Self-Solving Rubik's Cube

FINAL REPORT

Team 29

Dr. Zambreno

Taylor Burton (Systems Lead)
Jacob Campen (Hardware Lead)
Casey Cierzan (Materials Lead)
Joe Crowley (Testing Lead)
Annie Lee (Algorithms Lead)
Keegan Levings-Curry (Administrative Lead)
Luke Schoeberle (Software Design Lead)

sdmay20-29@iastate.edu
http://sdmay20-29.sd.ece.iastate.edu

Revised: April 25, 2020/Final Version

Executive Summary

Engineering Standards and Design Practices

- Follow IEEE standards
- Push early and push often
- Document every part of the process

Summary of Requirements

- Self-contained within the cube
- Easily scrambled by hand
- Solved within two minutes
- Lasts for three or more years
- Charges from an external source
- Fits within the typical senior design budget

Applicable Courses from the Iowa State University Curriculum

- ComS 309
- CprE 288
- ComS 228
- EE 224
- EE 311
- Phys 222

New Skills/Knowledge acquired that was not taught in courses

- Motors
- Arduino motor control software
- Rubik's cube algorithms
- 3D printing
- CAD tools

Table of Contents

List of figures and tables

1 Introduction

1.1 Acknowledgement

We would like to thank Dr. Joseph Zambreno, Lee Harker, and Boyd Lab for their support in this project. Dr. Zambreno is the project advisor, while Lee Harker has ordered the components for our project. Additionally, Boyd Lab has provided advice on motors, which was very helpful in the early stages of the project.

1.2 Problem and Project Statement

Our project is to create a self-solving Rubik's cube by adding components within a cube. After the user scrambles the cube by hand, the cube must solve itself within two minutes.

Dr. Joseph Zambreno introduced this project after seeing similar projects online around the world. This cube will be designed to inspire prospective students to pursue Electrical and Computer Engineering (ECpE). By witnessing the potential of a four-year undergraduate ECpE degree, they will be more motivated to study ECpE at Iowa State University (ISU). In this way, the cube is primarily intended for marketing purposes at ISU, so it is not intended for mass production.

1.3 Operational Environment

The finished cube must operate indoors on a fixed display table at room temperature.

1.4 Requirements

Here is a list of requirements:

- Can be scrambled by hand without requiring much force from the user
- Contains no external components (e.g. cameras, robot arms) except for charging devices
- Reliably solves the cube within two minutes
- Durable enough to last for at least three years
- The entire budget cannot exceed $750
- The cube's side length must be in the range [5.7, 18] (in cm)

1.5 Intended Users and Uses

Our cube's primary users are ISU tour guides and prospective families.

The tour guides will maintain the cube, which includes cleaning, charging, and lubing.

The prospective families, which may include high school students, parents, and small children, will scramble the cube.

1.6 Assumptions and Limitations

Here is a list of assumptions:

- The cube will always begin in a solved state
- Users will not need to indicate that they have finished scrambling the cube
- Users will only make turns in 90°-increments (e.g. 0°, 90°, or 180°)

Here is a list of limitations:

- The cube must be solved within two minutes
- The cost for the cube cannot exceed $750

- The cube's side length must be in the range [5.7, 18] (in cm)

1.7 Expected End Product and Deliverables

Our end product was originally intended to be a self-contained, self-solving Rubik's cube, but recently we have redefined our deliverables significantly due to the COVID-19 pandemic. We will provide the materials for the physical prototype, the completed solving algorithms, the untested system code, our CAD models, and documentation to Dr. Zambreno by 4/26/2020. Ultimately, we hope that our work will be used in future iterations of this project.

2 Specifications and Analysis

2.1 Proposed Design

Our cube's side length must be between 5.7 cm and 18 cm, and the cube will have a standard color scheme and layout. Each face will be independently controlled by electric motors. Since the cube must be easily scrambled by hand, we will be using hybrid stepper motors to ensure that the cube can be scrambled without damaging the motors. To sense face movement, two Hall Effect sensors under each face will be paired with magnets within each face. When a turn occurs, the sensors will transmit meaningful analog voltages to the microcontroller, which will be interpreted by embedded microcontroller software.

The software will adhere to IEEE Standards, such as the IEEE unit testing standard (IEEE 1008-1987). The software will control the motors, ensuring that the motors are disengaged during the scrambling process. It will also track the cube's physical rotations during the scramble, which is used to simulate the rotations in software. After a period of rotational inactivity, the software will initiate a solving algorithm based on the simulated cube's state. In doing so, it will send turn instructions to the motors until the cube is solved. By using algorithms that require few moves, the software will solve the cube in less than two minutes.

Because of the importance of user interaction in our project, our nonfunctional design choices will maximize the cube's interactivity. Specifically, we will ensure that the motors will not significantly impact the physical scrambling process. Similarly, when we finalize a CAD design, we will aim to make the rotations as fluid as possible. Additionally, during the solving process, we will attempt to minimize the solving area to simplify user interactions with the cube. In this way, our project is driven by our user's needs, which allows them to easily use and appreciate our work.

2.2 Design Analysis

As of 10/1, we have researched previous solutions, discussed multiple designs, and started our first prototype. We are currently using a large Rubik's cube for prototyping, and we have tested a few small magnets and Hall Effect sensors. We noticed that we may need different magnets since it is difficult to use the small magnets. Ultimately, we intend to use four magnets per face to detect rotations during the scrambling process.

