RUBIK’S CUBE SOLVER

1. INTRODUCTION

Rubik Cube is a 3-D combination puzzle is not nearly as impossible to solve as it at first appears. Rubik Cube solver is an automated machine that takes the input, perform the algorithm and gives the final solved cube in finite number of steps.

Rubik’s cube is designed by Erno Rubik’s because he wants that his students understand the 3D problems in a better way. In 18th century it became so much famous in Hungary. Once you look at the scrambled cube you know what you need to do but without the proper instructions it is nearly impossible to solve the cube.

Rubik's cube is a cube with edges approximately 56 mm (2.2 in) long. It consists of 26 smaller cubes; one side is made up of 3x3 such cubes. If you take an ordinary cube and cut it as you would a Rubik's cube, you are left with 27 smaller cubes. Rubik's cube does not contain the cube from the very middle. In fact, the smaller pieces of the Rubik's cube do not really have a shape of a cube. We call these 26 small pieces cubies. The implementation of this project is using Arduino, Python and PySerial.

Informal procedures to solve the Cube exist and are used as the starting point. A popular solving algorithm is decompose it into its core components. These components are composed to formally describe the solving algorithm. An appropriate representation of the Cube is derived with considerable detail. The efficacy of this representation is evaluated in relation to its ability to record the state of a Cube and allow for the intuitive manipulation of that state. The digital description of the Cube was manifested visually in 2 and 3 dimensions. The utility and features of these visualization are discussed. The solving algorithm was formally implemented and the key features of this implementation are demonstrated. The performance of this solver is evaluated in terms of speed and length. The solver is linked to the real world by a scanner application. This allows a real-life Cube to used as input to the solver. The performance and reliability of this application are discussed.

To conclude we note that even complex problems can be formalized and solved. The skills and experience gained from this project are acknowledged and areas for further investigation are suggested.

1.1 OBJECTIVE OF THE PROJECT

Although four decades have passed since The Rubik’s Cube was invented by Erno Rubik, few puzzles exist which are as iconic, renown and as recognizable. Since its inception in 1974 The Rubik’s Cube (hereafter also ’the Cube’) has enjoyed widespread popularity[Rub], entertaining and fascinating generations of children and adults alike. So what is it about the Cube that makes so popular? For the author the appeal is rooted in the stark contrast between the simplicity of the Cube itself and the complexities of solving it. Here we will examine how to computationally solve the Cube. The main topics that we will consider are how to formally model the Cube, how to visualize it and finally how to solve it.

The main purpose of humans creating a robot is to make their daily life simpler and time efficient. Since machines are so much more efficient than human and solving this cube using proper machine make our task a little bit easier. Now a day there are many robots which can play "chess”, "go" and this shows a milestone in the robots achievements. Solving the cube in the same way as is a bit difficult for ours. There is google ai which has beaten the smartest player of go in china. Solving a Rubik's cube is beneficial for future generation because implementation and augmentation of such machines can teach them the optimized code rather than all the techniques to solve the cube. (43 quintillion ways). In our project we are solving the Rubik’s cube but the faces of the cubes are entered manually by the person who is operating the program made in python but in future we can take this project to another level by using the appropriate sensor for this so that it can take the inputs for the face all by itself.

1.2 Description of the Project

An IOT(internet of things) device consisting of Arduino micro controller. It has 2 servo motors to control the arms of the robot to control Rubik's cube's movements. An optimized Arduino code for faster processing of results and a python interface for the user to input the faces of scrambled cube. The Machine takes the input and function according to the algorithm by rotations and moves for the final solved Cube. The inputs of the faces which we are adding by user can be taken by the machine also using a sensor and we will integrate the sensor in the mere future for the better functioning. The machine follows M2 Old Pochmann Algorithm that says, it is a basic method for blind solving that utilizes swapping only a few pieces at once using common algorithms found in methods like CFOP. It was created by Stefan Pochmann. You set up corners to a receiving spot and swap it with the bank piece. I highly suggest learning this before going into M2 method for edges.

2. SCOPE OF THE STUDY

To study about the Arduino
To study the python, PIP, pyserial and OpenCV
To study the Algorithm used for solving a puzzle
To study all dependent parameters
To study the feedback of the existing solving Technologies
To study the feedback of the implemented Technology

In the present scenario due to major changes in technical and commercial changes there are lots of default back drops in puzzle solving technologies. Even counter measures are same in the range of development but still lacking to scope up with the break-ins. This helps to study the counter measures and essential parameters to implement the technology.

The study helps to counter the implementation of the algorithms for the tricky puzzles takes became moderate because of the advanced technology and the programming tools. Solving a Rubik's cube is beneficial for future generation because implementation and augmentation of such machines can teach them the optimized code rather than all the techniques to solve the cube.

3. EXISTING SYSTEM

Cube-solving robots have been in existence for many years, with origins in the early 2000’s. One of the earliest documented designs was constructed with the Lego MindStorms robotics platform, which features motors and a small microprocessor. The robot, which took months to complete, solved cubes in 15 minutes with 95% accuracy. Lego-based designs are still popular today, and have broken multiple world records for solving speed. Figure 3 shows a Lego design from 2012 that solved a cube in only 5.2 seconds, a world record at the time. Although Lego designs have historically been dominant, many other structural implementations have been used in recent years, ranging from popsicle sticks, to custom machined and welded metal. Aside from physical design, existing robots all vary in their software infrastructure, peripherals, and hardware-software interface. Some projects place emphasis on advanced programming to make up for simplistic hardware, while others focus on system portability or overall speed. An important influence on these design considerations has been time: while hardware abilities have remained consistent over the last 15 years, there have been many changes in the enabling technologies for software development that are made evident by consistently innovative software implementations. Constant improvements in general computing performance have led to faster algorithms and faster robots.

3.1 PROPOSED SYSTEM

In the real world there are many ways to solve the Rubik's cube. The main way is by using the hands and the steps to solve it will be memorized. And the other option is implemented by technology and a physical robot. There are many existing machines available based on the same problem. It has many disadvantages like energy consumption and time consumption.

- Our proposed system gives faster and quick moves to get the output
- For this system we use Arduino, Servo motors.
- Arduino uno and python are the prime software and is merged with the hardware equipment for the project to execute.

4. PROBLEM ANALYSIS

4.1 PROBLEMS IDENTIFIED

- The input is manually given form computer by the user
- The project cannot detect the faces automatically
- The system uses both python and Arduino

4.2 PRODUCT DEFINITION

In to-day's daily routine finding the algorithms for the tricky puzzles is one of the important phenomena, so as to have a puzzle finding environment we have to propose a new systems in which the evolving IOT is included since IOT is growing faster in the technical world. This algorithm is monotonic in the sense that after each iteration we progress towards a solution. This is a consequence of the guarantee that at each stage at least one of the intermediate patterns will be matched. By implication the algorithm must eventually terminate with a solved cube, however it is impossible to determine in advance how many rotations the solution will involve. In fact it is not possible to determine in advance if a given Cube is actually solvable, that is of course other than to run it and see if it terminates within a sensible time frame.

4.3 FEASIBILITY ANALYSIS

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis are

➢ ECONOMICAL FEASIBILITY

➢ TECHNICAL FEASIBILITY

➢ SOCIAL FEASIBILITY

4.3.1 ECONOMICAL FEASIBILITY:

A system can be developed technically and that will be used if installed must still be a good investment for the organization.In this economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceeds the costs.

