Nama : Muhammad Daffa
NIM : 1103201258
Kelas : TK-44-G7

Chapter 1 : Introduction to ROS

Mengapa kita harus menggunakan ROS?

Robot Operating System (ROS) adalah kerangka kerja fleksibel yang menyediakan berbagai alat dan perpustakaan untuk menulis perangkat lunak robot. Ia menawarkan beberapa fitur canggih untuk membantu pengembang dalam tugas-tugas seperti penyampaian pesan, komputasi terdistribusi, penggunaan kembali kode, dan implementasi algoritma canggih untuk aplikasi robotik. Proyek ROS dimulai pada tahun 2007 oleh Morgan Quigley dan pengembangannya dilanjutkan di Willow Garage, sebuah penelitian robotika laboratorium untuk mengembangkan perangkat keras dan perangkat lunak sumber terbuka untuk robot. Tujuan ROS adalah untuk menetapkan cara standar untuk memprogram robot sambil menawarkan perangkat lunak siap pakai komponen yang dapat dengan mudah diintegrasikan dengan aplikasi robotik khusus. Ada banyak alasan memilih ROS sebagai framework pemrograman, dan beberapa di antaranya adalah sebagai
berikut:

- Kemampuan kelas atas:
  ROS hadir dengan fungsionalitas yang siap digunakan. Misalnya, Lokalisasi dan Pemetaan Simultan (SLAM) dan Adaptive Monte Paket Carlo Localization (AMCL) di ROS dapat digunakan untuk memiliki otonomi navigasi di robot seluler, sedangkan paket MoveIt dapat digunakan untuk bergerak perencanaan untuk manipulator robot. Kemampuan ini bisa langsung digunakan di kita perangkat lunak robot tanpa kerumitan. Dalam beberapa kasus, paket-paket ini sudah cukup memiliki tugas robotika inti pada platform yang berbeda. Selain itu, kemampuan ini sangat tinggi dapat dikonfigurasi; kita dapat menyempurnakan masing-masing menggunakan berbagai parameter.

- Banyak alat:
  Ekosistem ROS dilengkapi dengan banyak alat untuk debugging, memvisualisasikan, dan melakukan simulasi. Alat-alatnya, seperti rqt_gui, RViz, dan Gazebo, adalah beberapa alat sumber terbuka terkuat untuk debugging, visualisasi, dan simulasi. Kerangka perangkat lunak yang memiliki alat sebanyak ini sangatlah jarang.

- Dukungan untuk sensor dan aktuator kelas atas:
  ROS memungkinkan kita menggunakan perangkat yang berbeda driver dan paket antarmuka berbagai sensor dan aktuator dalam robotika. Seperti sensor kelas atas termasuk LIDAR 3D, pemindai laser, sensor kedalaman, aktuator, dan lagi. Kami dapat menghubungkan komponen-komponen ini dengan ROS tanpa kesulitan.

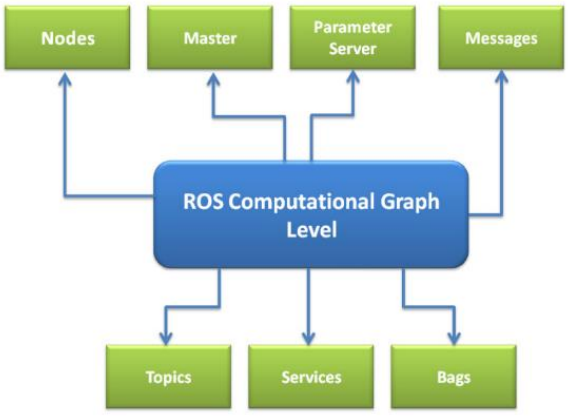
- Pengoperasian antar-platform:
  Middleware penyampaian pesan ROS memungkinkan komunikasi antar program yang berbeda. Di ROS, middleware ini dikenal sebagai node. Node ini dapat diprogram dalam bahasa apa pun yang memiliki klien ROS perpustakaan. Kita dapat menulis node dengan high-haance dalam C++ atau C dan node lainnya dengan Python atau Jawa.

