IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# A NOVEL AND IMPROVED SCHEME FOR SOLVING 3x3 RUBIK'S CUBE

**[1]Rakesh V S, [2]Mukul Singh, [3]Imtiyaz Ahmed, [4]K K Snehith Reddy, [5]Manit Srivastava**

[1]Assistant Professor, [2,3,4,5] UG Student, [1,2,3,4,5] Computer Science and Engineering,

[1,2,3,4,5] Cambridge Institute of Technology, Bengaluru, India

***Abstract:*** This project aims to create a sophisticated Rubik's Cube solver by blending human-designed algorithms with computer-based methods. By understanding the cube's structure and using efficient algorithms like Thistlethwaite, the solver analyzes the cube in real-time. It employs machine learning and deep learning techniques to enhance its efficiency. The software generates user-friendly 3D models and visualizations to help users understand the process better. Extensive testing ensures accuracy, with future plans to integrate physical devices for an even better user experience, potentially revolutionizing Rubik's Cube solving.

***Index Terms*** **– Rubik's Cube, Thistlethwaite, CFOP Algorithm, CUDA Architecture, OpenCV**

## I. INTRODUCTION

With its distinctive qualities, the well-loved Rubik's Cube has enthralled fans and drawn academic attention. Diverse studies have resulted from researchers' exploration of its possibilities in brain training and technological development. The creative methodology used in this project is consistent with the cube's use in mechanisms, suggesting potential future integrations with physical manipulators [1]. This project aims to revolutionize Rubik's Cube solving by blending advanced technologies, computer methods, and human algorithms seamlessly. By delving into the cube's complexity and utilizing the Thistlethwaite algorithm with CUDA Architecture, the solver achieves greater speed and adaptability. Machine learning and deep learning enhance its capabilities through real-time image processing and computer vision. The accompanying software offers user-friendly interactive visualizations and 3D models for all skill levels. Rigorous testing ensures efficiency and accuracy are maintained throughout development.

## II. BACKGROUND STUDY

Combining the groundbreaking contributions of C++, Python OpenCV, and the Thistlethwaite & CFOP algorithms, our software endeavors to revolutionize Rubik's Cube solving. Originating from Bjarne Stroustrup's visionary pursuit in the 1980s, C++ has evolved into a versatile programming language, offering the backbone for organized and modular code. Python OpenCV, born in the early 2000s, has democratized computer vision, empowering developers with its accessibility and extensive toolkit. Meanwhile, the Thistlethwaite algorithm and CFOP method [2] have pioneered efficient cube-solving strategies, leveraging group theory and computational techniques. By integrating these technologies, our software aims to provide an intuitive and efficient tool for solving Rubik's Cube puzzles, facilitating real-time cube recognition, state analysis, and optimal solutions.

## III. LITERATURE REVIEW

Various studies have explored color detection and Rubik's Cube solving techniques, employing diverse methodologies and technologies. These efforts include an Autonomous Rubik's Cube solver integrating color recognition with the Layer by Layer (LBL) algorithm and leveraging OpenCV for image evaluation [3], [4], as well as comparisons between the efficiency of the IDA* algorithm and Thistlethwaite's algorithm [5]. Additionally, research addresses challenges in color detection systems using OpenCV and Python [6], [7], while other studies focus on real-time color detection with applications in robotics and transportation [8], [9]. These collective endeavors significantly advance techniques in color detection and Rubik's Cube solving, enriching interdisciplinary understanding and propelling advancements in computer vision and algorithmic techniques. They demonstrate innovative approaches to tackling complex problems, emphasize efficiency and accuracy in contemporary applications, and push the boundaries of achievement in computer science and engineering.

## IV. RESEARCH METHODOLOGY

### 4.1 System Architecture

The proposed Rubik's Cube solving system boasts the capability to tackle any conceivable scramble of a 3x3 cube, employing standard Rubik's cube notation. Practical solutions, executable on an actual cube, are generated and accompanied by a 3D rendering, effectively simulating the solving process. At the core of the architecture lie the CFOP and Thistlethwaite algorithms, synergistically driving the system's proficiency. Fig 4.1 describes the key components of this architecture.

