|  |
| --- |
| CS101 EMbedded systems project |
| SuperCuber |
| Implementation of a Rubik’s cube solver using a Firebird V |

Rishi Vanukuru 140040014

Vasishta Garimella 140070044

Shantanu Choudhary 14D070044

Abhishek Andasu 140070057

|  |
| --- |
|  |

Abstract:

The Rubik’s cube over the past decade has become an object of immense passion and solving it is an engrossing hobby to many. It is the world’s top selling puzzle and is widely considered to be the world’s bestselling toy. Since the Rubik’s cube introduction, its popularity has inspired many students and researchers to build robots capable of solving this 3D puzzle. No matter what mechanics is used, the principle of real cube solving process remains the same including three phases: Detection of cube state, Generating the solution and realizing the particular moves.

The aim of this project is do exactly this, making use of the functionality of the Firebird bot. We have incorporated image processing to help automate the process. We have implemented the Thistlethwaite algorithm to solve the cube to simplify the movements executed by the bot.

Contents:

1. Introduction
   1. Problem Statement
   2. Objectives
2. Requirements
3. Background
   1. Motion Constraints

3.2 Image Processing

3.3 Solving

1. Mechanical Movement
2. Image Processing
3. Solving the Cube
4. Execution of Solution
5. Testing Strategy and Data
6. Discussion of System
7. Conclusions and future work
8. Unique features about the Project

Introduction:

Rubik’s Cube is a puzzle consisting of a plastic cube-like apparatus with 6 sides each of different colours, and each side consisting of a 3x3 grid of squares. The cube can be rotated about three different axes, and the aim of the puzzle is to solve – unscramble – the cube.

In the present day, Rubik’s cubes are exceedingly popular among the people all over the world. The Cube has inspired an entire category of similar puzzles, commonly referred to as twisty puzzles, which includes the cubes of different sizes mentioned above as well as various other geometric shapes. The puzzle is popular now in a large number of countries and there are numerous international contests where cube enthusiasts aim to solve the cube and its variations in the fastest possible time.

In this project we are going to solve a regular 3x3 Rubik’s cube. We have assimilated image processing to obtain a digital version of the cube which upon solving will return instructions to the Firebird bot which will execute them sequentially to obtain the solved cube.

Problem Statement:

1. Obtain a digital cube, in the form of an array which can represent the entire cube exhaustively.
2. Solve the cube
3. Execute the solution, mechanically using the Firebird.

**Objectives:**

1. Scan all the sides of the cube. From the images procured, recreate the cube digitally. This is done by superimposing the images to recreate each individual side the information of which is then stored in an array sequentially.
2. Pass the digital cube through the solving algorithm (which is implemented through the Thistlethwaite algorithm) to obtain four phase instructions to solve the cube.
3. The solution is resolved through the Firebird. The motions are converted into sequential movements that can be executed by the bot accounting for its constraints in motion.
4. The cube is then solved through the mechanised motions of the Firebird.

Requirements:

1. **Hardware requirements:**
2. Firebird V5
3. Plastic Framework
4. External AC-DC Adapter
5. Sealant
6. Servo Motor
7. Mechanix and K-Nex
8. USB converter to bootload the bot
9. Objects to provide inclination
10. Tape and adhesive
11. **Software Requirements:**
12. Open CV for image processing
13. AVR studio to bootload instructions to the bot
14. Atmel Studio 6 to program in embedded C
15. C++ Codeblocks

Background:

**Motion Constraints:**

Manipulation of the cube is done using a motor and a servo motor. This limits the motions possible. So the execution is done ensuring that all six faces of the cube can be enacted upon. The motions are executed by the Firebird.

**Image processing:**

Image processing is done using OpenCV. Colours are detected using pixel values and superimposed

**Solving:**

Thistlethwaite's algorithm is used to solve the cube. This involves solving the cube in four phases, each of which simplifies the manoeuvres required for subsequent phases. The input provided is in terms of the positions of various cubelets with respect to a fixed orientation.

Mechanical Movement:

The mechanical movements are restricted to – free rotation, hold rotation, flip.

Free rotation aims to change the orientation of the cube with respect to the front face. It changes the face that is towards us without changing the positions of the cubelets with respect to each other.

Hold rotation basically rotates the “down” layer.

Flip changes the face that is “down” by flipping the cube to change its orientation. Again, the positions of the cubelets with respect to each other do not change.

