Nama : Muhammad Daffa Satria

NIM : 1103204018

UAS Robotika

Chapter 1: Introduction to ROS

Technical requirements

Dalam bab ini, diperlukan komputer yang menggunakan sistem operasi Ubuntu 20.04 LTS atau distribusi Debian 10 GNU/Linux.

Why should we use ROS?

Sebuah framework yang sangat lentur dan bermanfaat untuk menulis perangkat lunak bagi robot dikenal dengan nama Robot Operating System (ROS). ROS menawarkan sejumlah alat dan perpustakaan yang mempermudah para pengembang dalam menjalankan berbagai tugas seperti pengiriman pesan, komputasi terdistribusi, penggunaan ulang kode, serta menerapkan algoritma tingkat lanjut untuk aplikasi robotika. Proyek ROS dimulai pada tahun 2007 oleh Morgan Quigley dan kemudian dikembangkan lebih lanjut di Willow Garage, sebuah laboratorium yang fokus pada pengembangan perangkat keras dan perangkat lunak sumber terbuka untuk robot. Tujuan utama dari ROS adalah menetapkan standar dalam proses pemrograman robot sambil menyediakan perangkat lunak yang siap pakai untuk diintegrasikan dengan mudah dalam aplikasi robotik spesifik. Ada beberapa alasan untuk memilih ROS sebagai kerangka kerja pemrograman, dan beberapa di antaranya adalah sebagai berikut:

- **Kemampuan Tinggi**: ROS dilengkapi dengan fungsionalitas siap pakai. Contohnya, paket Simultaneous Localization and Mapping (SLAM) dan Adaptive Monte Carlo Localization (AMCL) dalam ROS dapat digunakan untuk navigasi otonom pada robot mobile, sementara paket MoveIt dapat digunakan untuk perencanaan gerakan manipulator robot. Kemampuan-kemampuan ini dapat langsung digunakan dalam perangkat lunak robot tanpa kesulitan. Dalam beberapa kasus, paket-paket ini sudah cukup untuk menjalankan tugas-tugas inti pada berbagai platform robotik. Kemampuan-kemampuan ini juga sangat dapat dikonfigurasi; kita dapat menyesuaikan setiap paket dengan berbagai parameter.
- **Banyaknya Alat**: Ekosistem ROS penuh dengan beragam alat untuk debugging, visualisasi, dan simulasi. Alat-alat seperti rqt_gui, RViz, dan Gazebo adalah beberapa alat sumber terbuka terkuat untuk debugging,

visualisasi, dan simulasi. Kerangka kerja perangkat lunak yang memiliki banyak alat seperti ini sangatlah jarang.

- **Dukungan untuk Sensor dan Aktuator Tinggi**: ROS memungkinkan penggunaan driver perangkat dan paket antarmuka dari berbagai sensor dan aktuator dalam robotika. Sensor-sensor tinggi seperti LIDAR 3D, pemindai laser, sensor kedalaman, aktuator, dan lainnya dapat dihubungkan dengan ROS tanpa kesulitan.
- **Keteroperasian Antar-Platform**: Middleware pertukaran pesan pada ROS memungkinkan komunikasi antara program-program berbeda. Di ROS, middleware ini dikenal sebagai node. Node-node ini dapat diprogram dalam bahasa apapun yang memiliki pustaka klien ROS. Kita dapat menulis node-node berkinerja tinggi dalam C++ atau C dan node lainnya dalam Python atau Java.
- **Modularitas**: Salah satu masalah yang sering muncul dalam aplikasi robotik mandiri adalah jika salah satu benang dari kode utama mengalami kegagalan, seluruh aplikasi robot dapat berhenti. Namun, di ROS, situasinya berbeda; kita menulis node-node yang berbeda untuk setiap proses, dan jika satu node mengalami kegagalan, sistem masih dapat berfungsi.
- **Penanganan Sumber Daya Bersamaan**: Menangani sumber daya keras melalui lebih dari dua proses selalu menjadi masalah. Bayangkan kita ingin memproses gambar dari kamera untuk deteksi wajah dan deteksi gerakan; kita bisa menulis kode sebagai satu entitas yang bisa melakukan keduanya, atau kita bisa menulis kode satu utas untuk keberlangsungan. Namun, di ROS, kita dapat mengakses perangkat menggunakan topik-topik ROS dari driver ROS. Sejumlah node ROS dapat berlangganan pesan gambar dari driver kamera ROS, dan setiap node dapat memiliki fungsionalitas yang berbeda. Hal ini dapat mengurangi kompleksitas dalam komputasi dan juga meningkatkan kemampuan debugging dari keseluruhan sistem.