As of 11/1, we have started the CAD model and have tested multiple motors, which has helped us define our system's physical limitations. In doing so, we tested the torque and power needed to turn the cube. We have evaluated the advantages and disadvantages of a smaller cube in this scenario, but we have not yet decided on a final design. On the software side, we have also implemented rotation simulation algorithms, but we still need to thoroughly test those algorithms.

As of 12/1, we have secured an appropriate motor, and we have decided on a final cube size of 11 x 11 x 11 cm^3. We have also finished the majority of the CAD design, and we have thoroughly tested our rotation simulation algorithms. We have also started to write the hardware-software integration code, so we are now prepared to implement the first prototype.

In the second semester, we constructed the housing sphere and inserted the motors. However, we were unable to test this system fully due to the COVID-19 pandemic. During the rest of the semester, we finished the solving algorithms, wrote some untested system code, and completed the documentation for our project.

2.3 Development Process

For our project, we will be using a modified Agile workflow. We begin with the typical waterfall steps of high-level requirements specifications and design. However, we then follow iterative Agile processes, which will incrementally push us towards our final solution.

This is the best development process for our project because it allows us to create prototypes quickly. Because of spatial and monetary concerns, we will need to test many different designs before settling on our final product. Additionally, by frequently displaying our progress to our client, we will obtain a better understanding of our client's needs and desires. As a result, we will ultimately create a better final product by following Agile practices.

For more details about our development process, see Figure 1 below.

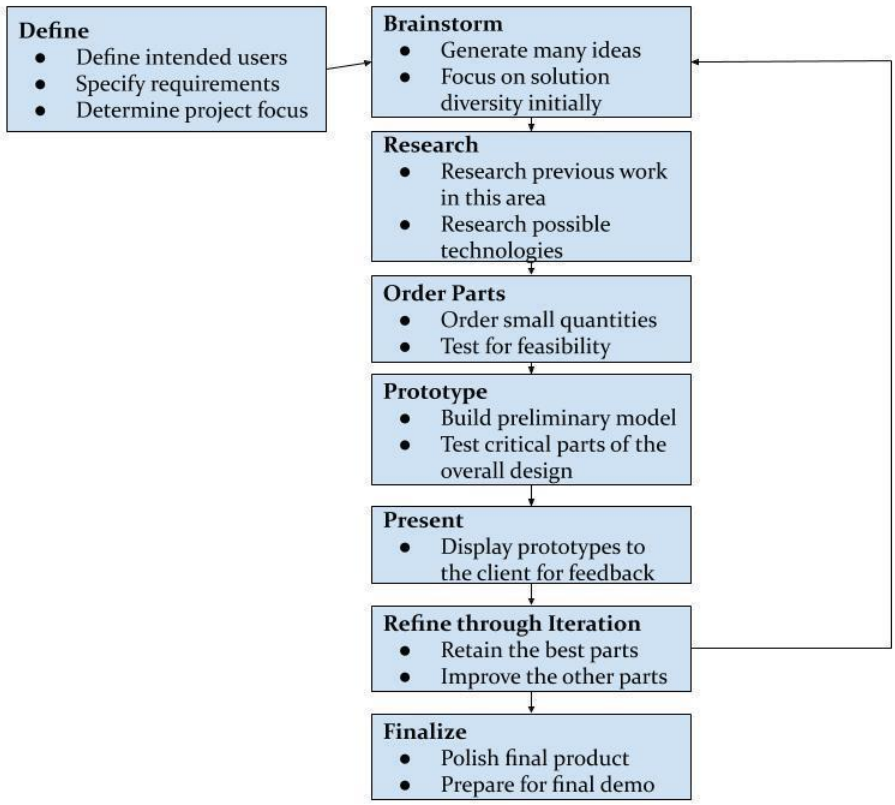

Figure 1: Development Process

2.4 Design Plan

We will initially design a large Rubik's cube (18 x 18 x 18 cm^3) that meets the basic requirements. Although this large cube satisfies the requirements, it will not be a perfect solution because larger cubes are more difficult to scramble. Once we have a complete larger cube, we will make a smaller cube by minimizing the size of the internal components as much as possible.

We will design a large cube first because it is easier and cheaper to design a large cube initially. Because of the larger cube's size, the internal space for our components will be much larger, so it will be much easier to find appropriate components for a larger cube. Additionally, miniature electrical components are exponentially more expensive than larger components, so it will be much cheaper to perform our initial testing on larger components. In this way, by designing a larger cube initially, we will limit the amount of time and money needed for this project.

Once we have finished an adequate large prototype, we will use our acquired knowledge to miniaturize the cube as much as possible. Our client and our team have agreed that a medium-sized cube (11 x 11 x 11 cm^3) would be much easier to scramble, so we plan to create a medium-sized cube for our final product.

3 Statement of Work

3.1 Previous Work and Literature

3.1.1 Similar Work

Two types of self-solving Rubik's cubes have been created in the past. The first type requires many external components, while the second cube uses only internal components.

Our client requires that we create an internal solver.

3.1.1.1 External Solvers

The first self-solving cube type is a standard cube augmented with large external components. The cube operates within a custom-made stand that is connected to motors, cameras, and a desktop application. Generally, the motors turn the cube, the cameras determine the state of the cube, and the desktop application runs the solving algorithm.

See the following link for an example: Fast External Cube.

