Cubr: 3x3x3 Puzzle Solver

JT Aceron, Lily Chen, Sam Fazel-Sarjui

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— Cubr is a system capable of solving a 3x3x3 Rubik's Cube. Cubr uses computer vision to scan a real cube and map its configuration into 2D space to create a cube string. This cube string is passed into our 3x3x3 solver, which determines the optimal set of moves to solve the puzzle, known as a solution string. The solution string is then handed over to the physical robot. The robot contains six motors, one for each face of the cube, and will execute the solution string to solve the Rubik's Cube in under 20 seconds.**

***Index Terms*— Arduino, computer vision, cube configuration string, cube notation, cube state detection, motors, motor drivers, Rubik's cube, solution string**

I. Introduction

Cubr, a puzzle cube solving robot, is an approach to introduce technology to a common but sophisticated household puzzle. The motivation of Cubr is to mechanically solve a 3x3x3 puzzle cube. We constitute Cubr's solution as successful in three ways. The first metric is to correctly classify the colors of each cube piece on all six sides of the cube 100% of the time. An error in color classification will result in an invalid cube configuration string and will lead to an inaccurate representation of the real cube. Furthermore, our solving algorithm will not find a solution for an invalid cube string. The second metric is for our solver module to find a solution string of rotation moves that is less than 300 moves using an intuitive approach known as the Beginner's Method. We consider a solution string with more than 300 moves to be an inefficient solution. We also classify a solution string that does not solve a valid cube configuration as an insufficient solution. This constrains our solver module from being too computationally expensive and naive. The last metric is the execution time for a motor arm to turn a face of the cube. With the Two-Phase Algorithm, we can obtain a solution string containing a maximum of 20 moves. With this, the motor arm must turn its designated cube face in less than 1 second and be able to perform the Two-Phase solution string in less than 20 seconds. Due to its proximity to human intuition, the Beginner's Method implementation in Cubr can be used as a learning tool. Cubr is important in proving that technology can be introduced to everyday objects that previously have not been integrated with technology.

II. Design Requirements

In mechanically solving the cube, there are three major components that are critical to the success of our project. The first is cube state detection, which is responsible for successfully mapping the cube to 2D space. The second component is the solving software which will take in a cube configuration string and will output a solution string. The third component is responsible for mechanically turning the cube faces to physically solve the cube.

To verify that the first component, cube state detection, has met our specification, we will scan 10 differently scrambled cubes and ensure that all pieces are correctly scanned and mapped. The testing and validation will not be complete until this requirement is met. We will be scanning a standard 3x3x3 cube with the traditional sticker color scheme through an external webcam.

To ensure algorithmic efficiency of the solving software, we are first evaluating whether or not a solution is found. If a solution is not found, then the solving software is not correct and we fail our success benchmark. We also fail our success benchmark if a solution of less than 300 moves is not found. The Beginner's Solving Method should approximately take 200 moves; therefore, we have a leniency margin of 100 moves. A solution string of greater than 300 moves indicates that our solving software is inefficient and is too naively solving the cube.

The hardware component is the most challenging for our group and requires the most attention. We will be testing the hardware components in steps. First, we want to make sure we can move a single motor with the control we desire. From that, we would like to map each possible move for each face to a keypress. If we can do this successfully, we'll be able to transition the code from reading keypress events to parsing a solution string and ultimately executing the set of moves to solve the puzzle.