The system is economically feasible but it does require addition hardware or software. Since the interface for this is developed using the existing resources and technologies.

A Functional requirement defines a function of system and its components. A function is described as a set of inputs, the behaviour and the outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish.

The key goal of determining functional requirements" in a software product design and implements is to catch the required behavior of a software system in terms of functionality and the technology implements of the business processes.

4.3.2 OPERATIONAL FEASIBILITY:

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirement. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following: -

- Sufficient support is to be guarantee for the management from the users.
- The system should be used and work properly even if it is being developed and implemented.
- To identify if there would be any resistance from the user that will undermine the possible application benefits.

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirement have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

4.3.3 TECHNICAL FEASIBILITY:

The technical issue usually raised during the feasibility stage of the investigation includes the following:

- The proposed equipment should need to have the technical capacity to hold the data that is required to use the new system
- The proposed system should also need to provide the adequate response to inquiries, regardless of the number of times user tries to login into the device.

There should be a possibility of system to be upgraded if developed Earlier no system existed to enter to the needs of 'Secure Implementation system', The current system developed is technically feasible. Thus it provides an easy access to users. Therefore, it provides the technical guarantee of accuracy, reliability and security. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth exists for providing a fast feedback to the users irrespective of the users using the system.

4.4 PROJECT PLAN

The aim of the project is to complete with a low budget and efficient model of Rubik cube solving robot which ensures faster performance and reliability and easy to use.

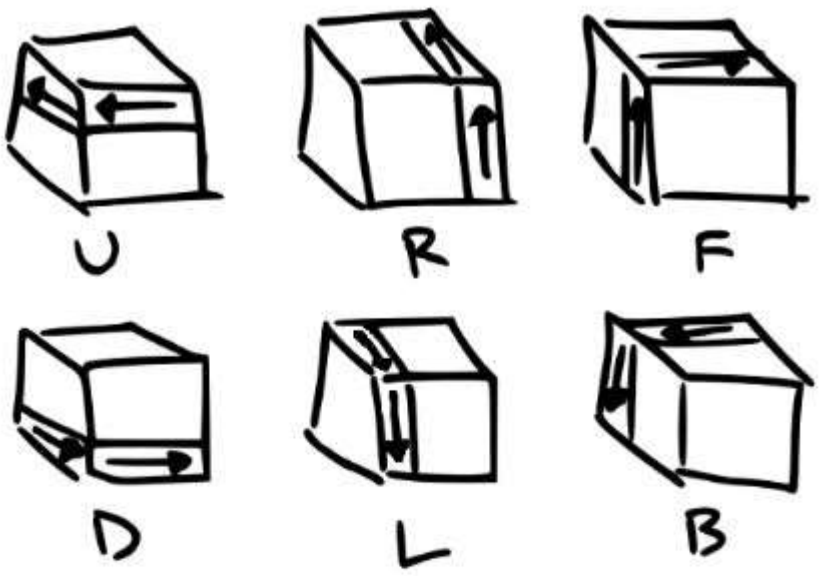


Fig 1: Moves of the cube

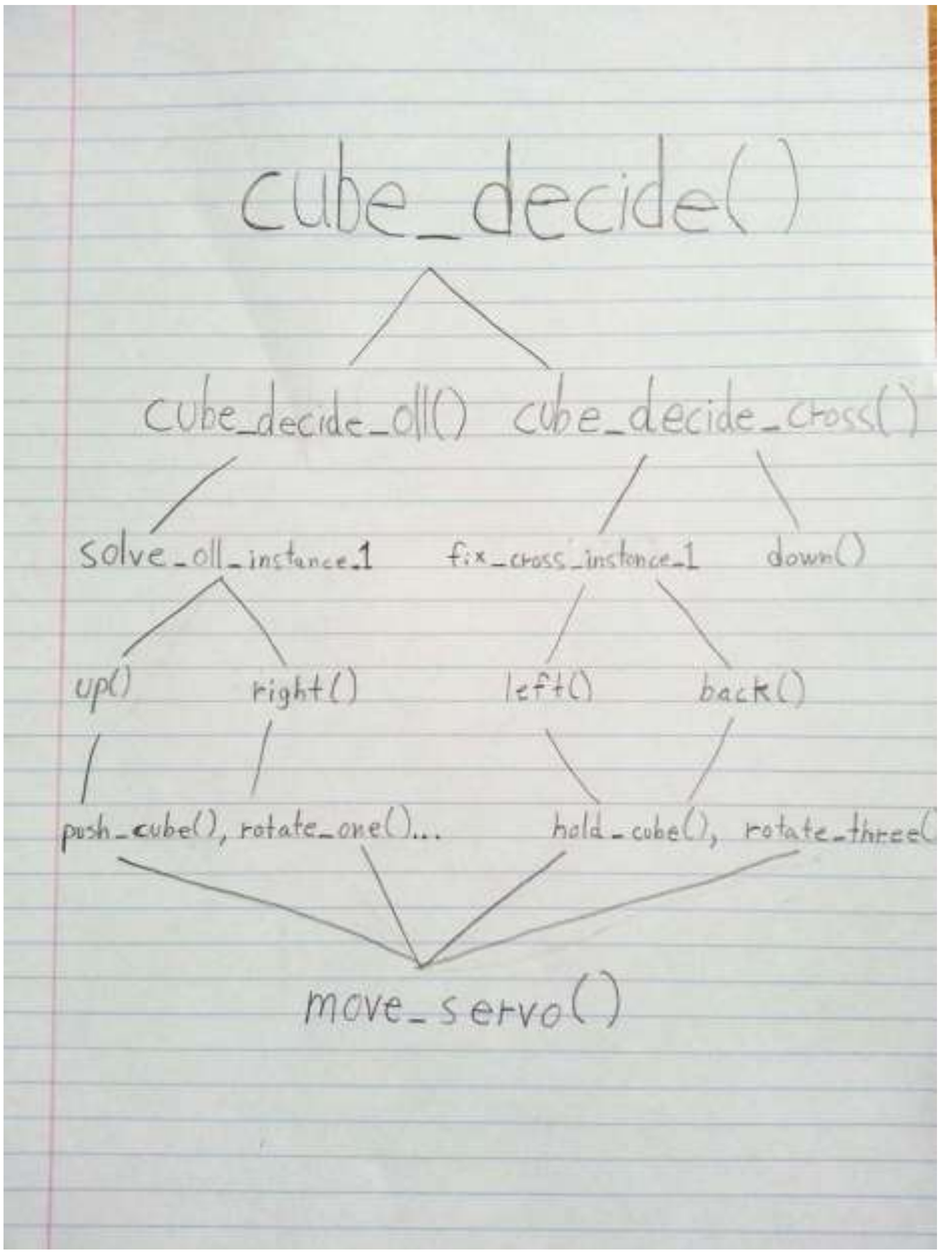


Fig 2: Steps followed by the Algorithm

5. SOFTWARE REQUIREMENT ANALYSIS

It is a outline of a software system that is to be used and developed according to the requirement of the project. It lays out the functional and also non-functional requirements which help the execution of our project. In this particular project software acts as third-party developing agent along with microcontroller and RFID module. Basically these software requirements specifies and establishes the basis for an agreement between user and developers. It permits a continuous assessment before we start the design and later have some changes. Used with care, Software requirements can help software from failing during implementation. These software requirements specifications helps to list out the necessary requirements that are required for the development of the project. To derive the requirements the user need to have proper understanding of the project that is being used.