3.1.1.2 Internal Solvers

The second self-solving cube type is a custom cube augmented with tiny internal components. The cube is composed of 3D-printed parts, and it contains motors and microcontrollers within the cube's hollow center. Generally, the 3D-printed parts increase the size within the cube, while the motors turn the cube and the microcontroller runs the solving algorithm.

See the following link for an example: Self-Rotating Cube.

3.1.2 Solving algorithms

Many Rubik's cube solving algorithms exist, and they can be easily found online. In our case, our primary goals are algorithm efficiency and simplicity. However, these goals are often at odds in the solving algorithms, which complicates this issue overall. Ultimately, we will choose the fastest algorithm that we can fully implement.

See Table 1 for a comparison of a few possible algorithms:

Algorithm	Advantages	Disadvantages
Petrus	• Very efficient • Popular in speed solving • Uses the fewest moves	• Difficult to understand • Difficult to implement • Not designed for computers
Thistlethwaite	• Reasonably efficient • Designed for computers	• Not extremely efficient • Difficult to understand
Old Pochmann	• Simple to implement • Requires simple memorization	• Requires many moves • Ineffective for speed solving

Table 1: Solving Algorithms

3.1.3 Other resources

Additionally, we discovered a few other resources that may be useful during our project.

One article describes their design of an internal self-solving Rubik's Cube (Human Controller 2018). It first illustrates how they created a large prototype before creating the standard-sized final cube. Additionally, it displays the additional internal components inside the cube. Overall, this article is very useful for us since we intend to create a very similar product.

See the following link for more information: Design of self-rotating cube.

We also found an open-source Rubik's cube applet ("Rubik's Cube Java Applet program"). This application provides a Rubik's cube simulator that allows us to visualize the cube easily, which is very helpful for testing. The applet also provides a few solving algorithms implemented in Java, which are very helpful examples of solving algorithms. Thus, this Java applet is an extremely useful resource for us. See the following link for more information: Open Source Java Applet.

3.2 Technology Considerations

See Table 2 for a comparison of the previous approaches:

Previous Approach	Advantages	Disadvantages
External Solvers	• Identifies cube state with cameras • Powered with external motors • No space concerns • Solves the cube very quickly • Uses mostly standard cubes	• Not interactive • Requires external setup • Requires machine learning knowledge • Not portable
Internal Solvers	• Very intriguing for the audience • Portable • Contains no external components • Solves the cube reasonably fast • Easily turned by hand	• Limited identification of cube state • Expensive • Many space concerns • Requires a custom cube

Table 2: Discussion of Previous Approaches

In our case, our client desires an internal solver, which limits our possible solutions. However, even with these limitations, there are many technology considerations, such as motor type, state identification, and system integration.

By investigating different motor types, we determined that a stepper motor is best for our cube. Since the cube must be manually turned during the scrambling process, the motors cannot damage other components during these turns, which is a problem for servo motors. For servo motors, gearboxes ensure that manual turns do not damage its internal components, but gearboxes are typically large and difficult to handle. As a result, although stepper motors are generally less powerful than servo motors, we decided to use stepper motors because eliminating the gearbox provides more benefits than improving turn performance. Consequently, we have experimented with stepper motors, ultimately choosing motors that can move the cube easily without damaging the internal components during manual turns.

We will use Hall Effect sensors to determine the state of the cube. These sensors will use magnets embedded in the cube's face to sense magnetic field changes, which will transmit the cube's rotations to the microcontroller via analog voltage signals. voltages. This will ultimately allow the microcontroller to simulate the cube's rotation, assuming that the cube starts in a solved state. We will use at least two sensors on each face to consistently detect rotations, which results in a total of at least twelve Hall Effect Sensors.

We will control the cube's operations with a Teensy 4.0 microcontroller, which was primarily selected for its small size and integration with Arduino libraries. This microcontroller has ample ports and uses a 3.3V power supply, which is ideal for our application. Ultimately, the Teensy will allow us to control the motors, receive rotation signals, and implement the solving algorithm within the cube, which is critical for system integration.

3.3 Task Decomposition

See Figure 2 for an overview of the tasks in this project.

