Servo Motor Control with 8051 M.C.

MCI Project Report

Course- 2EC701CC23 -Microcontroller and Interfacing

B. Tech. Semester IV

Submitted by:

Roll No: - 23BEC026 [Budania Niti Mahavir]

Roll No: - 23BEC027 [Vansh Champaneri]

School of Engineering
Institute of Technology
Nirma University
Ahmedabad 382481

April 2025

Abstract

Servo motors find extensive applications in robotics, automation, and industries because they have accurate control over position. In this project, an attempt has been made to control a servo motor with the help of an 8051 microcontroller (AT89C51) through the generation of Pulse Width Modulation (PWM) signals. The servo motor is set to rotate to three specific positions: 0°, 90°, and 180°, with a delay of 1 second between rotations.

A PWM signal is formed by using Timer 0 with Mode 1, where dedicated TH0 and TL0 are loaded to yield the desired pulse width for the respective angle. The servo control pin is high for the specified time (2ms, 1.5ms, or 1ms) and made low by the 8051 microcontroller in order to result in smooth motion.

The circuit is simulated in Proteus to generate waveforms accurately. The project showcases precise servo positioning and emphasizes the efficiency of 8051 timers for PWM-based control. The project has applications in robotic arms, automated systems, and mechatronics.

Peripheral Used :

- ✓ Microcontroller - AT89C51 (8051)
- ✓ Servo Motor - SG90 (0° - 180° rotation)
- ✓ Crystal Oscillator - 11.0592 MHz
- ✓ Capacitors - 22pF
- ✓ Resistors - 10kΩ
- ✓ Power Supply - 5V DC

Circuit Diagram:

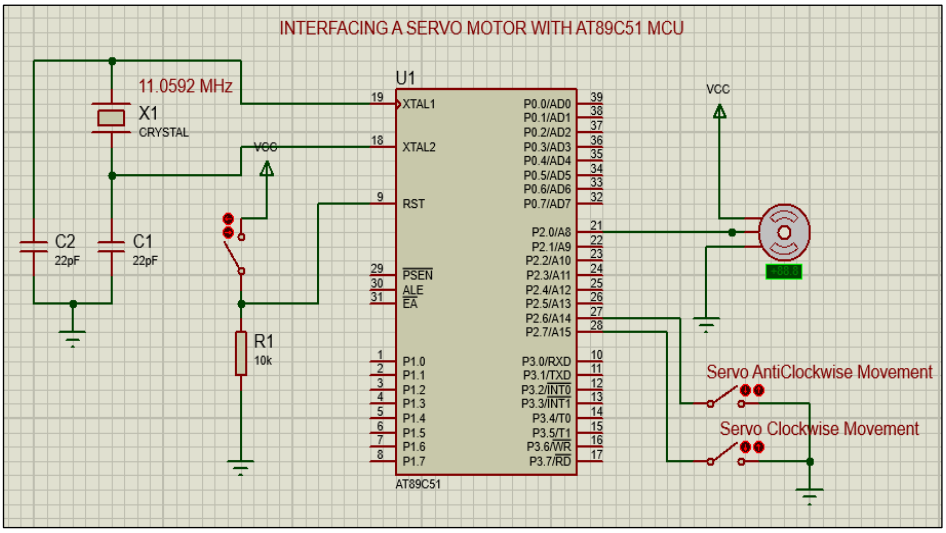


Figure 1 - Circuit Diagram

Working Principle:

A servo motor is a type of actuator that is controlled by a Pulse Width Modulation (PWM) signal. The angle of the servo is determined by the width of the PWM pulse, while the signal frequency remains constant at 50 Hz (20 ms period).

To control the servo motor, the 8051 microcontroller generates a PWM signal using Timer 0 in Mode 1 (16-bit mode). The ON duration of the signal (HIGH pulse) is varied to position the servo at different angles.

The standard PWM pulse widths for controlling a servo motor are:

Servo Angle	ON Time (Pulse Width)	OFF Time (for 20ms total)
0°	1.0 ms (1000 µs)	19.0 ms
90°	1.5 ms (1500 µs)	18.5 ms
180°	2.0 ms (2000 µs)	18.0 ms

In the program, we adjust the Timer 0 register values (TH0, TL0) to generate these specific pulse durations.

Timer-Based PWM Generation in 8051

Timer 0 operates in Mode 1 (16-bit timer mode) to generate the required delay for the HIGH pulse.

When the timer starts, the servo pin P2.0 is set HIGH.

The timer counts for the required ON duration (1 ms, 1.5 ms, or 2 ms).

Once the timer overflows, the servo pin is set LOW and remains LOW for the rest of the 20 ms period.

This process repeats continuously to maintain the servo position.

Timer Value Calculations:

The timer count is calculated using:

$$Timer\ Value = 65536 - (\frac{Delay\ in\ Microseconds}{Machine\ Cycles})$$

Where the 8051 machine cycle = 1.085 µs (for an 11.0592 MHz crystal).