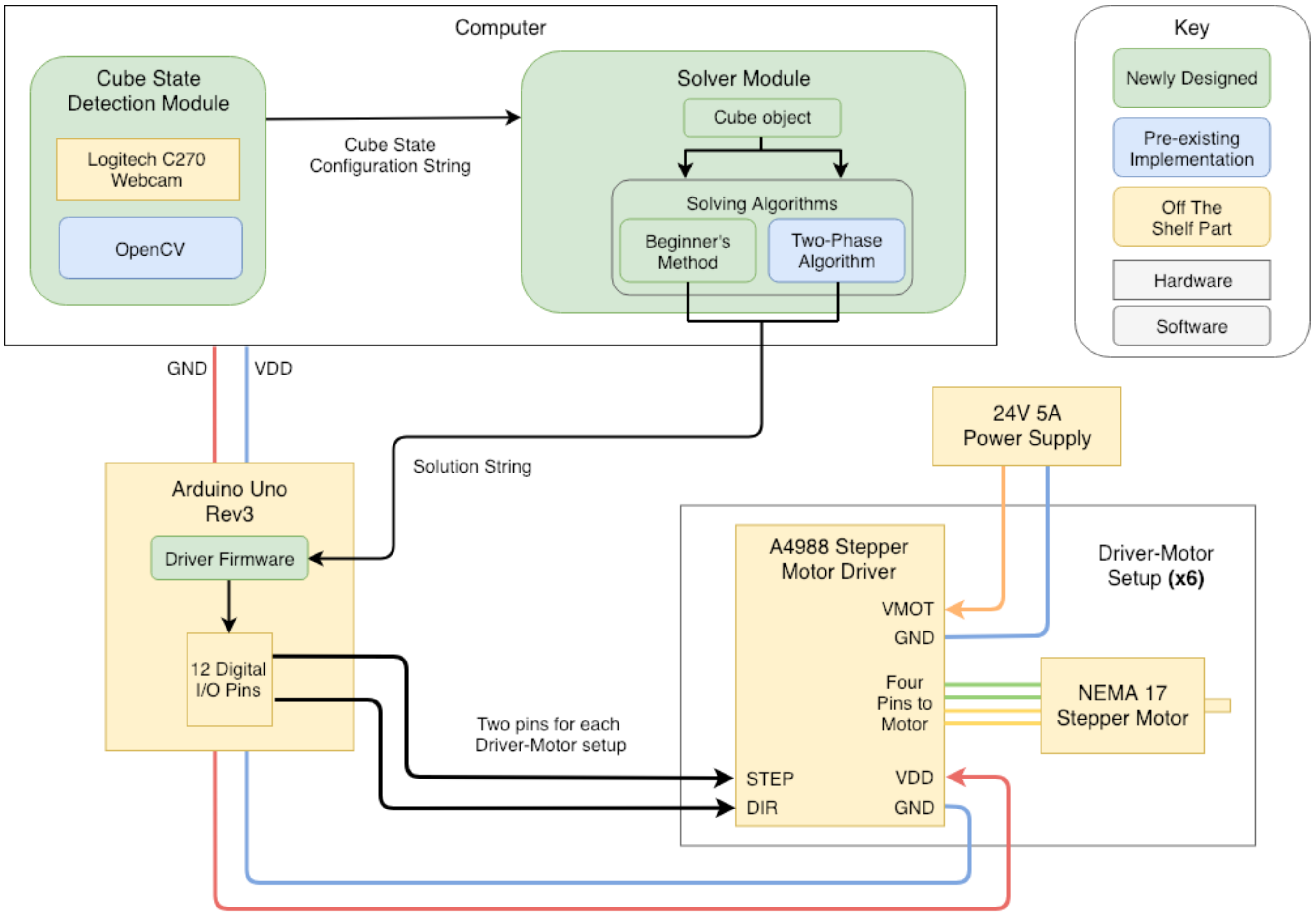


Fig. 1. Cubr Solution Approach Block Diagram Overview

III. Architecture and Principle of Operation

Our solution approach consists of three main modules to create the Cubr system. The first two, cube state detection and the solver, are software-based using OpenCV and Python. The third and final module, physical solving, is hardware-based with an Arduino as the mode of computation and control. Lastly, we need to ensure smooth integration between all three components. The Cubr system block diagram is depicted in Figure 1.

A. *Cube State Detection*

The first software module in Cubr's pipeline is the cube state detection. Cube state detection is an integral part of the process which identifies the given configuration of the cube and interfaces the mapped cube to our solver module. To scan all sides of the cube, a webcam is used with OpenCV to capture and identify the sides of the cube.

In the cube scanning portion of this module, there is a region to place a single face of the cube in front of the webcam which is indicated by nine small rectangles in the web frame. Additionally, there is a live color tracker in the top left corner of the window that shows the colors that the color detection algorithm is seeing for each cube piece of the current face. This acts as a validation step to ensure the right colors are being tracked. Then the user presses the spacebar to capture and record the side of the cube when the cube is aligned properly with the rectangles and the live color tracker is showing the correct colors. The software stores the colors of a given face of the cube in a data structure. The mapping of a given face is based on the color of the centerpiece as the centerpieces cannot be moved. This process is repeated for all six sides of the cube.

We standardize the scanning process by restricting the orientation in which we scan: the yellow centerpiece must be on top and the white centerpiece on the bottom. The user will make 90 degree turns while the white and yellow centerpieces are oriented correctly. Then, the user will make a 90 degree turn along another axis that allows the camera to scan the top yellow face and the bottom white face. This aligns with the 2D mapping of the cube as shown in Figure 2.

After all sides of the cube are scanned, the cube state detection module is responsible for properly mapping the cube and providing a cube string with the correct notation as defined in our software interface between the cube state detection and the solver modules.

B. *Solving*

The second module is the 3x3x3 puzzle cube solver. From the cube state detection module, the solver module will receive the cube configuration string for processing. We decided to implement the solver module in Python given its readability, dynamic typing, automatic memory management, and object-oriented programming (OOP) features.

Once the solver module instantiates a Cube object and identifies the individual cubie pieces and locations, the module will execute the Beginner's Method of solving, which reflects the way a human would solve a 3x3x3 puzzle cube. The Beginner's Method solves each layer of the cube by applying a given algorithm to each sublayer and maintaining the pieces of previous layer that are already in place.

As the module proceeds to solve the cube state by layers, it concatenates each sublayer solving algorithm to a string of moves. Once the cube object reaches its fully solved state, the resulting string of moves will be used to direct the hardware module for physical solving of the cube.

