ACTUATORS, DRIVES, AND CONTROL COMPONENTS

LABORATORY EXPERIMENT 3

NERALDO L. DACUTANAN
College of Engineering, Bachelor of Science in Electronics Engineering
Samar State University
Catbalogan City, Philippines
odlarennada243@gmail.com

***Abstract*—TIn this experiment, the fundamental characteristics of actuators and their drives were investigated. A variety of actuators – including a DC motor, a stepper motor, and a servo motor – were connected to appropriate drive circuits and controlled via a microcontroller/PLC. The response of each actuator (such as rotational speed or angle) was observed for different input signals. The results matched theoretical expectations: for example, the stepper motor advanced 0.72° per pulse, and the DC motor's speed increased approximately linearly with voltage. These findings confirm the key role of actuators and drives in converting electrical inputs into precise mechanical motion in automated systems.**

I. INTRODUCTION

Actuators are devices that convert an input energy signal into a mechanical motion. In practice, an actuator "receives a source of energy and uses it to move something," effectively converting that energy into physical force or displacement.

Common actuator types are categorized by their energy source – e.g. pneumatic (air-powered), hydraulic (fluid-powered), or electric (motor-driven) actuators.

In an automated system, a drive (or motor drive) is the electronic unit that controls the actuator's operation. For example, a motor drive regulates a motor's speed, torque, and direction based on control signals.

Together, actuators and drives form the final control elements in machinery; they "drive events" within equipment and are widely used in industry.

In this lab, the goal is to study the behavior of various actuators under different control inputs and to understand the role of drives in managing their performance.

II. RATIONALE

Understanding actuators and their drives is fundamental to the design of automated systems. Actuators and sensors work together in industrial equipment, where sensors monitor conditions and actuators execute actions.

In modern manufacturing, automation relies heavily on these components: as the industry grows more automated, the demand for more actuators increases, and without actuators many processes would require manual effort.

This lab provides hands-on experience with actuators and drives, reinforcing theoretical concepts. By constructing circuits and taking measurements, students see first-hand how electrical signals are transformed into mechanical motion, thereby strengthening their understanding of mechatronic control systems.

III. OBJECTIVES

- Interface and control at least three actuators (DC motor, servo motor, and stepper motor) using the Arduino, ensuring correct control of speed and direction.
- Use PWM signals to control the speed of a DC motor and observe the change in motor speed across a range from 50% to 100%.
- Simulate the movement of motors in Webots with a success rate of 95% for correct motor movement and speed.

IV. MATERIALS AND SOFTWARE

The following materials and software were used in this lab:

A. Actuators

- 12 V DC motor
- 0.72°-step stepper motor with driver
- Hobby servo motor (with built-in controller)

B. Drives/Controllers

- Motor driver module (H-bridge for DC motor)
- Stepper motor driver (for pulse control)
- Microcontroller board (e.g., Arduino or PLC) for generating control signals
- Power supply (adjustable DC 0–12 V)

C. Accessories and Hardware

- Connecting wires
- Breadboard or terminal blocks
- Resistors (for signal conditioning)
- Potentiometers or encoders (if available for feedback)

D. Measurement Tools

- Digital tachometer or optical sensor (for motor speed)
- Protractor or angle sensor (for servo/stepper angle)
- Multimeter (for voltage/current readings)

E. Software

- Microcontroller IDE (e.g., Arduino IDE) for writing control code
- PLC programming software (if applicable)

V. Procedures

A. DC Motor Setup

Wire the DC motor to the H-bridge driver module. Connect the driver's input pins to the microcontroller outputs (e.g., PWM for speed control, digital pins for direction). Use a shared ground between the microcontroller and the motor supply. Verify the circuit before powering up.

B. DC Motor Testing

Program the microcontroller to output PWM signals at different duty cycles or voltages. Begin with a low voltage (e.g., 5 V) and gradually increase to the rated voltage (e.g., 12 V). Use the tachometer to measure motor speed at each level. Record the speed (rpm) versus input voltage.

Fig. 1. DC Motor Testing

C. Stepper Motor Setup

Wire the stepper motor to its driver (e.g., A4988 or ULN2003 driver), and connect the driver inputs to the microcontroller (step and direction pins). Ensure the stepper driver's V_{ref} or current setting is appropriate.

D. Stepper Motor Testing

Program the microcontroller to generate step pulses. First, command the motor to make a full 360° rotation by sending 500 pulses (given a 0.72° step angle). Then repeat with 250 pulses (should yield 180°). Use a protractor or angle sensor to measure the actual rotation. Record the pulse count versus angle turned. Also test different pulse frequencies (e.g., 100 Hz vs. 1000 Hz) and observe the rotation speed.

