STRUCTURE CLIMBING MONKEY ROBOT

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Paul Bessent

May 2011

© 2011
Paul Bessent
ALL RIGHTS RESERVED

Committee Membership

TITLE: Structure Climbing Monkey Robot

AUTHOR: Paul Bessent

DATE SUBMITTED: May 2011

COMMITTEE CHAIR: Saeed Niku, Professor, Mechanical Engineering

COMMITTEE MEMBER: William Murray, Professor, Mechanical Engineering

COMMITTEE MEMBER: John Ridgely, Associate Professor, Mechanical
Engineering

Abstract

Structure Climbing Monkey Robot

Paul Bessent

This report describes the design, building, and testing of the Structure Climbing Monkey Robot (SCMR). It is composed of seven successive joints and linkages with two grippers at the two ends. Each gripper can act as the base or the end of the robot. The SCMR has the ability to climb any structure. The gripper plates can be changed to grab different kinds of structures, but this one is made to grab 2x4's. A program was written to assist the user to grab four non-coplanar, non-orthogonal points.

The SCMR is actuated by a total of nine motors: two to open and close the two grippers and seven to control the movement of the SCMR. Planetary gear motors are used with a worm gear to control the motion of each joint. The worm gear increases the torque of the motor and reduces the rotational speed to a usable value. The SCMR is just over 45 inches long and weighs about 30 pounds.

The motion of the SCMR is controlled by the microcontroller Arduino Mega 2560, Vex Robotic quadrature encoders, and Pololu 18v15 motor driver chips. Code was written in the languages Arduino and Processing to actuate the motors and create the GUI, respectively. The motors can be controlled individually or run simultaneously while incrementing a specified angle.

Table of Contents

	Page
List of Tables	viii
List of Figures	x
Chapter 1-Overview of Project	1
1.1-Introduction	1
1.2-Structure Climbing Monkey Robot	1
1.3-Similar Work	3
1.3.1-Carnegie Mellon's Self-Mobile Space Manipulator	3
1.3.2-Dr. Elon Rimon's Spider Robot	4
1.3.3-SRI International's Electroadhesion Wall-Climbing Robot	4
Chapter 2-Computer Modeling and Simulation	6
2.1-Modeling and Simulation Programs	6
2.1.1-Microsoft Robotics Studio	6
2.1.2-Dymola	6
2.1.3-SimulationX	10
2.1.4-MapleSim	12
2.2-Motor and Gear Selection	14
2.2.1-Motor Research and Selection	14
2.2.2-Worm Gear Selection	16

2.3-Forward Kinematic Equations of the Robot with Denavit-Hartenberg Modeling	
Representation	19
Chapter 3-Building and Testing the SCMR.....	26
3.1-SCMR Body Design	26
3.1.1-Motor with Worm Gearbox	26
3.1.2-Wrist Joints.....	27
3.1.3-Composite Links.....	28
3.1.4-Gripper.....	31
3.1.5-Preliminary Testing	32
3.2-Controller Hardware	35
3.2.1-Microcontroller Unit.....	35
3.2.2-Motor Driver Chip	37
3.2.3-Quadrature Encoder.....	39
3.2.4-Circuitry.....	41
3.3-Controller Software	42
3.3.1-Processing Code	43
3.3.2-Arduino Code	46
3.4-Testing Results	52
3.4.1-Climbing Outside of the Structure.....	53
3.4.2-Climbing Inside of the Structure	59
Chapter 4-Discussion.....	73
4.1-Conclusion.....	73

4.2-Future Work.....	74
Bibliography	76
Appendix.....	77
Source of Motors Researched	77
Instructions Manual	78
Installing the Programs	78
Setting Up the SCMR	78
Running the SCMR.....	80
Troubleshooting	83
Changing the Preset Motions	84
Replacing Hardware	85
Arduino Code.....	86
Processing Code.....	127

List of Tables

	Page
Table 1. Specifications of motors researched. The selected motors are highlighted.....	15
Table 2. D-H parameter table for SCMR with the left end as the base.....	20
Table 3. Motor angles to move SCMR from neutral to starting position.....	60
Table 4. Motor angles to move SCMR from point 1 to 1a.....	62
Table 5. Motor angles to move SCMR from point 1a to 1b.....	62
Table 6. Motor angles to move SCMR from point 1b to 1c.....	62
Table 7. Motor angles to move SCMR from point 1c to 2, to be actuated about 5 times.....	63
Table 8. Suggested motor angles to minimize load on gripping point 1 for Gripper 1 release....	64
Table 9. Suggested motor angles to move Gripper 1 away from gripping point 1 after release...	64
Table 10. Motor angles to move SCMR from point 2 to 2a.....	65
Table 11. Motor angles to move SCMR from point 2a to 2b.....	66
Table 12. Motor angles to move SCMR from point 2b to 2c.....	66
Table 13. Motor angles to move SCMR from point 2c to 3, to be actuated about 5 times.....	67
Table 14. Suggested motor angles to minimize load on gripping point 2 for Gripper 2 release...	68
Table 15. Suggested motor angles to move Gripper 2 away from gripping pt 2 after release.....	68
Table 16. Motor angles to move SCMR from point 3 to 3a.....	69
Table 17. Motor angles to move SCMR from point 3a to 3b.....	69
Table 18 Motor angles to move SCMR from point 3b to 3c.....	69
Table 19. Motor angles to move SCMR from point 3c to 3d, to be actuated about five times....	70
Table 20. Motor angles to move SCMR from point 3d to 4, to be actuated about four times.....	70
Table 21. Suggested motor angles to minimize load on gripping point 3 for Gripper 1 release...	71
Table 22. Suggested motor angles to move Gripper 1 away from gripping pt 3 after release.....	72

Table 23. List of motors' source website. The selected motors are highlighted.....	77
--	----

List of Figures

	Page
Figure 1. Structure Climbing Monkey Robot body in “neutral” position.	2
Figure 2. Dr. Elon Rimon’s Spider Robot.	4
Figure 3. SRI International’s Electroadhesion Wall-Climbing Robot.	5
Figure 4. Initial seven motor design in Dymola.	7
Figure 5. Alternative motor orientation designs of SCMR.	8
Figure 6. Dymola programming code for the SCMR.	9
Figure 7. Dymola plot of max motor torque (y) as the SCMR undergoes motion in time (x).	10
Figure 8. SimulationX model, the labels motor4 and motor6 were not used because of the labeling in the A-matrices: (a) Left end (b) large “motor1” (c) large “motor2” (d) small “motor3” (e) Left arm (f) small “motor5” (g) Right arm (h) small “motor7” (i) large “motor8” (j) large “motor9” (k) Right end.	11
Figure 9. Modeling diagram for SCMR in SimulationX.	11
Figure 10. Code for the SCMR model in MapleSim.	13
Figure 11. Simulation environment in MapleSim of SCMR code.	14
Figure 12. Banebot motor with planetary gearbox used for SCMR.	16
Figure 13. Motor torque with worm gear in critical position. Peak torque=320 in-lb _f =36 Nm. ...	18
Figure 14. Small motor torque with worm gear in critical position. Peak torque=175 in-lb _f =20 Nm.	18
Figure 15. Axes used for A-matrices.	20
Figure 16. SCMR with motors numbered.	26
Figure 17. Motor with custom worm gearbox attached.	27
Figure 18. Wrist attached to Gripper 1 with motors numbered.	28

Figure 19. Composite link test with empty tube.	29
Figure 20. Third link test with foam-filled tube.....	30
Figure 21. Link tests results.	30
Figure 22. Elbow connecting the two wrists via links.	31
Figure 23. Gripper designed and built for the SCMR.....	32
Figure 24. Lowered position of SCMR for full body lift test.	32
Figure 25. Raised position of SCMR for full body lift test.....	33
Figure 26. SCMR reaching for a new gripper position.....	33
Figure 27. Both grippers engaged.	34
Figure 28. Gripper 2 released from structure.	34
Figure 29. Arduino Mega 2560 used as the SCMR’s MCU.	36
Figure 30. Example of duty cycles for PWM signal.....	36
Figure 31. Example of an H-bridge for MDC.....	37
Figure 32. Pololu 18 volt 15 amp MDC used for the SCMR.....	38
Figure 33. Encoder disc with slots.....	39
Figure 34. Output signal of encoder for CW (top) and CCW (bottom) rotation.	40
Figure 35. Vex Robotics shaft quadrature encoder used for the SCMR.....	40
Figure 36. Circuit diagram for the SCMR.	42
Figure 37. Flowchart for Processing code to control the GUI for the SCMR.....	43
Figure 38. GUI used to control the SCMR.	44
Figure 39. Direction of worm gear shaft standardized for all motors.....	45
Figure 40. Flowchart for Arduino code.	47
Figure 41. Detailed flowchart for normalized motion in Arduino code.	48
Figure 42. Structure built to test the SCMR.....	52
Figure 43. Gripping points for the SCMR traversing the outside of the structure.....	53
Figure 44. SCMR in initial “neutral position at gripping point 1.	54

Figure 45. Intermediate steps for SCMR moving from gripping point 1 to 2.	55
Figure 46. SCMR with Gripper 1 engaged (left) and released (right) at gripping point 1.....	55
Figure 47. SCMR swinging Gripper 1 from gripping point 1 to 3.	56
Figure 48. SCMR swinging Gripper 1 from gripping point 1 to 3.	57
Figure 49. SCMR with grippers engaged at gripping points 2 and 3.....	57
Figure 50. Maximum reach of SCMR past front plane of structure.	58
Figure 51. SCMR limited in reaching past front plane of structure.....	59
Figure 52. Gripping points for SCMR climbing inside of structure.	60
Figure 53. SCMR in starting orientation to climb inside of structure.....	61
Figure 54. SCMR attached to gripping point 1 in start position.	61
Figure 55. SCMR in position 1c.	62
Figure 56. SCMR attached to gripping points 1 and 2 inside the structure.	63
Figure 57. Gripper 1 moved away from gripping point 1, and ready for preset motion.....	65
Figure 58. SCMR in position 2c.	66
Figure 59. SCMR attached to gripping points 2 and 3 inside the structure.	67
Figure 60. Gripper 2 moved away from gripping point 2, and ready for preset motion.....	68
Figure 61. SCMR in position 3d.....	70
Figure 62. SCMR attached to gripping points 3 and 4 inside the structure.	71
Figure 63. Gripper 1 moved away from gripping point 3, and ready for next motion.....	72

Chapter 1-Overview of Project

1.1-Introduction

This report describes the research and building of a prototypical Structure Climbing Monkey Robot (SCMR) that has the ability to climb any structure. The SCMR is composed of seven successive joints and linkages with two grippers at the two ends. Each gripper can act as the base or the end of the robot. The main purpose for the SCMR is to be able to climb where it is too dangerous or difficult for a person to go in order to do inspections and/or maintenance. The research began as an Honor's research project at Cal Poly in the winter of 2008 and then became a thesis in the spring of 2009.

1.2-Structure Climbing Monkey Robot

The goal of this project was to create a robot to traverse and climb a structure by grabbing non-coplanar beams at non-orthogonal angles. The application of the SCMR is to climb buildings, bridges, or other similar structures to perform inspections and possibly maintenance. Although the SCMR does not include the sensors and tools necessary for inspections and maintenance, it proves that such a robot can be built. The advantage of the SCMR is that it does not have to follow a predetermined path while traversing the structure, thus making it still usable if parts of the structure are damaged.

The ability to traverse and climb a structure was accomplished by using seven motors to control the motion of the SCMR and a motor for each of the two grippers. The seven motors used for motion are DC motors with a planetary gearbox and a set of worm gears attached. The SCMR is nearly symmetrical for the motor orientations so that either end can act as the base of the robot.

The grippers use a motor to turn a screw that moves one plate relative to another in order to open and close the gripper.

The SCMR weighs about 30 pounds and the distance from end to end is just over 45 inches. The body of the SCMR is shown in Figure 1, in the “neutral” position. Different versions of the SCMR are depicted throughout this report due to different methods of wiring.

Figure 1. Structure Climbing Monkey Robot body in “neutral” position.

The motors and encoders are connected to the motor driver chips (MDC's) and microcontroller unit (MCU), respectively, via a cable tether. Two programs must be running to control the motions of the SCMR. The Processing program running on a computer creates the GUI and allows the user to send messages to the MCU. The Arduino program runs on the MCU to interpret the messages from the user and perform functions to increment the motor angles.

1.3-Similar Work

1.3.1-Carnegie Mellon's Self-Mobile Space Manipulator

A similar project to the SCMR was undertaken by a group at the Carnegie Mellon University Robotics Institute many years ago [1]. A report in the 1990 Annual Research Review explains their findings and the problems they encountered. The Self-Mobile Space Manipulator (SM^2) was designed for the space station in order to perform routine inspection, maintenance, and light assembly. The SM^2 is desired because it is a much safer alternative than using an astronaut to do the inspections.

The SM^2 has much value to the space station, but it has many limitations. The design of the SM^2 is similar to a human arm. A gripper is at each end, connected by two links, with an elbow in the middle. The gripper for the SM^2 is a mechanism that screws into holes on a truss structure. This means that the design is limited to walk on the monkey bar-like structure that must be made around the space station. If the truss structure is damaged, the SM^2 is not able to move everywhere.

A 1/3 scale model of the SM^2 was built and tested at Carnegie Mellon University. Because the robot's links are constructed out of thin-walled aluminum tubing to be lightweight, the SM^2 is very flexible. Since the SM^2 is to be used in space that has very little gravity, a servoed gravity compensation system was developed to make the testing similar to a zero gravity testing.

The SM^2 would not be able to be used on earth because the force of gravity would be too much for the flexible links. Also, the SM^2 would not be able to be used on a variety of structures. The research of the SM^2 is no longer active. The successful design of a SCMR that can climb any structure would have many more uses on the space station and on earth.

1.3.2-Dr. Elon Rimon's Spider Robot

A successful design of a robot with a similar purpose is Dr. Elon Rimon's spider robot [2], shown in Figure 2. This three-legged robot can climb over obstacles and get into places too dangerous for people. The robot is able to move through underground cavities, pipes, and tunnels for the purpose of finding survivors in a collapsed building. It is also able to perform control and maintenance operations of complex systems in dangerous structures [3].

Figure 2. Dr. Elon Rimon's Spider Robot.

1.3.3-SRI International's Electroadhesion Wall-Climbing Robot

A third similar robot is SRI International's Wall-Climbing Robot [4]. This robot uses electroadhesion to induce a charge on wall surfaces, and produces up to 2.3 psi of clamping pressure [5]. This allows the robot climb up vertical walls made out of concrete, wood, steel, glass, drywall, and brick surfaces repeatedly even if they are covered in dust, debris, or water.

The robot propels itself using “tank” styled tracks to roll along the surfaces, and can be seen in Figure 3 attached to a brick wall.

Figure 3. SRI International's Electroadhesion Wall-Climbing Robot.

Chapter 2-Computer Modeling and Simulation

2.1-Modeling and Simulation Programs

The purpose of modeling the SCMR in the computer was to determine the best orientation for the motors. The modeling programs also helped to determine how many motors are needed and what motors should be used. Four programs were used and compared for their ability to model the SCMR: Microsoft Robotics Studio, Dymola, SimulatinX, and MapleSim.

2.1.1-Microsoft Robotics Studio

The Microsoft Robotics Studio was the first program used to attempt to model the SCMR. However, this program does not allow the user to create a robotic model. The user can only simulate premade models. This program is useful for controlling compatible hardware, but not simulating and testing the SCMR.

2.1.2-Dymola

Unlike the Microsoft Robotics Studio, Dymola is capable of creating a user specified robot. Using estimates for the dimensions, a computer model of the SCMR was successfully created in Dymola. The model is simulated in a graphic environment where motion to the joints can be applied and analyzed.

A model of the SCMR using only six motors to control the motion was built and tested in Dymola. Since there are six degrees of freedom in space (three distances for position and three angles for orientation), the SCMR would need at least six motors to control the motion. However, a six motor SCMR could not be made symmetrical such that it had full mobility. The six motor SCMR was able to reach any point in space, but its gripping ability was limited. At some positions, the gripper would be stuck in its orientation and was not able to turn in order to

grab the structure correctly for a firm grip. The SCMR needs to be symmetrical so that either end can act as the base with no loss of mobility.

With the failure of the six motor SCMR design, the testing of a seven motor SCMR began. Since one of the problems with the six motor SCMR design was that the gripper would be in the wrong orientation at the new gripping point, a motor must be attached to the gripper to rotate it. This was incorporated into the successful design shown in Figure 4. This seven motor SCMR was able to reach any point and the gripper was able to rotate to any orientation for a secure grip of any structure. The grippers were omitted from the Dymola model to make the design trials faster and easier; however their masses were considered and added as point masses at the end of the model. The red cylinders in the figure represent the motors; the axis of rotation for a motor runs through the center of the circular faces of the cylinder. The blue cylinders represent linkages between the joints to increase the SCMR's reach.

Figure 4. Initial seven motor design in Dymola.

Although the SCMR had maximum mobility, the axes of rotation of the motors at the “wrist” had to be arranged so that they would intersect. This would later allow the motion of the SCMR to be solved more easily mathematically. The initial design of the SCMR had the axes of

rotation of two of the motors in the “wrist” intersecting while the third axis of rotation was offset by a small distance. The adjustments were made to the SCMR while preserving the full mobility. This new design also allows the repositioning of the two motors closest to the “elbow” to either be at the “wrist” or at the “elbow” as seen in Figure 5.

Figure 5. Alternative motor orientation designs of SCMR.

The programming code that made the simulated model of the SCMR in Dymola is shown in Figure 6. The `pointMass` was used to give the motors their specified masses. The `actuatedRevolute` is the motor and is controlled by a ramp function that specifies the angular position of the joint. The `bodyBox` was used to connect the motors together and the `bodyCylinder` represents the link that makes up the body of the SCMR.

Figure 6. Dymola programming code for the SCMR.

Using a variety of motors available online, different weights of the motors were tested to calculate the necessary torque to move the SCMR through a range of motions. Such calculations were plotted as shown in Figure 7. The results of the calculations showed that the use of two different motors would provide the most available torque while keeping the weight as low as possible. Since not as much torque is needed by the elbow, a smaller motor can be used in order to save weight. Two large motors are needed in each wrist because they will experience the most torque during certain SCMR orientations. The third motor in the wrist can be the same as the elbow because they will experience the same maximum torque.

Figure 7. Dymola plot of max motor torque (y) as the SCMR undergoes motion in time (x).

The two motors that performed best were heavier than initially desired, but the motors have the highest torque to weight ratio; this means that the motors can lift their own weight better than the other motors. The motors used for comparison are explained later in section 2.2.1.

2.1.3-SimulationX

Similar to Dymola, the program SimulationX was also successful in modeling and simulating the SCMR. However, the mass of the motors and gears could not be easily and accurately positioned. For testing purposes, the missing masses were added to the mass of the adjacent gripper because they are close in proximity. This created a nearly identical moment on the joints as is required for the motor torque calculations even though the moment of inertia is different. For the model, motor numbers four and six were skipped to match the axes used for the A-matrices described later in section 2.3. Figure 8 shows the SimulationX model of the SCMR

with blocks representing the grippers. The left and right arms were cut short on the end toward the elbow so that interference between the two arms could be seen. SimulationX was used to model the SCMR in order to observe its range of motion. The modeling diagram for the SCMR is shown in Figure 9.

Figure 8. SimulationX model, the labels motor4 and motor6 were not used because of the labeling in the A-matrices: (a) Left end (b) large “motor1” (c) large “motor2” (d) small “motor3” (e) Left arm (f) small “motor5” (g) Right arm (h) small “motor7” (i) large “motor8” (j) large “motor9” (k) Right end.

Figure 9. Modeling diagram for SCMR in SimulationX.

Before the masses of the gears and joint brackets were known, a simulation was ran without those masses to calculate the necessary torque that the motors must output just to overcome the weight of the motors, links, and grippers. The large motor had a peak torque of

about 186 in-lb_f (21 Nm) while the small motor experienced a peak torque of about 97 in-lb_f (11 Nm). The goal was to use this data to determine an appropriate set of worm gears, and then the masses of those gears would be added to the simulation for another calculation. That resulting data would either confirm or disprove those worm gears to be suitable for the SCMR. These results are shown later in section 2.2.2.

2.1.4-MapleSim

The last modeling program used is the MapleSim toolbox of the computer algebra system Maple. In this program, the user is able to specify links and joints to make up a robot, but there was much difficulty in controlling the motion.

The results of the simulation were not consistent in MapleSim. Even with simplified models to run just one motor, the results would vary even though the input was identical. Sometimes the program would freeze up and not output any information.

The times that the simulation did work, the control of the motion was still limited. Only the gain of the motor could be specified, not the rotational speed or position. The result was that the motor would slowly speed up, then gain speed and start rotating rapidly. The calculated torque output of the motor when moving slowly was close to the SimulationX estimate. Another drawback of this program is that MapleSim did not specify the units of the torque calculated. It was assumed that SI units were used since that was what matched SimulationX.

Lastly, in trying to specify an angular velocity for the motor, the simulation would not move the joint as desired. The joint starts to move up, as intended, and then falls down and swings like a pendulum while another joint spins freely even though it was “fixed”.

The layout of the code used to model the SCMR in MapleSim is shown in Figure 10. The simulation environment where the model is built and tested is shown in Figure 11.

Figure 10. Code for the SCMR model in MapleSim.

Figure 11. Simulation environment in MapleSim of SCMR code.

2.2-Motor and Gear Selection

2.2.1-Motor Research and Selection

The first type of motor researched for the SCMR is a servo motor. The benefit of these motors is that they are easy to control and can turn to a precise angle; this is what is desirable to control the motion of the SCMR. However, the servo motors found were not strong enough for the SCMR base upon the calculations from the modeling programs. Due to the weight of the motors and the number of motors needed, a torque much higher than the strongest servo was needed. Although mounting a gear to the servo would increase the torque, the speed would drop to an undesirable amount. Also, a servo has limited range of motion and cannot complete a revolution without a modification to the servo, so a set of gears would further limit the range of motion.

The next type of motor researched is a planetary gear motor. For this type of motor, a planetary gearbox is attached to the output shaft of the motor to reduce the speed and greatly increase the torque. The advantage of planetary gears is that high gear ratios can be achieved with a relatively small gearbox by using multiple stages; some motors have a gear ratio as high as 256:1. These motors are able to produce much more torque than the servos while maintaining a usable speed for the SCMR. A list of the specifications of the motors considered for the SCMR is compiled in Table 1. The sources for the motors are compiled in Table 23 attached in the appendix. These motors were tested in the modeling programs to determine which motors would be most suitable for the SCMR. The two motors with the best specifications are highlighted in Table 1. These motors were chosen because of their high torque to weight ratio and their relatively high rate of rotation. Although the torque of these motors is not enough for the SCMR, the worm gear increases the torque and reduces the fast speed of the motor to a proper rate.

Table 1. Specifications of motors researched. The selected motors are highlighted.

Motor #	Weight (oz)	Stall Torque (oz-in)	Rated Torque (oz-in)	Stall Torque / Weight	Rated Torque / Weight	No-Load RPM (Efficient)
1	2.1	70		33.3		48
2	2.5	250		100.0		90
3	2.5	190		76.0		115-300
4	0.9	70		77.8		220-410
5	2.7	170	50	63.0	18.5	220 (131)
6	1.8 servo	116		64.4		
7	0.28 servo	18		64.3		
8	1.12 servo	76		67.9		
9	3.6	342	130	95.0	36.1	64 (47)
10	6.7	255	76	38.1	11.3	200 (112)
11	7.2	232		32.2		152
12	21.3	18061	2125	847.9	99.8	75 (67)
13	10	952		95.2		164
14	13.5	3700	540	274.1	40.0	(138)
15	9.376	500		53.3		14
16	8.8	390		44.3		65
17	12.6	4935	775	391.7	61.5	134 (116)
18	11	2437		221.5		64
19	1.3	534		410.8		26
20	5.8	2448		422.1		67

Due to availability, three types of motors were used for the SCMR. The four large motors are an RS-550 with a planetary gearbox that has a ratio of 144:1. Although the gear ratio is smaller than the one used for testing of 256:1, the weight is the same and the worm gear will compensate for the lower torque value. The small motor for the elbow is an RS-540 with a planetary gearbox that has a ratio of 125:1; this is the same as the one researched. The last two small motors in the wrist are also an RS-540, but with a planetary gearbox that has a ratio of 64:1. All three types of motors with their gearbox are similar to the one pictured in Figure 12.

Figure 12. Banebot motor with planetary gearbox used for SCMR.

2.2.2-Worm Gear Selection

The purpose of using a worm gear on top of the planetary gearbox of the motor is to further increase the torque output of the motor and reduce the speed. The reason for using worm gears as opposed to other types of gears is that the small lead angle is very small; the torque applied to the worm by the worm gear is unable to overcome the friction necessary to drive the worm. This makes the gears self-locking and allows the motors to maintain their position without the use of power. This means that the SCMR will remain in its orientation unless power is given to the motors.

Before determining the worm gear ratio necessary for the motors, the torque output of the motors must be known. The calculated specifications of the motors were given explicitly from the supplying company, Banebots. The large motor has a calculated torque at peak efficiency of 75 in-lb_f (8.4 Nm) with an angular velocity of 120 rpm. The small motor with the 64:1 gear ratio has a calculated torque at peak efficiency of 25 in-lb_f (2.6 Nm) with an angular velocity of 227 rpm. The small motor with the 125:1 gear ratio will provide more torque, but just at a lower speed. Since these values are calculated and not measured they do not take into account the effects of friction that come from the internal planetary gears. To verify these specifications, the large motor was tested with a 12V source and found to have a no-load speed of 85 rpm. The calculated 12V no-load speed from the motor specifications is 112 rpm. Using this ratio, the expected torque of the large motor is estimated to be 57 in-lb_f (6.5 Nm). To be conservative, the estimated torque output of the large and small motors was reduced to be 45 in-lb_f (5 Nm) and 20 in-lb_f (2.3 Nm), respectively.

While neglecting the masses of the gears and joint brackets, the calculations from the modeling programs showed that the large and small motors must have a gear ratio of at least 5:1. The lowest worm gear ratio that would fit the motor shafts and that were available is 20:1. This would reduce the calculated rotational speed of the large and small motors to 6 rpm and 11rpm, respectively, at peak efficiency. This is much more practical for the SCMR because the robot should not whip around in a split second due to safety.

The worm gear set selected for the large motor is a 10 pitch, single threaded, 20:1 ratio, cast iron worm with a steel worm. The worm gear set for the smaller motors is similar, but has a pitch of 12.

The weights of these gears were added to the simulation as well as an estimated weight for the brackets that mount the gears to the motor. The simulation has the SCMR move through critical positions that maximize the torque applied to the motors. The resulting calculations of the joint torques are depicted in Figure 13 for the large motor and Figure 14 for the small motor. The

peak torques for the large and small motors are 320 in-lb_f (36 Nm) and 175 in-lb_f (20 Nm), respectively. Using the expected 45 in-lb_f (5 Nm) and 20 in-lb_f (2.3 Nm) output torques of the motors, a gear ratio of 9:1 would suffice for both. Since a gear ratio of 20:1 will be used, this gives a comfortable factor of safety to take into account gear inefficiency.

Figure 13. Motor torque with worm gear in critical position. Peak torque=320 in-lb_f=36 Nm.

Figure 14. Small motor torque with worm gear in critical position. Peak torque=175 in-lb_f=20 Nm.

Before these gears were confirmed to be suitable for the SCMR, the ratings of the gears had to be checked. The Browning handbook for gears rates the 10 pitch and single threaded

worm gear with a 20:1 ratio to output 402 in-lb_f (45 Nm) of torque for 100 rpm input from the worm. This gives an estimated factor of safety of only 1.25 for the large worm gear set. This low but is the best available. The handbook also rates the 12 pitch and single threaded worm gear with a 20:1 ratio to output 231 in-lb_f (26 Nm) of torque for 100 rpm input from the worm. This gives an estimated factor of safety of 1.3 for the small worm gear set.

2.3-Forward Kinematic Equations of the Robot with Denavit-Hartenberg Modeling Representation

Denavit and Hartenberg (D-H) standardized the way to represent the modeling of robotic motions [6]. Through the use of a series of transformation matrices to move to each reference frame of each joint, the motion of a robot can be represented by a matrix. As the robot moves, so do the local reference frames. The local reference frame of each joint uses the z-axis to represent the motion of either rotational or linear movement, according to the right-hand rule for rotation. All of the joints in the SCMR, excluding the grippers, are rotational. The x_n -axis is set to be in the direction of the common normal of the z_{n-1} -axis and z_n -axis, where the common normal is the line mutually perpendicular to any two skew lines. In the case that the z-axes are parallel, the common normal that is collinear with the common normal of the previous joint will be used. Lastly, if the z-axes intersect, then the x-axis will be in the direction perpendicular to the plane of the two z-axes. The local reference frames used for the SCMR is shown in Figure 15. An extra reference frame was created to help with symmetry; this allows for the local x and z axes to be identical regardless of which end is the base of the robot. For this reason I skipped motor numbers four and six in the SimulationX code, as mentioned earlier.

Figure 15. Axes used for A-matrices.

These reference frames were used to fill out the D-H parameters in Table 2, using the left end as the base of the robot. The variable T_n represents the angle between the x_n -axis and x_{n+1} -axis about the z_n -axis. The variable d_{n+1} represents the distance needed to translate along the z_n -axis in order to make the x_n -axis and x_{n+1} -axis collinear. The variable a_{n+1} represents the distance needed to translate along the x_n -axis in order to make the origins of the x_n -axis and x_{n+1} -axis the same. Lastly, the variable α_{n+1} represents the angle needed to rotate about the x_{n+1} -axis in order to align the z_n -axis and z_{n+1} -axis.

Table 2. D-H parameter table for SCMR with the left end as the base.

Joint	x-angle, T_n	z-distance, d	x-distance, a	z-angle, α
0→1	0	-d1	0	0
1→2	T_1	0	0	+90°
2→3	T_2	0	0	-90°
3→4	T_3	-d2	0	0
4→5	0	0	0	+90°
5→6	T_5	0	0	+90°
6→7	0	+d3	0	0
7→8	T_7	0	0	-90°
8→9	T_8	0	0	+90°
9→10	T_9	+d4	0	0

The values in the D-H parameter table are used to form each A-matrix following the form of Eqn 1. This is a result of postmultiplying the four matrices representing the four motions in the table. The term ${}^nM_{n+1}$ represents the transformation from frame n to $n+1$.

$${}^nM_{n+1} = A_{n+1} = \begin{bmatrix} \cos T_n & -\sin T_n \cos \alpha_{n+1} & \sin T_n \sin \alpha_{n+1} & a_{n+1} \cos T_n \\ \sin T_n & \cos T_n \cos \alpha_{n+1} & -\cos T_n \sin \alpha_{n+1} & a_{n+1} \sin T_n \\ 0 & \sin \alpha_{n+1} & \cos \alpha_{n+1} & d_{n+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{Eqn 1})$$

The resulting A-matrices for the left end of the robot acting as the base are compiled into Eqn 2.

$$\begin{aligned} A_1 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} & A_6 &= \begin{bmatrix} \cos(T5) & 0 & \sin(T5) & 0 \\ \sin(T5) & 0 & -\cos(T5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ A_2 &= \begin{bmatrix} \cos(T1) & 0 & \sin(T1) & 0 \\ \sin(T1) & 0 & -\cos(T1) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & A_7 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ A_3 &= \begin{bmatrix} \cos(T2) & 0 & -\sin(T2) & 0 \\ \sin(T2) & 0 & \cos(T2) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & A_8 &= \begin{bmatrix} \cos(T7) & 0 & -\sin(T7) & 0 \\ \sin(T7) & 0 & \cos(T7) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{Eqn 2}) \\ A_4 &= \begin{bmatrix} \cos(T3) & -\sin(T3) & 0 & 0 \\ \sin(T3) & \cos(T3) & 0 & 0 \\ 0 & 0 & 1 & -d2 \\ 0 & 0 & 0 & 1 \end{bmatrix} & A_9 &= \begin{bmatrix} \cos(T8) & 0 & \sin(T8) & 0 \\ \sin(T8) & 0 & -\cos(T8) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ A_5 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & A_{10} &= \begin{bmatrix} \cos(T9) & -\sin(T9) & 0 & 0 \\ \sin(T9) & \cos(T9) & 0 & 0 \\ 0 & 0 & 1 & d4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The variables T1, T2, T3, T5, T7, T8, and T9 are the angular positions of the seven joints comprised of a motor with a worm gear. The ten matrices were multiplied together in Matlab to get the forward kinematic solution. This forward kinematic solution is set equal to Eqn 3. The variables n, o, and a are unit vectors that describe the desired orientation and P is the desired position of the end of the robot.

$$M = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{Eqn 3})$$

The variables in Eqn 3 are set equal to the terms in the forward kinematic solution to be easier to read:

$$\begin{aligned} n_x = & \cos(T9)*(\sin(T8)*(\sin(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) - \\ & \cos(T1)*\cos(T5)*\sin(T2)) - \cos(T8)*(\cos(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \\ & \cos(T1)*\cos(T2)*\cos(T3)) + \cos(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T3)*\sin(T1) + \\ & \cos(T1)*\cos(T2)*\sin(T3)))) + \sin(T9)*(\sin(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \\ & \cos(T1)*\cos(T2)*\cos(T3)) + \cos(T1)*\sin(T2)*\sin(T5)) + \cos(T7)*(\cos(T3)*\sin(T1) + \\ & \cos(T1)*\cos(T2)*\sin(T3))) \end{aligned}$$

$$\begin{aligned} n_y = & -\cos(T9)*(\sin(T8)*(\sin(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) + \\ & \cos(T5)*\sin(T1)*\sin(T2)) - \cos(T8)*(\cos(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \\ & \cos(T2)*\cos(T3)*\sin(T1)) - \sin(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T1)*\cos(T3) - \\ & \cos(T2)*\sin(T1)*\sin(T3)))) - \sin(T9)*(\sin(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \\ & \cos(T2)*\cos(T3)*\sin(T1)) - \sin(T1)*\sin(T2)*\sin(T5)) + \cos(T7)*(\cos(T1)*\cos(T3) - \\ & \cos(T2)*\sin(T1)*\sin(T3))) \end{aligned}$$

$$\begin{aligned} n_z = & \cos(T9)*(\cos(T8)*(\cos(T7)*(\cos(T2)*\sin(T5) + \cos(T3)*\cos(T5)*\sin(T2)) + \\ & \sin(T2)*\sin(T3)*\sin(T7)) + \sin(T8)*(\cos(T2)*\cos(T5) - \cos(T3)*\sin(T2)*\sin(T5))) - \\ & \sin(T9)*(\sin(T7)*(\cos(T2)*\sin(T5) + \cos(T3)*\cos(T5)*\sin(T2)) - \\ & \cos(T7)*\sin(T2)*\sin(T3)) \end{aligned}$$

$$\begin{aligned} o_x = & \cos(T9)*(\sin(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) + \\ & \cos(T1)*\sin(T2)*\sin(T5)) + \cos(T7)*(\cos(T3)*\sin(T1) + \cos(T1)*\cos(T2)*\sin(T3))) - \end{aligned}$$

$$\begin{aligned} & \sin(T9)*(\sin(T8)*(\sin(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) - \\ & \cos(T1)*\cos(T5)*\sin(T2)) - \cos(T8)*(\cos(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \\ & \cos(T1)*\cos(T2)*\cos(T3)) + \cos(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T3)*\sin(T1) + \\ & \cos(T1)*\cos(T2)*\sin(T3)))) \end{aligned}$$

$$\begin{aligned} o_y = & \sin(T9)*(\sin(T8)*(\sin(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) + \\ & \cos(T5)*\sin(T1)*\sin(T2)) - \cos(T8)*(\cos(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \\ & \cos(T2)*\cos(T3)*\sin(T1)) - \sin(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T1)*\cos(T3) - \\ & \cos(T2)*\sin(T1)*\sin(T3)))) - \cos(T9)*(\sin(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \\ & \cos(T2)*\cos(T3)*\sin(T1)) - \sin(T1)*\sin(T2)*\sin(T5)) + \cos(T7)*(\cos(T1)*\cos(T3) - \\ & \cos(T2)*\sin(T1)*\sin(T3))) \end{aligned}$$

$$\begin{aligned} o_z = & -\cos(T9)*(\sin(T7)*(\cos(T2)*\sin(T5) + \cos(T3)*\cos(T5)*\sin(T2)) - \\ & \cos(T7)*\sin(T2)*\sin(T3)) - \sin(T9)*(\cos(T8)*(\cos(T7)*(\cos(T2)*\sin(T5) + \\ & \cos(T3)*\cos(T5)*\sin(T2)) + \sin(T2)*\sin(T3)*\sin(T7)) + \sin(T8)*(\cos(T2)*\cos(T5) - \\ & \cos(T3)*\sin(T2)*\sin(T5))) \end{aligned}$$

$$\begin{aligned} a_x = & -\cos(T8)*(\sin(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) - \\ & \cos(T1)*\cos(T5)*\sin(T2)) - \sin(T8)*(\cos(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \\ & \cos(T1)*\cos(T2)*\cos(T3)) + \cos(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T3)*\sin(T1) + \\ & \cos(T1)*\cos(T2)*\sin(T3))) \end{aligned}$$

$$\begin{aligned} a_y = & \cos(T8)*(\sin(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) + \cos(T5)*\sin(T1)*\sin(T2)) \\ & + \sin(T8)*(\cos(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) - \\ & \sin(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T1)*\cos(T3) - \cos(T2)*\sin(T1)*\sin(T3))) \end{aligned}$$

$$a_z = \sin(T8)*(\cos(T7)*(\cos(T2)*\sin(T5) + \cos(T3)*\cos(T5)*\sin(T2)) + \sin(T2)*\sin(T3)*\sin(T7)) \\ - \cos(T8)*(\cos(T2)*\cos(T5) - \cos(T3)*\sin(T2)*\sin(T5))$$

$$P_x = d2*\cos(T1)*\sin(T2) - d4*(\cos(T8)*(\sin(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) - \\ \cos(T1)*\cos(T5)*\sin(T2)) + \sin(T8)*(\cos(T7)*(\cos(T5)*(\sin(T1)*\sin(T3) - \\ \cos(T1)*\cos(T2)*\cos(T3)) + \cos(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T3)*\sin(T1) + \\ \cos(T1)*\cos(T2)*\sin(T3)))) - d3*(\sin(T5)*(\sin(T1)*\sin(T3) - \cos(T1)*\cos(T2)*\cos(T3)) \\ - \cos(T1)*\cos(T5)*\sin(T2))$$

$$P_y = d3*(\sin(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) + \cos(T5)*\sin(T1)*\sin(T2)) + \\ d4*(\cos(T8)*(\sin(T5)*(\cos(T1)*\sin(T3) + \cos(T2)*\cos(T3)*\sin(T1)) + \\ \cos(T5)*\sin(T1)*\sin(T2)) + \sin(T8)*(\cos(T7)*(\cos(T5)*(\cos(T1)*\sin(T3) + \\ \cos(T2)*\cos(T3)*\sin(T1)) - \sin(T1)*\sin(T2)*\sin(T5)) - \sin(T7)*(\cos(T1)*\cos(T3) - \\ \cos(T2)*\sin(T1)*\sin(T3)))) + d2*\sin(T1)*\sin(T2)$$

$$P_z = d4*(\sin(T8)*(\cos(T7)*(\cos(T2)*\sin(T5) + \cos(T3)*\cos(T5)*\sin(T2)) + \\ \sin(T2)*\sin(T3)*\sin(T7)) - \cos(T8)*(\cos(T2)*\cos(T5) - \cos(T3)*\sin(T2)*\sin(T5))) - d1 - \\ d2*\cos(T2) - d3*(\cos(T2)*\cos(T5) - \cos(T3)*\sin(T2)*\sin(T5))$$

This type of equation with seven degrees of freedom, from the SCMR motor angles, usually requires an additional constraint equation since there are only six degrees of freedom that can be specified. In order to be able to solve these equations, one variable must be held constant by the user by specifying one of the motor angles. If a solution is not found, that angle is changed and a new solution is attempted. More work is needed to be done here to simplify the equations using trigonometric identities in order to get the inverse kinematic solution. The inverse

kinematic solution can then be used to specify all of the motor angles for a given gripping position.

