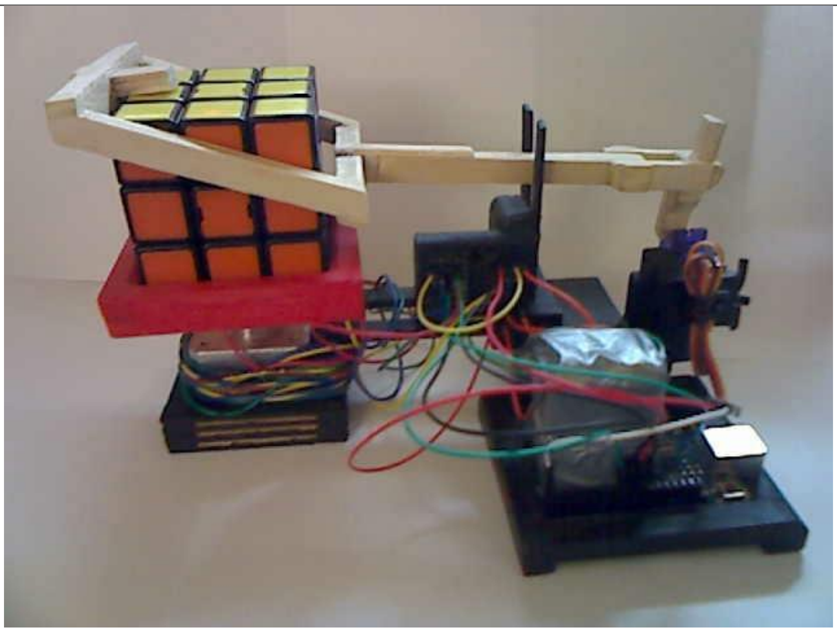
Rubik's Cube Manipulator (RCM)
By – Jules Manalang
CSE321 Fall 2014

1 Objective

The objective of the Rubik's Cube Manipulator is create a simple, compact, Arduino based robot that is able to turn the faces of the Rubik's Cube given inputs from the user.

2 Hardware Designs and Parts List

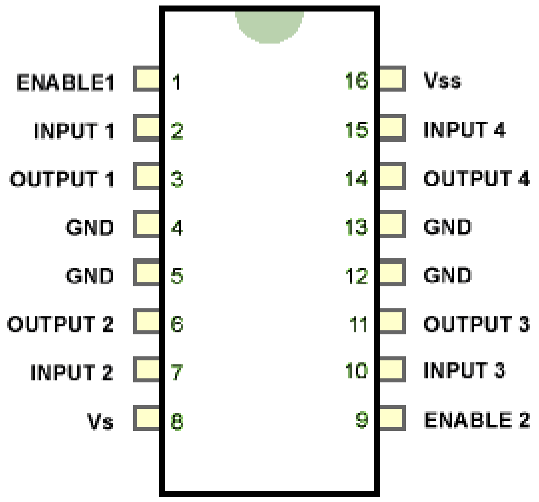


The following design is based on the MindCuber by David Gilday, a LEGO NXT Rubik's cube solver built using a singular NXT. As with the MindCuber, RCM was built using very few parts and designed as such that it required only 2 motors (a stepper motor and a servo) to be able to turn all the faces of the cube. The Following parts were used to build RCM.

Arduino UNO	Cables
Mini Breadboard	12V Bipolar Stepper Motor
Servo Motor	9v Battery
L293D or SN754410 Motor Driver (Both Interchangeable)	Popsicle Sticks