The Main goal of this software requirement analysis are

1)To describe the quality and scope of project

2)To provide the inputs to software developers

3)To provide the Framework to test the project/software

4)To give proper idea about the steps involve

5.1 FUNCTIONAL REQUIREMENTS

5.1.1 Arduino IDE - (Algorithms in C++)

5.1.2 Python 2.7+ and Tkinter - (GUI)

5.1.3 PySerial

5.1.4 Arduino UNO Board R3

5.1.5 Servo Motors

5.1.1 ARDUINO IDE

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers -

students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

5.1.2 PYTHON AND TKINTER GUI

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk.

Tkinter is not the only Gui Programming toolkit for Python. It is however the most commonly used one. Cameron Laird calls the yearly decision to keep Tkinter "one of the minor traditions of the Python world". With the help of Tkinter we have created the Gui for our project in which we have entered the details of all the faces regarding it. For creating Gui using a Tkinter library we have simple steps like import the Tkinter library create Gui application window add the required widgets than add the event for each module.

5.1.3 PYSERIAL

PySerial is a Python API module to access the serial port. PySerial provides a uniform API across multiple operating systems, including Windows, Linux, and BSD. We will use the PySerial API to access a serial port. Destructor in our classes will basically close the port when the serial port instance is freed. Read size bytes from the serial port. If a timeout is set it may return less characters as requested. With no timeout it will block until the requested number of bytes is read. Flush that has been used in our program will wait until everything is written.

5.1.4 ARDUINO UNO BOARD

The Arduino Uno is a microcontroller board based on the ATmega328. Arduino Uno R3 specifications are ATmega328 microcontroller, operating voltage at 5v, input voltage 7 to 12v, input voltage limit up to 20v, digital I/O pins 14, analog pins 6, DC current 40mA, flash memory 32KB including 0.5KB used by boot loader. SRAM of 2KB, EEPROM of 1KB and clock speed of 16 MHz some of the Features of Arduino UNO are power: can be USB connection or external power supply, with 7 to 12 volts recommended. The Arduino UNO provides power pins for other devices, the variants are 5v 3.3v and vin IOREF pin for optional power.
Arduino Uno is a 2KB of SRAM and 1KB of EEPROM (Electrically Erasable Programmable Read Only Memory). There are various input and output pins where 14 of them are digital pins with serial transfer and external interrupts and PWM (Pulse Width Modulation) pins and 6 analog pins. Arduino differs from all the preceding boards which does not use the FTDI USB-to-serial driver chip.

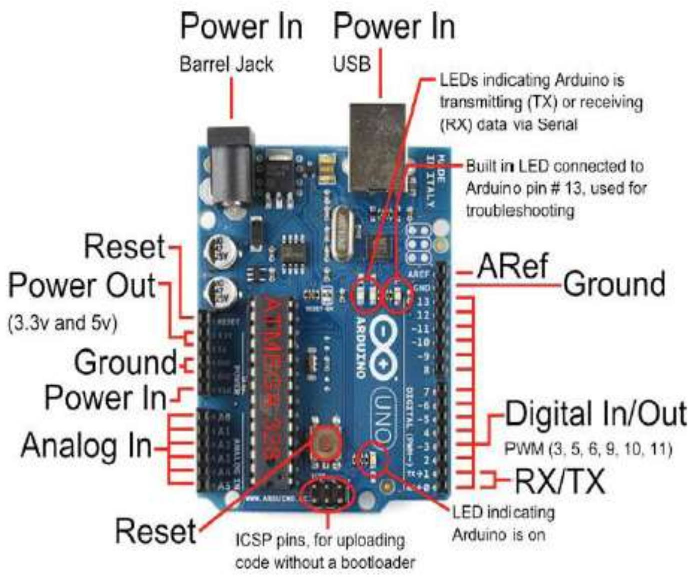

Power In
Barrel Jack
Power In
USB
LEDs indicating Arduino is transmitting (TX) or receiving (RX) data via Serial
Built in LED connected to Arduino pin # 13, used for troubleshooting
Reset
Power Out
(3.3v and 5v)
ARef
Ground
Ground
Power In
Analog In
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
RX/TX
Reset
ICSP pins, for uploading code without a bootloader
LED indicating Arduino is on

Fig 3: Pin Configuration of Arduino

5.1.5 SERVO MOTORS

A servo motor is an electrical device which can push or rotate an object with great precision. If you want to rotate and object at some specific angles or distance, then you use servo motor. It is just made up of simple motor which run through servo mechanism. If motor is used is DC powered then it is called DC servo motor, and if it is AC powered motor then it is called AC servo motor. We can get a very high torque servo motor in a small and light weight packages. Doe to these features they are being used in many applications like toy car, RC helicopters and planes, Robotics, Machine etc.

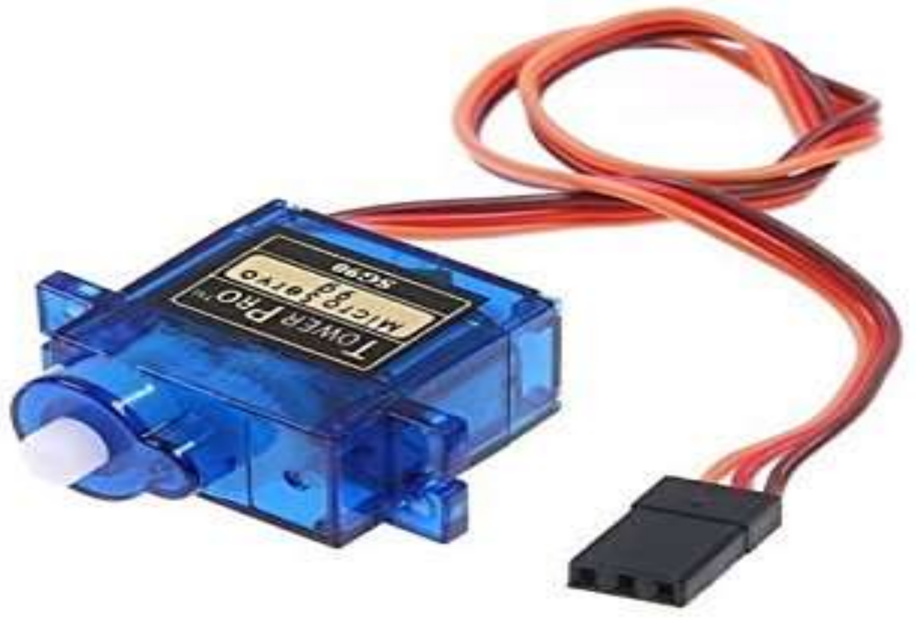


Fig 4: Servo motor

5.2 NON-FUNCTIONAL REQUIREMENTS

5.2.1 DURABILITY

The Durability is also one of the important factor of our project since it takes a lot of time in solving a single cube. So it should be durable enough so that it last upto the day when we have to present it to the concert authority the material which is used servo motor is basically a simple dc motor and sometimes it can be break so we have to consider these liabilities also.

5.2.2 MAINTAINABILITY

If in any case there occur any failure we should be able to restore to the operational status within a specific time and this is also a challenging situation for all of us.

5.2.3 OPERABILITY

When we are designing the Rubik's cube solver it should be easily operable the model which we are developing is can be operable through any person whom we are demonstrating the project. So that everyone can have a hands on experience using this.

