Code to Move a Single Servo Moter

// Include the Wire library for I2C communication, which is used to talk to the servo controller.

#include <Wire.h>

// --- Configuration ---

// I2C Slave address for the Robokits controller board.

// This is for servos 1-18. The address is 0x08.

#define SERVO\_CONTROLLER\_1 (16 >> 1)

// The index of the primary servo you want to control via Serial Monitor.

const int SERVO\_INDEX = 0;

// The total number of servos connected to the controller.

const int TOTAL\_SERVOS = 18;

/\*\*

\* @brief Initializes the Arduino, sets other servos to 90 degrees, and starts Serial/I2C.

\*/

void setup() {

// Start serial communication for debugging and receiving commands from the Serial Monitor.

// Make sure to set the Serial Monitor's baud rate to 9600.

Serial.begin(9600);

// Initialize the I2C bus to communicate with the servo controller.

Wire.begin();

// These values configure the I2C clock speed.

// It's best to keep them from your original code as they are specific to your hardware.

TWSR = 3;

TWBR = 18;

// Wait a moment for everything to initialize.

delay(500);

// --- Initialize other servos to 90 degrees ---

Serial.println("Initializing other servos to 90 degrees...");

for (int i = 0; i < TOTAL\_SERVOS; i++) {

// We only want to initialize the servos that are NOT the main one we're controlling.

if (i != SERVO\_INDEX) {

setServo(i, 90);

delay(20); // Small delay between commands to the controller.

}

}

Serial.println("Initialization complete.");

// Also set the primary servo to 90 degrees to start.

setServo(SERVO\_INDEX, 90);

// Print a welcome message to the Serial Monitor to let the user know it's ready.

Serial.println();

Serial.println("--- Servo Control Ready ---");

Serial.println("Controlling Servo 0. Enter an angle (0-180) and press Enter.");

Serial.println(); // Add a blank line for readability.

}

/\*\*

\* @brief Main loop, constantly checks for new input from the Serial Monitor for the primary servo.

\*/

void loop() {

// Check if there is any data available to read from the Serial Monitor.

if (Serial.available() > 0) {

// Read the incoming text as an integer. This will read numbers like "90", "120", etc.

int angle = Serial.parseInt();

// The Serial.parseInt() function might leave a newline character in the buffer.

// This loop clears any remaining characters to prevent misreading the next command.

while (Serial.available() > 0) {

Serial.read();

}

// Echo the command back to the user so they can see what was received.

Serial.print("Received command. Moving Servo ");

Serial.print(SERVO\_INDEX);

Serial.print(" to angle: ");

Serial.println(angle);

// Call the function to move the servo to the specified angle.

setServo(SERVO\_INDEX, angle);

// Add a small delay to prevent spamming commands.

delay(50);

}

}

/\*\*

\* @brief Sends a command to the Robokits I2C servo controller to set a servo's angle.

\*

\* @param index The servo number (0-17).

\* @param angleDeg The target angle in degrees (0-180).

\*/

void setServo(int index, int angleDeg) {

// Constrain the angle to be within the valid range of 0 to 180 degrees.

angleDeg = constrain(angleDeg, 0, 180);

// This logic from your original code reverses the direction of specific servos.

// It's kept here for consistency. It won't affect the 90-degree setting (180-90=90).

if (index == 1 || index == 3 || index == 4 || index == 5) {

angleDeg = 180 - angleDeg;

}

// Map the 0-180 degree angle to the servo's pulse width range in microseconds (500-2500 us).

int pulse\_us = map(angleDeg, 0, 180, 500, 2500);

// The servo controller expects a PWM value, which is calculated from the pulse width.

int pwm\_val = ((pulse\_us - 2) \* 2) - 1000;

// The servo number sent over I2C is 1-based.

uint8\_t servo\_num = index + 1;

// Begin an I2C transmission to the servo controller's address.

Wire.beginTransmission(SERVO\_CONTROLLER\_1);

// Send the data:

Wire.write(servo\_num - 1); // 1. The servo channel on the controller.

Wire.write(pwm\_val >> 8); // 2. The high byte of the PWM value.

