Technical University of Cluj-Napoca

Image Processing

Laboratory – Final Project

Rubik Cube Configuration Detection

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Laboratory Assistant:

Florin Oniga

Student:

Enoiu Diana-Cristina

Contents

1. Assignment Objective

The goal of this assignment is to be able to detect a Rubik cube configuration by taking 3 images of a Rubik cube covering two faces. The algorithm might be useful for robots made for solving the Rubik cube in order to detect the configuration by using multiple cameras, instead of manually inputting each color for each corresponding square.

2. Short description of the algorithm

We will attempt to implement this algorithm first by identifying the color for each square. We need to be careful here for two main problems:

1. The edges are not very clear, therefore when if we have two adjacent squares that have the same color, we might extract them as being only one square.
2. The orange and red are very similar, especially in shadow. It might be hard to differentiate between them if the light is not very strong.

After this step we can find the center of mass for each extracted square, giving us information about the position of that square in the image. Knowing the center of mass and the color for each square, we can extract an order of colors form left to right and form top down for each face, by comparing the positions for the centers of mass.

Once we have the colors ordered for each face, we still need to find out where the face belongs in the final configuration and whether or not it has to be rotated.

3. Steps

3.1 Edge enhancement

We will be using a particular type of Rubik cube that does not have the edges very clearly defined. This step might not be necessary in case the Rubik cube would have had dark edges for each square.

In this step, I will use a Canny Edge Detection algorithm that will attempt to identify the edges that the Rubik cube has. After that I can draw the edges of the cube with black. This is helping us to avoid identifying adjacent squares that have the same color as only one square, when we will use the color detection algorithm.

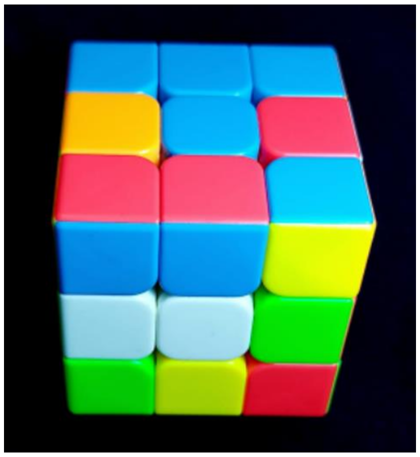


Figure 1. Initial cube

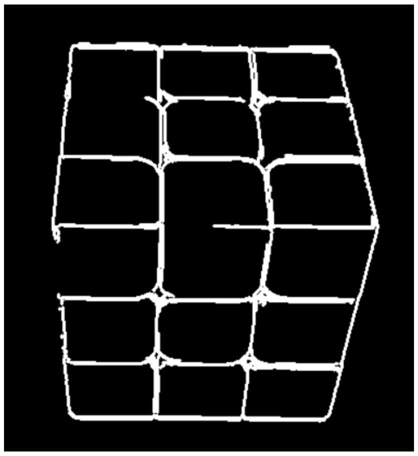


Figure 2. Canny edge detection result

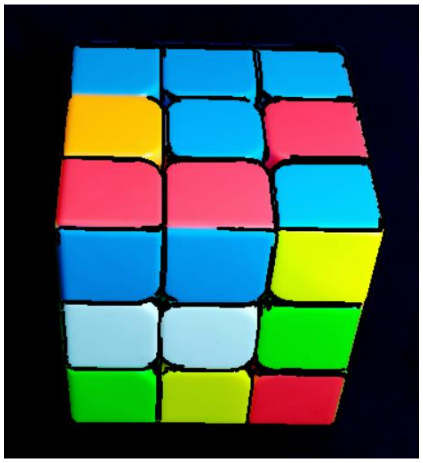


Figure 3. Results after applying canny edge detection result to original image

3.2 Transform to YUV and identify colors

At this point we can convert from RGB to YUV and we can extract the colors from the image by identifying some specific intervals that the YUV values should have.

