# LAB EXPERIMENT 1

ROBOTICS AND EMBEDDED SYSTEMS

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

***Abstract*—This experiment focuses on simulating the movement of a robot using Webots. The primary goal is to modify the robot's behavior through changes in its programming, such as adjusting motor velocities and incorporating new movement cycles. Through this simulation, students learn to control a robot's motion by modifying control code, providing practical insights into robotics and embedded systems.**

## I. Introduction

Robotics is integral to modern technology, and learning how to control robots through programming is essential for engineers. This experiment introduces students to basic robotic control by simulating movement in Webots, a powerful simulation environment. The aim is to help students understand the relationship between code changes and robot behavior. This activity lays the foundation for more advanced robotics tasks such as autonomous navigation and sensor-based control.

## II. Rationale

Simulation allows students to safely and efficiently experiment with robotic control systems in a virtual environment. By adjusting the robot's behavior through programming in Webots, students can observe the effects of their code changes in real time. This hands-on experience teaches the fundamental concepts of robotics and embedded systems, which are crucial for developing real-world robotic applications.

## Objectives

The main objective of this experiment is:

- To simulate robot movement based on the program.

## Materials and Software

**Materials:**

- Computer

**Software:**

- Webots Robotics Simulation Software

## III. Procedures

1) Download and install the Webots simulation software from the Cyberbotics website.
2) Open Webots, then go to the **File** menu and select the project file `spot.wbt` located in the `boston_dynamics` directory.
3) On the right panel of the Webots window, locate the C program that controls the robot's behavior.
4) Edit the program to change how the robot moves.
5) Click the **Play** button at the top of the screen to run the simulation.
6) Observe how the robot moves based on the updated code.
7) Record the robot's movement using screen recording for documentation.

## IV. Observations and Results

During the simulation, several key changes were made to the robot's movement code. Initially, the robot followed a basic walking cycle. After modifying the program, including adjusting the motor speeds and adding a crouch-walk cycle, the robot exhibited different movement patterns.

The following key observations were made:

- The robot successfully completed the new crouch-walk cycle after the code was edited.
- Adjustments to the motor velocity allowed for smoother and more controlled movements.
- The robot's movements were more fluid and responsive to the program's instructions.

## V. Data Analysis

Although the experiment did not require complex data analysis, the modification of the motor velocity and walk cycle directly impacted the robot's movement. By observing the changes in speed and behavior during the simulation, it was clear that the adjustments to motor control were successful in achieving the intended outcomes. The crouch-walk cycle, in particular, was a notable success, demonstrating effective control of robot posture and step sequence.

## VI. Discussion

This experiment demonstrated the basic principles of robotic control using code. The main takeaway was how altering the motor velocity and movement sequence can change the robot's behavior. There were no major challenges during the experiment, though it could be improved by introducing more complex control algorithms or adding sensor feedback for autonomous movement. Future experiments might explore the integration of real-world data to simulate more realistic robot behavior.

## VII. Conclusion

The experiment successfully met its objectives by allowing the robot's movement to be controlled and modified through changes in the program. It provided valuable insight into the control mechanisms behind robotics. The crouch-walk cycle functioned as expected, and the robot responded to programming changes with accuracy. Future improvements may include adding more complex movement patterns and integrating sensor data for enhanced control.

## VIII. References


- Webots: https://cyberbotics.com/


## Appendix

### Appendix: Program Listing with Comments

```
#include <webots/robot.h>
#include <webots/motor.h>

#define NUMBER_OF_JOINTS 24

WbDeviceTag motors[NUMBER_OF_JOINTS];
const char *motor_names[NUMBER_OF_JOINTS] = {
  "front_left_hip", "front_left_knee", "
      front_left_ankle",
  "front_right_hip", "front_right_knee", "
      front_right_ankle",
  "rear_left_hip", "rear_left_knee", "
      rear_left_ankle",
  "rear_right_hip", "rear_right_knee", "
      rear_right_ankle",
  "front_left_shoulder", "front_right_shoulder
      ",
  "rear_left_shoulder", "rear_right_shoulder",
  "extra_motor1", "extra_motor2", "
      extra_motor3", "extra_motor4",
  "extra_motor5", "extra_motor6", "
      extra_motor7", "extra_motor8"
};

void movement_decomposition(const double *
    target_positions, double duration);

void init_motors() {
  for (int i = 0; i < NUMBER_OF_JOINTS; ++i) {
    motors[i] = wb_robot_get_device(
        motor_names[i]);
    wb_motor_set_position(motors[i], 0.0);
    wb_motor_set_velocity(motors[i], 0.5);
  }
}

static void crouch_position(double duration) {
  const double motors_target_pos[
      NUMBER_OF_JOINTS] = {
    -0.50, -0.50, 1.2,  // Front left leg
     0.50, -0.50, 1.2,  // Front right leg
    -0.40, -0.90, 1.0,  // Rear left leg
     0.40, -0.90, 1.0,  // Rear right leg
    -0.30, -0.30, 0.8,  // Front left shoulder
     0.30, -0.30, 0.8,  // Front right
         shoulder
    -0.20, -0.70, 0.9,  // Rear left shoulder
     0.20, -0.70, 0.9   // Rear right shoulder
  };
  movement_decomposition(motors_target_pos,
      duration);
}

static void crouch_walk_cycle(double duration)
    {
  const double time_per_step = duration / 4;
  const double step_1[NUMBER_OF_JOINTS] = { /*
      Forward movement values */ };
  movement_decomposition(step_1, time_per_step
      );
  crouch_position(time_per_step);
  const double step_3[NUMBER_OF_JOINTS] = { /*
      Alternate step values */ };
  movement_decomposition(step_3, time_per_step
      );
  crouch_position(time_per_step);
}
```

1. Removed LEDs and Cameras: The new version focuses purely on motor control.

2. New Functions Introduced:

- `init_motors()` – Initializes motor devices and sets default positions.
- `crouch_position(double duration)` – Moves the robot to a slow crouch position.
- `crouch_walk_cycle(double duration)` – Introduces a crouch-walking sequence.

3. Velocity Adjustment:

- `wb_motor_set_velocity(motors[i], 0.5);` – Slows down motor movements for smoother control.

4. Modified Loop Behavior:

- Instead of performing various motions, the updated version continuously executes a crouch-walk cycle.