Rubik's cube Recognition, Identification and Solving using Computer Vision

Saichand Bandarupalli, Rahul Dev Appapogu

Colorado School of Mines

Introduction:

The world’s best selling puzzle, Rubik’s cube was invented by Hungarian sculptor and professor of Architecture, Erno Rubik in 1974. Originally called ‘The Magic Cube’, Rubik’s cube became immensely popular in the 80’s, and still remains equally popular till date. It is so popular, that it is said at least every one in seven living person has tried to solve the cube. Although it is a toy that is sold in shops, it has been a keen interest of subject to many mathematicians and engineers being a very complex combination puzzle. So complex, that it took 36 years after its invention to find out the exact number of possible configurations a cube has. In July 2010, it was revealed that the total number of possible configurations was over 43 Quintillion where, 1 Quintillion = 10^{18}. If a cube was placed for each of its possible configurations in a straight line, the total distance would be approximately 261 light years. So not until 2010, not even computers could not compute the least number of moves to solve the Rubik’s cube, in any given state. However, it was found out that the upper bound for the least number of moves required to solve the cube in any given state is 20, also known as *God’s number*. The algorithm used to solve is called *God’s Algorithm*.

Solving Rubik’s cube is not just solving a puzzle, but also a well defined mathematical problem. It took 1 month for its creator Erno Rubik to learn how to solve the cube. Although our team knows how to solve the cube manually, there are algorithms that can solve the cube at a much faster rate. It is not possible by a human alone to apply the same algorithms as it requires enormous effort to compute the possible moves applicable to solve in a given configuration. But using a computer that has enough configuration to compute the possibility of moves, Rubik’s cube can be solved a lot faster. (*‘Faster’* here is represented in terms of number of moves to solve, not amount of time taken) Considering this problem as a challenge, our team stepped forward to design an algorithm that could recognise, identify and solve the Rubik’s cube at the fastest pace possible. Our goals for this project could be broadly mentioned as:

- Recognise each face of the Rubik’s cube in any environment
- Exactly identify all the colors in a given face
- Construct the 3D model of the Rubik’s cube

- Enlist the moves required to solve the cube as an output text file.
- Incorporate the Computer Vision part to a GUI designed to solve the Rubik's cube

With these goals and knowledge of computer vision, our team started to design an algorithm for recognition and identification of the cube. Our team exclusively used MATLAB's Image Processing and Computer Vision toolboxes for this project. The scope of the study done in this project, algorithms developed and software designed on MATLAB, are limited to the standard 3x3x3 Rubik's cube only. Also, we used the official Rubik's© Cube for all testing purposes.

Brief note on Cube Theory:

The Rubik's cube has 6 solid color faces, White, Yellow; Red, Orange; Green, Blue with each color pair on opposite sides respectively. It is consisted of 20 small cubes, called *'cubies'* attached to the cube centres. The 6 cube centres are immovable, that is they never change their position in space. Also, being only one face full of solid color, its orientation in any configuration is the same. Cubies, can be further classified as 'corner cubies', cubies present in the corner and 'edge cubies' present in between two corners. Each corner cubie has 3 faces, with 3 different colors thus having 3 possible orientations and edge cubie has 2 faces with 2 different colors thus having 2 possible orientations. In total, there are 8 corner cubies and 12 edge cubies. Number of possible orientations for total corner and edge cubies are 3^8 and 2^{12} respectively. Number of ways to permute the corner and edge cubies are 8! and 12! respectively. Therefore, total number of number of configurations is: $\mathbf{3^8 \times 2^{12} \times 8! \times 12! = \sim 5.3 \times 10^{20}}$

Out of these, only half permutations, half edge and one third corner orientations are permitted because of the geometry of the cube. Thus reducing the total possible reachable states are **43 252 003 274 489 856 000** ~ 43 Quintillion

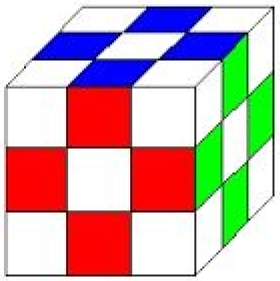
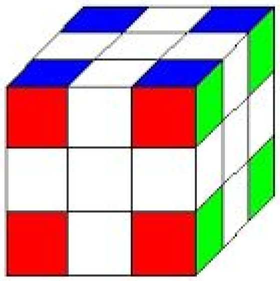

