LAB Experiment 1: Robotics and Embedded Systems

Renzyl Mae N. Abarracoso
Samar State University - College of Engineering
Bachelor of Science in Electronics Engineering
Catbalogan City, Samar, Philippines
Email: abarracosorenzylgmail.com

***Abstract*—This laboratory experiment introduces the fundamental concepts of robotic control systems through simulation. Students gain hands-on experience by modifying embedded C programs that control robot behavior in Webots. The experiment emphasizes the critical relationship between embedded programming and robot motion while building foundational skills in robotic simulation.**

***Index Terms*—Robotics, Embedded Systems, Simulation, Webots, Robot Programming**

I. Rationale

The rapid advancement of robotics and automation has significantly impacted various industries, necessitating proficiency in robotic control systems among electronics engineering students. This laboratory activity serves as an introductory exercise to familiarize students with fundamental robotic concepts using simulation environments. Simulating robot movements enables safe experimentation, rapid iteration, and early error detection before real-world deployment [1]. Moreover, mastering embedded programming is essential in robotics, where software directly dictates hardware behavior.

II. Objectives

The objectives of this laboratory experiment are as follows:

- To familiarize students with basic robotic control through a virtual simulation environment.
- To edit and modify an embedded C program that controls robot movements within the Webots simulation platform.
- To observe and analyze the effect of programmatic changes on robot behavior.
- To develop foundational skills in embedded system programming and robot behavior interpretation through simulation.

III. Materials and Software

- **Software:** Webots Robotic Simulation Software (Latest Version), C Programming Environment (Webots Editor)
- **Virtual Hardware:** Boston Dynamics-inspired Quadruped Robot (SPOT model)
- **Operating System:** Windows 10/11

IV. Procedures

1) Download and install Webots simulation software from the Cyberbotics website.
2) Open Webots and load the file `spot.wbt` under the `boston_dynamics` directory.
3) Access and edit the embedded C program controlling the robot.
4) Modify the dance routine using functions like `movement_decomposition()` and `dance_move()`.
5) Save the program and simulate the movement.
6) Observe and record the behavior changes.

V. Observations and Data Collection

TABLE I
Summary of Observed Robot Movements

Edited Movement	**Observed Behavior**
Lean Left	Robot tilts smoothly to the left.
Lean Right	Robot shifts weight to the right.
Bounce	Quick crouch and stand motion.
Shake	Right elbow periodically shakes.
Spin	Shoulder joints twist to spin.
Final Neutral	Balanced standing posture.

VI. Data Analysis

The robot's smooth and accurate movements confirmed that:

- The `movement_decomposition()` function provided gradual joint motion over calculated steps.
- Timing adjustments significantly affected movement dynamics.
- Real-time simulation accurately reflected the programmed sequence without lag or instability.

An example formula used:

$$\text{Steps} = \frac{\text{Duration} \times 1000}{\text{Time Step}} \tag{1}$$

VII. Discussion and Interpretations

Program changes successfully translated into observable robot behaviors, showcasing the importance of:

- Embedded system control over robotic hardware.
- Smooth motion planning for realistic robotic actions [2].

Minor sources of error included extreme angle settings or short movement durations causing jerky motions. Future work could involve dynamic balance control and sensor-based adaptive behaviors.

VIII. Conclusion

This laboratory activity successfully met its objectives, providing foundational skills in robotic simulation, embedded programming, and motion analysis. Students observed first-hand how embedded software controls robot behavior, setting the stage for more advanced robotic system development.

Appendix: Spot Robot Dance Code

```
/*
* 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.
48.
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;
}
```

REFERENCES

[1] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.