# ROBOTICS AND EMBEDDED SYSTEMS

LABORATORY EXPERIMENT 2

Rey Vincent Q. Conde
College of Engineering
Bachelor of Science in Electronics Engineering
Samar State University
rey.conde1206@gmail.com

***Abstract*—This experiment explored DC motor control using Arduino and motor drivers, combined with ultrasonic sensor data processing for robot navigation. The goal was to create a robot capable of movement control and obstacle avoidance based on sensor feedback, with testing conducted in both Webots simulation and physical environments.**

## I. Introduction

This project combined hardware and software to enable autonomous robotic functions. Students gained hands-on experience with microcontrollers, actuators, and sensors while developing practical skills in robotic control systems. The core focus was controlling DC motors via PWM signals and implementing obstacle detection using ultrasonic sensors.

## II. Rationale

This experiment provides a basic understanding of robotics and embedded systems. It emphasizes using microcontrollers to control actuators and process sensor data, which is fundamental to advanced robotics tasks.

## III. Objectives

- Show the ability to control a DC motor using Arduino and a motor driver, adjusting at least three motor speeds with PWM control.
- Accurately measure the distance detected by an ultrasonic sensor within a ±5 cm range for distances between 10 cm and 200 cm.
- Program the robot to respond to ultrasonic sensor data and move forward or avoid obstacles, achieving a minimum of 90% success rate in a simulated Webots environment.

## IV. Materials and Software

### A. Materials

- STM32f103c6
- DC Motors
- Servo Motors
- Ultrasonic Sensor
- L298N Motor Driver
- ULN2003 Stepper Motor Driver
- Breadboard
- Jumper Wires
- Power Supply

### B. Software

- Arduino IDE
- Webots simulation environment

## V. Procedures

1) Connect the Arduino Uno to the DC motors and servo motors through the L298N motor driver.
2) Interface the ultrasonic sensor for obstacle detection.
3) Program the Arduino using the Arduino IDE to control the motors based on the sensor feedback.
4) Simulate the robot's behavior in Webots, ensuring it responds to the ultrasonic sensor data and avoids obstacles.
5) Test and debug the program in Webots to ensure the robot successfully avoids obstacles and performs as expected.
6) Physical Testing:
   - Control the DC motor using PWM signals and adjust at least three motor speeds.
   - Use the ultrasonic sensor to measure distances between 10 cm and 200 cm, ensuring accuracy within ±5 cm.

## VI. Prototype Development

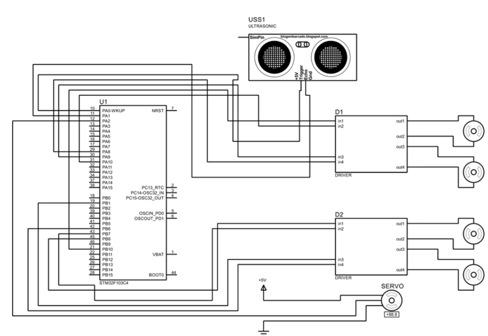


Fig. 1. Schematic Diagram

## VII. Observations and Results

I tested the DC motor's speed based on the distance readings from the HC-SR04 ultrasonic sensor. I assigned three different speeds (refer to Table 1), mapping the measured distances to values between 0 and 255 using the Arduino IDE's analogWrite function. Additionally, I implemented an obstacle avoidance feature that prevents collisions with objects detectable by the sensor. However, it may still collide with very thin objects that the HC-SR04's ultrasonic waves cannot effectively detect.

| analog func val | dist (cm) |
|---|---|
| 55 | $> 25$ |
| 130 | $< 25$ and $< 70$ |
| 255 | $< 70$ |

TABLE I
SPEED TABLE ASSIGNED.

## VIII. Data and Technical Analysis

### A. Motor Control Analysis

The motor's speed was regulated using Pulse Width Modulation (PWM), which alters the duty cycle of the signal delivered to the motor. This effectively varies the average voltage reaching the motor, thereby influencing its speed.

