Computer Vision: Lab 1

This lab includes three parts. We will use the Matlab environment throughout. Part one uses different kinds of filters to manipulate the image. In part two, we will write a convolution function to understand the process of convolution. Part three introduces an example of fitting a line to detected edges. Attached files are two example images, and the skeleton lab1part3.m. You may wish to read the documentation on matlab functions mentioned in the following by `doc functioname`. The instructions also include supplementary exercises for keen students. These can be skipped first, and tried after the core exercises are complete.

Part 1: Warmup with Convolution and Correlation

1.1 Denoising

In this section, we will warm up by trying different filters to manipulate the image. You can read an image into matlab, and view it using commands like:

```
img =double(rgb2gray(imread('picture/input.png')))/255;
imshow(img);
```

We can create a noisy version of this image to work with denoising:

```
imgn = imnoise(img, 'salt & pepper', 0.02);
imshow(imgn);
```

Now we can make a 3x3 box filter and convolve it with the image for denoising.

```
filter = 1/9*[1 1 1;1 1 1;1 1 1];
imgdn = conv2(imgn, filter,'same');
imshow([imgn,imgdn]);
```

You should see a denoised but slightly blurred version of the input image.

Points to Explore:

1. Increase the size of the box filter (5x5, 7x7, etc) and observe the tradeoff between increased denoising effect and increased blurring effect.
   - Hint 1: You can use `ones(h,w)` to generate a size hxw matrix of 1s.
   - Hint 2: Remember to normalise the filter so that the values sum to 1.0.
2. Verify that you can achieve the same effects with the matlab <u>correlation</u> rather than <u>convolution</u> routine `filter2()`.
   - Hint 1: filter2 expects image/filter arguments in reverse order to conv2.
   - Hint 2: You can compare the numeric similarities of two matricies like: `norm(imgdn(:)-imgdn2(:))`. The (:) notation reads out a matrix as a vector, and `norm()` gets the magnitude of a vector.
   - Why do convolution and correlation give the same result here?
3. Compare the output of convolution with a Gaussian kernel to the box filter kernel.

- Hint: You can use `filter=fspecial('gaussian',h,s)` to create a Gaussian filter. You can also visualise the filter with imshow(filter).
- Observe that changing the standard deviation of the Gaussian can also be used to create different degrees of blur.
    - Bonus [Moderate] : Implement your own Gaussian kernel rather than using fspecial.

4. Use the <u>median filtering</u> routine `medfilt2` to achieve more effective de-noising with less blurring.

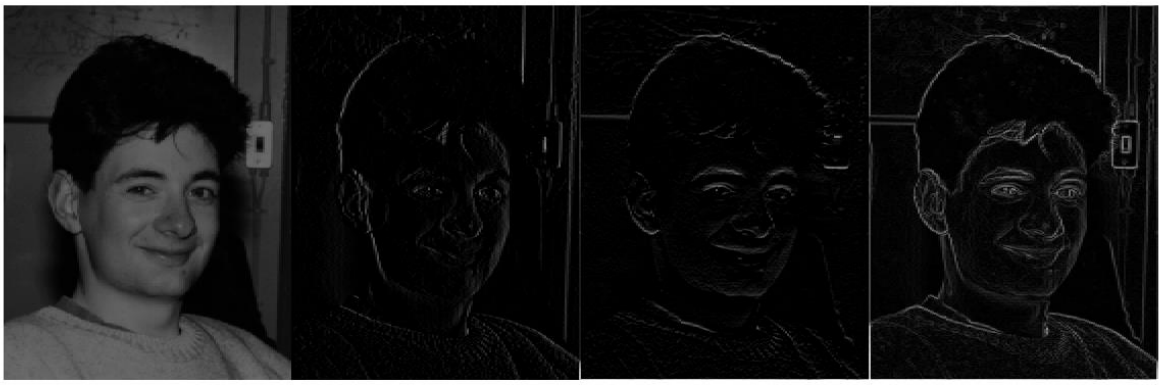
1.2 Edge Detection

Now define and apply an edge detection filter like:

```
img =double(rgb2gray(imread('picture/input.png')))/255;
filter_edge = [-1,0,1];
img_edge = conv2(img, filter_edge,'same');
imshow([img,img_edge])
```

Points to Explore:

1. Modify this example to perform horizontal edge detection.
2. Filter (`filter2`) rather than convolve the image with `filter_edge`.
    - This time the output of filter2 and conv2 are different. Why?
    - What has filter2 computed instead?
    - Adjust the call to filter2 to make it produce exactly the same output as conv2.
    - Hint: Check out `rot90()`.
3. Compute the overall <u>magnitude</u> of edge pixels in both x and y directions. Recall the formula: $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$. It should look like this, showing the original image, the vertical edges, the horizontal edges, and the magnitude (the edges are multiplied by 3.0 to increase visual contrast):



4. Explore the impact of filtering the image first with a blur filter (box or Gaussian), before performing edge detection, as discussed in the lecture notes. You should be able to use this to reduce the "noise" edges, and detect thicker/thinner edges.
5. Bonus [Moderate]: Verify the associativity of convolution numerically. If you convolve the edge filter and blur filter first, then convolve the result with the image; it should achieve the same result as sequentially blurring then edge detecting. You can use `tic` and `toc` in matlab to measure execution time. Which way is faster?

6. Bonus [Moderate]: Compute and visualise the edge direction at each pixel.

Part 2: Understanding Convolution

To make sure you understand convolution properly, implement your own convolution routine that inputs an image, a filter, and returns an image that is the result of convolving the image with the filter.

- For the simplest implementation, it is probably easiest to use the semantics of shape='valid' (see lecture notes, documentation of matlab `conv2`) so the output image is the difference of the input image and filter size.
- To check your implementation, take an example image and filter, and verify that your routine gives the same output as the built in matlab `conv2`.
    a. Hint: Call `conv2` with the same shape parameter that matches the edge semantics of your convolution routine.
- Bonus [Advanced]: Vanilla convolution takes $O(N^2M^2)$ cost. Implement fast O(Nlog(N)Mlog(M)) convolution with FFT (Hint: matlab `fft2`) https://en.wikipedia.org/wiki/Convolution_theorem

Part 3: Line Detection

In this section we will detect edges in a road image and use these to fit lines to the image. A self-driving car may need to do this in order to localise the road for steering. Open the provided skeleton code `lab1part3.m`. This code performs edge detection, and extracts a list of coordinates of every edge point in the left (`edge1`) and right (`edge2`) parts of the image. These will correspond to the two sides of the road.

- Implement the simple least squares line-fitting from the lecture notes to fit a line to the left and right sides of the road. It should look like this:

- Bonus [Advanced]: Implement a RANSAC line detector for robust line fitting.