

RUBIKS CUBE SOLVER

**A Project Report submitted in partial fulfillment of the requirements for the award
of the degree of**

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

CS Venkata Adithya 221910301015

Reesu B D Venkata Krishna 221910301047

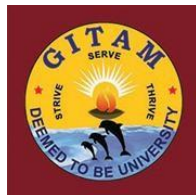
Yalaka Varsha 221910301056

Yerramilli Akhil 221910301058

Under the esteemed guidance of

Mrs. V. Rekha

Teaching Assistant



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

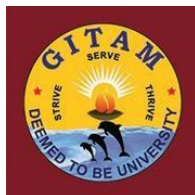
GITAM

(Deemed to be University)

HYDERABAD

NOVEMBER 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM INSTITUTE OF TECHNOLOGY
GITAM
(Deemed to be University)



DECLARATION

I/We, hereby declare that the project report entitled “**RUBIKS CUBE SOLVER**” is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s)	Name(s)	Signature
221910301015	CS Venkata Adithya	
221910301047	Reesu B D Venkata Krishna	
221910301056	Yalaka Varsha	
221910301058	Yerramilli Akhil	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)



CERTIFICATE

This is to certify that the project report entitled “**RUBIKS CUBE SOLVER**” is a bonafide record of work carried out by **CS Venkata Adithya (221910301015)**, **Reesu B D Venkata Krishna (221910301047)** , **Yalaka Varsha (221910301056)**, **Yerramilli Akhil (221910301058)** submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide

Head of the Department

Mrs. V. Rekha

Dr. S. Phani Kumar

Teaching Assistant

Professor

TABLE OF CONTENTS

1.	Abstract	1
2.	Introduction	
2.1	Cube Faces	2-4
2.2	Cube Rotations	
2.3	Cube Turns	
3.	Literature Review	5-6
4.	Problem Identification & Objectives	7
5.	System Methodology	
5.1	UML Diagram	8-10
5.2	Class Diagram	
5.3	Incremental Design	
6.	Overview of Technologies	
6.1	VS Code	
6.2	Python	
6.2.1	Python Installation	11-18
6.3	Libraries Used	
6.3.1	Numpy	
6.3.2	Open CV	
7.	Implementation	
7.1	Kociemba Algorithm	
7.2	HSV Color Scale	
7.2.1	HSV Range in OpenCV	19-33
7.2.2	Finding HSV Values	
7.2.3	Masking of HSV Images	
7.3	Coding	
8.	Result Analysis	34-35
9.	Conclusion & Future Scope	36
10.	References	37

TABLE OF FIGURES

Figure 2.1	Shows the depiction of faces of the cube	3
Figure 5.1	Class Diagram	8
Figure 5.2	Incremental Design Process	9
Figure 7.1	Typical Representation	19
Figure 7.1.1	Shows all the different pieces	20
Figure 7.2	HSV Color Scale	22
Figure 7.2.3	Rubik's Cube	25
Figure 7.2.4	Masking of Rubik's Cube	25
Figure 7.3	Snapshots of setcube.py	29
Figure 8.1	Detecting faces of the cube	34

1. ABSTRACT

Only one of the 43 quintillion possible states of the original-size Rubik's cube (3x3x3) is desired solved state. The challenge in solving the cube is to find an algorithm that solves the cube in the fewest steps possible. The Kociemba algorithm, which requires some pre-processing tasks like depth first search and tree pruning, can solve the cube less moves. As a result, we will be developing a programme to solve a Rubik's cube using Python and OpenCV as a team. To accomplish this, we will extract all of the cubies' colours and then use Kociemba's algorithm to return a solution to the users. To do this, we will extract all of the cubies colours and then apply HSV color scale to provide a solution to convert the BGR to HSV and detect the faces of the cube in each and every environment possible. The aim of a Rubik's Cube problem is to start with a randomised, scrambled, and jumbled cube configuration and, by rotating the faces, return it to the original solved pattern with each side being a single colour.

2. INTRODUCTION

The Rubik's cube was invented in 1974 by the Hungarian architecture professor Erno Rubik. Over the years it has captured the imagination of millions of people and quickly became one of the world's best selling toys ever and the most famous combinatorial puzzle of its time. It is remarkable that so many years after its invention there are still debates about the underlying mathematical principles of the Rubik's cube. Its popularity can be attributed to the simplicity of its design, which is combined with the mathematical complexity of the puzzle. Solving Rubik's cube is a challenge for everyone who likes solving puzzles and it is as old as the cube itself. The original-size Rubik's cube (3x3x3) has 43 quintillion possible states, of which only one is desired- the solved state. First algorithms for solving the cube were simple and had only one task-to solve the cube successfully. During the past three decades, as the cube gained popularity, better and better algorithms were arising. Most of the algorithms are designed in a way suitable for manipulation with human hands. Using a variety of algorithms, the legal arrangement of the Rubik's Cube can be solved in 20 moves or less. To simplify equations, the Rubik's cube is notated based on side, turns, and cube rotation to allow for easier understanding of the algorithms. The "**Singmaster notation**," proposed by David Singmaster in 1979, is used to denote a sequence of moves on the 3x3x3 Rubik's Cube.

2.1. CUBE FACES

These faces make the most sense when you hold the cube with one face parallel to the ground and one face facing you, but algorithm pages frequently display the cube with the front, right, and top faces visible.

Figure 2.1: Shows the depiction of faces of the cube

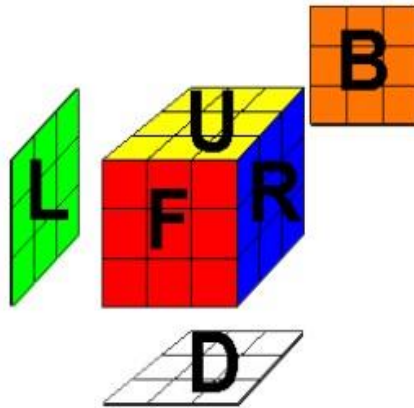


Figure 2.1 shows the faces of the cube

- **F (Front)** - the side facing you.
- **U (Up)** - the side facing upwards.
- **R (Right)** - the side facing to the right.
- **B (Back)** - the side facing away from you.
- **D (Down)** - the side facing downwards.
- **L (Left)** - the side facing to the left.

2.2. CUBE ROTATIONS

Cube rotations entail rotating the entire cube. Although it does not count as "move," changing cube perspective aids in shortening algorithms.

The cube rotations that can be modified with (90 degree counter-clockwise) or 2 (180 degree) turn clockwise or anti-clockwise) like a face turn are as follows:

- **x or [r]** - a rotation of the entire cube as if doing an R turn.
- **y or [u]** - a rotation of the entire cube as if doing a U turn.
- **z or [f]** - a rotation of the entire cube as if doing an F turn.

2.3. CUBE TURNS

A turn of one layer of one of the cube's six faces is written by appending a suffix (F, U, R, B, L, or D) to the face's name. There are three different ways to turn a face, and all moves should be performed as if you were looking at the face straight in the eyes.

Using the U face as an example, the following are possible turns:

- **U** - A 90-degree clockwise turn of the U face.
- **U'** - A 90-degree counter clockwise turn of the U face.
- **U2** - A 180-degree turn (either clockwise or counter clockwise) of the U face.