Wire.write(pwm\_val & 0xFF); // 3. The low byte of the PWM value.

// End the transmission, sending the data to the controller.

Wire.endTransmission();}

**Controlling all Servo Motors by index and Degree One at A time**

// Include the Wire library for I2C communication, which is used to talk to the servo controller.

#include <Wire.h>

// --- Configuration ---

// I2C Slave address for the Robokits controller board.

// This is for servos 1-18. The address is 0x08.

#define SERVO\_CONTROLLER\_1 (16 >> 1)

// The total number of servos connected to the controller.

const int TOTAL\_SERVOS = 18;

/\*\*

\* @brief Initializes the Arduino, sets up Serial/I2C, and positions servos.

\* - Servos 0 and 1 are set to 90 degrees.

\* - Servos 2 through 17 are set to 90 degrees.

\*/

void setup() {

// Start serial communication for debugging and receiving commands from the Serial Monitor.

// Make sure to set the Serial Monitor's baud rate to 9600.

Serial.begin(9600);

// Initialize the I2C bus to communicate with the servo controller.

Wire.begin();

// These values configure the I2C clock speed.

// It's best to keep them from your original code as they are specific to your hardware.

TWSR = 3;

TWBR = 18;

// Wait a moment for everything to initialize.

delay(500);

// --- Initialize servos ---

Serial.println("Initializing all servos to 90 degrees...");

for (int i = 0; i < TOTAL\_SERVOS; i++) {

setServo(i, 90);

delay(20); // Small delay between commands to the controller.

}

Serial.println("Initialization complete.");

// Print a welcome message to the Serial Monitor to let the user know it's ready.

Serial.println();

Serial.println("--- Servo Control Ready ---");

Serial.println("Enter the Servo Index (0 - 5)");

Serial.println("Enter command as '<servo> <angle>' (e.g., '0 90' or '1 180') and press Enter.");

Serial.println(); // Add a blank line for readability.

}

/\*\*

\* @brief Main loop, constantly checks for new input from the Serial Monitor.

\* Listens for commands in the format "<servo\_index> <angle>".

\*/

void loop() {

// Check if there is any data available to read from the Serial Monitor.

if (Serial.available() > 0) {

// Read the servo index (the first integer in the input).

int servoIndex = Serial.parseInt();

// Check if there is a second number (the angle) waiting.

// We can do this by peeking for a space or a digit.

if (Serial.peek() == ' ') {

// Read the angle (the second integer).

int angle = Serial.parseInt();

// Clear any remaining characters (like newline) from the input buffer.

while (Serial.available() > 0) {

Serial.read();

}

// --- Validate the input ---

// We only want to control servos 0 and 1.

if (servoIndex == 0 || servoIndex == 1 || servoIndex == 2 || servoIndex == 3 || servoIndex == 4 || servoIndex == 5) {

// Echo the command back to the user.

Serial.print("Received command. Moving Servo ");

Serial.print(servoIndex);

Serial.print(" to angle: ");

Serial.println(angle);

// Call the function to move the specified servo to the specified angle.

setServo(servoIndex, angle);

} else {

// If the user enters an invalid servo index, print an error message.

Serial.print("Invalid servo index: ");

Serial.print(servoIndex);

Serial.println(". Please use 0 or 1.");

}

}

// Add a small delay to prevent spamming commands.

delay(50);

}

}

/\*\*

\* @brief Sends a command to the Robokits I2C servo controller to set a servo's angle.

\*

\* @param index The servo number (0-17).

\* @param angleDeg The target angle in degrees (0-180).

\*/

void setServo(int index, int angleDeg) {

// Constrain the angle to be within the valid range of 0 to 180 degrees.

angleDeg = constrain(angleDeg, 0, 180);

// This logic from your original code reverses the direction of specific servos.

// Kept for consistency. Servo 1 will be reversed.

if (index == 1 || index == 3 || index == 4 || index == 5) {

angleDeg = 180 - angleDeg;

}

// Map the 0-180 degree angle to the servo's pulse width range in microseconds (500-2500 us).

int pulse\_us = map(angleDeg, 0, 180, 500, 2500);

// The servo controller expects a PWM value, which is calculated from the pulse width.

int pwm\_val = ((pulse\_us - 2) \* 2) - 1000;

// The servo number sent over I2C is 1-based.

uint8\_t servo\_num = index + 1;

// Begin an I2C transmission to the servo controller's address.

