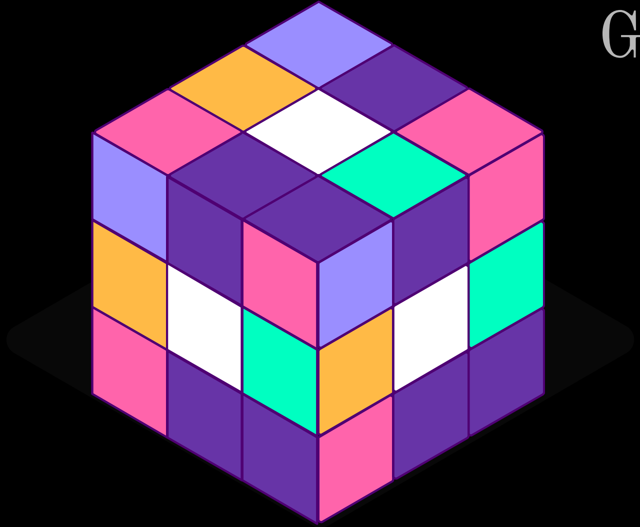
Autonomous Rubik's Cube Solver



Group Members :

IIB2019030	Kandagatla Meghana Santhoshi
IIT2019184	Pratyush Pareek
IIT2019185	R Shwethaa
IIT2019204	Mitta Lekhana Reddy
IIT2019205	Sanskar Polaki Patro
IIT2019208	Dhanush Vasa
IIT2019219	Gitika Yadav

The Program

Objective Of the Project

For the scope of this project, our objective is to develop a program that detects Rubik’s cube through a camera in real time, identifies its faces and provides an optimal solution.

The problem can be divided into three steps:

1. Detecting the initial state of the cube using a set of image .
2. Creating a set of instructions to optimally solve the cube.
3. Expressing the instructions to the user in an efficient and intuitive way.

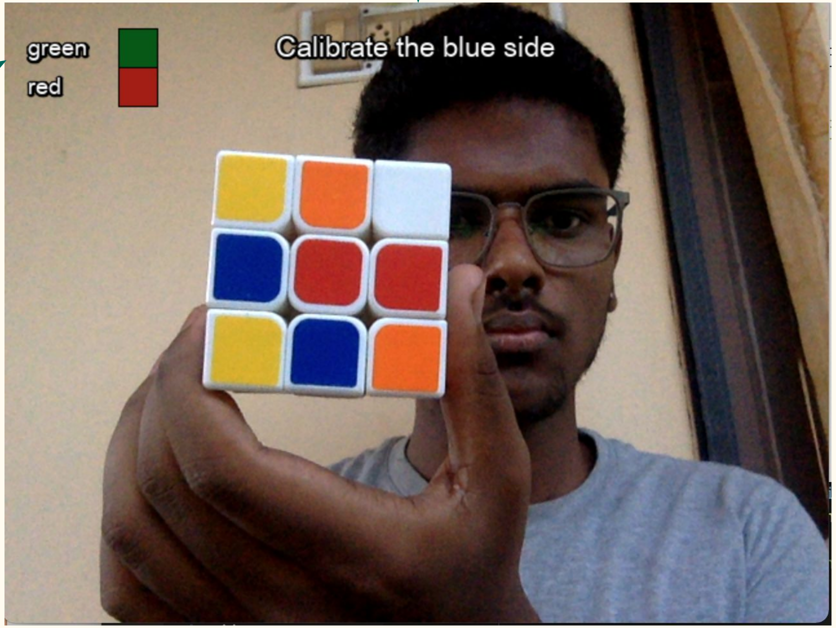
Technologies Used

- ★ Python
- ★ openCV - for solving computer vision problems
- ★ Numpy - supports large, multidimensional array and matrices and high level mathematical array functions.
- ★ **iMutils** - provides image processing functions such as : translation, rotation, resizing, sorting contours, etc.
- ★ i18n - translational module with JSON storage
- ★ **Kociemba** - contains two-phase algorithm for solving rubix cube

User Interface - Calibration Mode

Shows which color is currently being calibrated

Shows the colours that have already been calibrated.



User Experience:

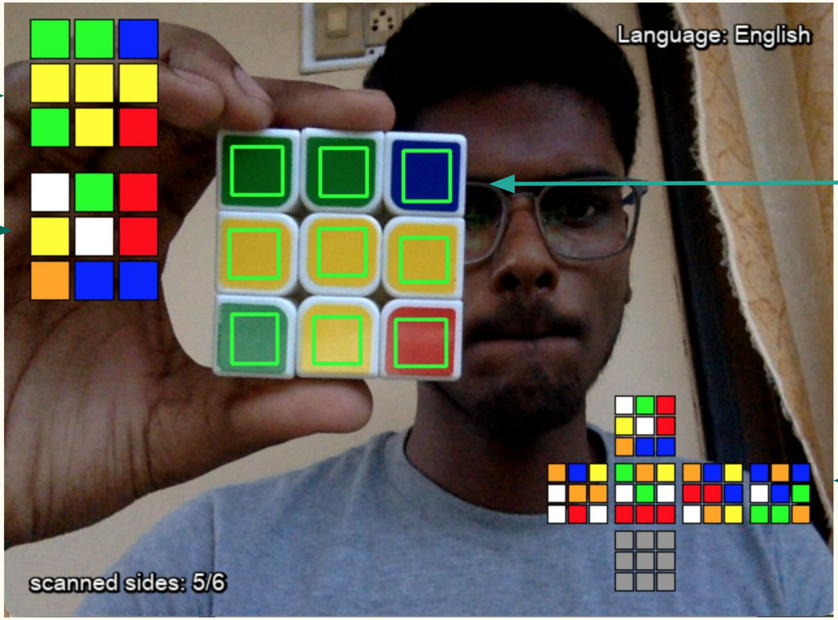
- To enter this mode, the user is required to press the ‘c’ key on the keyboard.
- The center square colour is checked for calibration.
- To confirm a colour, the user presses the spacebar.

