Computer vision Lab 2
Feature extraction

David Van Hamme

1 Edge detection

Many computer vision tasks rely on edge detection to find object boundaries. An edge in an image is a region where the intensity suddenly jumps. They can be detected by calculating the horizontal and vertical discrete derivatives of the image.

The simplest way to calculate the horizontal derivative of an image would be to filter it with the following kernel:

$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

however, kernels of even dimensions are typically not used because they calculate a value "in between pixels" so a better choice would be

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

To reduce the sensitivity to noise of the horizontal derivative, you can blur the image vertically prior to filtering with this kernel. Remember that vertical blurring could be achieved with a vertical all-positive kernel like

$$1/4 \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

Combining the two operations (blurring vertically and horizontal differencing) results in the following kernel, which is called the Sobel operator:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Note that this kernel sums to zero, meaning if you apply it to an image of uniform intensity, you will get zero response (which is what you expect for a derivative).

Exercise 1

Write a simple program that detects vertical edges in *building.png*.

Assignment 1 Use the Sobel operator to calculate the vertical first-order derivative. Ensure you capture and visualize the negative filter response values by using appropriate datatypes. Your result should look similar to the one shown in Figure 1.

Useful functions: **Sobel**.

Sometimes you want to detect edges in specific orientations. There are multiple ways to achieve this. The first is by making orientation-selective edge detection kernels. One example of such a filter is the Derivative of Gaussian (DoG) filter. As the name implies, it is obtained by taking the derivative of a two-dimensional Gaussian filter. Some examples are shown below. The more elliptical the Gaussian filter is, the more orientation selective the filter will be.

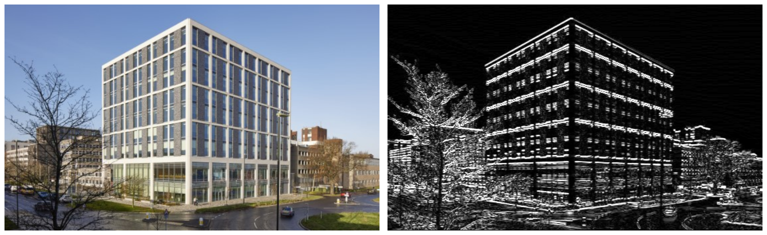
Figure 1: Vertical first-order derivative of an image.

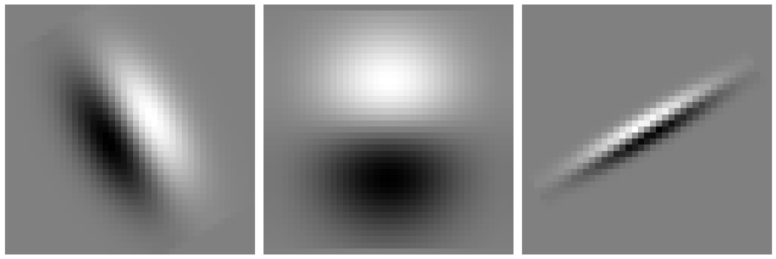
Figure 2: Examples of DoG filters.

Exercise 2

Write a simple program to create a DoG filter and detect the edges of the yellow strips in *rays.png* with it.

Assignment 2 Create a 15x15 DoG filter using the workflow below.

- create a 1D Gaussian kernel with the function **getGaussianKernel**;
- copy it to the middle row of a square matrix;
- create another 1D Gaussian kernel with a smaller standard deviation;
- filter the square matrix (containing the row kernel) with this column kernel to obtain an elliptical 2D Gaussian;
- derive this 2D Gaussian vertically with **Sobel** to obtain a DoG filter;
- make a rotation matrix for a 45 degree rotation with **getRotationMatrix2D**;
- rotate your DoG filter with this rotation matrix (**warpAffine**);

The result of the steps is shown below, but for a different orientation and filter size.

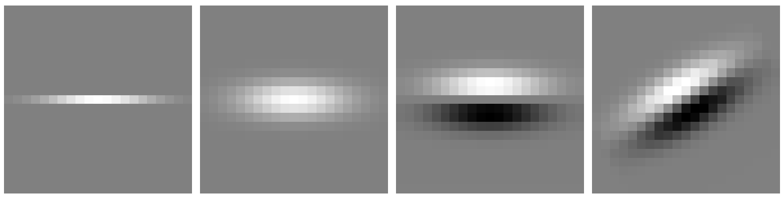
Figure 3: Steps in creating a DoG filter

Assignment 3 Filter *rays.png* with a well chosen DoG filter so that in the resulting image, the edges of the yellow strips stand out (you will need to take the absolute value of the responses).
Question 1 What happens when your filter goes "across the border" of the image?

