ETH Zürich
Computer Science Department

Prof. Otmar Hilliges
Prof. Markus Gross
AS 16

Visual Computing

Exercise 2

All programming has to be done in Matlab. Some Matlab commands you might find useful:

- *conv2*: Two-dimensional convolution
- *fspecial*: Creates a two-dimensional gaussian filter
- *edge*:Built-in matlab function for finding edges
- *bwselect*: Returns a binary image containing the objects that overlaps the pixel (R, C).
- *atan2*: Four quadrant inverse tangent

2.1 Edge Detection

The purpose of this exercise is to investigate different edge detection algorithms. In this first section you are asked to implement edge detection using the Sobel and Prewire kernels. You can use the image lighthouse.png to test your algorithms and output a binary image with edge pixels marked.

a) Implement the Sobel edge detector (Sobel filters and thresholding)

b) Implement the Prewitt edge detector (Prewitt filters and thresholding).

c) Compare your outputs to MATLABs built-in functions and comment onn which you think is best. Note, the default parameter settings are not necessarily the best, and your results may differ from the built-in functions. List the unable parameters in each edge detector and explain how you chose settings for each.

2.2 Canny Edge Detectorl

Using the same input file, the goal of this section is to implement your own Canny edge detector. The Canny edge detector uses a filter based on the first derivative of a Gaussian. The steps are

a) Implement the first-order derivatives of Gaussian kernels to remove noise from the image.

b) Optionally, you can also smooth the image with a Gaussian kernel first and then compute the gradients of the resulted blurred image. Compare the results with the one obtained in step a). You should use the same Gaussian kernels in both a) and b) for proper comparisons.

c) After either step a) or b), you should get a relatively thick edge image. A nonmaximal suppression algorithm should be implemented to thin the edges. The edge direction angle can be rounded to one of four angles representing vertical, horizontal and the two diagonals (e.g., 90, 0, 45, and 135 degrees). By passing a 3x3 grid over the image to carry out a search to determine if the gradient magnitude is a local maximum in the gradient direction. (If the central pixel is a local maximum, keep it. If the central pixel is not a local maximum, set it as a background pixel)

d) From the output of the previous section, implement the hysteresis thresholding and generate binary edge maps. Thresholding with hysteresis requires two thresholds T_low and T_high. If the magnitude of the non-max suppressed edge map is below T_low, it is set as a background pixel (i.e., 0). If it is above T_high, set it to be 1. If the magnitude is between the 2 thresholds, then it is set to 0 unless these is a path from this pixel to a pixel (in its 8-neighbours) with a gradient above T_high. One possibility is to use the MATLAB function bwselect which finds overlapping objects. Here, you can ignore orientations.

e) Compare your results with the built-in MATLAB Canny edge detector.

f) Compare your results with the edge detectors in previous section.