Technische Universität München
Fakultät für Informatik
Prof. Dr. Nassir Navab, Dr.
Selim Ben Himane

SoSe 2007
Exercise Sheet 2
May 08, 2007

Exercises in 3D Computer Vision

Image Processing, Edge Detection, Feature Extraction and Their Uncertainty

Exercise 4 (H) Discrete Convolution

The two-dimensional convolution is one of the central operations in image processing. Whenever we apply a linear filter to an image that takes small neigbourhoods of pixels into account we can achieve this by using the convolution. Each pixel within the filter mask is multiplied by the weighting factor of the mask. The products are summed up and the result is written at the position of the central pixel. If a and b are functions with two discrete variables, n_1 and n_2, then the formula for the two-dimensional convolution $c = a * b$ is

$$c(n_1,n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} a(k_1,k_2)b(n_1-k_1,n_2-k_2)$$

In our case of filtering an image a is a $K \times K$ filter mask and b is $M \times N$ image. In MATLAB the 2D-convolution is done with the command `conv2`.

a) Design a 3×3 filter mask that computes the mean intensity value within the mask and assigns the result to the center pixel! *Such a filter is called Mean-Filter.*

b) Apply your Mean-Filter on the image '`synthetic_noise.gif`' by using `conv2`.

c) Design a 3×3 Binomial-Filter that approximates the Gauss-Function for smoothing an image.

d) Look at both filter masks, Mean- and Gauss-Filter. What are the advantages of the Gauss-Filter?

Remark: *Additional to your written solutions for a, c and d, Please send us by email a m-file that shows the results of your mean filter and gauss filter on the image 'synthetic_noise.gif'.*

Exercise 5 (H) Edge Detection

Edges characterize boundaries and are therefore a problem of fundamental importance in image processing. The definition of an edge in image processing is a point within an image where there is a large variation in the intensity values between neighboring pixels. Therefore, edges in images are areas with strong intensity contrasts a jump in intensity from one pixel to the next. Edge detecting an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image. Often this edges in images correspond directly to the real edges between objects that are imaged. Edge detection is usually done by convolving the image with some appropriate linear filters.

a) Plot row 55 of the image '`synthetic.gif`'. *Set the the y-axis of the figure to minimum -300 and maximum 300 for a better visualization.*

b) Why is a filter that approximates the first or second derivative operator a good idea to detect edges?

c) Compute the approximation for the first and second derivative of the row 55 and plot them in different colors in the same figure as above.

d) Design filter masks that approximates the first order derivatives ($\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$).

e) Design a filter masks that approximates the second order derivative!
Hint : $(a * b) * c = a * (b * c)$
Reminder : The second derivative can be calculated by applying the first derivative successively.

f) Apply all your designed edge detection filters to the two images '`synthetic.gif`' and to '`synthetic_noise.gif`'. Especially the second derivative is extremely noise dependant. Plot row 55 after applying the second derivative filter on the noisy image.

g) How could the edge detection with a second derivative filter be improved for noisy images? How does the improved filter mask look like?

h) Apply your improved filter to the image '`synthetic_noise.gif`' and plot again row 55.

MATLAB Hint: The functions diff and gradient can be used for the approximation of the derivatives. After applying a derivative filter the result images contains usually also negative values which are not visible after calling image(I). Then, for better visualization, use imagesc(I).

Remark: *Please send us by email a m-file that shows the results of a, c, e and g.*

Exercise 6 (P) Binary Thresholding

In image segmentation thresholding is one of the simplest methods to do a segmentation. Therefore a user has to define a lower and an upper intensity threshold. The input image is transformed into a binary image in which all pixels are set to zero that lie outside the range that is defined by the thresholds. The pixels within the range are set to 255.

a) Write a function `thresholding(I,lower,upper)` that takes an image `I` and two intensities `lower` and `upper` as parameters. The function should return the filtered image.

