DEPARTMENT OF ELECTRICAL ENGINEERING & ELECTRONICS

Quadruped Robot Group Project

(ELEC330)

Assignment 3 report

Author: Zhibin Mo

Haoran Lu

Alex

Aidan

Project Supervisor: Lakany, Heba

Declaration of academic integrity

The standard University of Liverpool statement of academic integrity [6] should go here as follows:

I confirm that I have read and understood the University's Academic Integrity Policy.

I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

Introduction

This is a report for the ROS 2 Jazzy project called Peter. It is committed to designing and simulating a robot in ROS 2 Jazzy in gazebo. With a series of objectives such as successfully simulating in a realistic environment. Getting the robot to move, this would require successful implementation of gait and obstacle avoidance. The robot also has the ability to detect obstacles. Using an inbuilt camera, the robot can identify objects.

Gazebo is in charge of simulating the physics of the simulation.

Recent developments in legged robotics have brought attention to how well quadruped systems may function in unstructured settings. This project uses contemporary robotics frameworks to develop a fully open-source solution, building on previous research in dynamic locomotion and vision systems. Before being physically deployed, ROS 2 Jazzy's interaction with Gazebo offers a reliable environment for creating and evaluating autonomous behaviours in simulation.

System Architecture

Three main goals were the focus of this project; the first was to create a model of a quadruped robot that has realistic sensor setups and dynamics. The second is to put in place adaptive gait control that can react to barriers in the environment. The third step is to develop a perception pipeline that combines vision and LiDAR data to interpret the surroundings. The purpose of these goals was to confirm that autonomous navigation works alongside being safe and efficient.

The quadrupedal architecture of the simulated robot platform includes twelve individually controlled revolute joints, three on each leg. An RGB camera that takes 640x480 resolution photos at 30 frames per second and a 90-degree field-of-view LiDAR that operates at 10Hz with a resolution of 1 cm are examples of sensor integration. Realistic physics characteristics, such as a gravity acceleration of 9.8 m/s^2 and ground friction coefficients that correspond to concrete surfaces, are incorporated into the Gazebo simulation.

Implementation

In accordance with ROS 2 principles, the system architecture uses a distributed node-based design. Core functions are handled by four main nodes: the perception node processes sensor input, the navigation node implements obstacle avoidance, the joint controller controls movement via a finite state machine implementation, and the system monitor performs diagnostics.

To make sure the robot model would work with Gazebo simulation, it was created in SDF format. The inertial characteristics of each link in the kinematic chain were given consideration in order to ensure stability during dynamic motion. The LiDAR sensor provided depth data through the /lidar_scan topic, and camera images were published to

/boxes_image at 30 frames per second. The sensor setups were built using Gazebo plugins. The entire model is made up of 15 rigid links joined by 12 actuated joints.

A state machine with four separate phases, is implemented using the gait control algorithm, yielding a stepping frequency of 1.25 Hz. To guarantee smooth motion, joint trajectories are produced by interpolating between critical places using cubic spline technology. Using a segmentation algorithm, the obstacle avoidance subsystem splits the 90-degree scan into left and right sectors and calculates the minimum distances in each. The system initiates avoidance manoeuvres by altering the gait pattern to divert away from the impediment while keeping balance when obstacles are recognised under the 0.3m threshold.

Through the cv_bridge ROS 2 interface, the implementation makes use of OpenCV's optimised functions.

Individual efforts

Haoran Lu – gmapping and SLAM design

As part of the team, I was assigned responsibility for implementing the robot's autonomous mapping and navigation subsystem. This involved constructing a 2D occupancy grid map using the ROS 2 package slam_toolbox and deploying the Nav2 stack for autonomous goal-directed navigation. These functionalities form the core of a mobile robot's environmental awareness and decision-making system.

Simultaneous Localization and Mapping (SLAM) has become a cornerstone of autonomous mobile robotics, enabling a robot to incrementally construct a map of an unknown environment while simultaneously estimating its own pose, without relying on external localization systems such as GPS. Among various SLAM techniques, GMapping remains a widely used 2D algorithm based on Rao-Blackwellized Particle Filters (RBPF). It maintains multiple trajectory hypotheses, each associated with a corresponding map, and uses probabilistic updates to select the most likely path-map combination as the robot moves and collects laser scan data.

In recent ROS 2 implementations, slam_toolbox has emerged as a robust alternative to traditional GMapping. It supports both synchronous and asynchronous SLAM modes, provides advanced capabilities such as pose-graph optimization and loop closure, and allows for efficient map serialization. These features improve SLAM performance in dynamic or large-scale environments and are more compatible with ROS 2 modularity and real-time constraints.

Navigation in ROS 2 is handled by the Navigation 2 (Nav2) stack, a modular replacement for the legacy move_base framework from ROS 1. Nav2 incorporates multiple components, including AMCL for probabilistic localization using laser scans, global planners (e.g., Dijkstra, A*) for path generation, and local controllers that compute real-time velocity commands for smooth and collision-free path following. Behavior Trees (BT) orchestrate high-level logic, enabling recovery actions and flexible navigation behavior. Both global and

local planners rely on dynamic costmaps, which integrate sensor data to model obstacles in the environment.

Successful deployment of SLAM and Nav2 requires synchronized integration of sensor inputs, transform broadcasts (TF) between coordinate frames such as map, odom, and base_link, and particularly reliable odometry data. Inaccuracies or disruptions in any of these systems can compromise localization accuracy, map integrity, and navigation reliability—challenges that became critical in this project due to the unique motion model of the quadrupedal platform and the absence of conventional odometry sources.

A critical dependency for both SLAM and Nav2 is accurate odometry. In wheeled differential drive robots, odometry is typically computed by integrating the rotation of left and right wheels over time using wheel encoders. This method, while susceptible to drift, is straightforward and well-supported by existing ROS packages such as diff_drive_controller or robot_localization. Because the wheel-ground contact is continuous and planar, small errors accumulate slowly and can often be corrected through SLAM loop closures.

In contrast, quadrupedal robots present a fundamentally more complex challenge. Unlike wheels, legs do not rotate continuously but follow discrete gait cycles involving lifting, swinging, and placing. Ground contact is intermittent and non-uniform, and slippage or variation in terrain further complicates the estimation of linear and angular displacement. Additionally, the robot's motion depends on coordination among multiple actuators—typically 12 or more in a full quadruped—making inverse kinematics and proprioception-based odometry estimation computationally intensive and highly error-prone.