Chapter 3-Building and Testing the SCMR

3.1-SCMR Body Design

The body of the SCMR is made up of three main joints that are connected with links. The joints are made up of multiple motors with a worm gearbox attached. The wrist joints are at both ends of the robot and are attached to the grippers. Links were used to connect the wrists to the elbow joint.

From this point on, the numbering of the motors no longer skips four and six as it did for SimulationX. This was changed to be easier to understand for the code that controls the motion of the SCMR, described later. Figure 16 shows the numbering of the motors and grippers used.

Figure 16. SCMR with motors numbered.

3.1.1-Motor with Worm Gearbox

Both the large and small motors are mounted with a custom worm gearbox that attaches to the planetary gearbox. The gearbox is made out of $\frac{1}{4}$ " aluminum to be lightweight, strong, and thick enough for press-fitted ball bearing. Thrust and ball bearings are used to minimize the friction. The design was tested to ensure it worked properly. Figure 17 shows an individual motor with its gearboxes. The output shaft of the planetary gearbox is used to drive the worm.

The output shaft of the worm gear is the overall output of the motor and gearboxes to be connected to the next motor.

Figure 17. Motor with custom worm gearbox attached.

3.1.2-Wrist Joints

The wrist is comprised of three joints driven by two large and one small motor. In an effort to keep the design compact, the three axes of rotation in the wrist do not intersect. The axes were positioned as close as possible to intersecting while keeping the assembly easy. Figure 18 shows how motors 1, 2, and 3 were connected. Although there are some offset distances between the axes of rotation, the design was reversed for the motor orientations of the wrist connected to Gripper 2. This returns those offset values so that the gripper are lined up together.

Figure 18. Wrist attached to Gripper 1 with motors numbered.

3.1.3-Composite Links

The elbow joint is composed of just one motor and connects the two halves together via links. The length of the links determines how long the SCMR is. It was suggested to make the link out of a tube of composite material. The tube is very strong, lightweight, but also flexible. To stiffen the tube while keeping it lightweight, foam was used to fill the inside of the tube. The foam-filled tube was tested to observe the effect of the foam. The tube was clamped to a table top and known masses were hung at the other end. The deflection of the tip was measured to compare the three tests: empty tube, tube filled with one inch of foam, and tube completely filled with foam.

The first test of an empty tube is shown in Figure 19. The tube at the clamped end is severely compressed compared to the other end.

Figure 19. Composite link test with empty tube.

In an attempt to correct the reshaping of the tube, foam was injected in the tube to a thickness of one inch at the clamped end. A circular piece of cardboard was cut and inserted in the tube to keep the foam in place as it set. The foam stuck to the cardboard and was not removable without damaging the foam. This method of modifying the tube drastically improved the deflection of the link under the same loads. The piece of cardboard may have had a significant impact, but the true effect was not tested.

For the third test, the tube was completely filled with foam (no cardboard inside) as shown in Figure 20. This still showed improvement in the deflection compared to the empty tube, but not as much as the second test. Much of the bending was done only at the clamped end where a section of the foam was cut out in order to affix the clamp.

Figure 20. Third link test with foam-filled tube.

The results of the link tests are represented in Figure 21. The graph clearly shows that the empty tube deflected the most, then the filled tube, and the one inch filled tube deflected the least—possibly due to the cardboard inside. When handling the foam filled tube, the effect is much more apparent; the tube is much stiffer with the foam. The tube can be placed on your knee and you can push about as hard as you can on the ends with little or no effect on the tube. The tube was still very lightweight so it could be used to extend the SCMR without adding much weight.

Figure 21. Link tests results.

However, during the construction of the joints, the SCMR became much longer than originally planned. Due to this, the use of the composite tubes filled with foam would be impractical. There is not an easy way to attach the composite tube to the SCMR without adding a heavy bracket. The weight of the bracket would then neglect the purpose of using the composite tube. Also, the composite tube was bulky and could not fit in between the joints because it would interfere with the motors. Lastly, the composite tubes would only possibly replace short ½” steel bars, so not much weight could be saved anyway.

The steel links attaching the wrists to the elbow are shown in Figure 22. The links were made as short as possible while keeping the necessary clearance while the motors ran and remaining symmetrical for the distance from the elbow to the wrists.

Figure 22. Elbow connecting the two wrists via links.

3.1.4-Gripper

The gripper was designed and built to grab a 2x4. One gripper plate slides along the mounting plate to engage or disengage the grip. The gripper plates fit tightly around the 2” part of the 2x4. To create a secure grip, the clearance between the 2x4 and the gripper plates was minimized. After much testing, the final gripper is shown in Figure 23.

Figure 23. Gripper designed and built for the SCMR.

3.1.5-Preliminary Testing

Some testing was performed to validate the SCMR before the controller was added. The first test was to prove that the motors are strong enough to lift the SCMR through critical positions. For this test, Gripper 2 was attached to a 2x4 and then Motor 6 was actuated to raise and lower the SCMR, as shown in Figure 24 and Figure 25.

Figure 24. Lowered position of SCMR for full body lift test.

Figure 25. Raised position of SCMR for full body lift test.

The second test was to have the SCMR change gripper position. The SCMR started with Gripper 2 grabbing a 2x4 structure. Multiple motors were actuated to move the Gripper 1 into a grabbing position. Gripper 1 then closed onto the 2x4 and Gripper 2 was released. The following figures show the progress of the test.

Figure 26. SCMR reaching for a new gripper position.

Figure 27. Both grippers engaged.

Figure 28. Gripper 2 released from structure.

Both of these tests were successful, although it was difficult to align the gripper with the 2x4 for the second test. Gripper 1 was secured to the new gripping position and Gripper 2 was opened, Gripper 2 did not release itself from the 2x4. Motors had to be actuated for Gripper 2 to break free from its grip. Due to backlash, the SCMR shifted and swung away from the structure. After the success of these tests, the controller was developed.

3.2-Controller Hardware

The electronics used to control the motion of the SCMR are a Microcontroller Unit (MCU), nine motor driver chips (MDC's), and seven quadrature encoders.

3.2.1-Microcontroller Unit

A microcontroller unit (MCU) is a small, cheap, and low-power computer [7]. They are often dedicated to one specific program to perform tasks specified by the user. The MCU has inputs and outputs to communicate with the user. MCU's are used in many modern devices including: microwave, automobiles, TV, VCR, digital cameras, cell phones, laser printers, and many more.

There are nine motors, each requiring two outputs from the MCU, plus seven encoders, each requiring two input signals. Because of the 34 I/O's required, an Arduino Mega 2560 was selected for the MCU. This board has 54 digital I/O's, including 14 pulse width modulation (PWM) outputs. The board also features 16 analog inputs that could be used for potentiometers (instead of encoders), 4 serial ports (one USB connection), and runs at 16 MHz. The MCU is shown in Figure 29.

Figure 29. Arduino Mega 2560 used as the SCMR's MCU.

A PWM signal is used in this project to give an analog signal to the motors using a digital output of the MCU. A PWM uses a square wave and changes the duty cycle to control the analog signal. This means that the power supply is fully on or off throughout the cycle. Depending on how much of the cycle the power is high will determine what the analog signal will be. In using a 12V power supply, if the digital output has a duty cycle of 10%, then the analog signal will be 10% of the 12V power supply at 1.2V. This is an easy way to control multiple motors running at different speeds off of the same power supply. An example of duty cycles of a digital output is shown in Figure 30 [8].

Figure 30. Example of duty cycles for PWM signal.

An advantage of the PWM signal is that it resists noise. Since the output is either high or low, electrical noise is not large enough to change the signal. Many MCU's and MDC's use PWM signals, making it cheap and easy to implement. Since the transistors are fully on or off for a PWM signal, heat generation is also minimal.

3.2.2-Motor Driver Chip

A motor driver chip (MDC) is a board that uses an external power supply to apply a specified voltage to a motor. The voltage of the external power supply can be scaled down using a PWM signal from the MCU. This makes the MDC act like an amplifier since a low power signal goes into the MDC and a high power signal goes out to the motor. The voltage can easily be reversed to drive the motor in the opposite direction without changing the power supply. The MDC used in this project has an n-type MOSFET H-bridge to control the flow of the current. A MOSFET is a metal–oxide–semiconductor field-effect transistor that acts as an electronic switch. An example of an H-bridge can be seen in Figure 31.

Figure 31. Example of an H-bridge for MDC.

In this example of an H-bridge, closing the different switches will result in the motor behaving three different ways. If the S2 and S3 switches are closed, then the current will flow from right to left. If the S1 and S4 switches are closed, then the current will flow from left to right, causing the motor to spin in the opposite direction. The S3 and S4 switches can be closed to dampen the motor's rotation. Since the two leads of the motor are connected together, the induced current produced from the rotating motor will act to slow down the motor. The same effect is produced by closing the S1 and S2 switches. The S1 and S3 switches should never be closed simultaneously because this would create a short circuit. For the same reason, the S2 and S4 switches should never be closed at the same time.

The MDC chosen for the SCMR is a Pololu 18V15, shown in Figure 32. This MDC can provide 18 volts and 15 amps to the motor without a heat sink. The MDC receives a PWM signal from the MCU and amplifies it using an external power supply. The MDC also receives a direction input of high or low; when high, the output current will flow from pin A to pin B; when low, the output current will flow in the reverse direction. This makes it easy to reverse the direction of the motor.

Figure 32. Pololu 18 volt 15 amp MDC used for the SCMR.

3.2.3-Quadrature Encoder

A quadrature encoder outputs two signals that are each either high or low. This signal is read by the MCU to keep track of the position of the encoder. Many quadrature encoders use an infrared light and sensor to detect slots in a disc as it rotates. The output signals from the encoder will either be high or low depending on if it is over a slot or an opaque part of the disc. The disc of the encoder is shown in Figure 33.

Figure 33. Encoder disc with slots.

The advantage of a quadrature encoder is that the direction of the disc's rotation can be determined. A quadrature encoder uses two sensors that output signals that are 90 degrees out of phase from each other. This means that if the direction is clockwise (CW), one of the sensors will go high first; if the direction is counter clockwise (CCW), then the other sensor will go high first. The output signals for CW and CCW motion is shown in Figure 34 [9].

Figure 34. Output signal of encoder for CW (top) and CCW (bottom) rotation.

Vex Robotic shaft encoders, shown in Figure 35, were used for the SCMR because they are cheap and easily mounted to the motors. The encoder uses an infrared LED and sensor to generate 90 pulses per revolution. Since the direction of the motor rotation is known by the MCU and cannot be changed due to the self-locking worm gear, only one output signal of the quadrature encoder is used. Using only one signal meant that there are 180 positions for every 360 degrees, so the encoder has a resolution of 2 degrees. By mounting the encoder to the motor shaft with worm, the resolution increased to 0.1 degrees (excluding backlash) for reading the position of the worm gear because of the gear ratio.

Figure 35. Vex Robotics shaft quadrature encoder used for the SCMR.

The seven motors that control the motion of the SCMR have been fitted with an encoder. The two grippers do not have an encoder or any kind of sensor. Since the SCMR is tested within sight of the user, the user will visually assess if the gripper is engaged.

3.2.4-Circuitry

The MCU was connected directly to all nine MDC's and all seven encoders. The MDC requires three wires to the MCU: PWM, direction, and ground. The MDC's are powered by an external power supply to amplify the PWM signal to the motors. Since only one encoder output signal is used, the encoders also require three wires to the MCU: power (5V), ground, and encoder output.

Figure 36 is the circuit diagram of the wiring used for the SCMR. Red wire was used to carry the PWM signal from the MCU to the MDC. The PWM signal is split to an LED and resistor pair to show the user how strong the signal is to each motor. Red LED's are used for the seven motors controlling the motion while green and yellow LED's are used for Gripper 1 and Gripper 2, respectively. Red wire is also used to power the MDC and encoders. The green wire carries the direction signal for the motors. White wire was used to carry the output signal of the encoder to the MDC, represented by light blue in the circuit diagram. Black wire was used for ground. The Out A and Out B pins of the MDC connect to the motors. The terms M1-M7 on the MDC's and next to the encoders represent Motors 1-7. The terms G1 and G2 on the MDC's represent Gripper 1 and Gripper 2, respectively. The MDC's are not shown connected to the motors, but the outputs Out A and Out B of the MDC's would be connected to the two motor leads.

Figure 36. Circuit diagram for the SCMR.

3.3-Controller Software

The software written to control the SCMR was broken up into two different codes. The code written in the Processing language runs on the computer that is used by the operator of the SCMR. This code creates the GUI which consists of several buttons and scroll bars. When a button is pressed, a message is sent to the MCU over the serial port. The code written in the Arduino language runs on the MCU and interprets the message from the serial port to control the motors. The Arduino code controls all of the I/O's of the Arduino board to turn the motors on/off

and read the encoders. To start the controller for the SCMR, the Arduino code must first be uploaded to the MCU. Once the Arduino code is uploaded and running, then the Processing code can be run.

3.3.1-Processing Code

The Processing code runs in a continuous loop to update the GUI and send messages to the MCU. The flowchart of the code is shown in Figure 37.

Figure 37. Flowchart for Processing code to control the GUI for the SCMR.

The GUI created by the Processing code is shown in Figure 38. The value of the scrolling bars is shown to the right of the bar for user feedback. When a button is pressed, the color changes to dark and the related buttons are changed to a light color. The buttons for preset motions and angle increment remain dark until the motion is completed.

Figure 38. GUI used to control the SCMR.

When a motor is turned on via the “On” button, the motor will run at the value set by the power scroll bar. This value can be changed while the motor is running. The motors can also be turned on using the keyboard. By pressing 1-7, the corresponding motor will toggle on/off. The direction that the motor turns is controlled by the “Up” and “Down” buttons. The direction can be toggled using the keyboard by holding shift and pressing the motor number. The direction of all of the motors follows the rotation of the worm gear shown in Figure 39.

Figure 39. Direction of worm gear shaft standardized for all motors.

The angles of the motors can be incremented two different ways: individually or simultaneously. For incrementing the motors individually, the power must be set. The power can be adjusted by the user while the motor increments. The amount that the motors increment is determined by the “Angle Change” scroll bars. This function is initiated by the “Increment” buttons to the right of the “Angle Change” scroll bars. Multiple motors can be individually incremented simultaneously using these buttons, or the motion can be normalized using the “Normalized Increment of All Angles” button.

For normalized motion, the user can only specify the maximum power for all of the motors using the high, medium, and low buttons in the lower right of the GUI. The normalized motion adjusts the speeds of all of the motors to be incremented so that they all start and finish at the same time. When the normalized button is pressed, all of the angles set for the individual motors are sent to the MCU. The code that uses those angles for normalized motion is in the Arduino code and is described in section 3.3.2.

The “Point to Point Motion” buttons are preset motions written into the code. The angles and motor directions were already determined so that the SCMR follows a preset path. When one of the preset motion buttons is pressed, the first thing the Processing code does is send all of the motor directions to the MCU and change the direction buttons’ colors. Next, all of the angles are sent to the MCU to begin the normalized motion. These buttons should only be pressed in order

after starting at the beginning. Using these buttons is described later during testing in section 3.4.2.

Three buttons are used for each gripper: open, close, and off. The grippers run at full speed to either open or close. Since the gripper motors run at 7V, the PWM is used to scale the external power supply down to the 7V; this is done in the Arduino code. Since there are no sensors to tell the MCU that the gripper is fully open or fully closed, the user must turn the grippers off.

Lastly, the “STOP EVERYTHING!!!” button is a convenient way to stop all of the motors simultaneously. The user can also press the spacebar on the keyboard to stop all of the motors. This action cancels all of the incrementing functions of the motors and resets the buttons’ colors on the GUI. The full Processing code is attached in the appendix.

3.3.2-Arduino Code

The Arduino code runs on the MCU and continually checks for messages from the GUI. The code also manages incrementing the motors. When a motor is incrementing, the Arduino code reads the state of the encoders every loop. Since the motor rotation is relatively slow, the MCU is fast enough to read the encoders without using interrupts. This was tested for up to four motors running at full speed simultaneously. The flowchart of the Arduino code is shown in Figure 40. The full Arduino code is attached in the appendix.

In this code, there are two ways to increment the motors a specified angle: individually or normalized. To individually increment the motors, the angle to be incremented and the motor power is specified in the GUI. Those values are received by the MCU and the motor is ramped up to the specified power. The encoder is tracked and the angle value is decremented until it reaches zero, at which point the motor turns off. The MCU then sends a message to the GUI indicating that the motor is done incrementing.

Figure 40. Flowchart for Arduino code.

The second method for incrementing the motors a specified angle is normalized increment. In this part of the code, the motor powers are scaled and adjusted so that all the motors start and stop at the same time. To initiate the increment, the user must first set all the desired angles and directions and then press the “Normalized Increment of All Angles” button in the GUI. The Processing code first sends the value “1” to stop all the motors and clear individual increment motor angles. The second value sent is “2” to prepare the MCU to receive all of the motor angles. Next, the angles for all seven motors are sent on the serial. Figure 41 is a detailed flowchart of how the motor speeds are adjusted to maintain normalized motion.

Figure 41. Detailed flowchart for normalized motion in Arduino code.

Once the angles are sent to the MCU, the MCU determines which motor has the largest angle to increment and sets the “motor_max” variable equal to the motor number. The motor power values are then calculated by scaling them to the largest angle. For example, if the largest angle is 40 degrees and Motor 2 is to increment 10 degrees, then Motor 2’s power will be set to 25% of maximum power. Then the code goes into a repetitive process of checking the encoders and tracking the process of the angles. A counter was added so that the motor power values will change every so often.

If the user presses a button during the normalized motion, the program will cancel the normalized motion and stop all of the motors. This was added to the code so that the normalized motion is not altered during execution. However, the button command is not executed. If a motor on button or direction button is pressed, the GUI will show it as dark, but the MCU will not reflect this. Therefore, when running the normalized motion and before it has finished, it is best to stop all the motors using the spacebar or “STOP EVERYTHING!!!” button before executing a new command.

To make the motion of the SCMR smoother and reduce the current draw from the power supply, a ramp up function is used to start the motors. The initial calculated motor power of all the motors was divided by the variable “ramp_divisions”. The resulting value was set to be the corresponding motor’s power. The lead motor’s value was set to the variable “ramp_inc”, for ramp increment. A timing buffer, “ramp_timing”, was used so that function would ramp up at specified times set by the user. In this code, each time the code was to ramp up the lead motor’s power, it would add the “ramp_inc” value to the lead motor’s power. This process is repeated until the lead motor’s power is incremented to full power. Since all of the motors adjust their power to follow the progress of the lead motor, ramping up the lead motor’s power will cause the other motors to ramp up their power.

The progress of the motor with the largest angle to increment (lead motor) is calculated and scaled to a set value (“progress_scale” variable). The rest of the motors’ progresses are scaled to the same value. If a motor’s progress is less than the lead motor’s scaled progress, then that motor’s power will increase. If the motor’s progress is more than the lead motor’s scaled progress, then the motor’s power will decrease. If a motor is falling behind on the progress and is at max power, then the lead motor’s power will decrease. If the lead motor is not told to slow down, then its power will increase unless it is already at its maximum power.

The ramp down function for normalized motion does not turn off the motor, but slows it down at certain angles before the specified angle for when the motor is to turn off. This was

added in an attempt to make the motion smoother. Since the code tracks the amount of encoder ticks, the variable “ramp_down_ticks” was created to calibrate how far away from the end to start ramping the power down. The variables “ramp_down_mult” and “ramp_down_div” were created so that the user can easily calibrate how much the ramp down function slows the motor. These variables can be edited at the beginning of the code as part of the “Values to calibrate”. The first variable is multiplied by the motor value and the latter is divided from the motor power; this allows the user to create any fraction without using float variables. This process is repeated at position intervals based on the “ramp_down_ticks” variable.

As soon as a motor has reached its desired angle, it will turn off and increment the variable “motors_done”. Once that variable reaches seven, then all the motors are done and the value “200” is sent to the Processing code to indicate to the user that the normalized incrementing of the motors has completed by turning the normalized button off.

One problem with the normalization code was that the speeds of the motors oscillate as the lead motor slows down and then speeds up for the other motors to catch up. To prevent the lead motor from slowing down so much, a timing buffer called “slow_timing” was added. Every time a motor turns the “slowdown” flag on, the “slow_counter” will increment. Once the “slow_counter” reaches the value of “slow_timing”, then the lead motor will slow down. Also, the lead motor’s speed will not increase unless the “slowdown” flag is off. The “slow_timing” variable can still be calibrated to improve the performance of the SCMR.

The power setting buttons in the GUI control the maximum power the motors are allowed to reach during normalized motion. When one of the buttons is clicked in the GUI, a message is sent to the MCU. The MCU then scales all of the motors’ power and does not allow the motors to exceed that value. This does not affect the power setting for the individual motors when using the on/off or individual angle increment functions.

Ramp up/down functions are applied to the on/off functions to make the motion more smooth. For ramp up, the initial power setting is divided by the “ramp_divisions” variable. The

state of the ramp for each motor is tracked by variables using the motor number, such as “ramp_up1” for Motor 1. For each state, the power for the motor is set to be: $(\text{initial power}) * (\text{ramp_up state}) / (\text{ramp_divisions})$. After a set amount of time, determined by the variable “ramp_up_time”, the state is incremented to increase the motor power. Once the state reaches the value of the “ramp_divisions”, the state returns to zero because now the motor is at its operating power. If the operating power is changed during the ramping phase of the motor turning on, the ramping function will turn off and the motor power will jump to the new operating power. This is so that the motor does not jump down and ramp up every time the power is changed while the motor is on. Also, when the ramp up function is turned on, the ramp down function is turned off so that they do not work against each other.

The ramp down function, for when the motor “Off” button is pressed in the GUI, divides the motor power in half at two different times, and then turns the motor off. A timing buffer, “ramp_down_time”, sets the time interval for when to divide the power in half and when to turn the power off.

Before the code is run, the power input variables must be set. The variable “psupply” is the variable used to scale the power given to the seven motors that control the motion of the SCMR. Since those motors perform best by running at 12V, the power supply is scaled down in case the power supply is greater than 12V. For the testing in section 3.4, the SCMR was powered by a 13.8V power supply. The variable “psupply” was set to 14, and the PWM signal scales the power down to run the motors at 12V by using the equation: $\text{power} = \text{incoming} * 12 / \text{psupply}$, where incoming is the value sent from the GUI. In case the power supply is less than 12V, the variable should be set to 12 so that the full voltage is applied without exceeding the maximum value of the byte.

The second power input variable was used for the grippers, and is called “gpower”. Since these motors run at 7.2V, the minimum value of this variable should ever be is 7.

3.4-Testing Results

The preliminary tests shown in section 3.1.5 proved that the SCMR can move itself through critical positions and can grab a new gripping position by actuating the motors. The tests performed in this section assessed the SCMR's agility and found its limitations in climbing a structure. The first testing had the SCMR traverse across the outside of the structure. The second testing had the SCMR climb inside of the structure following the preset motions mentioned in 3.3.1. The structure used for these tests was made of wood 2x4's with the dimensions 68"x24"x36". Support beams run along the ends to keep the structure stable when the SCMR is stretched out. The structure is shown in Figure 42.

Figure 42. Structure built to test the SCMR.

3.4.1-Climbing Outside of the Structure

The gripping points chosen for the SCMR to grab while traversing the outside of the structure are shown in Figure 43. These points were chosen to prove that the SCMR can traverse along a plane by grabbing beams at different angles. Gripping points 1 and 2 are perpendicular to each, and gripping point 3 is on a beam that is 45 degrees from the beam for gripping point 2. Any angle can be reached in this plane because Motor 1 and Motor 7 allow the grippers to rotate while keeping the same location. Three points were chosen so that each gripper is engaged and released from the structure. The angles that the motors moved were recorded in order to write them into the preset motions. However, due to the limitation of the SCMR's reach discovered during this test, the preset motions were changed to follow a path for the SCMR to climb inside of the structure, discussed in section 3.4.2.

Figure 43. Gripping points for the SCMR traversing the outside of the structure.

For this test, the electronic hardware was mounted to the SCMR. Only three wires were tethered to the SCMR: serial, motor power, and ground. The SCMR began in the “neutral” position and Gripper 1 was attached to gripping point 1, shown in Figure 44.

Figure 44. SCMR in initial “neutral position at gripping point 1.

During the motion from gripping point 1 to 2, the SCMR went through the critical positions. The SCMR was easily controlled to move Gripper 2 to the new gripping point using the preset motion buttons. To power the SCMR, a 13.8V 19A power supply was used. The current draw for running more than two motors exceeded the 19A value in positions where the load on the motors was high. Four motors were able to run simultaneously under light loads, but this action was not very consistent. For this reason, the preset motions written into the Processing code only ran two motors simultaneously. The motion was broken up into four steps to get Gripper 2 near gripping point 2. The rest of the motion was performed by actuating the individual motors to line up the SCMR for a secure grip. The intermediate steps are shown in Figure 45.

Figure 45. Intermediate steps for SCMR moving from gripping point 1 to 2.

Before releasing Gripper 1, the motors were actuated to minimize the load on Gripper 1. This was mostly accomplished by rotating Motor 7 to pick up the SCMR. A couple of other motors had to be adjusted, as well. Upon release, Gripper 1 remained near gripping point 1, as shown in Figure 46.

Figure 46. SCMR with Gripper 1 engaged (left) and released (right) at gripping point 1.

To move the SCMR to gripping point 3, the SCMR swung around the front of the structure as shown in Figure 47 and Figure 48. This motion shows that the SCMR can continuously traverse along a plane. The application of this is to have the SCMR climb up a tall tower. The SCMR can climb along the outside of a structure as long as the gripping points are within reach of each other. The maximum reach of the SCMR for having the two grippers grabbing the same plane is just over two feet. The SCMR can also climb along the same beam repeatedly in the same manner that it traverses in the same plane.

Figure 47. SCMR swinging Gripper 1 from gripping point 1 to 3.

Figure 48. SCMR swinging Gripper 1 from gripping point 1 to 3.

The SCMR easily grabbed gripping point 3, shown in Figure 49. Gripper 2 was able to release, which proved that both grippers can be engaged and released during a climb.

Figure 49. SCMR with grippers engaged at gripping points 2 and 3.

While moving in this plane, Motor 3 and Motor 5 did not change their position. This was because the motions were simple while traversing along the plane. This means that a simpler and lighter SCMR can be made if the climbing is to be done in a plane. This would reduce the cost of the SCMR and the loads applied to the joints. These motors were needed later while climbing inside of the structure due to the more complex motion and space constraint, shown in section 3.4.2.

While Gripper 1 was engaged on gripping point 3, Gripper 2 was moved to reach inside of the structure. Due to motor interference, the SCMR was limited in how far it could penetrate the plane that Gripper 1 was grabbing. Some motors were at their maximum position, but it was not far enough to give the SCMR enough reach and mobility. Figure 50 shows the maximum reach of the SCMR inside of the structure.

Figure 50. Maximum reach of SCMR past front plane of structure.

Gripper 2 was oriented to point down, but was still unable to reach a new gripping point due to the motor interference, as shown in Figure 51.

Figure 51. SCMR limited in reaching past front plane of structure.

In order for the SCMR to have more agility in reaching past the front plane of the structure, a via point is needed. The SCMR can grab a beam that is in a plane that intersects with the front plane of the structure. From there, the SCMR would then be able to reach inside of the structure. The SCMR was tested for grabbing points in multiple planes in section 3.4.2.

3.4.2-Climbing Inside of the Structure

The gripping points chosen for the SCMR to grab inside of the structure are shown in Figure 52. These points were chosen to demonstrate that the SCMR can grab four non-coplanar points at varying angles. The preset buttons were configured to follow the path of gripping points sequentially. Gripping point 1 is on the diagonal beam on the side. Gripping points 2 and 3 are in the back plane of the structure. Gripping point 4 is on the top cross beam.

Figure 52. Gripping points for SCMR climbing inside of structure.

Before the SCMR was attached to the structure, it was put into the “neutral” position as shown previously in Figure 1. While holding Gripper 1 up off of the ground to prevent damage from the motion, the “Neutral to start” button was pressed to get the SCMR in the position shown in Figure 53. From this position, the SCMR was attached to gripping point 1 of the structure, shown in Figure 54. The motion for this position is shown in Table 3. This was done so that the SCMR can easily fit into the structure to be secured. Due to the power supply used, only two motors were programmed to run simultaneously.

Table 3. Motor angles to move SCMR from neutral to starting position.

Neutral to Start		
Motor	Direction	Angle
1	Down	45°
4	Down	80°

Figure 53. SCMR in starting orientation to climb inside of structure.

Figure 54. SCMR attached to gripping point 1 in start position.

Once attached to the structure, the next three buttons in the sequence were pressed: “Move pt 1 to 1a”, “Move pt 1a to 1b”, “Move pt 1b to 1c”. The motions for each of these

buttons are described in Table 4, Table 5, and Table 6. These buttons get the SCMR close to gripping point 2, but not in the final gripping position. This was done so that if the SCMR was slightly off in its position, it would not hit the structure. The position that the SCMR was in after these motions, point 1c, is shown in Figure 55.

Table 4. Motor angles to move SCMR from point 1 to 1a.

Point 1 to 1a		
Motor	Direction	Angle
5	Down	55°
6	Down	65°

Table 5. Motor angles to move SCMR from point 1a to 1b.

Point 1a to 1b		
Motor	Direction	Angle
3	Up	15°
5	Down	20°

Table 6. Motor angles to move SCMR from point 1b to 1c.

Point 1b to 1c		
Motor	Direction	Angle
2	Down	30°
7	Up	20°

Figure 55. SCMR in position 1c.

To move Gripper 2 into a gripping position, the “***Move pt 1c to 2***” button was pressed repeatedly, about 5 times. The button was programmed this way to nudge the gripper into position slowly at the user’s control. A couple other motors had to be actuated to get the gripper lined up with the gripping point. The motion applied by this button is listed in Table 7, and the SCMR attached to both gripping points is shown in Figure 56.

Table 7. Motor angles to move SCMR from point 1c to 2, to be actuated about 5 times.

Point 1c to 2		
Motor	Direction	Angle
3	Up	2°
6	Up	3°

Figure 56. SCMR attached to gripping points 1 and 2 inside the structure.

To minimize the load applied to Gripper 1, the motion described in Table 8 was executed. Once the load was minimized, Gripper 1 was able to release. Several motors were then actuated to move Gripper 1 into the position shown in Figure 57; the motion is described in Table 9. This action is necessary before pressing the next preset motion button. Since these motions were done by sight and by checking if Gripper 1 could open, they were not written into the code as a preset motion. These angles are a suggestion, and will not always be the necessary motion.

Table 8. Suggested motor angles to minimize load on gripping point 1 for Gripper 1 release.