3. LITERATURE REVIEW

We have taken the inspiration to work on this project from the IEEE paper . **Two approaches in solving Rubik's cube with Hardware-Software Co- design.**

The methodology for solving the Rubik's Cube usually includes some or all of the following phases:

- Identifying the initial state (may include colour recognition).
 - The generation of a solution for a randomly scrambled cube.
 - Implementation of the cube-solving moves.
-
- In this paper, two algorithms were used to solve the Rubik's cube, but we would like to implement the most effective of the two.
 - **Hardware design:** The mechanical design of the robot was one of its most important components. One stepper motor is mounted with extreme precision on each of the cuboid's four side walls, so that motors on opposite faces have coaxial shafts. A gripper driven by servo motors has been mounted on each of these motors.
 - **Electronics:** The electronic component includes a microcontroller, which serves as the robot's brain. The microcontroller used is a "Arduino Mega," and the motors are connected to the microcontroller's pins, which give engine commands based on code stored inside the microcontroller's chip or Atmega. Image processing is a technique for storing the colours of any object in a format that can be processed using codes and program.
 - **Software Architecture:** The image is being processed by taking images of all the cube's faces. The kociemba algorithm determines the order in which the images are captured. Image Binary Conversion: The camera records a 3-channel RGB image. Each colour has three values that distinguish it from others (hue, saturation, and value). Using this property, a binary image for each of the cube's six colours can be generated.

- **Centroid Detection & Filtration:** For each colour, blob detection is performed on the binary image. The CV blob library provided by OpenCV is used for blob recognition. Algorithm Koceimba Once we've determined the matrix that stores the colour of each block on the cube's six faces, we'll need an algorithm to solve the puzzle.

As you are aware, building a robotic arm from hardware components is a time-consuming and difficult process. A breakdown at work can be extremely costly. To achieve a cost-effective solution, we attempted to make it entirely software-based by eliminating hardware components and introducing computer vision.

4. PROBLEM IDENTIFICATION AND OBJECTIVES

The main problem in solving Rubix Cube is color recognition. In some cases, different colors may not be detected correctly due to different lighting conditions, shadows, and even the noise the camera adds as it clicks and processes the image. This cube cannot be resolved and the whole cube cannot be seen. Therefore, we use the HSV color model. The objective of the Rubik's Cube puzzle is to start with a random, shuffled, jumbled configuration of cubes and rotate the faces back to the original solved pattern. Each side is solid color.

As a result, we employ the HSV colour model. The goal of a Rubik's Cube puzzle is to start with a randomised, shuffled, and jumbled cube configuration and, by rotating the faces, return to the original solved pattern with each side being a single colour.

Above all, solving the rubix cube is a problem statement to be considered, with the goal of completing it in the fewest steps possible. In this project, opencv is used to scan each face of the cube.

The HSV colour model is used to obtain the exact colour of the images with no colour variations on the cube images. We face difficulties in defining colour boundaries because RGB values are not unique.

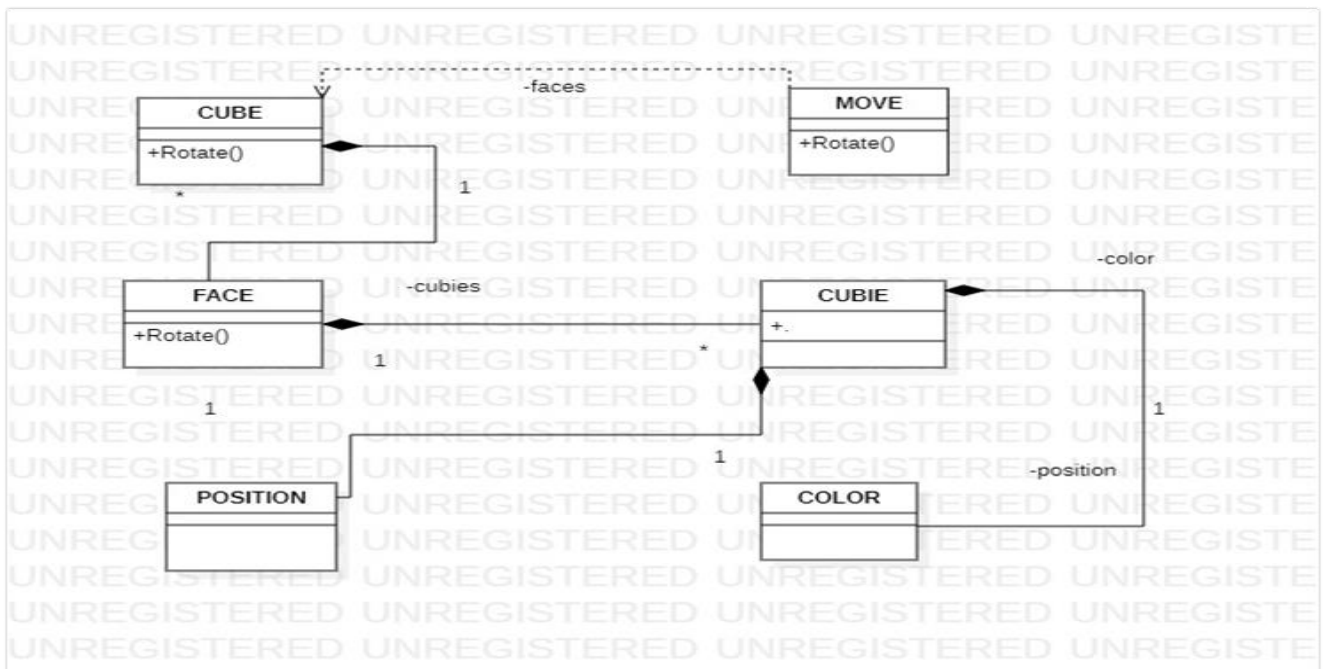
5. SYSTEM METHODOLOGY

5.1. UML Diagram

Unified Model Diagram is a versatile modeling language. The main goal of UML is to define a standard way of visualizing how systems are designed. This is very similar to blueprints used in other engineering disciplines.

5.2. Class Diagram

Class diagrams are used in the UML diagrams used to design the project. A static diagram is a class diagram. A class diagram is a static representation of an application. Class diagrams are used to create executable code for software applications as well as to visualise, describe, and document various aspects of a system. increase.

