Kinematics and Controls Robot Arm

By: Ryan Bezerra, William Denning, Brian Scholes

EEL 4664: Kinematics and Controls of Robotic Systems

12/02/2025

Introduction:

This semester, the team has built a robot arm. This arm can be controlled wirelessly via Bluetooth using an app and autonomously pick up a duck and place it in another location. Additionally, a simple loop was created to manually execute the arm, picking up and dropping objects in a specific area. This document discusses the materials and controls for this bot.

Materials:

Robot Arm Bill of Materials	
Part	Quantity
Arduino	1
SG90 Micro Servo	3
MG996R Servo	3
Wood board	1
5V 2A DC Power Supply	1
3D Printed Parts	14
HC-05 Bluetooth module	1

Methods

It was decided to keep the code contained in simple Arduino scripts. This was done for the following two reasons:

1. It was an easy-to-integrate format, as we were already using Arduino boards, and programming through its own IDE made for straightforward implementation.

2. The guide we followed for much of the project already had reference code in (.ino) files, starting us within the Arduino IDE for that aspect of arm control.

Bluetooth Control Code

Initially, we focused on Bluetooth control of the arm to make full use of the arm provided in the guide for this project, focusing on debugging the provided Arduino code. During this process, it became apparent that the original code was sending too much information at once for the Arduino to properly handle. This caused the arm to move erratically when sending commands through the provided app. The code also mismatched the baud rates between the Arduino and the HC-05 Bluetooth module. Various small errors within the code made Bluetooth control of the arm virtually unusable. The issues in baud rate, command frequency, and code complexity were addressed in the following code to ensure the app functioned as expected.

```
#include <Servo.h>

// Note: SoftwareSerial is REMOVED.
// We are using Hardware Serial (Pins 0 and 1) for smoother performance.

Servo servo01;
Servo servo02;
Servo servo03;
Servo servo04;
Servo servo05;
Servo servo06;

// Variables
int servo1Pos, servo2Pos, servo3Pos, servo4Pos, servo5Pos, servo6Pos;
int servo1PPos, servo2PPos, servo3PPos, servo4PPos, servo5PPos, servo6PPos;
int servo01SP[50], servo02SP[50], servo03SP[50], servo04SP[50], servo05SP[50],
servo06SP[50];
int speedDelay = 20;
int index = 0;
String dataIn = "";
unsigned long lastServoCmdTime = 0;
int pendingServoID = 0;
int pendingServoPos = -1;

void setup() {
  // PRE-SET POSITIONS (Fixes the startup slam)
```

```
  servo1PPos = 40;
  servo01.write(servo1PPos);

  servo2PPos = 160;
  servo02.write(servo2PPos);

  servo3PPos = 45;
  servo03.write(servo3PPos);

  servo4PPos = 90;
  servo04.write(servo4PPos);

  servo5PPos = 30;
  servo05.write(servo5PPos);

  servo6PPos = 130;
  servo06.write(servo6PPos);

  // START SERIAL (Hardware Serial on Pins 0 & 1)
  // IMPORTANT: Unplug Bluetooth RX/TX when uploading this code!
  Serial.begin(9600);
  Serial.setTimeout(5); // Reduced timeout for faster response

  delay(20);

  // ATTACH SERVOS
  // (Using extended range on Waist if you want that extra reach)
  servo01.attach(5, 500, 2600);
  servo02.attach(6);
  servo03.attach(7);
  servo04.attach(8);
  servo05.attach(9);
  servo06.attach(10);
}

void loop() {
  // ----- READ DATA (Using Hardware Serial) -----
  if (Serial.available()) {
    dataIn = Serial.readStringUntil('#'); // Reads until '#' or timeout
    lastServoCmdTime = millis();
  }

  // ----- HANDLE SERVO COMMANDS -----
```

```
  if (dataIn.startsWith("s")) {
