Rubik's Cube Solving Robot

Daniel Truesdell[1], Corey Holsey[2], Tony Verbano[3]
Department of Engineering and Computer Science
University of Central Florida
Orlando, Fl.
Email: [1]danieltruesdell@knights.ucf.edu,[2]holseyc@knights.ucf.edu,[3]tonyverbano@knights.ucf.edu

***Abstract*—In this paper we present the design and implementation of a robotic system that is capable of autonomously solving a Rubik's Cube puzzle. The four main components of our system are an integrated image sensing device, a custom embedded processing platform, a PC-based software application, and a physical robotic structure. These components function together to accurately decode and solve the Rubik's Cube puzzle in a timely manner.**

I. Introduction

The Rubik's Cube is a timeless puzzle that has challenged people since its creation in 1974. Significant mathematical investigations of the Rubik's Cube over the past decades have provided many frameworks and algorithms for systematically decoding and solving it. In recent years, cube enthusiasts have leveraged the power and speed of modern computers to analyze scrambled cubes in real time and determine what manipulations are necessary to solve them. The computerization of this process has prompted the creation of robotic devices that are capable of carrying out the computer-generated manipulation sequences in order to physically solve a cube from start to finish. The speed and accuracy of these systems showcase the power of engineering to perform tasks far beyond human capability.

Existing implementations of these systems range in complexity from simple hobbyist weekend projects to high-level university projects such as the present work. Some systems take minutes to solve a cube while others are finished in under a second, and some systems are hardware-oriented while others place emphasis on elaborate software programming. Each project contributes a unique solution to a growing pool of knowledge and resources that collectively advance our ability to solve the challenge. To this end, we herein present the design and implementation of a robotic system that is capable of autonomously solving a Rubik's Cube puzzle.

II. System Overview

The robotic system, shown in Figure 1, consists of four main functional components: An image sensing device, a software application, an embedded system, and a physical structure. The following subsections briefly describe these components and their functionality within the system.

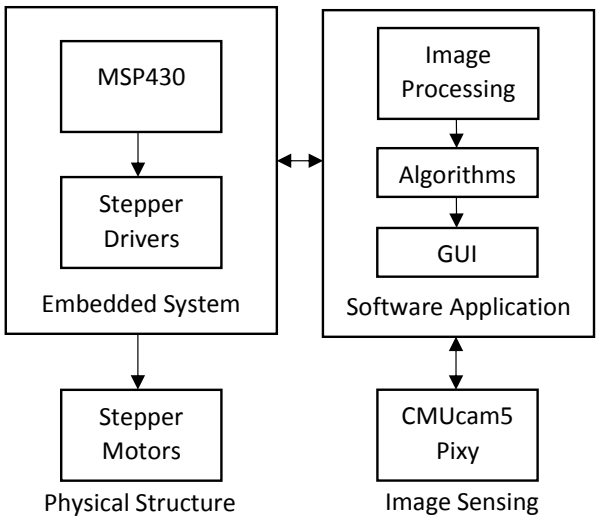

Fig. 1: Functional System Block Diagram

A. Image Sensing

This project implements the CMUcam5 Pixy as an integrated image sensing solution. The Pixy is a palm-sized camera with an on-board 204MHz NSP LPC4330 processor that allows it to perform image processing on raw data before it is sent over USB to our software application. Pixy's convenient libraries can be used to identify the colors of the cube as well as their positions so that the software can determine how the cube needs to be manipulated.

B. Software Application

Our robot software consists of several components for image processing and visualization, cube deciphering, a

solving algorithm, a GUI, and physical structure control. This software is collectively responsible for detecting the current state of the cube, deciphering the cube positions, applying the solving algorithm, and sending a string of information to the embedded system that tells it how it needs to manipulate the cube.

C. Physical Structure

The physical structure, shown in Figure 2, is what holds and manipulates the cube. The frame was designed using Autodesk Inventor and is intended to be laser cut from a variety of materials. The design of the structure allows any face of the cube to be turned with a dedicated stepper motor, which allows the cube to be manipulated in any way without needing any prior reorientation. This decreases the number of instructions needed to solve the cube which in turn reduces the amount of time it takes to do so.

Fig. 2: Robot Structure

D. Embedded System

The embedded system, shown in Figure 3, is designed with a Texas Instruments (TI) MSP430F6659 microcontroller that is interfaced with six TI DRV8825 stepper driver integrated circuits (IC's). The device is designed to receive a string of commands from the software application over a serial connection that will be decoded and used to actuate the stepper motors in order to manipulate the cube.

