Robotics and XR: Robotic Rubik's Cube Solver

Mohammad Jaber Hossain
mohahoss@student.uef.fi

Alvaro Flores-Romero
alvarofl@stud.ntnu.no

Austin Ryan English
aenglish@student.uef.fi

Akash Harijan
akashari@student.uef.fi

I. Introduction

The Rubik's cube has six internal degrees of freedom. The Rubik's cube may be scrambled by rotating each of its six faces. It has 27 cubelets that are joined together by a network of joints and springs. When all six of a Rubik's cube's faces are changed back to their original colors (the color of the center square), the cube is said to be solved. In figure 1, the flat image of the unscrambled cube with the solved orientation is shown. Due to the problem's design, which only has one best solution among the 43 quintillion other alternatives, it continues to rank among the best puzzle games [7]. The Rubik's Cube problem has captivated millions of people since the 1970s, and in recent years, a new trend of robots solving the cube has arisen [3]. The goal of this project was to look into the time and accuracy requirements of a robot that solves the Rubik's Cube. It could be quicker to solve such challenges as the Rubik's cube puzzle using the expertise of computer vision, robot kinematics and by developing algorithms, which might give a more efficient way to solve it than humans in fewer steps. A Field Programmable Gate Arrays (FPGA) chip is designed in [4] to solve a Rubik's Cube and to identify the color of the cubes with a CMOS camera, besides to build a hand like mechanical structure NXT motors and a Lego building blocks used that can push and spin the Rubik's Cube. Numerous techniques for both online and offline color identification of the Rubik's cube were described in [2], where scatter balance and extreme learning machine (SB-ELM), an offline technique, was discovered to have a successful way for effective segmentation of the colors. Automatic domain randomization (ADR) is a new algorithm that is introduced in [1] and to run this machine learning approach, a robot platform resembling a humanoid robot hand was constructed, besides domain randomization techniques [6, 9] were used for models that had only been trained in simulation and needed to transfer to the actual robot system . Thistlethwaite's method, Kociemba's algorithm, and Korf's algorithm are three well-known solutions to solve the Rubik's cube [10]. In [8], the study presents a method that uses Kociemba's algorithm combined with certain novel concepts and incremental advancements in some strategies to solve the problem for all possible combinations in 25 moves or less. The Kociemba's Algorithm will be used in this project to address the challenge.

II. Initial Concept Plan

At the conception stage, the project was intended to be developed in a default setup like the one shown in Figure 2. The results and the programming logic were created to only work with the designed platform. The objective was to implement the "Solving arm" in a mobile setup that could be transported easily, but the intricacies of it made it challenging. Calibration and illumination conditions (for the camera) were expected to be challenging but were included in the goals to guarantee it's functionality.

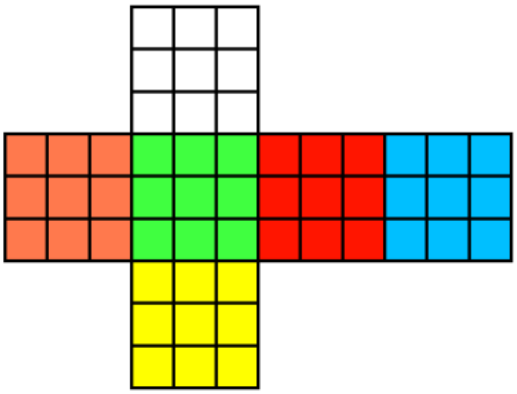

Fig. 1: Unscrambled cube in flat perspective.[7]

