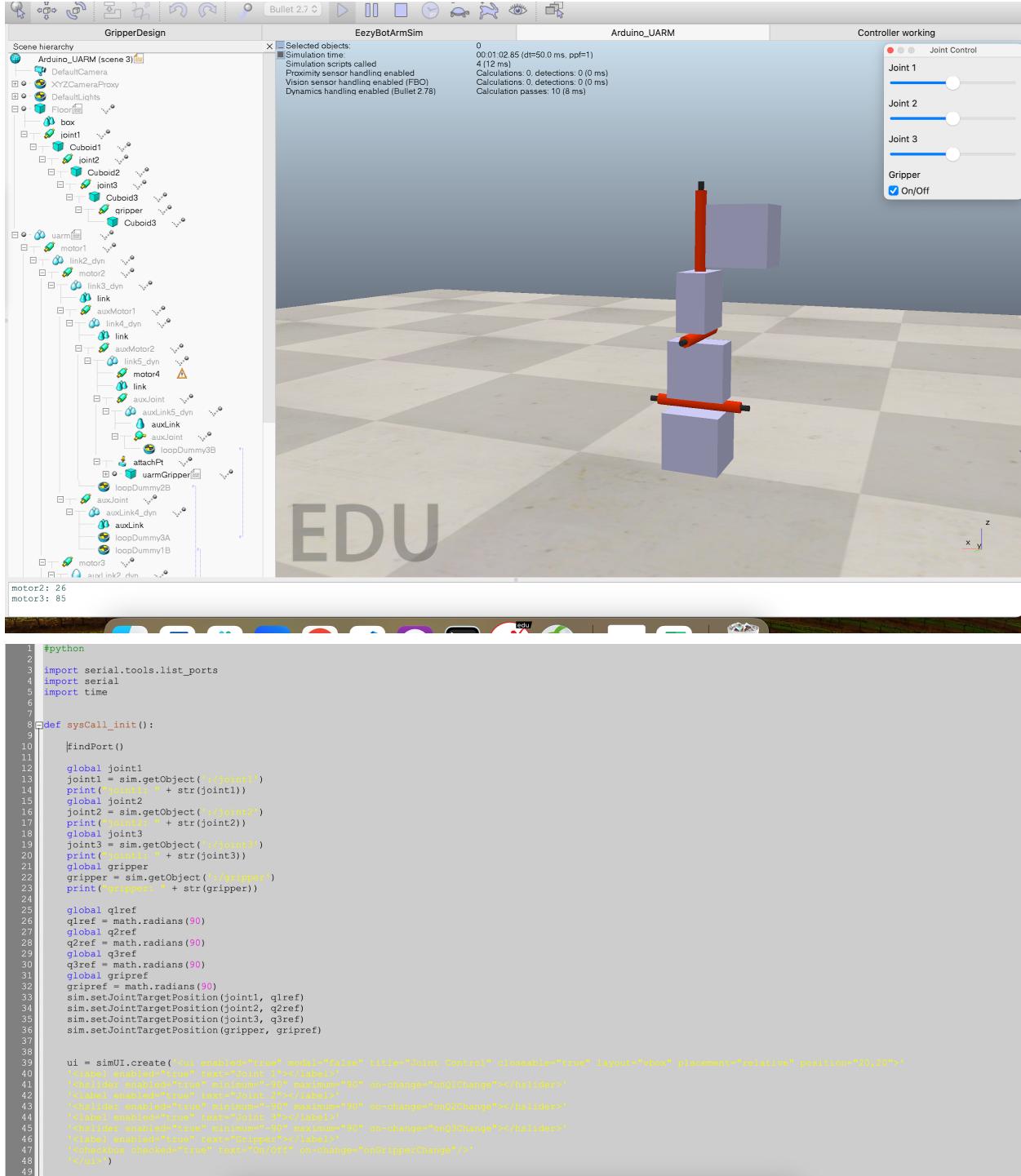


## **Data Collection**

To collect data as SCRAP was developed, a Google spreadsheet was used to record various observations throughout each iteration. The spreadsheet helped to organize what successes and failures throughout each step of my design process. The final spreadsheet is included below, followed by significant iterations. For each of the seven significant iterations, a description of the achieved function at that point of the project is given, and an explanation of why that iteration was important is provided. Also, pictures of SCRAP at each iteration are included.

**Iteration # 2:** A model made of 4 blocks with 4 joints controlled by sliders prints the position of each joint continuously to the terminal in the Simulation when a joint is changed. This iteration was important as it was the first model that achieved motion control with the sliders successfully.