ROS metapackages

Metapaket ROS merupakan suatu paket khusus yang hanya memerlukan satu berkas, yakni package.xml. Secara esensial, metapaket menggabungkan beberapa paket menjadi satu kesatuan paket yang terstruktur secara keseluruhan. Di dalam file package.xml, metapaket mengandung tag ekspor, seperti yang ditunjukkan di bawah ini:

```
<export>
        <metapackage/>
</export>
```

Pada metapaket, tidak ada dependensi <buildtool_depend> untuk catkin; hanya ada dependensi <run_depend>, yang merupakan paket-paket yang tergabung dalam metapaket tersebut. Tumpukan navigasi ROS adalah contoh bagus dari suatu lokasi yang memuat metapaket. Jika instalasi ROS dan paket navigasinya sudah selesai, kita bisa mencoba menggunakan perintah berikut setelah berpindah ke direktori metapaket navigasi:

```
roscd navigation
```

Selanjutnya, buka file package.xml menggunakan editor teks favorit Anda. Sebagai contoh, kita dapat menggunakan gedit dengan perintah:

```
gedit package.xml
```

ROS distributions

Versi terbaru dari ROS keluar bersamaan dengan distribusi terbaru dari ROS. Distribusi baru ROS ini mencakup pembaruan terkini dari inti perangkat lunak, serta kumpulan ROS yang baru atau yang diperbarui. Siklus rilis ROS mengikuti jadwal yang mirip dengan distribusi Ubuntu Linux: biasanya, versi baru dari ROS dirilis setiap 6 bulan. Secara umum, setiap versi Ubuntu LTS memiliki juga versi ROS LTS yang sejalan. Label 'Dukungan Jangka Panjang' (LTS) menandakan bahwa perangkat lunak yang dirilis akan mendapatkan dukungan untuk jangka waktu yang cukup lama, contohnya 5 tahun, baik untuk ROS maupun 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)

Running the ROS master and the ROS parameter

Sebelum menjalankan node ROS apapun, langkah pertama yang perlu dilakukan adalah memulai ROS master dan parameter server ROS. Proses ini dapat diinisiasi menggunakan perintah bernama roscore, yang secara otomatis memulai beberapa program berikut:

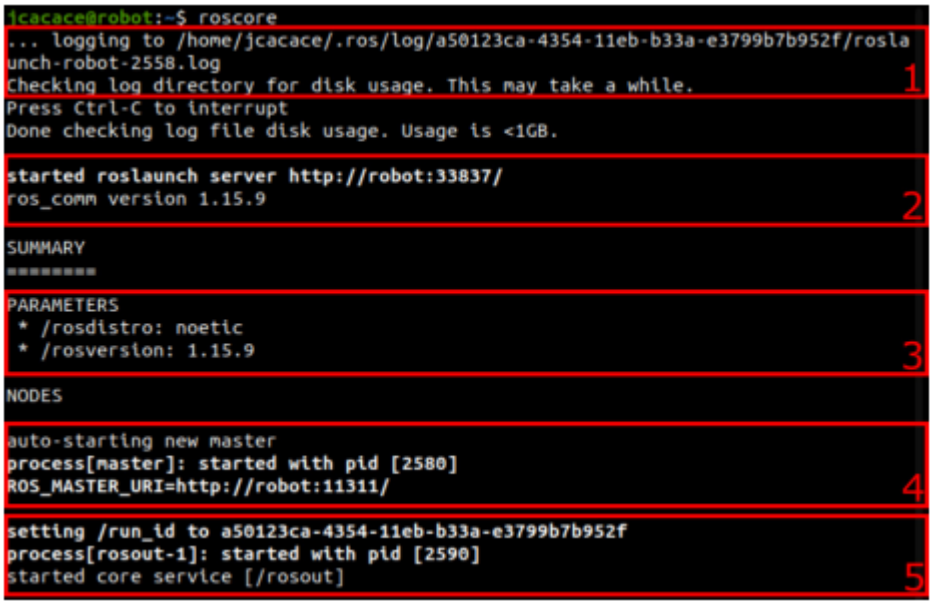
- ROS master
- Server parameter ROS
- Node logging rosout

