Computervision Lab 5
Feature detection

David Van Hamme, Sanne Roegiers

February 19, 2019

1 Edge detection

Edge detection is a basic component of many practical image processing systems (e.g. quality control, lane departure warning). The most commonly used algorithm for this is the Canny edge detector. The main advantages of this edge detector are that the output lines are only 1 pixel wide, and that hysteresis is used. This means that weaker edges are also detected on condition that they are connected to a stronger edge.

For some applications, however, it is interesting to detect only edges in a particular orientation. An orientation-selective edge filter is used for this, such as the vertical Sobel kernel from lab 2, for example.

2 Exercise 10

Write a program that extracts the edges of the yellow strips in **rays.png**. The name of the image is still a command line parameter. You can do this as follows:

- create a 1D Gaussian kernel with the function **getGaussianKernel**;
- copy it to the middle column of a square matrix (in C++ you may find **Mat::col** and **Mat::copyTo** useful functions);
- create another 1D Gaussian kernel with a smaller standard deviation and transpose it (**Mat::t**) to obtain a row matrix;
- filter the square matrix (containing the column kernel) with this row kernel to obtain a 2D Gaussian;
- derive them horizontally or vertically with **Sobel** to obtain a DoG filter (*differential of Gaussian*);
- make a rotation matrix for the correct angle of the yellow strips (about 75 degrees) with **getRotationMatrix2D**;
- rotate your DoG filter with this rotation matrix (**warpAffine**);
- convert the image to grayscale and filter it with the rotated DoG filter;
- take the absolute value of the filter response with **abs**;
- tweak the parameters of the Gaussians and the kernel size so that you get a good orientation selectivity;

Tip: to visualize floating point arrays like the DoG filter, you can determine the minimum and maximum with **minMaxLoc**, scale them to $[-.5, .5]$ and add .5 to them.

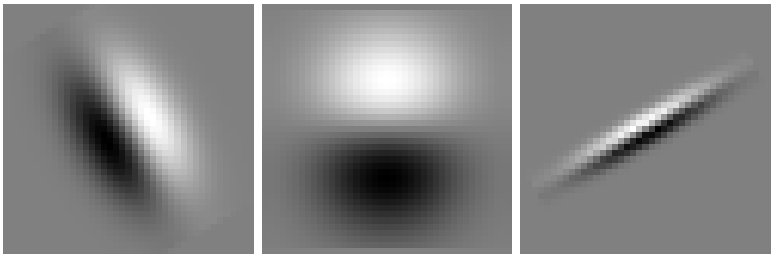

Figure 1: Examples of DoG filters.

3 Line detection

Once the edges in the image have been determined, we can look for straight lines. The most common method is the Hough transformation. 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. For each combination, the edge pixels that are located near this line are counted. This principle is called *parameter space voting* and can also be used for other parametrical objects (e.g. circles). Each parameter combination that received a number of votes higher than a threshold value is considered as a detected line.

4 Exercise 11

Write a program that detects and draws all lines in **rays.png**. New features: **Canny, HoughLines, line**.

5 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.

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.

6 Exercise 12

Write a program that reads two images (name through command line) and detects Harris corners in both . Test this on **shot1.png and shot2.png** and see if you detect the same features. New features: **goodFeaturesToTrack, circle**.

7 Exercise 13

Write a program that detects 96 ORB features in each of the two images that you provide through the command line, calculates the ORB descriptors for them, and matches the descriptors between the two images. Visualize the 48 best matches. New functions and classes: **ORB, BFMatcher, drawMatches**.

8 Report

Write a short report about how you solved the exercises. Include in this report your input and output images. Describe every new function that you used. Explain the basic algorithm and purpose of the functions and clarify the parameters and how you selected them. Upload your report in the form of **LabX_name.pdf** to the dropbox on Minerva.