Motion Detection

Keyvan Yaghmayi

The goal of this project is to write a software that detects moving objects. The idea, which is used in security cameras, is basically the process of comparing sequential images and determining whether the differences between them represent motion.

Example 1: The top two images are our inputs. The software detects the person. The black-and-white image is the raw output of the program that is denoised and visualized.

We do this project in three steps:

1. *Extracting Foreground:* We write a program that extracts foreground of an image.
2. *Denoising:* The program would probably extract and visualize some small pieces that we are not interested in, therefore, we should denoise our result from previous step.
3. *Motion Detection:* We write a program that compares two consecutive images and highlights their differences.

1 Extracting Foreground:

Consider an image that contains some dark objects on bright background (or vice versa). We separate the image into foreground and background and visualize the connected pieces of foreground. Our program has two functions:

- *Dual Threshold Function* which takes a gray-scale image and two threshold values. It returns a binary image, *i.e.* a black and white image, which lies in between of those thresholds.
- We also have *Flood Fill Function* base on the well-known *Flood Fill Algorithm* which assigns a label to a connected component. It takes three parameters: a binary image, a seed point which is white, and a label. The algorithm looks for all pixels connected to the seed point by a path of white pixels and labels them. Importing *LinkedList* class from Java into Matlab, made our function short and efficient.

A few examples:

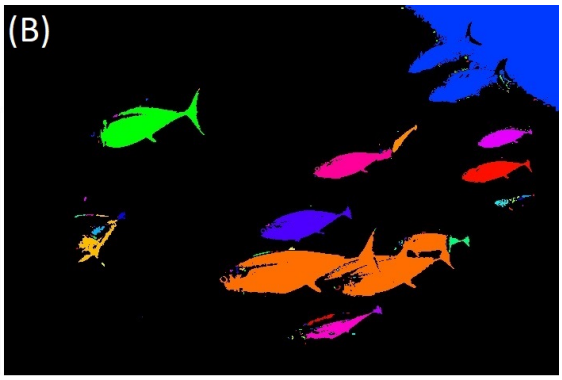
Example 2: (left) The image of brain in black background. (middle) The lighter foreground is extracted. Using threshold values 0 and 85, there are 19 connected components in total. (right) The interesting components are the skull, visualized in orange, and the brain tissue visualized in green. There is also a small connected component, visualized in blue, on the right side between the skull and the brain tissue.



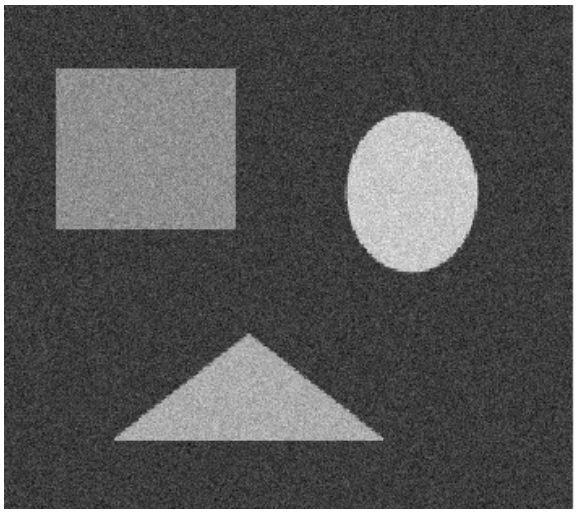
Example 3:

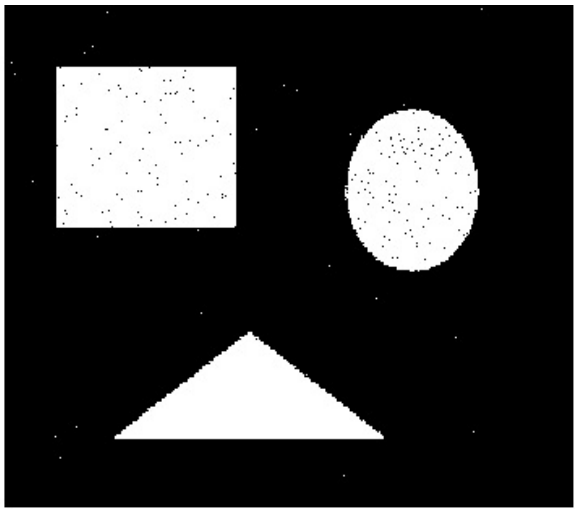
In this image the grayness level of the background is changing from 104 at the top right to 14 at the bottom left. This will challenge us in choosing proper thresholds. We used 75 and the results are images (A) and (B).