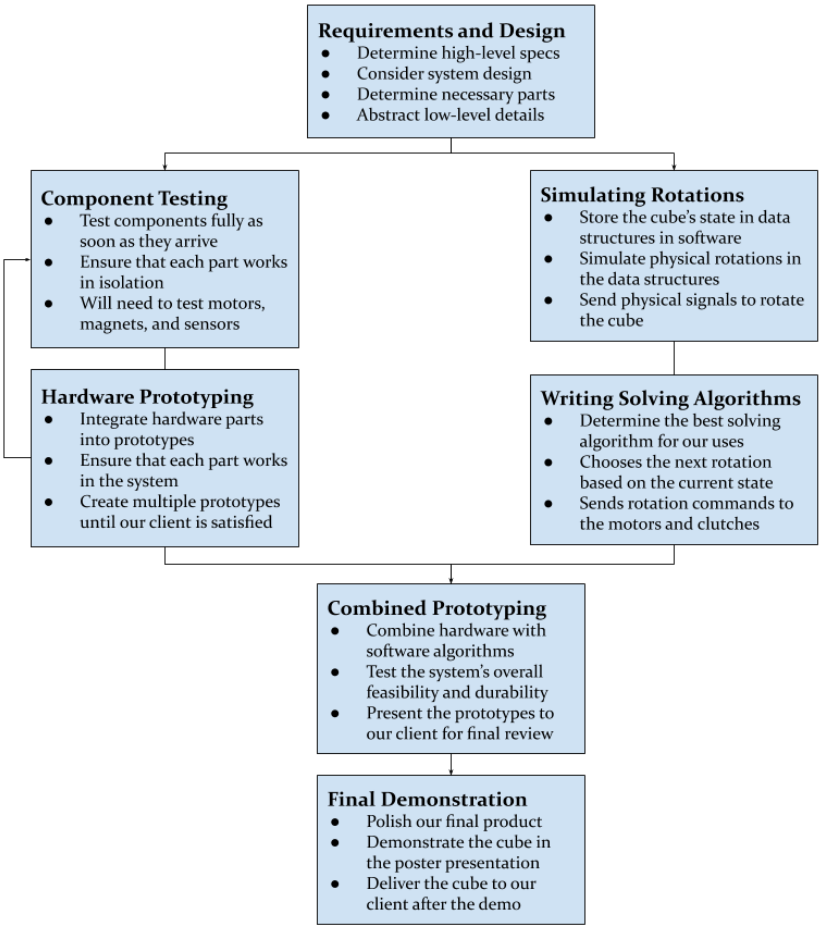

Figure 2: Task Decomposition

3.4 Possible Risks And Risk Management

Our project is potentially risky because of size concerns and our limited budget.

Since we need to create an internal solver, we will have very limited space for the internal hardware components, which introduces many possible issues. We may need to test multiple components because the magnets and other electrical components may interact within the cube's small center. Additionally, the space restrictions will greatly increase the costs of our electrical components, which may ultimately exceed our budget.

We will manage these risks in two ways. We will carefully consider the interactions between our electrical components before we physically test them, which will mitigate the risk of interactions within the cube. We will also manage our budget by meticulously tracking our purchases and buying cheap components as much as possible.

3.5 Project Proposed Milestones and Evaluation Criteria

To ensure timely project completion, we have established milestones for this project. These milestones currently include Solving Algorithm Implementation and Hardware Prototype Completion, which are defined in the following subsections.

3.5.1 Solving Algorithm Implementation

This milestone is satisfied when our solving algorithm is fully functional in software. To achieve this milestone, we must develop and test the solving algorithms with simulated rotations. We also must send the correct hardware signals during this process.

We will test this milestone by manual testing and with the assistance of the open-source Java applet described above in section 3.1.3. Initially, we will manually ensure that the simulated rotations are functional, focusing on the simulated cube's new state and the signals for the motors. Later, we will ensure that the algorithms work correctly within the Java simulator.

3.5.2 Hardware Prototype Completion

This milestone is satisfied when our hardware prototype can consistently rotate the cube without damaging internal components. To achieve this milestone, we must first produce functional components in isolation before testing the full prototype. Later, we must design and implement a reliable system for performing rotations on the cube.

We will test this milestone by manual testing. Initially, we will ensure that each component is functional individually. Finally, we will test that the system can turn the cube as desired without damaging the insides of the cube. In doing so, we will specifically check for tangling wires by performing the same turn repeatedly.

3.6 Project Tracking Procedures

To ensure timely project completion by next May, we will follow the schedule described below in section 4.1 as closely as possible.

If we miss any deadlines, we will immediately schedule an emergency team meeting to resolve the issue. Ideally, this will never occur, but our team is prepared to attend this meeting if necessary.

Due to COVID-19, we ultimately missed many deadlines, but we have included our original deadlines to illustrate our design process.

3.7 Expected Results and Validation

Our desired outcome was intended to be a self-contained, self-solving Rubik's cube. See Sections 1 and 2 for more details about the final outcome.

If users can consistently scramble the cube, let it lie on a flat surface, and watch the cube solve itself within two minutes, our solution will be fully functional.

4 Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

See Table 3 below for our project timeline. Week 1 denotes the first week of the Fall 2019 semester, and then Week 2 denotes the following week (and so on). Note that this timeline is not rigid, so it may be modified slightly in 2020 if unexpected complications arise.

Schedule	Tasks
Week 1, 8/26/-9/1 (2019)	• Attend first class • Study projects • Choose favorite projects
Week 2, 9/2-9/8	• Form teams • Discuss project requirements • Determine initial meeting times
Week 3, 9/9-9/15	• Assign roles • Research solving algorithms • Research motors and sensors
Week 4, 9/16-9/22	• Acquire sensors and magnets • Test the magnets and sensors • Obtain a large cube
Week 5, 9/23-9/29	• Deconstruct the large cube • Evaluate the large cube's feasibility • Order a stepper motor for testing
Week 6, 9/30-10/6	• Begin rotation simulation algorithms • Research alternatives to clutches • Evaluate feasibility of the large cube
Week 7, 10/7-10/13	• Acquire and test the stepper motors • Improve rotation simulation algorithms • Determine power requirements
Week 8, 10/14-10/20	• Complete software rotation algorithms • Begin the design of the 3D CAD model • Determine requirements for motors
Week 9, 10/21-10/27	• Test software rotation algorithms • 3D-print a housing hemisphere • Order stronger, geared motors
Week 10, 10/28-11/3	• Improve rotation algorithm testing • Discuss smaller cube sizes • Discuss future software work
Week 11, 11/4-11/10	• Finalize rotation code testing