    pendingServoID  = dataIn.substring(1, 2).toInt();
    pendingServoPos = dataIn.substring(2).toInt();
    dataIn = ""; // Clear buffer
  }

  // ----- EXECUTE MOVE -----
  // We wait 40ms after command to ensure no more data is flooding in
  if (pendingServoID != 0 && (millis() - lastServoCmdTime > 40)) {

    int* prev;
    Servo* s;

    // Pointer assignment
    switch (pendingServoID) {
      case 1: prev = &servo1PPos; s = &servo01; break;
      case 2: prev = &servo2PPos; s = &servo02; break;
      case 3: prev = &servo3PPos; s = &servo03; break;
      case 4: prev = &servo4PPos; s = &servo04; break;
      case 5: prev = &servo5PPos; s = &servo05; break;
      case 6: prev = &servo6PPos; s = &servo06; break;
      default: pendingServoID = 0; return; // Invalid ID check
    }

    // SAFETY CHECK: Fix erratic movements
    // Sometimes Bluetooth drops a digit (e.g., "180" becomes "18").
    // We ignore big, instant jumps if they look suspicious, or just clamp the
value.
    if (pendingServoPos < 0) pendingServoPos = 0;
    if (pendingServoPos > 180) pendingServoPos = 180;

    // Perform the move
    if (pendingServoPos != *prev) {
      s->write(pendingServoPos);
      *prev = pendingServoPos;
    }

    // Clear pending command
    pendingServoID = 0;
    pendingServoPos = -1;
  }

  // ----- SAVE -----
```

```
  if (dataIn.startsWith("SAVE")) {
    // Check array bounds to prevent crashing
    if(index < 50) {
        servo01SP[index] = servo1PPos;
        servo02SP[index] = servo2PPos;
        servo03SP[index] = servo3PPos;
        servo04SP[index] = servo4PPos;
        servo05SP[index] = servo5PPos;
        servo06SP[index] = servo6PPos;
        index++;
    }
    dataIn = "";
  }

  // ----- RUN -----
  if (dataIn.startsWith("RUN")) {
    dataIn = "";
    runservo();
  }

  // ----- RESET -----
  if (dataIn == "RESET") {
    // Standard way to clear arrays
    memset(servo01SP, 0, sizeof(servo01SP));
    memset(servo02SP, 0, sizeof(servo02SP));
    memset(servo03SP, 0, sizeof(servo03SP));
    memset(servo04SP, 0, sizeof(servo04SP));
    memset(servo05SP, 0, sizeof(servo05SP));
    memset(servo06SP, 0, sizeof(servo06SP));
    index = 0;
    dataIn = "";
  }
}

// Automatic mode custom function
void runservo() {
  while (dataIn != "RESET") {
    for (int i = 0; i <= index - 2; i++) {
      if (Serial.available() > 0) {
        dataIn = Serial.readString();

        // PAUSE LOGIC
        if (dataIn == "PAUSE") {
```

```
          while (dataIn != "RUN") {
            if (Serial.available() > 0) {
              dataIn = Serial.readString();
              if (dataIn == "RESET") break;
            }
          }
        }

        // SPEED SLIDER
        if (dataIn.startsWith("ss")) {
          String dataInS = dataIn.substring(2, dataIn.length());
          speedDelay = dataInS.toInt();
        }
      }

      // Execute Moves using Helper Function to save space/cleanliness
      moveStoredServo(servo01, servo01SP, i);
      moveStoredServo(servo02, servo02SP, i);
      moveStoredServo(servo03, servo03SP, i);
      moveStoredServo(servo04, servo04SP, i);
      moveStoredServo(servo05, servo05SP, i);
      moveStoredServo(servo06, servo06SP, i);
    }
  }
}

// Helper function to clean up the big "runservo" loop
void moveStoredServo(Servo &s, int sp[], int i) {
    if (sp[i] == sp[i+1]) return; // No move needed

    if (sp[i] > sp[i+1]) {
        for (int j = sp[i]; j >= sp[i+1]; j--) {
            s.write(j);
            delay(speedDelay);
        }
    } else {
        for (int j = sp[i]; j <= sp[i+1]; j++) {
            s.write(j);
            delay(speedDelay);
        }
    }
}
```