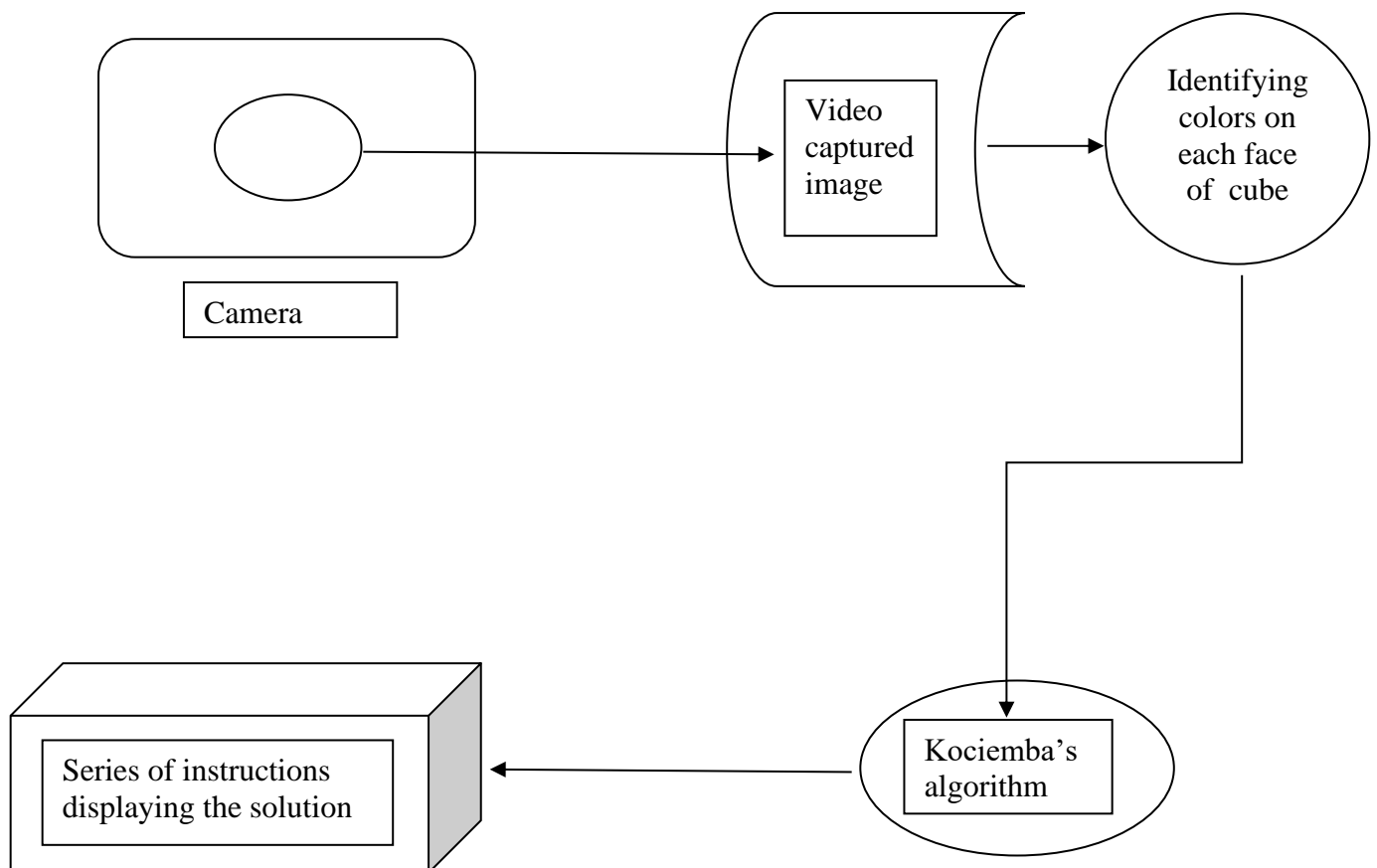


5.1 Class Diagram

- There are six Faces in the Cube class. There are nine Cubies on each Face.
- Each Cubie has two attributes: Position and Color.
- There are totally 18 one layer moves, 18 two layer moves, and 3 roll cube moves.
- There are helper classes, such as FaceLayout2D and FaceLayout3D, which describe the relations of faces in the Cube.
- The Rubik Pattern class records the interesting patterns.
- The Cube Reader class helps input the state of a Cube into the program.

5.3. Incremental Design

The concepts introduced in test-driven development are applied to all levels of design in incremental design. Programmers, like test-driven development, work in small steps, proving each before moving on to the next. This is done in three stages: first, create the simplest design that could possibly work, then incrementally add to it as the software's needs evolve, and finally, continuously improve the design by reflecting on its strengths and weaknesses.



5.2 Incremental Design Process

To program and test all of the functions, we used an incremental design process. We used a Python library called open CV after we had captured all of the photos. To determine the colour of each square. The kociemba algorithm will then be used to solve the cube. Kociemba's algorithm employs a two-phase approach that allows it to find a solution more quickly. We ran the algorithm on a laptop and sent the output over.

The output string of Kociemba's algorithm was slightly modified to delineate the start and end of the moves.

6. OVERVIEW OF TECHNOLOGIES

The technologies which we used in the project are written below with description about them in detail.

6.1. VS Code

Visual Studio Code is a lightweight, powerful source code editor for Windows, macOS, and Linux that runs on your desktop. It includes built-in support for JavaScript, Typescript, and Node.js, as well as a robust ecosystem of extensions for other languages and runtimes.

6.2. PYTHON 3.6

Python is a well-known programming language. Guido van Rossum discovered it. Python is an open source programming language with many built-in features. It is the most basic language because it has extensive libraries and modules. In the project, Opencv, Tesseract, Matplotlib, and Numpy are primarily used for image recognition and text extraction, as well as creating web forms with the obtained data.

Prerequisites

- A system running Windows 10 with admin privileges
- Command Prompt (comes with Windows by default)
- A Remote Desktop Connection app (use if you are installing Python on a remote Windows server).

6.2.1 Python 3 Installation on Windows

Step 1: Select Version of Python to Install

The installation procedure entails downloading and running the official Python.exe installer on your system. The version you require is determined by what you intend to do with Python. For example, if you're working on a Python version 2.6 project, you'll most likely want that version.

Step 2: Download Python Executable Installer

- Open your web browser and go to the Python website's Downloads for Windows section.
- **Browse to the** desired version of Python. At the time of this **article's publication**, the latest **version of** Python 3 is version **3.7.3** and the latest **version of** Python 2 is version 2.7.16.

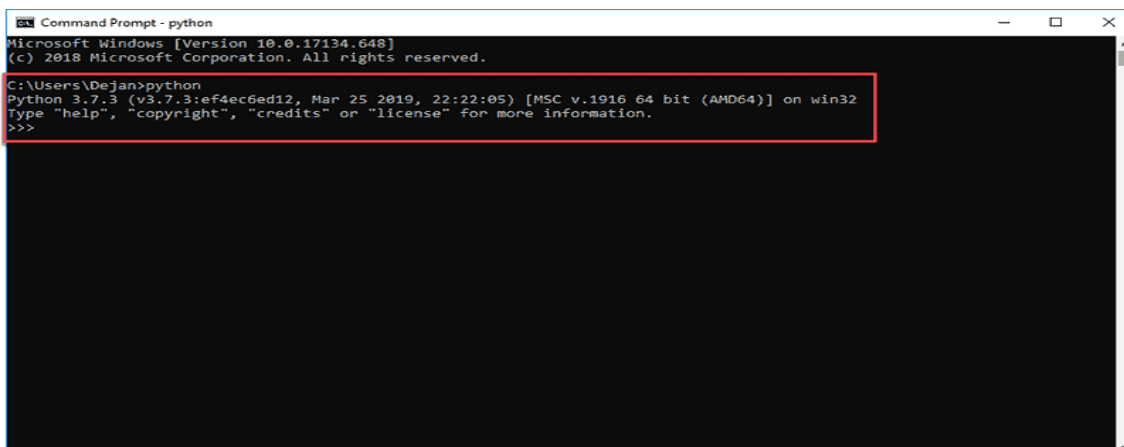
Step 3: Run Executable Installer

- Run the Python Installer when it has been downloaded. Check the Install launcher for all users and Add Python 3.7 to PATH checkboxes. The interpreter is inserted into the execution path by the former. For previous Python versions that do not have the Add Python to Path checkbox enabled.
- Select **Install Now** – the recommended installation options.

Step 4: Verify Python Was Installed On Windows

Navigate to the directory in which Python was installed on the system. In our case, it is `C:\Users\Username\AppData\Local\Programs\Python\Python37` since we have installed the latest version.

Double-click **python.exe**.