• Modularitas:
Salah satu masalah yang dapat terjadi pada sebagian besar robot yang berdiri sendiri aplikasinya adalah jika salah satu thread dari kode utama crash, seluruh robot aplikasi bisa berhenti. Di ROS, situasinya berbeda; kami menulis berbeda node untuk setiap proses, dan jika salah satu node mengalami crash, sistem masih dapat bekerja.
Memahami sistem file ROS level 5

• Penanganan sumber daya secara bersamaan:
Menangani sumber daya perangkat keras melalui lebih dari dua proses selalu memusingkan. Bayangkan kita ingin mengolah gambar dari a kamera untuk deteksi wajah dan deteksi gerakan; kita dapat menulis kode sebagai entitas tunggal yang dapat melakukan keduanya, atau kita dapat menulis sepotong kode berulir Tunggal konkurensi. Jika kita ingin menambahkan lebih dari dua fitur ke thread, aplikasinya perilaku akan menjadi kompleks dan sulit untuk di-debug. Tapi di ROS, kita bisa mengakses perangkat yang menggunakan topik ROS dari driver ROS. Node ROS berapa pun bisa berlangganan pesan gambar dari driver kamera ROS, dan setiap node bisa mempunyai fungsi yang berbeda. Hal ini dapat mengurangi kompleksitas dalam komputasi dan juga meningkatkan kemampuan debugging seluruh sistem.

Komunitas ROS berkembang sangat pesat, dan terdapat banyak pengguna dan pengembang di seluruh dunia. Sebagian besar perusahaan robotika kelas atas kini mem-porting perangkat lunak mereka ke ROS. Tren ini juga terlihat dalam robotika industri, tempat perusahaan beralih aplikasi robot berpemilik untuk ROS. Sekarang kita tahu mengapa mempelajari ROS itu mudah, kita bisa mulai memperkenalkan intinya konsep. Pada dasarnya ada tiga level dalam ROS: level sistem file, grafik komputasi tingkat, dan tingkat komunitas. Kami akan melihat secara singkat setiap level.

Memahami tingkat grafik perhitungan ROS
Perhitungan dalam ROS dilakukan dengan menggunakan jaringan node ROS. Jaringan komputasi ini disebut grafik komputasi. Konsep utama dalam grafik komputasi adalah ROS node, master, server parameter, pesan, topik, layanan, dan tas. Setiap konsep di grafik dikontribusikan ke grafik ini dengan cara yang berbeda. Paket terkait komunikasi ROS, termasuk perpustakaan klien inti, seperti roscpp dan rospython, dan implementasi konsep, seperti topik, node, parameter, dan layanan, disertakan dalam tumpukan yang disebut ros_comm (http://wiki.ros.org/ros_comm). Tumpukan ini juga terdiri dari alat-alat seperti rostopic, rosparam, rosservice, dan rosnode untuk introspeksi konsep-konsep sebelumnya. Tumpukan ros_comm berisi paket middleware komunikasi ROS, dan inipaket secara kolektif disebut lapisan grafik ROS:



Distribusi ROS

Pembaruan ROS dirilis dengan distribusi ROS baru. Distribusi ROS yang baru adalah terdiri dari versi terbaru dari perangkat lunak intinya dan serangkaian ROS baru/yang diperbarui paket. ROS mengikuti siklus rilis yang sama dengan distribusi Linux Ubuntu: baru versi ROS dirilis setiap 6 bulan. Biasanya, untuk setiap versi Ubuntu LTS, sebuah Versi LTS dari ROS dirilis. Long Term Support (LTS) dan artinya dirilis perangkat lunak akan dipertahankan untuk waktu yang lama (5 tahun untuk ROS dan Ubuntu):

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)