SLAM (Simultaneous Localization and Mapping) enables a robot to incrementally build a map of its environment while estimating its own pose. The chosen implementation relied on async_slam_toolbox_node, a modern ROS 2 alternative to traditional GMapping, offering features such as loop closure and pose-graph optimization. The configuration was defined in the file mapper_params_online_async.yaml, specifying critical parameters such as:

```
odom_frame: odometry/filtered
map_frame: map
base_frame: body
scan_topic: /scan
map_update_interval: 5.0
resolution: 0.05
do_loop_closing: true
```

These settings determined the scan source, coordinate frames, map resolution, update intervals, and enabled loop closure to correct long-term drift. The system was launched using a custom ROS 2 launch file slam_launch.py, which included the following SLAM node and topic remappings:

```
Node(
        package='slam_toolbox',
        executable='async_slam_toolbox_node',
        parameters=[configured_params],
        remappings=[('/peter1/body/gpu_lidar/scan', 'scan')]
)
```

The launch file also included conditional logic to check whether the user-supplied parameters were valid using:

```
HasNodeParams(source_file=params_file, node_name='slam_toolbox')
```

Although both the configuration and launch scripts were correctly implemented, the system could not be deployed successfully due to the absence of reliable odometry. The robot employed a 12-motor quadrupedal configuration, which significantly complicates pose estimation compared to wheeled robots. Without a stable odometry source, SLAM failed to initialize properly, as accurate motion updates are essential for scan registration and map alignment. Attempts to approximate odometry using inertial or kinematic estimators were unsuccessful within the scope and timeframe of the project.

Despite this limitation, the process provided valuable experience in SLAM configuration, ROS 2 launch mechanics, TF tree debugging, and scan integration. Future work may incorporate advanced estimation methods such as an Extended Kalman Filter or external localization systems to overcome odometry-related constraints in legged robotic platforms.

System Integration Attempted

Although the mapping functionality could not be fully realized due to the lack of reliable odometry, considerable effort was invested in preparing the surrounding system architecture for SLAM and navigation. The ROS 2 transform tree was configured with appropriate static and dynamic TF broadcasters to establish spatial relationships among the `map`, `odom`, and `base_link` frames. Sensor topics, including laser scans from `/peter1/body/gpu_lidar/scan`, were successfully remapped to the standardized `scan` topic to comply with SLAM input expectations.

RViz2 was configured to visualize sensor data, frame transformations, and prospective pose estimates. Visualization confirmed that laser data was being published correctly, and the SLAM node was receiving partial inputs; however, the missing `odom → base_link` transform prevented the SLAM system from performing scan registration and map updates.

Preparations were also made for downstream navigation by establishing compatible topic names and organizing the launch files to support modular activation of localization, planning, and control components once valid odometry becomes available. These integration steps ensured that the overall system could be resumed with minimal changes when future improvements to the state estimation pipeline are implemented.

Testing and Debugging Process

To validate the configuration and identify the root causes of failure, a structured testing and debugging process was carried out throughout the development cycle. ROS 2 tools such as `ros2 topic echo` and `ros2 interface show` were used to monitor message streams on key topics, particularly `/scan` and various TF-related transforms. The laser scan topic was verified to be publishing correctly at expected frequencies, and its remapping to `scan` was confirmed using RViz2.

However, the absence of the `odom → base_link` transform was immediately evident during runtime, as the SLAM node repeatedly issued transform timeout warnings. This confirmed

that the failure to initialize mapping originated from the lack of valid odometry input, which is critical for estimating relative motion between successive laser scans.

Multiple approaches were explored to emulate or approximate odometry data. These included fusing IMU readings using a preliminary robot_localization setup and implementing kinematic models based on joint encoders. Nevertheless, the complexity of the quadruped's gait and the lack of accurate ground-contact feedback made it infeasible to generate usable pose estimates within the available project timeline.

These findings were instrumental in refining the system's diagnostic pipeline and improving the team's understanding of the dependencies within the ROS 2 SLAM and navigation stacks. It also established a foundation for future development of a dedicated state estimation module tailored for legged locomotion.

Skills Acquired and Technical Growth

Despite the incomplete implementation of the mapping and navigation functionalities, this project provided significant opportunities for technical growth. Through configuring and launching SLAM modules in ROS 2, I developed a deeper understanding of the slam_toolbox architecture, including its parameter hierarchies, operational modes (synchronous vs. asynchronous), and integration requirements with transform and sensor data.

I gained practical experience in writing modular and reusable launch files using Python-based ROS 2 launch syntax, handling parameter substitution, and remapping topics for system compatibility. Working with TF transformations highlighted the importance of frame hierarchy integrity in robotic systems, and I became proficient in using visualization tools like RViz2 for debugging data flow and frame consistency.

Furthermore, this project emphasized the challenges of perception and localization in non-wheeled robots. By attempting to generate odometry from nontraditional sources such as IMU and proprioceptive feedback, I improved my understanding of state estimation pipelines and the limitations of standard ROS localization tools when applied to legged locomotion.

Collectively, these experiences reinforced my ability to architect and debug complex robotic systems in ROS 2 and laid the groundwork for future research in state estimation, SLAM adaptation, and navigation in bio-inspired mobile platforms.

Future Work and Recommendations

The primary limitation encountered in this project—the absence of reliable odometry—can be addressed in future work through several potential strategies. One viable approach is to implement a state estimator based on an Extended Kalman Filter (EKF), fusing data from inertial measurement units (IMUs), joint encoders, and foot contact sensors. This would provide a continuous pose estimate compatible with SLAM requirements and suitable for use in the Nav2 stack.

Alternatively, vision-based localization methods such as AprilTag tracking or visual-inertial odometry (VIO) could be explored to generate external pose estimates independent of proprioception. These methods are increasingly robust and may be particularly effective in indoor environments where GPS is unavailable and ground contact is inconsistent.

The modular system architecture and parameterized launch structure developed in this project make it feasible to reintroduce SLAM and navigation functionalities once odometry becomes available. It is also recommended to build a lightweight simulation environment to evaluate different odometry strategies before full deployment on the physical robot.

Ultimately, addressing the odometry gap is key to enabling full autonomy in quadrupedal robotic platforms. By building upon the current foundation, future iterations of this project can realize robust SLAM integration and enable safe, reliable navigation in complex environments.

Alex

The methodology in which I helped the robot task is through annotating the code, troubleshoot various functions such as node mapping and renaming thing so that they were relevant to each other.

I also conducted research and created an economical Bill of Materials (BoM), carefully choosing parts to satisfy financial restrictions without sacrificing functionality. I made a substantial contribution to the finished report by writing important passages and guaranteeing technical correctness. I also assisted with cross-team work by troubleshooting dependencies for Mo's gait control and Haoran's SLAM/Nav2 interface.

In terms of the writing and report I wrote the majority of this report including but not limited to these sections: Introduction, System architecture, Implementation, Challenges, Future improvements and Conclusion.

Aidan

In this project I was tasked with environment modeling and scene configuration, system testing and tuning optimization as well as being a proofreader and editor to ensure that grammar and spelling is correct. This is mainly due to the group being multicultural and being the only group member in which english is their first language. Due to this I wrote the executive summary which will be the main part that is entirely read by the marker as well as doing quality assurance on this report. The methodology in which the system testing and tuning optimization was carried out was through a mixture of learning from the tutorials available online as well as trial and error. Whenever a new part was developed for the robot it was tested in the environment I made in gazebo to ensure it could work seamlessly with the rest of the components. There were times in which we tried to add new things that didn’t work that we had to remove due to us not being able to figure out the issue. This was down to the lack of resources available online. I did not want to turn to things like chatGPT to answer my questions or do my work for me. I used real world data and data used in previous projects to create the sdf file of the world to include the objects as well as the physics, such as the coefficient of friction and the gravity. During the scheduled 3-hour lab sessions, Alex and I

were consistently present from the start and more often than not used the full duration for development and testing instead of working on new developments or work for the project. As a result, the bulk of the hands-on work and troubleshooting during lab hours was carried out by Alex and I.