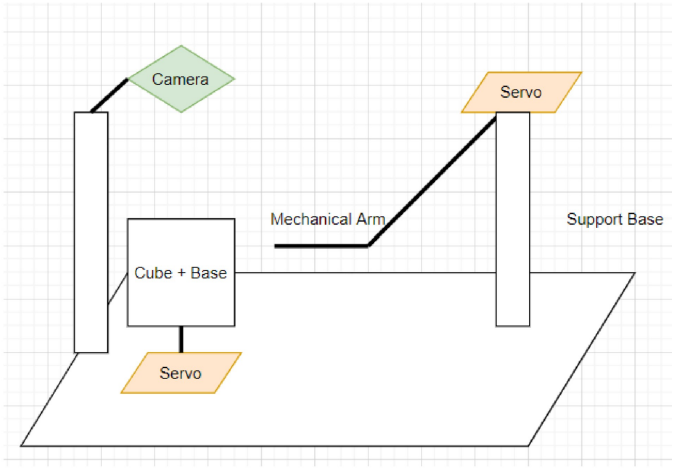

Fig. 2: Mechanical System

A. Initial Considered Challenges

The electro-mechanical system is to solve a 3x3 Rubik's Cube with no external help on the major movements or algorithm steps . The cube is to be pre-scrambled OFF the platform and placed in the base when it's ready to be solved. There was no speed requirement established as depending on

the complexity of the cube arrangement (after scrambling), the longer it could take to solve the present cube combination.

B. Technologies Considered

Realistically 3-degrees of motion (rotation around each axis) are needed to solve the problem optimally. This would imply that there's a mechanical locking system around the cube. This level of design implies 3D printing and more precise features as the complexity increases. One degree of freedom was dropped of the system and another one was changed from rotation to a push-motion. This increases by 3 the amount of time (and movements) needed to solve the cube but it also reduces the calibration complexity and the required component precision.

C. Required Knowledge for Initial Considerations

Certain knowledge was required for the expected functionality. The most important ones are listed below:

1) Machine/Deep Learning: Train the system into identifying the Rubik's cube presence. QR code could potentially be the alternative strategy. In the color identification with the camera images this is of particular interest.
2) Image Processing: To extract the information from the cube and update the virtual color faces based on the rotation of the faces. Working with different illumination conditions is important but this can also lead to non-homogeneous scenarios where we required not only post processing but also blur to reduce the image noise on low light conditions.
3) Servo Motor movement and Calibration: Servos are to be calibrated before the run to decrease the accumulated error when spinning the sides of the cube. The parallel offset between the rotation face and the rest should be kept at a minimum. An essential step that was implemented but 90° turns as they had more error than 180° ones.

D. Team Strategy

The workload was distributed on every member of the group such that each one of us was in charge of developing one of the previously mentioned areas (Machine learning, image processing, physical structure and Motor Movement). Communication between each of the "building blocks" was tested to ensure that the integration would work by the end for the final presentation. A general functionality diagram is included in Figure 3.

The programming language of choice was Python and C (Arduino-Based) as the platforms were involved into the implementation strategy). Some initial considerations and assumptions are listed down below:

1) Machine Learning and Image Processing will be done in Ubuntu OS using Python. Because of camera restrictions with the Raspberry Pi, Ubuntu on a virtual machine was used so the code could be translated to a Raspberry once a better camera was obtained (more on this later).
2) Servo Control will be done with Arduino IDE

