INFOIBV – 2013/2014

Supporting documentation: practical assignment

This document is part of the practical assignment program it shortly describes the working of the different programmed pipelines to segment images'

The program we have developed is a terminal only program. It is programmed in C++ and compiled with a C++11 compiler. It's originally developed in Linux and later on compiled for Windows.
The program can be started by executing INFOIBV from a command prompt in Windows. There are six different pipelines programmed each according to a different image.

Because we miss understood the assignment we made different pipelines for different images instead of just one pipeline which can be used to detect the same object in different images. Therefor we discussed this issue and were granted to make a bit longer explanation to describe each of the pipelines.
In order to fulfill the assignment we have chose the xray pipeline as our most mature pipeline and we will describe this pipeline a bit more extensive than the others.

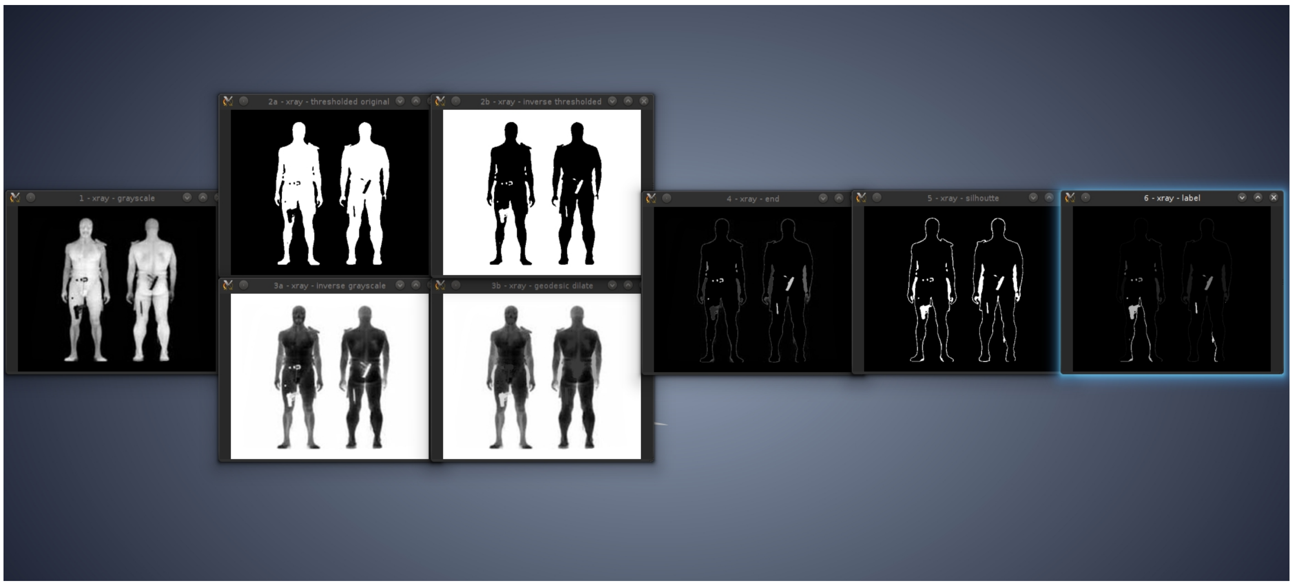
During the development of our toolkit we made an agreement that every function has to give an 8 bit unsigned char image back. Therefor we made our toolkit modular and extensible. This is the same for the input image of our functions. Some mathematical operations are more precise when executed with floating point precision. We made a special function to convert our unsigned char image to floating point notation and back. Our toolkit can easily be used to build your own pipeline for basic image processing.
The functions we have build are sorted in different category's: filters, morphology, statistics and a stockpile of basic functions.

