

MECCSCARA ROBOT

CONCEPT SELECTION AND DETAIL DESIGN REPORT

Dragon Engineering Solutions, Inc.

Dominic Silva

Iriona Gravley

Jianhao Lin

Foufouo Marielle Flora

May 28th, 2025

Philadelphia, PA



Table of Content

1.	INTRODUCTION	3
2.	CONCEPT GENERATION AND SELECTION	4
3.	TECHNICAL SPECIFICATIONS.....	5
4.	BILL OF MATERIALS (BOM)	7
5.	ENGINEERING DRAWINGS	8
6.	CODE AND ELECTRONICS	12
7.	WIRING DIAGRAM	13
8.	ASSEMBLY INSTRUCTIONS	14
9.	Appendix	17

1. INTRODUCTION

Our project aims to design and develop a cost-effective, high-performance SCARA robot. The design we have created enhances productivity and eliminates the issues of human error in repetitive tasks. To complete the design process of this product, we identified the main problem, brainstormed potential solutions, selected the best idea, tested our product, and refined our final design. The need for innovation in automation directly aligns with our project's goals and solutions our SCARA robot is designed to deliver. As technology continues to advance, industries increasingly rely on automation to meet the demands regarding efficiency and cost reduction. Offering an accurate, affordable robot that large companies can use to complete repetitive tasks. The SCARA robot we have designed improves the efficiency of these tasks by increasing the load capacity, accurate precision, and consistent output. Doing so will allow industrial companies to implement an affordable solution in settings such as assembly lines, packaging, and material handling. Additionally, the design allows for the control system to adapt to future upgrades and adjustments that even people with little experience are able to modify. By combining affordability, precision, and user-friendly adaptability, our SCARA robot provides a system that adapts to the demands of large-scale manufacturing.

2. CONCEPT GENERATION AND SELECTION

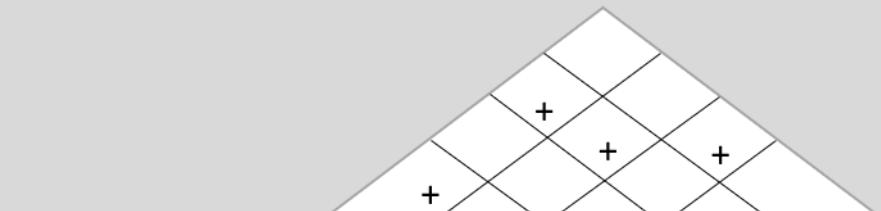
Morphological Chart

Design Problem: Scara Robot to Move 1-inch Cubes

Selected combination of functions used for final design: Highlighted in Blue

FUNCTION	SOLUTIONS		
Power	USB	Battery	
Move joint	Servo Motor	Stepper Mottor	
Z motion	Lead Screw	Rack and Pinion	Linear Actuator
End Effector	Finger Gripper	Vaccum Gripper	
User Interface	Serial Monitor	key pad	Web Interface (buetooth)
Control Logic	typed input	hard written program	button sequence
Home Position	Limit Switch	Gyroscope	

Quality Function Deployment



The house diagram illustrates the relationships between functional requirements and customer requirements. The top horizontal row shows correlations: + (Positive), + (Positive), + (Positive). The bottom horizontal row shows correlations: + (Positive), + (Positive), + (Positive).

Desired direction of improvement ($\uparrow, 0, \downarrow$)							Correlation:				
Customer importance rating 1: low, 5: high	Functional Requirements → Customer Requirements \downarrow	Energy-Efficient Gear	Proper Arm Dimensions	Arduino Controller	Stable Joints	Limit Switch For Homing	+	+	-		
		Smooth Movement	9	3	3	9	3	108	4	5	
		High Precision	1	3	9	9	9	155	4	5	
		Easy to Control			9		3	36	4	3	
		Stability	1	3	1	9	1	75	4	5	
		Less Space Occupied		9	3			24	3	2	
		Low Cost	1	3	9		3	80	5	2	
		Low Weight		9	1	3	1	14	3	3	
		Technical importance score	51	84	141	129	87	492			
		Importance %	10%	17%	29%	26%	18%	100%			
		Priorities rank	5	4	1	2	3				
		Current performance	2	4	3	2	1				
		Target	5	4	4	5	4				
		Difficulty	5	2	3	4	2				
Competitive evaluation (1: low, 5: high)											
		Weighted Score	Our Robot Rating	Yamaha Robot Rating	Epson Robot Rating						

1: very easy, 5: very difficult

The design is the most suitable because it addresses the most important customer requirements, such as the cost, the precision, and the stability. It outperforms most competitors mainly in terms of its cost efficiency and difficulty of use, through using components such as the Arduino controller. Although the performance of our robot is not the best among our competitors, it offers the best balance of overall cost and usability.

3. TECHNICAL SPECIFICATIONS

ROBOT SIZE

The total dimensions of our robot are as follows (close approximations):

Width: 5 inches

Length: 7 inches

Height: 7 inches

It should be noted that the width and length do not account for the arm reach of the robot, only the dimensions of the main base/body. With our prototype being made from lightweight PLM, it is fairly maneuverable and lightweight. Obviously, this will change as the final product is developed with much more sturdy, reliable materials like steel.

ARM REACH

Our SCARA robot will have an approximate reach of 6 inches when both arm segments are fully extended. The prismatic joint has around 7 inches of motion along the z axis, allowing us to pick up objects as far down as the base of the robot.

POWER REQUIREMENTS

The selected stepper motors used in our design require 5V of power to operate, similar to most other electrical components used in the model. This will be supplied by our Arduino MEGA microcontroller which also contains the program for said motors. As for the power needed to deliver the weight of itself and the item, the motors are rated at 300 gram-force centimeters. The parts we used in our prototype are very lightweight, and so too is the load we will be carrying, so we are not worried about the possibility that the motors cannot deliver enough power to operate the robot.

