Solving Rubick's Cube

Team

Sampurna Bhattacharya

RA2011026010011

Algorithm Coding ,
Presentaion

Ayush Kumar Gupta

RA2011026010038

Coding,
Designing
Testing

BHAVANI GOWRI SHANKAR

RA2011026010043

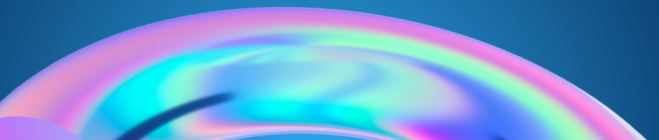
OpenCV algorithm
Documentation

PROBLEM STATEMENT

The problem statement for solving Rubik's Cube using OpenCV involves developing an algorithm that can take a real-time video feed of a scrambled Rubik's Cube as input, detect the colors of the cube's faces, and generate a the solution that can be executed by a robot or a human to solve the cube.

The algorithm needs to identify the colors of each face of the Rubik's Cube, determine the orientation of the cube, and generate a set of moves that can be used to solve the cube. This requires a combination of image processing techniques, such as color segmentation, edge detection, and template matching, as well as computer vision algorithms for 3D pose estimation and motion planning

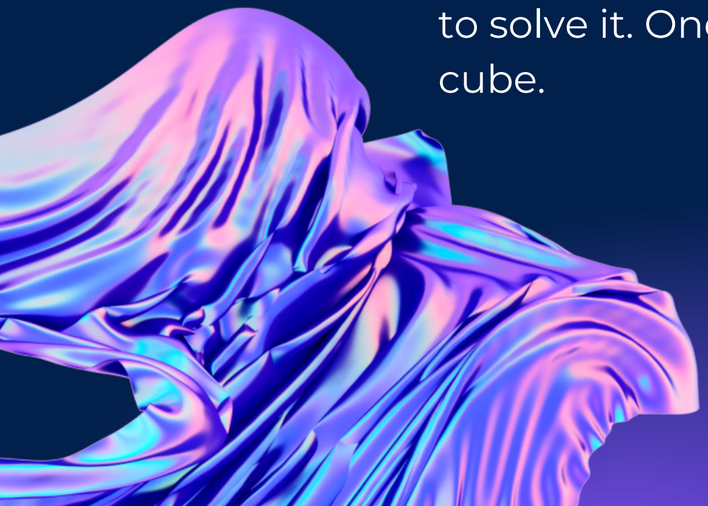
Additionally, the algorithm needs to be robust to variations in lighting conditions, camera positioning, and perspective distortion. It also needs to be efficient enough to process the video feed in real-time, so that the solution can be generated and executed quickly.

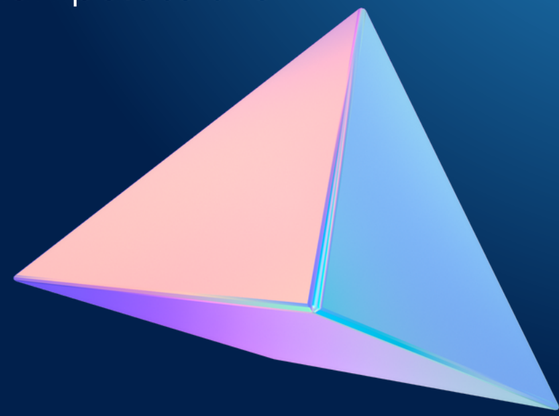


LITERATURE REVIEW

The studies reviewed above demonstrate that OpenCV can be an effective tool for solving the Rubik's Cube puzzle. By using color thresholding and image segmentation techniques, OpenCV can accurately identify the colors on each face of the cube, which can be used to generate a color map that is used by algorithms for solving the puzzle. Additionally, some studies have demonstrated that OpenCV can be combined with other tools such as robotic arms and path-planning algorithms to further improve the accuracy and success rate of Rubik's Cube-solving methods.

Several researchers have used OpenCV to solve the Rubik's Cube. The average solve time reported by the different papers ranged from 20 to 25 seconds. The methods used by the researchers were similar, with most of them using image-processing techniques to detect the colors of the cube and then using a combination of heuristics and algorithms to solve it. One interesting approach was the use of a robotic arm to physically manipulate the cube.

