

# **3D Printed Robotic Arm Project Report**

EEL 4664.01

December 2<sup>nd</sup>, 2025

## **Group 11**

Garrett Mislevy

Roland Diaz

Kyle Blanchard

## **Introduction**

This report will detail the steps and actions needed to build and operate a robotic arm. The arm is made of 3D printed components as well as servos operated by an Arduino Uno. The Arduino runs C code that is adapted from the tutorial written for How To Mechatronics by Dejan. Not all aspects of this tutorial were followed to a tee as parts were difficult or unnecessary to implement. For example, there is no app control for this robot as Android emulation did not allow for Wi-Fi or Bluetooth pass-through. However, the result is still a robot that can repeat a preprogrammed cycle or respond to user inputs from the serial monitor.

## **Materials**

1. 3 Micro Servo Motors (for wrist and claw)
2. 3 Mini Servo Motors (for base, shoulder, and elbow)
3. 3D Printed Components (from How To Mechatronics tutorial)
4. Assorted Screws
5. Rubber Band
6. Wooden Plank (to attach base of robot to create stability)
7. Arduino Uno
8. 9V Lithium Battery
9. Arduino Uno Servo Driver (PCA9685)
10. Buck Converter (on/off switch)

## **Methods**

The completion of this robotic arm begins attaching the servos to their respective 3D printed parts, as seen in How To Mechatronics. From there, put the pieces together to have a built arm. At this point, there is a diversion from the tutorial as this arm does not have a Bluetooth module and thus does not have wireless control. The motor controller is attached to the top of the Arduino Uno, and all the servos are connected to the controller. Power for the controller is provided through a buck converter that receives its power from a DC battery. This means that the battery provides power to the servos while the Arduino controls the servos. The power needed for the Arduino is given through its connection to a computer.