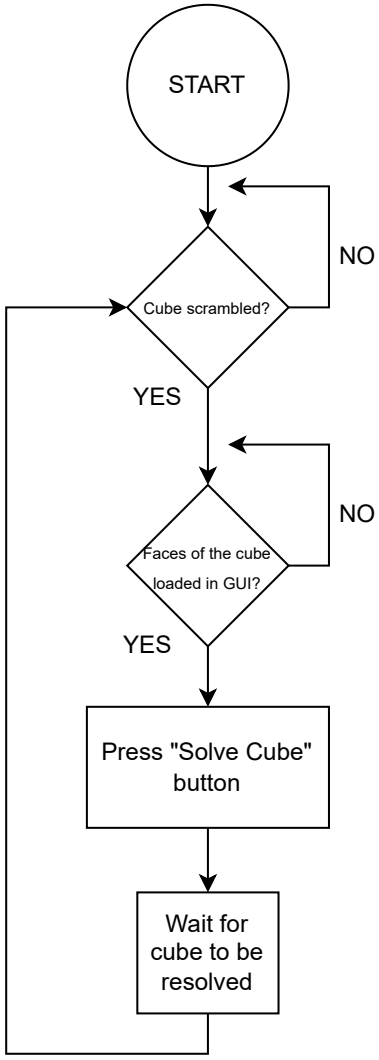

Fig. 3: Cube-solver logic diagram

3) Physical Construction and assembly will be done with "Popsicle wooden sticks" as it gives freedom for design with a good trade-off weight and resistance

Communication between Raspberry Pi and Arduino was done via Serial Port USB communication protocol to share directives of movement. Each of the main blocks will have its only version control decided by the team member (Github was suggested as it is widely known by every member of the team).

III. Final Technical Implementation

The final system is composed of several subsystems, described below and pictured in Figure 4.

1) Python Cube Solver
   - Computes solution steps from initial cube state using the Kociemba algorithm (Ubuntu Virtual Machine)
2) Mechanics Controller

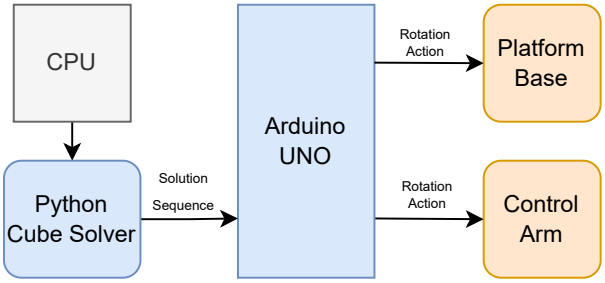

Fig. 4: Implementation System Diagram

- Arduino UNO to control the servo-motors with a PWM signal
- Receives solving steps of the cube over serial communication from the cube solver system and operates the platform base and control arm depending on each solution instruction step

3) Platform Base
   - Servo-motor controlled rotating platform which the Rubik's cube is cradled in
4) Control Arm
   - Servo-motor controlled arm to flipping and immobilizing the upper portion of the cube

A. Python Cube Solver

This subsystem is fairly concise and serves two roles:

1) Compute solution sequence
2) Transmit the solution to the Mechanics Controller subsystem.

Beginning with a manually set cube configuration, the subsystem performs two-phase solving, implemented in Python and described in detail by the author Herbert Kociemba[5]. Two-phase solving is inspired by the Thistlethwaite algorithm and computes two solution sequences, phase 1 to get from current state to a simplified state, and phase 2 to get from the simplified state to the solved state. The simplified state is a Rubik's cube position known as G1-Domino where the cube's degrees of freedom can be consolidated to a [3 3 2] structure rather than a [3 3 3] and a "Domino solution" calculated.

Once a solution is found in which phase 1 steps are minimized, the algorithm continues to search for solutions where although phase 1 may require more steps, phase 2 steps drop low enough to decrease the total number of steps required. Phase 1 will at most require 12 steps and phase 2 at most 18 steps.

The implementation by Herbert Kociemba outputs the solution string as a sequence of face and rotation codes. The solution sequence is then written out in simple characters with a start and end flag. This is sent to the Arduino over the machine's serial communication port to be read by the Mechanical Controller subsystem. The start and end character as "¡" and "¿" respectively and they were introduced to reduce the communication errors there could be.

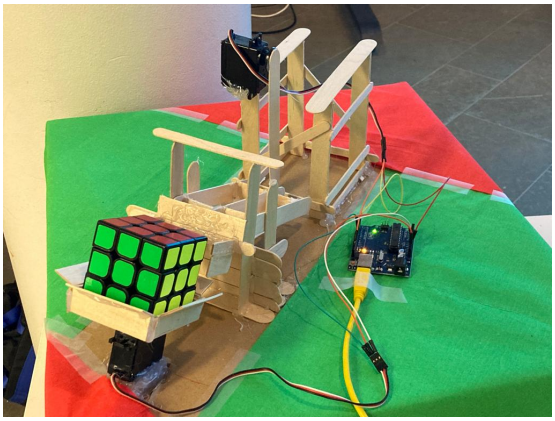

Fig. 5: Physical design of the Rubik's cube solver system.