Node rosout bertugas mengoleksi pesan log dari node ROS lainnya serta menyimpannya dalam sebuah file log. Selain itu, node tersebut mengirim ulang pesan log yang telah terkumpul ke topik lain. Topik /rosout dipublikasikan oleh node ROS menggunakan pustaka klien ROS seperti roscpp dan rospy. Node rosout sendiri berlangganan topik ini dan mengirim ulang pesan-pesannya ke dalam topik lain yang disebut /rosout_agg. Dalam topik ini, terdapat aliran pesan log yang telah digabungkan menjadi satu.

Penting untuk dicatat bahwa perintah roscore harus dijalankan sebagai prasyarat sebelum menjalankan node ROS apa pun. Berikut adalah tangkapan layar yang menunjukkan pesan yang muncul ketika roscore dijalankan di Terminal. Untuk memulai roscore, Anda dapat menggunakan perintah berikut di Terminal Linux:

Roscore

Setelah menjalankan perintah ini, kita akan melihat teks berikut di Terminal Linux:



Berikut ini adalah isi dari roscore.xml:

```
<launch>
    <group ns="/">
        <param name="rosversion" command="rosversion roslaunch" />
        <param name="rosdistro" command="rosversion -d" />
        <node pkg="rosout" type="rosout" name="rosout"
    respawn="true"/>
    </group>
</launch>
```

Isi dari roscore.xml adalah sebagai berikut: Ketika perintah roscore dijalankan, langkah awalnya adalah memeriksa argumen baris perintah untuk menemukan nomor port baru yang akan digunakan oleh rosmaster. Jika nomor port tersebut ada, roscore akan memulai proses mendengarkan pada port baru tersebut. Namun, jika tidak ada nomor port yang disediakan, roscore akan menggunakan port bawaan/default. Port ini, bersama dengan berkas peluncuran roscore.xml, akan diteruskan ke sistem roslaunch. Sistem roslaunch diimplementasikan sebagai modul Python; modul ini akan menguraikan nomor port serta menjalankan berkas roscore.xml.

Dalam file roscore.xml, parameter dan node ROS dikemas dalam sebuah grup tag XML yang berada dalam ruang nama /. Penggunaan grup tag XML menandakan bahwa

semua node di dalamnya memiliki pengaturan yang mirip. Parameter rosversion dan rosdistro menyimpan hasil dari perintah rosversion-roslaunch dan rosversion-d. Informasi ini diambil menggunakan tag command, yang merupakan bagian dari tag param ROS. Tag command bertugas untuk menjalankan perintah yang disebutkan di dalamnya dan menyimpan keluaran perintah tersebut ke dalam dua parameter yang disebutkan sebelumnya.

Rosmaster dan parameter server dijalankan menggunakan modul roslaunch melalui ROS_MASTER_URI. Ini merupakan proses yang terjadi di dalam modul Python roslaunch. ROS_MASTER_URI adalah gabungan antara alamat IP dan port yang akan dipantau oleh rosmaster. Port tersebut dapat diubah sesuai dengan port yang ditentukan dalam perintah roscore.