Tangible change in position only happens upon hold rotation, so the movements involve manipulation to bring the face to be rotated “down”. This is done by flipping and free rotation.

Through these three motions, all the faces of the cube can be manipulated thus reducing the need for more mechanised parts.

Flip is actualised using the servo motor. Servoindex() is used to vary the frequency and speed of the motion of the servo. Through progressive angles the mechanical arm moves forwards and then backwards to execute a flip. Due to the slope, gravity ensures that the cube falls back in place after flipping.

Free rotation is actualised by the rotation of the wheel of the Firebird. Motioncontrol() is used to vary the speed of rotation of the wheel and specify rotation.

Hold rotation is actualised by a combination of the two, holding of the cube in place by the mechanical arm while rotation of the wheel takes place.

All possible motions of the cube are expressed in terms of these fundamental motions.

Image Processing:

The function videocapture() initiates the camera and displays the video in real time. Using Cvtcolor() and inRange() real time thresholding of the video is done. Cvtcolor() converts RGB values to hue saturation values. inRange() sets the filters and converts the real time video to a binary form (in terms of black and white). Filters are set for all the colours prevalent in the cube to enable thresholding for all individual colours.

Images are then captured individually for all the colours for each face by using the respective filter values. Using findContours() and drawContours() the thresholded images are filled with the corresponding colours. These images now have coloured spaces depicting the exact positions of various squares with their colours. These images are saved.

Now, we basically have digital images of the faces of the cube, albeit for individual colours.

Each individual face is now created in the program by scanning the images, sequentially in order of colour and using the pixel values to exactly depict each face. This is stored in an array in a particular order.

Solving the Cube:

In accordance with the colours (representative of the faces they are from) the cubelets are designated names. Using the array, their positions are represented in correspondence with their names. These positions act as an input for the solving algorithm.

The cube is solved using Thistlethwaite's algorithm.

The way the algorithm works is by restricting the positions of the cubes into groups of cube positions that can be solved using a certain set of moves.

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

This group contains all possible positions of the Rubik's Cube.

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

This group contains all positions that can be reached (from the solved state) with quarter turns of the left, right, front and back sides of the Rubik's Cube, but only double turns of the up and down sides.

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

In this group, the positions are restricted to ones that can be reached with only double turns of the front, back, up and down faces and quarter turns of the left and right faces.

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

Positions in this group can be solved using only double turns on all sides.

G4 = {I}

The final group contains only one position, the solved state of the cube.

The cube is solved by moving from group to group, using only moves in the current group, for example, a scrambled cube likely lies in group G0. A look up table of possible permutations is used that uses quarter turns of all faces to get the cube into group G1. Once in group G1, quarter turns of the up and down faces are disallowed in the sequences of the look-up tables, and the tables are used to get to group G2, and so on, until the cube is solved.

