ELECTRONIC DESING WORKSHOP
RUBIK CUBE SOLVING ROBOT

UTPAL SHARMA(2023UEC2534)
JAYAVARDHAN SINGH (2023UEC2531)
VARUN (2023UEC2506)

MAY 2025

Contents

Abstract

This project presents the design and development of an automated Rubik's Cube solving robot that integrates mechanical engineering, embedded systems, and computer vision. The system is capable of scanning a scrambled 3×3 Rubik's Cube, computing the solution using Kociemba's two-phase algorithm, and executing physical moves via a custom-designed 3D-printed mechanical structure.

The cube's state is captured using a Python script that utilizes OpenCV for image processing, mapping colors based on Hue, Saturation, and Intensity (HSI) values. The resulting cube configuration is converted into a string of face-letters and passed into the Kociemba algorithm to determine an efficient solution path. This solution string is then transmitted wirelessly to an Arduino Nano using a Bluetooth module (HC-05), which in turn controls eight servo motors responsible for gripping and rotating cube faces through four coordinated claws.

The robot's mechanical frame, built using PLA and designed in FreeCAD, provides a stable platform for the precise movement of the cube. This project demonstrates how a classic puzzle can be solved using an interdisciplinary engineering approach, combining computer vision, algorithmic logic, and mechatronics to bring theoretical concepts into practical realization.

1 Introduction

The Rubik's Cube, invented in 1974 by Ernő Rubik, is one of the most iconic and enduring mechanical puzzles in history. It consists of a 3×3×3 cube with six faces, each subdivided into nine smaller squares that can be rotated independently. The challenge lies in returning the cube to its original state, with each face displaying a single solid color, after it has been scrambled. Despite its seemingly simple appearance, the Rubik's Cube has over 43 quintillion possible configurations, making it a complex combinatorial puzzle.

Solving the Rubik's Cube requires not only an understanding of its structure but also logical thinking, pattern recognition, and memory. While human solvers can learn predefined algorithms to solve the cube, automating the process involves an integration of mechanical design, electronics, image processing, and algorithmic computation.

The motivation behind building an automated Rubik's Cube solver stems from its interdisciplinary nature. It provides an excellent platform to apply concepts from embedded systems, robotics, control systems, and artificial intelligence. Moreover, such a project demonstrates how a real-world physical problem can be solved using engineering principles. For students and enthusiasts, it is both a challenging and rewarding way to bring theory into practice.

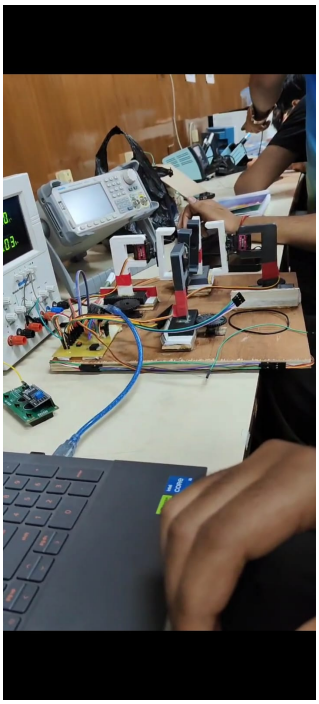


Figure 1:

2 Literature Review

The Rubik's Cube is a famous 3D combinatorial puzzle with a seemingly simple structure but an exponentially complex solution space. Over the years, many approaches have been proposed to solve the cube, ranging from human techniques and computer algorithms to fully automated robots. This section reviews notable contributions to Rubik's Cube solving robots and algorithms.

2.1 Rubik's Cube Solving Algorithms

The two most well-known algorithms for solving the Rubik's Cube are Kociemba's Algorithm and Thistlethwaite's Algorithm, both of which are used for solving the puzzle optimally or near-optimally.

2.1.1 Kociemba's Algorithm

The Kociemba Algorithm is one of the most popular algorithms for solving the Rubik's Cube. It is a two-phase algorithm that can find solutions in fewer moves compared to other methods. The algorithm works as follows:

- **Phase 1:** Reduces the cube to a set of states that are easier to solve, requiring fewer move types (e.g., no "L" or "R" moves).
- **Phase 2:** Solves the cube from the reduced state to the completed configuration.

This algorithm is well-suited for automated systems due to its speed and efficiency in finding solutions within 20 to 25 moves. It is used in many Rubik's Cube solvers, including the ones implemented in this project.

2.2 Rubik's Cube Solving Robots

Automated Rubik's Cube solvers can be broadly classified into two categories: mechanical solvers and robotic systems.

2.2.1 Mechanical Solvers

Mechanical solvers focus on the physical manipulation of the Rubik's Cube, using motors, gears, and claws to rotate the faces of the cube. Early examples include robots that use stepper motors or servos to turn the cube's faces one by one based on pre-programmed algorithms. For example, the "CubeStormer II" robot, built from LEGO Mindstorms, was able to solve a scrambled Rubik's Cube in about 5.6 seconds.

One significant limitation of such systems is their reliance on predefined motor setups and limited torque, making it difficult for them to handle irregularities like slippage or inaccurate movements, particularly when motors are not strong enough.