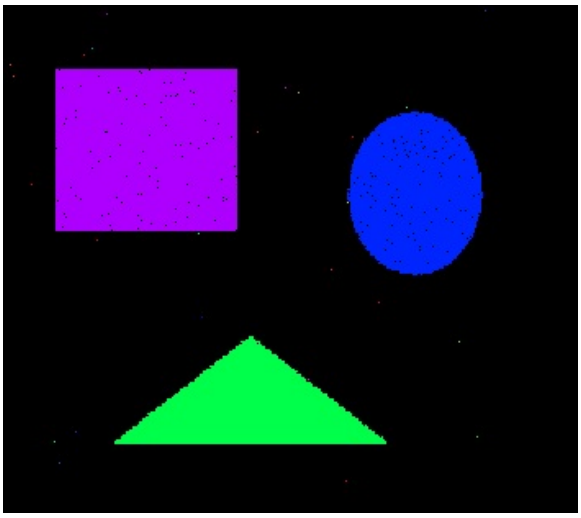
Note that with a simple modification in our code, for instance by using the command *graythresh()*, we can let the software choose the threshold levels automatically. In this example *graythresh()* returns 82 and the results are images (C) and (D).



Example 4: This time our image has some noise.



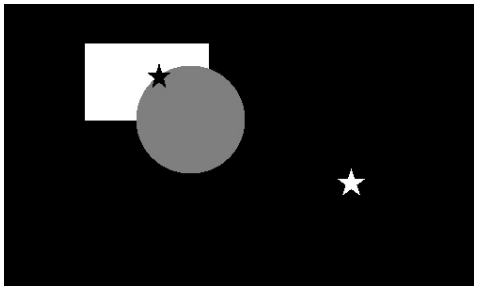




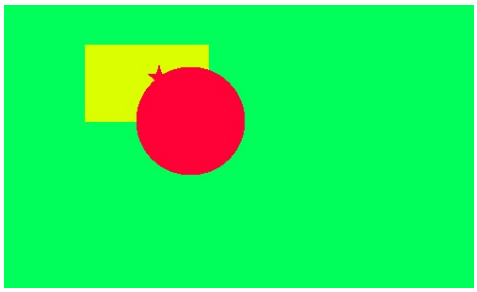
Applying our software with threshold values of 110 and 230 does a decent job of extracting all three geometric shapes. Because of the noise, there are some dots in our binary image and its visualization.

2 Topological Denoising

Example 5: Consider a black and white image (or black and gray and white image) with some noises. For instance, in the following image the stars are noise and we like to remove them.



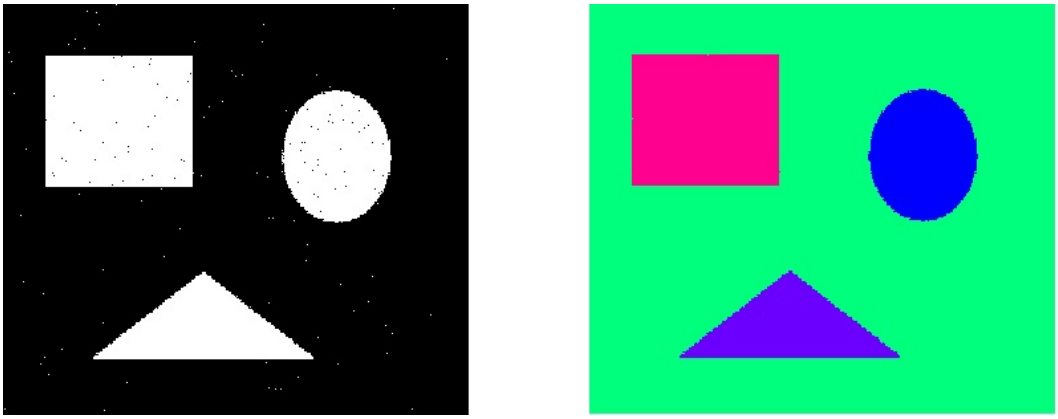
We want to visualize connected components and get rid of noises such that each noise get the same color of the adjacent component that has the largest boundary with the noise. In image above, the white star is completely located inside a black component and we get rid of it by coloring it with the same green color of its adjacent component. The black star has two neighbors: the white rectangle and the gray circle. We remove this noise by assign it the same color of the circle because it has larger boundary with the circle.



The denoising program is based on the extracting foreground program and the flood fill routine. We modified flood fill algorithm to keep track of the size (number of pixels) of connected components. If this size is smaller than a certain number, then the component will be considered as a noise.