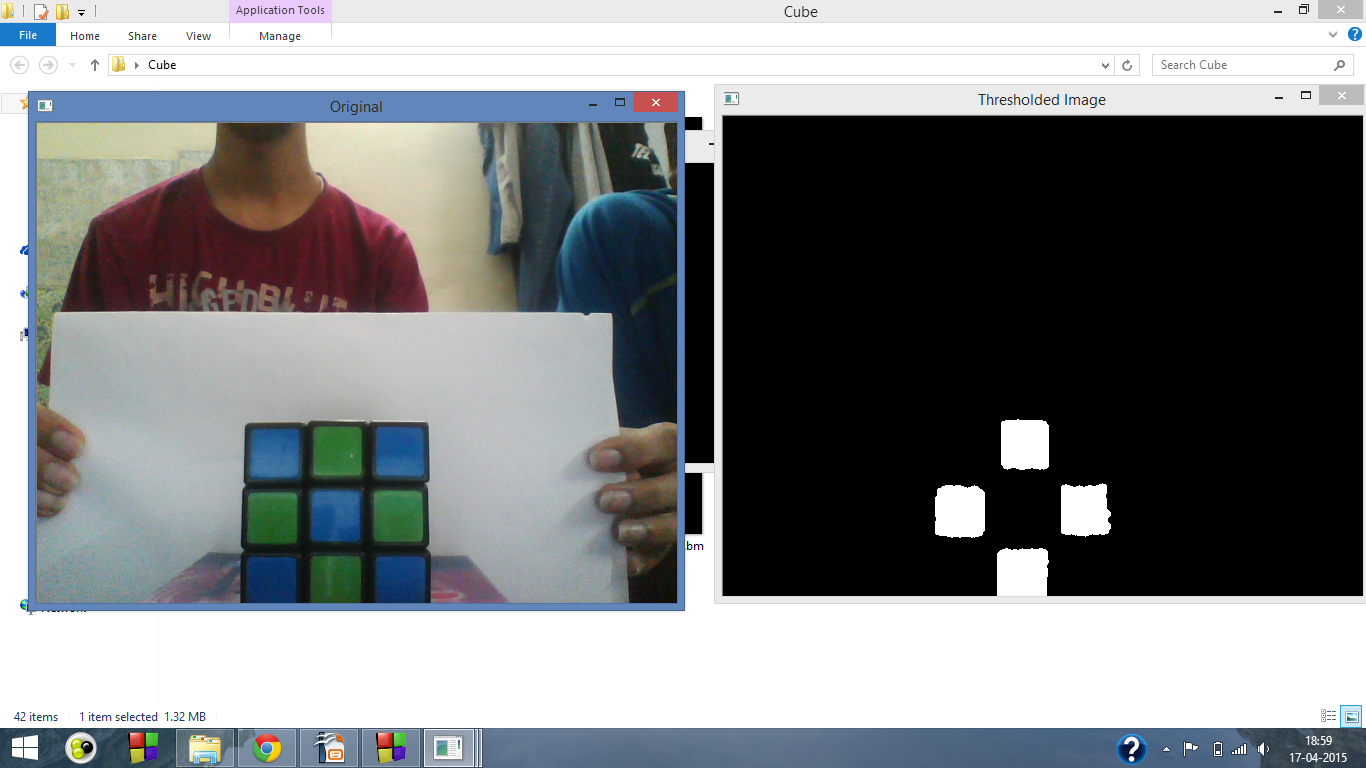
The solution is represented as rotation of the faces of the cube ensuring that the orientation of the cube is constant.

Execution of Solution:

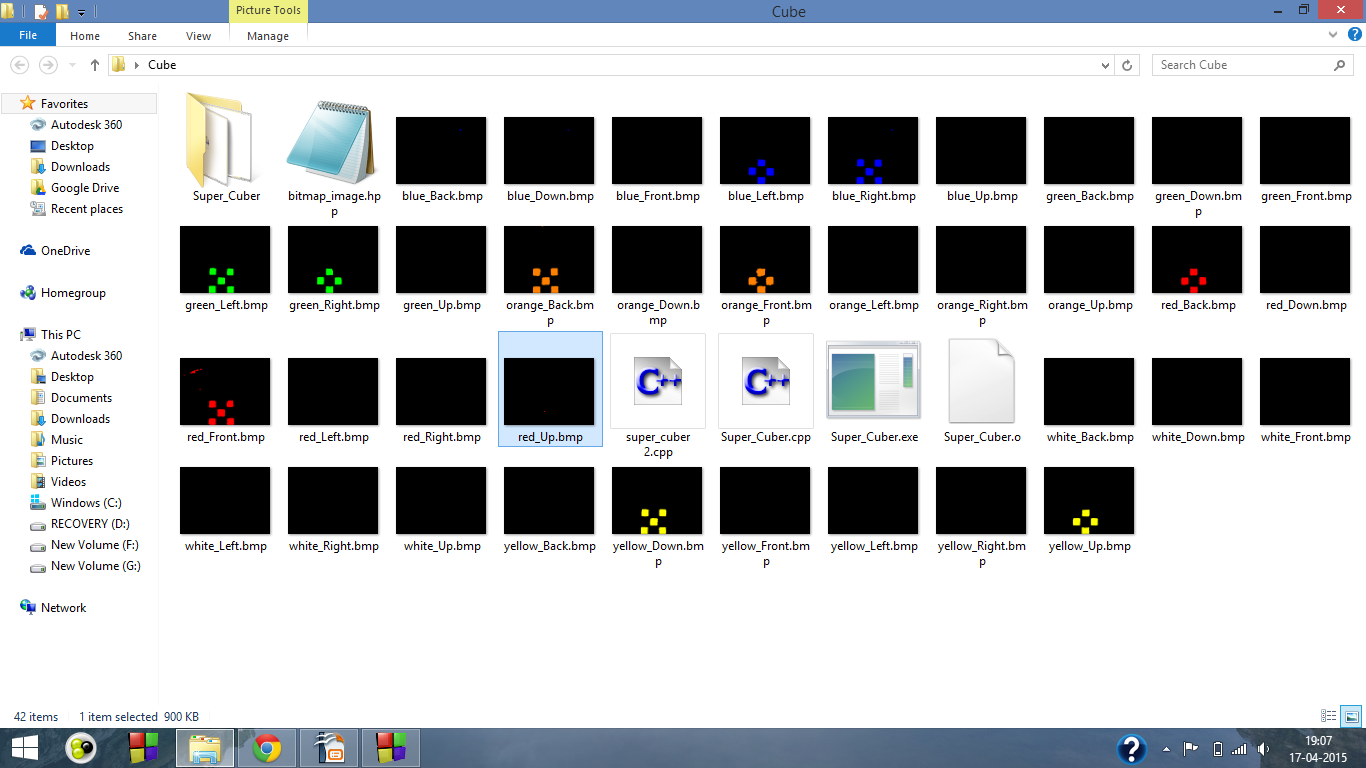
The solution is converted into a combination of hold rotation, free rotation, flipping to account for both the motion constraint and the change in orientation that is undergone for each successful manoeuvre. These are executed successively by the Firebird to obtain a solved cube.

