Course Project Documentation

CS101 Project : **Pyraminx..the Story Untold**

TEAM : 318

Karan Chadha, 140070014

Kalpesh Krishna, 140070017

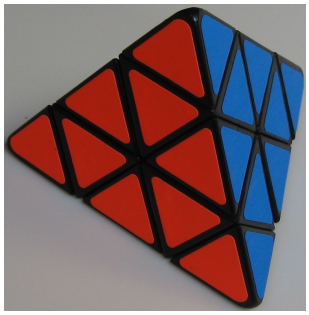
Harita Parmar, 140110014

Roshan Nayak, 140040097

TABLE OF CONTENTS

INTRODUCTION

The Pyraminx is a tetrahedron shaped puzzle loosely based on the famous “Rubik's Cube”. It was invented before the Rubik's Cube and is used as a puzzle in today's official speedcubing championships.
The motivation behind making a project on the Pyraminx was a fascination for cubing puzzles. Most people don't know that cube variations of the “Rubik's Cube” exist, let alone own them. That's the reason that speedcubing softwares are rather limited for these variations. Filling this gap was the major motivation behind making this project.
In this project we have attempted to not just provide the user with the faster solution to the Pyraminx Puzzle, but also provide a complete set of tools to help in learn how to solve it, get better at it, and even solve it on a computer.



A bit on Pyraminx cubing notation :-
Each possible move on the Pyraminx is represented by a letter and its inverse is represented by the letter followed by a '. There are eight moves possible on a Pyraminx and their corresponding inverses along with move ID as implemented by is:-

- Right – R (Its inverse is R') (Move ID – 4 and 5)
- Right Tip – r (Its inverse is r') (Move ID – 12 and 13)
- Left – L (Move ID – 2 and 3)
- Left Tip – l (Move ID – 10 and 11)
- Up – U (Move ID – 0 and 1)
- Up Tip – u (Move ID – 8 and 9)
- Back – B (Move ID – 6 and 7)

- Back Tip – b (Move ID – 14 and 15)

Color ID :-

- Blue – Colour ID 0
- Red – Colour ID 1
- Green – Colour ID 2
- Yellow – Colour ID 3
- White(used in algorithm finder) - Colour ID 4
- Pink (used in algorithm finder) – Colour ID 5

FOR USERS :- We have written all user instructions for operation in readme.txt. This is the project report.

PROBLEM STATEMENTS

We identified the following problems and have developed our project based on this :-

- To detect colors of stickers of pyraminx using image processing and send this data to computer.
- To find the fastest solution for the input configuration of the Pyraminx and make a display GUI for the same.
- To generate scrambling algorithm and to implement a GUI in order to solve the Pyraminx on a computer.
- To provide analysis on the user's solve.
- To generate cubing algorithms for tasks on Pyraminx and to implement a GUI for the same.

REQUIREMENTS

- HARDWARE :-
  - An android phone with a camera app. The phone must have atleast android 4.0
  - A black base Pyraminx
- SOFTWARE :-
  - A computer must have Allegro installed. We have created a bash script to do the same.

IMPLEMENTATION

1. To detect colors of stickers of pyraminx using image processing and send this data to computer.

We decided to implement the above requirement on an Android app for user's convinience. We made use of simple camera app integration in the Android app.
For the image processing, we processed the bitmap images produced by the camera app on the android app itself. We started off by "**colour thresholding**", which isolated all the pixels in the picture satisfying certain HSV conditions. There were a set of four conditions, one for each colour. The next step was "**blob detection**" which was implemented using a frontier based iterative approach. All blobs smaller than a certain area were removed. Once all blobs of a certain colour are detected, we find out its top most, right most, left most pixels and try to find whether they form an equilateral triangle. All blobs having a standard deviation less than a threshold were accepted as pyraminx stickers. These stickers were then organized in a logical manner using their coordiantes.

2. To find the fastest solution for the input configuration of the Pyraminx and make a display GUI for the same.

We made use of the "Breadth First Search" algorithm in order to compute the shortest solution to the Pyraminx. This solution treated each configuration of the Pyraminx as a node, and the key values were 4x9 arrays for 36 stickers respectively. For saving computation time, we assumed that all trivial tips were solved. This ensured that the total number of nodes in the graph were just under a million. For the adjacent list of the BFS algorithm, we encoded all changed made in the 4x9 array when one of the eight major moves were performed on the Pyraminx. To speed up the process of checking whether we had encountered a certain node,

we made use of Binary Search Trees instead of simple arrays. Here, key values were compared by using the 4x9 array, element by element. To balance the trees, we used Adel'son-Vel'skii-Landis (AVL) logic.This reduced **O(n)** computation time of the function to **O(log n)** We then solved the trivial tips by simply matching face stickers with tip stickers and displaying the corresponding move.
All this helped us generate solutions within 15 seconds, and on an average in 4 seconds.
To make a GUI, we made use of the Allegro library. Our main goal was to draw equilateral triangles placed correctly and to draw arrows showing the various moves on the face of the pyraminx. We used coordinate geometry to do this, and multiplied this by a scale factor to convert it to pixel coordinates. The arrows were drawn between mid points of stickers and the arrow heads were drawn using formulae derived using complex numbers.