5.2.4 PERFORMANCE

Performance is the biggest issue here as due to lack of proper hardware the model won't we working fast it takes a lot of time to solve the cube and hence it's a key area in our model in future we will be able to modify and use our hardware in the efficient way as much as possible.

5.2.5 REUSABILITY

Our project is totally reusable since we can easily change the code and improve the algorithm for the better performance and faster operations. The hardware which we are using is easy to modify.

6. DESIGN

System Description:-

The Machine that is built is implemented on the system that has the following configuration and framework:

- Framework: PY Serial
- GPU: Integrated Graphics or above
- Programming Languages used: Arduino and Python 2.7x

The final result is a cost effective machine that runs using Arduino and moves using only 2 servo motors.

Assumptions and Dependencies:-

- The user has a bare minimum computing device and set of these software's:

- o Arduino IDE(To edit and upload the code onto the Arduino)
- o Tkinter - GUI (Normally pre-installed with Python)
- o PySerial
- o Python 2.7x

• The user is expected to input the faces of the cube in the exact order that will be mentioned in the GUI interface.

• The user needs to be familiar with uploading code into an Arduino device.

Functional Requirements:-

The primary requirements for the proper working of the Rubik's cube solver are:-

A. Parameters of the faces of the scrambled cube.

B. Placing the cube in correct orientation.

6.1 E- R DIAGRAM

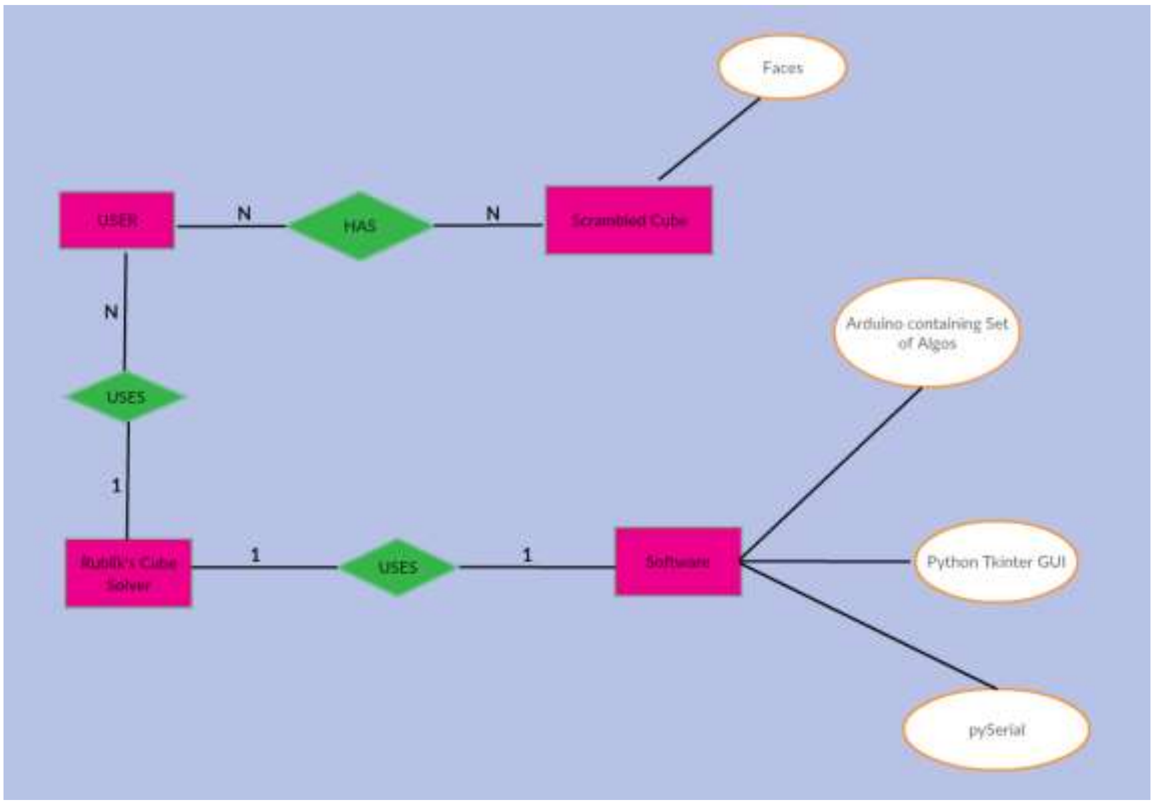


Fig 5: E-R Diagram for the System

- User is an Entity and he has a scrambled cube with him.

- Rubik’s Cube solver is an Entity, Software is another Entity that has attributes Arduino Containing set of Algorithms for solving, Python Tkinter GUI and PySerial.
- The User uses the Rubik’s cube solver for making an unscrambled Cube.

6.2. DFD DIAGRAMS

DFD-0

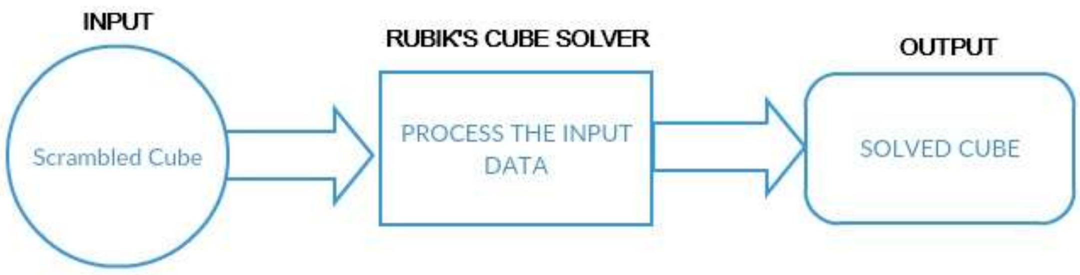


Fig 6: DFD -0 for the system

DFD -1

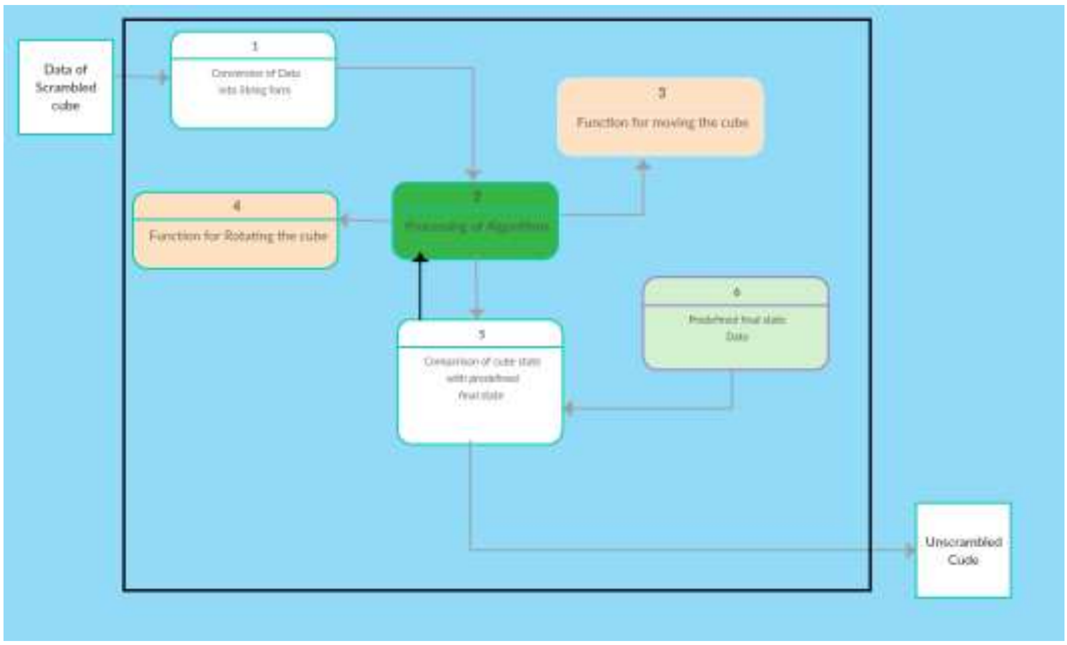

Data of Scrambled cube
1
Conversion of Data into String form
3
Function for moving the cube
4
Function for Rotating the cube
2
Processing of Algorithm
6
Predefined final state Data
5
Comparison of cube state with predefined final state
Unscrambled Cude