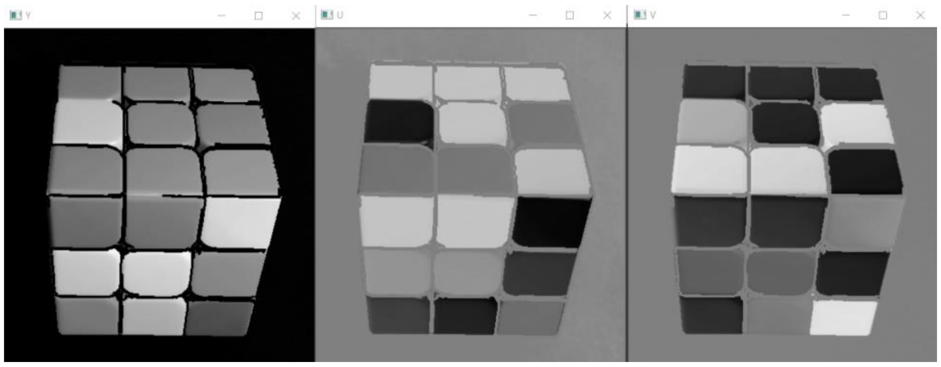


Figure 4. YUV conversion result

Some corresponding YUV values intervals are given in this article: https://www.ia.pw.edu.pl/~wkasprza/PAP/ICCVG06.pdf. The interval values present in the project have been modified a bit to detect a wider spectrum of colors. Using those intervals on a image that has a spectrum of colors will give us the following results:

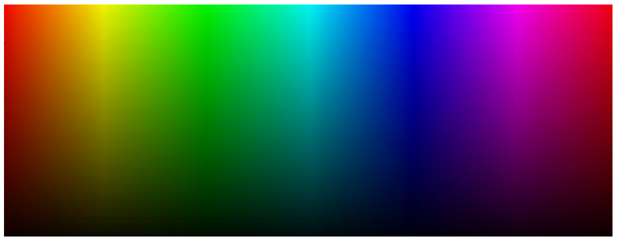


Figure 5. Initial image

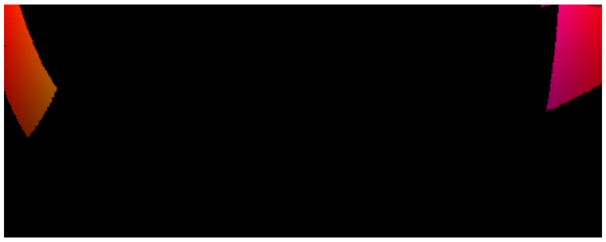


Figure 6. Identified as red

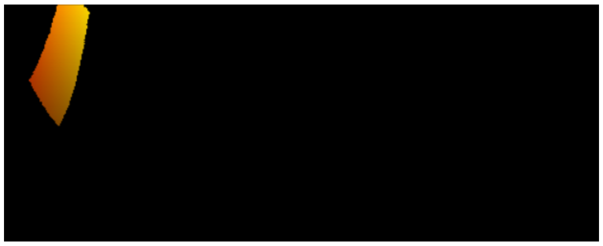


Figure 7. Identified as orange

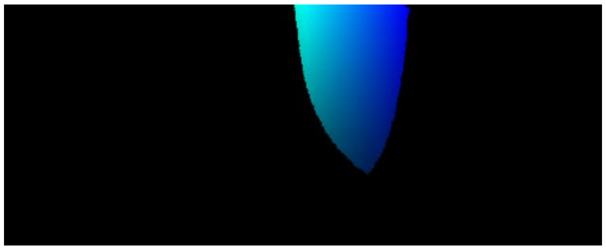


Figure 8. Identified as blue

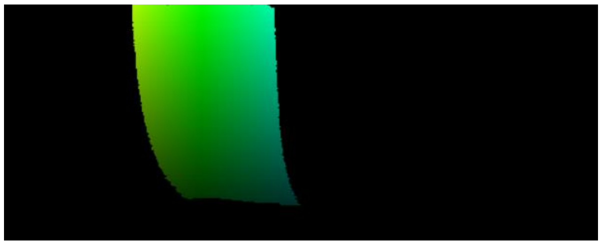


Figure 9. Identified as green

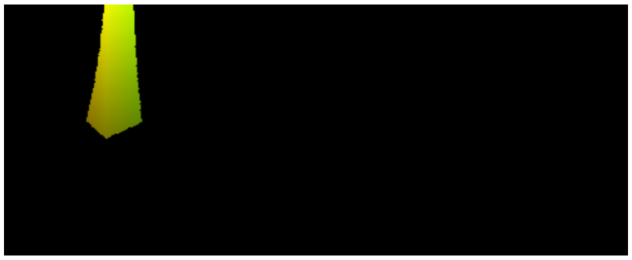


Figure 10. Identified as yellow

Of course, we haven't managed to identify all the possible color values for each color, therefore the algorithm might fail due to that fact. Also, the difference between what is considered to be orange or red might depend a lot based on the illumination conditions. For this algorithm to work properly we expect the cube to be well illuminated.

3.3 Identification of each square based on its color

After applying YUV conversion we can now extract the colors from the cube in separate images. The extraction is not going to be perfect and we might still identify some dots or lines that belong to a specific color, but it does not connect to a square. The goal is to obtain 9 different components each having a corresponding color.

One way to solve this is by applying multiple times erosion on an image until we remain only with the big components corresponding to our squares, and everything else that is identified will be deleted. Another alternative would be to delete everything that has an area smaller than some threshold value, but we haven't applied this solution.