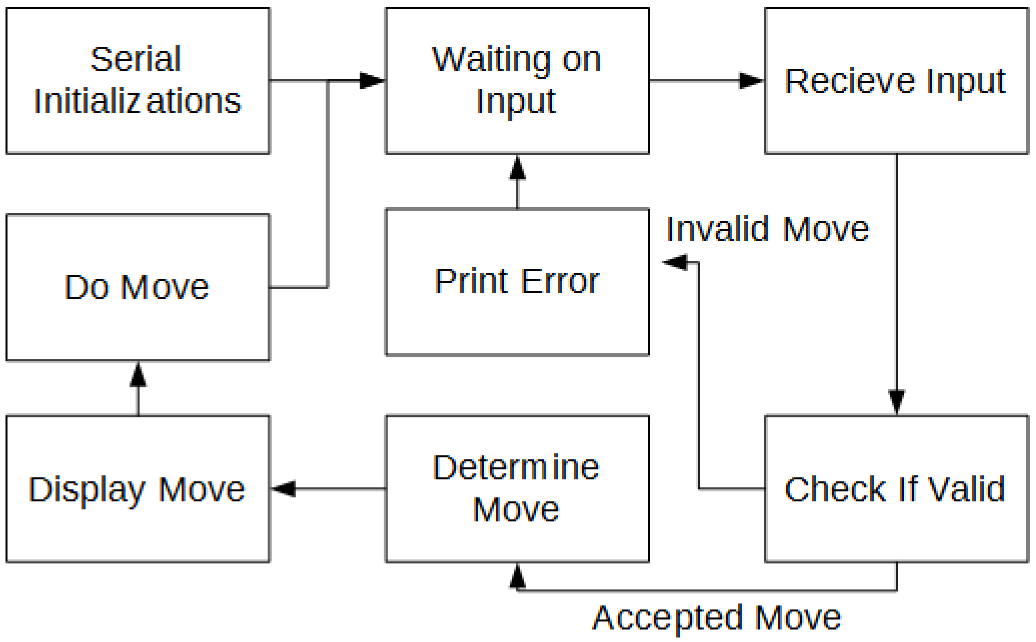
3 Wiring

The L293D Motor Driver is a Dual H Bridge controller that enables the usage of clockwise and counter clockwise coils on the Bi-Polar stepper motor used for the design. Utilizing the mini breadboard, the Motor is attached to the Driver which is then attached to the Arduino



Inputs pins are connected to the Arduino. Outputs are connected to the user. The rest of the pins are to provide electrical power to the motor.
The servo is attached to the 5V output on the Arduino and then is grounded with its input signal directly from the Arduino.

4 IMPLEMENTATION



The above flow chard describes the code. It takes advantage of the Arduino IDE's serial monitor and takes input through that using the following functions.

Serial.available();	This tells the Arduino that an input has been made through the serial monitor. This used to prevent reading something that doesn't exist
Serial.readString();	This reads whatever was input as a string.

Using this, the RCM will then determine if the input is valid and what move it should do. If the string matches a corresponding move. The RCM will then output the appropriate motor movements needed to turn that face. In order to use the motors, the following libraries were used

Servo.h	Enables servo controls
Stepper.h	Enables stepper controls.

