

Chapter 4 Simulating Robots Using ROS and Gazebo

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Setelah merancang model 3D sebuah robot, tahap berikutnya adalah mensimulasikannya. Simulasi robot memberikan gambaran tentang bagaimana robot beroperasi dalam lingkungan virtual.

Dalam bab ini, kita akan menggunakan simulator Gazebo (<http://www.gazebo-sim.org/>) untuk mensimulasikan robot dengan Seven Degrees Of Freedom (DOF) arms dan robot beroda mobile.

Gazebo adalah simulator multi-robot untuk simulasi robot kompleks di dalam dan di luar ruangan. Simulator ini memungkinkan simulasi robot kompleks, sensor robot, dan berbagai objek 3D. Gazebo sudah memiliki model simulasi untuk robot, sensor, dan objek 3D populer di repositorinya (https://bitbucket.org/osrf/gazebo_models/). Model-model ini dapat digunakan langsung tanpa harus membuat yang baru.

Gazebo terintegrasi dengan ROS dengan baik berkat antarmuka ROS yang memungkinkan kendali penuh terhadap Gazebo di ROS. Meskipun Gazebo dapat diinstal tanpa ROS, antarmuka ROS-Gazebo harus diinstal untuk berkomunikasi dari ROS ke Gazebo.

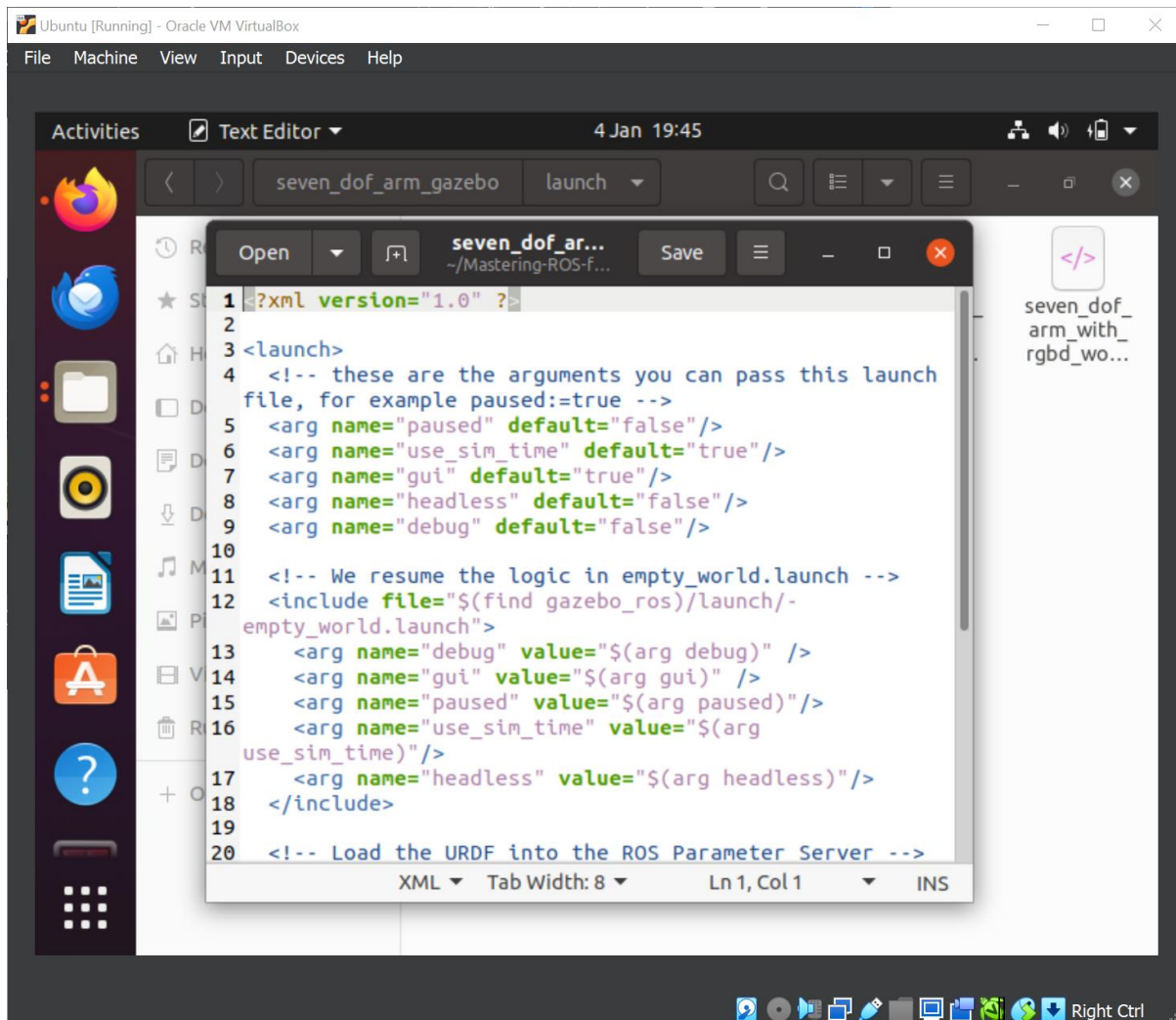
Simulating the robotic arm using Gazebo and ROS

- Setelah merancang lengan robot dengan tujuh derajat kebebasan (DOF), langkah selanjutnya adalah mensimulasikannya di Gazebo menggunakan ROS.
- Instal package-package berikut sebelum memulai dengan Gazebo dan ROS.
- Gazebo memiliki beberapa package seperti gazebo_ros_pkgs, gazebo-msgs, gazebo-plugins, dan gazebo-ros-control, yang dibutuhkan untuk berinteraksi dengan ROS.