EXPECTED PERFORMANCE

The SCARA robot should be able to pick up block nearly 100% of the time, meaning the vacuum end effector's programming and functionality must be extremely accurate and precise. There would be no point in making a robot which failed the first step of picking the object up. As for the motion and placement of items, we are hoping to obtain an accuracy of within 0.5 inches, meaning that when placed, the item should not be more than half an inch from the intended target. This level of accuracy for a prototype is difficult, yes, but we hope that troubleshooting and debugging after testing can get us closer towards this goal. With the first and second revolute joints, the robot will have a full 360 degrees of motion, meaning any target within it's reach should be movable, and the prismatic joint should allow the pickup of any object as far down as the base of the robot.

TOLERANCES

See engineering drawings below

4. BILL OF MATERIALS (BOM)

The estimation of the 3D printed parts were done with the help of a 3D printing Calculator Tool, link: [Makershop | 3D Print Cost Calculation Tool](#)

PART	ESTIMATED COST PER UNIT (\$)	QUANTITY	SUPPLIER	PART NUMBER/LINK
SCARA BASE	20	1	3D PRINTED BY DREXEL'S INNOVATION STUDIO	-
SCARA PLANETARY GEAR BASE	5	1		-
GEARS	2	5		-
RACK	2	1		-
INTERNAL GEAR	3	1		-
STEPPER MOTORS	2.3	3	Drexel College of Engineering	28BYJ-48
BREAD BOARD	0.91	1	Drexel College of Engineering	Jumper & breadboard
MICRO-CONTROLLER and USB	22.99	1	Drexel College of Engineering	USB Cable & Arduino
LIMIT SWITCHES	0.60	3	Drexel College of Engineering	K12-3
WIRES	0.06	30	Drexel College of Engineering	Jumper & breadboard
SERVO CONTROLLERS	6.66	3	Drexel College of Engineering	ULN2003
VACUUM GRIPPER	13.99	1	Drexel College of Engineering	Vacuum Gripper
Power Adaptor	6.99	1	Drexel College of Engineering	Power Adaptor

Total Cost: \$115.37

5. ENGINEERING DRAWINGS

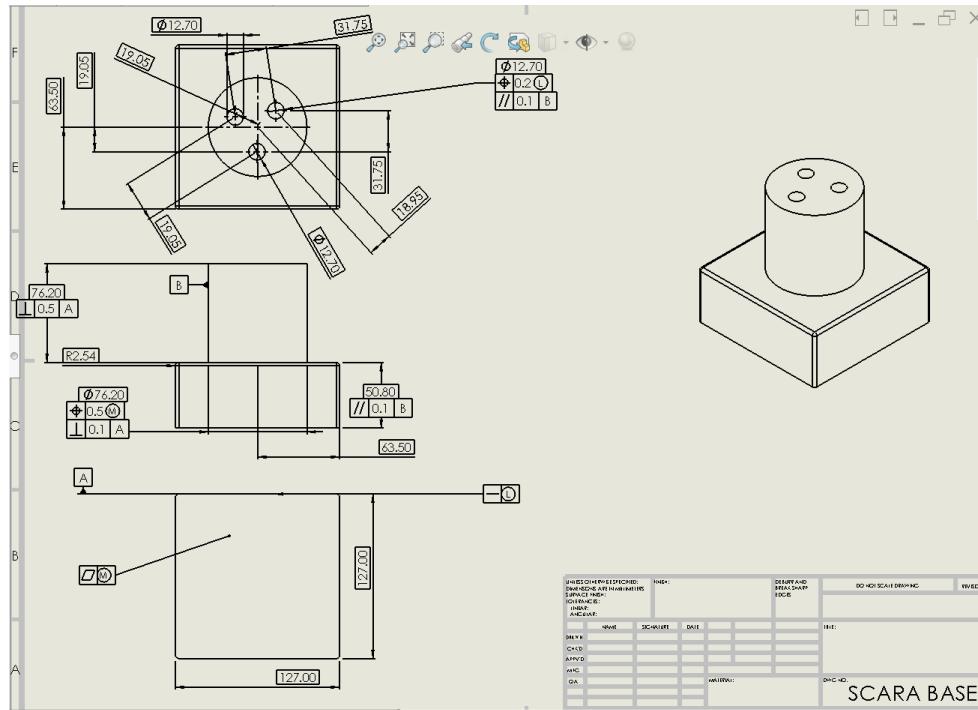


Figure 5.1: Engineering Drawing of SCARA BASE (millimeter precision)

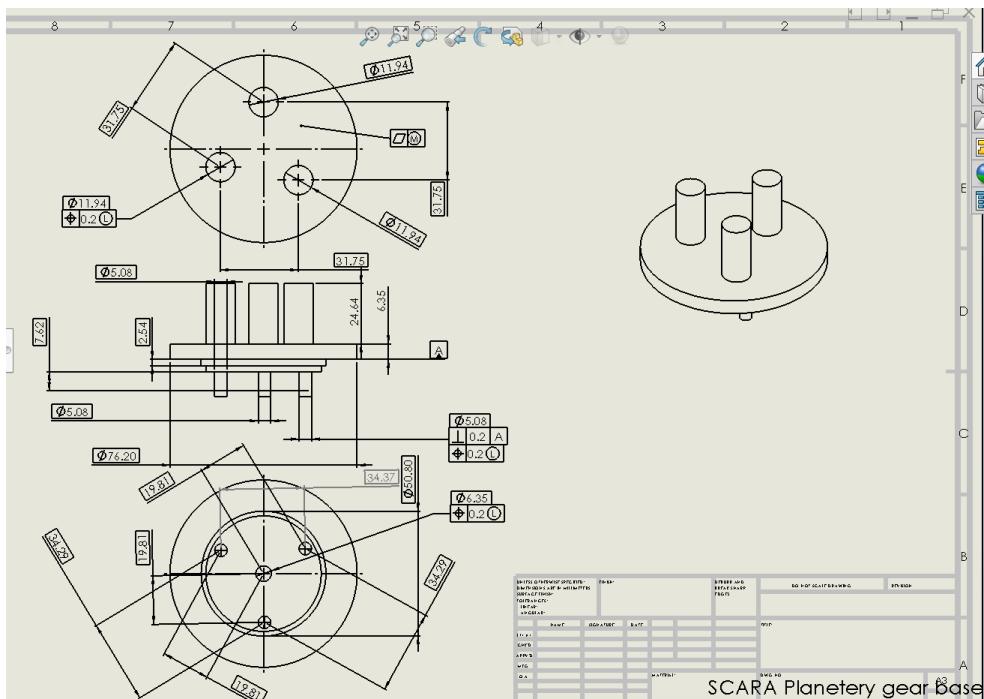


Figure 5.2: Engineering Drawing of SCARA Planetary Gear Base (millimeter precision)

