Smoothed image: ps1-5-a-1.png (this may be identical to ps1-4-a-1.png)
- Edge image: ps1-5-a-2.png (this may be identical to ps1-4-b-1.png)
- Original monochrome image with the circles drawn in color: ps1-5-a-3.png

b. Implement a function

find_circles that combines the above two steps, searching for circles within a given radius range, and returns circle centers along with their radii:

```
[centers, radii] =  
find_circles(img_edges,  
[20 50]);
```

Function file: find_circles.m**Output:**

- Original monochrome image with the circles drawn in color: ps1-5-b-1.png
- Text response: Describe what you had to do to find circles.

6. More realistic images. Now that you have



PS1

Updated automatically every 5 minutes

- that seems to work best in terms of finding all the pen edges. Don't worry (until b) about whether you are finding other lines.
Output: Smoothed image you used with the Hough lines drawn on them: ps1-6-a-1.png
- b. Likely the last step found lines that are not the boundaries of the pens. What are the problems present?
Output: Text response
- c. Attempt to find only the lines that are the **boundaries** of the pen. Three operations you need to try are better thresholding in finding the lines (look for stronger edges), checking the minimum length of the line, looking for nearby parallel lines.
Output: Smoothed image with new Hough lines drawn: ps1-6-c-1.png
7. Finding circles on the same clutter image (ps1-input2.png).
- a. Apply your circle finder. Use a smoothing filter that seems to work best in terms of finding all the coins.
Output: the smoothed image you used with the circles drawn on them: ps1-7-a-1.png
- b. Are there any false alarms? How would/did you get rid of them?
Output: Text response (if you did these steps, mention where they are in the code by file, line no. and also include brief snippets)
8. Sensitivity to distortion. There is a distorted version of the scene at ps1-input3.png.
- a. Apply the line and circle finders to the distorted image. Can you find lines? Circles?
Output: Monochrome image with lines and circles (if any) found: ps1-8-a-1.png
- b. What might you do to fix the circle problem?
Output: Text response describing what you might try



Published using Google Docs

[Report abuse](#)

[Learn more](#)

PS1

Updated automatically every 5 minutes

tried and what worked best (with snippets).