SIMBOT Documentation

Muhammad Anis

July 2025

1 Introduction

SIMBOT is a simulation of the ABB Flexley Tug, an Autonomous Mobile Robot (AMR) developed by ABB, primarily used in warehouse logistics. The robot is equipped with multiple cameras and two lidars, one at the front and one at the rear,which enable visual SLAM and obstacle detection for safe navigation.

The goal of this project is to create a Gazebo based simulation of the Flexley Tug, replicating its real world sensor setup. This simulated environment serves as a platform for research into the robot's perception and navigation capabilities.For the simulation of this robot, ROS Humble is used alongside Ignition Gazebo Fortress for simulation and RViz2 for visualization.

2 ROS Package Overview

The ROS package for SIMBOT consists of the following directories and subdirectories.

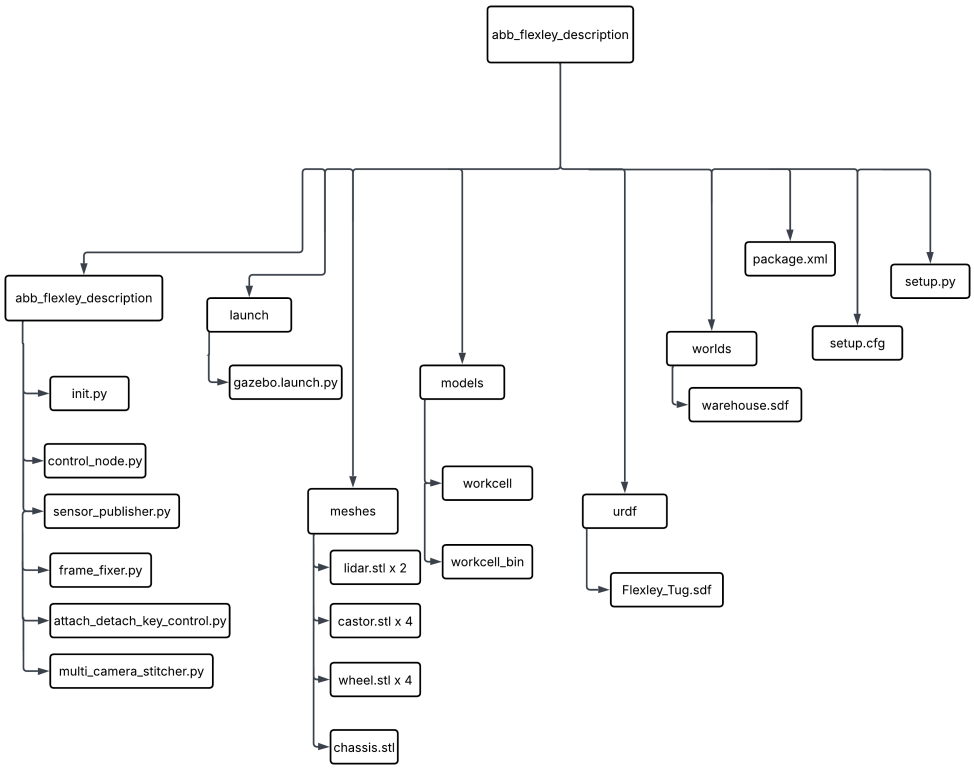

Figure 1: Package Overview

2.1 abb flexley description

This directory contains the nodes responsible for the control and sensor operations of the robot. All the nodes for this package are in python. The control node enables teleoperation of the robot, allowing it to be manually controlled through keyboard.The sensor publisher node publishes data from the sensors topics in ignition, such as lidar and cameras, to relevant ROS topics for further processing and visualization. The lidar frame merger node is responsible for merging the output of both the lidars(front & back) in to a single stream for visualization in rViz. The multi camera stitcher node is responsible for providing a unified view of the six cameras in rViz. The attach detach key control node is responsible for dynamically attaching and detaching the trolley and robot during simulation. The detailed description of each node is provided in section 5.

2.2 launch

This directory contains the launch file for the robot. The gazebo launch file can be used to run the simulation in ignition gazebo. The launch file for this package is written in python and is responsible for launching the gazebo, spawning the robot and also creating a bridge between gazebo and ROS2. The ros-gazebo bridge is responsible for publishing the output from gazebo topics to ros topics.

2.3 meshes

This directory contains all the 3D model files in STL format, which are referenced by the SDF file to construct the robot's visual and collision geometry in both Gazebo and RViz environments. These mesh files represent physical components such as wheels, lidar units, and chassis.

2.4 models

The sub-directory "models" contain the necessary models for developing the simulated world environment in gazebo. It contains three models workcell, workcell bin and trolley. The workcell and workcell bin models are being used in the world file to create warehouse environment, while trolley model is used to spawn the trolley inside the warehouse world which can then be attached and detached with the robot as per requirement.

2.5 urdf

This directory contains SDF file that is being used for the simulation of the robot in Gazebo. SDF file is an XML format file which defines visual, inertial and collision properties for robot's chassis, wheels, sensors and also defines the necessary plugins. The detailed description of SDF is provided in section 3.

2.6 worlds

This directory contains two world files, empty world is just an empty gazebo world environment, while warehouse world is the gazebo environment replicating a warehouse environment inside gazebo.

3 SDF Explanation

For simulation of a robot in the gazebo, the robot is modeled using an sdf file. SDF is used instead of URDF because it is a more comprehensive format, as it supports the plugins for simulation in gazebo too. It can also define the surrounding environment along with the robot description. The sdf for this model is structured as shown in the figure below:

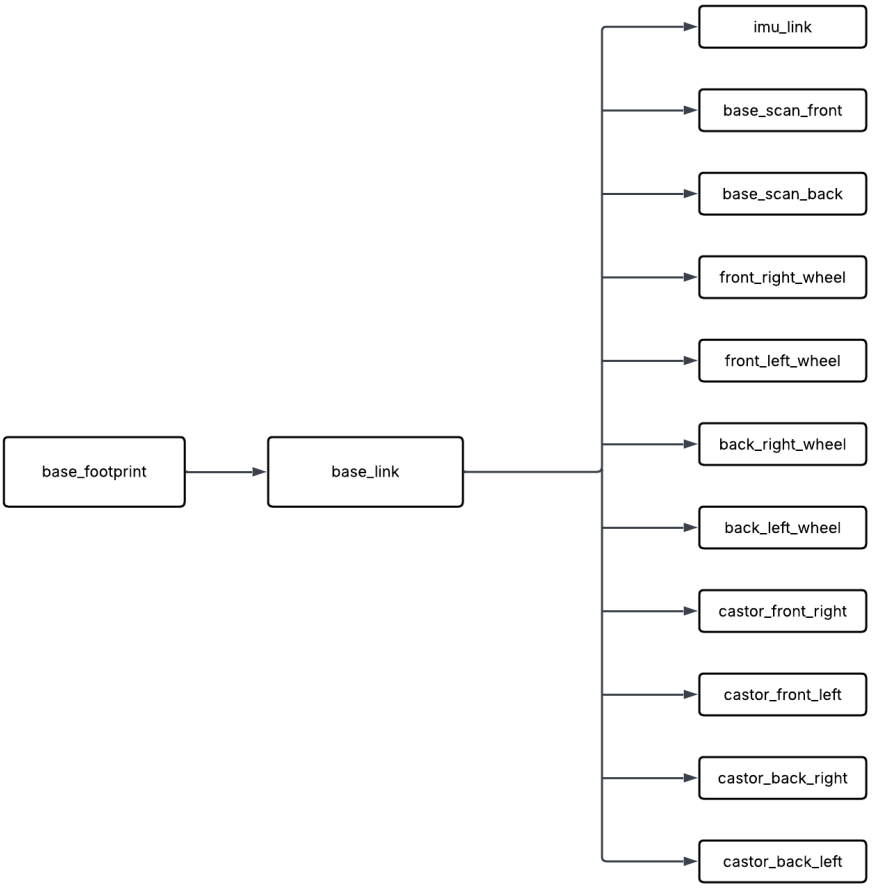

Figure 2: SDF Structure

The `Flexley_Tug.sdf` file can be found at the following path inside the package:
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

3.1 Base Link

The base footprint serves as the canonical link of the robot, to which the implicit frame of the model is attached. It acts as a reference or shadow frame for the entire robot. The base link, which represents the robot's chassis, is connected to this frame via a fixed joint located 0.09 m above the origin along the z-axis. The base link defines the robot's chassis and includes inertial, visual, and collision properties. These properties are based on the original physical model of the robot:

- Inertial: The mass of the chassis is taken as 100 Kg same as the original robot, and the inertial tensor is calculated based on this mass and original dimensions.
- Collision: Modeled as a box with dimensions 2.1 m (length) × 0.5 m (width) × 0.23 m (height). These dimensions are identical to those of the actual robot.
- Visual: Uses the original STL file of the chassis, scaled by 0.001 to convert millimeter units to meters.

Each property tag includes a pose element that defines the location of the link relative to the origin. The base link is positioned at 0.09 m along the z-axis. Additionally, the base link also defines six cameras (one at front, one at back, two at each side) identical to the original robot to provide real-time environmental visualization. Each camera publishes to a separate topic, while their combined output can be visualized using multi camera stitcher node which is explained in section 5. The complete base link implementation is shown in appendix A.1 Base Link, the link shown can be found in `Flexley_Tug.sdf` file and the path for this file is mentioned above. The link can be modified as per requirement.

3.2 IMU Link

The IMU link defines the frame to which the Inertial Measurement Unit (IMU) is attached. It is connected to the base link using a fixed joint named the imu joint, located 0.06 meters above the chassis along the z-axis. The properties of the IMU sensor are defined inside the imu link. The sensor is configured at 200 HZ, which is a common frequency for IMU sensors in robotics. Generally, a higher update rate would offer more accuracy and vise versa. The sensor data is published on the imu scan topic in gazebo which can then be bridged to ROS.

The IMU sensor measures both angular velocity and linear acceleration. To simulate real world sensor behavior, Gaussian noise is added to each axis:

- Angular velocity noise has a standard deviation of 0.0002 rad/s. This low value reflects a high-quality gyroscope with minimal drift, suitable for stable orientation tracking over time.
- Linear acceleration noise has a higher standard deviation of 0.0017 m/s^2. This is typical for accelerometers, which tend to have slightly more variability. The added noise helps mimic the uncertainty seen in real IMU readings, which is important for testing sensor fusion algorithms like an Extended Kalman Filter (EKF).

The link is attached to the base link using at fixed joint at 0.06 meters in z-axis. For modification, the link can be found in `Flexley_Tug.sdf` file. The complete source code for IMU link is available in Appendix A.2.

3.3 Lidar Links (Base Scan Front and Base Scan Back)

The robot features two identical 2D lidar sensors, one at the front (base scan front) and one at the back (base scan back) for obstacle detection and environmental awareness. Both sensors are modeled as cylindrical links with a mass of 1.17 kg and appropriate inertial properties for realistic simulation. The collision geometry is defined as a simple cylinder (0.05 m radius, 0.05 m length), while the visual representation uses the original STL mesh scaled to 0.001 to have units in meters.

Each lidar is implemented as a gpu lidar sensor with:

- 720 horizontal samples covering approximately 160° field of view (±1.396 rad)
- 2D operation using a single vertical scan line
- A detection range from 0.08 m to 10.0 m with 0.01 m resolution
- Real-time visualization enabled in the simulation

The front lidar is positioned 1.0 meters in front of the robot's base and rotated 3.14 radians around the yaw axis to face forward. Its sensor element is placed slightly above the base at 0.22 m height and publishes data at 5 Hz on the lidar front topic. The back lidar is mounted 0.9 meters behind the base, at 0.04 m height, with the sensor head raised to 0.22 m. It shares the same orientation (yaw 3.14) as the front lidar to maintain a consistent scanning direction and operates at a frequency of 5 Hz, publishing on the lidar back topic. Both sensors are fixed to the base using fixed joints, ensuring that they remain rigidly attached to the robot frame throughout the operation. The complete implementation of front and back lidars can be found in appendix A.3 and A.4 respectively. Both `base_scan_front` & `base_scan_back` links can be found inside `Flexley_Tug.sdf`, where they can be modified as per requirement.