Testing Strategy And Data

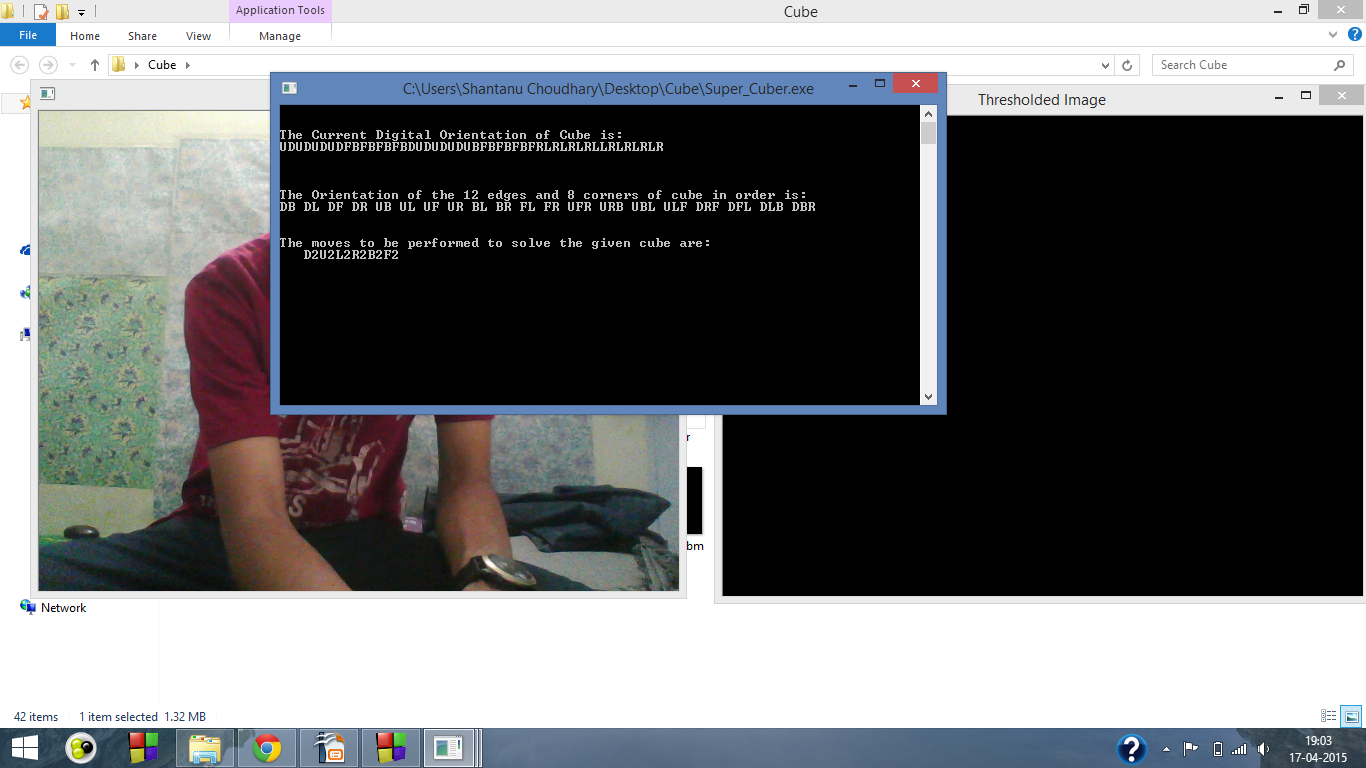
We begin by taking images of the various faces of the cube. Throughout our code we have consistently maintained a fixed orientation for the cube where white is on top and red is in front. The storage of the cube as an array is also done in the same orientation. The execution therefore follows with the same orientation in mind.