Chapter 2 : Getting Started with ROS Programming

Membuat paket ROS

Paket ROS adalah unit dasar program ROS. Kita dapat membuat paket ROS, build itu, dan merilisnya ke publik. Distribusi ROS yang kami gunakan saat ini adalah Noetic Ninjamys. Kami menggunakan sistem build catkin untuk membangun paket ROS. Sebuah bangunan sistem bertanggung jawab untuk menghasilkan target (dapat dieksekusi/perpustakaan) dari sumber tekstual kode yang dapat digunakan oleh pengguna akhir. Di distribusi lama, seperti Electric dan Fuerte, rosbuild adalah sistem pembangunan. Karena berbagai kekurangan rosbuild, catkin muncul. Ini juga memungkinkan kami untuk mendekatkan sistem kompilasi ROS Pembuatan Lintas Platform (CMake). Ini mempunyai banyak keuntungan, seperti porting paket ke OS lain, seperti Windows. Jika OS mendukung CMake dan Python, berbasis catkin paket dapat di-porting ke sana.

Persyaratan pertama untuk bekerja dengan paket ROS adalah membuat catkin ROS ruang kerja. Setelah menginstal ROS, kita dapat membuat dan membangun catkinworkspace Bernama catkin_ws:

```
mkdir -p ~/catkin_ws/src
```

Untuk mengkompilasi ruang kerja ini, kita harus mencari sumber lingkungan ROS untuk mendapatkan akses ke ROS
fungsi:

```
source /opt/ros/noetic/setup.bash
```

Beralih ke folder source src yang kita buat sebelumnya:

```
cd ~/catkin_ws/src
```

Inisialisasi ruang kerja catkin baru:

```
catkin_init_workspace
```

Kita dapat membangun ruang kerja meskipun tidak ada paket. Kita dapat menggunakan yang berikut ini perintah untuk beralih ke folder ruang kerja:

```
cd ~/catkin_ws
```

Perintah catkin_make akan membangun ruang kerja berikut:

```
catkin_make
```

Perintah ini akan membuat direktori devel dan build di ruang kerja catkin Anda. File setup yang berbeda terletak di dalam folder devel. Untuk menambahkan ROS yang dibuat ruang kerja ke lingkungan ROS, kita harus mengambil salah satu file ini. Selain itu, kami dapat mengambil file setup ruang kerja ini setiap kali sesi bash baru dimulai dengan perintah berikut:

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Setelah mengatur ruang kerja catkin, kita dapat membuat paket kita sendiri yang memiliki sampel node untuk mendemonstrasikan cara kerja topik, pesan, layanan, dan actionlib ROS. Perhatikan bahwa jika Anda belum menyiapkan ruang kerja dengan benar, Anda tidak akan dapat menggunakannya perintah ROS. Perintah catkin_create_pkg adalah cara paling nyaman untuk melakukannya membuat paket ROS.

Perintah ini digunakan untuk membuat paket yang akan kita tuju untuk membuat demo berbagai konsep ROS.

Beralih ke folder src ruang kerja catkin dan buat paket dengan menggunakan perintah berikut:
catkin_create_pkg package_name [dependency1] [dependency2]

Folder kode sumber: Semua paket ROS, baik dibuat dari awal atau diunduh dari repositori kode lain, harus ditempatkan di folder src ruang kerja ROS; jika tidak, data tersebut tidak akan dikenali oleh sistem ROS dan dikompilasi.

Berikut adalah perintah untuk membuat contoh paket ROS:
catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs
actionlib actionlib_msgs

Membuat node ROS
Node pertama yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini akan mempublikasikan nilai integer pada topik yang disebut /numbers. Salin kode saat ini ke yang baru paket atau gunakan file yang ada dari repositori kode buku ini.
Berikut kode lengkapnya:

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
int main(int argc, char **argv) {
    ros::init(argc, argv,"demo_topic_publisher");
    ros::NodeHandle node_obj;
    ros::Publisher number_publisher = node_obj.advertise<std_
msgs::Int32>("/numbers", 10);
    ros::Rate loop_rate(10);
    int number_count = 0;
    while ( ros::ok() ) {
        std_msgs::Int32 msg;
        msg.data = number_count;
        ROS_INFO("%d",msg.data);
        number_publisher.publish(msg);
        loop_rate.sleep();
        ++number_count;
    }
    return 0;
}
```

Membangun node
Kita harus mengedit file CMakeLists.txt dalam paket untuk mengkompilasi dan membangun sumbernya kode. Navigasikan ke mastering_ros_demo_pkg untuk melihat CMakeLists.txt yang ada mengajukan.
Cuplikan kode berikut dalam file ini bertanggung jawab untuk membangun dua node ini:

```
include_directories(
include
${catkin_INCLUDE_DIRS}
)
#This will create executables of the nodes
add_executable(demo_topic_publisher src/demo_topic_publisher.
cpp)
add_executable(demo_topic_subscriber src/demo_topic_subscriber.
cpp)

#This will link executables to the appropriate libraries
target_link_libraries(demo_topic_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_topic_subscriber ${catkin_
LIBRARIES})
```

Membuat file peluncuran

File peluncuran di ROS sangat berguna untuk meluncurkan lebih dari satu node. Dalam contoh sebelumnya, kita melihat maksimal dua node ROS, tapi bayangkan sebuah skenario di dalamnya yang mana kita harus meluncurkan 10 atau 20 node untuk sebuah robot. Akan sulit jika kita harus melakukannya jalankan setiap node di terminal satu per satu. Sebagai gantinya, kita dapat menulis semua node di dalam sebuah File berbasis XML disebut file peluncuran dan, menggunakan perintah bernama roslaunch, kami menguraikannya file dan luncurkan node. Perintah roslaunch akan secara otomatis memulai master ROS dan parameternya server. Jadi, pada dasarnya, tidak perlu memulai perintah roscore dan individu mana pun node; jika kita meluncurkan file tersebut, semua operasi akan dilakukan dalam satu perintah. Perhatikan itu jika Anda memulai sebuah node menggunakan perintah roslaunch, hentikan atau mulai ulang ini perintah akan memiliki efek yang sama seperti memulai ulang roscore.

Mari kita mulai dengan membuat file peluncuran. Beralih ke folder paket dan buat yang baru meluncurkan file bernama demo_topic.launch untuk meluncurkan dua node ROS untuk penerbitan dan berlangganan nilai integer. Kami akan menyimpan file peluncuran di folder peluncuran, yang mana ada di dalam paket:

```
roscd mastering_ros_demo_pkg
mkdir launch
cd launch
gedit demo_topic.launch
```

Setelah membuat file peluncuran demo_topic.launch, kita dapat meluncurkannya menggunakan perintah berikut:

```
roslaunch mastering_ros_demo_pkg demo_topic.launch
```

Kita dapat memeriksa daftar node dengan menggunakan perintah berikut:

```
rosnode list
```

Kita juga dapat melihat pesan log dan men-debug node menggunakan alat GUI yang disebut rqt_ console:

```
rqt_console
```

Chapter 3 : Bekerja dengan ROS untuk Pemodelan 3D

Creating the ROS package for the robot description

Sebelum membuat file URDF untuk robot, mari kita buat paket ROS di catkin ruang kerja agar model robot tetap menggunakan perintah berikut:

```
catkin_create_pkg mastering_ros_robot_description_pkg roscpp tf
geometry_msgs urdf rviz xacro
```

Paket ini terutama bergantung pada paket urdf dan xacro. Jika paket ini punya belum diinstal pada sistem Anda, Anda dapat menginstalnya menggunakan manajer paket:

```
sudo apt-get install ros-noetic-urdf
sudo apt-get install ros-noetic-xacro
```

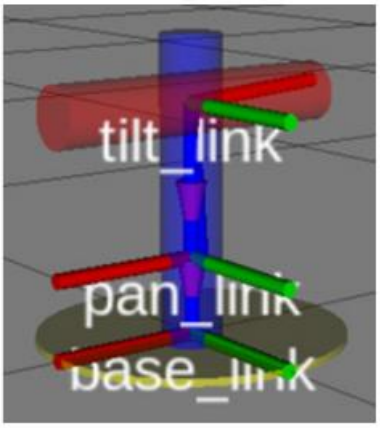
Kita dapat membuat file urdf robot di dalam paket ini dan membuat file peluncurannya menampilkan file urdf yang dibuat di RViz. Paket lengkap tersedia di Git berikut gudang; Anda dapat mengkloning repositori untuk referensi untuk mengimplementasikan paket ini, atau Anda bisa mendapatkan paket dari kode sumber buku:

```
git clone https://github.com/qboticslabs/mastering_ros_3rd_edition.git
cd mastering_ros_robot_description_pkg/
```

Sebelum membuat file URDF robot ini, mari kita buat tiga folder bernama urdf, meshes, dan luncurkan di dalam folder paket. Folder urdf dapat digunakan untuk menyimpan URDF dan file xacro yang akan kita buat. Folder jerat menyimpan jerat yang kita buat perlu disertakan dalam file urdf, dan folder peluncuran menyimpan file peluncuran ROS.

Creating our first URDF model

Setelah mempelajari tentang URDF dan tag pentingnya, kita dapat memulai beberapa pemodelan dasar menggunakan URDF. Mekanisme robot pertama yang akan kami rancang adalah mekanisme pan-and-tilt, seperti terlihat pada diagram berikut. Ada tiga mata rantai dan dua sambungan dalam mekanisme ini. Tautan dasarnya statis, dan sebagainya tautan lain dipasang ke sana. Sambungan pertama dapat bergerak pada porosnya; tautan kedua adalah dipasang pada tautan pertama, dan dapat dimiringkan pada porosnya. Dua sambungan dalam sistem ini adalah tipe berputar:



Mari kita lihat kode URDF dari mekanisme ini. Arahkan ke mastering_direktori ros_robot_description_pkg/urdf dan buka pan_tilt.urdf.

Kita akan mulai dengan mendefinisikan link dasar dari model root:

```
<?xml version="1.0"?>
<robot name="pan_tilt">
    <link name="base_link">
      <visual>
          <geometry>
          <cylinder length="0.01" radius="0.2"/>
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 0"/>
          <material name="yellow">
              <color rgba="1 1 0 1"/>
          </material>
      </visual>
    </link>
```

Kemudian, kita akan mendefinisikan pan_joint untuk menghubungkan base_link dan pan_link:

```
<joint name="pan_joint" type="revolute">
  <parent link="base_link"/>
  <child link="pan_link"/>
  <origin xyz="0 0 0.1"/>
  <axis xyz="0 0 1" />
</joint>
<link name="pan_link">
  <visual>
      <geometry>
      <cylinder length="0.4" radius="0.04"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0.09"/>
      <material name="red">
```

```
            <color rgba="0 0 1 1"/>
        </material>
    </visual>
</link>
```

Demikian pula, kita akan mendefinisikan tilt_joint untuk menghubungkan pan_link dan tilt_link:

```
<joint name="tilt_joint" type="revolute">
    <parent link="pan_link"/>
    <child link="tilt_link"/>
    <origin xyz="0 0 0.2"/>
    <axis xyz="0 1 0" />
</joint>
<link name="tilt_link">
    <visual>
        <geometry>
<cylinder length="0.4" radius="0.04"/>
        </geometry>
        <origin rpy="0 1.5 0" xyz="0 0 0"/>
        <material name="green">
            <color rgba="1 0 0 1"/>
            </material>
        </visual>
    </link>
</robot>
```

Memvisualisasikan model robot 3D di RViz

Setelah mendesain URDF, kita bisa melihatnya di RViz. Kita dapat membuat view_demo. luncurkan file peluncuran dan masukkan kode berikut ke dalam folder peluncuran. Navigasi ke direktori mastering_ros_robot_description_pkg/launch untuk kode:

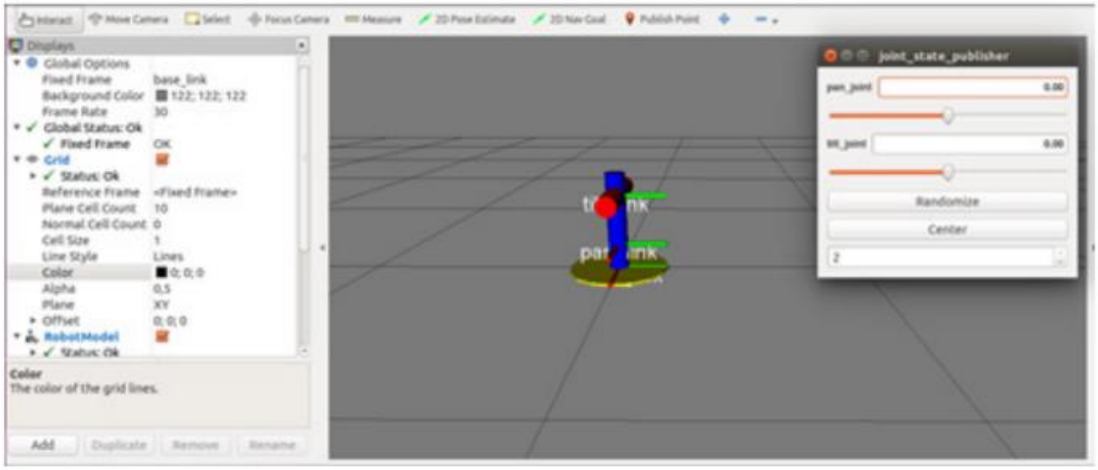
```
<?xml version="1.0" ?>
<launch>
```

```
    <arg name="model" />
    <param name="robot_description" textfile="$(find mastering_
    ros_robot_description_pkg)/urdf/pan_tilt.urdf" />
    <node name="joint_state_publisher_gui" pkg="joint_state_
    publisher_gui" type="joint_state_publisher_gui" />
    <node name="robot_state_publisher" pkg="robot_state_
    publisher" type="robot_state_publisher" />
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
    mastering_ros_robot_description_pkg)/urdf.rviz" required="true"
    />
</launch>
```

Kita dapat meluncurkan model menggunakan perintah berikut:

```
roslaunch mastering_ros_robot_description_pkg view_demo.launch
```

Jika semuanya berfungsi dengan benar, kita akan mendapatkan mekanisme pan-and-tilt di RViz, seperti yang ditunjukkan Di Sini:



Chapter 4 : Simulasi Robot Menggunakan ROS dan Gazebo

Simulasi lengan robot menggunakan Gazebo dan ROS

Pada bab sebelumnya, kami merancang lengan tujuh DOF. Pada bagian ini, kita akan melakukan simulasi robot di Gazebo menggunakan ROS.

Sebelum memulai dengan Gazebo dan ROS, kita harus menginstal paket berikut agar berfungsi dengan Gazebo dan ROS:

```
sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-
gazebo-msgs ros-noetic-gazebo-plugins ros-noetic-gazebo-ros-
control
```

Versi default yang diinstal dari paket Noetic ROS adalah Gazebo 11.x. Kegunaannya masing-masing paketnya adalah sebagai berikut:

- gazebo_ros_pkgs: Berisi wrapper dan alat untuk menghubungkan ROS dengan Gazebo.
- gazebo-msgs: Berisi pesan dan struktur data layanan untuk antarmuka dengan Gazebo dari ROS.
- gazebo-plugins: Ini berisi plugin Gazebo untuk sensor, aktuator, dan sebagainya.
- gazebo-ros-control: Ini berisi pengontrol standar untuk berkomunikasi antara ROS dan Gazebo.

Setelah instalasi, periksa apakah Gazebo sudah terpasang dengan benar menggunakan perintah berikut:

```
roscore & rosrun gazebo_ros gazebo
```

Perintah ini akan membuka Gazebo GUI. Kalau kita punya simulator Gazebo, kita bisa melanjutkan pengembangan model simulasi tujuh lengan DOF untuk Gazebo.

Membuat model simulasi lengan robot untuk Gazebo

Kita dapat membuat model simulasi lengan robot dengan mengupdate robot yang sudah ada deskripsi dengan menambahkan parameter simulasi.

Kita dapat membuat paket yang diperlukan untuk mensimulasikan lengan robot menggunakan perintah berikut:

```
catkin_create_pkg seven_dof_arm_gazebo gazebo_msgs gazebo_
plugins gazebo_ros gazebo_ros_control mastering_ros_robot_
description_pkg
```

Alternatifnya, paket lengkap tersedia di repositori Git berikut; kamu bisa mengkloning repositori untuk referensi untuk mengimplementasikan paket ini, atau Anda bisa mendapatkan paketnya kode sumber buku:

```
git clone https://github.com/PacktPublishing/Mastering-ROS-for-
Robotics-Programming-Third-edition.git
cd Chapter4/seven_dof_arm_gazebo
```

Model simulasi robot selengkapnya dapat Anda lihat di seven_dof_arm.xacro file, ditempatkan di folder mastering_ros_robot_description_pkg/urdf/. File tersebut diisi dengan tag URDF, yang diperlukan untuk simulasi. Kami akan mendefinisikannya bagian tumbukan, inersia, transmisi, sambungan, sambungan, dan Gazebo.

Untuk meluncurkan model simulasi yang ada, kita dapat menggunakan seven_dof_arm_gazebo paket, yang memiliki file peluncuran bernama seven_dof_arm_world.launch. Berkas definisinya adalah sebagai berikut:

```
<launch>
  <!-- these are the arguments you can pass this launch file,
for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

<!-- We resume the logic in empty_world.launch -->
<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="debug" value="$(arg debug)" />
  <arg name="gui" value="$(arg gui)" />
  <arg name="paused" value="$(arg paused)"/>
  <arg name="use_sim_time" value="$(arg use_sim_time)"/>
  <arg name="headless" value="$(arg headless)"/>
</include>

