RUBIK'S CUBE SOLVER BOT

MASTER'S THESIS REPORT

FOR THE DEGREE OF

MASTER OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

BY

AMRITESH SINGH IRM2013020

UNDER THE SUPERVISION OF
Dr. PAVAN CHAKRABORTY
IIIT-ALLAHABAD

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
(A UNIVERSITY ESTABLISHED UNDER SEC.3 OF UGC ACT, 1956 VIDE NOTIFICATION NO. F.9-4/99-U.3 DATED 04.08.2000 OF THE GOVT. OF INDIA)

A CENTRE OF EXCELLENCE IN INFORMATION TECHNOLOGY ESTABLISHED BY GOVT. OF INDIA

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled "RUBIK'S CUBE SOLVER BOT", submitted towards the fulfilment of the MASTER'S THESIS of M.Tech. (IT) at Indian Institute of Information Technology, Allahabad, is an authenticated record of my original work carried out under the guidance of Dr. Pavan Chakraborty. Due acknowledgements have been made in the text to all other material used. The project was completed in full compliance with the requirements and constraints of the prescribed curriculum.

Place : Allahabad
Date: / /

AMRITESH SINGH (IRM2013020) :

CERTIFICATE FROM SUPERVISOR

This is to certify that the declaration made by the candidate is correct to the best of my knowledge and belief. The project titled ”RUBIK’S CUBE SOLVER BOT” is a record of the candidate’s work carried out by him under my guidance and supervision. I do hereby recommend that it should be accepted in the fulfilment of the requirements of the MASTER’S THESIS at Indian Institute of Information Technology, Allahabad.

Date: / /

(Dr. PAVAN CHAKRABORTY)

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompanies the successful completion of any project work would be impossible without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts. This thesis was not only an endeavour but also an interesting learning experience for me and it bears the imprint of a number of people who directly or indirectly were a great source of help and constant encouragement.

I would like to express my sincere thanks to my mentor Prof. Pavan Chakraborty for his continuous motivation and guidance. His valuable suggestions, comments and support were an immense help for me. I am grateful to him for taking out time from his busy schedule and being very supportive in guiding my work.

Special thanks to my parents and my peers Utkarsh Kumar and Saptak Sengupta for their efforts and cooperation. The interesting and informative discussions we had together greatly contributes to the completion of this work.

Amritesh Singh

ABSTRACT

The Rubik's cube is a three-dimensional combination puzzle. Though several different variations of the cube have been invented differing in both number of layers and their shape, the classical Rubik's cube has 6 faces with each face containing 9 squares, each square having one of the 6 colours - yellow, white, blue, red, green, and orange. The goal is to solve the cube such that all the squares on each face have the same colour. In order to solve it manually, there are various strategies based on the position of the squares and colour with respect to the other squares that can be used to make a sequence of moves.

However, except for the professionals, the steps are not always perfect to get to the final result. A Rubik's cube can be solved in a considerably more efficient manner using algorithms on a computer system. The approach taken by a computer to solve a Rubik's cube is to first find out the exact steps to solve the Rubik's cube given a particular state of the cube. But it can never really execute those steps. Therefore, the main idea behind this thesis is to design a proper algorithm and to perform the steps using a mechanical robot to operate upon an actual Rubik's cube.

Contents

Chapter 1

Introduction

There exists a significant body of previous work to compute the exact steps for solving a Rubik's cube. A Rubik's cube has some standard properties. Each square on a face of a Rubik's cube can consist of one of only 6 solid colours - yellow, white, blue, red, green, and orange. There is also a fixed pattern in which these colours should appear in a solved Rubiks cube. The pattern resembles this: white is opposite to yellow, blue is opposite to green, red is opposite to orange, and red, white and blue appear in a clockwise order. Each face of the Rubik's cube can be rotated individually independent of the other faces. The middle square on a face is always in the correct position and can be used as a reference to understand the colour that each face should have once solved.

There are 4.3252 x 10$\hat{1}$9 different states that can be reached from any scrambled configuration or state of the cube. There are 43,252,003,274,489,856,000 (approximately 43 quintillion) ways to scramble a Rubik's cube.[1] To solve the cube, one needs a general strategy which consists of a sequence of moves or operations involving the rotating of the sides of the cubes. The standard way of representing each move is F, F', R, R', B, B', L, L', U, U', D and D'.[23] The physical representation of each of these moves is shown in figure 1.1.

Several previous works have given forward papers on the different algorithms to solve the Rubik's cube. The general idea is to use a heuristic to reach the final state and then use different techniques like iterative deepening to find out the final solution. One can use a pattern database along with the iterative deepening to get a the result in a median of 18 moves.[2] The number of steps required to solve any scrambled cube has been varied over time. This number is also often referred to as the God's number. At the very beginning it was considered to be 29.[18] The current top steps required is 26 but it requires a large amount of resources to create these 26 steps since it uses a lot of brute force mechanism.[3]

After the solution has been found, the bot needs to perform those steps to solve the Rubik's cube. For this we need to construct a mechanical model for the bot.

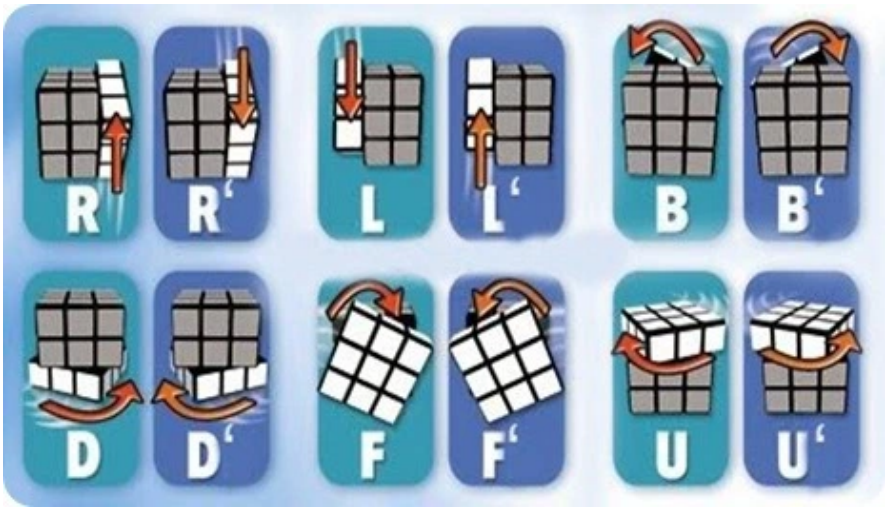


Figure 1.1: Different moves available on Rubik's cube

Work has been done on the modelling of a bot to rotate the Rubik's cube as well. The one with the most efficient mechanical design to solve a Rubik's cube is the Cubestormer[4] created by Mike Dobson and David Gilday which can solve the Rubik's cube in a record 3.25 seconds. However the resources required to make this mechanical design is very complex and costly and requires a lot of infrastructure to be built.

The aim here on the other hand is to make a mechanical design which can solve the bot in a comparatively greater time but using very less resources that can be created at a very low cost. So the main inspiration behind the mechanical design is from the Tilted Twister[5] by Hans Andersson. It uses a very minimalist and mechanical approach. It makes use of just 2 servo motors to perform all the movements.

Chapter 2

Previous Work

2.1 Image Processing

There can be different ways of identifying the exact colours of the various cells on each face of a Rubik's cube and then reconstructing the Rubik's cube to 3 dimensions.

An erosion and dilation technique has been proposed in research papers to separate each cell from the other and form a proper separation between them.[7] This way the internal homogeneous parts also remain flooded with the colour.

OpenCV can be used for masking and separately identifying the colours of each of the cells.[8]

2.2 Cube Solving Algorithm

There are several approaches to solving the Rubik's cube:

One of the earliest solutions for the cube was proposed by David Singmaster, solving the cube layer by layer. A modification of this method, called the CFOP method, was proposed by Jessica Fridrich and involves more pattern matching and more algorithms than the layer by layer method.