3.4 Wheels Link

The robot is equipped with four rotating wheels,two at the front and two at the rear,connected to the base via revolute joints that allow rotation about the y-axis. Each wheel is modeled with appropriate physical and visual characteristics. For inertial properties, each wheel has a mass of 0.765 kg, and the inertial tensor is defined based on the cylindrical shape and approximate mass distribution, enabling realistic dynamic simulation. For collision properties, each wheel is modeled as a cylinder with a radius of 0.075 m and a length of 0.05 m. This simplification ensures computational efficiency while maintaining physical accuracy. Friction parameters (mu = 0.90, mu2 = 0.25) define the grip between the wheels and the ground, helping simulate slippage or traction during motion.

Each wheel uses an STL mesh for accurate visual appearance, scaled to meters (0.001). A green material is applied to make the wheels easily visible in Gazebo.

Each wheel is connected to the base link using a revolute joint, allowing it to rotate freely around the y-axis (x=0, y=1, z= 0 in axis). The joint is configured with:

- Effort limit of 10000.0
- Velocity limit of 100.0
- Damping of 0.5 for smooth rotation
- Unlimited rotation

The joint pose matches the wheel's physical location on the robot:

- Front wheels are placed at x = +0.580 m, with y = ±0.200 m
- Back wheels are placed at x = -0.580 m, with y = ±0.200 m
- All wheels are mounted 0.02 m below the base frame on the z-axis

Each wheel is assigned a unique name but shares the same basic structure and behavior, ensuring consistency across the drivetrain. For complete implementation, please refer to appendix A.5.

3.5 Castor Wheel

The model includes four castor wheels: two at the front and two at the back, supporting free movement of the robot base. Each castor wheel is connected to the base link using a ball joint, allowing it to swivel freely in any direction. This setup is typical for passive omnidirectional support.

- Inertial properties: Each wheel has a mass of 0.103 kg, and an inertia tensor approximated for a small sphere.
- Collision model: Defined as a sphere with radius 0.025 m to simulate physical contact and dynamics.
- Visual representation: Uses a scaled STL mesh for visual realism.
- Joint type: Ball joint to enable 360° rotation in pitch and yaw.

All the castor links can be found in appendix A.6. The link can be found inside `Flexley_Tug.sdf` for any modification.

3.6 Gazebo Plugins

Three Gazebo plugins are used to simulate the robot. The differential drive plugin models the robot's motion using a differential drive system, allowing control via velocity commands. It drives two set of wheels on each side, with a separation of 0.4 m and a wheel radius of 0.075 m. The plugin subscribes to the cmd vel topic for velocity input and publishes estimated position and velocity to the odom topic.

The joint state publisher plugin continuously broadcasts the real-time position and velocity of specified joints to the joint states topic at 30 Hz. This data is essential for bridging the sensor data between gazebo and ROS and which is then used for visualization in Rviz.

The detachable joint plugin is also defined, which is responsible for dynamically attaching and detaching the trolley with the robot. The attachment and detachment is managed using a node named as `attach_detach_key_control` which is explained in section 5. This plugin defines a parent link which in this case is "`base_link`" of the robot and also defines a child link which is the base of the base of the trolley. These two links can be attached and detached using a keyboard key when the node is running.

All these plugins are defined inside `Flexley_Tug.sdf` for any modification and are shown in appendix A.7.

4 Package Setup

To use this simulation package, a system running Ubuntu 22.04 is required. Before setting up the package, ensure that ROS 2 Humble and Ignition Gazebo Fortress are installed on your PC. If ROS 2 Humble has not yet been installed, please follow the instructions on the following link to install it:

```
https://docs.ros.org/en/humble/Installation/Ubuntu-
Install-Debs.html
```

For installation of Ignition Gazebo Fortress, Follow the instructions in the following link:

```
https://gazebosim.org/docs/fortress/install/
```

Once both ROS Humble and Ignition Gazebo are installed in your PC, you can get started with the package. Open terminal in your PC and write the following command to make a directory and navigate to that directory.

```
mkdir -p ~/ros2_ws
cd ros2_ws
```

Once inside the directory, clone the package using the following command.

```
https://github.com/autonomymobilitylab/SimBot.git
```

Once you have cloned the package in your PC, you'll see a folder named SimBot. Navigate to the folder containing the src files.

```
cd SimBot
```

Once inside this folder, source the environment in the terminal and then use the following build command:

```
source /opt/ros/humble/setup.bash
colcon build --packages-select abb_flexley_description
```

Once the package is built, you can have visulaization in Rviz or simulation in Gazebo. To view the model in Rviz, source the environment and run the following command:

```
source install/setup.bash
ros2 launch abb_Flexley_description display.launch.py
```

This will launch the model of the robot in Rviz. For simulation of robot in gazebo, first make sure to change the IP of the gazebo to local IP using following command:

```
export IGN_IP=127.0.0.1
```

Once you have set the IP you can launch the simulation using the following command.

```
ros2 launch abb_Flexley_description gazebo.launch.py
```

After launching the simulation in gazebo you can control the robot by launching the control node, open a new terminal, source the environment and use the following command:

```
source install/setup.bash
ros2 run abb_flexley_description control_node
```

The robot can be controlled using the keys W,A,S,D for forward, backward and sideways movement. For stopping the robot X can be used. Similarly, for viewing the output of the sensors open a new termianl, source the environment and use the following commands:

```
source install/setup.bash
ros2 run abb_flexley_description publisher_node
```

Once you have launched both the nodes, you can control the robot using the keys and see the robot moving in gazebo and the output of the sensor in the terminal. For attachment and detachment of the robot, you can launch the attach detach key control node, this node can be launched using following command.

```
ros2 run abb_flexley_description attach_detach_key_control
```

Once the node is launched, you can attach the robot with the trolley using 't' key on the keyboard and detach it by pressing 'r'. The attachment only works once the robot is close to the trolley. For visualization of sensor's data in RViz, you need to launch the lidar frame merger and multi camera stitcher node to view unified response of the lidars and cameras in RViz. The nodes can be launched using following commands.

```
ros2 run abb_flexley_description lidar_frame_merger
ros2 run abb_flexley_description multi_camera_stitcher
```

5 Nodes

For smooth and efficient operation of this robot, certain ROS2 nodes are defined. These nodes are explained in the subsections below. These nodes are written in python and can be found at the following path inside the package for any modification:
`src>abb_flexley_description>abb_flexley_description>example_node.py`

5.1 Control Node

The control node allows the robot to be operated using keyboard inputs while also preventing collisions with nearby obstacles. It contains a class named Controller, which handles movement commands based on user input and monitors front and rear Lidar data to detect obstacles. The node subscribes to the lidar front and lidar back topics to receive sensor data and publishes velocity commands to the cmd vel topic. Default values for linear and angular velocities are defined within the class but can be modified as needed.

The get key function, defined outside the class, reads the currently pressed key in a non-blocking manner. The keyboard listener function continuously listens for key presses and updates the control state. The control loop function runs at regular intervals to check which keys are pressed and whether any obstacles are detected. Based on this, it sends the appropriate velocity commands to the robot. The robot moves forward, backward, left, or right in response to the 'w', 's', 'a', and 'd' keys respectively. Pressing 'x' or the spacebar stops the robot, and pressing 'q' cleanly shuts down the node. The complete pyhton code for control node is available in appendix A.8.

5.2 Sensor Publisher Node

The sensor publisher node receives the data from gazebo and publishes the sensor data. This node creates three publishers to publish IMU data to imu scan topic, front lidar data on lidar front topic and back lidar data on lidar back topic. The publish data function, the node generates synthetic sensor messages: the IMU message includes simulated linear acceleration and angular velocity with some random noise to mimic real sensor readings. For both front and back lidar, it publishes LaserScan messages with predefined angle ranges and random distance measurements within a specified range. These messages include metadata like timestamps and frame identifiers for correct interpretation. This data can also be sent to ROS2 using a ros gz bridge but that is being done using launch file. The complete implementation of sensor publisher node is available in appendix A.9.

5.3 Lidar Frame Merger Node

The Lidar Frame Merger node synchronizes and combines Lidar data from two separate lidars,front and back, into a unified 360° scan. It ensures that all published scans share a consistent reference frame, simplifying downstream processing such as mapping or obstacle detection.

The node is implemented through a class named LidarFrameMerger, which inherits from the ROS 2 Node class. It subscribes to two input topics, `lidar_front` and `lidar_back`, each providing LaserScan messages from the front and rear Lidar sensors. For each incoming message, the node creates a copy of the scan data with a fixed frame id (`base_link`) to standardize the coordinate reference. These "fixed" messages are then published to `lidar_front_fixed` and

lidar_back_fixed topics for visualization or debugging.

Once both front and back scans are received, the node attempts to merge them into a single LaserScan message representing a complete 360° view of the surroundings. The merging process involves:

- Using consistent timing and frame information (base_link_frame).
- Defining the merged scan's angular range from to radians, covering the full circle.
- Combining the range and intensity data from both sensors into one continuous dataset.

The resulting merged scan is then published to the /scan_topic, which can then be utilized for visualization in rViz.

Overall, this node acts as a Lidar data fusion module, harmonizing multi-sensor inputs into a single, consistent scan topic to support robust robot perception and environment awareness.

5.4 Multi Camera Stitcher Node

The Multi Camera Stitcher node combines live image feeds from six different cameras into a single panoramic 360° view. It synchronizes all camera streams, processes them into OpenCV format, stitches them together, and publishes the result as a unified image and camera information message.

The node is implemented through a class named Multi Camera Stitcher, which inherits from the ROS 2 Node class. It subscribes to six image topics corresponding to cameras positioned around the robot:

- camera_front
- camera_back
- camera_front_right
- camera_back_right
- camera_front_left
- camera_back_left

To ensure that all camera frames correspond to the same moment in time, the node uses an ApproximateTimeSynchronizer, which aligns incoming image messages based on their timestamps with a small tolerance (slop). Once all six images are synchronized, they are passed to the callback function for processing.

Inside the callback:

- Each incoming ROS Image message is converted to an OpenCV image using the CvBridge library.

- The images are resized to a common resolution to maintain consistency.
- The images are then concatenated horizontally (cv2.hconcat) to form a single panoramic strip representing a full 360° view around the robot.
- A new ROS Image message is created from the merged image and assigned the `base_link` frame for proper visualization in tools like RViz.
- A corresponding CameraInfo message is also generated, containing image dimensions, intrinsic camera parameters, and distortion coefficients.
- Finally, the node publishes the stitched 360° image on the `/camera_360/image_raw` topic and the camera calibration information on the `/camera_360/camera_info` topic.

Overall, this node functions as a multi-camera fusion and visualization module, creating a real-time panoramic image that can be used for perception, navigation, or situational awareness in robotic applications.

5.5 Attach Detach Key Control Node

The Attach Detach Node allows a user to manually control the attachment and detachment of two simulated models (in this case, robot and trolley) in Gazebo Ignition, using simple keyboard commands. The node continuously monitors the distance between the two models and only allows attachment when they are within a predefined proximity threshold.

The node is implemented through a class named AttachDetachNode, which inherits from the ROS 2 Node class. It publishes empty messages to two topics:

- /attach — to trigger the attachment of the models.
- /detach — to release or separate them.

When started, the node prints a brief control guide:

- Press 't' to attach the models (only possible when they are close enough).
- Press 'r' to detach them.
- Press 'q' to quit the program safely.

To determine whether attachment is possible, the node periodically retrieves each model's pose . It extracts their positions and orientations, then calculates the XY plane distance between them. If the models remain within the proximity threshold (default: 0.55 m) for a sustained number of consecutive checks, the node marks them as close. Optionally, a yaw alignment tolerance can be defined to ensure the models are facing the same direction before attaching.