Release from grip pt 1		
Motor	Direction	Angle
2	Up	20°
3	Down	4°
4	Down	10°
5	Down	3°
7	Up	4°

Table 9. Suggested motor angles to move Gripper 1 away from gripping point 1 after release.

Away from grip pt 1		
Motor	Direction	Angle
1	Down	8°
2	Down	6°
4	Down	5°
5	Down	13°

Figure 57. Gripper 1 moved away from gripping point 1, and ready for preset motion.

After Gripper 1 was in the ready position away from gripping point 1, the next three buttons in the sequence of preset motion were pressed: “Move pt 2 to 2a”, “Move pt 2a to 2b”, “Move pt 2b to 2c”. The motions for each of these buttons are described in Table 10, Table 11, and Table 12. Again these motions get Gripper 1 close to the gripping point 3, as shown in Figure 58. The order of the motion was chosen carefully so that Gripper 1 would not hit the structure during the motion to point 2c.

Table 10. Motor angles to move SCMR from point 2 to 2a.

Point 2 to 2a		
Motor	Direction	Angle
2	Down	50°
3	Down	25°

Table 11. Motor angles to move SCMR from point 2a to 2b.

Point 2a to 2b		
Motor	Direction	Angle
1	Down	35°
7	Up	34°

Table 12. Motor angles to move SCMR from point 2b to 2c.

Point 2b to 2c		
Motor	Direction	Angle
5	Up	85°
6	Up	20°

Figure 58. SCMR in position 2c.

To move Gripper 1 into gripping point 3, the “*Move pt 2c to 3*” button was pressed repeatedly, about six times. Still, some other motors had to be slightly adjusted during this

motion to get Gripper 1 lined up with the 2x4. The motors actuated during this motion are described in Table 13, and the resulting position is shown in Figure 59.

Table 13. Motor angles to move SCMR from point 2c to 3, to be actuated about 5 times.

Point 2c to 3		
Motor	Direction	Angle
4	Down	1°
6	Down	3°

Figure 59. SCMR attached to gripping points 2 and 3 inside the structure.

The load applied to Gripper 2 was minimized to prepare it for release. Motors were actuated individually until the gripper was able to open. The motion applied to the motors is described in Table 14. Gripper 2 was then moved into a position ready for the preset motion to gripping point 4. The motors actuated to get Gripper 2 into the ready position are described in

Table 15, and the position is shown in Figure 60. The gripper plate is in line with the top of the 2x4. These motions are suggestions and will not always be the necessary movement.

Table 14. Suggested motor angles to minimize load on gripping point 2 for Gripper 2 release.

Release from grip pt 2		
Motor	Direction	Angle
1	Up	23°
4	Up	5°
6	Up	2°
7	Down	6°

Table 15. Suggested motor angles to move Gripper 2 away from gripping pt 2 after release.

Away from grip pt 2		
Motor	Direction	Angle
2	Down	7°
4	Up	18°
6	Down	11°
7	Down	7°

Figure 60. Gripper 2 moved away from gripping point 2, and ready for preset motion.

The next three preset motions were pressed to get Gripper 2 close to gripping point 4: “Move pt 3 to 3a”, “Move pt 3a to 3b”, “Move pt 3b to 3c”. The fourth preset motion button, “*Move pt 3c to 3d*”, requires the user to press it repeatedly, about five times. This was added because Motor 2 must be near its maximum down position. A preset motion to cover the whole distance might go too far and cause interference between the adjacent motors. This motion is more complicated than the other preset motions because Gripper 2 could hit the structure. To prevent the SCMR from hitting the structure, Motor 1 is actuated down and then up. Also, the motion of Motor 2 is broken up among two buttons. The user may have to actuate some motors before or while pressing the “*Move pt 3c to 3d*” button multiple times to avoid hitting the top cross beam of the structure. This occurs because the angular motion is incremental instead of absolute. The motion for each button is described in Table 16, Table 17, Table 18, and Table 19. The final position of the SCMR after these motions is shown in Figure 61.

Table 16. Motor angles to move SCMR from point 3 to 3a.

Point 3 to 3a		
Motor	Direction	Angle
1	Down	40°
3	Down	53°

Table 17. Motor angles to move SCMR from point 3a to 3b.

Point 3a to 3b		
Motor	Direction	Angle
2	Down	15°
4	Up	40°

Table 18. Motor angles to move SCMR from point 3b to 3c.

Point 3b to 3c		
Motor	Direction	Angle
1	Up	15°
7	Down	53°

Table 19. Motor angles to move SCMR from point 3c to 3d, to be actuated about five times.

Point 3c to 3d		
Motor	Direction	Angle
2	Down	3°
5	Up	2°

Figure 61. SCMR in position 3d.

To move Gripper 2 into a gripping position, the “***Move pt 3d to 4***” button was pressed repeatedly, about four times. The motion performed by this button is described in Table 20. Figure 62 shows the result of the SCMR attached to gripping points 3 and 4.

Table 20. Motor angles to move SCMR from point 3d to 4, to be actuated about four times.

Point 3d to 4		
Motor	Direction	Angle
1	Up	2°
5	Up	1°

Figure 62. SCMR attached to gripping points 3 and 4 inside the structure.

The load applied to Gripper 1 was minimized to prepare it for release. Motors were actuated individually until the gripper was able to open, described in Table 21. Gripper 1 was then moved away from gripping point 3 so that it could be moved to a new position; the motion is described in Table 22 and the final position is shown in Figure 63. The SCMR was not moved to a new position because it proved it could climb a structure by grabbing four non-coplanar points at varying angles.

Table 21. Suggested motor angles to minimize load on gripping point 3 for Gripper 1 release.

Release from grip pt 3		
Motor	Direction	Angle
2	Up	9°
4	Up	13°
6	Down	23°

Table 22. Suggested motor angles to move Gripper 1 away from gripping pt 3 after release.

Away from grip pt 3		
Motor	Direction	Angle
2	Down	9°
4	Down	2°
6	Down	8°
7	Up	6°

Figure 63. Gripper 1 moved away from gripping point 3, and ready for next motion.

Throughout this test, the normalized motion was not as smooth as desired. Not all of the motors started and stopped simultaneously. Some motors oscillated their speed throughout the motion while trying to catch up to the lead motor. Although the calibration variables were adjusted, the problem still exists.

Chapter 4-Discussion

4.1-Conclusion

Overall, the SCMR was successful. The SCMR was able to climb a structure by grabbing four non-coplanar points at non-orthogonal angles. The planetary gear motors with worm gears provided enough torque to power the SCMR through the critical positions of maximum load. The grippers are able to grab and release the structure using only commands from the computer. Releasing the gripper takes some practice to minimize the load on the gripper to be released.

A simpler SCMR can be made to traverse along a plane using only five motors to control the motion. This would reduce the cost and load applied to the joints. However, this design would be limited in its ability to grab beams out of plane. The seven motor design was necessary to climb complex structures where space was limited.

The modeling program preferred to build the SCMR was SimulationX. It was easy to learn and allowed for the building and testing of a user created model. The motion of the robot was simulated and torque values were calculated to help determine what motors would work best for the SCMR. Dymola was also a good modeling program, but the tech support was not as helpful.

The next progressive step for the SCMR is to make it more autonomous. The inverse kinematic equation still needs to be solved so that it is easier to control the motion of all of the joints. This solution can then be applied to have the SCMR climb a known structure. A CAD drawing of the structure can be used to identify gripping points and then plan a path for the SCMR to follow. The SCMR can then go on to inspect certain areas on the structure and possibly perform maintenance if the correct tools are attached to the SCMR.

A major need for the SCMR is to better position the grippers into a secured grip. The small motions to get the gripper in position are very time consuming and make the climbing process slow. This could improve with the solution to the inverse kinematic equation or a redesign of the gripper. The grippers used for the SCMR have a very small clearance for the 2x4 in order to minimize the gripper from sliding along the 2x4. This high tolerance for the gripper position is what takes up most of the time for moving the SCMR to a new position.

4.2-Future Work

In order to improve the SCMR, there are a few more things that can be done to the hardware and software. As for the hardware, the SCMR can be made lighter and stronger. Holes can be drilled into the aluminum plates to make the SCMR lighter without making it weaker. More of the material can be milled away to create the same effect.

To make the SCMR stronger, the screws need to be better secured. The set screws are holding much better by using Loctite and doubling up the set screws. However, some of the set screws still loosen, increasing the backlash. Also, the #8-32 screws that hold the brackets to the motors have loosened after much testing. Although the motor specification indicated screw holes for #10-32, the screws did not fit. The #8-32 screws were able to fit in the holes, but with some extra room. Because of this, some screws are barely held in place by their threads. This might be acceptable because there are many other screws that hold the bracket firmly in place, but the screws should be tightened before each test.

A power supply with more current is needed for the SCMR. This could be done by buying multiple power supplies and connecting them in parallel to increase the current supplied to the SCMR. Since two motors can run off of the 19A power supply from RadioShack, at least two more are needed in case six motors run simultaneously; there might be enough current to run the seventh motor as well.

Lastly for the hardware, some of the gears were not aligned perfectly. This misalignment results in some backlash for the joint that cannot be minimized without rebuilding the joint. To fix the center-to-center distance of the gears, the aluminum brackets can be remade with higher tolerances. When making the brackets, transfer punches were used to line up the holes from the first bracket plate to make the holes on the new bracket plates. The bracket plate and transfer punch could have shifted to center the hole for the ball bearing far enough away to make the backlash noticeable.

As for the software, the main work to do is to solve for the inverse kinematics of the SCMR. The solution can then be used in a new code to control the motion more easily. Before this can be implemented and tested, the power supply issue must be addressed since multiple motors will run during this part of the code. Also, the SCMR codes can be edited to be more efficient and user friendly.

Also, the calibration variables in the software need to be further tested for the best operating values. During normalized motion, the motors are not always synchronized to start and stop simultaneously. Lastly, some motors oscillate their speed while MCU tries to maintain the same progress for each motor.

The next progressive step for the SCMR would be to add sensors to help make the SCMR more autonomous. Since adjusting the gripper manually is time consuming, sensors would be needed to assist the user to position the gripper correctly. In order to perform inspections once the SCMR has climbed a structure, a camera must be mounted to the SCMR.

Bibliography

- [1] Brown, H., M. Friedman, T. Kanade, and Y. Xu, "Self Mobile Space Manipulator Project," *Annual Research Review*, 1990, pp. 66-77.
- [2] "Technion 'Spider Robots' are Designed to Save Lives." American Technion Society. 14 March 2008. <<http://www.ats.org/robot/?id=178>>
- [3] "Robotic Spider Will Save Lives." IsraCast, 1 July 2004. Web. 24 Mar. 2010. <<http://www.isracast.com/article.aspx?id=104>>.
- [4] "Wall-Climbing Robots Use SRI International's Electroadhesion Technology." *SRI International*. 20 May 2008. Web. 20 Apr. 2011. <<http://www.sri.com/news/releases/052008.html>>.
- [5] "SRI International's Electroadhesive Robots." *SRI International*. Feb. 2011. Web. 20 Apr. 2011. <<http://www.sri.com/rd/electroadhesion.html>>.
- [6] Niku, Saeed B. "Robot Kinematics: Position Analysis." *An Introduction to Robotics Analysis, Systems, Applications*. Upper Saddle River, N.J.: Prentice Hall, 2001. 67-80. Print.
- [7] Brain, Marshall. "How Microcontrollers Work." *Howstuffworks "Electronics"*. Web. 21 Apr. 2011. <<http://electronics.howstuffworks.com/microcontroller1.htm>>.
- [8] Barr, Michael. "Pulse Width Modulation," *Embedded Systems Programming*, September 2001, pp. 103-104.
- [9] "A Rock Solid Scratch Encoder" *DJ TechTools*. Web. 25 Apr. 2011. <<http://www.djtechtools.com/forum/showthread.php?t=13910>>.

Appendix

Source of Motors Researched

Table 23. List of motors' source website. The selected motors are highlighted.

Motor #	Source
1	http://www.robotmarketplace.com/products/AB-72004.html
2	http://www.robotmarketplace.com/products/0-B104.html
3	http://www.robotmarketplace.com/products/0-B62.html
4	http://www.robotmarketplace.com/products/0-COPAL60.html
5	http://www.lynxmotion.com/Product.aspx?productID=581&CategoryID=71
6	http://h1071118.hobbyshopnow.com/products/description.asp?prod=JRPS2721
7	http://www.rctoys.com/rc-products/HIT-HS55.html
8	http://www.rctoys.com/rc-toys-and-parts/HIT-HS5245MG/RC-PARTS-HITEC-SERVOS.html?utm_source=googlebase_froogle&utm_medium=US&utm_term=HIT-HS5245MG
9	http://www.lynxmotion.com/Product.aspx?productID=580&CategoryID=71
10	http://www.lynxmotion.com/Product.aspx?productID=597&CategoryID=11
11	http://www.lynxmotion.com/Product.aspx?productID=250&CategoryID=11
12	http://banebots.com/p/MP-42256-550
13	http://banebots.com/pc/MP-36XXX-385/MP-36100-385
14	http://banebots.com/pc/MP-36XXX-545/MP-36100-545
15	http://www.lynxmotion.com/Product.aspx?productID=516&CategoryID=71 > http://www.lynxmotion.com/Product.aspx?productID=516&CategoryID=71
16	http://www.lynxmotion.com/Product.aspx?productID=515&CategoryID=71
17	http://banebots.com/pc/MP-36XXX-540/MP-36125-540
18	http://www.robotshop.ca/home/products/robot-parts/motors/gear-motors/planetary-gearmotors/rbban24-banebots-planetary-gearmotor.html
19	http://www.robotshop.ca/home/products/robot-parts/motors/gear-motors/spur-gearmotors/rbban51-banebots-spur-gearmotor.html
20	http://www.trossenrobotics.com/store/p/4228-256-1-28mm-Planetary-Gearmotor-RS-385-Motor.aspx

Instructions Manual

Installing the Programs

The two programs used for the SCMR are Arduino and Processing. Both programs must be downloaded and installed. The installation process for Arduino can be found at <http://arduino.cc/en/Guide/HomePage>. The installation process for Processing can be found at <http://processing.org/download/>. Step by step instructions are included on the websites.

In the Arduino program, go to the “Tools” dropdown menu, scroll down to “Board”, and then select the board used: Arduino Mega 2560. Once the board is connected to the computer, as explained later, go to the “Tools” dropdown menu, scroll down to “Serial Port” and verify what channel the board is on (i.e. “COM3”). If the Serial Port is not COM3, make the correction in the Processing code of the SCMR so that the GUI can communicate with the Arduino board. In the void setup() function of the SCMR Processing code, change the line:

```
arduino = new Serial(this, "COM3", 9600); // Setup the Serial communication
```

Troubleshooting this is explained in more detail on the websites of the programs.

Setting Up the SCMR

Before running the SCMR, ALL of the screws should be checked to make sure they are tight. If the backlash in some of the joints are too much, the set screws need to be tightened. Most of the set screws are doubled up, so the first one needs to be taken out so that the other one can be tightened. Most of the set screws were secured with Loctite, so they might feel tighter than usual.

Check that all of the wires in the breadboard have not come loose. Refer to Figure 36 if any wires come loose.

Connect the encoders via two cables with 22 gauge wire. The power for the encoders is the two wired cable: black and red. Red is +5V and black is ground. The colors should be matched when connecting and secured with electrical tape.

The cable carrying the encoder signals from the robot is 6 black wires and one green wire. This cable is to be connected to the 6 white wires and one red wire attached to the breadboard. The green and red wire should be connected together and carries the signal for Motor 1. This connection should also be secured with electrical tape.

There are nine cables that run the output of the motor driver chips to the motors. Each cable is labeled with the motor number and the ends have male and female quick disconnects. The disconnects were organized so that the end of each cable has one male and one female disconnect. This prevents connecting the motor backwards which would reverse the direction of spin.

The power for all of the motor driver chips connect together into two wires; this is the 10 gauge white wire. The end of the wires have the same quick disconnect (male-male or female-female), so it is important to make sure the wires are connected correctly to the power supply. One of the long white wires has an extra band of black tape around it on each end, marking it to be ground. The second long white wire is intended for the higher voltage. The recommended power for the motors is 12V, so if the power supply is different, the code needs to be adjusted.

For this project, a 13.8V power supply was used, so the code was told it was using 14V. To change this, edit the beginning of the Arduino code. The third and fourth variables defined are `psupply` and `gpower`. The variable `psupply` is the variable used by the seven motors that control the motion of the SCMR that should not exceed 12V. If the power supply is greater than 12V, change the `psupply` variable to the power supply's voltage (i.e. for the 13.8V power supply, `int psupply = 14;`). The code will scale the high power supply down to 12V using the PWM signal. If the power supply is less than 12V, set `psupply = 12` so that the motors get the full voltage of the power supply.

The second power variable in the code is gpower for the grippers. These motors run at 7V, so the gpower variable should not be set to below 7V. If the power supply is greater than 7V, set the gpower variable to the power supply's voltage (i.e. for the 13.8V power supply, int gpower = 14;). If the power supply is less than 7V, set gpower = 7 so that the motors get the full voltage of the power supply.

Lastly, use a printer cable to connect the computer to the Arduino board. This will power the board and allow for communication between the GUI and the board. If the power cuts out, there could be a short somewhere in the wiring. This occurred during testing because the unused encoder signal was shorting out during motion. That is why all of the unused encoder wires are insulated with electrical tape.

Running the SCMR

Once everything is connected, “upload” the Arduino code to the board using the button . After it is done uploading, “run” the Processing code using the button . Then the GUI will pop up on the computer screen.

If anything goes wrong during testing, or if the user wants to turn off all of the motors simultaneously, press the “STOP EVERYTHING!!!” button in the bottom left corner of the GUI. The same action is achieved by pressing the spacebar. This will stop all nine motors and set the buttons appropriately. The motors and grippers are labeled according the picture below. Note that Gripper 1 has electrical tape marking it. Also, the cables connected to the motors are labeled.

To control the motors individually, the power must be set. The power is set by clicking and dragging the white box with the mouse along the black line. The amount of power that will be applied to the motor when turned on is reported to the right of the black line. The value goes up to 100 which corresponds to 12V (unless the power supply is less than that).

To turn the motors on, the user can click the “On” button next to the power scroll bars, at which point the button will turn dark and the “Off” button will turn light. If a mouse is moved over a button, the buttons will highlight, but the mouse must still be clicked to activate the button. The motor can be toggled on/off using the numbers on the keyboard. Pressing “3” will toggle Motor 3 on/off and the buttons in the GUI will change colors. While the motor is running, the power can be adjusted in real time.

The direction of the motors is set by the “Up” and “Down” buttons. The motor direction can be toggled using the keyboard. By holding shift and pressing the motor number, the motor’s direction will toggle up/down. If the user holds shift and presses “5” (typing “%”) then Motor 5’s direction will change. This should be avoided when the motor is running at high speeds because

the sudden acceleration could cause damage. All of the motors are wired to rotate the output shaft in the same manner, shown below:

To increment the individual motors, adjust the angle scroll bars similar to the power scroll bars. The power for the motor must be set to increment the motor individually. Press the “Increment” button for the desired motor to move. For the setting shown below, Motor 4 would increment 10.0 degrees down while moving at 75% power, which is 9V.

The “Increment” will remain dark until the motor has reached its desired angle. The motor will turn off and the button will become light again. The power of the motor can be adjusted while the motor increments.

The normalized motion will increment multiple motors simultaneously. The motors are supposed to start and stop at the same time, but some calibration of the variables are still needed. The variables to adjust the normalized code are at the beginning of the Arduino code, under the comment “Values to calibrate”. A description of each variable is next to where the variable is defined.

To run the normalized motion, the motor directions and angles must be set by the user.

The next step is to simply press the button Normalized Increment of All Angles . This button will remain dark until all of the motors have reached their desired angles. The power scroll bar has no

effect on the normalized motion, but the maximum power can be set using one of the three buttons in the lower right corner of the GUI:

These buttons can be pressed during the normalized motion to adjust how fast the SCMR will move. The values for each button is written in the beginning of the Arduino code under the “Values to calibrate”: “phigh”, “pmed”, and “plow”. If any other buttons are pressed during normalized motion, all of the motors will turn off. The MCU will ignore the message sent, even if the GUI shows otherwise. If a motor direction was changed, the GUI will show the change, but the MCU will still apply the old direction. For this reason, the normalized motion should finish or be stopped only by the stop button or by pressing the spacebar.

The “Point to Point Motion” buttons act the same as the normalized motion button does, but the motor direction and angles are predetermined and written into the code. The use of these buttons is described in full detail in section 3.4.2 of the report. The red buttons indicate that they should be pressed multiple times.

Troubleshooting

Encoder signal not read by MCU

1. Make sure that the encoder is spinning with the motor. If not, remove the encoder and secure the screw attached to the motor shaft with Loc-Tite.
2. If the signal is always off, check that the encoder is receiving power. Some of the soldered connections could have broken loose.
3. If the signal is always on, check that the encoder is receiving ground signal. Some of the soldered connections could have broken loose.

Motor not turning

1. If the MCU is giving a PWM signal to the motor, an LED should be illuminated on the breadboard.
2. Check that the motor driver chip is receiving power, and power supply is on.
3. Check the the motor driver chip connected to the output wires to the motor.

Motor not turning off when done incrementing

1. Check encoder signal.

Motor only turns one direction

1. Check direction signal from MCU to motor driver chip.

Gripper stalling when attempting to open

1. Adjust motors to minimize load on gripper

MCU powers off on its own

1. Check for short circuit in wiring
2. Make sure the unused cable from encoder is insulated. A short here will short the MCU.

Changing the Preset Motions

To change what the Point to Point Motion buttons do, first find (Ctrl+f) where the button is pressed in the Processing code, example:

```
else if(rectpoint_1_1a.pressed()) { // Increment all the angles, normalized
```

Set the angles of all of the motors, example:

```
arduino.write(13); // Tell the board to change direction to down
mpower[1-1].dir = 0; // Turn the direction flag to down
rectm1up.basecolor = color(light); // Make the up button light
```



```
rectm1down.basecolor = color(dark); // Make the down button dark

arduino.write(22); // Tell the board to change direction to up
mpower[2-1].dir = 1; // Turn the direction flag to up
rectm2up.basecolor = color(dark); // Make the up button dark
rectm2down.basecolor = color(light); // Make the down button light
```

Set all of the motor angles, example:

```
arduino.write(75*2+2); // Motor 1 // Send all of the angles (multiply by 2 then add 2)
arduino.write(10*2+2); // Motor 2 // Limit the number of motors running due to
// power supply
```

Replacing Hardware

Most of the screws used for the brackets of the SCMR are #8-32 at lengths ranging from 3/8" up to 1". The larger set screws are 1/4"-20 and the smaller set screws are #10-24. The corner braces are from Home Depot, ranging from 1" to 2". The drills for the grippers are 7.2V Black and Decker available at Target or Kmart. The screw attached the gripper is 3/8" and is 7" long. The brackets and load bars are made mostly from 1/4" aluminum. The aluminum plate that the motor shaft with the worm penetrates is 3/8". The gripper plates were milled from a block of aluminum. All of the bearings and retaining rings are from McMaster-Carr:

60355K505 - Steel Ball Bearing--abec-1, Open Bearing No. R8 For 1/2" Shaft Dia, 1-1/8" Od
6383K16 - Steel Ball Bearing, Plain Open For 3/8" Shaft Dia, 7/8" Od, 1/4"w
6655K1717 - Steel Thrust Ball Bearing, Steel, For 1/2" Shaft Diameter, 15/16" Od
6655K15 - Steel Thrust Ball Bearing, Steel, For 3/8" Shaft Diameter, 13/16" Od
98420A138 - Heavy Duty Side-mount External Retaining Ring, Black-finish Steel, Style 2, For 1/2" Shaft Dia

Arduino Code

```
/*
 * Paul Bessent
 * Created: 2/10/11
 * Last updated: 5/2/11
 *
 * Version 2.4 for the SCMR
 * Run this code first, then run the corresponding Processing code
 *
 * This code receives messages from the Processing code to actuate
 * the motors. This code also performs normalized motion for angle
 * increment of multiple motors.
 *
 */

int reading = 0;          // This int indicates if the code will read a
                           // power or angle value for a specified motor
byte incoming = 0;       // Value incoming from the processing code

// Set voltage of power supply in order to scale to 12 volts
// if less than 12, set value to 12
int psupply = 14;
// Set voltage applied to grippers in order to scale to 7 volts
int gpower = 14;

// Values to calibrate-----

long progress_scale = 500; // Factor multiplied by progress fraction
int power_inc = 5;        // Amount to increment/decrement the motor
                           // power during normalized angle
                           // incrementation
long timing_buffer = 100; // Timing buffer for adjusting motor speeds
                           // during
                           // normalized incrementation
long ramp_down_time = 2000; // When signal to turn motor off, ramp the
                           // speed
                           // down for smooth motion
long ramp_up_time = 1000;  // When signal to turn motor on, ramp the
                           // speed
                           // up for smooth motion
int ramp_divisions = 10;   // Number of steps to take to max speed
int rdticks_high = 50;     // Number of ticks from end to start
                           // ramping down
                           // when power is high
int rdticks_med = 30;      // Number of ticks from end to start
                           // ramping down
                           // when power is medium
int rdticks_low = 20;      // Number of ticks from end to start
                           // ramping down
                           // when power is low
int ramp_down_mult = 2;    // For ramping down, value to multiply by
                           // motor\
                           // power
int ramp_down_div = 3;     // For ramping down, value by which to
                           // divide the
```

```

//      motor power
long slow_timing = 200; // Time when to slowdown lead motor
long ramp_timing = 1000; // Timing buffer for ramp during normalized
motion
int phigh = 250; // High power setting for normalized motion
int pmed = 200; // Medium power setting for normalized
motion
int plow = 150; // Low power setting for normalized motion

// Set the pins for motors' pwm: must be sequential
int m1onpin = 2;
int m2onpin = 3;
int m3onpin = 4;
int m4onpin = 5;
int m5onpin = 6;
int m6onpin = 7;
int m7onpin = 8;

// Set the pins for the motors' direction: must be sequential
int m1dirpin = 22;
int m2dirpin = 23;
int m3dirpin = 24;
int m4dirpin = 25;
int m5dirpin = 26;
int m6dirpin = 27;
int m7dirpin = 28;

// Set the pins for the motor encoders: must be sequential
int m1encpin = 42;
int m2encpin = 43;
int m3encpin = 44;
int m4encpin = 45;
int m5encpin = 46;
int m6encpin = 47;
int m7encpin = 48;

// Set pins for gripper motors' pwm: must be sequential
// after m7onpin
int g1onpin = 9;
int g2onpin = 10;

// Set pins for gripper motors' direction: must be sequential
// after m7dirpin
int g1dirpin = 29;
int g2dirpin = 30;

// Power value for specified motor
int m1power = 0;
int m2power = 0;
int m3power = 0;
int m4power = 0;
int m5power = 0;
int m6power = 0;
int m7power = 0;

// Angle value to increment for specified motor
long m1angle = 0;

```

```

long m2angle = 0;
long m3angle = 0;
long m4angle = 0;
long m5angle = 0;
long m6angle = 0;
long m7angle = 0;

// Angle value to increment during normalized incrementation
long m1angle_end = 0;
long m2angle_end = 0;
long m3angle_end = 0;
long m4angle_end = 0;
long m5angle_end = 0;
long m6angle_end = 0;
long m7angle_end = 0;

// Current angle value during normalized incrementation
long m1angle_now = 0;
long m2angle_now = 0;
long m3angle_now = 0;
long m4angle_now = 0;
long m5angle_now = 0;
long m6angle_now = 0;
long m7angle_now = 0;

// Largest angle to incrment
long angle_max = 0;

// Motor with angle_max
int motor_max = 0;

// Progress of lead motor
long angle_progress = 0;

// Flag indicating to slow the lead motor down during
//   normalized angle incrementation
int slowdown = 0;

// Number of motors that completed the normalized incrementation
int motors_done = 0;

// Timing counter for normalized incrementation
long timing = 0;

// Ramp down variables-----
// Time counter for each motor while ramping down
long ramp_down_time1 = 0;
long ramp_down_time2 = 0;
long ramp_down_time3 = 0;
long ramp_down_time4 = 0;
long ramp_down_time5 = 0;
long ramp_down_time6 = 0;
long ramp_down_time7 = 0;

// State of motor for ramp down when turning off
int ramp_down1 = 0;
int ramp_down2 = 0;

```

```

int ramp_down3 = 0;
int ramp_down4 = 0;
int ramp_down5 = 0;
int ramp_down6 = 0;
int ramp_down7 = 0;

// State of motor for ramp down during normalized motion
int ramp_down1_norm = 0;
int ramp_down2_norm = 0;
int ramp_down3_norm = 0;
int ramp_down4_norm = 0;
int ramp_down5_norm = 0;
int ramp_down6_norm = 0;
int ramp_down7_norm = 0;

// Ramp up variables-----
// Time counter for each motor while ramping up
long ramp_up_time1 = 0;
long ramp_up_time2 = 0;
long ramp_up_time3 = 0;
long ramp_up_time4 = 0;
long ramp_up_time5 = 0;
long ramp_up_time6 = 0;
long ramp_up_time7 = 0;

// State of ramp up for each motor when turning on
int ramp_up1 = 0;
int ramp_up2 = 0;
int ramp_up3 = 0;
int ramp_up4 = 0;
int ramp_up5 = 0;
int ramp_up6 = 0;
int ramp_up7 = 0;

// Amount to increment each motor during ramp up
int ramp_inc1 = 0;
int ramp_inc2 = 0;
int ramp_inc3 = 0;
int ramp_inc4 = 0;
int ramp_inc5 = 0;
int ramp_inc6 = 0;
int ramp_inc7 = 0;
//-----

// State of ramp up for normalized motion
int ramp = 0;
// Amount to increment the lead motor
int ramp_inc = 0;
// Value incremented every pass through
long ramp_counter = 0;
// Counter for slowdown of lead motor
long slow_counter = 0;
// Flag for normalized motion
int normalized = 0;
// Number of encoder ticks from end to start ramping down
int ramp_down_ticks = rdticks_low;

```

```

// Motor progress
long m1progress = 0;
long m2progress = 0;
long m3progress = 0;
long m4progress = 0;
long m5progress = 0;
long m6progress = 0;
long m7progress = 0;

// Old progress value of motor, currently not used
long m1progress_old = 0;
long m2progress_old = 0;
long m3progress_old = 0;
long m4progress_old = 0;
long m5progress_old = 0;
long m6progress_old = 0;
long m7progress_old = 0;

// Flag indicating the encoder value is high
int m1encon = 0;
int m2encon = 0;
int m3encon = 0;
int m4encon = 0;
int m5encon = 0;
int m6encon = 0;
int m7encon = 0;

int pmax = plow; // Set the max power setting to low at start

/* Value to be multiplied by the angle received to scale correctly
 * Ticks per revolution = 180
 * Degrees incremented per revolution = 360/20 = 18 (20:1 gear ratio)
 * Incoming angle value scale factor = 2
 * ang_scale = 180/(18*2)
 */
int ang_scale = 5;

void setup()
{
    for(int i=2; i<=10; i++) { // Set pins to be output for the motor
pwm
        pinMode(i, OUTPUT);
    }
    for(int i=22; i<=30; i++) { // Set the pins to be output for motor
direction
        pinMode(i, OUTPUT);
    }
    for(int i=40; i<=53; i++) { // Set the pins to be input for
encoders
        pinMode(i, INPUT);
    }
    Serial.begin(9600); // Start the Serial communication at a rate of
9600 bps (baud)
}

//-----Main loop-----
void loop()

```

```

{
  if (Serial.available() > 0) {    // Check if a value has been sent
    from Processing
    incoming = Serial.read();      // Set the value sent to the
    "incoming" int
    // Check for emergency stop, also stop if button (other than power
    setting) is pressed
    // during normalized motion
    if ((incoming == 1)||((reading == 1)||(reading == 2))&&((incoming
    != 101)&&(incoming != 102)&&(incoming != 103)))) {
      for(int i=g1onpin; i<=g2onpin; i++) {    // stop the gripper
motors
        analogWrite(i, 0);
      }
      // Ramp all motors down for smoother motion
      ramp_down1 = 1;
      ramp_down2 = 1;
      ramp_down3 = 1;
      ramp_down4 = 1;
      ramp_down5 = 1;
      ramp_down6 = 1;
      ramp_down7 = 1;
      // Stop ramping up all of the motors
      ramp_up1 = 0;
      ramp_up2 = 0;
      ramp_up3 = 0;
      ramp_up4 = 0;
      ramp_up5 = 0;
      ramp_up6 = 0;
      ramp_up7 = 0;
      // Reset the direction pin of the grippers
      digitalWrite(g1dirpin, 0);
      digitalWrite(g2dirpin, 0);
      // Clear ALL angles in case they were incremented
      m1angle = 0;
      m2angle = 0;
      m3angle = 0;
      m4angle = 0;
      m5angle = 0;
      m6angle = 0;
      m7angle = 0;
      // Turn off normalization flag
      normalized = 0;
      // If in normalized motion, clear appropriate variables
      if((reading == 1)||(reading == 2)) {
        reading = 0;
        motors_done = 0;
        m1progress_old = 0;
        m2progress_old = 0;
        m3progress_old = 0;
        m4progress_old = 0;
        m5progress_old = 0;
        m6progress_old = 0;
        m7progress_old = 0;
        Serial.write(200);
      }
    }
  }
}

```

```

        Serial.flush(); // Clear the Serial
    }
    else if (reading == 0) { // When reading==0, then a power/angle
value is not expected
        switch (incoming) { // Use switch-case to execute desired
command

            // Setup normalized incrementation of all motors-----
        case 2:
            reading = 101; // Indicate that all the motor
values are coming
            normalized = 1; // Turn on normalized flag
            ramp_down1 = 0; // Stop ramp down functions
            ramp_down2 = 0;
            ramp_down3 = 0;
            ramp_down4 = 0;
            ramp_down5 = 0;
            ramp_down6 = 0;
            ramp_down7 = 0;
        break;

            // MOTOR 1-----
        case 10: // Turn motor 1 off
            ramp_down1 = 1; // Turn off using ramp down
function
            ramp_up1 = 0; // Stop ramping up
            mangle = 0; // Clear the angle
        break;
        case 11: // Turn motor 1 on
            ramp_down1 = 0; // Stop ramping down
            ramp_up1 = 1; // Turn on ramping up
function
            mlpower = mlpower/ramp_divisions; // Divide power for ramp up
function
        break;
        case 12: // Turn motor 1 to the up
direction
            digitalWrite(mldirpin, HIGH);
        break;
        case 13: // Turn motor 1 to the down
direction
            digitalWrite(mldirpin, LOW);
        break;
        case 14: // Change motor 1 power
            ramp_down1 = 0; // Stop ramping down
            ramp_up1 = 0; // Stop ramping up
            reading = 11; // Change the reading value to
expect motor 1 power value
// as the next incoming value
        break;
        case 15: // Change motor 1 angle
            mangle = 0; // Clear the angle
            reading = 12; // Change the reading value to
expect motor 1 angle value
// as the next incoming value
        break;

