Tribhuvan University

Academia International College

Final Year Project Report

On

Cube Master

[CSC 412]

Under the supervision of

"Mr. Bishwas Mathema"

Submitted by

Mansij Maharjan (T.U. Exam Roll No. 29014/078)

Rinka Maharjan (T.U. Exam Roll No. 29023/078)

Sesema Limbu (T.U. Exam Roll No. 29028/078)

Submitted to

Department of Computer Science and Information Technology

Academia International College

Institute of Science and Technology

Tribhuvan University

September, 2025

Tribhuvan University

Academia International College

Final Year Project Report

On

Cube Master

[CSC 412]

A final year project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Submitted by

Mansij Maharjan (T.U. Exam Roll No. 29014/078)

Rinka Maharjan (T.U. Exam Roll No. 29023/078)

Sesema Limbu(T.U. Exam Roll No. 29028/078)

Submitted to

Department of Computer Science and Information Technology

Academia International College

Institute of Science and Technology

Tribhuvan University

September, 2025

Tribhuvan University

Institute of Science and Technology

Academia International College

Department of Computer Science and Information Technology

Email: mail@academiacollege.edu.np

Supervisor's Recommendation

I hereby recommend that the project work report prepared under my supervision by Mr. Mansij Maharjan (29014), Ms. Rinka Maharjan (29023) and Ms. Sesema Limbu (29015) entitled "Cube Master" be accepted as fulfilling in partial requirements for the degree of Bachelor of Science in Computer Science and Information Technology. In my best knowledge, this is an original work in Computer Science and Information Technology.

.....

Mr. Bishwas Mathema
Project Supervisor
HOD/Program Coordinator
Department of Computer Science and Information Technology
Academia International College
Gwarko, Lalitpur

Tribhuvan University

Department of Computer Science and Information Technology

Academia International College

Certificate of Approval

This is to certify that this project prepared by Mr. Mansij Maharjan, Ms. Rinka Maharjan and Ms. Sesema Limbu entitled "Cube Master" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

……...…….. Mr. Bishwas Mathema Project Supervisor Department of Computer Science and IT Academia International College	…….. Mr. Bishwas Mathema HOD/Program Coordinator Department of Computer Science and IT Academia International College
…….. Internal Examiner	…….. External Examiner

Acknowledgement

We sincerely thank Academia International College for providing us with this fantastic chance to engage in this project as a component of our academic program. This experience has been incredibly beneficial in utilizing our knowledge in a real-world context. We want to sincerely thank our respected supervisor, Mr. Bishwas Mathema (HOD/Program Coordinator, Academia International College), for his constant support, helpful advice, and useful feedback during the entire process of creating this report. His guidance has played a key role in improving our knowledge and abilities. We express our gratitude to all the individuals, families, friends, colleagues, and faculty who supported us throughout this journey. Their considerate recommendations, collaboration, and encouragement were essential in enabling us to finish this project successfully within the specified period.

Thanking You,

Mansij Maharjan (T.U.Exam Roll No. 29014)

Rinka Maharjan (T.U.Exam Roll No. 29023)

Sesema Limbu (T.U.Exam Roll No. 29028)

Abstract

Cube Master is a contemporary, online platform aimed at streamlining and improving the Rubik's Cube solving process by incorporating computer vision, smart solving techniques, and engaging visualization. The system functions in two primary phases to ensure efficiency and precision. At first, users have the option to either enter the cube's setup by hand or capture it through a webcam. The collected data is analyzed using advanced computer vision methods that employ Python, OpenCV, and K-means clustering to accurately identify and map the colors of each cube face. This approach guarantees high precision even in different lighting environments by concentrating on prominent color groups. Upon successfully mapping the cube, Cube Master offers users two solving choices: the user-friendly Layer-by-Layer technique suitable for novices, and the Kociemba algorithm, which produces the shortest and most efficient solution pathway. To enhance engagement and comprehension, the platform offers the solution as both a detailed text guide and an interactive 3D cube animation. Moreover, a virtual cube interface is provided, allowing users to engage in interactive practice for solving online. Through the integration of artificial intelligence and computer vision, Cube Master not only makes a typically challenging puzzle easier but also acts as an educational resource, demonstrating the capabilities of technology in addressing real-world issues. Designed for enthusiasts, educators, and learners and computer vision in enhancing everyday problem-solving experiences.

KEYWORDS: ***Rubik's Cube, Computer Vision, Layer-by-Layer Approach, Python, K-means Clustering, OpenCV, NumPy, Optimizing Algorithm, Koceimba Algorithm, Virtual Cube***

Table of Content

List of Figures

List of Tables

List of Abbreviations

3D	Three-Dimensional
AR	Augmented Reality
CFOP	Cross, F2L, OLL, PLL (Rubik's Cube Solving Method)
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CV	Computer Vision
F2L	First Two Layers
GB	Gigabyte
HSV	Hue, Saturation, Value
HTTPS	Hypertext Transfer Protocol Secure
LBL	Layer By Layer
OLL	Orientation of the Last Layer
PLL	Permutation of the Last Layer
RAM	Random Access Memory
RGB	Red, Green, Blue
ROI	Region of Interest
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
VScode	Visual Studio Code

Chapter 1: Introduction

1.1 Introduction

The Cube Master, a timeless puzzle recognized for its intricacy and intellectual challenge, has captivated problem solvers, learners, and teachers for many years. Although manually solving the cube involves memorizing algorithms and grasping spatial patterns, many novices struggle to understand the procedure. Cube Master is a cutting-edge online platform that aims to enhance and modernize the cube-solving experience through the use of computer vision, machine learning, and advanced solving algorithms. It enables users to enter the cube's state either manually or via a webcam, utilizing Python, OpenCV, and K-means clustering to effectively identify and categorize the cube's face colors—even in changing lighting conditions.

After scanning the cube, Cube Master provides two solving methods: a layer-by-layer approach suited for novices wanting to grasp the cube's logic gradually, and the Kociemba algorithm, which offers the most efficient solution with minimal moves. The platform boosts learning and usability by offering a text-based move list, a 3D visual showcase, and an interactive virtual cube for practical experience. By employing this dual-solving method and user-friendly interface, Cube Master demonstrates the real-world capabilities of artificial intelligence and computer vision in transforming intricate problem-solving into an accessible, educational, and engaging experience for a wider audience.

1.2 Problem Statement

Although the Rubik’s Cube is widely known for enhancing problem-solving and cognitive skills, existing digital solvers and learning tools have several limitations. Many require users to manually enter the color of each face, which is slow and prone to mistakes, especially for beginners. Others provide only text-based solutions without helpful visual guides, making it harder to follow the steps. Real-time camera detection is rare, and when available, it often struggles with accuracy under different lighting or device conditions. Additionally, some advanced solvers are too technical, or work only on specific platforms, making them less accessible to the general public and students.

1.3 Objective

The main objective of cube master is:

- To design and develop a web-based Rubik's Cube solver that offers users the choice between the Layer-by-Layer beginner method.
- To provide users with step-by-step visual guidance and animations to make the solving process easier to follow, especially for beginners.
- To create an interactive, educational, and user-friendly platform that makes solving the Rubik's Cube accessible and fun for users of all skill levels.
- To develop a system that can accurately detect colors from the faces of Rubik's cube using Computer Vision with K-means Clustering.

1.4 Scopes and Limitation

1.4.1 Scopes

The scopes of cube master include: -

- The platform integrates Open-CV for accurately detecting and scanning the colors of all six faces of a physical 3x3 Rubik's cube
- Users are guided step-by-step during the scanning process to ensure proper data capture for generating the solution.
- It allows user to apply the layer-by-layer solving algorithm, which is widely recognized as the simplest method for learning to solve a Rubik's cube.
- It also allows user to apply the kociemba solving algorithm, which is widely recognized as the most optimal method to solve a Rubik's cube.
- Solutions are presented through interactive, visual instructions to enhance user understanding and engagement.

1.4.2 Limitation

The limitation of cube master include: -

- Users must use a standard 3x3 Rubik's Cube with standard color schemes for accurate detection and solving.
- Scanning accuracy may vary due to lighting conditions, camera quality, or non-standard or faded cube colors.

- The system is unable to process or resolve cubes that have been altered, such as cubes with swapped pieces, missing stickers, or twisted parts, as these are unsolvable under normal rules.
- The platform depends on the user to accurately place and orient the cube while scanning; improper positioning could result inaccurate solutions.
- The solution focuses only on physical cubes and does not support virtual cube solving or custom cube configurations beyond the standard 3x3.

1.5 Methodology

The development of Cube Master was carried out through an iterative process to ensure ongoing improvement and enhancement. This methodology breaks down the complex project into manageable increments, allowing for focused development, testing, and integration at each step.

Iteration 1:

The project commenced with thorough exploration of relevant literature and methods for addressing the Rubik's Cube. Following an assessment of various methods, the Layer-by-Layer (LBL) technique was selected because of its ease of use and organized procedures, which facilitate implementation and adherence. This stage concentrated on developing the fundamental solving logic using the LBL method. The algorithm was first evaluated with simulated cube data to confirm its accuracy. Additional studies suggested that the integration of Computer Vision (CV) with K-Means Clustering would be an effective approach for analyzing the cube's colors.[2]

Iteration 2:

The second phase concentrated on completely executing the Layer-by-Layer algorithm, starting from the bottom layer and advancing through the middle and top layers. Every phase was broken down into specific sub-steps to guarantee accurate performance of rotations and movements. At the same time, the CV system was upgraded to enhance the detection and classification of cube colors. The clustering algorithm was refined to more effectively differentiate between comparable shades, enhancing scanning precision. The solution for each layer was tested separately to avoid errors impacting the following layers.

Iteration 3:

Once the core algorithm was finalized and operational, attention turned to developing an intuitive web interface. The goal was to develop an interactive and visually captivating platform for users to engage with the solving process. A Three.js-based 3D visualization of the Rubik's Cube was created, allowing for real-time representation of the cube's configuration as the solution progresses. The interface was created to refresh in real-time with every step of the algorithm. Furthermore, the Kociemba algorithm was examined and incorporated to offer an ideal solution route for more complex solving situations.[3]

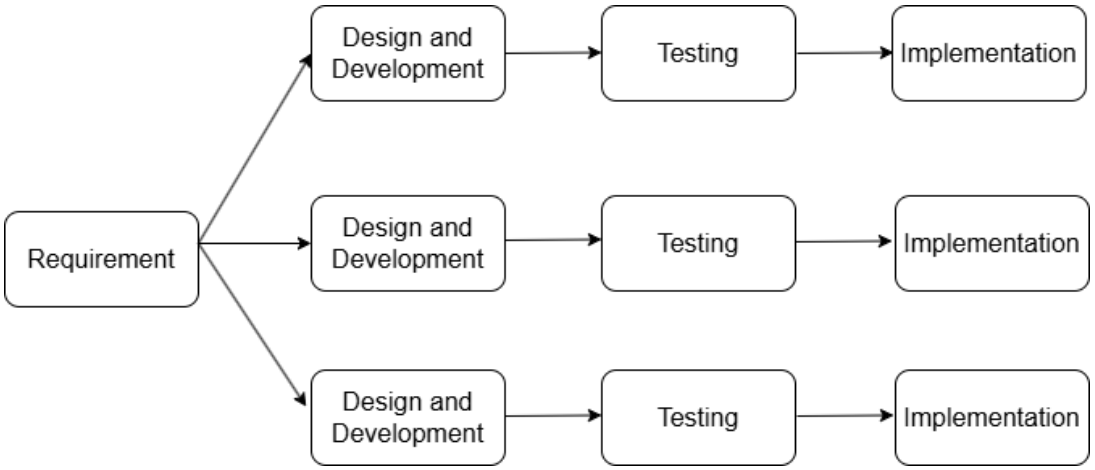

Figure 1.1: Incremental Method of Development

1.6. Report Organization

This project, "Cube master: The Rubik's Cube Solver", is organized into six chapters as follows:

Chapter 1 Introduction:

The project is introduced by highlighting its objectives, scope, and limitations. It also outlines the development methodology followed in creating the system.

Chapter 2:

Provides a background study, including fundamental theories and a literature review of similar projects and research, establishing the context and relevance of the project.

Chapter 3:

Focuses on analysis, detailing system analysis, requirement analysis, and feasibility analysis to define project requirements and assess viability.

Chapter 4:

Presents the overall system design, including architectural diagrams and design patterns that outline the structure and interaction of system components.

Chapter 5:

Covers implementation and testing, describing the tools used and the methodologies employed to ensure system reliability and functionality.

Chapter 6:

Concludes the report with a summary of findings, highlighting the project's outcomes and providing recommendations for future enhancements of the system.

Chapter 2: Background Study and Literature Review

2.1 Background Study

The Rubik's Cube, created by Ernő Rubik in 1974, is a three-dimensional combination puzzle featuring six faces, each made up of nine smaller squares that can be rotated independently. The objective of the puzzle is to restore it to its initial configuration, where each face shows a uniform color. Despite its seemingly straightforward design, the Rubik's Cube represents a complex mathematical entity that has captivated mathematicians, computer scientists, and puzzle enthusiasts for many years.

Solving a Rubik’s Cube is commonly perceived as highly difficult without a systematic solution approach or algorithm. The closer the cube is to being solved, with more cubies accurately positioned, the greater the chance that an unintended move will disturb already solved areas. To prevent this, meticulously crafted sequences of rotations are essential to ensure that only designated cubies are shifted to their intended locations. Over the years, a variety of solution techniques have emerged, differing in complexity and efficiency based on the identification and utilization of specific patterns and conditions. Typically, quicker algorithms that necessitate fewer rotations require the mastery of a larger number of sub-algorithms and their application in appropriate situations. The Layer by Layer (LBL) method, although simpler with fewer sub-algorithms to learn, is significantly less efficient in comparison.[4]

Manually solving a Rubik's Cube demands spatial reasoning, pattern recognition, and a comprehensive understanding of permutation groups. With advancements in modern technologies such as computer vision, artificial intelligence, and web development frameworks, automating this process has become quite achievable.

2.2 Literature Review

In the paper, it is discussed that the original Rubik's Cube has 8 corners and 12 edges. Corners can be arranged in 8! ways, and there are 37 possible orientations because the orientation of the eighth corner depends on the preceding ones. Edges can be arranged by 12!/2 ways, restricted from 12! because edges must be in an even permutation exactly when the corners are. And there are 211 possibilities because 11 edges can be flipped independently, with the flip of the 12th depending on the preceding ones. The number of Rubik's Cube configuration species is about

$8!\times3^7\times12!/2\times2^{11} = 43252003274489856000$

which is approximately 43 quintillion. To put this into perspective, if one had one standard-sized Rubik's Cube for each permutation, one could cover the Earth's surface 275 times or stack them in a tower 261 light-years high. The preceding figure is limited to permutations that can be reached solely by turning the sides of the cube. If one considers permutations reached through disassembly of the cube, the number becomes twelve times larger:

$8!\times3^8\times12!\times2^{12} = 519024039293878272000$

which is approximately 519 quintillion possible arrangements of the pieces that make up the Cube, but only one in twelve of these are actually solvable. This is because there is no sequence of moves that will swap a single pair of pieces or rotate a single corner or edge cube. Thus there are twelve possible sets of reachable configurations, sometimes called "universes" or "orbits", into which the Cube can be placed by dismantling and reassembling it.

Several algorithms have been developed to solve the Rubik's Cube, each varying in complexity, execution time, and efficiency. Among these, the Layer-by-Layer (LBL) approach is the most used method, especially for beginners. This method involves solving the cube systematically layer by layer, beginning with the first layer's cross, followed by positioning the corners, solving the middle-layer edges, and finally addressing the last layer. While the LBL method is intuitive and easy to grasp, it is less efficient in terms of the number of moves required compared to more advanced solving techniques.[5]

For speed-cubers aiming for efficiency, the Fridrich method, also known as CFOP (Cross, F2L, OLL, PLL), is widely favored. This method optimizes solving by combining multiple steps. After creating the cross on the first layer, the Fridrich method solves the first two layers (F2L) simultaneously. For the last layer, it uses two steps: Orientation of the Last Layer (OLL) and Permutation of the Last Layer (PLL), significantly reducing the number of moves required and improving solve times.[6]

The Kociemba algorithm is a two-phase algorithm designed to efficiently solve the Rubik's Cube, often used in virtual cube solvers. It finds an optimal solution with fewer moves by reducing the cube to a simpler state in the first phase and solving it completely in the second. It is closely linked to the concept of "God's Number," the maximum number of moves required to solve any cube configuration, which is 20. This algorithm is a cornerstone in computational Rubik's Cube solving.[7]

In the context of Rubik's Cube face detection, K-means clustering has been applied to improve the accuracy and efficiency of identifying individual cube faces. By clustering the RGB color data from images of the cube, K-means clustering helps to identify patterns that represent the distinct colored stickers on each face. This clustering method groups similar color pixels, making it easier to separate and detect each of the six cube faces. The use of K-means ensures better segmentation and identification of the colors, which is crucial for solving the cube through computer vision techniques. This approach improves the robustness of face detection even under varying lighting conditions, ultimately enhancing the automation of Rubik’s Cube solving algorithms.

