Solving the Rubik's Cube - Modelling 1

Chen, Dongdong & Mohammad Hariz bin Mohd Hisham

April 27, 2017

Abstract

This report details the overall procedure for the creation and use of a program to solve a Rubik's Cube. Various challenges while writing the code are brought forth and are contained in the report. The layout and use of the graphical user interface (GUI) is also explained in this report. Overcoming the challenges faced and making a program from scratch has given a greater understanding as to the process of modeling data in computers and of programming syntax in general.

1 Introduction

The Rubik's cube is 3D puzzle invented by Hungarian sculptor Ern Rubik. It is a cube with six differently coloured faces, which are usually red, orange,yellow, green, white and blue. The cube itself is made up of 26 smaller cubes for a simple 3x3x3 cube. The cube can be rotated around a central, '3D Cross' which acts as a pivot for each of the six faces. This essentially leads to a total of 43,252,003,274,489,856,000 number of configurations for a single 3x3x3 cube. Although regarded as a toy, this extremely high number of configurations speaks strongly to the complexity of this seemingly innocuous puzzle.

We set out to create a program that can solve a 3x3x3 Rubik's cube. The code should be able to solve the cube from any one of the possible 43,252,003,274,489,856,000 configurations. The program should also be able to generate any one of these combinations either by random generation or by manual entry of the different possible rotations. In addition to this, we wanted the program to be able to display the cube in both 3D and a spread out 2D representation to ease visualisation of the cube for the user.

With these criteria in mind, there were various functions which we needed to create in order to solve the cube. Firstly, we needed a way to translate the physical cubes of the Rubik's cube into a language the computer could understand. Next, we needed to alter the stored values of the cube to simulate the turning of an actual cube. We also set out to create an interface for the user to interact with to rearrange the cube however he/she sees fit or as a means to randomly scramble the cube. Finally, the program should be able to consistently solve the cube regardless of the configuration of the cube.

2 Flowchart/Pseudocode

```
Program Solve the Rubik's Cube
Create an array of variables as a representation of each cube face;
Create functions to rearrange the values of the array which corresponds to rotations of the cube;
Create a 3D cube with the colours of a standard Rubik's Cube;
Create a solver to rotate the 2D cube;
Generate strings so that when a 2D rotation is undergone, the 3D cube is rotated in the same manner;
while First layer is not solved
    Rotate cube in specific ways until first layer is solved
end
while Second layer is not solved
    Rotate cube in specific ways until second layer is solved
end
while Third layer is not solved
    Rotate cube in specific ways until third layer is solved
end
Create a GUI to display the unsolved and solved cube;
end
```

