Lab Assignment #5
Due Friday, November 8 by 11:59pm
(upload as a single PDF to CatCourses)

Title: Hough transform

In this lab, you will implement the Hough transform to detect the most prominent line in an image.

1) Carefully review the section on "Global processing using the Hough transform" in chapter 10 of the text. In particular, make sure you understand figure 10.32 in the 3rd edition or figure 10.29 in the 4th edition. You should also review my course notes for the Hough transform from November 8 and 20 (they are available on CatCourses).

2) Create the file `my_hough_transform.py` and in it, write the function `my_hough_transform` that takes as input an edge image and determines the most prominent line using the method described in the text and in my notes. This function should output the parameters of the line in normal form: theta (θ), the angle the line makes with the vertical axis, and rho (ρ), the length of the perpendicular bisector of the line. It should also output the accumulator matrix.

   Your `my_hough_transform` function should have the following calling syntax:

   ```
   theta_out, rho_out, accumulator = my_hough_transform(i_edge)
   ```

   where `i_edge` is an edge image (a 2D numpy array) which has value 0 for non-edge points and 255 for edge points. `theta_out` and `rho_out` are scalars and `accumulator` is a 2D numpy array

   Here is a sketch of what your `hough_tranform` should do:

   a) Determine the size of `i_edge`.

   b) Create an empty accumulator matrix (a 2D numpy array). This matrix should have one column for each degree that theta can take for a line: -89 to 90. It should have one row for each value rho can take. Rho, the length of the perpendicular bisector, can range between –D and D where D is the diagonal size of the image. Thus, your accumulator matrix should have 2*D+1 rows and 180 columns.

c) For every edge point in `i_edge`, plot the corresponding sinusoid in the accumulator matrix. That is, an edge point determines an x,y pair which in turn determines the relationship between rho and theta via the formula: x * cos(theta) + y * sin(theta) = rho. Let theta range over all possible values (-89 to 90) and compute rho. This determines which accumulator cell to add another "vote" to. Notes and warnings: we are working in degrees. math.sin() and math.cos() expect the input to be in radians so convert theta to radians using math.radians() before calling math.sin() and math.cos(). Also, remember the ranges of the indices of your accumulator matrix when indexing using the theta and rho computed above.

d) Once you have visited every edge point in your edge image, determine the cell with the most votes in your accumulator matrix. This should give you the theta and rho of the most prominent line in the image (again, remember the ranges of the indices).

e) Return `theta_out, rho_out, accumulator` once they have been computed.

3) Debugging your `my_hough_transform` function will be tricky so I'm providing some test scripts and results to help you. There are four scripts:

- test_horizontal_line.py
- test_vertical_line.py
- test_pos_diagonal_line.py
- test_neg_diagonal_line.py

I have provided my results from running these scripts with my `my_hough_transform` function below on pages 5-8. You should make sure your results are similar. Note that my estimated parameters are not always exactly equal to the true values. But, your theta should be within 1-2 degrees and your rho should be within 1-2 pixels. These scripts call the function `draw_line_on_image` in the file `Lab05_Functions.py` so make sure to download it with the test scripts.

4) To prove to me that your `my_hough_transform` function works correctly, run the script test_random_line.py THREE TIMES and provide the results in a similar format to what I provided for the test scripts above. Page 9 of this lab contains an example of running it once with my `my_hough_transform` function. You report should have three pages like this with different lines. Your theta should be within 1-2 degrees and your rho should be within 1-2 pixels of the true values.

5) Finally, run the script test_real_image.py to apply your `my_hough_transform` to the real image runway_image.tif. My results for this are shown on pages 10-12. You don't need to turn in your results. I just want you to see that the technique actually works for finding lines in real images (assuming your function works correctly). Note that test_real_image.py calls the OpenCV function Canny() to compute the edge image of the runway image using the Canny edge detector. This function takes the lower and upper threshold values as the second and third parameters. I have set these to 50 and 100. Play around with different thresholds to observe how they affect the produced edge image.

Questions:

1. Briefly describe how the location of the intersection of the curves in horizontal_line_accumulator.tif (your result or mine) indicate the theta and rho parameters of the line in horizontal_line.tif. Repeat this for the vertical line, positive diagonal line, and negative diagonal line examples.
2. Repeat question 1 for ONE of your random line examples.
3. What was the most difficult part of this assignment?

What to submit:

A lab report as a single PDF containing:

- Lab number and title, your name, your lab section, your TA's name, and the date.
- Abstract: In a few sentences, describe the purpose of this lab.
- Your results of running test_random_line.py THREE TIMES:
    - The true and estimated theta and rho values.
    - The edge image.
    - The accumulator image.
- Questions: Your answers to the three questions above.
- Your Python file `my_hough_transform.py` which contains your function `my_hough_transform`.

