DUAL-ARM ROBOTIC RUBIK'S CUBE SOLVER: SYSTEM DESIGN AND SUBSYSTEM INTEGRATION

Keeron Huang
qixuan3@illinois.edu

Yiming Xu
yx42@illinois.edu

Rong Wang
rongw5@illinois.edu

Zhuoyang Shen
zshen26@illinois.edu

ABSTRACT

This report presents the design and current implementation of an autonomous dual-arm Rubik's cube solving robot developed as a four-person ECE 445 senior design project. The system decomposes the task into four functional modules linked by three narrow, formally specified interfaces: a visual perception module converts an overhead camera frame into a $6 \times 3 \times 3$ colour matrix; a simulation and control-signal module integrates the Kociemba two-phase solver with a MuJoCo dynamics model and emits a discrete primitive-packet stream; a symmetric dual-arm mechanical platform physically realises the layer rotations; and a custom servo-control PCB distributes power and translates primitive packets into PWM-driven motor commands. The computational and electromechanical components have been independently verified: a four-move warm-up sequence (`R U R' U'`) is executed end-to-end in simulation and expands into 36 primitive packets, the Kociemba solver passes its self-test on representative cube states, the vision pipeline localises and classifies all visible stickers under locked exposure, and the servo-control PCB provides regulated 5 V and 3.3 V rails with a 16-channel PWM front end. Subsystem-level requirements, verification procedures, and observed results are summarised in Section 7.

1 INTRODUCTION

A Rubik's cube is a compact but well-studied testbed for autonomous manipulation. The workspace is small, the goal state is discrete, and the solver search space is tightly bounded by group theory; yet the task forces a controller to coordinate two grippers, plan around mechanically unreachable layers, and maintain confidence that the cube state observed at scan time remains valid throughout execution. The project described in this report constructs a tabletop robot that takes a scrambled cube as input, scans its faces with an overhead camera, computes a solving sequence using the two-phase Kociemba algorithm [1], and physically executes the sequence with a symmetric dual-arm mechanism driven by a custom servo-control PCB.

1.1 HIGH-LEVEL REQUIREMENTS

The project is constrained by the following high-level functional and non-functional requirements, which together define what the system must deliver to be considered a complete senior-design prototype:

1. **R1 — Autonomy.** The system shall scan, plan, and execute a complete cube-solving sequence with no operator intervention beyond a single physical start signal.
2. **R2 — Generality.** The solver shall accept any legal $3 \times 3 \times 3$ cube state and emit a sequence of at most 25 face turns in Singmaster notation, consistent with the upper bound of the two-phase algorithm.
3. **R3 — Perception accuracy.** The vision pipeline shall correctly identify all 54 stickers under fixed overhead lighting with a target colour-classification accuracy of $\geq 95\%$.

4. **R4 — Mechanical reachability.** The dual-arm mechanism shall be able to apply each of the six face turns $\{U, D, L, R, F, B\}$ to the cube without re-grasping by an external operator, using a combination of single-gripper cube reorientations and dual-grip layer turns.
5. **R5 — Deterministic execution.** The control-signal layer shall expose a discrete, finite-vocabulary primitive packet stream to the embedded controller, with a packet schema that is invariant across simulation and hardware execution.
6. **R6 — Electrical isolation.** The servo-control PCB shall derive both the 5 V servo rail and the 3.3 V logic rail from a single protected DC input and shall isolate the servo current path from the logic ground reference.
7. **R7 — Operator safety.** The system shall start only on a deliberate physical input and shall expose a visible status indication of its current operational state.

1.2 System Concept

Figure 1 shows the four-module system block diagram. The edge-computing host runs the vision pipeline and the simulation-based controller; the embedded controller and PCB driver layer convert discrete primitive commands into motor signals; and the dual-arm mechanism applies those signals to the cube. The remainder of the report is organised as follows. Section 2 establishes the system architecture and the inter-module interfaces. Sections 3–6 detail the design of each subsystem in turn. Section 7 presents subsystem-level requirements and verification. Section 8 addresses ethical and safety considerations, and Section 9 summarises the contributions and outlines the remaining integration work. Author contributions are listed at the end of Section 9.

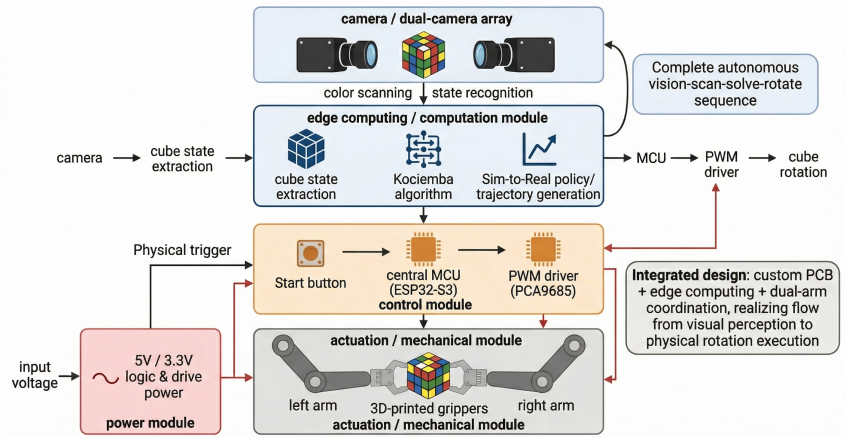


Figure 1: System-level block diagram. The edge-computing / computation module covers cube state extraction, the Kociemba solver, and sim-to-real trajectory generation in MuJoCo. The camera array, the central MCU with its PCB driver layer, and the mechanical actuation block close the loop from perception to motion.

2 System Architecture

2.1 Functional Decomposition

The system is partitioned into four functional modules and a clean operator-input layer. The decomposition follows the natural separation between perception, planning, execution, and actuation:

- **Vision Module** — camera capture, sticker localisation, colour classification, and assembly of the $6 \times 3 \times 3$ cube state.
- **Simulation & Control-Signal Module** — Kociemba two-phase solver, dual-arm controller running over a MuJoCo dynamics model, and emission of a discrete primitive-packet stream.