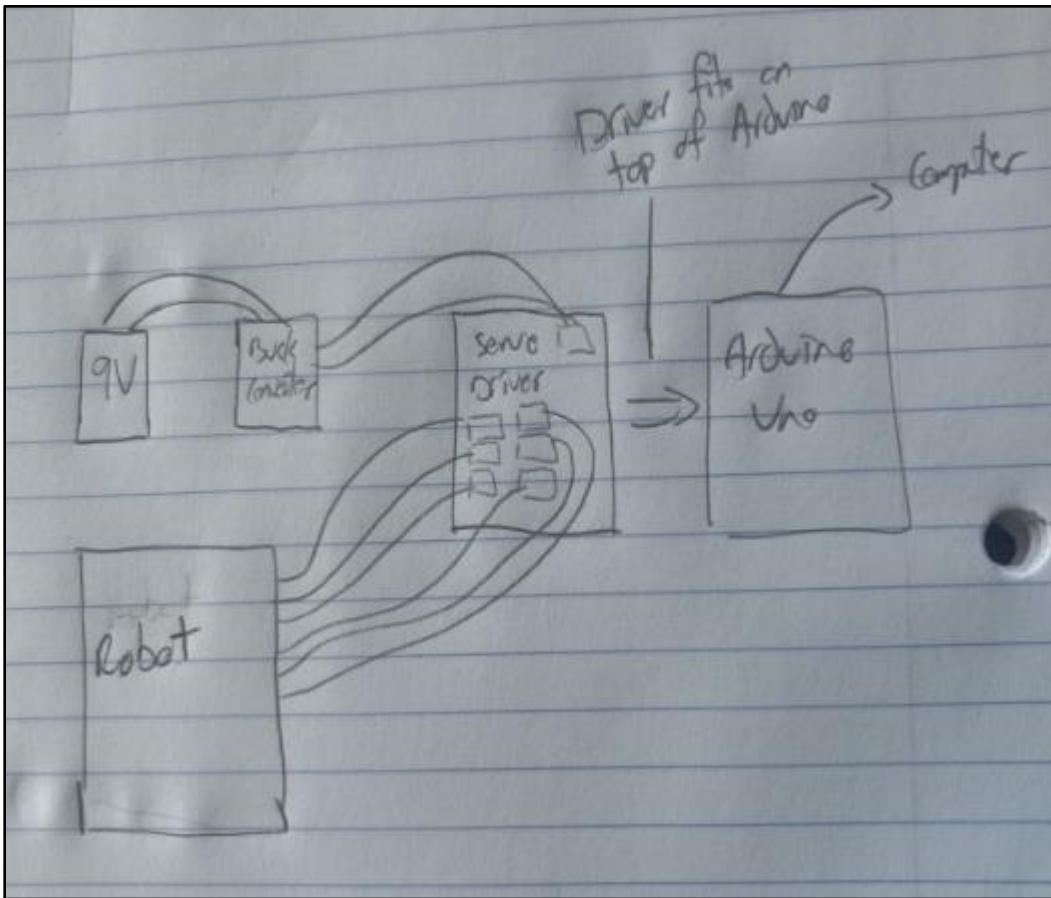


Figure 1: General Wiring Diagram

The code for the Arduino can be broken into its functions. First, there is the setup that sets a baud rate of 115200, begins the PWM, sets the robot to an initial position of pointing straight up, and prompts for next action. In the next loop, the program waits for a command to be sent from the serial monitor. If the command is ‘start’, the repeated movement of the arm picking up and moving the duck is started, and the case of ‘stop’, all movements are halted. While the robot runs, it loops through the functions for moving the robot into key positions.

```

void loop() {
    // Check for serial commands
    if (Serial.available() > 0) {
        String command = Serial.readStringUntil('\n');
        command.trim();
        command.toLowerCase();

        if (command == "start") {
            isRunning = true;
            Serial.println("Starting movement sequence...");
        } else if (command == "stop") {
            isRunning = false;
            Serial.println("Stopping movement sequence");
        }
    }
    // Only run movement sequence if enabled
    if (isRunning) {
        moveSequentialTo(movement1, sequence1, NUM_SERVOS);
        if (!isRunning) return; // Check if stopped during movement
        delay(50);

        moveSmoothTo(movement2);
        if (!isRunning) return;
        delay(50);

        moveSequentialTo(movement3, sequence3, NUM_SERVOS);
        if (!isRunning) return;
        delay(50);
    }
}

```

Figure 2: The main loop

The first part of these key movements sends the claw over to the duck's location to pick it up. This uses the sequential movement function that receives arguments for the final position, the sequence in which the servos should be actuated, and the number of servos being moved. The next call is to a smooth movement function that moves the arm into a neutral position without worrying about the sequence of the servos being moved. From the neutral position, the sequential function is called again to position the claw over the box and drop the duck.

```

// Move servos in a specific sequence
void moveSequentialTo(const int target[], const int sequence[], int seqLength) {
    for (int s = 0; s < seqLength; s++) {
        if (!isRunning) return; // Stop immediately if commanded

        int servoIndex = sequence[s];

        // Move this servo to target position
        while (currentPos[servoIndex] != target[servoIndex]) {
            if (!isRunning) return; // Check during movement

            if (currentPos[servoIndex] < target[servoIndex]) {
                currentPos[servoIndex]++;
            } else {
                currentPos[servoIndex]--;
            }
            writeServo(servoIndex, currentPos[servoIndex]);
            delay(10);
        }
    }
}

```

Figure 3: Sequential movement loop

The backbone function to all previously stated controls a single servo with PWM. Every function that changes the position of the arm calls this function.

```

// Write a single servo
void writeServo(int index, int angle) {
    int pulse = map(angle, 0, 180, SERVO_MIN, SERVO_MAX);
    pwm.setPWM(servоСhannels[index], 0, pulse);
}

```

Figure 4: The PWM control of servo function

## **Results and Discussion**

The finalized robot works as intended with a few drawbacks. The programming of the arm was set up for both manual inputs and repeated loops. For both cases, the expected movement was generally achieved. However, when the battery was low on charge or the servos heated up, the robot experienced erratic and sluggish movement. This behavior was seen in the competition for this robot where it failed to pick up and transfer a single duck. Just minutes before the competition, the robot was consistently moving the ducks, but that practice seems to have clearly hampered the performance when it was needed most.

Although there is not much to be done about overheating, other than letting the robot rest, there are some solutions to avoid loss of torque due to low battery. One of the simpler solutions would be to use a larger battery as the one for this arm was merely a single cell. However, the best solution could be to connect the motor controller to a clean, consistent source of power, like a bench DC power supply.

## **Conclusions**

The conclusion to take from this report is that changes from an expected path can be made to create a successful project. The final robotic arm did not use Bluetooth but was able to work with the necessary actions. Changes were also made to how power was delivered and how the servos were controlled, and the arm improved its capabilities.

## **GitHub Link**

[https://github.com/abasichuuman/RoboticArm\\_Group11\\_KinematicsAndCRS\\_FinalProject.git](https://github.com/abasichuuman/RoboticArm_Group11_KinematicsAndCRS_FinalProject.git)

## **Acknowledgements**

The creators of this robotic arm, Garrett Mislevy, Roland Diaz, and Kyle Blanchard, would like to thank those that supported the completion of this project. Dr. Hoan Ngo was an excellent mentor and greatly assisted in finding solutions to the problems that cropped up throughout the semester. Also, the Print Lab was very helpful and efficient in providing 3D printed parts for the arm.

## **References**

[DIY Arduino Robot Arm with Smartphone Control - How To Mechatronics](#)

[Overview | Adafruit 16-channel PWM/Servo Shield | Adafruit Learning System](#)

[How to control Robotic arm from Terminal](#)