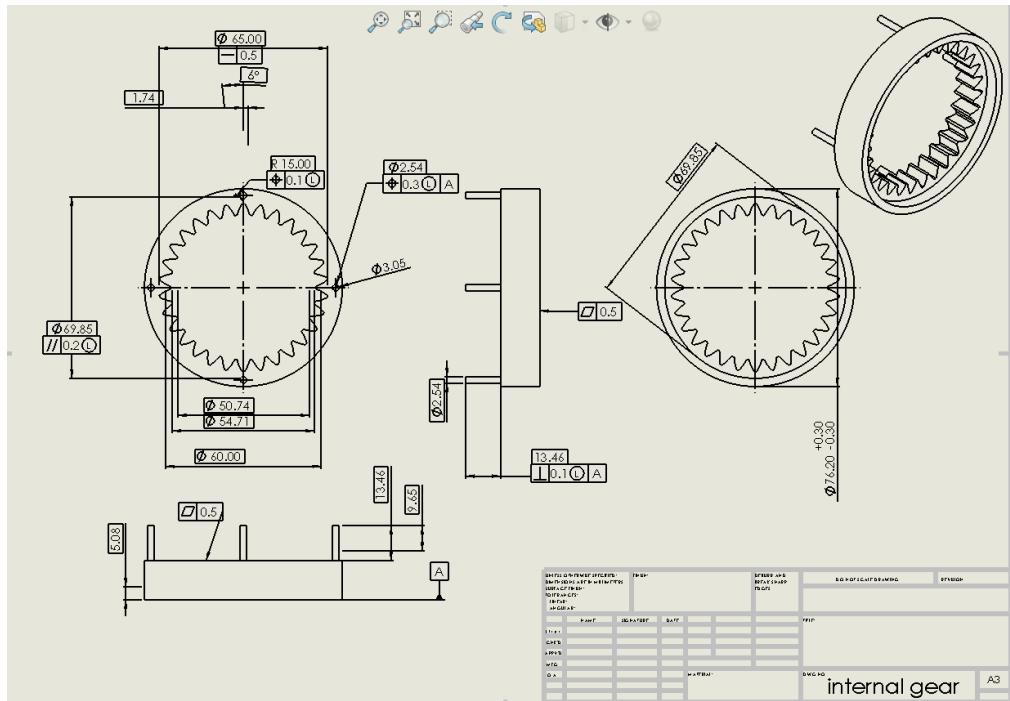


Figure 5.3: Engineering Drawing of SCARA Internal Gear (millimeter precision)

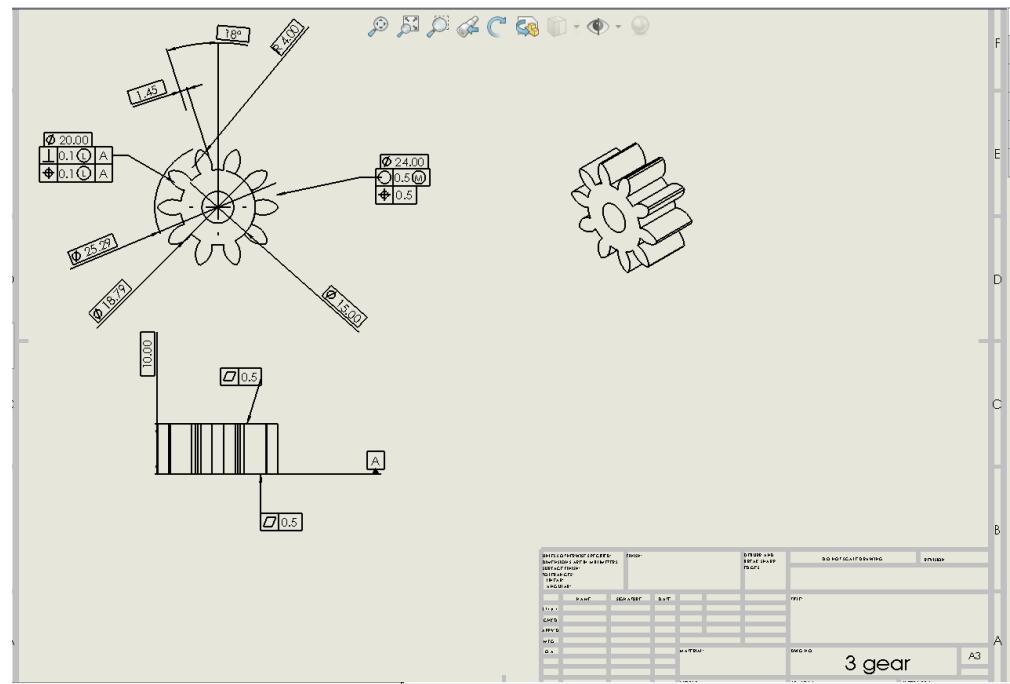


Figure 5.4: Engineering Drawing of SCARA Gear (millimeter precision)