- **Servo-Control PCB Module** — regulated power distribution, I^2C-driven 16-channel PWM expansion, and connector layout for the actuators and the embedded controller.
- **Mechanical Module** — symmetric dual-arm CAD design, stepper-motor driving circuit, and the physical mounting of the camera and the cube cradle.

This separation is intentional: it isolates the heavy planning workload on the host while keeping the embedded side responsible only for the deterministic execution of a small, well-defined packet vocabulary. The inter-module interfaces are summarised below.

2.2 Inter-Module Interfaces

The four modules exchange data along three primary boundaries (Figure 2, Table 1). Each boundary is fixed to a specific format and direction so that the modules can be developed and verified independently.

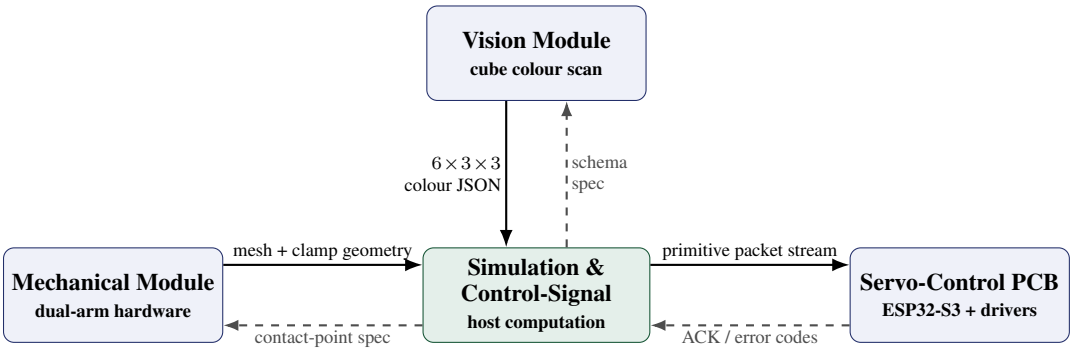


Figure 2: Inter-module interface diagram. Solid arrows indicate the primary data direction across each boundary; dashed arrows indicate the specification flowing in the opposite direction, so that each receiving module has a fixed target to design against.

Table 1: Inter-module interface specification. The three boundaries fully define the data crossing the system; all other state remains internal to the originating module.

Boundary	Specification crossing	Format / schema	Transport
Vision → Sim	$6 \times 3 \times 3$ cube colour matrix, face order `U,R,F,D,L,B`	JSON, `sample_state.json` schema	file / IPC
Sim → PCB	ordered stream of discrete primitive events; five-op vocabulary	JSON (development), packed binary (production option)	USB serial / UART
PCB → Mechanical	PWM drive signals for two wrist motors and two clamp motors	PCA9685 channel mapping; ctrlrange in Table 3	3-pin servo headers

The narrowest of these interfaces is the primitive packet stream: a single ordered list of discrete events and nothing else. Section 3 describes the rationale for this design choice and how the controller is implemented to keep the stream a stable contract while the underlying simulation continues to evolve.

2.3 Design Choices

Three architectural decisions structure the design. First, the simulation platform is MuJoCo rather than a general-purpose physics engine: MuJoCo provides actuator-level abstractions and equality constraints that match the mechanical reality of a geared dual-arm without requiring a high-fidelity contact solver. Second, the vision pipeline is decoupled into class-agnostic sticker localisation

and post-hoc colour classification rather than an end-to-end colour-class detector, which removes a known failure mode in which a single network overfits to specific lighting. Third, the embedded controller is positioned as a deterministic execution engine driven by a host-side planner rather than as an inference engine, so that the heavy computation (Kociemba search, vision inference, trajectory generation) is concentrated on a workstation while the ESP32 only enforces real-time motor scheduling and safety invariants.

3 Simulation and Control-Signal Module

The simulation and control-signal module is the bridge between the high-level cube-solving plan and the discrete events that the embedded controller will execute. Its design is organised around two stable contracts: a $6 \times 3 \times 3$ colour matrix on the input side and a primitive-packet stream on the output side. Everything between them is internal simulation state.

3.1 From CAD Assembly to MuJoCo Model

The starting point is `assemble/fullassemble.step`, the complete mechanical assembly. Because MuJoCo cannot build a controllable model directly from a STEP file, the first step is decomposition rather than control: the CAD assembly is broken into the sub-modules listed in Table 2, and each sub-module is exported as a mesh under `cube_mujoco/model/meshes/`. The original STEP file is preserved for downstream mechanical work.

Table 2: CAD decomposition into MuJoCo-compatible sub-modules. Only the four controllable units (two wrist motors, two clamp drives) cross the sim-to-real boundary; the remaining geometry is visual or kinematic support.

CAD module	Mesh file (under `cube_mujoco/model/meshes/`)	Role
base / frame	`base_vis.stl`	fixed global reference
left motor body	`left_motor_vis.stl`	visual + collision shell
right motor body	`right_motor_vis.stl`	visual + collision shell
left output shaft	`left_motor/motorgear_l.stl`	drives left wrist rotation
right output shaft	`right_motor/motorgear_r.stl`	drives right wrist rotation
left gripper assembly	`left_gripper/{gripper_base, centergear, left_claw, right_claw, cover}_l.stl`	clamps left side of cube
right gripper assembly	`right_gripper/{...}_r.stl`	clamps right side of cube

3.2 Kinematic Abstraction

The current MuJoCo model is `cube_mujoco/model/cube_dualarm.xml`. CAD meshes are registered in `<asset>` with `scale="0.001 0.001 0.001"`, converting CAD millimetres into MuJoCo metres. The assembled mechanism is constructed under `<worldbody>`, and a single `assembly_root` node carries a global `pos` and `euler` correction so that the imported model preserves its FreeCAD coordinate identity while sitting flat in the simulation world.

The mechanism is deliberately under-actuated. Each side uses a coaxial-shaft abstraction: the outer shaft controls gripper rotation; the inner shaft drives the centre gear, and a MuJoCo equality constraint couples the centre gear to the two claw slides (`polycoef="0 0.005 0 0 0"`). The controller therefore never writes to individual claw degrees of freedom — the simulation enforces the coupling automatically. After this abstraction, the entire dual-arm system is exposed as four actuator channels (Table 3), and these four channels are the only signals that eventually cross the wire to the PCB driver. All other simulation state (centre-gear position, individual claw slides, cube freejoint pose) remains internal.