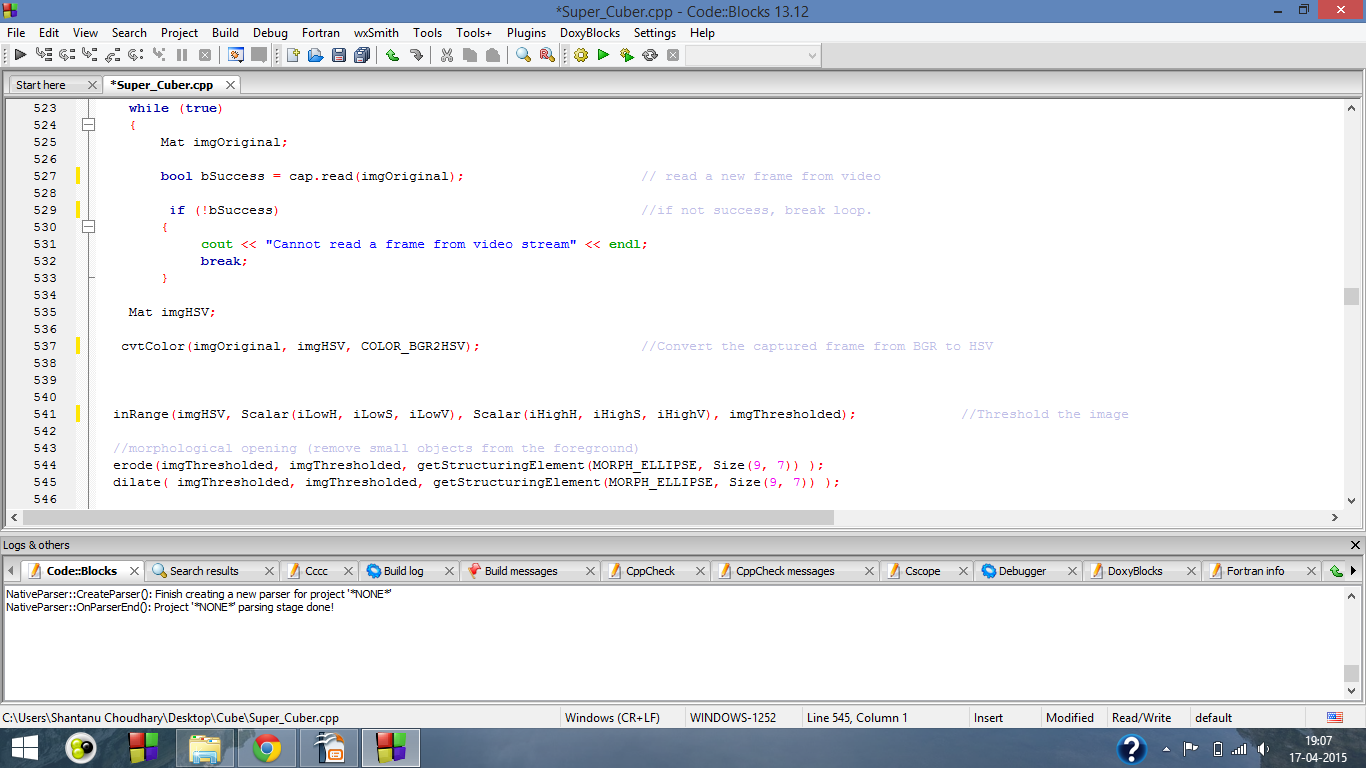
Thresholded images are obtained for specific colours by applying the filters for the specific colours.