C. *Physical Execution*

The third and final module in our approach is the physical execution of a solution string to solve the puzzle. The cube solving robot takes a single solution string as its sole input. Regardless if the string is a valid solution or not, this module will read the set of instructions and perform each move serially via a configuration of bipolar stepper motors and motor drivers.

As for the main hardware components of the robot, this module consists of a microcontroller, six bipolar stepper motors, six stepper motor drivers, and a power source. Secondary components include a casing or housing for the motors, cube coupling arms, a simple breadboard, and wires to make all the necessary connections.

The computation power of the robot will be from an Arduino Uno Rev3 microcontroller. This is also how the robot will communicate with our solving module to receive a solution string. The Arduino is also responsible for controlling each of the six motor drivers in order to execute all 18 possible moves on a 3x3x3 cube. A Rubik's Cube has six faces and each face only has three possible moves. These 18 possible moves are defined in a universal cubing standard called cube notation used in official World Cube Association Speedcubing competitions.

Cubr uses NEMA-17 Stepper Motors to attach to the center of each cube face with custom cube coupling arms to turn the puzzle. These motors can turn either direction, clockwise or counterclockwise, at any number of steps between 0 and 200, where 200 steps is a full revolution. To drive and move these stepper motors, Cubr utilizes A4988 Stepper Motor Driver Carriers. These drivers allow the Arduino to interface with the stepper motors without the need to write excess and pre-existing code or to implement third-party libraries.

One driver is used for each stepper motor, and all stepper motors directly communicate with the Arduino to receive direction and step instructions through the Arduino's digital input/output pins. A stepper motor and driver setup is repeated six times in parallel to the power supply to create the core base of the robot. From here, each setup is positioned into a housing or framework that allows each stepper motor to perpendicularly connect with each of the six faces on the puzzle. To turn the face of the cube, custom 3D-printed coupling arms are created to attach to the stepper motor and center piece of each cube face.

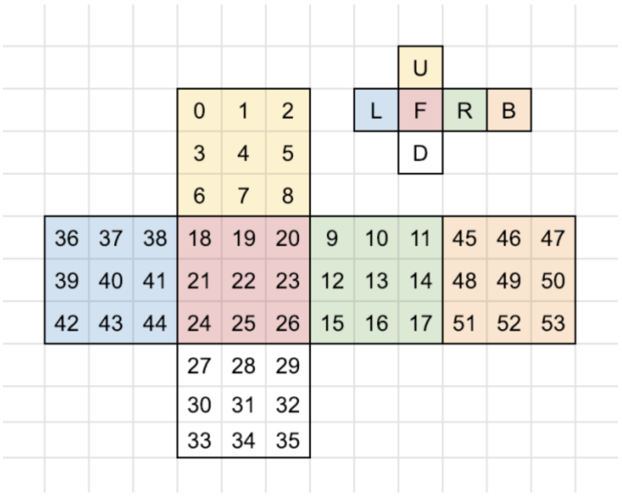


Fig. 2. 2D Mapping of 3x3x3 Cube

D. *Integration*

The cube itself is first scanned in front of the webcam for our cube state detection module. From cube state detection, a cube configuration string is outputted to STDOUT. From this, the solver module reads this cube string as input and creates a solution string as its output. The Arduino Uno Rev3 reads each move in the solution string and communicates with the appropriate motor driver to control the motor arms to physically solve the cube.

After it is scanned, the cube is then placed in the housing with the correct orientation. We predefine this as the yellow side facing upwards, and the red side facing forwards. Once the cube is firmly attached into the coupling arms, we start the solving process on the Arduino.

IV. Design Trade Studies

A. *Cube State Detection*

In designing a software solution for cube state detection, our top priorities were to have the highest accuracy in color detection and to design the most lightweight solution so that we can spend the majority of our efforts on the cube solving robot. The first design tradeoff for cube state detection was to use OpenCV for color detection as opposed to using color sensors. We chose this option as there is less physical moving parts needed in OpenCV, which is a computer vision library. It is also easier to customize our cube scanner to account for different cubes and their color profiles.

For validation and testing, we are scanning all sides of 10 differently scrambled cubes and are recording the number of centerpieces, edge pieces, and corner pieces scanned correctly. We will record which colors were incorrectly scanned against which colors were correctly scanned to get a percentage of the colors that were misread. We will also note how successful the color detection is with the logo sticker that is typically on the white centerpiece.