Chapter 3: System Analysis

3.1 System Analysis

The Cube Master system is designed to assist users in solving a standard 3x3 Rubik's Cube through an interactive web platform. The system combines computer vision, cube-solving algorithms, and user-friendly interfaces to deliver accurate solutions efficiently.

3.1.1 Requirement Analysis

The requirements can be functional and non-functional.

i. Functional Requirements

The project considered the following functional requirements:

- Helps users take pictures of all six sides of a Rubik's Cube with the device's camera.
- It finds and matches the colors from the pictures to a digital version of the cube.
- It shows users clear pictures and instructions to follow along and solve the cube.
- Capture cube colors via webcam or manual input.
- Provide Layer-by-Layer and Kociemba solving options.
- Display solutions in text and 3D animation.
- Offer a virtual interactive cube.

ii. Non-Functional Requirements

The project considered the following non-functional requirements:

- Simple and user-friendly interface.
- Accurate color detection under normal conditions.
- Quick response for scanning and solving.
- Works with basic webcams and average internet.

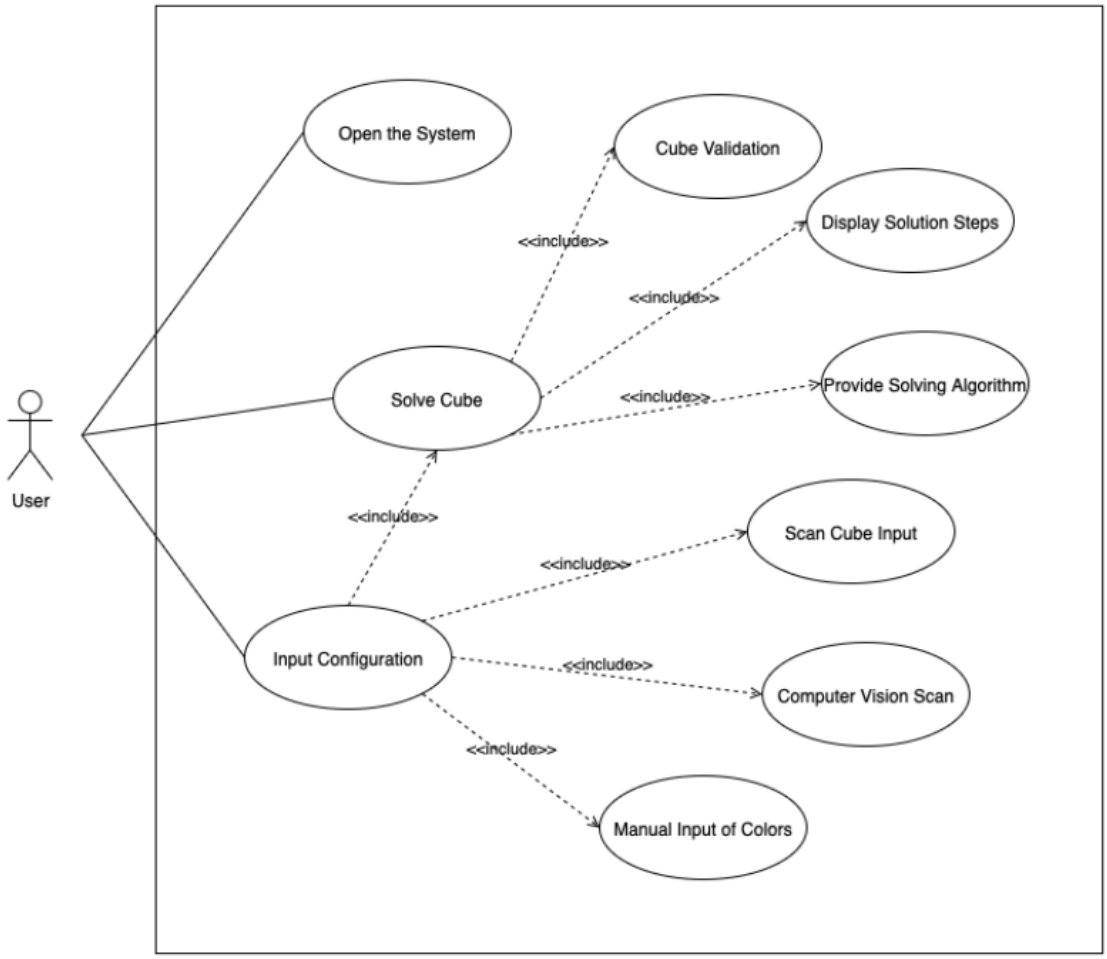

Figure 3.1: Use Case Diagram for Cube Master

Figure 3.1 shows the user interaction with the Cube Master by scanning the cube through the system's interface.

Table 3.1: Use Case Scenario of Cube Master

Use case: Solving a Scrambled Rubik's Cube using Cube Master	
Primary Actor:	Beginner User
Secondary Actor:	None

Flow of Events	1. The user wants to solve a scrambled Rubik's Cube using Cube Master by scanning the cube. 2. User navigates to Cube Master website and clicks on scan the cube button on system interface, activating the camera(). 3. The user is prompted to align their cube so that all six faces are clearly visible to the camera(). 4. Once all faces are captured, the app processes the image and automatically detects() the colors of each square on the cubes. 5. If there is any incorrect detection of colors in the cub, then the user will manually edit() the color for each square of the cube. 6. The system analyzes the color data either from the scan or manual input() and identifies the scrambled state of the cube. 7. The system processes the current cube configuration and the user chooses whether to the layer-by-layer method or kociemba algorithm to solve the cube. 8. The user is presented with a clear, easy-to-follow step-by-step guide on how to solve the cube, accompanied by visual representation.
Pre-condition	The system must be operational, and the user must have valid scrambled Rubik's Cube for scan.
Post-condition	The system must accurately scan all the faces of the cube and solve the Rubik's cube and the user must be presented with step-by-step instructions on how to solve the cube.
Quality	The solution must be accurate and the scrambled Rubik's Cube should be solved.

3.1.2. Feasibility Analysis

Feasibility analysis involves assessing a project or system concept to establish if it is feasible and workable technically, financially, and operationally prior to committing substantial time or resources

i. Technical Feasibility

The project is technically feasible using Python libraries such as OpenCV for image processing, NumPy for matrix manipulation, and scikit-learn for clustering. The Kociemba algorithm is available as a Python package. 3D visualizations can be handled using libraries like Three.js for web-based environments. No specialized hardware is required, making development accessible on standard systems.[8]

- Hardware Requirements: To effectively run the Cube Master system, a powerful CPU (Central Processing Unit) is required to manage the execution, process input data, and inference tasks. Adequate RAM is necessary to efficiently handle scanned cube, while sufficient storage capacity is required for quick data access and processing. A reliable internet connection is also important for data transfer and integration with cloud services.
- Software Requirements: The project primarily utilizes Python as the programming language due to its extensive libraries and frameworks for algorithm and image processing. Key libraries and frameworks include Django for backend, OpenCV for image processing tasks and NumPy for efficient numerical computations. Additionally, the project uses React for frontend and Three.js for displaying 3D Output. An Integrated Development Environment (IDE) or code editor, such as PyCharm, Visual Studio Code, or Jupyter Notebook, is recommended for writing and testing code. Implementing a version control system like Git is essential for tracking changes and facilitating collaboration. These software components are critical to ensuring the successful implementation and functionality of the brain tumor prediction system.[9]

ii. Operational Feasibility

Cube Master will run as a web-based application, accessible via browsers, making it user-friendly and widely available. Users can either upload images or use webcam input for cube scanning. The interface is designed for beginners, educators, and enthusiasts, ensuring easy operation without prior technical knowledge.

iii. Economic Feasibility

The project has a low development cost because it makes use of open-source libraries and technologies. During the development stage, hosting can be done on free or inexpensive platforms. All things considered, the project is financially feasible to develop and implement.

iv. Schedule Feasibility

Schedule feasibility is the process of developing a thorough project plan that outlines every stage of the project, including data collection, system design, development, testing, and deployment, utilizing tools such as Gantt charts. The Gantt chart shows the amount of work that is finished over time in comparison to the amount of time that is scheduled for future work.[10]

Table 3.2: Schedule Feasibility

Task	**Start Date**	**End Date**	**Duration**
Documentation	12/05/2025	7/01/2025	160
Iteration 1			
Planning	12/05/2025	13/05/2025	2
Design	14/05/2025	16/05/2025	3
Development	17/05/2025	02/06/2025	17
Testing	03/06/2025	07/06/2025	4
Implementation	08/06/2025	11/06/2025	4
Iteration 2			
Planning	12/06/2025	14/06/2025	3
Design	15/06/2025	22/06/2025	8
Development	23/06/2025	30/06/2025	25
Testing	01/07/2025	05/07/2025	5
Implementation	06/07/2025	08/07/2025	3
Iteration 3			
Planning	09/07/2025	11/07/2025	2
Design	12/07/2025	14/07/2025	2
Development	15/07/2025	25/07/2025	11

Testing	26/07/2025	28/07/2025	2
Implementation	29/07/2025	01/08/202	4

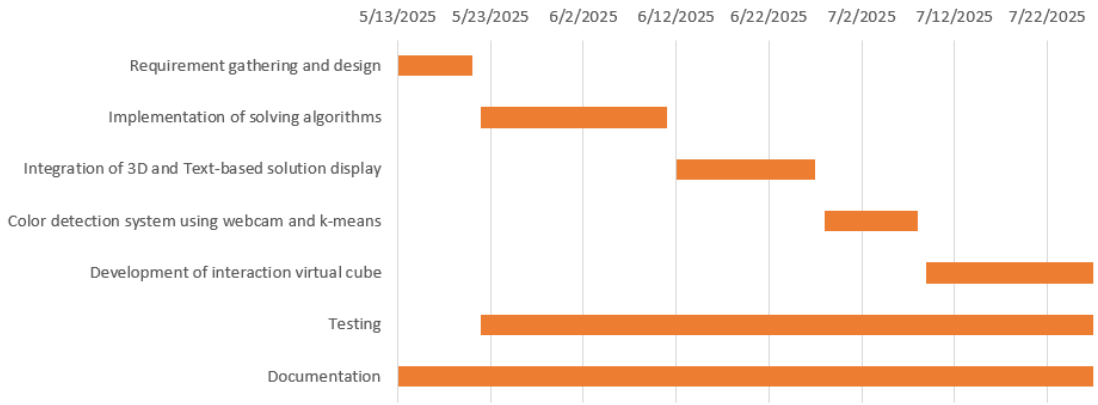

Figure 3.2: Gantt Chart

3.1.3. Object Modeling using Class and Object Diagrams

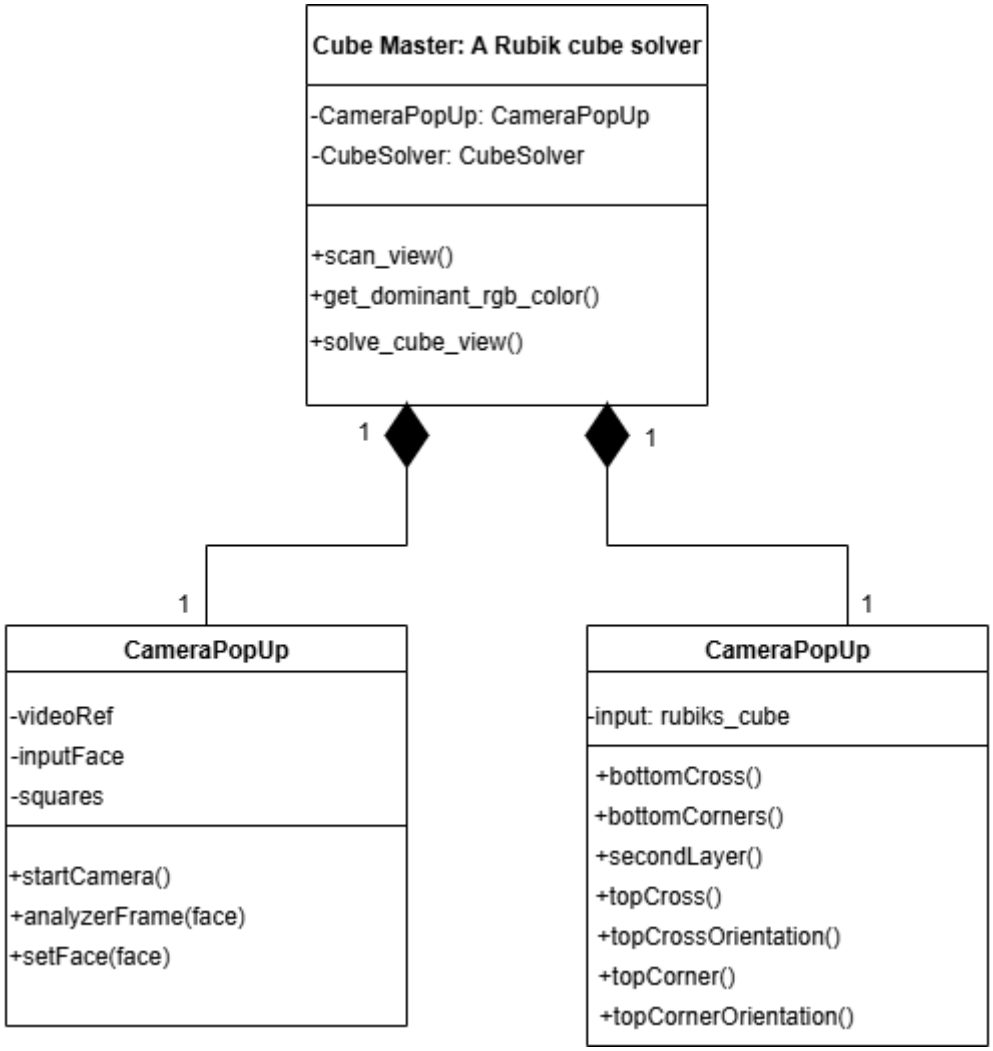

Figure 3.3: Object Diagram for Cube Master

Figure 3.3 The core classes, attributes, and methods of the Rubik's Cube Solver application illustrate its static structure. The core component is the Cube Master class, serving as the primary controller that oversees the entire operation flow, including cube scanning, solution processing, and visualization rendering. It works in conjunction with the CameraPopUp class, which handles the activation of the device camera, capturing images of the cube's faces, and identifying colors from specific square locations on each face. After scanning all six faces, the SolveCube class assumes control, employing a classic layer-by-layer solving method—like creating the bottom cross, completing the middle layer, and orienting the top face—or utilizing the Kociemba algorithm for a more optimized and effective solution

3.1.4. Dynamic modelling using state and sequence diagram

The sequence diagram for Cube Master is used to show the interactions between objects in the sequential order that those interactions occur. It provides a dynamic perspective on the system's behavior over time.