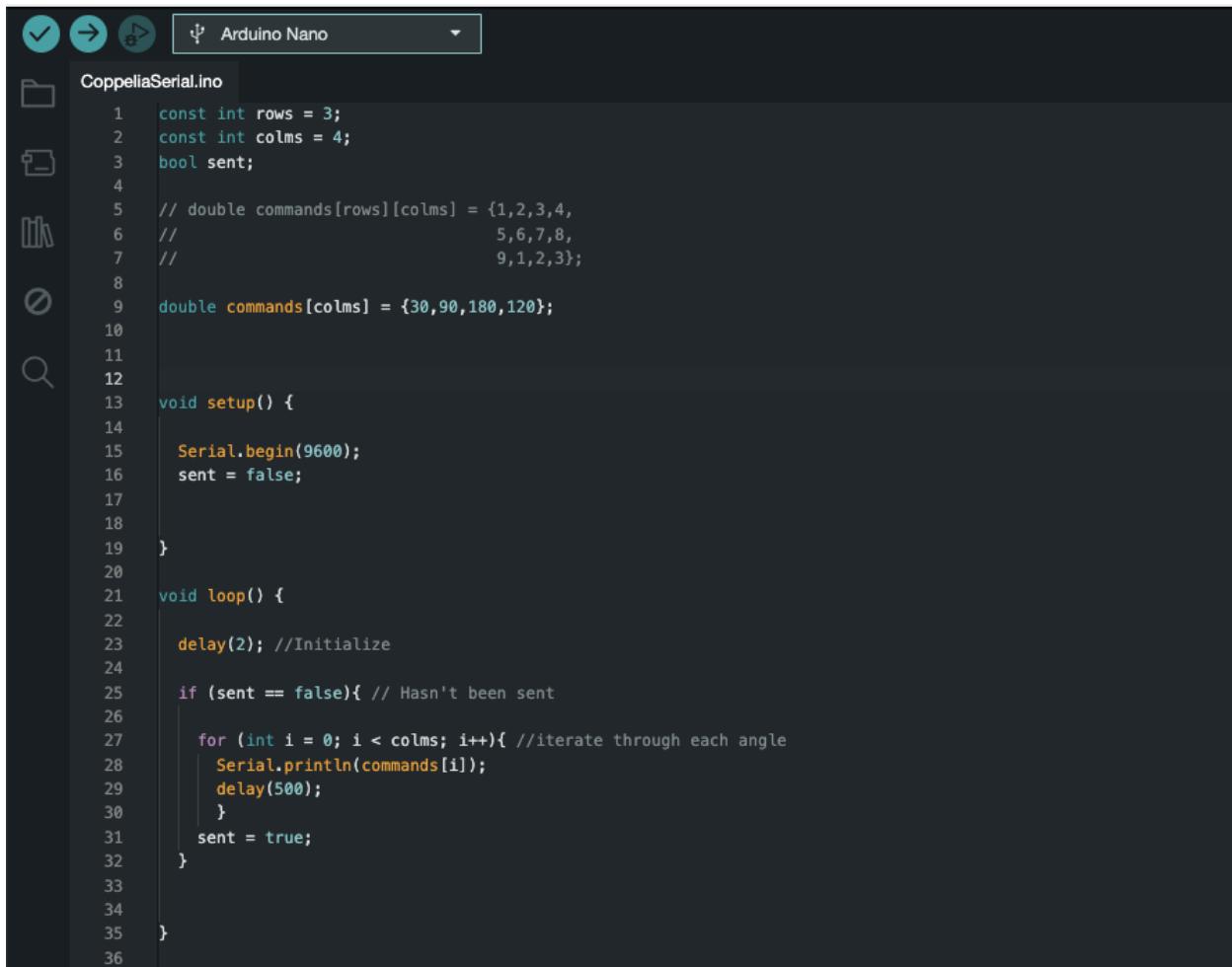


```

50 L
51 def findPort():
52     ports = serial.tools.list_ports.comports()
53     global finalPort
54     finalPort = ''
55     portList = []
56
57     for onePort in ports:
58         portList.append(str(onePort))
59     for port in portList:
60         usb = ''
61         for i in range (8,17):
62             usb += port[i]
63             if usb == "usbserial":
64                 for i in range (0,22):
65                     finalPort += port[i]
66                 break
67     print("Final port: " + finalPort)
68
69     global ser_com
70     ser_com = serial.Serial(finalPort, 9600)
71     print("Serial opening...")
72     time.sleep(3)
73
74
75 def onQ1Change(uiHandle, id, newValue):
76
77     global qlref
78     qlref = math.radians((newValue + 90))
79     sim.setJointTargetPosition(joint1, qlref)
80     sendNewCoords()
81
82 def onQ2Change(uiHandle, id, newValue):
83
84     global q2ref
85     q2ref = math.radians((newValue + 90))
86     sim.setJointTargetPosition(joint2, q2ref)
87     sendNewCoords()
88
89 def onQ3Change(uiHandle, id, newValue):
90
91     global q3ref
92     q3ref = math.radians((newValue + 90))
93     sim.setJointTargetPosition(joint3, q3ref)
94     sendNewCoords()
95
96
97 def onGripperChange(uiHandle, id, newValue):
98     global gripref
99
100    if (newValue == 0):
101        gripref = math.radians(180)
102        sim.setJointTargetPosition(gripper, gripref)
103        #open the gripper by moving it's motor
104    elif (newValue == 2):
105        gripref = math.radians(0)
106        sim.setJointTargetPosition(gripper, gripref)
107        #close the gripper by moving it's motor
108        sendNewCoords()
109
110
111 def sendNewCoords():
112
113     global qlref
114     global q2ref
115     global q3ref
116     global gripref
117
118     currentCoordinates = [math.degrees(qlref), math.degrees(q2ref), math.degrees(q3ref), math.degrees(gripref)]
119     toWrite = ','.join(str(coord) for coord in currentCoordinates)
120     print(toWrite)
121     toWrite = toWrite + '\n'
122
123     global ser_com
124     ser_com.write(toWrite.encode())
125
126
127
128 def sysCall_actuation():
129
130     pass
131
132
133