5 Source Code

```
#include<Servo.h>     //servo library for the ARM
#include<Stepper.h>   //stepper library for the Turning Tray

String input = "";
Servo arm;                     //instantiate arm servo
Stepper tray(200, 3, 4, 5, 6); //instantiate tray motor attaches to pins 8 9
10 11 and set at 200 steps per revulution

void setup() {
  Serial.begin(9600);                    //Initialize serial communication,
baud rate to 9600
  arm.attach(7);                         //attach servo to pin 7 enabling
control
  arm.write(50);                         //default "up" postition for the arm
  tray.setSpeed(90);                     //set the speed of stepper motor to
be 90PRM
  tray.step(0);                          //send pinout so torque on motor
increases
}

void loop() {
  if (Serial.available()>0){            //waits for user input in the serial
window.
    input = Serial.readString();        //read the user input as a string
    if(input == "U"){                   //compares the input with Valid
Inputs
      Serial.print("U Move \n");        //prints out what move is made
      tilt();                           //series of tilts holds and twists to
achieve that move
      tilt();
      hold();
      oCW();
      tilt();
      tilt();
      up();
    }
    else if(input == "U'"){           //the rest of this void loop is compare 
and dos.
      Serial.print("U' Move \n");
      tilt();
      tilt();
      hold();
      oCCW();
      tilt();
      tilt();
      up();
    }
    else if(input == "U2"){
      Serial.print("U2 Move \n");
      tilt();
      tilt();
      hold();
      oDT();
      tilt();
      tilt();
      up();
    }
    else if(input == "F"){
      Serial.print("F Move \n");
      tilt();
      hold();
      oCW();
      up();
      DT();
      tilt();
      up();
      DT();
    }
    else if(input == "F'"){
      Serial.print("F' Move \n");
      tilt();
      hold();
      oCCW();
      up();
      DT();
      tilt();
      up();
      DT();
    }
    else if(input == "F2"){
      Serial.print("F2 Move \n");
      tilt();
      hold();
      oDT();
      up();
      DT();
      tilt();
      up();
      DT();
```

```
}
else if(input == "R"){
  Serial.print("R Move \n");
  up();
  CCW();
  tilt();
  oCW();
  up();
  DT();
  tilt();
  up();
  CCW();
}
else if(input == "R'"){
  Serial.print("R' Move \n");
  up();
  CCW();
  tilt();
  oCCW();
  up();
  DT();
  tilt();
  up();
  CCW();
}
else if(input == "R2"){
  Serial.print("R2 Move \n");
  up();
  CCW();
  tilt();
  oDT();
  up();
  DT();
  tilt();
  up();
  CCW();
}
else if(input == "B"){
  Serial.print("B Move \n");
  up();
  DT();
  tilt();
  hold();
  oCCW();
  up();
  DT();
  tilt();
  up();
}
else if(input == "B'"){
  Serial.print("B' Move \n");
  up();
  DT();
```

```
    tilt();
    hold();
    oCW();
    up();
    DT();
    tilt();
    up();
  }
  else if(input == "B2"){
    Serial.print("B2 Move \n");
    up();
    DT();
    tilt();
    hold();
    oDT();
    up();
    DT();
    tilt();
    up();
  }
  else if(input == "L"){
    Serial.print("L Move \n");
    up();
    CW();
    tilt();
    hold();
    oCW();
    up();
    DT();
    tilt();
    up();
    CW();
  }
  else if(input == "L'"){
    Serial.print("L' Move \n");
    up();
    CW();
    tilt();
    hold();
    oCCW();
    up();
    DT();
    tilt();
    up();
    CW();
  }
  else if(input == "L2"){
    Serial.print("L2 Move \n");
    up();
    CW();
    tilt();
    hold();
    oDT();
    up();
    DT();
    tilt();
    up();
    CW();
  }
  else if(input == "D"){
    Serial.print("D Move \n");
    hold();
    oCCW();
    up();
  }
  else if(input == "D'"){
    Serial.print("D' Move \n");
    hold();
    oCW();
    up();

  }
  else if(input == "D2"){
    Serial.print("D2 Move \n");
    hold();
    oDT();
    up();
  }
  else if(input == "x"){
    Serial.print("x rotation \n");
    tilt();
    up();
  }
  else if(input == "x'"){
    Serial.print("x' rotation \n");
    up();
    DT();
    tilt();
    up();
    DT();
  }
  else if(input == "x2"){
    Serial.print("x2 rotation \n");
    tilt();
    tilt();
    up();
  }
  else if(input == "y"){
    Serial.print("y rotation \n");
    up();
    CCW();
  }
  else if(input == "y'"){
    Serial.print("y' rotation \n");
    up();
    CW();
    }
    else if(input == "y2"){
      Serial.print("y2 rotation \n");
      up();
      DT();
    }
    else if(input == "z"){
      Serial.print("z rotation \n");
      up();
      CCW();
      tilt();
      up();
      CW();
    }
    else if(input == "z'"){
      Serial.print("z' rotation \n");
      up();
      CW();
      tilt();
      up();
      CCW();
    }
    else if(input == "z2"){
      Serial.print("z2 rotation \n");
      up();
      CCW();
      tilt();
      tilt();
      up();
      CW();
    }
    else{
      Serial.print("Invalid \n");       //if the input is not valid prints
out invalid and waits for input again
    }
  }
}
//End Loop
```

```
/*=================================================================
MOTOR MOVEMENT CODE
tilt       -  tilts the cube
hold       -  makes the arm hold the cube
up         -  raises the arm
(o)CW      - (overdrive) clockwise quarter turn
(o)CCW     - (overdrive) counter clockwise quarter turn
(o)DT      - (overdrive) half turn direction doesn't matter
Overdrive  - While turning, the cube resists the force of the motor.
              This causes the tray to not complete its motion,
              while in code it has. To make up for this, overdrive
              is introduced to add 1 or 2 more steps ensuring that
              intended motion is completed
=================================================================*/

void tilt(){
  arm.write(85);
  delay(700);
  arm.write(175);
  delay(700);
  arm.write(80);
  delay(400);
  arm.write(90);
  delay(400);
}

void hold(){
  arm.write(90);
  delay(1000);
}

void up(){
  arm.write(50);
  delay(1000);
}

void CCW(){
  tray.step(-50);
  delay(500);
}

void CW(){
  tray.step(50);
  delay(500);
}

void DT(){
  tray.step(10);
  delay(500);
}

void oCCW(){
  tray.step(-51);
  delay(500);
}

void oCW(){
  tray.step(51);
  delay(500);
}

void oDT(){
  tray.step(102);
  delay(500);
}
```