Two background threads run concurrently:

- A proximity loop continuously checks model positions every 0.2 seconds and updates whether they are near or far apart.

- A keyboard listener waits for user input and sends the appropriate ROS 2 messages (/attach or /detach) based on proximity and key presses.

If the models are too far apart, pressing 't' will display a warning instead of publishing an attach command.

Overall, this node acts as a manual attach/detach controller with proximity safety, providing a simple way to test or simulate docking mechanisms between two mobile models in a Gazebo environment.

6 Robot Simulation

The robot simulation can be launched by following the steps stated in package overview section. Once you launch the simulation using the launch command, the terminal reads the python file named "gazbo.launch.py" in the launch directory.

This launch file specifies the file paths for the model, world, and robot SDF files. It sets the necessary environment variables and launches the Gazebo simulation window using the ExecuteProcess action. During launch, the Gazebo world is loaded from the predefined world file path. After the world is running, the robot model is spawned using another ExecuteProcess command that references the robot's SDF file. The robot's initial position in the simulation can be adjusted by modifying the x, y, and z coordinate values in this launch file. Please refer to appendix A.10 for source code of launch file. Also, for any modification launch file can be found at the following path inside package.
`src>abb_flexley_description>launch>gazebo.launch.py`
Once both the world and robot are launched, all sensors defined in the robot's SDF file become active, publishing their data to Ignition Gazebo topics specific to each sensor. The launch file then sets up a bridge between Ignition Gazebo and ROS 2 using the ros gz bridge package. This bridge forwards sensor data including LiDAR, IMU, and camera data as well as command velocity, odometry, joint states, and robot state information from Gazebo to corresponding ROS 2 topics. Users can verify the data flow by subscribing to these ROS 2 topics using standard ROS 2 command-line tools.

```
ros2 topic list
```

Then to listen to any topic use the following command in the termianl. '

```
ros2 topic echo /topic_name
```

The data can also be visualized by launching rviz2 in a separate terminal and then adding the corresponding topic for lidar or camera.

7 Launch Example

This section provides an example of the complete simulation to help users better understand how it works. Once the simulation is launched by following the steps outlined in the Package Setup section, the following window will appear:

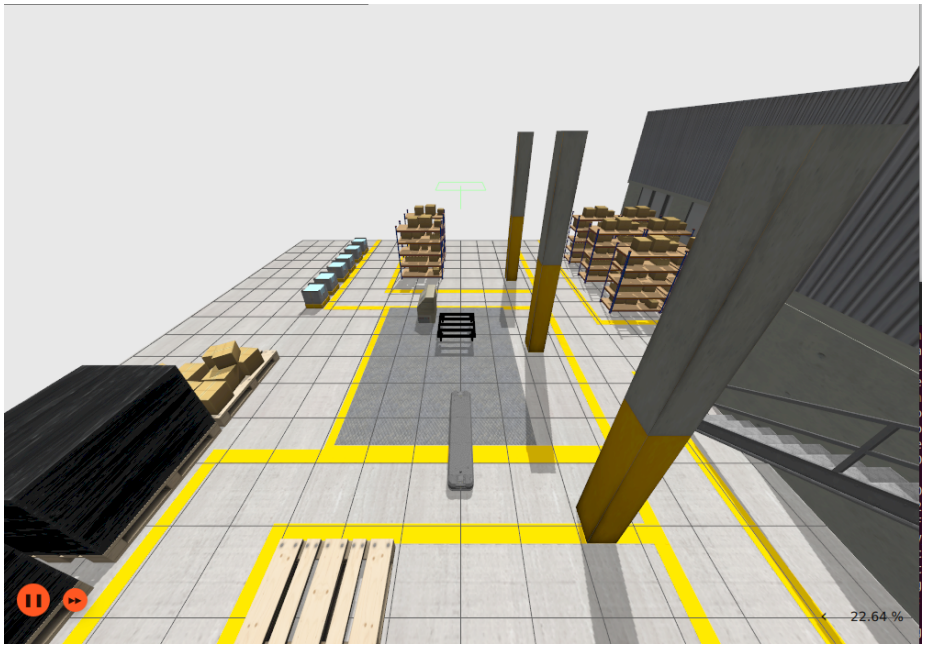

Figure 3: Gazebo Window

To view the camera feed and LiDAR data, use the search bar in the top-right corner to add Image Display and Visualize LiDAR panels. Once added, the image and LiDAR visualization will appear, and the window will look similar to the one shown below.

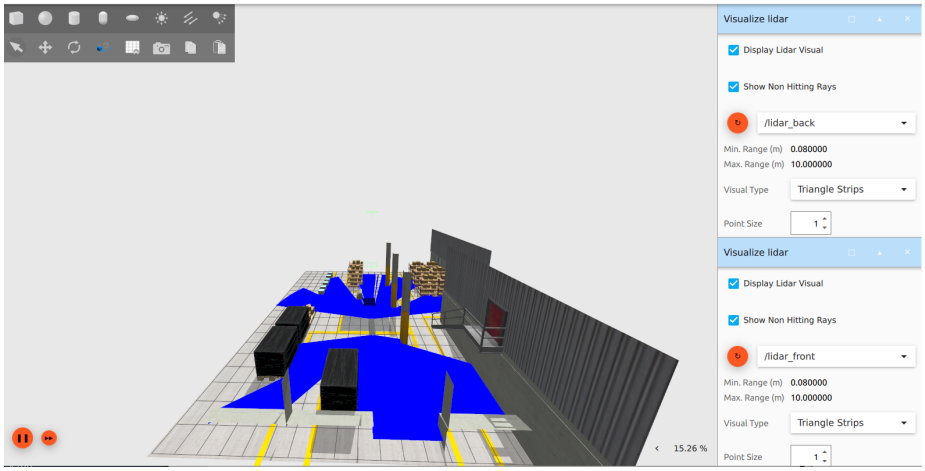

Figure 4: Camera and lidar View

The camera's view is visible at the side and the lidar rays can also be seen on the screen. Now, you can teleoperate the robot by launching the control node from the command specified in the package setup section. Once the node is running you can control the robot by using 'w' for forward. 's' for backward, 'a' for left and 'd' for right and 'spacebar' or 'x' to stop. To quit the node press 'q'. For visualization of data in rVIz, launch multi camera stitcher node and lidar frame merger node as explained in section 4 and then launch the rviz2 in the separate terminal using the following command.

```
ros2 launch abb_Flexley_description display.launch.py
```

The window will appear empty like this:

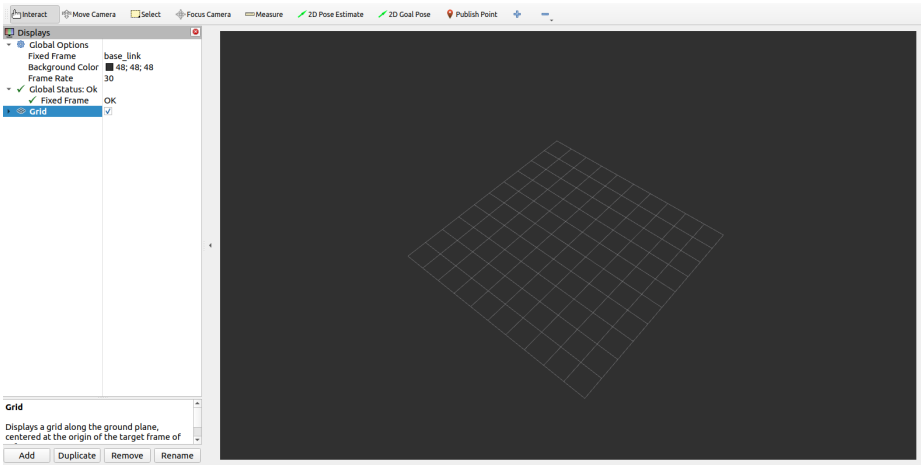

Figure 5: Rviz Window

Add the topic from the bottom left, and set the corresponding frame. For image display, add the camera topic from image raw under camera 360 and for lidars add the laserscan topic from scan and change the frame ids to corresponding frame ids. The response of the sensors will be visible in rviz.

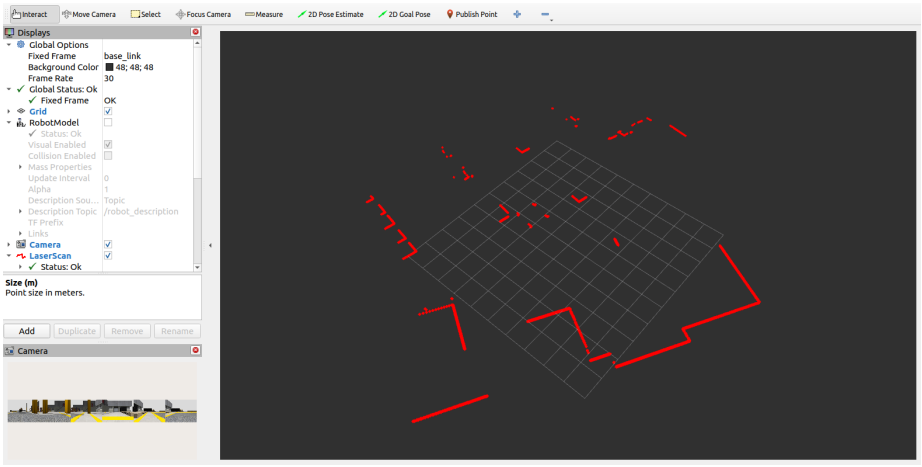

Figure 6: Sensor Data in rViz

You can also add the robot model to the rviz by adding the robot model from the bottom left corner and subscribing to robot description topic. The robot will be visible now in the window.

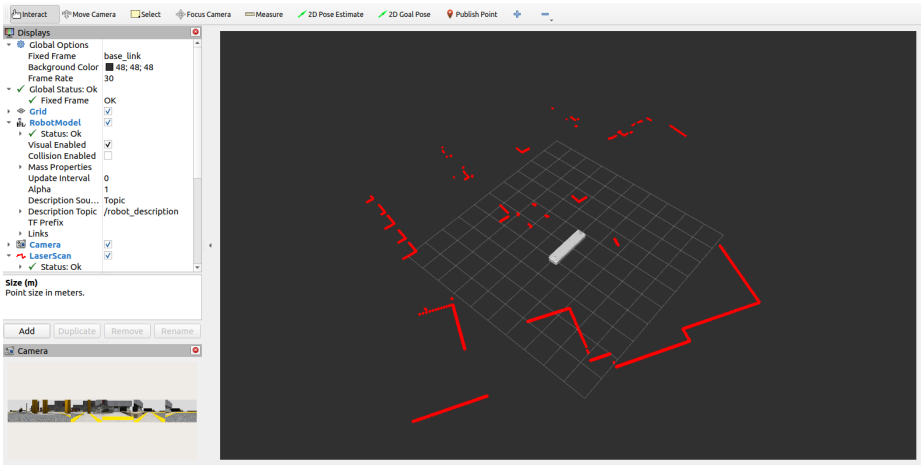

Figure 7: Robot Model Visualization

You can teleoperate the model in gazebo to see the corresponding results in rViz.