Wire.beginTransmission(SERVO\_CONTROLLER\_1);

// Send the data:

Wire.write(servo\_num - 1); // 1. The servo channel on the controller.

Wire.write(pwm\_val >> 8); // 2. The high byte of the PWM value.

Wire.write(pwm\_val & 0xFF); // 3. The low byte of the PWM value.

// End the transmission, sending the data to the controller.

Wire.endTransmission();

}

**Controlling all servo Motors using Comma Separated Values at a time**

// Include the Wire library for I2C communication, which is used to talk to the servo controller.

#include <Wire.h>

// --- Configuration ---

// I2C Slave address for the Robokits controller board.

// This is for servos 1-18. The address is 0x08.

#define SERVO\_CONTROLLER\_1 (16 >> 1)

// The index of the primary servo you want to control via Serial Monitor.

const int SERVO\_INDEX = 5;

// The total number of servos connected to the controller.

const int TOTAL\_SERVOS = 18;

/\*\*

\* @brief Initializes the Arduino, sets other servos to 90 degrees, and starts Serial/I2C.

\*/

void setup() {

// Start serial communication for debugging and receiving commands from the Serial Monitor.

// Make sure to set the Serial Monitor's baud rate to 9600.

Serial.begin(9600);

// Initialize the I2C bus to communicate with the servo controller.

Wire.begin();

// These values configure the I2C clock speed.

// It's best to keep them from your original code as they are specific to your hardware.

TWSR = 3;

TWBR = 18;

// Wait a moment for everything to initialize.

delay(500);

// --- Initialize other servos to 90 degrees ---

Serial.println("Initializing other servos to 90 degrees...");

for (int i = 0; i < TOTAL\_SERVOS; i++) {

// We only want to initialize the servos that are NOT the main one we're controlling.

if (i != SERVO\_INDEX) {

setServo(i, 90);

delay(20); // Small delay between commands to the controller.

}

}

Serial.println("Initialization complete.");

// Also set the primary servo to 90 degrees to start.

setServo(SERVO\_INDEX, 90);

// Print a welcome message to the Serial Monitor to let the user know it's ready.

Serial.println();

Serial.println("--- Servo Control Ready ---");

Serial.println("Controlling Servo 0. Enter an angle (0-180) and press Enter.");

Serial.println(); // Add a blank line for readability.

}

/\*\*

\* @brief Main loop, constantly checks for new input from the Serial Monitor for the primary servo.

\*/

void loop() {

// Check if there is any data available to read from the Serial Monitor.

if (Serial.available() > 0) {

// Read the incoming text as an integer. This will read numbers like "90", "120", etc.

int angle = Serial.parseInt();

// The Serial.parseInt() function might leave a newline character in the buffer.

// This loop clears any remaining characters to prevent misreading the next command.

while (Serial.available() > 0) {

Serial.read();

}

// Echo the command back to the user so they can see what was received.

Serial.print("Received command. Moving Servo ");

Serial.print(SERVO\_INDEX);

Serial.print(" to angle: ");

Serial.println(angle);

// Call the function to move the servo to the specified angle.

setServo(SERVO\_INDEX, angle);

// Add a small delay to prevent spamming commands.

delay(50);

}

}

/\*\*

\* @brief Sends a command to the Robokits I2C servo controller to set a servo's angle.

\*

\* @param index The servo number (0-17).

\* @param angleDeg The target angle in degrees (0-180).

\*/

void setServo(int index, int angleDeg) {

// Constrain the angle to be within the valid range of 0 to 180 degrees.

angleDeg = constrain(angleDeg, 0, 180);

// This logic from your original code reverses the direction of specific servos.