Table 3: The four MuJoCo actuators that survive the kinematic abstraction. Values are taken from `cube_mujoco/model/cube_dualarm.xml` lines 303–308.

MuJoCo actuator	Joint type	`ctrlrange`	Op. values	Hardware meaning
`left_gripper_rotate_act`	position, $k_p = 20$	± 1.57	$\{\mp\pi/2,\ 0\}$	left wrist motor
`left_centergear_act`	general (375/−25)	$[0,\ 0.16]$	$\{0,\ 0.123\}$	left clamp drive
`right_gripper_rotate_act`	position, $k_p = 20$	± 1.57	$\{\mp\pi/2,\ 0\}$	right wrist motor
`right_centergear_act`	general (375/−25)	$[0,\ 0.16]$	$\{0,\ 0.123\}$	right clamp drive

3.3 Cube Model and Layer Rotation

Early visual assets such as `cube_mujoco/model/meshes/cube/textured.obj` represent the cube as a single rigid body. This is sufficient for rendering but cannot express independent layer turns. The controllable cube is therefore modelled as a logical $3 \times 3 \times 3$ grid of 27 cubies, each with its own body. The cube root body carries a `freejoint` so the whole cube can be reoriented during manipulation. Individual cubies are named `cubie_x{m,0,p}_y{m,0,p}_z{m,0,p}`, where `m`, `0`, and `p` encode the logical coordinate $\in \{-1, 0, +1\}$. Figure 3 illustrates the convention.

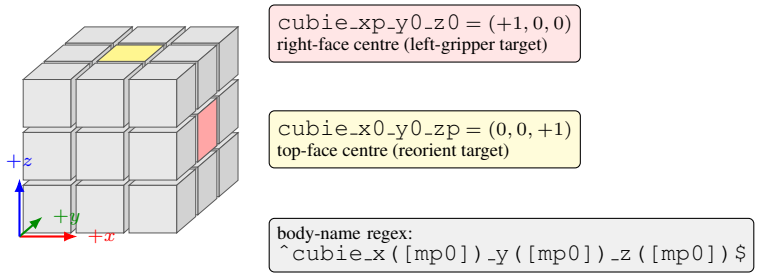


Figure 3: The 27-cubie logical cube under MuJoCo's $(+x, +y, +z)$ frame. The right-face centre (`cubie_xp_y0_z0`) is the natural target for the left gripper, and the top-face centre (`cubie_x0_y0_zp`) is what a reorientation move brings into a gripper slot. Controller initialisation requires all 27 cubies to be present, otherwise the run aborts.

The layer-rotation logic is implemented in `cube_mujoco/scripts/rubiks_cube_control.py`. The numerical constants that govern the motion are listed in Table 4. Initialisation scans the MuJoCo model with the regex shown in Figure 3, builds a body-id table and a logical grid, and aborts the run if any cubie is missing. A single layer turn selects the nine cubies whose grid coordinate matches the target axis and layer, rotates their positions with a rotation matrix and their orientations with a quaternion, calls `mj_forward` / `mj_step` to advance the simulation, and finally snaps the grid coordinates back to integers to suppress floating-point drift.

Table 4: Core constants in `rubiks_cube_control.py`. These five values are the only numerical knobs the controller exposes; all other quantities are derived.

Constant	Value	Meaning
`PITCH`	0.019 m	cubie grid spacing
`QUARTER_TURN`	$\pi/2$	one basic layer rotation
`CLAMP_CTRL`	0.122667	clamp-closed actuator setpoint
`RELEASE_CTRL`	0.0	clamp-open actuator setpoint
`NEUTRAL_GRIPPER_ROTATION`	0.0 rad	gripper home angle

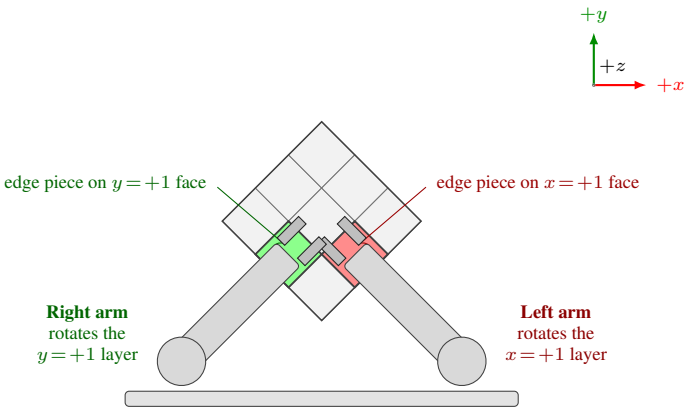
The same script parses Singmaster notation: `R`, `L`, `U`, `D`, `F`, `B` each map to an axis and layer, the suffix `'` reverses the direction, and the suffix `2` doubles it.

3.4 Cube–Gripper Spatial Relation

The controller does not assume that any logical layer is reachable from any orientation. The grippers are rigidly bolted to the frame, and only two outer layers can be clamped at any one time. This constraint is codified by two mappings, shown together with a front-view schematic in Figure 4:

```
GRIPPER_LAYER_SLOT          = {"left": ("x", +1), "right": ("y", +1)}
GRIPPER_ROTATION_AXIS_SLOT = {"left": ("x", -1), "right": ("y", -1)}
```

The left gripper engages the $x = +1$ outer layer; the right gripper engages the $y = +1$ outer layer; the rotation axis of each gripper points along the opposite face. When a logical move requires a layer that is not aligned with either slot, the controller first reorients the entire cube using a single-gripper pivot and then performs the layer turn. A matrix `presentation_to_cube` tracks the running transform between machine coordinates and logical cube coordinates and is updated after every whole-cube reorientation.



Each gripper clamps the middle *edge piece* of a bottom-facing cube edge, not a face centre. Rotating that gripper then rotates the entire 9-cubie outer layer behind it. The `presentation_to_cube` matrix tracks the running transform between this fixed machine frame and the logical cube frame (Singmaster $U/R/F/D/L/B$), so any solver-requested layer can be brought into one of the two reachable slots through a whole-cube reorient.