Frank Uijtewaal, F130694
Jeroen van Prooijen, F130693

Pipelines

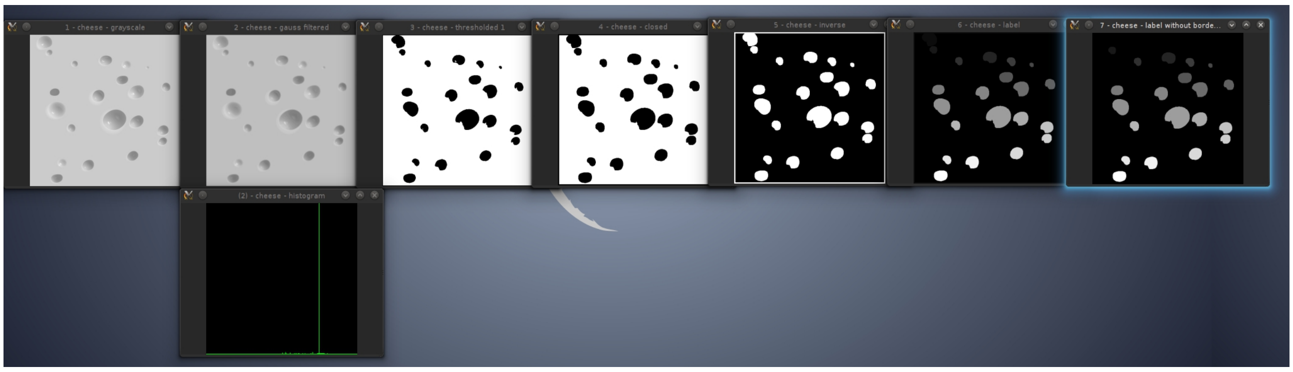
All the descriptions below are also shortly stated in the 'images.h' file.

Xray


1 - xray - grayscale
2a - xray - thresholded original
2b - xray - inverse thresholded
3a - xray - inverse grayscale
3b - xray - geodesic dilate
4 - xray - end
5 - xray - silhoutte
6 - xray - label

The pipeline to segment the *xray.png* image is done in two separated paths. First we make sure that the input image is converted to a gray level image.
The gray image gets thresholded and inverted. The result of these two operations will be used later on as a mask *(the upper two images of the four centered images).*
The gray image gets invert itself, then we create a matrix with only white borders. The inverse of the gray image and the white bordered image are then used to do a geodesic dilate with the inverse of the gray image as a control image. The result is a person where the objects in the body are grayed out. The objects which exceed the body are still white *(lower two images).*
The resulting image in the previous step is then subtracted from the thresholded and inverted gray image from step one. The result is a dark image with white objects. These objects are the black objects from the starting image and a small edge around the body.
After thresholding the image it turned into a binary image and can then be labeled which is the concluding function of this pipeline.

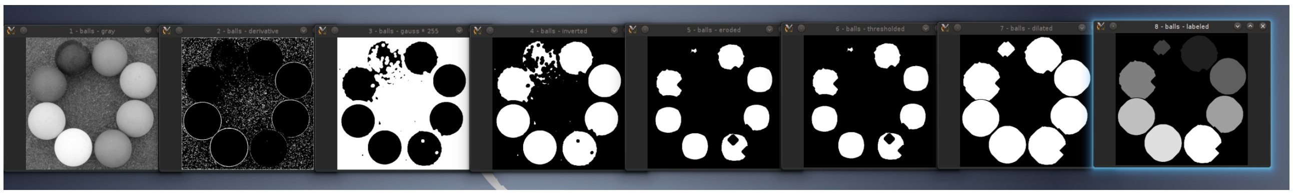
Cheese



The cheese picture is quite clean by itself. To make sure we have filtered it with a Gaussian filter which causes the least artifacts in an image. Therefor we needed a Gaussian kernel, we first

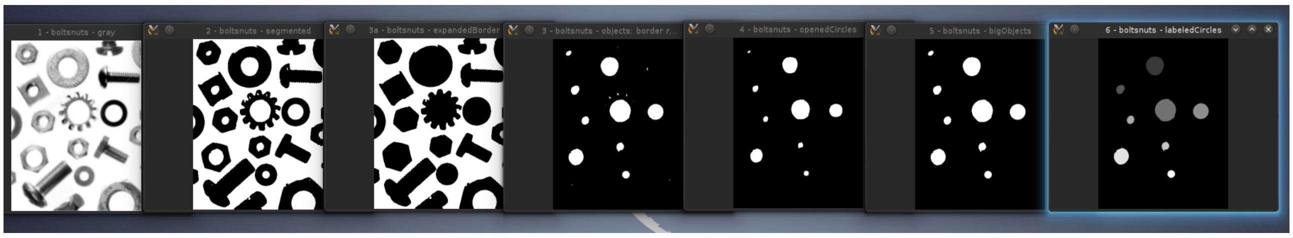
manually made this kernel, later on we made a function of it to return a Gaussian kernel with a given width and sigma. To illustrate we left the manually programmed kernel, since we didn't want, and didn't knew, to perfection it.
After the first function we show the histogram of the Gaussian filtered gray image, which illustrates the big amount of uniform background color.
This could easily be used to segment the background to leave us with the background only as a result.
We then perform an erosion on the background image, which can been seen as a closing operation on the holes of the cheese. As an artifact the background shrinks with half of the kernel size used during the erode function. This results in a border around the image, we'll be using that later on.
After the erosion we invert the image which returns all the holes. These are then be labeled.
The image has a border surrounding the image which is labeled as well. The objects located in the vicinity of the border which touch the border do have the same label because they are one object.
We use the label of the border to delete the holes connected to the border of the image.