3. To generate scrambling algorithm and to implement a GUI in order to solve the Pyraminx on a computer.

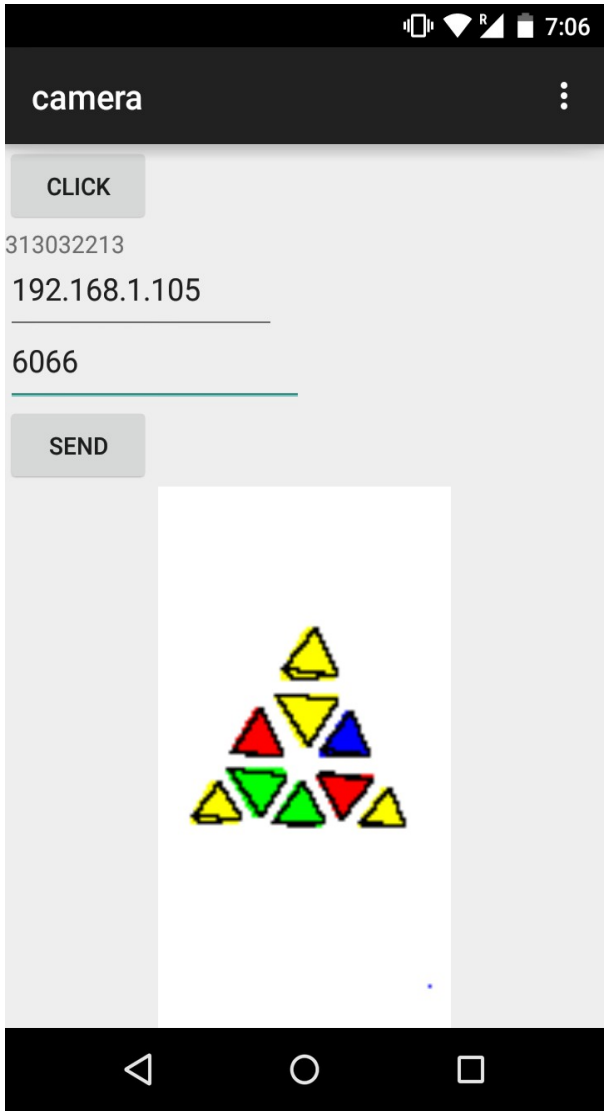
We used a random time based seed to scramble the Pyraminx. This generates a random sequence of move Ids. Care has been taken not to have two complementary moves next to each other (R and R') and also not to have two similar moves together (R and R which is same as R'). Also, each trivial tip were shuffled just once as further shuffles are redundant.
We passed this data via file handling to an Allegro interface which worked on the lines of the the solution display interface. It had 4 faces instead of just one. Each face was similar but with differences in sticker colours and in orientation and position. We displayed the corresponding scramble on the Pyraminx faces drawn. There are buttons which allow the user to perform moves on the Pyraminx. Every move is stored in a text file used later for analysis.

4. To provide analysis on the user's solve.
We used the text file generated by the above GUI and read it to determine the initial state of the Pyraminx and the set of moves performed by the user on it. We repeatedly reused our BFS code to display the shortest solution after each of the user's moves, as well as the initial shortest solution. This is in some sense telling the user where he went wrong while designing his solution.