The problem in thresholding is always to find the right range boundaries. To support the decision of defining thresholds it is often useful to have a look at the image histogram. The histogram is a plot of the intensity distribution. Peaks in the image histogram often correspond to objects in the image so a threshold at the minimum between two peaks could be a right choice for separating objects. When creating the histogram the user defines the number of bins in which the intensities are subdivided. The MATLAB function call `H = sum(hist(I,256)')` computes the histogram of the image `I` with 256 bins. A single bin for each intensity from 0 to 255. Try plotting the histogram with `bar(H)` instead of `plot(H)`.

b) Plot the image histogram for the image '`objects.gif`'.

c) Segment only the ellipses in the image '`objects.gif`' with your function `thresholding(I,lower,upper)`. Choose the right thresholds from the image histogram.

Remark: *Please send us by email a m-file that shows the results of b and c.*

Exercise 7 (P) Feature Extraction

In this exercise we want to determine where the segmented ellipses are located within the image. Therefore we want to compute the object centroids. First of all we have to distinguish between the single objects. This can be done by a so called 'labeling'. Labeling assigns an ID to each object. In the 2D case there exists an operator mask which does the labeling „on-the-fly". The mask consists of three coordinates, the actual pixel C (center), the pixel above U (up) and the pixel on the left-side L (left). The corresponding coordinates in the temporary image are identified with C^*, U^* and L^*.

1. The mask is shifted along the rows of the segmented binary image and equal-sized temporary image. If $C = 0$ the pixel belongs to the background. The corresponding pixel in the temporary image is set to 0. If $C \neq 0$ the following cases has to be distinguished:
   - $C \neq L; C \neq U \Rightarrow$ A new region has been found. The pixel C^* in the temporary image gets a new ID.
   - $C \neq L; C = U \Rightarrow$ The pixel C^* in the temp image gets the label of U^*.
   - $C = L \Rightarrow$ The actual pixel C^* is labeled with L^*.

   After the first step, the result will need to be refined and some classes have to be merged.
2. Scan the image, to find and replace the classes which are equivalents.
   Hint : *U^* and C^* belong to two equivalant classes, if $C^* \neq 0$ and $U^* \neq 0$ and $L^* \neq 0$, and $U^* \neq L^*$.*

In the temporary image all pixels are reassigned corresponding to the equivalence list.

a) Write the function `labeling(I)` that takes a segmented binary image `I` as a parameter and returns the image with labeled objects!

b) Write a function `computeCentroids(I)` that takes the result of the labeling as a parameter and returns a matrix containing all computed centroids!

c) Show the image '`objects.gif`' and plot the computed centroids as blue crosses overlaid onto the image!

Remark: *Please send us by email a m-file that shows the results to the question c.*

Exercise 8 (P) Moment Extraction

Moment extraction can be done by the mathematical tool of Principal Component Analysis (PCA). The PCA is a powerful method in the image processing topic known as feature selection. A nice introduction in the mathematics that are necessary for computing the principal components or moments can be found in the additional reference „A tutorial on Principal Components Analysis" by Lindsay I Smith. Please familiarize with these methods by reading chapter 3 of this tutorial.

We already know where the ellipses are located. Now we want to compute the main axis of one object via PCA.

a) Write a function `getLabelList(I)` that takes a labeled image `I` and returns a vector containing all labels that are used within the labeled image.

b) Write a function `getPixelList(I,id)` that takes a labeled image `I` and a label `id` and returns a vector containing the pixel coordinates that belong to the objects with the label `id`.

c) Get the pixel list of the ellipse that has its centroid at position $(105.6/217)$ and plot the points.

d) Do the Eigenvalue decomposition as it is described in the PCA tutorial on this ellipse. Overlay the image '`objects.gif`' with the computed main axes.
Hint: An Eigenvalue λ does not directly correspond to the extension of the main axis. Scale them by the following formula: $\lambda' = \sqrt{\lambda} * 2$. The covariance matrix of a data set can be computed with the function cov(x). The Eigenvalue decomposition of a matrix can be simply done by calling eig(M) that return two matrices, one containing the Eigenvectors, the other containing the corresponding Eigenvalues.

Remark: *Please send us by email a m-file that shows the results of c and d.*