Fig 7: DFD - 1 for the system

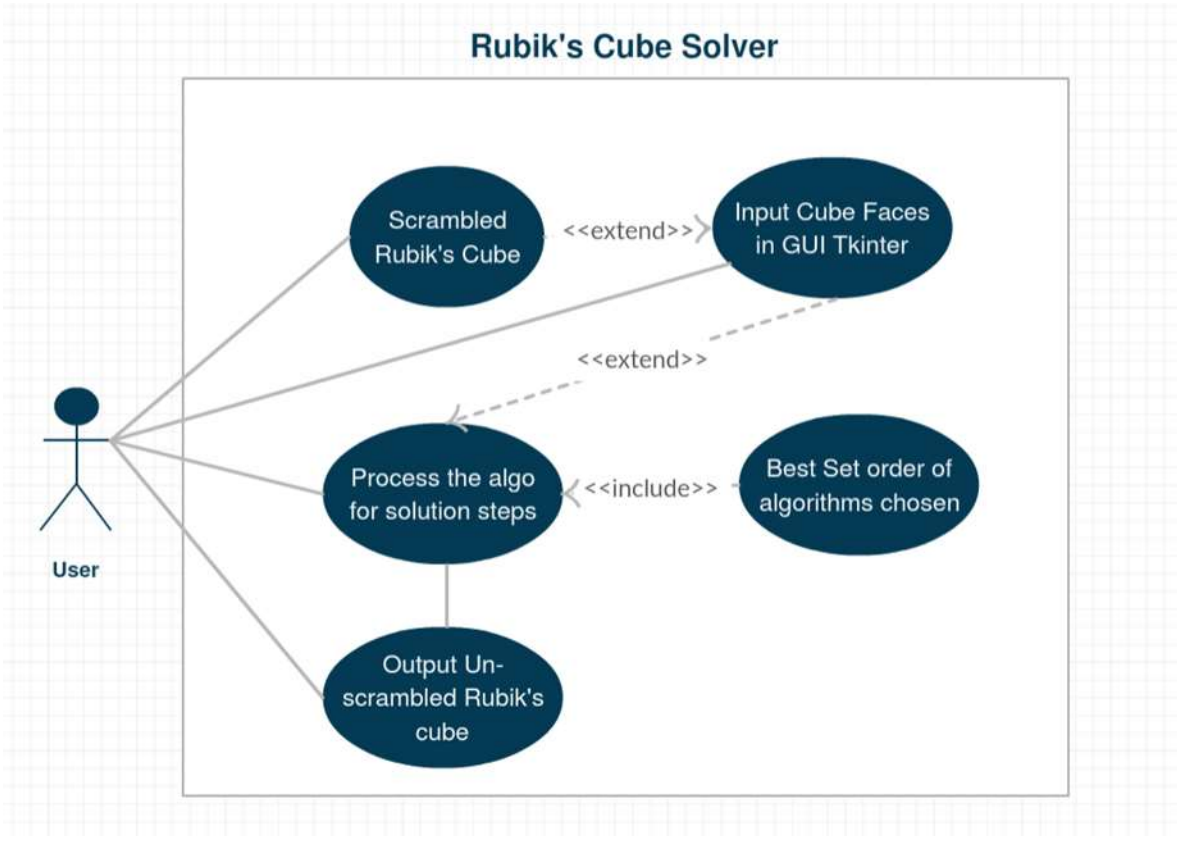
Level 0:

- The first entity contains the information of the user and the input data to be inputted.
- The second one process the data came from the first one using many certain rules.
- The third entity is the output of the processing part of rubik's cube solver.

Level 1:

- The inputted data is converted into string form for processing.
- This string is passed to the Algorithm processor which decides the moves and rotations to be implemented by comparing the current cube state with the predefined final state cube.
- After performing the rotations and moves the algorithm send the completion command to the output part.

6.3. USE -CASE MODEL:



Rubik's Cube Solver
Scrambled Rubik's Cube
<<extend>>
Input Cube Faces in GUI Tkinter
<<extend>>
User
Process the algo for solution steps
<<include>>
Best Set order of algorithms chosen
Output Un-scrambled Rubik's cube

Fig 8: Use – Case Model

Use Case Model Description:-

In the shown use case model for the Rubik's Cube solver, User is considered as the main actor who is in control of the machine or uses the machine.

- The Scrambled Rubik’s cube module is the one where the user puts the scrambled cube in its allocated base platform.
- In the Input Cube Faces module, a python Tkinter module is used for inputting the 6 faces of the cube in its scrambled form.
- The aforementioned modules are connected to the processing module, where all the different algorithms are checked upon.
- The processing module take help of best set order module to make a map of the optimized way of solving the cube from inside out.
- The output module is the one where the cube gets unscrambled and the final result is produced.

7. TESTING

After the completion of software and Hardware is prepared, a test plan will be implemented to make sure that product is meeting the needs that are required to be performed. The main objective of this testing is to delete all the errors in the product.

The varying color qualities of each cube will make error more likely. However, this error has to be held to a great minimum or the code with incorrectly interpret the cube resulting in a failed attempt of solving it. After, the cube has been identified and mapped to each matrix properly in a unsolved state it will now be tested against lighting conditions again. Light can cause color values to spike so making sure the code handles this conditions well will increase the success rate of it.

The solving algorithm will be tested to see the percentage of times it can solve the Rubik’s Cube successfully. The algorithm itself should have the least percentage of error because the pixy camera has moving parts as well as the hardware has moving parts.

Hardware Communication Test Hardware communication test is essential because if the code cannot tell the robot to move properly then it will result in the Rubik’s Cube not being solved. The fist hardware communication test would be making sure the python and Arduino is available with the serial communication between them.

Finally, the timing of the arms to turn after the code tells it to is incredibly important. The timing of each arm should be measured and accounted for when the code changes algorithms into arms movements. The arm movement must be in sync or an arm could get jammed trying to go at the same time as another arm causing an error in solving the cube.

7.1. FUNCTIONAL TESTING:

7.1.1 Software Testing:- the software we used in this project is Arduino and python along with Tkinter Library. Our test begins by entering null code into the software and then executing it with series of different and appropriate commands so that the output should be given error.

7.2 STRUCTURAL TESTING:

7.2.1 Hardware Testing: The hardware consists of microcontroller, Arduino board, Servo motors and Bread Board. The testing was done for this using the power supply. When we upload the sketch into the Arduino board, it must execute the hardware by moving the handle front and back also the base part will be rotated 360 degrees.

7.3 LEVELS OF TESTING

7.3.1 Motor Testing:- The motors attached to the hardware is tested individually by uploading the sketch to it and see the movements of the servo motors.