B. Mechanical Controller Subsystem

The Mechanical Controller subsystem is built into the Arduino UNO micro-controller and contains all the functional logic for processing a cube solution string into operational outputs. It's operational outputs can be summarized as:

- Spin front face F n times
- Spin back face B n times
- Flip cube n times access face
- Rotate and flip cube n times to access face
- Spin left face L n times
- Spin right face R n times
- Spin upper face U n times
- Spin bottom face D n times

The two servo motors that control the platform base and the control arm each have preset position definitions that are tuned according to the implemented mechanical setup, and allow for the Mechanical Controller system to designate the operations as servo positional modes. A different setup would require to realign and recalculate the preset conditions to give the same results.

After initializing inputs and outputs, the system loops through the instructional solution sequence, applying a subsequence of mechanical operations on the cube per step and recording the resulting orientation of the cube faces. Recording the final cube orientation state after each step allows the subsequent step movements to be minimized as opposed to returning to the original position, then applying an orientation change for the next step. Due to the nature of the system, there's a limit on the optimization that could be done.

C. Platform Base

The Platform Base subsystem, along with the Control Arm, is constructed of birchwood, which provides a smooth surface for strong adhesive contact and durable structure for mechanical stress applied by the motors. It is fashioned as square shaped cradle for the Rubik's cube, mounted at a slight angle onto a servo motor. The servo motor can rotate the cradle about its tilted z axis as in Figure 6. The platform base serves two purposes:

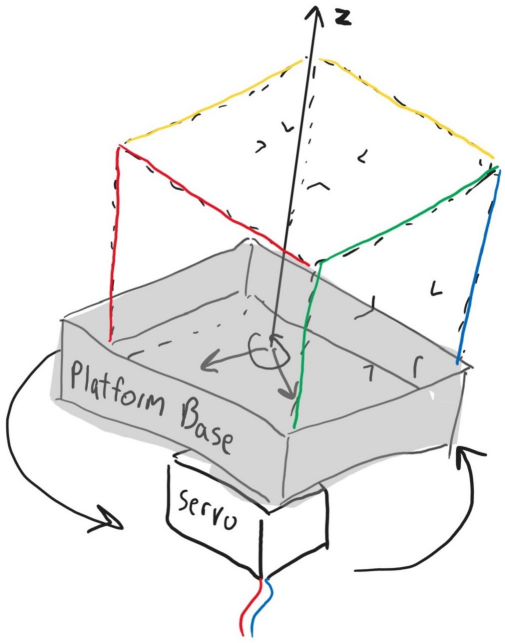

Fig. 6: Platform Base Structure

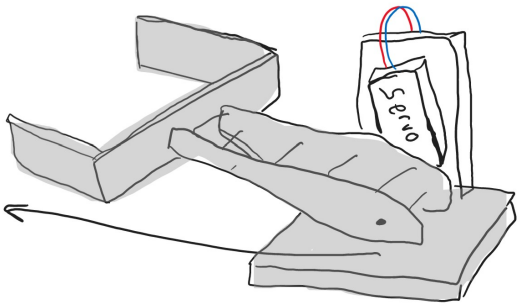

Fig. 7: Control Arm Structure

1) Hold cube securely
2) Rotate cube's lower face

The cradle is shallow which allows the platform base's rotation to be multi-functional when used in combination with the Control Arm.