Creating the robotic arm simulation model for Gazebo

- Pembuatan model simulasi untuk lengan robotik dilakukan dengan memperbarui deskripsi robot yang sudah ada dan menambahkan parameter simulasi.
- Package **seven_dof_arm_gazebo** dapat dibuat menggunakan perintah **catkin_create_pkg**.
- Model simulasi lengkap terdapat dalam file **seven_dof_arm.xacro** di folder **mastering_ros_robot_description_pkg/urdf/**.

seven_dof_arm_world.launch



```
1 <?xml version="1.0" ?>
2
3 <launch>
4   <!-- these are the arguments you can pass this launch
   file, for example paused:=true -->
5   <arg name="paused" default="false"/>
6   <arg name="use_sim_time" default="true"/>
7   <arg name="gui" default="true"/>
8   <arg name="headless" default="false"/>
9   <arg name="debug" default="false"/>
10
11   <!-- We resume the logic in empty_world.launch -->
12   <include file="$(find gazebo_ros)/launch/empty_world.launch">
13     <arg name="debug" value="$(arg debug)" />
14     <arg name="gui" value="$(arg gui)" />
15     <arg name="paused" value="$(arg paused)" />
16     <arg name="use_sim_time" value="$(arg use_sim_time)" />
17     <arg name="headless" value="$(arg headless)" />
18   </include>
19
20   <!-- Load the URDF into the ROS Parameter Server -->
```

Adding colors and textures to the Gazebo robot model

- Tag gazebo di dalam file **.xacro** digunakan untuk memberikan tekstur dan warna pada link-link robot.
- Setiap tag gazebo merujuk pada link tertentu dari model robot.

Adding transmission tags to actuate the model

- Tag **<transmission>** digunakan untuk mengaktifkan robot menggunakan kontroler ROS.
- Macro **transmission_block** digunakan untuk mendefinisikan elemen **<transmission>** dan menghubungkannya dengan joint (joint) dan aktuator.

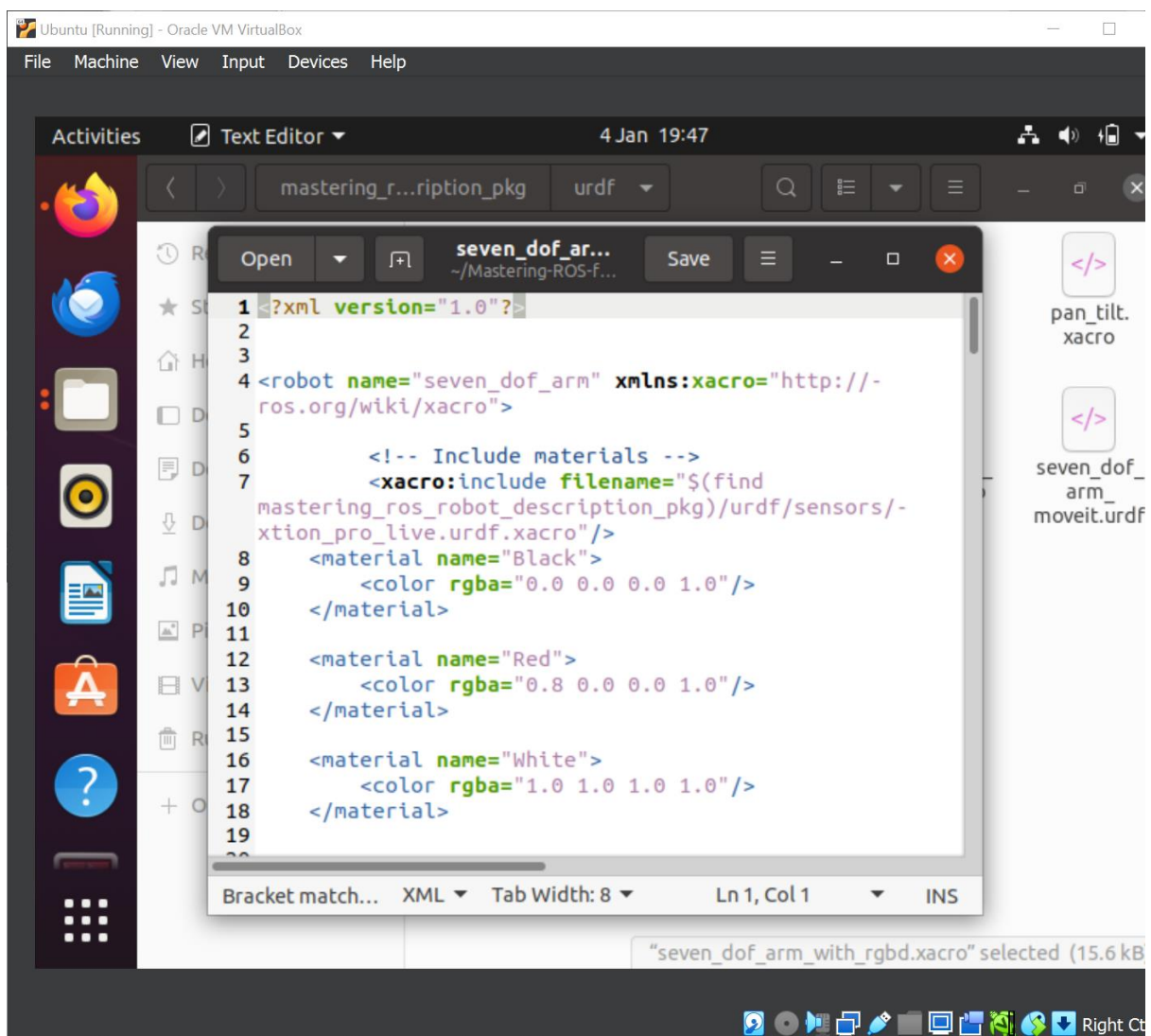
Adding the gazebo_ros_control plugin

- Plugin ini diperlukan untuk mengurai tag-tag transmisi dan menetapkan antarmuka perangkat keras yang sesuai serta manajemen kontrol.
- Ditambahkan ke dalam file **.xacro** dengan konfigurasi berupa namespace robot dan interface perangkat keras default.

