ACTUATORS, DRIVES, AND CONTROL COMPONENTS

LABOERATORY 3

Hannah Palanas Servande
College of Engineering
Bachelor of Science in Electronics Engineering
Samar State University
hannahservande01@gmail.com

***Abstract*—This experiment provides an in-depth exploration of the various types of actuators and control components commonly utilized in the field of robotics. It focuses on understanding their operational principles and how they interface with microcontrollers. Students will engage in hands-on activities that involve operating different actuators through Pulse Width Modulation (PWM) and other relevant control signals. Through a combination of simulation-based analysis and physical hardware testing, participants will assess and validate the performance, response characteristics, and overall behavior of the actuators. This comprehensive approach not only enhances theoretical knowledge but also strengthens practical skills in designing and implementing actuator control systems in robotic applications.**

***Index Terms*—Actuators, PWM, DC motor, Servo motor, Stepper motor, Robotics, Webots.**

I. Rationale

This experiment is designed to develop a comprehensive understanding of the different types of actuators and control components employed in robotic systems, with particular emphasis on their integration with microcontrollers. Participants will learn the principles governing actuator operation and will apply various control techniques, such as Pulse Width Modulation (PWM) and other signal protocols, to manipulate and regulate actuator behavior. Through guided exercises, students will gain practical experience in configuring microcontroller outputs to achieve precise and efficient actuator control, thereby deepening their knowledge of system dynamics and control strategies in robotics.

II. Objectives

- Interface and control a minimum of three different types of actuators—specifically a DC motor, a servo motor, and a stepper motor—using an Arduino microcontroller. Ensure accurate regulation of both speed and direction for each actuator, adhering to their respective operational requirements.
- Employ Pulse Width Modulation (PWM) techniques to modulate the speed of the DC motor. Systematically vary the duty cycle from 50 to 100 percent , carefully observing and recording the corresponding changes in motor speed to evaluate the effectiveness of PWM control.
- Simulate motor movements within the Webots robotic simulation environment, aiming for a minimum success rate of 95 Percent in achieving correct motor behavior. This includes accurate replication of intended direction, speed, and response characteristics, as compared to the expected physical performance.

III. Materials and Software

A. Materials

- Arduino Uno
- DC motors
- Servo motors
- Stepper motor
- L298N motor driver
- ULN2003 stepper motor driver
- Breadboard
- Jumper Wires

B. Software

- Arduino IDE
- Webots Simulation Environment

IV. Procedures

1) Set up Arduino to control DC motors, servo motors, and stepper motors.
2) Generate PWM signals to vary motor speed and direction.
3) Write the control code for motor management in Arduino IDE.
4) Simulate the actuators in Webots and check for correct performance.

V. Results

The stepper motor has been set to complete one full revolution approximately every 15 seconds, which results in a frequency of around 0.067 Hz. This configuration allows for precise and controlled rotation over a set period of time.

For the DC motor, three distinct speed modes have been defined, with PWM values of 55, 125, and 255 used to regulate its speed. In addition, the servo motor has been programmed to sweep continuously from 0 to 180 degrees with each loop iteration, providing a full range of back-and-forth motion.

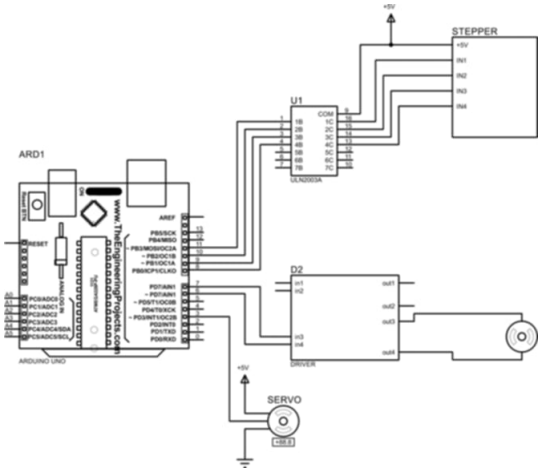

Fig. 1. Schematic.

VI. Discussion

The stepper motor exhibited precise, albeit relatively slow, rotational movement, completing a full 360-degree rotation in approximately 15 seconds. This performance corresponds to an operational frequency of approximately 0.067 Hz. The observed behavior was consistent with the expected output when driven by a ULN2003 motor driver at the selected stepping rate. Throughout the operation, the motor demonstrated stable and controlled motion, indicating reliable interfacing and proper configuration of the step sequence and timing parameters.

The DC motor displayed a clear and linear response to variations in the PWM duty cycle. By applying analogWrite values of 60, 130, and 255, distinct low, medium, and high-speed operating modes were observed, respectively. The results confirmed that as the PWM duty cycle increased, there was a proportional enhancement in both the torque output and rotational speed (measured in RPM). This behavior validated the effectiveness of PWM-based speed control for DC motors and highlighted the predictability of the motor's dynamic response under different input conditions.

Simultaneously, the servo motor reliably performed complete sweeps between 0° and 180° during each operational cycle. The servo consistently demonstrated accurate and repeatable position control, driven through standard PWM signal modulation techniques. This consistent performance underscored the servo motor's suitability for applications requiring precise angular positioning and rapid response to control inputs.

VII. Conclusion

This experiment effectively showcased the integration and control of a stepper motor, DC motor, and servo motor using PWM signals produced by an Arduino Uno. The DC motor displayed distinct speed modes that clearly corresponded to adjustments in the PWM duty cycle, demonstrating predictable and consistent behavior.