```



```

        case 16:                                // Change motor speed while
running
        ramp_down1 = 0;                        // Stop ramping down
        ramp_up1 = 0;                          // Stop ramping up
        analogWrite(m1onpin, m1power);        // Set the pwm
        break;

// MOTOR 2-----
        case 20:                                // Turn motor 2 off
        ramp_down2 = 1;                        // Turn off using ramp down
function
        ramp_up2 = 0;                          // Stop ramping up
        m2angle = 0;                          // Clear the angle
        break;
        case 21:                                // Turn motor 2 on
        ramp_down2 = 0;                        // Stop ramping down
        ramp_up2 = 1;                          // Turn on ramping up
function
        m2power = m2power/ramp_divisions; // Divide power for ramp up
function
        break;
        case 22:                                // Turn motor 2 to the up
direction
        digitalWrite(m2dirpin, HIGH);
        break;
        case 23:                                // Turn motor 2 to the down
direction
        digitalWrite(m2dirpin, LOW);
        break;
        case 24:                                // Change motor 2 power
        ramp_down2 = 0;                        // Stop ramping down
        ramp_up2 = 0;                          // Stop ramping up
        reading = 21;                          // Change the reading value to
expect motor 2 power value
        // as the next incoming value
        break;
        case 25:                                // Change motor 2 angle
        m2angle = 0;                          // Clear the angle
        reading = 22;                          // Change the reading value to
expect motor 2 angle value
        // as the next incoming value
        break;
        case 26:                                // Change motor speed while
running
        ramp_down2 = 0;                        // Stop ramping down
        ramp_up2 = 0;                          // Stop ramping up
        analogWrite(m2onpin, m2power);        // Set the pwm
        break;

// MOTOR 3-----
        case 30:                                // Turn motor 3 off
        ramp_down3 = 1;                        // Turn off using ramp down
function
        ramp_up3 = 0;                          // Stop ramping up
        m3angle = 0;                          // Clear the angle
        break;
        case 31:                                // Turn motor 3 on

```

```

        ramp_down3 = 0; // Stop ramping down
        ramp_up3 = 1; // Turn on ramping up
function
    m3power = m3power/ramp_divisions; // Divide power for ramp up
function
    break;
    case 32: // Turn motor 3 to the up
direction
    digitalWrite(m3dirpin, HIGH);
    break;
    case 33: // Turn motor 3 to the down
direction
    digitalWrite(m3dirpin, LOW);
    break;
    case 34: // Change motor 3 power
        ramp_down3 = 0; // Stop ramping down
        ramp_up3 = 0; // Stop ramping up
        reading = 31; // Change the reading value to
expect motor 3 power value // as the next incoming value

    break;
    case 35: // Change motor 3 angle
        m3angle = 0; // Clear the angle
        reading = 32; // Change the reading value to
expect motor 3 angle value // as the next incoming value

    break;
    case 36: // Change motor speed while
running
        ramp_down3 = 0; // Stop ramping down
        ramp_up3 = 0; // Stop ramping up
        analogWrite(m3onpin, m3power); // Set the pwm
        break;

// MOTOR 4-----
    case 40: // Turn motor 4 off
        ramp_down4 = 1; // Turn off using ramp down
function
        ramp_up4 = 0; // Stop ramping up
        m4angle = 0; // Clear the angle
        break;
    case 41: // Turn motor 4 on
        ramp_down4 = 0; // Stop ramping down
        ramp_up4 = 1; // Turn on ramping up
function
        m4power = m4power/ramp_divisions; // Divide power for ramp up
function
        break;
    case 42: // Turn motor 4 to the up
direction
        digitalWrite(m4dirpin, HIGH);
        break;
    case 43: // Turn motor 4 to the down
direction
        digitalWrite(m4dirpin, LOW);
        break;
    case 44: // Change motor 4 power

```

```

        ramp_down4 = 0;           // Stop ramping down
        ramp_up4 = 0;           // Stop ramping up
        reading = 41;           // Change the reading value to
expect motor 4 power value      // as the next incoming value

        break;
        case 45:                 // Change motor 4 angle
        m4angle = 0;           // Clear the angle
        reading = 42;           // Change the reading value to
expect motor 4 angle value      // as the next incoming value

        break;
        case 46:                 // Change motor speed while
running                           running
        ramp_down4 = 0;           // Stop ramping down
        ramp_up4 = 0;           // Stop ramping up
        analogWrite(m4onpin, m4power); // Set the pwm
        break;

// MOTOR 5-----
        case 50:                 // Turn motor 5 off
        ramp_down5 = 1;           // Turn off using ramp down
function
        ramp_up5 = 0;           // Stop ramping up
        m5angle = 0;           // Clear the angle
        break;
        case 51:                 // Turn motor 5 on
        ramp_down5 = 0;           // Stop ramping down
        ramp_up5 = 1;           // Turn on ramping up
function
        m5power = m5power/ramp_divisions; // Divide power for ramp up
function
        break;
        case 52:                 // Turn motor 5 to the up
direction
        digitalWrite(m5dirpin, HIGH);
        break;
        case 53:                 // Turn motor 5 to the down
direction
        digitalWrite(m5dirpin, LOW);
        break;
        case 54:                 // Change motor 5 power
        ramp_down5 = 0;           // Stop ramping down
        ramp_up5 = 0;           // Stop ramping up
        reading = 51;           // Change the reading value to
expect motor 5 power value      // as the next incoming value

        break;
        case 55:                 // Change motor 5 angle
        m5angle = 0;           // Clear the angle
        reading = 52;           // Change the reading value to
expect motor 5 angle value      // as the next incoming value

        break;
        case 56:                 // Change motor speed while
running                           running
        ramp_down5 = 0;           // Stop ramping down

```

```

        ramp_up5 = 0; // Stop ramping up
        analogWrite(m5onpin, m5power); // Set the pwm
        break;

// MOTOR 6-----
case 60: // Turn motor 6 off
    ramp_down6 = 1; // Turn off using ramp down
function
    ramp_up6 = 0; // Stop ramping up
    m6angle = 0; // Clear the angle
    break;
case 61: // Turn motor 6 on
    ramp_down6 = 0; // Stop ramping down
    ramp_up6 = 1; // Turn on ramping up
function
    m6power = m6power/ramp_divisions; // Divide power for ramp up
function
    break;
case 62: // Turn motor 6 to the up
direction
    digitalWrite(m6dirpin, HIGH);
    break;
case 63: // Turn motor 6 to the down
direction
    digitalWrite(m6dirpin, LOW);
    break;
case 64: // Change motor 6 power
    ramp_down6 = 0; // Stop ramping down
    ramp_up6 = 0; // Stop ramping up
    reading = 61; // Change the reading value to
expect motor 6 power value
// as the next incoming value
    break;
case 65: // Change motor 6 angle
    m6angle = 0; // Clear the angle
    reading = 62; // Change the reading value to
expect motor 6 angle value
// as the next incoming value
    break;
case 66: // Change motor speed while
running
    ramp_down6 = 0; // Stop ramping down
    ramp_up6 = 0; // Stop ramping up
    analogWrite(m6onpin, m6power); // Set the pwm
    break;

// MOTOR 7-----
case 70: // Turn motor 7 off
    ramp_down7 = 1; // Turn off using ramp down
function
    ramp_up7 = 0; // Stop ramping up
    m7angle = 0; // Clear the angle
    break;
case 71: // Turn motor 7 on
    ramp_down7 = 0; // Stop ramping down
    ramp_up7 = 1; // Turn on ramping up
function

```

```

        m7power = m7power/ramp_divisions; // Divide power for ramp up
function
    break;
    case 72:                                // Turn motor 7 to the up
direction
        digitalWrite(m7dirpin, HIGH);
    break;
    case 73:                                // Turn motor 7 to the down
direction
        digitalWrite(m7dirpin, LOW);
    break;
    case 74:                                // Change motor 7 power
        ramp_down7 = 0;                    // Stop ramping down
        ramp_up7 = 0;                      // Stop ramping up
        reading = 71;                      // Change the reading value to
expect motor 7 power value
                                                // as the next incoming value
    break;
    case 75:                                // Change motor 7 angle
        m7angle = 0;                      // Clear the angle
        reading = 72;                      // Change the reading value to
expect motor 7 angle value
                                                // as the next incoming value
    break;
    case 76:                                // Change motor speed while
running
        ramp_down7 = 0;                    // Stop ramping down
        ramp_up7 = 0;                      // Stop ramping up
        analogWrite(m7onpin, m7power);     // Set the pwm
    break;

// Gripper 1-----
case 80:                                // Set Gripper 1 off
    analogWrite(g1onpin, 0);              // Set pwm to zero
    digitalWrite(g1dirpin, LOW);          // Set direction to open
break;
case 81:                                // Set Gripper 1 to close
    digitalWrite(g1dirpin, HIGH);         // Set direction to close
    analogWrite(g1onpin, 255*7/gpower);    // Set pwm to 7 V
break;
case 82:                                // Set Gripper 1 to open
    digitalWrite(g1dirpin, LOW);          // Set the direction to open
    analogWrite(g1onpin, 255*7/gpower);    // Set pwm to 7 V
break;

// Gripper 2-----
case 90:                                // Turn Gripper 2 off
    analogWrite(g2onpin, 0);              // Set pwm to zero
    digitalWrite(g2dirpin, LOW);          // Set direction to open
break;
case 91:                                // Set Gripper 2 to close
    digitalWrite(g2dirpin, HIGH);         // Set direction to close
    analogWrite(g2onpin, 255*7/gpower);    // Set pwm to 7 V
break;
case 92:                                // Set Gripper 2 to open
    digitalWrite(g2dirpin, LOW);          // Set direction to open
    analogWrite(g2onpin, 255*7/gpower);    // Set pwm to 7 V

```

```

        break;
    }
}
// Set the power-----
else if(reading == 11) {    // Reading motor 1 power value
    m1power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 21) {    // Reading motor 2 power value
    m2power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 31) {    // Reading motor 3 power value
    m3power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 41) {    // Reading motor 4 power value
    m4power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 51) {    // Reading motor 5 power value
    m5power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 61) {    // Reading motor 6 power value
    m6power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
else if(reading == 71) {    // Reading motor 7 power value
    m7power = incoming*12/psupply;    // Set the power value
    reading = 0;    // Reset the reading value
}
// Set the angle-----
else if(reading == 12) {    // Reading motor 1 angle
    m1angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle    // A value of 2 was added when sent to
avoid sending a 1 or 0    // A value of 1 was added after scaling in
case the angle is 0    // so that the incrementation code will
run
    analogWrite(m1onpin, m1power);    // Turn the motor on
    reading = 0;    // Reset the reading value
}
else if(reading == 22) {    // Reading motor 2 angle
    m2angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle    // A value of 2 was added when sent to
avoid sending a 1 or 0    // A value of 1 was added after scaling in
case the angle is 0    // so that the incrementation code will
run
    analogWrite(m2onpin, m2power);    // Turn the motor on
    reading = 0;    // Reset the reading value
}

```

```

        else if(reading == 32) {    // Reading motor 3 angle
            m3angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle
// A value of 2 was added when sent to
avoid sending a 1 or 0
// A value of 1 was added after scaling in
case the angle is 0
// so that the incrementation code will
run
            analogWrite(m3onpin, m3power);    // Turn the motor on
            reading = 0;    // Reset the reading value
        }
        else if(reading == 42) {    // Reading motor 4 angle
            m4angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle
// A value of 2 was added when sent to
avoid sending a 1 or 0
// A value of 1 was added after scaling in
case the angle is 0
// so that the incrementation code will
run
            analogWrite(m4onpin, m4power);    // Turn the motor on
            reading = 0;    // Reset the reading value
        }
        else if(reading == 52) {    // Reading motor 5 angle
            m5angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle
// A value of 2 was added when sent to
avoid sending a 1 or 0
// A value of 1 was added after scaling in
case the angle is 0
// so that the incrementation code will
run
            analogWrite(m5onpin, m5power);    // Turn the motor on
            reading = 0;    // Reset the reading value
        }
        else if(reading == 62) {    // Reading motor 6 angle
            m6angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle
// A value of 2 was added when sent to
avoid sending a 1 or 0
// A value of 1 was added after scaling in
case the angle is 0
// so that the incrementation code will
run
            analogWrite(m6onpin, m6power);    // Turn the motor on
            reading = 0;    // Reset the reading value
        }
        else if(reading == 72) {    // Reading motor 7 angle
            m7angle = (incoming-2)*ang_scale+1;    // Scale the incoming
value to correct angle
// A value of 2 was added when sent to
avoid sending a 1 or 0
// A value of 1 was added after scaling in
case the angle is 0
// so that the incrementation code will
run

```

```

        analogWrite(m7onpin, m7power);          // Turn the motor on
        reading = 0;                             // Reset the reading value
    }

    // Set all angles for normalized motion-----
    else if(reading == 107) {
        m7angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m7angle_now = 0;                         // Clear current angle
        if (m7angle_end == 0) {                  // If no angle to increment
            motors_done += 1;                    // Increment the amount of motors
        }
        reading = 1;                             // Indicate that all
the angles are set
    }
    else if(reading == 106) {
        m6angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m6angle_now = 0;                         // Clear current angle
        if (m6angle_end == 0) {                  // If no angle to increment
            motors_done += 1;                    // Increment the amount of motors
        }
        reading = 107;    // Tell the code to set motor 6 angle with the
next incoming value
    }
    else if(reading == 105) {
        m5angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m5angle_now = 0;                         // Clear current angle
        if (m5angle_end == 0) {                  // If no angle to increment
            motors_done += 1;                    // Increment the amount of motors
        }
        reading = 106;    // Tell the code to set motor 6 angle with the
next incoming value
    }
    else if(reading == 104) {
        m4angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m4angle_now = 0;                         // Clear current angle
        if (m4angle_end == 0) {                  // If no angle to increment
            motors_done += 1;                    // Increment the amount of motors
        }
        reading = 105;    // Tell the code to set motor 5 angle with the
next incoming value
    }
    else if(reading == 103) {
        m3angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m3angle_now = 0;                         // Clear current angle
        if (m3angle_end == 0) {                  // If no angle to increment
            motors_done += 1;                    // Increment the amount of motors
        }
    }
}

```



```

        reading = 104;    // Tell the code to set motor 4 angle with the
next incoming value
    }
    else if(reading == 102) {
        m2angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m2angle_now = 0;    // Clear current angle
        if (m2angle_end == 0) {    // If no angle to increment
            motors_done += 1;    // Increment the amount of motors
done
        }
        reading = 103;    // Tell the code to set motor 3 angle with the
next incoming value
    }
    else if(reading == 101) {
        m1angle_end = (incoming-2)*ang_scale;    // Scale the incoming
value to correct angle
        m1angle_now = 0;    // Clear current angle
        if (m1angle_end == 0) {    // If no angle to increment
            motors_done += 1;    // Increment the amount of motors
done
        }
        reading = 102;    // Tell the code to set motor 2 angle with the
next incoming value
    }
    switch (incoming) {
        // Max Power Setting for normalized motion
        case 101:    // High power setting
            if(normalized == 1) {
                // Scale current power during normalized motion
                m1power = m1power*phigh/pmax;
                m2power = m2power*phigh/pmax;
                m3power = m3power*phigh/pmax;
                m4power = m4power*phigh/pmax;
                m5power = m5power*phigh/pmax;
                m6power = m6power*phigh/pmax;
                m7power = m7power*phigh/pmax;
            }
            // Change ramp_down_ticks variable
            ramp_down_ticks = rdticks_high;
            // Set max power value
            pmax = phigh;
            break;
        case 102:    // Medium power setting
            if(normalized == 1) {
                // Scale current power during normalized motion
                m1power = m1power*pmed/pmax;
                m2power = m2power*pmed/pmax;
                m3power = m3power*pmed/pmax;
                m4power = m4power*pmed/pmax;
                m5power = m5power*pmed/pmax;
                m6power = m6power*pmed/pmax;
                m7power = m7power*pmed/pmax;
            }
            // Change ramp_down_ticks variable
            ramp_down_ticks = rdticks_med;
            // Set max power value

```

```

    pmax = pmed;
break;
case 103:           // Low power setting
    if(normalized == 1) {
        // Scale current power during normalized motion
        m1power = m1power*plow/pmax;
        m2power = m2power*plow/pmax;
        m3power = m3power*plow/pmax;
        m4power = m4power*plow/pmax;
        m5power = m5power*plow/pmax;
        m6power = m6power*plow/pmax;
        m7power = m7power*plow/pmax;
    }
    // Change ramp_down_ticks variable
    ramp_down_ticks = rdticks_low;
    // Set max power value
    pmax = plow;
break;
}
}

// Normalized Motor Incrementation-----
if (reading == 1) {    // Find maximum angle to increment
    angle_max = max(m1angle_end, m2angle_end);
    angle_max = max(angle_max, m3angle_end);
    angle_max = max(angle_max, m4angle_end);
    angle_max = max(angle_max, m5angle_end);
    angle_max = max(angle_max, m6angle_end);
    angle_max = max(angle_max, m7angle_end);

    // If all angles are zero, abort normalized motion
    if (angle_max == 0) {
        reading = 0;    // Reset the reading flag
        for(int i=m1onpin; i<=g2onpin; i++) {        // stop ALL motors
            analogWrite(i, 0);
        }

        m1angle = 0;           // Clear ALL angles
        m2angle = 0;
        m3angle = 0;
        m4angle = 0;
        m5angle = 0;
        m6angle = 0;
        m7angle = 0;

        normalized = 0;    // Clear normalized flag

        Serial.write(200);    // Tell the processing code that all motors
are done

        Serial.flush();    // Clear the Serial
        goto BAILOUT;    // Jump to end of code
    }

    // Determine what motor has the maximum angle to increment
    // Then set the motor_max variable to the motor number
    if (angle_max == m1angle_end) {

```

```

    motor_max = 1;
}
else if (angle_max == m2angle_end) {
    motor_max = 2;
}
else if (angle_max == m3angle_end) {
    motor_max = 3;
}
else if (angle_max == m4angle_end) {
    motor_max = 4;
}
else if (angle_max == m5angle_end) {
    motor_max = 5;
}
else if (angle_max == m6angle_end) {
    motor_max = 6;
}
else {
    motor_max = 7;
}

// Scale the motor values based on angle to increment
m1power = m1angle_end*pmax/angle_max;
m2power = m2angle_end*pmax/angle_max;
m3power = m3angle_end*pmax/angle_max;
m4power = m4angle_end*pmax/angle_max;
m5power = m5angle_end*pmax/angle_max;
m6power = m6angle_end*pmax/angle_max;
m7power = m7angle_end*pmax/angle_max;

// Scale the motor power for power supply and divide by
ramp_division
// for ramp up function
m1power = m1power*12/psupply/ramp_divisions;
m2power = m2power*12/psupply/ramp_divisions;
m3power = m3power*12/psupply/ramp_divisions;
m4power = m4power*12/psupply/ramp_divisions;
m5power = m5power*12/psupply/ramp_divisions;
m6power = m6power*12/psupply/ramp_divisions;
m7power = m7power*12/psupply/ramp_divisions;

// Turn all motors on to their calculated power value
analogWrite(m1onpin, m1power);
analogWrite(m2onpin, m2power);
analogWrite(m3onpin, m3power);
analogWrite(m4onpin, m4power);
analogWrite(m5onpin, m5power);
analogWrite(m6onpin, m6power);
analogWrite(m7onpin, m7power);

// Set the ramp increment value
ramp_inc = max(m1power, m2power);
ramp_inc = max(ramp_inc, m3power);
ramp_inc = max(ramp_inc, m4power);
ramp_inc = max(ramp_inc, m5power);
ramp_inc = max(ramp_inc, m6power);
ramp_inc = max(ramp_inc, m7power);

```

```

        reading = 2;           // Tell the code to do active speed control to
normalize the motors
        ramp = 1;             // Set the ramp variable to the first state
    }

    // Continuous speed control for normalized motion-----
    if(reading == 2)  {

        // Check encoders and increment angle when necessary

        // Motor 1 encoder
        if (mlangle_now < mlangle_end)  { // Check encoder if still need
to increment angle
            // Read the encoder, if high then check if it was low
            if (digitalRead(mlencpin) == HIGH)  {
                if (mlencon == 0)  { // Check if the encoder was low
                    mlencon = 1; // Turn encoder flag high
                    mlangle_now += 1; // Increment the angle
                }
            }
            else { // If the encoder is reading low, check
if it was high
                if (mlencon == 1)  { // Check if the encoder was high
                    mlencon = 0; // Turn the encoder flag low
                    mlangle_now += 1; // Increment the angle
                }
            }
            if(mlangle_now >= mlangle_end)  { // Check if the desired angle
has been reached
                mlpower = 0; // Set the power to zero
                motors_done += 1; // Increment the amount of motors
done
            }

            // Ramp power down when it reaches specified angles
            if (((mlangle_end-mlangle_now) ==
ramp_down_ticks)&&(ramp_down1_norm == 0)&&(ramp == 0))  {
                ramp_down1_norm = 1; // Set the state of the ramp for motor
                mlpower = mlpower*ramp_down_mult/ramp_down_div; // Decrease
motor power
            }
            if (((mlangle_end-mlangle_now) ==
(ramp_down_ticks/2))&&(ramp_down1_norm == 1))  {
                ramp_down1_norm = 2; // Set the state of the ramp for motor
                mlpower = mlpower*ramp_down_mult/ramp_down_div; // Decrease
motor power
            }
            if (((mlangle_end-mlangle_now) ==
(ramp_down_ticks/4))&&(ramp_down1_norm == 2))  {
                ramp_down1_norm = 0; // Clear the state of the ramp
                mlpower = mlpower*ramp_down_mult/ramp_down_div; // Decrease
motor power
            }
        }
        // Motor 2 encoder

```

```

    if (m2angle_now < m2angle_end) { // Check encoder if still need
to increment angle
    // Read the encoder, if high then check if it was low
    if (digitalRead(m2encpin) == HIGH) {
        if (m2encon == 0) { // Check if the encoder was low
            m2encon = 1; // Turn encoder flag high
            m2angle_now += 1; // Decrement the angle
        }
    }
    else { // If the encoder is reading low, check
if it was high
        if (m2encon == 1) { // Check if the encoder was high
            m2encon = 0; // Turn the encoder flag low
            m2angle_now += 1; // Decrement the angle
        }
    }
    if(m2angle_now >= m2angle_end) { // Check if the desired angle
has been reached
        m2power = 0; // Set the power to zero
        motors_done += 1; // Increment the amount of motors
done
    }

    // Ramp power down when it reaches specified angles
    if (((m2angle_end-m2angle_now) ==
ramp_down_ticks)&&(ramp_down2_norm == 0)&&(ramp == 0)) {
        ramp_down2_norm = 1; // Set the state of the ramp for motor
        m2power = m2power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
    if (((m2angle_end-m2angle_now) ==
(ramp_down_ticks/2))&&(ramp_down2_norm == 1)) {
        ramp_down2_norm = 2; // Set the state of the ramp for motor
        m2power = m2power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
    if (((m2angle_end-m2angle_now) ==
(ramp_down_ticks/4))&&(ramp_down2_norm == 2)) {
        ramp_down2_norm = 0; // Clear the state of the ramp
        m2power = m2power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
}
// Motor 3 encoder
    if (m3angle_now < m3angle_end) { // Check encoder if still need
to increment angle
    // Read the encoder, if high then check if it was low
    if (digitalRead(m3encpin) == HIGH) {
        if (m3encon == 0) { // Check if the encoder was low
            m3encon = 1; // Turn encoder flag high
            m3angle_now += 1; // Decrement the angle
        }
    }
    else { // If the encoder is reading low, check
if it was high
        if (m3encon == 1) { // Check if the encoder was high
            m3encon = 0; // Turn the encoder flag low

```

```

        m3angle_now += 1;           // Decrement the angle
    }
}
if(m3angle_now >= m3angle_end) {    // Check if the desired angle
has been reached
    m3power = 0;                   // Set the power to zero
    motors_done += 1;              // Increment the amount of motors
done
}

// Ramp power down when it reaches specified angles
if (((m3angle_end-m3angle_now) ==
ramp_down_ticks)&&(ramp_down3_norm == 0)&&(ramp == 0)) {
    ramp_down3_norm = 1; // Set the state of the ramp for motor
    m3power = m3power*ramp_down_mult/ramp_down_div; // Decrease
motor power
}
if (((m3angle_end-m3angle_now) ==
(ramp_down_ticks/2))&&(ramp_down3_norm == 1)) {
    ramp_down3_norm = 2; // Set the state of the ramp for motor
    m3power = m3power*ramp_down_mult/ramp_down_div; // Decrease
motor power
}
if (((m3angle_end-m3angle_now) ==
(ramp_down_ticks/4))&&(ramp_down3_norm == 2)) {
    ramp_down3_norm = 0; // Clear the state of the ramp
    m3power = m3power*ramp_down_mult/ramp_down_div; // Decrease
motor power
}
m3progress_old = m3progress; // Set the old progress value
}
// Motor 4 encoder
if (m4angle_now < m4angle_end) { // Check encoder if still need
to increment angle
    // Read the encoder, if high then check if it was low
    if (digitalRead(m4encpin) == HIGH) {
        if (m4encon == 0) { // Check if the encoder was low
            m4encon = 1; // Turn encoder flag high
            m4angle_now += 1; // Decrement the angle
        }
    }
    else { // If the encoder is reading low, check
if it was high
        if (m4encon == 1) { // Check if the encoder was high
            m4encon = 0; // Turn the encoder flag low
            m4angle_now += 1; // Decrement the angle
        }
    }
    if(m4angle_now >= m4angle_end) { // Check if the desired angle
has been reached
        m4power = 0; // Set the power to zero
        motors_done += 1; // Increment the amount of motors
done
    }

// Ramp power down when it reaches specified angles

```

```

        if (((m4angle_end-m4angle_now) ==
ramp_down_ticks)&&(ramp_down4_norm == 0)&&(ramp == 0)) {
            ramp_down4_norm = 1; // Set the state of the ramp for motor
            m4power = m4power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
        if (((m4angle_end-m4angle_now) ==
(ramp_down_ticks/2))&&(ramp_down4_norm == 1)) {
            ramp_down4_norm = 2; // Set the state of the ramp for motor
            m4power = m4power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
        if (((m4angle_end-m4angle_now) ==
(ramp_down_ticks/4))&&(ramp_down4_norm == 2)) {
            ramp_down4_norm = 0; // Clear the state of the ramp
            m4power = m4power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
    }
    // Motor 5 encoder
    if (m5angle_now < m5angle_end) { // Check encoder if still need
to increment angle
        // Read the encoder, if high then check if it was low
        if (digitalRead(m5encpin) == HIGH) {
            if (m5encon == 0) { // Check if the encoder was low
                m5encon = 1; // Turn encoder flag high
                m5angle_now += 1; // Decrement the angle
            }
        }
        else { // If the encoder is reading low, check
if it was high
            if (m5encon == 1) { // Check if the encoder was high
                m5encon = 0; // Turn the encoder flag low
                m5angle_now += 1; // Decrement the angle
            }
        }
        if(m5angle_now >= m5angle_end) { // Check if the desired angle
has been reached
            m5power = 0; // Set the power to zero
            motors_done += 1; // Increment the amount of motors
done
        }

        // Ramp power down when it reaches specified angles
        if (((m5angle_end-m5angle_now) ==
ramp_down_ticks)&&(ramp_down5_norm == 0)&&(ramp == 0)) {
            ramp_down5_norm = 1; // Set the state of the ramp for motor
            m5power = m5power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
        if (((m5angle_end-m5angle_now) ==
(ramp_down_ticks/2))&&(ramp_down5_norm == 1)) {
            ramp_down5_norm = 2; // Set the state of the ramp for motor
            m5power = m5power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
    }
}

```



```

        if (((m5angle_end-m5angle_now) ==
(ramp_down_ticks/4))&&(ramp_down5_norm == 2)) {
            ramp_down5_norm = 0; // Clear the state of the ramp
            m5power = m5power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
    }
    // Motor 6 encoder
    if (m6angle_now < m6angle_end) { // Check encoder if still need
to increment angle
        // Read the encoder, if high then check if it was low
        if (digitalRead(m6encpin) == HIGH) {
            if (m6encon == 0) { // Check if the encoder was low
                m6encon = 1; // Turn encoder flag high
                m6angle_now += 1; // Decrement the angle
            }
        }
        else { // If the encoder is reading low, check
if it was high
            if (m6encon == 1) { // Check if the encoder was high
                m6encon = 0; // Turn the encoder flag low
                m6angle_now += 1; // Decrement the angle
            }
        }
        if(m6angle_now >= m6angle_end) { // Check if the desired angle
has been reached
            m6power = 0; // Set the power to zero
            motors_done += 1; // Increment the amount of motors
done
        }

        // Ramp power down when it reaches specified angles
        if (((m6angle_end-m6angle_now) ==
ramp_down_ticks)&&(ramp_down6_norm == 0)&&(ramp == 0)) {
            ramp_down6_norm = 1; // Set the state of the ramp for motor
            m6power = m6power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
        if (((m6angle_end-m6angle_now) ==
(ramp_down_ticks/2))&&(ramp_down6_norm == 1)) {
            ramp_down6_norm = 2; // Set the state of the ramp for motor
            m6power = m6power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
        if (((m6angle_end-m6angle_now) ==
(ramp_down_ticks/4))&&(ramp_down6_norm == 2)) {
            ramp_down6_norm = 0; // Clear the state of the ramp
            m6power = m6power*ramp_down_mult/ramp_down_div; // Decrease
motor power
        }
    }
    // Motor 7 encoder
    if (m7angle_now < m7angle_end) { // Check encoder if still need
to increment angle
        // Read the encoder, if high then check if it was low
        if (digitalRead(m7encpin) == HIGH) {
            if (m7encon == 0) { // Check if the encoder was low

```

```

        m7encon = 1;           // Turn encoder flag high
        m7angle_now += 1;      // Decrement the angle
    }
}
else {                         // If the encoder is reading low, check
if it was high
    if (m7encon == 1) {       // Check if the encoder was high
        m7encon = 0;         // Turn the encoder flag low
        m7angle_now += 1;     // Decrement the angle
    }
}
if(m7angle_now >= m7angle_end) { // Check if the desired angle
has been reached
    m7power = 0;              // Set the power to zero
    motors_done += 1;         // Increment the amount of motors
done
}

    // Ramp power down when it reaches specified angles
    if (((m7angle_end-m7angle_now) ==
ramp_down_ticks)&&(ramp_down7_norm == 0)&&(ramp == 0)) {
        ramp_down7_norm = 1; // Set the state of the ramp for motor
        m7power = m7power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
    if (((m7angle_end-m7angle_now) ==
(ramp_down_ticks/2))&&(ramp_down7_norm == 1)) {
        ramp_down7_norm = 2; // Set the state of the ramp for motor
        m7power = m7power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
    if (((m7angle_end-m7angle_now) ==
(ramp_down_ticks/4))&&(ramp_down7_norm == 2)) {
        ramp_down7_norm = 0; // Clear the state of the ramp
        m7power = m7power*ramp_down_mult/ramp_down_div; // Decrease
motor power
    }
}

    // If ramping up, increment the counter for timing buffer
    if (ramp > 0) {
        ramp_counter += 1;
    }

    // Check if ramp_counter has reached the value of ramp_timing while
still ramping up
    if ((ramp_counter >= ramp_timing)&&(ramp > 0)&&(ramp <=
ramp_divisions)) {
        ramp_counter = 0; // clear the counter
        ramp += 1; // increase the state of the ramp
        if (ramp > ramp_divisions) { // If done ramping up
            ramp = 0; // turn off ramp flag
        }
        switch(motor_max) { // Increment the lead motor's power if not at
max power
            case 1:

```

```

        if(m1power < ((pmax*12/psupply)-ramp_inc)) {
            m1power += ramp_inc; // Increase power to lead motor
        }
        break;
    case 2:
        if(m2power < ((pmax*12/psupply)-ramp_inc)) {
            m2power += ramp_inc; // Increase power to lead motor
        }
        break;
    case 3:
        if(m3power < ((pmax*12/psupply)-ramp_inc)) {
            m3power += ramp_inc; // Increase power to lead motor
        }
        break;
    case 4:
        if(m4power < ((pmax*12/psupply)-ramp_inc)) {
            m4power += ramp_inc; // Increase power to lead motor
        }
        break;
    case 5:
        if(m5power < ((pmax*12/psupply)-ramp_inc)) {
            m5power += ramp_inc; // Increase power to lead motor
        }
        break;
    case 6:
        if(m6power < ((pmax*12/psupply)-ramp_inc)) {
            m6power += ramp_inc; // Increase power to lead motor
        }
        break;
    default:
        if(m7power < ((pmax*12/psupply)-ramp_inc)) {
            m7power += ramp_inc; // Increase power to lead motor
        }
    }
}

// Increment the timing buffer to adjust motor speed every so often
timing += 1;
// Check if it is time to adjust motor speed
if(timing >= timing_buffer) {
    timing = 0;

    // Set angle progress of lead motor-----
    if(motor_max == 1) {
        angle_progress = m1angle_now*progress_scale/angle_max; // Set
        progress of lead motor
    }
    else if(motor_max == 2) {
        angle_progress = m2angle_now*progress_scale/angle_max; // Set
        progress of lead motor
    }
    else if(motor_max == 3) {
        angle_progress = m3angle_now*progress_scale/angle_max; // Set
        progress of lead motor
    }
    else if(motor_max == 4) {
        angle_progress = m4angle_now*progress_scale/angle_max; // Set
        progress of lead motor
    }
}

```

```

    }
    else if(motor_max == 5) {
        angle_progress = m5angle_now*progress_scale/angle_max;    // Set
progress of lead motor
    }
    else if(motor_max == 6) {
        angle_progress = m6angle_now*progress_scale/angle_max;    // Set
progress of lead motor
    }
    else {
        angle_progress = m7angle_now*progress_scale/angle_max;    // Set
progress of lead motor
    }

    // Adjust the motor speed based on angle progress-----
    // Motor 1
    if (m1angle_now < m1angle_end) {        // If motor is still
incrementing, check speed
        m1progress = m1angle_now*progress_scale/m1angle_end;    //
Calculate angle progress
        if(m1progress < angle_progress) {        // If motor is running
slow
            if(m1power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
                m1power += power_inc;                // increment the
motor power
            }
            else {
                slowdown = 1; // or else tell lead motor to slowdown
            }
        }
        else if(m1progress > angle_progress) { // If motor is running
fast
            if(m1power > power_inc) {                // and if power is not
off
                m1power -= power_inc;                // decrement the power
            }
            else {
                m1power = 0;                // or else turn power
off
            }
        }
        m1progress_old = m1progress;    // Set the old progress value
    }
    // Motor 2
    if (m2angle_now < m2angle_end) {        // If motor is still
incrementing, check speed
        m2progress = m2angle_now*progress_scale/m2angle_end;    //
Calculate angle progress
        if(m2progress < angle_progress) {        // If motor is running
slow
            if(m2power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
                m2power += power_inc;                // increment the
motor power
            }
            else {

```

```

        slowdown = 1; // or else tell lead motor to slowdown
    }
}
else if(m2progress > angle_progress) { // If motor is running
fast
    if(m2power > power_inc) { // and if power is not
off
        m2power -= power_inc; // decrement the power
    }
    else {
off
        m2power = 0; // or else turn power
    }
}
m2progress_old = m2progress; // Set the old progress value
}
// Motor 3
if (m3angle_now < m3angle_end) { // If motor is still
incrementing, check speed
    m3progress = m3angle_now*progress_scale/m3angle_end; //
Calculate angle progress
    if(m3progress < angle_progress) { // If motor is running
slow
        if(m3power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
            m3power += power_inc; // increment the
motor power
        }
        else {
            slowdown = 1; // or else tell lead motor to slowdown
        }
    }
    else if(m3progress > angle_progress) { // If motor is running
fast
        if(m3power > power_inc) { // and if power is not
off
            m3power -= power_inc; // decrement the power
        }
        else {
off
            m3power = 0; // or else turn power
        }
    }
}
m3progress_old = m3progress; // Set the old progress value
}
// Motor 4
if (m4angle_now < m4angle_end) { // If motor is still
incrementing, check speed
    m4progress = m4angle_now*progress_scale/m4angle_end; //
Calculate angle progress
    if(m4progress < angle_progress) { // If motor is running
slow
        if(m4power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
            m4power += power_inc; // increment the
motor power
        }
    }
}