Figure 4: Cube–gripper spatial relation in the operating (diamond) orientation. The cube is held with one edge pointing down, cradled between two arms in an inverted-V configuration. Each gripper closes on the middle edge cubie of a reachable bottom edge (highlighted red and green); the gripper rotation then carries the entire 9-cubie layer attached to that edge, which is the physical mechanism behind one Singmaster layer turn.

3.5 Motion Logic and Per-Move Finite-State Machine

The `RubiksCubeController` class implements motion in terms of three action classes (Table 5) and one per-move finite-state machine (Figure 5). The three action classes correspond exactly to the primitive event types emitted at the sim-to-real boundary, so the in-simulation abstraction and the firmware contract remain one-to-one.

Table 5: Action classes implemented by the controller. "#prims" is the number of primitive events typically emitted by one invocation.

Action class	API entry	#prims	Behaviour
Clamp primitives	`clamp/release/dual_hold`	1	open / close one or both grippers
Whole-cube reorient	`reorient_cube_with_single_gripper`	7–9	pivot cube on one gripper to align the target layer
Dual-grip layer turn	`turn_bottom_layer_with_dual_grip`	6–8	both grippers hold; active side rotates target layer

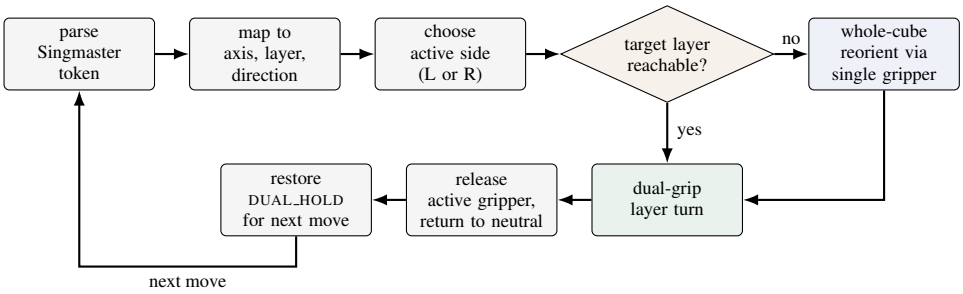


Figure 5: Per-move finite-state flow. The conditional reorient step is what allows a fixed-frame dual-arm to execute arbitrary Singmaster sequences: every unreachable layer is first brought into a reachable slot through a whole-cube reorientation.

3.6 PRIMITIVE-PACKET OUTPUT

The sim-to-real boundary is deliberately narrow: the simulation does not stream MuJoCo body poses, freejoint quaternions, or per-step interpolation. It emits a short, ordered stream of discrete primitive packets defined by the five-op vocabulary in Table 6 and the per-event schema in Table 7. This is the only contract the firmware is required to parse.

Table 6: The five primitive operations emitted by the controller. The firmware parser need only recognise these tokens.

`op`	Required fields	Semantics
CLAMP	`side`	close one gripper; mark `clamped[side]=true`
RELEASE	`side`	open one gripper; require `clamped[other]=true` first
DUAL_HOLD	–	close both grippers; safe rest configuration
ROTATE	`side`, `angle_rad`	drive wrist motor to $\{-\pi/2, 0, +\pi/2\}$
SETTLE	`steps`	dwell; no motor commands

Table 7: Per-event schema produced by `cube_mujoco/scripts/primitive_logger.py`. The schema is transport-agnostic: USB serial is used during development; CBOR or a packed binary frame is reserved as a future transport option.

Field	Type	Used by	Notes
`t`	float	host log only	MuJoCo simulation time; not a wall-clock schedule
`op`	enum	firmware	one of the five values in Table 6
`side`	enum	firmware	`"left"` or `"right"`; omitted for DUAL_HOLD/SETTLE
`angle_rad`	float	firmware (ROTATE)	target angle, currently in $\{-\pi/2, 0, +\pi/2\}$
`steps`	int	host (SETTLE)	sim-side dwell, advisory for firmware
`note`	string	log only	e.g. `layer_turn`, `cube_reorient`

Putting these pieces together, the end-to-end pipeline is shown in Figure 6. The Kociemba solver and the dual-arm controller are owned by the simulation-and-control module; the camera array, the PCB driver layer, and the dual-arm mechanism belong to the other three modules and connect through the interfaces defined in Section 2.

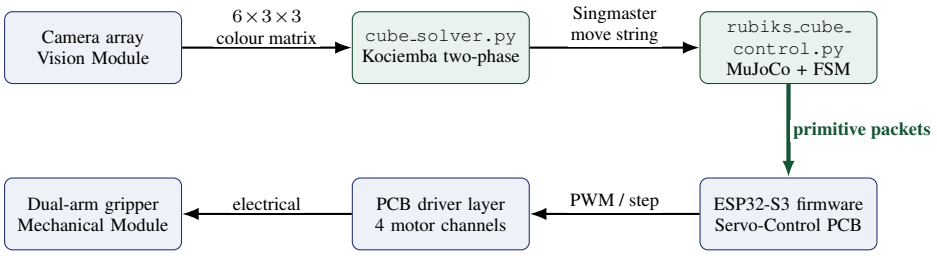


Figure 6: End-to-end pipeline from camera frame to motor command. The thick vertical arrow is the sim-to-real boundary, carrying only the discrete primitive packet stream. All other transports along the pipeline are conventional message passing (JSON, USB serial, PWM) and are described in Table 1.

4 Vision Module

The vision module is responsible for converting the physical cube state into a $6 \times 3 \times 3$ colour matrix consumed by the Kociemba solver. The pipeline (Figure 7) is intentionally decoupled into class-agnostic sticker localisation followed by an independent colour classification stage, which removes a known failure mode of single-network colour detectors that overfit to specific lighting conditions.

Figure 7: Decoupled vision pipeline. Locking the camera's auto-exposure and white balance stabilises RGB values when the white 3D-printed arms enter the field of view; YOLOv8 then performs class-agnostic sticker detection, KNN assigns a colour to each bounding box, and a geometric sort emits the $6 \times 3 \times 3$ matrix consumed by the Kociemba solver.

4.1 Hardware Selection and ISP Configuration

Because image capture only occurs when the mechanical arms and the cube are completely stationary, an electronic rolling shutter is sufficient and the camera selection optimises cost rather than dynamics. A 1080p USB 2.0 camera module (Table 8) provides the required resolution and field of view.

Table 8: Camera module key parameters.

