Rubik's Cube Solver using Real-World Logic

A Submission for AeroHack Design Challenge

Rubik's Cube Solver
Solving a scrambled 3x3 cube using camera input and valid move logic
Submitted by: PUSHADAPU MOHANA VENKATA SIVA NAGA SAI
Track: Design Challenge (Computer Science)

Overview:
Design and implement an algorithm that can solve a standard 3x3 Rubik's Cube from any scrambled state using valid real-world moves — no Kociemba library allowed.

Challenge Areas:

- Model cube state and track transformations
- Predict states after each move
- Efficiently reach solved state with minimal steps

How we approached it:

- **Step 1:** Input cube data via webcam using OpenCV
- **Step 2:** Extract each face's color configuration
- **Step 3:** Map colors to cube faces (U, D, F, B, L, R)
- **Step 4:** Validate color counts and correctness
- **Step 5:** Solve the white cross (first step of any Rubik's Cube method)
- **Step 6:** Output valid move steps (U, F', R, etc.) to guide solving

---

Face Notation:

- U (Up), D (Down), F (Front), B (Back), L (Left), R (Right)

Internal Data Structure:

- Each face as a 3x3 2D list:
  `self.cube['F'] = [['R', 'G', 'B'], ..., ...]`

Center Piece Reference:

- Used to determine face identity and guide correct alignment
  (e.g., `F[1][1] = Green` ⇒ Front face = Green)

HSV-based detection:

- Each face scanned using webcam in real-time

- Divided into 3x3 grid, extracting average HSV for each square
- Mapped to closest known color using defined hue/saturation/value thresholds

Correction step:

- After each face is scanned, user is asked to confirm or re-scan
- Prevents misclassification (e.g., red ↔ orange)

Why white cross first?

- White cross is the foundation of standard cube-solving methods (CFOP, beginner method, etc.)

Steps:

- Locate all white edge pieces
- Rotate and align them with corresponding center colors
- Bring them to correct position on Up face using valid moves (U, R, F', etc.)

Output:

- Step-by-step moves to solve the cross, shown as notation sequence

---

Color Confusion

- Red vs Orange often confused under different lighting

Center-Piece Reference

- Required consistent mapping from scanned colors to standard cube faces

No Kociemba Shortcut

- Had to implement step-by-step solving logic manually

Visualization

- Visual UI (e.g., Blender animation) planned but not fully completed

---

```
camera.py              → Color detection + OpenCV scanning
rubik_cube_with_kociemba.py          → (Optional) For future comparison or validation
rubik_cube_without_kociemba.py  → Our actual solving engine (white cross only)
rubik_cube_solver.ipynb           → Combined demo and step-by-step UI
```

You can view the complete code and resources at:

GitHub Repository:
https://github.com/MOHANSAI2810/Rubix-s-Cube-Solver

Summary:

- Successfully implemented a real-world Rubik's Cube solving engine from camera to move list
- Avoided external shortcut solvers to demonstrate true algorithmic thinking

Learnings:

- HSV color space mastery
- Face rotation logic & modeling
- Edge detection and solving heuristics
- Data structure optimization
- OpenCV integration for real-time input