B. *Physical Execution*

As shown in the design requirements, we need a fast and reliable method of physically solving a 3x3x3 Rubik's Cube. We chose to configure the robot with six bipolar stepper motors, one for each face. Having a stepper motor for each

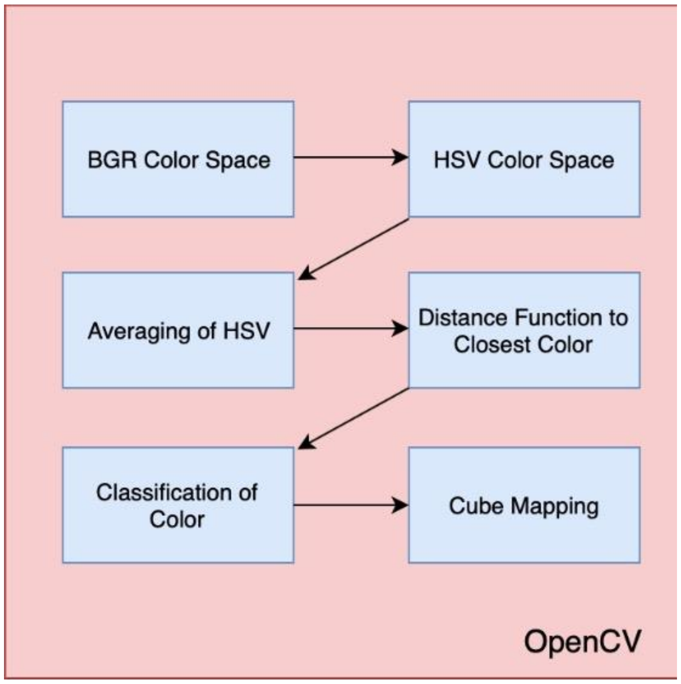


Fig. 3. Cube State Detection Workflow

side of the cube allows for the fastest solve time as no moves are wasted to reposition the cube. Other methods such as the claw and gripper method for physically solving the robot require less motors, but necessitate more fine-tuning and precision of the gripping movements. On top of this. the solve time is significantly slower due to no direct connection with each cube face. However, our method requires the most moving parts and ultimately has more power requirements. To address these new constraints, a 24V 5A Power Supply will be used to power all stepper motor and driver setups in parallel. This power supply was specifically chosen to operate at the higher voltage range for each of the six A4988 drivers while still providing enough current for each of the motors. Since each of the stepper motors operate at 350 mA, the power supply needs a total of at least 2.1 A to supply adequate current to all 6 motors at once since the motors pull current while stationary.

The NEMA-17 Stepper Motors are rated for 12 V and operate with a maximum speed of 600 rpm. To operate at its higher end speeds, we introduce the use of A4988 drivers. These drivers allow us to push the stepper motor's voltage rating to achieve higher step rates through adjustable current control. These drivers also allow us to directly communicate to the motors through the use of two digital input/output pins from the Arduino for each configuration. With this setup, we believe that we can achieve an operating speed of 1 cube turn per second at the very minimum. We will test our robot's turn speed with a baseline solution string of 20 moves. Since the Two-Phase Algorithm can output solutions around 20 moves or less, we want to see how fast we can execute 20 sequential moves. Once the robot is operating with a cube inside of it, we will time how long it takes to correctly execute the 20 moves.

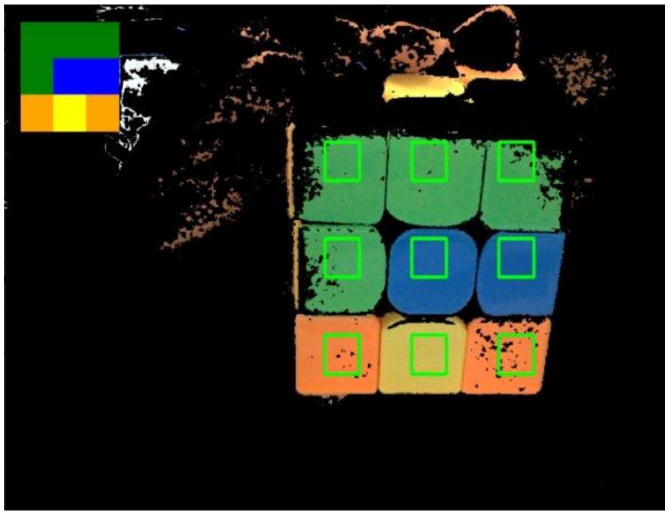


Fig. 4. Cube State Detection User Interface

V. System Description

We are currently developing Cubr on Unix-based systems. However, the execution of our software is platform agnostic. This allows for future work in migrating our software to different operating systems or for others to download our repository and create or modify the robot for their personal uses.

A. *Cube State Detection*

For the cube state detection subsystem, the modules required are OpenCV 4.0.0 and Python 3.6+. The cube state detection software reads each frame in the live video stream from a Logitech C270 webcam using OpenCV. Since OpenCV's default color space is BGR (blue, green, red), we convert each frame into the HSV (Hue, Saturation, Value) colorspace due to its robust lighting invariance for color detection. This basic workflow is shown on Figure 3.

Regions are indicated in the web frame with small green rectangles. Within these regions, the HSV values are averaged frame by frame for the live color tracker which shows what colors are being detected for each cube piece. The spacebar key event captures a frame and maps the side of the cube in an internal data structure. In the color detection pipeline, after the HSV values are averaged, the L2-Norm distance function is then used to find the closest color detected in those regions. We calculate the HSV value of a color by taking the midpoint of the non-overlapping HSV lower and upper bounds set for each color.