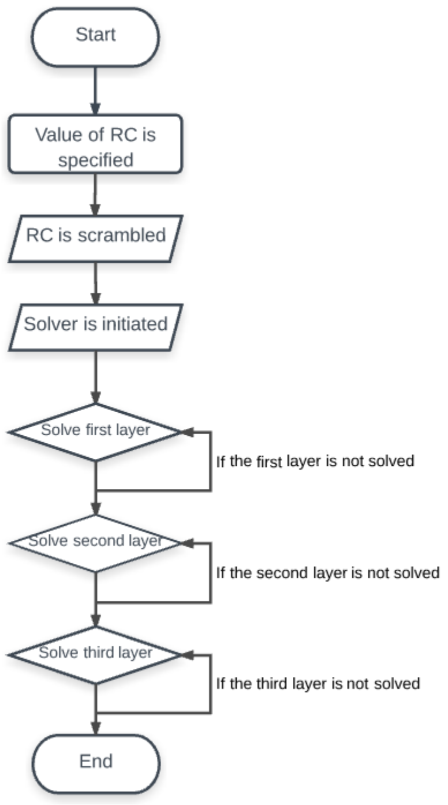
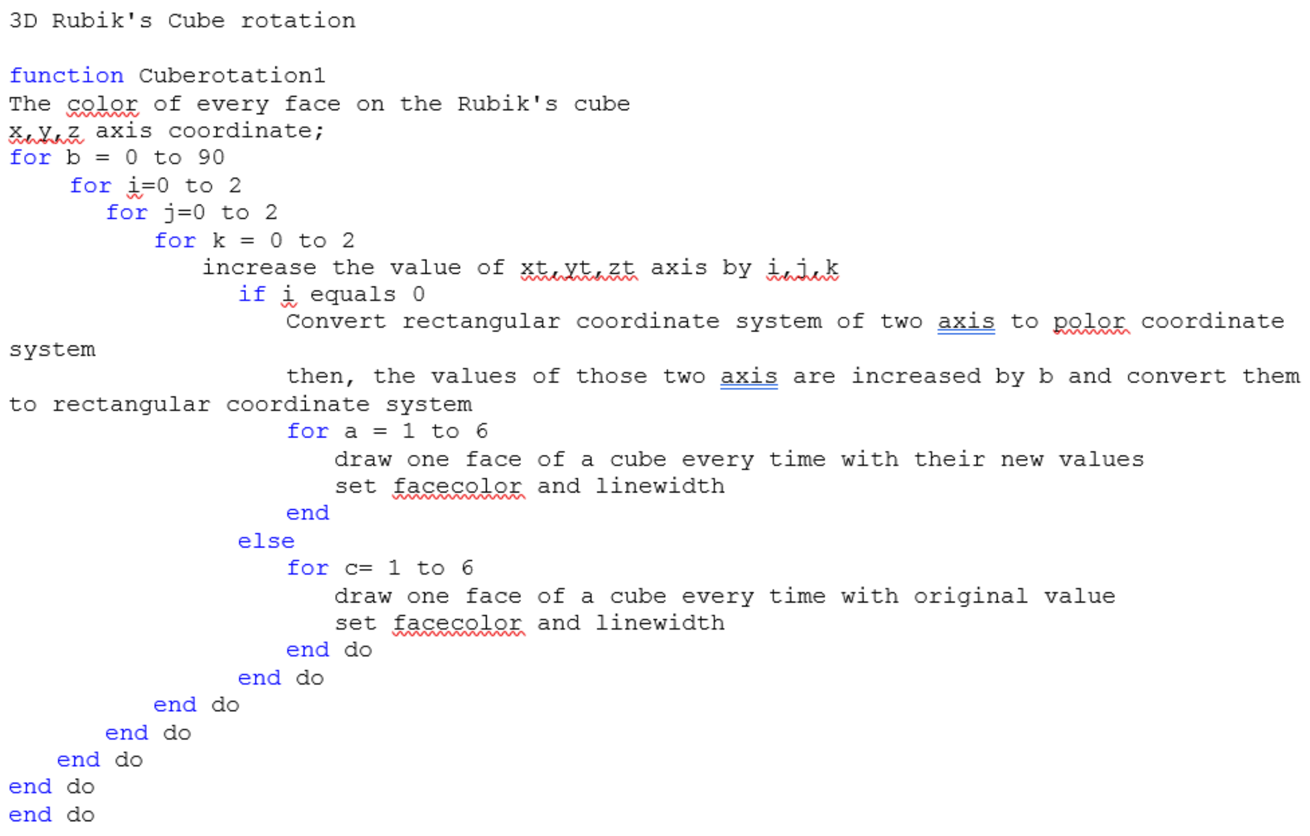