User Interface - Solving Mode

Shows the colors that are being detected and changes in real-time as the user moves or flips the cube.

Once the user verifies the face, they can click the **spacebar** to save that face. This is shown in the second square

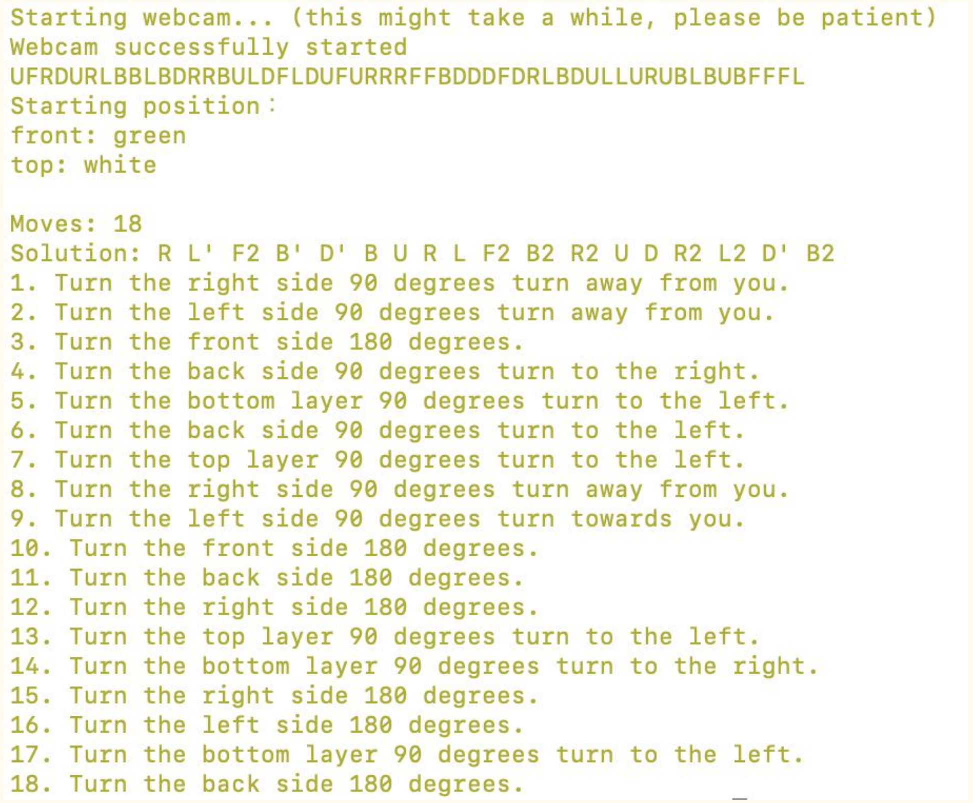
Shows the number of sides that have already been scanned



Bounding boxes denoting the squares whose colors are being detected.

Shows all the detected states on an intuitive 2D mapping of the cube.

User Interface - The Solution



```
Starting webcam... (this might take a while, please be patient)
Webcam successfully started
UFRDURLBBLBDRRBULDFLDUFURRRFFBDDDFDRLBDULLURUBLBUBFFFL
Starting position:
front: green
top: white

Moves: 18
Solution: R L' F2 B' D' B U R L F2 B2 R2 U D R2 L2 D' B2
1. Turn the right side 90 degrees turn away from you.
2. Turn the left side 90 degrees turn away from you.
3. Turn the front side 180 degrees.
4. Turn the back side 90 degrees turn to the right.
5. Turn the bottom layer 90 degrees turn to the left.
6. Turn the back side 90 degrees turn to the left.
7. Turn the top layer 90 degrees turn to the left.
8. Turn the right side 90 degrees turn away from you.
9. Turn the left side 90 degrees turn towards you.
10. Turn the front side 180 degrees.
11. Turn the back side 180 degrees.
12. Turn the right side 180 degrees.
13. Turn the top layer 90 degrees turn to the left.
14. Turn the bottom layer 90 degrees turn to the right.
15. Turn the right side 180 degrees.
16. Turn the left side 180 degrees.
17. Turn the bottom layer 90 degrees turn to the left.
18. Turn the back side 180 degrees.
```

Once the cube is scanned completely and perfectly, all the moves required to solve the cube are displayed in the terminal in layman's language.

All solutions are upto 20 moves.

The starting position and the number of moves is also displayed.