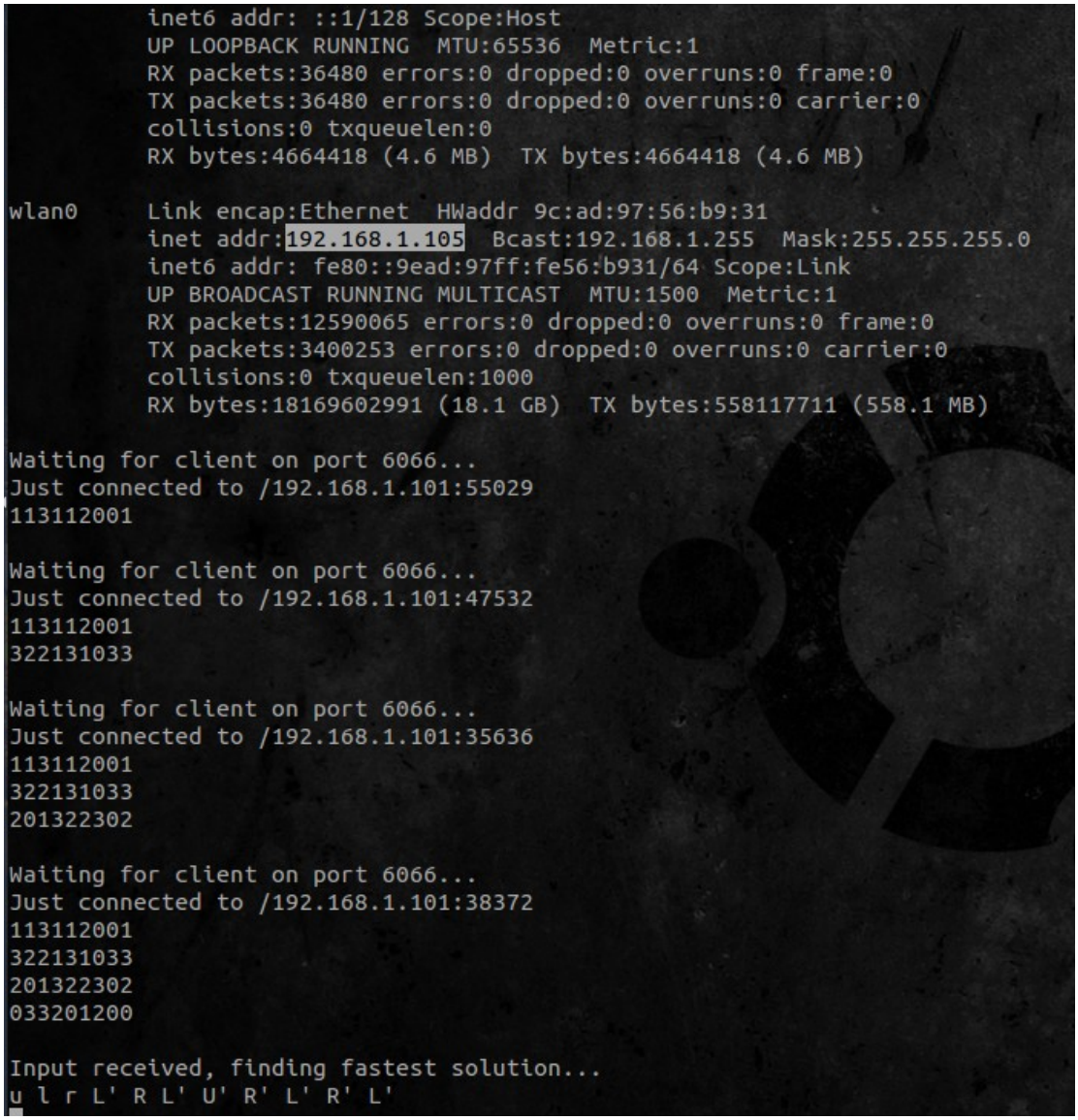
5. To generate cubing algorithms for tasks on Pyraminx and to implement a GUI for the same.
We started off with the GUI used in the scrambler GUI. We modified the colours to include a “pink” sticker which displays the selected sticker and a “white” sticker which displays an unimportant sticker in the algorithm. We modified the buttons to have “Left” and “Right” buttons to move the pink sticker, and five colour buttons to assign colours to the Pyraminx. We then input our start configuration using these five colours. “White” tells the BFS algorithm that this sticker does not matter, or any colour would do here eventually. This is essential in algorithms as we often don't care about certain parts of the cube while solving it. Once done, the algorithm finder computes the shortest solution (the “algorithm”) from this state to a solved cube and displays it using the above utility.

TESTING STRATEGY AND DATA

Image Processing :-
As input to the Android app, we click photographs by pressing on the “click” button. Here is a sample input and output :-