A pattern database has been used along with iterative deepening to get a result in a median of 18 moves.[2] This was done using an iterative deepening algorithm with a table containing the various different nodes or patterns.

There is also a machine learning approach available in which a program learns the scrambled state to end state steps of a large number of scenarios and computes a heuristic based on the learning.[6]

A brute force technique can be used with a high powered CPU to get great and

assured results for solving the cube.[3]

2.3 Mechanical Design

There are different mechanical designs which use different techniques to rotate the different faces of the Rubik's cube:

Cubot uses two orthogonal manipulators to grip the cube and then to perform the various moves that are computed by the algorithm.[9]

Figure 2.1: Cubot

Cubestormer, on the other hand, uses 4 mechanical arms which are controlled by 8 Lego bricks which control the motor sequencing.[10]

Figure 2.2: Cubestormer

However, the most cost effective design is the Tilted Twister which uses just two servo motors and support system to both hold and rotate the cube's faces.[5]

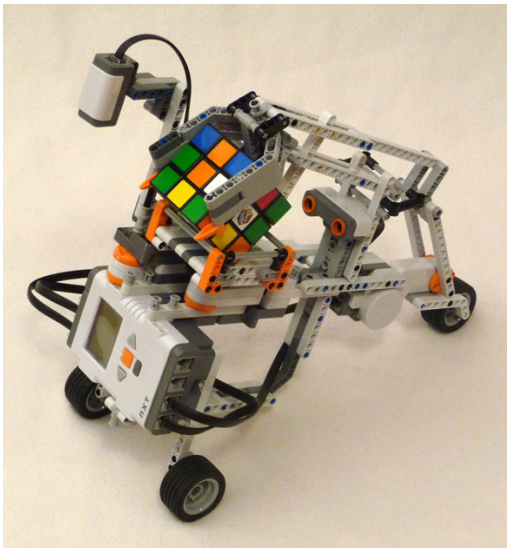


Figure 2.3: Tilted Twister

A dual arm system has also been designed for the purpose of manipulating a Rubik's cube.[21]

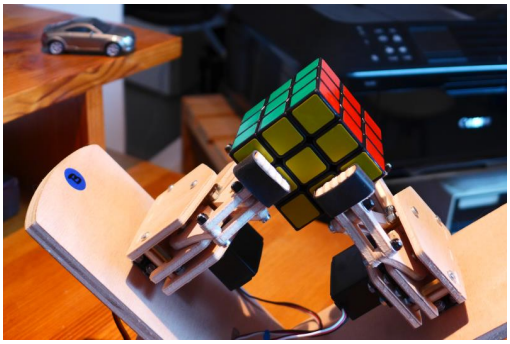


Figure 2.4: Solver with dual arms

Chapter 3

Problem Statement

3.1 Motivation

Rubik's cube has been one of the most popular and famous puzzles for a long time and a lot of people try to solve it using various methods. There are many different algorithms available for solving it as well. But compared to the number of algorithms, there are not many mechanical designs available to test the solutions of that result and it is mostly done manually or computed mathematically without seeing the actual result. The ones with a mechanical design have the code and algorithm for solving infused in them so they can't be used for testing other algorithms. So the idea is to make a bot which is simple and cost effective and can be used to test any algorithm that is pushed to some microcontroller.

The code and the mechanical design remain relatively decoupled from each other so that changing one does not affect the other too much. Also we use colour sensors and image processing to detect the state of the Rubik's cube so that the user does not have to manually enter the state of the Rubik's cube for testing an algorithm every time. Therefore, this would not be just another Rubik's cube solving robot but also a bot that can be used for testing purposes since a really essential part of algorithm development is testing for the correctness and effectiveness of it.

3.2 Problem Definition

To design a mechanical robot that is capable of bringing a Rubik's cube in a scrambled initial state to a final solved state.

The work consists of 3 basic modules:

- Cube colours detection (Image processing)
- Cube solving algorithm
- Mechanical Design

3.3 Objectives

In this work, we aim to construct a Rubik's cube solving bot that is both mechanically simple and uses the least possible amount of resources to make but that is also effective in solving all the steps. To the end of achieving this goal, only two servo motors are used along with a support system that consists of only an arm and a platform.

The colour detection of the cube using image processing is done with the aim of removing the tedious process of manually entering the colour in each cell in each face of the cube, thus saving a considerable amount of time in reading the cube.

The algorithm should lay out the steps for solving the cube using the standard cube notations, so that the user can solve the cube manually too if needed.

Chapter 4

Cube Colours Detection

4.1 Overview

In order to detect the colour of each cell on each face in a Rubik's cube, the cube is first placed on the platform component of the robot, directly underneath the external webcam fitted on a stand. The cube is pushed and rotated by the arm such that the webcam is able to capture each face of the cube. Subsequently, image processing is performed on each captured image in order to detect the cell colours on each face. OpenCV is used for this purpose.

4.2 Canny Edge Detection

Edge detection is a technique used in image processing to find the boundaries of objects within images. It functions by performing the detection of discontinuities in brightness in the image. Canny, Prewitt, Sobel, Roberts edge detection are some common edge detection techniques.[20]

In order to apply the canny edge detection process on an image, the image is first converted to a grayscale image from a truecolour (RGB) image.

The canny edge detection algorithm comprises of the following steps :

- The application of a Gaussian filter is done to reduce noise in the image and remove the fine details by blurring the image.
- The image's intensity gradients are found.
- In order to remove all inaccurate results obtained after applying edge detection, non-maximum suppression is applied.
- Double thresholding is performed to resolve potential edges.
- Hysteresis is applied in order to track the edge. The detection of edges is finalized by suppressing all the other edges not connected to strong edges, that is, the

edges that are considered to be weak.[11]

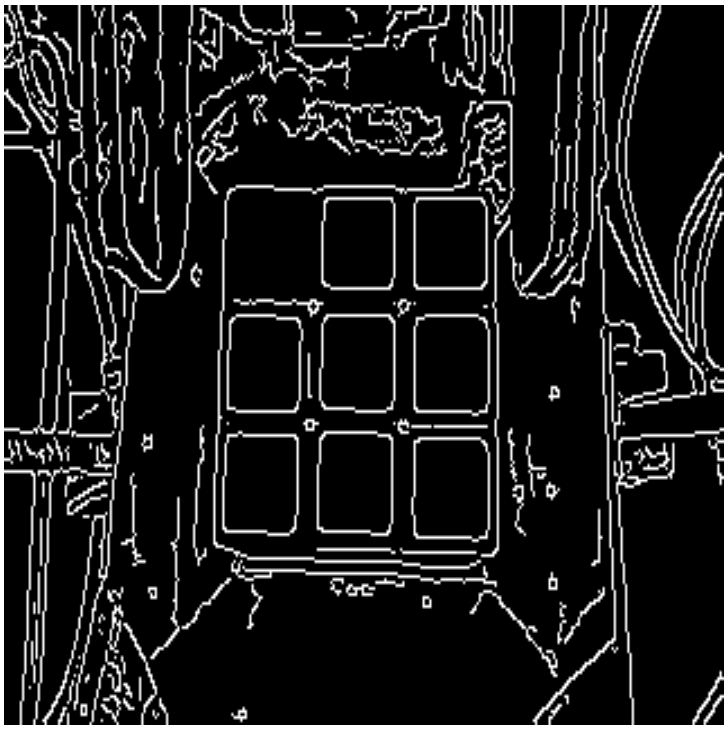


Figure 4.1: Image after applying Canny edge detection

4.3 Dilation

The image is dilated to enhance the foreground areas of the pixels in order to make the boundaries even more prominent than the one obtained after canny edge detection.