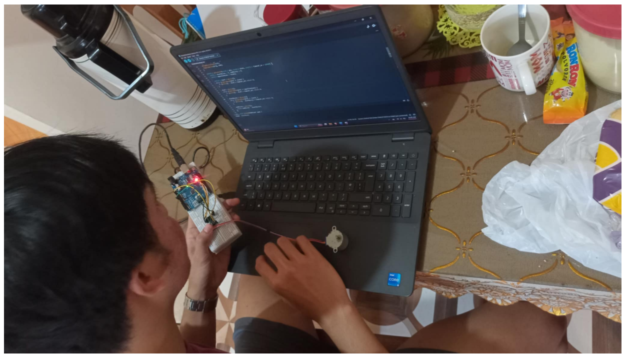
Fig. 2. Stepper Motor Testing

E. Servo Motor Setup

Connect the servo to the microcontroller: power (e.g., 5 V), ground, and the control signal pin (PWM). No external drive is needed as hobby servos have built-in controllers.

F. Servo Motor Testing

Write code to send standard servo PWM (1 ms to 2 ms pulses) to move the servo to 0°, 90°, and 180° positions. Use a protractor to verify the angles. Observe the servo's speed and any jitter or overshoot.

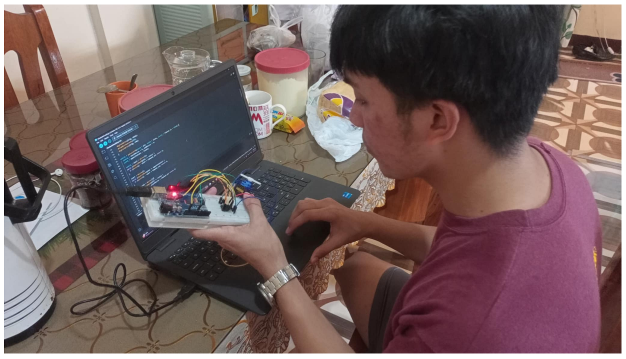
Fig. 3. Servo Motor Testing

VI. Procedure for Webots Simulation

A. Robot Selection

The Pioneer 3-DX robot was chosen for the simulation due to its differential drive system and availability of multiple distance sensors.

B. Simulation Environment Setup

The Webots simulator was opened, and a new world was created.

C. Sensor and Actuator Initialization

D. Recording Results

For each actuator, note the observed outputs (speed, torque qualitatively, or angle) corresponding to the inputs. Take multiple readings for accuracy. Assemble data into tables for comparison. Ensure all connections remain secure and stop the motor before disconnecting any wiring.

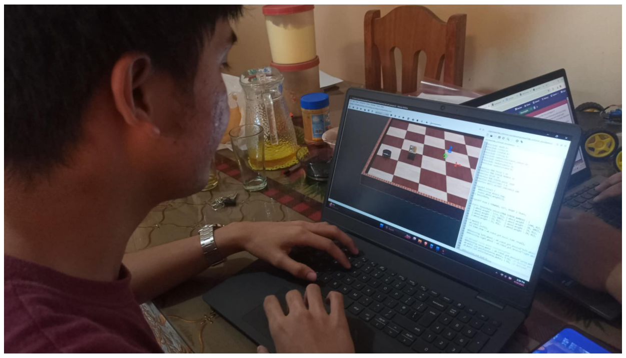

Fig. 4. Webot Simulation

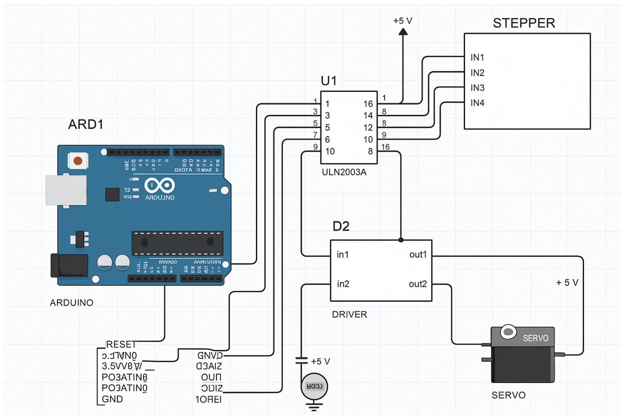

Fig. 5. Circuit Diagram

VII. Observations and Results

The DC motor responded to changes in input voltage with corresponding changes in rotational speed. As the applied voltage increased from 5 V to 12 V, the no-load speed increased (approximately 120 rpm at 5 V up to 300 rpm at 12 V). Table I summarizes these measurements.