Parameter	**Specification**
Sensor	1/2.7-inch CMOS
Resolution	1920×1080
FOV	DFOV: 95°
Focus type	Fixed Focus (FF)
Shutter type	Electronic rolling shutter
Interface	High-Speed USB 2.0

A critical constraint of the vision system is lighting consistency. The camera's automatic ISP features — Auto-Exposure and Auto-White-Balance — are explicitly disabled through the OpenCV API at initialisation, preventing the sensor from automatically rescaling colour representations when the white 3D-printed arms enter or leave the field of view. This fixes the radiometric mapping from scene to pixel, which is a precondition for the KNN colour classifier.

4.2 Decoupled Detection and Classification

An initial design attempted to train a YOLO model to directly output six colour classes; this configuration overfit to the lighting conditions present at training time and degraded under any spectral shift. The pipeline is therefore decoupled into two independent stages.

Sticker localisation. A YOLOv8-nano model is trained on a single-class detection task: identifying the plastic stickers on the cube regardless of their colour. The physical geometry of the cube and its black grid lines are stable across all cube states and provide reliable features for this task. Training parameters are summarised in Table 9; Figure 8 shows the training curves. Per-frame inference output is shown together with the colour classifier output in Figure 9, because the two stages operate on the same frame.

Table 9: YOLOv8-nano training parameters.

Parameter	**Value**	**Parameter**	**Value**
Epochs	50	Image size	640×640
Batch size	16	Optimiser	AdamW
Initial LR	0.001	Target class	1 (`sticker`)

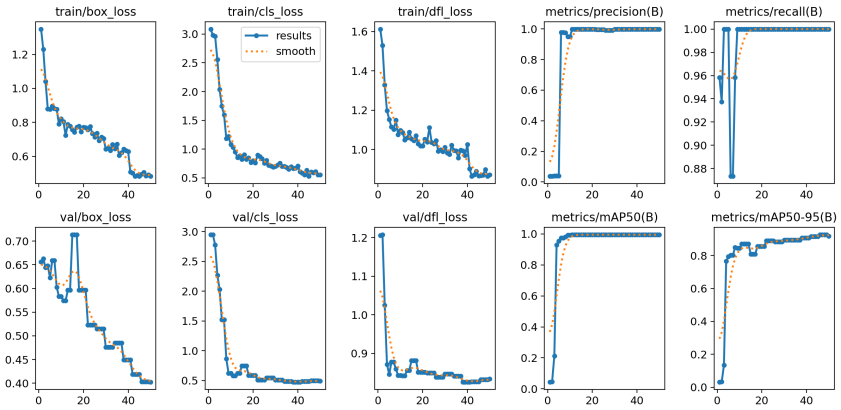


Figure 8: Training curves of the YOLOv8-nano sticker detector.

Colour classification. The colour of each detected sticker is assigned by a K-Nearest Neighbours (KNN) classifier trained on a dataset of several hundred colour samples collected directly from the deployment camera under operating illumination. Avoiding hard-coded RGB thresholds in favour of a learned classifier provides robustness against small spectral variations without requiring a vision retrain.

4.3 State Vector Assembly

YOLO emits bounding boxes in an unordered sequence. A geometric sorting stage orders the boxes by their y-coordinate (into rows) and then by their x-coordinate (into columns), producing a consistent 54×1 array. The overhead camera initially captures the top and front faces; the dual-arm mechanism then executes a pre-programmed flipping sequence that exposes the remaining four faces to the camera, building the full state vector. The vector is reshaped into the $6 \times 3 \times 3$ matrix and serialised to the schema consumed by `cube_solver.py`.

Figure 9: Integrated output of the vision pipeline on a representative frame: YOLOv8 has localised every visible sticker (bounding boxes), and KNN has assigned a colour class to each one (label colour). Partially occluded pieces are still classified correctly.

5 Mechanical Module

The mechanical module realises the physical cube manipulation: the dual-arm structural design, the assembled hardware, the stepper-motor driving circuit, and the camera mount that ties the perception subsystem to the frame.

5.1 Dual-Arm Structural Design

The mechanism is designed around a symmetric two-arm structure. The left and right arms approach the cube from two fixed directions and cooperate to hold, reorient, and rotate it. Each arm carries a motorised wrist mechanism and a clamping gripper. The cube sits at the geometric centre of the mechanism, with both arms maintaining contact with the reachable bottom edges.

Figure 10 shows the rendered CAD model. The assembly consists of a rigid base frame, two inclined support sections, two motor mounting regions, and the upper gripping mechanisms that interact directly with the cube. The inclined arm arrangement forms a cradle-like configuration that supports the cube while leaving room for rotation, and the upper gripper design secures the cube during motion while allowing one arm to release for reorientation when required.

Table 10 summarises the main sub-assemblies and their mechanical role. This decomposition feeds directly into the MuJoCo import described in Section 3; the physical structure and the simulated model therefore remain in one-to-one correspondence.

Table 10: Main sub-assemblies of the mechanical platform and their roles.

Sub-assembly	Qty	Role
base / frame	1	rigid global reference; fixes the relative pose of the two arms
inclined arm support	2	angled mounts bringing each arm into the cradle configuration
stepper motor + bracket	2	per-arm wrist actuation; drives the layer rotation
gripper assembly	2	clamps the bottom edge of the cube; rotates with the wrist motor
camera mount (3D-printed)	1	rigidly attaches the overhead camera to the frame

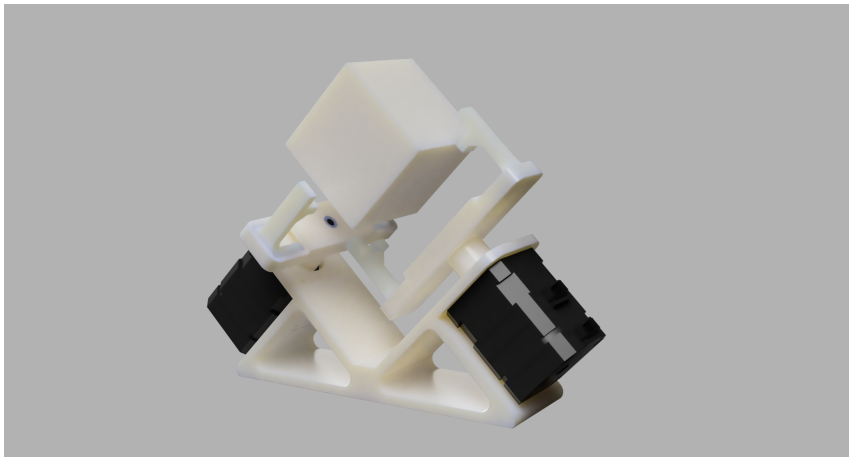


Figure 10: Rendered CAD model of the dual-arm Rubik's cube robot. The design includes a central cube-holding region, two symmetrically arranged actuation arms, and dedicated motor mounting locations.

