EN.601.461/661 – Computer Vision
Fall 2017
Homework #1
Due: 11:59 PM, Sunday, October 1, 2017
(10% off for each late day)

Note: All updates will be marked in **green**

10/1 2:27pm: Removed advice about consecutive label numbers.

9/28 4:23pm: added point value for each item

9/27 11:09pm: added notification of lateness policy

9/27 12:37am: clarified input and output parameters

9/24 10:55pm: added file name conventions for programming question 2.

9/24 9:48pm: clarified binary images

9/15 2:13pm: changed due date. Heavily clarified and modified Programming 1c.

9/13 8:53pm: added note about binary images using 0s and 1s

9/13 8:33pm: added note about using python 2

9/13 2:55pm: changed imagesc to pyplot.imshow()

9/13 12:50pm: changed Gradescope submission to Blackboard submission

All solutions (e.g., code, README write-ups, output images) should be zipped up as HW1_yourJHED.zip and posted on **Blackboard**, where 'yourJHED' is your JHED ID (e.g. areiter2). If you have hand-written the written assignment, please make a scan and attach the .pdf in the submission zip. Only basic Python functions are allowed, unless otherwise specified. If you are unsure of an allowable function, please ask on Piazza before assuming! You will get no credit for using a "magic" function to answer any questions where you should be answering them. When in doubt, ASK!

Similar rule applies with the use of OpenCV. Basic image IO functions such as `cv2.imread`, `cv2.imwrite` etc. are allowed. When in doubt, ask on Piazza!

About Piazza usage: Piazza is a great tool for collaboration and questions. However, **never post big chunks of source code as a public post**. Make use of this tool to talk about ideas, concepts and implementation details.

Please use python 2 for all programs. There isn't a huge difference between python 2 and 3 nowadays, but for testing purposes, it'll be better if we can predict which version you'll use.

Written Assignment

1) Consider a pinhole camera with perspective projection.
    a. (10 points) Consider a circular disk that lies in a plane parallel to the image plane. The disk does not necessarily lie on the optical axis. What is the shape of the image of the disk? Show your work and derive your equations.
    b. (5 points) Suppose the area of the image of the circular disk is 1 mm^2 when the distance from the pinhole to the plane of the disk is 1 meter. What is the area of the image of the circular disk if the distance is doubled? Again, show your work and how you arrived at your solution.
    c. (7 points) Now, replace the disk with a sphere. What is the shape of the image of the sphere? (Be careful with this one.)

2) ***(600.661: Required for Graduate Students)*** (10 points)
Once again, consider a pinhole camera. Where on the image would the vanishing points of ALL lines on the plane $Ax+By+Cz+D = 0$ lie? The coordinate frame is located at the pinhole with the z-axis pointing towards the image and the effective focal length is f.

Programming Assignment

1) Our goal is to develop a vision system that recognizes two-dimensional objects in images. The two objects we are interested in are shown in the image **two_objects.pgm**. (All of the images given to you are gray-level PGM images). Given an image such as **many_objects_1.pgm**, we would like our vision system to determine if the two objects are in the image and if so compute their positions and orientations. This information is valuable as it can be used by a robot to sort and assemble manufactured parts.

    The task is divided into four parts, each corresponding to a Python function you need to write and submit. Each of the following parts a)-d) requires a separate Python function, one for each (so 4 in total). Then write a "driver" program which loads the images that are needed, calls each of the functions, and reports/displays/writes the results. Therefore there should be 5 total files handed in with this question: **p1.py, p2.py, p3.py, p4.py** and **test_objects.py**. Please stick to these naming conventions to make the grading easier.

a. (5 points) Write a Python function named **p1** that converts a gray-level image to a binary one using a threshold value:

```
def p1(gray_image, thresh_val): # return binary_image
```

Select any threshold that results in "clean" binary images for the gray-level ones given to you (A binary image can be saved as a PGM file, and is no different from a regular image, but will have only 2 values, preferably 0 and 255). You should be able to use the same threshold value for all the images. (When submitting your assignment, please indicate the value you used in a separate README file.) Apply function **p1** to the image **two_objects.pgm**.

b. (15 points) Implement the sequential labeling algorithm (and name the function **p2**) that segments a binary image into several connected regions:

```
def p2(binary_image): # return labeled_image
```

Note that you may have to make two passes of the image to resolve possible equivalences in the labels. In the "labeled" output image each object region should be painted with a different gray-level: the *gray-level assigned to an object is its label*. The labeled images can be displayed to check the results produced by your program. Note that your program must be able to produce correct results given any binary image. You can test it on the images given to you. Apply function **p2** to the binary version of the image **two_objects.pgm**.

c. (15 points) Write a Python function named **p3** that takes a labeled image and computes object attributes, and generates the objects database:

```
def p3(labeled_image): #return [database, output_image]
```

The generated object database should be a dictionary of dictionaries ({0:dict1, 1:dict2, ...}), where each inner dictionary represents an object attached to its label (remember that the label is the gray level in the image). Each inner dictionary will have keys referring to attribute names, and values corresponding to those attributes' values. **We use an outer dictionary rather than an array because labels are not necessarily consecutive.**

For example, the `position` attribute of the first object in the `database_out` list (with label 0) would be accessed this way:

```
database[0]['position']
```

Whereas the `orientation` attribute of the second object, which happens to have label 41 (as a random example), would be accessed this way:

```
database[41]['orientation']
```

Since you will be using these attributes to recognize objects, it's up to you to experiment and find good attributes to list in the database. Every attribute will be added as a key to all the objects. Be sure to mention the attributes you chose in your README file. These attributes will serve as your object model database.