1. User Interface: Serving as the user's gateway, this interface facilitates efficient engagement with the software. Users can configure algorithms, capture cube states via webcam, and witness solving animations firsthand.
2. Event Driver: Bridging the User Interface and Algorithm Driver, this component ensures smooth information exchange for effective solving.
3. Algorithm Driver: Anchoring the workflow, this driver applies designated algorithms to inputs, yielding diverse solutions. It comprises three pivotal sub-states:
    a. Read Cube Data: Processing inputs to analyze the current puzzle state, laying the groundwork for subsequent processes.
    b. Apply Algorithm: Systematically executing designated algorithms to solve the cube, generating a spectrum of solutions.
    c. Optimize Generated Solution: Scrutinizing and refining all solutions to enhance efficiency and ensure optimal performance.
4. Solution Simulator: Bringing generated solutions to life, this phase employs dynamic animations to illustrate moves, engaging users visually.
    a. Convert Moves into 3D Rotations: Precisely translating optimized moves into accurate 3D rotations for virtual manipulation.
    b. Create Animation from Rotation: Converting rotation data into detailed animations, guiding users through solving steps effectively.

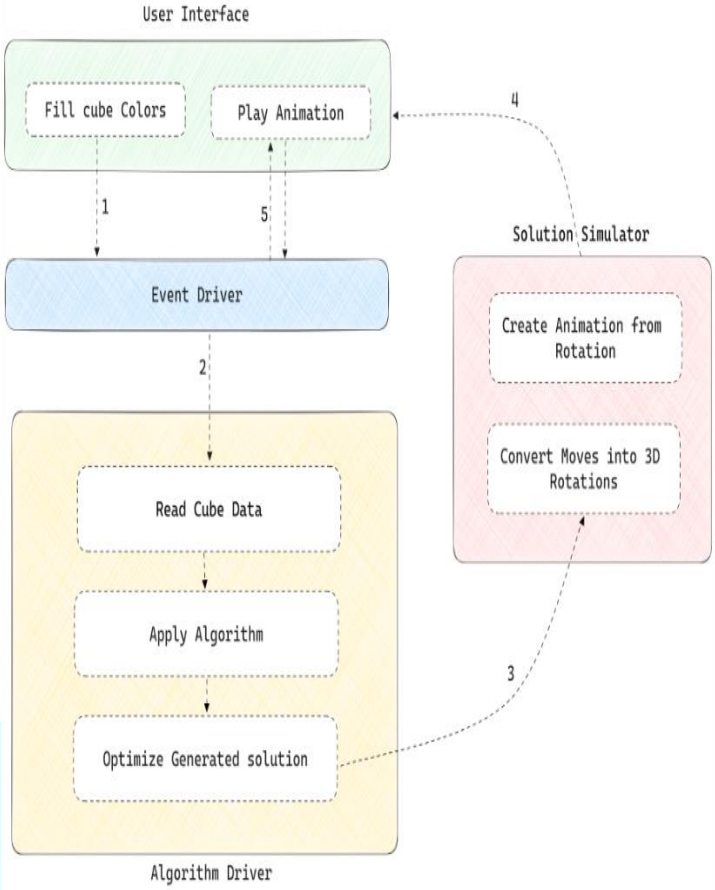


Figure 4.1: System Architecture Workflow

**4.2 Flow of the Algorithm**

The diagram below provides a comprehensive overview of the Rubik's Cube solving algorithm, incorporating various methodologies. It serves as a detailed guide, illustrating the sequential progression of the process from initiation to the eventual generation of solutions.

1. Map all the states of the pieces of Rubik's Cube: Numeric encoding condenses the Rubik's Cube state into six 1D arrays, each representing a side, and a 3-slice array capturing the cube's slices for streamlined computational processing.
2. Solve for Reverse Bottom Cross and move cross pieces into position: Establishing a sturdy base involves first solving the bottom cross, followed by creating a mirrored cross on the top and relocating the correctly positioned pieces to the bottom face for a reliable solution.
3. Perform F2L to solve all corner-edge pairs: Efficiently solve the first two layers (F2L) by targeting and orienting corners, pairing them with suitable edges using predefined algorithms. This completes the step, successfully resolving the two bottom slices.
4. Detect OLL cases and check to implement PLL: Identify potential OLL cases and create the top cross, skipping it when necessary. OLL ensures final layer orientation, preparing for the ultimate PLL algorithm in Rubik's Cube solving.
5. Perform OLL Algorithm and PLL Algorithm: PLL (Permute Last Layer) is the concluding step in solving the Rubik's Cube, utilizing a specific algorithm to complete the puzzle from its random state.

The solution is given in Rubik's Cube notations, with S representing clockwise, S' for anticlockwise, and S2 for a double rotation. These notations cover the rotations of right (R), left (L), top (T), down (D), front (F), and back (B) faces, providing a standardized method for executing the solution steps.