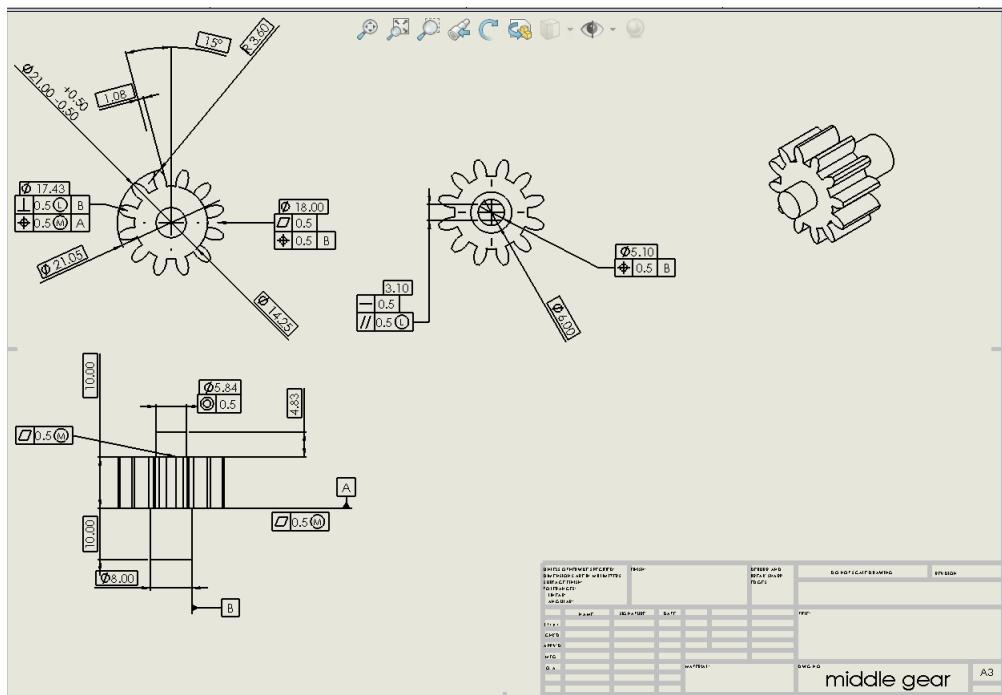


Figure 5.5: Engineering Drawing of SCARA Middle Gear (millimeter precision)

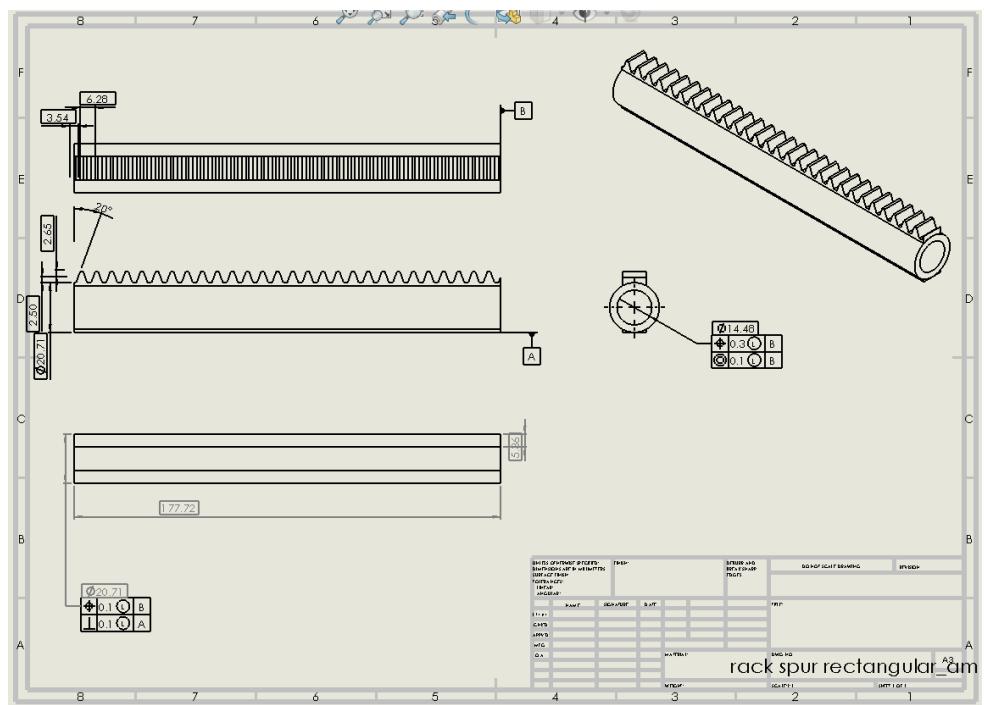


Figure 5.6: Engineering Drawing of SCARA Rack (millimeter precision)

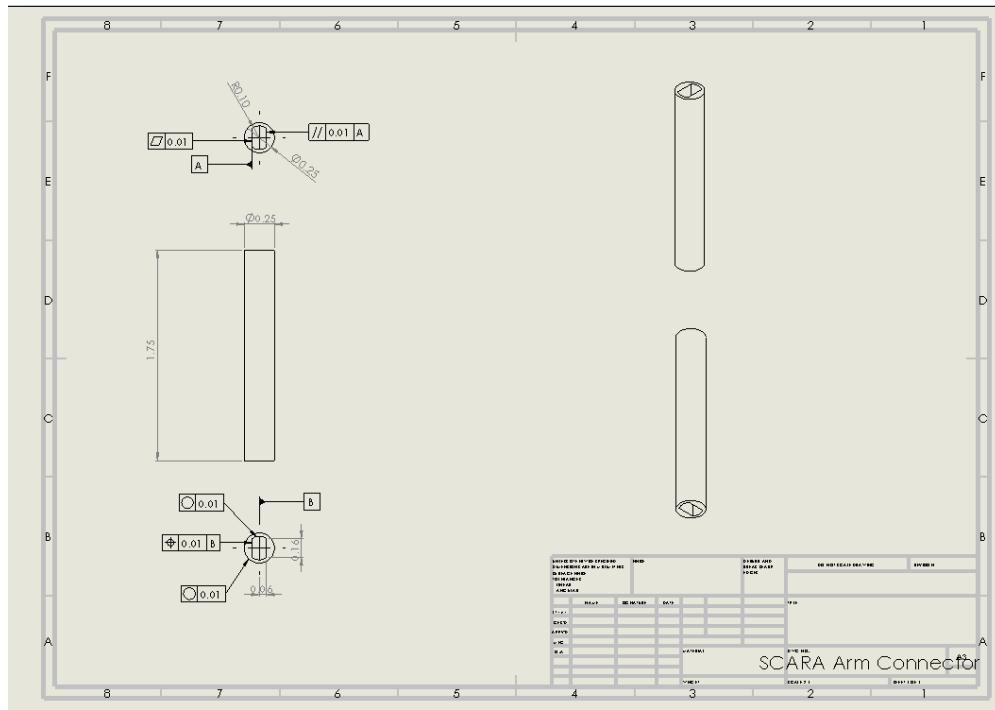


Figure 5.7: Engineering Drawing of SCARA Arm Connector (inch precision)