Appendix A: Source Codes

A.1 Base Link

The code refers to Base Link in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<sdf version='1.9'>
  <model name='Flexley_Tug'>
    <pose>0 0 0 0 0 0</pose>

    <link name="base_footprint"/>

    <!-- -Base Link -->
    <link name='base_link'>
      <inertial>
        <pose>0 0 0.09 0 0 0</pose> <!-- x=0, y=0 , z=0.09m, r=0, p=0,
        ↪  y=0 -->
        <mass>100</mass> <!-- mass of chassis -->
        <inertia>
            <ixx>2.0833</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>36.75</iyy>
            <iyz>0</iyz>
            <izz>38.83</izz>
        </inertia>
      </inertial>

      <collision name='base_collision'>
        <pose>0 0 0.09 0 0 0</pose> <!-- x=0, y=0 , z=0.09m, r=0, p=0,
        ↪  y=0 -->
        <geometry>
          <box>
            <size>2.1 0.5 0.23</size> <!-- Box 2.1m length, 0.5m width,
            ↪  0.23m height -->
          </box>
        </geometry>
      </collision>

      <visual name='base_link_visual'>
        <pose>0 0 0.09 0 0 0</pose> <!-- x=0, y=0 , z=0.09m, r=0, p=0,
        ↪  y=0 -->
        <material>
         <ambient>0.2 0.2 0.2 1</ambient>
         <diffuse>0.2 0.2 0.2 1</diffuse>
        </material>
        <geometry>
```

```
      <mesh>
        <scale>0.001 0.001 0.001</scale> <!-- scale 0.001 for units
        ↪  in meters -->
        <uri>../meshes/chassis.stl</uri>
      </mesh>
    </geometry>
  </visual>

  <sensor name="front_camera" type="camera">
   <pose>1.0 0 0.39 0 0 0</pose>  <!-- x=1m, y=0 , z=0.39m, r=0,
   ↪  p=0, y=0 -->
   <always_on>1</always_on> <!-- Keeps camera always on -->
   <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
   <visualize>true</visualize>
   <topic>front_cam</topic> <!-- ign topic -->
   <camera>     <!-- Camera Properties -->
      <horizontal_fov>2.0944</horizontal_fov>
      <image>
        <width>640</width>
        <height>480</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.05</near>
        <far>50</far>
        </clip>
   </camera>
  </sensor>

  <sensor name="back_camera" type="camera">
   <pose>-1.0 0 0.39 0 0 3.14</pose>  <!-- x=-1m, y=0 , z=0.39m,
   ↪  r=0, p=0, y=0 -->
   <always_on>1</always_on> <!-- Keeps camera always on -->
   <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
   <visualize>true</visualize>
   <topic>back_cam</topic> <!-- ign topic -->
   <camera>     <!-- Camera Properties -->
      <horizontal_fov>2.0944</horizontal_fov>
      <image>
        <width>640</width>
        <height>480</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.05</near>
        <far>50</far>
        </clip>
   </camera>
  </sensor>
```

```
<sensor name="front_right_camera" type="camera">
 <pose>1.0 -0.5 0.39 0 0 -1.57</pose>  <!-- x=1m, y=0.5 ,
 ↪  z=0.39m, r=0, p=0, y=0 -->
 <always_on>1</always_on> <!-- Keeps camera always on -->
 <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
 <visualize>true</visualize>
 <topic>front_right_cam</topic> <!-- ign topic -->
 <camera>     <!-- Camera Properties -->
    <horizontal_fov>2.0944</horizontal_fov>
    <image>
      <width>640</width>
      <height>480</height>
      <format>R8G8B8</format>
    </image>
    <clip>
      <near>0.05</near>
      <far>50</far>
      </clip>
 </camera>
</sensor>

<sensor name="back_right_camera" type="camera">
 <pose>-1.0 -0.5 0.39 0 0 -1.57</pose>  <!-- x=1m, y=0.5 ,
 ↪  z=0.39m, r=0, p=0, y=0 -->
 <always_on>1</always_on> <!-- Keeps camera always on -->
 <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
 <visualize>true</visualize>
 <topic>back_right_cam</topic> <!-- ign topic -->
 <camera>     <!-- Camera Properties -->
    <horizontal_fov>2.0944</horizontal_fov>
    <image>
      <width>640</width>
      <height>480</height>
      <format>R8G8B8</format>
    </image>
    <clip>
      <near>0.05</near>
      <far>50</far>
      </clip>
 </camera>
</sensor>

<sensor name="front_left_camera" type="camera">
 <pose>1.0 0.5 0.39 0 0 1.57</pose>  <!-- x=1m, y=0.5 , z=0.39m,
 ↪  r=0, p=0, y=0 -->
 <always_on>1</always_on> <!-- Keeps camera always on -->
 <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
 <visualize>true</visualize>
```

```
    <topic>front_left_cam</topic> <!-- ign topic -->
    <camera>     <!-- Camera Properties -->
       <horizontal_fov>2.0944</horizontal_fov>
       <image>
         <width>640</width>
         <height>480</height>
         <format>R8G8B8</format>
       </image>
       <clip>
         <near>0.05</near>
         <far>50</far>
         </clip>
    </camera>
   </sensor>

   <sensor name="back_left_camera" type="camera">
    <pose>-1.0 0.5 0.39 0 0 1.57</pose>  <!-- x=1m, y=0.5 , z=0.39m,
    ↪  r=0, p=0, y=0 -->
    <always_on>1</always_on> <!-- Keeps camera always on -->
    <update_rate>30</update_rate> <!-- 30 fps uprade rate -->
    <visualize>true</visualize>
    <topic>back_left_cam</topic> <!-- ign topic -->
    <camera>     <!-- Camera Properties -->
       <horizontal_fov>2.0944</horizontal_fov>
       <image>
         <width>640</width>
         <height>480</height>
         <format>R8G8B8</format>
       </image>
       <clip>
         <near>0.05</near>
         <far>50</far>
         </clip>
    </camera>
   </sensor>
 </link>

 <joint name="base_joint" type="fixed">
   <parent>base_footprint</parent>
   <child>base_link</child>
   <pose>0 0 0.09 0 0 0</pose> <!-- Fixed joint at 0.09m z-axis
   ↪  attache to Base Footprint -->
 </joint>
```

A.2 IMU Link

The code refers to IMU Link in `Flexley_Tug.sdf`. The file path is as follows inside the package.

src>abb_flexley_description>urdf>Flexley_Tug.sdf

```
<!-- IMU Link -->
    <link name="imu_link">
      <sensor name='imu_sensor' type='imu'>
        <always_on>1</always_on> <!-- For continuous operation of
        ↪  sensor -->
        <update_rate>200</update_rate>  <!-- 200 Hz update rate -->
        <pose>0 0 0.06 0 -0 0</pose>  <!-- x=0, y=0 , z=0.06m, r=0,
        ↪  p=0, y=0-->
        <visualize>true</visualize>  <!-- Enables visualization in
        ↪  gazebo -->
        <topic>/imu_scan</topic>    <!-- ign topic for publishing the
        ↪  data -->
        <frameName>imu_link</frameName>
        <imu>
          <angular_velocity>   <!-- Parameters for measuring angular
          ↪  velocity -->
            <x>
              <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
              </noise>
            </x>
            <y>
              <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
              </noise>
            </y>
            <z>
              <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
              </noise>
            </z>
          </angular_velocity>
          <linear_acceleration>  <!-- Parameters for measuring linear
          ↪  acceleration -->
            <x>
              <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>1.7e-2</stddev>
              </noise>
            </x>
            <y>
              <noise type="gaussian">
              <mean>0.0</mean>
```

```
            <stddev>1.7e-2</stddev>
            </noise>
          </y>
          <z>
            <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>1.7e-2</stddev>
            </noise>
          </z>
        </linear_acceleration>
      </imu>
    </sensor>
  </link>

  <joint name="imu_joint" type="fixed"> <!-- Fixed joint attached to
  ↪  base link -->
    <parent>base_link</parent>
    <child>imu_link</child>
    <pose>0 0 0.06 0 0 0</pose>
    <axis>
      <xyz>0 0 1</xyz>
    </axis>
  </joint>
```

A.3 Front Lidar

The code refers to `base_scan_front` in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<!-- Front Lidar -->
    <link name="base_scan_front">
      <pose relative_to="base_footprint" >1.0 0 0.04 0 0 3.14</pose>
      ↪  <!-- x=1.0m, y=0, z=0.04m, r=0,p=0,y=3.14 rad -->
      <inertial>
        <inertia>
          <ixx>0.00098</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.00098</iyy>
          <iyz>0</iyz>
          <izz>0.000146</izz>
        </inertia>
        <mass>1.17</mass> <!-- mass=1.17 kg -->
      </inertial>
      <collision name='lidar_sensor_front_collision'>
        <geometry>
```

```
    <cylinder>
      <length>0.05</length>
      <radius>0.05</radius>
    </cylinder>
  </geometry>
</collision>
<visual name='lidar_sensor_front_visual'>
  <material>
   <ambient>1 1 0 1</ambient>
   <diffuse>1 1 0 1</diffuse>
  </material>
  <geometry>
    <mesh>
      <scale>0.001 0.001 0.001</scale> <!-- scaled to 0.001 for
      ↪  units in meters -->
      <uri>lidar_front.stl</uri>
    </mesh>
  </geometry>
</visual>
<sensor name='lidar_front' type='gpu_lidar'> <!-- sensor
↪  properties -->
   <pose> 0.01 0 0.22 0 0 3.14 </pose>  <!-- x=0.01m y=0m,
   ↪  z=0.22m, r=0, p=0, y=3.14 rad -->
   <topic>lidar_front</topic>  <!-- ign topic -->
   <frame_name>base_scan_front</frame_name>  <!-- ign frame -->
   <update_rate>5</update_rate>  <!-- 5 Hz suitable for low speed
   ↪  robots -->
   <always_on>1</always_on> <!-- Sensor active all the time -->
   <visualize>true</visualize> <!-- visualization in gazebo -->
   <ray>
        <scan>
            <horizontal>
                 <samples>720</samples>  <!-- Number of
                 ↪  horizontal laser rays emitted -->
                 <resolution>1</resolution>  <!-- Angular
                 ↪  resolution multiplier -->
                 <min_angle>-1.396263</min_angle> <!-- minimum
                 ↪  angle in horizontal direction-->
                 <max_angle>1.396263</max_angle>  <!-- maximum
                 ↪  angle in horizontal direction-->
            </horizontal>
            <vertical>
                 <samples>1</samples>  <!-- Number of horizontal
                 ↪  laser rays emitted -->
                 <resolution>0.01</resolution> <!-- vertical
                 ↪  resolution -->
                 <min_angle>0</min_angle> <!-- zero spreed in
                 ↪  vertical direction(2D LIdar)-->
                 <max_angle>0</max_angle> <!-- zero spreed in
                 ↪  vertical direction(2D LIdar)-->
```

```
            </vertical>
          </scan>
          <range>
              <min>0.08</min>  <!-- minimum range (meters) -->
              <max>10.0</max>  <!-- maximum range (meters) -->
              <resolution>0.01</resolution>  <!-- lidar rays
              ↪  resolution -->
          </range>
      </ray>
    </sensor>
  </link>
  <joint name="lidar_front_joint" type="fixed">
    <pose>1.1 0 0 0 0 0</pose> <!-- x=1.1m y=0m, z=0, r=0, p=0, y=0
    ↪  -->
    <parent>base_footprint</parent>
    <child>base_scan_front</child>
    <axis>
      <xyz>0 0 1</xyz>
    </axis>
  </joint>
```

A.4 Back Lidar

The code refers to `base_scan_back` in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<!-- Back Lidar -->
    <link name="base_scan_back">
     <pose relative_to="base_footprint" >-0.9 0 0.04 0 0 0</pose> <!--
     ↪  x=-0.9m, y=0, z=0.04m, r=0,p=0,y=0 -->
     <inertial>
        <inertia>
          <ixx>0.00098</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.00098</iyy>
          <iyz>0</iyz>
          <izz>0.000146</izz>
        </inertia>
        <mass>1.17</mass>
      </inertial>

     <collision name='lidar_sensorback_collision'>
        <geometry>
          <cylinder>
            <length>0.05</length>
```

```
      <radius>0.05</radius>
    </cylinder>
  </geometry>
</collision>


<visual name='lidar_sensor_back_visual'>

  <material>
    <ambient>1 1 0 1</ambient>
    <diffuse>1 1 0 1</diffuse>
  </material>
  <geometry>
    <mesh>
      <scale>0.001 0.001 0.001</scale> <!-- scaled to 0.001 for
      ↪  units in meters -->
      <uri>lidar_front.stl</uri>
    </mesh>
  </geometry>
</visual>