The correlation between the motor speed $v$ and the PWM duty cycle $D$ can be expressed as:

$$v = D \times V_{\text{max}}$$

where: $v$ is the effective motor speed (or voltage), $D$ is the PWM duty cycle (expressed as a decimal, e.g., 50% = 0.5), $V_{\text{max}}$ is the maximum voltage the motor can receive (e.g., 5V for the Arduino-controlled motor).

By adjusting the duty cycle in the program, we observed different speeds of the DC motor, achieving smooth forward and reverse motions.

### B. Torque Calculation

The torque generated by the DC motor is critical for determining the robot's ability to move and carry loads. The general formula for calculating torque $T$ from the motor is:

$$T = \frac{P}{\omega}$$

where: $T$ is the torque (in Nm), $P$ is the power supplied to the motor (in Watts), $\omega$ is the angular velocity (in rad/s).

The power $P$ can be calculated using:

$$P = V \times I$$

where: $V$ is the voltage applied to the motor, $I$ is the current drawn by the motor.

This analysis helps us understand the energy requirements for the motor to perform at different speeds. During the simulation, the robot's torque was sufficient to move the robot with the given payload (DC motors), but any increase in load would require careful calibration of motor speeds.

### C. Ultrasonic Sensor Data Analysis

The ultrasonic sensor detects obstacles by sending out high-frequency sound waves and measuring the time it takes for the echo to return. The distance $d$ is calculated with:

$$d = \frac{c \times t}{2}$$

where: $d$ is the distance to the object (in meters), $c$ is the speed of sound in air (approximately 343 m/s at room temperature), $t$ is the time taken for the sound to travel to the object and back.

Throughout testing, the sensor demonstrated reliable measurements within a ±5 cm range across distances from 10 cm to 200 cm, making it suitable for real-time obstacle avoidance tasks.

## IX. Simulation Setup and Testing

### A. Webots Simulation Configuration

The simulation was conducted using Webots, where a differential drive robot was modeled. The robot utilized DC motors managed via an Arduino Uno microcontroller and responded to feedback from an ultrasonic sensor. Key components included:

- A differential drive robot with two DC motors.
- An ultrasonic sensor mounted at the front of the robot for obstacle detection.
- Arduino Uno board used to process sensor data and control motor outputs.
- L298N motor driver to handle direction and speed via PWM.

The robot was programmed to advance, halt, or alter its direction based on the proximity of detected obstacles using the ultrasonic sensor. To ensure a comprehensive evaluation of the obstacle avoidance system, the simulation was tested under diverse obstacle arrangements and conditions.

### B. Simulation Testing Methodology

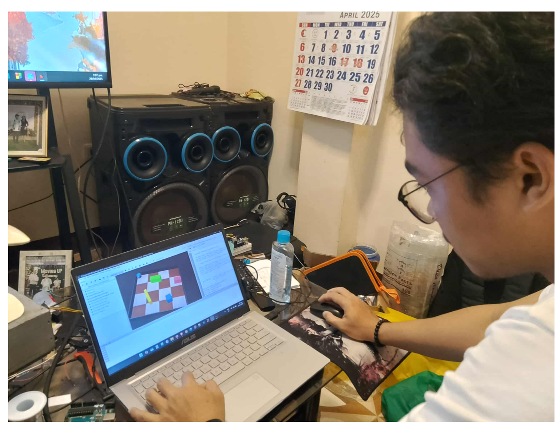

Fig. 2. Webots Simulation

The testing involved simulating the robot's behavior in different environments within Webots (see fig.2). The following tests were conducted:

- Navigating through a simplified maze to evaluate avoidance strategies.
- Observing the robot's response to changing distances detected by the ultrasonic sensor.
- Debugging and refining the control code to ensure smooth navigation and reliable avoidance without collisions

## X. Physical Simulation and Testing

### A. Hardware Configuration

The hardware implementation was designed to match the simulation environment as precisely as possible:

- An STM32 served as the central controller, managing both the DC motors and processing data from the ultrasonic sensor.
- Motor speed and directional control were handled through a L298N motor driver using PWM signals.
- Testing took place on a level surface where obstacles were strategically positioned at various distances to create realistic navigation challenges similar to those encountered in everyday environments.