D. Control Arm

The Control Arm, also constructed from birchwood, consists of a rigid claw and supporting arm which is thrust towards the cube on the rotation of a mounted servo motor (Figure 7). The control arm serves three main roles:

1) Flipping the cube
2) Holding the cube, causing a face to be turned as the base rotates
3) Releasing the cube, allowing free rotation of the full cube

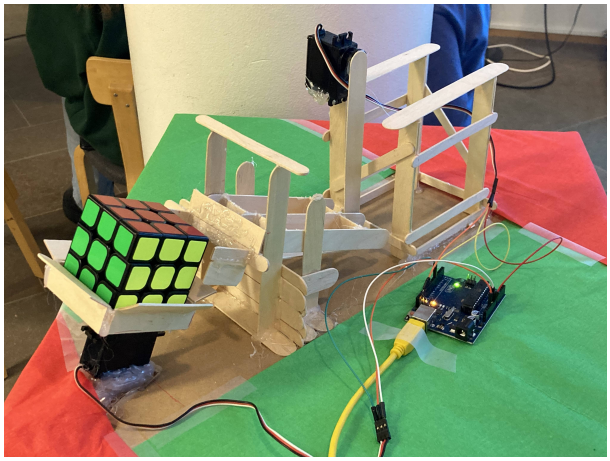

Fig. 8: Final System Build

E. Final System

The final physical system was demoed during the Robotics and XR exposition and can be seen pictured in Figure 8.

F. Initial Challenges

After moving forward with the project some challenges were presented. These required a couple of main modifications to the initial plan. The highlights are:

1) Raspberry Pi and it's proprietary camera PiCam were not used. The CPU architecture of the raspberry made it really hard to find compatible libraries that were able to do basic machine learning color identification. On top of this, the camera spectral sensitivity had a really difficult time differentiating orange from red. Because of this the project was run on an "Ubuntu Virtual-Machine" so it could be ported later to a raspberry when possible
2) Because the camera was discarded from setup itself, a GUI was included to allow the user to manually input the current faces of the cube.

Another important challenge was with the physical model's construction and tuning being a key one. If we compare our arm with the other class projects, they were already working with models that are either 3D printed or that have a physical pre-established configuration, but for our project, we have to deal with no already specified hardware (which by definition makes it harder). As a result, the arm must be built to its full extent in order to flip and rotate the cube. This requires first measuring the arm, then building it by trial and error.

Some of the following issues were resolved to some degree. The offset for rotation is the first. There is an offset caused by the device's inability to keep the cube firmly in place during the rotation since the servomotors are not powerful or accurate enough. After a few spins, this leads to cumulative inaccuracy that may be quickly fixed by adding extra revolutions in the opposite direction to make up for it. The installation of a camera color recognition system for the labels on each side of the cube is the second. As stated before, separating orange from red is a particularly difficult challenge that hasn't been

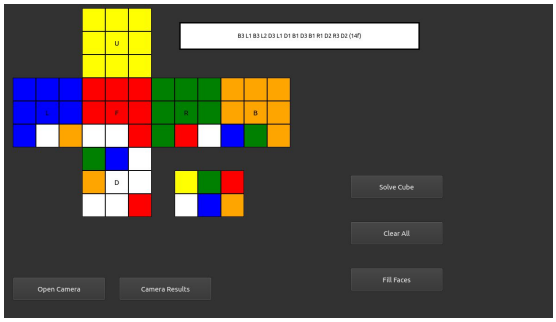

Fig. 9: A Simple Front-End to show the Rubik's cube state.

addressed due to the constantly changing lighting conditions and the camera's spectral sensitivity. In an effort to address this, we explored manually positioning the color recognition device in a "closed room" with controlled lighting rather than mounting it on a spinning base. In this manner, the camera may have a higher chance of seeing the differences in color. It may be determined that the camera is the major issue if the situation does not improve after doing this. As an alternative, a QR detector code could be used to determine when the cube has been placed on the platform so that the procedure may begin automatically (while keeping the color input manually).

IV. User Interfaces

As part of the two-phase solver library, a user interface is provided for displaying the Rubik's cube starting configuration (with the 6 faces laid out flat onto a 2D plane) and the subsequent solution steps visualized.

