

Nama : Indra Andriansyah Dody Misnadin
NIM : 1103200005
Kelas : TK-44-G7

UAS Robotika

Chapter 1: Introduction to ROS

Technical Requirements

Pada Chapter ini, memerlukan komputer yang menjalankan OS Ubuntu 20.04 LTS atau distribusi Debian 10 GNU/Linux

Why should we learn ROS?

Sistem Operasi Robot (ROS) adalah kerangka kerja yang fleksibel, menyediakan alat dan perpustakaan untuk menulis perangkat lunak robot, dengan tujuan membentuk cara standar memprogram robot dan menawarkan komponen perangkat lunak siap pakai yang dapat dengan mudah diintegrasikan dengan aplikasi robotika kustom, serta mendukung pengembangan fitur tinggi dan berbagai sensor dan aktuator canggih tanpa kesulitan. Dalam ROS, banyak alat sumber terbuka untuk debugging, visualisasi, dan simulasi, dan ekosistemnya terus berkembang pesat dengan banyak pengguna dan pengembang di seluruh dunia. Tren adaptasi perangkat lunak ROS juga terlihat di industri robotika, di mana perusahaan beralih dari aplikasi robotika proprietary ke ROS.

Proyek ROS dimulai pada tahun 2007 oleh Morgan Quigley dan pengembangannya dilanjutkan di Willow Garage, sebuah laboratorium untuk mengembangkan perangkat keras dan perangkat lunak sumber terbuka untuk robot. Tujuan dari ROS adalah untuk menetapkan cara standar untuk memprogram robot sambil menawarkan perangkat lunak siap pakai yang dapat dengan mudah diintegrasikan dengan aplikasi robotik khusus. Ada banyak alasan untuk memilih ROS sebagai kerangka kerja pemrograman, dan beberapa di antaranya adalah sebagai berikut:

- Kemampuan kelas atas: ROS hadir dengan fungsi yang siap digunakan. Sebagai contoh, Pelokalan dan Pemetaan Simultan (SLAM) dan Adaptive Monte Carlo Localization (AMCL) di ROS dapat digunakan untuk memiliki otonom navigasi di robot seluler, sedangkan paket MoveIt dapat digunakan untuk gerakan perencanaan untuk manipulator robot. Kemampuan ini dapat langsung digunakan dalam perangkat lunak robot tanpa kerumitan. Dalam beberapa kasus, paket-paket ini cukup untuk

memiliki tugas robotika inti pada platform yang berbeda. Juga, kemampuan ini sangat tinggi dapat dikonfigurasi; kita dapat menyempurnakan masing-masing menggunakan berbagai parameter.

- Banyak sekali alat: Ekosistem ROS dilengkapi dengan banyak sekali alat untuk melakukan debugging, memvisualisasikan, dan melakukan simulasi. Alat-alat tersebut seperti `rqt_gui`, `RViz`, dan `Gazebo`, adalah beberapa alat open source terkuat untuk debugging, visualisasi, dan simulasi. Kerangka kerja perangkat lunak yang memiliki banyak alat ini sangat jarang.
- Dukungan untuk sensor dan aktuator kelas atas: ROS memungkinkan kita untuk 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 banyak lagi. Kita dapat menghubungkan komponen-komponen ini dengan ROS tanpa kerumitan.
- Penanganan sumber daya secara bersamaan: Menangani sumber daya perangkat keras melalui lebih dari dua proses selalu memusingkan. Bayangkan kita ingin memproses sebuah gambar dari sebuah kamera untuk deteksi wajah dan deteksi gerakan; kita dapat menulis kode sebagai entitas tunggal yang dapat melakukan keduanya, atau kita dapat menulis kode berulir tunggal untuk konkurensi. Jika kita ingin menambahkan lebih dari dua fitur ke dalam sebuah thread, aplikasi-aplikasi akan menjadi kompleks dan sulit untuk di-debug. Tetapi di ROS, kita dapat mengakses perangkat menggunakan topik ROS dari driver ROS. Sejumlah node ROS dapat berlangganan pesan gambar dari driver kamera ROS, dan setiap node bisa memiliki fungsi yang berbeda. Hal ini dapat mengurangi kompleksitas dalam komputasi dan juga meningkatkan kemampuan debugging seluruh sistem.