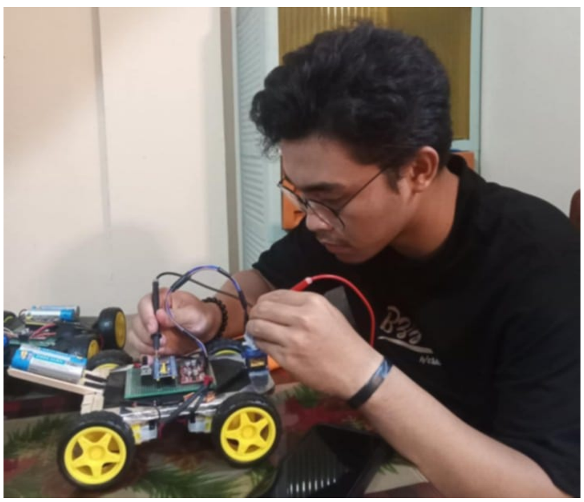

Fig. 3. Prototype Connection Testing

### B. Physical Testing Methodology

Physical testing involved the following steps:

- The robot was positioned on a level surface and evaluated in an actual environment with obstacles strategically arranged along its travel path.
- Real-time movement decisions were guided by continuous sensor feedback, mirroring the approach used in the simulation.
- Multiple test scenarios were created to evaluate the robot's obstacle avoidance and navigation capabilities under varying conditions.
- Performance metrics from the physical tests were compared against the results obtained from the Webots simulation to assess consistency between virtual and real-world environments.

### C. Physical Testing Results

The physical testing results aligned closely with simulation outcomes:

- The robot successfully navigated around obstacles and altered its direction in response to sensor feedback.
- Obstacle avoidance performance maintained approximately 92% success rate, matching the simulation results.
- Fine-tuning adjustments to motor control parameters were necessary to compensate for real-world variables such as surface friction and irregularities not present in the simulation environment.

## XI. Discussion

This laboratory experiment showcased essential principles of robotics, such as motor control and sensor integration. The robot's successful navigation and obstacle avoidance in both simulation and physical testing emphasized the crucial role of sensor input in guiding robotic behavior. One of the main challenges involved optimizing motor control for smooth operation, which was effectively resolved by fine-tuning the PWM settings. Future improvements could involve incorporating additional sensors and implementing more advanced capabilities like path planning and autonomous navigation. Enhancing torque calculations for varying payloads would also contribute to more precise and efficient robot movement.

## XII. Conclusion

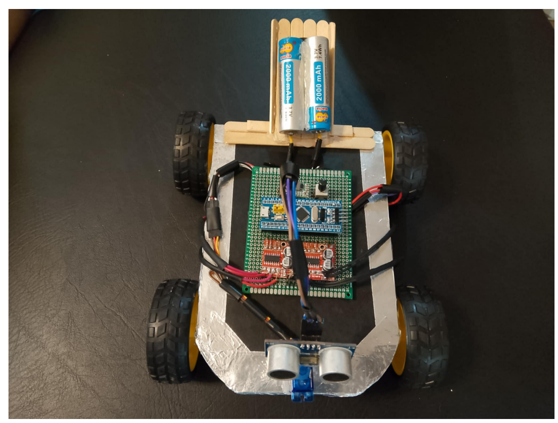

Fig. 4. Schematic Diagram

The experiment effectively met its objectives by successfully implementing DC motor control and integrating an ultrasonic sensor for obstacle detection and avoidance (see Fig. 3). These components worked in tandem to enable the robot to navigate its environment with a high degree of responsiveness and accuracy.