Dilation is a highly popular image processing technique and an operation of mathematical morphology, the other being erosion.[12] In this, the output pixel's value is the greatest value of all the pixels that occur in the neighbourhood of the input pixel. For the case of a binary image, the output pixel is set to 1 when any of the neighbourhood pixels are already set to the value 1.[22]

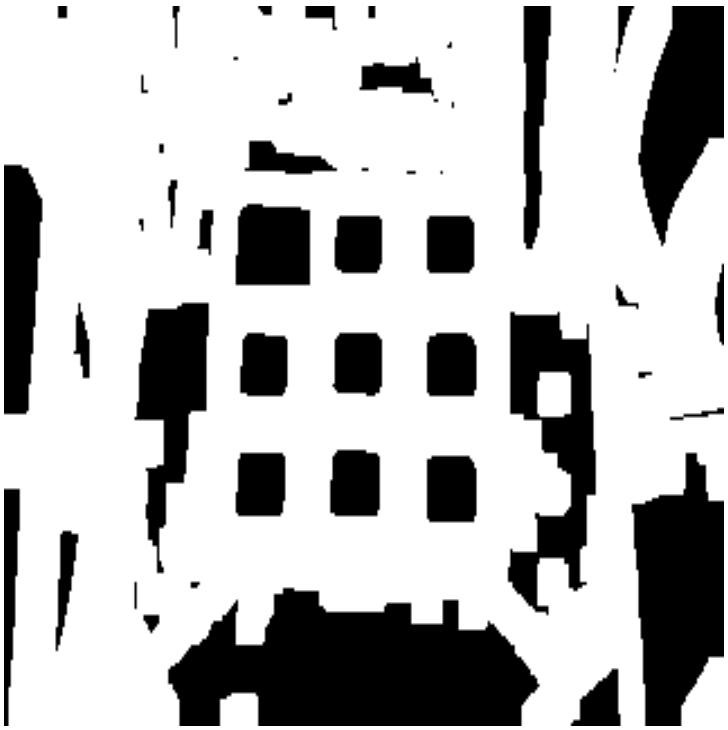


Figure 4.2: Image after applying dilation

4.4 Contour Detection

Contours are curves that join all the continuous points that lie along the boundary, that possess the same intensity or colour. They are helpful tools that are used for object detection and recognition and shape analysis.

A function for finding contours is available in OpenCV, that takes three arguments - the source image, a mode for contour retrieval, and a method for contour approximation. As output, the function gives a modified image, the contours, and hierarchy. These contours are Numpy arrays of coordinates of the object's boundary points.

4.5 Contour Approximation

Each contour obtained is then approximated by removing vertices that are not needed, to obtain a shape that is basically similar to the source image, with the precision varying depending on the requirements. After approximating all ignorable edges, a polygon-like contour area is obtained. This is an implementation of the Douglas-Peucker algorithm.[13]

The Douglas-Peucker algorithm, also known as the iterative end-point fit algorithm, has found widespread use in several fields such as road detection.[14] The algorithm works by dividing the lone recursively, automatically marking the first and last points in the line to be kept. Considering the first point and the last point as the end points, the algorithm then finds the point at maximum distance from the line segment, the point thus obtained is observed as being at the greatest distance on the curve from the approximating line segment that lies between the end points. Epsilon is the largest distance between the contour and the approximated contour. Any points not marked to be kept at present can be removed without the simplified curve being worse than epsilon, in the case when the point is closer to the line segment than epsilon. The lower the value of epsilon, the greater is the precision of the approximated shape. If the point that is at maximum distance from the line segment is greater than epsilon, then that point is kept. The recursive calls keep occurring with the first point and the furthest point and then with the furthest point and the last point, which includes the furthest point being marked as kept. Upon completion, a new output curve is generated consisting of the points that were marked as having been kept.[15][16]

The cells on each face are square in face and thus have 4 corner x-coordinates and 4 corner y-coordinates, thus giving a total of 8 corner coordinates. Thus, the approximated contour array's length should be 8. A range is set for the area of the contour and this is the range in which the area of each cell lies. Each contour is then checked whether it contains 4 sides or not. In the event of both the mentioned conditions being satisfied, the position of a cell is said to have been determined.

4.6 Colour Detection

The position of the top-left pixel of each cell obtained after contour approximation is determined. Using these positions, the position of the cell is easily determinable in terms of where it lies in each row of 3 cells and each column of 3 cells. But there is a problem with this approach: there are 9 x-coordinates and 9 y-coordinates remaining. Subsequently, with the aim of removing removing some of these coordinates, the difference between a point's coordinate (both x-coordinates and y-coordinates) and the next point's coordinate is checked. If this value lies above a certain threshold, it is considered a cluster point and it is possible that this lies on the next row or column level. This threshold represents the distance between the two points. After finding all the cluster points, there remain 3 x-coordinates and 3 y-coordinates, and thus their combination is used to yield the positions of the cells with respect to the

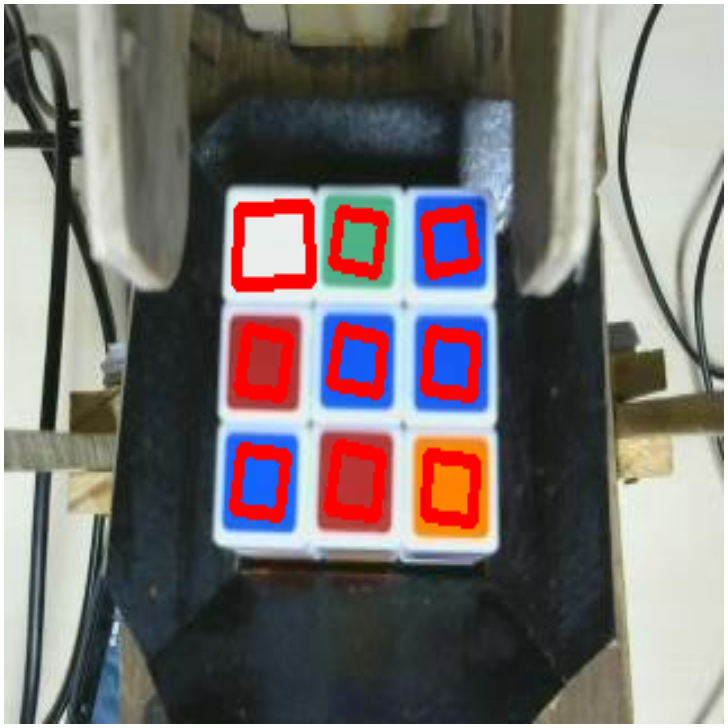


Figure 4.3: Image after detection and polygonal approximation of contours

other cells on that face.

A small area on each cell is considered, that is, all points along both the x-axis and y-axis in that are are considered, and Numpy mean is calculated for that area. The RGB ranges for each of the six colours - yellow, white, blue, red, green, and orange - are specified and if the RGB value obtained after the Numpy mean operation lies in that range, then that colour is determined to be the colour of the cell.

Chapter 5

Cube Solving Algorithm

For the purpose of solving the Rubik's cube, the layer by layer method of solving the cube has been used. This is the method used by most beginners in order to solve the cube.[17] The steps involved in solving the cube using this method are:

- Solving the cross.
- Solving the first two layers.
- Orienting the last layer.
- Permuting the last layer.[23]

This method is preferred by most speedcubers due to its heavy reliance on pattern recognition, muscle memory, and algorithms.

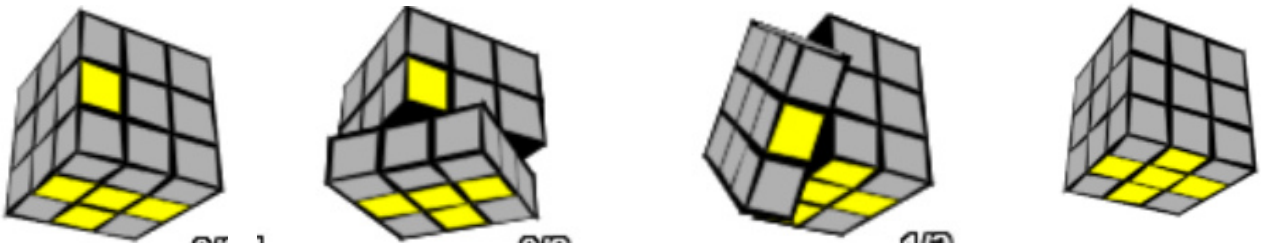
Solving the cross

All the moves in the algorithm used to solve the cube are taking the yellow face as the front and the blue face as the top of the cube. We start by solving the yellow cross, until we arrive at one of two possibilities.