```

**Iteration # 3:** This iteration uses the CoppeliaSerial Sketch to communicate between the simulation and Arduino to send the lines of coordinates of each joint as the sliders are moved. This iteration was important as it was the first model that utilized serial communication between the simulation and Arduino to send joint coordinates.



The screenshot shows the Arduino IDE interface with the following details:

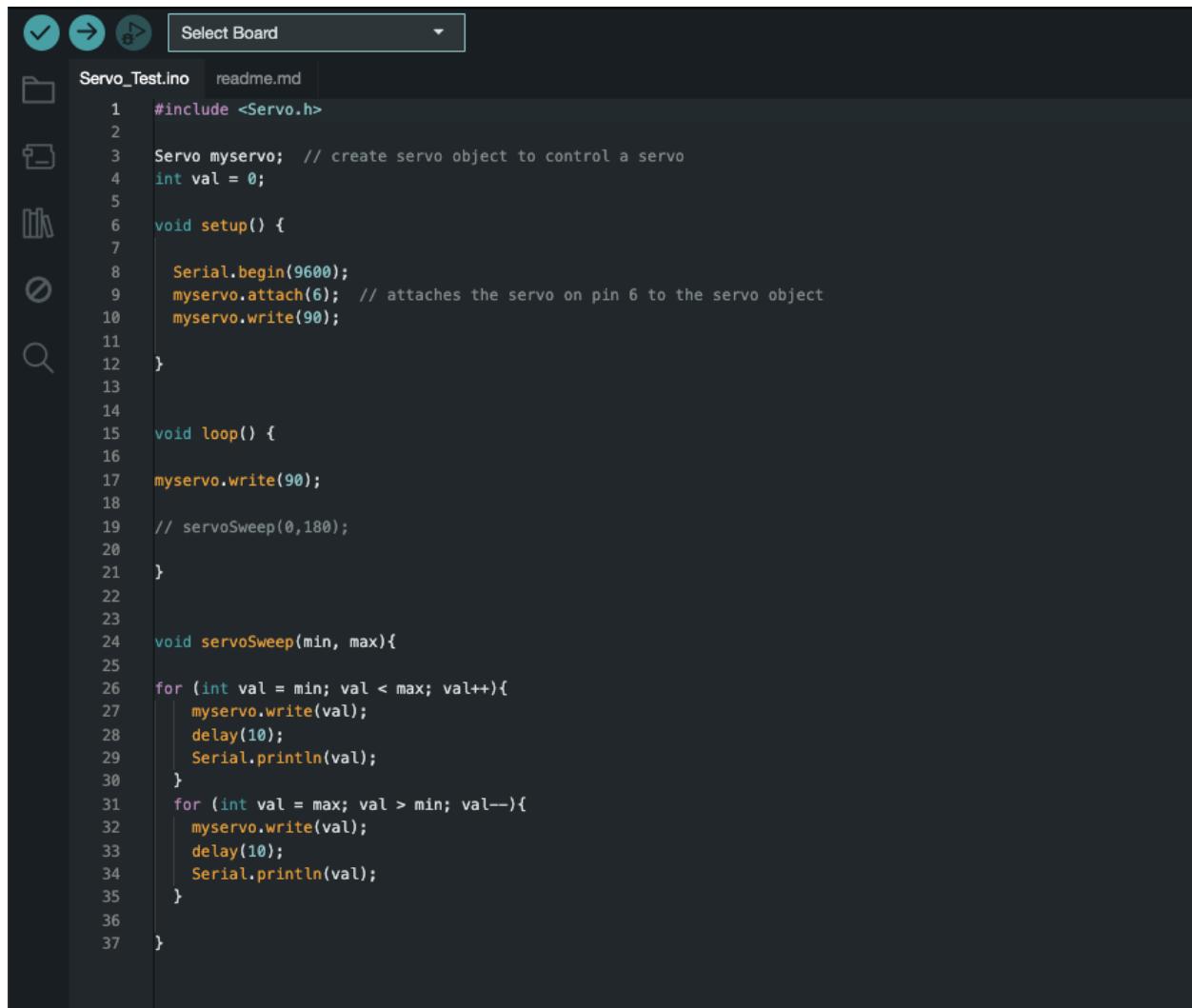
- Sketch Name:** CoppeliaSerial.ino
- Board:** Arduino Nano
- Code Content:** The code is a sketch named "CoppeliaSerial.ino" for an Arduino Nano. It defines variables for rows (3), columns (4), and a boolean "sent". It initializes the serial port at 9600 bps and sets "sent" to false. In the loop, it checks if "sent" is false (indicating it hasn't been sent yet) and iterates through each column (i from 0 to 3). For each column, it prints the corresponding command value (e.g., 30, 90, 180, 120) to the serial port and waits 500ms using delay(500). After sending all commands, it sets "sent" to true. The code is numbered from 1 to 36.

```
const int rows = 3;
const int colms = 4;
bool sent;
// double commands[rows][colms] = {1,2,3,4,
//                                  5,6,7,8,
//                                  9,1,2,3};
double commands[colms] = {30,90,180,120};

void setup() {
    Serial.begin(9600);
    sent = false;
}

void loop() {
    delay(2); //Initialize
    if (sent == false){ // Hasn't been sent
        for (int i = 0; i < colms; i++){ //iterate through each angle
            Serial.println(commands[i]);
            delay(500);
        }
        sent = true;
    }
}
```