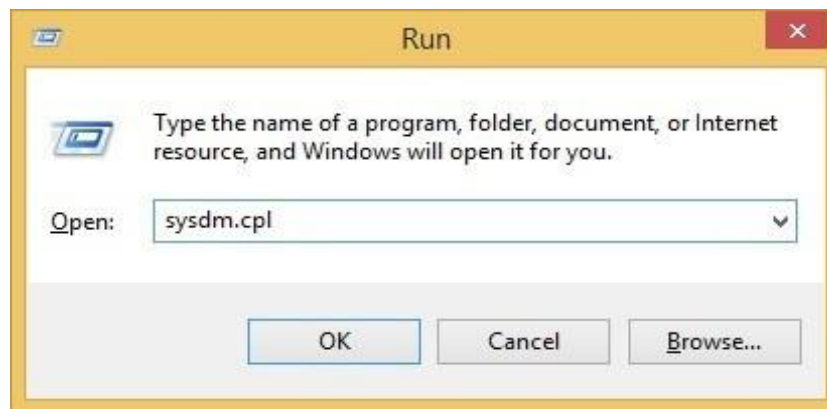


```
Command Prompt - python
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Dejan>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Step 5: Verify Pip Was Installed

- It's likely that if you installed an older version of Python, it didn't come with Pip preinstalled. Pip is a strong Python software package management system. As a result, ensure that you have it installed.
- For most Python packages, we suggest Pip, especially when working in virtual environments.
- To verify whether Pip was installed
- Open the **Start** menu and type "**cmd**."
- Select the **Command Prompt** application.
- Enter **pip -V** in the console. If Pip was installed successfully, you should see the following output.



- Type **sysdm.cpl** and click **OK**. This opens the **System Properties** window.
- Navigate to the **Advanced** tab and select **Environment Variables**.
- Under **System Variables**, find and select the **Path** variable.
- Click **Edit**.
- Select the **Variable value** field. Add the path to the **python.exe** file preceded with a **semicolon (;)**. For example, in the image below, we have added **";C:\Python34**.
- Click **OK** and close all windows.
- By setting this up, you can execute Python scripts like this: **Python script.py**
- Instead of this: **C:/Python34/Python script.py**
- As you can see, it is cleaner and more manageable.

6.3. LIBRARIES USED

6.3.1. Numpy

NumPy is a Python programming language module that adds functionality for huge, multi-dimensional arrays and matrices. The array object in NumPy is named `ndarray`, and it comes with a slew of helper methods that make dealing with `ndarray` a breeze. NumPy arrays, unlike lists, are kept in a single continuous location in memory, allowing program to access and modify them relatively fast.

6.3.2. Open Computer Vision(Open CV)

- **COMPUTER VISION**

Computer Vision is the process of understanding how photos and videos are stored and how we may change and retrieve data from them. Computer Vision is the foundation or primary application of Artificial Intelligence. Computer vision is becoming increasingly important in self-driving cars, robots.

- **OpenCV**

OpenCV is a large open-source library for computer vision, machine learning, and image processing, and it currently plays a significant part in real-time operation, which is critical in today's systems.

It can analyse photos and movies to recognise items, people, and even human handwriting. Python can process the OpenCV array structure for analysis when combined with other libraries such as NumPy. We employ vector space and execute mathematical operations on these characteristics to identify visual patterns and their different features.

OpenCV Installation

To install OpenCV, one must have Python and PIP, preinstalled on their system.

Step1: Check if your system already contains Python

- go through the following instructions:
- Open the Command line(search for cmd in the Run dialog(+ R)).
- Now run the following command:
 - **python --version**
- If Python is already installed, it will generate a message with the Python version available.
 - **python-version-check-windows**
- If Python is not present, install Python.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Yashwanth> pip -V
```

Step2: Check if your system already contains Pip

- To check if PIP is already installed on your system, just go to the command line and execute the following command:
 - **pip -V**
- Verification of pip

```
PS C:\Users\Yashwanth> pip install opencv-python
```

- If PIP is not present, install PIP.

Step3 : Downloading and Installing OpenCV

- OpenCV can be directly downloaded and installed with the use of pip (package manager).
- To install OpenCV, just go to the command-line and type the following command
 - **pip install opencv-python**
- Beginning with the installation

- Type the command in the Terminal and proceed

```
PS C:\Users\Yashwanth> pip install opencv-python
Collecting opencv-python
  Using cached opencv_python-4.6.0.66-cp36-abi3-win_amd64.whl (35.6 MB)
Requirement already satisfied: numpy>=1.14.5 in c:\users\yashwanth\appdata\local\programs\python\python310\lib\site-packages (from opencv-python) (1.22.4)
```

- Collecting Information and downloading data:

```
PS C:\Users\geetu> pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.6.0.66-cp36-abi3-win32.whl (26.2 MB)
    | 26.2 MB 726 kB/s
Collecting numpy>=1.17.3; python_version >= "3.8"
```

- Downloading-Data-for-OpenCV

```
Collecting opencv-python
  Downloading opencv_python-4.6.0.66-cp36-abi3-win32.whl (26.2 MB)
    | 26.2 MB 726 kB/s
Collecting numpy>=1.17.3; python_version >= "3.8"
  Downloading numpy-1.23.4-cp38-cp38-win32.whl (12.2 MB)
    | 12.2 MB 1.3 MB/s
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.23.4 opencv-python-4.6.0.66
```

- Finished Installation

```
Collecting opencv-python
  Downloading opencv_python-4.6.0.66-cp36-abi3-win32.whl (26.2 MB)
    | 26.2 MB 726 kB/s
Collecting numpy>=1.17.3; python_version >= "3.8"
  Downloading numpy-1.23.4-cp38-cp38-win32.whl (12.2 MB)
    | 12.2 MB 1.3 MB/s
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.23.4 opencv-python-4.6.0.66
```

- Finishing-Installation-OpenCV
- To check if OpenCV is correctly installed, just run the following commands to perform a version check:

python

>>>import cv2

>>>print(cv2.version)

7. IMPLEMENTATION

7.1. KOCIEMBA ALGORITHM

Herbert Kociemba enhanced Thistlethwaite's method in 1992. He decreased the number of groups to two, resulting in a significant reduction in needed moves to a maximum of 29 and a minimum of 19.

The core of the two-phase strategy is by solving the cube into a state with a certain attribute, then solve the rest of it.

If we choose F2L as our property, we obtain the following results:

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

You'll get a pretty quick solution if you solve F2L with the fewest moves feasible, then solve the ensuing LL case optimally. However, there may be a quicker solution in such case, the first step may require more movements but result in an easier solution.