III. Software Design

A. Image Processing

One of the main components of our project is taking in the image of the cube from its mixed up state and

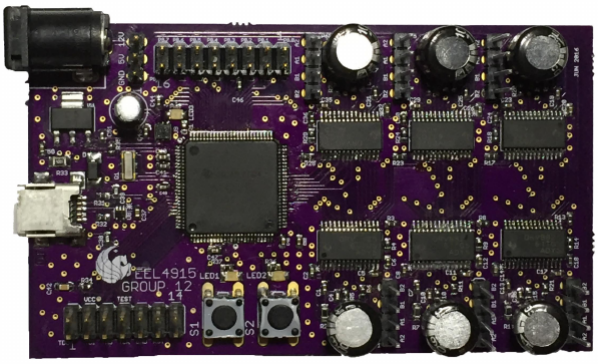

Fig. 3: Embedded System with MSP430F6659

placing its orientation into an array. We initially start this by with our Pixy CMU5 camera. This camera is set up to have a set of 7 signature colors that are recognizes easily. Luckily we will only need 6 signature colors for our cube leaving one value unknown. Each color will be given a signature color number as its initial value and that will be used to make our cube array. The camera is positioned to look at the bottom row of the top face and the top row of the front face. We have determined that from this position the whole cube can be viewed and oriented back to its initial state in 11 move sets. The camera takes a picture of the state of these rows at this position than rotates one face at a time till the whole cube is recognized.

After each initial picture is taken and before the cube is rotated, the image is the deciphered for the significant colors which are then placed into a string corresponding with their position on the cube. The image is than replaced with the next image after the rotation of the cube and the process of deciphering the state of the cube starts over until the whole cube is processed. After the whole cube is processed and the total cube is laid out into a string, our string is sent to our main program where it is placed into a matrix. The matrix is used with our GUI and our algorithm to help visualize and solve for the cubes correct orientation.

B. Kociemba's Algorithm

The algorithm we chose was Kociembas algorithm which is also known as Gods algorithm because it can solve the cube optimally. It is a two-phase algorithm that solves the cube in at most twenty moves when used optimally. It was founded in 2010 by Herbert Kociemba and was basically a refinement of another Rubiks cube algorithm [1]. Rather than using five groups it was cut down to merely three groups. The groups were identified

as G0, G1, and G2. The G0 group identified the initial state of the cube. The algorithm also utilizes symmetries and by reducing the number of symmetries that are available on the cube decreases the number of possible moves.

The Rubiks cube has billions of different symmetries 164,604,041,664 to be exact. Reducing those possible symmetries by finding correlation between symmetry, anti-symmetry, and conjugations greatly reduces the number of relevant symmetries to a little over a million. Once twenty turns have been completed utilizing the algorithm then exactly 32,625 different symmetric cubes can be solved using the twenty move algorithm by Herbert Kociemba.

The G1 state has a goal state which utilizes conjugations. The conjugation to move the edges of the middle cube in a way that once three moves are done it will result in the opposite of those three moves being done. It does not allow for the edges or corners to change orientation once in this state. The conjugate is sought by having millions of lookup tables that are generated and then pruned to find a solution to the cube. Pruning the cube while in this state is very important as to make the solution much faster on solving the cube. Deciding when to prune made the biggest difference in speed for our application. Then, that completes the phase 1 and it moves on to phase 2 of the algorithm.

While in phase 2 the corners and edges of the cube are permutated. Finally, once one solution to the cube is found then another solution is sought out that does not conflict with any of the possibilities that were discovered from the first solution. The second solution only goal is to make the first solution shorter. When phase 2 finally gets to zero then an optimal solution has been found and the algorithm is complete. A problem we had with this algorithm was there being so many lookup tables is the reason it was not possible to implement Kociembas algorithm directly on our microcontroller. It abuses entirely too much RAM for our MCU to handle. The easiest solution to this problem because RAM cannot be added to our MCU was to make a desktop application. A desktop computer (laptop) can handle a significant amount of RAM and still talk to our MCU by sending the solution string over UART.

Another difficulty is where to prune for the millions of lookup tables that are stored utilizing this algorithm. Pruning the tables in certain places can be the difference between these algorithm taking seconds or it taking a quarter hour. The prune tables could be loaded automatically when they are called upon which would make the time significantly longer. However, we chose to manually load the prune tables from the beginning of the program. Manually loading the prune tables is a bit more inefficient, but it is much faster which we were more concerned about.