Fig.1. Edge and Corner Cubies

Existing Methodologies:

Our literature review for similar lines of work revealed two copyrighted existing softwares to solve the Rubik's cube using MATLAB. One of them was *Rubik's Cube Simulator and Solver*, by Joren Heit. It consisted of a GUI, which could take cube configuration input from webcam as well as manually entered data. It also had a randomize option, which would give the cube a pseudo-random configuration. There are 3 solution algorithms which could be selected using the GUI. Solve option would solve the cube in any given configuration using the selected algorithm. Most of this work was on exploring various solving methodologies and less of identification using computer vision. Algorithm used for the computer vision part was barely mentioned in the notes. It just suggested that we use blob features and feature extraction for identification of the cube. Another software to solve the Rubik's cube in real time was found in our literature survey, namely *Rubik's cube solver using Arduino and MATLAB*. Clearly, this project was a next level achievement of our current goals for our project. It explored more about Arduino - MATLAB interface, mechanism built to hold and solve the Rubik's cube in real time. Our team would like continue our project to that level.

Major similarity in our goals and the goals of the projects mentioned above was using MATLAB and camera for identification of the cube. We initially thought we could get an advanced algorithm for identification of the Rubik's cube using MATLAB. But after exploring the projects, our team found out that there was no specific mention of the algorithms used for computer vision/cube recognition. The only significant help we found was to use contour features and their properties for recognition and identification of the cube. Moreover, there was no mention about the robustness of the computer vision algorithms. The Rubik's cube solver using Arduino and MATLAB has a fixed mechanism, and thus the cube lies static on a surface. For that application, there would be no need for recognition of the cube, as the cube always lies in the same place. Pixel coordinates of the cube corners would suffice for recognition and one could directly proceed to the color identification part. The Rubik's cube simulator and solver was much closer to our project in this perspective. Due to lack of a clear cut algorithm, our team decided to design our own robust - algorithm for the identification of the cube.

Our Approach:

First objective of our project was to identify the cube in any random environment. We first approached the problem, by fixing the location of the cube in the field of view of the camera. That is, the face of the cube is to be shown to the camera only at a particular location in space. After one face of the cube was placed in the mentioned location, the cube's face would be cropped using the fixed coordinates and the cropped face goes in for color identification.

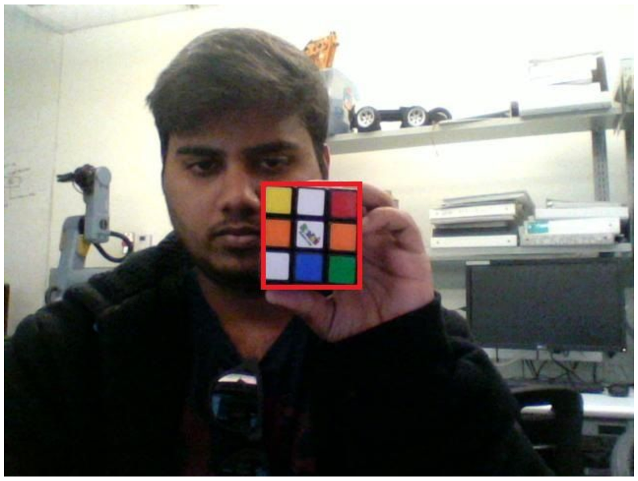

Fig.2. User trying to fit in one face of the cube in the mentioned square (red bordered)

One issue with this method of recognition is that is is not robust. In reality, the cube is not identified in a random background, but its location is merely fixed and the computer 'knows' where the cube is going to be located and which part to crop. This would be a constraint and would make the model less robust.

Our team tried many other approaches to recognise the cube, such as edge detection, finding squares, usage of black lines in between the squares but all those approaches were simply not very effective due to noise, existence of similar patterns/lines/squares in the background. We then tried to exploit the property that the cube had only solid colors. So a thresholded black and white image of one particular frame could extract the 9 squares on one face the cube as white squares, given an optimum threshold. But that would not comply with the lighting conditions as in dim light situations, green and blue would seem dark and appear black, losing 2 colors.