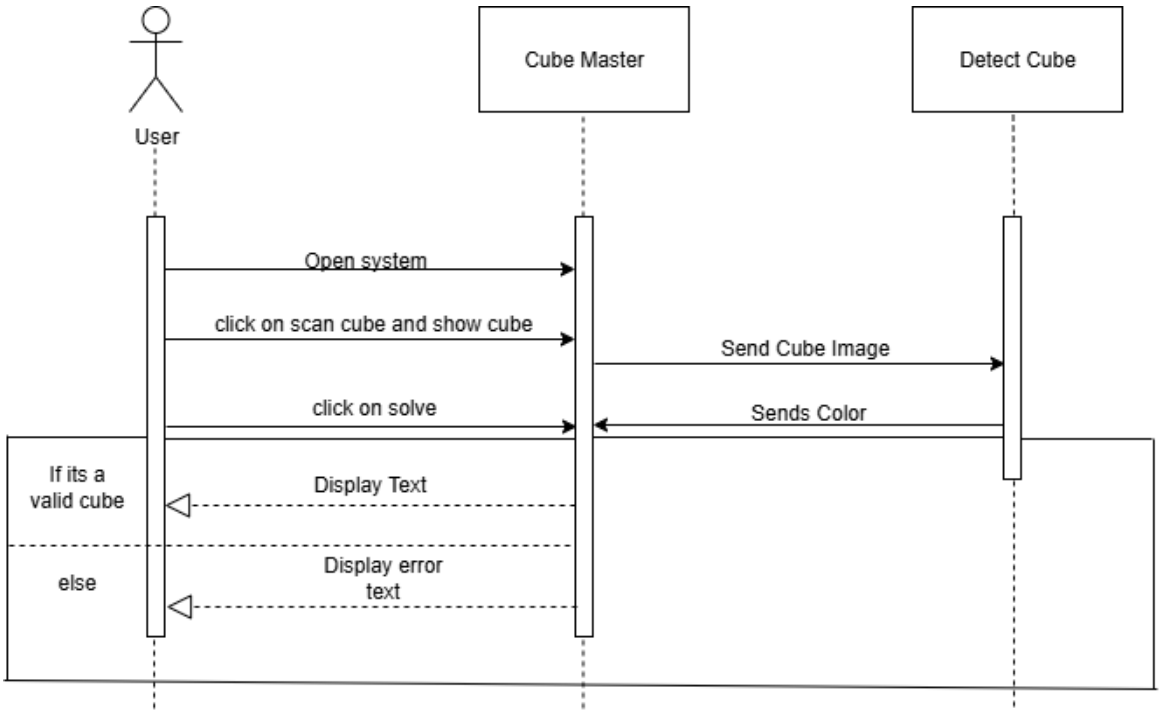

Figure 3.4: Sequence Diagram for Cube Master

Figure 3.4 illustrates the interaction between the user, the Cube Master system, and the Detect Cube component. The interaction starts when the user opens the app and chooses the “Scan Cube” option. In reaction, the system activates the camera of the device to take pictures of the cube’s six sides. The Detect Cube module receives these images, employing K-means clustering to assess the colors and link each area to the appropriate Rubik’s cube color code. After color detection is finished, the manipulated color information is sent back to the Cube Master system. When the cube arrangement is legitimate and can be solved, the system moves on to create a detailed solution, featuring a text-based step-by-step sequence along with a three-dimensional visual representation of the solving procedure, which is subsequently shown to the user. If the cube is in an invalid state or cannot be solved, the system provides an error message explaining the issue. This organized engagement guarantees a smooth user experience from cube scanning to results presentation whether it’s a successful outcome or distinct error notifications.

3.1.5 Process modelling using Activity Diagrams

An activity diagram is a type of UML diagram that graphically depicts a system's control flow or a series of events. It shows detailed work procedures that support concurrency, choice, and iteration.

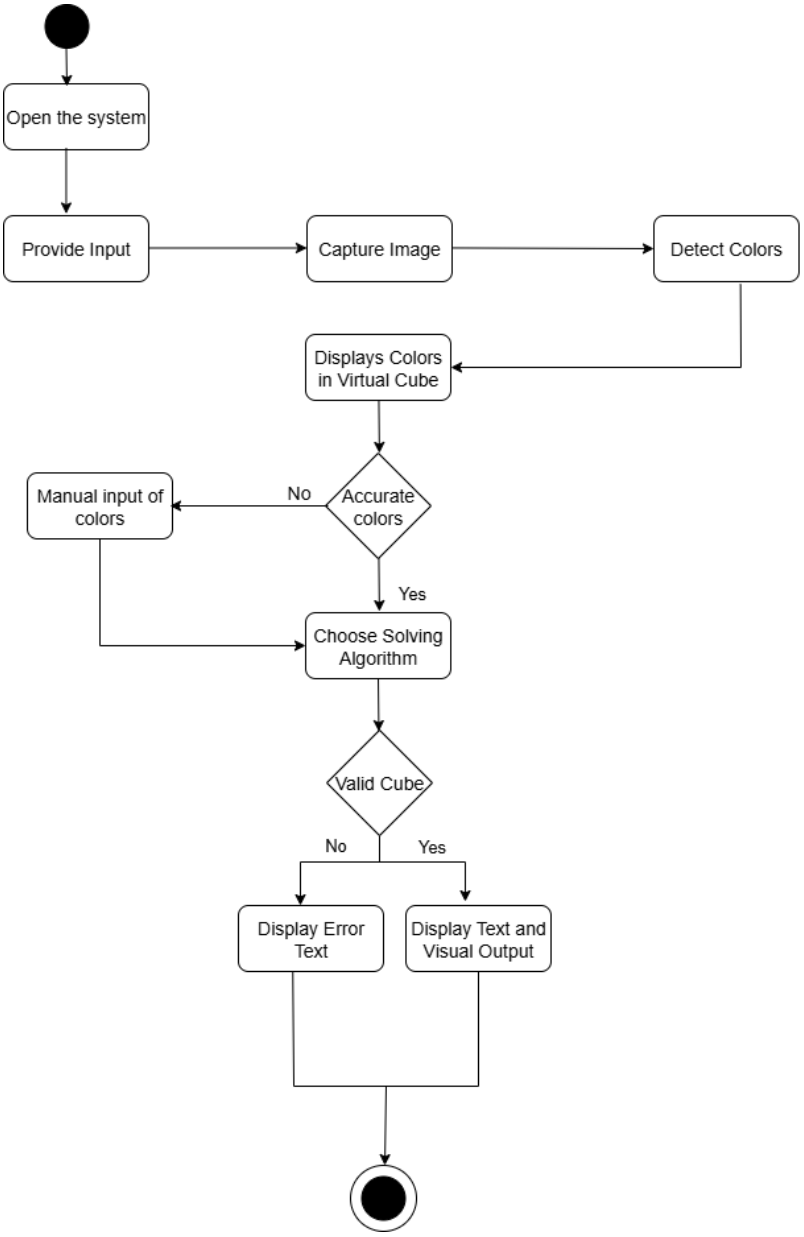

Figure 3.5: Activity Diagram for Cube Master

Figure 3.5 show that the Cube Master process begins when the user opens the app and turns on the camera. They then hold the Rubik's Cube so that all six sides are visible to the camera. The system takes pictures of each face. These pictures go to a color recognition part of the app, which checks and finds the colors on every side of the cube.

Once the colors are identified, they are sent to a solving part of the app. There, a digital copy of the cube is made based on the colors found. If the cube's colors are correct, the app uses either the Layer-by-Layer method or the Kociemba algorithm to figure out how to solve it. The answer is shown to the user as a list of moves and also as a 3D picture of the solved cube. If the cube's colors don't match or are in the wrong position, the app shows an error message explaining the problem, like mismatched colors or the cube not being set up properly.

Chapter 4: System Design

4.1 Design

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It focuses on planning the structure and behavior of a system, ensuring that all parts work together efficiently to achieve the intended objectives.

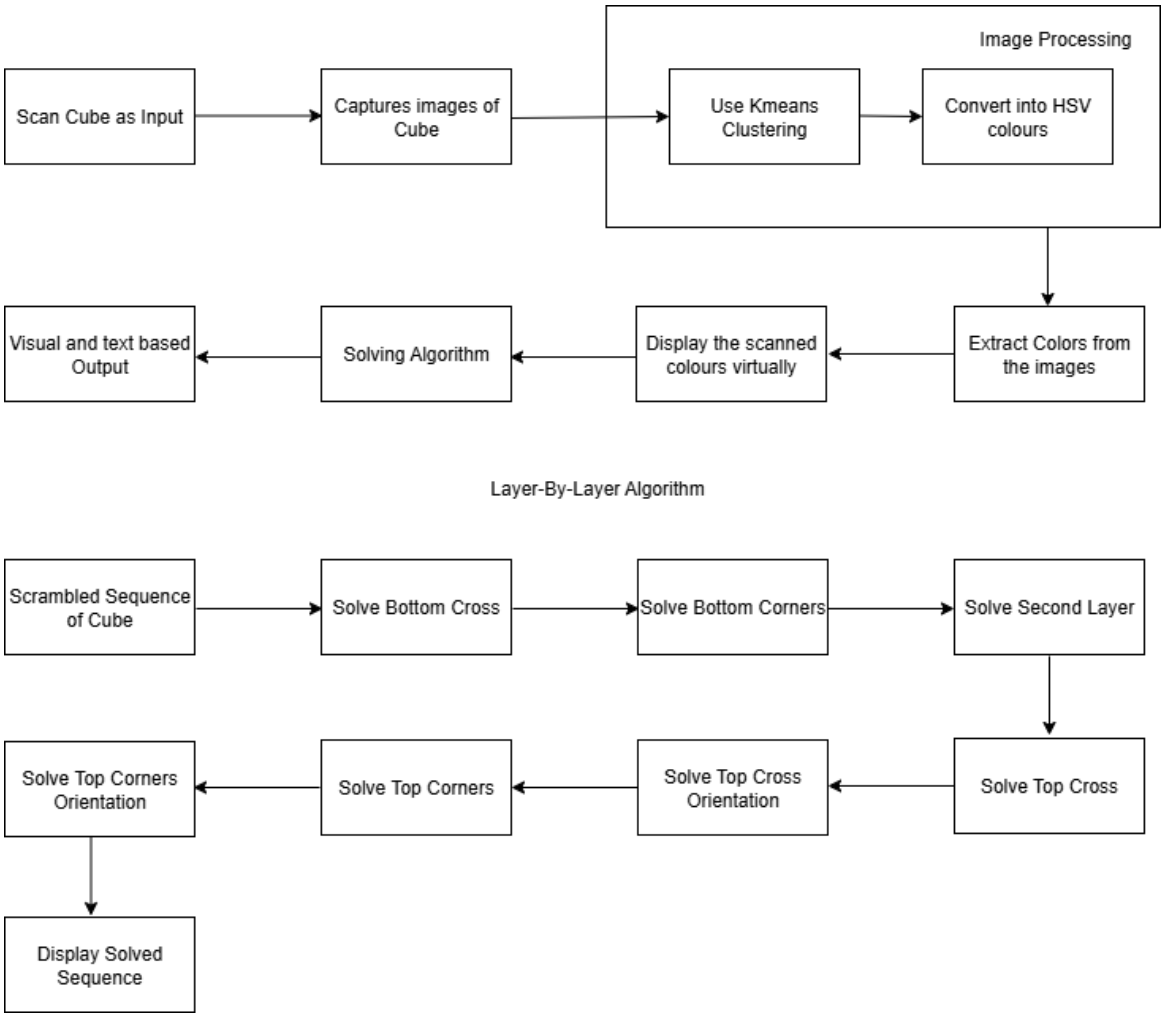

Figure 4.1: System Design for Cube Master

- Image Capturing

The process begins when the user decides to solve a Rubik's Cube with the Cube Master system. They can choose to use their device's camera to scan the cube. The system then helps the user turn the cube and take clear pictures of all six sides so that each face is properly recorded. At this stage, the system takes pictures with a resolution of 30x30 pixels to accurately identify the color of each square on the cube. This step is important for creating a complete and exact virtual copy of the cube's current state.

- Image Preprocessing

Once the images are taken, the system goes through a preparation step to make the scanned images clearer and more uniform. The system then looks at the pixel data with a method called K-means clustering, which helps group similar colors together and find the main color in each area. This helps prepare the data better for accurate color detection when solving the problem.

- Manual Input (if the scan is incorrect)

If the scanned data is not complete or correct, like when a cube face is missing or not captured properly, the system gives users the chance to enter information manually. They can pick and assign colors to each of the cube's six faces themselves, making sure the virtual version looks exactly like the real cube. This option helps the system continue solving the cube even if the scanning process has problems.

- Layer-by-Layer Algorithm

Once the cube's configuration is successfully captured, users can go on to solve it at any point that they use the layer-by-layer method. Cube Master starts with validating and checking if the cube's state is solvable. The system executes on this trusted and structured solving technique. Confirmation is required beforehand. Because it starts with the bottom layer, goes to the middle layer, then ends with the top layer, the algorithm finds what moves solve the cube in stages. Users solve the cube accurately coupled with efficiency via this clear step-by-step solution.

- Kociemba Algorithm

Alternatively, users can opt for the Kociemba algorithm, which focuses on computing the most optimal solving sequence. Once Cube Master confirms the cube's solvability,

it utilizes the Kociemba algorithm to calculate the minimal number of moves required to solve the cube. This approach serves as a benchmark for efficiency, offering advanced users a powerful and precise solution path.[8]

- Visual Display of Output

A visual representation of the Rubik's Cube in 3D as well as solved sequence, showing the current configuration to the solved state. This allows users to track the progress of the solution in a dynamic, interactive format.

- Error message

If the system spots any problems—like an invalid scan or a cube configuration that just can't be solved—it quickly lets the user know with straightforward error messages. These messages guide users in tackling the issue, whether that means rescanning the cube properly or checking if it can actually be solved. This way, users stay informed about any hurdles that might get in the way of solving the cube. The system's method makes working on the Rubik's Cube simpler and more intuitive, providing clear instructions and feedback through both visual signals and text.[9]

4.1.1 Refinement of Class/Object Diagram

The refined object diagram provides a detailed representation of all the classes within a system and illustrates the relationships between them.

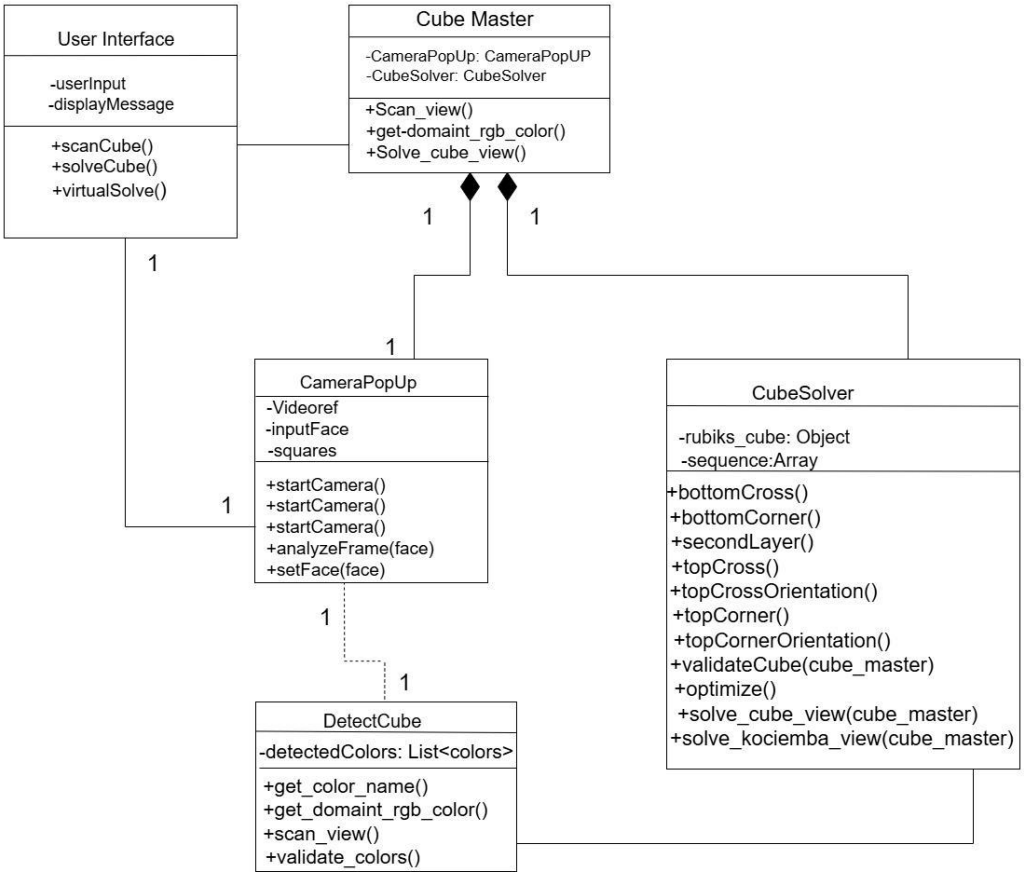

Figure 4.2: Refined Object Diagram for Cube Master

Figure 4.2 The Rubik's Cube Solver app is built with a clear structure using several main parts, each with its own job. At the center is the Cube Master class, which acts as the main manager. It handles taking pictures of the cube, figuring out how to solve it, and showing the steps visually. The CameraPopUp class starts the camera on the phone, takes a picture of the cube, and picks out the colors from specific spots on the cube. These color details are then given to the DetectCube part, which uses special calculations to recognize and arrange the cube's colors correctly. Once the cube's current state is known, the CubeSolver class finds the solution using either the Layer-by-Layer method or the Kociemba algorithm, based on what the user chooses. It goes through steps like solving the bottom cross, arranging the middle layer, and fixing the top face. For showing the solving process, the SolutionCube class creates animations in a 3D space, while the VirtualCube lets users play with a digital cube on their own. The UserInterface class makes sure everything works smoothly by letting users pick different modes, showing visual clues, and giving step-by-step instructions. All these parts work together to make the Cube Master system an easy and effective way to solve Rubik's Cubes both in real life and on a screen.

