CENG 421/ELEC 536 Assignment 2

Posted: February 15 2011. Due: March 4 2011 4:00pm	Aras Balali Moghaddam

This assignment consists of several steps. The main goal is to familiarize yourself with thresholding algorithms and morphological operations discussed in class and in Chapter 6 of your textbook.

1. Foreground Segmentation:

Note: For this section, you will need the files "stuff.pgm" and "stufmin1.pgm".
1a. Write a program which takes as input two grayscale images: The first of a scene containing a foreground object, the second of the same scene with the foreground object removed. Your program must isolate the missing object using an image differencing or background subtraction method. The output of the program will be a binary image in which background pixels are set to 0, and foreground pixels are set to 1.

I perform this to separate the eliminated object from the rest of the scene:

$$result = (img1 - img2) + (img2 - img1);$$

Here is the result image before thresholding. There is still some noise.

Here you can see the image after thresholding. The noise is caused by slight differences between the two images.

1b. Write a program to automatically threshold the foreground object from the background using any method discussed in class (indicate in your write up which method you used). The program will take as input 2 greyscale images as described in 1a. Your program must not use a manually selected threshold. The output of the program will be a binary image in which background pixels are set to 0, and foreground pixels are set to 1. **(Use of the MatLab function "imthresh" is not allowed. You must write your own function.)**

I have implemented an iterative optimal automatic threshold selection algorithm, based on the lecture notes. Here is the commandline output of the asssignment application:

```
Question 1 part b: Automatic threshold selection
Automatic threshold selection ...

            1 - Previous T: [100] Next T: [66]

            2 - Previous T: [66] Next T: [57]
```

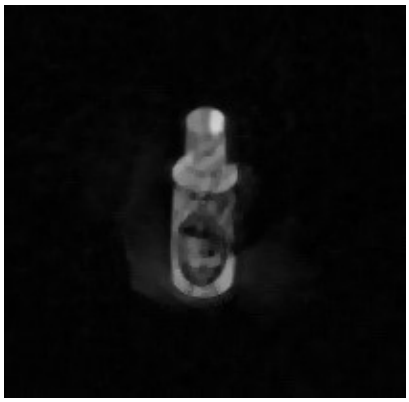
```
3 - Previous T: [57] Next T: [55]

4 - Previous T: [55] Next T: [55]

Selected threshold: [[ 55 ]]
```

Automatic thresholding binary result image.

1c. Write a program to perform noise removal on an input image using a median filter. The program must be able to accept either grayscale or binary images as input, and the output image will be of the same format as the input image**.** Select the image from 1a or 1b that gave you the best result, and use your program to perform median filtering on that image. **(Use of the MatLab functions "imfilter" and "medfilt2" are not allowed. You must write your own function to perform the median filtering of the image.)**