In order to relax constraints and make the recognition model more robust, our team came up with an algorithm that is described below in steps:

Cube-Face Recognition Algorithm:

1. Take R, G, B indexed images of each frame. Convert all 3 images to black and white
2. Extract blob features of each image (Centroid, Area, MajorAxisLength, MinorAxisLength)
3. Assuming that the cube is placed in front of the camera at a certain distance range, a certain threshold range for area is set, and all blobs in that area range are identified for each indexed image
4. Using the fact that for a square, MajorAxisLength = MinorAxisLength all blobs meeting this requirement from the ones that come under the area threshold are categorized as squares. This is done in all 3 indexed images and bounding boxes are drawn on the main RGB image for better visualization.
5. In order to eliminate any unnecessary squares from the background, median distance technique is employed. In this technique, median centroid of all centroids of identified squares is taken and the distance of all square's centroids from the median centroid is computed. Assuming all squares 9 squares on the face of the cube are identified (the median would mostly be the square in the middle) and using the fact that all 9 squares are closely packed, the squares with centroids far away from the median, which do not come in the given distance-threshold range are eliminated. Thus only the closely packed 9 squares are identified therefore, identifying the face
6. After all 9 squares (only) are identified, the face of the image is cropped for color identification. To crop the face, the least and highest x and y coordinates of the centroids of the 9 squares are taken. Least one represents the top-left corner cubie, and highest one represents the bottom-right corner cubie. Given a small offset to both the centroids, a bounding box is drawn using the two points and the image in the bounding box is cropped.

   Thus, we recognise one face of Rubik's cube and crop it for color recognition. This is done for all 6 faces of the cube. The advantage of taking R, G and B indexed images and converting them to black and white, is that we identify all 6 colors overall (using 3 images) as blobs without any loss of any color and without any noise.

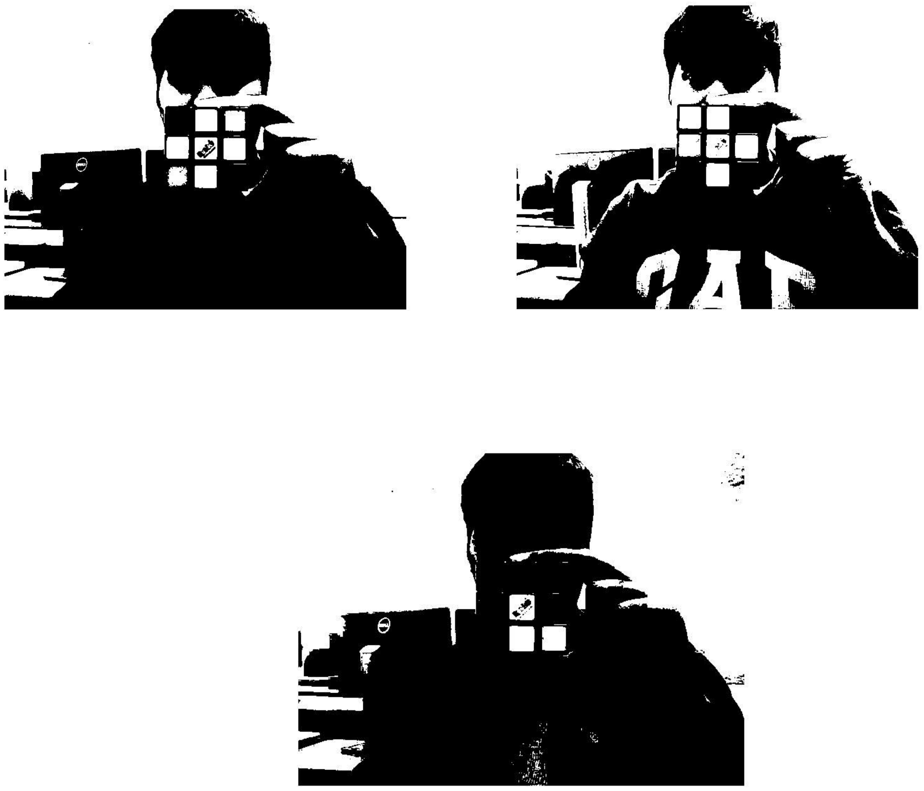

Fig.3. 3 Black and White indexed R, G, B images top left, top right and bottom one respectively. orange, red, yellow and white are identified in red space; yellow, orange, green and white are identified in green space; white and blue are identified in blue space

We can see that each square is identified completely, at least once in 3 image components, without any black patches, comfortably identified as squares.