Checking the roscore command's output

Mari kita periksa topik dan parameter ROS yang dibuat setelah menjalankan roscore. Perintah berikut akan menampilkan daftar topik yang sedang aktif di Terminal:

```
rostopic list
```

Berikut adalah daftar topik yang muncul, sesuai dengan pembahasan kita tentang langganan simpul rosout/topik rosout. Ini mencakup semua pesan log dari node ROS. /rosout_agg akan menyiarkan ulang pesan log:

- /rosout
- /rosout_agg

Perintah berikut memberikan daftar parameter yang tersedia saat menjalankan roscore. Perintah berikut digunakan untuk mencantumkan parameter ROS yang aktif:

```
rosparam list
```

Berikut adalah parameter-parameter yang diidentifikasi; parameter-parameter ini menyediakan informasi seperti nama distribusi ROS, versi, alamat server roslaunch, dan run_id, di mana run_id merupakan ID unik yang terkait dengan proses spesifik dari roscore:

- /rosdistro
- /roslaunch
- /uris/host_robot_virtualbox__51189
- /rosversion
- /run_id

Daftar layanan ROS yang dihasilkan setelah menjalankan roscore dapat diperiksa menggunakan perintah berikut:

rosservice list

Daftar layanan yang sedang berjalan termasuk:

- /rosout/get_loggers
- /rosout/set_logger_level

Layanan ROS ini dibuat untuk setiap node ROS dan digunakan untuk mengatur tingkat log.

Chapter 2: Getting Started with ROS Programming

Creating a ROS package

Paket ROS merupakan komponen esensial dalam kerangka kerja ROS yang dapat dibuat, dikompilasi, dan dirilis untuk penggunaan publik. Pada distribusi ROS Noetic Ninjemys yang saat ini digunakan, penggunaan sistem build catkin menjadi standar untuk pembuatan paket ROS. Sistem build bertanggung jawab dalam menghasilkan hasil akhir dari kode sumber menjadi sesuatu yang dapat digunakan oleh pengguna akhir. Sebelumnya, distribusi ROS seperti Electric dan Fuerte menggunakan sistem build bernama rosbuild. Namun, karena keterbatasan yang dimiliki oleh rosbuild, munculnya catkin sebagai alternatif yang mendekatkan sistem kompilasi ROS dengan Cross Platform Make (CMake). Hal ini membawa sejumlah keunggulan, seperti kemampuan untuk mentransfer paket ke sistem operasi lain, termasuk Windows. Dengan dukungan CMake dan Python, paket yang berbasis catkin dapat dipindahkan ke berbagai sistem operasi tersebut.

Langkah pertama dalam bekerja dengan paket ROS adalah membuat ruang kerja catkin ROS. Setelah ROS terpasang, langkah pertama adalah membuat dan membangun ruang kerja catkin dengan nama catkin_ws:

```
mkdir -p ~/catkin_ws/src
source /opt/ros/noetic/setup.bash
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws
catkin_make
```

Perintah-perintah ini membuat struktur dasar ruang kerja catkin dan menginisialisasinya. `catkin_make` akan membuat direktori 'devel' dan 'build' di dalam ruang kerja catkin untuk pengembangan. Kemudian, untuk mengintegrasikan ruang kerja catkin ke lingkungan ROS, Anda dapat menambahkan baris ke file `.bashrc` agar setiap kali sesi bash baru dimulai, setup dari ruang kerja ini diakses:

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

Setelah membuat ruang kerja catkin, langkah selanjutnya adalah membuat paket ROS sendiri. Ini bisa dilakukan dengan perintah `catkin_create_pkg`, yang akan membuat paket untuk menunjukkan berbagai konsep ROS:

```
cd ~/catkin_ws/src
catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs
```

Setelah membuat paket, Anda bisa membangunnya menggunakan `catkin_make`. Ini akan membangun paket tanpa menambahkan node apa pun:

```
cd ~/catkin_ws && catkin_make
```

Semua paket ROS harus ditempatkan di dalam direktori 'src' di ruang kerja ROS. Ini penting agar paket diakui oleh sistem ROS dan dapat dikompilasi.