The preloaded timer values for different angles are:

Angle	Pulse Width	Timer Value	TH0	TL0
-90°	1.0 ms	65536 - 1000 = 64536	0FCH	66H
0° (neutral)	1.5 ms	65536 - 1500 = 64036	0FAH	A0H
+90°	2.0 ms	65536 - 2000 = 63536	0F7H	4DH

Working of the Program

1. Timer Initialization:

   The program starts by setting Timer 0 in Mode 1 (16-bit timer mode) using:

   MOV TMOD, #01H

2. Button Monitoring:

   The program continuously checks the status of two buttons:

   P2.6 for Clockwise Rotation (i.e., +90°)

   P2.7 for Anti-Clockwise Rotation (i.e., -90°)

   If no button is pressed, the servo is moved to the center position (0°) by default.

3. Angle Subroutines:

   Depending on the button pressed, one of the following subroutines is called:

   ninety_degrees: Sends a 2ms pulse → moves servo to +90°

   negative_ninety_degrees: Sends a 1ms pulse → moves servo to -90°

   zero_degrees: Sends a 1.5ms pulse → moves servo to 0°

4. Generating the PWM Pulse:

   Inside each subroutine:

   Set P2.0 HIGH to start the PWM pulse (servo signal pin).

Load Timer 0 registers TH0 and TL0 with pre-calculated values for the required delay.

Start Timer 0 by setting TR0 = 1.

Wait for overflow flag (TF0) to indicate the desired ON time has passed.

Clear P2.0 to LOW to end the pulse.

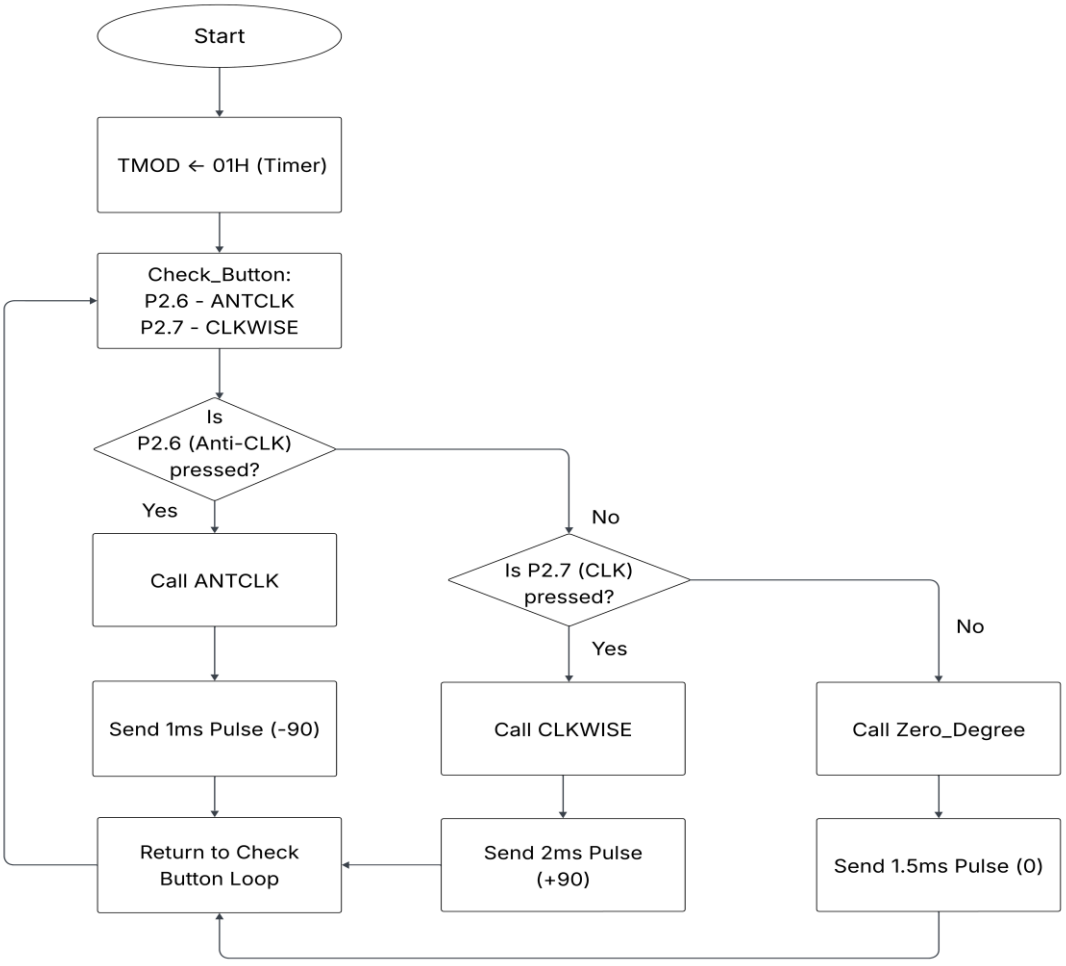
Stop Timer 0 and clear the TF0 flag to prepare for the next operation.