```

```

        else {
            slowdown = 1; // or else tell lead motor to slowdown
        }
    }
    else if(m4progress > angle_progress) { // If motor is running
fast
        if(m4power > power_inc) { // and if power is not
off
            m4power -= power_inc; // decrement the power
        }
        else {
            m4power = 0; // or else turn power
off
        }
    }
    m4progress_old = m4progress; // Set the old progress value
}
// Motor 5
if (m5angle_now < m5angle_end) { // If motor is still
incrementing, check speed
    m5progress = m5angle_now*progress_scale/m5angle_end; //
Calculate angle progress
    if(m5progress < angle_progress) { // If motor is running
slow
        if(m5power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
            m5power += power_inc; // increment the
motor power
        }
        else {
            slowdown = 1; // or else tell lead motor to slowdown
        }
    }
    else if(m5progress > angle_progress) { // If motor is running
fast
        if(m5power > power_inc) { // and if power is not
off
            m5power -= power_inc; // decrement the power
        }
        else {
            m5power = 0; // or else turn power
off
        }
    }
    m5progress_old = m5progress; // Set the old progress value
}
// Motor 6
if (m6angle_now < m6angle_end) { // If motor is still
incrementing, check speed
    m6progress = m6angle_now*progress_scale/m6angle_end; //
Calculate angle progress
    if(m6progress < angle_progress) { // If motor is running
slow
        if(m6power < ((pmax*12/psupply)-power_inc)) { // If power is
not at maximum
            m6power += power_inc; // increment the
motor power

```

```

    }
    else {
        slowdown = 1;    // or else tell lead motor to slowdown
    }
}
else if(m6progress > angle_progress) {    // If motor is running
fast
    if(m6power > power_inc) {                // and if power is not
off
        m6power -= power_inc;                // decrement the power
    }
    else {
        m6power = 0;                        // or else turn power
off
    }
}
m6progress_old = m6progress;    // Set the old progress value
}
// Motor 7
if (m7angle_now < m7angle_end)    {        // If motor is still
incrementing, check speed
    m7progress = m7angle_now*progress_scale/m7angle_end;    //
Calculate angle progress
    if(m7progress < angle_progress)    {        // If motor is running
slow
        if(m7power < ((pmax*12/psupply)-power_inc))    {    // If power is
not at maximum
            m7power += power_inc;                // increment the
motor power
        }
        else {
            slowdown = 1;    // or else tell lead motor to slowdown
        }
    }
else if(m7progress > angle_progress)    {    // If motor is running
fast
    if(m7power > power_inc)    {                // and if power is not
off
        m7power -= power_inc;                // decrement the power
    }
    else {
        m7power = 0;                        // or else turn power
off
    }
}
m7progress_old = m7progress;    // Set the old progress value
}
// Slow down lead motor if running too fast
if(slowdown == 1)    {    // Check flag
    slowdown = 0;                // clear the flag for next iteration

    slow_counter += 1;                // increment the counter
    if (slow_counter >= slow_timing)    {    // Check if it is time to
slowdown
        slow_counter = 0;                // clear the counter

```

```

        switch(motor_max) { // Check which motor is lead
motor
        case 1:
            if(m1power > power_inc) { // Decrement motor if still
running
                m1power -= power_inc;
            }
            break;
        case 2:
            if(m2power > power_inc) { // Decrement motor if still
running
                m2power -= power_inc;
            }
            break;
        case 3:
            if(m3power > power_inc) { // Decrement motor if still
running
                m3power -= power_inc;
            }
            break;
        case 4:
            if(m4power > power_inc) { // Decrement motor if still
running
                m4power -= power_inc;
            }
            break;
        case 5:
            if(m5power > power_inc) { // Decrement motor if still
running
                m5power -= power_inc;
            }
            break;
        case 6:
            if(m6power > power_inc) { // Decrement motor if still
running
                m6power -= power_inc;
            }
            break;
        case 7:
            if(m7power > power_inc) { // Decrement motor if still
running
                m7power -= power_inc;
            }
            break;
        }
    }

    // If not told to slow down, make sure it is running fast
    else {
        switch(motor_max) { // Check which motor is lead
motor
        case 1:
            if(m1power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
                m1power += (power_inc/2); // Increment the motor power by
half to slowly

```



```

                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
    }
    break;
    case 2:
        if(m2power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m2power += (power_inc/2); // Increment the motor power by
half to slowly
                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
        }
        break;
    case 3:
        if(m3power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m3power += (power_inc/2); // Increment the motor power by
half to slowly
                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
        }
        break;
    case 4:
        if(m4power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m4power += (power_inc/2); // Increment the motor power by
half to slowly
                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
        }
        break;
    case 5:
        if(m5power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m5power += (power_inc/2); // Increment the motor power by
half to slowly
                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
        }
        break;
    case 6:
        if(m6power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m6power += (power_inc/2); // Increment the motor power by
half to slowly
                                                                    //    increase speed. This will
keep the motor                                                                    //    speed high
        }
        break;
    case 7:

```

```

        if(m7power < ((pmax*12/psupply)-power_inc)) { // If power
is not at maximum
            m7power += (power_inc/2); // Increment the motor power by
half to slowly
                                                    // increase speed. This will
keep the motor
                                                    // speed high
        }
        break;
    }
}
}
if(motors_done >= 7) { // If all of the motors are done, make
sure power is off

    motors_done = 0; // Clear the motors_done counter

    // Make sure all motors are turned off
    m1power = 0;
    m2power = 0;
    m3power = 0;
    m4power = 0;
    m5power = 0;
    m6power = 0;
    m7power = 0;

    // Clear old progress values
    m1progress_old = 0;
    m2progress_old = 0;
    m3progress_old = 0;
    m4progress_old = 0;
    m5progress_old = 0;
    m6progress_old = 0;
    m7progress_old = 0;

    reading = 0; // Stop the normalized motion

    normalized = 0; // Turn off normalized flag

    // Tell the processing code that the normalized incremenation is
done
    Serial.write(200);
}

analogWrite(m1onpin, m1power); // Set the pwm
analogWrite(m2onpin, m2power); // Set the pwm
analogWrite(m3onpin, m3power); // Set the pwm
analogWrite(m4onpin, m4power); // Set the pwm
analogWrite(m5onpin, m5power); // Set the pwm
analogWrite(m6onpin, m6power); // Set the pwm
analogWrite(m7onpin, m7power); // Set the pwm

}

// Individual Motor Incrementation-----
// Check encoder 1 for change of angle if necessary

```

```

if (mlangle > 0) {           // Check if angle is to be incremented
    // Read the encoder, if high then check if it was low
    if (digitalRead(mlencpin) == HIGH) {
        if (mlencon == 0) {   // Check if the encoder was low
            mlencon = 1;       // Turn encoder flag high
            mlangle -= 1;       // Decrement the angle
        }
    }
    else {                    // If the encoder is reading low, check
if it was high
        if (mlencon == 1) {   // Check if the encoder was high
            mlencon = 0;       // Turn the encoder flag low
            mlangle -= 1;       // Decrement the angle
        }
    }
    if(mlangle <= 1) {        // Check if the desired angle has been
reached
        mlangle = 0;          // Clear the angle
        ramp_up1 = 0;         // Stop ramping up
        analogWrite(mlonpin, 0); // Turn off the motor
        Serial.write(110);     // Tell the processing code that
the angle has
                                // been reached
    }
}

// Check encoder 2 for change of angle if necessary
if (m2angle > 0) {           // Check if angle is to be incremented
    // Read the encoder, if high then check if it was low
    if (digitalRead(m2encpin) == HIGH) {
        if (m2encon == 0) {   // Check if the encoder was low
            m2encon = 1;       // Turn encoder flag high
            m2angle -= 1;       // Decrement the angle
        }
    }
    else {                    // If the encoder is reading low, check
if it was high
        if (m2encon == 1) {   // Check if the encoder was high
            m2encon = 0;       // Turn the encoder flag low
            m2angle -= 1;       // Decrement the angle
        }
    }
    if(m2angle <= 1) {        // Check if the desired angle has been
reached
        m2angle = 0;          // Clear the angle
        ramp_up2 = 0;         // Stop ramping up
        analogWrite(m2onpin, 0); // Turn the motor off
        Serial.write(120);     // Tell the processing code that
the angle has
                                // been reached
    }
}

// Check encoder 3 for change of angle if necessary
if (m3angle > 0) {           // Check if angle is to be incremented
    // Read the encoder, if high then check if it was low
    if (digitalRead(m3encpin) == HIGH) {

```

```

        if (m3encon == 0) { // Check if the encoder was low
            m3encon = 1; // Turn encoder flag high
            m3angle -= 1; // Decrement the angle
        }
    }
    else { // If the encoder is reading low, check
if it was high
        if (m3encon == 1) { // Check if the encoder was high
            m3encon = 0; // Turn the encoder flag low
            m3angle -= 1; // Decrement the angle
        }
    }
    if(m3angle <= 1) { // Check if the desired angle has been
reached
        m3angle = 0; // Clear the angle
        ramp_up3 = 0; // Stop ramping up
        analogWrite(m3onpin, 0); // Turn the motor off
        Serial.write(130); // Tell the processing code that
the angle has
                                // been reached
    }
}

// Check encoder 4 for change of angle if necessary
if (m4angle > 0) { // Check if angle is to be incremented
    // Read the encoder, if high then check if it was low
    if (digitalRead(m4encpin) == HIGH) {
        if (m4encon == 0) { // Check if the encoder was low
            m4encon = 1; // Turn encoder flag high
            m4angle -= 1; // Decrement the angle
        }
    }
    else { // If the encoder is reading low, check
if it was high
        if (m4encon == 1) { // Check if the encoder was high
            m4encon = 0; // Turn the encoder flag low
            m4angle -= 1; // Decrement the angle
        }
    }
    if(m4angle <= 1) { // Check if the desired angle has been
reached
        m4angle = 0; // Clear the angle
        ramp_up4 = 0; // Stop ramping up
        analogWrite(m4onpin, 0); // Turn the motor off
        Serial.write(140); // Tell the processing code that
the angle has
                                // been reached
    }
}

// Check encoder 5 for change of angle if necessary
if (m5angle > 0) { // Check if angle is to be incremented
    // Read the encoder, if high then check if it was low
    if (digitalRead(m5encpin) == HIGH) {
        if (m5encon == 0) { // Check if the encoder was low
            m5encon = 1; // Turn encoder flag high
            m5angle -= 1; // Decrement the angle

```

```

    }
  }
  else { // If the encoder is reading low, check
if it was high
    if (m5encon == 1) { // Check if the encoder was high
      m5encon = 0; // Turn the encoder flag low
      m5angle -= 1; // Decrement the angle
    }
  }
  if(m5angle <= 1) { // Check if the desired angle has been
reached
    m5angle = 0; // Clear the angle
    ramp_up5 = 0; // Stop ramping up
    analogWrite(m5onpin, 0); // Turn the motor off
    Serial.write(150); // Tell the processing code that
the angle has
// been reached
  }
}

// Check encoder 6 for change of angle if necessary
if (m6angle > 0) { // Check if angle is to be incremented
  // Read the encoder, if high then check if it was low
  if (digitalRead(m6encpin) == HIGH) {
    if (m6encon == 0) { // Check if the encoder was low
      m6encon = 1; // Turn encoder flag high
      m6angle -= 1; // Decrement the angle
    }
  }
  else { // If the encoder is reading low, check
if it was high
    if (m6encon == 1) { // Check if the encoder was high
      m6encon = 0; // Turn the encoder flag low
      m6angle -= 1; // Decrement the angle
    }
  }
  if(m6angle <= 1) { // Check if the desired angle has been
reached
    m6angle = 0; // Clear the angle
    ramp_up6 = 0; // Stop ramping up
    analogWrite(m6onpin, 0); // Turn the motor off
    Serial.write(160); // Tell the processing code that
the angle has
// been reached
  }
}

// Check encoder 7 for change of angle if necessary
if (m7angle > 0) { // Check if angle is to be incremented
  // Read the encoder, if high then check if it was low
  if (digitalRead(m7encpin) == HIGH) {
    if (m7encon == 0) { // Check if the encoder was low
      m7encon = 1; // Turn encoder flag high
      m7angle -= 1; // Decrement the angle
    }
  }
}

```

```

        else { // If the encoder is reading low, check
if it was high
            if (m7encon == 1) { // Check if the encoder was high
                m7encon = 0; // Turn the encoder flag low
                m7angle -= 1; // Decrement the angle
            }
        }
        if(m7angle <= 1) { // Check if the desired angle has been
reached
            m7angle = 0; // Clear the angle
            ramp_up7 = 0; // Stop ramping up
            analogWrite(m7onpin, 0); // Turn the motor off
            Serial.write(170); // Tell the processing code that
the angle has
                                // been reached
        }
    }
    //-----Ramp Down-----
    // Motor 1
    if (ramp_down1 > 0) { // Check if turned on
        ramp_down_time1 += 1; // increment counter
        if (ramp_down_time1 >= ramp_down_time) { // Check if time to ramp
down
            ramp_down_time1 = 0; // clear the counter
            if (ramp_down1 == 1) { // check state of ramp
down
                m1power = m1power/2; // decrease power
                ramp_down1 = 2; // set next state
            }
            else if (ramp_down1 == 2) { // check state of ramp
down
                m1power = m1power/2; // decrease power
                ramp_down1 = 3; // set next state
            }
            else if (ramp_down1 == 3) { // check state of ramp
down
                m1power = 0; // turn motor off
                ramp_down1 = 0; // turn off ramp down
            }
            analogWrite(m1onpin, m1power); // set pwm
        }
    }

    // Motor 2
    if (ramp_down2 > 0) { // Check if turned on
        ramp_down_time2 += 1; // increment counter
        if (ramp_down_time2 >= ramp_down_time) { // Check if time to ramp
down
            ramp_down_time2 = 0; // clear the counter
            if (ramp_down2 == 1) { // check state of ramp
down
                m2power = m2power/2; // decrease power
                ramp_down2 = 2; // set next state
            }
            else if (ramp_down2 == 2) { // check state of ramp
down
                m2power = m2power/2; // decrease power

```

```

        ramp_down2 = 3;                                // set next state
    }
    else if (ramp_down2 == 3) {                          // check state of ramp
down        m2power = 0;                                // turn motor off
            ramp_down2 = 0;                            // turn off ramp down
        }
        analogWrite(m2onpin, m2power);                // set pwm
    }
}

// Motor 3
if (ramp_down3 > 0) {                                  // Check if turned on
    ramp_down_time3 += 1;                             // increment counter
    if (ramp_down_time3 >= ramp_down_time) {           // Check if time to ramp
down        ramp_down_time3 = 0;                       // clear the counter
            if (ramp_down3 == 1) {                     // check state of ramp
down                m3power = m3power/2;              // decrease power
                    ramp_down3 = 2;                   // set next state
            }
            else if (ramp_down3 == 2) {                 // check state of ramp
down                m3power = m3power/2;              // decrease power
                    ramp_down3 = 3;                   // set next state
            }
            else if (ramp_down3 == 3) {                 // check state of ramp
down                m3power = 0;                      // turn motor off
                    ramp_down3 = 0;                  // turn off ramp down
            }
            analogWrite(m3onpin, m3power);              // set pwm
        }
    }

// Motor 4
if (ramp_down4 > 0) {                                  // Check if turned on
    ramp_down_time4 += 1;                             // increment counter
    if (ramp_down_time4 >= ramp_down_time) {           // Check if time to ramp
down        ramp_down_time4 = 0;                       // clear the counter
            if (ramp_down4 == 1) {                     // check state of ramp
down                m4power = m4power/2;              // decrease power
                    ramp_down4 = 2;                   // set next state
            }
            else if (ramp_down4 == 2) {                 // check state of ramp
down                m4power = m4power/2;              // decrease power
                    ramp_down4 = 3;                   // set next state
            }
            else if (ramp_down4 == 3) {                 // check state of ramp
down                m4power = 0;                      // turn motor off
                    ramp_down4 = 0;                  // turn off ramp down
            }
        }
    }
}

```

```

        analogWrite(m4onpin, m4power);          // set pwm
    }
}

// Motor 5
if (ramp_down5 > 0) {                          // Check if turned on
    ramp_down_time5 += 1;                      // increment counter
    if (ramp_down_time5 >= ramp_down_time) {   // Check if time to ramp
down
        ramp_down_time5 = 0;                  // clear the counter
        if (ramp_down5 == 1) {                // check state of ramp
down
            m5power = m5power/2;              // decrease power
            ramp_down5 = 2;                   // set next state
        }
        else if (ramp_down5 == 2) {           // check state of ramp
down
            m5power = m5power/2;              // decrease power
            ramp_down5 = 3;                   // set next state
        }
        else if (ramp_down5 == 3) {           // check state of ramp
down
            m5power = 0;                      // turn motor off
            ramp_down5 = 0;                   // turn off ramp down
        }
        analogWrite(m5onpin, m5power);        // set pwm
    }
}

// Motor 6
if (ramp_down6 > 0) {                          // Check if turned on
    ramp_down_time6 += 1;                      // increment counter
    if (ramp_down_time6 >= ramp_down_time) {   // Check if time to ramp
down
        ramp_down_time6 = 0;                  // clear the counter
        if (ramp_down6 == 1) {                // check state of ramp
down
            m6power = m6power/2;              // decrease power
            ramp_down6 = 2;                   // set next state
        }
        else if (ramp_down6 == 2) {           // check state of ramp
down
            m6power = m6power/2;              // decrease power
            ramp_down6 = 3;                   // set next state
        }
        else if (ramp_down6 == 3) {           // check state of ramp
down
            m6power = 0;                      // turn motor off
            ramp_down6 = 0;                   // turn off ramp down
        }
        analogWrite(m6onpin, m6power);        // set pwm
    }
}

// Motor 7
if (ramp_down7 > 0) {                          // Check if turned on
    ramp_down_time7 += 1;                      // increment counter

```



```

    if (ramp_down_time7 >= ramp_down_time) { // Check if time to ramp
down
        ramp_down_time7 = 0; // clear the counter
        if (ramp_down7 == 1) { // check state of ramp
down
            m7power = m7power/2; // decrease power
            ramp_down7 = 2; // set next state
        }
        else if (ramp_down7 == 2) { // check state of ramp
down
            m7power = m7power/2; // decrease power
            ramp_down7 = 3; // set next state
        }
        else if (ramp_down7 == 3) { // check state of ramp
down
            m7power = 0; // turn motor off
            ramp_down7 = 0; // turn off ramp down
        }
        analogWrite(m7onpin, m7power); // set pwm
    }
}
//-----Ramp Up-----
if (ramp_up1 > 0) { // Check if ramp up is
turned on
    ramp_up_time1 += 1; // Increment the
counter
    if(ramp_up_time1 >= ramp_up_time) { // Check if time to
ramp up
        ramp_up_time1 = 0; // Clear the counter
        m1power = m1power*(ramp_up1+1)/ramp_up1; // Increase motor power
        ramp_up1 += 1; // Increment ramp state
        if (ramp_up1 >= ramp_divisions) { // Check if done
ramping up
            ramp_up1 = 0; // Turn off ramp up for
motor
        }
        analogWrite(m1onpin, m1power); // Set pwm
    }
}

if (ramp_up2 > 0) { // Check if ramp up is
turned on
    ramp_up_time2 += 1; // Increment the
counter
    if(ramp_up_time2 >= ramp_up_time) { // Check if time to
ramp up
        ramp_up_time2 = 0; // Clear the counter
        m2power = m2power*(ramp_up2+1)/ramp_up2; // Increase motor power
        ramp_up2 += 1; // Increment ramp state
        if (ramp_up2 >= ramp_divisions) { // Check if done
ramping up
            ramp_up2 = 0; // Turn off ramp up for
motor
        }
        analogWrite(m2onpin, m2power); // Set pwm
    }
}
}

```

```

    if (ramp_up3 > 0) { // Check if ramp up is
turned on
        ramp_up_time3 += 1; // Increment the
counter
        if(ramp_up_time3 >= ramp_up_time) { // Check if time to
ramp up
            ramp_up_time3 = 0; // Clear the counter
            m3power = m3power*(ramp_up3+1)/ramp_up3; // Increase motor power
            ramp_up3 += 1; // Increment ramp state
            if (ramp_up3 >= ramp_divisions) { // Check if done
ramping up
                ramp_up3 = 0; // Turn off ramp up for
motor
            }
            analogWrite(m3onpin, m3power); // Set pwm
        }
    }

    if (ramp_up4 > 0) { // Check if ramp up is
turned on
        ramp_up_time4 += 1; // Increment the
counter
        if(ramp_up_time4 >= ramp_up_time) { // Check if time to
ramp up
            ramp_up_time4 = 0; // Clear the counter
            m4power = m4power*(ramp_up4+1)/ramp_up4; // Increase motor power
            ramp_up4 += 1; // Increment ramp state
            if (ramp_up4 >= ramp_divisions) { // Check if done
ramping up
                ramp_up4 = 0; // Turn off ramp up for
motor
            }
            analogWrite(m4onpin, m4power); // Set pwm
        }
    }

    if (ramp_up5 > 0) { // Check if ramp up is
turned on
        ramp_up_time5 += 1; // Increment the
counter
        if(ramp_up_time5 >= ramp_up_time) { // Check if time to
ramp up
            ramp_up_time5 = 0; // Clear the counter
            m5power = m5power*(ramp_up5+1)/ramp_up5; // Increase motor power
            ramp_up5 += 1; // Increment ramp state
            if (ramp_up5 >= ramp_divisions) { // Check if done
ramping up
                ramp_up5 = 0; // Turn off ramp up for
motor
            }
            analogWrite(m5onpin, m5power); // Set pwm
        }
    }

    if (ramp_up6 > 0) { // Check if ramp up is
turned on

```

```

        ramp_up_time6 += 1; // Increment the
counter
        if(ramp_up_time6 >= ramp_up_time) { // Check if time to
ramp up
            ramp_up_time6 = 0; // Clear the counter
            m6power = m6power*(ramp_up6+1)/ramp_up6; // Increase motor power
            ramp_up6 += 1; // Increment ramp state
            if (ramp_up6 >= ramp_divisions) { // Check if done
ramping up
                ramp_up6 = 0; // Turn off ramp up for
motor
            }
            analogWrite(m6onpin, m6power); // Set pwm
        }
    }

    if (ramp_up7 > 0) { // Check if ramp up is
turned on
        ramp_up_time7 += 1; // Increment the
counter
        if(ramp_up_time7 >= ramp_up_time) { // Check if time to
ramp up
            ramp_up_time7 = 0; // Clear the counter
            m7power = m7power*(ramp_up7+1)/ramp_up7; // Increase motor power
            ramp_up7 += 1; // Increment ramp state
            if (ramp_up7 >= ramp_divisions) { // Check if done
ramping up
                ramp_up7 = 0; // Turn off ramp up for
motor
            }
            analogWrite(m7onpin, m7power); // Set pwm
        }
    }

    BAILOUT:
    { //do nothing here
    }

}

```

Processing Code

```
/*
 * Paul Bessent
 * Created: 2/10/11
 * Last updated: 5/12/11
 *
 * Version 2.4 for the SCMR
 * Run this code after the corresponding Arduino code
 *
 * This is the GUI and sends messages to the MCU to actuate the motors.
 * For individual motor motion, the power must be set individually.
 * For normalized motion, only the max power can be set to a high,
 * medium, or low setting.
 * The keyboard functions are:
 * Spacebar = emergency stop of all motors
 * 1-7 = toggle that motor # on/off
 * Shift+(1-7) = toggle that motor #'s direction
 */

// Setup communication with Arduino board over the serial
import processing.serial.*;
Serial arduino;

// Create color variable for buttons
color currentcolor;

// Initialize buttons for GUI

// Motor on/off buttons
RectButton rectm1on, rectm1off, rectm2on, rectm2off, rectm3on,
rectm3off;
RectButton rectm4on, rectm4off, rectm5on, rectm5off, rectm6on,
rectm6off;
RectButton rectm7on, rectm7off;

// Motor up buttons
RectButton rectm1up, rectm2up, rectm3up, rectm4up, rectm5up, rectm6up;
RectButton rectm7up;

// Motor down buttons
RectButton rectm1down, rectm2down, rectm3down, rectm4down, rectm5down;
RectButton rectm6down, rectm7down;

// Motor individual increment buttons
RectButton rectm1angle, rectm2angle, rectm3angle, rectm4angle,
rectm5angle;
RectButton rectm6angle, rectm7angle;

// Gripper open/off/close buttons
RectButton rectg1open, rectg2open, rectg1off, rectg2off;
RectButton rectg1close, rectg2close;

// Emergency stop and normalized motion button
RectButton rectstop, rectnormalize;
```

```

// Preset motion buttons
RectButton rectneutstart;
RectButton rectpoint_1_1a, rectpoint_1a_1b, rectpoint_1b_1c;
RectButton      rectpoint_1c_2,      rectpoint_2_2a,      rectpoint_2a_2b,
rectpoint_2b_2c;
RectButton      rectpoint_2c_3,      rectpoint_3_3a,      rectpoint_3a_3b,
rectpoint_3b_3c;
RectButton rectpoint_3c_3d, rectpoint_3d_4;

// Max power setting button
RectButton rectpmaxhigh, rectpmaxmed, rectpmaxlow;

// Flag for when mouse button is pressed while over a button
boolean locked = false;

// Font for text in GUI
PFont fontA;

//Classes for scrolling bars
Handle[] mpower;
Handle[] mangle;

char keyboard; // Character pressed on the keyboard
int click = 0; // Flag indicating the mouse button is
clicked
// and not yet released
int motorx = 20; // x position to print motor text
int mpowerx = 80; // x position to start the handle for motor
power
int scrollx = mpowerx+115; // x position to print motor power value
int onx = scrollx+30; // x position for on button
int offx = onx+40; // x position for off button
int upx = offx+40; // x position for up button
int downx = upx+40; // x position for down button
int dark = 51; // color for when button is pressed
int light = 200; // color for when button is not pressed
int num = 7; // number of motors
int manglex = downx+40; // x position for angle handle
int scrollx2 = manglex+115; // x position to print angle change value
int anglex = scrollx2+30; // x position for angle button
int incoming; // Value received from Arduino code over
the
// Serial
int normalized = 0; // Normalized flag off
int pt_but = 135; // x-dir spacing for Point Buttons on GUI
int pointx = anglex+150; // x position of Point Buttons

// Setup-----
void setup(){
  size(580+pt_but, 190+(num-1)*40); // Size of the display box
  smooth(); // Smooths curved edges

  arduino = new Serial(this, "COM3", 9600); // Setup the Serial
communication
  arduino.buffer(1);

```

```

color baseColor = color(200);           // Set color
currentcolor = baseColor;

mpower = new Handle[num];              // Create object for the motor
power                                     // scrolling bars
int hsize = 10;                         // Box size for the scrolling bar
for(int i=0; i<num; i++) {              // Initialize the motor power
object
    // set values ( x position, y position, length, size, motor #, on
flag,
    // power, state, direction, angle, Handle[])
    mpower[i] = new Handle(mpowerx, 30+i*40, 0, 10, i+1, 0, 0, 0, 0, 0,
mpower);
}

// Repeat object creation for motor angle
mangle = new Handle[num];
hsize = 10;
for(int i=0; i<num; i++) {
    mangle[i] = new Handle(manglex, 30+i*40, 0, 10, i+1, 0, 0, 1, 0, 0,
mangle);
}

// Load the font. Fonts must be placed within the data
// directory of your sketch. Use Tools > Create Font
// to create a distributable bitmap font.
// For vector fonts, use the createFont() function.
fontA = loadFont("TimesNewRomanPSMT-12.vlw");

// Set the font and its size (in units of pixels)
textFont(fontA, 12);

// Define and create rectangle button: Motor on
color buttoncolor = color(light);
color highlight = color(0);
// Set values (x position, y position, size, button color, highlight
color)
rectm1on = new RectButton(onx, 20, 20, buttoncolor, highlight);
rectm2on = new RectButton(onx, 20+(2-1)*40, 20, buttoncolor,
highlight);
rectm3on = new RectButton(onx, 20+(3-1)*40, 20, buttoncolor,
highlight);
rectm4on = new RectButton(onx, 20+(4-1)*40, 20, buttoncolor,
highlight);
rectm5on = new RectButton(onx, 20+(5-1)*40, 20, buttoncolor,
highlight);
rectm6on = new RectButton(onx, 20+(6-1)*40, 20, buttoncolor,
highlight);
rectm7on = new RectButton(onx, 20+(7-1)*40, 20, buttoncolor,
highlight);

// Define and create rectangle button: Motor off
buttoncolor = color(dark);
highlight = color(0);

```

```

    // Set values (x position, y position, size, button color, highlight
    color)
    rectm1off = new RectButton(offx, 20, 20, buttoncolor, highlight);
    rectm2off = new RectButton(offx, 20+(2-1)*40, 20, buttoncolor,
    highlight);
    rectm3off = new RectButton(offx, 20+(3-1)*40, 20, buttoncolor,
    highlight);
    rectm4off = new RectButton(offx, 20+(4-1)*40, 20, buttoncolor,
    highlight);
    rectm5off = new RectButton(offx, 20+(5-1)*40, 20, buttoncolor,
    highlight);
    rectm6off = new RectButton(offx, 20+(6-1)*40, 20, buttoncolor,
    highlight);
    rectm7off = new RectButton(offx, 20+(7-1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Direction up
    buttoncolor = color(light);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectm1up = new RectButton(upx, 20, 20, buttoncolor, highlight);
    rectm2up = new RectButton(upx, 20+(2-1)*40, 20, buttoncolor,
    highlight);
    rectm3up = new RectButton(upx, 20+(3-1)*40, 20, buttoncolor,
    highlight);
    rectm4up = new RectButton(upx, 20+(4-1)*40, 20, buttoncolor,
    highlight);
    rectm5up = new RectButton(upx, 20+(5-1)*40, 20, buttoncolor,
    highlight);
    rectm6up = new RectButton(upx, 20+(6-1)*40, 20, buttoncolor,
    highlight);
    rectm7up = new RectButton(upx, 20+(7-1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Direction down
    buttoncolor = color(dark);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectm1down = new RectButton(downx, 20, 20, buttoncolor, highlight);
    rectm2down = new RectButton(downx, 20+(2-1)*40, 20, buttoncolor,
    highlight);
    rectm3down = new RectButton(downx, 20+(3-1)*40, 20, buttoncolor,
    highlight);
    rectm4down = new RectButton(downx, 20+(4-1)*40, 20, buttoncolor,
    highlight);
    rectm5down = new RectButton(downx, 20+(5-1)*40, 20, buttoncolor,
    highlight);
    rectm6down = new RectButton(downx, 20+(6-1)*40, 20, buttoncolor,
    highlight);
    rectm7down = new RectButton(downx, 20+(7-1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Execute angle change
    buttoncolor = color(light);
    highlight = color(0);

```

```

    // Set values (x position, y position, size, button color, highlight
    color)
    rectm1angle = new RectButton(anglex, 20, 20, buttoncolor, highlight);
    rectm2angle = new RectButton(anglex, 20+(2-1)*40, 20, buttoncolor,
    highlight);
    rectm3angle = new RectButton(anglex, 20+(3-1)*40, 20, buttoncolor,
    highlight);
    rectm4angle = new RectButton(anglex, 20+(4-1)*40, 20, buttoncolor,
    highlight);
    rectm5angle = new RectButton(anglex, 20+(5-1)*40, 20, buttoncolor,
    highlight);
    rectm6angle = new RectButton(anglex, 20+(6-1)*40, 20, buttoncolor,
    highlight);
    rectm7angle = new RectButton(anglex, 20+(7-1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Gripper Open
    buttoncolor = color(light);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectg1open = new RectButton(onx, 40+(num)*40, 20, buttoncolor,
    highlight);
    rectg2open = new RectButton(onx, 40+(num+1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Gripper Off
    buttoncolor = color(dark);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectg1off = new RectButton(offx, 40+(num)*40, 20, buttoncolor,
    highlight);
    rectg2off = new RectButton(offx, 40+(num+1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Gripper Close
    buttoncolor = color(light);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectg1close = new RectButton(upx, 40+(num)*40, 20, buttoncolor,
    highlight);
    rectg2close = new RectButton(upx, 40+(num+1)*40, 20, buttoncolor,
    highlight);

    // Define and create rectangle button: Stop all motors
    buttoncolor = color(150);
    highlight = color(0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectstop = new RectButton(mpowerx-40, 40+(num)*40, 60, buttoncolor,
    highlight);

    // Define and create rectangle button: Increment all angles,
    normalized
    buttoncolor = color(light);

```



```

    highlight = color(0);
    color redhighlight = color(255, 0, 0);
    // Set values (x position, y position, size, button color, highlight
    color)
    rectnormalize = new RectButton(anglex, 20+(num)*40, 20, buttoncolor,
    highlight);
    // Set values (x pos, y pos, size, button color, highlight color):
    Point Buttons
    rectneutstart = new RectButton(pointx+5, 20, 15, buttoncolor,
    highlight);
    rectpoint_1_1a = new RectButton(pointx+5, 33+22, 15, buttoncolor,
    highlight);
    rectpoint_1a_1b = new RectButton(pointx+5, 33+2*22, 15, buttoncolor,
    highlight);
    rectpoint_1b_1c = new RectButton(pointx+5, 33+3*22, 15, buttoncolor,
    highlight);
    rectpoint_1c_2 = new RectButton(pointx+5, 33+4*22, 15, buttoncolor,
    redhighlight);
    rectpoint_2_2a = new RectButton(pointx+5, 46+5*22, 15, buttoncolor,
    highlight);
    rectpoint_2a_2b = new RectButton(pointx+5, 46+6*22, 15, buttoncolor,
    highlight);
    rectpoint_2b_2c = new RectButton(pointx+5, 46+7*22, 15, buttoncolor,
    highlight);
    rectpoint_2c_3 = new RectButton(pointx+5, 46+8*22, 15, buttoncolor,
    redhighlight);
    rectpoint_3_3a = new RectButton(pointx+5, 59+9*22, 15, buttoncolor,
    highlight);
    rectpoint_3a_3b = new RectButton(pointx+5, 59+10*22, 15, buttoncolor,
    highlight);
    rectpoint_3b_3c = new RectButton(pointx+5, 59+11*22, 15, buttoncolor,
    highlight);
    rectpoint_3c_3d = new RectButton(pointx+5, 59+12*22, 15, buttoncolor,
    redhighlight);
    rectpoint_3d_4 = new RectButton(pointx+5, 59+13*22, 15, buttoncolor,
    redhighlight);

    // Set values (x pos, y pos, size, button color, highlight color):
    Power Max
    buttoncolor = color(light);
    rectpmaxhigh = new RectButton(pointx, height-30, 20, buttoncolor,
    highlight);
    rectpmaxmed = new RectButton(pointx-40, height-30, 20, buttoncolor,
    highlight);
    buttoncolor = color(dark);
    rectpmaxlow = new RectButton(pointx-80, height-30, 20, buttoncolor,
    highlight);

    println("Program Start"); // Print in text area below
}

void draw()
{
    background(currentcolor); // Refresh the screen to not have text
    write over itself
    stroke(0); // Change the color of lines to be black