	• Acquire and test new geared motors • Begin system CAD model
Week 12, 11/11-11/17	• Begin hardware-software integration • Order a stronger ungeared motor • Discuss cube size with our client
Week 13, 11/18-11/24	• Write Arduino code • Discuss final design with our client • Improve system CAD model
Week 14, 11/25-12/1	Thanksgiving Break
Week 15, 12/2-12/8	• Improve rotation code visualization • Acquire and test new ungeared motors • Finalize our design document
Week 16, 12/9-12/15	• Finalize final presentation slides • Attend final presentation • Finish final reflection
Week 17, 12/16-12/22	Finals week, start of winter break
Week 18-19, 1/13-1/26 (2020)	• Construct hardware prototype • Finish hardware-software integration • Start solving algorithms
Week 20-22, 1/27-2/16	• Test hardware prototype • Test hardware-software integration • Finish and test solving algorithms
Week 23-25, 2/17-3/8	• Finish CAD design for smaller cube • Test entire prototype • Start final cube
Week 26-28, 3/9-3/29*	• Optimize solving algorithms for speed • Consider solving space concerns • Finish final cube
Week 29-31, 3/30-4/19*	• Thoroughly test final cube • Present the cube to our client • Tweak the final product
Week 32- End of semester, 4/20-5/5*	• Prepare final poster and presentation • Present the final product to our client • Participate in the poster presentation

*Ultimately, most of these hardware deadlines were not achieved due to COVID-19.

Table 3: Project Timeline

4.2 Feasibility Assessment

Ultimately, creating a self-solving Rubik's cube will be challenging, but it is certainly possible for our team of electrical and computer engineers because we already have most of the skills needed to solve this project. For instance, we know how to use microcontrollers and Hall effect sensors. Overall, we will face three major challenges during the project: size constraints, motor systems, and component costs.

First, since we are creating an internal solver, we will need to place every internal component inside the cube's tiny center. Consequently, it will be very difficult to fit these components inside the cube without skillful 3D modeling, which is not our team's strength. To alleviate these size constraints, we have decided to create a larger prototype to test our ideas, but size remains a large challenge in our project.

Second, since we need to turn the cube programmatically, we will need to use motors systems inside the cube, which is a challenge since our team is unfamiliar with motors overall. Consequently, we will need to do extra research and testing with motors to determine the best options for our cube, which will be time-consuming and potentially expensive. In this way, choosing the correct motors will be difficult since we are not mechanical engineers.

Lastly, component costs will be a challenge due to the size constraints and motor experiments. In general, miniature electrical components are exponentially more expensive than larger components, so component costs will become a large concern if we decrease the size of our cube. Additionally, motors are fairly expensive, so our additional motor testing will become expensive over time if we are not careful. Consequently, managing our budget will be difficult in this project.

4.3 Personnel Effort Requirements

See Table 3 below for a detailed task breakdown of our project.

These tasks are roughly based on the tasks described in Figure 2 in Sect 3.3, but they are broken down further to assign personnel to these tasks. Additionally, keep in mind that the tasks are not ordered linearly because they may be repeated during the project. For instance, we plan on creating at least two self-solving cubes, so we will perform the "Test self-solving cube" task at least twice before we finish the project.

Task	**Projected Effort**
Determine project scope	**Entire team**: ● Discuss project scope with our client ● Determine acceptable solving time, cube size, and other requirements
Create design	**Entire team**: ● Discuss high-level ideas for our project ● Choose the best course of action for satisfying our client and creating a good final product
Evaluate design	**Joe, Taylor**: ● Consider other possible design improvements ● Ensure that the design satisfies the original requirements
Research materials	**Keegan, Casey, Jacob, Taylor**:

	• Research batteries for optimal size-to-current ratios • Research motors for optimal size-to-cost and size-to-torque ratios
Obtain materials	**Casey**: • Order parts from ETG when necessary • Manage our budget carefully
Research algorithms	**Joe, Annie, Luke**: • Research solving algorithms online • Choose the best algorithm for our cube
Implement algorithms	**Joe, Annie, Luke**: • Write C code for performing rotations and solving algorithms • Write code that integrates with our hardware
Construct self-solving cube*	**Jacob, Taylor**: • Create a 3D model of the system • Combine components into a single physical system based on the CAD model
Test self-solving cube*	**Keegan, Jacob, Taylor, Joe**: • Test requirements of the electrical components • Test algorithms in the physical system • Assess the system's reliability
Finalize our project*	**Entire team**: • Tweak the prototype to produce the final cube • Present the final cube to our client

*Ultimately, most of these hardware tasks were never performed due to COVID-19.

Table 4: Effort Requirements

4.4 Other Resource Requirements

We are currently using the following parts:

- 1 18x18x18 Rubik's cube
- 6 stepper motors
- 12 Hall Effect Sensors
- 18 magnets
- 1 Teensy 4.0 microcontroller
- 2 3D-printed hemispheres
- 2 rechargeable batteries
- 1 battery charging port

4.5 Financial Requirements

The project must fit within the typical Senior Design budget of $750.

5 Testing and Implementation

5.1 Interface Specifications

Here are brief specifications for the important physical components of our system:

Power Source:

The power source will be placed near the cube's center, and it will provide power to the Hall Effect sensors, microcontroller, and the motor control circuit. The power source will consist of the rechargeable battery of minimum size that can power all of our components simultaneously.

Magnets:

Two magnets will be placed on the outside of each face's center cube, and they will move with the rotated face during cube rotations. The magnets are needed by the Hall Effect sensors to transmit the cube's rotations to the microcontroller.