// It's kept here for consistency. It won't affect the 90-degree setting (180-90=90).

if (index == 1 || index == 3 || index == 4 || index == 5) {

angleDeg = 180 - angleDeg;

}

// Map the 0-180 degree angle to the servo's pulse width range in microseconds (500-2500 us).

int pulse\_us = map(angleDeg, 0, 180, 500, 2500);

// The servo controller expects a PWM value, which is calculated from the pulse width.

int pwm\_val = ((pulse\_us - 2) \* 2) - 1000;

// The servo number sent over I2C is 1-based.

uint8\_t servo\_num = index + 1;

// Begin an I2C transmission to the servo controller's address.

Wire.beginTransmission(SERVO\_CONTROLLER\_1);

// Send the data:

Wire.write(servo\_num - 1); // 1. The servo channel on the controller.

Wire.write(pwm\_val >> 8); // 2. The high byte of the PWM value.

Wire.write(pwm\_val & 0xFF); // 3. The low byte of the PWM value.

// End the transmission, sending the data to the controller.

Wire.endTransmission();

}

Communicating with Robotic Arm using Python and Adriano Uno one motor at a timee

**Python Code:**

import serial

import time

# --- Configuration ---

SERIAL\_PORT = 'COM6'

BAUD\_RATE = 9600

def connect\_to\_arduino(port, baud\_rate):

    """

    Establishes a serial connection and listens for initial messages.

    """

    try:

        print(f"Connecting to Arduino on {port} at {baud\_rate} bps...")

        # Add a write\_timeout to prevent the script from freezing indefinitely.

        ser = serial.Serial(port, baud\_rate, timeout=1, write\_timeout=1)

        # Give the Arduino time to reset and start sending messages.

        print("Connection established. Waiting for Arduino to boot...")

        time.sleep(2)

        # Read any startup messages from the Arduino (like setup errors).

        print("--- Arduino Startup Log ---")

        while ser.in\_waiting > 0:

            try:

                line = ser.readline().decode('utf-8').strip()

                if line:

                    print(line)

            except Exception as e:

                print(f"Error reading line: {e}")

        print("---------------------------")

        print("\nConnection successful!")

        return ser

    except serial.SerialException as e:

        print(f"Error: Could not open serial port {port}.")

        print(f"Details: {e}")

        print("Please ensure the Arduino is connected, the correct port is selected,")

        print("and no other program (like the Arduino IDE's Serial Monitor) is using it.")

        return None

def send\_command(serial\_connection, servo\_index, angle):

    """

    Formats and sends a command to the Arduino to move a servo.

    """

    if serial\_connection and serial\_connection.is\_open:

        command = f"{servo\_index},{angle}\n"

        try:

            # Encode the string to bytes and send it.

            serial\_connection.write(command.encode('utf-8'))

            print(f"Sent: Move Servo {servo\_index} to {angle}°")

            # Wait for a confirmation response from the Arduino.

            response = serial\_connection.readline().decode('utf-8').strip()

            if response:

                print(f"Arduino says: {response}")

            else:

                print("No response from Arduino. The servo may or may not have moved.")

        except serial.SerialTimeoutException:

             print("Error: Write to serial port timed out. The Arduino may be frozen.")

        except serial.SerialException as e:

            print(f"Error writing to serial port: {e}")

    else:

        print("Cannot send command: Serial connection is not open.")

def main():

    """

    Main function to run the command-line interface for controlling the arm.

    """

    arduino\_ser = connect\_to\_arduino(SERIAL\_PORT, BAUD\_RATE)

    if not arduino\_ser:

        return # Exit if connection failed

    print("\n--- Python Servo Controller ---")

    print("Enter commands in the format 'servo,angle' (e.g., '5,90').")

    print("Type 'exit' to quit.")

    try:

        while True:

            user\_input = input("\nEnter command: ")

            if user\_input.lower() == 'exit':

                print("Closing serial port and exiting.")

                break

            try:

                parts = user\_input.split(',')

                if len(parts) != 2:

                    raise ValueError("Invalid format. Please use 'servo,angle'.")

                servo\_id = int(parts[0].strip())

                angle\_val = int(parts[1].strip())

                if not (0 <= servo\_id < 18):

                     raise ValueError("Servo index must be between 0 and 17.")

                if not (0 <= angle\_val <= 180):

                     raise ValueError("Angle must be between 0 and 180.")

                send\_command(arduino\_ser, servo\_id, angle\_val)

            except ValueError as e:

                print(f"Error: {e}")

    finally:

        if arduino\_ser and arduino\_ser.is\_open:

            arduino\_ser.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Arduino Code (C++)**

// Include the Wire library for I2C communication with the servo controller.