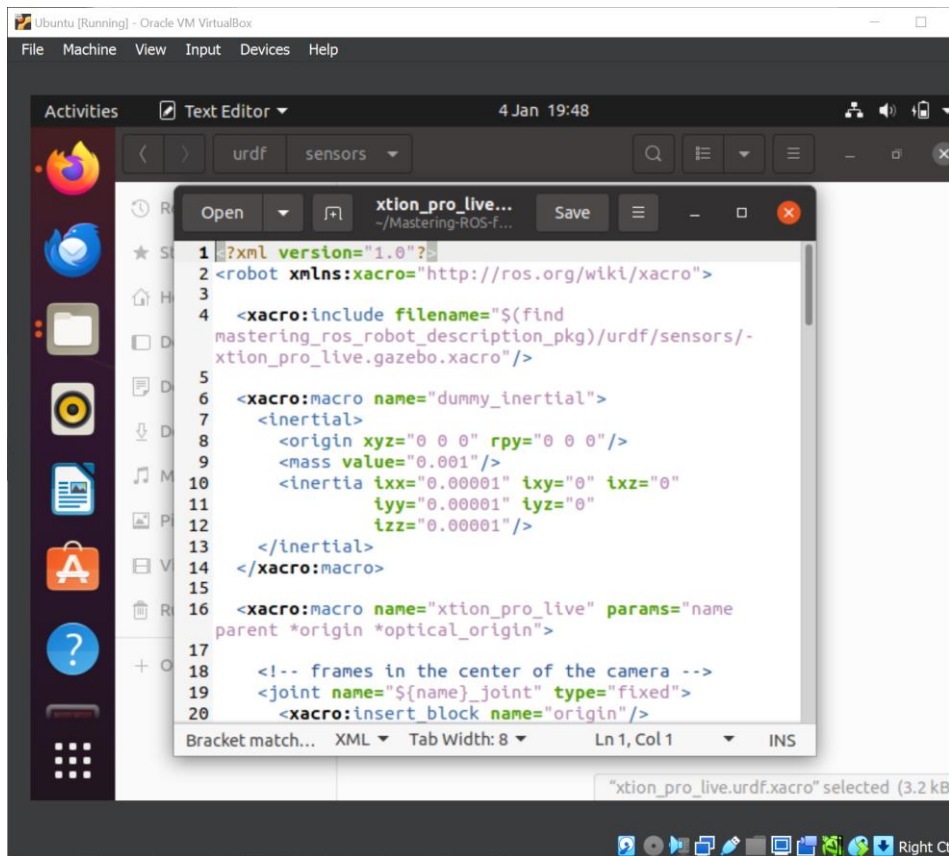
Adding a 3D vision sensor to Gazebo

- Robot di Gazebo dapat memiliki sensor yang mensimulasikan pergerakan dan fisika robot serta berbagai sensor.
- Untuk menambahkan sensor penglihatan 3D (seperti sensor RGB-D) seperti Asus Xtion Pro, kita memodelkannya dan menyertakan definisi Gazebo-ROS plugin dalam file **.xacro**.
- Pengaturan plugin termasuk nama kamera, topik gambar, dan pembaruan.

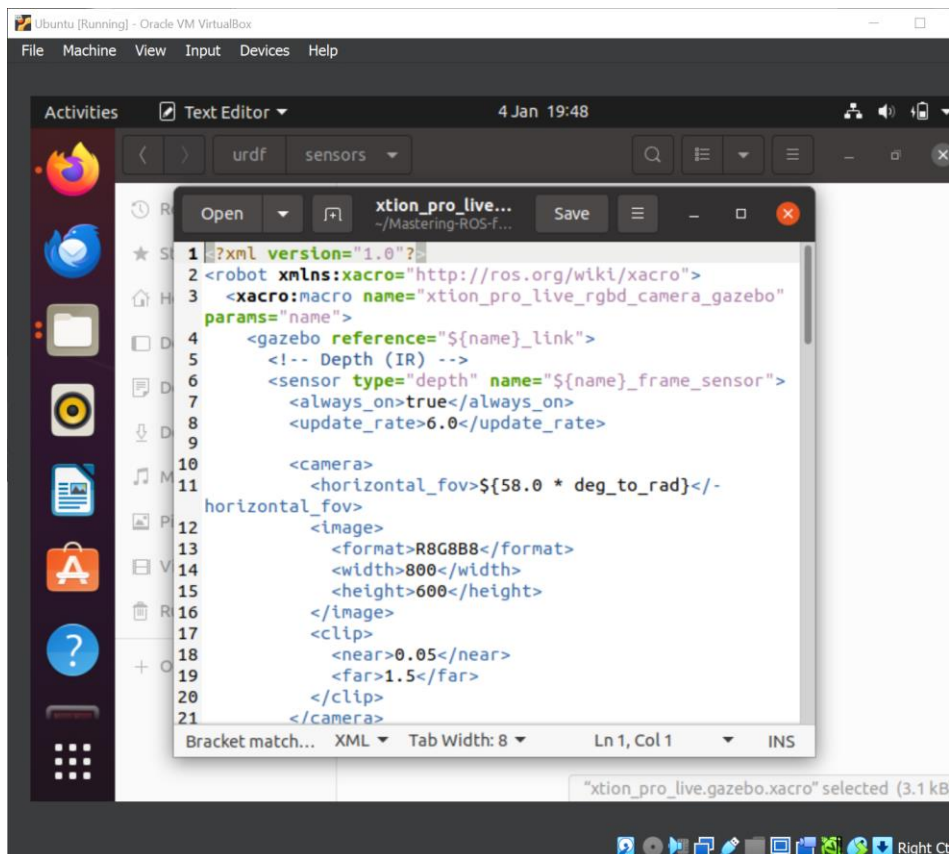
seven_dof_arm_with_rgb.d.xacro



xtion_pro_live.urdf.xacro



xtion_pro_live.gazebo.xacro



Simulating the robotic arm with Xtion Pro

Setelah merancang plugin kamera di Gazebo, kita dapat meluncurkan simulasi lengkap menggunakan perintah `roslaunch seven_dof_arm_gazebo seven_dof_arm_with_rgbd_world.launch`. Dalam simulasi ini, model robot dengan sensor di bagian atas lengannya dapat dilihat di lingkungan Gazebo. Langkah-langkah selanjutnya melibatkan pengujian sensor RGB-D yang mensimulasikan input kamera secara langsung.