There are two cases where cross needs to be fixed:

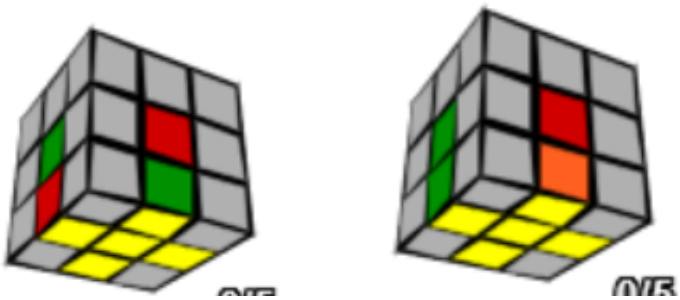
- Here, the face with the yellow cross piece is simply rotated till it is at its correct position.

- Here, the face with the yellow cross piece first needs to be brought to a position that matches the previous case. Afterwards, the face is simply rotated till the cross piece comes to rest on the yellow face.



Solving the first layer

After the yellow cross has been solved, the yellow face is rotated till any two of the cross pieces are able to assume their proper positions. The first layer of the Rubik's cube is solved by swapping the incorrect yellow cross pieces. Afterwards, a sequence of moves is implemented to swap the two cross pieces so that they are placed at their accurate positions. This involves two cases: when the two incorrect pieces are next to each other, and when they are on the opposite faces. To solve the two cases, one of the incorrect piece is moved to the last layer in order to be moved independently from the other piece. Afterwards, rotation is done to move the piece to its correct position and rotated back to its original layer. Both cases use a similar strategy. In the case when all four corners are already at their proper positions, no further action needs to be taken.



After the cross pieces have been solved, the four yellow corner pieces need to be solved. There are three cases for solving of corner pieces:

- This involves a sequence of three moves R U R' which brings the corner piece to its proper position.

- This case is simply a mirror image of the previous case and thus involves mirror image moves of the previous case.

- In this case, the cube is solved using the sequence of moves R U' R' U U so that it comes to rest in an orientation that is similar to either one of the previous two possible cases.

If a corner is already inserted but with the incorrect orientation, the cube is first solved to match one of the previous three cases. Subsequently, each yellow corner piece is brought to its correct position, thus resulting in the solving of the first layer.

Solving the second layer

The solution of the second layer involves correctly orienting the non-white edge pieces so that they are able to assume their correct positions in the Rubik's cube.

Three cases are possible in which the edge pieces may occur in the cube:

- This involves a sequence of eight moves U R U' R' U' F' U F that brings the edge piece to its correct position.

- This case is a mirror image of the previous case and thus we need to use moves that are mirror images of the moves used in the previous case.

- In this case, the corner is twisted in four moves so that it's similar to either one of the previous two cases. Then it can be solved accordingly.

Thus the edge pieces are brought to rest in their final orientations, and as a result the first two layers of the cube are solved.

Orienting the last layer

The next step involves correctly orienting the last layer of the Rubik's cube. For this, first the white edge pieces are made to face in a way so that a white cross is being formed on the white piece.

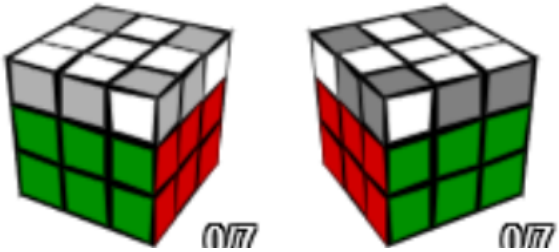
There are three cases possible as follows:

- Only one edge piece that is not on the white face is visible.

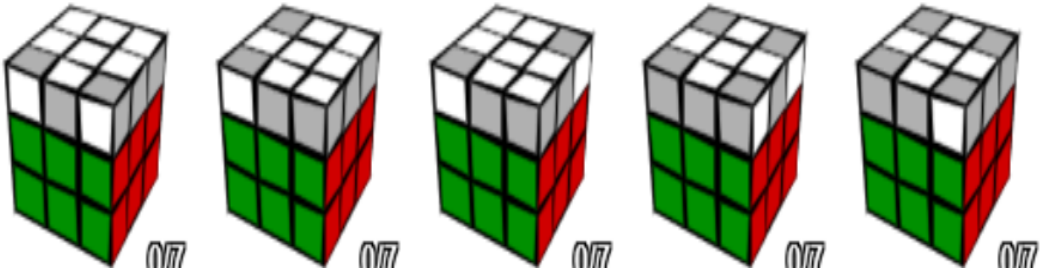
- One edge piece not on the white face and two edge pieces that are on the white face are visible. The edge pieces on the white face are in the same layer.

- One edge piece not on the white face and two edge pieces that are on the white face are visible. The edge pieces on the white face are in different layers of the cube.

After a sequence of six moves R' U' F' U F R that are the same for all the above three cases, the white cross is made. Now, the next step is to orient the last layer so as to solve the complete white face. When there are three corners to be twisted, two cases are possible, both mirror images of each other. The solution for this involves eight moves that are mirror images of each other. For the first case, these moves are R U R' U R U U R'.



In all other cases, we apply the first case as shown to bring the cube into a case that is the same as either of the two previous cases. Thus, the white face is solved and the last layer has been oriented.



Permuting the last layer

The last step for solving the cube involves the permutation of the last layer so that each piece is at its correct location. First, the corner pieces are solved.

A total of four cases are possible in order to solve the four corner pieces:

- The first case is solved using a sequence of twelve moves R' F R' B B R F' R' B B R R.

- This case is a mirror image of the previous one and thus mirror image moves are used here.

- The white face is rotated till a match occurs for either of the first two cases.

- In this case, the same sequence of moves used in the first case are applied to obtain a cube that matches one of the previous three cases.

Finally, the white edge pieces are solved. There are three cases possible listed as following:

- If there are exactly three wrong edges and they need to be rotated in a clockwise direction, then the cube is held so that those three edges are as shown, and two steps are followed to solve this. First, the corners are twisted in clockwise direction using the sequence of moves R U R' U R U U R'. Then, taking the mirror image case, the corners are twisted back to normal.

- If there are exactly three wrong edges and they need to be rotated in an anti-clockwise direction, then the cube is held so that those three edges are as shown, and two steps are followed to solve this. First, the corners are twisted in anti-clockwise direction. Then, taking the mirror image case, the corners are twisted back to normal. Therefore, this is a mirror image of the previous case.

- When all four edges are incorrectly placed, the solution for the first case is applied.

  This results in the complete solution of the cube.

Chapter 6

Mechanical Design

6.1 Tools Used

HARDWARE

For controlling of motor functions

- Arduino Uno
- Servo motors (x2)
- Connecting wires
- Power supply

For construction of basic framework

- Mechanix kit
- Pinewood
- Plywood board
- Glue gun

For cube colours detection

- External webcam

SOFTWARE

Coding Environment

- Fedora
- Pycharm

- Arduino IDE

Programming Languages

- C++ (for Arduino)
- Python 3

Python Modules

- OpenCV
- Pyserial

6.2 Basic Framework Construction

The bot consists of two components basically: an arm for pushing or holding the cube and platform for rotating the cube, each of which is attached to a servo motor that controls the movements of each of these components. A stand is erected between the platform and the arm on the axis along which the movement of the arm takes place. A steel rod is placed on the stand which supports the arm's movement.

