ECE332 Introduction to Computer Vision, MP#5

The due date is 10/27/2020 (Tu).

1 Canny Edge Detector

The purpose of this MP is to give you a chance to play with pixels by implementing the Canny edge detector which consists of many smart ideas in image processing. If you use Matlab, you do not need to consider the speed of your implementation at this point. To implement it, what you need to do in this MP are the following steps:

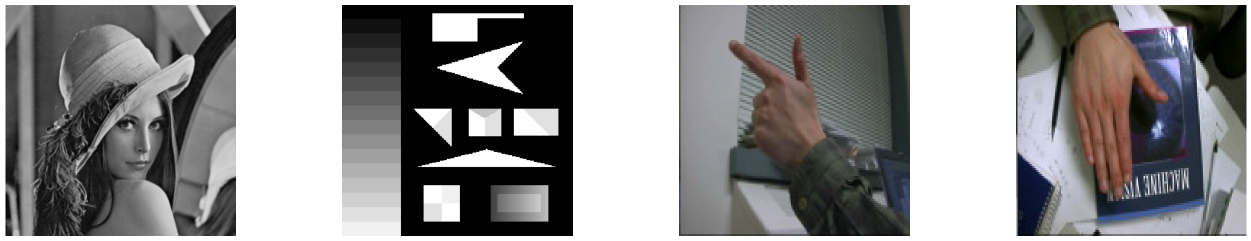
- *Gaussian Smoothing.* Using a Gaussian filter to smooth a given gray scale image. Your prototype could be `S = GaussSmoothing(I, N, Sigma)`.
- *Calculating Image Gradient.* Once you get the smoothed version of the input image, you need to get the image gradient of it. You can use the Roberts cross or Sobel or whatever to produce the magnitude and direction of your edge map. The prototype could be `[Mag, Theta] = ImageGradient(S)`.
- *Selecting High and Low Thresholds.* You need to determine two thresholds for later edge processing. As discussed in class, you'd better use the histogram of the image gradient to find the threshold, although you need some magic numbers, e.g. `percentageOfNonEdge` is the specified percentage of Non-edge area in `Mag`. Prototype could be `[T_low, T_high] = FindThreshold(Mag, percentageOfNonEdge)`, where You can use `T_low = 0.5*T_high`.
- *Supressing Nonmaxima.* One important idea in Canny edge detector is finding local maxima of image gradient based on nonmaxima supressing techniques. We've discussed two ideas in class, i.e., the quantization method and the interpolation method. You can use either of them (or both of them). You can use a LUT to simplify your implementation. The prototype could be `Mag = NonmaximaSupress(Mag, Theta, method)`;
- *Thresholding and Edge Linking.* The nonmaxima supressing techniques will give you quite good (thin) edges. The next step is to link the edges based on the two thresholds determined before. The high threshold produces strong edges, and the low threshold produces weak edges. The idea of Canny detector is to first recursively link the strong edges. When a strong edge ends, keep growing the weak edges to fill the gaps between strong edges. We've discussed that in class. The prototype could be `E = EdgeLinking(Mag_low, Mag_high)`.
- *Comparisons of Different Parameters.* You should play with different parameters, mainly, the Gaussian smoothing parameters (`N`, `Sigma`), and `percentageOfNonEdge`, to see different outputs.

- *Comparisons with Different Edge Detectors.* To see if Canny detector works better than other edge detectors, you'd better just compare different detectors on different testing images. Here, you can use the functions provided by Matlab, e.g.,

$$\begin{aligned}
\texttt{E} &= \texttt{edge}(\texttt{imginput},'\texttt{sobel}') \\
\texttt{E} &= \texttt{edge}(\texttt{imginput},'\texttt{roberts}') \\
\texttt{E} &= \texttt{edge}(\texttt{imginput},'\texttt{zerocross}')
\end{aligned}$$

You can also create your own test images. A special attention is that since the edge linking step uses recursion heavily, your Matlab setting may not allow you use a large stack. You can use `set(0,'RecursionLimit',N)` to specify a larger stack, say `N = 1000`. Some images are shown here[1]. Be careful that all the images are 24-bit RGB image, although some of them look gray scale.

Since this MP may take a little bit more than than previous MPs, I suggest you test and debug each of your smaller functions before put them together. You should create testing cases of your own. It is a good idea to check every intermediate results by plotting them out.



(a) (b) (c) (d)

Figure 1: testing images.

2 What to turn in

Each individual student should turn in his/her own solution. What you need to turn in includes:

- your code;
- a short report ($\leq$ 1 page is fine);
- your results on these testing images.

[1]you can download these images from our course website