5. Continuous Monitoring:

After executing the respective subroutine, the program jumps back to check the buttons again.

The entire process repeats continuously, allowing real-time response to button presses.

Flowchart:



Assembly Code Explanation

1.1 Timer Initialization:

Start of Program

ORG 00H

Sets the starting address of the program memory to 0x00, the reset vector of the 8051 microcontroller.

1.2 Main Program Block:

MAIN:

MOV TMOD, #01H

TMOD is the Timer Mode Register.

#01H sets Timer 0 in Mode 1, which is a 16-bit timer mode.

Bit 1-0 of TMOD: 01 → Timer 0 Mode 1

This allows delays up to 65536 counts.

1.3 Button Check Loop:

CHECK_BUTTONS:

JNB P2.6, ANTCLK

JNB P2.7, CLKWISE

JNB: Jump if Bit is Not set (i.e., if bit = 0).

- P2.6 and P2.7 are buttons:
  - If P2.6 is pressed (logic 0), jump to ANTCLK (Anti-clockwise).
  - If P2.7 is pressed, jump to CLKWISE.

1.4 Default Action: 0° Position:

LCALL zero_degrees

SJMP CHECK_BUTTONS

If no button is pressed, call zero_degrees (servo at neutral position).

SJMP loops back to check buttons continuously.

1.5 Clockwise Rotation (+90°):

```
CLKWISE:
   LCALL ninety_degrees
   SJMP CHECK_BUTTONS
```

Calls ninety_degrees subroutine (generates 2 ms pulse).

Loops back to button check.

1.6 Anti-Clockwise Rotation (−90°):

```
ANTCLK:
   LCALL negative_ninety_degrees
   SJMP CHECK_BUTTONS
```

Calls negative_ninety_degrees subroutine (generates 1 ms pulse).

Loops back to button check.

1.7 Subroutines (Servo Pulse Control):

A. Negative 90° (1 ms pulse)

```
negative_ninety_degrees:
   MOV TH0, #0FCH
   MOV TL0, #66H
   SETB P2.0        ; Start pulse (HIGH)
   SETB TR0         ; Start Timer 0
WAIT1:
   JNB TF0, WAIT1   ; Wait until Timer Overflows (TF0 = 1)
   CLR P2.0         ; End pulse (LOW)
   CLR TF0          ; Clear Timer Flag
```

```
    CLR TR0         ; Stop Timer
    RET
```

Sets timer for 1 ms delay (FC66H = 64536 → 65536 - 1000).

Sends 1 ms HIGH pulse on P2.0 → corresponds to -90° on servo.

B. Zero Degrees (1.5 ms pulse)

```
zero_degrees:
    MOV TH0, #0FAH
    MOV TL0, #0A0H
    SETB P2.0
    SETB TR0
WAIT2:
    JNB TF0, WAIT2
    CLR P2.0
    CLR TF0
    CLR TR0
    RET
```

Sets timer for 1.5 ms delay (F9A0H = 63936 → 65536 - 1500).

Pulse for servo to move to neutral (0°) position.

C. +90 Degrees (2 ms pulse)

```
ninety_degrees:
    MOV TH0, #0F7H
    MOV TL0, #4DH
    SETB P2.0
    SETB TR0
```

WAIT3:

JNB TF0, WAIT3

CLR P2.0

CLR TF0

CLR TR0

RET

Sets timer for 2 ms delay (F74DH = 63581 → 65536 - 2000).

Sends 2 ms HIGH pulse to rotate servo to +90°.

1.8 End of Program:

END

Indicates the end of the source file to the assembler.

Arduino Code Explanation:

1. **Library and Constant Definitions:**
   - The program includes the MsTimer2 library, which is used to generate timer interrupts every 20 milliseconds, simulating the servo PWM signal.
   - SERVO is defined as pin 2, connected to the servo signal wire.
   - PWM_RANGE_MIN and PWM_RANGE_MAX define the pulse width range (in microseconds) for 0° to 180° rotation.
2. **Setup Function:**
   - pinMode(2, OUTPUT) sets the servo pin as output.

- MsTimer2::set(20, setServo) sets a 20ms period to regularly call the setServo() function — the standard servo refresh rate.
- Serial communication is initialized at 115200 baud to take user input from the Serial Monitor.