Creating ROS nodes

Node pertama yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini bertugas mempublikasikan nilai integer ke dalam topik yang bernama /numbers. Anda dapat menyalin kode yang ada saat ini ke dalam berkas baru atau menggunakan berkas yang sudah ada dari repositori kode buku ini. Berikut adalah 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;
```

```
}
```

Building the nodes

Dalam paket mastering_ros_demo_pkg, kita perlu mengedit file CMakeLists.txt untuk melakukan kompilasi dan pembangunan dari kode sumber. Cari file CMakeLists.txt di dalam paket ini. Di dalam file tersebut, terdapat potongan kode yang 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})
```

Anda bisa menambahkan potongan kode sebelumnya untuk membuat file CMakeLists.txt baru yang mengompilasi kedua potongan kode tersebut. Untuk membangun paket, perintah catkin_make digunakan. Pertama, navigasi ke ruang kerja:

```
cd ~/catkin_ws
```

Lalu, buat ruang kerja ROS, termasuk mastering_ros_demo_package, seperti ini:

```
catkin_make
```

Anda dapat menggunakan perintah sebelumnya untuk membangun seluruh ruang kerja atau menggunakan opsi -DCATKIN_WHITELIST_PACKAGES. Dengan opsi ini, Anda dapat menetapkan satu atau lebih paket untuk dikompilasi:

```
catkin_make -DCATKIN_WHITELIST_PACKAGES="pkg1,pkg2,..."
```

Penting untuk diingat bahwa Anda perlu mengembalikan konfigurasi ini untuk mengompilasi paket lain atau seluruh ruang kerja. Ini bisa dilakukan dengan perintah:

```
catkin_make -DCATKIN_WHITELIST_PACKAGES=""
```

Setelah selesai pembangunan, Anda bisa mengeksekusi node. Pertama, mulai roscore:

roscore

Setelah itu, jalankan kedua perintah tersebut di dua shell terpisah. Di penerbit yang sedang berjalan, jalankan perintah:

rosrun mastering_ros_demo_package demo_topic_publisher

Di subscriber yang berjalan, jalankan perintah:

rosrun mastering_ros_demo_package demo_topic_subscriber

Creating launch files

File peluncuran dalam ROS memungkinkan kita untuk meluncurkan banyak node sekaligus. Bayangkan skenario di mana kita perlu menjalankan puluhan node untuk sebuah robot. Melakukan ini satu per satu di terminal akan merepotkan. Sebagai alternatif, kita dapat menuliskan semua node dalam file XML yang disebut file peluncuran. Dengan menggunakan perintah roslaunch, kita bisa mem-parse file ini dan meluncurkan semua node sekaligus.

Perintah roslaunch secara otomatis memulai ROS master dan parameter server. Ini berarti kita tidak perlu menjalankan perintah roscore atau perintah node secara terpisah. Dengan hanya meluncurkan file peluncuran, semua langkah ini dapat dilakukan dalam satu perintah. Harap diperhatikan bahwa jika kita memulai sebuah node menggunakan perintah roslaunch, menghentikan atau memulai ulang node tersebut akan memiliki efek yang sama dengan memulai ulang roscore.

Mari kita mulai dengan membuat file peluncuran. Beralihlah ke direktori paket dan buatlah sebuah berkas baru bernama demo_topic.launch yang akan meluncurkan dua node ROS, satu untuk menerbitkan dan satu lagi untuk berlangganan nilai bilangan bulat. Berkas peluncuran ini akan disimpan di dalam folder 'launch' yang telah kita buat 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 menjalankannya dengan menggunakan perintah:

```
roslaunch mastering_ros_demo_pkg demo_topic.launch
```

Kita bisa memeriksa daftar node yang berjalan dengan perintah:

```
rosnode list
```