Figure 1: The program flowchart

3 Major Challenges

3.1 Developing the 3D Rubik's Cube rotation

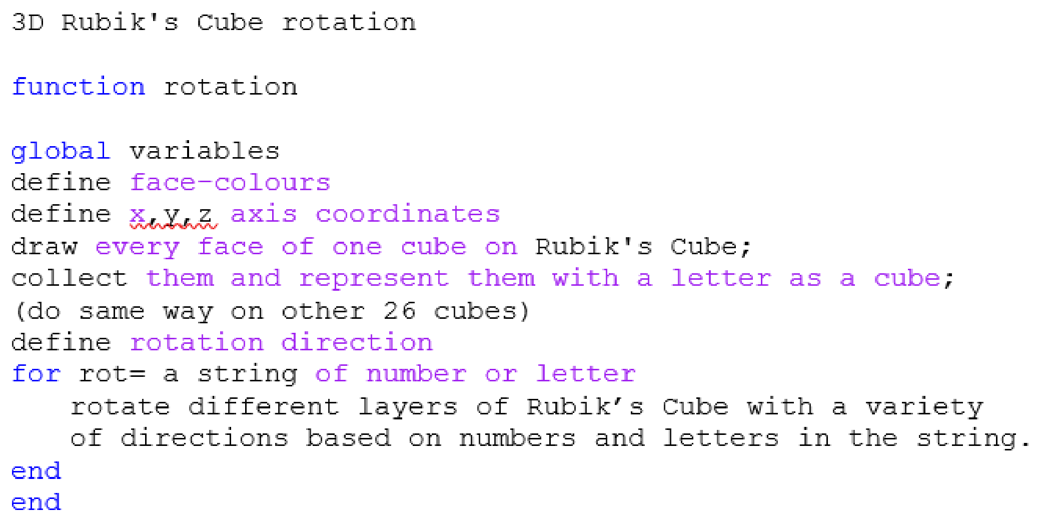
In this part, 2D square rotation was taught in the lecture and the first rotation method of 3D cube was an extension of it.
The first method:



```
3D Rubik's Cube rotation

function Cuberotation1
The color of every face on the Rubik's cube
x,y,z axis coordinate;
for b = 0 to 90
    for i=0 to 2
      for j=0 to 2
         for k = 0 to 2
            increase the value of xt,yt,zt axis by i,j,k
              if i equals 0
                 Convert rectangular coordinate system of two axis to polor coordinate
system
                 then, the values of those two axis are increased by b and convert them
to rectangular coordinate system
                 for a = 1 to 6
                    draw one face of a cube every time with their new values
                    set facecolor and linewidth
                 end
              else
                 for c= 1 to 6
                    draw one face of a cube every time with original value
                    set facecolor and linewidth
                 end do
              end do
         end do
      end do
   end do
end do
end do
```

Figure 2: The first rotation psuedocode

However, the layer of the Rubik's Cube is rotated so slowly that it became time-consuming. After analysis, this is because every small cube is drawn again after that layer changes angle. It also lead to the rotation of that layer to not be smooth.
Therefore, it was decided that a better method which could create a smoother and faster rotation needed to be applied and fortunately, it was figured out.
Firstly, all of faces of a cube is drawn in Matlab with their face-color. Then, faces on one cube are presented with different letters and every layer of Rubik's Cube is classified with different letters. After that, any one of the layers can be rotated.
The second method

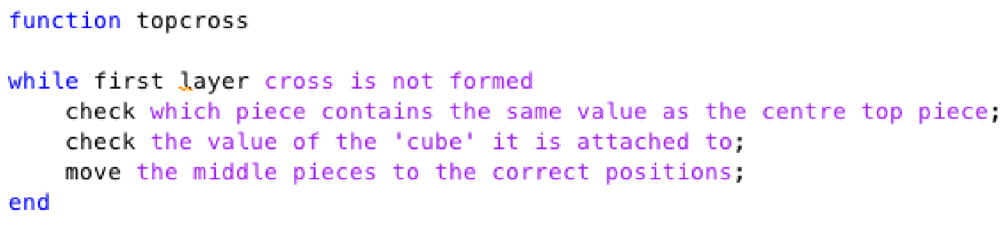


```
3D Rubik's Cube rotation

function rotation

global variables
define face-colours
define x,y,z axis coordinates
draw every face of one cube on Rubik's Cube;
collect them and represent them with a letter as a cube;
(do same way on other 26 cubes)
define rotation direction
for rot= a string of number or letter
    rotate different layers of Rubik’s Cube with a variety
    of directions based on numbers and letters in the string.
end
end
```