6 USAGE

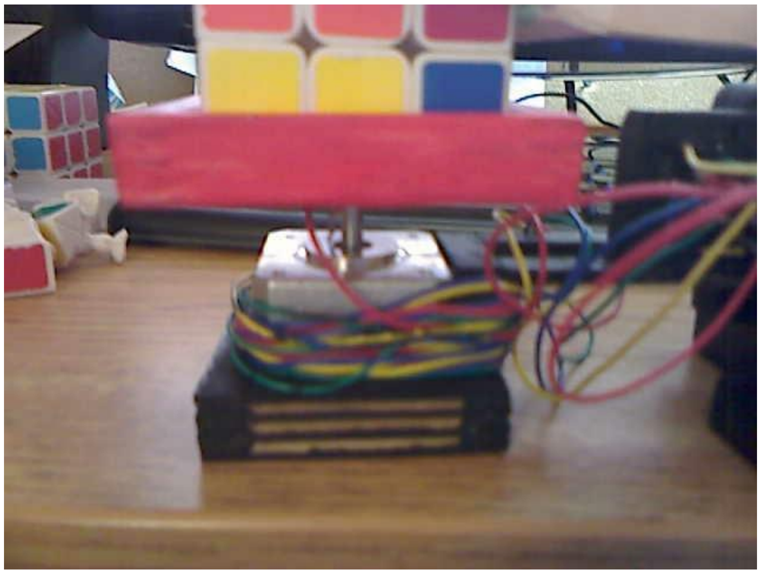
RCM takes in the standard World Cube Association turning notation.

- U F R L B D U’ F’ R’ L’ B’ D’ U2 F2 R2 L2 B2 D2 x y z x’ y’ z’ x2 y2 z2
- U, U’, U2 => Top Side Quarter Clockwise, Quarter Counter-clockwise, Half turn.
- F – Front, R – Right, L – Left, B – Back, D – Bottom
- x y z are full cube rotations with x following R, y following U and z following F in terms of direction

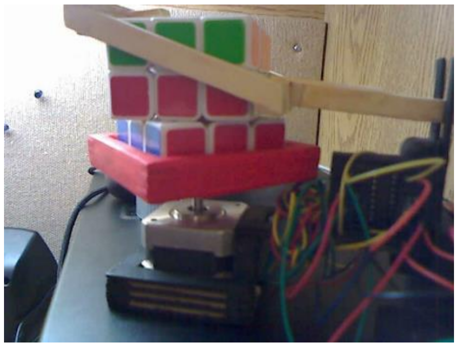
After uploading the code the Arduino, access the Arduino IDE’s serial monitor and change to the end line setting to “No Line Ending”

Then, type in what move is desired. These are case sensitive. All turns are uppercase whereas rotations are lower case.