<sensor name='lidar_back' type='gpu_lidar'> <!-- sensor
↪  properties -->
   <pose>-0.05 0.0 0.22 0 0 3.14</pose>  <!-- x=-0.05m y=0m,
   ↪  z=0.22m, r=0, p=0, y=3.14 rad -->
   <topic>lidar_back</topic>  <!-- ign topic -->
   <frame_name>base_scan_back</frame_name>  <!-- ign frame -->
   <update_rate>5</update_rate>  <!-- 5 Hz suitable for low speed
   ↪  robots -->
   <always_on>1</always_on> <!-- Sensor active all the time -->
   <visualize>true</visualize> <!-- visualization in gazebo -->
   <ray>
        <scan>
            <horizontal>
                 <samples>720</samples>  <!-- Number of
                 ↪  horizontal laser rays emitted -->
                 <resolution>1</resolution>  <!-- Angular
                 ↪  resolution multiplier -->
                 <min_angle>-1.396263</min_angle> <!-- minimum
                 ↪  angle in horizontal direction-->
                 <max_angle>1.396263</max_angle>  <!-- maximum
                 ↪  angle in horizontal direction-->
            </horizontal>
            <vertical>
                 <samples>1</samples>  <!-- Number of horizontal
                 ↪  laser rays emitted -->
                 <resolution>0.01</resolution> <!-- vertical
                 ↪  resolution -->
                 <min_angle>0</min_angle> <!-- zero spreed in
                 ↪  vertical direction(2D LIdar)-->
```

```
                    <max_angle>0</max_angle> <!-- zero spreed in
                    ↪  vertical direction(2D LIdar)-->
                </vertical>
            </scan>
            <range>
                <min>0.08</min>  <!-- minimum range (meters) -->
                <max>10.0</max>  <!-- maximum range (meters) -->
                <resolution>0.01</resolution>  <!-- lidar rays
                ↪  resolution -->
            </range>
        </ray>
      </sensor>

    </link>

    <joint name="lidar_back_joint" type="fixed">
      <pose relative_to='base_scan_back'/>
      <parent>base_footprint</parent>
      <child>base_scan_back</child>
      <axis>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>
```

A.5 Wheels

The code refers to all the wheels in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<!-- Front Right Wheel -->
    <link name='front_right_wheel'>
      <inertial>   <!-- Inertial properties-->
        <pose>0.580 -0.200 -0.02 -1.5707 0 0</pose>
        <mass>0.765</mass> <!-- Mass of the wheel in kg -->
        <inertia> <!-- Inertia tensor components in kg·m² -->
            <ixx>0.0019</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.0011</iyy>
            <iyz>0</iyz>
            <izz>0.0011</izz>
        </inertia>
      </inertial>

      <collision name='front_right_wheel_collision'>  <!-- Collision
      ↪  element  -->
        <pose>0.580 -0.200 -0.02 -1.5707 0 0</pose>
```

```
      <geometry>
        <cylinder>
          <radius>0.075</radius> <!-- Radius of the wheel in meters
          ↪  -->
          <length>0.05</length> <!-- Width (thickness) of the wheel
          ↪  -->
        </cylinder>
      </geometry>
      <surface>
         <friction>
          <ode>
           <mu>0.90</mu> <!-- Friction coefficient in longitudinal
           ↪  direction -->
           <mu2>0.25</mu2> <!-- Friction coefficient in lateral
           ↪  direction -->
           <fdir1>0 1 0</fdir1>
          </ode>
         </friction>
      </surface>

    </collision>

    <visual name='front_right_wheel_visual'> <!-- Visual element  -->
     <pose>0.580 -0.200 -0.02 0 0 0</pose>
     <material>
        <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
        <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
     </material>
     <geometry>
        <mesh>
          <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
          ↪  meters -->
          <uri>../meshes/wheel_front_right.stl</uri> <!-- Mesh file
          ↪  for visual appearance -->
        </mesh>
     </geometry>
    </visual>

  </link>

  <joint name="front_right_wheel_joint" type="revolute">
    <pose>0.580 -0.200 -0.02 0 -1.5707 0</pose>
    <parent>base_link</parent>
    <child>front_right_wheel</child>
    <dynamics>
      <damping>0</damping> <!-- Resistance to motion -->
    </dynamics>
    <effort>10000.0</effort> <!-- Max torque that can be applied -->
    <velocity>100.0</velocity> <!-- Max rotational velocity -->
```

```
    <axis>
      <xyz> 0 1 0</xyz> <!-- Rotation axis of the wheel (around
      ↪  y-axis) -->
      <limit> <!-- Very large limits simulate continuous rotation -->
        <lower>-1.7e+308</lower>
        <upper>1.7e+308</upper>
      </limit>
    </axis>
  </joint>

  <!-- Front Left Wheel -->
  <link name='front_left_wheel'>

    <inertial>  <!-- Inertial properties-->
      <pose>0.580 0.200 -0.02 -1.5707 0 0</pose>
      <mass>0.765</mass> <!-- Mass of the wheel in kg -->
      <inertia> <!-- Inertia tensor components in kg·m² -->
          <ixx>0.0019</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.0011</iyy>
          <iyz>0</iyz>

          <izz>0.0011</izz>
      </inertia>
    </inertial>

    <collision name='front_left_wheel_collision'> <!-- Collision
    ↪  element  -->
      <pose>0.580 0.200 -0.02 -1.5707 0 0</pose>
      <geometry>
        <cylinder>
          <radius>0.075</radius> <!-- Radius of the wheel in meters
          ↪  -->
          <length>0.05</length>  <!-- Width (thickness) of the wheel
          ↪  -->
        </cylinder>
      </geometry>
      <surface>
         <friction>
           <ode>
            <mu>0.90</mu> <!-- Friction coefficient in longitudinal
            ↪  direction -->
            <mu2>0.25</mu2> <!-- Friction coefficient in lateral
            ↪  direction -->
            <fdir1>0 1 0</fdir1>
           </ode>
          </friction>
      </surface>
```

```
  </collision>
  <visual name='front_left_wheel_visual'> <!-- Visual element  -->
    <pose>0.580 0.200 -0.02 0 0 0</pose>
    <material>
     <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
     <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
    </material>
    <geometry>
      <mesh>
        <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
        ↪  meters -->
        <uri>../meshes/wheel_front_left.stl</uri> <!-- Mesh file
        ↪  for visual appearance -->
      </mesh>
    </geometry>
  </visual>

</link>

<joint name="front_left_wheel_joint" type="revolute">
  <pose>0.580 0.200 -0.02 0 -1.5707 0</pose>
  <parent>base_link</parent>
  <child>front_left_wheel</child>
  <dynamics>
    <damping>0</damping> <!-- Resistance to motion -->
  </dynamics>
  <effort>10000.0</effort> <!-- Max torque that can be applied -->
  <velocity>100.0</velocity> <!-- Max rotational velocity -->
  <axis>
    <xyz> 0 1 0</xyz> <!-- Rotation axis of the wheel (around
    ↪  y-axis) -->
    <limit>  <!-- Very large limits simulate continuous rotation
    ↪  -->
      <lower>-1.7e+308</lower>
      <upper>1.7e+308</upper>
    </limit>
  </axis>
</joint>

<!-- Back Right Wheel -->
<link name='back_right_wheel'>
 <inertial> <!-- Inertial properties-->
    <pose>-0.580 -0.200 -0.02 -1.5707 0 0</pose>
    <mass>0.765</mass> <!-- Mass of the wheel in kg -->
    <inertia> <!-- Inertia tensor components in kg·m² -->
        <ixx>0.0019</ixx>
        <ixy>0</ixy>
        <ixz>0</ixz>
```

```
        <iyy>0.0011</iyy>
        <iyz>0</iyz>
        <izz>0.0011</izz>
      </inertia>
    </inertial>

    <collision name='back_right_wheel_collision'> <!-- Collision
    ↪  element  -->
      <pose>-0.580 -0.200 -0.02 -1.5707 0 0</pose>
      <geometry>
        <cylinder>
          <radius>0.075</radius> <!-- Radius of the wheel in meters
          ↪  -->
          <length>0.05</length> <!-- Width (thickness) of the wheel
          ↪  -->
        </cylinder>
      </geometry>
      <surface>
         <friction>
          <ode>
           <mu>0.90</mu> <!-- Friction coefficient in longitudinal
           ↪  direction -->
           <mu2>0.25</mu2> <!-- Friction coefficient in lateral
           ↪  direction -->
           <fdir1>0 1 0</fdir1>
          </ode>
         </friction>
      </surface>

    </collision>

    <visual name='back_right_wheel_visual'> <!-- Visual element  -->
      <pose>-0.580 -0.200 -0.02 0 0 0</pose>
      <material>
       <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
       <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
      </material>
      <geometry>
        <mesh>
          <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
          ↪  meters -->
          <uri>../meshes/wheel_front_right.stl</uri> <!-- Mesh file
          ↪  for visual appearance -->
        </mesh>
      </geometry>
    </visual>

  </link>
```

```
<joint name="back_right_wheel_joint" type="revolute">
  <pose>-0.580 -0.200 -0.02 0 -1.5707 0</pose>
  <parent>base_link</parent>
  <child>back_right_wheel</child>
  <dynamics>
    <damping>0</damping> <!-- Resistance to motion -->
  </dynamics>
  <effort>10000.0</effort> <!-- Max torque that can be applied -->
  <velocity>100.0</velocity> <!-- Max rotational velocity -->
  <axis>
    <xyz> 0 1 0</xyz> <!-- Rotation axis of the wheel (around
    ↪  y-axis) -->
    <limit>  <!-- Very large limits simulate continuous rotation
    ↪  -->
      <lower>-1.7e+308</lower>
      <upper>1.7e+308</upper>
    </limit>
  </axis>
</joint>

<!-- Back Left Wheel -->
<link name='back_left_wheel'>
  <inertial> <!-- Inertial properties-->
    <pose>-0.580 0.200 -0.02 -1.5707 0 0</pose>
    <mass>0.765</mass> <!-- Mass of the wheel in kg -->
    <inertia> <!-- Inertia tensor components in kg·m² -->
        <ixx>0.0019</ixx>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyy>0.0011</iyy>
        <iyz>0</iyz>

        <izz>0.0011</izz>
    </inertia>
  </inertial>

  <collision name='back_left_wheel_collision'> <!-- Collision
  ↪  element  -->
    <pose>-0.580 0.200 -0.02 -1.5707 0 0</pose>
    <geometry>
      <cylinder>
        <radius>0.075</radius> <!-- Radius of the wheel in meters
        ↪  -->
        <length>0.05</length> <!-- Width (thickness) of the wheel
        ↪  -->
      </cylinder>
    </geometry>
    <surface>
       <friction>
```

```
          <ode>
           <mu>0.90</mu> <!-- Friction coefficient in longitudinal
           ↪  direction -->
           <mu2>0.25</mu2> <!-- Friction coefficient in lateral
           ↪  direction -->
           <fdir1>0 1 0</fdir1>
          </ode>
         </friction>
    </surface>

  </collision>

  <visual name='back_left_wheel_visual'> <!-- Visual element  -->
    <pose>-0.580 0.200 -0.02 0 0 0</pose>
    <material>
     <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
     <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
    </material>
    <geometry>
      <mesh>
        <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
        ↪  meters -->
        <uri>../meshes/wheel_front_right.stl</uri> <!-- Mesh file
        ↪  for visual appearance -->
      </mesh>
    </geometry>
  </visual>
</link>

<joint name="back_left_wheel_joint" type="revolute">
  <pose>-0.580 0.200 -0.02 0 -1.5707 0</pose>
  <parent>base_link</parent>
  <child>back_left_wheel</child>
  <dynamics>
    <damping>0</damping> <!-- Resistance to motion -->
  </dynamics>
  <effort>10000.0</effort> <!-- Max torque that can be applied -->
  <velocity>100.0</velocity> <!-- Max rotational velocity -->
  <axis>
    <xyz> 0 1 0</xyz> <!-- Rotation axis of the wheel (around
    ↪  y-axis) -->
    <limit>  <!-- Very large limits simulate continuous rotation
    ↪  -->
      <lower>-1.7e+308</lower>
      <upper>1.7e+308</upper>
    </limit>
  </axis>
</joint>
```