3. **Loop Function:**
   - The loop checks for a flag that becomes true when a full angle input is received over serial.
   - The input string strAngle is converted into an integer (temp) and checked to ensure it lies between 0 and 180 degrees.
   - The entered angle is mapped to a corresponding PWM pulse width using the map() function:

dutyHIGH = map(temp, 0, 180, 544, 2400);

     - For example:
       - 0° → 544µs pulse
       - 90° → ~1500µs pulse
       - 180° → 2400µs pulse
   - This pulse width value is stored in the dutyHIGH variable and will be used in the next PWM cycle.

4. **Timer Interrupt Function (setServo):**
   - This function is called every **20ms** (standard servo control interval).
   - It sets pin 2 HIGH, waits for dutyHIGH microseconds (pulse duration), then sets the pin LOW to complete the PWM pulse.

- This produces the required control signal to rotate the servo to the desired position.
-

5. **Serial Event Function:**
   - This function reads incoming serial data **character by character**.
   - If numeric digits (0–9) are received, they are added to strAngle.
   - When a newline character (\n) is received, it indicates the **end of input**, and the flag is set to true so the loop can process the angle.

Simulation:

We have done simulation for both assembly and Arduino. For assembly code simulation is done in Proteus software. And for Arduino Simulation is done in Wokwi Site.

Proteus :

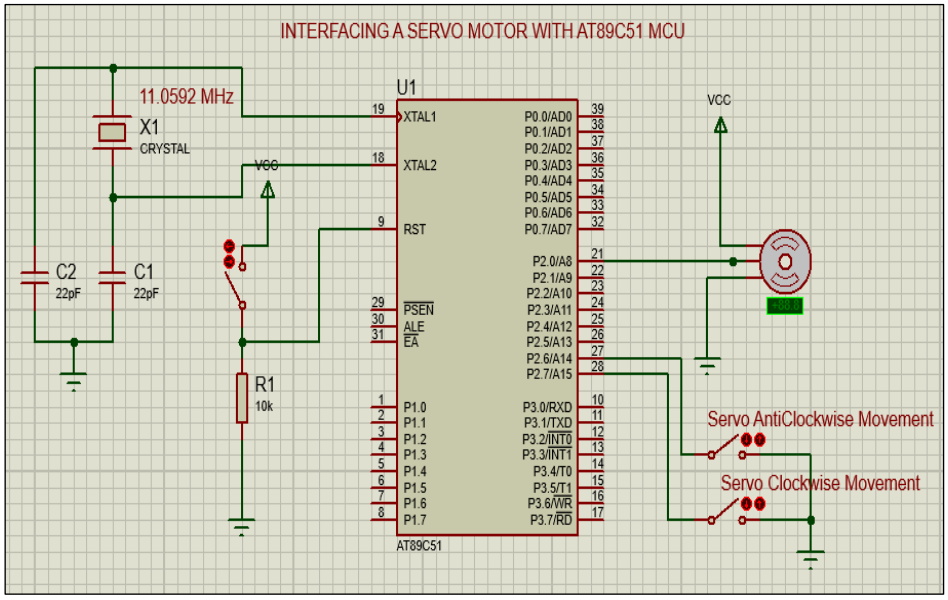


Figure 2- Circuit Diagram

Observaiton:

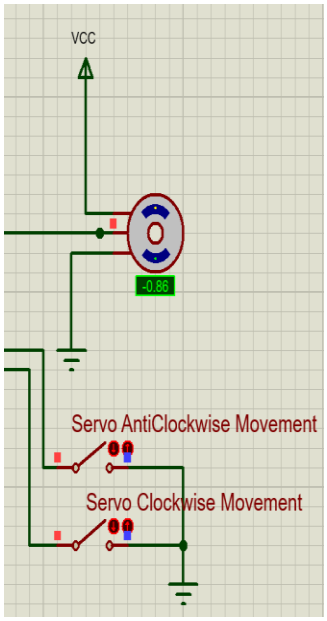


Figure 3 - 0 Positon

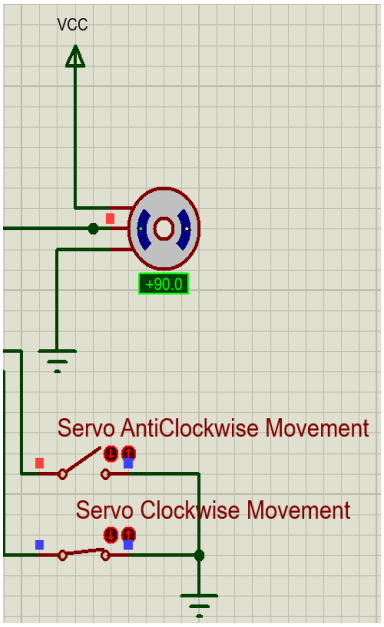


Figure 4 – +90 Position

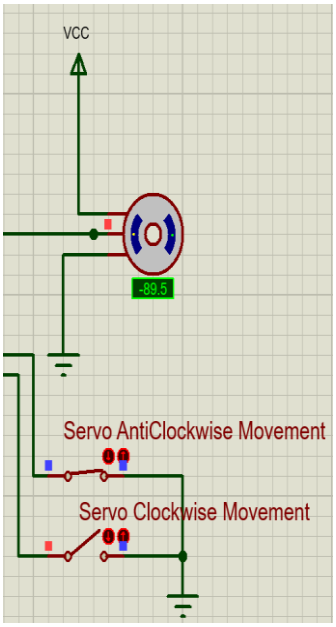


Figure 5 - -90 Position

Wokwi:

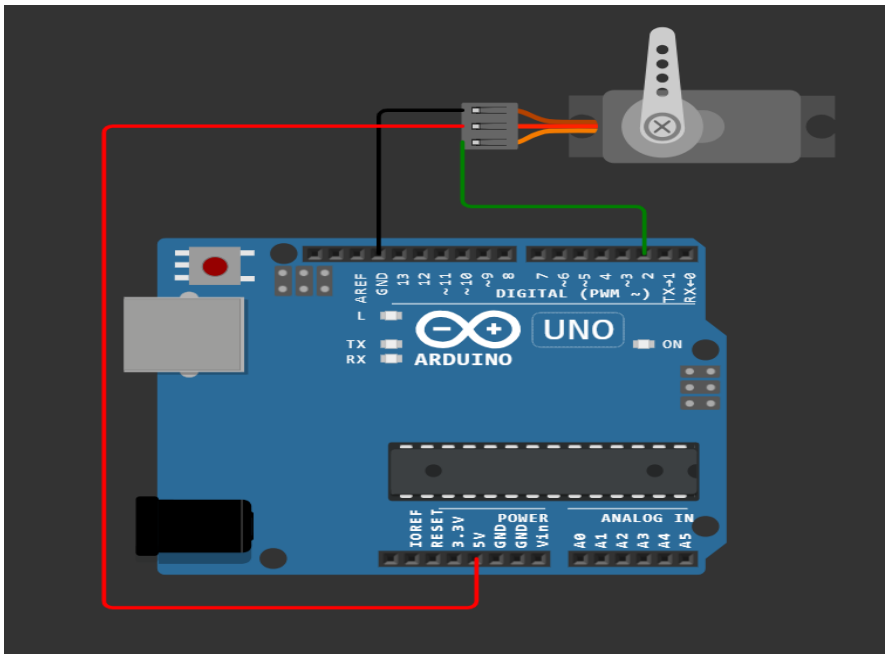


Figure 6 - Circuit Diagram

Observaiton:

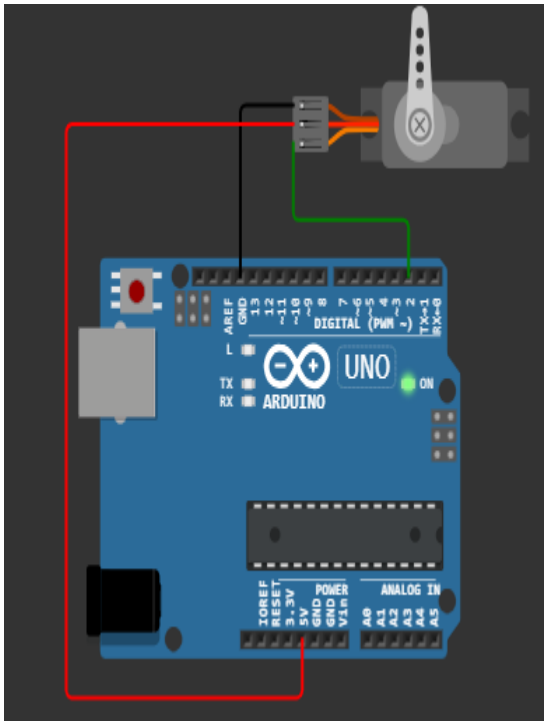


Figure 7 - 0 Position

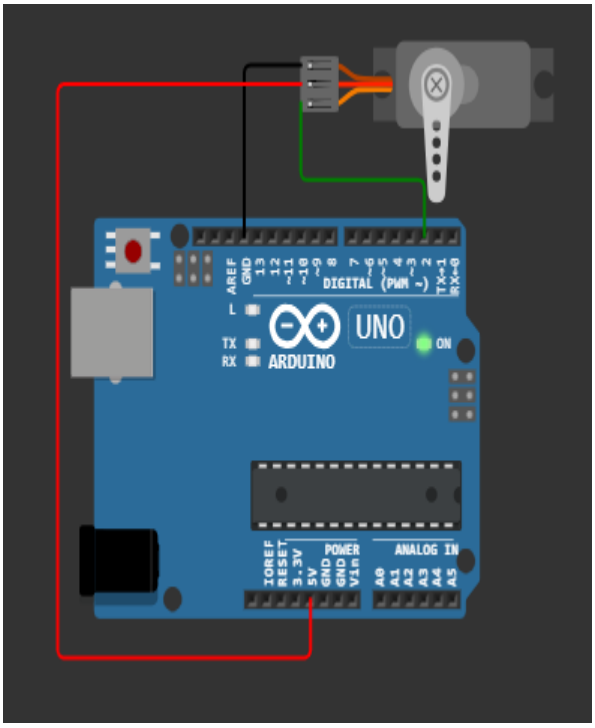


Figure 8 – 180 Position

Application:

Servo motors are widely used in various applications that require precise position control. The implementation of servo motor control using the 8051 microcontroller and PWM signals can be applied in numerous fields. Below are some key applications:

1. Robotics and Automation

- Robotic Arms: Used in industrial automation for pick and place operations.
- Humanoid Robots: Controls joints to mimic human-like movements.
- Autonomous Vehicles: Controls steering mechanisms in robotic cars.

2. Industrial Applications

- CNC Machines: Used for precise positioning of cutting tools.
- Conveyor Belt Systems: Controls the movement and sorting of objects.
- Automated Assembly Lines: Ensures accuracy in repetitive tasks.

3. Aerospace and Defense

- Missile Guidance Systems: Used for trajectory adjustments.
- Radar Positioning Systems: Controls the orientation of radar antennas.
- Unmanned Aerial Vehicles (UAVs): Adjusts flaps and control surfaces.

4. Medical Equipment

- Surgical Robots: Helps in precision-based medical procedures.
- Prosthetic Limbs: Enables smooth movement of artificial limbs.
- MRI and CT Scanners: Moves the scanner bed accurately.

5. Smart Home and IoT

- Automatic Door Openers: Detects presence and opens/closes doors.
- Home Security Systems: Controls surveillance cameras for tracking movement.
- Smart Windows and Blinds: Adjusts position based on light intensity.

6. Educational and Research Applications

- Electronics and Embedded Systems Training: Helps students learn about PWM control.
- Research on Control Systems: Used for experimentation and testing.
- DIY Projects: Common in Arduino and microcontroller-based projects.

Advantage & Limitation:

Advantages :

1. Precise Position Control
    - The PWM-based control ensures accurate positioning of the servo motor.
2. High Efficiency
    - Consumes power only when required, making it energy-efficient.
3. Fast Response Time
    - The servo motor can quickly reach the desired position with minimal lag.
4. Compact and Lightweight

- Servo motors are small and lightweight, making them ideal for embedded applications.

5. Simple Interface with 8051 Microcontroller
    - Requires minimal external components, reducing circuit complexity.
6. Wide Range of Applications
    - Used in robotics, automation, automotive systems, and smart home devices.
7. Stable and Reliable
    - Servo motors hold their position even when external force is applied.

Limitations :

1. Limited Rotation Angle
    - Most servo motors can only rotate between 0° and 180°, restricting applications that require continuous rotation.
2. Power Consumption
    - Servo motors consume more power compared to stepper motors for maintaining position.
3. Microcontroller Processing Load
    - The 8051 must continuously generate PWM signals, which can limit its ability to perform other tasks simultaneously.
4. External Power Requirement
    - High-torque servo motors require an external power supply, as the 8051 alone cannot provide enough current.
5. Limited Torque 🔧
    - Standard hobby servos may not provide sufficient torque for heavy-duty applications.
6. Positioning Error Due to External Load
    - If an external force is applied, the servo may not maintain its exact position without additional feedback mechanisms.

7. Noise and Heat Generation
    - Continuous operation can lead to heat buildup and mechanical wear, reducing motor lifespan.

Conclusion:

The use of servo motor control through the 8051 microcontroller illustrates a basic yet efficient means of producing precise angular motion. Through the generation of PWM signals, the microcontroller is able to effectively control the servo motor to attain positions of user-defined input.

This system is cost-effective, energy-efficient, and extensively suitable for use in fields like automation, robotics, and industrial control systems. However, it too has some shortcomings, such as limited rotation angles, power levels, and processor load on the microcontroller.

As a whole, the project sets the stage for understanding PWM-based motor control and is a prelude to more complex motor control implementations. Possible future expansions include adding feedback, multi-servo control, or combining with wireless communication for remote control.

Reference:

Ref. Book :

Mazidi, M.A., Mazidi, J.G., & McKinlay, R.D. (2007). The 8051 Microcontroller and Embedded Systems: Using Assembly and C. Pearson Education.