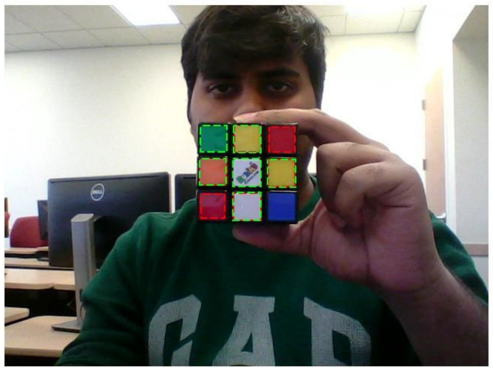

Fig.4. Original RGB image. Bounding boxes are drawn on identified squares.

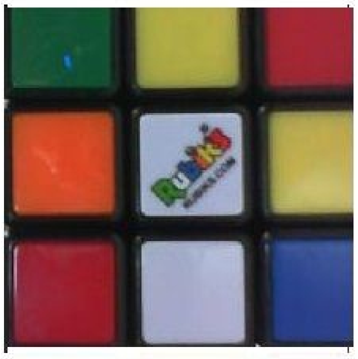

Fig. 5. Identified square - Cropped image

All 6 faces of the cube are recognised individually. Every time only one face is shown to the camera and all 6 faces are shown in a particular order (for 3D reconstruction of the model). The order our team chose to build the model is:

Front Face -> Top Face -> Bottom Face -> Left Face -> Right Face -> Back Face

Each cropped face is used for color identification. This algorithm is implemented on MATLAB, using the pre-defined functions from Image Processing and Computer Vision toolbox. Pseudocode for the algorithm described with predefined functions in MATLAB is mentioned below:

```
RGB_Image=I; I_red=I(:,:,1); I_blue=I(:,:,2); I_green=I(:,:,3);
for i=1:3 {I_respective_color = im2bw(I_color);
stats(i)=regionprops(I_respective_color);
if thresh_1<stats(i).area<thresh_2 => selected_blobs(i);
If selected_blobs(i).(MajorAxis-MinorAxis)<thresh
=>declare_square;
if norm(median(centroid)-other(centroid)>thresh_3 => eliminate;
}
draw_rectangle(min_centroid,max_centroid);
Face_n = imcrop(draw_rectangle);
```

Original MATLAB code developed is mentioned in appendix

Color identification:

The cropped face of the cube that is detected in the random environment using the RGB extraction is the input to the color identification algorithm. The cropped image is converted from RGB to HSV space for reliable color identification. The face of the cube is discretized into 9 areas, each area representing one cubie/one color. Assuming the face of the cube is cropped appropriately, the cropped face is divided into regions by 2 equally spaced horizontal and vertical lines, dividing the cube into 9 regions like a tic tac toe grid.

Each region ideally is a single color, so the median color value of the region should provide the actual color for the entire region. Hence the median 'hue' value is found for the given region and based on the hue, color of the region is identified. 'Hue' value for the solid colors in Rubik's cube are identified and the color regions are classified based on various experiments our team made. Below is a table containing the Hue region and respective identified color.

Color	Hue Min	Hue Max
Orange	0.01	0.08
Yellow	0.1	0.28
Green	0.32	0.49
Blue	0.56	0.68
White	0.61	0.73
Red	0.81	1.00

The 'hue' values in the table are found out by our team by performing various experiments in different conditions. All values mentioned in the table account for both good and bad lighting situations. Thus, this color identification algorithm runs reliably in most lighting situations. One observation in the table is that, Blue and White's hues range are overlapping. In order to eliminate this anomaly, we use saturation component of the region. Saturation below 0.5 is white and rest is identified as blue.