The output image should display positions and orientations of objects in the input image using a circle for the position and a short line segment originating from the center circle for orientation. Apply function **p3** to the labeled image of **two_objects.pgm**.

d. (10 points) Now you have all the tools needed to develop the object recognition system. Write a function named **p4** that recognizes objects from the database:

```
def p4(labeled_image, database): # return output_image
```

Your program should compare (using your own comparison criteria) the attributes of each object in a labeled image file with those from the object model database. It should produce an output image which would display the positions and orientations (using circles and line segments, as before) of only those objects that have been recognized. Using the object database generated from **two_objects.pgm**, test your function on the images **many_objects 1.pgm** and **many_objects_2.pgm.** In your README file, state the comparison criteria and thresholds that you used.

2) Your task here is to develop a vision system that recognizes lines in an image using the Hough Transform. Such a system can be used to automatically interpret engineering drawings, etc. We will call it the "line finder". Three images are provided to you: **hough_simple_1.pgm**, **hough_simple_2.pgm**, and **hough_complex_1.pgm**.

Each of the following parts a)-d) requires a separate Python function, one for each (so 4 in total), with d being only for graduate students. Write a "driver" program which loads the images that are needed, calls each of the functions, and reports/displays/writes the results. There should be 5 total files handed in with this question: **p5.py, p6.py, p7.py, p8.py** (for graduate students) and **test_line_finder.py**. Please stick to these naming conventions to make the grading easier.

a. (10 points) First you need to find the locations of edge points in the image. For this you may either use the squared-gradient operator or the Laplacian. Since the Laplacian requires you to find zero-crossings in the image, you may choose to use the square gradient operator. The convolution masks proposed by Sobel should work reasonably well. Else, try your favorite masks. Generate an edge image where the intensity at each

point is proportional to the edge magnitude. In the README, write down which mask(s) you used and why.

```
def p5(image): #return edge_image
```

You may **NOT** use the `edge` function from the Image Processing Toolbox (or the equivalent in OpenCV). However, you can compare your output to the output of this function for verification (in fact this is a good idea to test out your code!).

b. (15 points) Threshold the edge image so that you are left with only strong edges. Return this image. We will not use edge orientation information as it is generally inaccurate in the case of discrete images. Next, you need to implement the Hough Transform for line detection. As discussed in class, the equation $y = mx + c$ is not suitable as it requires the use of a huge accumulator array. So, use the line equation $xsin(\theta) - ycos(\theta) + \rho = 0$. What are the ranges of possible θ values and possible ρ values (be careful here)? You can use these constraints to limit the size of the accumulator array. Also note them down in the README file.

The resolution of the accumulator array must be selected carefully. Low resolution will not give you sufficient accuracy in the estimated parameters, but very high resolution will increase computations and reduce the number of votes in each bin. If you get bad results, you may want to vote for small patches rather than points in the accumulator array. Note that because the number of votes might exceed 255, you should create a new 2d array and store the votes in this array. Once you're done voting, copy over the values into an output image, scaling values so that they lie between 0 and 255. This image should be of the same resolution as your hough array, which will be different from the input image resolution, in general. In the README, write down what resolution you chose for your accumulator array (and why), what voting scheme you used (and why), and what edge threshold you chose.

```
def p6(edge_image, edge_thresh): # return
      [edge_thresh_image, hough_image]
```

c. (8 points) To find "strong" lines in the image, scan through the accumulator array looking for parameter values that have high votes. Here again, a threshold must be used to distinguish strong lines from short segments (write down the value of this threshold in the README; it can be different for each image). After having detected the line parameters that have high confidence, paint the detected lines on a copy of the original scene image (using the `cv2.line` function in OpenCV). Make sure that you draw the line using a color that is clearly visible in the output image.

```
def p7(image, hough_image, hough_thresh):
```

```
    #return line_image
```

d. (600.661: Required for Graduate Students) (10 points)

Note that the above implementation does not detect end-points of line segments in the image. Implement an algorithm that prunes the detected lines so that they correspond to the line segments from the original image (i.e. not infinite). Briefly explain your algorithm in the README.

```
def p8(image, hough_image, edge_thresh_image,
hough_thresh):
     #return cropped_lines_image
```

Total points: undergraduates 100, graduates 120