Visualizing the 3D sensor data

Setelah mensimulasikan robot dan sensor di Gazebo, kita dapat memeriksa topik-topik yang dihasilkan oleh plugin sensor. Image-view tool dapat digunakan untuk memeriksa gambar RGB, IR, dan kedalaman dari sensor 3D. Selain itu, kita dapat memonitor data awan titik sensor menggunakan perangkat lunak RViz.

Moving the robot joints using ROS controllers in Gazebo

Bagian ini membahas cara menggerakkan setiap joint robot di lingkungan simulasi Gazebo menggunakan kontroler ROS. Kontroler ROS membutuhkan antarmuka perangkat keras yang sesuai dengan parameter yang ditentukan dalam tag transmisi. Kontroler ROS ini menyediakan mekanisme umpan balik untuk menerima titik set dan mengontrol keluaran berdasarkan umpan balik dari aktuator.

Understanding the ros_control packages

Package `ros_control` terdiri dari berbagai modul dan alat yang dapat digunakan oleh kontroler. Ini mencakup `control_toolbox`, `controller_interface`, `controller_manager`, `controller_manager_msgs`, `hardware_interface`, dan `transmission_interface`. Setiap package menyumbang pada fungsionalitas kontroler, manajemen kontroler, dan antarmuka perangkat keras di ROS.

Different types of ROS controllers and hardware interfaces

Kontroler ROS seperti `joint_position_controller`, `joint_state_controller`, dan `joint_effort_controller` memberikan kemampuan kontrol yang berbeda pada robot. Sementara itu, antarmuka perangkat keras seperti `Effort Joint Interface`, `Velocity Joint Interface`, dan `Position Joint Interface` digunakan untuk mengirim perintah ke perangkat keras robot.

How the ROS controller interacts with Gazebo

Interaksi antara kontroler ROS dan Gazebo melibatkan penggunaan antarmuka perangkat keras. Antarmuka perangkat keras bertindak sebagai perantara antara kontroler ROS dan

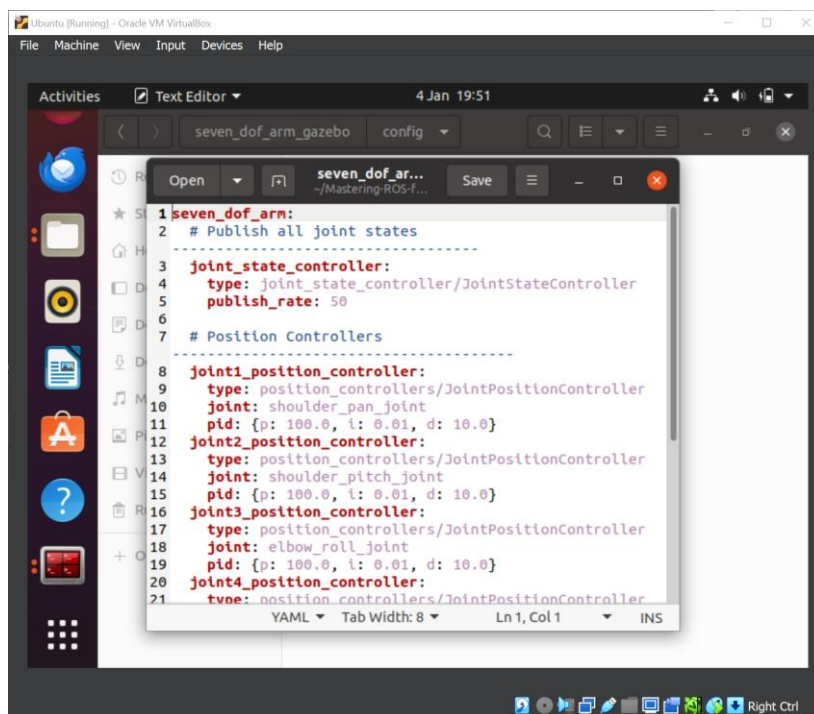
perangkat keras fisik atau simulasi. Informasi mengenai setiap joint dan aktuator disalurkan dari kontroler ke antarmuka perangkat keras, dan nilai-nilai tersebut dapat diteruskan ke simulasi Gazebo atau perangkat keras fisik sesungguhnya.

Interfacing the joint state controllers and joint position controllers with the arm

Menghubungkan kontroler robot dengan setiap joint merupakan tugas yang sederhana. Langkah pertama melibatkan penulisan file konfigurasi untuk dua kontroler. Kontroler status joint akan mempublikasikan status joint lengan, sementara kontroler posisi joint dapat menerima posisi tujuan untuk setiap joint dan dapat menggerakkan masing-masing joint. File konfigurasi untuk kontroler dapat ditemukan di `seven_dof_arm_gazebo_control.yaml`.

Pertama, kita mendefinisikan kontroler status joint yang mempublikasikan status joint lengan pada tingkat 50 Hz. Selanjutnya, kita perlu mendefinisikan kontroler posisi untuk setiap joint dengan parameter PID tertentu. Semua kontroler ini terletak di dalam namespace `seven_dof_arm`.