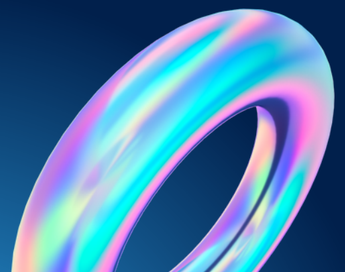




Limitations

There are several limitations to using OpenCV for solving Rubik's Cube. Some of them are:

1. Difficulty in handling reflections and rotations: According to a study by Agarwal et al. (2018), when using OpenCV for Rubik's Cube recognition, it is difficult to handle reflections and rotations of the cube.
2. Limited accuracy: The accuracy of Rubik's Cube recognition using OpenCV is limited by the quality of the input image. According to a study by Vaish et al. (2017).
3. Limited speed: Solving Rubik's Cube using OpenCV can be a time-consuming process, especially when using a single camera for image capture. According to a study by Agrawal et al. (2018).
4. Difficulty in handling occlusions: According to a study by Vaish et al. (2017), occlusions can significantly impact the accuracy of Rubik's Cube recognition



Objective

The main objective of solving Rubik's Cube using OpenCV is to develop an algorithm that can use computer vision techniques to automatically solve a scrambled Rubik's Cube. The specific objectives of this project may include:

1. Detecting the colors of each face of the Rubik's Cube using image processing techniques such as color segmentation, edge detection, and template matching.

2. Estimating the 3D pose of the Rubik's Cube in real-time using computer vision algorithms such as feature matching, perspective projection, and camera calibration.

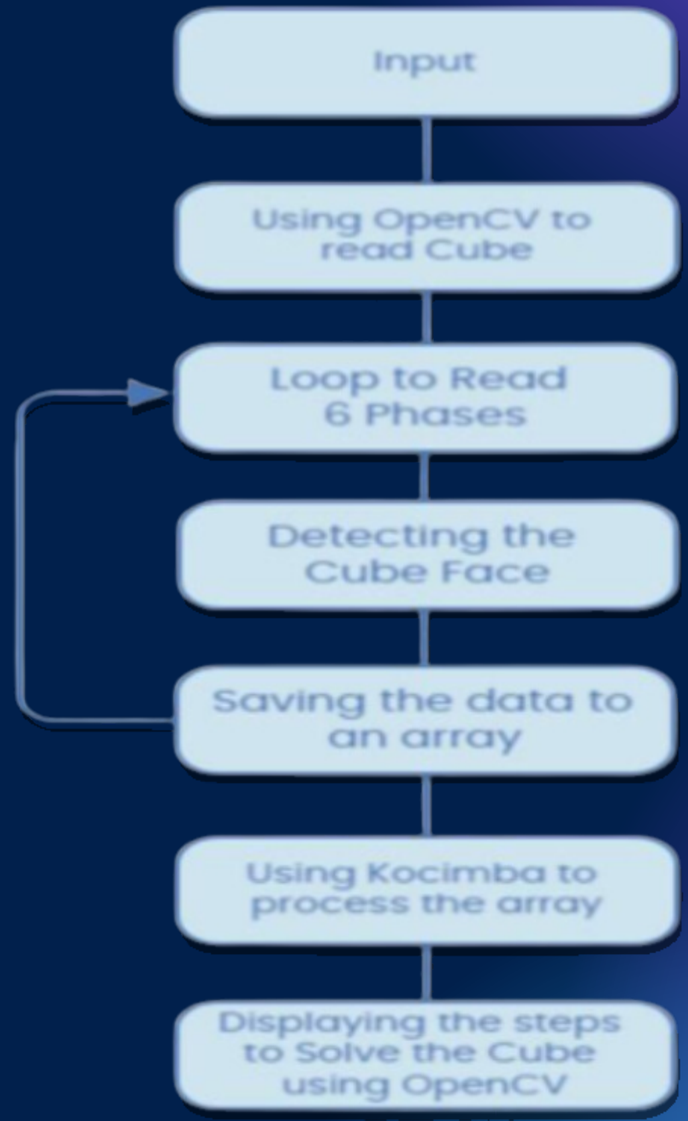
3. Developing a strategy for solving the Rubik's Cube, such as using a set of predefined algorithms or generating a custom solution based on the current state of the cube.