Calibrating the cube

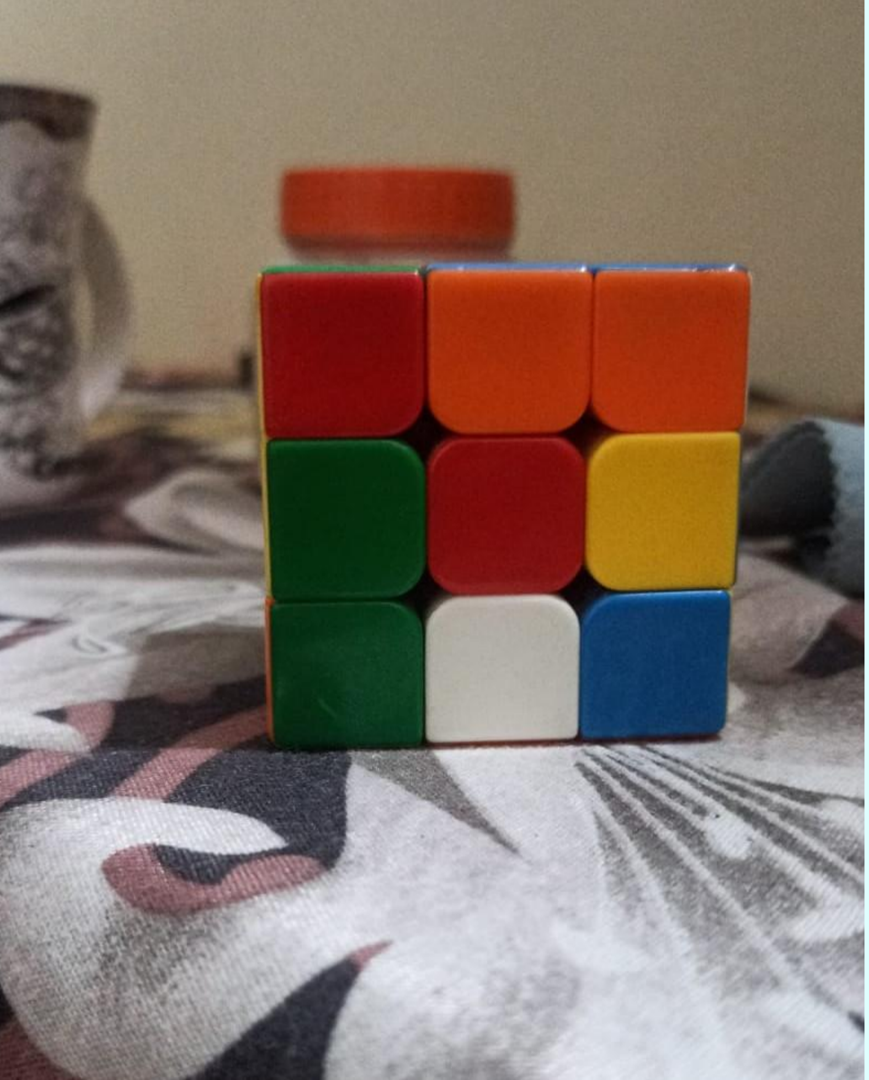
- Use the calibrate mode to let application be familiar with your cube's color scheme. If your room has proper lighting then this will give you a 99.9% guarantee that your colors will be detected properly.
- Default Colors to calibrate are 'green', 'red', 'blue', 'orange', 'white', 'yellow'.
- As the user calibrates, the program will replace these values with the ones detected during calibration.
- After Calibrating all the sides of the cube is done successfully we can proceed to the solving mode.
- The calibration mode adapts the software to the lighting and camera conditions of the user and the variety of the cube.
- After the calibration , the structure or skeleton of the cube is stored according to the Color detected at the center.