4.1.2 Refinement of Sequence Diagram

A refined sequence diagram offers a detailed depiction of the interactions between various objects or components within a system over time. It outlines the chronological flow of messages exchanged among these objects, providing a clear and structured view of their communication and behavior in a specific sequence of events.

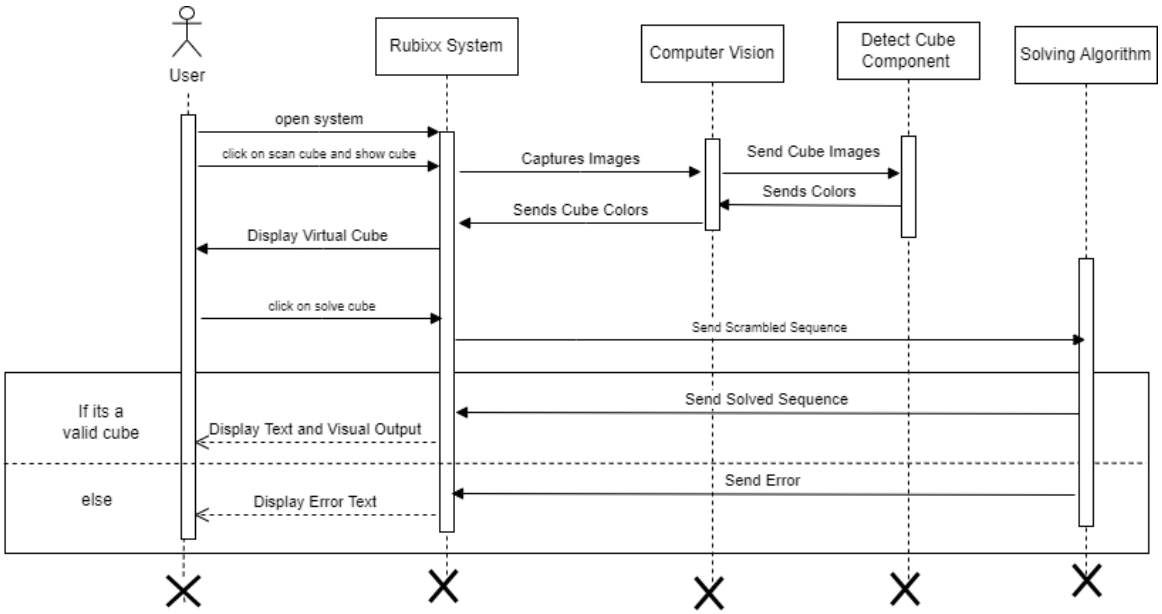

Figure 4.3: Refined Sequence Diagram for Scanning through CV

Figure 4.3 when the user starts the Cube Master system and chooses the "Scan Cube" option, the system turns on the device's camera to take pictures of each side of the Rubik's Cube. These pictures are sent to the Detect Cube part, which uses a method called K-means clustering to find and group the colors. It then changes these colors into the standard color names used for Rubik's Cubes. After the colors are identified, the information goes back to the Cube Master system. If the cube's setup is correct and can be solved, the system creates a clear step-by-step solution in text and also shows an animated 3D video of the cube being solved. These results are shown to the user. But if the system finds that the cube is not set up properly or has wrong color matching, an error message appears, telling the user exactly what the problem is. This whole process helps users go from scanning the cube to seeing the solution or fixing any issues with the scan.

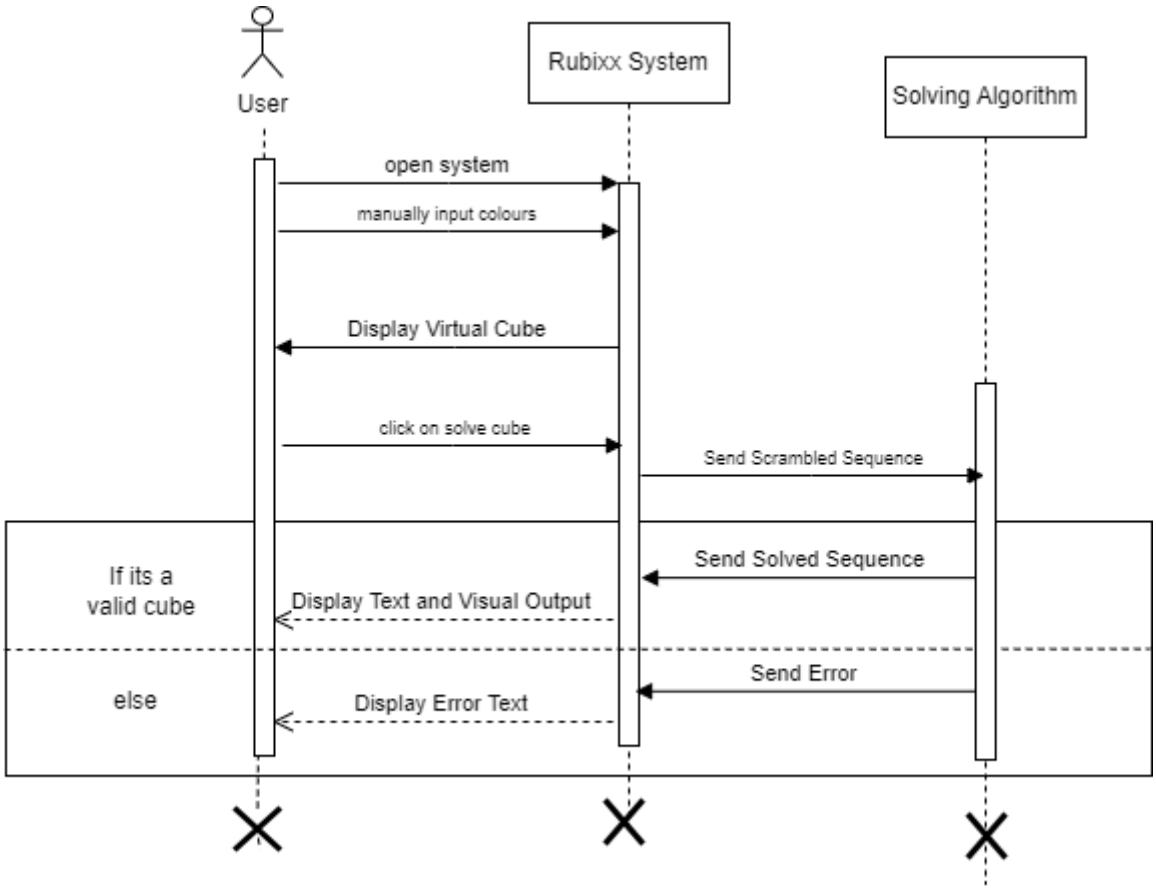

Figure 4.4: Refined Sequence Diagram by manually inputting the cube

Figure 4.4 when a user manually enters the colors of each square on all six faces of the Rubik's Cube, the system shows a clear diagram of the Cube Master process. The user starts by opening the system and entering the colors for every square. Once they finish and send the information, the system checks if the cube is in a solvable state. If the cube is valid, the Cube Master system creates a solution. This includes a written step-by-step guide and a moving 3D animation to show how to solve it. The user gets both the text and the animation as the solution. However, if the cube is not in a solvable state, the system shows an error message that explains what is wrong. This way of entering the cube's state makes the experience easy to use. The user gets clear messages and helpful results, whether the cube is solvable or not.

4.1.3 Refinement of Activity Diagram

A refined activity diagram is a type of UML diagram that offers a clear and detailed representation of the control flow and sequence of actions within a system, illustrating the progression of events in a structured manner.

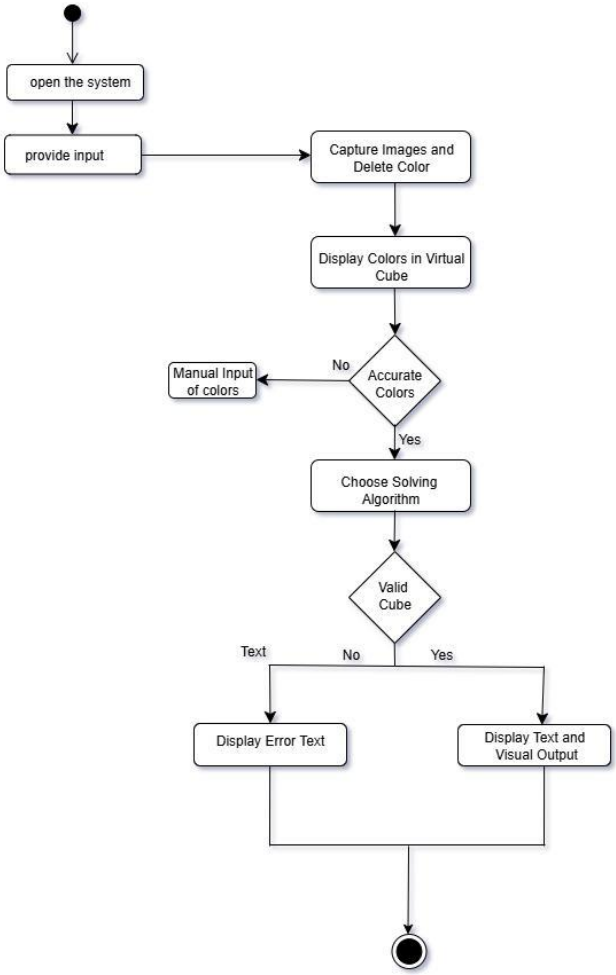

Figure 4.5: Refined Activity Diagram for Scanning through CV

Figure 4.5 when the user manually enters the Rubik's Cube into the system, the refined sequence diagram shows how the process works. It starts when the user opens the system, turns on the camera, and positions the Rubik's Cube so it fits in the camera's view. The system takes pictures of each face of the cube and sends those images to a part of the program that identifies the color of each sticker. These color labels are then given to another part of the program called the solver. The solver uses the colors to create a digital version of the cube and checks if the colors are set up correctly. If the cube's setup is correct and can be solved, the solver chooses a solving method and figures out the steps needed to solve it. The user then sees both a list of the moves written out in text and a 3D animation showing the solution step by step. If the cube's setup isn't correct, the system shows an error message that explains what's wrong, like having the wrong number of colors or faces that don't match up.

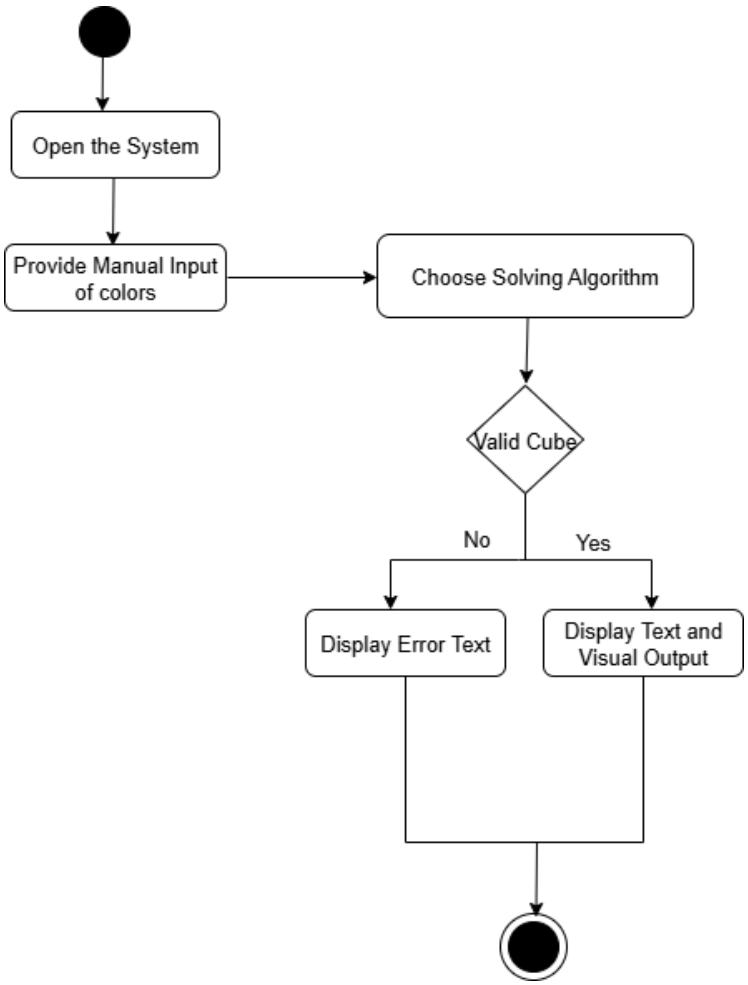

Figure 4.6: Refined Activity Diagram by manually inputting the cube squares

Figure 4.6 presents the refined activity diagram for Cube Master, focusing on the process of manually inputting the cube's configuration. The diagram illustrates the interaction between the user and the Cube Master system, beginning with the user opening the system and manually entering the colors for all squares across all cube faces. If the inputted cube configuration is valid and solvable, the system provides a text-based solution sequence along with a 3D visual representation of the solving process for the user depending on the choice of solving algorithm by the user. However, if the cube is invalid or unsolvable, the system displays an error message highlighting the specific issue, such as an incorrect color arrangement or misalignment of the cubes.

4.1.4 Component Diagram

A component diagram breaks down the system under development into separate functional units. Each component serves a distinct role within the overall system and interacts with

other components only when necessary, ensuring efficient and purpose-driven communication.

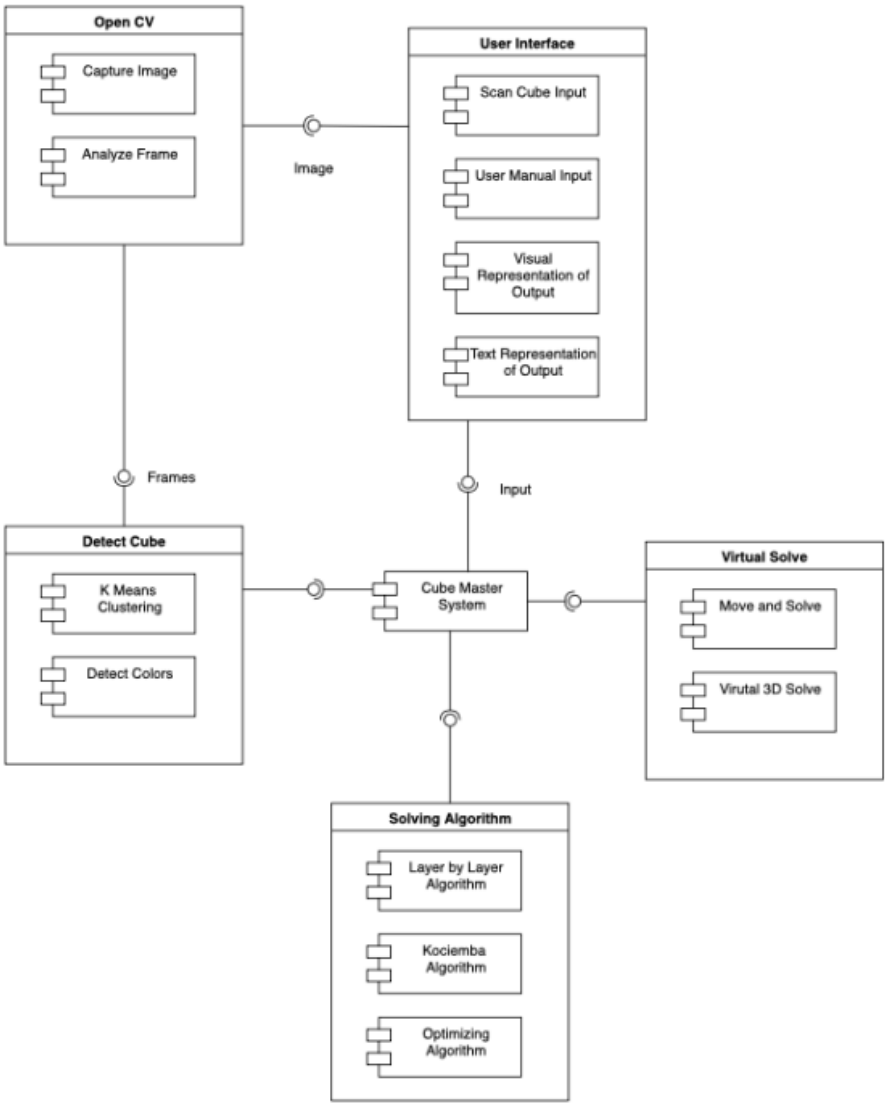

Figure 4.7: Component Diagram for Cube Master

Figure 4.7 shows the component diagram for Cube Master system. This project integrates OpenCV, Layer-by-Layer Algorithm or Kociemba's Algorithm for solving and Three.js for Visual Representation of Output. The OpenCV captures and converts image frames which are then clustered using K-means Frequent clustering for color detection. Then Layer-by-layer algorithm uses its 7-layer algorithm or the optimal Kociemba algorithm to solve the Rubik's cube. The optimizing algorithm makes sure the solution is most optimal. If an error occurs in a any layer of the solving method, it displays what the error is to the users. The User Interface accepts user input and provides Visual as well as Text- Sequence Output.

