CS6640 – Project 2
Assigned Sept. 19, 2012
Due Oct. 3 (Just before midnight)
Instructor: Guido Gerig
TA: Lingbing Jiang, Office MEB 3115, office hours Tue,Thu 3 - 5pm

Goals

The purpose of this assignment is to get familiar with implementatin and application of spatial filters for image smoothing, edge detection and template matching.

A note about scaling of filters: For smoothing filters there is a need to normalize the weights so that the sum adds up to 1, otherwise image intensity gets scaled. For derivative filters that sum up to 0 (see later), this is not really necessary for just getting edge maps. However, thinking about a function and its correct gradient would you do functional analysis, an appropriate normalization would be required. Here, the width of the mask would need to be considered, i.e. a filter (-1,1) would calculate signal difference across one pixel distance and no normalization required, whereas a filter (-1,0,1) calculates a difference across 2 pixel distance and the result has to be divided by 2. As you see from the book and some slides, this scaling is often not rigorously followed through as it mostly plays a role in multi-scale analysis schemes (discussed in advanced classes). In this project, feel free to make your choice. The scaling does not have an effect on the gradient direction (ratio) and only scales the edge magnitude image globally (see equation).

1 Image Smoothing

Implement a spatial filtering schemes for quadratic, symmetric filters. For simplification, do not filter the boundary as discussed in the course. Two schemes have to be implemented, one for 2D k*k-size kernels and one for separable filtering with 1D kernels of size k in horizontal and vertical directions.

1.1 k*k smoothing kernel with equal weights

Implement a 2D scheme for 2D square filters of size k*k, and design a filter with equal weights (remember to normalize to 1). Use a k*k smoothing kernel (3*3, 5*5) to reduce noise and smooth input image. Apply a 3*3 and 5*5 smoothing to your favorite black and white images.

1.2 Separable Filtering Scheme

Implement a filtering scheme for separable filters (see course slides) where 1D filtering in horizontal followed by vertical is applied. Use this separable scheme to achieve the same effect of smoothing filtering (3*3, 5*5) by applying 1D smoothing filters with 3 and 5 pixel width. Again, normalization of the filter masks is required.

1.3 Separable Gaussian Filter

Use the separable filtering scheme for 2D Gaussian filters. We will the standard equation of a 1D Gaussian filter $G(x,\sigma) = \frac{1}{\sqrt{(2\pi*\sigma^2)}} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$ to construct 1D filters with weights that fall off with distance from the center.

Please note that for a symmetric filter centered at a pixel, you can set the mean value μ to 0.

By choosing a specific filter of width σ pixels, you can estimate mask weights for a symmetric filter in the range of $\pm 3\sigma$. For example, a filter of width $\sigma = 2$ extends 6 pixels to both sides from the center and thus

forms a symmetric mask of size (2*6+1) pixels. Please note that the 1D filter weights need to be normalized to sum up to 1. This 1-D filter can then be applied in horizontal followed by vertical direction, which results in an efficient 2D filtering scheme.

Apply the Gaussian filtering to the same image(s) as before and compare results.

2 Edge Detection

Objects in images can be segmented by detection of object contours (edges) followed by description of these contours and classification.

2.1 Local Edge Filtering

As discussed in class, edge detection can be implemented as a horizontal and vertical differentiation filtering. Common practice are 3*3 masks as shown in class. **Please note that in the class slides, only strictly horizontal and vertical masks are shown, whereas in the book figure 3.41 last row full 3x3 masks are introduced - you can choose either or. The full 3x3 masks already include some local smoothing**. Application of these masks for vertical and horizontal filering results in estimates of image derivatives $I_x = \frac{\partial}{x} I(x, y)$ and $I_y = \frac{\partial}{y} I(x, y)$. These derivatives can be displayed as images (note that you get positive and negative values and that for display, it **might be necessary to map those into a positive range - but this should be done only for display purposes**. **Please also see comment in the introduction w.r.t. scaling of derivative filter masks.**

The two partial derivatives from a gradient $(I_x, I_y)^T$, which is a vector. The vector norm $\sqrt{I_x^2 + I_y^2}$ is calculated to result in an edge map (**see also book section 3.6.4**). The gradient orientation is calculated as $\alpha = \tan^{-1}(I_y/I_x)$ and this local edge orientation forms another important information for subsequent processing of edge maps.

Implement the calculation of partial derivatives and calculation of the edge map. You can also display the edge orientation by assigning an orientation angle to each pixel and displaying the image in the range of 0 to 360 degrees (**if your display is limited to 0 to 255 just scale the value range appropriately**). Display the **four** results as 4 images (dx, dy, edge map, orientation map) and discuss.