Methodology

1. Cube Detection
2. Color Detection
3. Determining the state of the cube
4. Solving the cube

Cube Detection

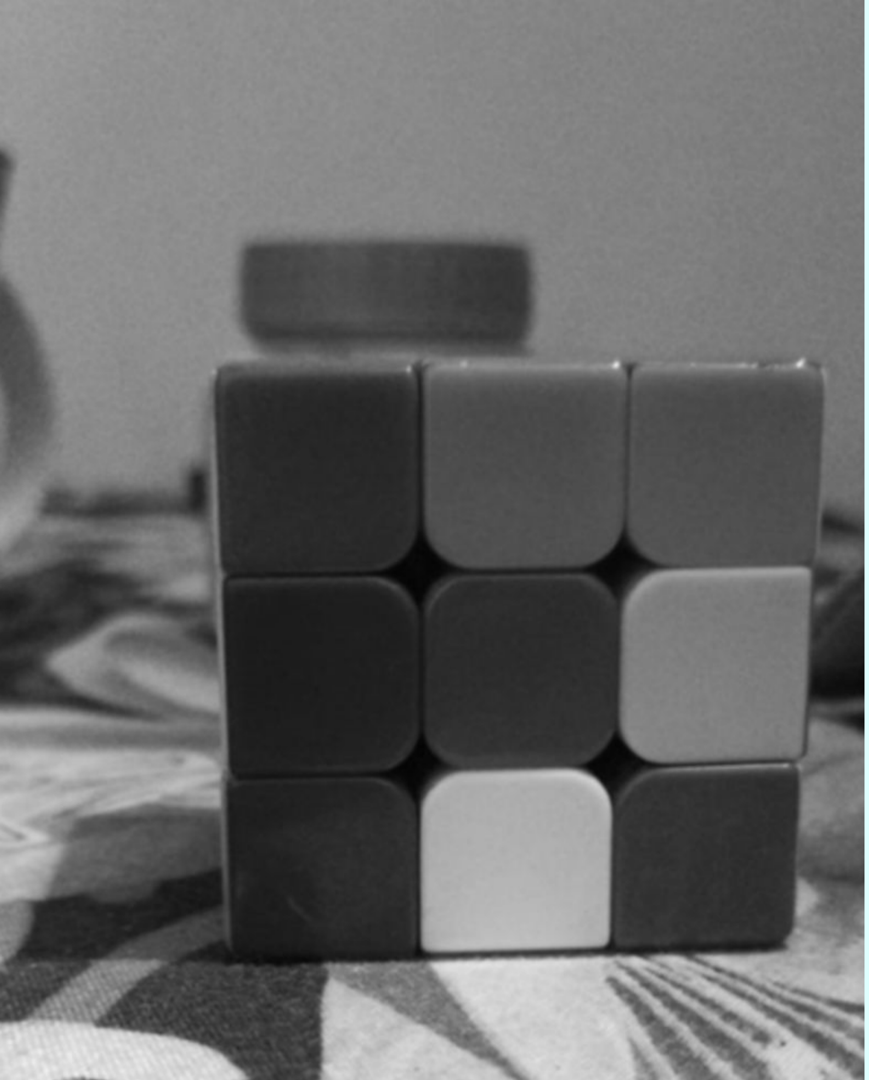
Since there may be a plethora of different objects of varying sizes,shapes and colours in the image, it is crucial to pre-process the image and remove the extra objects before we move on to detect colors.



INPUT IMAGE

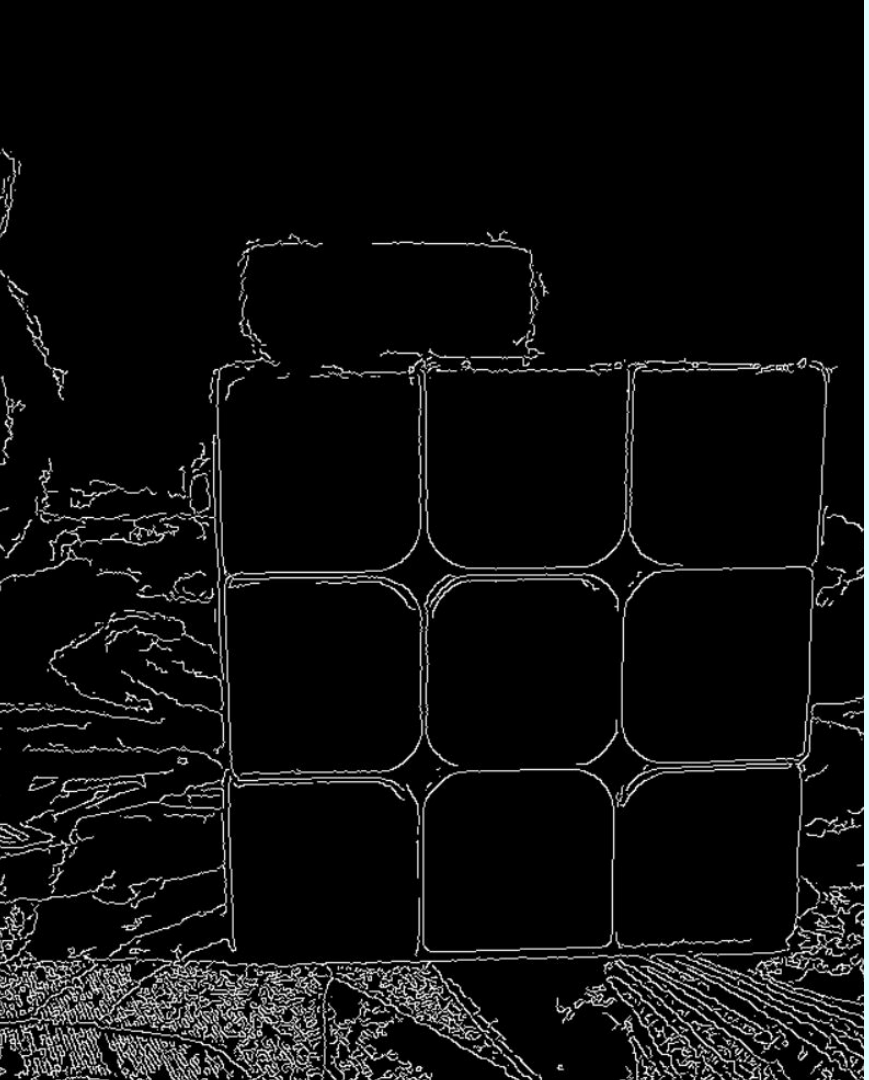
Notice the

- coffee mug
- the container in the background (that shares edges with the cube)
- The designs on the tablecloth
- The cloth on the right