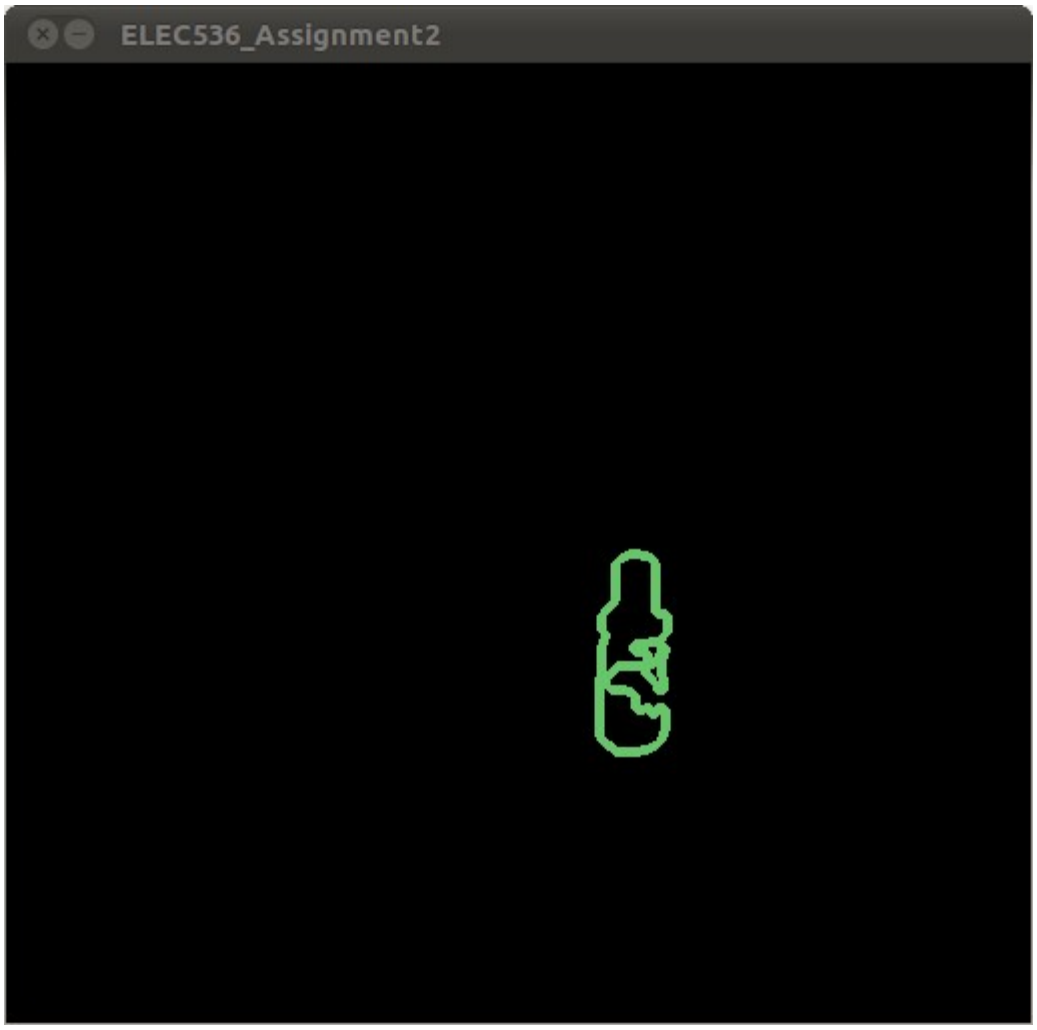


For this part I have implemented a 2D median filtering to remove noise it uses a square window which has edges of size "size" Precondition: size > 1 and it is an odd number

2. Connected Components:

2a. Write a program to perform a connected component labeling of the binary image produced in 1c. The output of your labeling should follow the same format as the MatLab bwlabel() function (see MatLab help files). You may define your connected components using either 4-connected or 8-connected regions, just make sure to state which you use in your written report. **(Use of the MatLab function "bwlabel" is not allowed. You must write your own function for connected component labeling.)**

I have not finished implementing this algorithm yet. So I used OpenCV equivalent of bwlabel which are findContour() and drawContour() functions to implement this part and move forward.


ELEC536_Assignment2

The result of connected component. The grayscale image is first filtered by a median filter of size 5. It is then processed using findContour() method to detect the isolated regions, in other words, perform connected component labeling. Next, the drawContour() method is used to draw and color each of these blobs, or distinguishable objects using a randomly assigned colour.

2b. Identify and display (using colour) the connected component corresponding to the object of interest. Any connected components not belonging to the object of interest (ie: noise) should remain white. Use this connected component as a mask to display the original grayscale object. Your output image should display the object in grayscale as determined by the mask, while the rest of the image remains black (ie: Everything but the object should be removed from the image).

The median filter does an excellent job of removing the salt and pepper noise that was originally present in the input images.

3. Measurement:

Using the connected component object labeling computed in 2a and the MatLab regionprops() function, write programs to do the following:

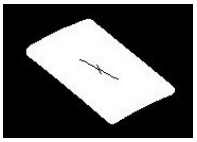
3a. Calculate the area of the object of interest (in pixels).

3b. Calculate the centroid of the object of interest. In addition, display the object with the centroid indicated by superimposing 'cross hairs' over the image.

3c. Calculate the orientation (in degrees) of the object of interest from the x-axis. Display the major axis by superimposing a line over the object image.

Note: For 3.b and 3.c, it is enough to superimpose the cross hairs or line on the binary connected component image of the object. You do not need to display the grayscale image of the object.

Example:



In this image, the cross hair position indicates the centroid location, while the long edge of the cross hair indicates the major axis of the object.

4. General Thresholding:

Using the image "things.pgm", test the algorithms that you designed in the previous sections on it.

Since you have only one image to work with, you cannot test the image differencing algorithm from 1a. You also cannot use a comparison of two images to determine an automatic threshold as in 1b. Instead, modify your 1b code to accept a single image as input, and use automatic thresholding directly on that image to look for bright objects.

Perform the connected component labeling from Part 2, and calculate the measurements from part 3 on each of the detected objects.

Discuss how well your code performs at detecting the various objects in the image. One object will be particularly difficult to find, do not worry if you are not able to detect it. Comment on how you might go about detecting it.

Optional:

For a little additional challenge (and a few bonus marks!), try to segment the objects in the images "testimage.pgm" and "testimagemin1.pgm". You may notice that the images are not perfectly aligned. Does this have any effect on your results? Are you able to clean up any noise

using the function you wrote in 1.c? If not, how might you obtain a cleaner result (ie: what other algorithms discussed in class might prove useful)?