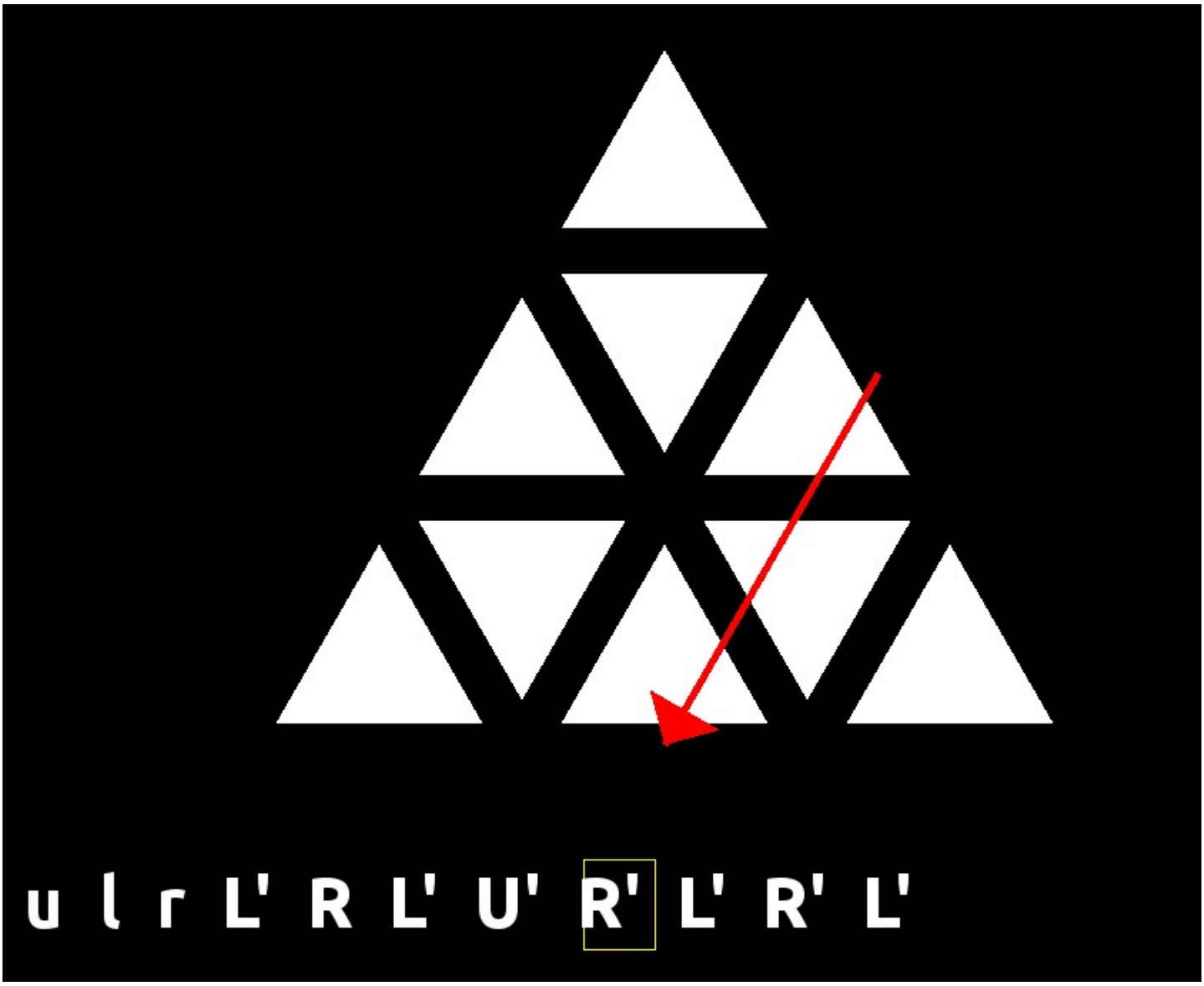


We are sending this data to a Java server. The input is the string of numbers displayed under “click”. It is the colour ID of the various colours as described in the introduction. Here is the output :-