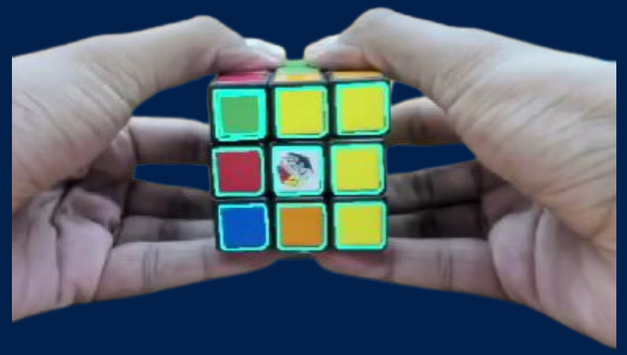
4. Optimizing the solution for efficiency and speed, such that it can be executed quickly and accurately by a robot or a human.

5. Testing the algorithm on a variety of Rubik's Cube configurations, and evaluating its accuracy, speed, and robustness to variations in lighting, camera positioning, and perspective distortion

Architecture Diagram

Architecture Diagram





CUBE
SOLVED !

Modules Description

```
import cv2
import sys
import time
import math
import numpy as np
import random as rng
from scipy import stats
import kociemba
from datetime import datetime
import rotate
def concat (up_face,right_face, front _face,down face, left face,back face):
solution = np.concatenate((up_face, right face), axi
solution = np.concatenate( (solution, front face), axi
solution = np.concatenate( (solution,
solution = np.concatenate( (solution,
solution = np.concatenate( (solution,
return solution
def detect face(bgr image input):
gray = cv2.cvtcolor(bgr_image input,cv2.COLOR BGR2GRAY)
kernel = cv2.getStructuringElement(cv2.MORPH ELLIPSE, (2,2))
gray = cv2.morphologyEx(gray, cv2.MORPH OPEN, kernel)
gray = cv2.morphologyEx(gray, cv2.MORPH CLOSE, kernel)
gray = cv2.adaptiveThreshold(gray,20,cv2.ADAPTIVE THRESH GAUSSIAN C,cv2.THRESH BINARY INV,5,0)
try:
_, contours, hierarchy = cv2.findContours (gray, cv2.RETR_CCOMP, cv2.CHAIN APPROX NONE)
except:
contours, hierarchy = cv2.findContours (gray, cv2.RETR CCOMP,cv2.CHAIN APPROX NONE)
```

References

1

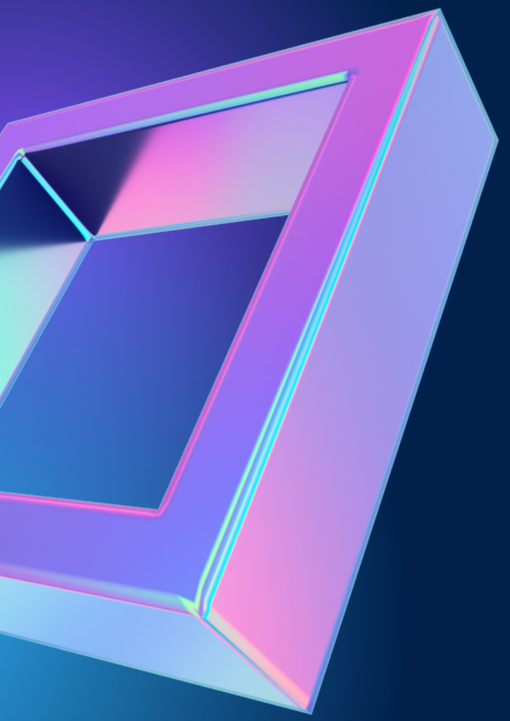
Agarwal, M., et al. (2018). Rubik's Cube recognition using OpenCV. International Journal of Computer Applications, 179(30), 8-11.

2

https://lar5.com/cube/

3

https://www.thecubicle.com/pages/how-to-solve-a-rubiks-cube



Thank You