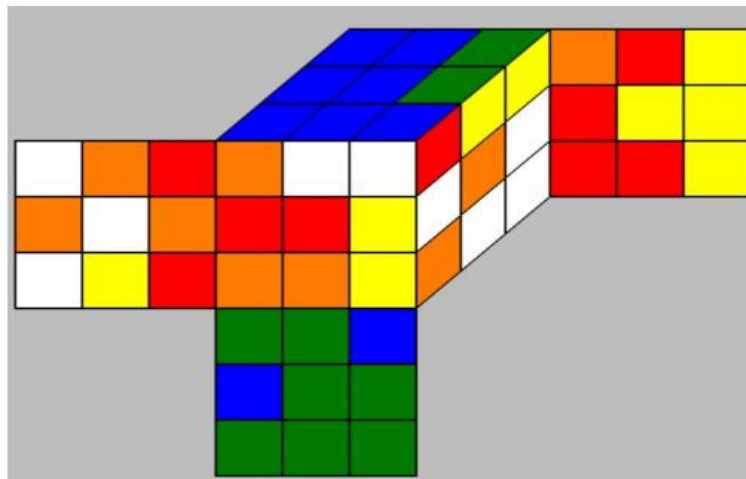


Figure: 7.1 Typical Representation

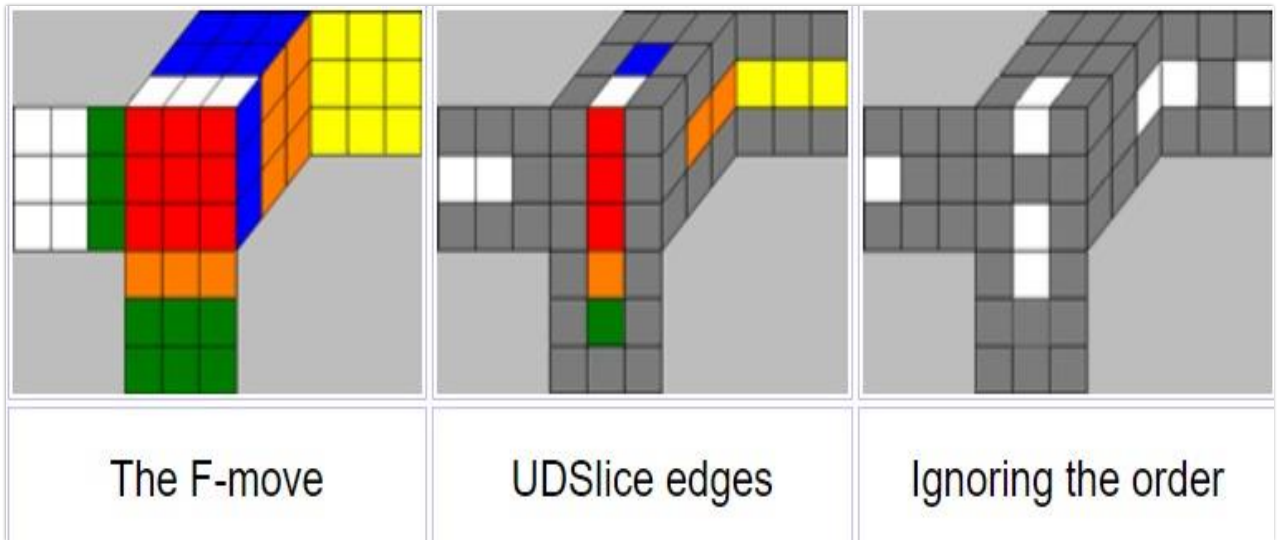


Figure: 7.1.1 shows all the different pieces

- $G_0 = \{U, D, L, R, F, B\}$
- $G_1 = \{U, D, Lx2, Rx2, Fx2, Bx2\}$
- $G_2 = \{1\}$

The principle of this algorithm is to start with group G_0 , and then move some target cubies to their expected positions by using only moves from this group.

- **Edge Piece Series** : $U' R U R'$
- **Corner Piece Series** : $U R U' L' U R' U' L$

Group G_0

- G_0 is the set of all states that can be reached with the movements L, R, F, B, U, D .
- Take note of how this is frequently merely all attainable states using any of the allowed moves provided.
- We can execute any L^2, R^2, F^2 , etc. move by modifying $L * L, R * R, F * F$. Similarly, by swapping moves $L * L * L, R * R * R, F * F * F$, we may do any L^3, R^3, F^3 .

- Our aim is to maneuver from $G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_4$. Where G_4 contains only the solved cube state.

Group G1

G_1 is the combination of these moves: L,R,F,B,U2,D2 to reach all states. G_1 differs from G_0 in that all smaller states are approachable. G_1 has only 'good' edges, hence we established that flipping any edges is impossible using just movements U, R, D, and L. rather using movements U, R, D, and L, enabling us to demonstrate that the same conclusion is feasible with moves L, R, F, B, U2, and D2.

Group G2

The group G_2 is simply the solved state. Using only moves in G_1 the state is changed directly from $G_1 \rightarrow G_2$.

7.2. HSV COLOR SCALE

- Hues are the three basic colours (red, blue, and yellow) as well as the three secondary colours (orange, green, and violet) found on the colour wheel or colour circle.
- Color saturation refers to the purity and intensity of a colour in a picture.
- The higher a color's saturation, the more vibrant and intense it is. The lower the saturation, or chroma, of a colour, the closer it is to pure grey on the grayscale.
- The relative brightness or darkness of a colour is referred to as its colour value. The amount of light reflected on a surface influences how we perceive colour value.

7.2.1. Working of HSV range in OpenCV

- Whenever we wish to address problems involving object detection, we must utilise HSV and determine the range of HSV.
- The Hue, Saturation, and Value in HSV each have their own value range.
- The HSV hue range is $[0,179]$, the Saturation range is $[0,255]$, and the Value range is $[0,255]$.
- In HSV, there is also an upper and lower limit range for each colour.

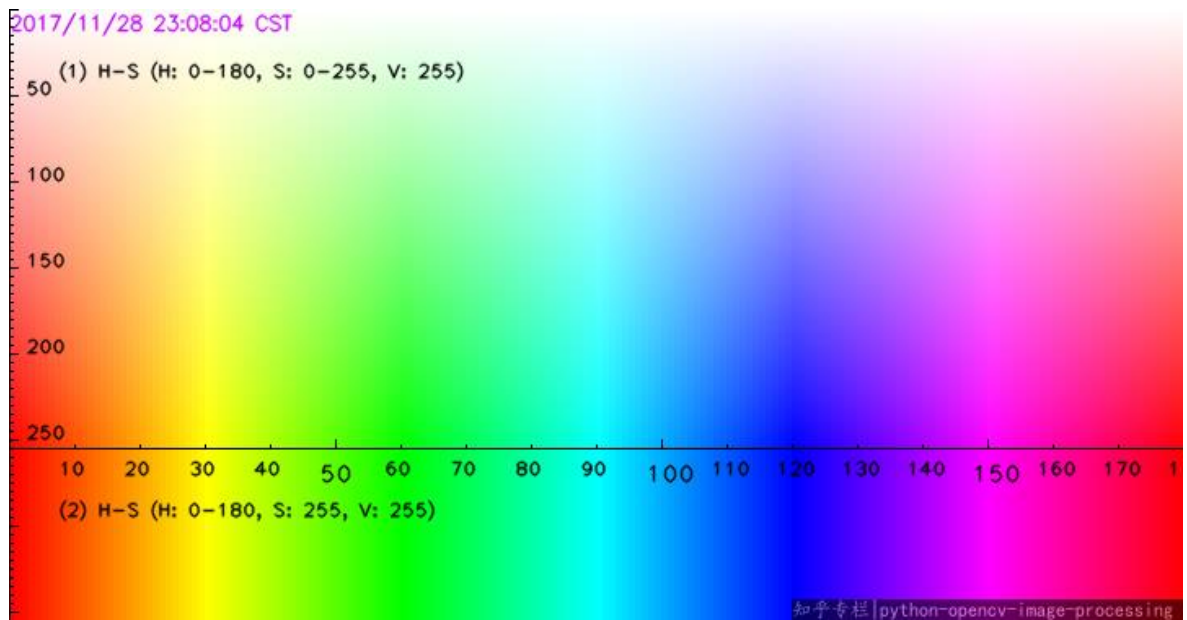


Figure: 7.2 HSV Color Scale

Figure 7.2 represents the following

- Red = Hue(0...9) AND Hue(151..180)
- Orange = Hue(10...15)
- Yellow = Hue(16..45)
- Green = Hue(46..100)
- Blue = Hue(101..150)
- White = Saturation(0..20) AND Value(230..255)