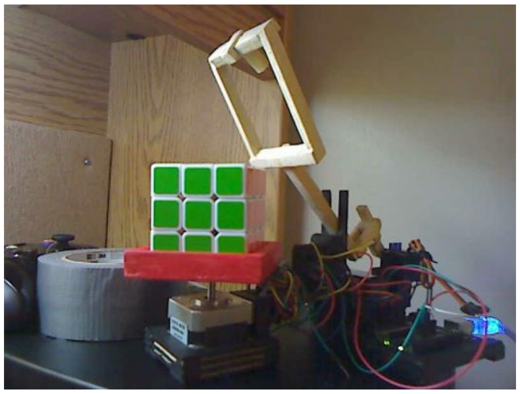
7 PHYSICAL MOVEMENTS



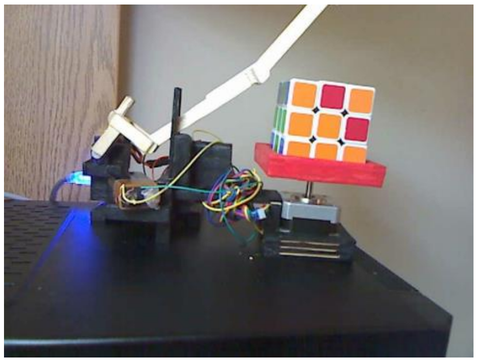
The 12V stepper motor is used to the turn the cube using the holding tray. This means that the only face that is able to turn is the one in the tray.



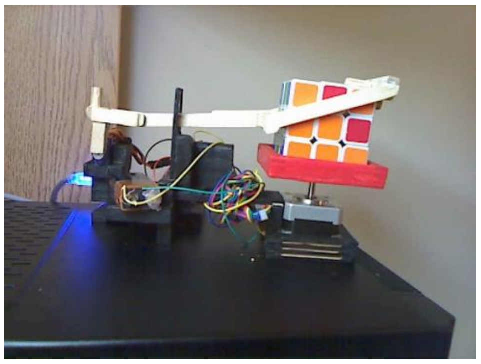
It can also rotate the whole cube when the holding arm is in the up position.



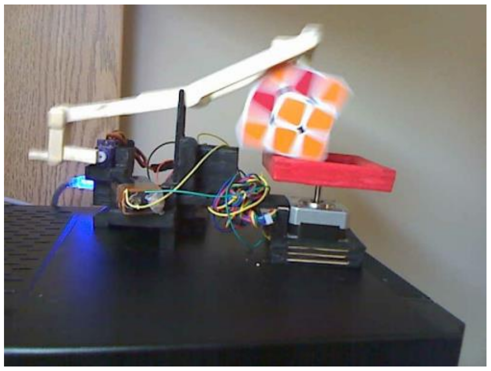
Moving onto the arm, the arm has 3 main functions; to hold the cube, release the cube, and the tilt the cube. The best illustration for all 3 is inputting x rotation into the serial monitor.



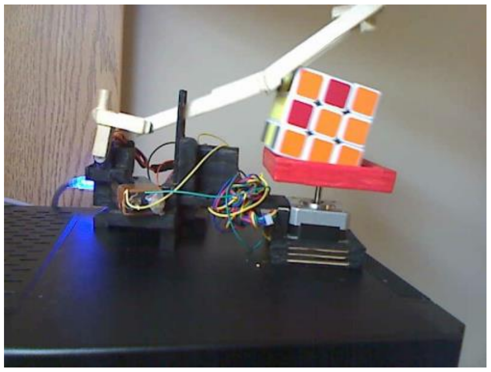
Arm is up by default (Up function)