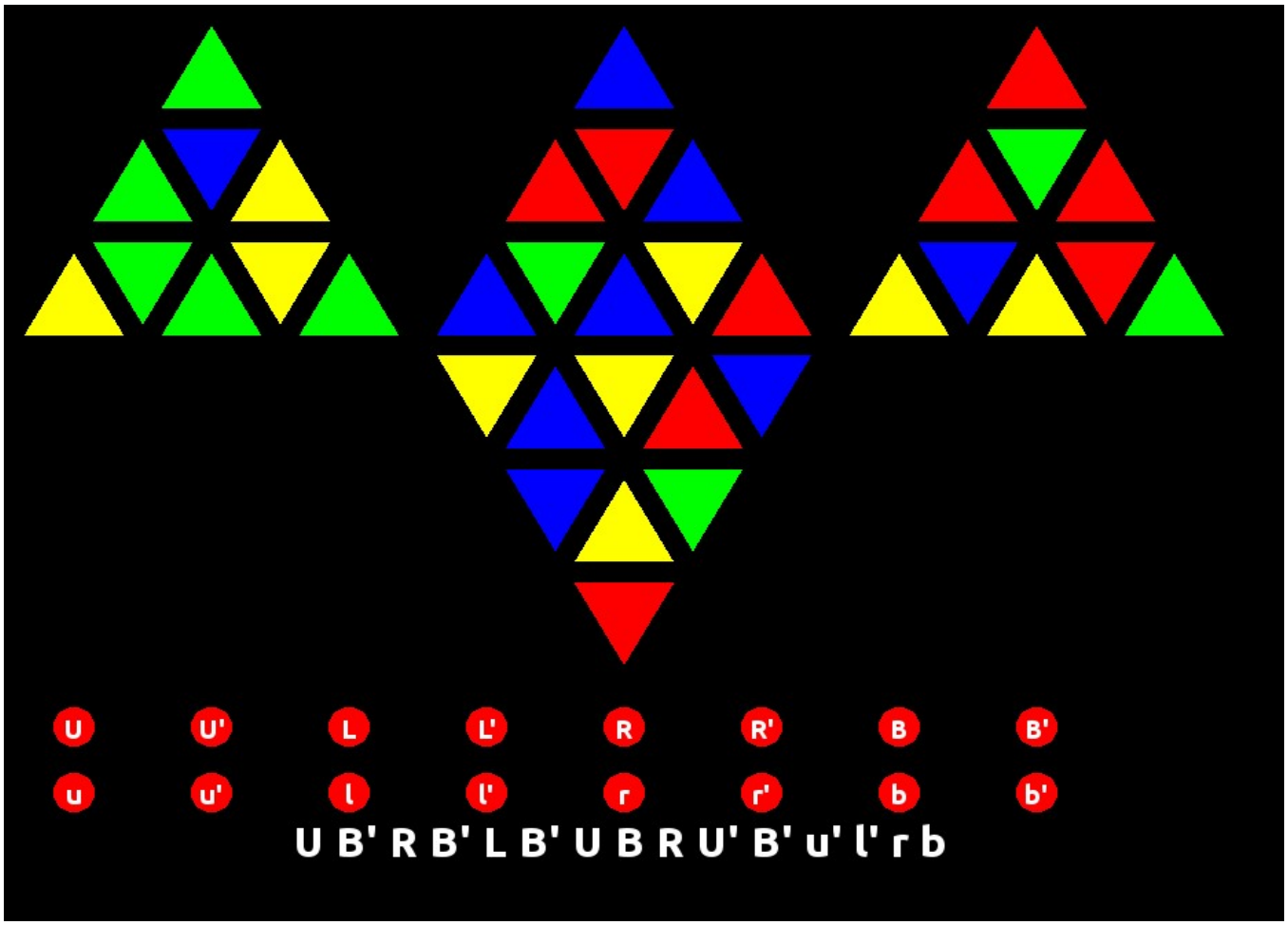
As one can see, the input is received and stored in a text file “test.txt”. This is read by another file and we get the shortest solution, which can be seen at the bottom of the above image. We store this in the text file “solved_moves.txt”

We have also shown this output in a GUI format. This is an Allegro GUI which reads “solved_moves.txt” and prints a graphical form of the solution.



Random Shuffler Mode

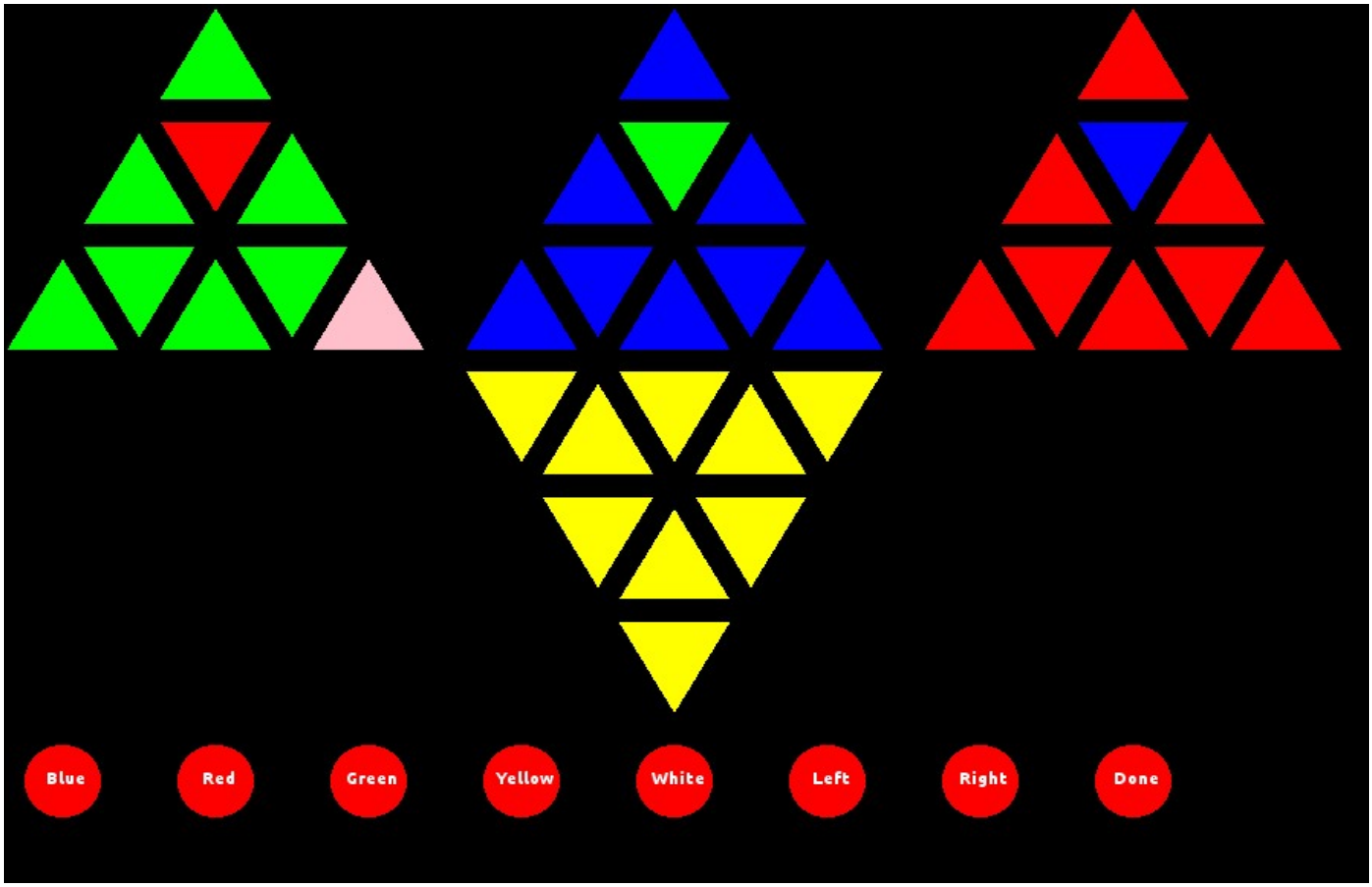
A random shuffling is generated. The user attempts on solving the cube in the following interface :-