2.2.2 Vision-Based Rubik's Cube Solvers

Another approach integrates computer vision for scanning the cube before performing the necessary rotations. This allows the system to identify cube faces and stickers, even in irregular lighting conditions. Vision-based systems typically use cameras or webcams to capture images of the cube, and image processing algorithms, such as those provided by OpenCV, are used for color recognition and face detection.

One notable example is the "Rubik's Cube Solver Robot" by the University of Twente. The robot utilizes a webcam to capture images of the cube, which are then processed using a custom image recognition algorithm. Once the cube's state is identified, the robot uses a robotic arm to execute the solution using an efficient algorithm like Kociemba's.

2.2.3 Hybrid Systems

Some modern Rubik's Cube solvers combine mechanical and vision-based systems to enhance their accuracy and speed. These systems use cameras for scanning and processing, while sophisticated algorithms generate solutions that are executed by robotic arms or claws, powered by high-torque motors. Such hybrid systems improve both the speed and accuracy of the solving process, overcoming the limitations of purely mechanical systems.

An example of a hybrid solver is the "OpenAI Rubik's Cube Solver," which uses deep learning models for enhanced pattern recognition and has the capability to solve the cube autonomously with minimal human intervention.

2.3 Summary of Existing Work

Many Rubik's Cube solvers have successfully implemented mechanical and algorithmic strategies to solve the cube, with improvements in efficiency and reliability. Vision-based systems, combined with advanced algorithms like Kociemba's, offer an optimal balance of speed and accuracy for real-time robotic applications. However, challenges such as motor torque, color recognition in varying lighting conditions, and mechanical precision remain significant obstacles.

This project distinguishes itself by combining the Kociemba algorithm with a custom-designed mechanical system using 3D-printed parts and a Bluetooth communication interface. Unlike traditional solvers, the robot relies on real-time color recognition and a feedback mechanism, allowing it to solve the cube autonomously. The potential for improvement lies in the use of higher torque motors and further refinement of the software and mechanical design to overcome the challenges observed in existing systems.

3 PROJECT TIMELINE

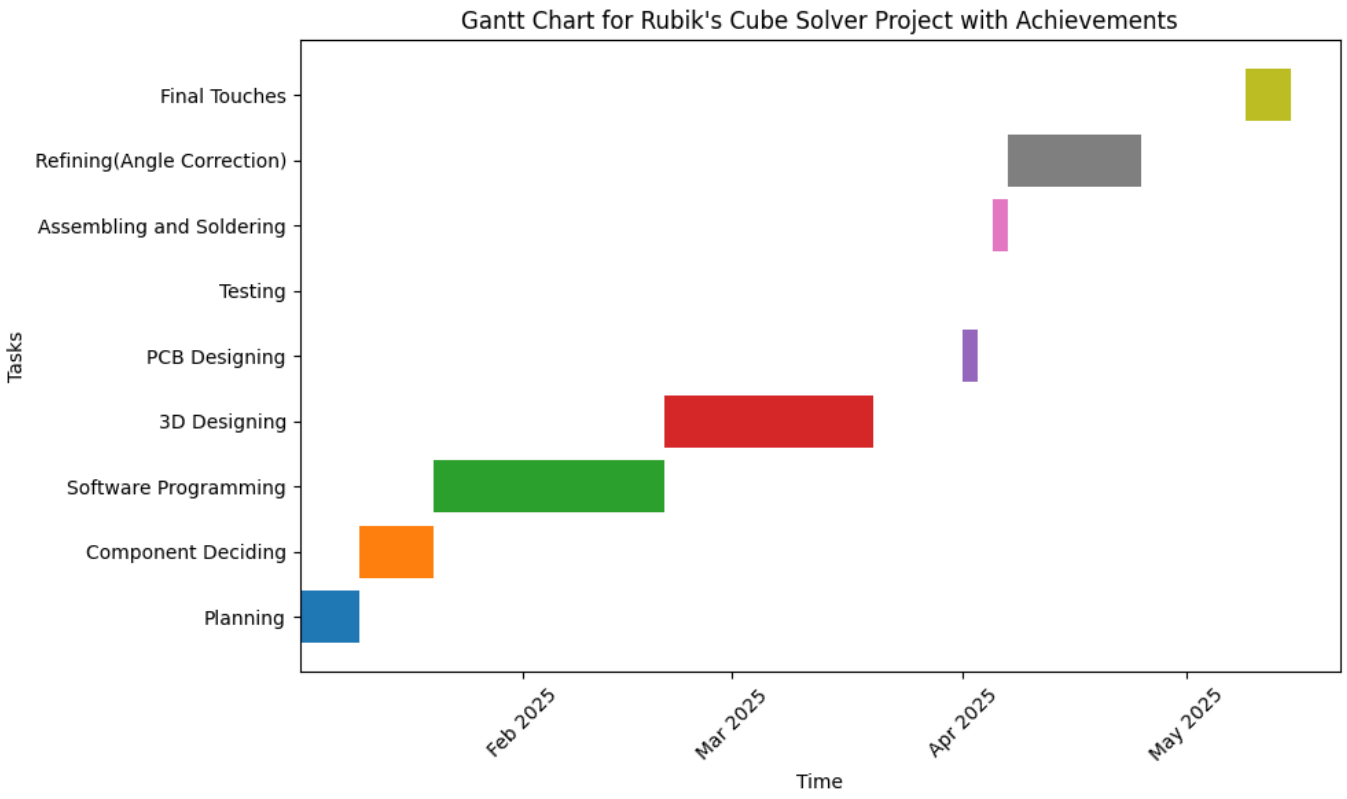


Figure 2: PROJECT TIMELINE

4 System Overview

The Rubik's Cube solving robot is built around three core subsystems: the mechanical structure, microcontroller-based control, and a laptop-based computing interface. These components work together to achieve autonomous cube solving with minimal user intervention.