With cube state detection, we found it difficult to classify each color correctly with any type of variance in lighting, which includes glare from the cube itself. To work around this, Cubr is using a lighting softbox to standardize the lighting in the cube scanning portion of the cube state detection module.

As another precautionary measure, the green rectangles in the web frame, as shown in Figure 4, were adjusted to be smaller. This adjustment made it easier for the user to align the cube in front of the webcam. The smaller the sampling area, the less variance of color, which improved the accuracy of the cube state detection. Showing a live color tracker also

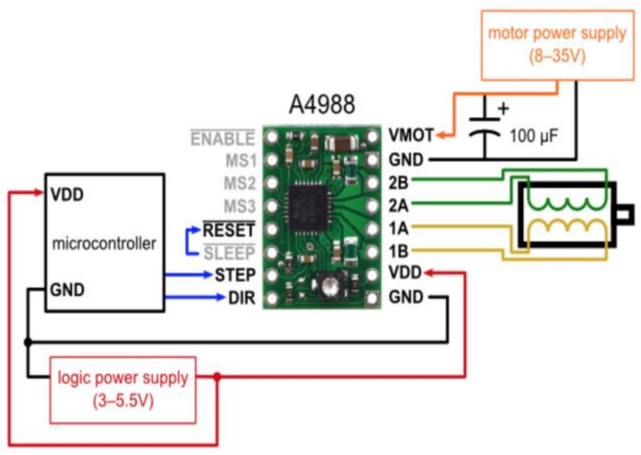


Fig. 5. Minimal Wiring Diagram for the Driver-Motor Setup

shifts the error towards the user. This will help assure high accuracy readings.

B. *Solver*

For the sake of continuity, the solver module is built on Python 3.6. All Python libraries used in the solver are part of the Python Standard Library and thus require no additional installation. The general structure of the code follows the object-oriented programming paradigm. After the solver module receives the cube configuration string from the cube state detection module, the solver module instantiates a Cube object for the given cube state and identifies the individual cubie pieces and locations prior to executing the solving algorithms.

With an object-oriented structure, we can create class instantiations for each input configuration string. Taking an OOP approach allows us to view and identify the cube based on the individual cubies, or pieces, instead of by colors; doing so will allow us to take a "colorblind" approach in mapping colors to the faces in the cube state detection module. Furthermore, OOP makes it easier to maintain and modify our existing code such that we can run both our implementation of the Beginner’s Method and pre-existing solving algorithms, such as the Two-Phase Algorithm, on the same cube states. These different method calls work with the basic but concrete structure, types, and properties for our modular OO classes.

The Beginner’s Method solves each layer of the cube by applying a given algorithm to each sublayer and maintaining the pieces already in place. Each sublayer-solving algorithm consists of 18 types of rotation moves, with three for each face: turns of 90 degrees clockwise, 90 degrees counterclockwise, and 180 degrees.

The Two-Phase Algorithm outputs a solution string of a maximum of 20 moves. In its initial run, the algorithm produces tables for memoization and lookup that amount to 140 million lookup references at 13 different depths. In addition, during subsequent runs of the Two-Phase Algorithm, the algorithm does not stop when a first solution is found, continuing its search for shorter solutions from suboptimal parts of the prior solution. Moreover, the Two-Phase Algorithm also uses symmetrical properties of the 3x3x3 to find and perform symmetry reductions.

As the module proceeds to solve the cube state by layers, it concatenates together each sublayer-solving algorithm's list of moves into what we call the solution string. Once the Cube object reaches its fully solved state, the resulting solution string will be used to direct the hardware module for physical solving of the cube.

C. *Physical Execution*

The NEMA-17 Stepper Motor is a four wire bipolar stepper that rotates 1.8 degrees per step. The motor operates at a maximum current of 350 mA and operates with a power rating ranging of 8-35 V. 200 steps are required to make one full rotation. Notable step inputs include 100 steps and 50 steps to execute 180-degree and 90-degree turns respectively. These motors provide ample torque to smoothly turn a single cube face.

The A4988 Stepper Motor Driver Carrier is the driver of choice to interface between the NEMA-17 motors and the Arduino. The driver allows for a simple step and direction control interface while having five different step resolutions, though the Cubr robot will operate at the full-step resolution. These drivers also have adjustable current control, which allows for the use of higher voltages above the NEMA-17’s rated voltage to achieve higher step rates. The A4988 has two pins called “STEP” and “DIR”: these are the pins that will each directly connect with one of the 14 digital input/output pins on the Arduino. Because there are six setups, Cubr requires at least 12 digital I/O pins.

These parts in conjugation as depicted in Figure 5 allows for control over a single NEMA-17 stepper motor. This setup will be repeated 6 times to create the core of the cube solving robot.