Mo

In our group project to simulate an autonomous quadruped robot with ROS2 and Gazebo, I was primarily responsible for the construction of the robot model, the development of the SDF file with plugin integration, the design of the quadrupedal walking gait, and the radar sensor-based obstacle avoidance functionality. These elements formed the structural and functional foundation that allowed the robot to perform motion control, respond to sensor inputs, and interact with the virtual environment. The project is divided into the following main modules:

My focus was to ensure that the robot loaded smoothly into the simulation environment and had a structurally stable model, controllable joints, and both linear and steering motions in accordance with the joint movements. This laid the groundwork for the rest of the team to implement navigation and object detection features.

Methodology and implementation
I modelled the robot using Creo, generated the URDF file using the plugin of SolidWorks, and further revised it to get the final robot SDF description file. Each leg consists of three rotational joints (hip, knee, and ankle) with a total of 12 controllable joints. Key considerations include the definition of the <link> and <joint> hierarchies; the application of physical parameters (e.g., friction): mass, inertia, and collision geometry; and the enhancement of stability through the introduction of flat foot contact surfaces. Loading of geometry through STL mesh files for more accurate visualization and collision.

To enable dynamic interaction, I embedded several key plugins into the SDF file:

Modules	**Descriptions**
JointStatePublisher	used to publish joint states in real-time for RViz or other debugging tools to visualize robot poses
JointPositionController	supports precise position control of individual joints, working with PID parameters to realize smooth movements
IMU and Lidar plug-in integration	provides the necessary sensory input interface for subsequent navigation modules.

Table 1. The plugin modules in the Robot SDF file

Using ROS2 and Python, I implemented a four-phase gait that cyclically lifts and moves opposing legs (e.g., left front leg and right back leg). It is controlled primarily through the joint_controller.py node, which posts to all 12 joints. The gait is also adjusted based on real-time LiDAR feedback to avoid detected obstacles. The gait control system allows the robot to not only move, but also to adjust its motion based on environmental inputs.

Testing scenarios
The test was conducted in a customized gazebo world with slopes, walls, and dynamic obstacles. I performed model integrity tests including joint motion, foot contact behaviour; step cycle validation by observing leg synchronization and stability; obstacle interaction tests using synthetic lidar inputs and video capture; and object detection by introducing OpenCV through the camera.

Challenges

The Gazebo-ROS 2 bridge arrangement presented another major challenge, requiring careful topic remapping and parameter adjustment to ensure stable connection. Although software-based picture stabilisation added extra delay to the processing pipeline, it helped to somewhat reduce the vision system's performance loss during motion.

Future improvements

Three crucial areas have been noted for further development: First, true autonomous navigation would be made possible by the deployment of a SLAM system employing the current LiDAR data. Second, reinforcement learning optimisation could help the gait controller manage more diverse terrain. With these improvements, the system would be ready for the switch to physical hardware deployment.

Conclusion

Using ROS 2 and Gazebo simulation, this project has effectively shown that autonomous quadruped navigation is feasible. In addition to meeting its primary goals of obstacle avoidance, environment awareness, and steady locomotion, the built system offers a modular framework for future development. The work lays a strong platform for further advancements in legged robotics, despite limitations in perception reliability during movement.