**Iteration # 5:** A servo is connected to the Arduino and uses the simulation with 4 blocks with the Servo\_Test sketch to move the motor in accordance with changes in the simulation. This iteration is important as it was the first test that successfully used the simulation to move a motor accordingly.



The screenshot shows the Arduino IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Find, Refresh), a "Select Board" dropdown, and a search icon.
- Project Explorer:** Shows two files: "Servo\_Test.ino" (selected) and "readme.md".
- Code Editor:** Displays the following C++ code for "Servo\_Test.ino":

```
#include <Servo.h>

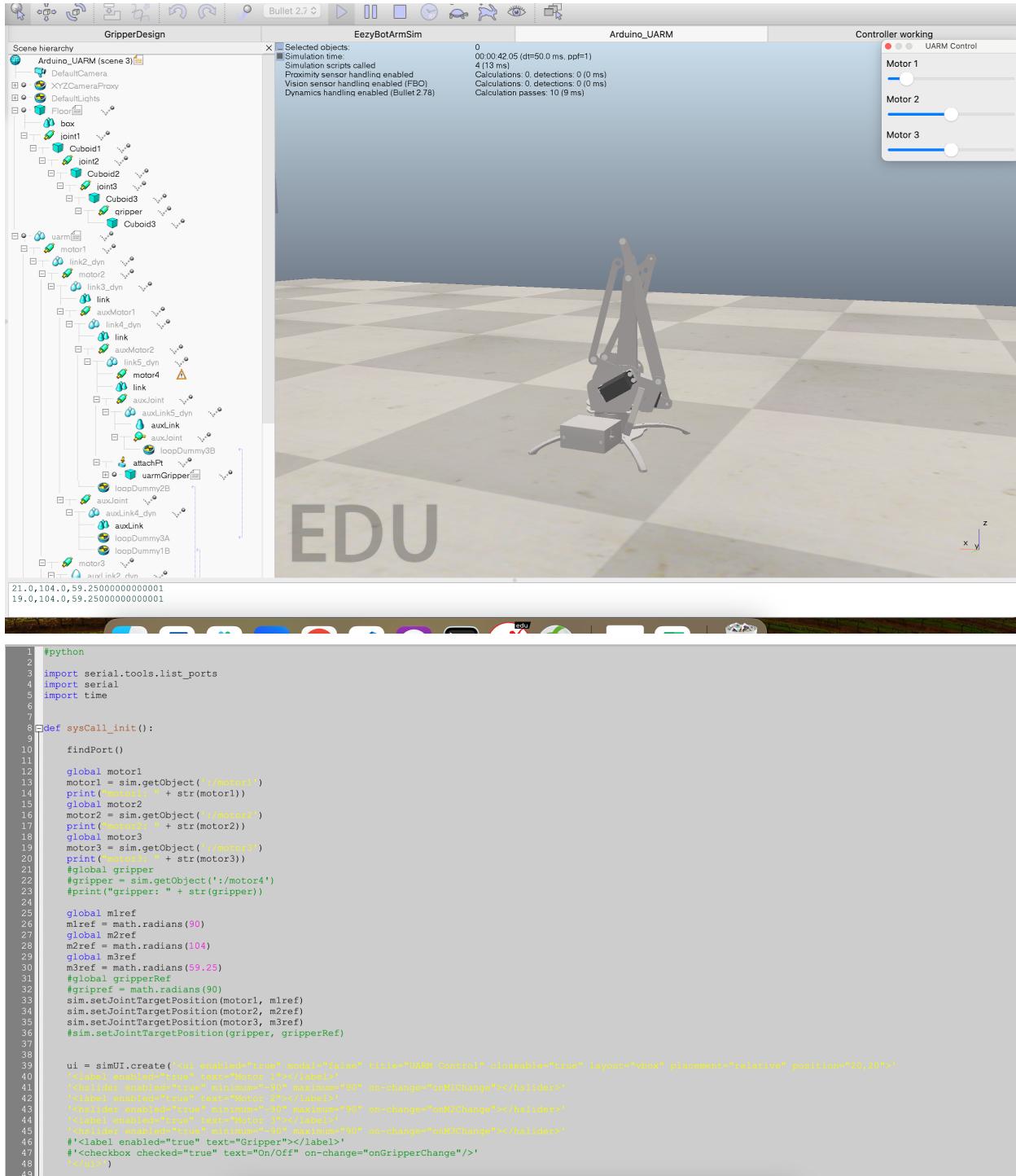
Servo myservo; // create servo object to control a servo
int val = 0;

void setup() {
    Serial.begin(9600);
    myservo.attach(6); // attaches the servo on pin 6 to the servo object
    myservo.write(90);
}

void loop() {
    myservo.write(90);
    // servoSweep(0,180);
}

void servoSweep(min, max){
for (int val = min; val < max; val++){
    myservo.write(val);
    delay(10);
    Serial.println(val);
}
for (int val = max; val > min; val--){
    myservo.write(val);
    delay(10);
    Serial.println(val);
}
}
```