Figure 3: The Rubik’s Cube rotation psuedocode

3.2 Creating the 2D solver

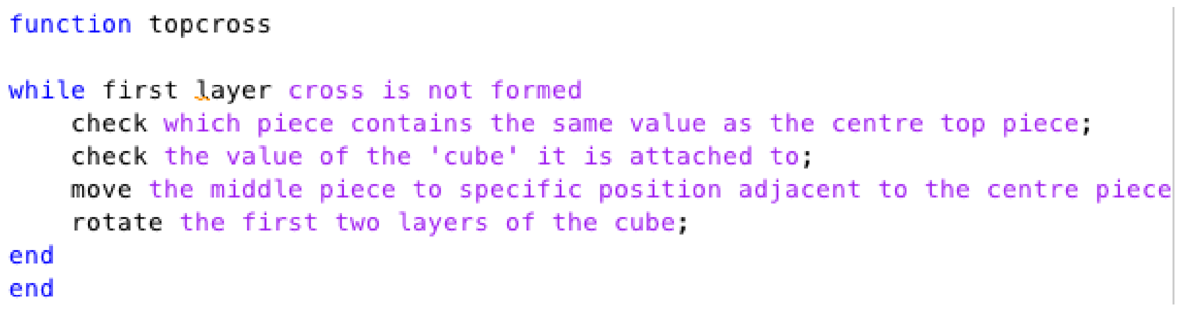
In writing the solver, the the algorithm used to solve the Rubik’s cube was the method that can be readily found from many sources. This method solves the cube layer by layer, starting with a cross for the first layer, followed by solving the corners of the first layer, followed by solving the second layer and so on. A major challenge was in forming the first cross. To form the cross, we would only need to be concerned with the positions of four cube pieces i.e. the middle pieces of the corresponding colour. With a physical cube, these four pieces can easily be spotted and moved to the correct positions. With the array, each possible position needs to be checked to determine whether those pieces should be moved.



```
function topcross

while first layer cross is not formed
    check which piece contains the same value as the centre top piece;
    check the value of the 'cube' it is attached to;
    move the middle pieces to the correct positions;
end
```

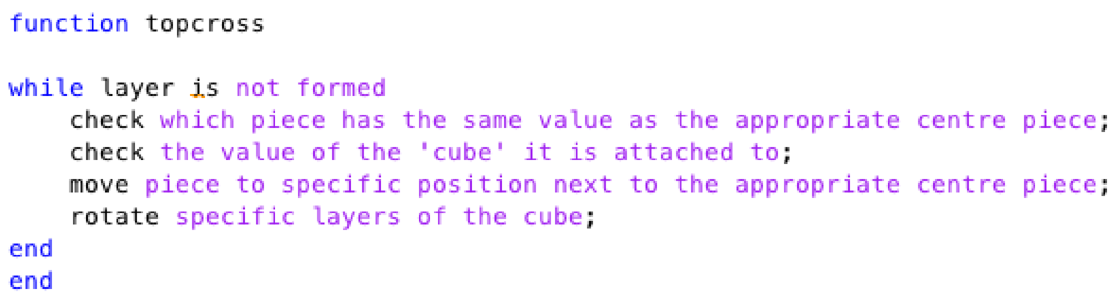
Figure 4: Pseudocode for the first layer cross

However, this required the consideration of 24 possible arrangements for each of the four middle pieces of the cross. Thus to circumvent this issue, the top two layers of the cube was rotated so only the first of the 24 arrangements needed to be considered.
With this in mind, to simplify the code, the following layers only needed to consider a few positions of the correct cube pieces, and if they weren’t present in those positions, the cube is rotated in specific ways. The general pseudocode for the following layers is as follows :



```
function topcross

while first layer cross is not formed
    check which piece contains the same value as the centre top piece;
    check the value of the 'cube' it is attached to;
    move the middle piece to specific position adjacent to the centre piece
    rotate the first two layers of the cube;
end
end
```