Appendices

Code 1. joint_controller.py

```
import rclpy
```

```
from rclpy.node import Node
from std_msgs.msg import Float64
from sensor_msgs.msg import LaserScan
import threading
class QuadrupedSpider(Node):
  def __init__(self):
    # Initialize the parent class with the node name 'joint_controller'
    super().__init__('joint_controller')
    # Create publishers for each leg joint
    # Each joint has a unique topic for publishing its position
    # 'jlf' -> front-left leg, 'jrf' -> front-right leg
    # 'jlr' -> rear-left leg, 'jrr' -> rear-right leg
    # Numbers (1, 2, 3) represent individual joints in a leg
    self.publishers_joints = {
      'jlf1': self.create_publisher(Float64, 'jlf1_topic', 100),  # Joint 1 of front-left leg
      'jlf2': self.create_publisher(Float64, 'jlf2_topic', 100),  # Joint 2 of front-left leg
      'jlf3': self.create_publisher(Float64, 'jlf3_topic', 100),  # Joint 3 of front-left leg
      'jrf1': self.create_publisher(Float64, 'jrf1_topic', 100),  # Joint 1 of front-right leg
      'jrf2': self.create_publisher(Float64, 'jrf2_topic', 100),  # Joint 2 of front-right leg
      'jrf3': self.create_publisher(Float64, 'jrf3_topic', 100),  # Joint 3 of front-right leg
      'jlr1': self.create_publisher(Float64, 'jlr1_topic', 100),  # Joint 1 of rear-left leg
      'jlr2': self.create_publisher(Float64, 'jlr2_topic', 100),  # Joint 2 of rear-left leg
      'jlr3': self.create_publisher(Float64, 'jlr3_topic', 100),  # Joint 3 of rear-left leg
      'jrr1': self.create_publisher(Float64, 'jrr1_topic', 100),  # Joint 1 of rear-right leg
      'jrr2': self.create_publisher(Float64, 'jrr2_topic', 100),  # Joint 2 of rear-right leg
      'jrr3': self.create_publisher(Float64, 'jrr3_topic', 100),  # Joint 3 of rear-right leg
    }
    # Subscribe to LIDAR data
    # The 'lidar_scan' topic provides LaserScan messages, which are processed in the
callback function
    self.lidar_subscription = self.create_subscription(
      LaserScan,       # Message type for LIDAR data
      'scan',      # Topic name for LIDAR data
      self.lidar_callback, # Callback function to process incoming data
      10             # Queue size for storing incoming messages
    )
    # Create a timer for periodic publishing of joint positions
    # The timer executes the 'publish_joint_positions' method every 0.2 seconds
    self.timer = self.create_timer(0.2, self.publish_joint_positions)
    # Initialize the robot's state variables
    self.active = False          # Indicates whether the robot is actively walking
    self.step = 0              # Counter to track the current gait phase
    self.obstacle_detected = False   # Flag to indicate if an obstacle is detected by LID
AR
```

```
        self.obstacle_side = None       # Tracks whether the obstacle is on the left or right si
de
    def lidar_callback(self, msg):
        """
        Process LIDAR data to detect obstacles and determine their position (left or right).
        Parameters:
        msg (LaserScan): The LIDAR scan message containing an array of distance readin
gs (ranges).
        Functionality:
       - Divides the LIDAR ranges into two halves: left and right.
       - Calculates the minimum distance in each half to identify obstacles.
       - Sets the `obstacle_detected` flag and the `obstacle_side` attribute based on the lo
cation of the closest obstacle.
       - Logs a message indicating obstacle detection and distance.
        """
        # Divide the LIDAR ranges into left and right sections
        # The first half (minus a small overlap) corresponds to the left side
        left_ranges = msg.ranges[:(len(msg.ranges)//2)-5]
        # The second half (including the small overlap) corresponds to the right side
        right_ranges = msg.ranges[(len(msg.ranges)//2)-5:]

        # Find the minimum distance in the left and right ranges
        min_left_distance = min(left_ranges)
        min_right_distance = min(right_ranges)

        # Check if any obstacle is within a critical distance (0.3 meters)
        if min_left_distance < 0.3 or min_right_distance < 0.3:
            self.obstacle_detected = True
             # Determine the side of the obstacle based on the closer minimum distance
            if min_left_distance < min_right_distance:
                self.obstacle_side = 'left'# Obstacle is closer on the left

            else:
                self.obstacle_side = 'right'# Obstacle is closer on the right
        else:
            # No obstacle detected within the critical distance
            self.obstacle_detected = False
            self.obstacle_side = None
    def toggle_active(self):
        """
        Toggle the movement state.
        Realize joint motion by releasing the angle information of the joints
        realize the forward motion and steering of the quadruped robot.
        """
```

```
    self.active = not self.active
    if self.active:
        self.get_logger().info("Motion started.")
    else:
        self.get_logger().info("Motion stopped.")
  def publish_joint_positions(self):
    if self.active:
        # Initialize messages for each joint
        joint_msgs = {name: Float64() for name in self.publishers_joints}
        if self.obstacle_detected:
            # Stop or modify movement based on obstacle position
            for msg in joint_msgs.values():
                msg.data = 0.0
            # When an obstacle is detected to the left, execute the right turn code
            if self.obstacle_side == 'left':
             if self.step % 4 == 0:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = -0.2, 0.7,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = -0.2, 0.7
, 0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
             elif self.step % 4 == 1:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = -0.2, 0.0,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = -0.2, 0.0
, 0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
             elif self.step % 4 == 2:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = -0.2, 0.7
, 0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = -0.2, 0.
7, 0.0
             elif self.step % 4 == 3:
```

```
            joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = -0.2, 0.0
, 0.0
            joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = -0.2, 0.
0, 0.0
        self.step += 1# Move to the next step in the gait cycle
        #When an obstacle is detected to the right, execute the left turn code
        elif self.obstacle_side == 'right':
        if self.step % 4 == 0:
            joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.2, 0.7,
0.0
            joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
 0.0
            joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.2, 0.7,
0.0
            joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
        elif self.step % 4 == 1:
            joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.2, 0.0,
0.0
            joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
 0.0
            joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.2, 0.0,
0.0
            joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
        elif self.step % 4 == 2:
            joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.2, 0.7,
 0.0
            joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.2, 0.7
, 0.0
        elif self.step % 4 == 3:
            joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.2, 0.0,
 0.0
```

```
            joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
            joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.2, 0.0
, 0.0
            self.step += 1# Move to the next step in the gait cycle
        else:
            # Otherwise keep walk straightly
            if self.step % 4 == 0:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.7, 0.7,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = -0.7, 0.7
, 0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
            elif self.step % 4 == 1:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.7, 0.0,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = -0.7, 0.0
, 0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.0, 0.0
, 0.0
            elif self.step % 4 == 2:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = -0.7, 0.7
, 0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.7, 0.7
, 0.0
            elif self.step % 4 == 3:
                joint_msgs['jlf1'].data, joint_msgs['jlf2'].data, joint_msgs['jlf3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrf1'].data, joint_msgs['jrf2'].data, joint_msgs['jrf3'].data = -0.7, 0.0
, 0.0
                joint_msgs['jlr1'].data, joint_msgs['jlr2'].data, joint_msgs['jlr3'].data = 0.0, 0.0,
0.0
                joint_msgs['jrr1'].data, joint_msgs['jrr2'].data, joint_msgs['jrr3'].data = 0.7, 0.0
, 0.0
            self.step += 1# Move to the next step in the gait cycle
```

```
        # Publish joint commands
        for joint, publisher in self.publishers_joints.items():
            publisher.publish(joint_msgs[joint])
def keyboard_listener(node):
    """
    Independent thread to listen for keyboard inputs to control motion state.

    """
    while rclpy.ok():
        cmd = input("start movement: enter s; quit: enter q ")
        match cmd:
            case 's':
                node.toggle_active()
            case 'q':
                rclpy.shutdown()
                break
            case _:
                print("Invalid input error. Please enter 's' to start/stop or 'q' to quit.")
def main(args=None):
    rclpy.init(args=args)
    node = QuadrupedSpider()
    # Start keyboard listener thread
    thread = threading.Thread(target=keyboard_listener, args=(node,))
    thread.start()
    # Start ROS2 event loop
    rclpy.spin(node)
    # Wait for the thread to finish
    thread.join()
if __name__ == '__main__':
    main()
```