Our software application will not look for solution that are completely optimal which would mean be done in 20 moves. This is because it increases the time for the solution to be found and having it solve in a consistent time was important to us. Therefore, we only look to solve the cube in approximately 24 moves.

C. Graphical User Interface

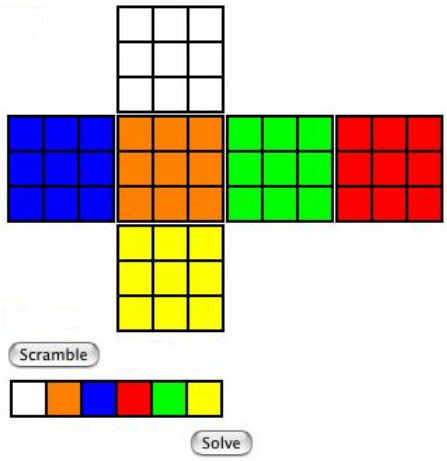

Fig. 4: Graphical User Interface

The software application will have a Graphical User Interface that will have a display of the cube. The display of the cube will be a 2D display that shows the different faces of the cube. The faces shown will be mapped appropriately once the initial string is interpreted to display the state of the cube. Each color of the cube is denoted by the center face that it is on. For example, the Green face is on the front face therefore it will be denoted as F rather than Green.

The Graphical User Interface will also consist of multiple buttons. One button that will made available is a solution button. The solution button will solve the cube at its current state by generating a solution string and sending it over to the robot over UART. The next button that is made available is that randomize button. The randomize button will change the state of the cube from its current state without looking to solve the cube. It will manipulate the array position in valid ways to give the cube a different state. In Figure 5 the randomize button is denoted by Scramble. While in the same figure the solution button is denoted by Solve. Other buttons or text may be added if we have time such as a timer or a

stop button. However, that is only if we have time in the end will we look to add additional functionality to the Graphical User Interface.

We chose to use java programming language to make the Graphical User Interface because it is friendlier to use for a Graphical User Interface than C programming language is. Java has libraries built in to make this process as seamless as possible. Visually the buttons are likely to simply be j-buttons from the standard java library. Both button will generate a solution string then convert that so that the our MCU can understand it and move the robotic arms appropriately.

In Figure 4 is an example of what our Graphical User Interface should look similar to once completed. We are still working on the Graphical User Interface because the front end it not as important nor difficult as the backend and actual assembly of the robot. Also, keeping things simple usually helps to generate as few errors as possible which is what we are seeking.

IV. Hardware Design

A. Processor

The block diagram for the embedded system is shown in Figure 5. The on-board MSP430F6659 offers connectivity and I/O through a micro-USB port with ESD protection, 4-wire JTAG pins, 16 GPIO pins that are mappable to various serial modules, 2 pushbuttons, and 2 LEDs. An additional 14 GPIO pins on the MSP430 are interfaced with the six DRV8824 stepper drivers.

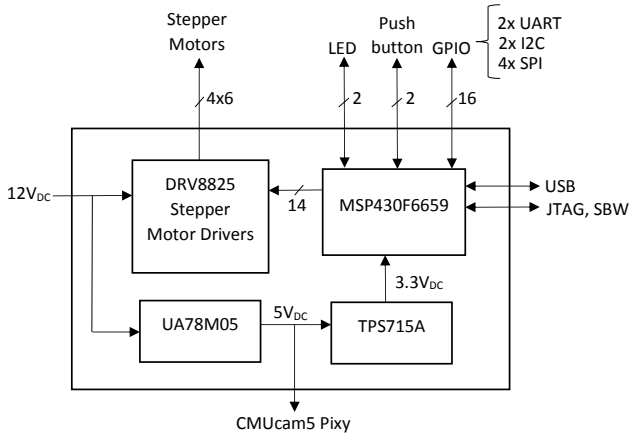

Fig. 5: Embedded System Block Diagram

B. Power

The board is intended to be supplied with $12V_{dc}$ via a 2.1mm barrel jack. This voltage is needed by the DRV8825 IC's to power the connected stepper motors, but it is too high for the MSP, so it is stepped down to $5V_{dc}$ by the UA78M05 and again to $3.3V_{dc}$ by the TSP715. A power LED indicates that the MSP430 is receiving the necessary $3.3V_{dc}$. If the $12V_{dc}$ connection is not used, the MSP430 can still be powered via the $5V_{dc}$ header pin, the USB port, or the JTAG V_{cc} pin.