Balls



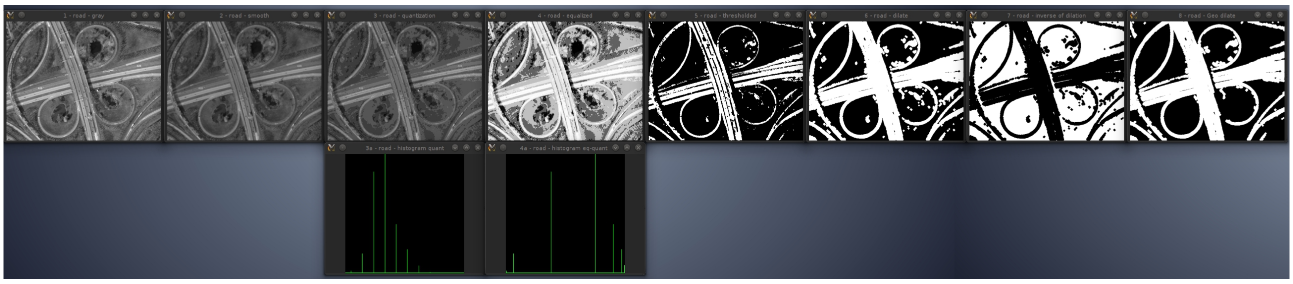
The *balls.png* is first transformed to gray scale.
We then calculate the derivatives in four directions, from left to right and vice versa, from top to bottom and vice versa. The four results are then added to eachother to get a single image. This resulting image has a lot of noise caused by the grass in the background.
The derivative image is then filtered with a Gaussian filter of size 15 and sigma 3. The result is also multiplied by 255 to enhance the difference.
The enhanced image is inverted to do some further processing.
After applying 15 erode operations with the default kernel, 3x3 matrix with a '+' sign, the image is thresholded. We didn't do that before, so actually there still were gray values left.
The image is dilated again to give the balls left over a bit more of their original size. This is done with the default kernel element of a 3x3 matrix with a '+' sign.
The last function used is the labeling function.

Boltsnuts


1 - boltsnuts - gray
2 - boltsnuts - segmented
3a - boltsnuts - expandedBorder
3 - boltsnuts - objects: border r...
4 - boltsnuts - openedCircles
5 - boltsnuts - bigObjects
6 - boltsnuts - labeledCircles

The bolts and nuts images are quite clean, as most of them come from websites selling them. Therefore, they lend themselves perfectly for an approach based on removing the background starting at the border.
The image is first thresholded to obtain the objects. After that, the background, outside of the nuts' inner parts, is removed using a geodesic dilation starting at the border. This 'expanded border' image is subtracted from the thresholded image obtained earlier, which leaves us with all white objects that were not connected to the border before. The smaller objects are eroded away after which a geodesic dilation is performed on the remaining markers (opening by reconstruction). After that, the only step now is to check the remaining objects for the correct compactness and label them.

Road



As with all the other pictures we first convert it to a gray image.
We then apply a small Gaussian filter with size 3 and sigma 1, we do this in order to make sure that all the very small details are smoothed away.
We quantize the smoothed image to an 8 different gray level image, to illustrate the different levels we calculate and display the histogram of the quantized image.
To process it further we apply a histogram stretch, by doing this the 8 gray levels are more widely spread and easier to threshold. For illustration we again calculate the histogram.
The image can now be thresholded, its clearly shown that there are some open spots in the image. To get rid of them we dilate the image three times with a default kernel.
We then calculate the inverse of the result to use it in a geodesic dilation. This operation deletes all the tiny single spots in the image. The result is a somewhat wobbly image of the highway.