Figure 5: The new pseudocode for the first layer cross

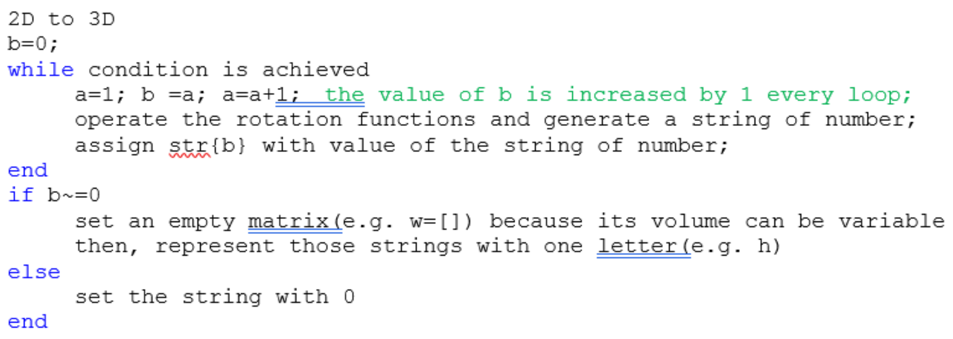


```
function topcross

while layer is not formed
    check which piece has the same value as the appropriate centre piece;
    check the value of the 'cube' it is attached to;
    move piece to specific position next to the appropriate centre piece;
    rotate specific layers of the cube;
end
end
```

Figure 6: General pseudocode for the layers

3.3 Mapping 2D solver to 3D solver

The idea was to make the 2D solver generate a string of numbers or letters. Then, this string is applied in the 3D rotation code to control the rotation order. Because colors on 2D has been matched with the 3D Rubik's Cube, the Rubik's Cube can be solved as long as the 2D cube can be solved. However, since it is unknown how many times the 'while' loop will operate, a specific matrix containing the strings of numbers could not be created. Therefore, a variable matrix is needed.



```
2D to 3D
b=0;
while condition is achieved
     a=1; b =a; a=a+1;  the value of b is increased by 1 every loop;
     operate the rotation functions and generate a string of number;
     assign str{b} with value of the string of number;
end
if b~=0
     set an empty matrix(e.g. w=[]) because its volume can be variable
     then, represent those strings with one letter(e.g. h)
else
     set the string with 0
end
```

Figure 7: A string of number as output

4 Running the Code

1. Method to run code
   First step: Open Matlab and find a GUI file called 'rubickscube.m';
   Second step: Click the 'Run' button on the tool bar. Then, a display panel with buttons will be shown.

2. Method to operate display panel and corresponding figures
   On the display panel, those buttons are divided into three main parts which are 'Scramble', 'Solver' and 'Single Step'. In addition, there is another area which is for displaying Rubik's cube. The display panel is shown below.