```

```

// Create lines to create separation of the buttons
line(onx-10, 0, onx-10, 50+(num-1)*40); // left of the on button
line(upx-10, 0, upx-10, 50+(num-1)*40); // left of the up button
line(downx+30, 0, downx+30, 50+(num-1)*40); // Right of the down
button
line(0, 50+(num-1)*40, width-pt_but, 50+(num-1)*40); // Below the
scrolling bars
line(onx-80, 50+(num-1)*40, onx-80, height); // Left of the gripper
buttons
line(downx-10, 50+(num-1)*40, downx-10, height); // Right of the
gripper buttons
line(downx-10, 50+(num)*40, pointx-100, 50+(num)*40); // Below
Normalized Increment button
line(pointx-100, 86+(num)*40, width, 86+(num)*40); // Above power
setting buttons
line(width-pt_but, 0, width-pt_but, height); //Right of version 2.3
buttons

// Write text for the GUI ("Text written", x position, y position)
for(int i=1; i<num+1; i++) {
    text("Motor ", motorx, 35+(i-1)*40);
    text(i, motorx+40, 35+(i-1)*40);
}
text("On", onx+3, 15);
text("Off", offx+2, 15);
text("Up", upx+3, 15);
text("Down",downx-2,15);
text("Power", mpowerx+45, 15);
text("Angle Change (deg)", manglex+15, 15);
text("Increment", anglex-13, 15);
text("Point to Point Motion", width-pt_but+15, 15);
text("STOP EVERYTHING!!!", motorx-10, 70+(num-1)*40);
text("Open", onx-4, 70+(num-1)*40);
text("Off", offx+2, 70+(num-1)*40);
text("Close", upx-2, 70+(num-1)*40);
text("Gripper 1", onx-60, 95+(num-1)*40);
text("Gripper 2", onx-60, 95+(num)*40);
text("Normalized Increment of All Angles", anglex-180, 35+(num)*40);
text("Type # to toggle that motor on/off", anglex-180, 65+(num)*40);
text("Hold shift and type # to toggle that", anglex-180,
85+(num)*40);
text("motor up/down", anglex-160, 105+(num)*40);
text("Press the spacebar to stop everything", anglex-180,
125+(num)*40);
line(pointx-100, 32-14, width, 32-14);
text("Neutral to start", pointx-85, 32);
line(pointx-100, 32+7, width, 32+7);
line(pointx-100, 32+20, width, 32+20);
text("Move pt 1 to 1a", pointx-85, 45+22);
text("Move pt 1a to 1b", pointx-85, 45+22*2);
text("Move pt 1b to 1c", pointx-85, 45+22*3);
line(pointx-100, 45+22*4+7, width, 45+22*4+7);
line(pointx-100, 58+22*4+7, width, 58+22*4+7);
text("Move pt 2 to 2a", pointx-85, 58+22*5);
text("Move pt 2a to 2b", pointx-85, 58+22*6);
text("Move pt 2b to 2c", pointx-85, 58+22*7);
line(pointx-100, 58+22*8+7, width, 58+22*8+7);

```

```

line(pointx-100, 71+22*8+7, width, 71+22*8+7);
text("Move pt 3 to 3a", pointx-85, 71+22*9);
text("Move pt 3a to 3b", pointx-85, 71+22*10);
text("Move pt 3b to 3c", pointx-85, 71+22*11);
text("Max Power Setting", pointx-77, height-50);
text("Low      Med      High", pointx-80, height-35);
fill(255,0,0); // Change text
color to red
text("*Move pt 1c to 2*", pointx-91, 45+22*4);
text("*Move pt 2c to 3*", pointx-91, 58+22*8);
text("*Move pt 3c to 3d*", pointx-91, 71+22*12);
text("*Move pt 3d to 4*", pointx-91, 71+22*13);
fill(0, 0, 255); // Change text
color to blue
text("Grip", pointx-45, 32+18);
text("Grip", pointx-45, 45+22*4+18);
text("Grip", pointx-45, 58+22*8+18);
fill(0); // Change text
color to black

// Update mouse position
update(mouseX, mouseY);

//Display all of the buttons
rectm1on.display();
rectm1off.display();
rectm2on.display();
rectm2off.display();
rectm3on.display();
rectm3off.display();
rectm4on.display();
rectm4off.display();
rectm5on.display();
rectm5off.display();
rectm6on.display();
rectm6off.display();
rectm7on.display();
rectm7off.display();

rectm1up.display();
rectm2up.display();
rectm3up.display();
rectm4up.display();
rectm5up.display();
rectm6up.display();
rectm7up.display();

rectm1down.display();
rectm2down.display();
rectm3down.display();
rectm4down.display();
rectm5down.display();
rectm6down.display();
rectm7down.display();

rectm1angle.display();
rectm2angle.display();

```

```

rectm3angle.display();
rectm4angle.display();
rectm5angle.display();
rectm6angle.display();
rectm7angle.display();

rectneutstart.display();
rectpoint_1_1a.display();
rectpoint_1a_1b.display();
rectpoint_1b_1c.display();
rectpoint_1c_2.display();
rectpoint_2_2a.display();
rectpoint_2a_2b.display();
rectpoint_2b_2c.display();
rectpoint_2c_3.display();
rectpoint_3_3a.display();
rectpoint_3a_3b.display();
rectpoint_3b_3c.display();
rectpoint_3c_3d.display();
rectpoint_3d_4.display();

rectstop.display();
rectnormalize.display();
rectpmaxhigh.display();
rectpmaxmed.display();
rectpmaxlow.display();

rectg1open.display();
rectg2open.display();
rectg1off.display();
rectg2off.display();
rectg1close.display();
rectg2close.display();

// Update all of the scrolling bars
for(int i=0; i<num; i++) {
    mpower[i].update();
    mpower[i].display();
    mangle[i].update();
    mangle[i].display();
}

fill(0); // not sure if this is necessary

incoff(); // Check if incremented motors are done
}

void incoff() // Check if incremented motors are done
{
    if(arduino.available() > 0) { // Check if anything has been sent
over the serial
        incoming = arduino.read(); // Read the Serial
        switch (incoming) { // Use switch-case to execute correct
code
            case 110: // Motor 1 done
                mpower[1-1].on=0; // Turn the on flag off

```

```

        arduino.write(10);                // Make sure the motor is
off    rectm1on.basecolor = color(light);    // Turn the on button
light  rectm1off.basecolor = color(dark);    // Turn the off button
dark   rectm1angle.basecolor = color(light); // Turn the angle button
light  break;

    case 120:                // Motor 2 done
        mpower[2-1].on=0;    // Turn the on flag off
        arduino.write(20);    // Make sure the motor is
off    rectm2on.basecolor = color(light);    // Turn the on button
light  rectm2off.basecolor = color(dark);    // Turn the off button
dark   rectm2angle.basecolor = color(light); // Turn the angle button
light  break;

    case 130:                // Motor 3 done
        mpower[3-1].on=0;    // Turn the on flag off
        arduino.write(30);    // Make sure the motor is
off    rectm3on.basecolor = color(light);    // Turn the on button
light  rectm3off.basecolor = color(dark);    // Turn the off button
dark   rectm3angle.basecolor = color(light); // Turn the angle button
light  break;

    case 140:                // Motor 4 done
        mpower[4-1].on=0;    // Turn the on flag off
        arduino.write(40);    // Make sure the motor is
off    rectm4on.basecolor = color(light);    // Turn the on button
light  rectm4off.basecolor = color(dark);    // Turn the off button
dark   rectm4angle.basecolor = color(light); // Turn the angle button
light  break;

    case 150:                // Motor 5 done
        mpower[5-1].on=0;    // Turn the on flag off
        arduino.write(50);    // Make sure the motor is
off    rectm5on.basecolor = color(light);    // Turn the on button
light  rectm5off.basecolor = color(dark);    // Turn the off button
dark   rectm5angle.basecolor = color(light); // Turn the angle button
light  break;

```

```

    case 160:                                // Motor 6 done
        mpower[6-1].on=0;                    // Turn the on flag off
        arduino.write(60);                  // Make sure the motor is
off    rectm6on.basecolor = color(light);    // Turn the on button
light   rectm6off.basecolor = color(dark);    // Turn the off button
dark    rectm6angle.basecolor = color(light); // Turn the angle button
light   break;

    case 170:                                // Motor 7 done
        mpower[7-1].on=0;                    // Turn the on flag off
        arduino.write(70);                  // Make sure the motor is
off    rectm7on.basecolor = color(light);    // Turn the on button
light   rectm7off.basecolor = color(dark);    // Turn the off button
dark    rectm7angle.basecolor = color(light); // Turn the angle button
light   break;

    case 200:                                // All motors are done
        mpower[1-1].on=0;                    // Turn the on flag off
        mpower[2-1].on=0;                    // Turn the on flag off
        mpower[3-1].on=0;                    // Turn the on flag off
        mpower[4-1].on=0;                    // Turn the on flag off
        mpower[5-1].on=0;                    // Turn the on flag off
        mpower[6-1].on=0;                    // Turn the on flag off
        mpower[7-1].on=0;                    // Turn the on flag off
        normalized = 0;                      // Normalized flag off
light   rectm1on.basecolor = color(light);    // Turn the on button
light   rectm2on.basecolor = color(light);    // Turn the on button
light   rectm3on.basecolor = color(light);    // Turn the on button
light   rectm4on.basecolor = color(light);    // Turn the on button
light   rectm5on.basecolor = color(light);    // Turn the on button
light   rectm6on.basecolor = color(light);    // Turn the on button
light   rectm7on.basecolor = color(light);    // Turn the on button
light   rectm1off.basecolor = color(dark);    // Turn the off button
dark    rectm2off.basecolor = color(dark);    // Turn the off button
dark    rectm3off.basecolor = color(dark);    // Turn the off button
dark    rectm4off.basecolor = color(dark);    // Turn the off button
dark

```

```

        rectm5off.basecolor = color(dark);        // Turn the off button
dark
        rectm6off.basecolor = color(dark);        // Turn the off button
dark
        rectm7off.basecolor = color(dark);        // Turn the off button
dark
        rectnormalize.basecolor = color(light);    // Turn the normalize
button light
        rectneutstart.basecolor = color(light);    // Turn the button
light
        rectpoint_1_1a.basecolor = color(light);    // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light
        rectpoint_1c_2.basecolor = color(light);    // Turn the button
light
        rectpoint_2_2a.basecolor = color(light);    // Turn the button
light
        rectpoint_2a_2b.basecolor = color(light);    // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light);    // Turn the button
light
        rectpoint_2c_3.basecolor = color(light);    // Turn the button
light
        rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
        rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light);    // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light);    // Turn the button
light
        rectpoint_3d_4.basecolor = color(light);    // Turn the button
light
    }
}
}

// Update position of mouse
void update(int x, int y)
{
    if(locked == false) { // If mouse button is not locked

        // Update for highlight of button
        rectm1on.update();
        rectm1off.update();
        rectm2on.update();
        rectm2off.update();
        rectm3on.update();
        rectm3off.update();
        rectm4on.update();
        rectm4off.update();
        rectm5on.update();
        rectm5off.update();
        rectm6on.update();

```

```

rectm6off.update();
rectm7on.update();
rectm7off.update();

rectm1up.update();
rectm2up.update();
rectm3up.update();
rectm4up.update();
rectm5up.update();
rectm6up.update();
rectm7up.update();

rectm1down.update();
rectm2down.update();
rectm3down.update();
rectm4down.update();
rectm5down.update();
rectm6down.update();
rectm7down.update();

rectm1angle.update();
rectm2angle.update();
rectm3angle.update();
rectm4angle.update();
rectm5angle.update();
rectm6angle.update();
rectm7angle.update();

rectstop.update();
rectnormalize.update();
rectpmaxhigh.update();
rectpmaxmed.update();
rectpmaxlow.update();

rectneutstart.update();
rectpoint_1_1a.update();
rectpoint_1a_1b.update();
rectpoint_1b_1c.update();
rectpoint_1c_2.update();
rectpoint_2_2a.update();
rectpoint_2a_2b.update();
rectpoint_2b_2c.update();
rectpoint_2c_3.update();
rectpoint_3_3a.update();
rectpoint_3a_3b.update();
rectpoint_3b_3c.update();
rectpoint_3c_3d.update();
rectpoint_3d_4.update();

rectg1open.update();
rectg2open.update();
rectg1off.update();
rectg2off.update();
rectg1close.update();
rectg2close.update();
}
else {                                     // If locked, then unlock

```



```

    locked = false;
}

if((mousePressed)&&(click == 0)) {    // When the mouse is clicked or
held down
    click = 1;                        // Click flag clears once
released
    if(rectstop.pressed()) {          // The stop button will stop
everything and reset the buttons, but does not change direction
        arduino.write(1);             // Tell the MCU to stop all the
motors
        for(int i=1; i<(num+1); i++) { // Turn the On flags off for all
motors
            mpower[i-1].on=0;
        }
        // Turn the on buttons light and off buttons dark
rectm1on.basecolor = color(light);
rectm1off.basecolor = color(dark);
rectm2on.basecolor = color(light);
rectm2off.basecolor = color(dark);
rectm3on.basecolor = color(light);
rectm3off.basecolor = color(dark);
rectm4on.basecolor = color(light);
rectm4off.basecolor = color(dark);
rectm5on.basecolor = color(light);
rectm5off.basecolor = color(dark);
rectm6on.basecolor = color(light);
rectm6off.basecolor = color(dark);
rectm7on.basecolor = color(light);
rectm7off.basecolor = color(dark);

        // Turn all of the angle buttons light
rectm1angle.basecolor = color (light);
rectm2angle.basecolor = color (light);
rectm3angle.basecolor = color (light);
rectm4angle.basecolor = color (light);
rectm5angle.basecolor = color (light);
rectm6angle.basecolor = color (light);
rectm7angle.basecolor = color (light);

        // Turn the gripper off buttons dark and the open/close buttons
light
rectg1open.basecolor = color(light);
rectg2open.basecolor = color(light);
rectg1off.basecolor = color(dark);
rectg2off.basecolor = color(dark);
rectg1close.basecolor = color(light);
rectg2close.basecolor = color(light);

        normalized = 0;                // Turn the normalized
flag off

    }
    // Motor on/off buttons pressed
    else if(rectm1on.pressed()) {      // Motor 1 on button
        arduino.write(14);            // Tell the MCU that
the next value is power

```

```

        arduino.write(mpower[1-1].power);           // Send the power
value
        arduino.write(10);                           // Turn the motor off
to clear any increment currently running
        arduino.write(11);                           // Turn the motor on
        mpower[1-1].on = 1;                           // Turn the motor flag
on
        rectm1on.basecolor = color(dark);             // Make the on button
dark
        rectm1off.basecolor = color(light);           // Make the off button
light
        rectm1angle.basecolor = color(light);         // Make the increment
button light
    }
    else if(rectm1off.pressed()) {                     // Motor 1 off button
        arduino.write(10);                           // Tell the MCU to
turn the motor off
        mpower[1-1].on = 0;                           // Turn the motor
flag off
        rectm1on.basecolor = color(light);           // Make the on button
light
        rectm1off.basecolor = color(dark);           // Make the off
button dark
        rectm1angle.basecolor = color(light);         // Make the increment
button light
    }
    else if(rectm2on.pressed()) {                     // Motor 2 on button
        arduino.write(24);                           // Tell the MCU that
the next value is power
        arduino.write(mpower[2-1].power);           // Send the power
value
        arduino.write(20);                           // Turn the motor off
to clear any increment currently running
        arduino.write(21);                           // Turn the motor on
        mpower[2-1].on = 1;                           // Turn the motor flag
on
        rectm2on.basecolor = color(dark);             // Make the on button
dark
        rectm2off.basecolor = color(light);           // Make the off button
light
        rectm2angle.basecolor = color(light);         // Make the increment
button light
    }
    else if(rectm2off.pressed()) {                     // Motor 2 off button
        arduino.write(20);                           // Tell the MCU to
turn the motor off
        mpower[2-1].on = 0;                           // Turn the motor
flag off
        rectm2on.basecolor = color(light);           // Make the on button
light
        rectm2off.basecolor = color(dark);           // Make the off
button dark
        rectm2angle.basecolor = color(light);         // Make the increment
button light
    }
    else if(rectm3on.pressed()) {                     // Motor 3 on button

```

```

        arduino.write(34); // Tell the MCU that
the next value is power
        arduino.write(mpower[3-1].power); // Send the power
value
        arduino.write(30); // Turn the motor off
to clear any increment currently running
        arduino.write(31); // Turn the motor on
        mpower[3-1].on = 1; // Turn the motor flag
on
        rectm3on.basecolor = color(dark); // Make the on button
dark
        rectm3off.basecolor = color(light); // Make the off button
light
        rectm3angle.basecolor = color(light); // Make the increment
button light
    }
    else if(rectm3off.pressed()) { // Motor 3 off button
        arduino.write(30); // Tell the MCU to
turn the motor off
        mpower[3-1].on = 0; // Turn the motor
flag off
        rectm3on.basecolor = color(light); // Make the on button
light
        rectm3off.basecolor = color(dark); // Make the off
button dark
        rectm3angle.basecolor = color(light); // Make the increment
button light
    }
    else if(rectm4on.pressed()) { // Motor 4 on button
        arduino.write(44); // Tell the MCU that
the next value is power
        arduino.write(mpower[4-1].power); // Send the power
value
        arduino.write(40); // Turn the motor off
to clear any increment currently running
        arduino.write(41); // Turn the motor on
        mpower[4-1].on = 1; // Turn the motor flag
on
        rectm4on.basecolor = color(dark); // Make the on button
dark
        rectm4off.basecolor = color(light); // Make the off button
light
        rectm4angle.basecolor = color(light); // Make the increment
button light
    }
    else if(rectm4off.pressed()) { // Motor 4 off button
        arduino.write(40); // Tell the MCU to
turn the motor off
        mpower[4-1].on = 0; // Turn the motor
flag off
        rectm4on.basecolor = color(light); // Make the on button
light
        rectm4off.basecolor = color(dark); // Make the off
button dark
        rectm4angle.basecolor = color(light); // Make the increment
button light
    }
}

```

```

        else if(rectm5on.pressed()) { // Motor 5 on button
            arduino.write(54); // Tell the MCU that
the next value is power // Send the power
            arduino.write(mpower[5-1].power);
value // Turn the motor off
            arduino.write(50); // Turn the motor on
to clear any increment currently running // Turn the motor flag
            arduino.write(51); // Make the on button
            mpower[5-1].on = 1; // Make the off button
on // Make the increment
            rectm5on.basecolor = color(dark);
dark // Make the on button
            rectm5off.basecolor = color(light);
light // Make the off button
            rectm5angle.basecolor = color(light);
button light // Make the increment
        }
        else if(rectm5off.pressed()) { // Motor 5 off button
            arduino.write(50); // Tell the MCU to
turn the motor off // Turn the motor
            mpower[5-1].on = 0; // Make the on button
flag off // Make the off
            rectm5on.basecolor = color(light); // Make the off
light // Make the increment
            rectm5off.basecolor = color(dark);
button dark // Make the increment
            rectm5angle.basecolor = color(light);
button light
        }
        else if(rectm6on.pressed()) { // Motor 6 on button
            arduino.write(64); // Tell the MCU that
the next value is power // Send the power
            arduino.write(mpower[6-1].power);
value // Turn the motor off
            arduino.write(60); // Turn the motor on
to clear any increment currently running // Turn the motor flag
            arduino.write(61); // Make the on button
            mpower[6-1].on = 1; // Make the off button
on // Make the increment
            rectm6on.basecolor = color(dark);
dark // Make the on button
            rectm6off.basecolor = color(light);
light // Make the off button
            rectm6angle.basecolor = color(light);
button light // Make the increment
        }
        else if(rectm6off.pressed()) { // Motor 6 off button
            arduino.write(60); // Tell the MCU to
turn the motor off // Turn the motor
            mpower[6-1].on = 0; // Make the on button
flag off // Make the off
            rectm6on.basecolor = color(light); // Make the off
light // Make the increment
            rectm6off.basecolor = color(dark);
button dark // Make the increment
            rectm6angle.basecolor = color(light);
button light
        }
    }

```

```

    }
    else if(rectm7on.pressed()) { // Motor 7 on button
        arduino.write(74); // Tell the MCU that
the next value is power
        arduino.write(mpower[7-1].power); // Send the power
value
        arduino.write(70); // Turn the motor off
to clear any increment currently running
        arduino.write(71); // Turn the motor on
        mpower[7-1].on = 1; // Turn the motor flag
on
        rectm7on.basecolor = color(dark); // Make the on button
dark
        rectm7off.basecolor = color(light); // Make the off button
light
        rectm7angle.basecolor = color(light); // Make the increment
button light
    }
    else if(rectm7off.pressed()) { // Motor 7 off button
        arduino.write(70); // Tell the MCU to
turn the motor off
        mpower[7-1].on = 0; // Turn the motor
flag off
        rectm7on.basecolor = color(light); // Make the on button
light
        rectm7off.basecolor = color(dark); // Make the off
button dark
        rectm7angle.basecolor = color(light); // Make the increment
button light
    }

    //-----Motor direction button pressed-----
    else if(rectmlup.pressed()) { // Motor 1 up button
        arduino.write(12); // Tell the board to
change direction to up
        mpower[1-1].dir = 1; // Set the direction
flag high
        rectmlup.basecolor = color(dark); // Make the up button
dark
        rectmldown.basecolor = color(light); // Make the down
button light
    }
    else if(rectmldown.pressed()) { // Motor 1 down button
        arduino.write(13); // Tell the board to
change direction to down
        mpower[1-1].dir = 0; // Set the direction
flag low
        rectmlup.basecolor = color(light); // Make the up button
light
        rectmldown.basecolor = color(dark); // Make the down
button dark
    }
    else if(rectm2up.pressed()) { // Motor 2 up button
        arduino.write(22); // Tell the board to
change direction to up
        mpower[2-1].dir = 1; // Set the direction
flag high

```

```

        rectm2up.basecolor = color(dark);           // Make the up button
dark
        rectm2down.basecolor = color(light);        // Make the down
button light
    }
    else if(rectm2down.pressed()) {                // Motor 2 down button
        arduino.write(23);                          // Tell the board to
change direction to down
        mpower[2-1].dir = 0;                        // Set the direction
flag low
        rectm2up.basecolor = color(light);          // Make the up button
light
        rectm2down.basecolor = color(dark);         // Make the down
button dark
    }
    else if(rectm3up.pressed()) {                  // Motor 3 up button
        arduino.write(32);                          // Tell the board to
change direction to up
        mpower[3-1].dir = 1;                        // Set the direction
flag high
        rectm3up.basecolor = color(dark);           // Make the up button
dark
        rectm3down.basecolor = color(light);        // Make the down
button light
    }
    else if(rectm3down.pressed()) {                // Motor 3 down button
        arduino.write(33);                          // Tell the board to
change direction to down
        mpower[3-1].dir = 0;                        // Set the direction
flag low
        rectm3up.basecolor = color(light);          // Make the up button
light
        rectm3down.basecolor = color(dark);         // Make the down
button dark
    }
    else if(rectm4up.pressed()) {                  // Motor 4 up button
        arduino.write(42);                          // Tell the board to
change direction to up
        mpower[4-1].dir = 1;                        // Set the direction
flag high
        rectm4up.basecolor = color(dark);           // Make the up button
dark
        rectm4down.basecolor = color(light);        // Make the down
button light
    }
    else if(rectm4down.pressed()) {                // Motor 4 down button
        arduino.write(43);                          // Tell the board to
change direction to down
        mpower[4-1].dir = 0;                        // Set the direction
flag low
        rectm4up.basecolor = color(light);          // Make the up button
light
        rectm4down.basecolor = color(dark);         // Make the down
button dark
    }
    else if(rectm5up.pressed()) {                  // Motor 5 up button

```

```

        arduino.write(52); // Tell the board to
change direction to up
        mpower[5-1].dir = 1; // Set the direction
flag high
        rectm5up.basecolor = color(dark); // Make the up button
dark
        rectm5down.basecolor = color(light); // Make the down
button light
    }
    else if(rectm5down.pressed()) { // Motor 5 down button
        arduino.write(53); // Tell the board to
change direction to down
        mpower[5-1].dir = 0; // Set the direction
flag low
        rectm5up.basecolor = color(light); // Make the up button
light
        rectm5down.basecolor = color(dark); // Make the down
button dark
    }
    else if(rectm6up.pressed()) { // Motor 6 up button
        arduino.write(62); // Tell the board to
change direction to up
        mpower[6-1].dir = 1; // Set the direction
flag high
        rectm6up.basecolor = color(dark); // Make the up button
dark
        rectm6down.basecolor = color(light); // Make the down
button light
    }
    else if(rectm6down.pressed()) { // Motor 6 down button
        arduino.write(63); // Tell the board to
change direction to down
        mpower[6-1].dir = 0; // Set the direction
flag low
        rectm6up.basecolor = color(light); // Make the up button
light
        rectm6down.basecolor = color(dark); // Make the down
button dark
    }
    else if(rectm7up.pressed()) { // Motor 7 up button
        arduino.write(72); // Tell the board to
change direction to up
        mpower[7-1].dir = 1; // Set the direction
flag high
        rectm7up.basecolor = color(dark); // Make the up button
dark
        rectm7down.basecolor = color(light); // Make the down
button light
    }
    else if(rectm7down.pressed()) { // Motor 7 down button
        arduino.write(73); // Tell the board to
change direction to down
        mpower[7-1].dir = 0; // Set the direction
flag low
        rectm7up.basecolor = color(light); // Make the up button
light

```

```

        rectm7down.basecolor = color(dark);           // Make the down
button dark
    }

    /-----Increment angle button pressed-----
    else if(rectm1angle.pressed()) {                 // Increment joint 1
        arduino.write(14);                           // Indicate the
power is coming
        arduino.write(mpower[1-1].power);           // Send the power
        arduino.write(15);                           // Indicate the
angle is coming
        arduino.write(mpower[1-1].angle+2);         // Send the angle
        mpower[1-1].on = 1;                           // Turn the flag on
        rectm1angle.basecolor = color(dark);         // Keep the button
dark until angle is reached
        rectm1on.basecolor = color(light);           // Turn the on
button light
        rectm1off.basecolor = color(dark);           // Turn the off
button dark
    }
    else if(rectm2angle.pressed()) {                 // Increment joint 2
        arduino.write(24);                           // Indicate the
power is coming
        arduino.write(mpower[2-1].power);           // Send the power
        arduino.write(25);                           // Indicate the
angle is coming
        arduino.write(mpower[2-1].angle+2);         // Send the angle
        mpower[2-1].on = 1;                           // Turn the flag on
        rectm2angle.basecolor = color(dark);         // Keep the button
dark until angle is reached
        rectm2on.basecolor = color(light);           // Turn the on
button light
        rectm2off.basecolor = color(dark);           // Turn the off
button dark
    }
    else if(rectm3angle.pressed()) {                 // Increment joint 3
        arduino.write(34);                           // Indicate the
power is coming
        arduino.write(mpower[3-1].power);           // Send the power
        arduino.write(35);                           // Indicate the
angle is coming
        arduino.write(mpower[3-1].angle+2);         // Send the angle
        mpower[3-1].on = 1;                           // Turn the flag on
        rectm3angle.basecolor = color(dark);         // Keep the button
dark until angle is reached
        rectm3on.basecolor = color(light);           // Turn the on
button light
        rectm3off.basecolor = color(dark);           // Turn the off
button dark
    }
    else if(rectm4angle.pressed()) {                 // Increment joint 4
        arduino.write(44);                           // Indicate the
power is coming
        arduino.write(mpower[4-1].power);           // Send the power
        arduino.write(45);                           // Indicate the
angle is coming
        arduino.write(mpower[4-1].angle+2);         // Send the angle

```



```

        mpower[4-1].on = 1;                                // Turn the flag on
        rectm4angle.basecolor = color(dark);              // Keep the button
        dark until angle is reached
        rectm4on.basecolor = color(light);                // Turn the on
        button light
        rectm4off.basecolor = color(dark);                // Turn the off
        button dark
    }
    else if(rectm5angle.pressed()) {                      // Increment joint 5
        arduino.write(54);                                // Indicate the
        power is coming
        arduino.write(mpower[5-1].power);                 // Send the power
        arduino.write(55);                                // Indicate the
        angle is coming
        arduino.write(mpower[5-1].angle+2);               // Send the angle
        mpower[5-1].on = 1;                                // Turn the flag on
        rectm5angle.basecolor = color(dark);              // Keep the button
        dark until angle is reached
        rectm5on.basecolor = color(light);                // Turn the on
        button light
        rectm5off.basecolor = color(dark);                // Turn the off
        button dark
    }
    else if(rectm6angle.pressed()) {                      // Increment joint 6
        arduino.write(64);                                // Indicate the
        power is coming
        arduino.write(mpower[6-1].power);                 // Send the power
        arduino.write(65);                                // Indicate the
        angle is coming
        arduino.write(mpower[6-1].angle+2);               // Send the angle
        mpower[6-1].on = 1;                                // Turn the flag on
        rectm6angle.basecolor = color(dark);              // Keep the button
        dark until angle is reached
        rectm6on.basecolor = color(light);                // Turn the on
        button light
        rectm6off.basecolor = color(dark);                // Turn the off
        button dark
    }
    else if(rectm7angle.pressed()) {                      // Increment joint 7
        arduino.write(74);                                // Indicate the
        power is coming
        arduino.write(mpower[7-1].power);                 // Send the power
        arduino.write(75);                                // Indicate the
        angle is coming
        arduino.write(mpower[7-1].angle+2);               // Send the angle
        mpower[7-1].on = 1;                                // Turn the flag on
        rectm7angle.basecolor = color(dark);              // Keep the button
        dark until angle is reached
        rectm7on.basecolor = color(light);                // Turn the on
        button light
        rectm7off.basecolor = color(dark);                // Turn the off
        button dark
    }
    //-----Gripper buttons pressed: open/off/close-----
    else if(rectglopen.pressed()) {                      // Gripper 1 open
        button

```

```

        arduino.write(82); // Tell the board
to open gripper 1
    rectg1open.basecolor = color(dark); // Make the open
button dark
    rectg1off.basecolor = color(light); // Make the off
button light
    rectg1close.basecolor = color(light); // Make the close
button light
    }
    else if(rectg2open.pressed()) { // Gripper 2 open
button
        arduino.write(92); // Tell the board
to open gripper 2
        rectg2open.basecolor = color(dark); // Make the open
button dark
        rectg2off.basecolor = color(light); // Make the off
button light
        rectg2close.basecolor = color(light); // Make the close
button light
    }
    else if(rectg1off.pressed()) { // Gripper 1 off button
        arduino.write(80); // Tell the board
to turn off gripper 1
        rectg1open.basecolor = color(light); // Make the open
button light
        rectg1off.basecolor = color(dark); // Make the off
button dark
        rectg1close.basecolor = color(light); // Make the close
button light
    }
    else if(rectg2off.pressed()) { // Gripper 2 off button
        arduino.write(90); // Tell the board
to turn off gripper 2
        rectg2open.basecolor = color(light); // Make the open
button light
        rectg2off.basecolor = color(dark); // Make the off
button dark
        rectg2close.basecolor = color(light); // Make the close
button light
    }
    else if(rectg1close.pressed()) { // Gripper 1 close
button
        arduino.write(81); // Tell the board
to close gripper 1
        rectg1open.basecolor = color(light); // Make the open
button light
        rectg1off.basecolor = color(light); // Make the off
button light
        rectg1close.basecolor = color(dark); // Make the close
button dark
    }
    else if(rectg2close.pressed()) { // Gripper 2 close
button
        arduino.write(91); // Tell the board
to close gripper 2
        rectg2open.basecolor = color(light); // Make the open
button light

```

```

        rectg2off.basecolor = color(light);           // Make the off
button light
        rectg2close.basecolor = color(dark);         // Make the close
button dark
    }
    //-----Normalized-----
    else if(rectnormalize.pressed()) { // Increment all the angles,
normalized
        normalized = 1; // Normalized flag on
        rectnormalize.basecolor = color(dark); // Turn the button dark
        rectneutstart.basecolor = color(light); // Turn the button
light
        rectpoint_1_1a.basecolor = color(light); // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
        rectpoint_1c_2.basecolor = color(light); // Turn the button
light
        rectpoint_2_2a.basecolor = color(light); // Turn the button
light
        rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
        rectpoint_2c_3.basecolor = color(light); // Turn the button
light
        rectpoint_3_3a.basecolor = color(light); // Turn the button
light
        rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
        rectpoint_3d_4.basecolor = color(light); // Turn the button
light
        arduino.write(1); // Stop everything
        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(mpower[1-1].angle+2); // Send all of the
angles
        arduino.write(mpower[2-1].angle+2);
        arduino.write(mpower[3-1].angle+2);
        arduino.write(mpower[4-1].angle+2);
        arduino.write(mpower[5-1].angle+2);
        arduino.write(mpower[6-1].angle+2);
        arduino.write(mpower[7-1].angle+2);

        println("Normalized Incrementation"); // Print in text area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on

```

```

    mpower[5-1].on=1;           // Turn the on flag on
    mpower[6-1].on=1;           // Turn the on flag on
    mpower[7-1].on=1;           // Turn the on flag on
    rectm1on.basecolor = color(light);    // Turn the on button light
    rectm2on.basecolor = color(light);    // Turn the on button light
    rectm3on.basecolor = color(light);    // Turn the on button light
    rectm4on.basecolor = color(light);    // Turn the on button light
    rectm5on.basecolor = color(light);    // Turn the on button light
    rectm6on.basecolor = color(light);    // Turn the on button light
    rectm7on.basecolor = color(light);    // Turn the on button light
    rectm1off.basecolor = color(dark);    // Turn the off button dark
    rectm2off.basecolor = color(dark);    // Turn the off button dark
    rectm3off.basecolor = color(dark);    // Turn the off button dark
    rectm4off.basecolor = color(dark);    // Turn the off button dark
    rectm5off.basecolor = color(dark);    // Turn the off button dark
    rectm6off.basecolor = color(dark);    // Turn the off button dark
    rectm7off.basecolor = color(dark);    // Turn the off button dark
}

// Set the Maximum Power sent to the motors during normalization
motion
// Set to high setting
else if(rectpmaxhigh.pressed()) {
    println("Motor Max Power High");           // Print in window below
    arduino.write(101);                       // Tell the MCU the new
power setting
    rectpmaxhigh.basecolor = color(dark);    // Make the high button
dark
    rectpmaxmed.basecolor = color(light);    // Make the med button
light
    rectpmaxlow.basecolor = color(light);    // Make the low button
light
}

// Set to medium setting
else if(rectpmaxmed.pressed()) {
    println("Motor Max Power Medium");           // Print in window
below
    arduino.write(102);                       // Tell the MCU the new
power setting
    rectpmaxhigh.basecolor = color(light);    // Make the high button
light
    rectpmaxmed.basecolor = color(dark);    // Make the med button
dark
    rectpmaxlow.basecolor = color(light);    // Make the low button
light
}

// Set to low setting
else if(rectpmaxlow.pressed()) {
    println("Motor Max Power Low");           // Print in window below
    arduino.write(103);                       // Tell the MCU the new
power setting
    rectpmaxhigh.basecolor = color(light);    // Make the high button
light
    rectpmaxmed.basecolor = color(light);    // Make the med button
light

