ECE4580 Homework #1

Due: Jan. 20, 2017

Below are the problems of the homework. Please try to submit as a pdf document your results. On t-square you should be able to find the necessary Matlab files containing all that you will need to implement the homework.

Note: There is no need to turn in any code. It is sufficient to include your lines of code in the homework document, or the values requested.

Problem 1. (10 pts) The goal of this problem is to get your Matlab juices flowing and to also demonstrate another kind of problem you may have to solve now and again (if you continue in the world of computer vision). In linear algebra, you all learned about eigenvalues and eigenvectors. Well, turns out there is this other, related concept called the singular value decomposition (SVD). We will discuss the math shortly, but for now, please note that there is a function called `svd` in Matlab that computes the decomposition.

Given the matrix

$$A = \begin{bmatrix} -7 & 6 & -2 & -9 \\ -14 & -5 & -4 & 0 \\ -12 & 14 & 8 & -2 \\ 10 & -14 & 9 & 4 \end{bmatrix}$$

run the command

```
> [UU, SS, VV] = svd(A);
```

and turn in the singular values from the matrix `SS`. You should notice that they are ordered largest to smallest along the diagonal. The columns of `UU` and `VV` relate to the singular values and are also ordered (first column belongs to the first value, and last to the last). Also turn in the last column of `VV`, which corresponds to the smallest singular value of `SS`. (These answers will be written and should be turned in on paper or electronically in a homework submission pdf).

For those who are interested, the singular value decomposition is related to the eigenvalue decomposition. It is spiritually similar.

Hint: The Matlab function turns a vector into a diagonal matrix, or it turns a diagonal matrix into a vector. All depends on what is passed. This can be used to vectorize the singular value matrix

Problem 2. (15 pts) Implement the projection of a point in the world to a point on the image plane. Use the pinhole projection equations given by,

$$r^1 = f\frac{x}{z} \qquad \text{and} \qquad r^2 = f\frac{y}{z},$$

where the focal length parameter is $f = 6/10$. These should be implemented in Matlab by a function `pcamera.m` that takes in $(x, y, z)^T$ world coordinate in meters (in 3x1 column vector form) and returns $(r^1, r^2)^T$ image plane coordinates in meters (in 2x1 column vector form). There should be a Matlab code stub called `pcamera.m` in the assignment workspace (for you to fill out).

Where do the following points project to under the pinhole model?

$$q_1 = \begin{Bmatrix} -36.6 \\ -25.7 \\ 66.0 \end{Bmatrix} \quad q_2 = \begin{Bmatrix} 23.1 \\ 0.1 \\ 77.0 \end{Bmatrix} \quad q_3 = \begin{Bmatrix} -45.0 \\ 49.7 \\ 150.0 \end{Bmatrix} \quad q_4 = \begin{Bmatrix} -81.0 \\ 89.5 \\ 270.0 \end{Bmatrix}$$

If the domain of the image plane was limited to the range $[-0.5, 0.5]$ in both coordinates, do all points project onto the image plane? How many unique projections are there onto the domain of the image plane?

Problem 3. (15 pts) Many times in computer vision, a decision must be made given some kind of numerical criteria. For scalar values, one simple decision is to threshold the quantity. Anything bigger than a given value belongs to class A, while anything smaller belongs to class B (the equality case is application dependent). Humans rarely apply such a technique due to its rather arbitrary and global nature. Let's explore how such a simple decision can vary when we let a computer numerically decide. You will partially play the role of the computer in this case.

As part of this homework problem, you will find a file called `edgethresh.mat` which has two Matlab variables in it, `edges1` and `edges2` plus an image `tsrb1`. Both `edges1` and `edges2` are images that represent the output of

edge scoring algorithms. The algorithms ouput a numerical quantity that indicates how strong the evidence is towards that particular region being an edge.

Look at the image and identify, in your head, where you think all of the edge looking parts of the image should be. Then do the following in order (the a, b order is more important):

a) Modify the `plotEdges.m` m-file to include the threshold on the edge score. For this part, turn in the thresholds used for each edge image, plus turn in the resulting edge plot images.

b) Now, most likely you will have noticed some unpleasant behavior in the thresholding process. Plus you wonder, how in the world would a computer decide the threshold. Well, some people like to write a program that analyzes the statistics of a property and use the statistics to decide the threshold.

   Use the Matlab's histogram function `hist` to generate a histogram of the edge scores. Edge's are sparse objects in the scene (meaning that the should represent a small percentage of the scene). Using the histogram plot, pick a threshold for the edge score based on a value that rejects the majority of the edge scores. (In statistics, one would say to avoid the peak and pick a value more in the tail of the distribution).

   Using the histogram alone, pick a threshold for each edge score and repeat part a. Use the `histEdges.m` m-file for this. Turn in the same information as part (a), but also include the histogram plot and mark where your threshold was in relation to the histogram.

   Hints: The `hist` *function in Matlab implements a histogram, but it takes a 1D array, not a 2D array. So be careful to send the proper argument. Also, the second argument to histogram are the histogram bins. I am not a fan of the default bin selection. You may want to consider passing in your own bin array. You can pick the bin centers, use* `linspace` *to generate them, or use the* `[xmin:dx:xmax]` *way of generating a regularly spaced linear array of values.*

c) Comment on the joy or frustration of selecting an edge set that roughly aligns with what you envisioned. What were you trying to achieve, but somehow not being able to using the thresholds? If you did get what you wanted, then explain why you feel satisfied by the final answer plots.

Problem 4. (15 pts) Matlab has several built in function to perform image processing or image filtering. One such function is the `imfilter` function. Let's get used to this function by applying an averaging filter to some image. To apply the filter requires invoking the Matlab command

```
> aFilt = ones(n)/(n^2);
```

which generates an $n \times n$ matrix of ones, then divides by the size of the matrix. When this matrix is applied to an image using Matlab's `imfilter` command, it basically replaces each pixel with an average of its neighbors. The neighborhood size is determined by the number n.

a) Apply the imfilter command to the `campus3` image in the `filterme.mat` Matlab file with a filter size of 5. Now the averaging filter ends up smoothing things, which tends to have the effect of blurring the image. Tell me one area that is no longer legibile due to the blurring, and one area that can still be read in spite of the blurring. Why did the area pass through OK? What would be needed to make it unreadable?

b) The averaging filter matrix is an example of a convolution kernel. Read the course textbook, the online books, or any other reliable source and describe another convolution kernel besides the averaging filter. What form does it take and what does it do?

Hint: The images are in `uint8` *format which* `imfilter` *cannot really handle. A good first step would be to convert them to* `double`*. An example conversion of the image* `I` *would be:*

```
> dI = double(I);
```

Also, when in doubt type `help MatlabCommand`*, with the actual matlab command you are interested in knowing more about, to get some help.*