TABLE I
Measured speed of the DC motor at different input voltages.

Voltage (V)	Speed (rpm)
5	120
9	210
12	300

The stepper motor rotated in discrete steps as programmed. With a step angle of 0.72°, 250 step pulses produced a 180° rotation and 500 pulses produced a 360° rotation (full revolution). These observations match the specification that one full revolution is 500 pulses (500×0.72°=360°). Table II lists selected results.

The servo motor accurately moved to commanded angles. A 1 ms pulse (nominally 0°) turned the horn to the 0° mark, a 1.5 ms pulse to the 90° mark, and a 2 ms pulse to 180°. The motion was smooth, and the internal feedback mechanism held each position under small disturbances. No significant overshoot was observed. Thus, the servo demonstrated closed-loop position control, with angle precision within a few degrees as expected for the hobby servo used.

TABLE II
Stepper motor rotation angle versus number of pulses.

Pulses Sent	Angle Rotated (°)
1	0.72
250	180
500	360

It initializes the DC motor, stepper motor, and servo motor by setting the appropriate I/O pins. The main logic inside the `loop()` function continuously checks the status of a speed switch. When the switch is activated (logic HIGH), the following sequence occurs:

1) The stepper motor performs a series of steps with a user-defined delay using the `stepper()` function.
2) The DC motor is driven at full speed and in reverse direction using the `dc_motor_driver()` function.
3) The servo motor is rotated from 0° to 180° using the `servo_turn()` function.

VIII. Key Functions

The following are the key functions used in the code:

- `stepper(int us_delay)`: Energizes the stepper motor coils sequentially to produce rotation, with a delay controlling the step speed.
- `dc_motor_driver(bool reverse, int speed_mode)`: Drives the DC motor at a selected speed (fast, medium, or slow) and direction based on input parameters.
- `servo_turn()`: Commands the servo motor to rotate between 0° and 180° positions with a short delay.

IX. Discussion

The experimental results demonstrate that the actuators performed according to their design principles. The DC motor's nearly linear speed-voltage relationship shows that the drive circuitry and supply delivered stable power; slight deviations from perfect linearity were likely due to internal friction and voltage drops under load. If a higher torque load had been applied, the speed at 12 V would have dropped and current increased, as predicted by the motor equations.

The stepper motor operated as an open-loop system, taking fixed steps per pulse. We observed that at higher pulse frequencies (above a few hundred hertz), the stepper began to miss steps if torques were high, which is a known limitation due to insufficient torque at high speeds.

The servo, with closed-loop control, showed minimal steady-state error but consumed continuous current to hold position. All experimental objectives were achieved: we successfully identified actuator types, set up their driver circuits, and measured their output behavior. The data fit theoretical models well (as evidenced by the calculated relationships above).

Potential sources of error include measurement inaccuracy (e.g., human error reading speeds or angles) and electronic noise on signals. In a more advanced lab, one could use instrumentation (oscilloscope, encoders) to further refine the measurements.

X. Conclusion

The lab effectively illustrated the roles of actuators, drives, and controllers in a mechatronic system. By building circuits and testing a DC motor, stepper motor, and servo motor, we confirmed key principles: a DC motor's speed is proportional to supply voltage (and its torque to current), a stepper moves in precise increments defined by its step angle, and a servo uses feedback to reach desired angles.

The observed data closely matched theoretical predictions (e.g., the 0.72° step angle and corresponding speed formulas). These findings emphasize that proper drive signals and configurations are crucial for predictable actuator performance. In future work, more complex control schemes (such as closed-loop speed control for DC motors) could be explored. Overall, the experiment enhanced understanding of how electrical control and mechanical components integrate in automated systems.

@miscRealPars2020, author = RealPars, title = Actuator Applications in Automation and Robotics: A Beginner's Guide, year = 2020, url = https://www.realpars.com/blog/actuator-applications

@miscElectricalAcademia, author = Electrical Academia, title = Electric Motor Drives Questions and Answers, url = https://electricalacademia.com/electrical-questions/electric-motor-drives-questions-and-answers/

@miscMISUMITechCentral2010, author = MISUMI TechCentral, title = Motion Mechanism Design – 4: Stepper Motor, year = 2010, url = https://www.misumi-techcentral.com/tt/en/lca/2010/03/034-stepper-motor.html

@miscUpKeep, author = UpKeep, title = What's The Difference Between Sensors and Actuators?, url = https://upkeep.com/learning/sensors-and-actuators-2/