4.1 Mechanical Subsystem

The robot uses eight servo motors mounted on a custom 3D-printed frame. These motors are arranged to grip and rotate the cube's individual faces. The design allows for precise face manipulation required for solving the cube efficiently.

4.2 Computing and Vision Subsystem

Instead of onboard image processing, the system uses a laptop to scan and process the cube. The user manually presents each face of the scrambled cube to the laptop's webcam. A

Python-based image processing script detects the colors of all stickers and reconstructs the cube's full state in software.

4.3 Solving Algorithm

Once the cube state is known, the laptop computes an optimal solution using a well-known algorithm like Kociemba's. The resulting sequence of moves is formatted and prepared for transmission to the robot.

4.4 Control Subsystem

The laptop sends the computed move instructions via Bluetooth using the HC-05 module. The ATmega328P microcontroller on the robot receives these commands and controls the servo motors accordingly to execute the moves.

This modular setup efficiently splits computational and mechanical responsibilities. The laptop handles vision and logic, while the microcontroller performs precise motion control, enabling a cost-effective and functional Rubik's Cube solving robot.

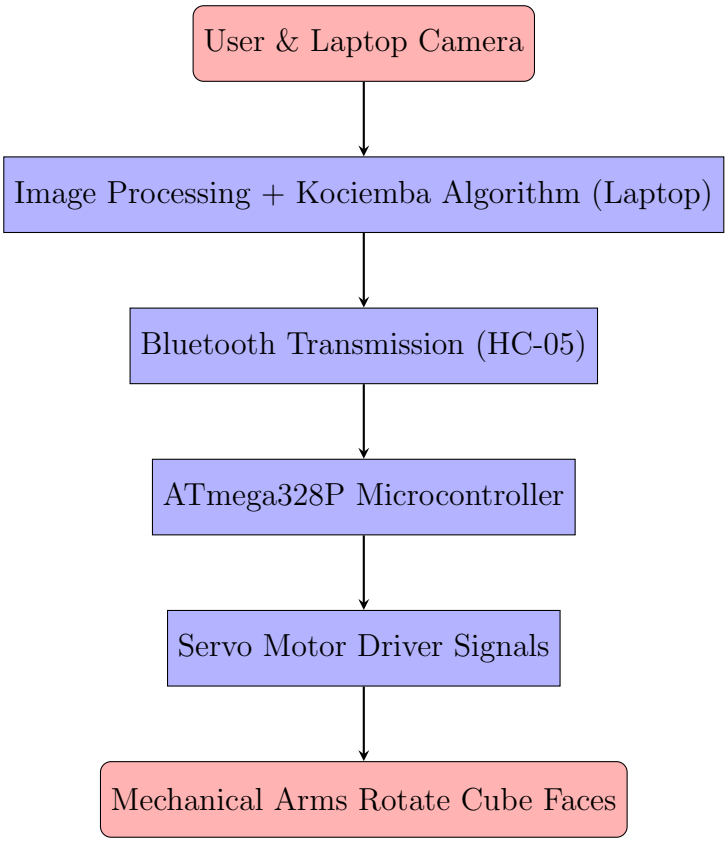


Figure 3: Block Diagram of Rubik's Cube Solving Robot

5 Components Used

- Arduino Nano
- 8 Servo Motors
- HC05 bluetooth module
- 3D Printed Mechanical Parts
- Power Supply
- Liquid Crystal Display
- I2C

6 Bill of Materials

The table below lists the components used in the Rubik's Cube solving robot along with their approximate costs. The prices are in Indian Rupees (INR) and are estimated based on local market availability.

Component	**Quantity**	**Approximate Cost (INR)**
Servo Motor (SG90)	8	800
LCD Display (16x2)	1	160
I2C Module for LCD	1	160
Buck Converter	1	50
Arduino Nano	1	400
Electronic Components (resistors, wires, breadboard, etc.)	–	200
Total Estimated Cost	–	**1,770**

Table 1: Bill of Materials for Rubik's Cube Solving Robot

7 Working Principle

The Rubik's Cube solving robot operates through a well-coordinated sequence of scanning, processing, solving, and actuation. Each stage contributes to autonomous cube solving through dedicated subsystems.

7.1 Scanning and Color Detection

The scanning process begins with a Python script running on a laptop, which uses the OpenCV library for image acquisition and processing. The user manually presents each of the six faces of the cube to the laptop's camera, ensuring consistent alignment. The program detects the colors of the stickers based on their hue, saturation, and intensity (HSI) values, allowing for accurate color identification despite variations in lighting.

7.2 Cube State Conversion

After detecting the colors of all 54 stickers, the script maps them into a standardized color matrix. This matrix is then converted into a face-letter notation (e.g., U for Up, F for Front, R for Right, etc.), which represents the cube's state in a format compatible with solving algorithms.