7.2.2. Finding of HSV Values

- Generally we can use `cv.cvtColor()` for finding the HSV values of color by passing the BGR values .
- But since we are working on live video capture we simply cant get the values to be as accurate.

```
In [7]: ► yellowBGR = np.uint8([[[0,255,255 ]]])

        hsv_yellow = cv2.cvtColor(yellowBGR,cv2.COLOR_BGR2HSV)
        print (hsv_yellow)

        [[[ 30 255 255]]]
```

```
In [5]: ► blueBGR = np.uint8([[[255,0,0 ]]])  
        hsv_blue = cv2.cvtColor(blueBGR,cv2.COLOR_BGR2HSV)  
        print (hsv_blue)  
        [[120 255 255]]
```

7.2.3. Masking of images using Python OpenCV

- In image processing, masking is used to produce the Region of Interest, or simply the section of the image that we are interested in.
- We typically employ bitwise operations for masking because they allow us to reject sections of the picture that we do not require.

The process of masking images

We have three steps in masking.

1. Creating a **black** canvas with the same dimensions as the image, and naming it mask.
2. Changing the values of the mask by drawing any figure in the image and providing it with a **white** color.
3. Performing the bitwise ADD operation on the image with the mask.

Understanding masking with example

```
In [2]: # importing cv2 and numpy library
import cv2
import numpy as np

# Reading an image
img = cv2.imread('rubix.jpg')

# The kernel to be used for dilation
# purpose
kernel = np.ones((5, 5), np.uint8)

# converting the image to HSV format
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# defining the lower and upper values
# of HSV, this will detect yellow colour
Lower_hsv = np.array([10, 100, 100])
Upper_hsv = np.array([20, 255, 255])

# creating the mask
Mask = cv2.inRange(hsv, Lower_hsv, Upper_hsv)

# Inverting the mask
mask_yellow = cv2.bitwise_not(Mask)
Mask = cv2.bitwise_and(img, img, mask = mask_yellow)

# Displaying the image
cv2.imshow('Mask', Mask)

# waits for user to press any key
cv2.waitKey(0)

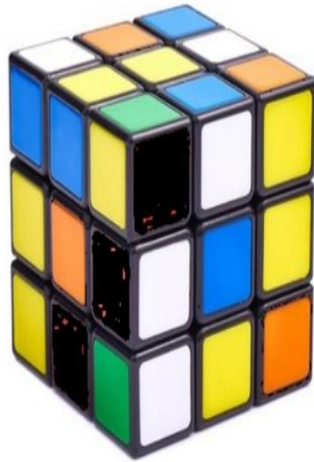
# closing all open windows
cv2.destroyAllWindows()
```