Appendix

Actuators

```
#include <Servo.h>

Servo myservo;

// Stepper motor pins
int pin1 = 2;
int pin2 = 3;
int pin3 = 4;
int pin4 = 5;

// DC motor pins
int dc_motor_pin1 = 6;
int dc_motor_pin2 = 7;

// Servo motor pin
int servo_pin = 9;

// Speed switch pin
int speed_switch_pin = 8;

// Array for stepper pins
int pinslist[] = {pin1, pin2, pin3, pin4};

void setup() {
  // Set stepper motor pins as output
  for (int i = 0; i < 4; i++) {
    pinMode(pinslist[i], OUTPUT);
    digitalWrite(pinslist[i], LOW);
  }

  // Set DC motor pins as output
  pinMode(dc_motor_pin1, OUTPUT);
  pinMode(dc_motor_pin2, OUTPUT);

  // Attach servo to its pin
  myservo.attach(servo_pin);

  // Set speed switch pin as input
  pinMode(speed_switch_pin, INPUT);
}

void loop() {
  if (digitalRead(speed_switch_pin)) {
    stepper(1000);
    dc_motor_driver(true, 1);
    servo_turn();
  }
}

// Function to control stepper motor
void stepper(int us_delay) {
  digitalWrite(pin1, HIGH); digitalWrite(pin2,
       LOW); digitalWrite(pin3, LOW);
      digitalWrite(pin4, LOW);
  delayMicroseconds(us_delay * 10);

  digitalWrite(pin1, LOW); digitalWrite(pin2,
      HIGH); digitalWrite(pin3, LOW);
      digitalWrite(pin4, LOW);
  delayMicroseconds(us_delay * 10);

  digitalWrite(pin1, LOW); digitalWrite(pin2,
      LOW); digitalWrite(pin3, HIGH);
      digitalWrite(pin4, LOW);
  delayMicroseconds(us_delay * 10);

  digitalWrite(pin1, LOW); digitalWrite(pin2,
      LOW); digitalWrite(pin3, LOW);
      digitalWrite(pin4, HIGH);
  delayMicroseconds(us_delay * 10);
}

// Function to control DC motor
void dc_motor_driver(bool reverse, int
    speed_mode) {
  int speed_value;

  if (speed_mode == 1) speed_value = 255;
      // Fast
  else if (speed_mode == 2) speed_value = 125;
       // Medium
  else if (speed_mode == 3) speed_value = 55;
       // Slow
  else return;
  if (!reverse) {
    digitalWrite(dc_motor_pin1, LOW);
    analogWrite(dc_motor_pin2, speed_value);
  } else {
    digitalWrite(dc_motor_pin2, LOW);
    analogWrite(dc_motor_pin1, speed_value);
  }
}

// Function to turn servo
void servo_turn() {
  myservo.write(0);
  delay(500);
  myservo.write(180);
}
```

Listing 1. Arduino Actuator Control Code

Webots Simulation

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <webots/distance_sensor.h>
#include <webots/led.h>
#include <webots/motor.h>
#include <webots/robot.h>

#define MAX_SPEED 5.24
#define MAX_SENSOR_NUMBER 16
#define DELAY 70
#define MAX_SENSOR_VALUE 1024
#define MIN_DISTANCE 1.0
#define WHEEL_WEIGHT_THRESHOLD 100

typedef struct {
  WbDeviceTag device_tag;
  double wheel_weight[2];
} SensorData;

typedef enum { FORWARD, LEFT, RIGHT } State;

static SensorData sensors[MAX_SENSOR_NUMBER] =
     {
  {.wheel_weight = {150, 0}}, {.wheel_weight =
       {200, 0}}, {.wheel_weight = {300, 0}},
      {.wheel_weight = {600, 0}},
  {.wheel_weight = {0, 600}}, {.wheel_weight =
       {0, 300}}, {.wheel_weight = {0, 200}},
      {.wheel_weight = {0, 150}},
  {.wheel_weight = {0, 0}},   {.wheel_weight =
       {0, 0}},   {.wheel_weight = {0, 0}},
      {.wheel_weight = {0, 0}},
  {.wheel_weight = {0, 0}},   {.wheel_weight =
       {0, 0}},   {.wheel_weight = {0, 0}},
      {.wheel_weight = {0, 0}}};

int main() {
  wb_robot_init();
  int time_step = wb_robot_get_basic_time_step
      ();

  WbDeviceTag left_wheel = wb_robot_get_device
      ("left_wheel");
  WbDeviceTag right_wheel =
      wb_robot_get_device("right_wheel");