A.6 Castor Wheels

The code refers to all the castor wheels in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<!-- Front Right Castor Wheel -->
 <link name='castor_front_right'>
   <inertial> <!-- Inertial properties-->
      <pose>0.920 -0.170 -0.05 0 0 0</pose>
      <mass>0.103</mass> <!-- Mass of the wheel in kg -->
      <inertia> <!-- Inertia tensor components in kg·m² -->
          <ixx>0.016</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.016</iyy>
          <iyz>0</iyz>
          <izz>0.016</izz>
      </inertia>
   </inertial>
   <visual name='visual_castor_front_right'> <!-- Visual element  -->
    <pose>0.920 -0.170 -0.05 0 0 0</pose>
    <material>
       <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
       <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
    </material>
    <geometry>
      <mesh>
       <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
       ↪  meters -->
       <uri>../meshes/castor_front_right.stl</uri> <!-- Mesh file for
       ↪  visual appearance -->
      </mesh>
    </geometry>
   </visual>
   <collision name='collision_castor_front_right'> <!-- Collision
   ↪  element  -->
      <pose>0.920 -0.170 -0.05 0 0 0</pose>
      <geometry>
        <sphere>
           <radius>0.025</radius> <!-- Radius in meters -->
        </sphere>
      </geometry>
   </collision>
 </link>
 <joint name='caster__front_right_joint' type='ball'>
    <pose>0.920 -0.170 -0.05 0 0 0</pose>
    <parent>base_link</parent>
    <child>castor_front_right</child>
```

```
</joint>
<!-- Front Left Castor Wheel -->
 <link name='castor_front_left'>
  <inertial> <!-- Inertial properties-->
     <pose>0.920 0.170 -0.05 0 0 0</pose>
     <mass>0.103</mass> <!-- Mass of the wheel in kg -->
     <inertia> <!-- Inertia tensor components in kg·m² -->
         <ixx>0.016</ixx>
         <ixy>0</ixy>
         <ixz>0</ixz>
         <iyy>0.016</iyy>
         <iyz>0</iyz>
         <izz>0.016</izz>
     </inertia>
  </inertial>
  <visual name='visual_castor_front_left'> <!-- Visual element  -->
   <pose>0.920 0.170 -0.05 0 0 0</pose>
   <material>
      <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
      <diffuse>0 1 0 1</diffuse>  <!-- Diffuse color: green -->
   </material>
   <geometry>
     <mesh>
      <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
      ↪  meters -->
      <uri>../meshes/castor_front_right.stl</uri> <!-- Mesh file for
      ↪  visual appearance -->
     </mesh>
   </geometry>
  </visual>
  <collision name='collision_castor_front_left'> <!-- Collision
  ↪  element  -->
     <pose>0.920 0.170 -0.05 0 0 0</pose>
     <geometry>
       <sphere>
          <radius>0.025</radius> <!-- Radius in meters -->
       </sphere>
     </geometry>
  </collision>
 </link>
 <joint name='caster_front_left_joint' type='ball'>
   <pose>0.920 0.170 -0.05 0 0 0</pose>
   <parent>base_link</parent>
   <child>castor_front_left</child>
 </joint>

 <!-- Back Right Castor Wheel -->
<link name='castor_back_right'>
  <inertial>  <!-- Inertial properties-->
```

```
      <pose>-0.790 -0.170 -0.05 0 0 0</pose>
      <mass>0.103</mass> <!-- Mass of the wheel in kg -->
      <inertia>  <!-- Inertia tensor components in kg·m² -->
          <ixx>0.016</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.016</iyy>
          <iyz>0</iyz>
          <izz>0.016</izz>
      </inertia>
   </inertial>
   <visual name='visual_castor_back_right'> <!-- Visual element  -->
    <pose>-0.790 -0.170 -0.05 0 0 0</pose>
    <material>
       <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
       <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
    </material>
    <geometry>
      <mesh>
       <scale>0.001 0.001 0.001</scale> <!-- Scale down from mm to
       ↪  meters -->
       <uri>../meshes/castor_front_right.stl</uri> <!-- Mesh file for
       ↪  visual appearance -->
      </mesh>
    </geometry>
   </visual>
   <collision name='collision_castor_back_right'> <!-- Collision
   ↪  element  -->
      <pose>-0.790 -0.170 -0.05 0 0 0</pose>
      <geometry>
        <sphere>
           <radius>0.025</radius> <!-- Radius in meters -->
        </sphere>
      </geometry>
   </collision>
</link>
<joint name='caster_back_right_joint' type='ball'>
   <pose>-0.790 -0.170 -0.05 0 0 0</pose>
   <parent>base_link</parent>
   <child>castor_back_right</child>
</joint>
<!-- Back Left Castor Wheel -->
 <link name='castor_back_left'>
  <inertial> <!-- Inertial properties-->
      <pose>-0.790 0.170 -0.05 0 0 0</pose>
      <mass>0.103</mass> <!-- Mass of the wheel in kg -->
      <inertia>  <!-- Inertia tensor components in kg·m² -->
          <ixx>0.016</ixx>
          <ixy>0</ixy>
```

```
          <ixz>0</ixz>
          <iyy>0.016</iyy>
          <iyz>0</iyz>
          <izz>0.016</izz>
       </inertia>
    </inertial>
    <visual name='visual_castor_back_left'> <!-- Visual element  -->
     <pose>-0.790 0.170 -0.05 0 0 0</pose>
     <material>
        <ambient>0 1 0 1</ambient> <!-- Ambient color: green -->
        <diffuse>0 1 0 1</diffuse> <!-- Diffuse color: green -->
     </material>
     <geometry>
       <mesh>
        <scale>0.001 0.001 0.001</scale>  <!-- Scale down from mm to
        ↪  meters -->
        <uri>../meshes/castor_front_right.stl</uri>  <!-- Mesh file
        ↪  for visual appearance -->
       </mesh>
     </geometry>
    </visual>
    <collision name='collision_castor_back_left'> <!-- Collision
    ↪  element  -->
       <pose>-0.790 0.170 -0.05 0 0 0</pose>
       <geometry>
         <sphere>
            <radius>0.025</radius>  <!-- Radius in meters -->
         </sphere>
       </geometry>
    </collision>
   </link>
   <joint name='caster_back_left_joint' type='ball'>
     <pose>-0.790 0.170 -0.05 0 0 0</pose>
     <parent>base_link</parent>
     <child>castor_back_left</child>
   </joint>
```

A.7 Gazebo Plugins

The code refers to gazebo plugins in `Flexley_Tug.sdf`. The file path is as follows inside the package.
`src>abb_flexley_description>urdf>Flexley_Tug.sdf`

```
<!-- Gazebo Plugins -->
    <plugin name="ignition::gazebo::systems::DiffDrive"
        filename="libignition-gazebo-diff-drive-system.so">
```

```
    <left_joint>front_left_wheel_joint</left_joint>
    <left_joint>back_left_wheel_joint</left_joint>
    <right_joint>front_right_wheel_joint</right_joint>
    <right_joint>back_right_wheel_joint</right_joint>
    <wheel_separation>0.400</wheel_separation> <!-- Distance between
    ↪  left and right wheels (meters) -->
    <wheel_radius>0.075</wheel_radius> <!-- Radius of the wheels
    ↪  (meters) -->
    <topic>/cmd_vel</topic> <!-- Topic to receive velocity commands
    ↪  -->
    <odom_topic>/odom</odom_topic> <!-- Topic to publish odometry
    ↪  -->
    <base_frame>base_link</base_frame> <!-- Base frame used in the
    ↪  odometry message -->
  </plugin>

  <plugin name="ignition::gazebo::systems::JointStatePublisher"
  ↪  filename="libignition-gazebo-joint-state-publisher-system.so">
   <update_rate>30</update_rate> <!-- Update rate in Hz for
   ↪  publishing joint states -->
   <joint_name>front_left_wheel_joint</joint_name>
   <joint_name>front_right_wheel_joint</joint_name>
   <topic>/joint_states</topic> <!-- Topic for publishing joint
   ↪  states (sensor_msgs/JointState) -->
  </plugin>

  <plugin name="ignition::gazebo::systems::DetachableJoint"
  ↪  filename="libignition-gazebo-detachable-joint-system.so" >

    <parent_link>base_link</parent_link>
    <child_model>trolley</child_model>
    <child_link>base</child_link>
    <output_topic>/detachable_joint_state</output_topic>
    <attach_topic>/attach</attach_topic>
    <detach_topic>/detach</detach_topic>

