Computer Vision 1: Exercise Sheet 5

Summary:

1. Smoothing before edge detection
2. Edge detection with first order methods: Sobel filter
3. Edge detection with second order methods: Laplacian

1 Preparation

As explained in the lecture, smoothing is applied before edge detection on images since noise negatively affects edge detection. To examine the effect of this in practice, we create a noisy image and smooth it.

- Load the image `woman.png` from Moodle.
- Using `skimage.util.random_noise`, add Gaussian distributed noise to the image with a variance of 0.01. This gives you a noisy image N.
- Apply `skimage.filters.gaussian` to filter N. Use $\sigma = 1.0$. This gives you a smoothed image S.

2 Sobel filter

The Sobel filter is an example of a first-order derivative method to edge detection.

- Use the function `skimage.filters.sobel`. This function applies the two Sobel kernels (horizontal and vertical) on the image, and then computes the root of the squared sum of the results.
- Visualize the result of applying the function on the noisy image N, and on the smoothed image S.

Let us call the output of the `sobel` function on the noisy image F_n and the output on the smoothed image as F_s.

- To detect edges, select threshold values t_n and t_s and create a binary mask by applying the logical operations $F_n > t_n$ and $F_s > t_s$ in NumPy. You can for example draw histograms of F_s and F_n to come up with suitable thresholds. Visualize the binary masks. If necessary, tune your threshold valuess so that the results show the outline of the woman's face.
- Why is edge detection improved by applying smoothing?

3 Laplacian operator

The result of applying the Laplacian operator on an image is roughly proportional to the average rate of change in the image intensity. The Laplacian is isotropic, that is, it is insensitive to the direction of edges or intensity changes in the image. The Laplacian operator is an example of a second-order derivative method for edge detection.

- Use the function `skimage.filters.laplace` on the noisy image N, and on the smoothed image S. Visualize the results.

Since applying the Laplacian is a second order method, to detect edges we find the zero crossings in the output L from the function `skimage.filters.laplace`. Your function should return a binary image B of the same size as L, where $B(i, j) = 1$ if (i, j) is a zero crossing, and $B(i, j) = 0$ otherwise. You can implement the following pseudocode as a function to detect zero crossings in L.

```
Input: Laplace filtered image $L$, a threshold value $t > 0$
Output: Binary image $B$ with ones indicating zero-crossings in $L$
  Initialize $B$ as all zeros matrix same size as $L$
  for all non-border pixels $(i, j)$ do
      $v \leftarrow$ value of $L(i, j)$
      $N \leftarrow$ values of neighbours of $L(i, j)$ (up, down, left, and right) with opposite sign to $v$
      if $N$ is not empty then
          $m \leftarrow \min_{n \in N} |n|$                    ▷ Smallest absolute value in $L$
          if $|v| < m$ then
              $d \leftarrow \max_{n \in N} |v - n|$            ▷ Greatest absolute difference of $v$ and any element of $L$
              if $d > t$ then                                  ▷ Clear enough zero-crossing?
                  $B(i, j) \leftarrow 1$                       ▷ $(i, j)$ is zero-crossing
              end if
          end if
      end if
  end for
```

- Use the strategy above to detect zeros in the Laplace filtered outputs, for both the noisy and the smoothed image.
- Visualize the edge detection results. Notice that without smoothing, it is almost impossible to get a meaningful edge detection result.