One advantage to eroding the image is that it will allow us to further separate the squares having the same color. As example for the blue in the image we still obtained two connected squares. After erosion they become two separate components. Applying erosion needs the image to be binary.

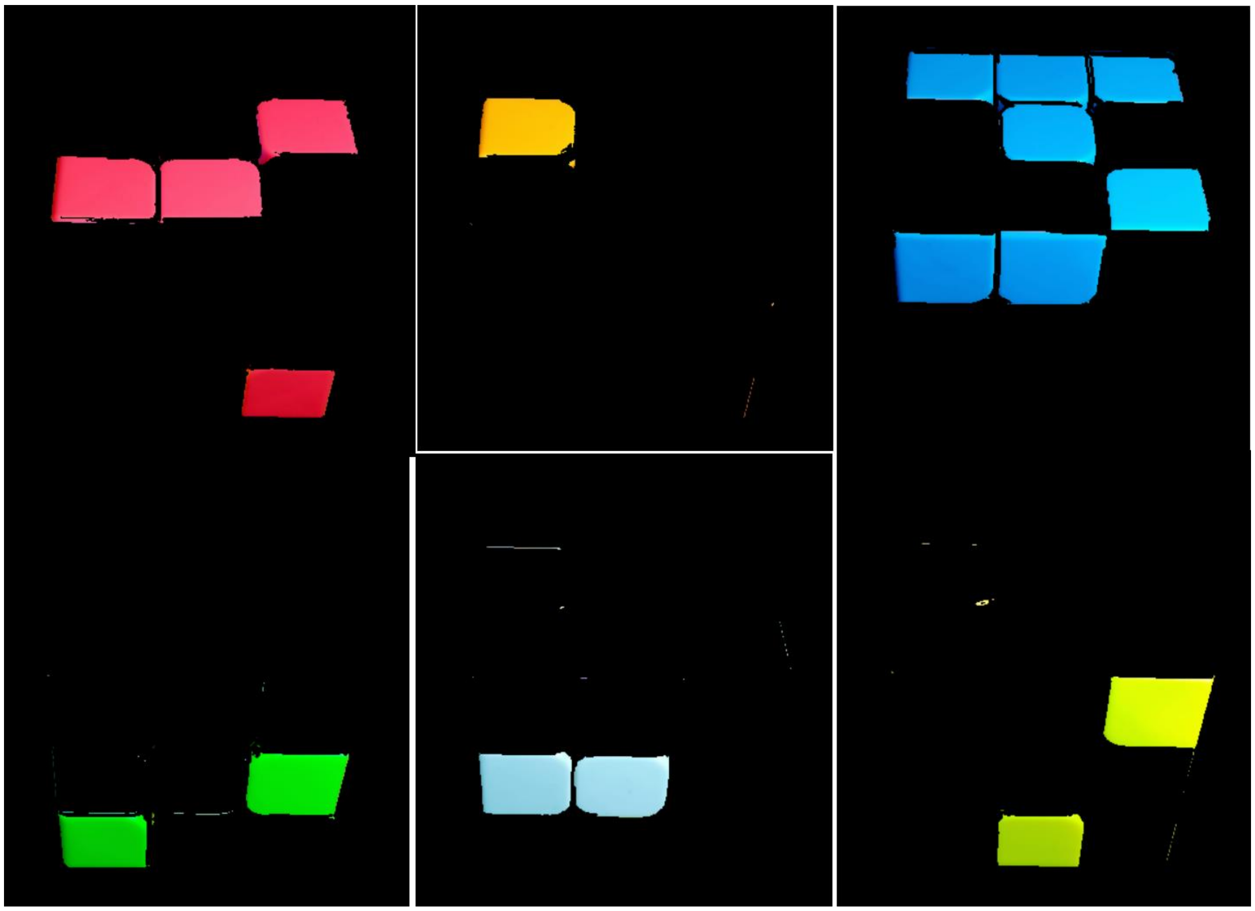


Figure 11. Color extraction

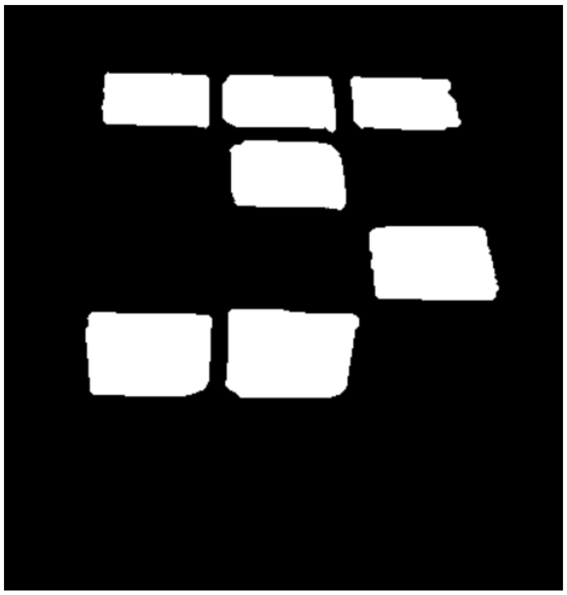


Figure 12. Blue squares identified after erosion

3.4 Finding the center of masses and sorting

Having the squares as separated components allows us to apply a breadth first search algorithm on the image to identify each separated component and label it. The center of masses will be computed taking into consideration this labeled components.

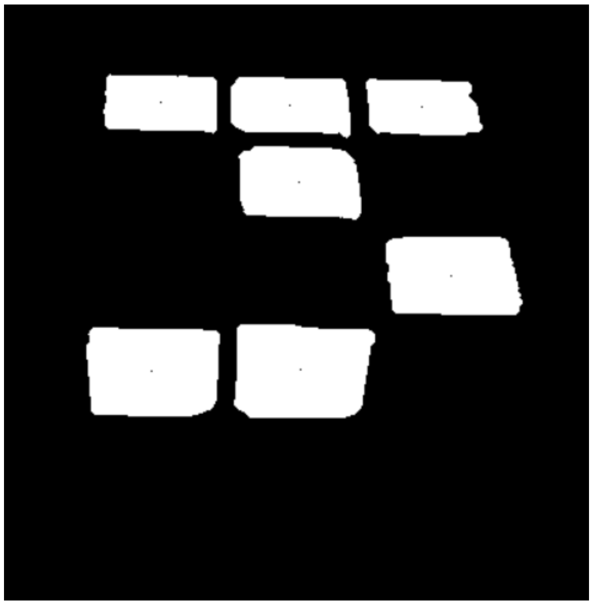


Figure 13. Center of masses found

Now that we know where the middle point (center of mass) lays for each square and its corresponding color, we can order the obtained squares by the position of the center of masses in the image. We know that the first 9 squares belong to the upper face and the next 9 to the bottom face.

3.5 Obtaining the final configuration

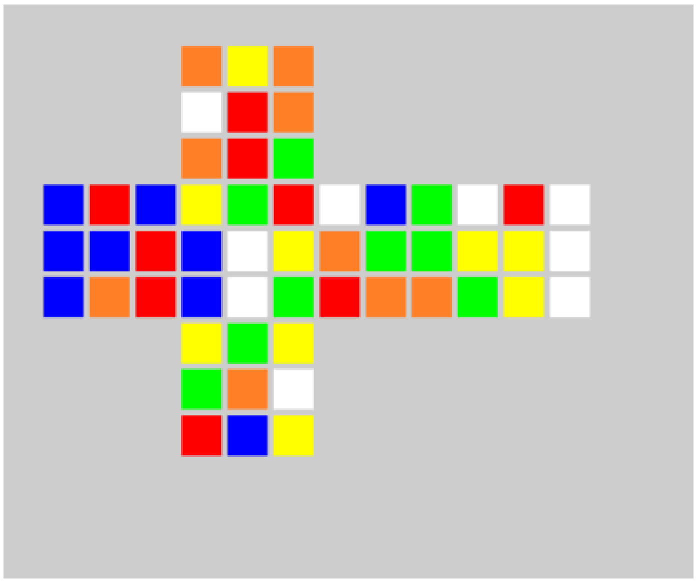
After the algorithm is done for one image we will repeat the same process on the remaining two images.

At this point we have identified the faces, but we still don't know where they belong in the final configuration. By taking the color of the middle square of the face, we can identify the position of color of the face (the color obtained when the Rubik cube is solved) and this will give us the information that we needed to know where to place those faces.

Another thing to consider is that we still need to rotate the faces in order for the edges to match one another. Considering that the images are taken from the same place every time, we can hardcode how many times we need to rotate the faces in order for the edges to mach.

4. Result

After the algorithm is done we print the final configuration as a flat view of the cube.



5. Conclusion

The following algorithm succeeds in identifying the Rubik cube configuration, but it has some very clear constraints, as taking the pictures from the same angles. Wanting to modify the angles from each we take the pictures leads to modifications in the code for this algorithm to still work.

Another problem comes from poorly identifying the edges of the squares of the Rubik cube, but this problem might be avoided when another type of Rubik cube is used (one that has dark edges). The background is also important to be black or a darker color that does not get identified as some color from the one above.

The most important drawback comes from misidentifying colors or not identifying them at all under certain light conditions, this might be avoided if the Rubik cube is well illuminated.