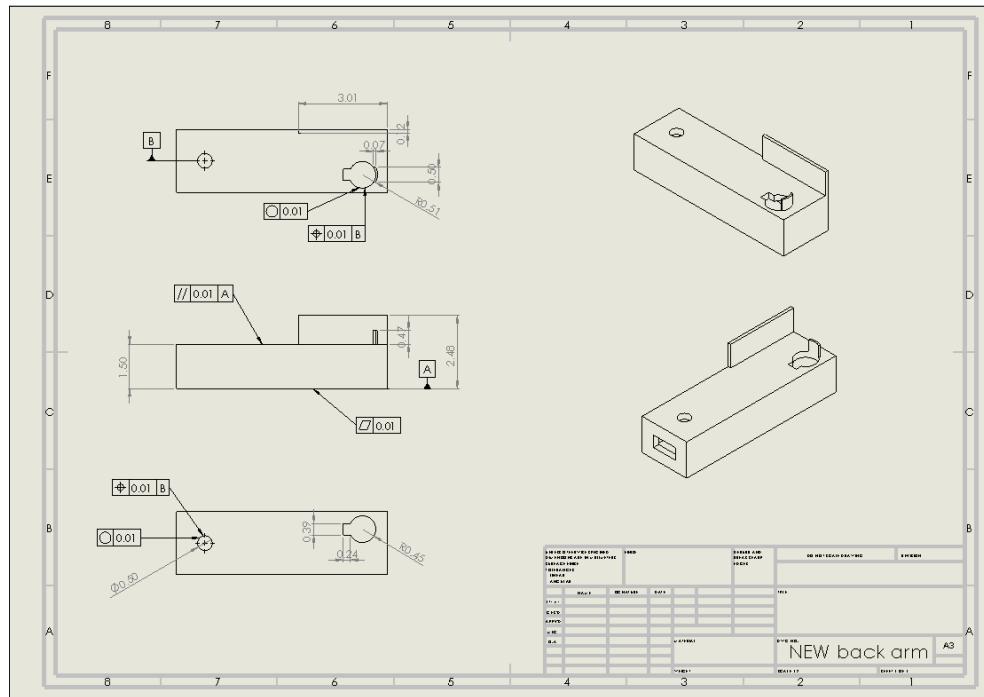


Figure 5.8: Engineering Drawing of SCARA Rear Arm (inch precision)

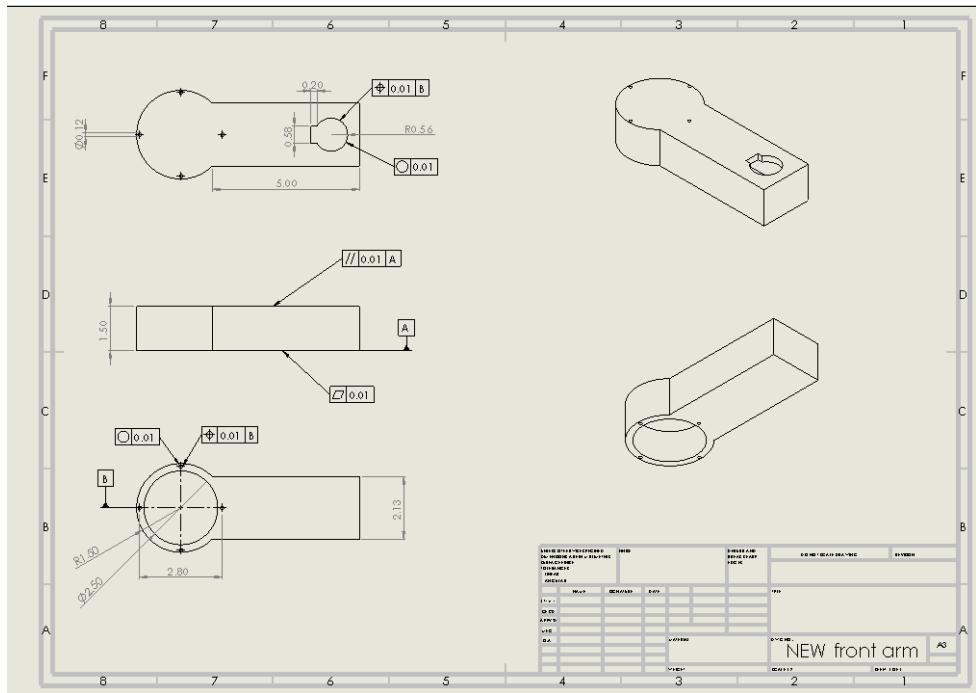


Figure 5.9: Engineering Drawing of SCARA Front Arm (inch precision)

6. CODE AND ELECTRONICS

This is a pre-programmed sequence of code that takes no inputs. Every time the program is launched, the robot will first move in the direction of the limit switch and stop when the switch is pressed and moves to the opposite direction a set number of steps to home. Then, it moves to the desired position and picks up the block, moving to another and places down the block.

Lines 1-2: The stepper motor and math library are imported.

Lines 3-26: Variables are initialized, and pins are assigned to stepper motors' wires.

Lines 28-38: Motor speeds are set to 10, limit switch pins are declared, and the home position function is called.

Lines 40-42: The control block of the program. The block is currently empty but will be hard programmed and call the “moveTo” function when the sequence of arm movements are decided.

