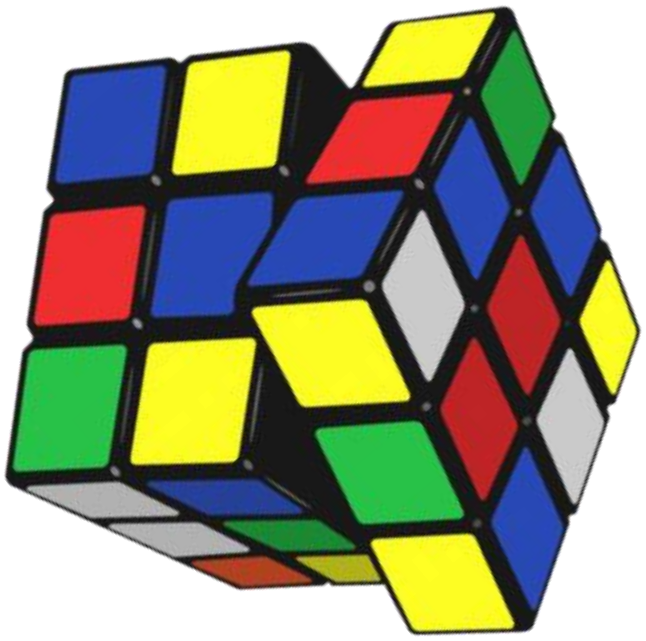
Rubik's cube configuration detection



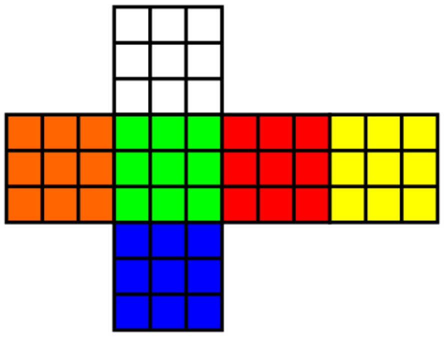
Andrei Gog, group 30433

prof: Florin Oniga

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

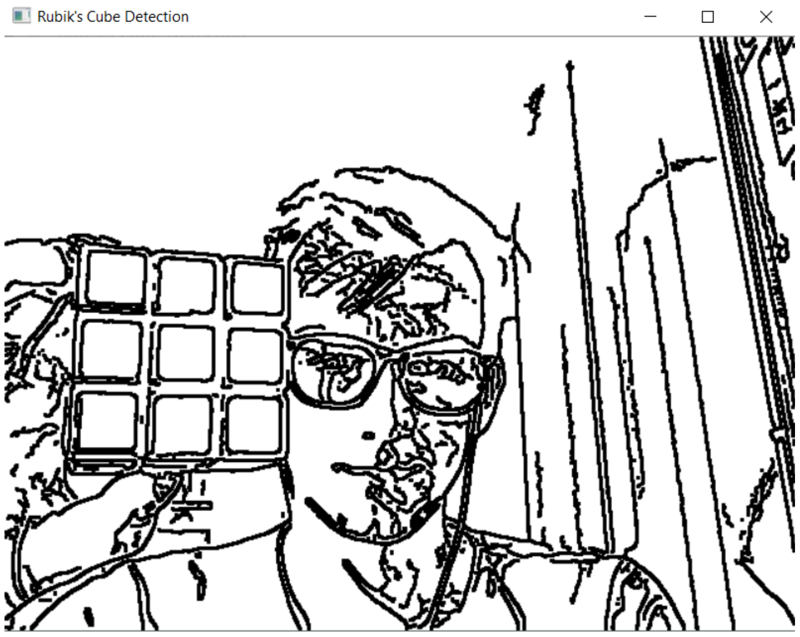
1. Problem statement

The Rubik's cube is a 3-D combination puzzle invented in 1974 by Erno Rubik, a Hungarian professor. The motivation for developing the cube was that he wanted to create a model that explains the three dimensional geometry to his students. The Rubik's cube was conceived as a teaching tool, where Erno Rubik could demonstrate the structural design to his students. The purpose of this project is to reconstruct the 3D structure of such a cube, having a video perspective of the cube and displaying the cube face by face, until it is completely scanned. The expected output is the cube state displayed like this:

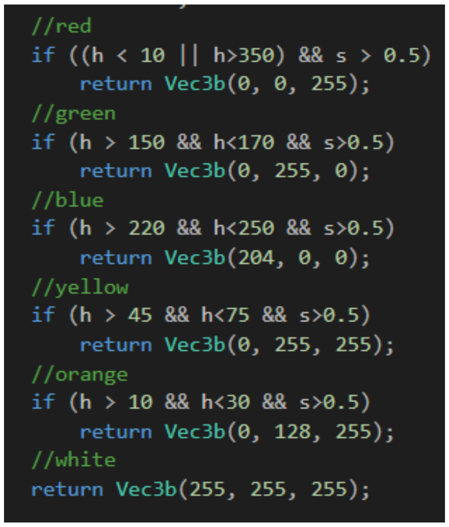


2. Explained algorithm

To solve the given problem, OpenCV and C++ offer some really useful built in structures and algorithms, specifically designed for image processing like this. OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. First of all, a video window will start, more precisely the laptop webcam, or any other recording device. OpenCV has a feature that allows each frame of that video capture to be processed as a simple image. The aimed goal, is to detect squares on the video frame, drop the non Rubik's cube squares and then scan the color from the scanned squares. To detect the squares, an edge detection algorithm is applied on the initial frame. The preferred algorithm was Canny Edge Detection. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Here is an example on how the output of a Canny Edge Detection would like, after the algorithm was applied on the video frame containing a face of the cube.