C. Stepper Drivers

Dedicated stepper driver IC's are vital to the operation of the embedded system. Figure 6 shows the schematic for the DRV8825 stepper driver IC. The motors used in this system require 12V for maximum torque which is far beyond what the MSP430 can provide [2][3]. The DRV8825 solves this problem as it can be interfaced with an external motor supply voltage of up to 45V while still accepting the low-voltage control signals from the MSP430 [4]. It also protects the MSP430 from potentially harmful back-EMF from the large inductive loads of the stepper motors. The maximum full-scale motor current (I_{FS}) for this application was limited to 300mA to stay within the limits of the PCB traces as well as the 350mA current limit of the motors. The voltage divider calculation for I_{FS} is adapted from the device datasheet and shown below in equation 1:

$$300mA = I_{FS}(A) = \frac{xVREF(V)}{A_V \times R_{SENSE}(\Omega)} \tag{1}$$

Where the gain $A_V = 5$, $R_{SENSE} = 0.2\Omega$, and the voltage reference $xVREF$ is given in equation 2 as

$$xVREF(V) = \frac{V3P3OUT(V) \times R17(\Omega)}{R17(\Omega) + R18(\Omega)} \tag{2}$$

Where $V3P3OUT = 3.3V$. Equation 1 yields $xVREF = 0.3V$. Using this value in equation 2, $R17$ and $R18$ are selected as $20\,\mathrm{k}\Omega$ and $220\,\mathrm{k}\Omega$, respectively.

Another benefit of the DRV8825 is that it offers a microcontroller-friendly control interface. Bipolar stepper motors such as the ones in this project rely on bidirectional current control on four separate motor coil wires. Complex, high-precision drive patterns are required on these four wires in order to achieve smooth rotation of the motor shaft. The DRV8825 is used to handle all of the timing and current control on these wires by accepting 1-bit step and direction inputs from the MSP430. A chip enable signal allows the device to be disabled which causes it to ignore input signals and consume essentially zero power. The embedded system design takes advantage of this by using a single step signal to control all six DRV8825s but having multiple enable signals to enable

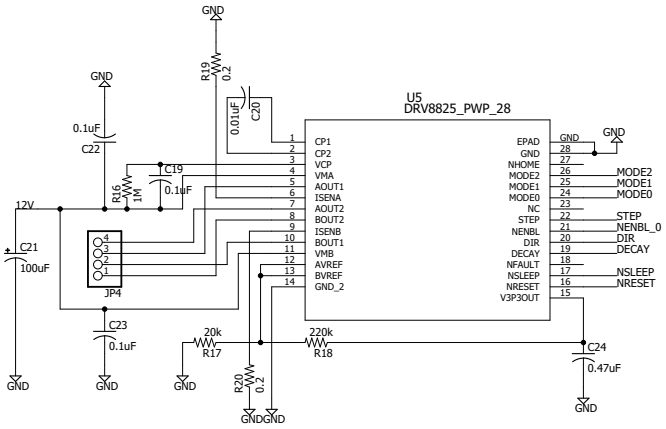

Fig. 6: Stepper Driver Circuit

the chips separately. This is shown in a simplified version of the interface in Figure 7. The result is the same as having a different step signal for each DRV8825 except that there is far less power consumption due to only one device being enabled at a time instead of all six.

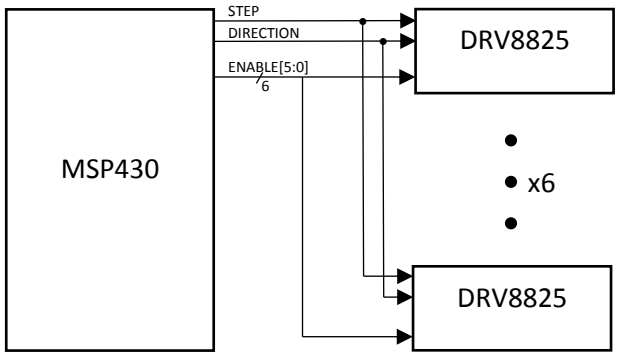

Fig. 7: Stepper Control Interface

D. Physical Structure

The physical structure of the robot is shown in Figure 2. The primary goal of the design was to allow any face of the cube to be turned without having to reorient the cube itself. The advantage of doing this is that it minimizes the number of commands that are needed to solve the cube. To implement this design, a motor is attached to the center tile of each face. Rotating the center tile will cause the entire face to be rotated. Because the cube is internally connected by the center tiles of each face, the positions of these tiles do not move relative to each other and thus the cube can be help stationary this way without limiting the ability to manipulate it.