`seven_dof_arm_gazebo_control.yaml`

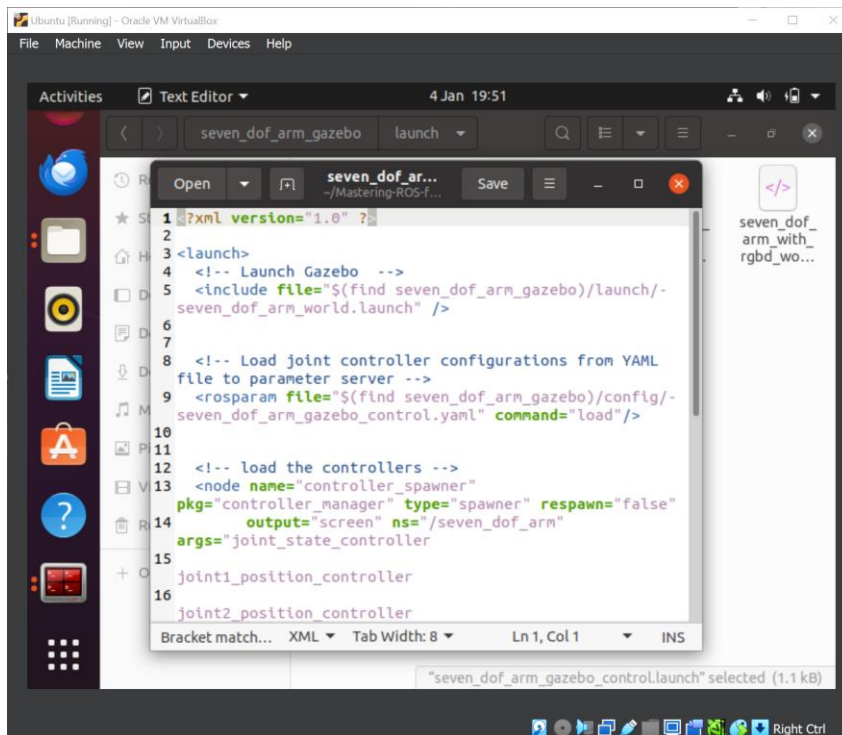


```
1 seven_dof_arm:
2   # Publish all joint states
3   -----
4   joint_state_controller:
5     type: joint_state_controller/JointStateController
6     publish_rate: 50
7   # Position Controllers
8   -----
9   joint1_position_controller:
10    type: position_controllers/JointPositionController
11    joint: shoulder_pan_joint
12    pid: {p: 100.0, i: 0.01, d: 10.0}
13   joint2_position_controller:
14    type: position_controllers/JointPositionController
15    joint: shoulder_pitch_joint
16    pid: {p: 100.0, i: 0.01, d: 10.0}
17   joint3_position_controller:
18    type: position_controllers/JointPositionController
19    joint: elbow_roll_joint
20    pid: {p: 100.0, i: 0.01, d: 10.0}
21   joint4_position_controller:
22    type: position_controllers/JointPositionController
```

Launching the ROS controllers with Gazebo

Setelah konfigurasi kontroler selesai, kita dapat membuat file peluncuran yang memulai semua kontroler bersama dengan simulasi Gazebo. File peluncuran ini mencakup pengaturan simulasi Gazebo, memuat konfigurasi kontroler, dan menjalankan robot state publisher untuk mempublikasikan status dan transformasi joint.

seven_dof_arm_gazebo_control.launch

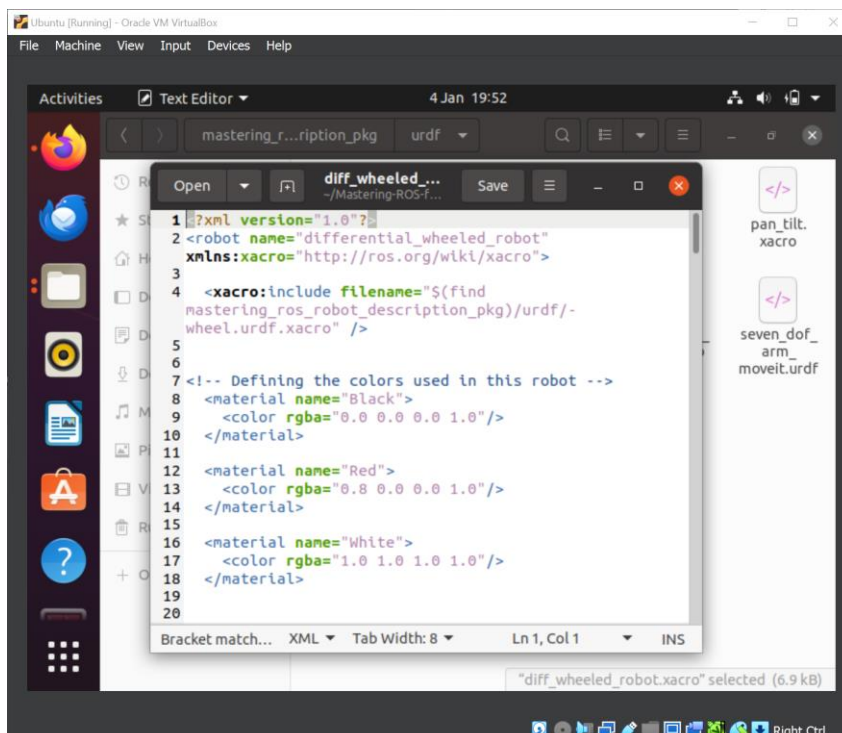


```
1 <?xml version="1.0"?>
2
3 <launch>
4   <!-- Launch Gazebo -->
5   <include file="$(find seven_dof_arm_gazebo)/launch/-
6     seven_dof_arm_world.launch" />
7
8   <!-- Load joint controller configurations from YAML
9     file to parameter server -->
10   <rosparam file="$(find seven_dof_arm_gazebo)/config/-
11     seven_dof_arm_gazebo_control.yaml" command="load"/>
12
13   <!-- load the controllers -->
14   <node name="controller_spawner"
15     pkg="controller_manager" type="spawner" respawn="false"
16     output="screen" ns="/seven_dof_arm"
17     args="joint_state_controller
18     joint1_position_controller
19     joint2_position_controller"/>
20 </launch>
```