7.3.2 GUI Testing:- The GUI part i.e., taking the input from the user to get the faces of all sides, these task is tested using the python shell

7.3.3 Unit Testing:- The coded programs are individually tested with every device based in prepared test data. Any unusual happenings will be noted and debugged wherever the error corrections are handled. The result of each device is shown in above lines. Keypad is physically connected with Arduino based on the pin configuration. All characters are typed and successful message is displayed.

8. IMPLEMENTATION OF THE PROJECT

The core of the solver subsystem is contained in two main classes: RubiksSolver and RubiksSolution. As covered in section 2.1 the thrust of the solving is algorithm is in matching patterns and then applying a certain sequence of moves in response. It is these patterns, their respective responses, and the match-execute logic that is contained in RubiksSolver. The output of this process is fundamentally a sequence of moves that takes the Cube in its current state to one in a solved state. This sequence is referred to as a solution and is represented by a RubiksSolution.

We aggregate moves represented by RubiksMove to form a solution. At its core the RubiksSolution class provides methods for accumulating such an aggregation. We can manipulate this aggregation in various ways, including inverting it and composing it with other such aggregations. Importantly the RubiksSolution class can associate notes alongside and between each move. This is intended to annotate the solution with helpful explanations of what is happening at each stage. The explanations are dynamically generated within the RubiksSolver class and they are to be presented to the user along with the moves involved in the solution.

8.1 POST IMPLEMENTATION AND SOFTWARE MAINTENANCE

It involves both operational and functional checks, reference and frequency checks so that the software failure does not lead to project failure. Arduino suits best in this scenario as it is a single slot software and hardware where the device is loaded with one particular code at a time and cannot be used for other purposes as the code does not help for another usages. One the code is entered into the Arduino there is no way of accessing or editing the code. the only way to change is to delete the code and equip it new code with new changes. So one should be careful before entering the code into the device as any wrong in code leads to time failure and even project chaos.

9. PROJECT LEGACY

9.1 CURRENT STATUS OF THE PROJECT

Currently the project is working by giving the inputs manually to the system. There is no interaction between the user and the machine after giving the inputs. The system is working in a better way as our thought that we wanted to build.

9.2 REMAINING AREAS OF CONCERN

Our system is working fine as expected. We are working on the OpenCV for image processing which helps in taking the inputs automatically using the camera and process to get the colors in each side. Also we are thinking to use Android phone to act as camera for capturing the faces.

9.3 TECHNICAL AND MANAGERIAL LESSONS LEARNT

In the process of developing this project, many technical aspects were needed to be focused on such as to make the project more users friendly so that the users won't face any difficulties in using it. During the making of this project, many managerial aspects were also taken considered such as the query tab in the users interface.

10. USER MANUAL

So basically the code we wrote uses combinations of pushes, holds and platform rotations to apply algorithms to the cube. There are 3 main parts of my code: the algortihm, GUI (enter cube state) and the serial communications. Let's start with the Arduino sketch. It basically uses algorithms to manipulate the cube colours:

1. There are 6 char arrays represent each face of the cube.
2. Using a basic layer by layer method of solving the Rubik's cube, we made a function, cube_decide(), that goes through each stage or the cube solving process.
3. Within each stage, such as cube_decide_cross() (solving the cross), the program would check for specific locations of colours in the cube, and if an algorithm was matched with the condition, cube notation like 'U' (Up) or entire algorithms, like fix_cross_instance_1() would be run.
4. The cube notation and algorithms control the servo functions, such as push_cube() or rotate_one().

Here's a basic overview of the code structure (layers of abstraction):

Cube Decide functions < Cube Algorithms < Cube Move functions < Servo functions < Move Function

One major breakthrough we had was that we actually simulated the cube's movements in the program. The program reassigns the values in the arrays to simulate and cube rotation. This way, the program can simulate the cube move before it physically executes it.

1. After the cube colors were entered in the GUI in Send_Cube_State.py, We converted the data into a string
2. Used a 'handshake', which in this case the Arduino tells the computer it's ready, then the computer transfers the data to the robot.
3. The Arduino converts the string received into chars, which are assigned to arrays in the sketch.
4. Now that the cube colours have been successfully transferred from the computer to the Arduino, the cube colours can be put through the algorithms and solve the Rubiks cube.

11. SOURCE CODE AND SYSTEM SNAPSHOTS

Arduino Code:

```
void setup()
{
  rotate_servo.attach(9);
  push_servo.attach(6);
  push_servo.write(push_pos);
  rotate_servo.write(rotate_pos);
  delay(1000);
  Serial.begin(9600);
  while (! Serial);
}
void loop()
{
  import_cube_colors();
  solve_cube();
  Serial.println("Done!");
  show_off_cube();
  while(true){}
};
void auto_test()
{
  Serial.println("Auto (Algorithm) Test:");
  int num_of_tests_ran = 1;
```

```
while(test_ongoing == true)
{
  Serial.println();
  Serial.println();
  Serial.println();
  Serial.print("Test number: ");
  Serial.print(num_of_tests_ran);
  scramble();
  delay(10000);
  print_whole_cube();
  solve_stage = 1;
  while(solve_stage != 10)
  {
    cube_decide();
  }
  if (cube_solved == false)
  {
    test_ongoing = false;
  }
  else
  {
    num_of_tests_ran++;
  }
}
Serial.println();
```

```
  Serial.println();

  Serial.println();

  Serial.print("Error found, test ended at Test number:");

  Serial.print(num_of_tests_ran);

}

void solve_cube()

{

  Serial.println("Solving Cube: ");

  print_whole_cube();

  solve_stage = 1;

  while(solve_stage != 10)

  {

    cube_decide();

  }

}

void show_off_cube()

{

  rotate_one();

  rotate_three();

  push_cube(2);

  rotate_one();

}
```

