University at Buffalo, Computer Science and Engineering

CSE 468/568: Robot Algorithms

Homework 2: Simple Image Processing*

In this assignment, you will be implementing some basic image processing algorithms to quickly clean up images obtained from a camera and detect features to be used for localization and mapping. To simplify this portion, we will implement these algorithms in MATLAB. MATLAB novices can get started using the tutorial from here.
`http://cseweb.ucsd.edu/~sjb/classes/matlab/matlab.intro.html`
As an alternative, you are welcome to use Octave, an open-source implementation of MATLAB that is freely available on Linux. Octave novices can get started using the tutorial from here.
`https://en.wikibooks.org/wiki/Octave_Programming_Tutorial`
You are not allowed to use calls to image processing functions in MATLAB/Octave in this assignment. You may however compare your output to the output generated by the image processing toolboxes to make sure you are on the right track.

Convolution

Write a function that convolves an image with a given convolution filter
`function [img1] = myImageFilter(img0, h)`

The function will input a greyscale image `img0` and a convolution filter stored in matrix `h`. The function will output an image `img1` of the same size as `img0` which results from convolving `img0` with `h`. You will need to handle boundary cases on the edges of the image. For example, when you place a cnvolution mask on the top left corner of the image, most of the filter mask will lie outside the image. One solution is to output a zero value at all these locations, the better thing to do is to pad the image such that pixels lying outside the image boundary have the same intensity value as the nearest pixel that lies inside the image. Your code can not call on MATLAB's `imfilter, conv2, convn, filter2` functions or other similar functions. You may compare your output to these functions for comparision and debugging.

*Parts of this assignment are borrowed from Prof. Srinivasa Narasimhan's Computer Vision course

Edge Detection

Sobel Filter

Based on this one-dimensional analysis, the theory can be carried over to two-dimensions as long as there is an accurate approximation to calculate the derivative of a two-dimensional image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. The Sobel edge detector uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The actual Sobel masks are shown below:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Magnitude of the gradient can be calculated using $\|G| = \sqrt{{G_x}^2 + {G_y}^2}$. An approximation is $|G| = |G_x| + |G_y|$

Edge detection homework

Write a function that finds edge intensity and orientation in an image.

```
function [Im Io Ix Iy] = myEdgeFilter(img, sigma)
```

The function will input an image (`img`) and `sigma` (scalar). `sigma` is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. The function will output `Im`, the edge magnitude image; `Io` the edge orientation image and `Ix` and and `Iy` which are the edge filter responses in the x and y directions respectively.
First, use your convolution function to smooth out the image with the specified Gaussian kernel. This helps reduce noise and spurious fine edges in the image. To find the image gradient in the x direction `Ix`, convolve the smoothed image with the x oriented Sobel filter. Similarly, find `Iy` by convolving the smoothed image with the y oriented Sobel filter.
The edge magnitude image `Im` and the edge orientation image `Io` can be calculated from Ix and Iy
In many cases, the high gradient magnitude region along an edge will be quite thick. For finding lines its best to have edges that are a single pixel wide. Towards this end, make your edge filter look at the two neighboring pixels along the gradient direction and if either of those pixels has a larger gradient magnitude then set the edge magnitude at the center pixel to zero. Your code can not call on MATLAB's edge function or other similar functions. You may use edge for comparision and debugging.

Harris Corner Detector

Harris Corner Detector is one of the most common feature point detector. It works by analysis the covariance of local gradient distribution using the following equation:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

where I_x, I_y are the x and y gradients respectively, and $w(x,y)$ is the windowing function (simplest case, w=1)
Remember, we discussed in class an efficient way of computing this is using the *deteminant* and *trace* of the matrix M

$$R = detM - k * (traceM)^2$$

where R is the corner strength function at a point. A point is a corner if the corner strength is above a threshold. k is usually 0.04.
Write a function that detects the Harris corner features from the edge gradients I_x and I_y.
`function [R] = myHarrisCorner(Ix, Iy, threshold)`
Play with the threshold to identify only the features that are visually significant. Your output is a set of feature points superimpose on the input images. Use small red or blue circles to superimpose the features.
Attached are five images that you can test your code on.

Submission

You will submit the following files
`myImageFilter.m`
`myEdgeFilter.m`
`myHarrisCorner.m`
.
Along with the MATLAB files, submit the five images with the harris corner features superimposed on each image. Submit all the files as `lab3.tar.gz` using the `submit` scripts as before. We will likely test on other images as well. The assignment is due on Oct 21, 2016 by midnight.