Figure 7.2.3 Shows input image named rubix.jpg

Mask

— □ ×



Mask

— □ ×

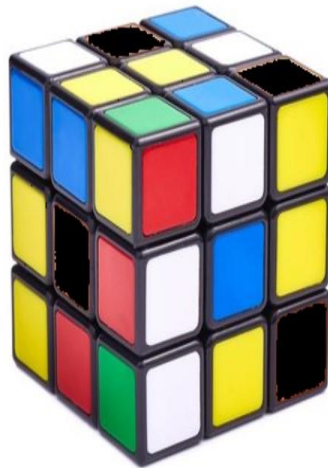


Figure: 7.2.4 Masking of different colors on the cube

7.3. CODING

As discussed above since we are working with live video capture of images we just cant confine the color to be determined based on just RGB value of the color we have to define a boundary values for colors.

```
import cv2
import sys
import numpy as np

def nothing(x):
    pass

# Create a window
cv2.namedWindow('image')
# create trackbars for color change
# Hue is from 0-179 for Opencv
cv2.createTrackbar('HMin', 'image', 0, 255, nothing)
cv2.createTrackbar('SMin', 'image', 0, 255, nothing)
cv2.createTrackbar('VMin', 'image', 0, 255, nothing)
cv2.createTrackbar('HMax', 'image', 0, 179, nothing)
cv2.createTrackbar('SMax', 'image', 0, 255, nothing)
cv2.createTrackbar('VMax', 'image', 0, 255, nothing)

# Set default value for MAX HSV trackbars.
cv2.setTrackbarPos('HMax', 'image', 179)
cv2.setTrackbarPos('SMax', 'image', 255)
cv2.setTrackbarPos('VMax', 'image', 255)

# Initialize to check if HSV min/max value changes
hMin = sMin = vMin = hMax = sMax = vMax = 0
phMin = psMin = pvMin = phMax = psMax = pvMax = 0
wait_time = 33
```

```

cap=cv2.VideoCapture(0)
while(1):
    ret,image=cap.read()
    image=cv2.flip(image,1)
    output = image
    # get current positions of all trackbars
    hMin = cv2.getTrackbarPos('HMin','image')
    sMin = cv2.getTrackbarPos('SMin','image')
    vMin = cv2.getTrackbarPos('VMin','image')

    hMax = cv2.getTrackbarPos('HMax','image')
    sMax = cv2.getTrackbarPos('SMax','image')
    vMax = cv2.getTrackbarPos('VMax','image')

    # Set minimum and max HSV values to display
    lower = np.array([hMin, sMin, vMin])
    upper = np.array([hMax, sMax, vMax])

    # Create HSV Image and threshold into a range.
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    output = cv2.bitwise_and(image,image, mask= mask)

    # Print if there is a change in HSV value
    if( (phMin != hMin) | (psMin != sMin) | (pvMin != vMin) | (phMax != hMax) | (psMax != sMax) | (pvMax != vMax) ):
        print("[%d,%d,%d],[%d,%d,%d]" % (hMin , sMin , vMin, hMax, sMax , vMax))
        phMin = hMin
        psMin = sMin
        pvMin = vMin
        phMax = hMax
        psMax = sMax
        pvMax = vMax

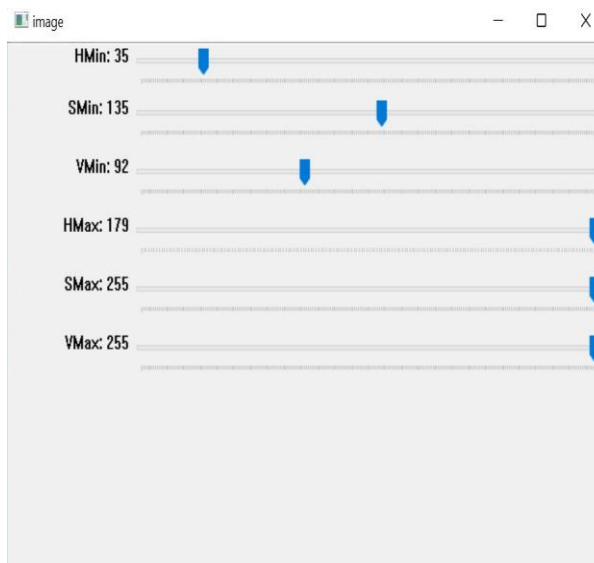
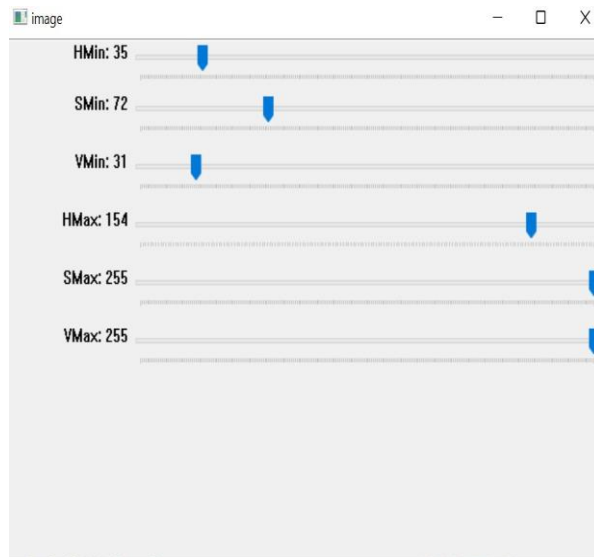
    # Display output image
    cv2.imshow('image',output)

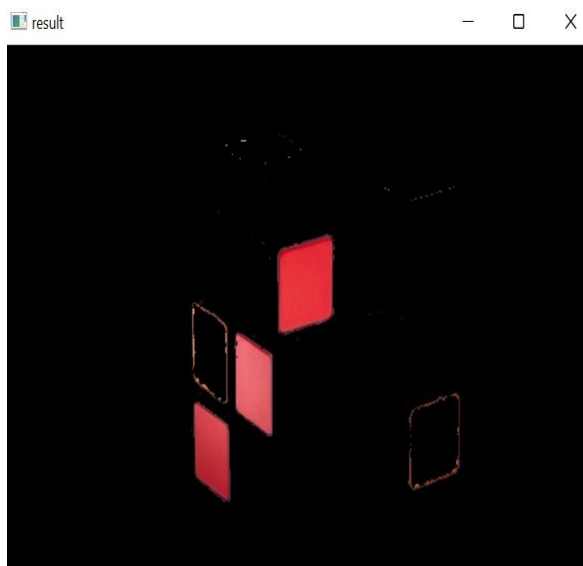
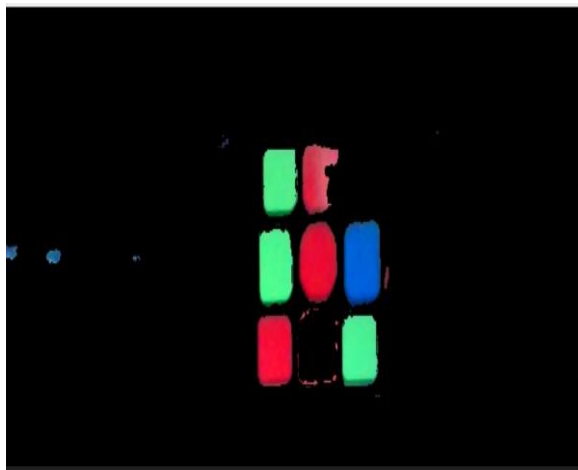
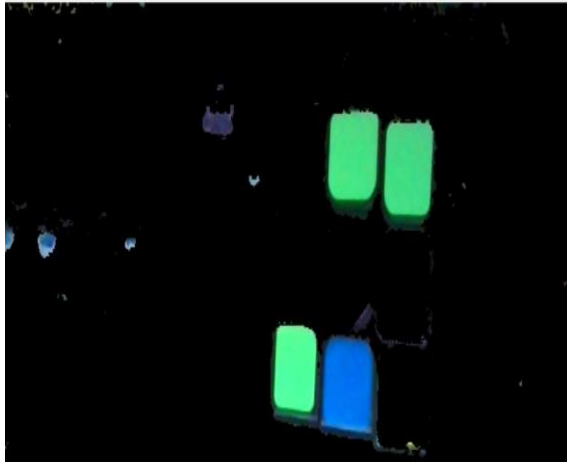
    # Wait longer to prevent freeze for videos.
    if cv2.waitKey(wait_time) & 0xFF == 27:
        break

cv2.destroyAllWindows()

```

7.3. Snapshots of setcube.py





- Now this is how we mask the images captured during the live video cam and mask the images for detecting the colors on faces of the cube .
- We can see that by changing the values of (hmin,Smin,Vmin,hmax,Vmax,Smax) we are able to detect different colors at a time.
- These kind of variations need to be consider which cannot be directly identified by using the `cv2.cvtColor()` but that is helpful in a way for finding one fixed value of the color.

```
import numpy as np
import cv2
import time
from twophase import solve
bounds = {
    "red" : (np.array([160, 75, 75]), np.array([180, 255, 255])),
    "blue" : (np.array([110, 75, 75]), np.array([130, 255, 255])),
    "green" : (np.array([35, 0, 0]), np.array([85, 255, 255])),
    "yellow" : (np.array([20, 75, 75]), np.array([40, 255, 255])),
    "white" : (np.array([0, 0, 20]), np.array([180, 30, 255])),
    "orange" : (np.array([10, 100, 100]), np.array([20, 255, 255]))
}

def density(img, color):
    lower = bounds[color][0]
    upper = bounds[color][1]
    mask = cv2.inRange(img, lower, upper)
    return np.sum(mask)/255

def cubestr(data):
    ret = ""
    for i in "URFDLB":
        ret += "".join(data[i])
    for i in "URFDLB":
        ret = ret.replace(data[i][4], i)
    return ret
```

- As discussed above we have to define the bounds of each and every color i.e we have to set lower and upper bound value for a particular color so that it can be detected easily.
- We have next defined a function named **density ()** and have given the lower and upper bounds of the color and perform the mask and return them back as BGR values.
- Now we have defined the **cubestr()** function which will take the string returned after scanning of the cube faces based on the kociemba algorithm which we will invoke later it will replace the sequence of moves to be performed. This function is useful in generating the final output sequence.
- After this we are working on the **screen_record()** function in this we are capturing the each face image of the rubix cube the initial face starting with FUDLRB.
- Now the **data** which is defined is a dictionary which stores key value pairs of corresponding Faces present on each side of the cube.
- After that we convert the image from **BGR2HSV** with the use of **cv2.cvtColor** method discussed earlier .

```

def screen_record():
    last_time = time.time()
    cv2.startWindowThread()
    # cv2.namedWindow("preview")
    cap = cv2.VideoCapture(0)
    faces = "FUDLRB"
    idx = 0

    data = {
        "F": ["", "", "", "", "", "", "", "", ""],
        "U": ["", "", "", "", "", "", "", "", ""],
        "D": ["", "", "", "", "", "", "", "", ""],
        "L": ["", "", "", "", "", "", "", "", ""],
        "R": ["", "", "", "", "", "", "", "", ""],
        "B": ["", "", "", "", "", "", "", "", ""],
    }

    while(True):
        _, img = cap.read()
        last_time = time.time()
        img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

```

```

colors = {
    "red" : (0, 0, 255),
    "blue" : (255, 0, 0),
    "green" : (0, 255, 0),
    "yellow" : (0, 255, 255),
    "white" : (255, 255, 255),
    "orange" : (0, 165, 255)
}

offset = 75
z = 0
for i in (-1, 0, 1):
    for j in (-1, 0, 1):
        px = 358 + j * offset
        py = 280 + i * offset

        maxDens = [0, "white"]
        crop = img_hsv[(py-35):(py+35), (px-35):(px+35)]
        for k in ("red", "blue", "green", "yellow", "white", "orange"):
            d = density(crop, k)
            if d > maxDens[0]:
                maxDens[0] = d
                maxDens[1] = k

        cv2.circle(img, (px, py), 5, colors[maxDens[1]], -1)
        data[faces[idx]][z] = maxDens[1][0]
        z += 1