Python Code:

```
from Tkinter import *
import serial
import time
import sys

ser = serial.Serial("COM5", 9600)

yellow_face = ['y', 'y', 'y',
               'y', 'y', 'y',
               'y', 'y', 'y']

white_face = ['w', 'w', 'w',
              'w', 'w', 'w',
              'w', 'w', 'w']

blue_face = ['b', 'b', 'b',
             'b', 'b', 'b',
             'b', 'b', 'b']

red_face = ['r', 'r', 'r',
            'r', 'r', 'r',
            'r', 'r', 'r']

green_face = ['g', 'g', 'g',
              'g', 'g', 'g',
              'g', 'g', 'g']

orange_face = ['o', 'o', 'o',
               'o', 'o', 'o',
               'o', 'o', 'o']

def reset_arduino():
    ser.setDTR(False)
    time.sleep(0.022)
    ser.setDTR(True)


raw_cube_string = ""

def generate_raw_cube():
        raw_cube_string = ""
        for color in yellow_face:
                raw_cube_string = raw_cube_string + color

        for color in white_face:
                raw_cube_string = raw_cube_string + color

        for color in red_face:
                raw_cube_string = raw_cube_string + color

        for color in blue_face:
                raw_cube_string = raw_cube_string + color
        for color in green_face:
                raw_cube_string = raw_cube_string + color

        for color in orange_face:
                raw_cube_string = raw_cube_string + color

        return raw_cube_string


def send_raw_cube():
        ready_signal = ser.readline()


        ardu_ready = False

        while ardu_ready == False:
                arduino_status = ser.readline()
                time.sleep(0.01)
                if arduino_status == ready_signal:
                        ardu_ready = True
                        time.sleep(0.01)


        ser.write(raw_cube_string)
        print ("color sent: ",raw_cube_string)

        time.sleep(0.01)

def initialize_connection():
    reset_arduino()

    while ser.readline() == "":

        print ("Waiting for Arduino...")


    raw_cube_string = generate_raw_cube()
    ready_signal = ser.readline()


    ardu_ready = False

    while ardu_ready == False:
        arduino_status = ser.readline()
        time.sleep(0.01)
        if arduino_status == ready_signal:
            ardu_ready = True
            time.sleep(0.01)
```

```
def enter_yellow_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]
    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):
        r=x/3
        c=x%3
        if b[r][c] == 0:
            b[r][c]= 5
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 0
            yellow_face[x] = 'w'
        elif counter[x] == 0:
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 1
            yellow_face[x] = 'b'
        elif counter[x] == 1:
            bb[x].ss.configure(bg='red', activebackground='red')
            counter[x] = 2
            yellow_face[x] = 'r'
        elif counter[x] == 2:
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 3
            yellow_face[x] = 'g'

        elif counter[x] == 3:
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 4
            yellow_face[x] = 'o'

        elif counter[x] == 4:
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 5
            yellow_face[x] = 'y'

        elif counter[x] == 5:
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 0
            yellow_face[x] = 'w'

    root = Tk()
    root.title('Enter Yellow Face')

    class Knop():
        """This is the docstring of the class"""

        def __init__(self, i, master=None):
            self.nummer = i
            self.row = i/3
            self.col = i%3
            def human_move():
                change_color(self.nummer)
            self.ss = Button(root, command=human_move, bg='yellow', activebackground='yellow', width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)

            next_face = Button(root, text="Next Face",  command=root.destroy)
            next_face.grid(row=4, column=1)

    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)

    mainloop()

def enter_white_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]
    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):

        r=x/3
        c=x%3
        if b[r][c] == 0:
            b[r][c]= 5
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 0
            white_face[x] = 'b'

        elif counter[x] == 0:
            bb[x].ss.configure(bg='red', activebackground='red')
            counter[x] = 1
            white_face[x] = 'r'

        elif counter[x] == 1:
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 2
            white_face[x] = 'g'


        elif counter[x] == 2:
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 3
            white_face[x] = 'o'
        elif counter[x] == 3:
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 4
            white_face[x] = 'y'

        elif counter[x] == 4:
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 5
            white_face[x] = 'w'

        elif counter[x] == 5:
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 0
            white_face[x] = 'b'

    root = Tk()
    root.title('Enter White Face')

    class Knop():
        """This is the docstring of the class"""

        def __init__(self, i, master=None):
            self.nummer = i
            self.row = i/3
            self.col = i%3
            def human_move():
                change_color(self.nummer)
            self.ss    =    Button(root,    command=human_move,    bg='white',
activebackground='white', width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)

            next_face = Button(root, text="Next Face",  command=root.destroy)
            next_face.grid(row=4, column=1)

    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)

    mainloop()

def enter_blue_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]

    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):
```

```
    r=x/3
    c=x%3
    if b[r][c] == 0:
        b[r][c]= 5
        bb[x].ss.configure(bg='red', activebackground='red')
        counter[x] = 0
        blue_face[x] = 'r'

    elif counter[x] == 0:
        bb[x].ss.configure(bg='green', activebackground='green')
        counter[x] = 1
        blue_face[x] = 'g'

    elif counter[x] == 1:
        bb[x].ss.configure(bg='orange', activebackground='orange')
        counter[x] = 2
        blue_face[x] = 'o'


    elif counter[x] == 2:
        bb[x].ss.configure(bg='yellow', activebackground='yellow')
        counter[x] = 3
        blue_face[x] = 'y'

    elif counter[x] == 3:
        bb[x].ss.configure(bg='white', activebackground='white')
        counter[x] = 4
        blue_face[x] = 'w'

    elif counter[x] == 4:
        bb[x].ss.configure(bg='blue', activebackground='blue')
        counter[x] = 5
        blue_face[x] = 'b'

    elif counter[x] == 5:
        bb[x].ss.configure(bg='red', activebackground='red')
        counter[x] = 0
        blue_face[x] = 'r'

root = Tk()
root.title('Enter Blue Face')

class Knop():
    """This is the docstring of the class"""

    def __init__(self, i, master=None):
        self.nummer = i
        self.row = i/3
        self.col = i%3
```

```
            def human_move():
                change_color(self.nummer)
            self.ss = Button(root, command=human_move, bg='blue',
activebackground='blue', width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)

            next_face = Button(root, text="Next Face",  command=root.destroy)
            next_face.grid(row=4, column=1)

    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)

    mainloop()

def enter_red_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]

    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):

        r=x/3
        c=x%3
        if b[r][c] == 0:
            b[r][c]= 5
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 0
            red_face[x] = 'g'

        elif counter[x] == 0:
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 1
            red_face[x] = 'o'

        elif counter[x] == 1:
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 2
            red_face[x] = 'y'


        elif counter[x] == 2:
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 3
            red_face[x] = 'w'

        elif counter[x] == 3:
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 4
            red_face[x] = 'b'

        elif counter[x] == 4:
            bb[x].ss.configure(bg='red', activebackground='red')
            counter[x] = 5
            red_face[x] = 'r'

        elif counter[x] == 5:
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 0
            red_face[x] = 'g'

    root = Tk()
    root.title('Enter Red Face')

    class Knop():
        """This is the docstring of the class"""

        def __init__(self, i, master=None):
            self.nummer = i
            self.row = i/3
            self.col = i%3
            def human_move():
                change_color(self.nummer)
            self.ss = Button(root, command=human_move, bg='red', activebackground='red',
width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)

            next_face = Button(root, text="Next Face",  command=root.destroy)
            next_face.grid(row=4, column=1)

    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)

    mainloop()

def enter_green_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]

    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):

        r=x/3
        c=x%3
        if b[r][c] == 0:
            b[r][c]= 5
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 0
            green_face[x] = 'o'

        elif counter[x] == 0:
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 1
            green_face[x] = 'y'

        elif counter[x] == 1:
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 2
            green_face[x] = 'w'


        elif counter[x] == 2:
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 3
            green_face[x] = 'b'

        elif counter[x] == 3:
            bb[x].ss.configure(bg='red', activebackground='red')
            counter[x] = 4
            green_face[x] = 'r'

        elif counter[x] == 4:
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 5
            green_face[x] = 'g'

        elif counter[x] == 5:
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 0
            green_face[x] = 'o'
    root = Tk()
    root.title('Enter Green Face')
    class Knop():
        """This is the docstring of the class"""
        def __init__(self, i, master=None):
            self.nummer = i
            self.row = i/3
            self.col = i%3
            def human_move():
                change_color(self.nummer)
            self.ss = Button(root, command=human_move, bg='green',
activebackground='green', width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)
            next_face = Button(root, text="Next Face",  command=root.destroy)
            next_face.grid(row=4, column=1)
    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)
    mainloop()
def enter_orange_face():
    top,mid,btm=[0,0,0],[0,0,0],[0,0,0]
    bord = [top,mid,btm]

    counter = [0, 0, 0,
               0, 0, 0,
               0, 0, 0]

    def change_color(x, b=bord):

        r=x/3
        c=x%3
        if b[r][c] == 0:
            b[r][c]= 5
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 0
            orange_face[x] = 'y'
        elif counter[x] == 0:
            bb[x].ss.configure(bg='white', activebackground='white')
            counter[x] = 1
            orange_face[x] = 'w'

        elif counter[x] == 1:
            bb[x].ss.configure(bg='blue', activebackground='blue')
            counter[x] = 2
            orange_face[x] = 'b'
            elif counter[x] == 2:
            bb[x].ss.configure(bg='red', activebackground='red')
            counter[x] = 3
            orange_face[x] = 'r'
        elif counter[x] == 3:
            bb[x].ss.configure(bg='green', activebackground='green')
            counter[x] = 4
            orange_face[x] = 'g'
        elif counter[x] == 4:
            bb[x].ss.configure(bg='orange', activebackground='orange')
            counter[x] = 5
            orange_face[x] = 'o'
        elif counter[x] == 5:
            bb[x].ss.configure(bg='yellow', activebackground='yellow')
            counter[x] = 0
            orange_face[x] = 'y'
    root = Tk()
    root.title('Enter Orange Face')
    class Knop():
        """This is the docstring of the class"""
        def __init__(self, i, master=None):
            self.nummer = i
            self.row = i/3
            self.col = i%3
            def human_move():
                change_color(self.nummer)
            self.ss = Button(root, command=human_move, bg='orange',
activebackground='orange', width=10, height=5)
            self.ss.grid(row=self.row, column=self.col)
            next_face = Button(root, text="Send Cube!",  command=root.destroy)
            next_face.grid(row=4, column=1)
    bb = range(9)
    for i in range(9):
        bb[i]= Knop(i, master=root)
    mainloop()
def print_face(current_face):
    print (current_face[0], current_face[1], current_face[2])
    print (current_face[3], current_face[4], current_face[5])
    print (current_face[6], current_face[7], current_face[8])
def print_cube():
    print ("Yellow Face: ")
    print_face(yellow_face)
    print ("White Face: ")
    print_face(white_face)
    print ("Blue Face: ")
    print_face(blue_face)
    print ("Red Face: ")
    print_face(red_face)
    print ("Green Face: ")
    print_face(green_face)
    print ("Orange Face: ")
    print_face(orange_face)
def legal_cube_check():
    if yellow_face[4] != 'y' or white_face[4] != 'w' or blue_face[4] != 'b' or red_face[4] !=
'r' or green_face[4] != 'g' or orange_face[4] != 'o':
        print ("Incorrect center pieces, Try Again")
        sys.exit()
def enter_cube():
    enter_yellow_face()
    enter_white_face()
    enter_blue_face()
    enter_red_face()
    enter_green_face()
    enter_orange_face()
initialize_connection()
enter_cube()
legal_cube_check()
print_cube()
```

```
reset_arduino()
while ser.readline() == "":
    print ("Waiting for Arduino...")
print ("sending cube string...")
raw_cube_string = generate_raw_cube()
send_raw_cube()
while True:
    time.sleep(0.01)
    print ser.readline()
```