Code 2. Peter1.sdf (Robot Description File

```
<?xml version="1.0" encoding="utf-8"?>
<sdf version='1.11'>
  <model name='peter1' canonical_link = 'body'>
    <frame name="lidar_frame" attached_to='body'>
        <pose>0 0 0.15 0 0 3.1415926535</pose>
    </frame>
    <frame name="imu_frame" attached_to='body'>
        <pose>0 0 0.15 0 0 3.1415926535</pose>
```

```
  </frame>
  <frame name="camera_frame" attached_to='body'>
      <pose>0 0 0.05 0 0 3.1415926</pose>
  </frame>
  <link name='body'>
   <inertial>
    <pose>0.0022555536637576899 0.00133065518558914 0.04306473137922620
1 0 0 0</pose>
    <mass>0.063478029760084195</mass>
    <inertia>
     <ixx>7.9230379603712002e-05</ixx>
     <ixy>2.9482940315218001e-07</ixy>
     <ixz>-1.20685983267543e-08</ixz>
     <iyy>7.1253492093192099e-05</iyy>
     <iyz>1.2420643100907299e-08</iyz>
     <izz>0.00014631818171768199</izz>
    </inertia>
   </inertial>
   <collision name='body_collision'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/body.STL</uri>
     </mesh>
    </geometry>
     <surface>
      <friction>
       <ode>
        <mu>1</mu>
        <mu2>1</mu2>
       </ode>
      </friction>
      <bounce/>
      <contact/>
     </surface>
   </collision>
   <visual name='body_visual'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/body.STL</uri>
     </mesh>
```

```
    </geometry>
    <material>
     <diffuse>0.995098054 1 1 1</diffuse>
     <ambient>0.995098054 1 1 1</ambient>
    </material>
   </visual>
<sensor name='gpu_lidar' type='gpu_lidar'>"
     <pose relative_to='lidar_frame'>0 0 0 0 0 0</pose>
     <topic>scan</topic>
     <update_rate>10</update_rate>
     <ray>
       <scan>
         <horizontal>
           <samples>90</samples>
           <resolution>1</resolution>
           <min_angle>-0.785</min_angle>
           <max_angle>0.785</max_angle>
         </horizontal>
         <vertical>
           <samples>1</samples>
           <resolution>0.01</resolution>
           <min_angle>0</min_angle>
           <max_angle>0</max_angle>
         </vertical>
       </scan>
       <range>
         <min>0.05</min>
         <max>10.0</max>
         <resolution>0.01</resolution>
       </range>
     </ray>
     <always_on>1</always_on>
     <visualize>true</visualize>
   </sensor>
   <sensor name="boundingbox_camera" type="boundingbox_camera">
    <pose relative_to='camera_frame'>0 0 0 0 0 0</pose>
    <topic>boxes</topic>
    <camera>
     <box_type>2d</box_type>
     <horizontal_fov>1.047</horizontal_fov>
     <image>
      <width>1280</width>
      <height>720</height>
     </image>
```

```
        <clip>
          <near>0.1</near>
          <far>10</far>
        </clip>
      </camera>
      <always_on>1</always_on>
      <update_rate>30</update_rate>
      <visualize>true</visualize>
    </sensor>
    <sensor name="imu_sensor" type="imu">
    <pose relative_to='imu_frame'>0 0 0 0 0 0</pose>
    <always_on>1</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize>
    <topic>imu/data</topic>
    </sensor>
  </link>
<plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
</plugin>
  <joint name='jlf1' type='revolute'>
   <pose relative_to='body'>-0.04 -0.04 0.04 -1.5708 5.5511151231257827e-17 -0.52
359999999999995</pose>
    <parent>body</parent>
    <child>llf1</child>
    <axis>
     <xyz>0 1 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
  </joint>
  <link name='llf1'>
   <pose relative_to='jlf1'>0 0 0 0 0 0</pose>
   <inertial>
    <pose>0.00286542457 -0.001117447 -0.027575894 0 0 0</pose>
    <mass>0.012789728747716301</mass>
    <inertia>
```

```
      <ixx>2.8085225283823598e-06</ixx>
      <ixy>-1.22116927896592e-08</ixy>
      <ixz>-8.7286357197039904e-08</ixz>
      <iyy>1.9479276383099401e-06</iyy>
      <iyz>-2.2831986328942301e-08</iyz>
      <izz>1.8524298404201401e-06</izz>
     </inertia>
    </inertial>
    <collision name='llf1_collision'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llf1.STL</uri>
      </mesh>
     </geometry>
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
         <mu2>1</mu2>
        </ode>
       </friction>
       <bounce/>
       <contact/>
      </surface>
    </collision>
    <visual name='llf1_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llf1.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
   </link>
   <joint name='jlf2' type='revolute'>
    <pose relative_to='llf1'>0.0022300000000000002 -0.005899999999999999 -0.03
32 0.34907000000000005 2.1175823681357508e-22 3.1415853071795872</pose>
```

```
    <parent>llf1</parent>
    <child>llf2</child>
    <axis>
     <xyz>1 0 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
   <link name='llf2'>
    <pose relative_to='jlf2'>0 0 0 0 0 0</pose>
    <inertial>
     <pose>0 3.46944695195361e-18 -0.024259718544271999 0 0 0</pose>
     <mass>0.0071256681620003801</mass>
     <inertia>
      <ixx>1.51184436411063e-06</ixx>
      <ixy>9.9261673506363302e-22</ixy>
      <ixz>4.7645603283054403e-22</ixz>
      <iyy>2.8950464625635802e-06</iyy>
      <iyz>2.64697796016969e-22</iyz>
      <izz>1.50597424566293e-06</izz>
     </inertia>
    </inertial>
    <collision name='llf2_collision'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llf2.STL</uri>
      </mesh>
     </geometry>
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
         <mu2>1</mu2>
        </ode>
```

```
        </friction>
        <bounce/>
        <contact/>
       </surface>
    </collision>
    <visual name='llf2_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llf2.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
   </link>
   <joint name='jlf3' type='revolute'>
    <pose relative_to='llf2'>-0.00027 6.2114999999999997e-05 -0.050126999999999
998 1.2401 0 0</pose>
    <parent>llf2</parent>
    <child>llf3</child>
    <axis>
     <xyz>-1 0 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
   <link name='llf3'>
    <pose relative_to='jlf3'>0 0 0 0 0 0</pose>
    <inertial>
     <pose>0.00052240477499151299 -0.00014146217495314401 0.007929754286
0836193 0 0 0</pose>
     <mass>0.0099422645031376194</mass>
```

```
    <inertia>
      <ixx>1.9963726602804399e-06</ixx>
      <ixy>4.0330485793124704e-09</ixy>
      <ixz>-4.3733666249029598e-08</ixz>
      <iyy>2.2864873535189799e-06</iyy>
      <iyz>-1.38843083458551e-09</iyz>
      <izz>6.0776447329995005e-07</izz>
    </inertia>
  </inertial>
  <collision name='llf3_collision'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
      <mesh>
        <scale>1 1 1</scale>
        <uri>model://description/meshes/llf3.STL</uri>
      </mesh>
    </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
          </ode>
        </friction>
        <bounce/>
        <contact/>
      </surface>
  </collision>
  <visual name='llf3_visual'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
      <mesh>
        <scale>1 1 1</scale>
        <uri>model://description/meshes/llf3.STL</uri>
      </mesh>
    </geometry>
    <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