Figure 6.1: (a) Arm which pushes or holds cube. (b) Platform on which cube is mounted.

The free end of the arm consists of a mechanism for holding the cube (cube holder) and another at the bottom of that end that pushes the cube (cube pusher).

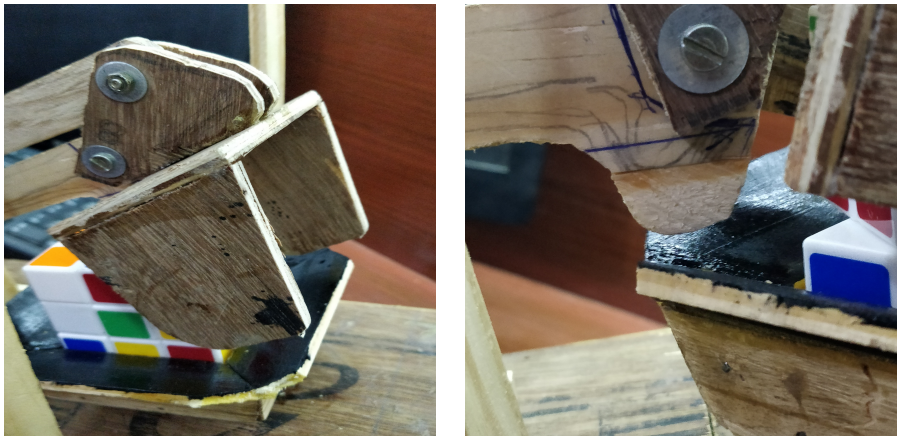


Figure 6.2: Mechanism for (a) holding the cube, and (b) pushing the cube.

The platform is hollowed out so that the cube can be placed in it. The upper boundaries of the platform are fitted with slanting sticks in order to prevent snagging while a sheet of sunmica is fitted on the inside surface of the platform, smoothening it and allowing the cube to slide freely along it.

The complete framework of the bot is mounted on a plywood tray with the arm and the platform components mounted above it, while the two servo motors and the Arduino board are mounted below it, with slots for guiding and supporting respectively of their corresponding components cut out in the tray.

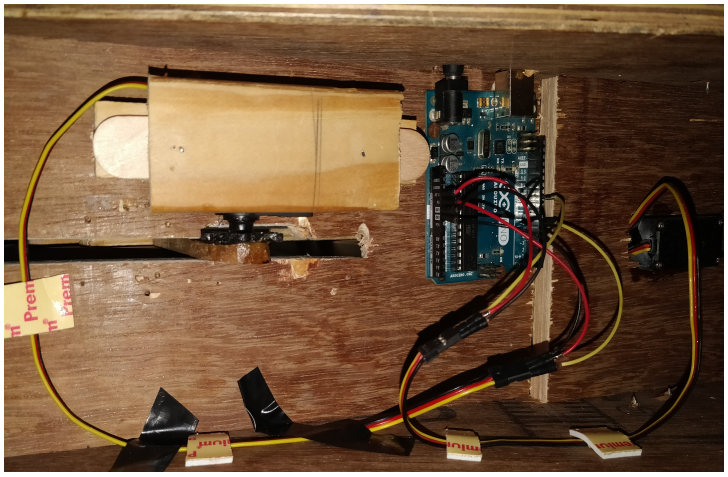


Figure 6.3: Arduino and servo motors mounted beneath the tray

The external webcam is fitted directly above the cube on a stand. The movement of the webcam is free along the length of the stand. The webcam can also

be partially rotated in order to accurately adjust its position for the purpose of cube detection.

Figure 6.4: External webcam

6.3 Arduino Connections

Arduino Uno, a microcontroller board that is based on the ATmega328P, comprises of 14 digital pins for input and output, 6 analog inputs, a power jack, a USB connection, a 16 MHz quartz crystal, a reset header and an ICSP header. It has been used in the model for the purpose of controlling the rotation of the shafts of the servo motors in order to accurately solve the scrambled Rubik's cube.

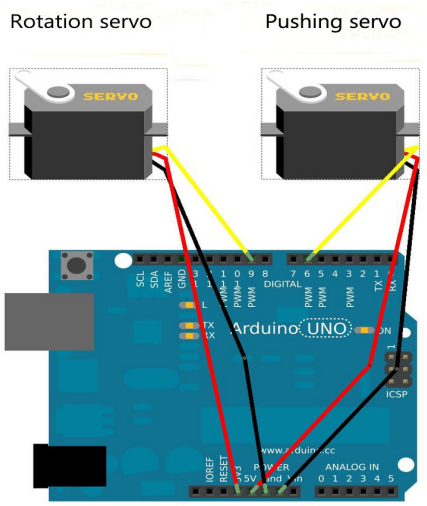


Figure 6.5: Connections between servo motors and Arduino

The connections between the Arduino and the servo motors have been done in the following manner:

- The yellow (signal) wire of the pushing servo is connected to input/output pin 6 on the Arduino board.
- The red (power) wire of the pushing servo is connected to the 5V pin on the Arduino board.
- The black (ground) wire of the pushing servo is connected to the GND pin on the Arduino board.
- The yellow (signal) wire of the rotation servo is connected to input/output pin 9 on the Arduino board.
- The red (power) wire of the rotation servo is connected to the 3.3V power source pin on the Arduino board.
- The black (ground) wire of the rotation servo is connected to the GND pin on the Arduino board.

6.4 Functional Methodology

The arm component of the bot is used to push or hold the Rubik's cube as per the requirement. The platform component of the bot is used to rotate the cube, which is placed inside it. The stand in the middle consists of a steel rod on which the arm comes to rest when it is being used to hold the cube using the cube holder fitted at its end. When the arm is used to push the cube, it moves along the steel rod and rises up eventually, at which point the cube pusher at the same end of the arm comes in contact with the cube and pushes it so that the cube tilts and falls on the face opposite to the face which came into contact. The platform rotates either the whole cube or just the down face of the cube depending on the conditions.

The pushing servo of the cube can perform three operations:

- **Push:** The arm comes forward and pushes the cube.
- **Hold:** The arm holds the cube along the two layers that form the top of the cube.
- **Release:** The arm remains or comes back to its initial state and the all the three layers of the cube are free as a result.

The rotation servo of the cube is capable of performing one operation:

- **Rotate:** The platform rotates along the shaft and as a result, the cube placed in it rotates too.

The combination of the operations of the pushing and rotation servos results in three basic operations for the bot:

- **Push:** The Push operation is done in isolation while the platform remain stationary. This is responsible for changing the cube face currently under operation.

Figure 6.6: Push operation on cube

- **Release and Rotate:** The Release operation of the arm is done in conjunction with the Rotate operation of the platform, resulting in the rotation of the whole cube.

Figure 6.7: Hold and Rotate operation on cube

- **Hold and Rotate:** The Hold operation of the arm is done in conjunction with the Rotate operation of the platform. This results in the rotation of just the bottom face of the cube.

All the basic moves of a cube, i.e., F, F', U, U', L, L', R, R', D, D', B, and B', can be then translated to a sequence of moves that involves bringing the face to be rotated to the bottom of the cube. For instance, if we need to perform the move F, the cube is first subjected to pushing and rotation operations that involve bringing the face that was at the front to the bottom of the cube and then rotating the platform in anti-clockwise direction. Afterwards, the cube is brought to its original orientation.

Chapter 7

Results

The cube state obtained as the output of the image processing module will be used as the input for the cube solving algorithm module, in order to obtain the set of moves that will then be fed to the mechanical design through Arduino.

For the purpose of running the code, the Arduino sketch is uploaded to the microcontroller board. Then the python file is run which captures video through the external webcam. After each face of the cube has been captured, the raw string generated of the cube faces is then sent to the Arduino where it is parsed and subsequently the moves required are carried out.