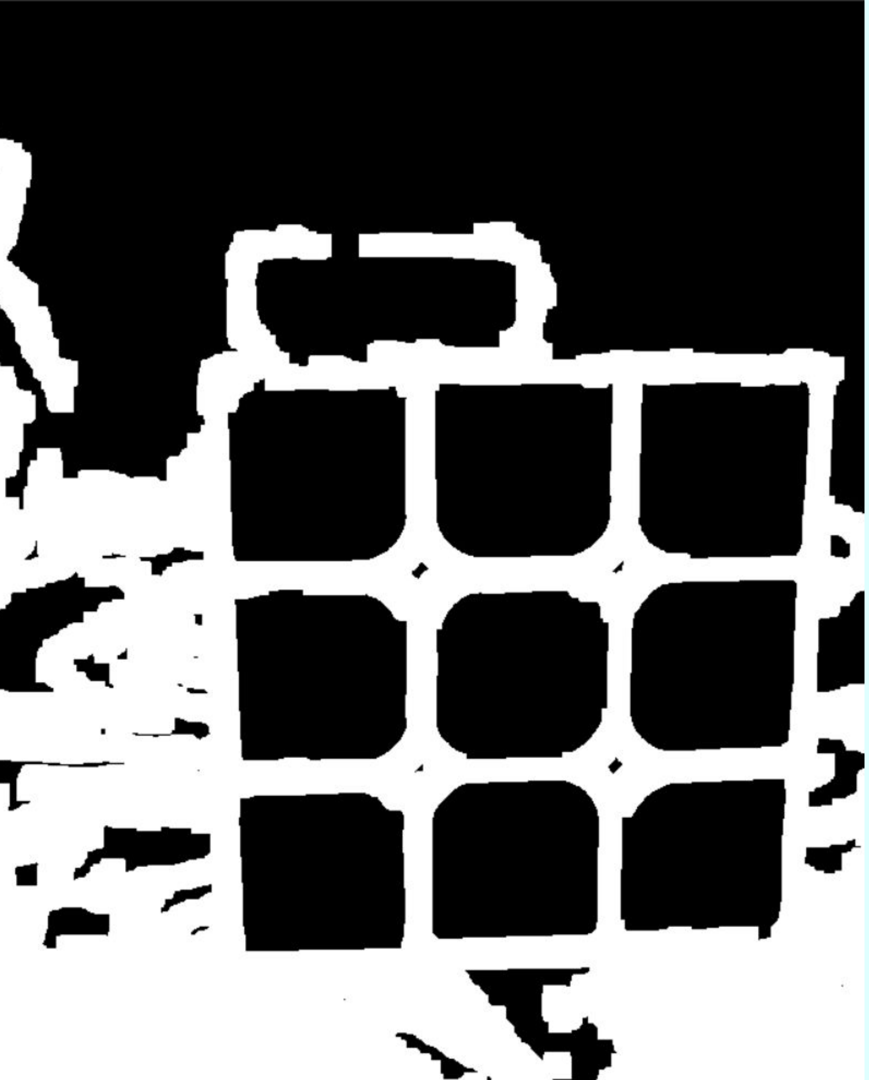
Grayscale and Blur

- Grayscale the image for better edge detection
- Blur the image to get rid of noise