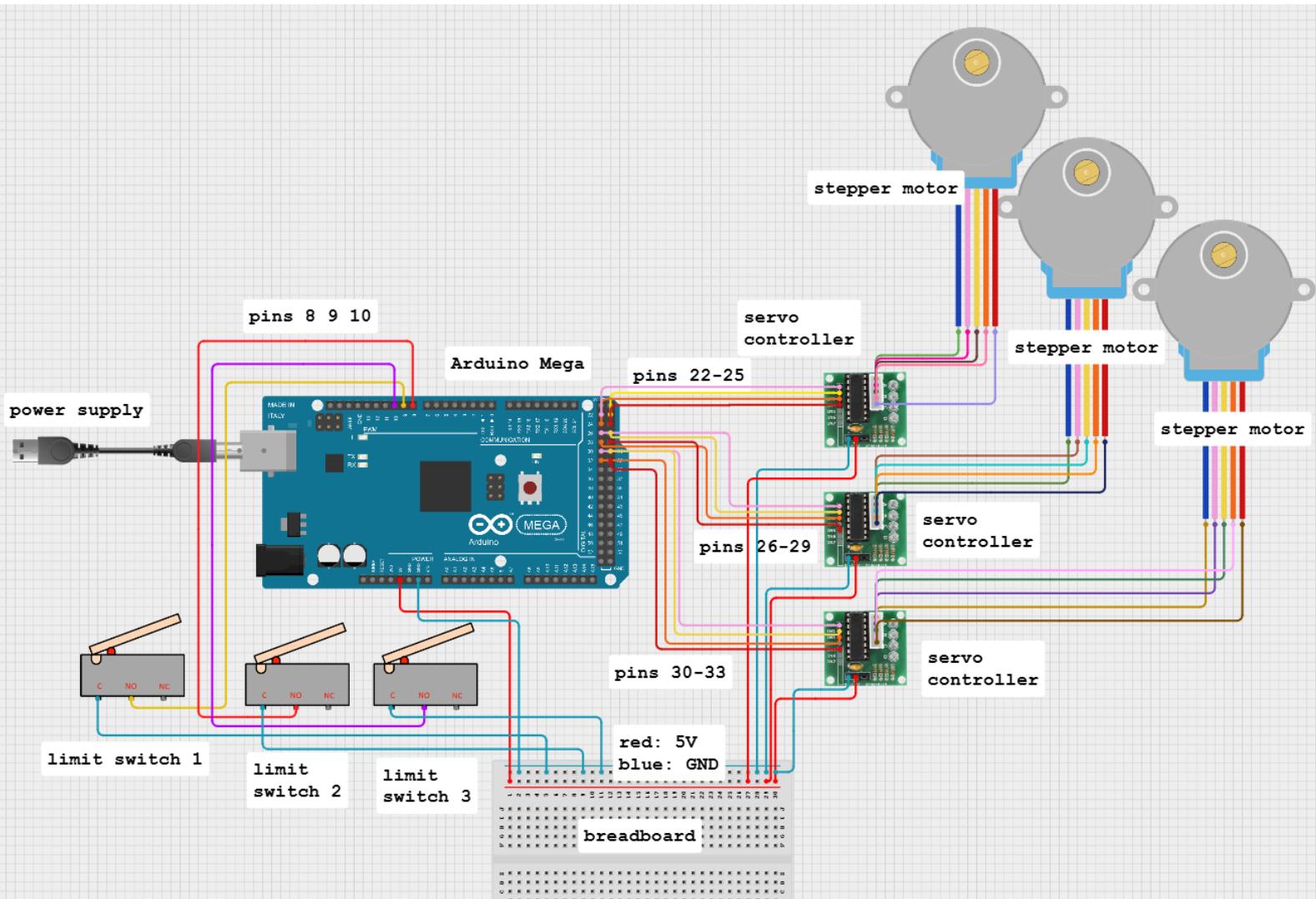
Lines 44-89: The function that moves each stepper a specific number of steps simultaneously.

Lines 91-110: The home position function moves each stepper until all of them hits their limit switch and then moves all of them a set number of steps to the opposite direction.

Lines 112-119: The inverse kinematics function calculates the angle when given a target position.

Lines 121-129: This function moves the robot arm to a position; the angles are calculated from their current position.

7. WIRING DIAGRAM



8. ASSEMBLY INSTRUCTIONS

TOOLS NEEDED

Drill, 2mm screws, glue/adhesive

STEPS

Download required part files, convert to STL, and 3D print

- a. Part and # needed → Base (1), Planetary gear base (1), Planetary gear ring (1), sun gear (1), planet gears (3), 1st arm segment (1), Arm connector (1), 2nd arm segment (1), Rack (1)
- 2.) Start with the base by placing it down on your workspace (A)
- 3.) Insert the axle of one of the stepper motors into the bottom of the planetary gear base and proceed to insert the planetary gear base simply by sliding the 3-cylinder configuration into the corresponding holes in the base (B)
- 4.) Insert the sun gear into the middle of the base (C)
- 5.) Put the corresponding three planetary gears into the three slots surrounding it (D)
- 6.) Insert the planetary gear ring onto the base surrounding the smaller gears. Some wiggling into place may be required, as the planetary gears may block it. (E)
- 7.) Using the 4 holes which link with the 4 posts on the planetary ring, slide the first arm segment, the one with the rounded back, into place on top of the planetary gear setup (F)
- 8.) Place the second stepper motor facing up into the hole at the end of the arm segment. Use screws and a drill to fix the motor into the arm at the two drill points on either side of the motor.
- 9.) Grab the arm connector and slide it over the axle of the stepper motor. Use superglue or some other adhesive to fix it in place
- 10.) Get the second arm segment and slide the connector though the two circular holes on one side of the arm. Again, use superglue or adhesive to fix the connector to the arm segment. (G)
- 11.) This last step deals with placing the rack and pinion. Grab the last gear and glue it to the axle of the last stepper motor. Grab the rack and slide it into the other hole on the opposite side of the arm segment. Hold it in place while (preferably another person) slides the stepper motor, back against the wall of the arm, until the gear teeth on the motor reach the teeth of the rack. At that spot, glue the back of the motor to the wall (wires facing up) and the rack should now be in place. (H)
- 12.) Place the breadboard and Arduino MEGA along the top of the first arm segment and glue them down. Use the wiring guide provided above to wire the robot. The limit switches should be placed on both sides of the second arm segment and one more at the end of the rack using glue again.