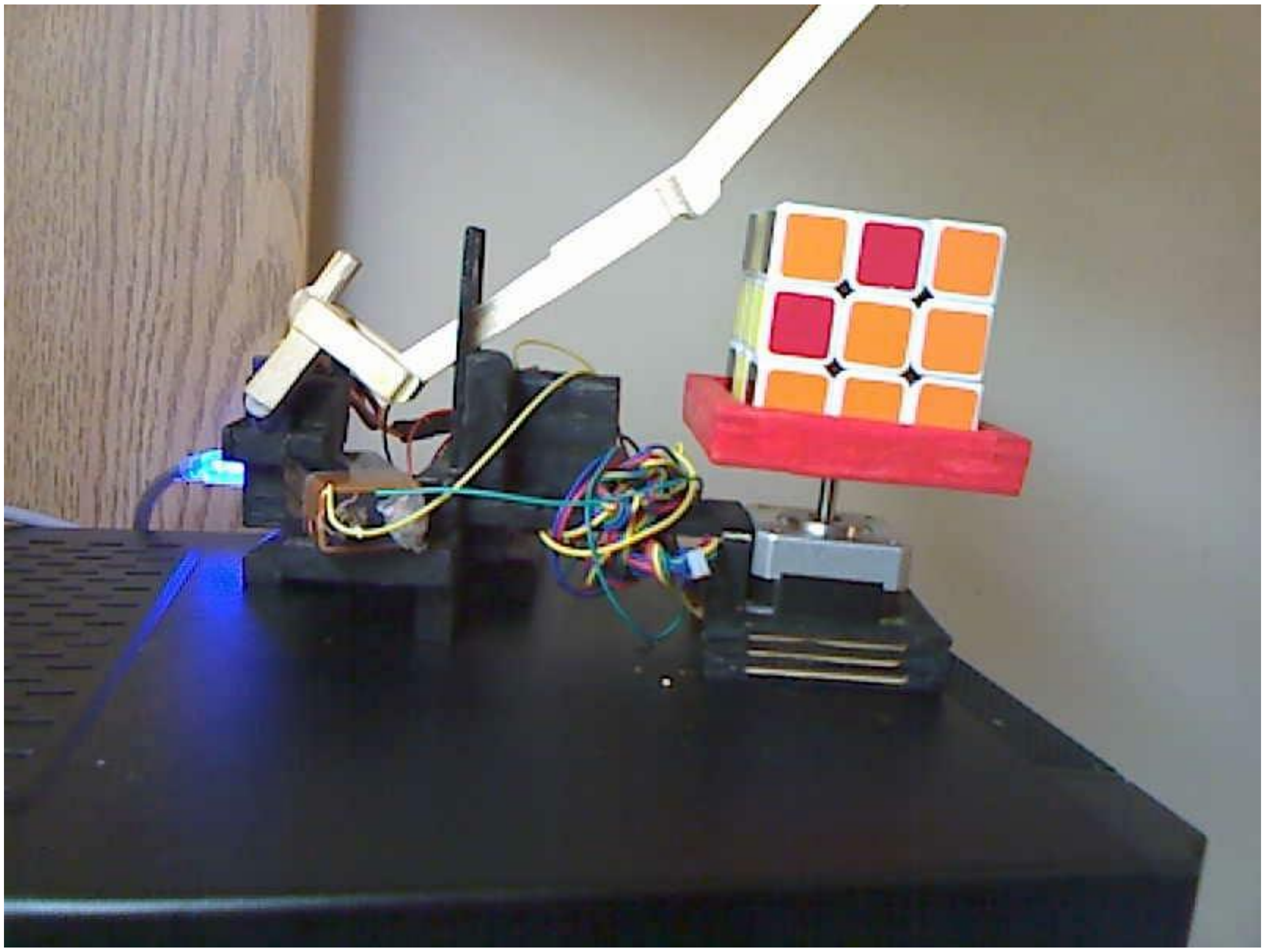
Arm is now holding the cube (hold function)



Arm tilts the cube (tilt function)



Arm pushes cube on back on the tray



The arm then returns back to its default up position.

RCM has access to 2 out of 3 axis which means that it rotate the cube so that ANY face will be on the turning tray. This in turn means that RCM is fully capable of turning all the faces of the Rubik's Cube.