```

```

        rectpmaxlow.basecolor = color(dark);    // Make the low button
dark
    }

    // Preset motions-----

    else if(rectneutstart.pressed()) {    // Increment all the angles,
normalized
        normalized = 1;                    // Normalized flag on
        rectnormalize.basecolor = color(light);    // Turn the button
light
        rectneutstart.basecolor = color(dark);    // Turn the button dark
        rectpoint_1_1a.basecolor = color(light);    // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light
        rectpoint_1c_2.basecolor = color(light);    // Turn the button
light
        rectpoint_2_2a.basecolor = color(light);    // Turn the button
light
        rectpoint_2a_2b.basecolor = color(light);    // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light);    // Turn the button
light
        rectpoint_2c_3.basecolor = color(light);    // Turn the button
light
        rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
        rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light);    // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light);    // Turn the button
light
        rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

        arduino.write(1);                    // Stop everything
        // Set all of the directions
        // Motor 1
        arduino.write(13);                    // Tell the board
to change direction to down
        mpower[1-1].dir = 0;                    // Turn the
direction flag to down
        rectmlup.basecolor = color(light);    // Make the up
button light
        rectmldown.basecolor = color(dark);    // Make the down
button dark
        // Motor 2
        arduino.write(23);                    // Tell the board
to change direction to down
        mpower[2-1].dir = 0;                    // Turn the
direction flag to down
        rectm2up.basecolor = color(light);    // Make the up
button light

```

```

        rectm2down.basecolor = color(dark);           // Make the down
button dark
    //Motor 3
    arduino.write(33);                               // Tell the board
to change direction to down
    mpower[3-1].dir = 0;                             // Turn the
direction flag to down
    rectm3up.basecolor = color(light);               // Make the up
button light
    rectm3down.basecolor = color(dark);              // Make the down
button dark
    // Motor 4
    arduino.write(43);                               // Tell the board
to change direction to down
    mpower[4-1].dir = 0;                             // Turn the
direction flag to down
    rectm4up.basecolor = color(light);               // Make the up
button light
    rectm4down.basecolor = color(dark);              // Make the down
button dark
    // Motor 5
    arduino.write(53);                               // Tell the board
to change direction to down
    mpower[5-1].dir = 0;                             // Turn the
direction flag to down
    rectm5up.basecolor = color(light);               // Make the up
button light
    rectm5down.basecolor = color(dark);              // Make the down
button dark
    // Motor 6
    arduino.write(63);                               // Tell the board
to change direction to down
    mpower[6-1].dir = 0;                             // Turn the
direction flag to down
    rectm6up.basecolor = color(light);               // Make the up
button light
    rectm6down.basecolor = color(dark);              // Make the down
button dark
    // Motor 7
    arduino.write(73);                               // Tell the board
to change direction to down
    mpower[7-1].dir = 0;                             // Turn the
direction flag to down
    rectm7up.basecolor = color(light);               // Make the up
button light
    rectm7down.basecolor = color(dark);              // Make the down
button dark

    arduino.write(2);                               // Tell arduino that all of the
angles will be sent
    arduino.write(45*2+2); // Motor 1    Send all of the angles
(multiply by 2 then add 2)
    arduino.write(0*2+2); // Motor 2    Limit the number of motors
running due to power supply
    arduino.write(0*2+2); // Motor 3
    arduino.write(80*2+2); // Motor 4

```

```

    arduino.write(0*2+2);    // Motor 5
    arduino.write(0*2+2);    // Motor 6
    arduino.write(0*2+2);    // Motor 7

    println("Normalized Motion from Neutral position to start
position"); // Print in text area below

    mpower[1-1].on=1;        // Turn the on flag on
    mpower[2-1].on=1;        // Turn the on flag on
    mpower[3-1].on=1;        // Turn the on flag on
    mpower[4-1].on=1;        // Turn the on flag on
    mpower[5-1].on=1;        // Turn the on flag on
    mpower[6-1].on=1;        // Turn the on flag on
    mpower[7-1].on=1;        // Turn the on flag on
    rectm1on.basecolor = color(light);    // Turn the on button light
    rectm2on.basecolor = color(light);    // Turn the on button light
    rectm3on.basecolor = color(light);    // Turn the on button light
    rectm4on.basecolor = color(light);    // Turn the on button light
    rectm5on.basecolor = color(light);    // Turn the on button light
    rectm6on.basecolor = color(light);    // Turn the on button light
    rectm7on.basecolor = color(light);    // Turn the on button light
    rectm1off.basecolor = color(dark);    // Turn the off button dark
    rectm2off.basecolor = color(dark);    // Turn the off button dark
    rectm3off.basecolor = color(dark);    // Turn the off button dark
    rectm4off.basecolor = color(dark);    // Turn the off button dark
    rectm5off.basecolor = color(dark);    // Turn the off button dark
    rectm6off.basecolor = color(dark);    // Turn the off button dark
    rectm7off.basecolor = color(dark);    // Turn the off button dark
}

//-----
else if(rectpoint_1_1a.pressed()) {      // Increment all the
angles, normalized
    normalized = 1;                      // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(dark); // Turn the button dark
    rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(light); // Turn the button
light

```

```

        rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
        rectpoint_3d_4.basecolor = color(light); // Turn the button
light

        arduino.write(1); // Stop everything
        // Set all of the directions
        // Motor 1
        arduino.write(13); // Tell the board
to change direction to down
        mpower[1-1].dir = 0; // Turn the
direction flag to down
        rectm1up.basecolor = color(light); // Make the up
button light
        rectm1down.basecolor = color(dark); // Make the down
button dark
        // Motor 2
        arduino.write(23); // Tell the board
to change direction to down
        mpower[2-1].dir = 0; // Turn the
direction flag to down
        rectm2up.basecolor = color(light); // Make the up
button light
        rectm2down.basecolor = color(dark); // Make the down
button dark
        //Motor 3
        arduino.write(33); // Tell the board
to change direction to down
        mpower[3-1].dir = 0; // Turn the
direction flag to down
        rectm3up.basecolor = color(light); // Make the up
button light
        rectm3down.basecolor = color(dark); // Make the down
button dark
        // Motor 4
        arduino.write(43); // Tell the board
to change direction to down
        mpower[4-1].dir = 0; // Turn the
direction flag to down
        rectm4up.basecolor = color(light); // Make the up
button light
        rectm4down.basecolor = color(dark); // Make the down
button dark
        // Motor 5
        arduino.write(53); // Tell the board
to change direction to down
        mpower[5-1].dir = 0; // Turn the
direction flag to down
        rectm5up.basecolor = color(light); // Make the up
button light
        rectm5down.basecolor = color(dark); // Make the down
button dark
        // Motor 6

```



```

        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(73); // Tell the board
to change direction to down
        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(0*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(55*2+2); // Motor 5
        arduino.write(65*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

        println("Normalized Motion from pt 1 to 1a"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on
        mpower[7-1].on=1; // Turn the on flag on
        rectm1on.basecolor = color(light); // Turn the on button light
        rectm2on.basecolor = color(light); // Turn the on button light
        rectm3on.basecolor = color(light); // Turn the on button light
        rectm4on.basecolor = color(light); // Turn the on button light
        rectm5on.basecolor = color(light); // Turn the on button light
        rectm6on.basecolor = color(light); // Turn the on button light
        rectm7on.basecolor = color(light); // Turn the on button light
        rectm1off.basecolor = color(dark); // Turn the off button dark
        rectm2off.basecolor = color(dark); // Turn the off button dark
        rectm3off.basecolor = color(dark); // Turn the off button dark
        rectm4off.basecolor = color(dark); // Turn the off button dark
        rectm5off.basecolor = color(dark); // Turn the off button dark
        rectm6off.basecolor = color(dark); // Turn the off button dark
        rectm7off.basecolor = color(dark); // Turn the off button dark
    }

```

```

//-----
else if(rectpoint_1a_1b.pressed()) { // Increment all the
angles, normalized
    normalized = 1; // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(light); // Turn the button
light
    rectpoint_1a_1b.basecolor = color(dark); // Turn the button dark
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(light); // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
    rectpoint_3d_4.basecolor = color(light); // Turn the button
light

    arduino.write(1); // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13); // Tell the board
to change direction to down
    mpower[1-1].dir = 0; // Turn the
direction flag to down
    rectmlup.basecolor = color(light); // Make the up
button light
    rectmldown.basecolor = color(dark); // Make the down
button dark
    // Motor 2
    arduino.write(23); // Tell the board
to change direction to down
    mpower[2-1].dir = 0; // Turn the
direction flag to down
    rectm2up.basecolor = color(light); // Make the up
button light
    rectm2down.basecolor = color(dark); // Make the down
button dark
    //Motor 3

```

```

        arduino.write(32); // Tell the board
to change direction to up
        mpower[3-1].dir = 1; // Turn the
direction flag to up
        rectm3up.basecolor = color(dark); // Make the up
button dark
        rectm3down.basecolor = color(light); // Make the down
button light
        // Motor 4
        arduino.write(43); // Tell the board
to change direction to down
        mpower[4-1].dir = 0; // Turn the
direction flag to down
        rectm4up.basecolor = color(light); // Make the up
button light
        rectm4down.basecolor = color(dark); // Make the down
button dark
        // Motor 5
        arduino.write(53); // Tell the board
to change direction to down
        mpower[5-1].dir = 0; // Turn the
direction flag to down
        rectm5up.basecolor = color(light); // Make the up
button light
        rectm5down.basecolor = color(dark); // Make the down
button dark
        // Motor 6
        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(73); // Tell the board
to change direction to down
        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(15*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(20*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

```

```
println("Normalized Motion from pt 1a to 1b"); // Print in text
area below
```

```

mpower[1-1].on=1;           // Turn the on flag on
mpower[2-1].on=1;           // Turn the on flag on
mpower[3-1].on=1;           // Turn the on flag on
mpower[4-1].on=1;           // Turn the on flag on
mpower[5-1].on=1;           // Turn the on flag on
mpower[6-1].on=1;           // Turn the on flag on
mpower[7-1].on=1;           // Turn the on flag on
rectm1on.basecolor = color(light); // Turn the on button light
rectm2on.basecolor = color(light); // Turn the on button light
rectm3on.basecolor = color(light); // Turn the on button light
rectm4on.basecolor = color(light); // Turn the on button light
rectm5on.basecolor = color(light); // Turn the on button light
rectm6on.basecolor = color(light); // Turn the on button light
rectm7on.basecolor = color(light); // Turn the on button light
rectm1off.basecolor = color(dark); // Turn the off button dark
rectm2off.basecolor = color(dark); // Turn the off button dark
rectm3off.basecolor = color(dark); // Turn the off button dark
rectm4off.basecolor = color(dark); // Turn the off button dark
rectm5off.basecolor = color(dark); // Turn the off button dark
rectm6off.basecolor = color(dark); // Turn the off button dark
rectm7off.basecolor = color(dark); // Turn the off button dark
}

//-----
else if(rectpoint_1b_1c.pressed()) { // Increment all the
angles, normalized
    normalized = 1; // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(light); // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
    rectpoint_1b_1c.basecolor = color(dark); // Turn the button dark
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(light); // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light); // Turn the button
light

```

```

    rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
    rectpoint_3d_4.basecolor = color(light); // Turn the button
light

    arduino.write(1); // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13); // Tell the board
to change direction to down
    mpower[1-1].dir = 0; // Turn the
direction flag to down
    rectm1up.basecolor = color(light); // Make the up
button light
    rectm1down.basecolor = color(dark); // Make the down
button dark
    // Motor 2
    arduino.write(23); // Tell the board
to change direction to down
    mpower[2-1].dir = 0; // Turn the
direction flag to down
    rectm2up.basecolor = color(light); // Make the up
button light
    rectm2down.basecolor = color(dark); // Make the down
button dark
    //Motor 3
    arduino.write(33); // Tell the board
to change direction to down
    mpower[3-1].dir = 0; // Turn the
direction flag to down
    rectm3up.basecolor = color(light); // Make the up
button light
    rectm3down.basecolor = color(dark); // Make the down
button dark
    // Motor 4
    arduino.write(43); // Tell the board
to change direction to down
    mpower[4-1].dir = 0; // Turn the
direction flag to down
    rectm4up.basecolor = color(light); // Make the up
button light
    rectm4down.basecolor = color(dark); // Make the down
button dark
    // Motor 5
    arduino.write(53); // Tell the board
to change direction to down
    mpower[5-1].dir = 0; // Turn the
direction flag to down
    rectm5up.basecolor = color(light); // Make the up
button light
    rectm5down.basecolor = color(dark); // Make the down
button dark
    // Motor 6
    arduino.write(63); // Tell the board
to change direction to down
    mpower[6-1].dir = 0; // Turn the
direction flag to down

```

```

        rectm6up.basecolor = color(light);           // Make the up
button light
        rectm6down.basecolor = color(dark);         // Make the down
button dark
        // Motor 7
        arduino.write(72);                          // Tell the board
to change direction to up
        mpower[7-1].dir = 1;                        // Turn the
direction flag to up
        rectm7up.basecolor = color(dark);           // Make the up
button dark
        rectm7down.basecolor = color(light);        // Make the down
button light

```

```

        arduino.write(2);                          // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2);                      // Motor 1           // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(30*2+2);                    // Motor 2           // Limit the
number of motors running due to power supply
        arduino.write(0*2+2);                      // Motor 3
        arduino.write(0*2+2);                      // Motor 4
        arduino.write(0*2+2);                      // Motor 5
        arduino.write(0*2+2);                      // Motor 6
        arduino.write(20*2+2);                    // Motor 7

```

```

        println("Normalized Motion from pt 1b to 1c"); // Print in text
area below

```

```

        mpower[1-1].on=1;                          // Turn the on flag on
        mpower[2-1].on=1;                          // Turn the on flag on
        mpower[3-1].on=1;                          // Turn the on flag on
        mpower[4-1].on=1;                          // Turn the on flag on
        mpower[5-1].on=1;                          // Turn the on flag on
        mpower[6-1].on=1;                          // Turn the on flag on
        mpower[7-1].on=1;                          // Turn the on flag on
        rectm1on.basecolor = color(light);          // Turn the on button light
        rectm2on.basecolor = color(light);          // Turn the on button light
        rectm3on.basecolor = color(light);          // Turn the on button light
        rectm4on.basecolor = color(light);          // Turn the on button light
        rectm5on.basecolor = color(light);          // Turn the on button light
        rectm6on.basecolor = color(light);          // Turn the on button light
        rectm7on.basecolor = color(light);          // Turn the on button light
        rectm1off.basecolor = color(dark);          // Turn the off button dark
        rectm2off.basecolor = color(dark);          // Turn the off button dark
        rectm3off.basecolor = color(dark);          // Turn the off button dark
        rectm4off.basecolor = color(dark);          // Turn the off button dark
        rectm5off.basecolor = color(dark);          // Turn the off button dark
        rectm6off.basecolor = color(dark);          // Turn the off button dark
        rectm7off.basecolor = color(dark);          // Turn the off button dark
    }
    //-----
    else if(rectpoint_1c_2.pressed()) {             // Increment all the
angles, normalized
        normalized = 1;                            // Normalized flag on

```

```

    rectnormalize.basecolor = color(light);    // Turn the button
light
    rectneutstart.basecolor = color(light);    // Turn the button
light
    rectpoint_1_1a.basecolor = color(light);    // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light
    rectpoint_1c_2.basecolor = color(dark);    // Turn the button dark
    rectpoint_2_2a.basecolor = color(light);    // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light);    // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light);    // Turn the button
light
    rectpoint_2c_3.basecolor = color(light);    // Turn the button
light
    rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light);    // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light);    // Turn the button
light
    rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

    arduino.write(1);                            // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13);                            // Tell the board
to change direction to down
    mpower[1-1].dir = 0;                            // Turn the
direction flag to down
    rectm1up.basecolor = color(light);                // Make the up
button light
    rectm1down.basecolor = color(dark);                // Make the down
button dark
    // Motor 2
    arduino.write(23);                            // Tell the board
to change direction to down
    mpower[2-1].dir = 0;                            // Turn the
direction flag to down
    rectm2up.basecolor = color(light);                // Make the up
button light
    rectm2down.basecolor = color(dark);                // Make the down
button dark
    //Motor 3
    arduino.write(32);                            // Tell the board
to change direction to up
    mpower[3-1].dir = 1;                            // Turn the
direction flag to up
    rectm3up.basecolor = color(dark);                // Make the up
button dark

```

```

        rectm3down.basecolor = color(light);           // Make the down
button light
    // Motor 4
    arduino.write(43);                               // Tell the board
to change direction to down
    mpower[4-1].dir = 0;                             // Turn the
direction flag to down
    rectm4up.basecolor = color(light);               // Make the up
button light
    rectm4down.basecolor = color(dark);              // Make the down
button dark
    // Motor 5
    arduino.write(53);                               // Tell the board
to change direction to down
    mpower[5-1].dir = 0;                             // Turn the
direction flag to down
    rectm5up.basecolor = color(light);               // Make the up
button light
    rectm5down.basecolor = color(dark);              // Make the down
button dark
    // Motor 6
    arduino.write(62);                               // Tell the board
to change direction to up
    mpower[6-1].dir = 1;                             // Turn the
direction flag to up
    rectm6up.basecolor = color(dark);               // Make the up
button dark
    rectm6down.basecolor = color(light);             // Make the down
button light
    // Motor 7
    arduino.write(73);                               // Tell the board
to change direction to down
    mpower[7-1].dir = 0;                             // Turn the
direction flag to down
    rectm7up.basecolor = color(light);               // Make the up
button light
    rectm7down.basecolor = color(dark);              // Make the down
button dark

    arduino.write(2);                               // Tell arduino that all of the
angles will be sent
    arduino.write(0*2+2);                           // Motor 1      // Send all of
the angles (multiply by 2 then add 2)
    arduino.write(0*2+2);                           // Motor 2      // Limit the
number of motors running due to power supply
    arduino.write(2*2+2);                           // Motor 3
    arduino.write(0*2+2);                           // Motor 4
    arduino.write(0*2+2);                           // Motor 5
    arduino.write(3*2+2);                           // Motor 6
    arduino.write(0*2+2);                           // Motor 7

    println("Normalized Motion from pt 1c to 2"); // Print in text
area below

    mpower[1-1].on=1;                               // Turn the on flag on

```



```

mpower[2-1].on=1;           // Turn the on flag on
mpower[3-1].on=1;           // Turn the on flag on
mpower[4-1].on=1;           // Turn the on flag on
mpower[5-1].on=1;           // Turn the on flag on
mpower[6-1].on=1;           // Turn the on flag on
mpower[7-1].on=1;           // Turn the on flag on
rectm1on.basecolor = color(light);    // Turn the on button light
rectm2on.basecolor = color(light);    // Turn the on button light
rectm3on.basecolor = color(light);    // Turn the on button light
rectm4on.basecolor = color(light);    // Turn the on button light
rectm5on.basecolor = color(light);    // Turn the on button light
rectm6on.basecolor = color(light);    // Turn the on button light
rectm7on.basecolor = color(light);    // Turn the on button light
rectm1off.basecolor = color(dark);    // Turn the off button dark
rectm2off.basecolor = color(dark);    // Turn the off button dark
rectm3off.basecolor = color(dark);    // Turn the off button dark
rectm4off.basecolor = color(dark);    // Turn the off button dark
rectm5off.basecolor = color(dark);    // Turn the off button dark
rectm6off.basecolor = color(dark);    // Turn the off button dark
rectm7off.basecolor = color(dark);    // Turn the off button dark
}
//-----
else if(rectpoint_2_2a.pressed()) {    // Increment all the
angles, normalized
    normalized = 1;                    // Normalized flag on
    rectnormalize.basecolor = color(light);    // Turn the button
light
    rectneutstart.basecolor = color(light);    // Turn the button
light
    rectpoint_1_1a.basecolor = color(light);    // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light
    rectpoint_1c_2.basecolor = color(light);    // Turn the button
light
    rectpoint_2_2a.basecolor = color(dark);    // Turn the button dark
    rectpoint_2a_2b.basecolor = color(light);    // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light);    // Turn the button
light
    rectpoint_2c_3.basecolor = color(light);    // Turn the button
light
    rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light);    // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light);    // Turn the button
light
    rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

    arduino.write(1);                    // Stop everything
    // Set all of the directions

```

```

        // Motor 1
        arduino.write(13);
to change direction to down
        mpower[1-1].dir = 0;
direction flag to down
        rectm1up.basecolor = color(light);
button light
        rectm1down.basecolor = color(dark);
button dark
        // Motor 2
        arduino.write(23);
to change direction to down
        mpower[2-1].dir = 0;
direction flag to down
        rectm2up.basecolor = color(light);
button light
        rectm2down.basecolor = color(dark);
button dark
        //Motor 3
        arduino.write(33);
to change direction to down
        mpower[3-1].dir = 0;
direction flag to down
        rectm3up.basecolor = color(light);
button light
        rectm3down.basecolor = color(dark);
button dark
        // Motor 4
        arduino.write(43);
to change direction to down
        mpower[4-1].dir = 0;
direction flag to down
        rectm4up.basecolor = color(light);
button light
        rectm4down.basecolor = color(dark);
button dark
        // Motor 5
        arduino.write(53);
to change direction to down
        mpower[5-1].dir = 0;
direction flag to down
        rectm5up.basecolor = color(light);
button light
        rectm5down.basecolor = color(dark);
button dark
        // Motor 6
        arduino.write(63);
to change direction to down
        mpower[6-1].dir = 0;
direction flag to down
        rectm6up.basecolor = color(light);
button light
        rectm6down.basecolor = color(dark);
button dark
        // Motor 7
        arduino.write(73);
to change direction to down

```

```

// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board
        // Turn the
        // Make the up
        // Make the down
// Tell the board

```

```

        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(50*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(25*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(0*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

        println("Normalized Motion from pt 2 to 2a"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on
        mpower[7-1].on=1; // Turn the on flag on
        rectm1on.basecolor = color(light); // Turn the on button light
        rectm2on.basecolor = color(light); // Turn the on button light
        rectm3on.basecolor = color(light); // Turn the on button light
        rectm4on.basecolor = color(light); // Turn the on button light
        rectm5on.basecolor = color(light); // Turn the on button light
        rectm6on.basecolor = color(light); // Turn the on button light
        rectm7on.basecolor = color(light); // Turn the on button light
        rectm1off.basecolor = color(dark); // Turn the off button dark
        rectm2off.basecolor = color(dark); // Turn the off button dark
        rectm3off.basecolor = color(dark); // Turn the off button dark
        rectm4off.basecolor = color(dark); // Turn the off button dark
        rectm5off.basecolor = color(dark); // Turn the off button dark
        rectm6off.basecolor = color(dark); // Turn the off button dark
        rectm7off.basecolor = color(dark); // Turn the off button dark
    }
    //-----
    else if(rectpoint_2a_2b.pressed()) { // Increment all the
angles, normalized
        normalized = 1; // Normalized flag on
        rectnormalize.basecolor = color(light); // Turn the button
light
        rectneutstart.basecolor = color(light); // Turn the button
light
        rectpoint_1_1a.basecolor = color(light); // Turn the button
light

```

```

    rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(dark); // Turn the button dark
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(light); // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
    rectpoint_3d_4.basecolor = color(light); // Turn the button
light

    arduino.write(1); // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13); // Tell the board
to change direction to down // Turn the
    mpower[1-1].dir = 0; // Make the up
direction flag to down // Make the down
    rectm1up.basecolor = color(light); // Make the up
button light // Make the down
    rectm1down.basecolor = color(dark); // Make the down
button dark
    // Motor 2
    arduino.write(23); // Tell the board
to change direction to down // Turn the
    mpower[2-1].dir = 0; // Make the up
direction flag to down // Make the down
    rectm2up.basecolor = color(light); // Make the up
button light // Make the down
    rectm2down.basecolor = color(dark); // Make the down
button dark
    // Motor 3
    arduino.write(33); // Tell the board
to change direction to down // Turn the
    mpower[3-1].dir = 0; // Make the up
direction flag to down // Make the down
    rectm3up.basecolor = color(light); // Make the up
button light // Make the down
    rectm3down.basecolor = color(dark); // Make the down
button dark
    // Motor 4
    arduino.write(43); // Tell the board
to change direction to down

```

```

        mpower[4-1].dir = 0; // Turn the
direction flag to down
        rectm4up.basecolor = color(light); // Make the up
button light
        rectm4down.basecolor = color(dark); // Make the down
button dark
        // Motor 5
        arduino.write(53); // Tell the board
to change direction to down
        mpower[5-1].dir = 0; // Turn the
direction flag to down
        rectm5up.basecolor = color(light); // Make the up
button light
        rectm5down.basecolor = color(dark); // Make the down
button dark
        // Motor 6
        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(72); // Tell the board
to change direction to up
        mpower[7-1].dir = 1; // Turn the
direction flag to up
        rectm7up.basecolor = color(dark); // Make the up
button dark
        rectm7down.basecolor = color(light); // Make the down
button light

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(35*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(0*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(0*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(34*2+2); // Motor 7

        println("Normalized Motion from pt 2a to 2b"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on

```

```

    mpower[7-1].on=1;           // Turn the on flag on
    rectm1on.basecolor = color(light);    // Turn the on button light
    rectm2on.basecolor = color(light);    // Turn the on button light
    rectm3on.basecolor = color(light);    // Turn the on button light
    rectm4on.basecolor = color(light);    // Turn the on button light
    rectm5on.basecolor = color(light);    // Turn the on button light
    rectm6on.basecolor = color(light);    // Turn the on button light
    rectm7on.basecolor = color(light);    // Turn the on button light
    rectm1off.basecolor = color(dark);    // Turn the off button dark
    rectm2off.basecolor = color(dark);    // Turn the off button dark
    rectm3off.basecolor = color(dark);    // Turn the off button dark
    rectm4off.basecolor = color(dark);    // Turn the off button dark
    rectm5off.basecolor = color(dark);    // Turn the off button dark
    rectm6off.basecolor = color(dark);    // Turn the off button dark
    rectm7off.basecolor = color(dark);    // Turn the off button dark
  }
  //-----
  else if(rectpoint_2b_2c.pressed()) {    // Increment all the
    angles, normalized
    normalized = 1;                        // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(light); // Turn the button
light
    rectpoint_1a_1b.basecolor = color(dark); // Turn the button dark
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(light); // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
    rectpoint_3d_4.basecolor = color(light); // Turn the button
light

    arduino.write(1);                // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13);                // Tell the board
to change direction to down
    mpower[1-1].dir = 0;              // Turn the
direction flag to down

```

```

        rectm1up.basecolor = color(light);           // Make the up
button light
        rectm1down.basecolor = color(dark);          // Make the down
button dark
        // Motor 2
        arduino.write(23);                           // Tell the board
to change direction to down
        mpower[2-1].dir = 0;                          // Turn the
direction flag to down
        rectm2up.basecolor = color(light);           // Make the up
button light
        rectm2down.basecolor = color(dark);          // Make the down
button dark
        //Motor 3
        arduino.write(33);                           // Tell the board
to change direction to down
        mpower[3-1].dir = 0;                          // Turn the
direction flag to down
        rectm3up.basecolor = color(light);           // Make the up
button light
        rectm3down.basecolor = color(dark);          // Make the down
button dark
        // Motor 4
        arduino.write(43);                           // Tell the board
to change direction to down
        mpower[4-1].dir = 0;                          // Turn the
direction flag to down
        rectm4up.basecolor = color(light);           // Make the up
button light
        rectm4down.basecolor = color(dark);          // Make the down
button dark
        // Motor 5
        arduino.write(52);                           // Tell the board
to change direction to up
        mpower[5-1].dir = 1;                          // Turn the
direction flag to up
        rectm5up.basecolor = color(dark);            // Make the up
button dark
        rectm5down.basecolor = color(light);          // Make the down
button light
        // Motor 6
        arduino.write(62);                           // Tell the board
to change direction to up
        mpower[6-1].dir = 1;                          // Turn the
direction flag to up
        rectm6up.basecolor = color(dark);            // Make the up
button dark
        rectm6down.basecolor = color(light);          // Make the down
button light
        // Motor 7
        arduino.write(73);                           // Tell the board
to change direction to down
        mpower[7-1].dir = 0;                          // Turn the
direction flag to down
        rectm7up.basecolor = color(light);           // Make the up
button light

```

```

        rectm7down.basecolor = color(dark);           // Make the down
button dark

        arduino.write(2);                             // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2);                         // Motor 1           // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2);                         // Motor 2           // Limit the
number of motors running due to power supply
        arduino.write(0*2+2);                         // Motor 3
        arduino.write(0*2+2);                         // Motor 4
        arduino.write(85*2+2);                       // Motor 5
        arduino.write(20*2+2);                       // Motor 6
        arduino.write(0*2+2);                         // Motor 7

        println("Normalized Motion from pt 2b to 2c"); // Print in text
area below

        mpower[1-1].on=1;                             // Turn the on flag on
        mpower[2-1].on=1;                             // Turn the on flag on
        mpower[3-1].on=1;                             // Turn the on flag on
        mpower[4-1].on=1;                             // Turn the on flag on
        mpower[5-1].on=1;                             // Turn the on flag on
        mpower[6-1].on=1;                             // Turn the on flag on
        mpower[7-1].on=1;                             // Turn the on flag on
        rectm1on.basecolor = color(light);             // Turn the on button light
        rectm2on.basecolor = color(light);             // Turn the on button light
        rectm3on.basecolor = color(light);             // Turn the on button light
        rectm4on.basecolor = color(light);             // Turn the on button light
        rectm5on.basecolor = color(light);             // Turn the on button light
        rectm6on.basecolor = color(light);             // Turn the on button light
        rectm7on.basecolor = color(light);             // Turn the on button light
        rectm1off.basecolor = color(dark);             // Turn the off button dark
        rectm2off.basecolor = color(dark);             // Turn the off button dark
        rectm3off.basecolor = color(dark);             // Turn the off button dark
        rectm4off.basecolor = color(dark);             // Turn the off button dark
        rectm5off.basecolor = color(dark);             // Turn the off button dark
        rectm6off.basecolor = color(dark);             // Turn the off button dark
        rectm7off.basecolor = color(dark);             // Turn the off button dark
    }
    //-----
    else if(rectpoint_2c_3.pressed()) {               // Increment all the
angles, normalized
        normalized = 1;                               // Normalized flag on
        rectnormalize.basecolor = color(light);        // Turn the button
light
        rectneutstart.basecolor = color(light);       // Turn the button
light
        rectpoint_1_1a.basecolor = color(light);     // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light

```



```

        rectpoint_1c_2.basecolor = color(light);    // Turn the button
light
        rectpoint_2_2a.basecolor = color(light);    // Turn the button
light
        rectpoint_2a_2b.basecolor = color(light);   // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light);   // Turn the button
light
        rectpoint_2c_3.basecolor = color(dark);     // Turn the button dark
        rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
        rectpoint_3a_3b.basecolor = color(light);   // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light);   // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light);   // Turn the button
light
        rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

        arduino.write(1);                          // Stop everything
        // Set all of the directions
        // Motor 1
        arduino.write(13);                          // Tell the board
to change direction to down
        mpower[1-1].dir = 0;                        // Turn the
direction flag to down
        rectm1up.basecolor = color(light);          // Make the up
button light
        rectm1down.basecolor = color(dark);         // Make the down
button dark
        // Motor 2
        arduino.write(23);                          // Tell the board
to change direction to down
        mpower[2-1].dir = 0;                        // Turn the
direction flag to down
        rectm2up.basecolor = color(light);          // Make the up
button light
        rectm2down.basecolor = color(dark);         // Make the down
button dark
        //Motor 3
        arduino.write(33);                          // Tell the board
to change direction to down
        mpower[3-1].dir = 0;                        // Turn the
direction flag to down
        rectm3up.basecolor = color(light);          // Make the up
button light
        rectm3down.basecolor = color(dark);         // Make the down
button dark
        // Motor 4
        arduino.write(43);                          // Tell the board
to change direction to down
        mpower[4-1].dir = 0;                        // Turn the
direction flag to down
        rectm4up.basecolor = color(light);          // Make the up
button light

```

```

        rectm4down.basecolor = color(dark);           // Make the down
button dark
        // Motor 5
        arduino.write(53);                           // Tell the board
to change direction to down
        mpower[5-1].dir = 0;                          // Turn the
direction flag to down
        rectm5up.basecolor = color(light);            // Make the up
button light
        rectm5down.basecolor = color(dark);           // Make the down
button dark
        // Motor 6
        arduino.write(63);                           // Tell the board
to change direction to down
        mpower[6-1].dir = 0;                          // Turn the
direction flag to down
        rectm6up.basecolor = color(light);            // Make the up
button light
        rectm6down.basecolor = color(dark);           // Make the down
button dark
        // Motor 7
        arduino.write(73);                           // Tell the board
to change direction to down
        mpower[7-1].dir = 0;                          // Turn the
direction flag to down
        rectm7up.basecolor = color(light);            // Make the up
button light
        rectm7down.basecolor = color(dark);           // Make the down
button dark

        arduino.write(2);                             // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2);                         // Motor 1           // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2);                         // Motor 2           // Limit the
number of motors running due to power supply
        arduino.write(0*2+2);                         // Motor 3
        arduino.write(1*2+2);                         // Motor 4
        arduino.write(0*2+2);                         // Motor 5
        arduino.write(3*2+2);                         // Motor 6
        arduino.write(0*2+2);                         // Motor 7

        println("Normalized Motion from pt 2c to 3"); // Print in text
area below

        mpower[1-1].on=1;                             // Turn the on flag on
        mpower[2-1].on=1;                             // Turn the on flag on
        mpower[3-1].on=1;                             // Turn the on flag on
        mpower[4-1].on=1;                             // Turn the on flag on
        mpower[5-1].on=1;                             // Turn the on flag on
        mpower[6-1].on=1;                             // Turn the on flag on
        mpower[7-1].on=1;                             // Turn the on flag on
        rectm1on.basecolor = color(light);             // Turn the on button light
        rectm2on.basecolor = color(light);             // Turn the on button light
        rectm3on.basecolor = color(light);             // Turn the on button light

```

```

    rectm4on.basecolor = color(light);    // Turn the on button light
    rectm5on.basecolor = color(light);    // Turn the on button light
    rectm6on.basecolor = color(light);    // Turn the on button light
    rectm7on.basecolor = color(light);    // Turn the on button light
    rectm1off.basecolor = color(dark);    // Turn the off button dark
    rectm2off.basecolor = color(dark);    // Turn the off button dark
    rectm3off.basecolor = color(dark);    // Turn the off button dark
    rectm4off.basecolor = color(dark);    // Turn the off button dark
    rectm5off.basecolor = color(dark);    // Turn the off button dark
    rectm6off.basecolor = color(dark);    // Turn the off button dark
    rectm7off.basecolor = color(dark);    // Turn the off button dark
  }
  //-----
  else if(rectpoint_3_3a.pressed()) {    // Increment all the
angles, normalized
    normalized = 1;                      // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(light); // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light
    rectpoint_3_3a.basecolor = color(dark); // Turn the button dark
    rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
    rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
    rectpoint_3d_4.basecolor = color(light); // Turn the button
light

    arduino.write(1);                    // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13);                    // Tell the board
to change direction to down
    mpower[1-1].dir = 0;                  // Turn the
direction flag to down
    rectmlup.basecolor = color(light);    // Make the up
button light
    rectmldown.basecolor = color(dark);   // Make the down
button dark