#include <Wire.h>

// --- Configuration ---

// I2C Slave address for the Robokits controller board.

#define SERVO\_CONTROLLER\_1 (16 >> 1)

// The total number of servos connected to the controller.

const int TOTAL\_SERVOS = 18;

// A string to store the incoming command from Python.

String serialCommand;

/\*\*

\* @brief Initializes the Arduino, sets servos to 90 degrees, and starts Serial/I2C.

\*/

void setup() {

// Start serial communication for receiving commands from Python.

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

// Initialize the I2C bus to communicate with the servo controller.

Wire.begin();

// These values configure the I2C clock speed.

TWSR = 3;

TWBR = 18;

delay(500);

Serial.println("Arduino boot complete. Initializing servos...");

// --- Initialize all servos to a default 90-degree position ---

for (int i = 0; i < TOTAL\_SERVOS; i++) {

// We pass 'true' to indicate this is a setup check.

setServo(i, 90, true);

delay(20);

// If we have an I2C error, we shouldn't continue trying.

// The error will be printed inside setServo.

}

Serial.println("\nInitialization complete. Ready for commands.");

}

/\*\*

\* @brief Main loop. Listens for and processes commands sent from Python.

\*/

void loop() {

if (Serial.available() > 0) {

char incomingChar = Serial.read();

if (incomingChar == '\n') {

Serial.print("Command received: ");

Serial.println(serialCommand);

int commaIndex = serialCommand.indexOf(',');

if (commaIndex != -1) {

String indexStr = serialCommand.substring(0, commaIndex);

String angleStr = serialCommand.substring(commaIndex + 1);

int servoIndex = indexStr.toInt();

int angle = angleStr.toInt();

// We pass 'false' because this is a regular command, not a setup check.

setServo(servoIndex, angle, false);

Serial.print("Moved servo ");

Serial.print(servoIndex);

Serial.print(" to ");

Serial.print(angle);

Serial.println(" degrees.");

} else {

Serial.println("Error: Invalid command format. Expected 'index,angle'.");

}

serialCommand = "";

} else {

if (incomingChar != '\r') {

serialCommand += incomingChar;

}

}

}

}

/\*\*

\* @brief Sends a command to the I2C servo controller and CHECKS for errors.

\* \* @param index The servo number (0-17).

\* @param angleDeg The target angle in degrees (0-180).

\* @param isSetupCheck A flag to print a more specific error during setup.

\*/

void setServo(int index, int angleDeg, bool isSetupCheck) {

angleDeg = constrain(angleDeg, 0, 180);

if (index == 1 || index == 3 || index == 4 || index == 5) {

angleDeg = 180 - angleDeg;

}

int pulse\_us = map(angleDeg, 0, 180, 500, 2500);

int pwm\_val = ((pulse\_us - 2) \* 2) - 1000;

uint8\_t servo\_num = index + 1;

Wire.beginTransmission(SERVO\_CONTROLLER\_1);

Wire.write(servo\_num - 1);

Wire.write(pwm\_val >> 8);

Wire.write(pwm\_val & 0xFF);

// \*\*\* THIS IS THE IMPORTANT NEW PART \*\*\*

// Check the status of the I2C transmission.

byte error = Wire.endTransmission();

if (error != 0) {

Serial.print("I2C Error on Servo ");

Serial.print(index);

Serial.print(". Error code: ");

Serial.println(error);

Serial.println("--> CHECK WIRING AND POWER for the servo controller!");

// During setup, we only want to show the error once.

if (isSetupCheck) {

// This will hang the loop so you just see the error message.

while(1);

}

}

}

Communicating with Robotic Arm using Python and Adriano Uno All motor at a timee

**Python Code:**

import serial

import time

# --- Configuration ---

# Update this to your Arduino's actual COM port

SERIAL\_PORT = 'COM6'

BAUD\_RATE = 960090

# This must match the TOTAL\_SERVOS constant in your Arduino sketch

# Changed from 18 to 6 as requested.

TOTAL\_SERVOS = 6

def connect\_to\_arduino(port, baud\_rate):

    """

    Establishes a serial connection and listens for initial messages.