Datasheet of 8051 Microcontroller:

https://ww1.microchip.com/downloads/en/DeviceDoc/doc4383.pdf

Servo Motor Datasheet:

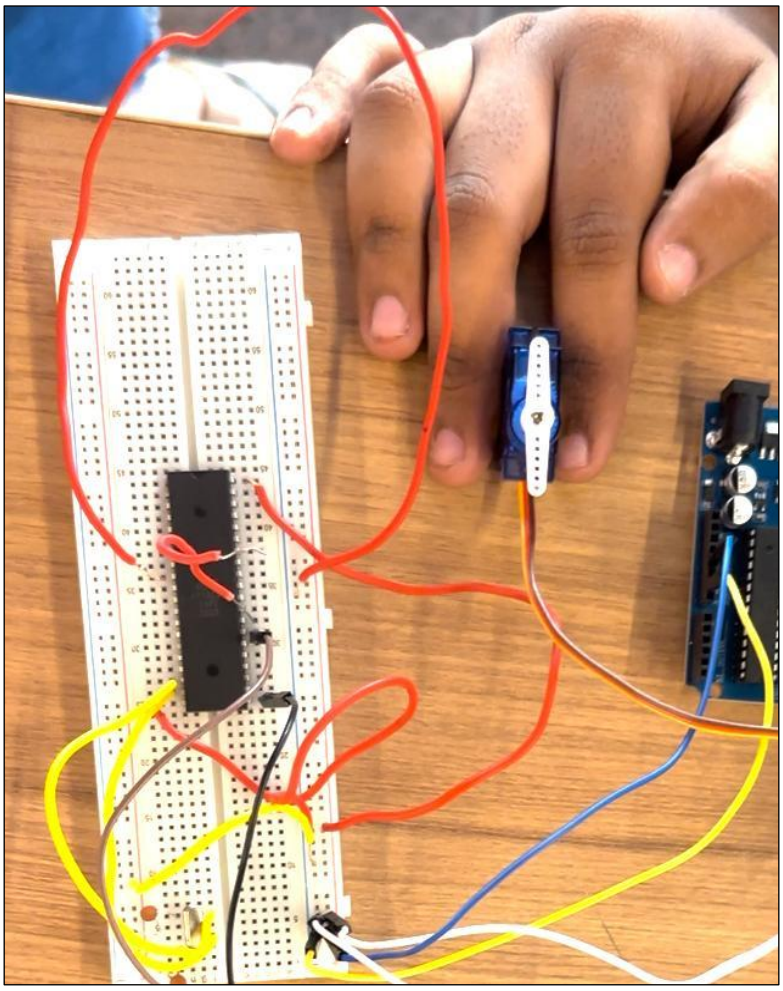
https://robojax.com/learn/arduino/robojaxservosg90_datasheet.pdf?srsltid=AfmBOoo_7NyKsj65PzGZlnE3eLm2xThqu9V6sEBzD5KSkWIz7d4lYMxP

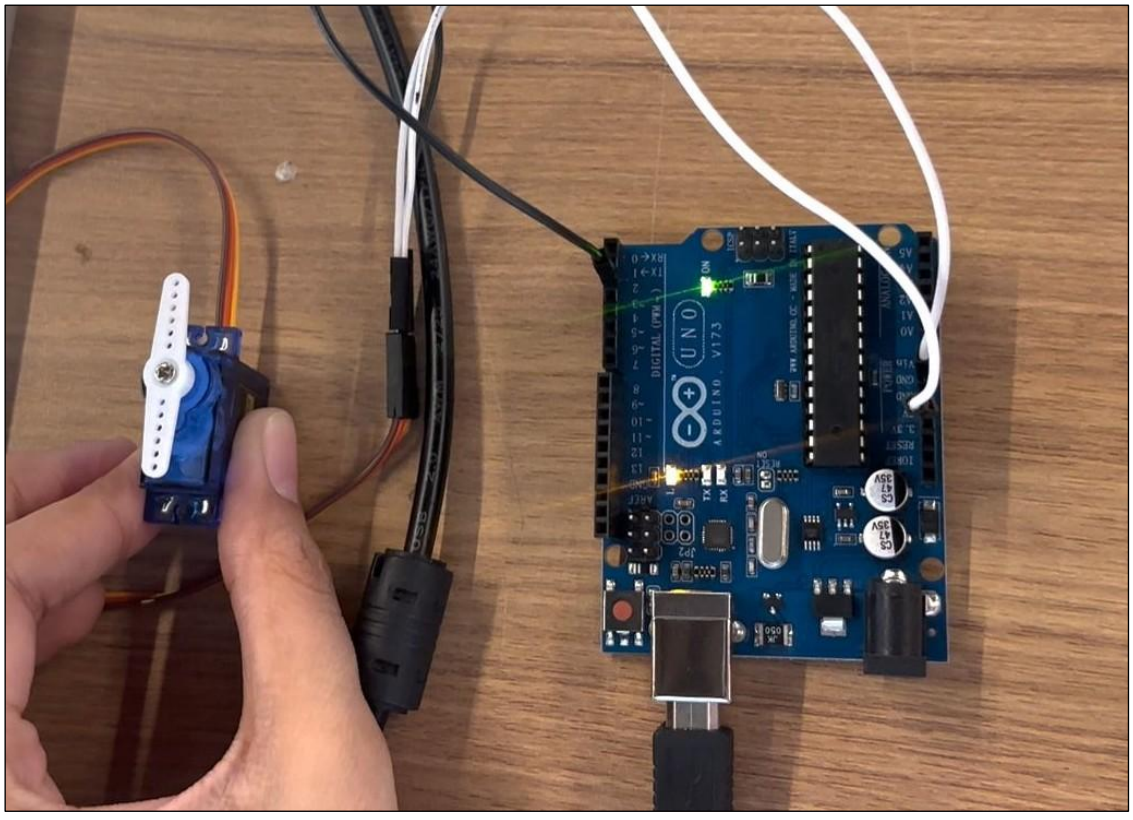
Bill of Materials (BoM):