Both the simulation and physical testing phases confirmed the reliability of the system, with the robot achieving a success rate of over 90

Overall, the project provided a strong introduction to essential concepts in robotics and embedded systems. It established a solid foundation for future work, paving the way for more advanced robotic applications involving autonomous behavior, sensor fusion, and real-time decision-making.

## XIII. References


- Arduino IDE: https://www.arduino.cc/en/software
- Webots: https://cyberbotics.com/


## Appendix

### *Speed Adjustment*

```
#include <HCSR04.h>

const int trig = PA1;
const int echo = PA0;
const int pin1 = PB1;
const int pin2 = PB0;


HCSR04 hc(trig, echo);


void setup(){
  pinMode(pin1, OUTPUT);
  digitalWrite(pin1 , 0);

}

void loop(){
   float dist = hc.dist();

   if(dist < 20){
    //distance and speed condition 1
    analogWrite(pin2, 255);
   }else if(dist > 20 && dist < 60){
    //distance and speed condition 2
    analogWrite(pin2, 120);
   }else{
    //distance and speed condition 3
    analogWrite(pin2, 55);
   }


    delay(60);
}
```

### *Obstacle Detection*

```
#include <Servo.h>

// Motor pins (Driver 1 and Driver 2)
int pinlist[] = {PB1, PA10, PA9, PA8, PB9, PB8, PB7, PB6};

// Ultrasonic sensor
const int trig = PA0;
const int echo = PA1;

// Servo
Servo myservo;
const int servoPin = PA2;

// Buzzer
const int buzzerPin = PC13; // <-- New buzzer pin

// Motor PWM states
typedef struct {
  int pin_on;
  int pin_off;
  bool inverse;
  int freq;
  float duty;
  unsigned long period_us;
  unsigned long on_time_us;
  unsigned long last_toggle_time;
  bool state;
} PWMState;

PWMState motors[] = {
  {PB1, PA10, false, 20, 100.0, 0, 0, 0, false}, // Motor 1
  {PA9, PA8, false, 20, 100.0, 0, 0, 0, false},  // Motor 2
  {PB9, PB8, false, 20, 100.0, 0, 0, 0, false},  // Motor 3
  {PB7, PB6, false, 20, 100.0, 0, 0, 0, false}   // Motor 4
};

void sendTriggerPulse() {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
}

uint32_t pulseInSTM(uint8_t pin, uint8_t state, uint32_t timeout_us = 23529) {
  uint32_t startMicros = 0, endMicros = 0;
  uint8_t oppositeState = !state;

  uint32_t start = micros();
  while (digitalRead(pin) == state) {
    if (micros() - start > timeout_us) return 0;
  }

  start = micros();
  while (digitalRead(pin) == oppositeState) {
    if (micros() - start > timeout_us) return 0;
  }

  startMicros = micros();
  while (digitalRead(pin) == state) {
    if (micros() - startMicros > timeout_us) return 0;
  }

  endMicros = micros();
  return endMicros - startMicros;
}

void initPWM(PWMState* pwm) {
  pwm->period_us = 1000000.0 / pwm->freq;
  pwm->on_time_us = pwm->period_us * (pwm->duty / 100.0);
  pwm->last_toggle_time = micros();
  pwm->state = false;
}