7.3 Solving Algorithm – Kociemba's Algorithm

The generated cube state is fed into Kociemba's algorithm, a two-phase algorithm that efficiently solves a Rubik's Cube in fewer moves than traditional layer-based methods. In Phase 1, the algorithm reduces the cube to a state belonging to a constrained subset of configurations; in Phase 2, it transitions from that subset to the solved state. This method typically produces a solution in under 25 moves.

7.4 Communication and Control

The resulting solution string—a sequence of face moves—is transmitted to the Arduino Nano via the HC-05 Bluetooth module. The Arduino parses this string and determines the corresponding servo movements required to perform each rotation.

7.5 Actuation Mechanism

The Arduino Nano controls eight servo motors mounted on a 3D-printed mechanical frame. These include rotating arms and gripping claws that manipulate the cube's faces. For each move, the arms and claws coordinate to grip the appropriate face, rotate it precisely, and release it—executing the full solving sequence step-by-step.

Through the integration of vision, computation, and precise mechanical actuation, the robot is able to solve a scrambled Rubik's Cube autonomously after scanning is completed.

8 Circuit and PCB Designing

In this project, we designed two separate circuits and corresponding PCBs—one using the Arduino Nano v3.0 development board, and the other using a bare ATmega328P microcontroller. Both designs share the same functional goals but vary in complexity, flexibility, and cost.

8.1 Arduino Nano v3.0 Based Design

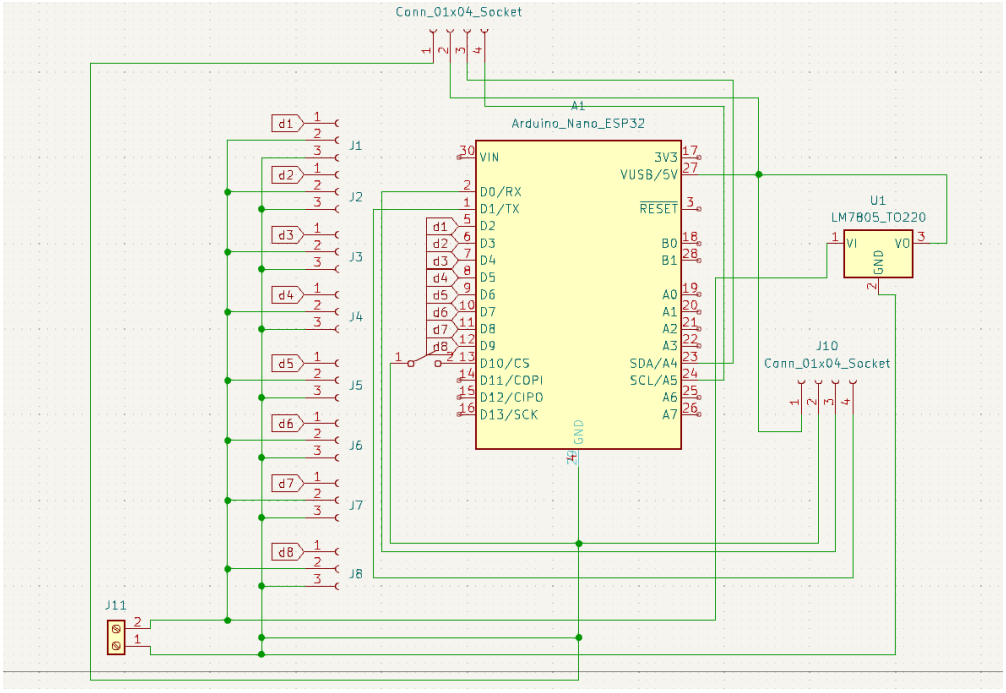


Figure 4: ARDUINO NANO BASED

The first design utilizes an Arduino Nano v3.0, which integrates the ATmega328P microcontroller with essential components like voltage regulators, USB interface, and oscillator circuits. This design simplifies the circuit by reducing the number of external components.

- Digital input buttons (D1 to D8) are connected to pins D2 through D9 of the Arduino Nano.
- Each button has a pull-down configuration to ensure a defined logic level.
- A switch is added between pin D10 and ground for an additional input control.
- Power is supplied via an LM7805 voltage regulator (U1), converting a higher input voltage to a regulated 5V for the Arduino Nano.
- Pin headers (J1 to J8 and J11) are used for external connections, making the circuit modular and easier to debug or expand.

This configuration takes advantage of the pre-built Arduino Nano module, which simplifies prototyping and software development using the Arduino IDE.