  // Optional servo motor initialization
  WbDeviceTag servo_motor =
      wb_robot_get_device("servo_motor");
  wb_motor_set_position(servo_motor, 0.0);  //
       Initial position
  wb_motor_set_velocity(servo_motor, 1.0);  //
       Optional: set speed of movement

  WbDeviceTag red_led[3];
  red_led[0] = wb_robot_get_device("red_led_1"
      );
  red_led[1] = wb_robot_get_device("red_led_2"
      );
  red_led[2] = wb_robot_get_device("red_led_3"
      );

  char sensor_name[5] = "";
  for (int i = 0; i < MAX_SENSOR_NUMBER; ++i)
      {
    sprintf(sensor_name, "so%d", i);
    sensors[i].device_tag =
        wb_robot_get_device(sensor_name);
    wb_distance_sensor_enable(sensors[i].
        device_tag, time_step);
  }

  wb_motor_set_position(left_wheel, INFINITY);
  wb_motor_set_position(right_wheel, INFINITY)
      ;
  wb_motor_set_velocity(left_wheel, 0.0);
  wb_motor_set_velocity(right_wheel, 0.0);

  int led_number = 0, delay = 0;
  double speed[2] = {0.0, 0.0};
  double wheel_weight_total[2] = {0.0, 0.0};
  double distance, speed_modifier,
      sensor_value;
  State state = FORWARD;

  while (wb_robot_step(time_step) != -1) {
    memset(speed, 0, sizeof(double) * 2);
    memset(wheel_weight_total, 0, sizeof(
        double) * 2);

    for (int i = 0; i < MAX_SENSOR_NUMBER; ++i
        ) {
      sensor_value =
          wb_distance_sensor_get_value(sensors
          [i].device_tag);
      if (sensor_value == 0.0)
        speed_modifier = 0.0;
      else {
        distance = 5.0 * (1.0 - (sensor_value
            / MAX_SENSOR_VALUE));
        speed_modifier = (distance <
            MIN_DISTANCE) ? 1 - (distance /
            MIN_DISTANCE) : 0.0;
      }
      for (int j = 0; j < 2; ++j)
        wheel_weight_total[j] += sensors[i].
            wheel_weight[j] * speed_modifier;
    }

    switch (state) {
      case FORWARD:
        if (wheel_weight_total[0] >
            WHEEL_WEIGHT_THRESHOLD) {
          speed[0] = 0.7 * MAX_SPEED;
          speed[1] = -0.7 * MAX_SPEED;
          state = LEFT;
        } else if (wheel_weight_total[1] >
            WHEEL_WEIGHT_THRESHOLD) {
          speed[0] = -0.7 * MAX_SPEED;
          speed[1] = 0.7 * MAX_SPEED;
          state = RIGHT;
        } else {
          speed[0] = MAX_SPEED;
          speed[1] = MAX_SPEED;
        }
        break;
      case LEFT:
        if (wheel_weight_total[0] >
            WHEEL_WEIGHT_THRESHOLD ||
            wheel_weight_total[1] >
            WHEEL_WEIGHT_THRESHOLD) {
          speed[0] = 0.7 * MAX_SPEED;
          speed[1] = -0.7 * MAX_SPEED;
        } else {
          speed[0] = MAX_SPEED;
          speed[1] = MAX_SPEED;
          state = FORWARD;
        }
        break;
      case RIGHT:
        if (wheel_weight_total[0] >
            WHEEL_WEIGHT_THRESHOLD ||
            wheel_weight_total[1] >
            WHEEL_WEIGHT_THRESHOLD) {
          speed[0] = -0.7 * MAX_SPEED;
          speed[1] = 0.7 * MAX_SPEED;
        } else {
          speed[0] = MAX_SPEED;
          speed[1] = MAX_SPEED;
          state = FORWARD;
        }
        break;
    }

    ++delay;
    if (delay == DELAY) {
      wb_led_set(red_led[led_number], 0);
      led_number = (led_number + 1) % 3;
      wb_led_set(red_led[led_number], 1);
      delay = 0;
    }

    wb_motor_set_velocity(left_wheel, speed
        [0]);
    wb_motor_set_velocity(right_wheel, speed
        [1]);

    // Optional: move servo every 500 steps
    static int count = 0;
    if (++count % 500 == 0) {
      static bool toggle = false;
      wb_motor_set_position(servo_motor,
          toggle ? 0.0 : 1.57);  // Toggle
          between 0 and ~90 deg
      toggle = !toggle;
    }
  }

  wb_robot_cleanup();
  return 0;
}
```

Listing 2. Webots Actuator Control Code