Simulating a differential wheeled robot in Gazebo

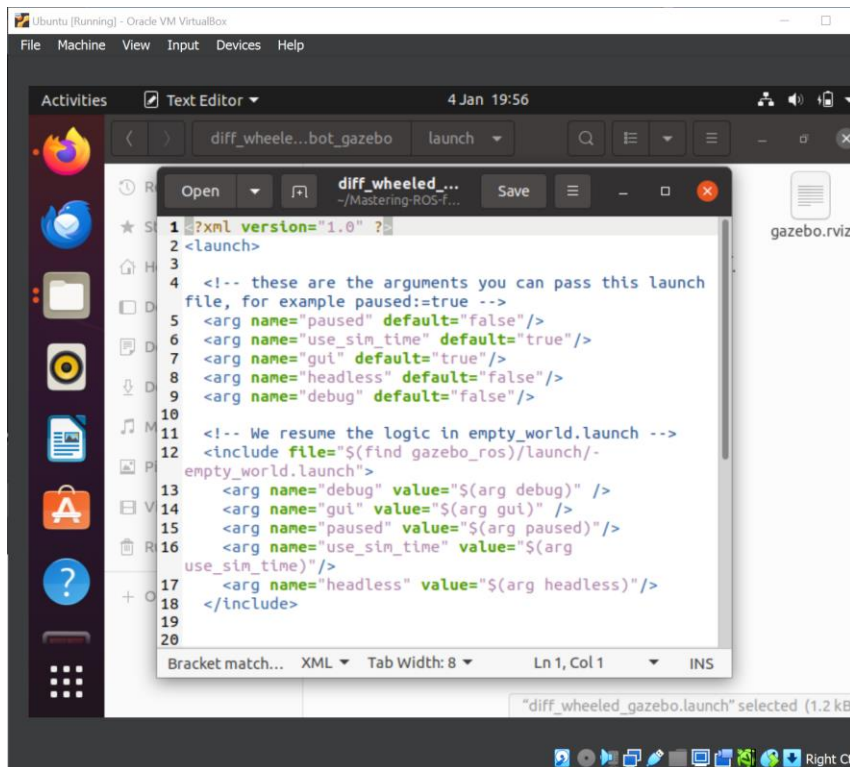
Setelah mensimulasikan lengan robot, langkah selanjutnya adalah menyiapkan simulasi untuk robot beroda diferensial yang telah dirancang sebelumnya. Dengan menggunakan file peluncuran diff_wheeled_gazebo.launch, kita dapat memuat model robot beroda diferensial ke dalam lingkungan Gazebo.

diff_wheeled_robot.xacro



```
1 <?xml version="1.0"?>
2 <robot name="differential_wheeled_robot"
3   xmlns:xacro="http://ros.org/wiki/xacro">
4   <xacro:include filename="$(find
5     mastering_ros_robot_description_pkg)/urdf/-
6     wheel.urdf.xacro" />
7
8   <!-- Defining the colors used in this robot -->
9   <material name="Black">
10     <color rgba="0.0 0.0 0.0 1.0"/>
11   </material>
12   <material name="Red">
13     <color rgba="0.8 0.0 0.0 1.0"/>
14   </material>
15   <material name="White">
16     <color rgba="1.0 1.0 1.0 1.0"/>
17   </material>
18 </robot>
```

diff_wheeled_gazebo.launch



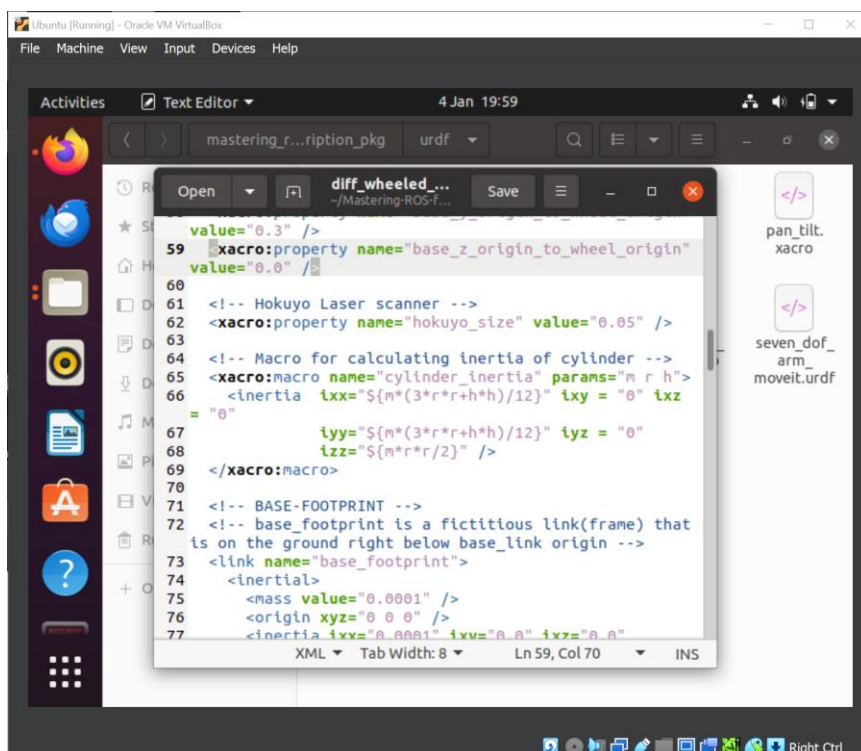
The screenshot shows a text editor window titled "diff_wheeled_gazebo.launch" with the following XML content:

```
1 <?xml version="1.0" ?>
2 <launch>
3
4 <!-- these are the arguments you can pass this launch
   file, for example paused:=true -->
5 <arg name="paused" default="false"/>
6 <arg name="use_sim_time" default="true"/>
7 <arg name="gui" default="true"/>
8 <arg name="headless" default="false"/>
9 <arg name="debug" default="false"/>
10
11 <!-- We resume the logic in empty_world.launch -->
12 <include file="$(find gazebo_ros)/launch/empty_world.launch">
13   <arg name="debug" value="$(arg debug)" />
14   <arg name="gui" value="$(arg gui)" />
15   <arg name="paused" value="$(arg paused)" />
16   <arg name="use_sim_time" value="$(arg use_sim_time)" />
17   <arg name="headless" value="$(arg headless)" />
18 </include>
19
20
```

Adding the laser scanner to Gazebo

Untuk melengkapi simulasi, kita menambahkan pemindai laser ke robot beroda diferensial. Kode tambahan diperlukan untuk mendefinisikan tautan dan sambungan pemindai laser serta konfigurasi plugin Gazebo untuk mensimulasikan pemindai laser.