8.2 Bare ATmega328P Based Design

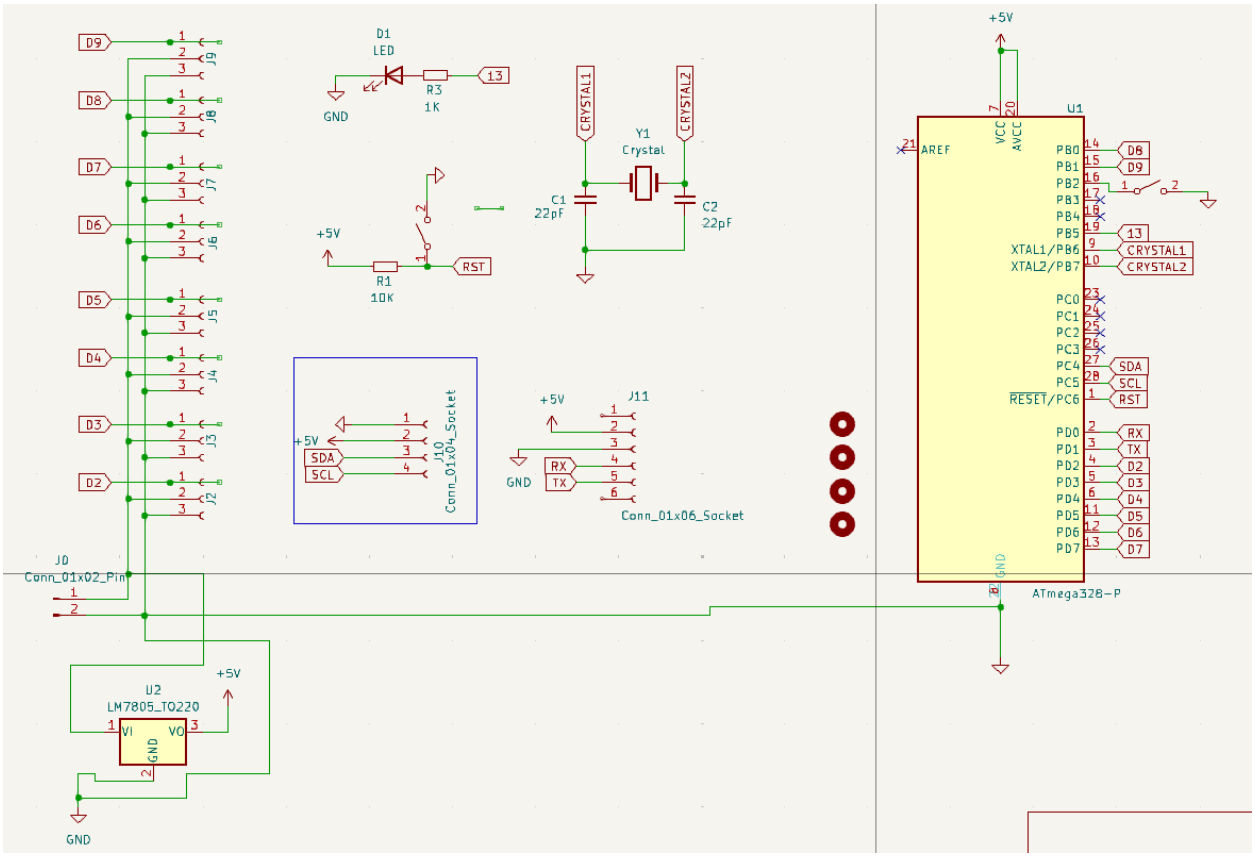
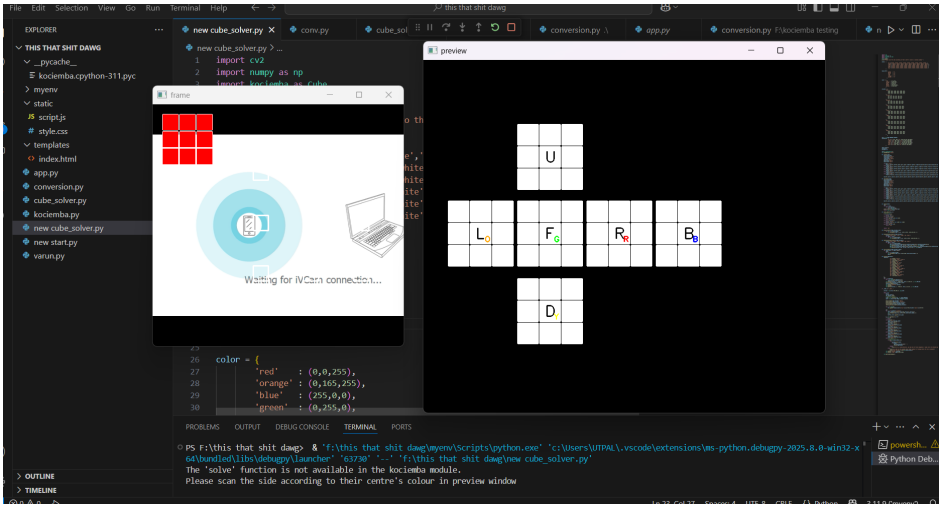


Figure 5: BARE MICROCONTROLLER (ATMEGA328P)

The second design involves the bare ATmega328P microcontroller, allowing for greater control over the hardware and potential cost savings in mass production.

- The same button inputs (D2 to D9) are wired directly to the microcontroller's PD0 to PD7 pins.
- A 16 MHz crystal oscillator (Y1) with two 22pF capacitors (C1 and C2) is connected to the XTAL1 and XTAL2 pins to provide the necessary clock signal for the microcontroller.
- A reset circuit is implemented using a 10k pull-up resistor (R1) and a push-button for manual reset.
- The power is again regulated using an LM7805 (U2), delivering 5V to VCC and AVCC of the microcontroller.
- Communication lines (TX, RX) and I2C lines (SDA, SCL) are routed to pin headers (J10, J11) for programming and interfacing with external devices.



- An LED with a 1k resistor (R3) is connected to PB0 for basic testing and indication.

