Elective 2 – Robotics Technology

LABORATORY ACTIVITY 1

Virtual Robotics Simulation

SY 2024-2025, 2nd Semester

Name:	**ABARRACOSO, Renzyl Mae N.**	**Date:**	**02/25/2025**
Course & Year:	BSECE 4A		

1. What are the key components of the robot in the project?

- The simulated virtual robot in this project comprises several key components that work together to create a cohesive and dynamic system. At its core, the robot features a robust structure and body that includes a frame serving as the skeleton, with links and joints (both rotary and prismatic) forming its limbs and facilitating movement. This structure is complemented by a suite of sensors—including position sensors for accurate joint tracking, force/torque sensors for safe interaction, and potentially vision, proximity, tactile, and temperature sensors—which collectively provide real-time feedback about the robot's state and environment.

2. How are components inter-related?

- These components are intricately inter-related. The control system, featuring a processor, memory, and specialized software, integrates sensor data and orchestrates the movement of joints and actuators through predefined routines like the movement_decomposition() and dance_move() functions. In the dance sequence, the robot smoothly transitions between movements—leaning left and right, bouncing, shaking, and spinning—by coordinating its structural components with precise sensor feedback and control algorithms. Additionally, the power system ensures that all operations are consistently supported, while the end-effectors, such as grippers or specialized tools, would enable interaction with external objects if needed.

3. In your opinion, explain where could be this kind of robot can be used for?

- In my opinion, this type of robot—capable of executing smooth, complex, and preprogrammed movements—has significant potential applications in several fields. It could be effectively used in entertainment and interactive exhibits where engaging, lifelike motion is essential. Furthermore, its precise movement and sensor integration make it ideal for educational demonstrations, research in robotics and automation, and even industrial applications where nuanced, controlled motions are required. Overall, the design and functionality of this simulated virtual robot underscore its versatility and promise across a broad spectrum of real-world scenarios.