    """

    try:

        print(f"Connecting to Arduino on {port} at {baud\_rate} bps...")

        # Add a write\_timeout to prevent the script from freezing indefinitely.

        ser = serial.Serial(port, baud\_rate, timeout=2, write\_timeout=2)

        # Give the Arduino time to reset and start sending messages.

        print("Connection established. Waiting for Arduino to boot...")

        time.sleep(2)

        # Read any startup messages from the Arduino (like setup errors).

        print("--- Arduino Startup Log ---")

        while ser.in\_waiting > 0:

            try:

                line = ser.readline().decode('utf-8').strip()

                if line:

                    print(line)

            except Exception as e:

                print(f"Error reading line: {e}")

        print("---------------------------")

        print("\nConnection successful!")

        return ser

    except serial.SerialException as e:

        print(f"Error: Could not open serial port {port}.")

        print(f"Details: {e}")

        print("Please ensure the Arduino is connected, the correct port is selected,")

        print("and no other program (like the Arduino IDE's Serial Monitor) is using it.")

        return None

def send\_positions\_command(serial\_connection, positions\_string):

    """

    Formats and sends a command string of all servo positions to the Arduino.

    Args:

        serial\_connection: The active PySerial object.

        positions\_string (str): A comma-separated string of angles.

    """

    if serial\_connection and serial\_connection.is\_open:

        # The command is already formatted, just add the newline terminator.

        command = f"{positions\_string}\n"

        try:

            # Encode the string to bytes and send it.

            serial\_connection.write(command.encode('utf-8'))

            print(f"Sent positions: {positions\_string}")

            # Wait for a confirmation response from the Arduino.

            response = serial\_connection.readline().decode('utf-8').strip()

            if response:

                print(f"Arduino says: {response}")

            else:

                print("No response from Arduino.")

        except serial.SerialTimeoutException:

             print("Error: Write to serial port timed out. The Arduino may be frozen.")

        except serial.SerialException as e:

            print(f"Error writing to serial port: {e}")

    else:

        print("Cannot send command: Serial connection is not open.")

def main():

    """

    Main function to run the command-line interface for controlling the arm.

    """

    arduino\_ser = connect\_to\_arduino(SERIAL\_PORT, BAUD\_RATE)

    if not arduino\_ser:

        return # Exit if connection failed

    print("\n--- Python Multi-Servo Controller ---")

    print(f"Enter {TOTAL\_SERVOS} comma-separated angle values (e.g., '90,45,180,...').")

    print("The first value is for servo 0, the second for servo 1, and so on.")

    print("Type 'exit' to quit.")

    try:

        while True:

            user\_input = input("\nEnter all servo positions: ")

            if user\_input.lower() == 'exit':

                print("Closing serial port and exiting.")

                break

            try:

                # Remove any accidental whitespace from the input string.

                user\_input = user\_input.replace(" ", "")

                parts = user\_input.split(',')

                # Validate that the correct number of positions were entered.

                if len(parts) != TOTAL\_SERVOS:

                    raise ValueError(f"Invalid input. Please provide exactly {TOTAL\_SERVOS} values. You provided {len(parts)}.")

                # Validate that each value is a number between 0 and 180.

                for part in parts:

                    angle = int(part)

                    if not (0 <= angle <= 180):

                        raise ValueError(f"Invalid angle '{angle}'. All angles must be between 0 and 180.")

                # If all validation passes, send the command string to the Arduino.

                send\_positions\_command(arduino\_ser, user\_input)

            except ValueError as e:

                # Catch errors from parsing (e.g., text instead of numbers) or validation.

                print(f"Input Error: {e}")

    finally:

        if arduino\_ser and arduino\_ser.is\_open:

            arduino\_ser.close()

            print("Serial port closed.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Ardino Uno Code:**

// Include the Wire library for I2C communication with the servo controller.