Sr. No.	Component	Specification	Quantity	Unit Price (INR)	Total Price (INR)
1	Microcontroller Board	AT89C51 / 8051 Dev Board	1	250	250
2	Arduino Uno Board	ATmega328P	1	1000	1000
3	Servo Motor	SG90 / 180° Rotation	1	150	150
4	Crystal Oscillator	11.0592 MHz	1	15	15
5	Capacitors	33pF (for crystal oscillator)	2	2	4
6	Resistors	10kΩ Pull-up	2	2	4
7	Wires	Male-to-Female Jumpers	1 set	20	20
8	Breadboard	Medium size	1	80	80

Total Cost: ₹1,523

- **Circuit Image :**





- # Appendix (Assembly Code):

```
ORG 00H

MAIN:
   MOV TMOD, #01H       ; Set Timer 0 in Mode 1 (16-bit timer)

CHECK_BUTTONS:
   JNB P2.6, ANTCLK    ; If P2.6 (Clockwise button) is pressed (logic 0), go to
CLKWISE
   JNB P2.7, CLKWISE     ; If P2.7 (Anti-clockwise button) is pressed (logic 0),
go to ANTCLK

   LCALL zero_degrees   ; Default position (0°)
   SJMP CHECK_BUTTONS

CLKWISE:
   LCALL ninety_degrees         ; Rotate to +90°
   SJMP CHECK_BUTTONS

ANTCLK:
   LCALL negative_ninety_degrees  ; Rotate to -90°
   SJMP CHECK_BUTTONS

RET

; -90 = 1ms pulse
negative_ninety_degrees:
   MOV TH0, #0FCH      ; Load TH0 with FC66H
   MOV TL0, #66H
   SETB P2.0
   SETB TR0
WAIT1:
   JNB TF0, WAIT1
   CLR P2.0
   CLR TF0
   CLR TR0
   RET
```

```
; 0 = 1.5ms pulse
zero_degrees:
   MOV TH0, #0FAH      ; Load TH0 with F9A0H
   MOV TL0, #0A0H
   SETB P2.0
   SETB TR0
WAIT2:
   JNB TF0, WAIT2
   CLR P2.0
   CLR TF0
   CLR TR0
   RET

; +90 = 2ms pulse
ninety_degrees:
   MOV TH0, #0F7H      ; Load TH0 with F74DH
   MOV TL0, #4DH
   SETB P2.0
   SETB TR0
WAIT3:
   JNB TF0, WAIT3
   CLR P2.0
   CLR TF0
   CLR TR0
   RET

END
```

- **Appendix (Arduino C Code):**

```
#include <MsTimer2.h>
#define SERVO 2
#define PWM_RANGE_MIN 544 // 0 degree
#define PWM_RANGE_MAX 2400 // 180 degree
String strAngle;
boolean flag = false;
int dutyHIGH = 1500;

void setup(){
   pinMode(2, OUTPUT);
```

```
    MsTimer2::set(20, setServo); // 20ms Period Timer Interrupt
    MsTimer2::start();
    Serial.begin(115200);
    Serial.print("Input Angle 0 ~ 180 : ");
}

void loop(){
    if(flag == 1){
        // Run when serial data is received
        flag = 0;
        int temp = strAngle.toInt();
        strAngle = "";
        if((temp >= 0) && (temp <= 180)){
            dutyHIGH = map(temp, 0, 180, PWM_RANGE_MIN,
PWM_RANGE_MAX);
        }
        else {
            Serial.println("Please Input 0 ~ 180..!");
        }
        Serial.print("Input Angle 0 ~ 180 : ");
    }
}

void setServo(){
    digitalWrite(SERVO, HIGH);
    delayMicroseconds(dutyHIGH);
    digitalWrite(SERVO, LOW);
}

void serialEvent(){
    char myChar = (char)Serial.read();
    if((myChar >= '0') && (myChar <= '9')){
        strAngle += myChar;
    }
    else if(myChar == '\n'){
        flag = true;
    }
}
```