4. The program you used with comments on the instruction you edited or added.

```
/*
 * Spot Robot Dance Routine
 * Routine steps:
 * 1. Lean Left
 * 2. Lean Right
 * 3. Neutral
 * 4. Bounce
 * 5. Shake
 * 6. Spin
 * 7. Final Neutral
 */

#include <webots/camera.h>
#include <webots/device.h>
#include <webots/led.h>
#include <webots/motor.h>
#include <webots/robot.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_LEDS 8
#define NUMBER_OF_JOINTS 12
#define NUMBER_OF_CAMERAS 5

// Devices: motors, cameras, LEDs.
static WbDeviceTag motors[NUMBER_OF_JOINTS];
static const char *motor_names[NUMBER_OF_JOINTS] = {
  "front left shoulder abduction motor", "front left shoulder rotation motor", "front left elbow motor",
  "front right shoulder abduction motor", "front right shoulder rotation motor", "front right elbow motor",
  "rear left shoulder abduction motor", "rear left shoulder rotation motor", "rear left elbow motor",
  "rear right shoulder abduction motor", "rear right shoulder rotation motor", "rear right elbow motor"
};

static WbDeviceTag cameras[NUMBER_OF_CAMERAS];
static const char *camera_names[NUMBER_OF_CAMERAS] = {
  "left head camera", "right head camera", "left flank camera", "right flank camera", "rear camera"
};

static WbDeviceTag leds[NUMBER_OF_LEDS];
static const char *led_names[NUMBER_OF_LEDS] = {
  "left top led", "left middle up led", "left middle down led",
  "left bottom led", "right top led", "right middle up led",
  "right middle down led", "right bottom led"
};

// Step simulation.
static void step() {
```

```
  const double time_step = wb_robot_get_basic_time_step();
  if (wb_robot_step(time_step) == -1) {
    wb_robot_cleanup();
    exit(0);
  }
}

// Smoothly move joints to target positions.
static void movement_decomposition(const double *target, double duration) {
  const double time_step = wb_robot_get_basic_time_step();
  const int steps = duration * 1000 / time_step;
  double diff[NUMBER_OF_JOINTS], current[NUMBER_OF_JOINTS];

  for (int i = 0; i < NUMBER_OF_JOINTS; ++i) {
    current[i] = wb_motor_get_target_position(motors[i]);
    diff[i] = (target[i] - current[i]) / steps;
  }

  for (int s = 0; s < steps; ++s) {
    for (int i = 0; i < NUMBER_OF_JOINTS; ++i) {
      current[i] += diff[i];
      wb_motor_set_position(motors[i], current[i]);
    }
    step();
  }
}

// Unused functions kept for reference.
static void lie_down(double duration) __attribute__((unused));
static void lie_down(double duration) {
  const double target[NUMBER_OF_JOINTS] = {
    -0.40, -0.99, 1.59,  0.40, -0.99, 1.59,
    -0.40, -0.99, 1.59,  0.40, -0.99, 1.59
  };
  movement_decomposition(target, duration);
}

static void stand_up(double duration) __attribute__((unused));
static void stand_up(double duration) {
  const double target[NUMBER_OF_JOINTS] = {
    -0.1, 0.0, 0.0,  0.1, 0.0, 0.0,
    -0.1, 0.0, 0.0,  0.1, 0.0, 0.0
  };
  movement_decomposition(target, duration);
}

static void sit_down(double duration) __attribute__((unused));
static void sit_down(double duration) {
  const double target[NUMBER_OF_JOINTS] = {
    -0.20, -0.40, -0.19,  0.20, -0.40, -0.19,
```

```
    -0.40, -0.90, 1.18,   0.40, -0.90, 1.18
  };
  movement_decomposition(target, duration);
}

static void give_paw() __attribute__((unused));
static void give_paw() {
  const double target1[NUMBER_OF_JOINTS] = {
    -0.20, -0.30, 0.05,  0.20, -0.40, -0.19,
    -0.40, -0.90, 1.18,   0.49, -0.90, 0.80
  };
  movement_decomposition(target1, 4);
  const double initTime = wb_robot_get_time();
  while (wb_robot_get_time() - initTime < 8) {
    wb_motor_set_position(motors[4], 0.2 * sin(2 * wb_robot_get_time()) + 0.6);
    wb_motor_set_position(motors[5], 0.4 * sin(2 * wb_robot_get_time()));
    step();
  }
  const double target2[NUMBER_OF_JOINTS] = {
    -0.20, -0.40, -0.19,  0.20, -0.40, -0.19,
    -0.40, -0.90, 1.18,   0.40, -0.90, 1.18
  };
  movement_decomposition(target2, 4);
}

// Dance routine with extra spin step.
static void dance_move() {
  // 1. Lean Left (2 sec)
  const double left[NUMBER_OF_JOINTS] = {
    -0.3, -0.1, 0.1,  0.0, 0.0, -0.1,
    -0.3, -0.1, 0.1,  0.0, 0.0, -0.1
  };
  movement_decomposition(left, 2.0);

  // 2. Lean Right (2 sec)
  const double right[NUMBER_OF_JOINTS] = {
    -0.0, 0.0, -0.1,  0.3, 0.1, 0.1,
    -0.0, 0.0, -0.1,  0.3, 0.1, 0.1
  };
  movement_decomposition(right, 2.0);

  // 3. Neutral (1 sec)
  const double neutral[NUMBER_OF_JOINTS] = {
    -0.1, 0.0, 0.0,  0.1, 0.0, 0.0,
    -0.1, 0.0, 0.0,  0.1, 0.0, 0.0
  };
  movement_decomposition(neutral, 1.0);

  // 4. Bounce: Quick crouch and stand-up (0.5 sec each)
  lie_down(0.5);
```

```
  stand_up(0.5);

  // 5. Shake: Move right elbow (3 sec)
  double startTime = wb_robot_get_time();
  while (wb_robot_get_time() - startTime < 3.0) {
    double pos = 0.2 * sin(3 * wb_robot_get_time());
    wb_motor_set_position(motors[5], pos);
    step();
  }

  // 6. Spin: Twist shoulders (2 sec)
  const double spin[NUMBER_OF_JOINTS] = {
    -0.15, 0.5, 0.0,  0.15, -0.5, 0.0,
    -0.15, 0.5, 0.0,  0.15, -0.5, 0.0
  };
  movement_decomposition(spin, 2.0);

  // 7. Final Neutral (1 sec)
  movement_decomposition(neutral, 1.0);
}

int main(int argc, char **argv) {
  wb_robot_init();

  const double time_step = wb_robot_get_basic_time_step();

  // Initialize cameras.
  for (int i = 0; i < NUMBER_OF_CAMERAS; ++i)
    cameras[i] = wb_robot_get_device(camera_names[i]);
  wb_camera_enable(cameras[0], 2 * time_step);
  wb_camera_enable(cameras[1], 2 * time_step);

  // Initialize LEDs.
  for (int i = 0; i < NUMBER_OF_LEDS; ++i) {
    leds[i] = wb_robot_get_device(led_names[i]);
    wb_led_set(leds[i], 1);
  }

  // Initialize motors.
  for (int i = 0; i < NUMBER_OF_JOINTS; ++i)
    motors[i] = wb_robot_get_device(motor_names[i]);

  // Run dance routine in a loop.
  while (true) {
    dance_move();
  }

  wb_robot_cleanup();
  return EXIT_FAILURE;
}
```