  </plugin>
```

A.8 Control Node

The code refers to control node in the package. The file path is as follows inside the package.
src>abb_flexley_description>abb_flexley_description>Control_Node.py

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
```

```
from geometry_msgs.msg import Twist
import sys
import termios
import tty
import threading
import select

def get_key(timeout=0.1):
    """Non-blocking keyboard input using select."""
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(fd)
        rlist, _, _ = select.select([sys.stdin], [], [], timeout)
        if rlist:
            return sys.stdin.read(1)
        return None
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

class Controller(Node):
    def __init__(self):
        super().__init__('controller')

        # Obstacle detection
        self.front_range = float('inf')
        self.back_range = float('inf')
        self.obstacle_threshold = 0.5
        self.obstacle_blocked = False  # To prevent repeated logging

        # Speed settings
        self.linear_speed = 0.5
        self.angular_speed = 0.5

        # State
        self.pressed_keys = set()
        self.running = True

        # ROS interfaces
        self.cmd_pub = self.create_publisher(Twist, '/cmd_vel', 10)
        self.front_sub = self.create_subscription(LaserScan,
        ↪  '/lidar_front', self.front_callback, 10)
        self.back_sub = self.create_subscription(LaserScan,
        ↪  '/lidar_back', self.back_callback, 10)
        self.create_timer(0.1, self.control_loop)

        # Keyboard input thread
        self.key_thread =
        ↪  threading.Thread(target=self.keyboard_listener,
        ↪  daemon=True)
```

```
        self.key_thread.start()

    def front_callback(self, msg):
        valid_ranges = [r for r in msg.ranges if 0.01 < r <
        ↪  float('inf')]
        self.front_range = min(valid_ranges) if valid_ranges else
        ↪  float('inf')

    def back_callback(self, msg):
        valid_ranges = [r for r in msg.ranges if 0.01 < r <
        ↪  float('inf')]
        self.back_range = min(valid_ranges) if valid_ranges else
        ↪  float('inf')

    def keyboard_listener(self):
        while self.running:
            key = get_key(timeout=0.1)
            if not self.running:
                break
            if key is None:
                continue
            if key == 'q':
                self.get_logger().info("Quit key pressed  shutting
                ↪  down.")
                self.running = False
                rclpy.shutdown()
                break
            elif key in [' ', 'x']:
                self.get_logger().info("Emergency stop triggered.")
                self.pressed_keys.clear()
                self.publish_stop()
            elif key in ['w', 'a', 's', 'd']:
                self.pressed_keys.add(key)
            elif key in ['W', 'A', 'S', 'D']:
                self.pressed_keys.discard(key.lower())

    def control_loop(self):
        twist = Twist()

        wants_forward = 'w' in self.pressed_keys
        wants_backward = 's' in self.pressed_keys
        wants_left = 'a' in self.pressed_keys
        wants_right = 'd' in self.pressed_keys

        # Detect if the current command would hit an obstacle
        forward_blocked = wants_forward and self.front_range <
        ↪  self.obstacle_threshold
        backward_blocked = wants_backward and self.back_range <
        ↪  self.obstacle_threshold
```

```
        movement_blocked = forward_blocked or backward_blocked

        if movement_blocked:
            if not self.obstacle_blocked:
                self.get_logger().warn(" Obstacle detected  stopping
                ↪  all motion.")
                self.publish_stop()
                self.obstacle_blocked = True
            return  # Do not continue motion
        else:
            # Clear the block state once movement becomes safe
            self.obstacle_blocked = False

        # Set linear velocity
        if wants_forward:
            twist.linear.x = self.linear_speed
        elif wants_backward:
            twist.linear.x = -self.linear_speed

        # Set angular velocity
        if wants_left:
            twist.angular.z = self.angular_speed
        elif wants_right:
            twist.angular.z = -self.angular_speed

        self.cmd_pub.publish(twist)

    def publish_stop(self):
        """Publishes zero velocity to stop the robot."""
        stop_twist = Twist()
        self.cmd_pub.publish(stop_twist)

def main(args=None):
    rclpy.init(args=args)
    node = Controller()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info("KeyboardInterrupt | exiting cleanly.")
    finally:
        node.running = False
        node.key_thread.join(timeout=1.0)
        node.publish_stop()
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A.9 Sensor Publisher

The code refers to sensor publisher node in the package. The file path is as follows inside the package.
src>abb_flexley_description>abb_flexley_description>sensor_publisher.py

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Imu, LaserScan
import math
import random

class SensorPublisher(Node):
    def __init__(self):
        super().__init__('sensor_publisher')

        # Publishers
        self.imu_pub = self.create_publisher(Imu, '/imu_scan', 10)
        self.front_lidar_pub = self.create_publisher(LaserScan,
        ↪  '/lidar_front', 10)
        self.back_lidar_pub = self.create_publisher(LaserScan,
        ↪  '/lidar_back', 10)


        self.timer = self.create_timer(0.1, self.publish_data)  # 10 Hz

    def publish_data(self):
        now = self.get_clock().now().to_msg()

        # --- IMU ---
        imu_msg = Imu()
        imu_msg.header.stamp = now
        imu_msg.header.frame_id = 'imu_link'
        imu_msg.linear_acceleration.x = random.uniform(-0.1, 0.1)
        imu_msg.linear_acceleration.y = 0.0
        imu_msg.linear_acceleration.z = 9.81
        imu_msg.angular_velocity.x = 0.0
        imu_msg.angular_velocity.y = 0.0
        imu_msg.angular_velocity.z = random.uniform(-0.1, 0.1)
        self.imu_pub.publish(imu_msg)

        # --- LIDAR parameters ---
        angle_min = -1.396263
        angle_max = 1.396263
        angle_increment = math.radians(0.5)
        range_min = 0.08
        range_max = 10.0
```

```
        num_readings = int((angle_max - angle_min) / angle_increment)

        # --- Front LIDAR ---
        front_scan = LaserScan()
        front_scan.header.stamp = now
        front_scan.header.frame_id = 'lidar_front'
        front_scan.angle_min = angle_min
        front_scan.angle_max = angle_max
        front_scan.angle_increment = angle_increment
        front_scan.range_min = range_min
        front_scan.range_max = range_max
        front_scan.ranges = [random.uniform(1.0, 5.0) for _ in
        ↪  range(num_readings)]
        self.front_lidar_pub.publish(front_scan)

        # --- Back LIDAR ---
        back_scan = LaserScan()
        back_scan.header.stamp = now
        back_scan.header.frame_id = 'lidar_back'
        back_scan.angle_min = angle_min
        back_scan.angle_max = angle_max
        back_scan.angle_increment = angle_increment
        back_scan.range_min = range_min
        back_scan.range_max = range_max
        back_scan.ranges = [random.uniform(1.0, 5.0) for _ in
        ↪  range(num_readings)]
        self.back_lidar_pub.publish(back_scan)

        self.get_logger().info(
        f"IMU → Acc: ({imu_msg.linear_acceleration.x:.2f}, "
        f"{imu_msg.linear_acceleration.y:.2f},
        ↪  {imu_msg.linear_acceleration.z:.2f}) "
        f"Gyro: (0.0, 0.0, {imu_msg.angular_velocity.z:.2f})\n"
        f"LIDAR Front Ranges (sample): {front_scan.ranges[:5]}\n"
        f"LIDAR Back Ranges (sample): {back_scan.ranges[:5]}"
        )

def main(args=None):
    rclpy.init(args=args)
    node = SensorPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

A.10 Lidar Frame Merger

The code refers to lidar frame merger node in the package. The file path is as follows inside the package.
src>abb_flexley_description>abb_flexley_description>lidar_frame_merger.py

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from std_msgs.msg import Header

class LidarFrameMerger(Node):
    def __init__(self):
        super().__init__('lidar_frame_merger')

        # Subscribers
        self.sub_front = self.create_subscription(
            LaserScan, '/lidar_front', self.front_cb, 10)
        self.sub_back = self.create_subscription(
            LaserScan, '/lidar_back', self.back_cb, 10)

        # Publishers for fixed frames
        self.pub_front_fixed = self.create_publisher(LaserScan,
        ↪  '/lidar_front_fixed', 10)
        self.pub_back_fixed = self.create_publisher(LaserScan,
        ↪  '/lidar_back_fixed', 10)

        # Publisher for merged 360° scan
        self.pub_scan = self.create_publisher(LaserScan, '/scan', 10)

        # Store latest scans
        self.front_scan = None
        self.back_scan = None

        self.get_logger().info('LidarFrameMerger node started')

    def front_cb(self, msg):
        fixed_msg = self.copy_scan_with_frame(msg, 'base_link')
        self.pub_front_fixed.publish(fixed_msg)
        self.front_scan = fixed_msg
        self.try_merge()

    def back_cb(self, msg):
        fixed_msg = self.copy_scan_with_frame(msg, 'base_link')
        self.pub_back_fixed.publish(fixed_msg)
        self.back_scan = fixed_msg
        self.try_merge()

    def copy_scan_with_frame(self, msg, frame_id):
```

```
        """Creates a copy of a LaserScan message with a fixed frame."""
        fixed = LaserScan()
        fixed.header = Header()
        fixed.header.stamp = self.get_clock().now().to_msg()
        fixed.header.frame_id = frame_id
        fixed.angle_min = msg.angle_min
        fixed.angle_max = msg.angle_max
        fixed.angle_increment = msg.angle_increment
        fixed.time_increment = msg.time_increment
        fixed.scan_time = msg.scan_time
        fixed.range_min = msg.range_min
        fixed.range_max = msg.range_max
        fixed.ranges = list(msg.ranges)
        fixed.intensities = list(msg.intensities)
        return fixed

    def try_merge(self):
        if self.front_scan is None or self.back_scan is None:
            return

        merged = LaserScan()
        merged.header = Header()
        merged.header.stamp = self.get_clock().now().to_msg()
        merged.header.frame_id = 'base_link'  # single merged frame

        # We'll keep min/max of the scans
        merged.angle_min = -3.14159  # -
        merged.angle_max = 3.14159   # +
        merged.angle_increment = self.front_scan.angle_increment
        merged.time_increment = self.front_scan.time_increment
        merged.scan_time = self.front_scan.scan_time
        merged.range_min = min(self.front_scan.range_min,
        ↪  self.back_scan.range_min)
        merged.range_max = max(self.front_scan.range_max,
        ↪  self.back_scan.range_max)

        # Concatenate the ranges and intensities
        merged.ranges = list(self.front_scan.ranges) +
        ↪  list(self.back_scan.ranges)
        if self.front_scan.intensities and self.back_scan.intensities:
            merged.intensities = list(self.front_scan.intensities) +
            ↪  list(self.back_scan.intensities)
        else:
            merged.intensities = []

        self.pub_scan.publish(merged)
        self.get_logger().debug(f'Published merged scan in frame:
        ↪  {merged.header.frame_id}')
```

```
def main(args=None):
    rclpy.init(args=args)
    node = LidarFrameMerger()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```

A.11 Multi Camera Stitcher

The code refers to multi camera stitcher node in the package. The file path is as follows inside the package.
src>abb_flexley_description>abb_flexley_description>multi_camera_stitcher.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
import cv2
import numpy as np
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge
from message_filters import ApproximateTimeSynchronizer, Subscriber


class MultiCameraStitcher(Node):
    def __init__(self):
        super().__init__('multi_camera_stitcher')

        self.bridge = CvBridge()

        # --- Subscriptions to 6 camera feeds ---
        self.sub_front = Subscriber(self, Image, '/camera_front')
        self.sub_back = Subscriber(self, Image, '/camera_back')
        self.sub_front_right = Subscriber(self, Image,
        ↪  '/camera_front_right')
        self.sub_back_right = Subscriber(self, Image,
        ↪  '/camera_back_right')
```

```
        self.sub_front_left = Subscriber(self, Image,
        ↪  '/camera_front_left')
        self.sub_back_left = Subscriber(self, Image,
        ↪  '/camera_back_left')

        # --- Synchronize all 6 streams ---
        self.ts = ApproximateTimeSynchronizer(
            [self.sub_front, self.sub_back,
             self.sub_front_right, self.sub_back_right,
             self.sub_front_left, self.sub_back_left],
            queue_size=10,
            slop=0.1
        )
        self.ts.registerCallback(self.callback)

        # --- Publishers ---
        self.pub_image = self.create_publisher(Image,
        ↪  '/camera_360/image_raw', 10)
        self.pub_info = self.create_publisher(CameraInfo,
        ↪  '/camera_360/camera_info', 10)

        self.get_logger().info('Multi-camera stitcher node started.')

    def callback(self, front, back, fr, br, fl, bl):
        try:
            # Convert ROS images to OpenCV
            imgs = [
                self.bridge.imgmsg_to_cv2(front, 'bgr8'),
                self.bridge.imgmsg_to_cv2(fr, 'bgr8'),
                self.bridge.imgmsg_to_cv2(br, 'bgr8'),
                self.bridge.imgmsg_to_cv2(back, 'bgr8'),
                self.bridge.imgmsg_to_cv2(bl, 'bgr8'),
                self.bridge.imgmsg_to_cv2(fl, 'bgr8')
            ]

            # Resize for consistency (optional)
            imgs = [cv2.resize(img, (320, 240)) for img in imgs]

            # Simple horizontal merge (side-by-side 360° strip)
            merged = cv2.hconcat(imgs)

            # Create ROS Image message
            img_msg = self.bridge.cv2_to_imgmsg(merged,
            ↪  encoding='bgr8')
            img_msg.header.stamp = self.get_clock().now().to_msg()
            img_msg.header.frame_id = "base_link"  # Important for RViz

            # --- Create CameraInfo message ---
            cam_info = CameraInfo()
```

```
            cam_info.header = img_msg.header
            cam_info.width = merged.shape[1]
            cam_info.height = merged.shape[0]
            cam_info.distortion_model = "plumb_bob"
            cam_info.d = [0.0, 0.0, 0.0, 0.0, 0.0]
            cam_info.k = [
                1.0, 0.0, cam_info.width / 2.0,
                0.0, 1.0, cam_info.height / 2.0,
                0.0, 0.0, 1.0
            ]
            cam_info.p = [
                1.0, 0.0, cam_info.width / 2.0, 0.0,
                0.0, 1.0, cam_info.height / 2.0, 0.0,
                0.0, 0.0, 1.0, 0.0
            ]

            # --- Publish both ---
            self.pub_image.publish(img_msg)
            self.pub_info.publish(cam_info)

        except Exception as e:
            self.get_logger().error(f'Error processing images: {e}')


def main(args=None):
    rclpy.init(args=args)
    node = MultiCameraStitcher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

A.12 Attach Detach Key Control

The code refers to attach detach key control node in the package. The file path is as follows inside the package.
src>abb_flexley_description>abb_flexley_description>attach_detach_key_control.py

```
#!/usr/bin/env python3
import rclpy
```

```python
from rclpy.node import Node
from std_msgs.msg import Empty
import sys
import termios
import tty
import subprocess
import re
import math
import time
import threading

class AttachDetachNode(Node):
    def __init__(self):
        super().__init__('attach_detach_key_control')
        self.attach_pub = self.create_publisher(Empty, '/attach', 10)
        self.detach_pub = self.create_publisher(Empty, '/detach', 10)
        self.get_logger().info("Attach/Detach node started. Press 't'
        ↪  to attach, 'r' to detach, 'q' to quit.")