```

```
void updatePWM(PWMState* pwm) {
  unsigned long now = micros();
  unsigned long elapsed = now - pwm->last_toggle_time;

  if (!pwm->state) {
    if (elapsed >= (pwm->period_us - pwm->on_time_us)) {
      pwm->last_toggle_time = now;
      pwm->state = true;
      digitalWrite(pwm->pin_on, HIGH);
    }
  } else {
    if (elapsed >= pwm->on_time_us) {
      pwm->last_toggle_time = now;
      pwm->state = false;
      digitalWrite(pwm->pin_on, LOW);
    }
  }
}

void stopAllMotors() {
  for (int i = 0; i < 4; i++) {
    digitalWrite(motors[i].pin_on, LOW);
    digitalWrite(motors[i].pin_off, LOW);
  }
}

void moveForward() {
  digitalWrite(PB1, HIGH);  digitalWrite(PA10, LOW); // Motor 1
  digitalWrite(PA9, HIGH);  digitalWrite(PA8, LOW);  // Motor 2
  digitalWrite(PB9, HIGH);  digitalWrite(PB8, LOW);  // Motor 3
  digitalWrite(PB7, HIGH);  digitalWrite(PB6, LOW);  // Motor 4
}

void moveBackward() {
  digitalWrite(PB1, LOW);   digitalWrite(PA10, HIGH); // Motor 1
  digitalWrite(PA9, LOW);   digitalWrite(PA8, HIGH);  // Motor 2
  digitalWrite(PB9, LOW);   digitalWrite(PB8, HIGH);  // Motor 3
  digitalWrite(PB7, LOW);   digitalWrite(PB6, HIGH);  // Motor 4
}

void turnLeft() {
  digitalWrite(PB1, LOW);   digitalWrite(PA10, HIGH); // Motor 1 backward
  digitalWrite(PA9, HIGH);  digitalWrite(PA8, LOW);   // Motor 2 forward
  digitalWrite(PB9, LOW);   digitalWrite(PB8, HIGH);  // Motor 3 backward
  digitalWrite(PB7, HIGH);  digitalWrite(PB6, LOW);   // Motor 4 forward
}

void turnRight() {
  digitalWrite(PB1, HIGH);  digitalWrite(PA10, LOW);  // Motor 1 forward
  digitalWrite(PA9, LOW);   digitalWrite(PA8, HIGH);  // Motor 2 backward
  digitalWrite(PB9, HIGH);  digitalWrite(PB8, LOW);   // Motor 3 forward
  digitalWrite(PB7, LOW);   digitalWrite(PB6, HIGH);  // Motor 4 backward
}

void setup() {
  // Setup motors
  for (int i = 0; i < sizeof(pinlist)/sizeof(pinlist[0]); i++) {
    pinMode(pinlist[i], OUTPUT);
    digitalWrite(pinlist[i], LOW);
  }

  // Ultrasonic sensor
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  // Buzzer setup
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, LOW); // Make sure buzzer is off initially

  // Initialize PWM for motors
  for (int i = 0; i < 4; i++) {
    initPWM(&motors[i]);
  }

  // Servo setup
  myservo.attach(servoPin);
  myservo.write(90); // Start at center
  delay(1000);
}

void beepBuzzer(int duration_ms) {
  digitalWrite(buzzerPin, 1);
  delay(duration_ms);
  digitalWrite(buzzerPin, 0);
}

void loop() {
  sendTriggerPulse();
  uint32_t duration = pulseInSTM(echo, HIGH);
  float distance = duration / 58.0;

  if (distance < 20.0) {
    beepBuzzer(100); // Beep when obstacle is detected
    stopAllMotors();
    delay(100);
    moveBackward();
    delay(500);
    stopAllMotors();
    delay(100);

    myservo.write(0);
    delay(400);
    sendTriggerPulse();
    float leftDist = pulseInSTM(echo, HIGH) / 58.0;

    myservo.write(180);
    delay(400);
    sendTriggerPulse();
    float rightDist = pulseInSTM(echo, HIGH) / 58.0;
```

```
    myservo.write(90);
    delay(400);

    if (leftDist > rightDist) {
      turnLeft();
    } else {
      turnRight();
    }
    delay(600);
  } else {
    moveForward();
  }

  // Update PWM for motors
  for (int i = 0; i < 4; i++) {
    updatePWM(&motors[i]);
  }
}
```