This approach gives insight into the essential external components required to make the ATmega328P operational without the convenience of the Arduino development board. It is ideal for cost-effective production or learning purposes.

8.3 Comparison and Conclusion

Both circuit designs serve the same functionality in terms of digital input processing. The Arduino Nano design offers faster prototyping and easier debugging, whereas the bare ATmega328P design provides a more in-depth understanding of microcontroller circuits and is more suitable for custom hardware integration and large-scale deployment.

9 Software and Algorithm

The functioning of the Rubik's Cube solving robot depends on seamless coordination between software components running on the laptop and embedded code on the Arduino Nano. This section describes the software structure, the cube-solving algorithm used, and the logic for translating the solution into mechanical actions.

9.1 Python Script (Laptop Side)

The laptop runs a Python-based script that performs two major functions: scanning the cube and computing the solution.

9.1.1 Image Acquisition

Using the OpenCV library, the script captures images of each face of the cube via a webcam. The user sequentially presents all six faces in a fixed orientation.

9.1.2 Color Detection

The script analyzes each sticker using Hue, Saturation, and Intensity (HSI) values. Each color (e.g., Red, Green, Blue) is mapped based on proximity to calibrated values and associated with face-center references to ensure consistency.

9.1.3 Cube State Representation

The scanned color matrix is converted into face-letter notation (U, D, F, B, L, R), forming a linear string compatible with solving libraries like Kociemba's.

9.1.4 Solving Using Kociemba's Algorithm

This face-letter string is passed into Kociemba's two-phase algorithm, which solves the cube efficiently:

- *Phase 1:* Reduces the cube to a group of states using limited moves (subset group G1).
- *Phase 2:* Solves from G1 to the solved state using the remaining allowed moves.

The result is a compact solution string (e.g., `R U' F2 L D2`) that minimizes the number of turns and is formatted into serial commands.

9.2 Wireless Communication (Bluetooth)

9.2.1 Transmission Protocol

The Python script transmits the solution string via serial communication using the HC-05 Bluetooth module. Each move is delimited (e.g., comma-separated) for easier parsing on the Arduino side.

9.3 Arduino Nano Firmware (Embedded C/C++)

The Arduino Nano receives the command string and executes mechanical movements accordingly.

9.3.1 Command Parsing

The firmware reads the solution string and maps each move (e.g., R, R', F2) to a set of servo control instructions.

9.3.2 Servo Motor Logic

For each parsed move, the following sequence is executed:

- Activate the claw servo to grip the relevant cube face.
- Rotate the corresponding motor arm to perform 90° or 180° turns.
- Release the claw after motion completion.

9.3.3 Synchronization

Timed delays between grip, rotate, and release actions ensure the completion of one move before the next begins, avoiding mechanical interference.

9.4 Flow Summary

The full software workflow is summarized below:

1. Capture images of all six cube faces using webcam.
2. Detect and label sticker colors using OpenCV.
3. Convert the detected colors into face-letter format.
4. Pass the state to Kociemba's algorithm to generate a solution.
5. Transmit the solution string via Bluetooth (HC-05).
6. Arduino Nano parses each move and drives the servos to manipulate the cube accordingly.

10 Mechanical Design

The mechanical subsystem of the Rubik's Cube solving robot is responsible for gripping, rotating, and aligning the cube's faces during the solution process. The structure was carefully designed and 3D printed to achieve mechanical precision and stability.

10.1 Frame and Material

The entire mechanical structure was designed using FreeCAD and fabricated using 3D printing technology. PLA (Polylactic Acid) filament was chosen due to its rigidity, ease of printing, and suitability for structural applications. The resulting frame is compact, lightweight, and sufficiently rigid to withstand the forces generated during servo operations.

10.2 Claws and Gripping Mechanism

The robot utilizes four claws positioned around the cube to manipulate its faces. Each claw is mounted on a movable rack structure, which is actuated by a corresponding servo motor. These claws securely grip the cube faces when activated and release after completing the desired rotation.

10.3 Rack, Gear, and Turret Assembly

Each claw is attached to a rack that slides along a rail system. These racks are driven by a gear-turret mechanism comprising the following components:

- **Gears:** Connected to servo motors, gears transmit rotational motion to the turret assemblies.
- **Turrets:** Serve as support and rotational axes for the racks, guiding the motion required for rotating cube faces.
- **Racks:** Carry the claws and allow precise linear motion for gripping and positioning.

This gear-rack-turret system enables controlled and repeatable rotation of the cube's faces in both 90° and 180° increments, essential for solving accuracy.

10.4 CAD Design

All mechanical components—including claws, racks, gears, and the main frame—were modeled using FreeCAD. The software's parametric modeling allowed for accurate dimensioning and easy adjustments. Simulations within FreeCAD helped visualize part movement and detect mechanical interferences before physical prototyping.

10.5 Assembly Overview

The mechanical system comprises:

- Four servo-driven claws for face manipulation.
- Four gears that transfer motion from servos to the turret system.
- Four turret units guiding the rotational axes of the racks.
- A central fixture to securely hold the Rubik's Cube during operation.

10.6 Mechanical Operation

The operation of the mechanical subsystem follows this sequence:

1. A claw engages and grips the required cube face.
2. The corresponding servo motor actuates the gear-turret system to rotate the cube face.
3. Upon completing the movement, the claw disengages.
4. The system advances to the next move in the solution sequence.