SNAPSHOTS:

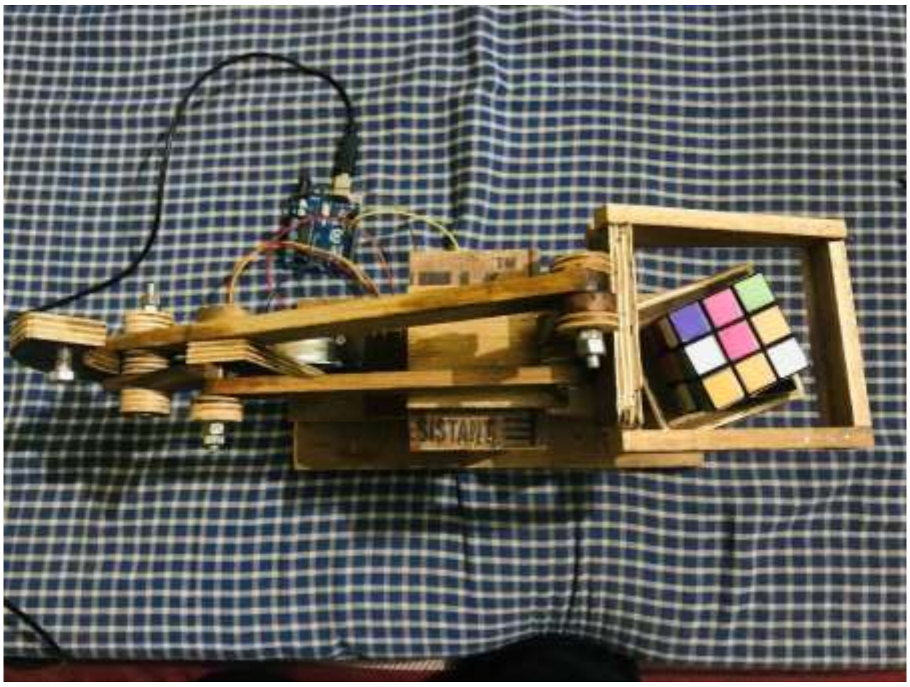


Fig 9: Snapshot of the Hardware

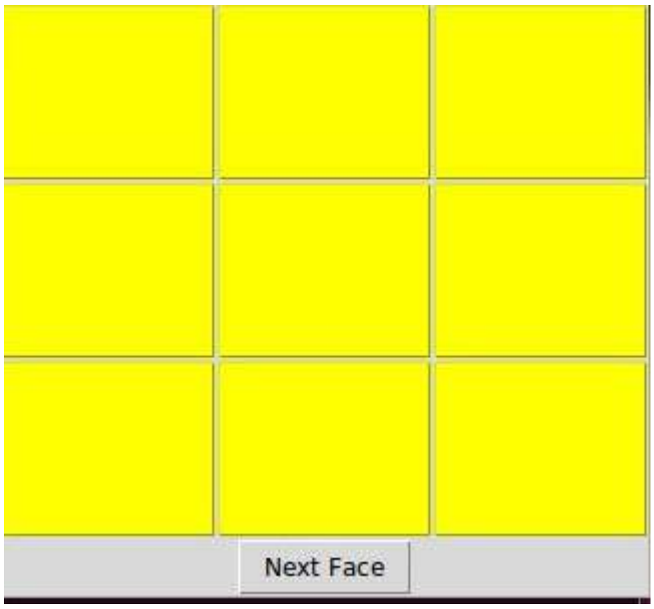


Fig 10: GUI – Faces Input

12. REFERENCES

1) https://www.youtube.com/watch?v=wmXretly7pg
2) https://www.grubiks.com/solvers/rubiks-cube-3x3x3/
3) https://www.youcandothecube.com/solve-it/3-x-3-solution
4) https://www.speedcubereview.com/blind-solving-algorithms.html
5) https://dzone.com/articles/gui-tic-tac-toe-less-100-lines
6) https://pythonhosted.org/pyserial/
7) http://studentnet.cs.manchester.ac.uk/resources/library/3rd-year-projects/2015/aryeh.grosskopf.pdf
8) http://www.eecs.ucf.edu/seniordesign/sp2016su2016/g12/pdfUploads/Project_Documet_Group_12.pdf
9) http://www.diva-portal.org/smash/get/diva2:953324/FULLTEXT01.pdf
10) http://www.instructables.com/id/Rubiks-Cube-Solver/
11) https://www.cs.swarthmore.edu/~knerr/helps/rcube.html
12) https://ruwix.com/the-rubiks-cube/gods-number/