The above mentioned algorithm mentioned is implemented in MATLAB using Image Processing and Computer Vision toolbox's predefined functions. Figures below are test inputs provided and the results obtained in MATLAB are mentioned below

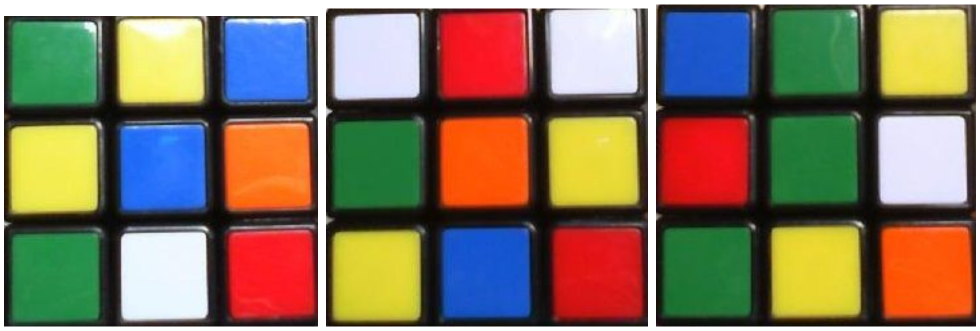

Fig.6. Test input images provided for testing algorithm in MATLAB

Results:

```
g y b            w r w             b g y
y b o            g o y             r g w
g w r            y b r             g y o
```

Our team was very much satisfied with the results our designed algorithms provided and we also made a 3D model reconstruction of the Rubik's cube in MATLAB using `fill3()` function. Result of the 3D reconstructed model with the original cube configuration is shown below for comparison

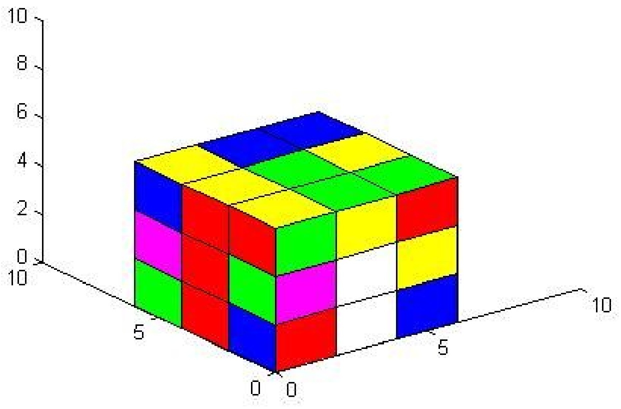

Fig.7 3D reconstructed model (Orange is represented as Magenta)