PHOTO A

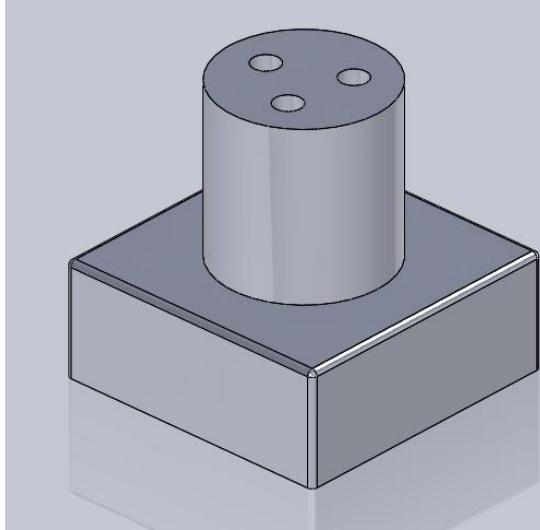


PHOTO B

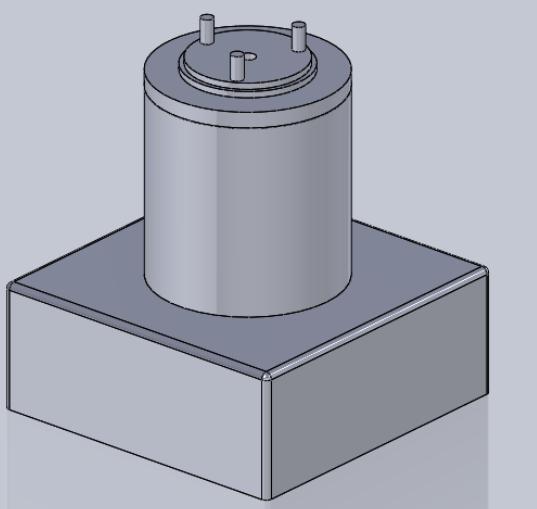


PHOTO C

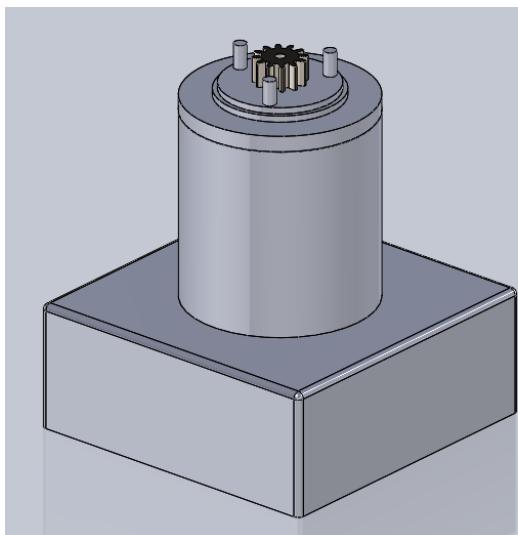


PHOTO D

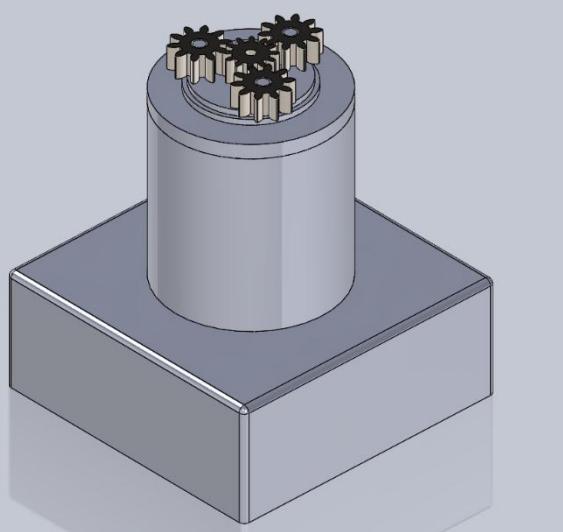


PHOTO E

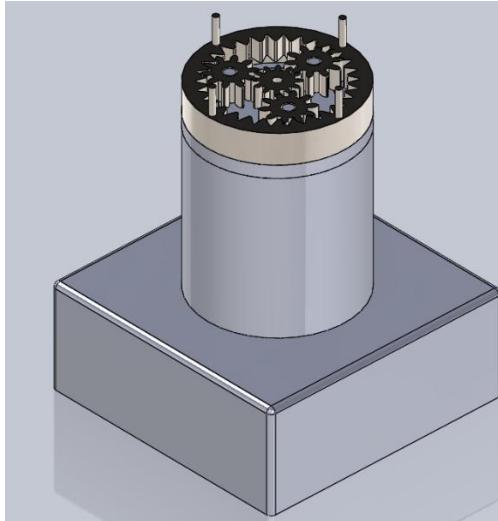


PHOTO F

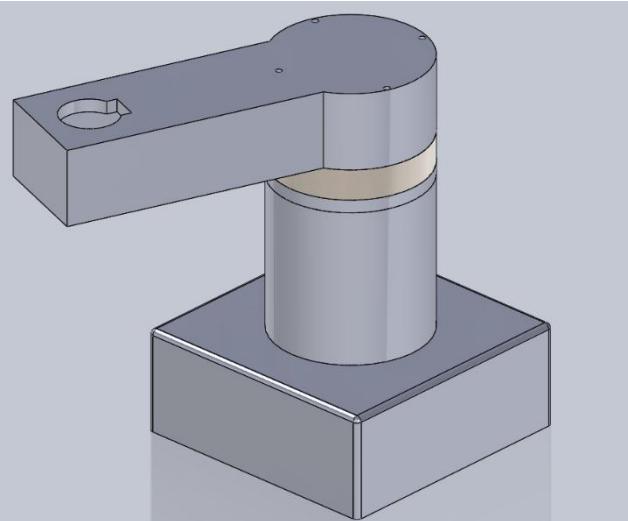


PHOTO G

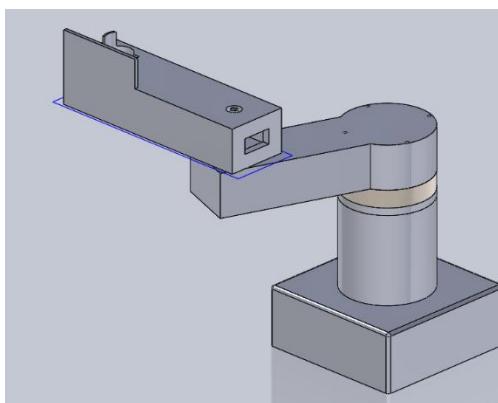
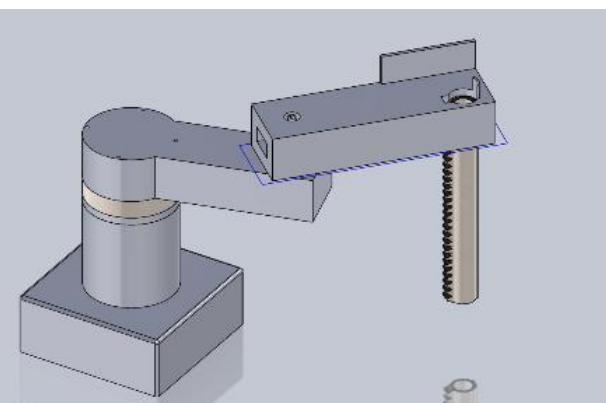
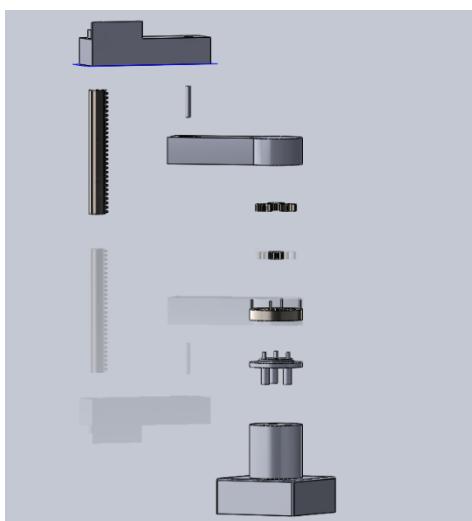


PHOTO H



BLOWUP TO SHOW ALIGNMENTS



9. Appendix

Microcontroller Code

```
1 #include <Stepper.h>
2 #include <math.h>
3
4 const int stepsPerRevolution = 2048;
5
6 int ls1 = 8;
7 int ls2 = 9;
8 int ls3 = 10;
9
10 bool limit1 = false;
11 bool limit2 = false;
12 bool limit3 = false;
13
14 Stepper motor1(stepsPerRevolution, 22, 24, 23, 25);
15 Stepper motor2(stepsPerRevolution, 26, 28, 27, 29);
16 Stepper motor3(stepsPerRevolution, 30, 32, 31, 33);
17
18 float l1 = 0;
19 float l2 = 0;
20
21 float theta1 = 0;
22 float theta2 = 0;
23 float theta3 = 0;
24 float tarTheta1 = 0;
25 float tarTheta2 = 0;
26 float tarTheta3 = 0;
27
28 void setup() {
29   motor1.setSpeed(10);
30   motor2.setSpeed(10);
31   motor3.setSpeed(10);
32
33   pinMode(ls1, INPUT);
34   pinMode(ls2, INPUT);
35   pinMode(ls3, INPUT);
36
37   homePosition();
38 }
39
40 void loop() {
41   //moveTo()
```

```

42  }
43
44 void moveMotors(long s1, long s2, long s3) {
45     long steps1 = abs(s1);
46     long steps2 = abs(s2);
47     long steps3 = abs(s3);
48
49     int dir1;
50     int dir2;
51     int dir3;
52
53     if (s1 >= 0) {
54         dir1 = 1;
55     } else {
56         dir1 = -1;
57     }
58     if (s2 >= 0) {
59         dir2 = 1;
60     } else {
61         dir2 = -1;
62     }
63     if (s3 >= 0) {
64         dir3 = 1;
65     } else {
66         dir3 = -1;