<!-- Load the URDF into the ROS Parameter Server -->
<param name="robot_description" command="$(find xacro)/xacro
```

'$(find mastering_ros_robot_description_pkg)/urdf/seven_dof_

arm.xacro'" />

<!-- Run a python script to the send a service call to

gazebo_ros to spawn a URDF robot -->

<node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"

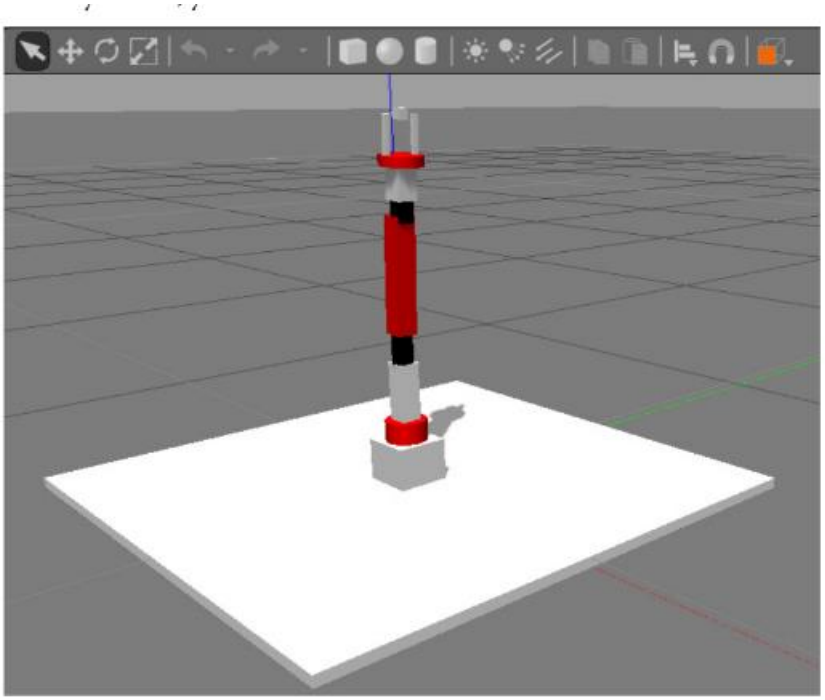
respawn="false" output="screen"

args="-urdf -model seven_dof_arm -param robot_description"/>

</launch>

Luncurkan perintah berikut dan periksa apa yang Anda dapatkan:

roslaunch seven_dof_arm_gazebo seven_dof_arm_world.launch



Chapter 5 : Simulasi Robot Menggunakan ROS, CoppeliaSim, dan Webot

Menyiapkan CoppeliaSim dengan ROS

Langkah awal sebelum mulai bekerja dengan CoppeliaSim adalah menginstalnya pada sistem dan mengonfigurasi lingkungan agar dapat menjembatani komunikasi antara ROS dan simulasi adegan. CoppeliaSim adalah perangkat lunak lintas platform yang dapat diakses di Windows, macOS, dan Linux. Dikembangkan oleh Coppelia Robotics GmbH, simulator ini memiliki lisensi pendidikan dan tersedia secara gratis untuk penggunaan komersial. Anda bisa mengunduh versi terbaru dari simulator ini dari situs resmi Coppelia Robotics di http://www.coppeliarobotics.com/downloads.html. Pastikan untuk memilih versi edu untuk Linux. Pada contoh ini, kita akan menggunakan versi CoppeliaSim 4.2.0.

Setelah proses unduh selesai, ekstrak arsip dengan menggunakan perintah tar vxf CoppeliaSim_Edu_V4_2_0_Ubuntu20_04.tar.xz. Selanjutnya, ganti nama folder hasil ekstraksi ke sesuatu yang lebih intuitif, misalnya: mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim. Agar lebih mudah diakses, atur variabel lingkungan COPPELIASIM_ROOT yang mengarah ke direktori utama CoppeliaSim dengan menjalankan perintah echo "export COPPELIASIM_ROOT=/path/to/CoppeliaSim/folder >> ~/.bashrc".

Setelah konfigurasi selesai, simulator dapat dijalankan dengan menggunakan perintah ./coppeliaSim.sh dari direktori $COPPELIASIM_ROOT. Pastikan juga untuk memulai roscore dengan menjalankan perintah roscore sebelum membuka CoppeliaSim untuk mengaktifkan antarmuka komunikasi ROS.

Mensimulasikan lengan robot menggunakan CoppeliaSim dan ROS

Berikutnya, langkah untuk mensimulasikan lengan robot tujuh derajat kebebasan (DOF) melibatkan impor model ke dalam adegan simulasi CoppeliaSim. Proses ini melibatkan konversi file xacro menjadi format URDF dan penyimpanannya di folder urdf dalam paket csim_demo_pkg. Sebagai contoh, perintah rosrun xacro seven_dof_arm.xacro > /path/to/csim_demo_pkg/urdf/seven_dof_arm.urdf dapat digunakan.

Sementara itu, untuk mengintegrasikan Webots dengan ROS, langkah-langkah serupa harus diikuti. Mulai dari mengotentikasi repositori Cyberbotics hingga menambahkan repositori tersebut ke manajer paket APT, lalu menginstal Webots melalui perintah sudo apt-get install webots. Setelah instalasi selesai, Webots dapat dijalankan menggunakan perintah webots.

Menyiapkan Webot dengan ROS

Anda perlu menginstal paket webots_ros menggunakan APT dengan perintah seperti sudo apt-get install ros-noetic-webots-ros untuk melakukan integrasi Webots-ROS. Setelahnya, Anda dapat membuat sebuah node teleoperasi yang memanfaatkan dependensi webots_ros untuk mengendalikan kecepatan roda robot melalui pesan geometry_msgs::Twist. Semua langkah ini dirancang untuk menyiapkan sistem agar dapat mengintegrasikan simulasi robot dengan lingkungan ROS menggunakan simulator seperti CoppeliaSim atau Webots.