Selain itu, kita dapat melihat pesan log dan melakukan debugging pada node menggunakan alat GUI bernama rqt_console:

rqt_console

Chapter 3: Working with ROS for 3D Modeling

Creating the ROS package for the robot description

Sebelum membuat file URDF untuk robot, mari kita buat sebuah paket ROS di dalam catkin untuk mempertahankan model robot. Gunakan perintah berikut untuk membuat paket:

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

Paket ini utamanya bergantung pada paket urdf dan xacro. Jika paket-paket ini belum terpasang di sistem Anda, Anda dapat menginstalnya menggunakan manajer paket:

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

Ini akan memastikan bahwa paket-paket yang diperlukan seperti urdf dan xacro telah terinstal, memungkinkan Anda untuk membuat file URDF untuk model robot Anda.

Explaining the URDF file

Simpan kode URDF sebelumnya dengan nama pan_tilt.urdf dan periksa apakah file URDF tersebut mengandung kesalahan menggunakan perintah berikut:

```
check_urdf pan_tilt.urdf
```

Untuk menggunakan perintah ini, pastikan paket liburdfdom-tools telah diinstal. Jika belum, Anda bisa menginstalnya menggunakan perintah:

```
sudo apt-get install liburdfdom-tools
```

Jika Anda ingin secara grafis melihat struktur tautan dan sambungan robot, Anda dapat menggunakan alat perintah bernama urdf_to_graphiz:

```
urdf_to_graphiz pan_tilt.urdf
```

Perintah ini akan menghasilkan dua file: pan_tilt.gv dan pan_tilt.pdf. Untuk melihat struktur robot dalam format visual, Anda bisa menggunakan perintah berikut:

```
evince pan_tilt.pdf
```

Ini akan membuka file PDF yang menunjukkan struktur robot secara grafis menggunakan aplikasi PDF viewer.

Visualizing the 3D robot model in Rviz

Anda dapat meluncurkan model dengan menggunakan perintah berikut:

```
roslaunch mastering_ros_robot_description_pkg view_demo.launch
```

Untuk melihat model lengan robot dengan tujuh derajat kebebasan (seven-DOF) di Rviz.

Untuk membuat berkas peluncuran berikut di dalam folder 'launch' dan kemudian membangun paket menggunakan perintah catkin_make:

```
roslaunch mastering_ros_robot_description_pkg view_arm.launch
```

Anda dapat melihat robot bergerak dengan menggunakan perintah:

```
roslaunch mastering_ros_robot_description_pkg view_mobile_robot.launch
```

Ini akan meluncurkan tampilan visual robot dan memungkinkan Anda untuk melihatnya dalam lingkungan simulasi yang disediakan.

Chapter 4: Simulating Robots Using ROS and Gazebo

Simulating the robotic arm using Gazebo and ROS

Dalam pengembangan sebelumnya, lengan robot dengan tujuh derajat kebebasan telah didesain. Sekarang, langkahnya adalah mensimulasikan robot ini menggunakan Gazebo melalui ROS.

Pertama-tama, instal paket-paket yang diperlukan untuk bekerja dengan Gazebo dan ROS. Ini mencakup paket seperti `ros-noetic-gazebo-ros-pkgs`, `ros-noetic-gazebo-msgs`, `ros-noetic-gazebo-plugins`, dan `ros-noetic-gazebo-ros-control`.

Setelah instalasi, pastikan Gazebo terpasang dengan benar menggunakan perintah `roscore & rosrun gazebo_ros gazebo`.

Creating the robotic arm simulation model for Gazebo

Berikutnya, untuk membuat model simulasi untuk lengan robot, Anda perlu memperbarui deskripsi robot yang sudah ada dengan parameter-parameter simulasi. Gunakan perintah `catkin_create_pkg` untuk membuat paket yang diperlukan untuk simulasi ini.

Selanjutnya, setelah mempelajari plugin kamera di Gazebo, Anda dapat meluncurkan simulasi lengkap lengan robot dengan kamera Xtion Pro menggunakan perintah `roslaunch`. Anda juga dapat melihat data point cloud dari sensor ini di RViz menggunakan perintah `rosrun rviz -f /rgbd_camera_optical_frame`.