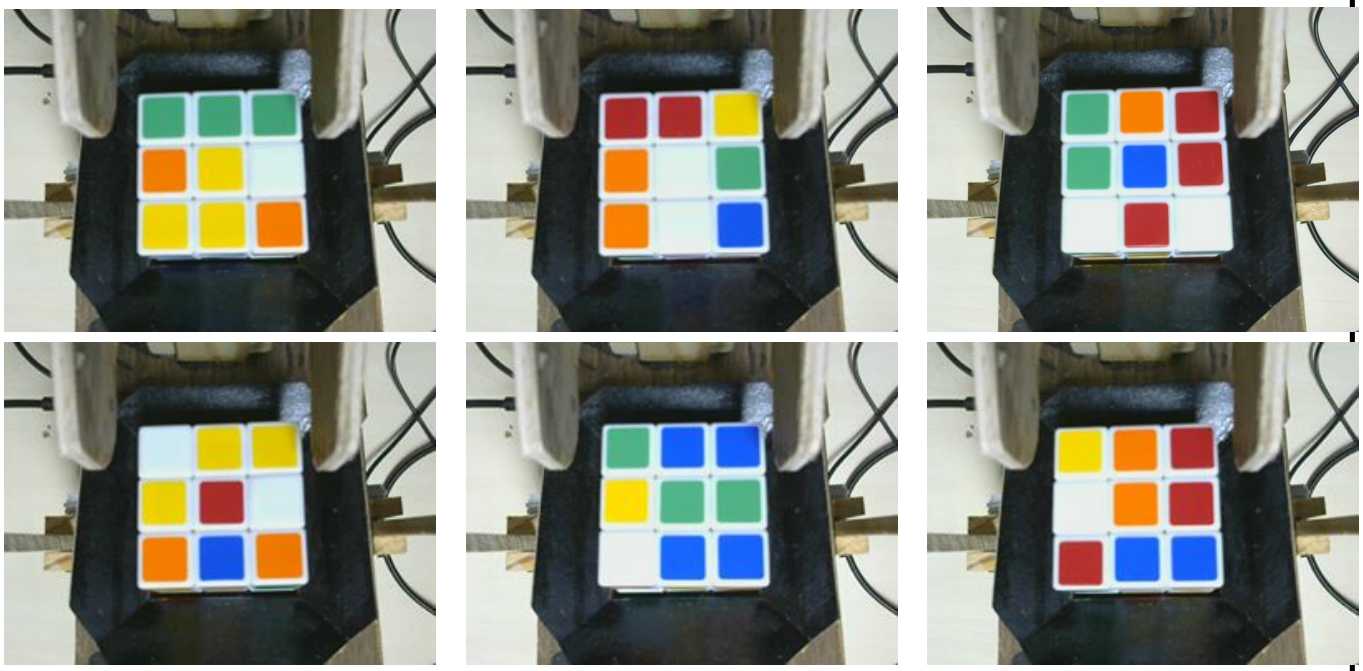


Figure 7.1: Cube faces captured through webcam

The cube faces are accurately being processed through the image processing operations applied on them and accurately generating the colours to be stored in the raw string.