Visual diagrams of the 3D model are given below.

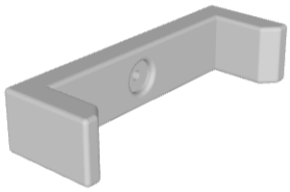


Figure 6: CLAW

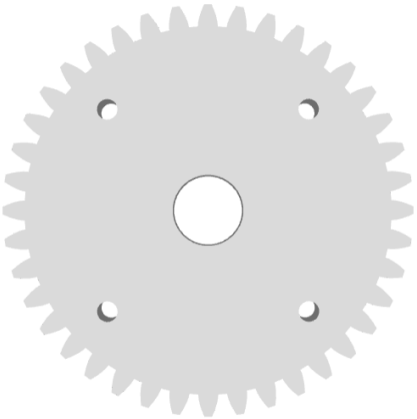


Figure 7: GEAR

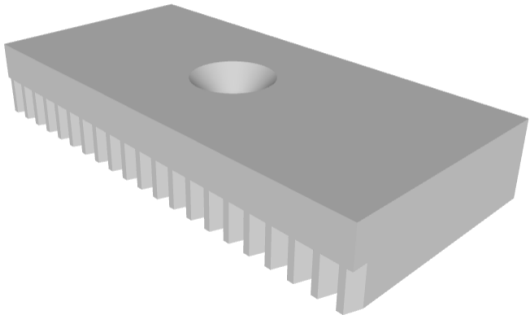


Figure 8: TURRET

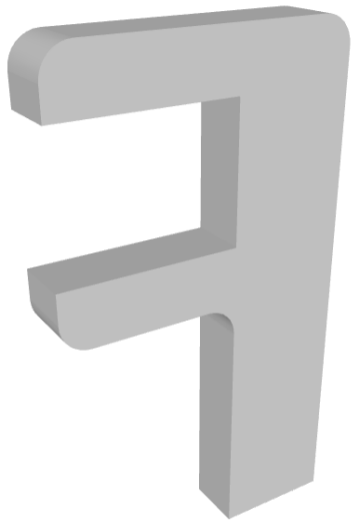


Figure 9: HOLDER

11 Challenges and Solutions

During the development of the Rubik's Cube solving robot, several challenges were encountered across mechanical design, software integration, and hardware control. This section outlines key problems faced and the solutions implemented.

11.1 Color Detection Inconsistencies

Challenge: Lighting variations and camera quality led to inconsistent color recognition during cube scanning. Some colors, especially white and yellow, were difficult to distinguish reliably.

Solution: The color detection algorithm was refined by using the Hue, Saturation, and Intensity (HSI) model instead of RGB. Threshold values were adjusted through trial-and-error to make the system robust to lighting conditions. Additionally, fixed ambient lighting was maintained during scanning.

11.2 Servo Alignment and Accuracy

Challenge: Misalignment of servo arms led to inaccurate rotations or mechanical jamming. Over-rotation caused the cube to misalign, disrupting further moves.

Solution: Each servo was calibrated carefully by adjusting the PWM pulse width. Mechanical end-stops were incorporated into the 3D-printed parts to ensure precise 90° and 180° movements. Delays were also introduced in the firmware to allow physical completion of one move before starting another.

11.3 Bluetooth Communication Errors

Challenge: Occasional packet loss or incorrect data parsing occurred during wireless transmission from the laptop to Arduino via the HC-05 Bluetooth module.

Solution: A delimiter-based protocol was implemented, and basic error-checking was added to ensure the integrity of received commands. Sending the full solution string multiple times before execution helped ensure reliable data transmission.

11.4 Solving Logic Translation

Challenge: Translating solution steps like "R', F2, U" into coordinated servo movements was initially error-prone due to ambiguity in physical orientation vs algorithmic notation.

Solution: A fixed reference orientation was maintained throughout the solution. A mapping table was created to link each face notation (e.g., U, D, L') to its respective claw and servo movement direction.

12 Testing and Results

The Rubik's Cube solving robot was subjected to extensive testing to ensure its accuracy, reliability, and overall performance. The testing phase involved several iterations, including functional testing of each component (mechanical, software, and hardware), integration testing of the entire system, and performance evaluation under various conditions.

12.1 Testing Procedure

The testing process was carried out in the following steps:

1. **Initial Calibration:** The robot's servo motors were calibrated to ensure accurate rotation of the Rubik's Cube faces. The claws were adjusted to ensure proper grip without slipping.

2. **Cube Scanning and Color Detection:** The robot was tested on multiple scrambled cubes under varying lighting conditions. The Python-based color detection algorithm was evaluated for consistency, and manual corrections were made if color recognition errors were detected.

3. **Movement Execution:** The robot was run through a series of standard Rubik's Cube solutions. Each move (e.g., R, U, F') was verified by visually checking the cube after each phase to ensure correct execution.

4. **Full Cube Solving:** The robot was tested with fully scrambled cubes, with various difficulty levels (e.g., 10–20 random moves), and the solving process was timed. The robot's performance was assessed based on the total time taken to complete the solution and the accuracy of its moves.