**Iteration # 6:** The UARM model is loaded into CoppeliaSim and sliders are functional but only the base rotation joint functions properly. This iteration is important as it uses a simulated robot similar to the physical robot, but the full functionality of the simulation was hindered.

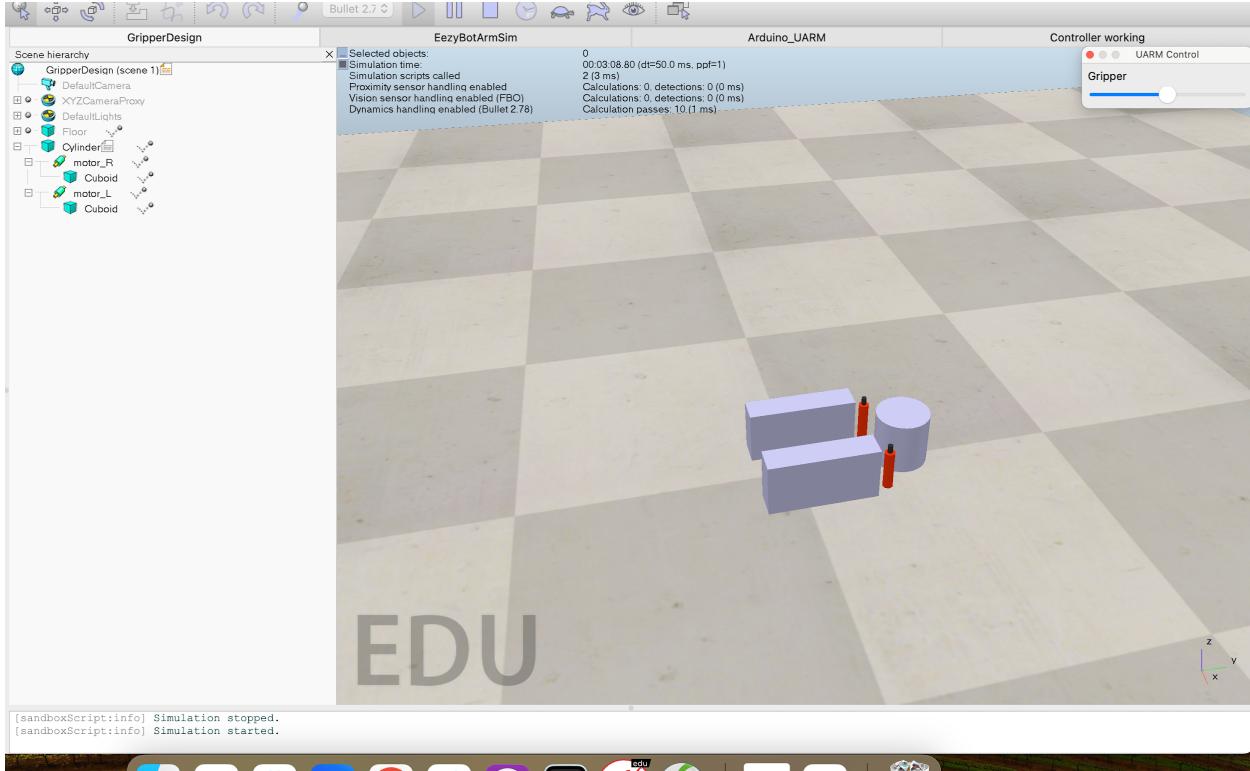


```

50 L
51 def findPort():
52     ports = serial.tools.list_ports.comports()
53     global finalPort
54     finalPort =
55     portList = []
56
57     for onePort in ports:
58         portList.append(str(onePort))
59     for port in portList:
60         usb = ""
61         for i in range (8,17):
62             usb += port[i]
63         if usb == "usbserial":
64             for i in range (0,22):
65                 finalPort += port[i]
66             break
67     print("final port: " + finalPort)
68
69     global ser_com
70     ser_com = serial.Serial(finalPort, 9600)
71     print("initializing...")
72     time.sleep(1)
73
74
75
76 def onM1Change(uiHandle, id, newValue):
77
78     global mlref
79     mlref = math.radians((newValue + 90))
80     sim.setJointTargetPosition(motor1, mlref)
81     sendNewCoords()
82
83 def onM2Change(uiHandle, id, newValue):
84
85     global m2ref
86     m2ref = math.radians((newValue + 90))
87     sim.setJointTargetPosition(motor2, m2ref)
88     sendNewCoords()
89
90 def onM3Change(uiHandle, id, newValue):
91
92     global m3ref
93     m3ref = math.radians((newValue + 90))
94     sim.setJointTargetPosition(motor3, m3ref)
95     sendNewCoords()
96
97 def onGripperChange(uiHandle, id, newValue):
98     global gripref
99
100    if (newValue == 0):
101        #print("opened")
102        gripref = math.radians(180)
103        sim.setJointTargetPosition(gripper, gripref)
104        #open the gripper by moving it's motor
105    elif (newValue == 2):
106        #print("closed")
107        gripref = math.radians(0)
108        sim.setJointTargetPosition(gripper, gripref)
109        #close the gripper by moving it's motor
110        sendNewCoords()
111
112
113
114 def sendNewCoords():
115
116     global mlref
117     global m2ref
118     global m3ref
119     #global gripperRef
120
121     currentCoordinates = [math.degrees(mlref), math.degrees(m2ref), math.degrees(m3ref)]#, math.degrees(gripperRef)]
122     toWrite = ','.join(str(coord) for coord in currentCoordinates)
123     print(toWrite)
124     toWrite = toWrite + '\n'
125
126     global ser_com
127     ser_com.write(toWrite.encode())
128
129
130
131 def sysCall_actuation():
132
133     pass
134
135
136 def sysCall_sensing():
137     # put your sensing code here
138     pass
139
140 def sysCall_cleanup():
141     # do some clean-up here
142     pass
143
144