VI. Project Management

A. *Schedule*

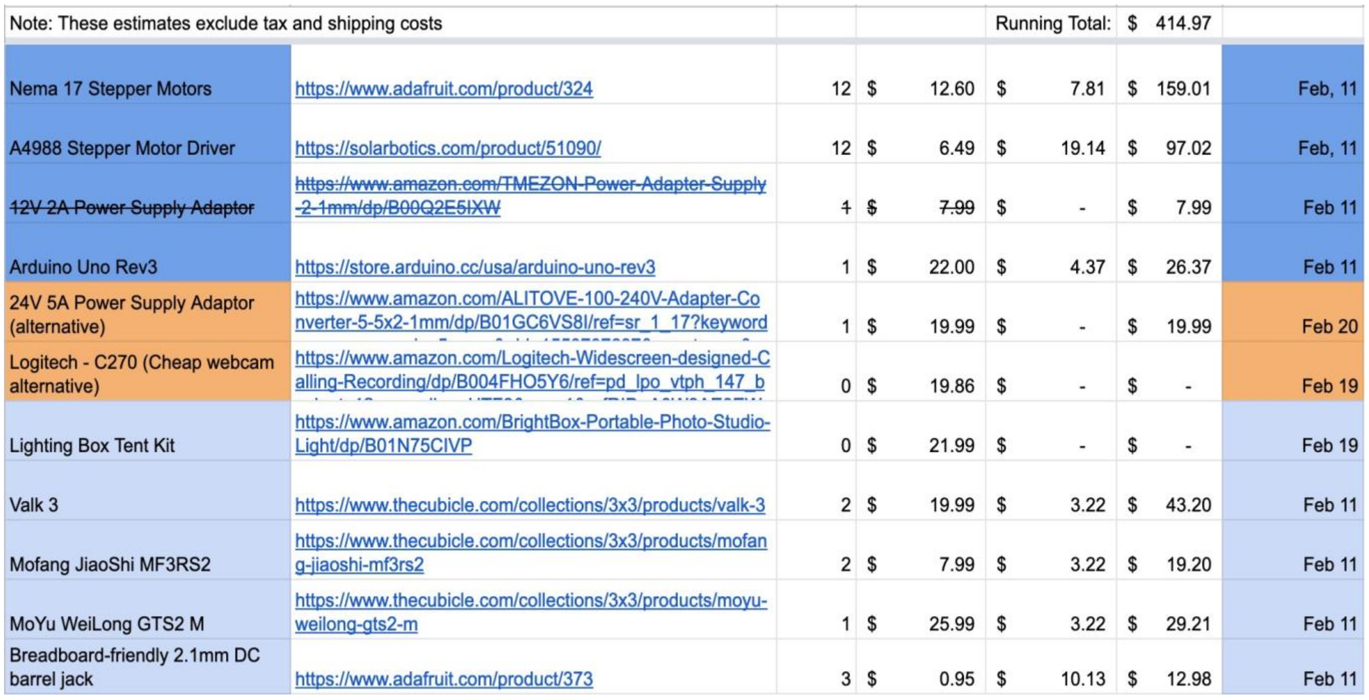
Cubr has three major parts that are modular. This allows the team to work simultaneously without too many blocks in the production pipeline. Each of the three major components have a well-defined input and output. Working within these constraints, the team can continue to make progress on their respective tasks without waiting on a task outside of their module to be completed. However, modularity is lost once all three tasks are completed, at least on a rudimentary level. As the team converges, the bottlenecks arise from tasks such as construction and print time.

As of now, one team member is assigned to each of the three core modules. After these are completed, the team will come back together to integrate, test, and assemble. The team schedule is located on page 7, Figure 7.

B. *Team Member Responsibilities*

As depicted in Figure 7, JT is in charge of items in red, Sam has tasks in blue, while Lily performs jobs in green, and multiple team members work on the yellow assignments. While we are all in charge of defined components, all three members have a responsibility to understand every aspect of the project.

JT is primarily focusing on firmware programming and hardware design for the physical robot. After this, he will



Note: These estimates exclude tax and shipping costs							Running Total:	$	414.97	
Nema 17 Stepper Motors	https://www.adafruit.com/product/324	12	$	12.60	$	7.81	$	159.01	Feb, 11
A4988 Stepper Motor Driver	https://solarbotics.com/product/51090/	12	$	6.49	$	19.14	$	97.02	Feb, 11
~~12V 2A Power Supply Adaptor~~	~~https://www.amazon.com/TMEZON-Power-Adapter-Supply-2-1mm/dp/B00Q2E5IXW~~	~~1~~	$	~~7.99~~	$	-	$	7.99	Feb 11
Arduino Uno Rev3	https://store.arduino.cc/usa/arduino-uno-rev3	1	$	22.00	$	4.37	$	26.37	Feb 11
24V 5A Power Supply Adaptor (alternative)	https://www.amazon.com/ALITOVE-100-240V-Adapter-Converter-5-5x2-1mm/dp/B01GC6VS8I/ref=sr_1_17?keyword	1	$	19.99	$	-	$	19.99	Feb 20
Logitech - C270 (Cheap webcam alternative)	https://www.amazon.com/Logitech-Widescreen-designed-Calling-Recording/dp/B004FHO5Y6/ref=pd_lpo_vtph_147_b	0	$	19.86	$	-	$	-	Feb 19
Lighting Box Tent Kit	https://www.amazon.com/BrightBox-Portable-Photo-Studio-Light/dp/B01N75CIVP	0	$	21.99	$	-	$	-	Feb 19
Valk 3	https://www.thecubicle.com/collections/3x3/products/valk-3	2	$	19.99	$	3.22	$	43.20	Feb 11
Mofang JiaoShi MF3RS2	https://www.thecubicle.com/collections/3x3/products/mofang-jiaoshi-mf3rs2	2	$	7.99	$	3.22	$	19.20	Feb 11
MoYu WeiLong GTS2 M	https://www.thecubicle.com/collections/3x3/products/moyu-weilong-gts2-m	1	$	25.99	$	3.22	$	29.21	Feb 11
Breadboard-friendly 2.1mm DC barrel jack	https://www.adafruit.com/product/373	3	$	0.95	$	10.13	$	12.98	Feb 11