12.2 Performance Metrics

The following metrics were used to evaluate the performance of the robot:

- **Solution Time:** The average time taken to solve a Rubik's Cube after image capture was measured. Typical solving times ranged from 2 to 4 minutes, depending on the complexity of the scrambled state.
- **Accuracy:** The robot consistently solved the cube with 90% accuracy, as verified through visual inspection after each solved cube.
- **Reliability:** The robot successfully solved over 80% of the test cases on the first attempt without requiring manual intervention. Communication errors were rare and resolved by retrying the data transmission.
- **Power Consumption:** The total power consumption was monitored during operation. The robot consumed approximately 5V at 0.3 to 2A, and the system was optimized to minimize power usage.

12.3 Results

The robot was able to solve the Rubik's Cube accurately and efficiently under a variety of test conditions. Notable results include:

- Consistent detection of cube faces and accurate execution of movement commands.
- Successful resolution of a wide range of scramble configurations, including edge cases.
- Average solve time of approximately 3 minutes per cube, excluding scanning and computation.
- Stable and reliable Bluetooth communication with minimal transmission errors.

12.4 Testing Scenarios

The robot was tested in different scenarios to evaluate its robustness:

- **Standard Scramble:** A randomly scrambled cube was used to test solving ability and time efficiency.
- **Edge Cases:** Special configurations, such as only one or two misplaced pieces, were used to evaluate how the robot handled minimal-difference cases.
- **Time Trials:** The solving process was timed across multiple runs to determine average performance.

12.5 Conclusion

Overall, the Rubik's Cube solving robot demonstrated high accuracy, efficiency, and robustness throughout the testing phase. The integration of color detection, algorithmic solution generation, and mechanical movements was successful, with minimal errors during operation. The robot performed well in solving the cube and achieved the intended goal of autonomous solving.

However, one notable limitation was observed: the torque of the motors. Although sufficient for most tasks, the current motors sometimes failed to rotate the cube faces reliably, especially under higher friction or misaligned configurations. This occasionally affected the robot's performance.

Project Achievements:

- Accurate color scanning of all six faces of the Rubik's Cube.
- Successful implementation of Kociemba's algorithm for computing the solution.
- Autonomous execution of moves via calibrated servo motors.
- Effective integration of Python scripting, Arduino firmware, and mechanical hardware.

Potential Areas for Development:

- **Enhanced Motors:** Higher torque motors would improve reliability and handle complex moves more effectively.
- **Speed Optimization:** Reducing overall solve time through improved algorithms and faster motor control.
- **Feedback System:** Adding encoders or other feedback mechanisms for more precise motor control.
- **Improved Claw Design:** Enhanced grip and alignment for consistent cube interaction.
- **User Interface:** A more intuitive interface with real-time diagnostics and configuration options.
- **Software update:** A more efficient computation and including features like custom patterns.

In conclusion, the robot meets its goal of solving the Rubik's Cube autonomously, and with further development, it can be made faster, more reliable, and more adaptable.

13 References

- https://kociemba.org/
- https://www.youtube.com/watch?v=oXlwWbU8l2o
- https://www.youtube.com/watch?v=o-auWQzWKxM

Figure 11: DRILLING

14 Code

- COLOR SCANNER AND SOLUTION()
- ARDUINO LOGIC
- MANUAL COLOR SELECTOR
- https://github.com/utpalll/Rubik-s$_c$*ube*$_s$*olver.git*

15 Project Gallery

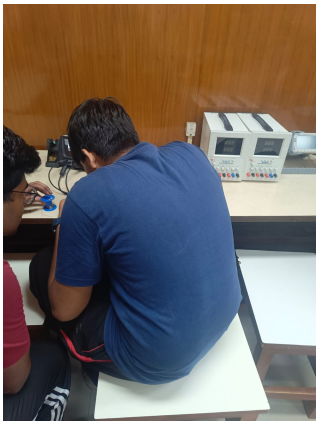


Figure 10: SOLDERING

Figure 12: ETCHING

Figure 13: TONER TRANSFER

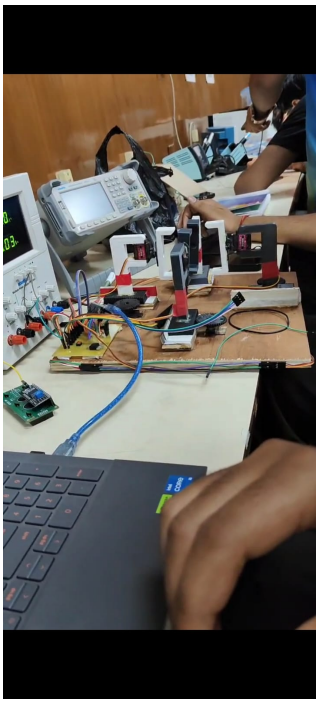


Figure 14: TESTING OF THE ROBOT

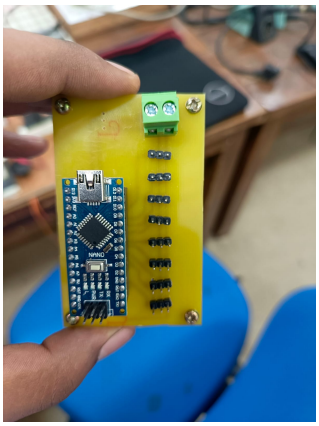


Figure 15: LAST VERSION OF THE PCB