**ELECTIVE 2 – ROBOTICS TECHNOLOGY**

**LABORATORY ACTIVITY 1**

**VIRTUAL ROBOTICS SIMULATION**

**SETH DANIEL U. YGRUBAY**

**BACHELOR OF SCIENCE IN ELECTRONICS ENGINEERING, SECTION 4A**

**S.Y. 2024-2025**

**2ND SEMESTER**

**KEY COMPONENTS**

**Structure and Body**

* Frame:

The Boston Dynamics robot, Spot, has a rigid central body or chassis that mimics the form of a dog. The frame houses its power system and control electronics, as well as providing a foundation to which the legs and other end-effectors may be attached.

* Joints and Motors:

As a robot that mimics a dog, Spot is a quadruped with four legs, each consisting of multiple joints. In the Webots simulation, the program specifically has 3 joints/motors, namely the shoulder abduction, shoulder rotation, and elbow joints. These are responsible for all motions required for movement, balancing, and, in the simulation’s case, “jumping”.

**Sensors**

* Vision and Depth Sensors:

Spot is equipped with cameras and depth sensors that allow it to obtain visual feedback for navigation and obstacle detection. In the Webots simulation, these components are not utilized, or at the very least, not in an obvious way.

* Force and Torque Sensors:

Spot is equipped with sensors on the feet to detect contact forces, which helps in adjusting the leg motions in real time to ensure proper push-off while walking or jumping. In the Webots simulation, this component is not obvious, but it is what allows Spot to “jump” and return to in original crouching position.

* Inertial Measurement Units (IMUs):

IMUs provide information about Spot’s orientation and acceleration. Though not obvious in the Webots simulation, it is another essential, yet not obvious component that allows Spot to not use too much or too little force in each “jump”.

**Control System**

* Central Processing Unit:

The control system includes onboard computers that process sensor data and execute control algorithms, which compute the necessary motor commands to achieve smooth movements.

* Motor Planning and Inverse Kinematics:

In the Webots simulation, this component is performed by the *move\_legs()* function, found in line 49, which interpolates joint positions over time based on the original position, the target position, and feedback from the sensors.

* Feedback Loops:

Sensor data is continuously fed back into the control system, allowing for real-time corrections to ensure stability and precise control during movement. In the Webots simulation, this component can be somewhat observed in the move\_legs() fuction, specifically the *Joint Movement Updates* loop, although it does not capture the full extent of an actual feedback loop. Specifically, the loop does not actually obtain feedback from sensors but assumes that the joints have perfectly done what is commanded of it.

**Power System**

* Battery Energy Source:

Spot is powered by high-capacity batteries that supply energy to the motors, sensors, and control electronics. This component is not present in the Webots simulation.

* Power Electronics and Distribution:

Power management circuits regulate the energy delivered to each component, ensuring that the joints/motors receive the necessary current for precise control without overloading the system.

**End-Effector**

* Feet:

Spot, as a mobile robot, has four feet that serve as its main end-effectors. They act as the point of contact with the environment, allowing for movement. In the Webots simulation, Spot uses these feet to generate enough force to push off the ground.

* Optional Manipulation Tools:

Later versions of Spot has an additional arm end-effector directly behind its “face”, which allows it to manipulate the environment with more precision. In the Webots simulaton, Spot does not have this grabber arm.

**COMPONENT INTER-RELATION**

**Mechanical Integration**

The rigid body chassis holds all components together, while the legs and joints provide mobility. The feet as end-effectors directly interact with the environment, translating motor commands into physical movement.

**Sensor-Driven Control**

Sensors feed real-time data into the control system, which are used to adjust motor commands to ensure that movements are accurate and responsive to changes in the environment.

**Energy Management**

The power system supplies the necessary energy for all operations. Efficient energy distribution is critical to maintaining continuous operation of sensors, processors, and joints.

**ROBOT APPLICATION**

In my opinion, the robot dog Spot can be used as a domestic and entertainment robot. Its grabber arm allows Spot to assist in simple tasks such as fetching, dragging, pulling, pushing, and more. As a dog mimicry, Spot can also act as a companion to replace biological pets for households that have issues with them, such as allergies or lack of capabilities to raise an additional living being.

**CODE**

|  |
| --- |
| // Libraries  #include <webots/motor.h>  #include <webots/robot.h>  #include <stdlib.h>  #define NUMBER\_OF\_JOINTS 12  #define JUMP\_DURATION 0.5  #define RECOVERY\_DURATION 0.5  // ---------------------------------------------------------------------------------------------------------------  // Arrays  // Motor Tag Array  static WbDeviceTag motors[NUMBER\_OF\_JOINTS];  // Motor Name Array  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"  };  // ---------------------------------------------------------------------------------------------------------------  // Functions  // Function to Perform Time Step  static void step() {  // Get Time Step  const double time\_step = wb\_robot\_get\_basic\_time\_step();    // End Simulation  if (wb\_robot\_step(time\_step) == -1) {  wb\_robot\_cleanup();  exit(0);  }  }  // Function for Leg Joint Movements  static void move\_legs(const double \*target, double duration) {  const double time\_step = wb\_robot\_get\_basic\_time\_step();    // Calculate number of steps needed for a duration  // Multiply by 1000 for milliseconds  const int steps = duration \* 1000 / time\_step;    // Joint Position Arrays  double current\_position[NUMBER\_OF\_JOINTS];  double step\_difference[NUMBER\_OF\_JOINTS];  // Joint Movement Calculation  for (int i = 0; i < NUMBER\_OF\_JOINTS; ++i) {  current\_position[i] = wb\_motor\_get\_target\_position(motors[i]);  step\_difference[i] = (target[i] - current\_position[i]) / steps;  }  // Joint Movement Updates  for (int i = 0; i < steps; ++i) {  for (int j = 0; j < NUMBER\_OF\_JOINTS; ++j) {  current\_position[j] += step\_difference[j];  wb\_motor\_set\_position(motors[j], current\_position[j]);  }  step();  }  }  // Jump Function  static void jump() {  // Joint Position Values - Original  const double crouch\_position[NUMBER\_OF\_JOINTS] = {  -0.5, -1.0, 1.59, // Front left leg, Shoulder Abduction Motor, Shoulder Rotation Motor, Elbow Motor  0.5, -1.0, 1.59, // Front right leg  -0.5, -1.0, 1.59, // Rear left leg,  0.5, -1.0, 1.59 // Rear right leg  };  // Joint Position Values - Target  const double launch\_position[NUMBER\_OF\_JOINTS] = {  -0.1, -0.1, 0.8,  0.1, -0.1, 0.8,  -0.1, -0.5, 0.5,  0.1, -0.5, 0.5  };  // Move to Crouch  move\_legs(crouch\_position, JUMP\_DURATION);    // Move to Jump  move\_legs(launch\_position, JUMP\_DURATION);    // Return to Cruch  move\_legs(crouch\_position, RECOVERY\_DURATION);  }  // ---------------------------------------------------------------------------------------------------------------  // Command Loop  int main() {  wb\_robot\_init();  // Loop to Retrieve Joint/Motor Name  for (int i = 0; i < NUMBER\_OF\_JOINTS; ++i)  motors[i] = wb\_robot\_get\_device(motor\_names[i]);  // Repeat Jump  while (true) {  jump(); // Perform one full jump sequence.  }  // Cleanup Resources - Just In Case  wb\_robot\_cleanup();  return 0;  } |