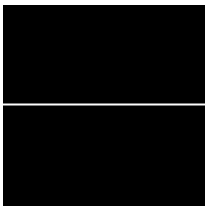
What your report will be graded on

- Whether you included an abstract.
- Whether your report includes the results you were asked to include and that your figures have appropriate captions.
- Whether you answered the assignment questions.

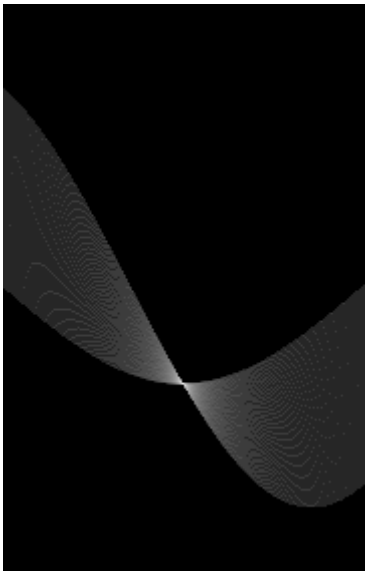
- Whether your report includes the code you were asked to include.
- Whether your code is correct.
- Whether your code is commented and your functions have headers.

My results of running test_horizontal_line.py:

```
true theta =  0  true rho =  50
estimated theta =  0  estimated rho =  50
```



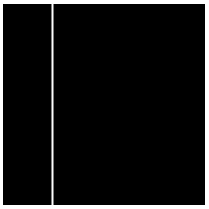
horizontal_line.tif



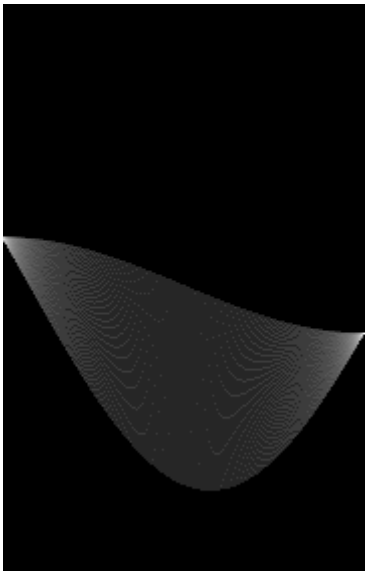
horizontal_line_accumulator.tif

My results of running test_vertical_line.py

```
true theta =  90  true rho =  25
estimated theta =  90  estimated rho =  25
```



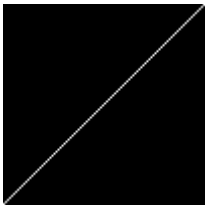
vertical_line.tif



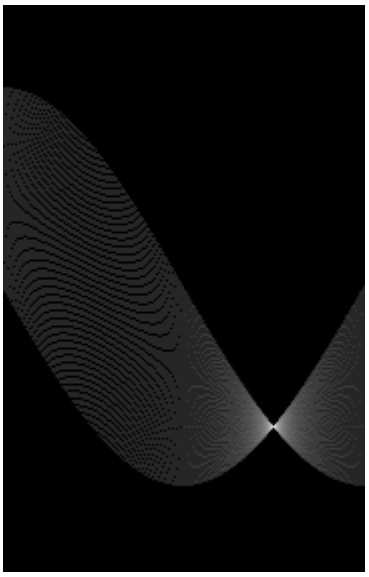
vertical_line_accumulator.tif

My results of running test_pos_diagonal_line.py

```
true theta =  45  true rho =  71
estimated theta =  45  estimated rho =  71
```



pos_diagonal_line.tif

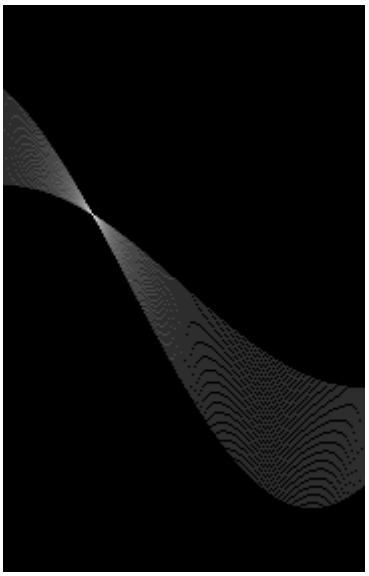


pos_diagonal_line_accumulator.tif

My results of running test_neg_diagonal_line.py

```
true theta =  -45  true rho =  -35
estimated theta =  -45  estimated rho =  -35
```