2.2 Edge filtering at specific scales

Whereas the previous section applies edge detection directly on the noisy images, a common approach is edge detection at a specific scale, i.e. estimation of edges after Gaussian filtering.

You can easily do that by first filtering images with a Gaussian filter (implementation above) and then applying local edge detection. Test this procedure by filtering with a specific filter width (e.g. $\sigma = 2$) followed by gradient calculation. Please note that due to the 2D Gaussian filtering, you can reduce the gradient calculation to simple 1D filters with length 3 ($\frac{1}{2}[-1, 0, 1]$) applied in horizontal and also vertical direction, (**since the filtering already provides noise reduction and correlation across the 2D domain**).

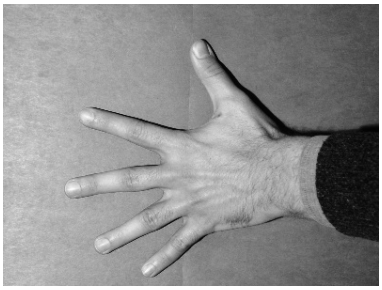
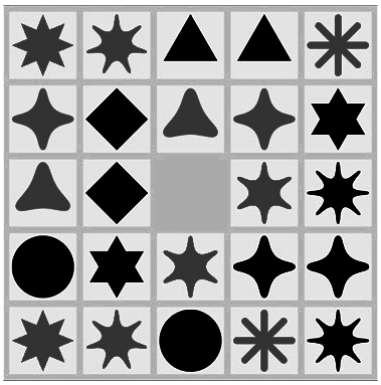
3 Template Matching

Spatial filtering is also used for finding specific objects in images by template matching. Given a template of a sought object, the template acts as a filter, and the maximum correlation indicates the position of the object.

Use the 2D spatial filtering technique implemented before to find/segment image objects. Now, the filter weights are not a user-defined mask but a mask that is represented as a small image template.

Steps:

- Select an image presented a repeated pattern of same size/orientation objects where the task is to find the position of objects (e.g., similar cars in a parking lot, specific letter in an image of a text document).

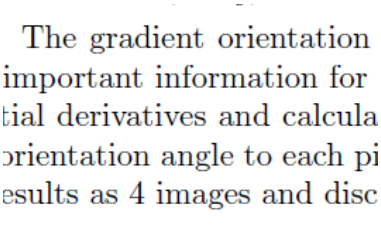



Figure 1: The following are test images that can be used for illustrating your solutions. Please feel free to use your own images. Filtering and edges: captitol.jpg , hand-bw.jpg. Ttemplate matching: shapes-bw.jpg , keys-bw.jpg, cars-bw.jpg, text.png.

- Select a template, i.e. a very small subimage containing your object.
- Use the template as a mask m*n, and filter the image with this mask to obtain a correlation image.
- Apply thresholding to the correlation image to get the set of correlation peaks, whih are indicating the locations of the template structure.
- For more advanced experts: Try to develop a procedure that finds these peaks and creates a list of their posion. Then, put the template at each of those positions to generate a resulting image that shows the segmentation result.
- Hint: the procedure above can also be applied to binarized/thresholded images, i.e. using thresholded images as input and a binary template. This way, the correlation peaks signify the number of pixels in the image and the template that overlap. The advantage would be that the correlation peak is independent of the absolute gray values. Would you have time, you can try this version in addition to gray-scale template matching.

4 Instructions, Requirements, and Restrictions

1. Please use your name **"NAME"** in all reports and submitted materials, to uniquely identify your projects.
2. Write your project code in a single directory, called `project1-NAME`.
3. For Matlab each individual function (including functions you define) should be a ".m" file, and your functions should call one another as necessary.

4. We **do not allow to use Matlab toolbox functions** (e.g. Imaging Toolbox) or other existing image processing libraries in order to give all students the same conditions for code development [1].

5. You should have in your report a short description of each algorithm you used and documentation on how your code is organized. Failure to do this will result in a loss of points. Please remmember to **add your name** to the report title.

6. Your project report will be in the form of an html file called `index.html`, contained in that directory. All links from that file must be relative and all other files necessary to read your report must be in that directory (or subdirectories).

7. You should use examples of images in your report. They should be viewable in the browser when we open your html file.

8. You will submit a single tar file created from from your project directory with the unix command *tar -czf project1-NAME.tgz./project1-NAME*.

9. **Submission has to be done through the CADE lab's handin system (you need an account but can do it via the web, see Project page for instructions).**

10. Please remember or look-up the honor code and requirement to provide your own solution as discussed in the syllabus.

11. **Please look up the late policy as defined in the syllabus**

[1] The core MATLAB package comes with several rudimentary functions that can be used to load, save, and perform custom functions on images. Taken from wikibooks