diff_wheeled_robot.xacro



The screenshot shows a text editor window titled "diff_wheeled_robot.xacro" with the following XML content:

```
59 <xacro:property name="base_z_origin_to_wheel_origin"
   value="0.3" />
60 <xacro:property name="base_z_origin_to_wheel_origin"
   value="0.0" />
61
62 <!-- Hokuyo Laser scanner -->
63 <xacro:property name="hokuyo_size" value="0.05" />
64
65 <!-- Macro for calculating inertia of cylinder -->
66 <xacro:macro name="cylinder_inertia" params="m r h">
67   <inertia ixx="{m*(3*r*r+h*h)/12}" ixy="0" iyz="0"
68     iyy="{m*(3*r*r+h*h)/12}" izz="{m*r*r/2}" />
69 </xacro:macro>
70
71 <!-- BASE-FOOTPRINT -->
72 <!-- base_footprint is a fictitious link(frame) that
   is on the ground right below base_link origin -->
73 <link name="base_footprint">
74   <inertial>
75     <mass value="0.0001" />
76     <origin xyz="0 0 0" />
77     <inertia ixx="0.0001" iyy="0.0001" izz="0.0001" />
78   </inertial>
79 </link>
```

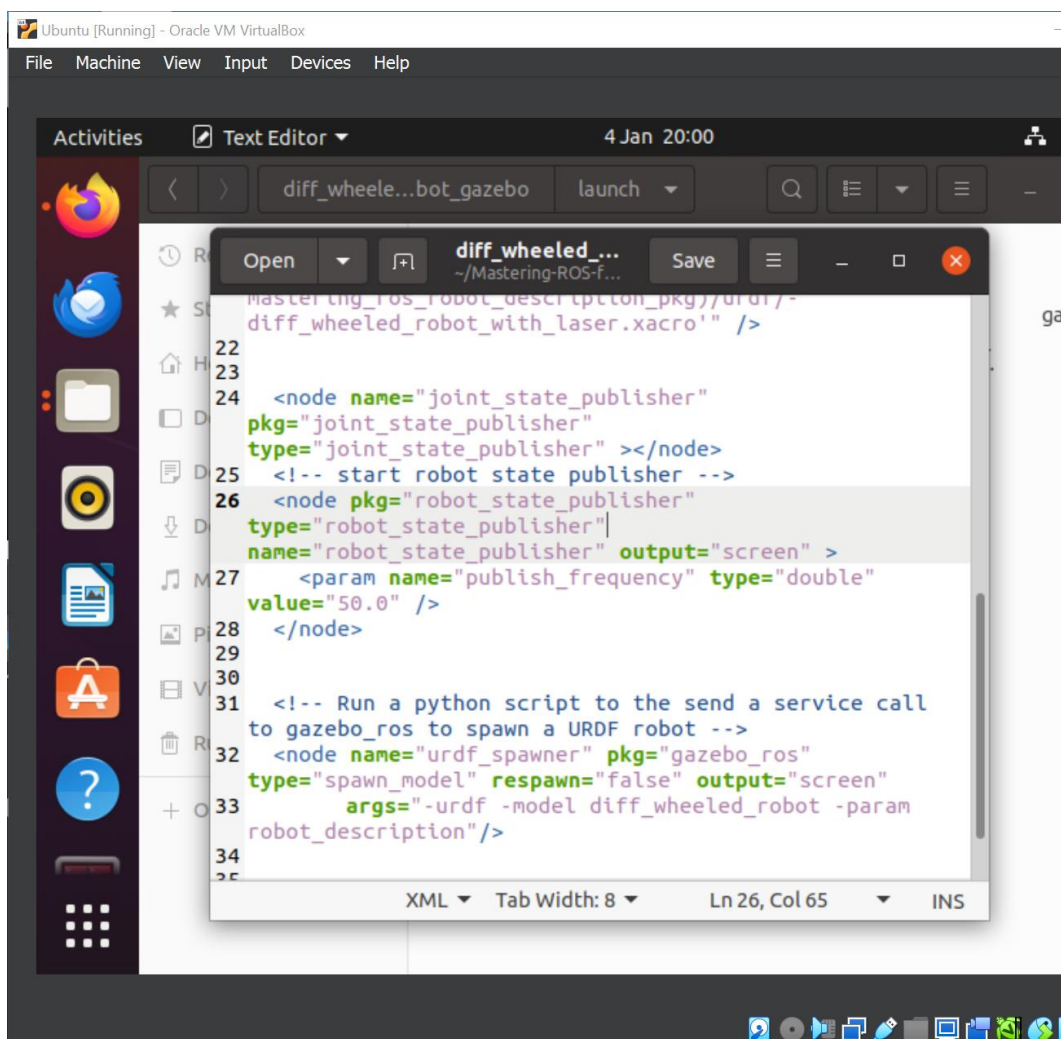

Moving the mobile robot in Gazebo

Robot yang digunakan adalah robot diferensial dengan dua roda dan dua roda penumpu. Untuk membuat robot bergerak di Gazebo, kita memerlukan plugin Gazebo-ROS bernama `libgazebo_ros_diff_drive.so`. Plugin ini memberikan perilaku kendaraan beroda diferensial ke robot kita dengan menerima perintah kecepatan melalui topik `cmd_vel`. Perintah kecepatan dapat dipublikasikan menggunakan node ROS `teleop` untuk menggerakkan robot di lingkungan simulasi Gazebo.