```

```

        // Motor 2
        arduino.write(23);
to change direction to down
        mpower[2-1].dir = 0;
direction flag to down
        rectm2up.basecolor = color(light);
button light
        rectm2down.basecolor = color(dark);
button dark
        //Motor 3
        arduino.write(33);
to change direction to down
        mpower[3-1].dir = 0;
direction flag to down
        rectm3up.basecolor = color(light);
button light
        rectm3down.basecolor = color(dark);
button dark
        // Motor 4
        arduino.write(43);
to change direction to down
        mpower[4-1].dir = 0;
direction flag to down
        rectm4up.basecolor = color(light);
button light
        rectm4down.basecolor = color(dark);
button dark
        // Motor 5
        arduino.write(53);
to change direction to down
        mpower[5-1].dir = 0;
direction flag to down
        rectm5up.basecolor = color(light);
button light
        rectm5down.basecolor = color(dark);
button dark
        // Motor 6
        arduino.write(63);
to change direction to down
        mpower[6-1].dir = 0;
direction flag to down
        rectm6up.basecolor = color(light);
button light
        rectm6down.basecolor = color(dark);
button dark
        // Motor 7
        arduino.write(73);
to change direction to down
        mpower[7-1].dir = 0;
direction flag to down
        rectm7up.basecolor = color(light);
button light
        rectm7down.basecolor = color(dark);
button dark

```

```

// Tell the board
// Turn the
// Make the up
// Make the down
// Tell the board
// Turn the
// Make the up
// Make the down
// Tell the board
// Turn the
// Make the up
// Make the down
// Tell the board
// Turn the
// Make the up
// Make the down
// Tell the board
// Turn the
// Make the up
// Make the down
// Tell the board
// Turn the
// Make the up
// Make the down

```

```

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(40*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(53*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(0*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

        println("Normalized Motion from pt 3 to 3a"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on
        mpower[7-1].on=1; // Turn the on flag on
        rectm1on.basecolor = color(light); // Turn the on button light
        rectm2on.basecolor = color(light); // Turn the on button light
        rectm3on.basecolor = color(light); // Turn the on button light
        rectm4on.basecolor = color(light); // Turn the on button light
        rectm5on.basecolor = color(light); // Turn the on button light
        rectm6on.basecolor = color(light); // Turn the on button light
        rectm7on.basecolor = color(light); // Turn the on button light
        rectm1off.basecolor = color(dark); // Turn the off button dark
        rectm2off.basecolor = color(dark); // Turn the off button dark
        rectm3off.basecolor = color(dark); // Turn the off button dark
        rectm4off.basecolor = color(dark); // Turn the off button dark
        rectm5off.basecolor = color(dark); // Turn the off button dark
        rectm6off.basecolor = color(dark); // Turn the off button dark
        rectm7off.basecolor = color(dark); // Turn the off button dark
    }
    //-----
    else if(rectpoint_3a_3b.pressed()) { // Increment all the
angles, normalized
        normalized = 1; // Normalized flag on
        rectnormalize.basecolor = color(light); // Turn the button
light
        rectneutstart.basecolor = color(light); // Turn the button
light
        rectpoint_1_1a.basecolor = color(light); // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
        rectpoint_1c_2.basecolor = color(light); // Turn the button
light
        rectpoint_2_2a.basecolor = color(light); // Turn the button
light

```

```

        rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
        rectpoint_2c_3.basecolor = color(light); // Turn the button
light
        rectpoint_3_3a.basecolor = color(light); // Turn the button
light
        rectpoint_3a_3b.basecolor = color(dark); // Turn the button dark
        rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
        rectpoint_3d_4.basecolor = color(light); // Turn the button
light

        arduino.write(1); // Stop everything
        // Set all of the directions
        // Motor 1
        arduino.write(13); // Tell the board
to change direction to down
        mpower[1-1].dir = 0; // Turn the
direction flag to down
        rectm1up.basecolor = color(light); // Make the up
button light
        rectm1down.basecolor = color(dark); // Make the down
button dark
        // Motor 2
        arduino.write(23); // Tell the board
to change direction to down
        mpower[2-1].dir = 0; // Turn the
direction flag to down
        rectm2up.basecolor = color(light); // Make the up
button light
        rectm2down.basecolor = color(dark); // Make the down
button dark
        //Motor 3
        arduino.write(33); // Tell the board
to change direction to down
        mpower[3-1].dir = 0; // Turn the
direction flag to down
        rectm3up.basecolor = color(light); // Make the up
button light
        rectm3down.basecolor = color(dark); // Make the down
button dark
        // Motor 4
        arduino.write(42); // Tell the board
to change direction to up
        mpower[4-1].dir = 1; // Turn the
direction flag to up
        rectm4up.basecolor = color(dark); // Make the up
button dark
        rectm4down.basecolor = color(light); // Make the down
button light
        // Motor 5
        arduino.write(53); // Tell the board
to change direction to down

```

```

        mpower[5-1].dir = 0; // Turn the
direction flag to down
        rectm5up.basecolor = color(light); // Make the up
button light
        rectm5down.basecolor = color(dark); // Make the down
button dark
        // Motor 6
        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(73); // Tell the board
to change direction to down
        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(15*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(0*2+2); // Motor 3
        arduino.write(40*2+2); // Motor 4
        arduino.write(0*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

        println("Normalized Motion from pt 3a to 3b"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on
        mpower[7-1].on=1; // Turn the on flag on
        rectm1on.basecolor = color(light); // Turn the on button light
        rectm2on.basecolor = color(light); // Turn the on button light
        rectm3on.basecolor = color(light); // Turn the on button light
        rectm4on.basecolor = color(light); // Turn the on button light
        rectm5on.basecolor = color(light); // Turn the on button light
        rectm6on.basecolor = color(light); // Turn the on button light
        rectm7on.basecolor = color(light); // Turn the on button light
        rectm1off.basecolor = color(dark); // Turn the off button dark

```

```

    rectm2off.basecolor = color(dark);    // Turn the off button dark
    rectm3off.basecolor = color(dark);    // Turn the off button dark
    rectm4off.basecolor = color(dark);    // Turn the off button dark
    rectm5off.basecolor = color(dark);    // Turn the off button dark
    rectm6off.basecolor = color(dark);    // Turn the off button dark
    rectm7off.basecolor = color(dark);    // Turn the off button dark
}
//-----
else if(rectpoint_3b_3c.pressed()) {    // Increment all the
angles, normalized
    normalized = 1;                      // Normalized flag on
    rectnormalize.basecolor = color(light);    // Turn the button
light
    rectneutstart.basecolor = color(light);    // Turn the button
light
    rectpoint_1_1a.basecolor = color(light);    // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light);    // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light);    // Turn the button
light
    rectpoint_1c_2.basecolor = color(light);    // Turn the button
light
    rectpoint_2_2a.basecolor = color(light);    // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light);    // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light);    // Turn the button
light
    rectpoint_2c_3.basecolor = color(light);    // Turn the button
light
    rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
    rectpoint_3b_3c.basecolor = color(dark);    // Turn the button dark
    rectpoint_3c_3d.basecolor = color(light);    // Turn the button
light
    rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

    arduino.write(1);                      // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(12);                      // Tell the board
to change direction to up
    mpower[1-1].dir = 1;                      // Turn the
direction flag to up
    rectmlup.basecolor = color(dark);          // Make the up
button dark
    rectmldown.basecolor = color(light);        // Make the down
button light
    // Motor 2
    arduino.write(23);                      // Tell the board
to change direction to down
    mpower[2-1].dir = 0;                      // Turn the
direction flag to down

```



```

        rectm2up.basecolor = color(light);           // Make the up
button light
        rectm2down.basecolor = color(dark);          // Make the down
button dark
        //Motor 3
        arduino.write(33);                           // Tell the board
to change direction to down
        mpower[3-1].dir = 0;                           // Turn the
direction flag to down
        rectm3up.basecolor = color(light);            // Make the up
button light
        rectm3down.basecolor = color(dark);           // Make the down
button dark
        // Motor 4
        arduino.write(43);                           // Tell the board
to change direction to down
        mpower[4-1].dir = 0;                           // Turn the
direction flag to down
        rectm4up.basecolor = color(light);            // Make the up
button light
        rectm4down.basecolor = color(dark);           // Make the down
button dark
        // Motor 5
        arduino.write(53);                           // Tell the board
to change direction to down
        mpower[5-1].dir = 0;                           // Turn the
direction flag to down
        rectm5up.basecolor = color(light);            // Make the up
button light
        rectm5down.basecolor = color(dark);           // Make the down
button dark
        // Motor 6
        arduino.write(63);                           // Tell the board
to change direction to down
        mpower[6-1].dir = 0;                           // Turn the
direction flag to down
        rectm6up.basecolor = color(light);            // Make the up
button light
        rectm6down.basecolor = color(dark);           // Make the down
button dark
        // Motor 7
        arduino.write(73);                           // Tell the board
to change direction to down
        mpower[7-1].dir = 0;                           // Turn the
direction flag to down
        rectm7up.basecolor = color(light);            // Make the up
button light
        rectm7down.basecolor = color(dark);           // Make the down
button dark

        arduino.write(2);                             // Tell arduino that all of the
angles will be sent
        arduino.write(15*2+2);                       // Motor 1           // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2);                       // Motor 2           // Limit the
number of motors running due to power supply

```

```

    arduino.write(0*2+2);          // Motor 3
    arduino.write(0*2+2);          // Motor 4
    arduino.write(0*2+2);          // Motor 5
    arduino.write(0*2+2);          // Motor 6
    arduino.write(53*2+2);         // Motor 7

    println("Normalized Motion from pt 3b to 3c"); // Print in text
area below

    mpower[1-1].on=1;              // Turn the on flag on
    mpower[2-1].on=1;              // Turn the on flag on
    mpower[3-1].on=1;              // Turn the on flag on
    mpower[4-1].on=1;              // Turn the on flag on
    mpower[5-1].on=1;              // Turn the on flag on
    mpower[6-1].on=1;              // Turn the on flag on
    mpower[7-1].on=1;              // Turn the on flag on
    rectm1on.basecolor = color(light); // Turn the on button light
    rectm2on.basecolor = color(light); // Turn the on button light
    rectm3on.basecolor = color(light); // Turn the on button light
    rectm4on.basecolor = color(light); // Turn the on button light
    rectm5on.basecolor = color(light); // Turn the on button light
    rectm6on.basecolor = color(light); // Turn the on button light
    rectm7on.basecolor = color(light); // Turn the on button light
    rectm1off.basecolor = color(dark); // Turn the off button dark
    rectm2off.basecolor = color(dark); // Turn the off button dark
    rectm3off.basecolor = color(dark); // Turn the off button dark
    rectm4off.basecolor = color(dark); // Turn the off button dark
    rectm5off.basecolor = color(dark); // Turn the off button dark
    rectm6off.basecolor = color(dark); // Turn the off button dark
    rectm7off.basecolor = color(dark); // Turn the off button dark
}
//-----
else if(rectpoint_3c_3d.pressed()) { // Increment all the
angles, normalized
    normalized = 1;                // Normalized flag on
    rectnormalize.basecolor = color(light); // Turn the button
light
    rectneutstart.basecolor = color(light); // Turn the button
light
    rectpoint_1_1a.basecolor = color(light); // Turn the button
light
    rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
    rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
    rectpoint_1c_2.basecolor = color(light); // Turn the button
light
    rectpoint_2_2a.basecolor = color(light); // Turn the button
light
    rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
    rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
    rectpoint_2c_3.basecolor = color(light); // Turn the button
light

```

```

    rectpoint_3_3a.basecolor = color(light);    // Turn the button
light
    rectpoint_3a_3b.basecolor = color(light);    // Turn the button
light
    rectpoint_3b_3c.basecolor = color(light);    // Turn the button
light
    rectpoint_3c_3d.basecolor = color(dark);    // Turn the button dark
    rectpoint_3d_4.basecolor = color(light);    // Turn the button
light

    arduino.write(1);                            // Stop everything
    // Set all of the directions
    // Motor 1
    arduino.write(13);                            // Tell the board
to change direction to down
    mpower[1-1].dir = 0;                            // Turn the
direction flag to down
    rectm1up.basecolor = color(light);            // Make the up
button light
    rectm1down.basecolor = color(dark);          // Make the down
button dark
    // Motor 2
    arduino.write(23);                            // Tell the board
to change direction to down
    mpower[2-1].dir = 0;                            // Turn the
direction flag to down
    rectm2up.basecolor = color(light);            // Make the up
button light
    rectm2down.basecolor = color(dark);          // Make the down
button dark
    //Motor 3
    arduino.write(33);                            // Tell the board
to change direction to down
    mpower[3-1].dir = 0;                            // Turn the
direction flag to down
    rectm3up.basecolor = color(light);            // Make the up
button light
    rectm3down.basecolor = color(dark);          // Make the down
button dark
    // Motor 4
    arduino.write(43);                            // Tell the board
to change direction to down
    mpower[4-1].dir = 0;                            // Turn the
direction flag to down
    rectm4up.basecolor = color(light);            // Make the up
button light
    rectm4down.basecolor = color(dark);          // Make the down
button dark
    // Motor 5
    arduino.write(52);                            // Tell the board
to change direction to up
    mpower[5-1].dir = 1;                            // Turn the
direction flag to up
    rectm5up.basecolor = color(dark);            // Make the up
button dark
    rectm5down.basecolor = color(light);          // Make the down
button light

```

```

        // Motor 6
        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(73); // Tell the board
to change direction to down
        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(0*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(3*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(0*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(2*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

        println("Normalized Motion from pt 3c to 3d"); // Print in text
area below

        mpower[1-1].on=1; // Turn the on flag on
        mpower[2-1].on=1; // Turn the on flag on
        mpower[3-1].on=1; // Turn the on flag on
        mpower[4-1].on=1; // Turn the on flag on
        mpower[5-1].on=1; // Turn the on flag on
        mpower[6-1].on=1; // Turn the on flag on
        mpower[7-1].on=1; // Turn the on flag on
        rectm1on.basecolor = color(light); // Turn the on button light
        rectm2on.basecolor = color(light); // Turn the on button light
        rectm3on.basecolor = color(light); // Turn the on button light
        rectm4on.basecolor = color(light); // Turn the on button light
        rectm5on.basecolor = color(light); // Turn the on button light
        rectm6on.basecolor = color(light); // Turn the on button light
        rectm7on.basecolor = color(light); // Turn the on button light
        rectm1off.basecolor = color(dark); // Turn the off button dark
        rectm2off.basecolor = color(dark); // Turn the off button dark
        rectm3off.basecolor = color(dark); // Turn the off button dark
        rectm4off.basecolor = color(dark); // Turn the off button dark
        rectm5off.basecolor = color(dark); // Turn the off button dark
        rectm6off.basecolor = color(dark); // Turn the off button dark
        rectm7off.basecolor = color(dark); // Turn the off button dark

```

```

    }
    //-----
    else if(rectpoint_3d_4.pressed()) { // Increment all the
angles, normalized
        normalized = 1; // Normalized flag on
        rectnormalize.basecolor = color(light); // Turn the button
light
        rectneutstart.basecolor = color(light); // Turn the button
light
        rectpoint_1_1a.basecolor = color(light); // Turn the button
light
        rectpoint_1a_1b.basecolor = color(light); // Turn the button
light
        rectpoint_1b_1c.basecolor = color(light); // Turn the button
light
        rectpoint_1c_2.basecolor = color(light); // Turn the button
light
        rectpoint_2_2a.basecolor = color(light); // Turn the button
light
        rectpoint_2a_2b.basecolor = color(light); // Turn the button
light
        rectpoint_2b_2c.basecolor = color(light); // Turn the button
light
        rectpoint_2c_3.basecolor = color(light); // Turn the button
light
        rectpoint_3_3a.basecolor = color(light); // Turn the button
light
        rectpoint_3a_3b.basecolor = color(light); // Turn the button
light
        rectpoint_3b_3c.basecolor = color(light); // Turn the button
light
        rectpoint_3c_3d.basecolor = color(light); // Turn the button
light
        rectpoint_3d_4.basecolor = color(dark); // Turn the button dark

        arduino.write(1); // Stop everything
        // Set all of the directions
        // Motor 1
        arduino.write(12); // Tell the board
to change direction to up
        mpower[1-1].dir = 1; // Turn the
direction flag to up
        rectmlup.basecolor = color(dark); // Make the up
button dark
        rectmldown.basecolor = color(light); // Make the down
button light
        // Motor 2
        arduino.write(23); // Tell the board
to change direction to down
        mpower[2-1].dir = 0; // Turn the
direction flag to down
        rectm2up.basecolor = color(light); // Make the up
button light
        rectm2down.basecolor = color(dark); // Make the down
button dark
        //Motor 3

```

```

        arduino.write(33); // Tell the board
to change direction to down
        mpower[3-1].dir = 0; // Turn the
direction flag to down
        rectm3up.basecolor = color(light); // Make the up
button light
        rectm3down.basecolor = color(dark); // Make the down
button dark
        // Motor 4
        arduino.write(43); // Tell the board
to change direction to down
        mpower[4-1].dir = 0; // Turn the
direction flag to down
        rectm4up.basecolor = color(light); // Make the up
button light
        rectm4down.basecolor = color(dark); // Make the down
button dark
        // Motor 5
        arduino.write(52); // Tell the board
to change direction to up
        mpower[5-1].dir = 1; // Turn the
direction flag to up
        rectm5up.basecolor = color(dark); // Make the up
button dark
        rectm5down.basecolor = color(light); // Make the down
button light
        // Motor 6
        arduino.write(63); // Tell the board
to change direction to down
        mpower[6-1].dir = 0; // Turn the
direction flag to down
        rectm6up.basecolor = color(light); // Make the up
button light
        rectm6down.basecolor = color(dark); // Make the down
button dark
        // Motor 7
        arduino.write(73); // Tell the board
to change direction to down
        mpower[7-1].dir = 0; // Turn the
direction flag to down
        rectm7up.basecolor = color(light); // Make the up
button light
        rectm7down.basecolor = color(dark); // Make the down
button dark

        arduino.write(2); // Tell arduino that all of the
angles will be sent
        arduino.write(2*2+2); // Motor 1 // Send all of
the angles (multiply by 2 then add 2)
        arduino.write(0*2+2); // Motor 2 // Limit the
number of motors running due to power supply
        arduino.write(0*2+2); // Motor 3
        arduino.write(0*2+2); // Motor 4
        arduino.write(1*2+2); // Motor 5
        arduino.write(0*2+2); // Motor 6
        arduino.write(0*2+2); // Motor 7

```

```
println("Normalized Motion from pt 3d to 4"); // Print in text
area below
```

```

mpower[1-1].on=1;           // Turn the on flag on
mpower[2-1].on=1;           // Turn the on flag on
mpower[3-1].on=1;           // Turn the on flag on
mpower[4-1].on=1;           // Turn the on flag on
mpower[5-1].on=1;           // Turn the on flag on
mpower[6-1].on=1;           // Turn the on flag on
mpower[7-1].on=1;           // Turn the on flag on
rectm1on.basecolor = color(light); // Turn the on button light
rectm2on.basecolor = color(light); // Turn the on button light
rectm3on.basecolor = color(light); // Turn the on button light
rectm4on.basecolor = color(light); // Turn the on button light
rectm5on.basecolor = color(light); // Turn the on button light
rectm6on.basecolor = color(light); // Turn the on button light
rectm7on.basecolor = color(light); // Turn the on button light
rectm1off.basecolor = color(dark); // Turn the off button dark
rectm2off.basecolor = color(dark); // Turn the off button dark
rectm3off.basecolor = color(dark); // Turn the off button dark
rectm4off.basecolor = color(dark); // Turn the off button dark
rectm5off.basecolor = color(dark); // Turn the off button dark
rectm6off.basecolor = color(dark); // Turn the off button dark
rectm7off.basecolor = color(dark); // Turn the off button dark
    }
}
}

//-----CLASSES-----
// Most of this code is from the example for buttons in Processing
class Button
{
    int x, y;
    int size;
    color basecolor, highlightcolor;
    color currentcolor;
    boolean over = false;
    boolean pressed = false;

    void update() // Update function for
buttons
    {
        if((over())&&(click == 0)) { // When the mouse is over
the button and the mouse has not been clicked
            currentcolor = highlightcolor; // highlight the button
        }
        else {
            currentcolor = basecolor; // return button to
unclicked color
        }
    }

    boolean pressed() // Pressed function for
buttons

```

```

{
    if(over) {
        locked = true;
        return true;
    }
    else {
        locked = false;
        return false;
    }
}

boolean over()
{
    return true;
}

boolean overRect(int x, int y, int width, int height) // mouse over
the button
{
    if (mouseX >= x && mouseX <= x+width &&
        mouseY >= y && mouseY <= y+height) {
        return true;
    }
    else {
        return false;
    }
}

}

class RectButton extends Button
{
    RectButton(int ix, int iy, int isize, color icolor, color ihighlight)
    {
        x = ix;
        y = iy;
        size = isize;
        basecolor = icolor;
        highlightcolor = ihighlight;
        currentcolor = basecolor;
    }

    boolean over()
    {
        if( overRect(x, y, size, size) ) {
            over = true;
            return true;
        }
        else {
            over = false;
            return false;
        }
    }

    void display()

```



```

    {
        stroke(255);
        fill(currentcolor);
        rect(x, y, size, size);
    }
}
void mouseReleased()
{
    click = 0; //added so that multiple buttons will not be
clicked
    for(int i=0; i<num; i++) {
        mpower[i].release();
        mangle[i].release();
    }
}
//-----Scrolling bars-----
// Most of this code is from the example for scrolling bars in
Processing
class Handle
{
    int x, y;
    int boxx, boxy;
    int motor;
    int on, power;
    int state;
    int dir;
    int angle;
    int length;
    int size;
    boolean over;
    boolean press;
    boolean locked = false;
    boolean otherslocked = false;
    Handle[] others;

    Handle(int ix, int iy, int il, int is, int imotor, int ion, int
ipower, int istate, int idir, int iangle, Handle[] o)
    {
        x = ix;
        y = iy;
        motor = imotor;
        on = ion;
        power = ipower;
        state = istate;
        dir = idir;
        angle = iangle;
        length = il;
        size = is;
        boxx = x+length - size/2;
        boxy = y - size/2;
        others = o;
    }

    void update()
    {
        boxx = x+length;
        boxy = y - size/2;

```

```

for(int i=0; i<others.length; i++) {
    if(others[i].locked == true) {
        otherslocked = true;
        break;
    } else {
        otherslocked = false;
    }
}

if(otherslocked == false) {
    over();
    press();
}

if(press) {
    if(state == 0) { // If power scroll bar
        length = lock(mouseX-mpowerx-size/2, 0, 100); //length for
power
        power = length*255/100;
        if ((on == 1)&&(normalized == 0)) {
            arduino.write((motor*10)+4); // Tell the MCU that the power
value is coming
            arduino.write(power); // Send the power value
            arduino.write((motor*10)+6); // Turn the motor on, this is
necessary
// to apply the new value to
the motor
        }
    }
    else if(state == 1) { // If angle scroll bar
        length = lock(mouseX-manglex-size/2, 0, 100); // length for
angle
        angle = length; // Set the angle value
        mpower[motor-1].angle = angle; // Set the motor's angle
    }
}

void over()
{
    if(overRect(boxx, boxy, size, size)) {
        over = true;
    } else {
        over = false;
    }
}

void press()
{
    if(over && mousePressed || locked) {
        press = true;
        locked = true;
    } else {
        press = false;
    }
}

```

```

void release()
{
    locked = false;
}

void display()
{
    fill(255);
    stroke(0);
    line(x, y, x+100+size, y);
    rect(boxx, boxy, size, size);
    if(over || press) {
        line(boxx, boxy, boxx+size, boxy+size);
        line(boxx, boxy+size, boxx+size, boxy);
    }
    fill(0);
    if(state == 0) {
        text(length, scrollx, y+5);           //display the value of the
        scrolled box
    }
    if(state == 1) {
        float ang_deg = float(length);       // Convert the length into a
        float for display.
        text(str(ang_deg/2), scrollx2, y+5);   // Display the angle
        value ranging from 0-50 degrees
                                           // use str() to limit the
        decimals displayed
    }
}

boolean overRect(int x, int y, int width, int height)
{
    if (mouseX >= x && mouseX <= x+width &&
        mouseY >= y && mouseY <= y+height) {
        return true;
    } else {
        return false;
    }
}

int lock(int val, int minv, int maxv)
{
    return min(max(val, minv), maxv);
}

//-----keyboard-----
void keyPressed() {    // Function in Processing language
    keyboard = key;     // Get the value of the key pressed and set it
    variable
    if(keyboard == ' ') { // Check if spacebar was pressed to stop
    everything
        arduino.write(1); // Tell MCU to stop the motors
        for(int i=1; i<(num+1); i++) { // Turn motor flags off
            mpower[i-1].on=0;
        }
    }
}

```

```

// Set buttons to appropriate color
rectm1on.basecolor = color(light);
rectm1off.basecolor = color(dark);
rectm2on.basecolor = color(light);
rectm2off.basecolor = color(dark);
rectm3on.basecolor = color(light);
rectm3off.basecolor = color(dark);
rectm4on.basecolor = color(light);
rectm4off.basecolor = color(dark);
rectm5on.basecolor = color(light);
rectm5off.basecolor = color(dark);
rectm6on.basecolor = color(light);
rectm6off.basecolor = color(dark);
rectm7on.basecolor = color(light);
rectm7off.basecolor = color(dark);

rectm1angle.basecolor = color (light);
rectm2angle.basecolor = color (light);
rectm3angle.basecolor = color (light);
rectm4angle.basecolor = color (light);
rectm5angle.basecolor = color (light);
rectm6angle.basecolor = color (light);
rectm7angle.basecolor = color (light);

rectg1open.basecolor = color(light);
rectg2open.basecolor = color(light);
rectg1off.basecolor = color(dark);
rectg2off.basecolor = color(dark);
rectg1close.basecolor = color(light);
rectg2close.basecolor = color(light);

normalized = 0; // Normalized flag off
}
//-----Toggle on/off-----
else if (keyboard == '1') { // Toggle the motor 1 on/off
  if (mpower[1-1].on == 1) { // If on, then turn off
    arduino.write(10); // Tell MCU to turn motor off
    mpower[1-1].on = 0; // Turn motor flag off
    //Set buttons to appropriate color
    rectm1on.basecolor = color(light);
    rectm1off.basecolor = color(dark);
    rectm1angle.basecolor = color(light);
  }
  else { // If off, then turn on
    arduino.write(14); // Tell MCU that the power value is
coming arduino.write(mpower[1-1].power); // Send the power
    arduino.write(11); // Tell MCU to turn motor on
    mpower[1-1].on = 1; // Turn motor flag on
    // Set buttons to appropriate color
    rectm1on.basecolor = color(dark);
    rectm1off.basecolor = color(light);
  }
}
else if (keyboard == '2') { // Toggle the motor 2 on/off
  if (mpower[2-1].on == 1) { // If on, then turn off
    arduino.write(20); // Tell MCU to turn motor off

```

```

        mpower[2-1].on = 0;          // Turn motor flag off
        //Set buttons to appropriate color
        rectm2on.basecolor = color(light);
        rectm2off.basecolor = color(dark);
        rectm2angle.basecolor = color(light);
    }
    else {                          // If off, then turn on
        coming arduino.write(24);    // Tell MCU that the power value is
        coming arduino.write(mpower[2-1].power); // Send the power
        arduino.write(21);          // Tell MCU to turn motor on
        mpower[2-1].on = 1;         // Turn motor flag on
        //Set buttons to appropriate color
        rectm2on.basecolor = color(dark);
        rectm2off.basecolor = color(light);
    }
}
else if (keyboard == '3') { // Toggle the motor 3 on/off
    if (mpower[3-1].on == 1) { // If on, then turn off
        arduino.write(30);    // Tell MCU to turn motor off
        mpower[3-1].on = 0;    // Turn motor flag off
        //Set buttons to appropriate color
        rectm3on.basecolor = color(light);
        rectm3off.basecolor = color(dark);
        rectm3angle.basecolor = color(light);
    }
    else {                      // If off, then turn on
        coming arduino.write(34); // Tell MCU that the power value is
        coming arduino.write(mpower[3-1].power); // Send the power
        arduino.write(31);    // Tell MCU to turn motor on
        mpower[3-1].on = 1;    // Turn motor flag on
        //Set buttons to appropriate color
        rectm3on.basecolor = color(dark);
        rectm3off.basecolor = color(light);
    }
}
else if (keyboard == '4') { // Toggle the motor 4 on/off
    if (mpower[4-1].on == 1) { // If on, then turn off
        arduino.write(40);    // Tell MCU to turn motor off
        mpower[4-1].on = 0;    // Turn motor flag off
        //Set buttons to appropriate color
        rectm4on.basecolor = color(light);
        rectm4off.basecolor = color(dark);
        rectm4angle.basecolor = color(light);
    }
    else {                      // If off, then turn on
        coming arduino.write(44); // Tell MCU that the power value is
        coming arduino.write(mpower[4-1].power); // Send the power
        arduino.write(41);    // Tell MCU to turn motor on
        mpower[4-1].on = 1;    // Turn motor flag on
        //Set buttons to appropriate color
        rectm4on.basecolor = color(dark);
        rectm4off.basecolor = color(light);
    }
}
}
}

```

```

else if (keyboard == '5') { // Toggle the motor 5 on/off
  if (mpower[5-1].on == 1) { // If on, then turn off
    arduino.write(50); // Tell MCU to turn motor off
    mpower[5-1].on = 0; // Turn motor flag off
    //Set buttons to appropriate color
    rectm5on.basecolor = color(light);
    rectm5off.basecolor = color(dark);
    rectm5angle.basecolor = color(light);
  }
  else { // If off, then turn on
    coming arduino.write(54); // Tell MCU that the power value is
    arduino.write(mpower[5-1].power); // Send the power
    arduino.write(51); // Tell MCU to turn motor on
    mpower[5-1].on = 1; // Turn motor flag on
    //Set buttons to appropriate color
    rectm5on.basecolor = color(dark);
    rectm5off.basecolor = color(light);
  }
}
else if (keyboard == '6') { // Toggle the motor 6 on/off
  if (mpower[6-1].on == 1) { // If on, then turn off
    arduino.write(60); // Tell MCU to turn motor off
    mpower[6-1].on = 0; // Turn motor flag off
    //Set buttons to appropriate color
    rectm6on.basecolor = color(light);
    rectm6off.basecolor = color(dark);
    rectm6angle.basecolor = color(light);
  }
  else { // If off, then turn on
    coming arduino.write(64); // Tell MCU that the power value is
    arduino.write(mpower[6-1].power); // Send the power
    arduino.write(61); // Tell MCU to turn motor on
    mpower[6-1].on = 1; // Turn motor flag on
    //Set buttons to appropriate color
    rectm6on.basecolor = color(dark);
    rectm6off.basecolor = color(light);
  }
}
else if (keyboard == '7') { // Toggle the motor 7 on/off
  if (mpower[7-1].on == 1) { // If on, then turn off
    arduino.write(70); // Tell MCU to turn motor off
    mpower[7-1].on = 0; // Turn motor flag off
    //Set buttons to appropriate color
    rectm7on.basecolor = color(light);
    rectm7off.basecolor = color(dark);
    rectm7angle.basecolor = color(light);
  }
  else { // If off, then turn on
    coming arduino.write(74); // Tell MCU that the power value is
    arduino.write(mpower[7-1].power); // Send the power
    arduino.write(71); // Tell MCU to turn motor on
    mpower[7-1].on = 1; // Turn motor flag on
    //Set buttons to appropriate color
    rectm7on.basecolor = color(dark);

```

```

        rectm7off.basecolor = color(light);
    }
}

//-----Toggle directions-----
else if (keyboard == '!') {    // Toggle motor 1 up/down
    if (mpower[1-1].dir == 1) { // If up, then change to down
        arduino.write(13);      // Tell MCU to change direction to
down
        mpower[1-1].dir = 0;      // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm1up.basecolor = color(light);
        rectm1down.basecolor = color(dark);
    }
    else {                      // If down, then change to up
        arduino.write(12);      // Tell MCU to change direction to
up
        mpower[1-1].dir = 1;      // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm1up.basecolor = color(dark);
        rectm1down.basecolor = color(light);
    }
}
else if (keyboard == '@') {    // Toggle motor 2 up/down
    if (mpower[2-1].dir == 1) { // If up, then change to down
        arduino.write(23);      // Tell MCU to change direction to
down
        mpower[2-1].dir = 0;      // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm2up.basecolor = color(light);
        rectm2down.basecolor = color(dark);
    }
    else {                      // If down, then change to up
        arduino.write(22);      // Tell MCU to change direction to
up
        mpower[2-1].dir = 1;      // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm2up.basecolor = color(dark);
        rectm2down.basecolor = color(light);
    }
}
else if (keyboard == '#') {    // Toggle motor 3 up/down
    if (mpower[3-1].dir == 1) { // If up, then change to down
        arduino.write(33);      // Tell MCU to change direction to
down
        mpower[3-1].dir = 0;      // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm3up.basecolor = color(light);
        rectm3down.basecolor = color(dark);
    }
    else {                      // If down, then change to up
        arduino.write(32);      // Tell MCU to change direction to
up
        mpower[3-1].dir = 1;      // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm3up.basecolor = color(dark);
        rectm3down.basecolor = color(light);
    }
}

```

```

    }
}
else if (keyboard == '$') { // Toggle motor 4 up/down
    if (mpower[4-1].dir == 1) { // If up, then change to down
        arduino.write(43); // Tell MCU to change direction to
down
        mpower[4-1].dir = 0; // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm4up.basecolor = color(light);
        rectm4down.basecolor = color(dark);
    }
    else { // If down, then change to up
        arduino.write(42); // Tell MCU to change direction to
up
        mpower[4-1].dir = 1; // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm4up.basecolor = color(dark);
        rectm4down.basecolor = color(light);
    }
}
else if (keyboard == '%') { // Toggle motor 5 up/down
    if (mpower[5-1].dir == 1) { // If up, then change to down
        arduino.write(53); // Tell MCU to change direction to
down
        mpower[5-1].dir = 0; // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm5up.basecolor = color(light);
        rectm5down.basecolor = color(dark);
    }
    else { // If down, then change to up
        arduino.write(52); // Tell MCU to change direction to
up
        mpower[5-1].dir = 1; // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm5up.basecolor = color(dark);
        rectm5down.basecolor = color(light);
    }
}
else if (keyboard == '^') { // Toggle motor 6 up/down
    if (mpower[6-1].dir == 1) { // If up, then change to down
        arduino.write(63); // Tell MCU to change direction to
down
        mpower[6-1].dir = 0; // Set motor direction flag to down
        //Set buttons to appropriate color
        rectm6up.basecolor = color(light);
        rectm6down.basecolor = color(dark);
    }
    else { // If down, then change to up
        arduino.write(62); // Tell MCU to change direction to
up
        mpower[6-1].dir = 1; // Set motor direction flag to up
        //Set buttons to appropriate color
        rectm6up.basecolor = color(dark);
        rectm6down.basecolor = color(light);
    }
}
else if (keyboard == '&') { // Toggle motor 7 up/down

```



```

        if (mpower[7-1].dir == 1) { // If up, then change to down
            arduino.write(73);      // Tell MCU to change direction to
down
            mpower[7-1].dir = 0;    // Set motor direction flag to down
            //Set buttons to appropriate color
            rectm7up.basecolor = color(light);
            rectm7down.basecolor = color(dark);
        }
        else {                      // If down, then change to up
            arduino.write(72);      // Tell MCU to change direction to
up
            mpower[7-1].dir = 1;    // Set motor direction flag to up
            //Set buttons to appropriate color
            rectm7up.basecolor = color(dark);
            rectm7down.basecolor = color(light);
        }
    }
}

```