```

```

lower = np.array([110, 100, 100])
upper = np.array([130, 255, 255])

mask = cv2.inRange(img_hsv, lower, upper)
output = cv2.bitwise_and(img, img, mask = mask)

cv2.imshow(faces[idx] + ' Face', img)
if cv2.waitKey(25) & 0xFF == ord('q'):
    cv2.destroyAllWindows()
    cap.release()
    break

if cv2.waitKey(25) & 0xFF == ord('h'):
    idx += 1
    if idx == len(faces):
        print(solve(cubestr(data)))
        cv2.destroyAllWindows()
        cap.release()
        break

screen_record()

```

8. RESULT ANALYSIS

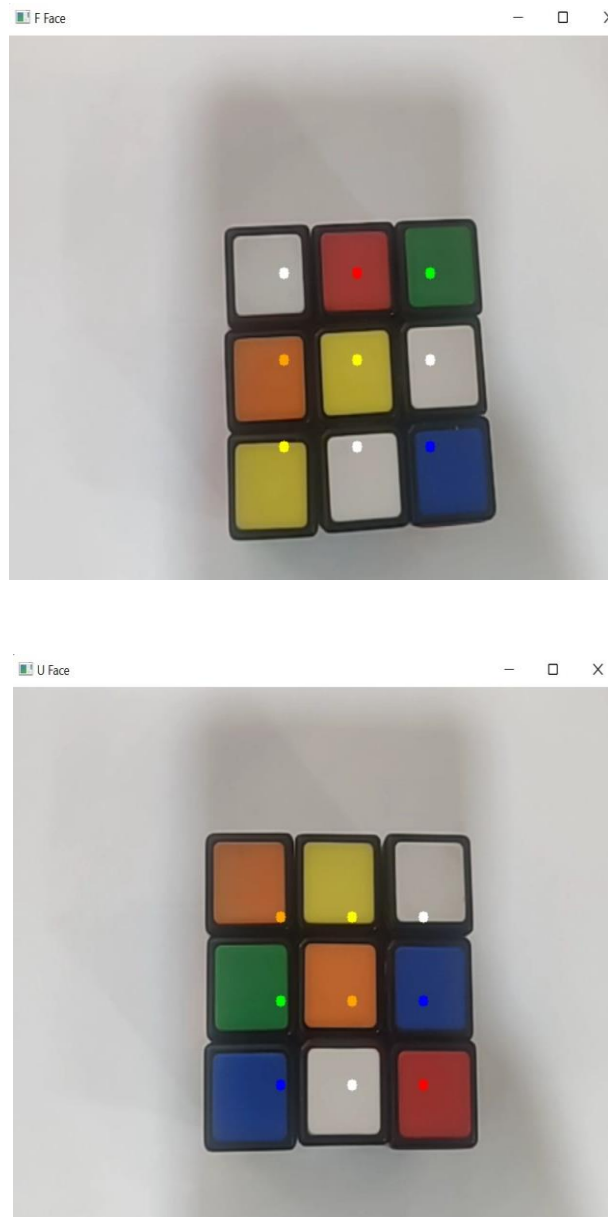


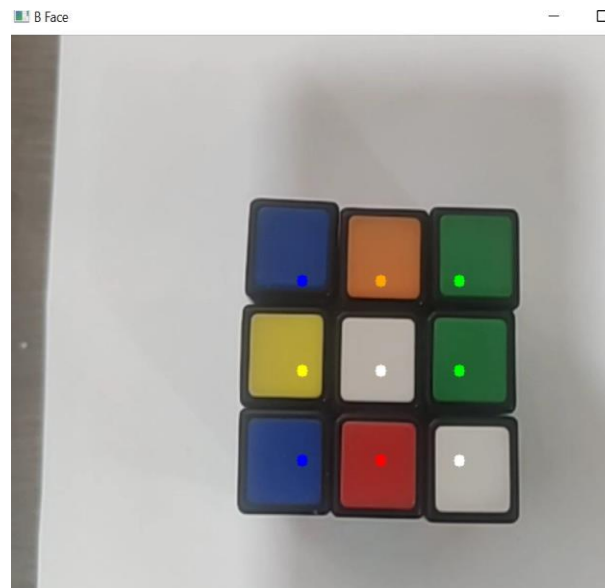
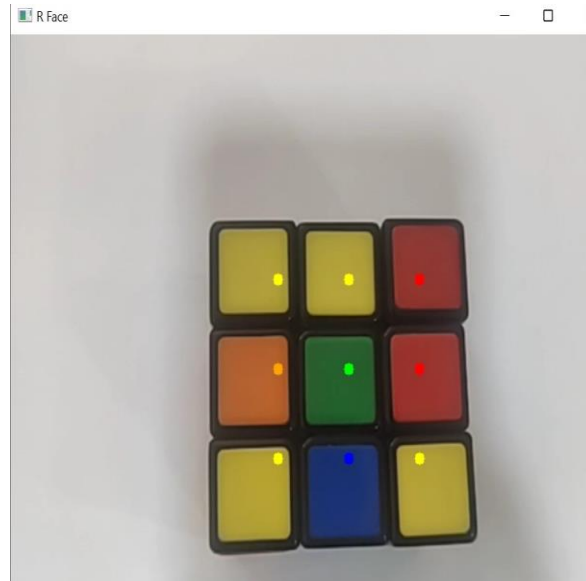
Figure 8.1 : Detecting faces of the cube

D Face



L Face





```
[Running] python -u "d:\mini\newcube.py"
```

```
R B U L B R2 F B' R B D B2 U F2 U B2 L2 D2 B2 R2 D' F2 D'
```


9. CONCLUSION AND FUTURE SCOPE

- Image processing based on proper predictions may accurately work out the blobs of the cube and so totally delivers the cube's original state. Finally, using Koceimba's technique yields the Rubik's cube solution, which can then be used to teach users on how to solve the cube appropriately.
- • A control system design is predicted to solve the Rubik's cube. Algorithm, open CV, and image processing are all used. It assists the user in resolving the jumbled Rubik's cube. The user can solve a Rubik's cube in the fewest steps possible. The Rubik's cube can be solved without the assistance of a machine and by the user's own hands.
- The above implemented project we can add some visual representation so we can change the outlook of the project instead of just displaying the sequences of steps to solve the cube.
- We can work on a program which generates a sequence of cube rotations learned from a deep neural network, solving a scrambled Rubik's Cube instead of directly using the predefined algorithms.

10. REFERENCES

A. Journals/Articles

E. Barucija, A. Akagic, S. Ribic and Z. Juric, "Two approaches in solving Rubik's cube with Hardware-Software Co-design," 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), 2020, pp. 128-133, doi: 10.23919/MIPRO48935.2020.9245201.

B. Books

- Learning OpenCV 3: Computer Vision in Python3 with the OpenCV Library .
- Python Cookbook: Recipes for Mastering Python 3 (3rd Edition)

C. E-websites / downloads

<https://pypi.org/project/opencv-python/>

<https://pyimagesearch.com/2021/04/28/opencv-color-spaces-cv2-cvtColor/>

<https://ruwix.com/the-rubiks-cube/herbert-kociemba-optimal-cube-solver-cube-explorer/>