Adding joint state publishers to the launch file

Setelah menambahkan plugin differential drive, kita perlu menambahkan joint state publishers ke dalam file peluncuran yang sudah ada, atau kita dapat membuat file peluncuran baru. File peluncuran terakhir yang baru dapat ditemukan dalam `diff_wheeled_robot_gazebo/launch` dengan nama `diff_wheeled_gazebo_full.launch`. File peluncuran ini berisi penerbit status joint, yang membantu pengembang untuk memvisualisasikan tf di rviz. Berikut adalah baris tambahan yang perlu ditambahkan ke dalam file peluncuran ini untuk penerbitan status joint:

diff_wheeled_gazebo_full.launch

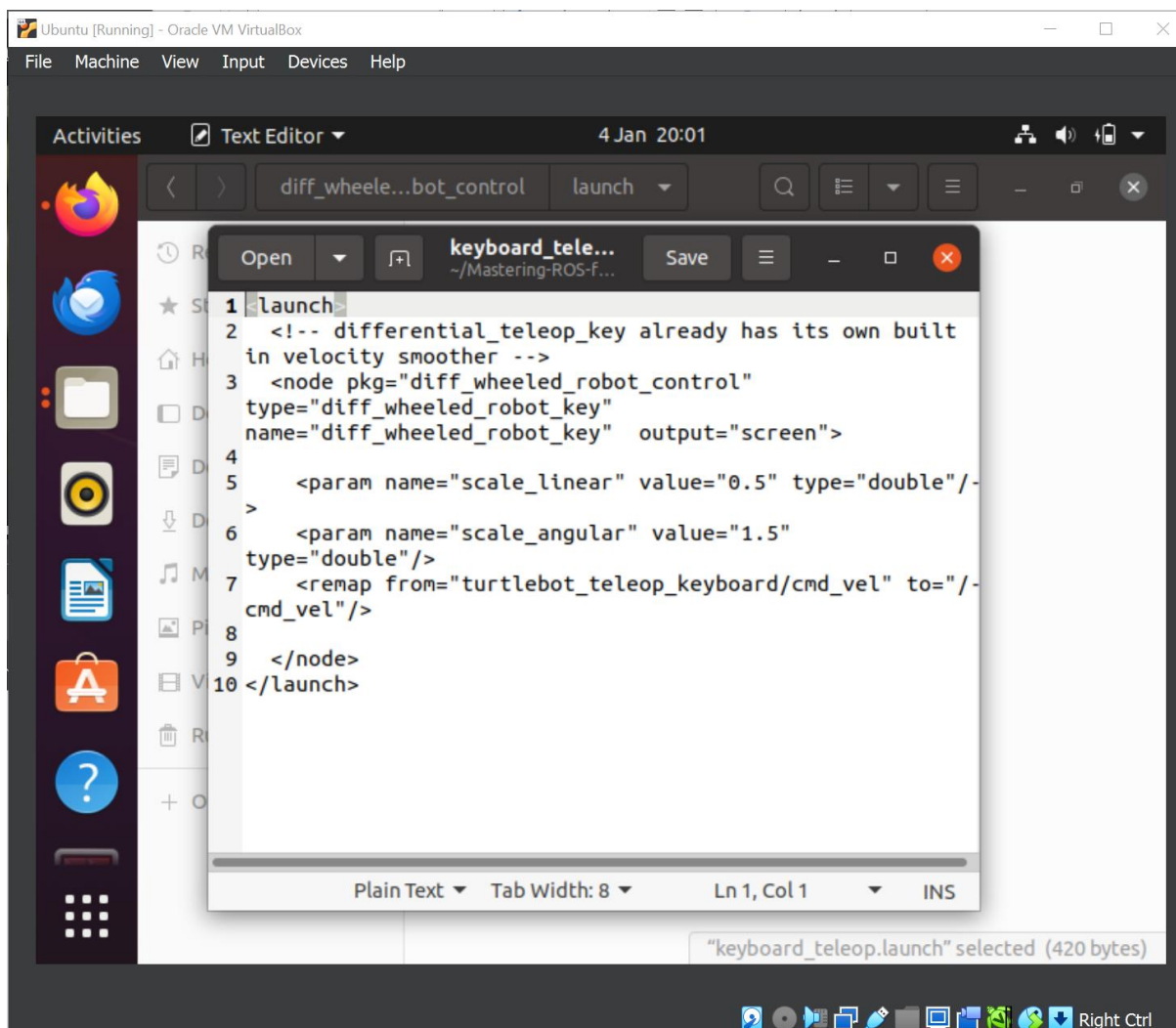


```
22 Mastering_Ros_Robot_Description_pkg/urdf/-
23 diff_wheeled_robot_with_laser.xacro' />
24 <node name="joint_state_publisher"
25   pkg="joint_state_publisher"
26   type="joint_state_publisher" ></node>
27 <!-- start robot state publisher -->
28 <node pkg="robot_state_publisher"
29   type="robot_state_publisher"
30   name="robot_state_publisher" output="screen" >
31   <param name="publish_frequency" type="double"
32     value="50.0" />
33 </node>
34
35 <!-- Run a python script to the send a service call
36 to gazebo_ros to spawn a URDF robot -->
37 <node name="urdf_spawner" pkg="gazebo_ros"
38   type="spawn_model" respawn="false" output="screen"
39   args="-urdf -model diff_wheeled_robot -param
40 robot_description"/>
```

Adding the ROS teleop node

Node teleop ROS mempublikasikan perintah Twist ROS dengan menggunakan input keyboard. Dari node ini, kita dapat menghasilkan kecepatan linear dan angular, dan sudah ada implementasi node teleop standar yang dapat kita gunakan kembali. Teleop diimplementasikan dalam package `diff_wheeled_robot_control`. File skrip `diff_wheeled_robot_key`, yang merupakan node teleop, dapat ditemukan di folder `script`. Untuk menggunakan package ini, Anda perlu mengunduhnya dari repositori Git sebelumnya dan menginstal package `joy_node` jika belum terpasang.

keyboard_teleop.launch



The screenshot shows a text editor window titled "keyboard_teleop..." with the following XML content:

```
1 <?xml version="1.0"?>
2 <!-- differential_teleop_key already has its own built
   in velocity smoother -->
3 <node pkg="diff_wheeled_robot_control"
   type="diff_wheeled_robot_key"
   name="diff_wheeled_robot_key" output="screen">
4
5   <param name="scale_linear" value="0.5" type="double"/>
6   <param name="scale_angular" value="1.5"
   type="double"/>
7   <remap from="turtlebot_teleop_keyboard/cmd_vel" to="/-
   cmd_vel"/>
8
9 </node>
10 </launch>
```

The status bar at the bottom of the editor indicates "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS". A notification at the bottom right says "keyboard_teleop.launch" selected (420 bytes).