V. System Usage

There are some steps that a user has to follow to use our system to solve the Rubik's cube.

1) Run the python code to read the current state of Rubik's cube in Ubuntu operating system, it can be either in raspberry pi or in VMWare of Ubuntu. It will use camera of raspberry pi to detect the colors of cube faces and will create virtual color faces as shown in Figure 7. If there's errors with the prediction the faces can be fixed manually by clicking on the corresponding colors.
2) Once current state of Rubik's cube is known it will send those results to Arduino system that is attached to the wooden robotic arm and Arduino system will start performing the steps to be taken to solve the Rubik's cube.

A complete visual implementation of the usage can be visualized in the YouTube video in Section VII.

VI. Codebase

All the source code is available on GitHub: Initial compilation may take 30 minutes if the user wants to run it. This time is only for the first run. Please visit: https://github.com/MJHossainS/Rubik-s-cube-solver-using-Arduino-UNO

VII. Demo

A demo video of the system's operations can be found on YouTube with the following link: https://youtu.be/UytTCMy3AUE

VIII. Self-Evaluation

To add to the project's successful implementation, the following is a list of the parts implemented by each team member:

- Alvaro Flores-Romero: Arduino servo-motor logic to implement the Kociemba's algorithm result and read serial information with error correction. Calibration and manual step-by-step adjustment was required to offset error.
- Mohammad Jaber Hossain: We started by reviewing the relevant previous work done to develop a reliable system. This literature analysis offered us the assurance we needed to evaluate several approaches and choose the most effective one (Kociemba's algorithm) for our project. We spent most of our time working as a team on the physical system design, and then we had to test and debug the algorithm's codebase using Arduino to make it efficient.
- Austin R English: Testing integrated systems and evaluating failure points in the mechanical structure. Designing prototype layouts and helping interface python control module with mechanical control module.
- Akash Harijan: Worked on the failures of the algorithm as it couldn't differentiate the red and orange color. Also worked on the front-end of the system in python using PyQt5.

After consideration it is our belief that the team deserves an evaluation of 4.8 out of 5. There's a couple of reasons behind this:

1) Successful results: Results were achieved and even though some modifications were done (as with every project), the core goal was achieved. Foundations for later improvements were implemented.
2) The great group dynamic allowed to integrate a fundamentally complex system on the programming, mechanical and communication (serial) sub-systems.
3) This turned out to be a challenging project that worked with 0 pre-defined structures, making that, by itself, something challenging for most people
4) The final system incorporates electrical, vision, mechanical, and GUI systems.
5) Although thoroughly challenging and cross-disciplinary, we have seen some design points that could be done differently and perhaps result in a more production-ready product in future iterations.

References

[1] Ilge Akkaya et al. "Solving rubik's cube with a robot hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[2] Lin Feng et al. “Color Recognition for Rubik’s Cube Robot”. In: *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE. 2019, pp. 269–274.

[3] DANIEL GRÖNÅS and FREDRIK MAZUR. *Robotic Cuber: A Rubik s Cube solving robot*. 2020.

[4] Chun-Fei Hsu et al. “Intelligent Rubik’s Cube Solver”. In: *Asia Pacific Workshop on FPGA Applications*. 2013.

[5] Herbert Kociemba. *Two-Phase Algorithm Details*. URL: http://kociemba.org/math/imptwophase.htm.

[6] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[7] SP Rohith et al. “Autonomous Rubik’s cube solver bot”. In: *International Journal of Scientific Research and Engineering Development* 2.3 (2019), pp. 146–151.

[8] Tomas Rokicki. “Twenty-five moves suffice for Rubik’s cube”. In: *arXiv preprint arXiv:0803.3435* (2008).

[9] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[10] Ekta S Toshniwal and Yogesh Golhar. “Rubik’s Cube Solver: A Review”. In: *2019 9th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-19)*. IEEE. 2019, pp. 1–5.