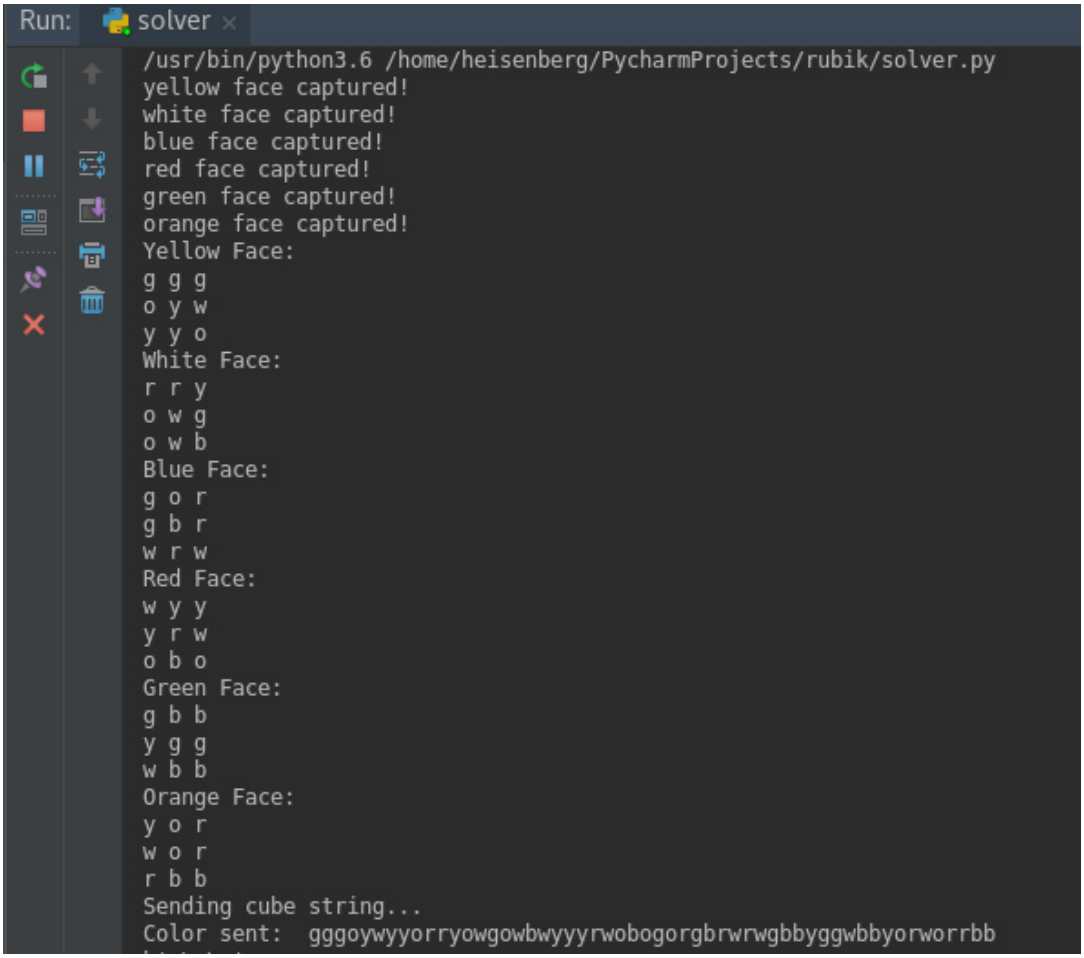
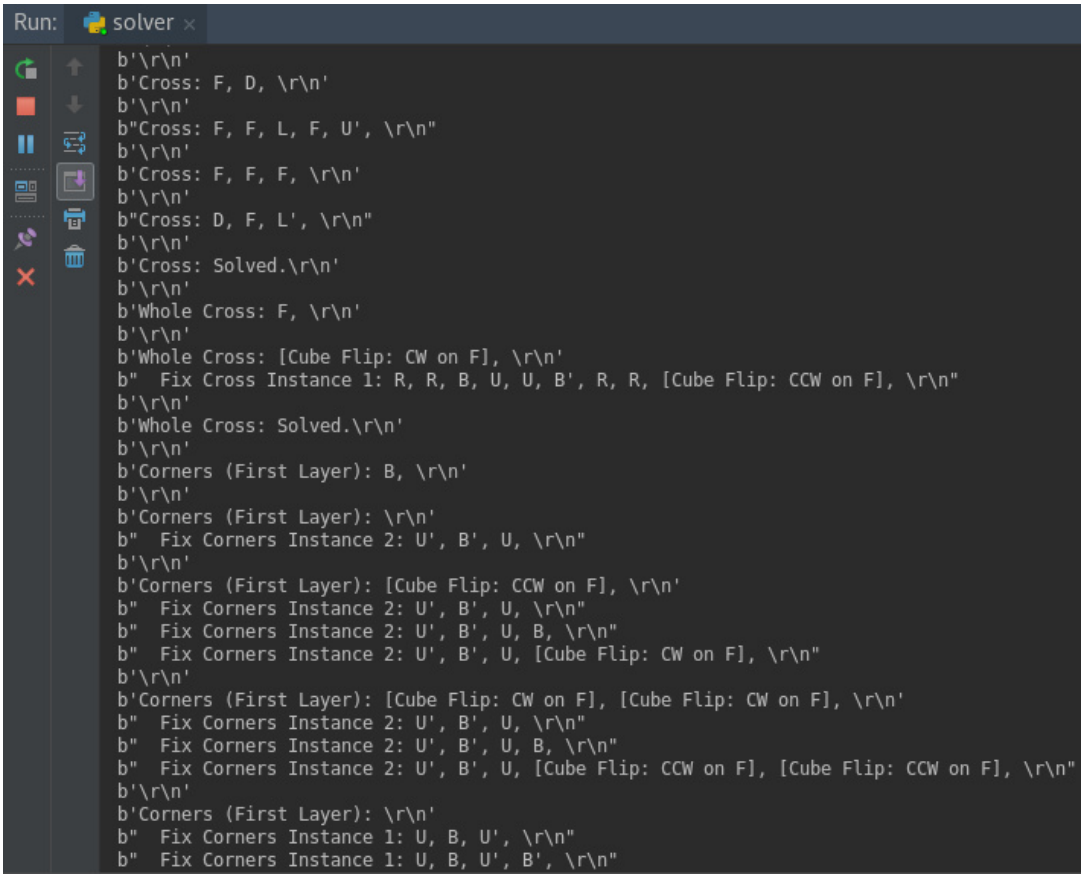
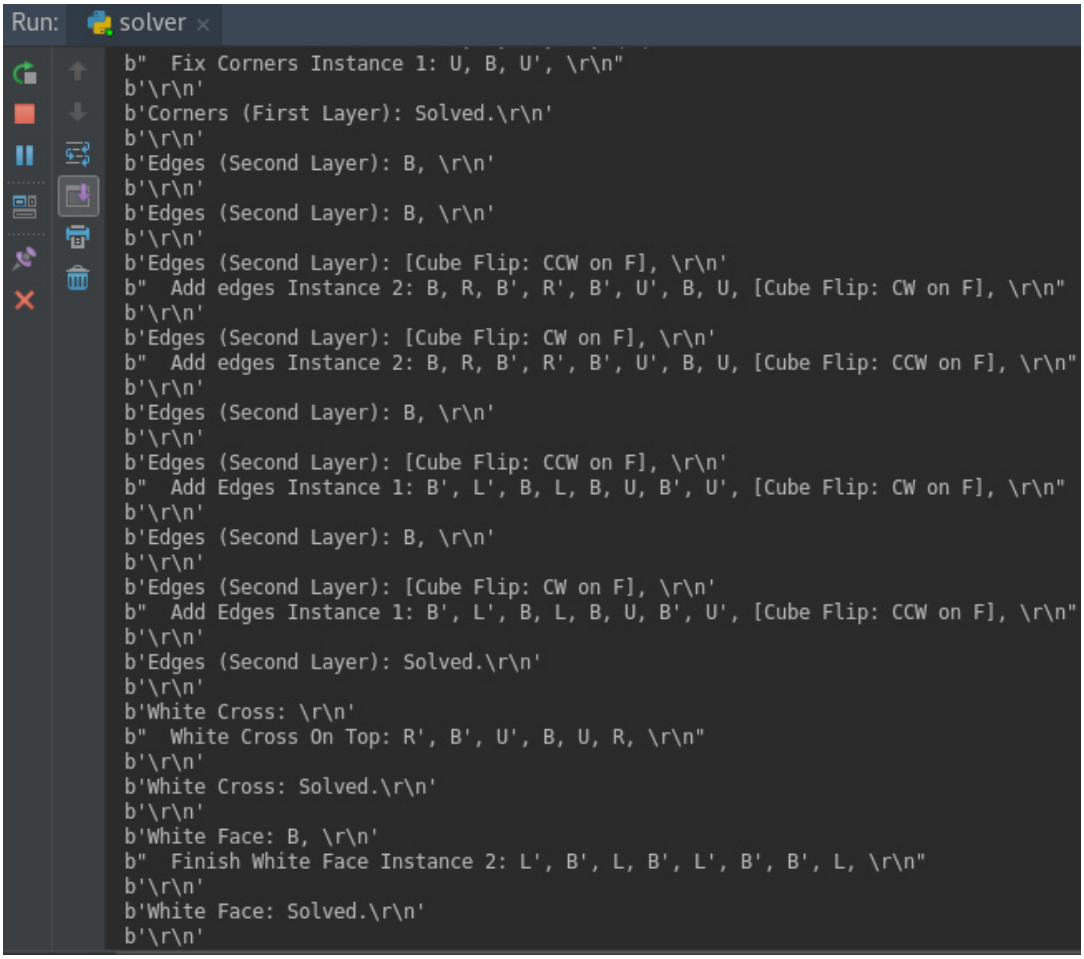


Figure 7.2: Raw string sent to Arduino post-image processing

For the purpose of solving the cube, it is then placed on the platform with the yellow face facing the arm and the blue face acting as the face on top. The raw string sent to the cube is parsed and the sequence of moves are generated allowing for either the manual solving of the cube or the solving of the cube through the bot by placing it on the platform.


Run:
solver
b'\r\n'
b'Cross: F, D, \r\n'
b'\r\n'
b"Cross: F, F, L, F, U', \r\n"
b'\r\n'
b'Cross: F, F, F, \r\n'
b'\r\n'
b"Cross: D, F, L', \r\n"
b'\r\n'
b'Cross: Solved.\r\n'
b'\r\n'
b'Whole Cross: F, \r\n'
b'\r\n'
b'Whole Cross: [Cube Flip: CW on F], \r\n'
b" Fix Cross Instance 1: R, R, B, U, U, B', R, R, [Cube Flip: CCW on F], \r\n"
b'\r\n'
b'Whole Cross: Solved.\r\n'
b'\r\n'
b'Corners (First Layer): B, \r\n'
b'\r\n'
b'Corners (First Layer): \r\n'
b" Fix Corners Instance 2: U', B', U, \r\n"
b'\r\n'
b'Corners (First Layer): [Cube Flip: CCW on F], \r\n'
b" Fix Corners Instance 2: U', B', U, \r\n"
b" Fix Corners Instance 2: U', B', U, B, \r\n"
b" Fix Corners Instance 2: U', B', U, [Cube Flip: CW on F], \r\n"
b'\r\n'
b'Corners (First Layer): [Cube Flip: CW on F], [Cube Flip: CW on F], \r\n'
b" Fix Corners Instance 2: U', B', U, \r\n"
b" Fix Corners Instance 2: U', B', U, B, \r\n"
b" Fix Corners Instance 2: U', B', U, [Cube Flip: CCW on F], [Cube Flip: CCW on F], \r\n"
b'\r\n'
b'Corners (First Layer): \r\n'
b" Fix Corners Instance 1: U, B, U', \r\n"
b" Fix Corners Instance 1: U, B, U', B', \r\n"