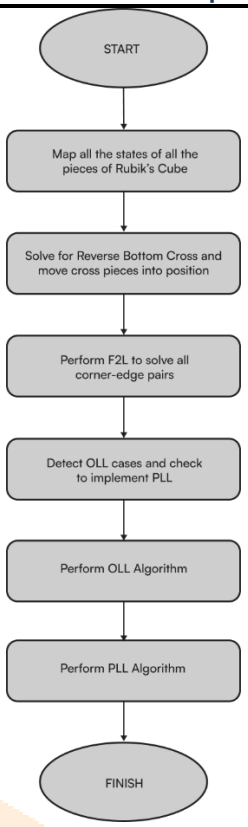


Figure 4.2: Algorithm Flowchart

**4.3 Proposed Algorithm**

The following pseudo-code algorithm outlines the software's comprehensive functionality and offers an in-depth depiction of its entire code structure.

```
for each side of cube
        build a reverse cross on the top face with misaligned centres
        align centres and bring cross pieces down.

        for each top edge corner
                check corner-edge orientation
                reduce corner-edge orientation to a reduced form
                pair edge corner for f2l
                apply set moves to perform f2l on the paired edge-corner
        build upper cross
        check upper corners orientation
       reduce upper corner orientation
        apply oll algorithm
        apply corner pll algorithm
        apply edge pll algorithm
        count the no. of moves
final moves are the moves with minimum length
reduce final moves to trim and merge moves
```

## V. RESULTS AND OUTPUT

### 5.1 Software Output

In Figure 5.1.1, the initial render of the cube is depicted, offering a starting point for users. Figure 5.1.2 showcases the innovative "Inside Out View," providing a perspective of the cube's backsides from within. Users can engage in the "Process of Filling Colors" demonstrated in Figure 5.1.3, where all colors are removed, allowing manual selection and application using a mouse, ideal for replicating physical cube configurations. Lastly, Figure 5.1.4 illustrates the "Cube Solving Phase," offering a midway point in the solving process, with the software generating solutions step by step upon user prompt, the speed of which is adjustable. This comprehensive software suite caters to Rubik's Cube enthusiasts, offering visualization, customization, and solving capabilities.