Tip: to visualize floating point arrays like the DoG filter, rescale the values to $[-.5, .5]$ and add .5 to them so that black represents negative, gray zero, and white positive values, like with the code snippet below.

```python
def showFilter(name, f):
    cv2.imshow(name, 0.5*f/f.max()+0.5)
    cv2.imwrite(name + ".png", 255*(0.5*f/f.max()+0.5))
```

If you want all the edges from an image, strong or weak, in all orientations, the most used method is Canny edge detection. Canny uses Sobel kernels and hysteresis thresholding, meaning that strong edges are always returned but weak edges are only returned when they are connected to a strong edge. This results in more complete edges, but they often still need post-processing anyway.

Exercise 3

Write a simple program that detects all the edges in *rays.png*.

Assignment 4 Apply Canny edge detection with thresholds chosen so that the edges of all strips are detected.

2 Line fitting

Getting a binary image with edge pixels is only the first step in interpretation of the image content. Usually it will be followed by some geometrical analysis to fit shapes to the edge image. Two candidate algorithms are RANSAC and Hough. RANSAC is an algorithm that selects random minimal subsets of edge pixels needed to define a shape, and then checks if a sufficient number of other edge pixels fall close to this defined shape. The Hough algorithm transforms the edge pixel coordinates to a parametric space where each pixel "votes" on the parameter combinations that correspond to shapes that pass through the pixel. Parameters where the votes accumulate, describe shapes present in the image. For lines, this works in the two-dimensional rho-theta parameter domain. Each point in the rho theta plane represents a line at angle theta with the horizontal axis, at a distance rho of the origin. Each parameter combination that received a number of votes higher than a threshold value is considered as a detected line.

Exercise 4

Write a simple program that does Hough line fitting to *painting4.jpg*.

Assignment 5 Apply Canny edge detection so that you get the four edges of the painting, and as few other edges as possible. It is inevitable that you find other edges however.
Assignment 6 Apply **HoughLines** to the result of Assignment 5 and visualize the lines on the original image (use the **line** function). It is normal too get too many lines, since you cannot get the Canny result perfect.

3 Corner detection

For many applications, including most tracking and recognition algorithms, it is necessary to detect characteristic points in the image so that they can be compared to characteristic points from a subsequent frame or another image. A characteristic point from image processing perspective is a point that is well-located. Well-localized points are e.g. corners of objects, because the environment of those points and the environment of points next to them is very different. Poorly localized points are e.g. points on lines or in large uniform areas: the environment of those points is very similar to the environment of points just next to it.

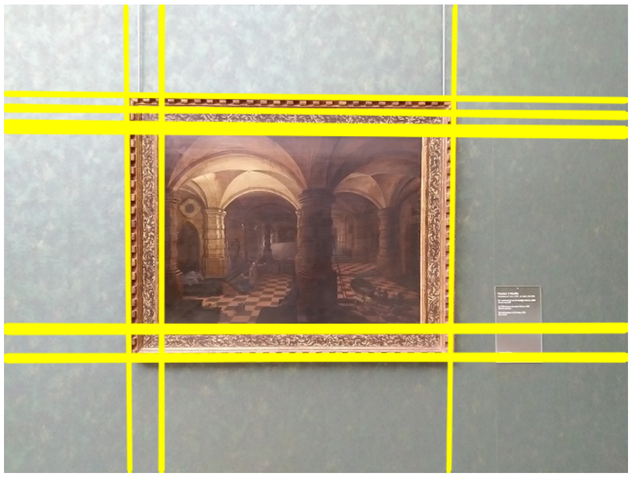

Figure 4: HoughLines result on a different painting

Algorithms that detect these points are called corner detectors. The most common is the Harris corner detector, but it is pretty slow. FAST features can be detected much faster, but are slightly less stable, for example with noise in the image or small intensity differences. To relate objects in different images to each other, SIFT or SURF features are often used. These calculate a descriptor of the environment around the point that can then be compared with the descriptors of the other image. A popular combination of detector and descriptor is FAST+BRIEF, which are combined into Oriented FAST and Rotated Brief (ORB).

Exercise 5

Write a simple program that detects corners in two images. Try to match the corners across the images.

Assignment 7 Detect Harris corners in *shot1.png and shot2.png* and visualize them side by side.
Question 2 Name two kinds of problems you foresee in trying to match these corners.
Assignment 8 Detect ORB features in each of the two original images, calculate the ORB descriptors for them, and match the descriptors between the two images. Visualize the 32 best matches.

Useful functions: **goodFeaturesToTrack, circle, ORB, BFMatcher, drawMatches**.

4 Reporting

Write a brief report about this lab session in which you answer each question and provide the output image of each assignment. Append your source code.