        # XY proximity threshold (meters)
        self.proximity_threshold_xy = 0.55

        # Debounce: require this many consecutive checks inside
        ↪  threshold
        self.sustain_count_required = 3
        self._sustain_counter = 0
        self.models_close = False

        # Optional yaw alignment (set to None to ignore)
        self.yaw_tolerance = None

        # Flag to stop threads on exit
        self._running = True

    def get_key(self):
        """Blocking single-key read from stdin."""
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            key = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return key

    def get_model_pose(self, model_name):
        """Return (x, y, z, roll, pitch, yaw) from `ign model -m <name>
        ↪  --pose`."""
        try:
```

```
            cmd = ["ign", "model", "-m", model_name, "--pose"]
            result = subprocess.run(cmd, capture_output=True,
            ↪  text=True, timeout=3.0)
            lines = result.stdout.splitlines()
            xyz, rpy = None, None
            for line in lines:
                line = line.strip()
                if line.startswith('[') and xyz is None:
                    xyz = [float(v) for v in line.strip('[]').split()]
                elif line.startswith('[') and xyz is not None:
                    rpy = [float(v) for v in line.strip('[]').split()]
                    break
            if xyz is not None and rpy is not None:
                return (xyz[0], xyz[1], xyz[2], rpy[0], rpy[1], rpy[2])
        except Exception as e:
            self.get_logger().warn(f"Failed to get pose for
            ↪  {model_name}: {e}")
        return None

    def update_proximity(self):
        """Update planar proximity using only X and Y."""
        tpose = self.get_model_pose("trolley")
        upose = self.get_model_pose("flexley_tug")

        if not tpose or not upose:
            # couldn't read poses -> treat as FAR
            if self.models_close:
                self.get_logger().info("Models are now FAR (pose read
                ↪  failed).")
            self.models_close = False
            self._sustain_counter = 0
            return

        dx = tpose[0] - upose[0]
        dy = tpose[1] - upose[1]
        dist_sq = dx*dx + dy*dy
        thresh_sq = self.proximity_threshold_xy ** 2

        planar_ok = dist_sq <= thresh_sq

        yaw_ok = True
        if self.yaw_tolerance is not None:
            tyaw = tpose[5]
            uyaw = upose[5]
            diff = (tyaw - uyaw + math.pi) % (2*math.pi) - math.pi
            yaw_ok = abs(diff) <= self.yaw_tolerance

        if planar_ok and yaw_ok:
            self._sustain_counter += 1
```

```
        else:
            self._sustain_counter = 0

        prev_state = self.models_close
        if self._sustain_counter >= self.sustain_count_required:
            self.models_close = True
        else:
            self.models_close = False

        # Only print message when state changes
        if prev_state != self.models_close:
            state_str = "NEAR" if self.models_close else "FAR"
            planar_dist = math.sqrt(dist_sq)
            self.get_logger().info(f"Models are {state_str} (XY
            ↪  distance = {planar_dist:.3f} m).")

    def proximity_loop(self):
        """Thread to continuously check model proximity."""
        while self._running and rclpy.ok():
            self.update_proximity()
            time.sleep(0.2)

    def key_loop(self):
        """Thread to handle key input."""
        while self._running and rclpy.ok():
            key = self.get_key()
            if key == 't':
                if self.models_close:
                    self.attach_pub.publish(Empty())
                    self.get_logger().info(" Sent /attach message.")
                else:
                    self.get_logger().warning(" Models too far apart
                    ↪  cannot attach.")
            elif key == 'r':
                self.detach_pub.publish(Empty())
                self.get_logger().info(" Sent /detach message.")
            elif key == 'q':
                self.get_logger().info("Exiting...")
                self._running = False
                break

    def run(self):
        # Start threads for proximity checking and key handling
        prox_thread = threading.Thread(target=self.proximity_loop,
        ↪  daemon=True)
        key_thread = threading.Thread(target=self.key_loop,
        ↪  daemon=True)
        prox_thread.start()
        key_thread.start()
```

```
        # Wait for key thread to finish
        key_thread.join()
        self._running = False  # stop proximity thread

def main(args=None):
    rclpy.init(args=args)
    node = AttachDetachNode()
    try:
        node.run()
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A.13 Launch File

The code refers to gazebo launch file in the package. The file path is as follows inside the package.
src>abb_flexley_description>launch>gazebo.launch.py

```
from launch import LaunchDescription
from launch.actions import ExecuteProcess, SetEnvironmentVariable
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
import os
import xacro
from launch.actions import TimerAction
from launch.actions import ExecuteProcess, OpaqueFunction
import subprocess
import time


def wait_and_publish_detach(context):
    print(" Waiting for Flexley Tug model to load in Gazebo...")

    # Wait for model to exist in Gazebo
    while True:
        result = subprocess.run(
            ['ign', 'topic', '-l'],
            capture_output=True, text=True
        )
        if '/model/flexley_tug' in result.stdout:
            print("Model 'flexley_tug' detected in Gazebo!")
```

```
            # Now also ensure /detach topic exists
            while True:
                result = subprocess.run(['ros2', 'topic', 'list'],
                ↪  capture_output=True, text=True)
                if '/detach' in result.stdout:
                    print(" /detach topic found! Sending detach
                    ↪  command...")
                    subprocess.run(['ros2', 'topic', 'pub', '--once',
                    ↪  '/detach', 'std_msgs/msg/Empty', '{}'])
                    print(" Detach message published.")
                    return []
                time.sleep(1)
        time.sleep(1)


def generate_launch_description():
    # Paths
    package_name = 'abb_flexley_description'
    pkg_share = get_package_share_directory(package_name)

    world_path = os.path.join(pkg_share, 'worlds', 'warehouse.sdf')
    robot_sdf_path = os.path.join(pkg_share, 'urdf', 'Flexley_Tug.sdf')
    model_path = pkg_share

    urdf_file = os.path.join(
        pkg_share,
        'urdf',
        'Flexley_Tug.urdf'
    )

    robot_description_config = xacro.process_file(urdf_file)
    robot_description = {'robot_description':
    ↪  robot_description_config.toxml()}


    # Set required environment variables
    set_ign_path = SetEnvironmentVariable(
        name='IGN_GAZEBO_RESOURCE_PATH',
        value=model_path + '/models:' + model_path + '/worlds'
    )
    set_file_path = SetEnvironmentVariable(
        name='IGN_FILE_PATH',
        value=model_path + '/models:' + model_path + '/worlds'
    )

    # Launch Ignition Gazebo with the world
    launch_gazebo = ExecuteProcess(
        cmd=['ign', 'gazebo','-r' , world_path],
        output='screen'
```

```
)

# Spawn robot model
spawn_model = ExecuteProcess(
    cmd=[
        'ros2', 'run', 'ros_gz_sim','create',
        '-file', robot_sdf_path,
        '-name', 'flexley_tug',
        '-x', '-2.5', '-y', '0', '-z', '0.1'
    ],
    output='screen'
)



# Sensor bridges...
imu_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/imu_scan@sensor_msgs/msg/Imu[ignition.msgs.IMU'],
    remappings=[('/imu_scan', '/imu_scan')],
    parameters=[{'use_sim_time': False}],
    output='screen'
)

front_lidar_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/lidar_front@sensor_msgs/msg/LaserScan[ignition.ms
    ↪  gs.LaserScan'],
    remappings=[('/lidar_front', '/lidar_front')],
    parameters=[{'use_sim_time': True}],
    output='screen'
)

back_lidar_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/lidar_back@sensor_msgs/msg/LaserScan[ignition.msg
    ↪  s.LaserScan'],
    remappings=[('/lidar_back', '/lidar_back')],
    parameters=[{'use_sim_time': True}],
    output='screen'
)

front_camera_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/front_cam@sensor_msgs/msg/Image[ignition.msgs.Ima
    ↪  ge'],
```

```
        remappings=[('/front_cam', '/camera_front')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    back_camera_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/back_cam@sensor_msgs/msg/Image[ignition.msgs.Imag
        ↪  e'],
        remappings=[('/back_cam', '/camera_back')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    front_right_camera_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/front_right_cam@sensor_msgs/msg/Image[ignition.ms
        ↪  gs.Image'],
        remappings=[('/front_right_cam', '/camera_front_right')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    front_left_camera_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/front_left_cam@sensor_msgs/msg/Image[ignition.msg
        ↪  s.Image'],
        remappings=[('/front_left_cam', '/camera_front_left')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    back_right_camera_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/back_right_cam@sensor_msgs/msg/Image[ignition.msg
        ↪  s.Image'],
        remappings=[('/back_right_cam', '/camera_back_right')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    back_left_camera_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/back_left_cam@sensor_msgs/msg/Image[ignition.msgs
        ↪  .Image'],
```

```
        remappings=[('/back_left_cam', '/camera_back_left')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )


    diff_drive_cmd_vel_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/cmd_vel@geometry_msgs/msg/Twist@ignition.msgs.Twi
        ↪  st'],
        remappings=[('/cmd_vel', '/cmd_vel')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    diff_drive_odom_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/odom@nav_msgs/msg/Odometry[ignition.msgs.Odometry
        ↪  '],
        remappings=[('/odom', '/odom')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    joint_state_pub_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/joint_states@sensor_msgs/msg/JointState[ignition.
        ↪  msgs.Model'],
        remappings=[('/joint_states', '/joint_states')],
        parameters=[{'use_sim_time': True}],
        output='screen'
    )

    joint_state_publisher_gui = Node(
        package='joint_state_publisher_gui',
        executable='joint_state_publisher_gui',
        name='joint_state_publisher_gui',
        arguments=[urdf_file],
        parameters=[{'use_sim_time': True}],
        output=['screen']
    )

    robot_state_publisher = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='robot_state_publisher',
```

```
        output='both',
        parameters=[robot_description]
    )

    tf_bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/tf@tf2_msgs/msg/TFMessage[ignition.msgs.Pose_V'],
        remappings=[('/tf', '/tf')],
        parameters=[{'use_sim_time': True}],
        output='screen'
)

    detachable_state_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/detachable_joint_state@std_msgs/msg/String[ignition.m
    ↪  sgs.StringMsg'],
    remappings=[('/detachable_joint_state',
    ↪  '/detachable_joint/state')],
    parameters=[{'use_sim_time': True}],
    output='screen'
)

    detach_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/detach@std_msgs/msg/Empty]ignition.msgs.Empty'],
    remappings=[('/detach', '/detach')],
    parameters=[{'use_sim_time': True}],
    output='screen'
)

    attach_bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/attach@std_msgs/msg/Empty]ignition.msgs.Empty'],
    remappings=[('/attach', '/attach')],
    parameters=[{'use_sim_time': True}],
    output='screen'
)



    wait_for_detach_and_publish = TimerAction(
    period=5.0,
    actions=[OpaqueFunction(function=wait_and_publish_detach)]
)
    return LaunchDescription([
```

```
        set_ign_path,
        set_file_path,
        launch_gazebo,
        spawn_model,
        imu_bridge,
        front_lidar_bridge,
        back_lidar_bridge,
        diff_drive_cmd_vel_bridge,
        diff_drive_odom_bridge,
        joint_state_pub_bridge,
        front_camera_bridge,
        back_camera_bridge,
        front_right_camera_bridge,
        front_left_camera_bridge,
        back_right_camera_bridge,
        back_left_camera_bridge,
        joint_state_publisher_gui,
        robot_state_publisher,
        tf_bridge,
        detachable_state_bridge,
        detach_bridge,
        attach_bridge,
        wait_for_detach_and_publish,

    ])
```