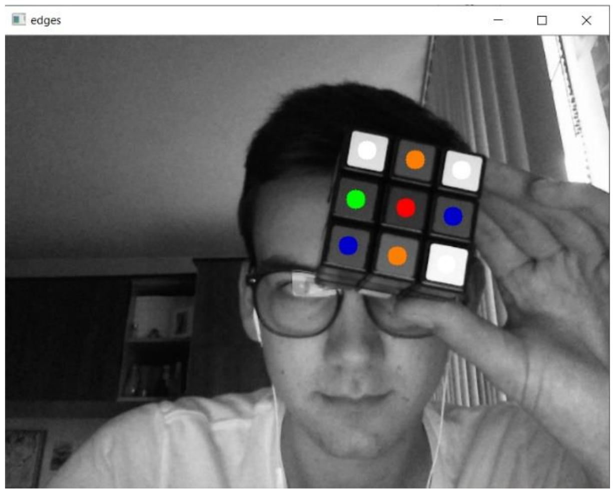
The next step is to make sure, that each square of the cube is a connected component, in order to be eligible for labeling. This need is satisfied by dilating the image once or twice. After the first part of the processing is completed, the refined image, as well as the original image are then passed to a method that will do most of the job. The edge-only image is a binary image, having the edges as black pixels and the other pixels white. It is traversed, until the following condition is satisfied: the current pixel is white and its northern and western neighbors are edge pixels. This means that the current pixel might be inside a square and a breadth first search labeling algorithm is started. When a possible square is found there are also some information about that possible square initialized. The features of interest are the RGB values from the original image, the center of weight, the area and the perimeter. After the labeling is done, the squares that do not have the aspect ration close to the float value 1.0 are dropped. The aspect ratio of the shape is computed using the area and the perimeter.



```
//red
if ((h < 10 || h>350) && s > 0.5)
    return Vec3b(0, 0, 255);
//green
if (h > 150 && h<170 && s>0.5)
    return Vec3b(0, 255, 0);
//blue
if (h > 220 && h<250 && s>0.5)
    return Vec3b(204, 0, 0);
//yellow
if (h > 45 && h<75 && s>0.5)
    return Vec3b(0, 255, 255);
//orange
if (h > 10 && h<30 && s>0.5)
    return Vec3b(0, 128, 255);
//white
return Vec3b(255, 255, 255);
```

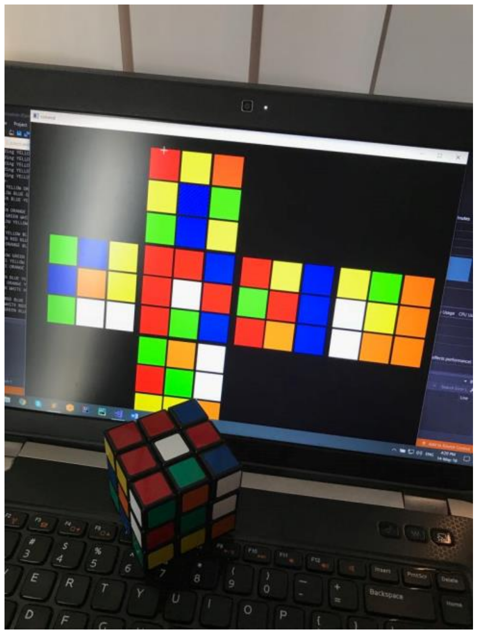
The steps described above are continuously executed for each frame of the video. Another interesting part of the project is detecting the actual Rubik's cube face, because sometimes in the background there will appear another square objects that might twist the expected result. To avoid this, some extra checks are performed on the scanned objects. The algorithm will know it has found a face, only when it will detect 9 squares in an area, having the centers of weight at a specific repeated distance between the neighbors.

For the remaining 9 squares, the algorithm will process the information stored about the average RGB value of those values. The RGB value is transformed to HSV (hue-saturation-value) color space. Comparing those values with the standard red, green, blue, yellow, white values, each square will receive the corresponding flat color. One of the last step in reading the face is sorting the read squares to map them to a Rubik's cube face template. In order to do this, the vector of squares is first sorted based on the X coordinate of each center of weight. After that, the first three squares, the second three squares and the last three squares of the sorted vector are sorted base on the Y coordinate value. Having the vector sorted this way, we obtained the logical order of the squares. Also, the color from the middle of the vector, respectively from the middle of the cube's face indicates the



scanned face. To let the user know when the face is scanned, there is a log in console and also some flat colors circles will be displayed on the video frame having their center in the square center. To store the cube state, there is a C++ class called cube having the all the faces as fields and the specific methods.

The last step of the implemented project is closing the video frame and checking the result. Here's an example of a trial:



3. User's guide

The user can start with any of the cube faces, because the algorithm does not have a specific order, and can also show the faces in whichever order. The only things that has to be taken into consideration is that each face must be displayed with respect to the already shown faces in terms of face orientation. The user can wait and scan all the face or just some of them, in both cases, when he wants to see the output, he has to press the Esc key.

4. Future improvements

The are still some issues with the red and orange squares because they have related HSV values and sometimes there are small errors, though the average success rate of the cube scanner is 50 squares out of 54.

Another step in developing this software would be after reading the cube state by displaying the set of moves needed to solve the cube in that particular state.

5. Bibliography

Stanford Automated Rubik's Cube Recognition

UTCN Image Processing Basic Notions