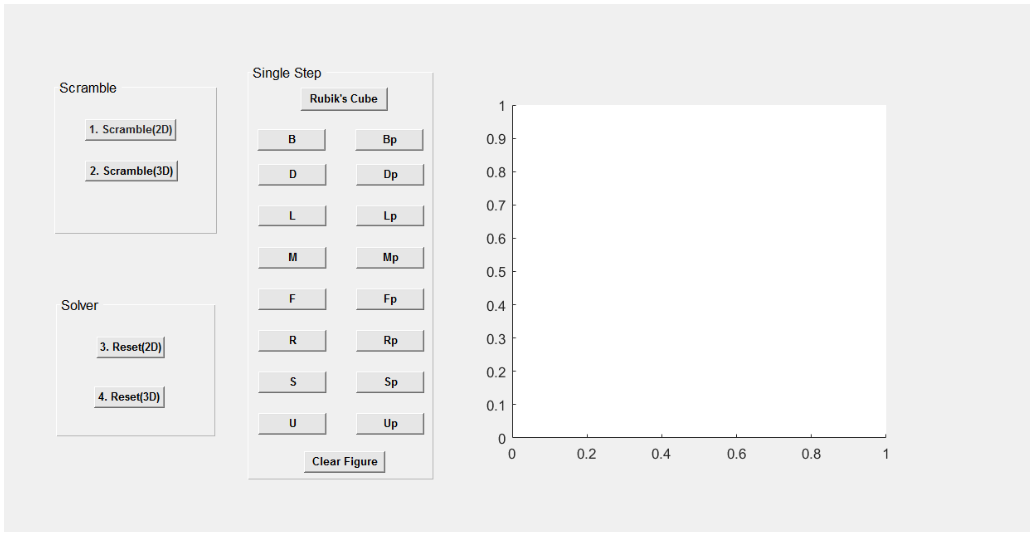


Figure 8: The copy of display panel

   (a) Button group: Scramble
   There are two buttons which are 'Scramble(2D)' and 'Scramble(3D)'. Firstly, the button 'Scramble(2D)' is pressed and a figure will be shown(e.g. Figure 9). Then, press the second button 'Scramble(3D)' and a 3D figure of a disordered Rubik's cube is shown on the display panel and this cube corresponds to Figure2(e.g. Figure 10). In order to obtain a 3D figure which corresponds to the 2D figure, these two button should be pressed in order, starting from 2D followed by 3D.

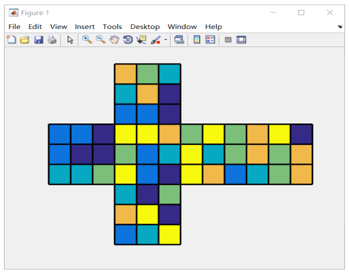


Figure 9: The image of disordered 2D Rubik's Cube

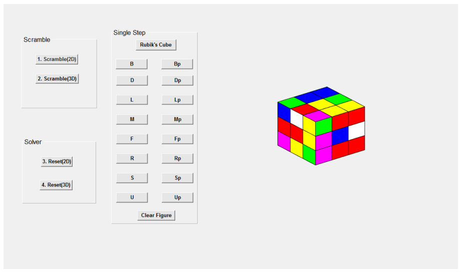


Figure 10: The image of disordered 3D Rubik's Cube

(b) Button group: Solver
Like the button group 'Scramble', there are two buttons which are 'Reset(2D)' and 'Reset(3D)'. To solve the 3D Rubik's cube, the 2D figure of Rubik's cube has to be solved first. Therefore, the 'Reset(2D)' button is to be pressed first. After the scrambled 2D figure is solved like Figure 11, the 'Reset(3D)' button is pressed to start the rotation of 3D Rubik's Cube. Figures below are examples of the expected results. Moreover, the 3D cube will be solved with three-dimensional animation.

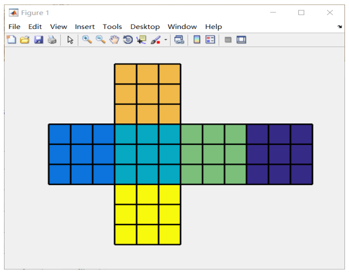


Figure 11: The image of reverted 2D Rubik's Cube

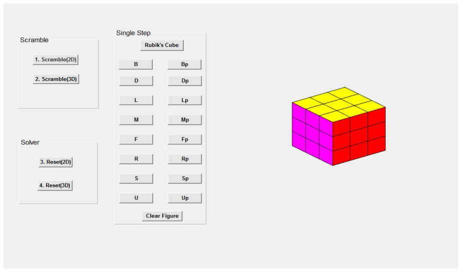


Figure 12: The image of reverted 3D Rubik's Cube

(c) Button group: Single Step
In this part, the buttons apart from 'Rubik's Cube' and 'Clear Figure' are used to control the rotation of the layers of the Rubik's cube. Thus, the buttons that control the rotations can be pressed because a disordered 3D Rubik's cube has been solved in last step and it is shown in the panel like Figure 5. However, if a new Rubik's Cube is preferred, the button 'Clear Figure' and 'Rubik's Cube' should be pressed in that order to avoid overlapping figures(e.g. Figure 13 and Figure 14). Then, buttons for controlling rotations can be pressed randomly and 3D cube will move corresponding to commands. Figure 15 demonstrates the state after button 'L' is pressed.