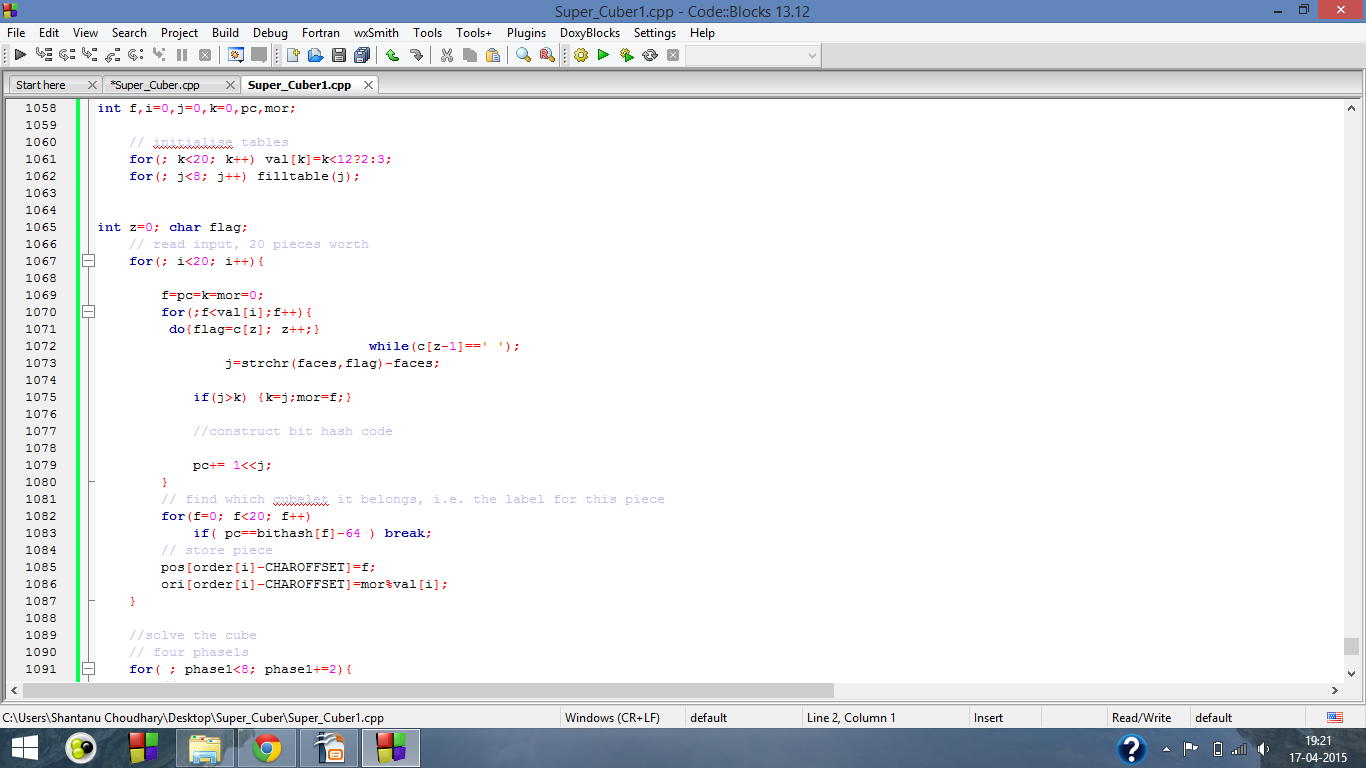
The thresholded images are coloured using contouring and are saved. The face they represent is documented.