These components are connected to a central “Cube Master System” component indicating their integration in this system.

4.1.5 Deployment Diagram

The deployment diagram illustrates how the components of the Cube Master systems are distributed across various hardware devices, ensuring a seamless flow of data and efficient operation of the system. It highlights the interaction between the client, web server, and application server, as well as the roles of the artifacts and components deployed on each.

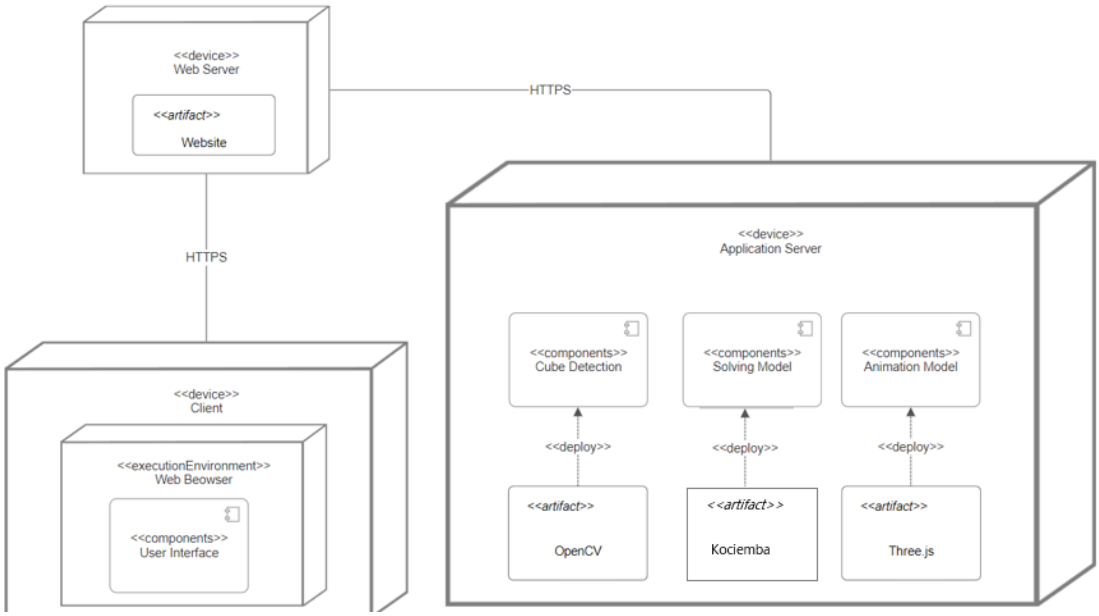

Figure 4.8: Deployment Diagram for Cube Master

Figure 4.8 shows that the client-side is hosted within a Web Browser Execution Environment and features a User Interface component. This component allows users to interact with the system and communicates securely with the server-side via the HTTPS protocol. The client acts as the entry point for user engagement, ensuring accessibility and ease of use. The web server plays a crucial role in hosting the Website Artifact, which acts as the intermediary between the client and the application server. It ensures secure communication with both sides through HTTPS, facilitating the smooth transmission of requests and responses while maintaining data security.

The application server is the backbone of the system, containing three primary components: the Cube Detection Model, the Solving Model, and the Animation Model. The Cube Detection Model leverages the OpenCV Artifact to process images and recognize colors,

providing accurate color analysis. The Solving Model performs logical computations to offer optimal solutions to solve a rubiks cube. The Animation Model, powered by the Three.js Artifact, generates dynamic 3D animations of the cube to enhance user experience and visualization.

4.2 Algorithm Detail

4.2.1 Cube Detection Algorithm (CV - K-Means Clustering)

The Cube Master system utilizes a robust computer vision-based algorithm powered by K-Means clustering to detect and interpret the state of a Rubik's Cube. This approach identifies and classifies the six distinct colors present on the cube's surface from captured images or video frames. The process begins with input preprocessing, where the system decodes and processes images or frames to enhance clarity and reduce noise through Gaussian blur. Following preprocessing, the cube is divided into regions of interest (ROIs), with each face segmented into a 3x3 grid. Individual cells within these grids are isolated and analyzed to determine their color.[10]

To identify colors, the pixel values of each ROI are flattened into a two-dimensional array, and K-Means clustering is applied with k=7 clusters (6 cube colors + background noise), representing the six cube colors plus background noise. The cluster containing the maximum number of pixels determines the dominant color of the cell. Once the dominant RGB color is identified, it is converted to the HSV color space, which provides better differentiation of the cube's colors. Using predefined thresholds for hue, saturation, and value, the system maps each ROI to one of six colors: red, orange, yellow, green, blue, or white. The K-Means algorithm minimizes the distance between data points and their respective cluster centroids, ensuring accurate color detection and mapping.

```
import numpy as np

import cv2

def get_color_name(h, s, v):

    # print(h, s, v)

    if s < 65 and v > 50:

        return "white"
```

```
    if h <= 1 or h > 160:
        return "red"
    if 178 <= h or 0 <= h < 27:
        return "orange"
    if 27 <= h < 45:
        return "yellow"
    if 45 <= h < 85:
        return "green"
    if 85 <= h < 160:
        return "blue"
    return "Unknown"
def get_dominant_rgb_color(roi):
    """Get dominant RGB from a certain region of interest.
    :param roi: the image array
    :returns: tuple
    """
    pixels = np.float32(roi.reshape(-1, 3))
    n_colors = 7
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200,
0.1)
    flags = cv2.KMEANS_RANDOM_CENTERS
    _, labels, palette = cv2.kmeans(pixels, n_colors, None, criteria, 10, flags)
    _, counts = np.unique(labels, return_counts=True)
    dominant = palette[np.argmax(counts)]
    hsv_frame = cv2.cvtColor(np.uint8([[dominant]]), cv2.COLOR_BGR2HSV)[0][0]
```

[h, s, v] = hsv_frame

color = get_color_name(h, s, v)

return color

Key Equation - K-Means Objective Function

$$J = \sum_{i=1}^{k} \sum_{x \in c_i} \| x - \mu i \|^2$$

4.2.2 LAYER-BY-LAYER Solving Algorithm

The Layer-by-Layer (LBL) solving algorithm is one of the most intuitive and widely used methods for solving the Rubik's Cube. It divides the solution process into three stages: solving the bottom layer, the middle layer, and finally the top layer. Each layer is solved sequentially, and the goal is to preserve the solved sections of the cube while progressing to the next layer.

Before diving into the detailed explanation of the LBL method, it is important to understand the standard notations used to describe cube movements.

Notations in Rubik's Cube Movements

The Rubik's Cube is divided into six faces, each represented by a unique letter:

- U (Up): The face on the top.
- D (Down): The face on the bottom.
- L (Left): The face on the left.
- R (Right): The face on the right.
- F (Front): The face facing you.
- B (Back): The face opposite to the front.

Each notation represents a 90-degree clockwise turn of the specified face unless stated otherwise:

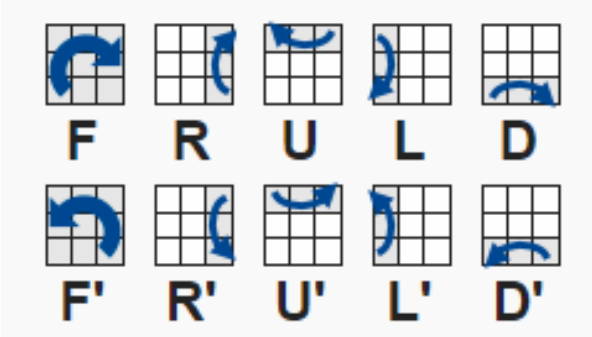

Figure 4.9: Cube Master Notations

- **U, D, L, R, F, B**: Rotate the respective face 90° clockwise.
- **U', D', L', R', F', B'**: Rotate the respective face 90° counterclockwise (denoted by the apostrophe or prime symbol ').
- **RR, RL, RU, RD:** Rotate the cube right, left, up and down respectively.

Understanding these notations is crucial for executing the steps of the LBL algorithm.

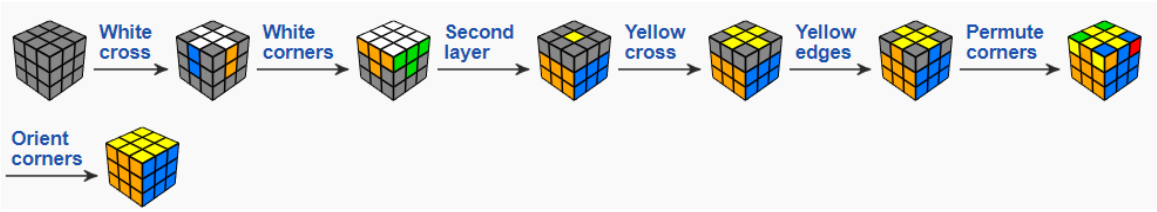

Figure 4.10: LBL Solving Method

- Solving the Bottom Layer

The first step in the layer-by-layer (LBL) method is all about tackling the bottom layer, starting with creating the cross. This means you need to position the edge pieces on the bottom face so they line up perfectly with the bottom center and the centers of the adjacent faces. Each edge piece has to match the colors of these center pieces just right. Once you've got the cross set up, the next challenge is to place and orient the corner pieces to finish off the bottom layer. You can spot these corners by their three-colored stickers, which should align with the centers of the three surrounding faces. You'll use specific moves to get each corner in the right spot, all while keeping that cross intact throughout the whole process.

For example:

- To place a corner piece correctly, you might use a sequence such as **R U R'** or **U R U' R'**, where **R** refers to a turn of the right face and **U** adjusts the orientation of the cube.

- Solving the Middle Layer

After you've tackled the bottom layer, it's time to shift your focus to the middle layer. Here, the goal is to carefully place and align the edge pieces that fit into this section. Each edge piece features two colors, and they need to match the colors of the center pieces on the adjacent faces. To insert these edges without disturbing the bottom layer, we rely on special algorithms. If you find an edge in the wrong spot, you can use a specific sequence of moves—like U R U' R' U' F' U F—to remove it and put it back in the right place. This stage is all about maintaining the integrity of the completed bottom layer while you work on building up the middle layer.[5]

- Solving the Top Layer

The top layer is solved in two stages: first, forming the cross on the top face and then positioning and orienting the corner pieces.

1. **Forming the Top Cross:**
   In the top layer, the edge pieces are arranged to create a cross, much like what you do for the bottom layer. Depending on the current pattern of the cube, you'll use specific algorithms—like F R U R' U' F'—to move the edges around and form that cross. These sequences carefully adjust the edge pieces one step at a time until the cross shape you want shows up on the top face.
2. **Positioning and Orienting the Corners:**
   The first step in solving the top layer corners is to get them into the right spots, even if they're not oriented correctly just yet. To do this, you can use algorithms like U R U' L' U R' U' L, which help swap and position the corners just right. Once all the corners are in their correct locations, you can apply some additional moves to fix their orientation while keeping the rest of the cube just as it is.
3. **Final Adjustment**
   The last step involves correcting the orientation of the edge pieces in the top layer, ensuring that the cube is fully solved.

4.2.3 Optimization Algorithm

The optimization algorithm refines the solving sequence by eliminating redundant moves and simplifying opposing moves, ultimately reducing the total number of steps required to solve the cube. The input to the algorithm is a sequence of moves, which include face

rotations (e.g., R, L, U, D, F, B) and their inverses (e.g., R', L', etc.). The algorithm works iteratively, processing moves in a sequence to identify patterns that can be optimized.
The first step in optimization is to identify and remove four consecutive identical moves, as these result in a net zero rotation of the cube. For example, four consecutive R moves cancel out entirely. Similarly, three consecutive identical moves are simplified to their inverse; for instance, R R R simplifies to R'. In cases where two opposing moves are found consecutively, such as R followed by R', they cancel each other out. The algorithm applies a sliding window of size four across the sequence to detect and process these patterns efficiently.

The optimization process continues iteratively until no further changes can be made. By simplifying redundant and opposing moves, the algorithm produces a more efficient and concise solving sequence. This not only improves computational performance but also reduces the physical effort required for manual or robotic cube solving. It enhances the overall user experience by delivering cleaner, optimized solutions for the Rubik's Cube.

Key Equation:

Four-Move Cancellation:

If $Mi = M_{i+1} = M_{i+2} = M_{i+3}$ then remove Mi, M_{i+1}, M_{i+2}, M_{i+3}

Three-Move Simplification:

If $Mi = M_{i+1} = M_{i+2}$ then M_i'

4.2.4 Kociemba's Solving Algorithm

Besides the usual way of solving the 3x3x3 Rubik's Cube one layer at a time, we also used Kociemba's algorithm to make the solving process more efficient. This algorithm was created by Herbert Kociemba and is a computer-based method that improves the solving process with smart computational techniques. It builds on the ideas from Thistlethwaite's Algorithm but makes the cube easier to solve in fewer steps by reducing its complexity in a more strategic way.

$P(s) = \min\{d(s,g)\} \mid g \in G_{target}$

Where,

- G_0: Full group of all possible moves.
- G_1: Subgroup with all edges oriented.
- G_2: Subgroup with restricted moves to solve completely.
- Pruning function: P(s): Minimum moves required from state s to the target group.
- d(s,g) represents the distance (in terms of moves) between state s and g.

The algorithm is divided into two primary phases. In the first phase, the cube is reduced to a restricted subgroup, G_1, where all edge pieces are correctly oriented and belong to a specific subset of states. This is achieved by applying moves from the full move set (G_0), which includes all possible cube rotations. A precomputed heuristic, called a pruning table, guides the solver to achieve this reduction in the minimal number of moves. Once the cube is in G_1, the second phase solves the cube completely using a reduced move set (G_2) that simplifies the task by focusing on permuting and orienting pieces within the restricted subgroup. The algorithm employs another pruning table in this phase to ensure efficiency and minimize the number of moves to reach the solved state. By leveraging group theory, Kociemba's algorithm breaks down the problem into manageable subgroups, significantly reducing the complexity of the state space. Its reliance on precomputed pruning tables allows it to achieve solutions in an average of 30-32 moves, often close to "God's number," which represents the theoretical minimum number of moves to solve any scramble. Although the algorithm requires substantial computational resources for precomputations, modern computers handle this efficiently, making it a practical choice for real-time applications.

Chapter 5: Implementation and Testing

5.1. Implementation

The Rubik's Cube Solver was created step by step: first, they set up the cube model, then moved on to color detection, followed by developing solving algorithms, and finally, they designed the user interface. Each component was carefully polished through a process of testing and debugging to make sure everything worked accurately and was easy to use.

5.1.2 Tools Used

To build the Cube Master system, a combination of modern tools and technologies was used to ensure a seamless and efficient experience. Here are the key tools:

- Software Development Tools

  Django forms the backbone of the backend, taking care of system logic and routing in an efficient way. It sets up the framework needed to build a scalable web application. For the frontend, React.js brings dynamic, interactive user interfaces to life, like the virtual Rubik's cube. Vite is the tool that powers the frontend build process, making everything fast and modern. Tailwind CSS allows us to style the app using utility-first classes, making it easy to create responsive and visually pleasing designs. Django-cors-headers ensures smooth communication between the frontend and backend by allowing Cross-Origin Resource Sharing (CORS). Our primary development environment is Visual Studio Code (VS Code), which is perfect for coding, debugging, and collaborating. Postman is our go-to tool for testing API endpoints and making sure data flows smoothly between the frontend and backend.