Fig. 6. Cubr Bill of Materials

work on housing assembly and hardware integration. In conjunction, JT will assist Sam with refining the cube state detection.

Lily will be focusing on overall software design and the Beginner's Method solver. She will also lead the hardware design of the housing CAD models. During housing design, she will help hardware integration and testing.

Sam's primary role is the oversight and development of the cube state detection. Upon completion, Sam will join the rest of the team in addressing problems in the hardware integration phase. Simultaneously, Sam will help with improving the software design.

C. *Budget*

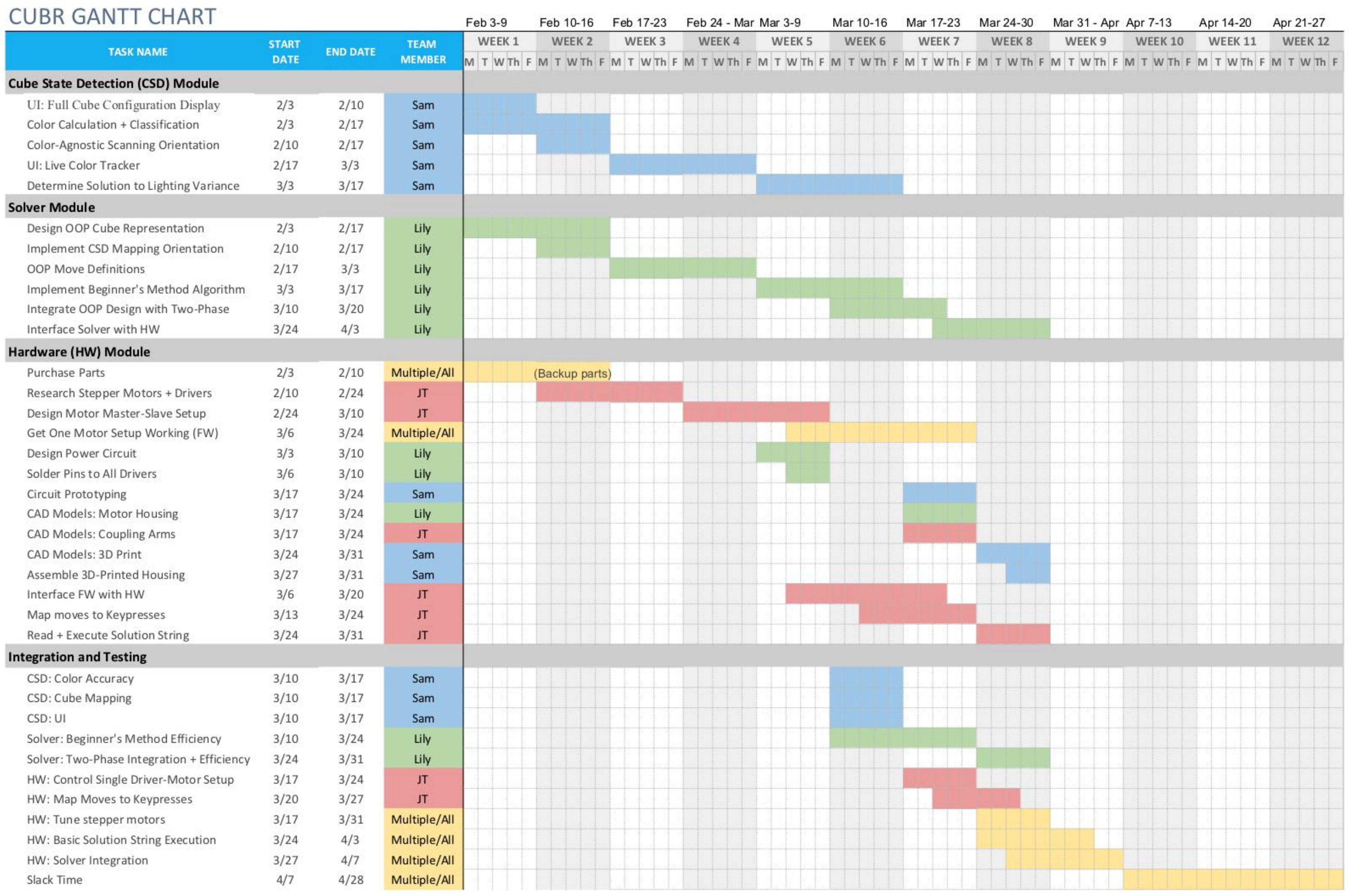
From Figure 6, blue purchases are priority and essential purchases; orange were alternative purchases that were made to better fit the project as its needs were further realized. Light blue items are secondary purchases that are not essential in the ongoing production of the project but have purpose in our final product. So far, we have upgraded to a new power supply that had a higher voltage and current rating to better accommodate the six stepper driver-motor setups in parallel.