67     }
68
69     long maxSteps = max(steps1, max(steps2, steps3));
70
71     long stepCounter1=0;
72     long stepCounter2=0;
73     long stepCounter3=0;
74     for (int i = 0; i < maxSteps; i++) {
75         if ((i * steps1) / maxSteps > stepCounter1) {
76             motor1.step(dir1);
77             stepCounter1++;
78         }
79         if ((i * steps2) / maxSteps > stepCounter2) {
80             motor2.step(dir2);
81             stepCounter2++;
82         }
83         if ((i * steps3) / maxSteps > stepCounter3) {
84             motor3.step(dir3);
85             stepCounter3++;
86         }

```

```

87     delay(2);
88 }
89 }
90
91 void homePosition() {
92     while(!limit1 | !limit2 | !limit3){
93         if(digitalRead(ls1)){
94             limit1 = true;
95         } else{
96             motor1.step(-1);
97         }
98         if(digitalRead(ls2)){
99             limit2 = true;
100        } else{
101            motor2.step(-1);
102        }
103        if(digitalRead(ls3)){
104            limit3 = true;
105        } else{
106            motor3.step(-1);
107        }
108    }
109    moveMotors(1024, 1024, 0);
110}
111
112void inverseKinematics(float x, float y, float z){
113    tarTheta2 = acos((sq(x) + sq(y) - sq(l1) - sq(l2)) / (2*l1*l2));
114    if(x < 0 & y < 0){
115        tarTheta2 *= -1;
116    }
117    tarTheta1 = (atan(y/x) - atan(l2 * sin(tarTheta2)/(l1 + l2 *
118        cos(tarTheta2)))) * 180/3.14159;
119    tarTheta2 *= 180/3.14159;
120}
121void moveTo(float x, float y, float z){
122    inverseKinematics(x, y, z);
123
124    int step1 = round((tarTheta1 - theta1)*256/45);
125    int step2 = round((tarTheta2 - theta2)*256/45);
126    int step3 = round((tarTheta3 - theta3)*256/45);
127
128    moveMotors(step1, step2, step3);
129}

```