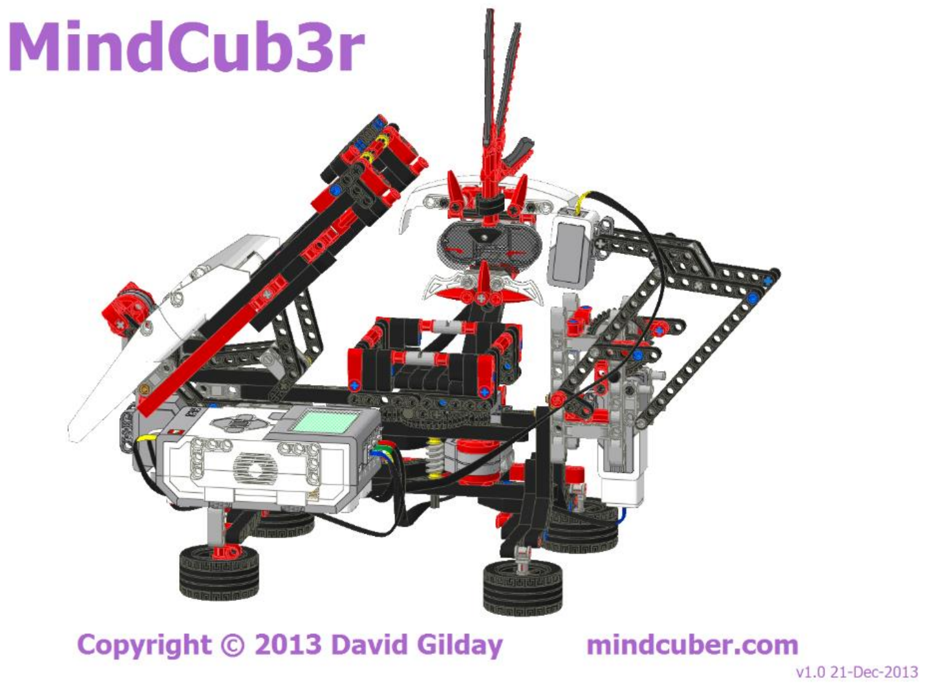
8 Previous Designs and Ideas

It is enough to make a cube robot that's huge and has many motors. As an added challenge, the design should use as little number of parts as possible as and smaller than a sheet of paper. Originally, RCM was supposed to a Rubik's Cube solver controlled using only 2 Servos. As such the need to the motor driver and external battery were nonexistent. The code implemented was only for two servos. And it was until the introduction of the stepper motor.

As for the solving portion of the code, originally implemented was Herbert Kociemba's 2-Phase Algorithm to find solutions that were less than 25 moves long. However, the code that was implemented exceeded the memory limit on the Arduino also without a powerful processor it would have taken about

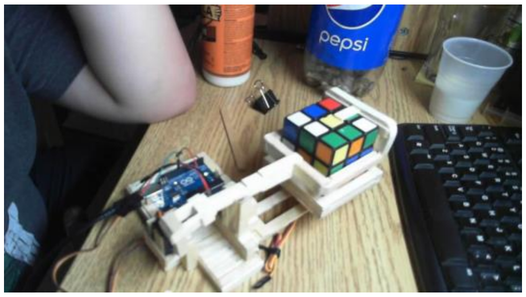
2-5 minutes to find a solution and was scrapped as well the intended web camera color input as the design no longer called for them.

This design was based on David Gilday's MindCuber, a LEGO NXT cube solver.



The first iteration of the design was narrow but extremely long at about 45cm or about a 1.5 feet. It was not elevated at all. In fact, the servos were touching the surface. This made the center of gravity of the design extremely low making it stable. The added length also helped with stability when the arm was moved. It was lightweight and took maybe 30 Popsicle sticks to make. However, the overall size goal was less than sheet of paper as such the design was scrapped

The iteration of the design changed how the arm hinged and at what angle this cut the length to fit within the sheet of paper originally ended. The Arduino mount was also moved off to the side. This made the whole thing fit within half a sheet of paper. Adding more weight to the frame added extra stability to make up for the lack of length. All in all took about 80 Popsicle sticks.



This iteration however lacked flexibility a full motor. The Servo had only 180 degrees of movement meaning, whatever move it had to do. It had to undo it later. While it was possible, it made creating functions for turning a difficult task.

To ease the problem of difficult functions, the turning tray servo was replaced with a Stepper Motor. This meant also adding the L293D motor driver and breadboard into the design. The breadboard was attached to the frame of the RCM. The motor makes the RCM heavier than the rest of the iterations. With the addition of the Stepper Motor the turning tray was redesigned to fit on the stepper shaft. Since the tray was raised quite a bit, the arm was also raised to be able to be functional. An external battery was added to the design to power the motor.

This was the final iteration for the RCM resulting in the picture on page 1.

9 Final Words

Despite being the final project for CSE 321, I feel as if this is only the beginning for this little project of mine. The RCM still has a long way to go from being what I want to be, a Rubik's Cube solver. There's so many things in the code to optimize and changes to hardware to make it run faster and better. These are limited by time rather than disinterest to continue. While it failed utterly as cube solver, I am quite happy with its performance as simply a manipulator.

10 Question and Comments

Contact Jules Manalang - julesman@buffalo.edu