Hall Effect Sensors:

Two Hall Effect sensors will be placed near the inside of each face's center cube, and they will detect the magnets' position during cube rotations. The sensors will then transmit meaningful voltages to the microcontroller that indicate the magnets' new position. These analog signals will be used by the microcontroller to simulate the rotations in software.

Microcontroller:

The microcontroller will be placed near the cube's center, and it will control our cube's operations overall. Initially, it will receive analog signals from the Hall Effect sensors during the physical scrambling process, and it will use those signals to simulate the rotations in software.

Once the cube is at rest for ten seconds, the microcontroller will run a solving algorithm to obtain a list of rotations to solve the cube. This list will be used by the motor control circuit to send signals to the motors to solve the cube.

Motor Control Circuit:

The motor control circuit will be placed near the cube's center, and it will control our cube's six motors. It will receive the rotation list from the microcontroller, and it will use that list to activate and deactivate the motors as necessary to solve the cube.

Stepper Motors:

Six stepper motors will be connected to each center cube, and they will physically turn the faces to solve the cube. The motors will be controlled by the motor control circuit.

5.2 Hardware and Software For Testing

Here is a list of our current hardware and software for testing:

Temporary Motor Control Circuit:

We have been using a large motor control circuit to independently test our motors. This circuit is essentially a large version of our cube's final motor control circuit, so we will be using a similar circuit in a PCB in our final product.

Open-Source Java Applet:

We have been using the open-source Rubik's Cube applet introduced in Section 3.3 to independently test solving algorithms. This applet allows us to easily compare solving algorithms in multiple ways, so it helps us to choose the optimal solving algorithm for our final solution.

5.3 Functional Testing

In our project, we will perform unit testing, integration testing, and system testing, as described in the following sections.

5.3.1 Unit Testing

We will test the components from Section 5.1 as follows:

Power Source:

To test the power source individually, we will verify that the power source produces the correct voltage and current on a breadboard. This ensures that the power source will properly power the other components in the system.

Magnets:

To test the magnets individually, we will verify that they react properly when placed near other magnets. This ensures that the magnets properly create a magnetic field for our Hall Effect sensors. While we perform these simple tests, we will also empirically determine the strength of the magnets, which will be useful later during higher-level testing.

Hall Effect Sensors and Magnets:

To test the Hall Effect sensors individually, we will verify that their output voltage reacts properly when placed near magnets. This ensures that they transmit the proper rotation information to the microcontroller. In doing so, we will empirically determine the necessary distance between the sensors and the magnets, which will be critical when we construct the final system. We will also measure the typical analog voltages of the Hall Effect sensors with our magnets to improve the accuracy of our microcontroller's rotation detection algorithm. While we perform these tests, we will measure the power needed to operate the Hall Effect sensors to determine the necessary power source for our cube.

Microcontroller:

To test the microcontroller individually, we will test the rotation detection algorithms and solving algorithms in software. This ensures that the microcontroller will properly detect rotations from Hall Effect sensor data and send proper rotation commands to the motor control circuit. While we perform these tests, we will simulate the physical inputs and outputs as accurately as possible to simplify the integration process.

Motor Control Circuit and Stepper Motors:

To test the motor control circuit and the stepper motors individually, we will verify that the motors spin properly with the motor control circuit. This ensures that they will properly rotate the cube during the solving process. While we perform these tests, we will measure the power needed to physically perform rotations to determine the necessary power source for our cube.

5.3.2 Integration Testing

We will test the integration of the components above as follows:

Rotation Sensing Integration:

To test the integration of the rotation sensing components, we will combine the magnets, Hall Effect sensors, and microcontroller into a rotation sensing subsystem for testing. Next, we will verify that the microcontroller properly uses the signals from the Hall Effect sensors to record the rotations of the cube. This ensures that the microcontroller will properly record the state of the cube before it attempts to solve

the cube. While we perform these simple tests, we will also adjust the code in the microcontroller to account for the interaction between multiple magnets and sensors.

Motor Integration:

To test the integration of the motor components, we will combine the motor control circuit, stepper motors, and microcontroller into a motor subsystem for testing. Next, we will verify that the microcontroller properly sends rotation signals to the motor control circuit to perform the physical rotations for solving the cube. This ensures that the motors will properly solve the cube based on instructions from the microcontroller.

5.3.3 System Testing

To test the entire system, we will combine the rotation sensing subsystem and the motor subsystem into the full system. Next, we will verify that our requirements are met for our system; specifically, we will check for the following requirements:

- Easily scrambled by hand
- Reliably finishes the solving process within two minutes
- Does not damage the internal components during the solving process
- Successfully solves the cube without needing additional power
- Can be charged reliably by replacing the rechargeable battery

5.4 Non-Functional Testing

We will test the following nonfunctional requirements as follows:

Durability:

To test the physical durability of our cube, we will perform rotational stress tests on the cube, and we will also measure the cube's battery life in each run. Regarding the rotational stress tests, we will turn the cube many times forcefully during the scrambling process to verify that the cube can handle forceful scrambling. Additionally, by measuring the battery life of the cube, we will ensure that the battery will last for at least one entire run.

Performance:

To test the performance of our cube, we will measure the solving time in each run to ensure that that the cube always solves itself within 2 minutes. In doing so, we will tweak the algorithms if necessary to ensure that the cube solves itself within the required time period.