5.2 Physical Assembly

Assembly proceeds from the CAD model with particular attention to the alignment between the two arms, because even small geometric deviations affect the security of the clamp and the smoothness of layer rotations. The grippers are positioned to contact the cube at repeatable locations: each gripper closes on the middle edge cubie of one reachable outer layer, matching the assumption embedded in the simulation model that the real claw centre lands on the cubie centre.

The dual-arm arrangement supports two essential mechanical behaviours: holding the cube stably with both arms (the rest configuration) and allowing one arm to manipulate the cube while the other remains fixed (the active configuration). These behaviours map directly onto the controller's action classes in Table 5.

5.3 Stepper-Motor Driving Circuit

The mechanism requires precise angular motion, for which stepper motors are well suited: they provide repeatable position-based rotation and are driven by discrete pulse commands. The driving circuit (Figure 11) consists of a regulated DC supply, the TB6600 driver stage, the STM32 control module, and the stepper motor.

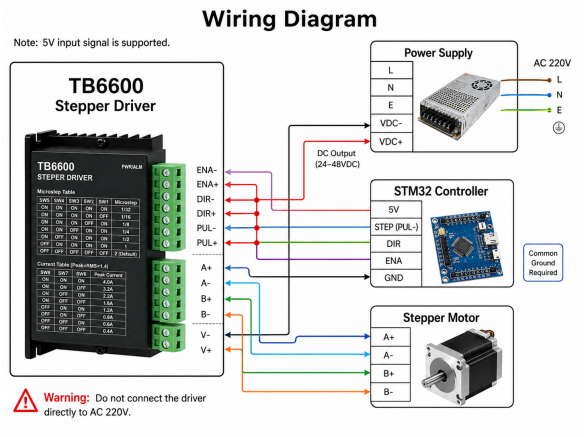


Figure 11: Stepper-motor driving circuit, including the power supply, TB6600 drivers, STM32 control module, and stepper motor connections.

At the system level, robot motion is represented as a sequence of discrete primitive commands such as clamping, releasing, and rotating (Tables 6 and 7). These commands are translated by the embedded layer into electrical motor signals, and the stepper-motor driving circuit closes that path by converting digital pulse trains into shaft rotation.

5.4 Camera Mounting

A 3D-printed bracket attaches the overhead camera to the robot frame. The bracket maintains a stable, fixed pose between the camera and the cube holding region, which is a precondition for locking the camera's exposure and white balance as discussed in Section 4. A photograph of the fully assembled platform with the camera installed is shown in Figure 13 as part of the integrated hardware summary.

6 Servo-Control PCB Module

The servo-control PCB is the electrical interface between the ESP32-S3 controller and the servo actuators. It distributes regulated power, expands the controller's outputs to 16 independent PWM channels, and exposes the debug connections required during integration.

6.1 Architecture

The design is built around the PCA9685 PWM controller. Instead of dedicating one ESP32 GPIO per servo, the ESP32 communicates with the PCA9685 over I^2C; the PCA9685 then generates independent PWM outputs for the servo connectors. This architecture reduces the number of direct control wires from the microcontroller and keeps the actuator wiring organised through standard three-pin headers.

The completed schematic includes the main power input, input protection, voltage regulation, PWM generation, servo connectors, communication interfaces, user input, and debug indicators. The external input rail is protected by a fuse and Schottky protection devices, then converted to 5 V for the servos by an XL4015 buck-converter stage. A secondary AMS1117 regulator generates the 3.3 V logic rail. Together these meet requirement R6: a single protected DC input supplies both rails, and the servo current path returns through its own copper region to limit ground displacement of the logic reference.

Bring-up and debug circuitry includes USB and UART connections for development access, a push-button input, a power indicator, a 3.3 V indicator, a status LED, and an error LED. The visible status indicators satisfy requirement R7 (operator-visible state).

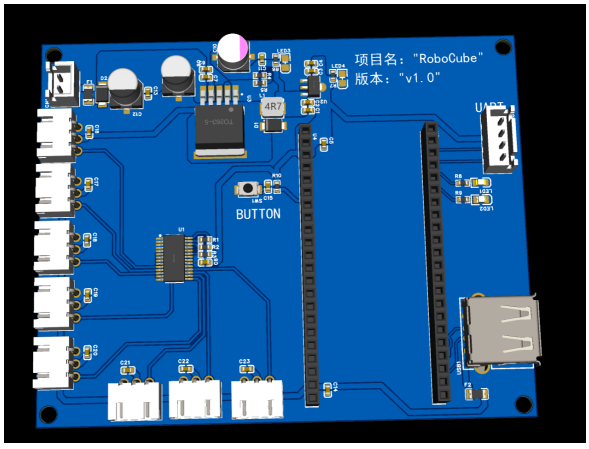


Figure 12: 3D preview of the completed servo-control PCB. The board integrates the PCA9685 PWM controller, the XL4015 5 V servo-power stage, the AMS1117 3.3 V logic regulator, servo connectors, controller headers, USB and UART interfaces, status LEDs, a push-button, mounting holes, and silkscreen labelling.

6.2 Layout and Stack-up

The PCB is implemented as a two-layer FR-4 board with the stack-up given in Table 11 (nominal total thickness 1.6 mm). The layout separates the power-conversion region, the logic-control region, and the actuator-output region. Servo connectors sit along the board edge to simplify external wiring to the robot arms; the ESP32 and controller headers are placed in the central area to shorten the I^2C and control-signal routes. Large copper-pour regions on both layers provide stable reference and return paths, and the four mounting holes define the mechanical interface to the robot structure.

Table 11: PCB stack-up of the servo-control board.