    </material>
  </visual>
</link>
<joint name='jlr1' type='revolute'>
```

```
    <pose relative_to='body'>0.04 0.04 0.04 -1.5708000000000002 -2.7755575615628
914e-17 2.6179999999999999</pose>
    <parent>body</parent>
    <child>llr1</child>
    <axis>
     <xyz>0 1 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
   <link name='llr1'>
    <pose relative_to='jlr1'>0 0 0 0 0 0</pose>
    <inertial>
     <pose>0.00286542457 -0.001117447 -0.027575894 0 0 0</pose>
     <mass>0.0127897287485531</mass>
     <inertia>
      <ixx>2.8085225284554198e-06</ixx>
      <ixy>-1.22116928234475e-08</ixy>
      <ixz>-8.72863572603713e-08</ixz>
      <iyy>1.9479276384354398e-06</iyy>
      <iyz>-2.2831986359843301e-08</iyz>
      <izz>1.8524298405056999e-06</izz>
     </inertia>
    </inertial>
    <collision name='llr1_collision'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llr1.STL</uri>
      </mesh>
     </geometry>
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
```

```
          <mu2>1</mu2>
         </ode>
        </friction>
        <bounce/>
        <contact/>
       </surface>
    </collision>
    <visual name='llr1_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/llr1.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
   </link>
   <joint name='jlr2' type='revolute'>
    <pose relative_to='llr1'>0.0022300000000000002 -0.0058999999999999999 -0.03
32 0.34907000000000005 2.1175823681357508e-22 3.1415853071795872</pose>
    <parent>llr1</parent>
    <child>llr2</child>
    <axis>
     <xyz>1 0 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
   <link name='llr2'>
    <pose relative_to='jlr2'>0 0 0 0 0 0</pose>
    <inertial>
```

```
    <pose>-6.9388939039072299e-18 3.46944695195361e-18 -0.024259718544271
999 0 0 0</pose>
    <mass>0.0071256681620003801</mass>
    <inertia>
     <ixx>1.51184436411063e-06</ixx>
     <ixy>7.9409338805090704e-22</ixy>
     <ixz>-5.2939559203393795e-23</ixz>
     <iyy>2.8950464625635802e-06</iyy>
     <iyz>1.8528845721187801e-22</iyz>
     <izz>1.5059742456629201e-06</izz>
    </inertia>
   </inertial>
   <collision name='llr2_collision'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/llr2.STL</uri>
     </mesh>
    </geometry>
     <surface>
      <friction>
       <ode>
        <mu>1</mu>
        <mu2>1</mu2>
       </ode>
      </friction>
      <bounce/>
      <contact/>
     </surface>
   </collision>
   <visual name='llr2_visual'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/llr2.STL</uri>
     </mesh>
    </geometry>
    <material>
     <diffuse>0.995098054 1 1 1</diffuse>
     <ambient>0.995098054 1 1 1</ambient>
    </material>
   </visual>
```

```
  </link>
  <joint name='jlr3' type='revolute'>
   <pose relative_to='llr2'>-0.0004499999999999999 5.9766999999999e-05 -0.
050127999999999999 1.2216999999999998 0 0</pose>
   <parent>llr2</parent>
   <child>llr3</child>
   <axis>
    <xyz>-1 0 0</xyz>
    <limit>
     <lower>-3.14</lower>
     <upper>3.14</upper>
     <effort>10</effort>
     <velocity>10</velocity>
    </limit>
    <dynamics>
     <spring_reference>0</spring_reference>
     <spring_stiffness>0</spring_stiffness>
    </dynamics>
   </axis>
  </joint>
  <link name='llr3'>
   <pose relative_to='jlr3'>0 0 0 0 0 0</pose>
   <inertial>
    <pose>0.00052240466158130804 -0.000141462048715624 0.00792975427728
35093 0 0 0</pose>
    <mass>0.0099422642876608892</mass>
    <inertia>
     <ixx>1.9963727548235902e-06</ixx>
     <ixy>4.0330423282316097e-09</ixy>
     <ixz>-4.3733671753696498e-08</ixz>
     <iyy>2.2864874548568501e-06</iyy>
     <iyz>-1.3884479500935699e-09</iyz>
     <izz>6.0776445098594797e-07</izz>
    </inertia>
   </inertial>
   <collision name='llr3_collision'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/llr3.STL</uri>
     </mesh>
    </geometry>
    <surface>
```

```
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
          </ode>
        </friction>
        <bounce/>
        <contact/>
      </surface>
    </collision>
    <visual name='llr3_visual'>
      <pose>0 0 0 0 0 0</pose>
      <geometry>
        <mesh>
          <scale>1 1 1</scale>
          <uri>model://description/meshes/llr3.STL</uri>
        </mesh>
      </geometry>
      <material>
        <diffuse>0.995098054 1 1 1</diffuse>
        <ambient>0.995098054 1 1 1</ambient>
      </material>
    </visual>
  </link>
  <joint name='jrf1' type='revolute'>
    <pose relative_to='body'>-0.04 0.04 0.04 -1.5708 -5.5511151231257827e-17 0.523
59999999999995</pose>
    <parent>body</parent>
    <child>lrf1</child>
    <axis>
      <xyz>0 1 0</xyz>
      <limit>
        <lower>-3.14</lower>
        <upper>3.14</upper>
        <effort>10</effort>
        <velocity>10</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
      </dynamics>
    </axis>
  </joint>
  <link name='lrf1'>
```

```
<pose relative_to='jrf1'>0 0 0 0 0 0</pose>
<inertial>
 <pose>0.00286542457 -0.001117447 0.027575894 0 0 0</pose>
 <mass>0.012789727077819699</mass>
 <inertia>
  <ixx>2.8085225922082198e-06</ixx>
  <ixy>-1.2211785859690101e-08</ixy>
  <ixz>8.7286419548063804e-08</ixz>
  <iyy>1.9479275047790999e-06</iyy>
  <iyz>2.28320672404181e-08</iyz>
  <izz>1.8524299282918799e-06</izz>
 </inertia>
</inertial>
<collision name='lrf1_collision'>
 <pose>0 0 0 0 0 0</pose>
 <geometry>
  <mesh>
   <scale>1 1 1</scale>
   <uri>model://description/meshes/lrf1.STL</uri>
  </mesh>
 </geometry>
  <surface>
   <friction>
    <ode>
     <mu>1</mu>
     <mu2>1</mu2>
    </ode>
   </friction>
   <bounce/>
   <contact/>
  </surface>
</collision>
<visual name='lrf1_visual'>
 <pose>0 0 0 0 0 0</pose>
 <geometry>
  <mesh>
   <scale>1 1 1</scale>
   <uri>model://description/meshes/lrf1.STL</uri>
  </mesh>
 </geometry>
 <material>
  <diffuse>0.995098054 1 1 1</diffuse>
  <ambient>0.995098054 1 1 1</ambient>
 </material>
```

```
    </visual>
  </link>
  <joint name='jrf2' type='revolute'>
    <pose relative_to='lrf1'>0.0020500000000000002 -0.0058999999999999999 0.03
32 2.7925 -2.1175823681357508e-22 3.1415853071795872</pose>
    <parent>lrf1</parent>
    <child>lrf2</child>
    <axis>
      <xyz>-1 0 0</xyz>
      <limit>
        <lower>-3.14</lower>
        <upper>3.14</upper>
        <effort>10</effort>
        <velocity>10</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
      </dynamics>
    </axis>
  </joint>
  <link name='lrf2'>
    <pose relative_to='jrf2'>0 0 0 0 0 0</pose>
    <inertial>
      <pose>0 -6.9388939039072299e-18 -0.024259718544271999 0 0 0</pose>
      <mass>0.0071256681620003896</mass>
      <inertia>
        <ixx>1.51184436411063e-06</ixx>
        <ixy>-3.4410713482205998e-22</ixy>
        <ixz>1.2705494208814499e-21</ixz>
        <iyy>2.8950464625635899e-06</iyy>
        <iyz>-4.2351647362714998e-22</iyz>
        <izz>1.50597424566293e-06</izz>
      </inertia>
    </inertial>
    <collision name='lrf2_collision'>
      <pose>0 0 0 0 0 0</pose>
      <geometry>
        <mesh>
          <scale>1 1 1</scale>