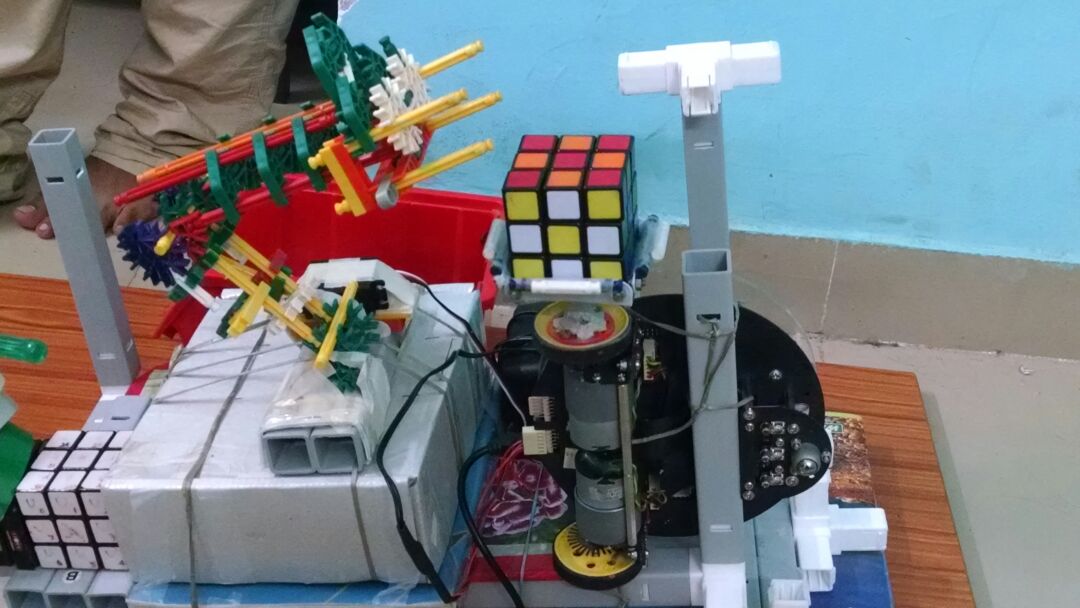
The solution for the given cube is obtained and those steps are represented as shown above.



Code for the image processing



Code for solving the cube



The bot solving the cube

Discussion of System

1. **What worked according to plan:**
2. Image Processing: We were able to effectively scan the all the faces of the cube to recreate the cube digitally on which we could carry out our manipulations
3. Cube Solving: Using the digital representation of the cube, we were able to run a successful program that could give us the solution to the cube.
4. System for solving: We were able to design a system that could convert the movements of the cube’s faces to motions of the system while accounting for its constraints in motion
5. **What we added more than discussed in the SRS:**
6. Framework: We managed to design a sturdier framework and a better designed mechanical arm than initially envisioned
7. Thistlethwaite algorithm: We managed to incorporate the Thistlethwaite algorithm which is far more efficient than the human algorithms we initially wanted to execute. It is a purely computational algorithm and is highly efficient. It is able to give us an accurate solution within an average of thirty moves.
8. **Changes made in plan:**

Segmentation of Image Processing and Solving: We separated the processes of image processing and solving by the bot. Image processing is now done using the computer and its result is fed back into the bot.  
Reasons: Implementing image processing using the Firebird was proving to be a challenge. It was harder to configure the Firebird to make it capable of image processing. This involved programming additional libraries. Also considering that it has a microcontroller and not a microprocessor, we were not very optimistic of its capabilities and its efficacy once executed.

1. **Challenges:**

Accuracy of movements of the wheel of the Firebird: We are relying on time to gauge the rotation of the wheel of the bot which we are using to rotate the bot. The problem with this is the tendency of the wheels to skid. Whenever this happens the alignment gets disturbed, at which point human intervention is required to realign the cube.

Conclusions and future work

1. We have been able to create a working Rubik’s cube solver which is nearly completely automated.
2. The image processing and the solving algorithm work in tandem, seamlessly to provide an accurate solution.
3. The mechanical arm is able to execute its motions satisfactorily.
4. To better the project, we need to make use of the wheel counter to make the motions of the wheel accurate and remove the need for human intervention.
5. Another objective in the future would be to integrate the image processing with the bot and enable it to solve the cube in real time. We aim to do this with the inclusion of a camera on the bot.

Unique features about the Project

1. Multidimensional: This project combines multiple aspects, mechanical execution, image processing and coding algorithms with the aim of automating the process.
2. Comprehensive: This project can effectively and independently utilise all the aspects involved in a multitude of applications.
3. Unique: Makes uncommon and unique use of the capabilities of the Firebird.
4. Well Integrated: All aspects have been effectively integrated to make seamless transitions.

References:

1. OpenCV Installation

<https://www.willowgarage.com/pages/software/opencv>

1. Bitmap Image Libraries

<https://www.partow.net/programming/bitmap/index.html>

1. Cube Solving Algorithm

<https://www.jaapsch.net/puzzles/thistle.htm>