Autonomous Code

With a greater understanding of the provided arm, a preset loop for the arm was created. This loop's main function is to move an object to the opposing side of the robot. Utilizing the same servo library used in the mobile app code, a simple loop was created. This loop contained a set of commands that dictated all servo movements of the arm. This allowed for pre-programmed movements within the loop to achieve the desired outcomes. This code is provided below. Additionally, a video of this code operating is provided within the GitHub for this project.

```
#include <Servo.h>

// define the 6 servos connected to the arduino
Servo servo01;
Servo servo02;
Servo servo03;
Servo servo04;
Servo servo05;
Servo servo06;

// SPEED SETTING: Higher number = Slower movement
// 10 is fast, 50 is slow
// this one only applies to the startup procedure
int moveSpeed = 10;

// Set to false by default, so the robot waits for a command
bool loopRunning = false;

void setup() {
  // Start the Serial Monitor so we can send/receive commands
  Serial.begin(9600);
  Serial.println("Ready! Type 'START' to begin sequence, 'STOP' to pause.");

  // pre set positions
  servo01.write(40);   // Waist start pos
  servo02.write(160);  // Shoulder start pos
  servo03.write(45);   // Elbow start pos
```

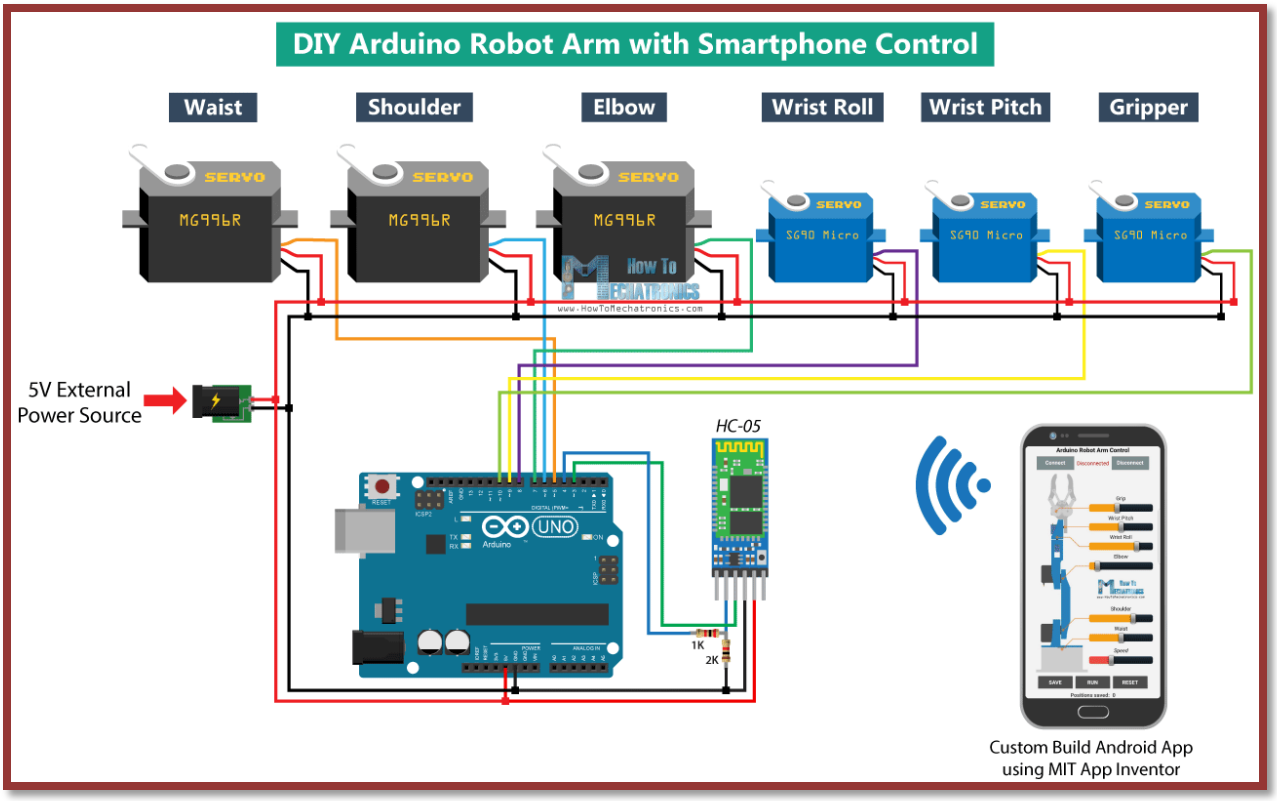
```
    servo04.write(90);   // Wrist Roll start pos
    servo05.write(30);   // Wrist Pitch start pos
    servo06.write(130);  // Gripper start pos

    // "attach" the servos to the arduino
    //servo01.attach(5);
    servo01.attach(5, 500, 2700); // attaches differently to avoid the software
limitation with 'servo.h' library
    servo02.attach(6);
    servo03.attach(7);
    servo04.attach(8);
    servo05.attach(9);
    servo06.attach(10);

    // awake dance
    servo06.write(130);
    delay(1000);
    servo06.write(70);
    delay(1000);
    servo06.write(130);
    delay(1000);
    servo06.write(70);
    delay(1000);
    servo06.write(130);
    delay(1000);
    servo06.write(70);
    delay(1000);
    servo06.write(130);
    delay(1000);
  }

  void loop() {
    if (Serial.available() > 0) {
      // Read the whole command until the newline character
      String command = Serial.readStringUntil('\n');
      command.trim(); // Remove any whitespace
      command.toUpperCase(); // Convert to uppercase for easy matching

      if (command == "START") {
        loopRunning = true;
        Serial.println("--- SEQUENCE STARTED ---");
      }
      else if (command == "STOP") {
```

```
        loopRunning = false;
        Serial.println("--- SEQUENCE STOPPED ---");
      }
    }

    // The loop only runs when the 'loopRunning' flag is true.
    if (loopRunning) {
      pickAndPlaceSequence();
    }
    // If loopRunning is false, the code simply checks for serial input very
fast.
  }

  void pickAndPlaceSequence() {
    // position to duck
    slowMove(servo01, 40); // Move waist to 40
    slowMove(servo05, 70); // Wrist pitch to 70
    slowMove(servo02, 130); // Move shoulder to 130
    slowMove(servo03, 87.5); // Elbow to 85
    servo06.write(130); // Open Gripper (130)
    delay(250); // Pause before grabbing

    // grab duck
    servo06.write(70); // gripper
    delay(250);

    // lift duck
    slowMove(servo02, 180); // Shoulder up
    slowMove(servo05, 130); // Wrist adjust
    slowMove(servo03, 55);  // Elbow adjust

    // turn around
    slowMove(servo01, 180);

    // drop duck
    delay(250);
    servo06.write(130); // gripper
    delay(250);
  }

  // function to move servos slowly
  void slowMove(Servo &myServo, int targetPos) {
    int startPos = myServo.read(); // Get current position
```

```
    // If we need to move UP (e.g. 0 to 90)
    if (startPos < targetPos) {
      for (int i = startPos; i <= targetPos; i++) {
        myServo.write(i);
        delay(moveSpeed);
      }
    }
    // If we need to move DOWN (e.g. 90 to 0)
    else {
      for (int i = startPos; i >= targetPos; i--) {
        myServo.write(i);
        delay(moveSpeed);
      }
    }
  }
```