Layer / material	Thickness (mm)
Top solder mask	0.010
Top copper	0.035
Core dielectric	1.510
Bottom copper	0.035
Bottom solder mask	0.010

Table 12 lists the external interfaces of the board and the role each plays in the integrated system: the UART header carries the primitive packet stream from the host, I^2C carries PWM expansion commands from the ESP32 to the PCA9685, and the servo connectors drive the four arm-actuation channels listed in Table 3.

Table 12: External interfaces exposed by the servo-control PCB.

Interface	Direction	Signal type	Role in the system
Main power input	in	DC, fused + Schottky-protected	raises rail to 5 V (XL4015) and 3.3 V (AMS1117)
USB	bi-dir.	5 V + USB data	development, flashing, optional logging
UART header	bi-dir.	TTL	host $\leftrightarrow$ ESP32 link carrying primitive packets and ACK / error codes
I^2C	bi-dir.	SDA / SCL	ESP32 $\leftrightarrow$ PCA9685 PWM controller
Servo connectors	out	3-pin (PWM / V_{servo} / GND)	drives the four arm-actuation channels of Table 3
Status / error LEDs	out	GPIO	visible bring-up indicators
Push button	in	GPIO	operator start / reset (R1, R7)

7 Requirements and Verification

This section maps the high-level requirements of Section 1.1 onto subsystem-level requirements with quantitative verification methods. Each table lists the requirement, the procedure used to verify it, and the result observed at the time of this report. Items that depend on the first full hardware bring-up are marked "Pending integration."

Figure 13 shows the integrated hardware that hosts the verification work to date.

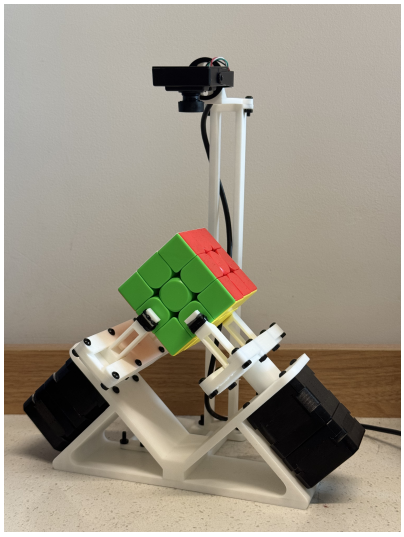


Figure 13: Fully assembled dual-arm Rubik's cube robot with the overhead camera mounted on the 3D-printed bracket. This is the integrated platform against which the verification procedures below are run.

7.1 Simulation and Control-Signal Module

Most of the requirements imposed on this module can be verified in software, since the simulation, the solver, and the controller all run on the host. The single open item (S6) is the host-to-embedded packet round-trip, which depends on firmware bring-up and is deferred until then. Table 13 lists the requirements together with the verification procedures applied.

Table 13: Requirements and verification: simulation and control-signal module.

#	Requirement	Verification method	Result
S1	Kociemba solver returns a finite, legal move sequence for any solvable cube state	Self-test on solved state (expected empty), checkerboard state (expected finite), and 100 random scrambles	Verified: all self-tests pass
S2	Layer rotation in MuJoCo preserves the 27-cubie logical grid without drift	Apply 1000 random face turns; assert grid coordinates remain integer-valued after each step	Verified: zero drift observed
S3	Singmaster parser handles $\{U, D, L, R, F, B\}$ with suffixes $\{', 2\}$	Unit tests against handwritten reference sequences	Verified
S4	Controller emits a discrete primitive packet for every layer turn and reorientation	Run `R U R' U'` with `--log-primitives`; inspect packet log	Verified: 36 packets, vocabulary matches Table 6
S5	Primitive packet log is replayable across executions	Bit-exact diff of two independent runs on the same move string	Verified
S6	End-to-end host $\rightarrow$ embedded packet round-trip	Send packets over USB serial; firmware ACKs each event	Pending integration

7.2 VISION MODULE

Verification for the vision module combines unit-level checks on each pipeline stage (ISP locking, sticker detection, colour classification) with integrated tests against scrambled-cube frames. The per-stage checks confirm that the architectural assumption of the decoupled pipeline holds; the integrated tests probe the system-level accuracy target in R3. Table 14 lists the requirements and the corresponding methods.

Table 14: Requirements and verification: vision module.

#	Requirement	Verification method	Result
V1	Camera ISP is locked at initialisation (constant exposure and white balance)	Capture before / after arm enters frame; verify pixel statistics unchanged in the cube region	Verified
V2	Sticker detector locates all visible stickers on a face	Inference on held-out test set of N images covering all face configurations	Verified (see Figure 8)
V3	Colour classifier assigns the correct colour for $\geq 95\%$ of detected stickers	Cross-validation on collected sample set; field test on `N` scrambled-cube frames	Field accuracy: pending end-to-end test
V4	Pipeline emits a $6 \times 3 \times 3$ matrix in the schema of `sample_state.json`	Run end-to-end on a known cube state; compare emitted JSON against ground truth	Verified on single-frame scans
V5	Full six-face scan is constructed via dual-arm flipping	Integrated test with the simulation-and-control module driving the flipping sequence	Pending integration

7.3 MECHANICAL MODULE

The mechanical module is verified through a mix of dimensional measurements (arm coaxiality, camera-mount stability) and bench tests on the actuators (pulse-train response of the stepper-motor circuit). The gripper-contact requirement (M2) cannot be fully evaluated until the firmware drives the wrist motors through a real layer turn; it is therefore deferred to first hardware bring-up. Table 15 summarises the requirements and verification methods.

Table 15: Requirements and verification: mechanical module.

#	Requirement	Verification method	Result
M1	Dual-arm assembly maintains arm coaxiality within tolerance such that the cube sits centred between the grippers	Direct measurement of inter-arm distance and gripper centreline; visual alignment with the cube cradle	Verified at build (recheck before solve)
M2	Each gripper closes on the middle edge cubie of its reachable outer layer without slipping during a quarter turn	Manual test: clamp, drive wrist motor through $\pm\pi/2$, inspect cubie pose	Pending: requires firmware bring-up
M3	Stepper-motor driving circuit produces controllable, repeatable wrist rotation	Pulse-train test on TB6600 with the stepper motor in isolation	Verified at bench
M4	Camera mount maintains a fixed pose between camera and cube	Repeatability check: capture frames before / after small mechanical disturbance; compare pixel coordinates	Verified

7.4 SERVO-CONTROL PCB MODULE

The PCB is verified at the bench level: the regulator stages (XL4015 and AMS1117) are measured under both no-load and operational current conditions, and the PCA9685 outputs are probed on the oscilloscope to confirm 16 independent PWM channels. The two items that require the end-to-end

system (UART link integrity P4 and firmware status indicators P6) remain pending until firmware bring-up. Table 16 lists the requirements and methods.