The structure is divided into six separate pieces that are meant to hold each of the six motors. As shown in Figure 2, one of the motor supports will support the Pixy camera so that it can capture the colors of the tiles on one edge of the cube. Although the robot will be able to scramble a cube on its own, the modular assembly of the robot allows it to be easily detached from the cube so that a human can remove it and scramble it manually.

The Rubik's Cube is connected to the stepper motors by a 3D-printed motor shaft attachment that is designed to replace the center tiles of the cube. It is nearly identical to the original cube tile except that it has an outward-facing female connector for the motor shaft. To ensure that the connection is tight, the plastic connector is heat-fitted to the motor shaft once it is in place.

V. System Operation

The operation of our system depends on all individual hardware and software components working together correctly. This section discusses the integration of our system components and reviews the flow of operation for the system to complete its task of solving a Rubik's Cube.

A. Determining Cube State

The system operation begins with a scrambled Rubik's Cube being present within the physical structure. If the cube is not already scrambled, the GUI can be used to instruct the robot to automatically scramble the cube. Following this, the first goal of the system is to use the Pixy camera to detect the signature colors of the cube. Because the Pixy is stationary and has a limited view of the cube, it is necessary to rotate the cube several times in order for the Pixy to gather all the necessary information. The software keeps track of the transformations that are made and will return the cube to its original scrambled state after all the cube data is collected. The information gathered is sent to the software application over a USB connection and is used to form a data structure that represents the positions and colors of the tiles of the scrambled cube.

B. Determining Cube Solution

Once a data structure has been formed within the software, The user can use the GUI to instruct the robot to solve the cube. Here, Kociemba's algorithm is applied to determine the shortest possible series of manipulations necessary to solve the cube. The output from this solving algorithm exists as a string of characters that indicates the order and direction that the faces of the cube need to be turned. An example of this string is shown below:

$$DLD'L'F'UL'RRU'BD'UL'UR'BBR'F$$

Where the letters correspond with the faces of the cube, and a tick mark (′) indicates a counter-clockwise direction of rotation. Clockwise direction is the default direction of rotation. This string of characters is sent over a UART connection to the embedded system.

C. Performing Cube Manipulation

Once the embedded system receives the solving sequence from the software application, it deciphers the information to determine which motors need to be turned as well as the directions of rotation. The timing of the motor actuation is optimized to perform the cube manipulation as fast as possible.

D. Assessment of Completion

The embedded system will communicate to the software when the prescribed manipulations have been completed. At this time, our software can instruct the Pixy to assess the cube once again to verify that the Cube has been solved correctly. If it is necessary, the solving process can be automatically initiated again to attempt to solve the cube to completion.

VI. Conclusion

In this paper we present the design of a robotic system for autonomously solving a Rubik's Cube puzzle. Our unique implementation demonstrates application-specific hardware and software design for a high-level solution to this challenge. The contributions of our work support the ongoing quest to solve a Rubik's Cube puzzle in the fastest time possible. As a result of this project, we have gained hands-on experience with engineering design, prototyping, and production.

Acknowledgments

The authors would like to thank Dr. Samuel Richie and Dr. Lei Wei for their guidance in Senior Design I and Senior Design II.

References

[1] H. Kociemba. Cube explorer 5.12 htm and qtm. [Online]. Available: "http://kociemba.org/cube.htm"

[2] *Adafruit NEMA-17 Stepper Motor*, Adafruit. [Online]. Available: "https://www.adafruit.com/product/324"

[3] *MSP430F665x, MSP430F645x, MSP430F565x, MSP430F535x Mixed Signal Microcontrollers*, Texas Instruments. [Online]. Available: "http://www.ti.com/lit/ds/symlink/msp430f5659.pdf"

[4] *DRV8825 Stepper Motor Controller IC*, Texas Instruments. [Online]. Available: "http://www.ti.com/lit/ds/symlink/drv8825.pdf"

[5] sample author, "sample title," *Proceedings of the IEEE*, vol. 103, no. 4, pp. 665–681, 2015.