Reliability:

To test the reliability of our cube, we will test our system many times to ensure that the cube always solves itself properly within the required time period. In doing so, we will also verify that the performance and durability of the cube are acceptable in each run.

Usability:

To test the usability of our cube, we will allow Dr. Zambreno's son to test our final product, and we will also test that our cube can be easily charged in standard outlets. This will ensure that the cube can be easily handled and charged safely by our typical users.

5.5 Testing Process

For our project, we will be using a testing process that is similar to our development process. We begin with high-level test planning, which requires us to review requirements and establish high-level testing goals. We then follow iterative testing processes, which will incrementally push us towards our final product.

This is the best testing process for our project because it allows us to dynamically test our prototypes efficiently. Because of spatial and monetary concerns, we will need to perform many small tests before settling on our final product. Additionally, by frequently displaying our tests to our client, we will improve our understanding of our client's needs and desires. As a result, we will ultimately create a better final product by following iterative testing practices.

For more details about our testing process, see Figure 3 below.

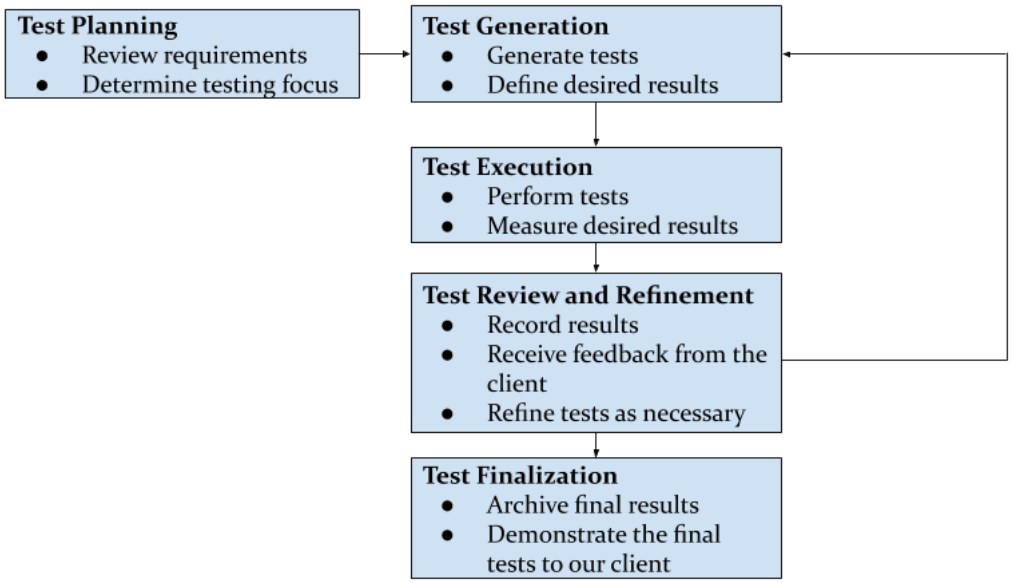

Figure 3: Testing Process

5.6 Test Results

At this point, we have successfully completed most of the unit testing process. Specifically, we have tested the magnets, sensors, motor control circuit, stepper motors, and some microcontroller code, but we have not tested most of the microcontroller's code. Due to COVID-19, we were unable to perform further testing of the microcontroller software.

Through our testing, we learned that powering our internal components will require large amounts of current, which will greatly increase the size of our power source. Additionally, we learned that the magnets have negligible interaction in our proposed design, which will simplify integration in the future.

In the future, we will face major issues with the power supply because of the necessary output current. Initially, since this output current requirement will greatly increase the battery's size, it will be difficult to fit the power supply within the cube. Additionally, we may not be able to perform rotations as quickly as possible because of the size constraints, which will complicate our efforts to finish the solving process within two minutes.

These results should guide future groups during the implementation of our design.

6 Closing Material

6.1 Conclusion

In conclusion, we have completed the initial design, and we have made good progress on our first prototype. Currently, we have designed a 3D model and have obtained and tested the majority of the individual units, but we have not yet started the integration process. Ultimately, we were unable to integrate our project due to COVID-19.

Ultimately, we plan to create a smaller cube with a side length of 11 cm. We will be 3D-printing a custom cube that contains our electrical components within a housing sphere. In doing so, we will minimize the cube's size as much as possible. At the end of the spring semester, we will present the components of the final cube to our client for final review.

Based on our initial testing, our solution is optimal because it minimizes the cube's size without exceeding our budget. If we used servo motors, we would need to include clutches, which would greatly increase the cube's size. The current stepper motors are small, durable, and fairly cheap, so we can easily afford six motors. In this way, our current design is the best option for this project.

6.2 References

Controller, Human. "Self Solving Rubik's Cube." *YouTube*, produced by Human Controller, YouTube, 17 Sept. 2018, https://www.youtube.com/watch?v=xCoH2AORcEQ.

Controller, Human. "全自動ルービックキューブ　Self Solving Rubik's Cube by Human Controller." *Self Solving Rubik's Cube*, DMM.make, 17 Sept. 2018, https://media.dmm-make.com/item/4462/.

Flatland, Jay, and Paul Rose. "World's Fastest Rubik's Cube Solving Robot - Now Official Record Is 0.900 Seconds." *YouTube*, produced by Jay Flatland and Paul Rose, 11 Jan. 2016, https://www.youtube.com/watch?v=ixTddQQ2Hs4.