Setelah itu, luncurkan pengontrol ROS dengan Gazebo menggunakan `roslaunch` untuk memeriksa topik pengontrol yang dihasilkan.

Untuk menggerakkan sendi robot di Gazebo, Anda perlu mempublikasikan nilai sendi yang diinginkan dengan pesan `std_msgs/Float64` pada topik pengontrol posisi sendi. Contohnya, untuk memindahkan sendi keempat ke 1,0 radian, gunakan perintah `rostopic pub`.

Anda juga dapat melihat keadaan sendi robot menggunakan perintah `rostopic echo`.

Selain simulasi lengan robot, Anda juga dapat mensimulasikan robot beroda diferensial di Gazebo. Untuk meluncurkan file ini, gunakan perintah

`roslaunch`.

Adding the ROS teleop node

Untuk menambahkan node teleop ROS, Anda dapat menggunakan node `diff_wheeled_robot_key`. Pastikan paket `diff_wheeled_robot_control` terpasang

dengan benar dan gunakan perintah yang disediakan dalam paragraf untuk memulai node teleop ini.

Terakhir, gunakan RViz untuk memvisualisasikan keadaan robot dan data laser.

Chapter 5: Simulating Robots Using ROS, CoppeliaSim, and Webots

Setting up CoppeliaSim with ROS

Sebelum memulai bekerja dengan CoppeliaSim, langkah pertama 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 dengan lisensi pendidikan dan gratis untuk komersial. Anda dapat mengunduh versi terbaru dari simulator ini dari halaman unduhan Coppelia Robotics di http://www.coppeliarobotics.com/downloads.html, pilih versi edu untuk Linux. Dalam contoh ini, kita akan menggunakan versi CoppeliaSim 4.2.0.

Setelah mengunduh, ekstrak arsip dengan perintah `tar vxf CoppeliaSim_Edu_V4_2_0_Ubuntu20_04.tar.xz` dan ganti nama folder ke sesuatu yang lebih intuitif, misalnya: `mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim`. Untuk mempermudah akses, atur variabel lingkungan `COPPELIASIM_ROOT` yang mengarah ke folder utama CoppeliaSim dengan perintah `echo "export COPPELIASIM_ROOT=/path/to/CoppeliaSim/folder >> ~/.bashrc"`.

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

Simulating a robotic arm using CoppeliaSim and ROS

Selanjutnya, untuk mensimulasikan lengan robot tujuh derajat kebebasan (DOF), kita perlu mengimpor model tersebut ke dalam adegan simulasi CoppeliaSim. Ini dapat dilakukan dengan mengonversi file xacro ke format URDF dan menyimpannya dalam folder `urdf` pada paket `csim_demo_pkg`. Contohnya adalah: `rosrun xacro seven_dof_arm.xacro > /path/to/csim_demo_pkg/urdf/seven_dof_arm.urdf`.

Demikian juga, untuk mengatur Webots dengan ROS, langkah-langkah serupa dapat diikuti. Mulai dari mengotentikasi repositori Cyberbotics, menambahkan repositori Cyberbotics ke manajer paket APT, hingga menginstal Webots dengan perintah `sudo apt-get install webots`. Setelah instalasi, Webots dapat dijalankan dengan perintah `webots`.

Setting up Webots with ROS

Integrasi Webots-ROS memerlukan instalasi paket `webots_ros` menggunakan APT, seperti: `sudo apt-get install ros-noetic-webots-ros`. Setelah itu, Anda dapat menulis

sebuah node teleoperasi dengan memanfaatkan `webots_ros` sebagai dependensi untuk mengontrol kecepatan roda robot menggunakan pesan `geometry_msgs::Twist`.

Semua langkah ini bertujuan untuk mempersiapkan sistem Anda agar dapat mengintegrasikan simulasi robot dengan lingkungan ROS menggunakan simulator seperti CoppeliaSim atau Webots.