

Phase 1 — Foundation (Understanding & Setup)

Timeline: 2–4 days

Goal: Set up environment + understand cube solving representation

Tasks

1. Read Rubik's Cube notation

- Understand U, D, L, R, F, B
- Clockwise (CW), anti-clockwise (CCW), double turn (2)

2. Study Kociemba algorithm

- How cube string is passed
- Output format

3. Install tools

- Python 3.10
- OpenCV
- numpy
- kociemba
- matplotlib
- imutils

4. Setup Git Repo

- `cube-ai-ar-solver/`
- README.md
- tasks.md

Deliverables

- Clean project folder

- Git initialized
 - Working environment
-

Phase 2 — Basic Solver Without Camera

Timeline: 1–3 days

Goal: Pass mock cube → get solution

Tasks

1. Write a Python script:
 - Input: manually typed cube state
 - Output: solver steps
2. Validate solver:
 - Give random cube state strings
 - Print human-readable solution
3. Build function:

```
def solve_cube(state_string): return steps
```

Deliverable

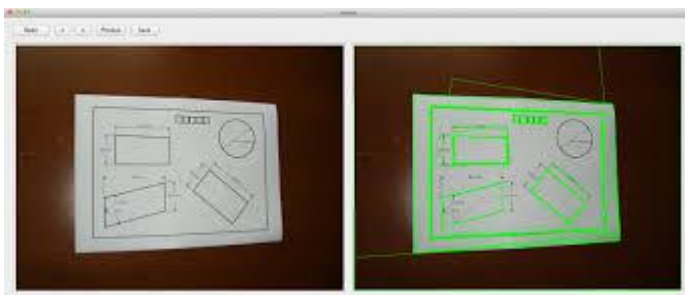
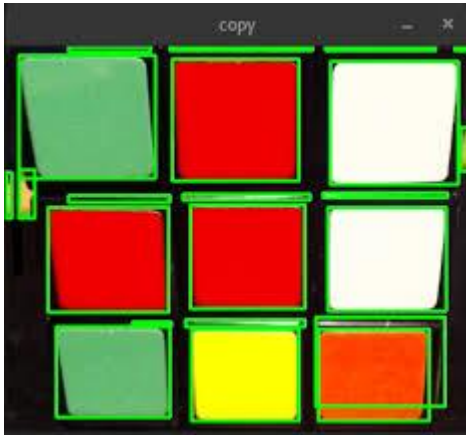
- `solver.py` that returns solution steps from a valid cube state

 This proves solver works before doing vision.

Phase 3 — Camera Input + Cube Face Scanning

Timeline: 5–10 days

Goal: Detect cube face + detect stickers via CV



4

Tasks

1. Webcam access

`cv2.VideoCapture(0)`

2. Find cube face:
 - Detect largest square region
 - Detect 3×3 mini squares
3. For each grid cell:
 - Get HSV value
 - Classify color

Output format:

`['G', 'G', 'G', ...]` → 9 tiles

4. Repeat 6 times:
 - U
 - R
 - F
 - D
 - L
 - B

Let user rotate cube → press key → capture

Deliverables

- `scanner.py`
- Stores all faces as 6×9 grid
- Saves face states as JSON

Phase 4 — Build Real Cube State + Validation

Timeline: 3–5 days

Goal: Convert scanned faces → solver input

Tasks

1. Fix scanning order
2. Convert faces → Kociemba string
3. Validate orientation
4. Add UI warnings
5. Recollect if invalid

Deliverable

- `state_builder.py`
 - Outputs a single 54-char cube string
-

Phase 5 — Show Steps to User (Desktop)

Timeline: 2–5 days

Goal: Present solution clearly

Tasks

- Display solution as:

Step 1: R

Step 2: U

Step 3: R'

...

- Add colors for split:

- CW = green arrow
- CCW = red arrow
- Double turn = yellow

Deliverables

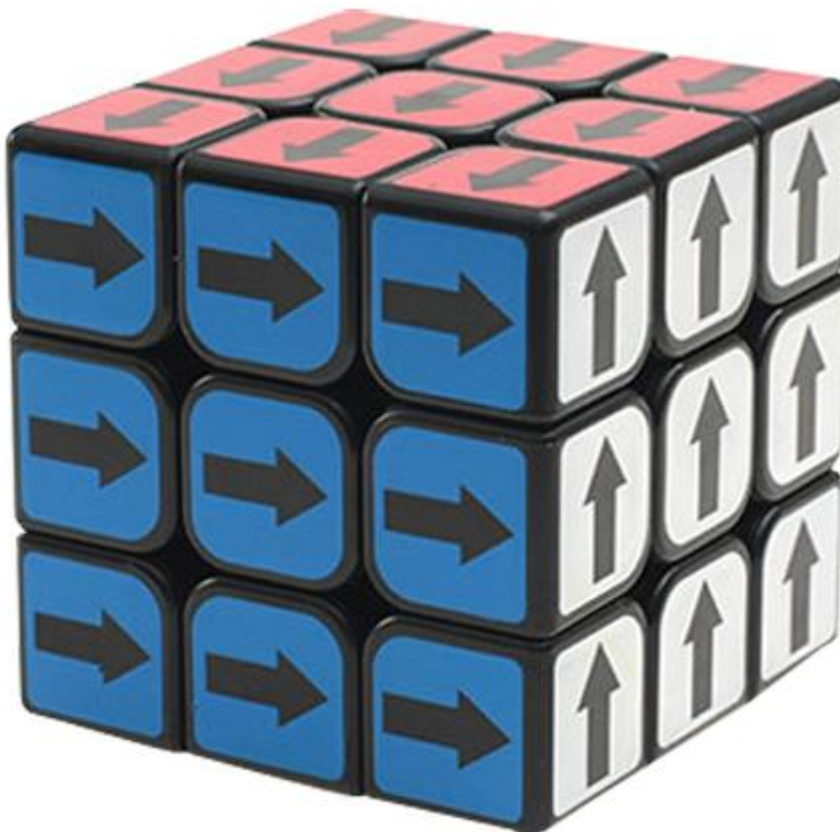
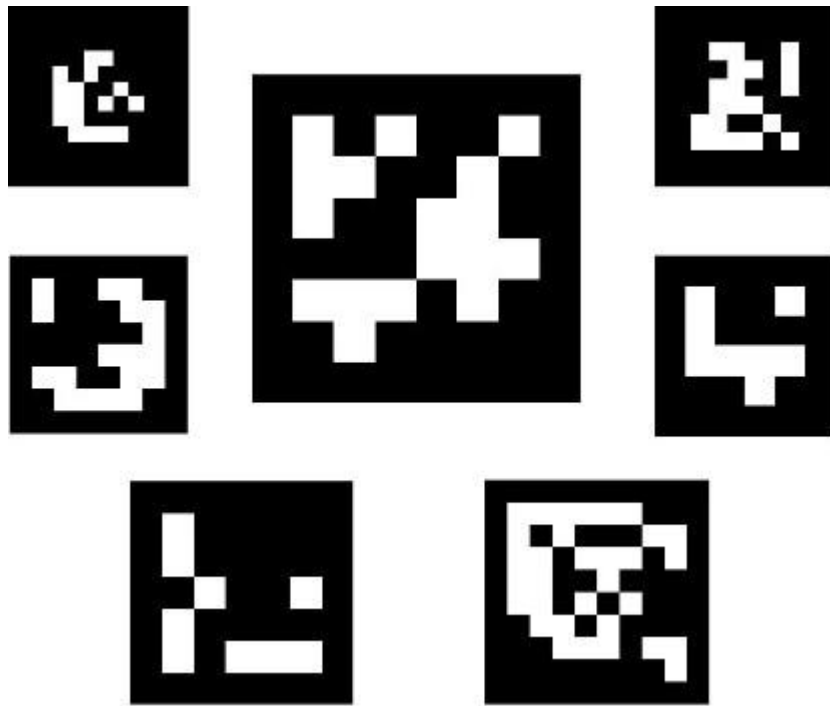
- `instructions.py`
- Displays steps cleanly

Phase 6 — AR Overlay (Desktop Prototype)

Timeline: 5–10 days

Goal: Visual arrows on the cube





Option A — ArUco Marker (Recommended)

1. Stick marker on cube
2. Detect marker → camera pose
3. Draw 3D arrow on face

Option B — Markerless

- Harder
 - You detect cube geometry yourself
-

Visual approach:

- Movement “R” → highlight right side face
- Draw arrow clockwise

Deliverable

- Real-time camera feed
 - Overlay arrows for each step
-

Phase 7 — Full AR Workflow + User Experience

Timeline: 5–12 days

Goal: Complete usable version

Tasks

1. Show next move only
2. Wait until rotate manually
3. Detect rotation

4. Move to next step
5. Reset system after solve

Deliverable

- Smooth AR guidance
 - Near real-world usability
-

Phase 8 — Documentation & Portfolio

Timeline: 1–3 days

Goal: Make it resume-ready

Tasks

1. README.md
 - Problem
 - Tech stack
 - Architecture diagram
 - How to run
 - Demo GIF/video
 2. GitHub commits
 3. Sample results
 4. Setup instructions
-

Folder Structure (Final)

cube-ai-ar-solver/

|— data/

```
|— models/
|— notebooks/
|— src/
|   |— scanner.py
|   |— color_classifier.py
|   |— state_builder.py
|   |— solver.py
|   |— ar_overlay.py
|   |— utils.py
|— tests/
|— docs/
|— README.md
|— requirements.txt
```

Breakdown in Trackable Tasks (Kanban style)

To Do

- Learn cube notation
- Read kociemba docs
- Setup environment
- Implement CLI solver

In Progress

- Camera scanning
- Cube face segmentation
- Color detection tuning

Validate

- Cube state reconstruction
- Solver integration

AR


- Marker tracking → overlay
- UI for move instructions

Polishing

- Error handling
- Stability
- Documentation



Estimated Total Time

 If you are working alone:

- **Beginner:** 6–8 weeks
- **Intermediate:** 3–4 weeks
- **Expert:** 10–14 days