Table 16: Requirements and verification: servo-control PCB.

#	Requirement	Verification method	Result
P1	XL4015 stage regulates the servo rail to $5.0\,\mathrm{V} \pm 5\%$	Bench measurement under no-load and at expected servo current draw	Verified
P2	AMS1117 stage regulates the logic rail to $3.3\,\mathrm{V} \pm 5\%$	Bench measurement under typical I^2C and PCA9685 load	Verified
P3	PCA9685 produces 16 independent PWM channels addressable via I^2C	Sweep duty cycle on each channel and observe with oscilloscope	Verified
P4	UART link carries primitive packets between host and ESP32 without packet loss	Loopback test: send 1000 packets; verify ACK count	Pending integration
P5	Servo and logic ground returns are physically separated on the layout	Visual layout review; impedance check between servo GND and logic GND	Verified
P6	Status / error LEDs indicate operating state (idle, running, fault)	Firmware writes to the corresponding GPIOs; LED state observed	Pending firmware

8 Ethics and Safety

The project is developed in accordance with the IEEE Code of Ethics [8], which emphasises the safety, health, and welfare of the public, honest and realistic claims about technical work, and the avoidance of harm. Three classes of safety concern are relevant to this prototype.

Electrical safety. The servo-control PCB ingests an external DC supply, regulates it to 5 V and 3.3 V rails, and drives motor actuators through the PCA9685 expansion. Input protection consists of a fuse and Schottky diodes upstream of the regulators, which limits damage in the event of polarity reversal or over-current. Servo and logic grounds are physically separated on the layout (requirement P5) to avoid ground-displacement disturbance of the controller from servo inrush currents. All bench testing is performed at low currents with the supply current-limited.

Mechanical safety. The dual-arm mechanism uses stepper-driven wrist motors and servo-driven clamp actuators. The grippers operate over a short travel (clamp range $[0, 0.16]$ in actuator units) and exert limited force; pinch hazards on operator fingers are mitigated by the small clamp aperture and slow motion profile. The robot is intended to be operated on a tabletop with the operator outside the workspace; the push-button start (R7) ensures that motion only begins on a deliberate operator action, and a planned hardware emergency stop will cut the servo rail at the regulator stage without disturbing logic supply.

Data and privacy. The vision pipeline captures imagery only of the cube within the camera's field of view, which is constrained by the camera's physical mounting. No human subjects, identifiable backgrounds, or off-task imagery are recorded or stored. All training data for the YOLO detector and the KNN classifier is collected internally under controlled conditions and is not redistributed.

The team commits to honesty in reporting: every quantitative claim in Section 7 is annotated with its verification status, and items that have not yet been measured end-to-end are explicitly marked "Pending integration" rather than being represented as verified.

9 Conclusion

This report has presented the design of a four-module autonomous dual-arm Rubik's cube solving robot. The functional decomposition into vision, simulation and control-signal, mechanical, and PCB modules is coupled only through three narrow interfaces: a $6 \times 3 \times 3$ cube colour matrix, a discrete primitive-packet stream, and a four-channel PWM map to the actuators. Independently of

one another, each module is functional on its own. The Kociemba solver produces verified solving sequences; the MuJoCo simulation drives the four-move warm-up sequence `R U R′ U′` end-to-end into 36 firmware-ready primitive packets; the vision pipeline localises and classifies all visible stickers under locked exposure; the mechanical platform is assembled and the stepper-driving circuit drives the wrist motors as designed; and the servo-control PCB provides regulated 5 V and 3.3 V rails with a 16-channel PWM front end.

The remaining work centres on the three module boundaries. A physical measurement pass is required to confirm that the real gripper contact lands on the middle cubie of the targeted outer layer and to reconcile any offset against the claw collision geometry in `cube_mujoco/model/cube_dualarm.xml`. The ESP32 firmware needs to implement the parser for the schema in Table 7 together with the state machine {IDLE, WAIT_PLAN, EXECUTE, DONE, ERROR}, calibrate the clamp setpoint and the rotation targets against real motor encoders, and enforce the RELEASE invariant in hardware. The integrated bring-up then proceeds in escalating order: a single `R` move, then `R U`, then `R U R′ U′`, and only after that round-trips cleanly with the simulation packet log, a complete Kociemba solution.

Author contributions. Keeron Huang led the design of the simulation environment, the dual-arm controller, the FSM, and the primitive-packet specification. Yiming Xu led the visual perception module, including hardware selection, the decoupled YOLOv8 + KNN pipeline, and the geometric sort that produces the $6 \times 3 \times 3$ state matrix. Rong Wang led the mechanical design and the physical assembly of the dual-arm platform, the stepper-motor driving circuit, and the camera mount. Zhuoyang Shen led the design and layout of the servo-control PCB, including the PCA9685-based PWM expansion, the dual-rail power stage, and the debug-interface design.

REFERENCES

1. H. Kociemba. *Cube Explorer / Two-Phase Algorithm*, 1995. `https://kociemba.org/cube.htm`.
2. E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ IROS*, 2012, pp. 5026–5033.
3. Ultralytics. *YOLOv8 Documentation*, 2023. `https://docs.ultralytics.com`.
4. G. Bradski. "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
5. Espressif Systems. *ESP32-S3 Series Datasheet*, v1.5, 2024.
6. NXP Semiconductors. *PCA9685: 16-channel, 12-bit PWM Fm+ I^2C-bus LED controller*, Rev. 4, 2015.
7. Toshiba. *TB6600HG: PWM chopper-type stepper motor driver IC*, 2014.
8. IEEE. *IEEE Code of Ethics*, 2020. `https://www.ieee.org/about/corporate/governance/p7-8.html`.