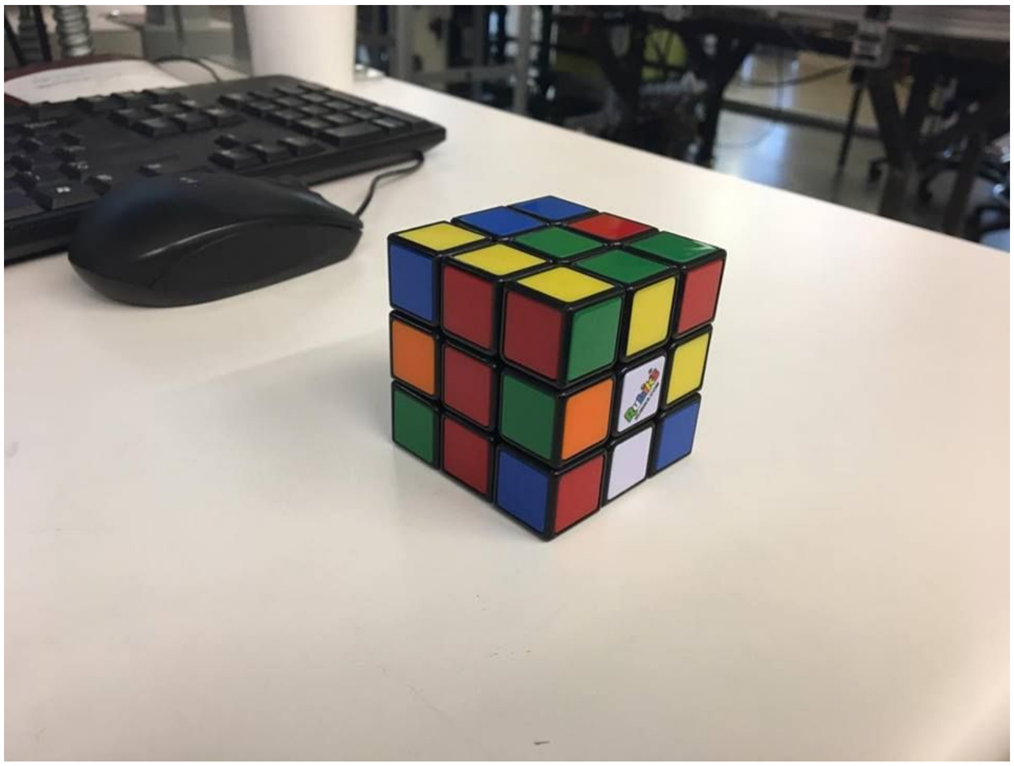

Fig.8. Configuration of Rubik's Cube used for 3D modelling

Solving the Rubik's Cube:

Now that we have constructed our 3D model, our team's next objective is to solve the Rubik's Cube using a computer. Although, solving the cube has nothing much to do with Computer Vision, we would like to present it briefly in our study.

The easiest way of solving a rubik's cube is to go brute force. That is given a possible configuration, draw a tree of all possible moves and the respective reachable configurations. That way, many trees are drawn until one reaches the solved configuration. Using this method, one may even have to face the entire possible configurations of the Rubik's cube and that would simply take forever to solve it. Another method to solve, a 'genuine' one is the 6 step guide mentioned in the official Rubik's cube website. Using the 6-step method, the average number of moves required to solve the cube would be around 120, which still is quite a large number.

Thistlethwaite's 45-Move Algorithm: (Brief Explanation)

Thistlethwaite's 45 move algorithm is recognized as one of the first genuine mathematical attempt to solve the Rubik's cube. This is a computer based algorithm and it's almost impossible for any person to execute it manually. This method requires to prepare a set of pruning-tables that are used to solve the cube and cannot be generated manually, as some have entries like over a million. Even if they were generated, it is impossible for anyone to solve the cube using those tables.

The algorithm is made up out of 4 stages, in which the following steps are solved:

1. Fix all edge-orientations.
2. Fix the corner-orientations and bring the LR-edges in their slice.
3. Bring the corners into their G3-orbit, and move all edges into their slice in an even permutation.
4. Solve the cube by simultaneously fixing both the corner- and edge-permutation.

To assure that each stage stays solved, the number of permitted moves decreases
each time we move to a new stage. For each stage, the permitted moves dene
a group Gi. These are given below:

G0 = L,R,F,B,U,D

G1 = L,R,F,B,U2,D2

G2 = L,R,F2,B2,U2,D2

G3 = L2,R2,F2,B2,U2,D2

G4 = I

Where L, R, F, B, U, D are 90 degree clockwise rotations about Left, Right, Front, Back, Up and Down faces respectively. Any face having 2, say D2 means, rotate D face clockwise by 180 degrees.

A cube is said to be in Gi when its state can be acquired (from a solved initial state) by using moves from Gi only. Since any desired move-sequence can be constructed by using moves from G0, any given cube is in G0. This is not true however for its subgroups Gi>0. The basic idea of this algorithm is to start with a cube in Gi and then move it to the next group Gi+1 by only using the moves in Gi and repeat this until it arrives in G4, meaning that it is solved. The groups are constructed in such a way that it is impossible to destroy a previous stage when only using the permitted moves. The number of elements in Gi is called the order of Gi, Gi, which decreases as i increases. Taking into account that the double and anticlockwise are counted as one move, thus implicitly included in the groups, the order of each group is given below:

G0 = 18; G1 = 14; G2 = 8; G3 = 6; G4 = 0

Using the above mentioned movesets and using the generated Pruning tables as lookup tables, Rubik's cube can be solved under 45 moves.

Discussion and Achievements:

After several hours of brainstorming, design of algorithms and several nights of restless programming, testing, experimenting with the Rubik's cube, our team was able to successfully recognise, identify and construct a 3D model of the Rubik's cube. The best part of it was that our team, rather relying on internet sources, (where clear algorithms are not provided), we designed our own algorithm for recognition and identification. The algorithm is also very much robust and continues to provide good results in different kinds of environments and backgrounds.