```

**Iteration # 9:** A simulation of only a gripper model is functional by controlling the left and right joints of the gripper simultaneously. To this point, the robot simulation was functional except for the gripper. The development of this targeted simulation to achieve gripper functionality allowed for easy implementation in the final stimulation.

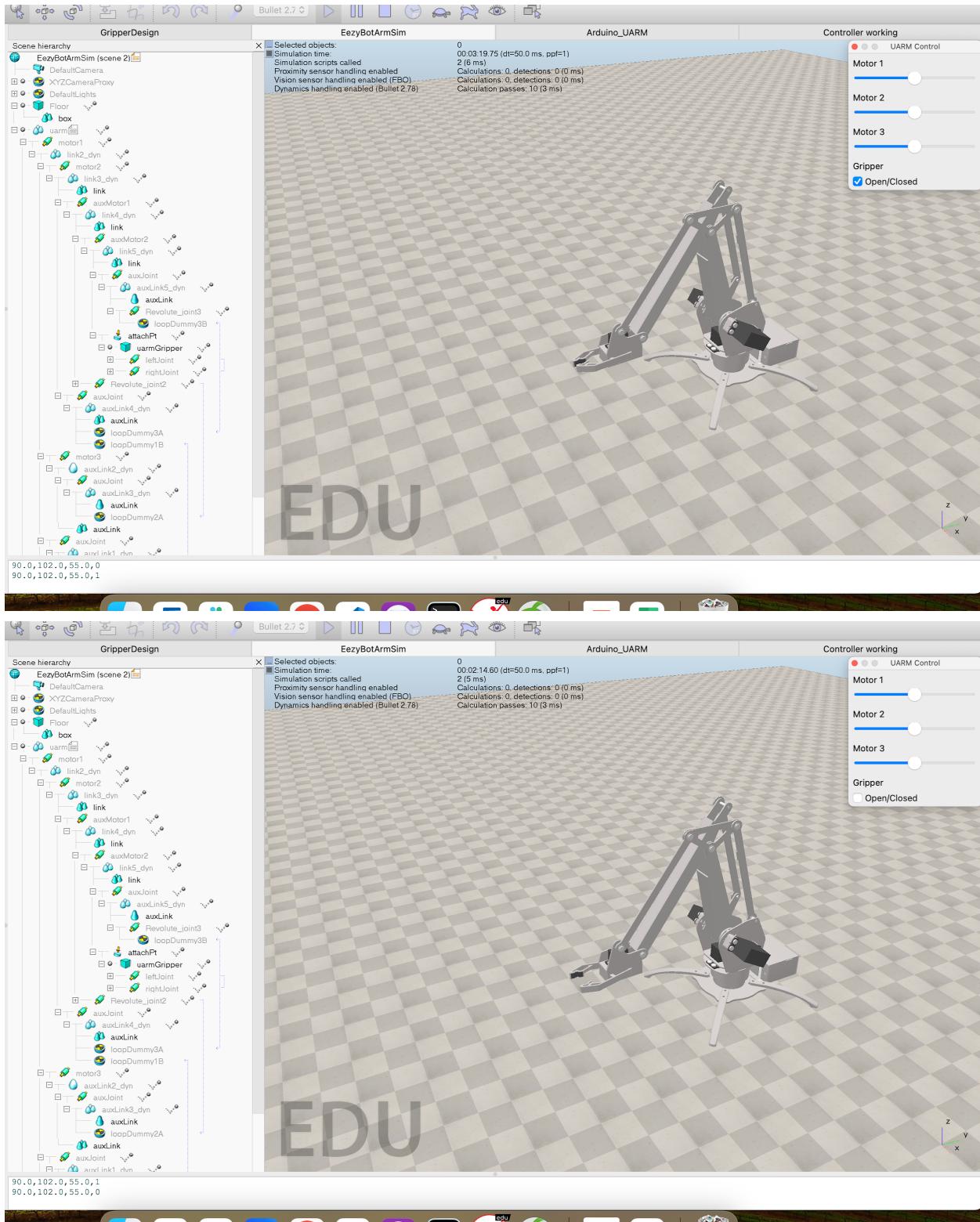


```

1  #python
2
3  def sysCall_init():
4
5      global gripperRight
6      gripperRight = sim.getObject('/motor_R')
7      global gripperLeft
8      gripperLeft = sim.getObject('/motor_L')
9
10     global gripperRef
11     gripperRef = math.radians(0)
12     sim.setJointTargetPosition(gripperRight, gripperRef)
13     sim.setJointTargetPosition(gripperLeft, gripperRef * -1)
14
15     # Creates the UI box with Sliders to control the robot
16
17     ui = simUI.create("ui_cuboid='true' modal='false' title='UARM Control' closable='true' layout='vbox' placement='relative' position='20,20'")
18     "label enabled='true' text='Gripper'></label>
19     <slider enabled='true' minimum='-90' maximum='90' on-change='onGripperChange'></slider>
20   </ui>")
21
22 def onGripperChange(uiHandle, id, newValue):
23
24     global gripperRef
25     gripperRef = math.radians(newValue)
26     sim.setJointTargetPosition(gripperRight, gripperRef)
27     sim.setJointTargetPosition(gripperLeft, gripperRef * -1)
28