D. *Risk Management*

To manage risk as much as possible, the team front loaded most of the design and solution approach for a better part of the first three weeks. Having a team comprised of software-focused engineers, we knew we had to plan out as much as possible in order to address our weakness in hardware design and execution. Everyone on the team is more than capable of accomplishing the software goals with high fidelity. However, none of us have worked with robotics or much hardware beyond the required courses for our undergraduate studies. JT was in charge of research and designing as much as he could before the team started to assemble physical pieces. We also defined clear modules that we assigned with well-defined inputs and outputs. This will allow us to ensure smooth integration between all three modules and adherence to the rules we defined at the beginning of the semester. These clear definitions also allow other members to lend their efforts towards the robot or other lacking modules after certain software milestones were completed.

From a schedule and communication standpoint, it was imperative that we were upfront and communicative. Slack and other team collaboration tools are constantly in use to provide timely status updates and to serve as a place for questions and clarifications about aspects of each module between the team members. As shown in our schedule, Figure 7, we broke down tasks into their simplest forms possible. This allows us to quickly see what was to be done and what needed work each week. Thanks to our solid planning, we were able to purchase all necessary tools and equipment in the first two weeks of the project. We have spent $415 out of the budgeted $600 and plan on allocating the rest for backup parts as needed.

The primary risk of the project was the physical construction of the robot module. With no real experience on the team, there were a lot of obstacles and problems that required attention from all three team members. We opted for a six bipolar stepper motor configuration because we felt it best suited the requirements of the capstone while being within the reach of a group of hardware novices. The use of off the shelf parts, such as the motors and drivers, would offload a lot of the tuning and precision of robotics onto the hardware.



CUBR GANTT CHART

TASK NAME	START DATE	END DATE	TEAM MEMBER
Cube State Detection (CSD) Module			
UI: Full Cube Configuration Display	2/3	2/10	Sam
Color Calculation + Classification	2/3	2/17	Sam
Color-Agnostic Scanning Orientation	2/10	2/17	Sam
UI: Live Color Tracker	2/17	3/3	Sam
Determine Solution to Lighting Variance	3/3	3/17	Sam
Solver Module			
Design OOP Cube Representation	2/3	2/17	Lily
Implement CSD Mapping Orientation	2/10	2/17	Lily
OOP Move Definitions	2/17	3/3	Lily
Implement Beginner's Method Algorithm	3/3	3/17	Lily
Integrate OOP Design with Two-Phase	3/10	3/20	Lily
Interface Solver with HW	3/24	4/3	Lily
Hardware (HW) Module			
Purchase Parts	2/3	2/10	Multiple/All
Research Stepper Motors + Drivers	2/10	2/24	JT
Design Motor Master-Slave Setup	2/24	3/10	JT
Get One Motor Setup Working (FW)	3/6	3/24	Multiple/All
Design Power Circuit	3/3	3/10	Lily
Solder Pins to All Drivers	3/6	3/10	Lily
Circuit Prototyping	3/17	3/24	Sam
CAD Models: Motor Housing	3/17	3/24	Lily
CAD Models: Coupling Arms	3/17	3/24	JT
CAD Models: 3D Print	3/24	3/31	Sam
Assemble 3D-Printed Housing	3/27	3/31	Sam
Interface FW with HW	3/6	3/20	JT
Map moves to Keypresses	3/13	3/24	JT
Read + Execute Solution String	3/24	3/31	JT
Integration and Testing			
CSD: Color Accuracy	3/10	3/17	Sam
CSD: Cube Mapping	3/10	3/17	Sam
CSD: UI	3/10	3/17	Sam
Solver: Beginner's Method Efficiency	3/10	3/24	Lily
Solver: Two-Phase Integration + Efficiency	3/24	3/31	Lily
HW: Control Single Driver-Motor Setup	3/17	3/24	JT
HW: Map Moves to Keypresses	3/20	3/27	JT
HW: Tune stepper motors	3/17	3/31	Multiple/All
HW: Basic Solution String Execution	3/24	4/3	Multiple/All
HW: Solver Integration	3/27	4/7	Multiple/All
Slack Time	4/7	4/28	Multiple/All

Fig. 7. Cubr Detailed Gantt Chart