The shuffling algorithm is show at the bottom of the window.
The user can try to solve the Pyraminx for himself using the buttons shown below the faces.
The output is the analyisis of the user's try which is the shortest solution after every move the user has made:

Algorithm Finder :-

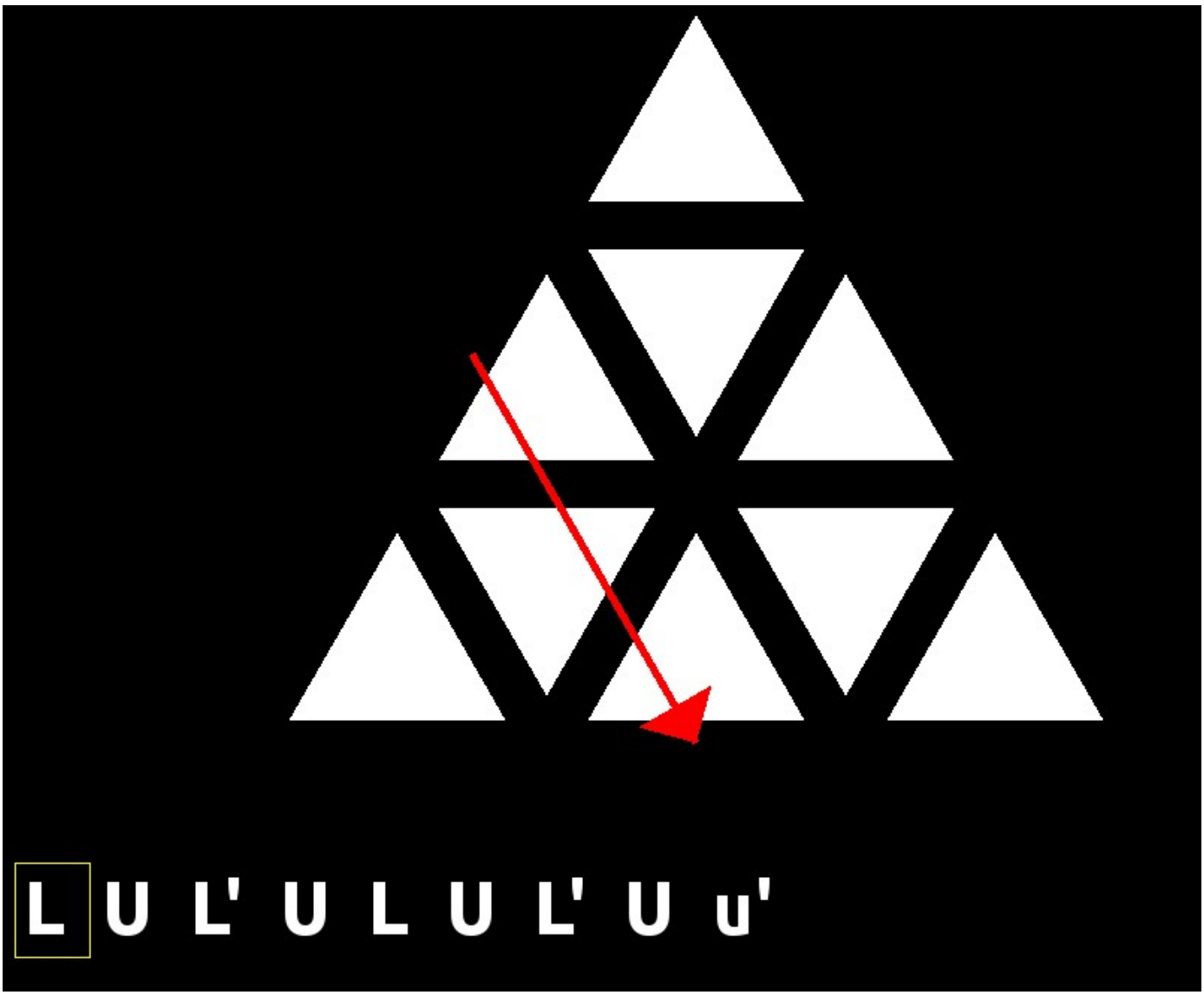
We have made an interface in order to generate an input conviniently. This is the interface :-