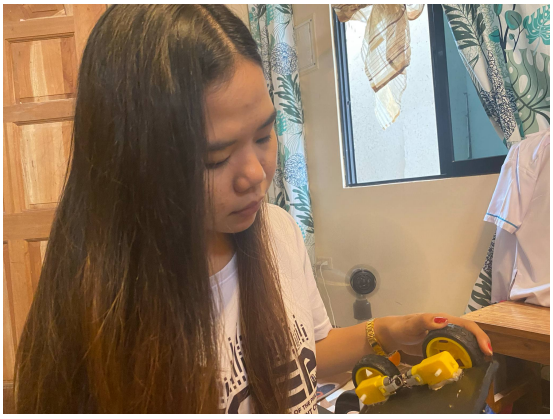

Fig. 2. DC motor testing.

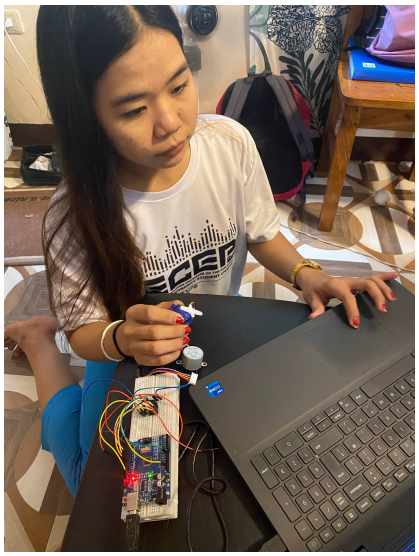

Fig. 3. Servo motor testing.

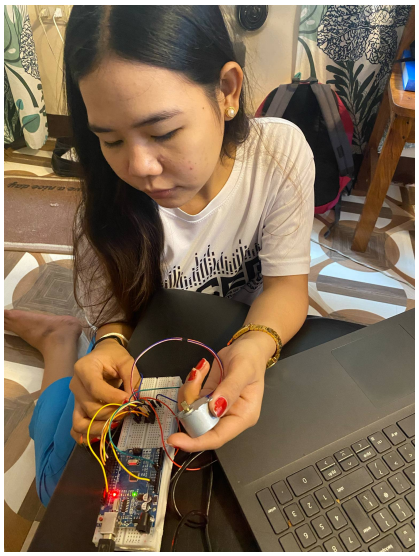

Fig. 4. Stepper motor testing.

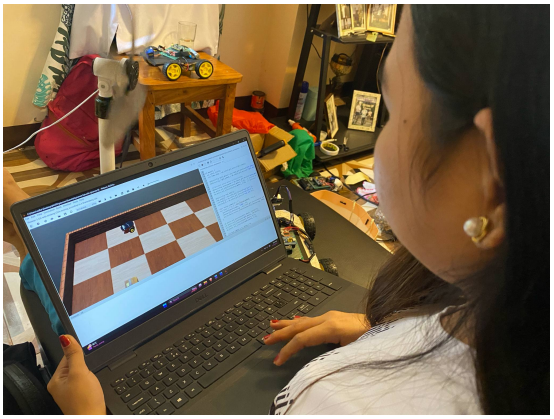

Fig. 5. Webots Simulation

The stepper motor operated with stable, low-frequency rotational movement, maintaining a steady pace as expected. Meanwhile, the servo motor consistently executed full-range sweeps from 0° to 180°, validating its ability to achieve accurate position control through PWM signal modulation.

REFERENCES

[1] , https://www.st.com/en/microcontrollers-microprocessors/stm32f103c6.html.

[2] Cyberbotics Ltd., "Webots User Guide", 2024.

[3] ULN2003 stepper motor driver, https://www.ti.com/product/ULN2003A.

APPENDIX

Actuators

```
/*
 Stepper Motor Control - one revolution

 This program drives a unipolar or bipolar stepper motor.
 The motor is attached to digital pins 8 - 11 of the Arduino.

 The motor should revolve one revolution in one direction, then
 one revolution in the other direction.


 Created 11 Mar. 2007
 Modified 30 Nov. 2009
 by Tom Igoe

 */

#include <Stepper.h>

const int stepsPerRevolution = 200;  // change this to fit the number of steps per revolution
// for your motor

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
  // set the speed at 60 rpm:
  myStepper.setSpeed(60);
  // initialize the serial port:
  Serial.begin(9600);
}

void loop() {
  // step one revolution  in one direction:
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);

  // step one revolution in the other direction:
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
/* Sweep
 by BARRAGAN <http://barraganstudio.com>
 This example code is in the public domain.

 modified 8 Nov 2013
 by Scott Fitzgerald
 https://www.arduino.cc/en/Tutorial/LibraryExamples/Sweep
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
}
barraganstudio.com
const int pin1 = 4;
```

```
const int pin2 = 5;


void setup(){
        Serial.begin(9600);
        pinMode(pin1, OUTPUT);
        digitalWrite(pin1, 0);


}

void loop(){

        int x = Serial.read();

        switch(x){
                case 1:
                        analogWrite(pin2, 255)
                            ;
                break;
                case 2:
                        analogWrite(pin2, 125)
                            ;
                break;
                case 3:
                        analogWrite(pin2, 50);
                break;
                default:
                break;
        }

}
```