CS4495 Computer Vision – Fall 2014

Problem Set 1 – Edges and Lines

Due Tuesday, September 9, 2014 *11:55pm*

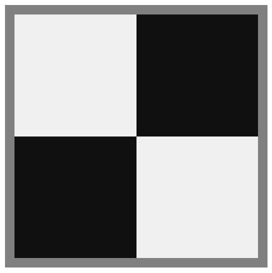
This problem set is your first "vision" project where you compute an "answer" – that is some structural or semantic description as to what is in an image. You'll find edges and objects. And you'll learn that some methods work well for carefully controlled situations and hardly at all when you relax those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

RULES: You may use the Matlab (or other image processing library) to find the edges, such as the 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.

As a reminder as to what you hand in: A Zip file that has

(1) Images (either as JPG or PNG or some other easy to recognize format) clearly labeled using he convention PS<number>-<question number>-<question sub>-counter.jpg For example, for this PS, the second image used as input would be listed as ps0-1-a-2.jpg since question 1(a) asks to get two images as input for the rest of the PS.
(2) Code you used for each question. It should be clear how to run your code to generate the results. Code should be in different folders for each main part with names like PS0-1-code. For some parts – especially if using Matlab – the entire code might be just a few lines.
(3) Finally a PDF file that shows all the results you need for the problem set. This will include the images appropriately labeled so it is clear which section they are for and the small number of written responses necessary to answer some of the questions. Also, for each main section, if it is not obvious how to run your code please provide brief but clear instructions.

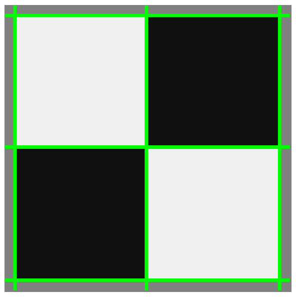
1 In the Problem Set directory there is a Data director with a few images. For this question use the first one ps1-input0.png which looks like this:



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

a. Do “doc edge” in Matlab and read about edge operators. Using one of your choosing – for this image it probably won’t matter much – create an edge image which is a binary image with white pixels on the edges and black pixels elsewhere. If your edge operator uses parameters (like `'canny'`) play with those until you get the edges you would expect to see.
Output: the edge image

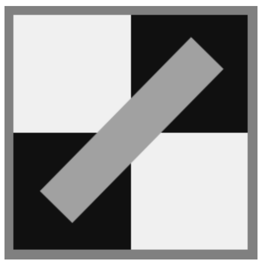
2 Write a Hough method for finding lines. Remember to worry about *d* being negative if θ goes from 0 to π. Apply it to the edge image. Draw the lines in color on the monochrome intensity (not edge) image. The lines can extend to the edge of the images (aka infinite lines). You should see an image that looks 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).

Output: Hough accumulator array image with peaks circled or somehow labeled.
Output: intensity image with lines drawn on them
Output: written response describing your accumulator bin sizes and why/how you picked those.

3 Just to make sure that your system can handle lines that are not vertical or horizontal, there is another test image: ps1-input0a.png :



Now you should get the long diagonal edges too. You may or may not the short ones.

Output: Hough accumulator array image with peaks circled or somehow labeled.
Output: intensity image with lines drawn on them
Output: written response describing your accumulator bin sizes and why/how you picked those.

4 Now were going to add noise. For this question use the first one ps1-input0-noise.png .
   a. This image is the same 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
   b. Using an edge operator of your choosing, create a binary edge image for both the raw monochrome image and the smoothed version above.
      Output: the two edge images
   c. Now apply your Hough method to the smoothed version of the edge image. Your goal is to adjust the filtering, the edge finding, and the Hough algorithms to find the lines as best you can in this test case
      Output: Hough accumulator array image with peaks circled or somehow labeled.
      Output: intensity image (original one with the noise) with lines drawn on them
      Output: describe what you had to do to get the best result you could.

5 For this question use the first one ps1-input1.jpg .
   a. This image has objects in it whose boundaries are circles (coins) or lines (pens). For this question you're still finding lines. Create a monochrome version of the image and compute a modestly smoothed version of this image by using a Gaussian filter. Make σ at least a few pixels big.
      Output: smoothed image
   b. Using an edge operator and parameters of your choosing, create an edge image for the smoothed version above.
      Output: the edge image
   c. Apply your Hough algorithm to find the lines along the pens. Draw the lines in color on the smoothed monochrome intensity (not edge) image. The lines can extend to the edge of the images (aka infinite lines).
      Output: Hough accumulator array image with peaks circled or somehow labeled.
      Output: intensity images with lines drawn on them
      Output: describe what you had to do to get the best result you could

6 Now for circles. Write a circle finding version of the Hough transform. You can implement either the single point method or the point plus gradient method. **THIS PART IS (SOMETIMES) HARDER THAN IT LOOKS – LEAVE EXTRA TIME FOR THIS!!!! TO TEST THIS YOU MIGHT MAKE YOUR OWN TEST IMAGE.** 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. Using the same original image as above. Smooth it, find the edges, find the circles.
   Output: edge image and images with the circles drawn in color
   Output: describe what you had to do to find the circles.

7 More realistic images. Now that you have Hough methods working, we're going to try them on images that have *clutter* in them: visual elements that are not part of the objects to be detected. The image to use is ps1-input2.jpg

   a. Apply your line finder. Use a smoothing filter and edge detector 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: the smoothed image you used with the Hough lines drawn on them.
   b. Likely the last step found lines that are not the boundaries of the pens. What are the problems present?
      Output: written 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.

8 Finding circles on the same clutter image.

   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.
   b. Are there any false alarms? How would/did you get rid of them?
      Output: written response (if you did these steps mention that they are in the code)

9 Sensitivity to distortion. There is a distorted version of the scene at ps1-input3.jpg

   a. Apply the line and circle finder to the distorted image. Can you find lines? The circles?
      Output: Image with lines and circles (if any) found.
   b. What might you do to fix the circle problem?
      Output: written response describing what you might try.
   c. EXTRA CREDIT: Try to fix the circle problem **THIS IS HARD**.
      Output: Written response describing what tried and what worked best.
      Output: Image that is the best shot at fixing the circle problem.