The user may navigate the pink “active” sticker and select the colour he desires in that position. He can choose one of five colours.

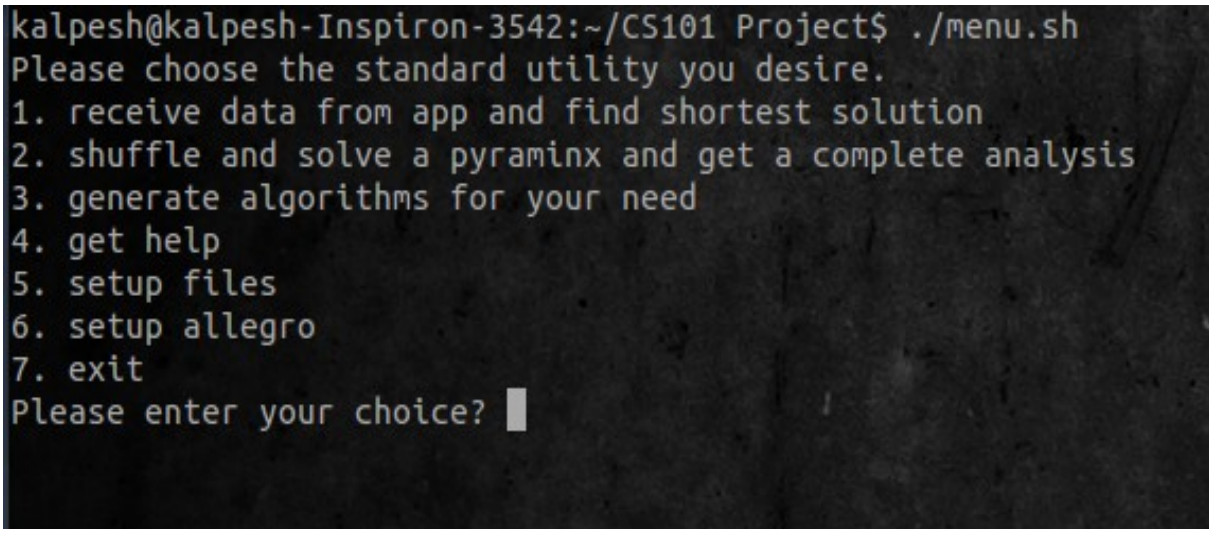
This colour data, a 4x9 array is stored in “test.txt” and is read by a C++ file to compute the shortest path from this to a solved cube.

We print this solution in “solved_moves.txt” and use the same display GUI. The solution to the above state is :-