#include <Wire.h>

// --- Configuration ---

// I2C Slave address for the Robokits controller board.

#define SERVO\_CONTROLLER\_1 (16 >> 1)

// The total number of servos connected to the controller.

const int TOTAL\_SERVOS = 6;

// A string to store the incoming command from Python.

String serialCommand;

/\*\*

\* @brief Initializes the Arduino, sets servos to 90 degrees, and starts Serial/I2C.

\*/

void setup() {

// Start serial communication for receiving commands from Python.

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

// Initialize the I2C bus to communicate with the servo controller.

Wire.begin();

// These values configure the I2C clock speed.

TWSR = 3;

TWBR = 18;

delay(500);

Serial.println("Arduino boot complete. Initializing servos...");

// --- Initialize all servos to a default 90-degree position ---

for (int i = 0; i < TOTAL\_SERVOS; i++) {

// We pass 'true' to indicate this is a setup check.

setServo(i, 90, true);

delay(20);

// If we have an I2C error, the program will halt and display an error.

}

Serial.println("\nInitialization complete. Ready for commands.");

}

/\*\*

\* @brief Main loop. Listens for and processes a comma-separated string of servo positions.

\*/

void loop() {

if (Serial.available() > 0) {

char incomingChar = Serial.read();

// Check if the character is a newline, which signifies the end of a command.

if (incomingChar == '\n') {

Serial.print("Command string received: ");

Serial.println(serialCommand);

int servoIndex = 0;

int lastIndex = 0;

// Loop through the command string to find commas and parse values.

for (int i = 0; i < serialCommand.length(); i++) {

if (serialCommand.charAt(i) == ',') {

// Extract the value between the last comma (or start) and this one.

String angleStr = serialCommand.substring(lastIndex, i);

int angle = angleStr.toInt();

// Move the servo if the index is within our total servo count.

if (servoIndex < TOTAL\_SERVOS) {

setServo(servoIndex, angle, false);

}

servoIndex++;

lastIndex = i + 1; // Update the starting point for the next value.

}

}

// Handle the very last value in the string (after the last comma).

String angleStr = serialCommand.substring(lastIndex);

int angle = angleStr.toInt();

if (servoIndex < TOTAL\_SERVOS) {

setServo(servoIndex, angle, false);

}

// Send one confirmation message after all servos are moved.

Serial.println("All servo positions updated based on command.");

// Clear the command string to be ready for the next one.

serialCommand = "";

} else {

// If it's not a newline, append the character to our command string.

// We also ignore carriage returns for better cross-platform compatibility.

if (incomingChar != '\r') {

serialCommand += incomingChar;

}

}

}

}

/\*\*

\* @brief Sends a command to the I2C servo controller and CHECKS for errors.

\* \* @param index The servo number (0-17).

\* @param angleDeg The target angle in degrees (0-180).

\* @param isSetupCheck A flag to print a more specific error during setup.

\*/

void setServo(int index, int angleDeg, bool isSetupCheck) {

angleDeg = constrain(angleDeg, 0, 180);

if (index == 1 || index == 3 || index == 4 || index == 5) {

angleDeg = 180 - angleDeg;

}

int pulse\_us = map(angleDeg, 0, 180, 500, 2500);

int pwm\_val = ((pulse\_us - 2) \* 2) - 1000;

uint8\_t servo\_num = index + 1;

Wire.beginTransmission(SERVO\_CONTROLLER\_1);

Wire.write(servo\_num - 1);

Wire.write(pwm\_val >> 8);

Wire.write(pwm\_val & 0xFF);

byte error = Wire.endTransmission();

if (error != 0) {

Serial.print("I2C Error on Servo ");

Serial.print(index);

Serial.print(". Error code: ");

Serial.println(error);

Serial.println("--> CHECK WIRING AND POWER for the servo controller!");

if (isSetupCheck) {

// This will hang the program so you only see the error message on startup.

while(1);

}

}

}