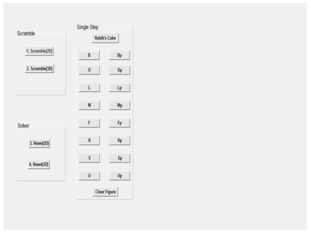


Figure 13: The image of clearing 2D Rubik's Cube

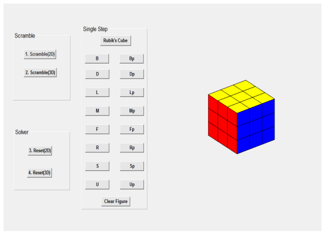


Figure 14: The image of new 3D Rubik's Cube

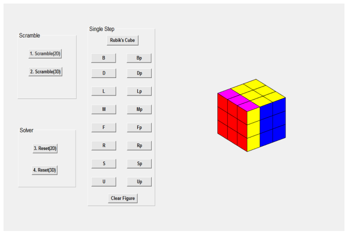


Figure 15: Image of 3D Rubik's Cube with left rotation

3. Special issues

(a) Those code are tested on the PC with windows system;

(b) The 2D figures are shown in display frame 'Figure 1'.

5 Conclusions

This project is about creating, scrambling and solving a Rubik's Cube with programming language in Matlab. During this process, the representation of a 2D and 3D Rubik's Cube was achieved and a 2D scrambled Rubik's Cube can be automatically solved. Additionally, the 2D solver was successfully mapped onto the 3D Rubik's Cube. It means that a 3D Rubik's Cube can be reverted to its original state as well. Then, to simplify operation all of these functions were displayed in a control panel, i.e. a GUI. To enrich the panel's functions, every single rotation is implemented on the panel. However, there are some improvements which could be done, such as image processing i.e. using a webcam input and making a machine to solve the cube After this project, the understanding and application of Matlab is more familiar to us, especially the usage of its various commands. Moreover, it tested and hence, improved our problem solving abilities. Therefore, it can be concluded that a Rubik's Cube is successfully solved by our program and numerous processes have been achieved although there is still room for adding functionality for the program.

Appendix

The application of every function in Matlab

2D Rubik's Cube	
Function	Application
B	Turning clockwise, the back face
F	Turning clockwise, the front face
R	Turning clockwise, the right face
L	Turning clockwise, the left face
U	Turning clockwise, the upper face
D	Turning clockwise, the face opposite to the upper face
M	Turning clockwise, the middle layer parallel to the U and D faces
S	Turning clockwise, the middle layer parallel to the F and B faces
twoDcube	original 2D state of Rubik's Cube
scramble	Disorder the 2D state of Rubik's Cube

3D Rubik's Cube	
Function	Application
cube	Draw 3D Rubik's Cube
cuberotaion	Rotate the Rubik's Cube based on command
bottomcorners bottomcorners1	Solve cubes on corners of the bottom layer
bottomcross bottomcross2	Achieve a cross on the bottom layer
bottomlayer	The last step for completing the bottom layer
flc	Solve cubes on corners of the first layer
secondlayer	Complete the secondlayer
topcross	Achieve a cross on the first layer
rotation3Dcube	The first way to rotate 3D cube

Table 1: Main part of functions and corresponding applications in Matlab

References

[1] Rubiks-cube-solver.com. (2017). Rubik's Cube Solver - Optimal solution in 20 steps. [online] Available at: *https://rubiks-cube-solver.com/* [Accessed 23 Apr. 2017].