- Diagram Tools

  For designing and visualizing the system's architecture, we use draw.io. It's an intuitive tool that helps us create everything from use case diagrams to class diagrams, giving us a clear picture of how the system works and how the components interact with each other.

- Hardware Tools

A powerful CPU (central processing unit) is crucial for processing data, running system logic, and ensuring the application runs smoothly. To handle all the data and tasks efficiently, we recommend a minimum of 8 GB of RAM to support multiple processes at once without hiccups.

5.1.2 Implementation Details of Modules

The project is broken down into several key modules, each responsible for specific tasks within the Cube Master system.

- **Cube Solver Algorithm Module**

```
class RubiksCubeSolver:
    def __init__(self, rubiks_cube):
        self.rubiks_cube = rubiks_cube
        self.sequence_cube = rubiks_cube
        self.sequence = []

    def solve(self):
        self._solve_bottom_layer()
        self._solve_second_layer()
        self._solve_top_layer()

        self.sequence = self._optimize(self.sequence)

        for seq in self.sequence:
            self.sequence_cube = moves.execute_move(self.sequence_cube, seq)

        if self.sequence_cube != self.rubiks_cube:
            raise serializers.ValidationError(
                "The cube state does not match the expected solved state. Sequence Error!"
            )

        return self.sequence, self.sequence_cube

    def _solve_bottom_layer(self):
        bottom_solver = BottomLayerSolver(self.rubiks_cube)
        moves, updated_cube = bottom_solver.solve()
        self.sequence.extend(moves)
        self.rubiks_cube = updated_cube

    def _solve_second_layer(self):
        second_solver = SecondLayerSolver(self.rubiks_cube)
        moves, updated_cube = second_solver.solve()
        self.sequence.extend(moves)
        self.rubiks_cube = updated_cube

    def _solve_top_layer(self):
        top_solver = TopLayerSolver(self.rubiks_cube)
        moves, updated_cube = top_solver.solve()
        self.sequence.extend(moves)
        self.rubiks_cube = updated_cube
```

Figure 5.1: Solving Process of Cube Master Solver

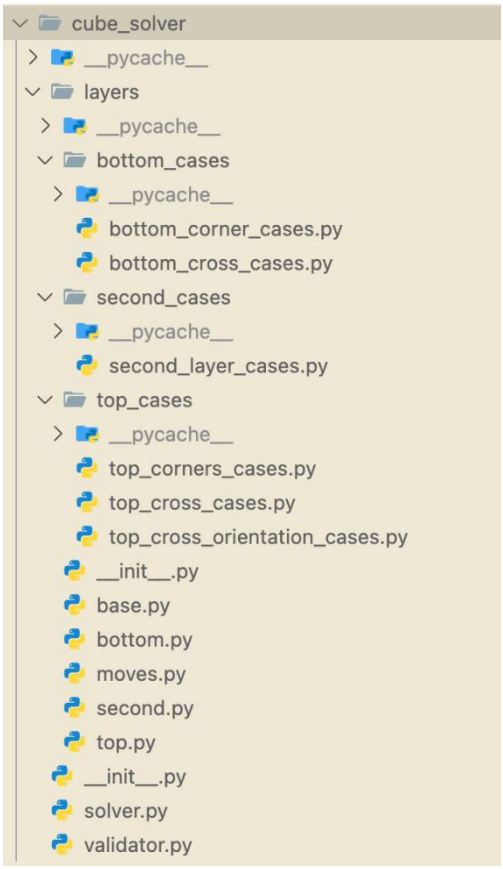

Figure 5.2: Solving Modules of Cube Master Solver

- **Cube Detection Module with OpenCV**

```
def get_color_name(h, s, v):
    # print(h, s, v)
    if s < 65 and v > 50:
        return "white"
    if h <= 1 or h > 160:
        return "red"
    if 178 <= h or 0 <= h < 27:
        return "orange"
    if 27 <= h < 45:
        return "yellow"
    if 45 <= h < 85:
        return "green"
    if 85 <= h < 160:
        return "blue"
    return "Unknown"


def get_dominant_rgb_color(roi):
    """Get dominant RGB from a certain region of interest.

    :param roi: the image array
    :returns: tuple
    """
    pixels = np.float32(roi.reshape(-1, 3))

    n_colors = 7
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, 0.1)
    flags = cv2.KMEANS_RANDOM_CENTERS
    _, labels, palette = cv2.kmeans(pixels, n_colors, None, criteria, 10, flags)
    _, counts = np.unique(labels, return_counts=True)
    dominant = palette[np.argmax(counts)]
    hsv_frame = cv2.cvtColor(np.uint8([[dominant]]), cv2.COLOR_BGR2HSV)[0][0]
    [h, s, v] = hsv_frame
    color = get_color_name(h, s, v)
    return color
```

Figure 5.3: Cube Detection Module with OpenCV

5.2 Testing

5.2.1 Test Case for Unit Testing

Unit testing focuses on verifying individual components of the system to ensure that each part functions as expected. Below are the test cases for the Cube Solver Algorithm, UI components, and other modules.

Table 5.1: Test Case for Solving Algorithm Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_001	Solving a Valid Scrambled Cube (Simple Scramble)	Simple scrambled configuration with first two layers scrambled	Correct solving steps for first layer, middle layer, and last layer with the cube solved	The cube was solved layer by layer with the correct solving steps for the first, middle, and last layers.	Pass
2	TC_002	Solving a Valid Scrambled Cube	Random scrambled configuration with all layers scrambled	Cube is solved layer by layer: first layer, middle layer, and last layer. Sequence of moves is correct.	The cube was solved layer by layer with the correct sequence of moves for each layer.	Pass
3	TC_003	Solving an Invalid Cube (Unsovable Configuration)	Unsolvable cube configuration (e.g., corner and edge swap)	System detects the unsolvable configuration and provides an error or	The system detected the unsolvable configuration and displayed an error	Pass

				warning message	message as expected.	
4	TC_004	Solving a Cube with Incorrectly Inputted Colors	Invalid manual configuration (e.g., repeating colors)	System detects invalid input and prompts the user to correct the configuration	The system detected invalid input and prompted the user to correct the configuration.	Pass
5	TC_005	Solving a Cube with No Scramble (Already Solved)	Solved cube configuration	Since the cube is already solved, no solving steps will be required. The system will simply confirm that the cube is solved.	The system recognized the already solved cube and confirmed that no solving steps were needed.	Pass

Table5.2: Test Case for UI Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_006	Inputting Cube Configuration Manually	Manual color input for cube configuration	Cube is recognized, and input is validated correctly.	The cube was recognized, and the input was validated	Pass

				Solving steps are generated.	correctly. The solving steps were generated.	
2	TC_007	Scanning Cube Using Computer Vision	Scanning the cube using the camera	Cube's colors are detected, and the configuration is validated. Solving steps are displayed.	The system correctly detected the cube's colors, validated the configuration, and displayed the solving steps.	Pass

Table 3.3: Test Case for Animation and Visualization Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_008	Inputting Cube Configuration Manually	Manual color input for cube configuration	Cube is recognized, and input is validated correctly. Solving steps are generated.	The cube was recognized, and the input was validated correctly. The solving steps were generated.	Pass
2	TC_009	Scanning Cube Using Computer Vision	Scanning the cube using the camera	Cube's colors are detected, and the	The system correctly detected the cube's colors,	Pass

				configuration is validated. Solving steps are displayed.	validated the configuration, and displayed the solving steps.	

5.2.2 Test Case for System Testing

System testing evaluates the interaction between multiple components to ensure that the entire system functions correctly as a whole. The tests below involve running the entire system, including the cube solver, UI, animation, and random scramble generation, to verify that the system meets the required functionality.

Table 5.4: Test Case for Full System Integration Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_010	Full System Integration	Combination of manual input and scanning configuration	Cube is scanned or manually input, and solving steps are generated and displayed.	The system integrated the input and scanning process and generated solving steps correctly.	Pass
2	TC_011	User Interface Functionality	Manual and scanning inputs combined	User is able to interact with the virtual cube, input	The UI was functional with real-	Pass

				configurations, and view animations.	time updates and interaction.	

Table 5.5: Test Case for Performance Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_012	Animation Rendering Speed	Scrambled cube configuration	Animation of solving steps renders smoothly with no lag or stutter.	Animation rendered smoothly without any delays.	Pass

Table 5.6: Test Case for Usability Test

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_013	Usability of 3D Cube Color Reflection.	User scans a physical Rubik's Cube using the camera.	The 3D virtual cube updates instantly to reflect the scanned cube colors, ensuring	The 3D cube displayed the exact scanned colors correctly, allowing the user to compare with the real cube	Pass

				easy visual verification		
2	TC_014	Usability of Move Visualization.	User applies a solving move from the solution sequence.	The 3D virtual cube updates instantly to show the move applied, making it easy for the user to follow.	The 3D cube rotated correctly according to the applied move and was easy to track visually.	Pass

Table 5.7: Test Case for Kociemba's Algorithm vs Layer-By-Layer Algorithm

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC_015	Testing move optimization with kociemba's algorithm	25 random scrambles.	Kociemba's algorithm gives more optimized and optimal solution for solving the cube.	Kociemba's average move required to solve the cube was 32 whereas with the LBL algorithm it was 154.	Pass

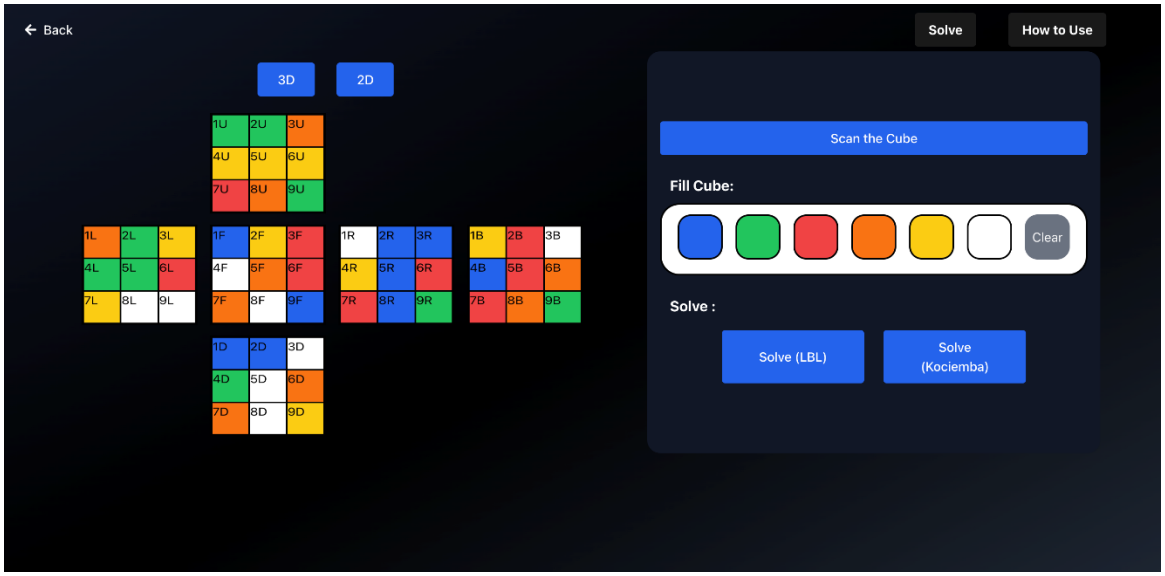

Figure 5.4: UI Representation of 2D Cube & Solving Algorithm

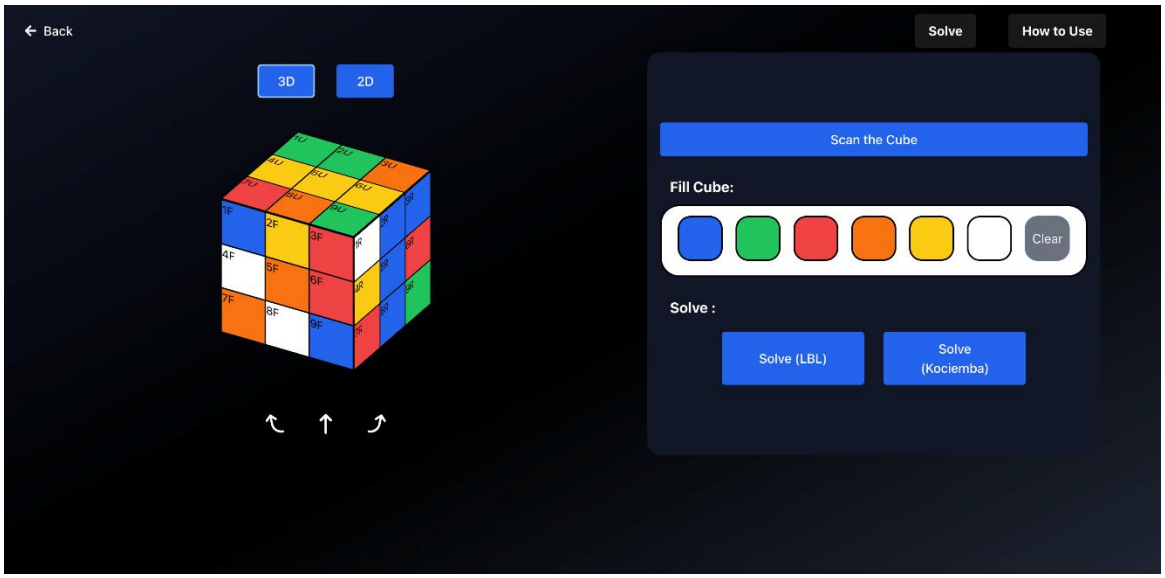

Figure 5.5: UI Representation of 3D Cube & Solving Algorithm

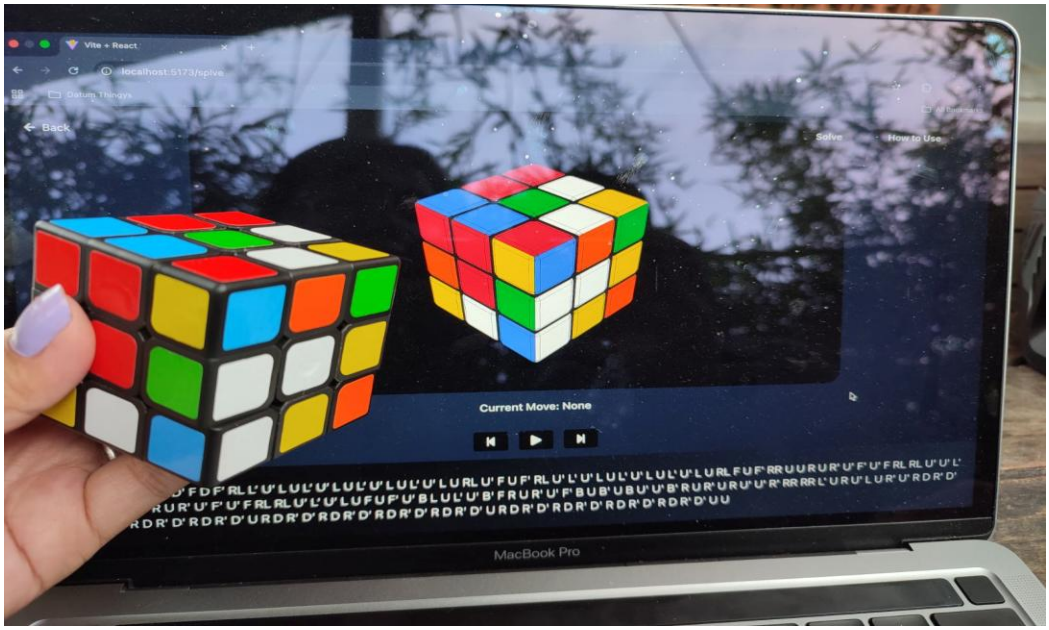

Figure 5.6: Scanning Cube and Getting Solving Algorithm

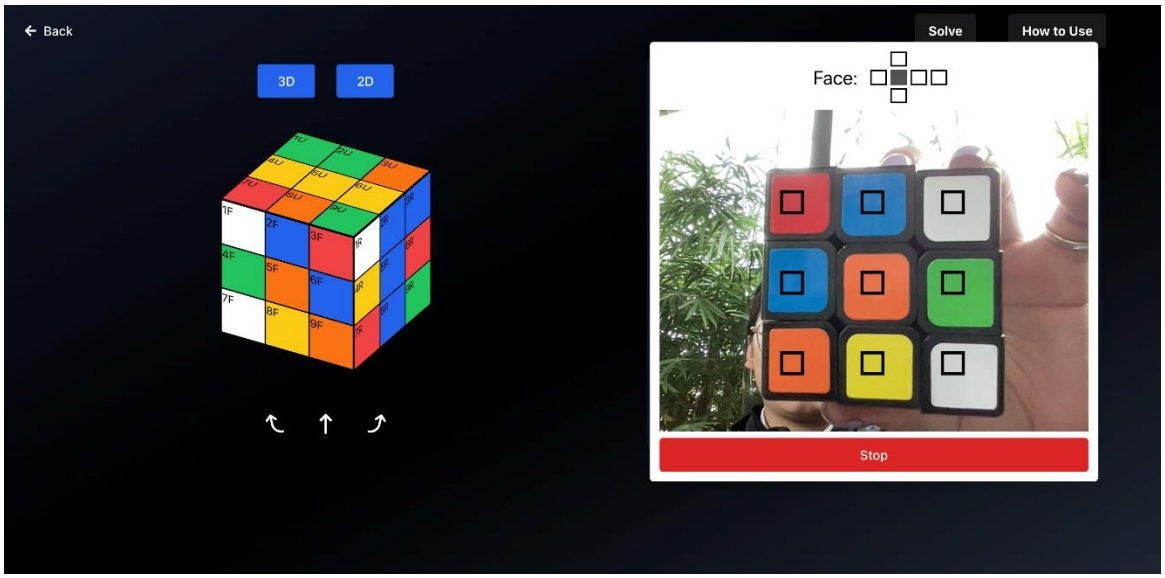

Figure 5.7: Solving the cube using the provided instructions

5.3 Result Analysis

This section discusses the performance and efficiency of the Cube Master's solver, analyzing its effectiveness in detecting the cube's state, identifying the optimal solution steps, and executing the solving algorithm. The system underwent various testing procedures, including unit testing, system testing, and edge-case scenarios to ensure reliable performance. The results indicate that the system is effective under standard conditions, but there are areas that can be improved to ensure robustness across different environments.