Limitations and Improvements:

Our team faced lots of problems, initially while working with the recognition and the identification algorithm. We kept checking each problem and devised solutions accordingly. However, we were never short of any limitations. One major limitation is that the Rubik's cube's face colors are reflective in nature. If light is incident directly on the face of the cube, due to its reflective nature, some colors like Yellow and White, Orange and Red cannot be differentiated as the reflected light is received by the camera lens.

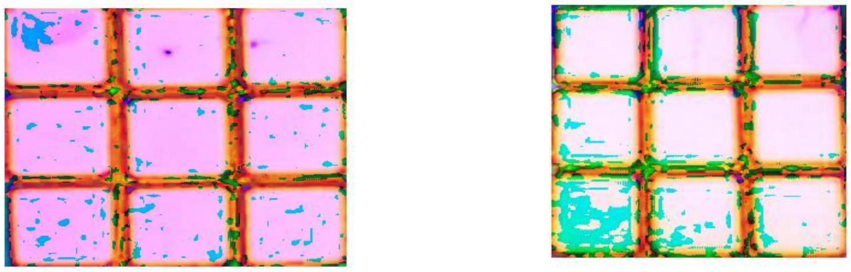

Fig.9. Red Face of the cube in different lighting situations (HSV Space)

In Figure 9, on the the left is the red face of the cube in low light. On the right, it is the same red face in bright lighting situation. Both images are in HSV space. In the bright light condition, if we look at the bottom left cubie, it is polluted by the light reflection, and the median ‘hue’ value is changed, leading to a wrong color identification. This is one limitation.

Another factor that affect identification of colors on the cube is the light intensity in the space under observation. When the light intensity is very low the detection of colors on the face of the cube becomes hard. For example, darker shades like green and blue in dark are difficult to differentiate. To balance the effect of the bad lighting condition we could change the intensity of the image in MATLAB, but it is not very reliable.

The distance between the cube to the point of observation also affects detection of the cube this is because the algorithm that we had employed for cube detection search areas within a specific range. We have to limit ourselves to detecting the cube within in specific range to avoid bad detections in the background.

One more considerable limitation is bad detection of cubes. Although we employed median technique to eliminate unnecessary squares, if there is a cluster of squares present in the background, the detection fails miserably. Also, if the cube is placed on a reflective surface, the surface reflects the bottom most squares, and they are also detected as they are present close to the median square.

We tried to incorporate the cube recognition and identification to a previously developed GUI, which could solve the Rubik’s cube. Unfortunately, due to change of versions in MATLAB, several commands in MATLAB are no longer available and we were unable to fix the error in solving.

Future Work and Improvements:

As our future work, we would like to reduce the number of limitations by applying more efficient algorithms for recognition and identification. For now, our algorithm is very much robust but not as reliable. So, we would work on the reliability part to improve the working efficiency. We also would like to take the project to a different level, where we could design a mechanism, that can solve the Rubik’s cube in real time, based on its current configuration, identified by our recognition and identification algorithm. We also would like to fix the GUI which has not been working because of change in MATLAB versions.

Note:

Our team would like to thank Dr. William Hoff for giving us this opportunity to work under him and guiding us through the project, giving valuable suggestions. Our team enjoyed and learnt a lot, working for this project.

Note 2: The report for this project is made using Google Docs and it does not have source formatting while pasting. Hence the pasted codes are occupying more space and are arranged in a haphazard manner. Instead of pasting the code, our team attached the codes along with the report in a zipped folder. Please find the codes. We apologize for this inconvenience.

References:

[1] https://www.rubiks.com/about/cube-facts/

[2] https://en.wikipedia.org/wiki/Rubik's_Cube

[3] https://www.rubiks.com/solve-it/3x3/

[4] https://www.rubiks.com/about/cube-facts/

[5] Image Courtesy for edge and corner cubies: Google Images

[6] Dr. Hoff’s code for blob feature extraction, CSCI507

[7] http://makerzone.mathworks.com/resources/arduino/rubiks-cube-solver/