neg_diagonal_line.tif

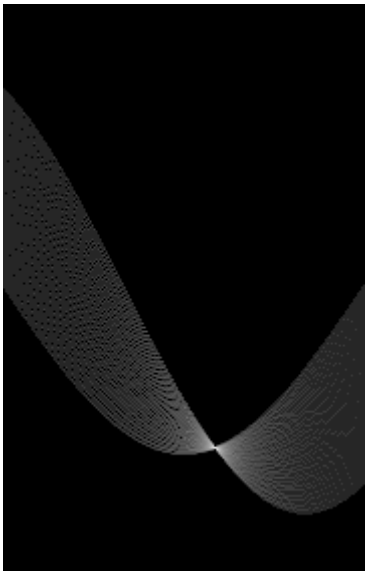


neg_diagonal_line_accumulator.tif

My results of running test_random_line.py

```
true theta =  16  true rho =  82
estimated theta =  16  estimated rho =  82
```

random_line.tif

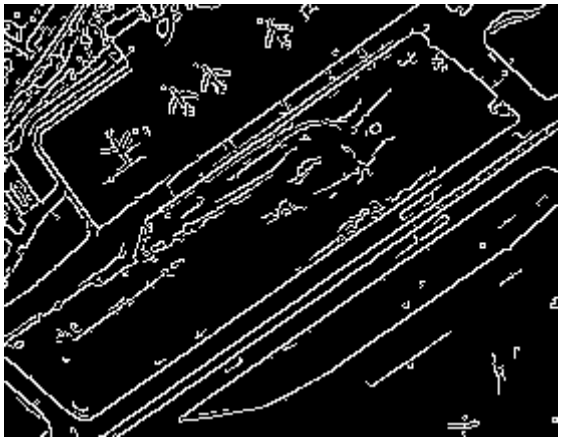


random_line_accumulator.tif

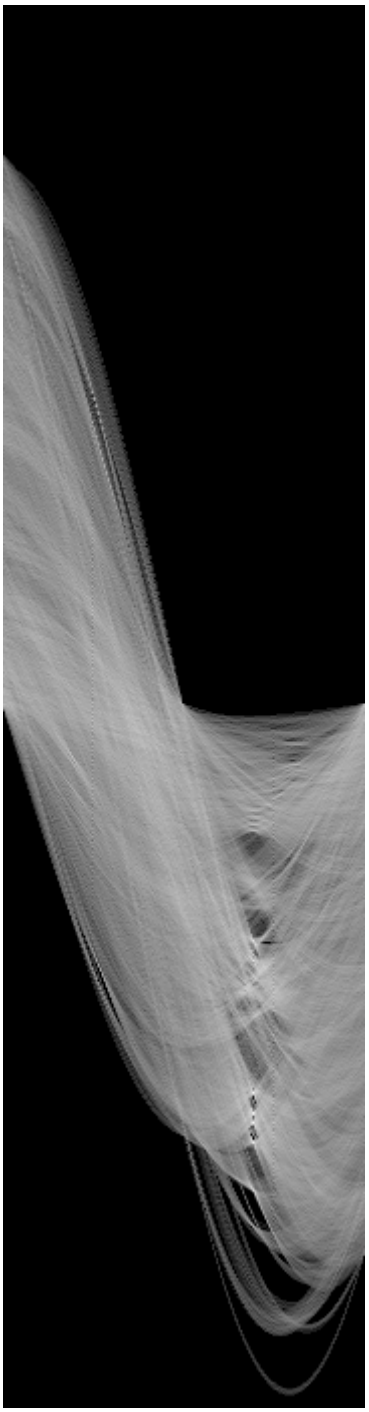
My results of running test_real_image.py

```
estimated theta =  35  estimated rho =  195
```

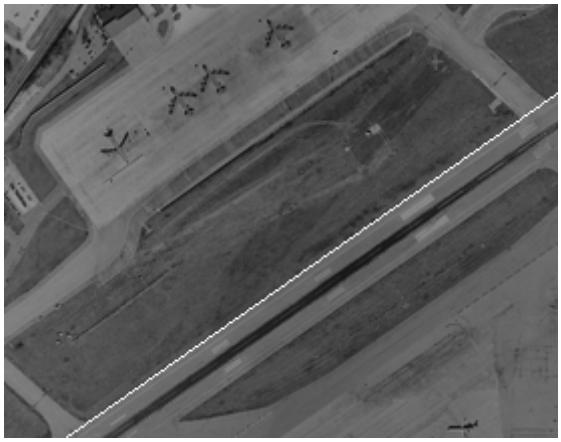
runway_image.tif



runway_image_edge.tif



runway_image_accumulator.tif



runway_image_with_line.tif