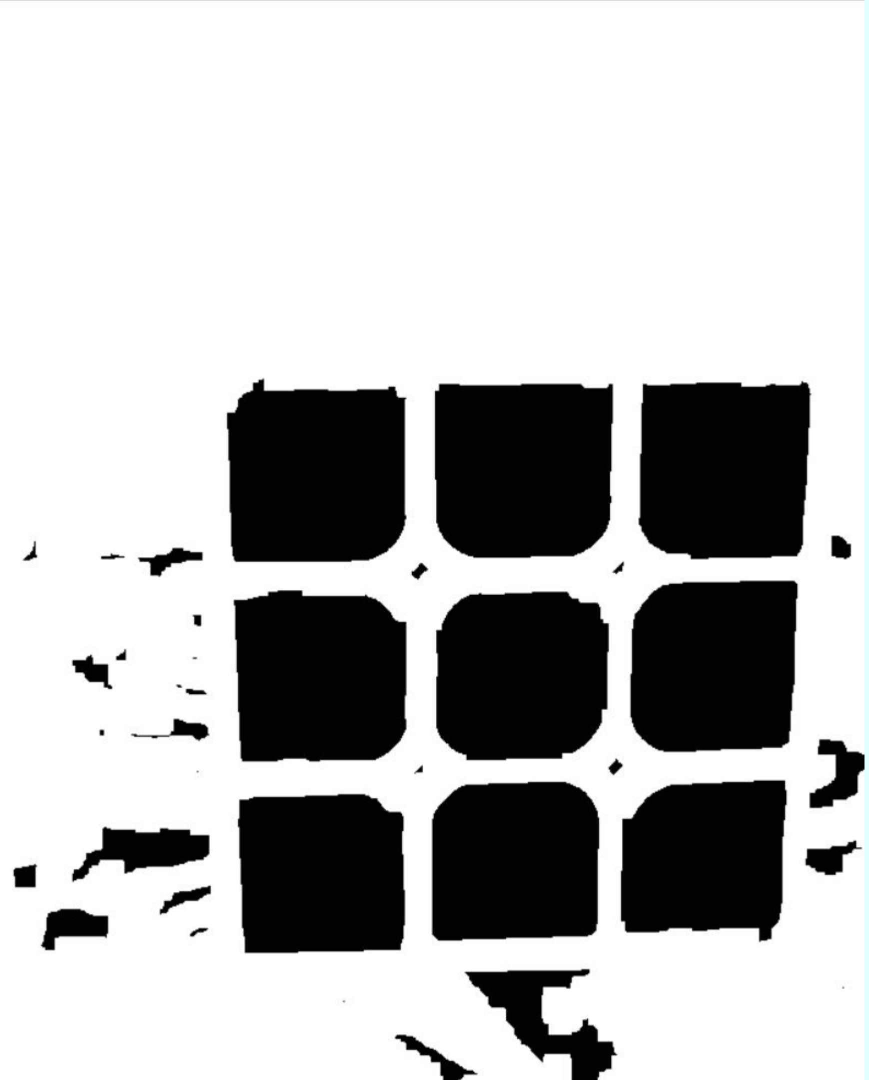
Canny Edge Detection

- Gradient Calculation
- Non maximum Suppression
- Double Thresholding
- Hysteresis



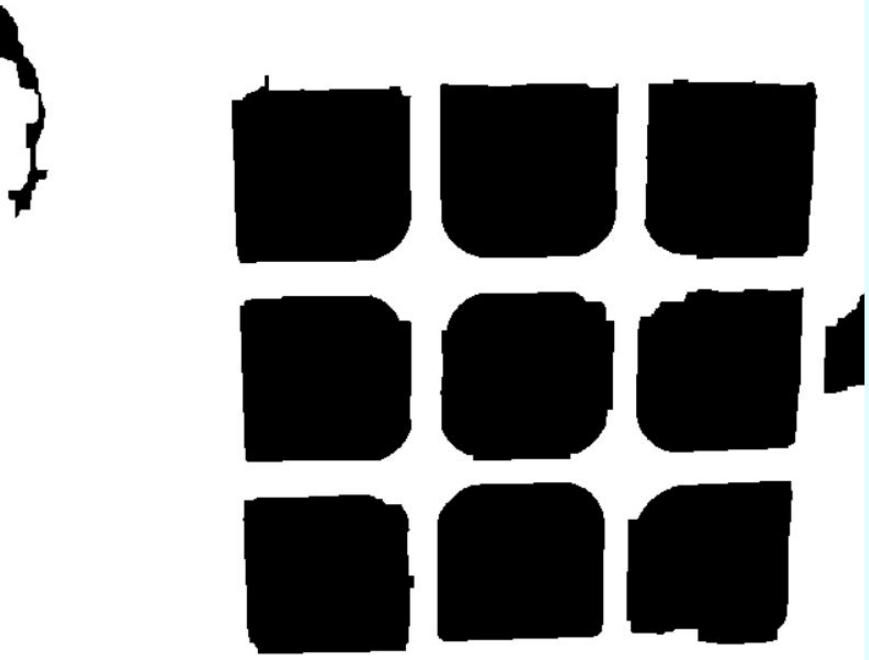
Dilation

- About 4 times
- Completes all cube edges, external and internal
- Completes all impure edges as well



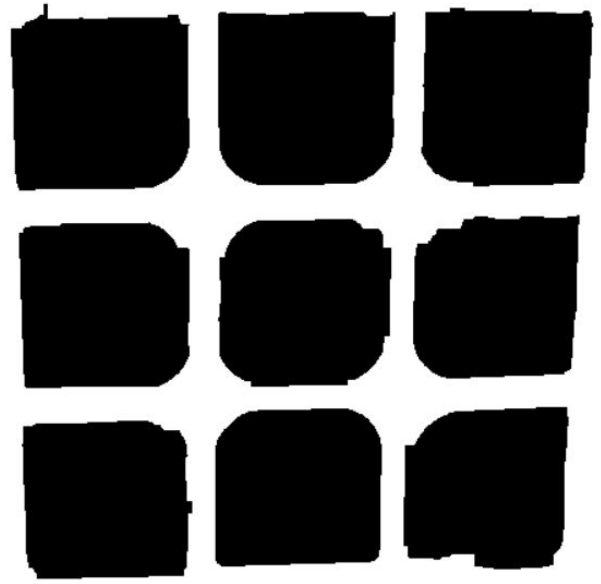
Floodfill (optional)

- Floodfill the image from each point on the edge
- Gets rid of most large impurities
- Takes advantage of the fact that all squares of the cube would be inside outlines



Area Filter

- Approximated the size of a square by hit and trial
- Any area beyond the upper limit of the expected size gets deleted



Polygon Filter

- Approximates a polygon around each contour
- Checks if the polygon has 4 sides
- Checks if the length and width are very close (i.e, the rectangle is a square)

Neighbourhood Filter

01 Check if there are atleast 9 contours still left

02 Approximate where the 8 neighbours’ centers should be for each contour

03 Iterate over all contours to see if any of them has 8 neighbours

04 The one that satisfies the conditions is the center contour, the ones other than it and its neighbours are removed

Color Detection

In Color Detection we had 2 main goals that we had to achieve.

1. Identification of Dominant Color
2. Identification of Closest Color

Color Detection : Identification of Dominant Color

In this part we help of K-Mean Clustering to identifying the Dominant Color.