Arm Construction

This robot arm was assembled using the instructions found on GitHub. The instructions provided cover all parts of assembly from arm construction to wiring. However, the issue that some servo wires are rather short and require splicing in additional wire to reach the Arduino is not addressed. To begin assembly, an MG996R servo is attached inside the base. Depending on the tolerances of the 3D printer, the base may have noticeable gaps. The main arm is attached to the base, and an MG996R servo is then attached to this arm. A rubber band is attached to give extra support. Using the last MG996R servo, the top half of the arm is attached to the bottom half. The claw manipulator is constructed and attached using all three SG90 Micro servos. Additionally, you may wish to use Loctite to secure the fastenings on the claw of the arm to prevent vibrations from loosening the screws. Lastly, you may wish to install a solid exterior base to allow for easier transport and better stability. Below is a wiring diagram to help illustrate how to wire the arm.



Results and Discussion

The main result of this project is a robot arm capable of autonomously operating, able to pick up an object, and move it to a secondary location. During testing, a few issues emerged, with the only major problem being that a wire can disconnect during movement. This issue could have been solved with wires soldered to a circuit board placed in a stable location. The robot's movement could be optimized through various methods and testing. Due to time constraints, neither of these options was possible.

Conclusion

The robot arm functions as intended. It can be controlled via Bluetooth with the app or run autonomously. The robot, though with some consistency issues, can pick up and move six rubber ducks in a minute. The robot arm meets all goals presented, with only minor improvements needed for better reliability.

References

GitHub: https://github.com/cyberturtl/flpoly-roboticsproject?tab=readme-ov-file

Robot Arm Guide: https://howtomechatronics.com/tutorials/arduino/diy-arduino-robot-arm-with-smartphone-control/