```

**Iteration # 12:** The robot has smooth motion at all joints and is able to move with the servo motors. This iteration is important as it is the first fully functional iteration of the simulated robot. It works effectively using the slider UI.



```

1 #python
2
3 import serial.tools.list_ports
4 import serial
5 import time
6
7
8 def sysCall_init():
9     findPort()
10
11     global motor1
12     motor1 = sim.getObject(':/motor1')
13     global motor2
14     motor2 = sim.getObject(':/motor2')
15     global motor3
16     motor3 = sim.getObject(':/motor3')
17     global gripperRight
18     gripperRight = sim.getObject('/rightJoint')
19     global gripperLeft
20     gripperLeft = sim.getObject('/leftJoint')
21     gripperRef = sim.getObject('/refJoint')
22
23     global m1ref
24     m1ref = math.radians(90)
25     sim.setJointTargetPosition(motor1, m1ref)
26     global m2ref
27     m2ref = math.radians(90)
28     sim.setJointTargetPosition(motor2, m2ref)
29     global m3ref
30     m3ref = math.radians(90)
31     sim.setJointTargetPosition(motor3, m3ref)
32     global gripperRef
33     gripperRef = math.radians(0)
34     sim.setJointTargetPosition(gripperRight, gripperRef)
35     sim.setJointTargetPosition(gripperLeft, gripperRef)
36
37     # Creates the UI box with Sliders to control the robot
38
39     ui = simUI.create('ui combobox="true" model="false" circle="true" layout="vbox" placement="relative" position="20,20">' +
40         '<label enabled="true" text="Motor 1"></label>' +
41         '<label enabled="true" text="Motor 2"></label>' +
42         '<label enabled="true" text="Motor 3"></label>' +
43         '<label enabled="true" text="Gripper"></label>' +
44         '<label enabled="true" text="Open/Closed"></label>' +
45         '<checkbox checked="true" text="Gripper" on-change="onGripperChange"/>' +
46         '<checkbox checked="true" text="Open/Closed" on-change="onOpenClose"/>' +
47     '</ui>')
48
49
50 def findPort():
51     ports = serial.tools.list_ports.comports()
52     global finalPort
53     finalPort =
54     portList = []
55
56     # form a list of 11 available ports
57     for onePort in ports:
58         portList.append(str(onePort))
59     # Check to see if any of the available ports match the expected name of an Arduino nano
60     for port in portList:
61         usb = ''
62         for i in range(8,17):
63             usb += port[i]
64             if usb == "usbserial":
65                 for i in range(0,22):
66                     finalPort += port[i]
67                     break
68     print("Com Port: " + finalPort)
69
70     # Open the com port with the Arduino
71     global ser_com
72     ser_com = serial.Serial(finalPort, 9600)
73
74
75
76
77 def onM1Change(uiHandle, id, newValue):
78
79     global m1ref
80     m1ref = math.radians(newValue + 90)
81     sim.setJointTargetPosition(motor1, m1ref)
82     sendNewCoords()
83
84 def onM2Change(uiHandle, id, newValue):
85
86     global m2ref
87     m2ref = math.radians(newValue + 90)
88     sim.setJointTargetPosition(motor2, m2ref)
89     sendNewCoords()
90
91 def onM3Change(uiHandle, id, newValue):
92
93     global m3ref
94     m3ref = math.radians(newValue + 90)
95     sim.setJointTargetPosition(motor3, m3ref)
96     sendNewCoords()
97