Here we run the K-Means model inside cv2 for 200 iterations in order for us to reduce the number of colours to 1 while preserving the overall appearance of the region of color being detected.

In the end what happens is that we are able to return a tuple with the color that was detected.

In conclusion, the main goal of this function is to identify the dominant color in a particular region.

Color Detection : Identification of Closest Color

In this part to identify Closest Color we use 2 different source codes:

1. bgr2lab : This source code was used to convert the BGR colour identified in the region to be converted to LAB in order for use to use the next source code.
2. ciede2000 : This source code takes input of the LAB color and calculates the colour difference to the source color.

Hence as we can see we iterate through the color palette and find the color closest to the imputed color in the region. Where we end up returning that back.

Failed Methodologies

Before using the above Color Detection Method , We tried several methods that were simpler and more intuitive, and built upon them or changed them to improve the color detection in the program.

Method 1 : BGR Ranges : To match with the different ranges of colors

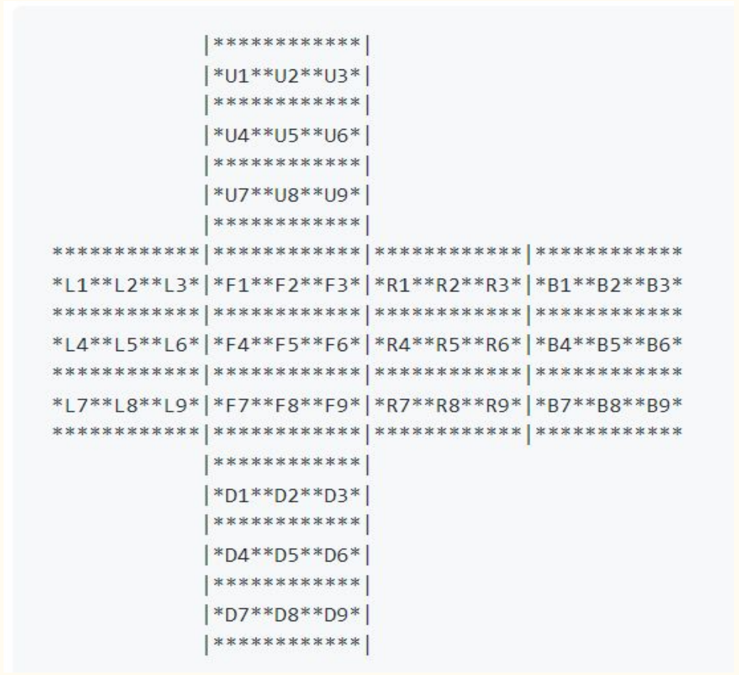
Method 2 : BGR Closest : To approximate the colours using distance between the standardised colors.

Method 3 : HSV Ranges : BGR values are converted to HSV values and then tried to match with different range of values

Determining the State of the Cube - Cube’s Data Structure

- Based on the [x,y] values stored for every calibrated color during input process, we can calculate their positions.
- After each and every side of Cube are scanned convert all these sides and BGR colors to Cube’s notation.
- User shall face the Green-centered side with the White-centered side on the top.
- This allows the normalization of the centers to the symbols: U(Up), R(Right), F(Front), D(Down), L(Left), B(Back).
- This normalized form is used for the generation of the solution. The substitution is URFDLB(white, red, green, yellow, orange, blue).

Determining the State of the Cube - Cube's Data Structure



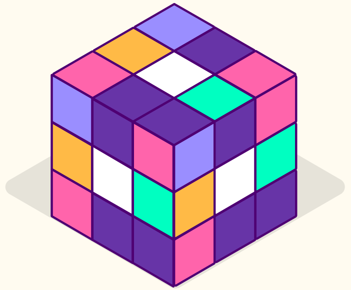
```
            |************|
            |*U1**U2**U3*|
            |************|
            |*U4**U5**U6*|
            |************|
            |*U7**U8**U9*|
            |************|
************|************|************|************
*L1**L2**L3*|*F1**F2**F3*|*R1**R2**R3*|*B1**B2**B3*
************|************|************|************
*L4**L5**L6*|*F4**F5**F6*|*R4**R5**R6*|*B4**B5**B6*
************|************|************|************
*L7**L8**L9*|*F7**F8**F9*|*R7**R8**R9*|*B7**B8**B9*
************|************|************|************
            |************|
            |*D1**D2**D3*|
            |************|
            |*D4**D5**D6*|
            |************|
            |*D7**D8**D9*|
            |************|
```

- The Contours that have been detected for each face are sorted according to the positions of their centers and their colors are stored in 6 matrices of 3x3 each.

Determining states of the Cube: Cube Verification

1. We need to find out that the Cube hasn’t been solved already. So, for that:
   - Get the center color of current side.
   - Compare the center color to all neighbors, if we come across at least one different color then, we assume that the Cube isn’t solved yet.
2. User should make sure that, while scanning a side all 9 tiles must be recognised, and overall 54 tiles must be recognised for whole cube.
3. Examples of the restrictions are that the blue centerpiece must be on opposite as the green centerpiece, there is only one blue-yellow side piece, a red-orange side piece does not exist

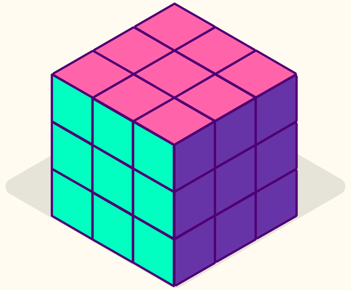
Solving cube using kociemba algorithm



Input

accepts a cube definition string

E.g.DRLUUBFBRBLURRLRUBLRDDFDLFU FUFFDBRDUBRUFLLFDDBFLUBLRBD



Output

returns a solution string in standard notation

e.g. u"D2 R' D' F2 B D R2 D2 R' F2 D' F2 U' B2 L2 U2 D R2 U"

Two phase approach

The core of the 2-phase approach is this:

- Solve the cube into a state with a certain property.
- Solve the rest of the cube.