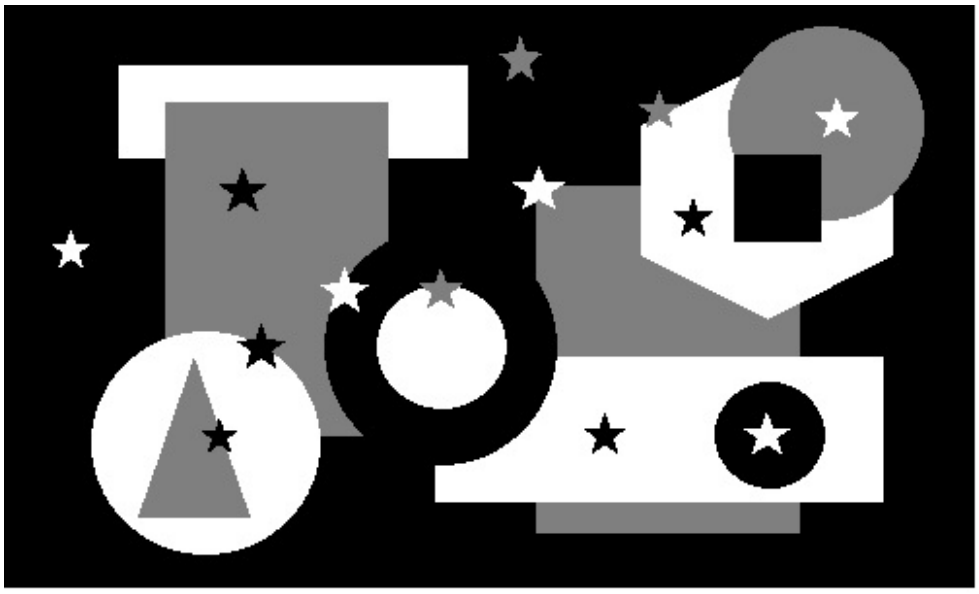
First, the program visualizes all connected components including the noises. Then, for noise components, it looks at the boundary pixels and their colors. After finding the dominant color, it calls the flood fill function to fill up the noise component with this color.

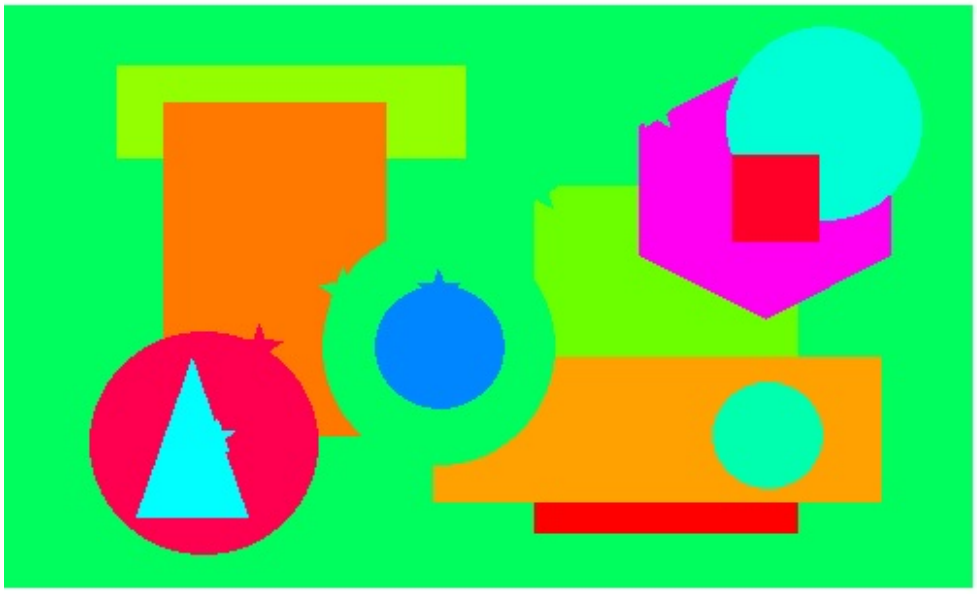
Example 6: Denoising our images from *Example 4*. Left side image has 29 connected components which all are noise except 4 of them: the rectangle, the circle, the triangle, and the background.



Example 7: On the left we have image (C) from *Example 3*. We applied denoisation by re-coloring each component with 300 pixels or less with the same color of its dominant neighbor.

Example 8: Another example that stars are noise.





3 Motion Detection

We write a software that automatically detects moving objects. The idea, which is used in security cameras, is basically the process of comparing sequential images and determining whether the differences between them represent motion. If there are significant differences between two consecutive images, the cameras conclude that there has been motion within the camera view.

In the following we have two pictures, the program analyzes them and highlights the difference in white color. Then we implement denoisation from previous section to refine the result.

Example 9: The moving parts are the dog and tall grasses on the right side.




Example 10 (A) Photo from a parking lot. *(B)* Another photo after one second. *(C)* Detecting the motion in black-and-white with high sensitivity. *(D)* Denoisation and colorization of (C). *(E)* Detecting motion with low sensitivity. Comparing to (C), there are less noise and the cars and the person have better visualization. *(F)* Denoisation and colorization of (E); In (D) we have better result.