```

```

98     def onGripperChange(uiHandle, id, newValue):
99         global gripperRef
100
101        if (newValue == 0):
102            gripperRef = 0
103            sim.setJointTargetPosition(gripperRight, math.radians(0))
104            sim.setJointTargetPosition(gripperLeft, math.radians(0))
105        elif (newValue == 2):
106            gripperRef = 1
107            sim.setJointTargetPosition(gripperRight, math.radians(-9))
108            sim.setJointTargetPosition(gripperLeft, math.radians(-9))
109
110    sendNewCoords()
111
112
113    def final_map(value, in_min, in_max, out_min1, out_max1, out_min2, out_max2, threshold):
114        if value < threshold:
115            return map_value(value, in_min, threshold, out_min1, out_max1)
116        else:
117            return map_value(value, threshold, in_max, out_min2, out_max2)
118
119    def map_value(value, in_min, in_max, out_min, out_max):
120        return (value - in_min) * (out_max - out_min) // (in_max - in_min) + out_min
121
122
123    def sendNewCoords():
124
125        global mlref
126        global m2ref
127        m2send = final_map(math.degrees(m2ref), 0, 180, 180, 90, 90, 0, 104)
128        global m3ref
129        m3send = final_map(math.degrees(m3ref), 0, 180, 180, 90, 90, 0, 35)
130        global gripperRef
131
132        currentCoordinates = [math.degrees(mlref), m2send, m3send, gripperRef]
133        toWrite = ' '.join(str(coord) for coord in currentCoordinates)
134        print(toWrite)
135        toWrite = toWrite + '\n'
136
137        global ser_com
138        ser_com.write(toWrite.encode())
139
140
141
142    def sysCall_actuation():
143        # put your repeated code here
144        pass
145

```

**Iteration # 15:** Both the simulation and the robot function as expected when controlled by the simulation at this point. This iteration is important as it sees the development of a functional Python script in the simulation and a successful Arduino script. The two programs work together effectively to control the robot.



The screenshot shows the Arduino IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Find, Copy, Paste), a search bar, and a "Select Board" dropdown.
- Code Area:** Displays the Arduino sketch "coppeliaSim\_Input.ino".
- Code Content:** The code is a C++ program for an Arduino. It includes declarations for four servos (baseServo, rightServo, leftServo, gripServo) and variables for baseValue, rightValue, and leftValue. It defines a constant colms = 4 and an array commands[cols]. The setup() function initializes the serial port at 9600 bps and attaches the servos to pins 3, 5, 6, and 9. It sets all motors to initial simulation positions (commands[0] = 90, commands[1] = 90, commands[2] = 90, commands[3] = 1) and calls setServos(). The loop() function reads coordinates from the serial port, sets servos to new positions, and delays for 5 milliseconds. The coordinates() function is a helper that takes CSV input and parses it into an array of doubles. The code uses standard Arduino libraries like <Servo.h>.

```
59
60
61 if (Serial.available() > 0) {
62     coords = Serial.readStringUntil('\n');
63
64     while (index < coords.length()) { // Iterate through the CSV line
65         int commaIndex = coords.indexOf(',', index);
66
67         if (commaIndex != -1) { // Check if a comma was found
68             // Extract the substring between start and commaIndex
69             String substr = coords.substring(start, commaIndex);
70             commands[j] = substr.toInt();
71             start = commaIndex + 1; // Move the start index to the character after the comma
72
73         } else {
74             // If no comma is found, extract the substring from start to the end of the line
75             String substr = coords.substring(start);
76             commands[j] = substr.toInt();
77             break; // Exit the loop since we reached the end of the coordinate line
78         }
79         // Move the index to the character after the comma
80         index = commaIndex + 1;
81         j++;
82     }
83 }
84
85
86 }
87
88 void setServos(){
89
90     baseServo.write(commands[0]);
91     rightServo.write(commands[1]);
92     leftServo.write(commands[2]);
93
94
95     if (commands[3] == 1) {
96         //Closed
97         gripServo.write(35);
98     } else {
99         //Opened
100        gripServo.write(90);
101    }
102
103 }
104
105
106 void servoArc(){
107
108     for (int i = 0; i < 181; i += 5){
109         Serial.println(i);
110         baseServo.write(i);
111         delay(4500);
112     }
113 }
114 }
```

**Iteration # 17:** The robot can be given the coordinates of an object to which it will move and pick up the object. Then a new set of coordinates can be given to move the object to. This iteration showcases the final functionality of SCRAP as a pick-and-place system.