If we select F2L as our property, we get:

- Solve the first two layers.
- Solve the last layer.

If you solve F2L with the fewest moves possible, then solve the resulting LL case optimally, you'll end up with a decently short solution. However, there might be a shorter solution: in that case, the first phase might take more moves.

Phase one uses iterative deepening to find solution.

Solution string consists of space-separated parts, each of them represents a single move:

- A single letter by itself means to turn that face clockwise 90 degrees.
- A letter followed by an apostrophe means to turn that face counterclockwise 90 degrees.
- A letter with the number 2 after it means to turn that face 180 degrees.
- e.g. R U R’ U R U2 R’ U

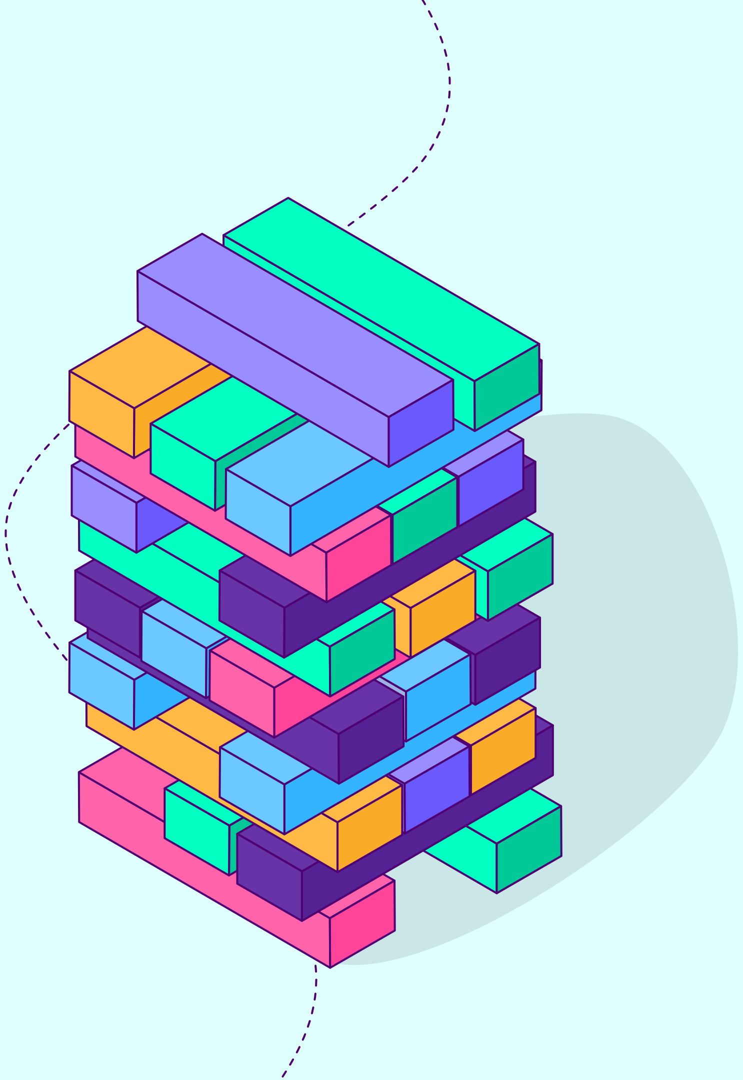
If we keep going, we reach a point where any solution we haven't considered must be longer than the one we have -- so our solution is optimal. For 3x3x3, we happen to know that the optimal solution must be at most 20 moves... but that's only because we know God's number which is the upper bound

Intuitive Solutions

- Many people won't understand the cube and won't bother learning.
- For them, they can just add "-n" to their command (while running the program) to get a normalized solution at the end.
- The normalized uses a hard-coded decision tree and substitution to replace Kociemba instructions like "R" to "Rotate the right side by 90 degrees away from you".

Results and Conclusion

- The lighting of the room plays a major role in the detected colours and even edges.
- It was much easier to detect cube with thick internal boundaries and timid colours as compared to those without internal boundaries and deeper colours.•
- Lab format performed significantly better than HSV format which performed better than BGR when it came to detecting the color of a square and classifying it.
- Color detection improved significantly after adding a Calibration mode to orient the software to the lighting conditions and camera.
- Color classification improved significantly by using K-means clustering.
- The software has been designed such that it can be extended to several other3x3x3 cubes, including but not limited to the mirror cube, diagonal cube, andmany more.
- CNNs and large datasets aren't always necessary to solve image-processingproblems. Softwares can be hard-programmed more specifically to the taskand may outperform soft-programmed models.



THANK YOU