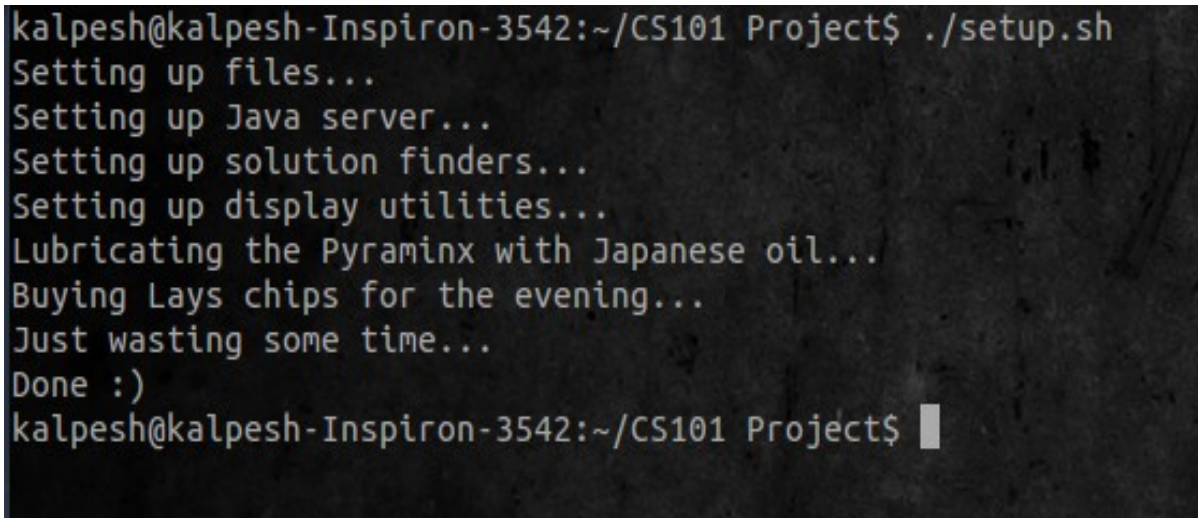

L U L' U L U L' U U'

Bash Scripts :-

menu.sh



setup.sh



DISCUSSION OF SYSTEM

A) What all worked as per plan?

1. Android App:

The app worked perfectly as per plan. It was integrated with the mobile camera . The image processing was initially done on the PC in java. Later it worked perfectly fine on the phone also after making a few changes in the classes used , since the android code is java itself witha few changes in the functions.

2. Data Transfer:

We used sockets in java for data transfer. It worked well with both WiFi- Hotspot of the mobile and also the WiFi connection using routers by entering in the app the appropriate IP address decided by the WiFi and port number initially decided by us and stored at the recieving station.

3. Image Proccessing :

The Image proccessing part worked properly after changing the blob detection code to an iterative one from a recurrsive one, which was giving the stack overflow error.Also changing to the HSV format from the RGB for color detection gave us better accuracy.

4. Breadth First Search Algorithm :

The BFS algorithm initially took a lot of time to be executed (not finished even after a day of the code running). This was because it is a linear order in time algorithm and it required about 10^14 iterations to be done since it had to compare 4x9 arrays before putting them into the graph . So we added the idea of using BST(Binary Search Trees) using AVL to make the trees balanced, which reduced the order from linear to log(n). Hence the code ran fast enough now and the solution was genarated correctly.

5. Tips-solving algorithm:

The tips solving algorithm worked perfectly as desired. It had to be done differently than the BFS since it is trivial to solve the tips and any solution to the tips will be the shortest solution itself.

B) Work done besides points in SRS

1. Menu using Bash scripts:

We basically made a menu using bash scripts from which we can switch between the different modes of the project to explore each end without running the file again and again.

2. Setup Files:

We also made two setup files which would initially compile the code and also for installing the graphic library ALLEGRO which we have used to display the graphics on the screen.

(C) Changes made in plan:

1. Graphics:

We initially planned to use the OpenGL(GLUT) library, but we later switched to the ALLEGRO library since Open GL(GLUT) couldn't be configured to include buttons.

FUTURE WORK

- We can improve algorithm finder by adding links between stickers .
- Implement 3-D pyraminx .
- Change menu bar from a bash script to a GUI .
- Improve interface for Android app.
- Convert **setup.sh** to **make** file
- **Transfer entire project to an android app or a web application for portability. Could be a useful app for speedcubers.**

CONCLUSION

The project idea was initially basically to find the shortest solution the Pyraminx, which was very efficiently done. Also a graphic interface was made so that the user can try to solve the Pyraminx himself on the computer itself and get the analysis of each of his step so that he can see where he can improvise. Since we have included the Algorithm Finder in the project we have enabled the user to himself make a few useful algorithms for himself to solve the pyraminx in an efficient way. Also, we have included the solution display so that the people who don't know the cube notation can also solve the Pyraminx. To conclude, we have made a complete package of the pyraminx enough to make a person from a beginner to a prodigy in speed-cubing the pyraminx.

REFERENCES

Android App :- http://developer.android.com/index.html
BFS Algorithm :- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-13-breadth-first-search-bfs/
BST/AVL Data Structures :- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-6-avl-trees-avl-sort/
Socket Programming :- http://www.tutorialspoint.com/java/java_networking.htm
Allegro Setup :- https://wiki.allegro.cc/index.php?title=Install_Allegro5_From_Git/Linux/Debian
Allegro Basics :- https://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials
Allegro Functions :- http://alleg.sourceforge.net/api.html
Bash Scripting :- http://linuxconfig.org/bash-scripting-tutorial
C++ Programming :- CS101 Lectures, http://www.tutorialspoint.com/cplusplus/, http://www.cplusplus.com/reference/