          <uri>model://description/meshes/lrf2.STL</uri>
        </mesh>
      </geometry>
        <surface>
```

```
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
          </ode>
        </friction>
        <bounce/>
        <contact/>
      </surface>
    </collision>
    <visual name='lrf2_visual'>
      <pose>0 0 0 0 0 0</pose>
      <geometry>
        <mesh>
          <scale>1 1 1</scale>
          <uri>model://description/meshes/lrf2.STL</uri>
        </mesh>
      </geometry>
      <material>
        <diffuse>0.995098054 1 1 1</diffuse>
        <ambient>0.995098054 1 1 1</ambient>
      </material>
    </visual>
  </link>
  <joint name='jrf3' type='revolute'>
    <pose relative_to='lrf2'>-0.00044999999999999999 -5.9766999999999999e-05 -0
.050127999999999999 1.9198999999999999 0 0</pose>
    <parent>lrf2</parent>
    <child>lrf3</child>
    <axis>
      <xyz>1 0 0</xyz>
      <limit>
        <lower>-3.14</lower>
        <upper>3.14</upper>
        <effort>10</effort>
        <velocity>10</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
      </dynamics>
    </axis>
  </joint>
  <link name='lrf3'>
```

```
    <pose relative_to='jrf3'>0 0 0 0 0 0</pose>
    <inertial>
     <pose>0.0005223984142396780 2 -0.000141459359524615 -0.00792975909639
33497 0 0 0</pose>
     <mass>0.0099422546587472193</mass>
     <inertia>
      <ixx>1.9963740226929001e-06</ixx>
      <ixy>4.0329498722230403e-09</ixy>
      <ixz>4.3734180051889302e-08</ixz>
      <iyy>2.28648720039974e-06</iyy>
      <iyz>1.3881003758252501e-09</iyz>
      <izz>6.0776469273260595e-07</izz>
     </inertia>
    </inertial>
    <collision name='lrf3_collision'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/lrf3.STL</uri>
      </mesh>
     </geometry>
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
         <mu2>1</mu2>
        </ode>
       </friction>
       <bounce/>
       <contact/>
      </surface>
    </collision>
    <visual name='lrf3_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/lrf3.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
```

```
      </material>
    </visual>
  </link>
  <joint name='jrr1' type='revolute'>
    <pose relative_to='body'>0.04 -0.04 0.04 -1.5708000000000002 2.7755575615628
914e-17 -2.6179999999999999</pose>
    <parent>body</parent>
    <child>lrr1</child>
    <axis>
      <xyz>0 1 0</xyz>
      <limit>
        <lower>-3.14</lower>
        <upper>3.14</upper>
        <effort>10</effort>
        <velocity>10</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
      </dynamics>
    </axis>
  </joint>
  <link name='lrr1'>
    <pose relative_to='jrr1'>0 0 0 0 0 0</pose>
    <inertial>
      <pose>0.00286542457 -0.001117447 0.027575894 0 0 0</pose>
      <mass>0.012789727077659701</mass>
      <inertia>
        <ixx>2.80852259219697e-06</ixx>
        <ixy>-1.22117858410542e-08</ixy>
        <ixz>8.7286419553580596e-08</ixz>
        <iyy>1.94792750477442e-06</iyy>
        <iyz>2.2832067239878298e-08</iyz>
        <izz>1.8524299282579899e-06</izz>
      </inertia>
    </inertial>
    <collision name='lrr1_collision'>
      <pose>0 0 0 0 0 0</pose>
      <geometry>
        <mesh>
          <scale>1 1 1</scale>
          <uri>model://description/meshes/lrr1.STL</uri>
        </mesh>
      </geometry>
```

```
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
         <mu2>1</mu2>
        </ode>
       </friction>
       <bounce/>
       <contact/>
      </surface>
    </collision>
    <visual name='lrr1_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/lrr1.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
   </link>
   <joint name='jrr2' type='revolute'>
    <pose relative_to='lrr1'>0.0020500000000000002 -0.0058999999999999999 0.03
32 -2.7925 0 0</pose>
    <parent>lrr1</parent>
    <child>lrr2</child>
    <axis>
     <xyz>1 0 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>10</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
```

```
  <link name='lrr2'>
   <pose relative_to='jrr2'>0 0 0 0 0 0</pose>
   <inertial>
    <pose>6.9388939039072299e-18 -6.9388939039072299e-18 -0.024259718544271999 0 0 0</pose>
    <mass>0.0071256681620003896</mass>
    <inertia>
     <ixx>1.51184436411063e-06</ixx>
     <ixy>-6.3527471044072497e-22</ixy>
     <ixz>-1.2705494208814499e-21</ixz>
     <iyy>2.8950464625635899e-06</iyy>
     <iyz>2.64697796016969e-22</iyz>
     <izz>1.50597424566293e-06</izz>
    </inertia>
   </inertial>
   <collision name='lrr2_collision'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/lrr2.STL</uri>
     </mesh>
    </geometry>
     <surface>
      <friction>
       <ode>
        <mu>1</mu>
        <mu2>1</mu2>
       </ode>
      </friction>
      <bounce/>
      <contact/>
     </surface>
   </collision>
   <visual name='lrr2_visual'>
    <pose>0 0 0 0 0 0</pose>
    <geometry>
     <mesh>
      <scale>1 1 1</scale>
      <uri>model://description/meshes/lrr2.STL</uri>
     </mesh>
    </geometry>
    <material>
     <diffuse>0.995098054 1 1 1</diffuse>
```

```
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
   </link>
   <joint name='jrr3' type='revolute'>
    <pose relative_to='lrr2'>0.00044999999999999999 5.9766999999999999e-05 -0.0
50127999999999999 1.9198999999999999 -4.2351647362715017e-22 3.14158530
71795872</pose>
    <parent>lrr2</parent>
    <child>lrr3</child>
    <axis>
     <xyz>1 0 0</xyz>
     <limit>
      <lower>-3.14</lower>
      <upper>3.14</upper>
      <effort>10</effort>
      <velocity>100</velocity>
     </limit>
     <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
     </dynamics>
    </axis>
   </joint>
   <link name='lrr3'>
    <pose relative_to='jrr3'>0 0 0 0 0 0</pose>
    <inertial>
     <pose>0.00052239841485524801 -0.00014145935954890099 -0.007929759095
75863 0 0 0</pose>
     <mass>0.0099422546595867908</mass>
     <inertia>
      <ixx>1.99637402276075e-06</ixx>
      <ixy>4.0329498745485196e-09</ixy>
      <ixz>4.3734179997038598e-08</ixz>
      <iyy>2.2864872005159102e-06</iyy>
      <iyz>1.38810037893004e-09</iyz>
      <izz>6.0776469278123602e-07</izz>
     </inertia>
    </inertial>
    <collision name='lrr3_collision'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
```

```
        <uri>model://description/meshes/lrr3.STL</uri>
      </mesh>
     </geometry>
      <surface>
       <friction>
        <ode>
         <mu>1</mu>
         <mu2>1</mu2>
        </ode>
       </friction>
       <bounce/>
       <contact/>
      </surface>
    </collision>
    <visual name='lrr3_visual'>
     <pose>0 0 0 0 0 0</pose>
     <geometry>
      <mesh>
       <scale>1 1 1</scale>
       <uri>model://description/meshes/lrr3.STL</uri>
      </mesh>
     </geometry>
     <material>
      <diffuse>0.995098054 1 1 1</diffuse>
      <ambient>0.995098054 1 1 1</ambient>
     </material>
    </visual>
  </link>
<joint name='fit1' type='revolute'>
<pose relative_to='llf3'>0 0 0.06 0 0 0</pose>
<parent>llf3</parent>
<child>fitl1</child>
<axis>
  <xyz>1 0 0</xyz>
  <limit>