5.3.1 Image Recognition and Cube State Detection

The first key component of the system is the accurate identification of the cube's current configuration. During testing, various Rubik's Cube states were captured under different lighting conditions, and the system was evaluated on its ability to correctly classify the color patterns on each face. The cube's six faces were identified and mapped to the standard color scheme (white, yellow, blue, green, red, and orange).

- Accuracy: The system achieved an accuracy rate in correctly recognizing the colors of the cube's faces under normal lighting conditions.
- Lighting Conditions: While the system performed well under optimal lighting, there were significant variations in accuracy when tested in not optimal light environments. The recognition accuracy dropped by approximately in such conditions.
- Speed: The time taken to recognize the cube's state averaged , making the detection process quick enough for real-time solving.

The system demonstrated robustness in detecting cube states, but further improvements in lighting adaptability are recommended for performance in varied environments.

5.3.2 Solving Algorithm Efficiency

The solver algorithm, which calculates the sequence of moves needed to solve the Cube Master, was tested to ensure it correctly solved the cube in the least number of moves possible. Several different cube configurations, ranging from scrambled to nearly solved states, were used to evaluate the algorithm's effectiveness.

Along with the normal LBL method, kociemba's algorithm for solving cube master was also utilized, which generated drastic differences in move required to solve a cube master.

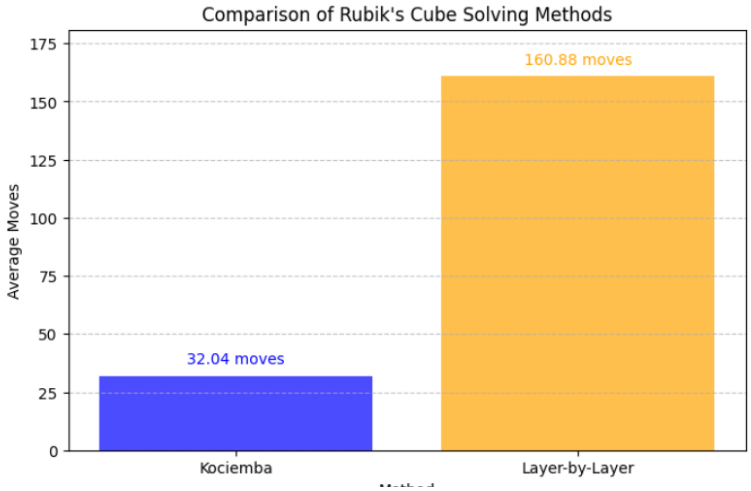

Figure 5.8: Comparison of Cube Master Solving Methods

- **Accuracy**: The solver demonstrated a high degree of accuracy, with a success rate in solving the cubes correctly. In some edge cases, such as certain cube configurations with multiple centers in an unsolved state, the solver needed a few extra steps but still reached the correct solution.
- **Solver Time**: On average, the solver algorithm completed a full solve after recognizing the cube's state. The solver's response time was consistent across a variety of tests, making it efficient for real-time solving.
- **Move Optimization**: The algorithm utilized a basic solving technique. Automated testing was used to compare the unoptimized and optimized moves needed to solve the cube.

5.3.3 Usability and User Interaction

The system's usability was also tested by evaluating how easy it was for users to interact with the solver, capture the cube's state, and understand the provided solution. Users were given various cube configurations, and they interacted with the system to obtain the solution steps.

- **User Experience**: The system performed well in terms of user-friendliness. Users were able to easily capture the cube's state by placing it in front of the camera and receiving step-by-step instructions for solving it.

- **Instructions**: The system provided visual step-by-step guidance, highlighting the moves needed to solve the cube. This feature was particularly helpful for beginners who wanted to learn how to solve the cube manually.

5.3.4 System Performance Under Different Conditions

Finally, the system's performance was tested under varying conditions, including changes in cube orientation, cube size, and environmental factors.

- **Cube Orientation**: The system encountered challenges when handling the cube in various orientations, such as tilted or rotated angles. The algorithm struggled to accurately recognize the cube's state when held at extreme angles, leading to a failure in some test cases. This resulted in a longer recognition time and a failure to determine the correct cube configuration in certain orientations. Future improvements are needed to better handle cube orientation and to ensure consistent performance in different holding positions.
- **Cube Size**: The system was designed and tested for standard 3x3 Rubik's Cubes.
- **Environmental Variability**: The system showed satisfactory performance in controlled environments with good lighting. However, as mentioned earlier, performance in not optimal light conditions decreased, and further enhancements in lighting adaptability are recommended.

5.3.5 Conclusion of the Results Analysis

Overall, the Cube Master solver demonstrated strong performance, with high accuracy in state recognition and an efficient solver algorithm that can complete the puzzle in a reasonable amount of time. Also, with the implementation of the kociemba's algorithm the efficiency of the solving algorithm saw a drastic improvement. While the system excels under controlled conditions, there is room for improvement in handling low-light environments, larger cubes, and optimizing the solver algorithm for fewer moves. These enhancements will further increase the system's effectiveness and make it more versatile for a wider range of users and scenarios.

Chapter 6: Conclusion And Future Recommendations

6.1 Conclusion

In conclusion, this Rubik's Cube solving system created in the project uses computer vision to automatically solve a standard 3x3 cube. It combines an image detection model with a strong solving algorithm to analyze the cube's current state and figure out the best moves to solve it. The solving algorithm works well in different situations and is very quick. The vision part takes pictures of the cube and reads the colors on each face, turning them into a clear layout. The system can recognize the cube's pattern with accuracy. The solving part is very reliable, always finding the right solution even if the lighting is poor or there are distractions around. In total, this project offers a dependable, automatic Rubik's Cube solver that works in real time with little need for user input. It's helpful for both people who are just starting to learn how to solve a cube and for those who want to practice more efficiently.

6.2 Future Recommendation

While this Rubik's Cube solver system performs efficiently in its current form, several improvements and future enhancements can be implemented to expand its functionality and accuracy. The following are some key recommendations for future developments:

1.Support for Multiple Cube Sizes:

Expanding the solver to handle cubes of different sizes, such as 4x4x4 or 5x5x5, would greatly increase the versatility of the system. Developing algorithms that can efficiently solve these larger cubes would open the solver to a broader audience of Rubik's Cube enthusiasts.

2.Incorporation of Augmented Reality (AR):

Integrating Augmented Reality (AR) into the system could allow users to interact with the Rubik's Cube in a more immersive manner. For instance, AR could be used to visually display the moves required to solve the cube directly onto the physical cube, offering real-time guidance and feedback.

3.Support for More Lighting Conditions:

While the system performs well under good lighting conditions, its accuracy could be further improved in low-light environments. Enhancing the robustness of the image

recognition model to handle a wider range of lighting conditions would ensure consistent performance in varying settings.

4.User Interface Enhancement:

Improving the user interface (UI) could make the system more accessible and user-friendly, especially for beginners. A more intuitive UI could allow users to easily capture the cube's state and receive step-by-step instructions to solve it.

5.Integration with Educational Platforms:

Collaborating with educational platforms or offering the Rubik's Cube solver as a learning tool could help students and enthusiasts understand the mechanics of solving a Rubik's Cube. Integrating tutorials and interactive guides that explain the algorithms behind the solver would make the system an excellent resource for learning.

6. Optimization of the Layer-by-Layer (LBL) Algorithm:

Although the current solver implements the Layer-by-Layer (LBL) method effectively, there exist more efficient algorithms for solving individual layers. By incorporating and optimizing these advanced algorithms, the solver could reduce the total number of moves required, making the solution faster and more efficient. This enhancement would not only improve performance but also align the solver closer to competitive speedcubing standards.

References

[1] Rubik's, "History of the Rubik's Cube," *Rubik's*, 2024. [Online]. Available: https://www.rubiks.com/history. [Accessed: Oct. 16, 2024].

[2] R Hod. Finding the total number of legal permutations of the Rubik's Cubic. Trondheim: Trondheim Katedralskole, 2010. [Online]. Available: https://link.springer.com/article/10.1186/s10033-018-0269-7 [Accessed:, : Oct. 16, 2024].

[3] M. Cube, "Rubik's Cube," Academia.edu, [Online]. Available: https://www.academia.edu [Accessed: : Oct. 16, 2024].

[4] Herbert Kociemba's Optimal Cube Solver - Cube Explorer [Online]. Available: https://ruwix.com/the-rubiks-cube/herbert-kociemba-optimal-cube-solver-cube-explorer/[Accessed: Jan. 14, 2025].

[5] C. Müller and K.-L. Ma, "Visual Analysis of Rubik's Cube Solving Strategies," in EuroVA: International Workshop on Visual Analytics, 2019. [Online]. Available: http://data.caleydo.org/papers/2019_eurova_rubik.pdf [Accessed: Oct. 16, 2024].

[6]" Ernő Rubik" [Online]. Available: https://en.wikipedia.org/wiki/Ern%C5%91_Rubik [Accessed: Oct. 26, 2024].

[7] "How to Solve the Rubik's Cube: Beginner's Method," Ruwix, [Online]. Available: https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/ [Accessed: Oct. 26, 2024].

[8] "kociemba 1.2.1": Python/C implementation of Herbert Kociemba's Two-Phase algorithm for solving Rubik's Cube [Online]. Available:https://pypi.org/project/kociemba/ [Accessed: Jan. 14, 2025].

[9] "50 Years of Rubik's Cube," *Rubik's*, 2024. [Online]. Available: https://www.rubiks.com/history. [Accessed: Jul. 30, 2025]. Wikipedia+15rubiks.com+15Worldcrunch+15

[10] A. Beard, "Life's Work: An Interview with Ernő Rubik," *Harvard Business Review*, Nov.–Dec. 2020. [Online]. Available: HBR article. [Accessed: Jul. 29, 2025].