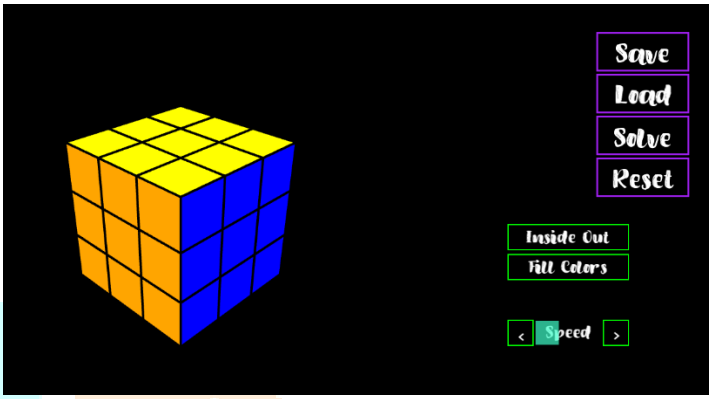


Figure 5.1.1: Initial Render and Output

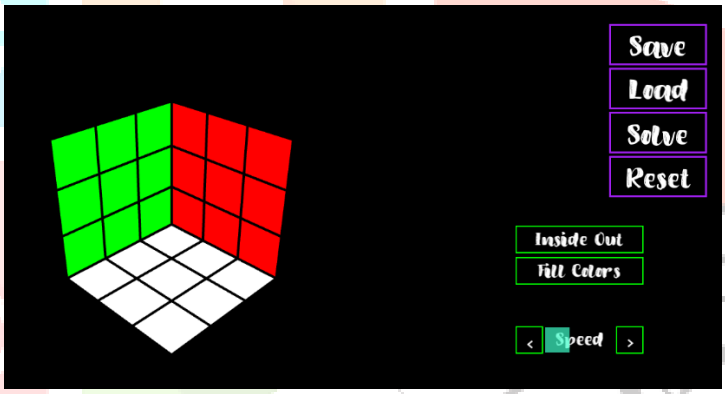


Figure 5.1.2: Inside out view of the cube

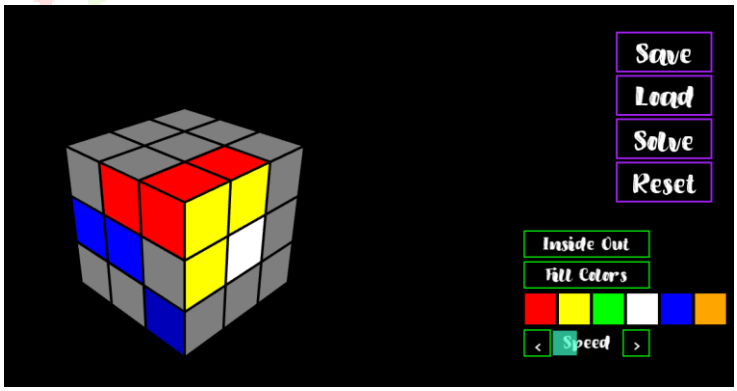


Figure 5.1.3: Process of filling colors

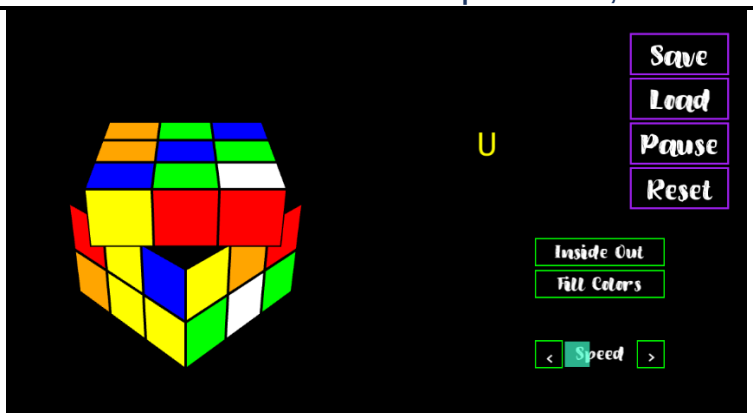


Figure 5.1.4: Cube solving phase

### 5.2 Data Statistics

Table 5.2.1: Descriptive Statistics with best side moves optimization enabled

| Time (Milliseconds) | Number of Moves |
|---|---|
| 36 | 89 |
| 49 | 77 |
| 56 | 74 |
| 73 | 91 |
| 48 | 78 |

Table 5.2.2: Descriptive Statistics with best side moves optimization disabled

| Time (Milliseconds) | Number of Moves |
|---|---|
| 10 | 108 |
| 8 | 105 |
| 6 | 94 |
| 4 | 103 |
| 12 | 91 |

Based on the observed results data displayed in Table 5.2.1 and Table 5.2.2, the average solving time achieved by our Rubik's Cube solver is 8 milliseconds. Additionally, the solver exhibits an average of 100.2 moves required to solve the Rubik's Cube.

### Conclusion

In summary, our analysis reveals a notable trade-off between the speed and number of moves in our Rubik's Cube solver algorithm. While the algorithm demonstrates impressive speed, it encounters a disadvantage by doubling the maximum potential required moves in the fastest solves. To address this challenge, three key approaches have been outlined for improvement: integrating additional Thistlethwaite states, designing a more efficient state reducer using machine learning, and refining move trimming through alternative rotation perspectives. Looking ahead, this project presents a promising frontier for advancing Rubik's Cube solving and computational problem-solving domains. By prioritizing user-friendly algorithms and leveraging advanced computational techniques, the project holds significant potential for future development. Anticipated advancements, including the integration of OpenCV for color detection, utilization of Nvidia CUDA architecture, and harnessing AI through native Neural Networks training, underscore the project's commitment to innovation and its potential to revolutionize not only Rubik’s Cube solving but also broader computational problem-solving endeavors.

## REFERENCES

**[1]** Rohith, S. P., Mohamed Sharif, A., Jayasankar, S., & Harikrishnan, M. (2019). "Autonomous Rubik's Cube Solver Bot." International Journal of Scientific Research and Engineering Development, Volume 2, Issue 3, May-June 2019

**[2]** Kaur, H.H. (2015). "Comparison of IDA* Algorithm and Thistlethwaite's Algorithm for Solving the Rubik's Cube." Degree Project in Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden.

**[3]** Gupta, S., Rathee, A., Kathariya, A., Ashish, & Channi, H. K. (2017). Modeling and Designing of Color Detector using Arduino. International Journal of Scientific Research in Computer Information Science, Engineering, 3(7), 2164-2171. DOI: 10.13140/ RG.2.2.26143.87203.

**[4]** Kaushal, M., & Singh, B. (2022). Real-Time Color Detection Using Python and OpenCV. International Research Journal of Engineering and Technology (IRJET), 09(05), 1399-1400.

**[5]** Farooqi, H.J., Chauhan, A.A., Siddiqui, A.W. (2020). Naïve Bayes Approached in Color Detection using Pandas & OpenCV. International Journal of Scientific & Engineering Research, 11(12), 288-292. ISSN 2229-5518.

**[6]** Deborah T. Joy, Gitika Kaur, Aarti Chugh, and Shalini B. Bajaj. "Computer Vision for Color Detection." International Journal of Innovative Research in Computer Science & Technology (IJIRCST), Volume 9, Issue 3, May 2021, Pages 53-59.