    <lower>-3.14</lower>
    <upper>3.14</upper>
    <effort>100</effort>
    <velocity>1</velocity>
  </limit>
  <dynamics>
    <spring_reference>0</spring_reference>
    <spring_stiffness>0</spring_stiffness>
  </dynamics>
```

```
 </axis>
</joint>
<link name='fitl1'>
 <pose relative_to='fit1'>0 0 0 0 0 0</pose>
 <inertial>
  <pose>0 0 0 0 0 0</pose>
  <mass>0.01</mass>
  <inertia>
   <ixx>2.083e-06</ixx>
   <ixy>0.0</ixy>
   <ixz>0.0</ixz>
   <iyy>2.083e-06</iyy>
   <iyz>0.0</iyz>
   <izz>2.083e-06</izz>
  </inertia>
 </inertial>
 <collision name='fit1_collision'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <surface>
   <friction>
    <ode>
     <mu>1</mu>
     <mu2>1</mu2>
    </ode>
   </friction>
   <bounce/>
   <contact/>
  </surface>
 </collision>
 <visual name='fit1_visual'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <material>
   <diffuse>0.8 0.8 0.8 1</diffuse>
   <ambient>0.8 0.8 0.8 1</ambient>
```

```
  </material>
 </visual>
</link>
<joint name='fit2' type='revolute'>
 <pose relative_to='lrf3'>0 0 -0.06 0 0 0</pose>
 <parent>lrf3</parent>
 <child>fitl2</child>
 <axis>
  <xyz>1 0 0</xyz>
  <limit>
   <lower>-3.14</lower>
   <upper>3.14</upper>
   <effort>100</effort>
   <velocity>1</velocity>
  </limit>
  <dynamics>
   <spring_reference>0</spring_reference>
   <spring_stiffness>0</spring_stiffness>
  </dynamics>
 </axis>
</joint>
<link name='fitl2'>
 <pose relative_to='fit2'>0 0 0 0 0 0</pose>
 <inertial>
  <pose>0 0 0 0 0 0</pose>
  <mass>0.01</mass>
  <inertia>
   <ixx>2.083e-06</ixx>
   <ixy>0.0</ixy>
   <ixz>0.0</ixz>
   <iyy>2.083e-06</iyy>
   <iyz>0.0</iyz>
   <izz>2.083e-06</izz>
  </inertia>
 </inertial>
 <collision name='fit2_collision'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <surface>
   <friction>
```

```
      <ode>
        <mu>1</mu>
        <mu2>1</mu2>
      </ode>
    </friction>
    <bounce/>
    <contact/>
  </surface>
 </collision>
 <visual name='fit2_visual'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <material>
   <diffuse>0.8 0.8 0.8 1</diffuse>
   <ambient>0.8 0.8 0.8 1</ambient>
  </material>
 </visual>
</link>
<joint name='fit3' type='revolute'>
 <pose relative_to='llr3'>0 0 0.06 0 0 0</pose>
 <parent>llr3</parent>
 <child>fitl3</child>
 <axis>
  <xyz>1 0 0</xyz>
  <limit>
   <lower>-3.14</lower>
   <upper>3.14</upper>
   <effort>100</effort>
   <velocity>1</velocity>
  </limit>
  <dynamics>
   <spring_reference>0</spring_reference>
   <spring_stiffness>0</spring_stiffness>
  </dynamics>
 </axis>
</joint>
<link name='fitl3'>
 <pose relative_to='fit3'>0 0 0 0 0 0</pose>
 <inertial>
  <pose>0 0 0 0 0 0</pose>
```

```
  <mass>0.01</mass>
  <inertia>
   <ixx>2.083e-06</ixx>
   <ixy>0.0</ixy>
   <ixz>0.0</ixz>
   <iyy>2.083e-06</iyy>
   <iyz>0.0</iyz>
   <izz>2.083e-06</izz>
  </inertia>
 </inertial>
 <collision name='fit3_collision'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <surface>
   <friction>
    <ode>
     <mu>1</mu>
     <mu2>1</mu2>
    </ode>
   </friction>
   <bounce/>
   <contact/>
  </surface>
 </collision>
 <visual name='fit3_visual'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <material>
   <diffuse>0.8 0.8 0.8 1</diffuse>
   <ambient>0.8 0.8 0.8 1</ambient>
  </material>
 </visual>
</link>
<joint name='fit4' type='revolute'>
 <pose relative_to='lrr3'>0 0 -0.06 0 0 0</pose>
 <parent>lrr3</parent>
```

```
 <child>fitl4</child>
 <axis>
  <xyz>1 0 0</xyz>
  <limit>
   <lower>-3.14</lower>
   <upper>3.14</upper>
   <effort>100</effort>
   <velocity>1</velocity>
  </limit>
  <dynamics>
   <spring_reference>0</spring_reference>
   <spring_stiffness>0</spring_stiffness>
  </dynamics>
 </axis>
</joint>
<link name='fitl4'>
 <pose relative_to='fit4'>0 0 0 0 0 0</pose>
 <inertial>
  <pose>0 0 0 0 0 0</pose>
  <mass>0.01</mass>
  <inertia>
   <ixx>2.083e-06</ixx>
   <ixy>0.0</ixy>
   <ixz>0.0</ixz>
   <iyy>2.083e-06</iyy>
   <iyz>0.0</iyz>
   <izz>2.083e-06</izz>
  </inertia>
 </inertial>
 <collision name='fit4_collision'>
  <pose>0 0 0 0 0 0</pose>
  <geometry>
   <box>
    <size>0.030 0.030 0.005</size>
   </box>
  </geometry>
  <surface>
   <friction>
    <ode>
     <mu>1</mu>
     <mu2>1</mu2>
    </ode>
   </friction>
   <bounce/>
```

```
    <contact/>
   </surface>
  </collision>
  <visual name='fit4_visual'>
   <pose>0 0 0 0 0 0</pose>
   <geometry>
    <box>
     <size>0.030 0.030 0.005</size>
    </box>
   </geometry>
   <material>
    <diffuse>0.8 0.8 0.8 1</diffuse>
    <ambient>0.8 0.8 0.8 1</ambient>
   </material>
  </visual>
 </link>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlf1</joint_name>
   </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlf2</joint_name>
   </plugin>
     <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlf3</joint_name>
    </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jrf1</joint_name>
   </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jrf2</joint_name>
   </plugin>
     <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
```

```
    <joint_name>jrf3</joint_name>
    </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlr1</joint_name>
   </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlr2</joint_name>
   </plugin>
     <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jlr3</joint_name>
    </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jrr1</joint_name>
   </plugin>
   <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jrr2</joint_name>
   </plugin>
     <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <joint_name>jrr3</joint_name>
    </plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlf1</joint_name>
 <topic>jlf1_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
```

```
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlf2</joint_name>
 <topic>jlf2_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlf3</joint_name>
 <topic>jlf3_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlr1</joint_name>
 <topic>jlr1_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlr2</joint_name>
```

```
 <topic>jlr2_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jlr3</joint_name>
 <topic>jlr3_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrf1</joint_name>
 <topic>jrf1_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrf2</joint_name>
 <topic>jrf2_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
```

```
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrf3</joint_name>
 <topic>jrf3_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrr1</joint_name>
 <topic>jrr1_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrr2</joint_name>
 <topic>jrr2_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
<plugin
```

```
 filename="gz-sim-joint-position-controller-system"
 name="gz::sim::systems::JointPositionController">
 <joint_name>jrr3</joint_name>
 <topic>jrr3_topic</topic>
 <p_gain>1</p_gain>
 <i_gain>0.1</i_gain>
 <d_gain>0.01</d_gain>
 <i_max>1</i_max>
 <i_min>-1</i_min>
 <cmd_max>1000</cmd_max>
 <cmd_min>-1000</cmd_min>
</plugin>
 </model>
</sdf>
```