Jelinek, Josef. "Rubik's Cube Java Applet Program." *Rubik's Cube Java Applet Program*, Ruwix, https://ruwix.com/the-rubiks-cube/rubiks-cube-java-applet-software-program-josef-jelinek-animating-rubix/.

Petrus, Lars. "Petrus Method." *Petrus Method*, SpeedSolving.com Wiki, https://www.speedsolving.com/wiki/index.php/Petrus_Method.

Pochmann, Stefan. "Blind Solving Algorithms." *Blind Solving Algorithms*, SpeedCubeReview.com, https://www.speedcubereview.com/blind-solving-algorithms.html.

Thistlethwaite, Morwen. "Thistlethwaite's Algorithm." *Thistlethwaite's Algorithm*, SpeedSolving.com Wiki, https://www.speedsolving.com/wiki/index.php/Thistlethwaite's_algorithm.

6.3 Appendices

6.3.1 Operations Manual

In this section, we will describe the usage of our proposed self-solving cube.

6.3.1.1 Hardware

From a user's perspective, the physical operation of the cube is simple.

First, the user should pick up the solved cube and scramble the cube as desired. Once the user has sufficiently scrambled the cube, he/she should place the cube in any orientation on a flat surface, such as a table. After a brief delay, the cube will solve itself, so the user can watch the solving process as desired. Once the cube is fully solved, the user may pick up the cube to restart this process.

Eventually, the cube will need to be charged by connecting the charging port to a power source through a micro-USB cable.

6.3.1.2 Software

From a user's perspective, the cube does not contain any software. All of the software will be embedded onto the Teensy 4.0, and the typical user will not need to run this software directly. Instead, the software will run automatically once the cube has been scrambled.

If an ambitious user wishes to test the solving algorithms, they must understand how to test *mainDriver.c*, which is the main driver for the solving algorithms. To do this, the user must encode the rotations of the scramble properly in *inputRotationFile.txt*, and then they must run *runMainDriver.sh* to run *mainDriver.c*. This produces *outputRotationFile.txt*, which contains an encoded rotation list for solving the cube.

Additionally, an ambitious user can test the algorithms for the first, second, and third layers directly. To do so, the user can run *runFirstLayerTests.sh*, *runSecondLayerTests.sh*, and *runThirdLayerTests.sh*, respectively.

Finally, if an ambitious user wishes to test the Arduino software, he/she can test each module individually. To do so, the user can run the *rotationDetection*, *fileReadWrite*, *motorRotations*, and *stepperSpeedControl* modules, which perform the expected operations. By combining these modules, the user can test subsystems in our design; for instance, the combination of *rotationDetection* and *fileReadWrite* will create a list of rotations for the solving algorithms.

6.3.2 Other Design Versions

As noted in Section 3.1, we considered external solvers in our design. However, this does not fit our client's needs since it would reduce the sense of magic introduced by internal solvers.

Additionally, we considered servo motors, but we eliminated these motors for various reasons. First, we needed to be able to turn the motor easily without sacrificing the motor's torque. Furthermore, the motors needed to fit comfortably within our housing sphere. The only other possible solution involves a custom gearbox, but we were unable to implement this solution due to our lack of mechanical knowledge.

On the software side, we also considered modifying the solving algorithms to limit the cube's movement to a fixed area. To do so, we planned to test the movements of the prototype during each rotation, which would allow us to adjust our algorithms based on the simulated position of the cube in software. Unfortunately, we were unable to implement this modification due to COVID-19.

6.3.3 Other Considerations

6.3.3.1 Hardware

Late in the spring semester, we determined that our Teensy would need an SD card for transferring information between the Arduino code and the solving algorithm executable, which was incorporated into the final hardware design in Figure 4. Ultimately, due to COVID-19, we were unable to construct the final cube, so fully assembling and testing the cube will be essential in future iterations of this project.

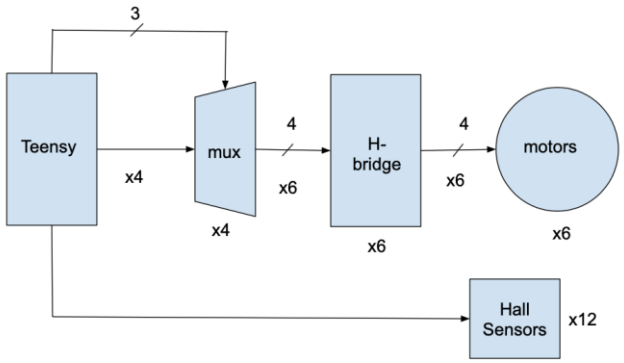

Figure 4: General Wiring Diagram

6.3.3.2 Software

For our solving algorithms, we did not implement multiple algorithms for solving the cube due to time constraints. In future iterations of this project, it would be useful to consider multiple algorithms to minimize the solving time.

For our Arduino code, we did not integrate the modules due to COVID-19. Consequently, in future iterations of this project, the Arduino modules will need to be integrated and tested fully on breadboards. This will ensure that the system-level Arduino code functions properly.

6.3.3.3 Final Thoughts

Overall, we enjoyed this project in many ways, and we were very disappointed that we were unable to complete the project. Ultimately, we applied many skills from our ECpE curriculum, but we noticed that we lacked a lot of motor knowledge. As a result, we sought assistance from the Mechanical Engineering department to complete this project. Consequently, in future iterations of this project, it would be helpful to include mechanical engineers as well as ECpE students.