Appendices

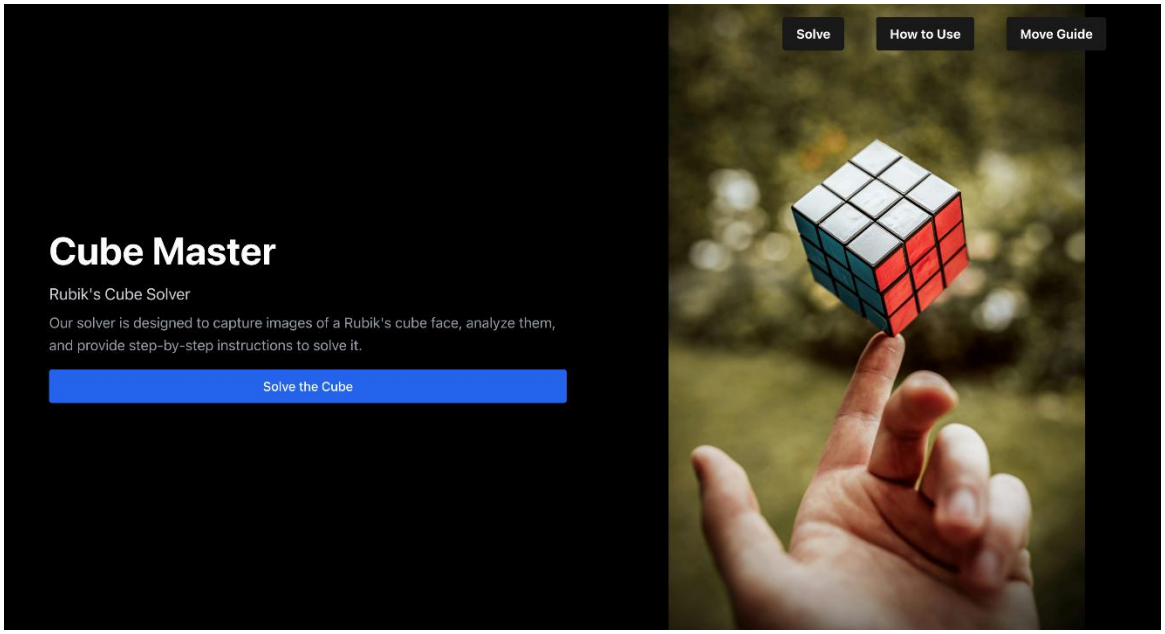

Figure: Home Page

Figure: How to use Cube Master Section

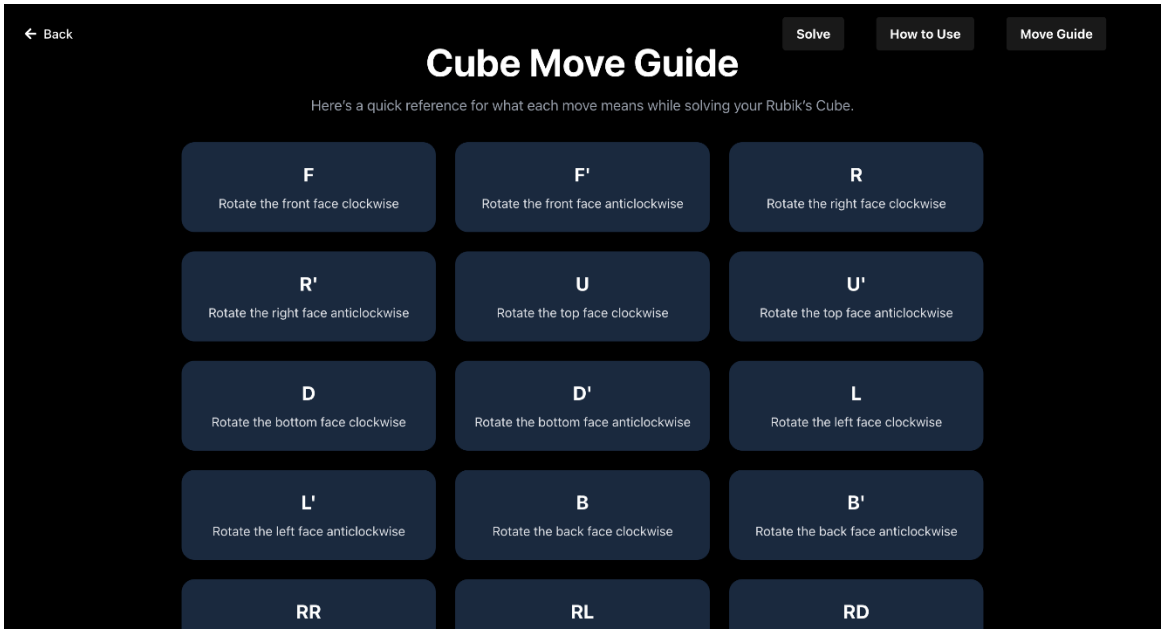

Figure: Move Guide to use Cube Master

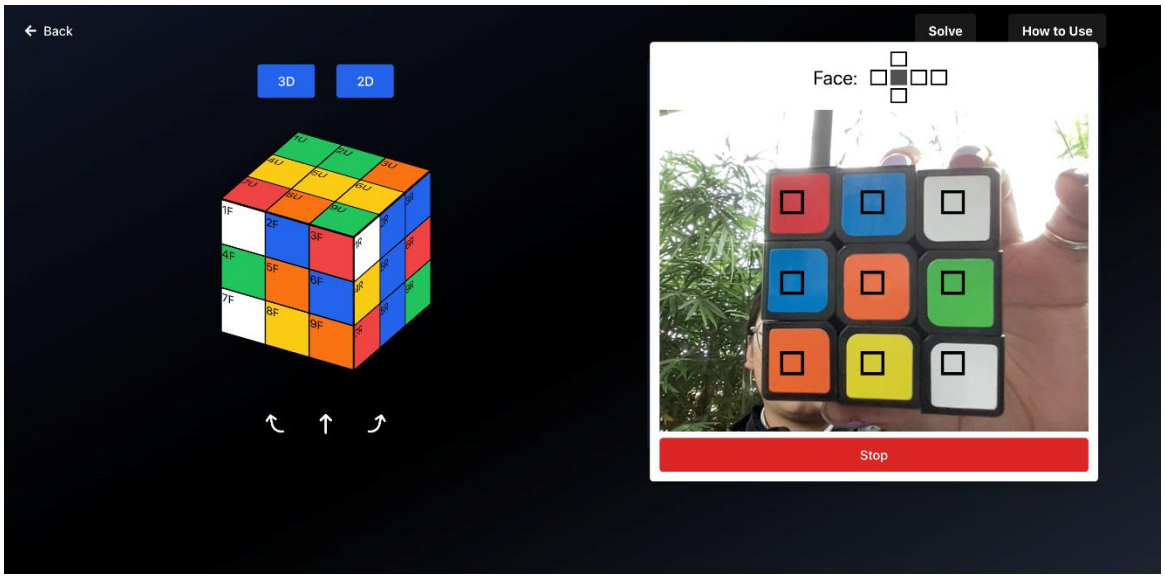

Figure: Scanning the Cube

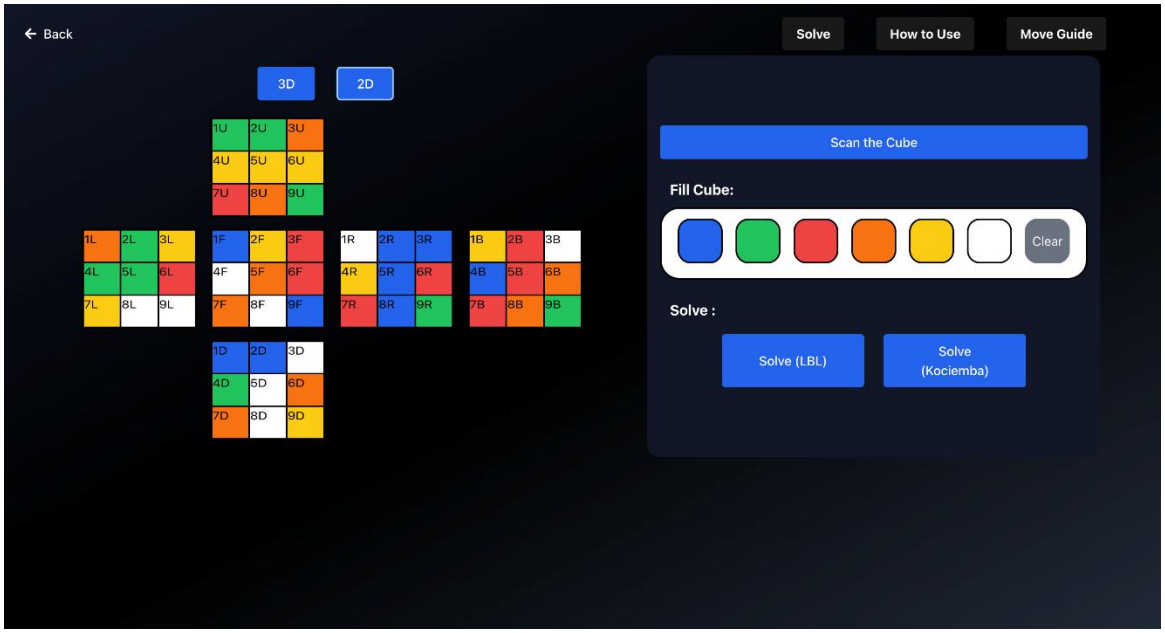

Figure: 2D Representation of Scanned Cube

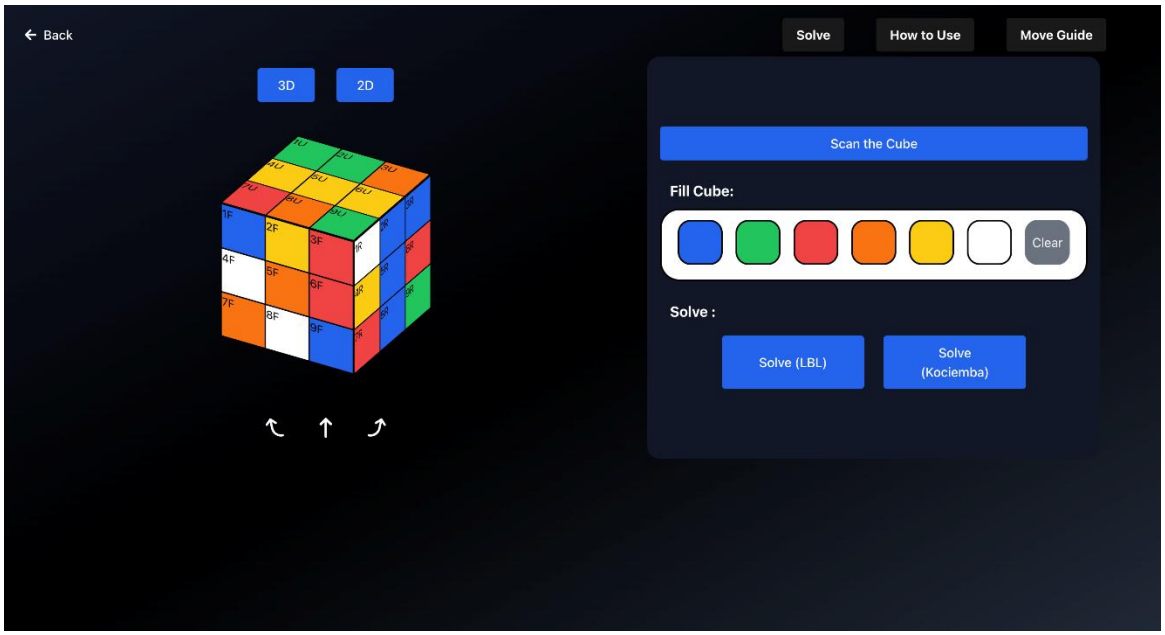

Figure: 3D Representation of Scanned Cube

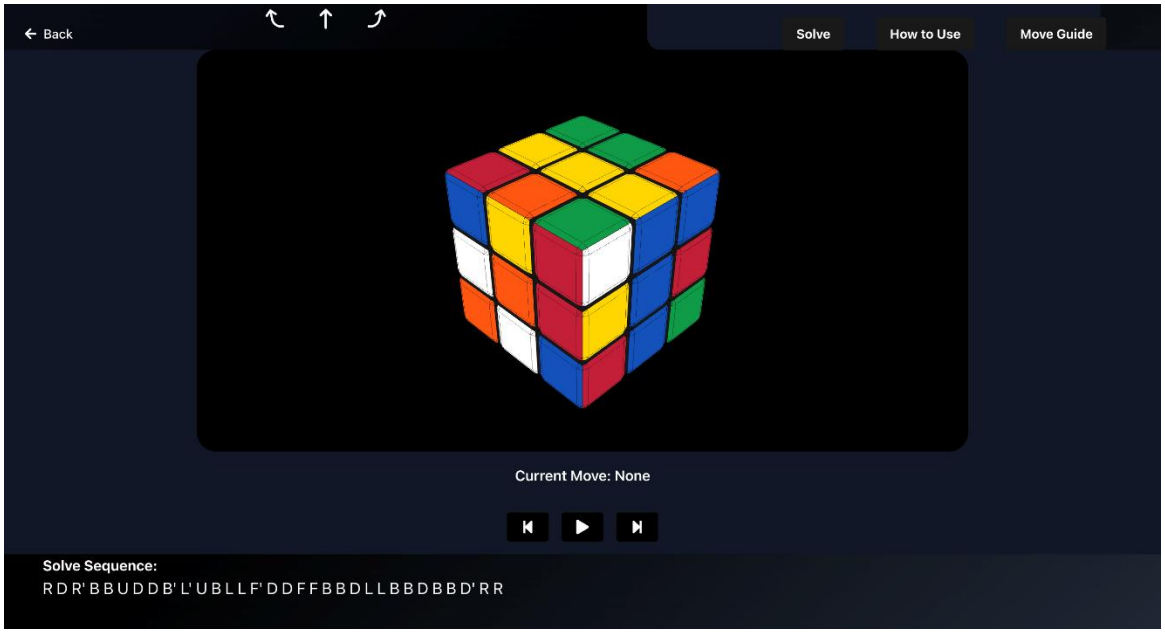

Figure: Solved State using Kociemba Algorithm

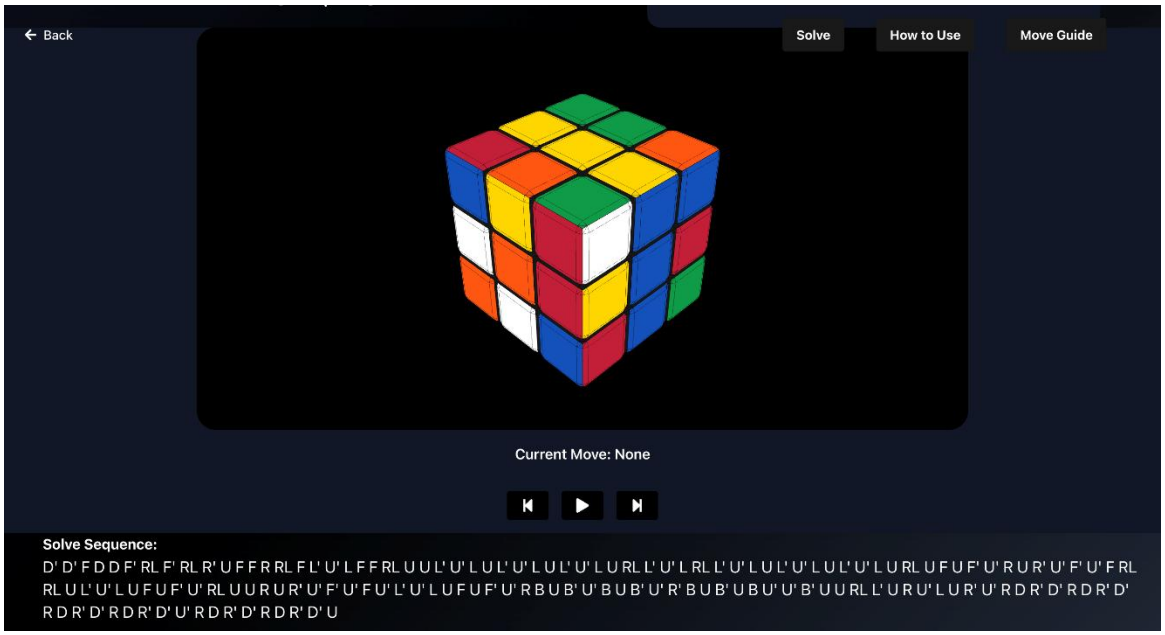

Figure: Solved Sequence using Layer by Layer Algorithm

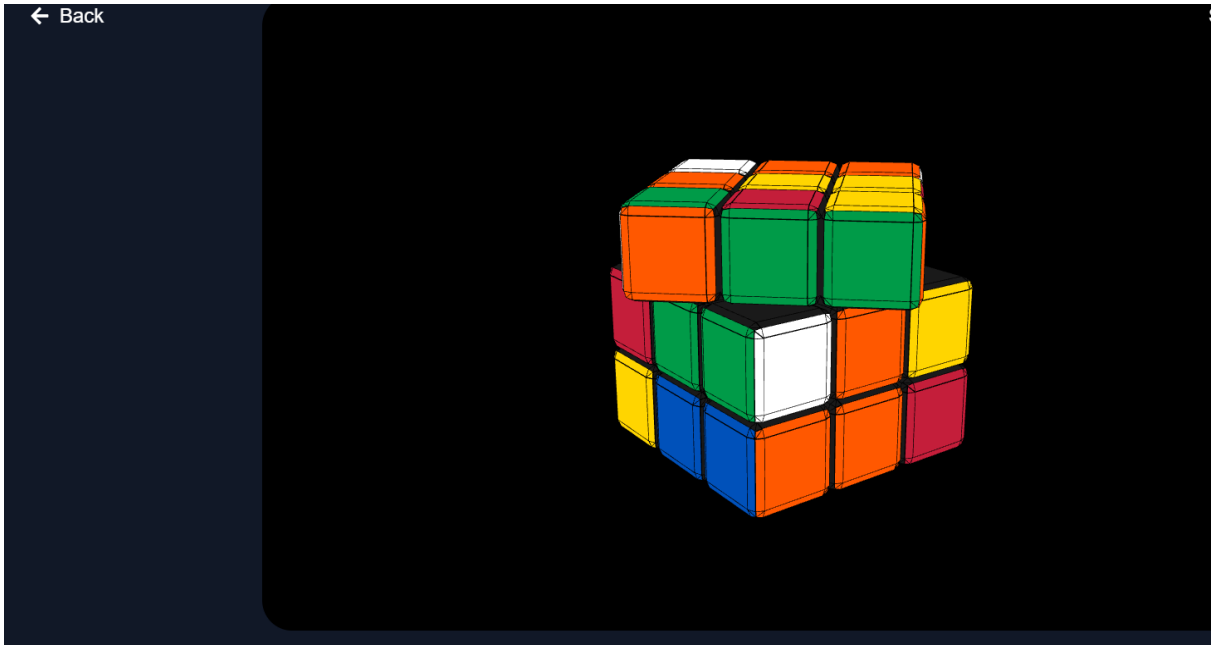

Figure: 3D Representation of Solving Cube

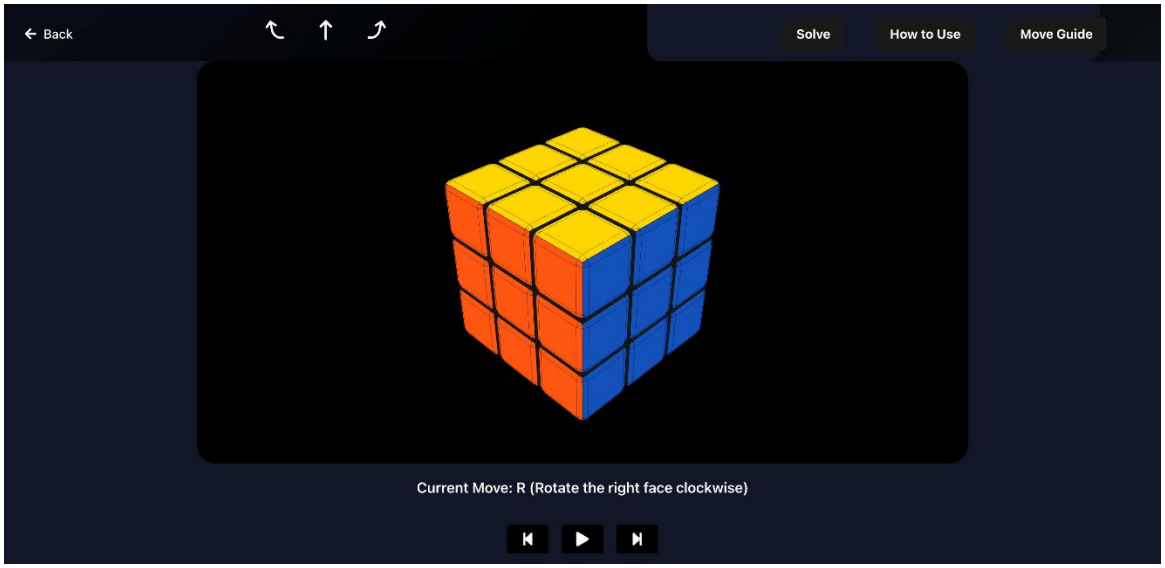

Figure: 3D Representation of Solved State