Run:
solver
b" Fix Corners Instance 1: U, B, U', \r\n"
b'\r\n'
b'Corners (First Layer): Solved.\r\n'
b'\r\n'
b'Edges (Second Layer): B, \r\n'
b'\r\n'
b'Edges (Second Layer): B, \r\n'
b'\r\n'
b'Edges (Second Layer): [Cube Flip: CCW on F], \r\n'
b" Add edges Instance 2: B, R, B', R', B', U', B, U, [Cube Flip: CW on F], \r\n"
b'\r\n'
b'Edges (Second Layer): [Cube Flip: CW on F], \r\n'
b" Add edges Instance 2: B, R, B', R', B', U', B, U, [Cube Flip: CCW on F], \r\n"
b'\r\n'
b'Edges (Second Layer): B, \r\n'
b'\r\n'
b'Edges (Second Layer): [Cube Flip: CCW on F], \r\n'
b" Add Edges Instance 1: B', L', B, L, B, U, B', U', [Cube Flip: CW on F], \r\n"
b'\r\n'
b'Edges (Second Layer): B, \r\n'
b'\r\n'
b'Edges (Second Layer): [Cube Flip: CW on F], \r\n'
b" Add Edges Instance 1: B', L', B, L, B, U, B', U', [Cube Flip: CCW on F], \r\n"
b'\r\n'
b'Edges (Second Layer): Solved.\r\n'
b'\r\n'
b'White Cross: \r\n'
b" White Cross On Top: R', B', U', B, U, R, \r\n"
b'\r\n'
b'White Cross: Solved.\r\n'
b'\r\n'
b'White Face: B, \r\n'
b" Finish White Face Instance 2: L', B', L, B', L', B', B', L, \r\n"
b'\r\n'
b'White Face: Solved.\r\n'
b'\r\n'

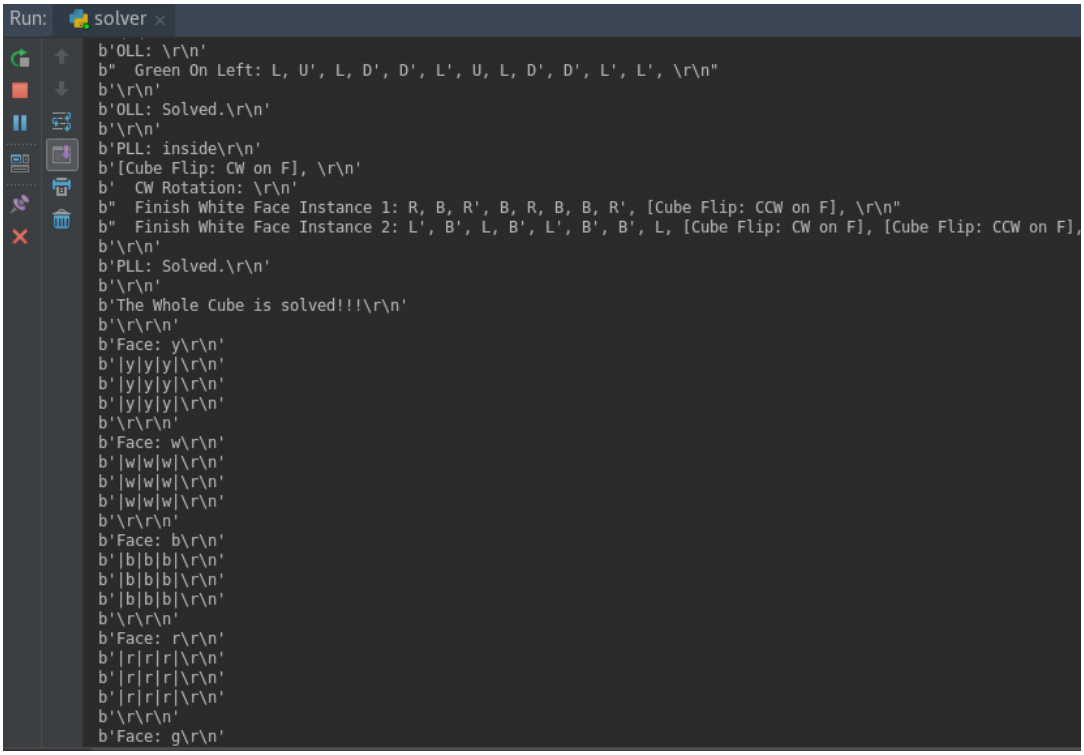


Figure 7.3: Sequence of moves to solve the cube

The cube is accurately being solved through the mechanism without any glitches. Therefore, the mechanical design has been implemented perfectly.

Chapter 8

Conclusion

The mechanical robot is able to detect the colours of the cube cells and subsequently solve the scrambled Rubik's cube. Since the model is using only two servo motors, the goal to make the cube as simple as possible mechanically and to use less resources than other existing models was successfully achieved. The cube solving algorithm also displays the steps required for cube solving in the case when manual solving of the Rubik's cube is desired. Therefore, the proposed model successfully achieved all the objectives laid out for it.

References

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

[2] Korf, Richard E. *Finding optimal solutions to Rubik's cube using pattern databases*. AAAI/IAAI. 1997.

[3] Kunkle, Daniel, and Gene Cooperman. *Twenty-six moves suffice for Rubik's cube*. Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation. ACM, 2007.

[4] http://www.techrepublic.com/article/lego-cubestormer-3-robots-and-the-future-of-mobility/

[5] http://www.tiltedtwister.com/tt2download.html

[6] Korf, Richard E. *A Program That Learns to Solve Rubik's Cube*. AAAI. 1982.

[7] Kasprzak, Wodzimierz, Wojciech Szynkiewicz, and ukasz Czajka. *Rubik's cube reconstruction from single view for Service robots*. Machine Graphics Vision International Journal 15.3 (2006): 451-459.

[8] Reich, F. R., et al. Robot to Solve Rubiks Cube. No. 830341. SAE Technical Paper, 1983.

[9] Interface News, *Measurement and Control*, Vol 47, Issue 5, pp. 140 - 145

[10] Nguyen, Viet; Gchter, Stefan; Martinelli, Agostino; Tomatis, Nicola; Siegwart, Roland (2007). *A comparison of line extraction algorithms using 2D range data for indoor mobile robotics*. Autonomous Robots. 23 (2): 97.

[11] Amruta L. Kabade and Dr. V.G. Sangam, *Canny edge detection algorithm*, International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), Volume 5, Issue 5, May 2016

[12] Rudwan A. Husain, Ali S. Zayed, Wesam M. Ahmed, Hanan S. Elhaji, *Image segmentation with improved watershed algorithm using radial bases function neural networks*, IEEE.

[13] Bradski, G., Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".

[14] Mahes Visvalingam Peter J. Williamson, *Simplification and Generalization of Large Scale Data for Roads: A Comparison of Two Filtering Algorithms*, Cartography and Geographic Information Systems, Volume 22, 1995 - Issue 4.

[15] M. Visvalingam, and J. D. Whyatt, *The Douglas-Peucker Algorithm for Line Simplification: Reevaluation through Visualization*, Computer Graphics Forum, Volume 9, Issue 3, September 1990, pp. 213-225.

[16] David Douglas & Thomas Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer 10(2), 112122 (1973)

[17] J. A. Eidswick, *Cubelike Puzzles–What are They and How do you Solve Them*, The American Mathematical Monthly, Vol. 93, No. 3 (Mar., 1986), pp. 157-176.

[18] David Joyner, *The Man Who Found God's Number*, The College Mathematics Journal, Volume 45, 2014 - Issue 4.

[19] Hana M. Bizek, *The Rubik's-Cube Design Problem*, BRIDGES Mathematical Connections in Art, Music, and Science

[20] G.T. Shrivakshan and Dr. C. Chandrasekar, *A Comparison of various Edge Detection Techniques used in Image Processing*, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 1, September 2012.

[21] Cezary Zieliski, Tomasz Winiarski , Wojciech Szynkiewicz, Maciej Staniak, Witold Czajewski, Tomasz Kornuta1, *MRROC++ Based Controller of a Dual Arm Robot System Manipulating a Rubiks Cube*, Institute of Control and Computation Engineering, Institute of Control and Industrial Electronics, Grant no 4 T11A 003 25, Report no 0610, Warsaw, June 2007.

[22] Roushdy, Mohamed. *Comparative study of edge detection algorithms applying on the grayscale noisy image using morphological filter*, ICGST, GVIP Journal, Volume 6, Issue 4, December, 2006.

[23] David Singmaster (1980-08-06). *A Step by Step Solution of Rubik's "Magic Cube"*