ROS metapackages

Metapaket ROS adalah paket khusus yang membutuhkan hanya satu file, yaitu file `package.xml`. Metapaket secara sederhana mengelompokkan sejumlah paket menjadi satu paket logis tunggal. 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 metapaketnavigasi:

```
roscd navigation
```

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

```
gedit package.xml
```

Running the ROS master and the ROS parameter

Sebelum menjalankan node ROS apa pun, kita harus memulai master ROS dan parameter ROS server. Kita dapat memulai master ROS dan server parameter ROS dengan menggunakan satu yang disebut roscore, yang akan memulai program-program berikut:

- ROS master
- ROS parameter server
- rosout logging nodes

Node rosout akan mengumpulkan pesan log dari node ROS lain dan menyimpannya dalam file log, dan juga akan menyiarkan ulang pesan log yang dikumpulkan ke topik lain. Topik /rosout dipublikasikan oleh node ROS menggunakan pustaka klien ROS seperti roscpp dan rospy, dan topik ini dilanggan oleh node rosout, yang menyiarkan ulang pesan dalam topik lain yang disebut /rosout_agg. Topik ini berisi aliran agregat log pesan. Perintah roscore harus dijalankan sebagai prasyarat untuk menjalankan ROS node. Tangkapan layar berikut ini menunjukkan pesan-pesan yang dicetak ketika kita menjalankan perintah roscore di Terminal. Gunakan perintah berikut untuk menjalankan roscore pada Terminal Linux:

Roscore

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

```
jacace@robot:~$ roscore
... logging to /home/jacace/.ros/log/a50123ca-4354-11eb-b33a-e3799b7b952f/rosla
unch-robot-2558.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://robot:33837/
ros_comm version 1.15.9

SUMMARY
=====
PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.9

NODES
auto-starting new master
process[master]: started with pid [2580]
ROS_MASTER_URI=http://robot:11311/

setting /run_id to a50123ca-4354-11eb-b33a-e3799b7b952f
process[rosout-1]: started with pid [2590]
started core service [/rosout]
```

Berikut ini adalah isi dari roscore.xml:

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

Berikut adalah penjelasan setiap bagian saat menjalankan perintah roscore di Terminal:

1. Pada bagian 1, dapat diamati bahwa sebuah file log dibuat di dalam folder ~/.ros/log untuk mengumpulkan log dari node-node ROS. File ini dapat digunakan untuk tujuan debugging.
2. Pada bagian 2, sebuah file peluncuran ROS yang disebut roscore.xml dimulai oleh perintah. Ketika sebuah file peluncuran dimulai, rosmaster dan server parameter ROS secara otomatis dimulai. Perintah roslaunch adalah skrip Python yang dapat memulai rosmaster dan server parameter ROS setiap kali mencoba menjalankan file peluncuran. Alamat server parameter ROS di dalam port ditunjukkan oleh bagian ini.
3. Pada bagian 3, parameter seperti rostdistro dan rosversion ditampilkan di Terminal. Parameter-parameter ini ditampilkan saat menjalankan roscore.xml. Rincian lebih lanjut tentang roscore.xml akan dibahas pada bagian berikutnya.
4. Pada bagian 4, terlihat bahwa node rosmaster dimulai dengan menggunakan ROS_MASTER_URI, yang telah ditentukan sebelumnya sebagai variabel lingkungan.
5. Pada bagian 5, terlihat bahwa node rosout dimulai, yang akan memulai berlangganan ketopik /rosout dan mengirimkannya kembali ke /rosout_agg.

Chapter 2: Getting Started with ROS Programming

Creating a ROS package

Paket ROS adalah unit dasar dari program ROS. Kita dapat membuat paket ROS, membangunnya, dan merilisnya ke publik. Distribusi ROS yang sedang kita gunakan saat ini adalah Noetic Ninjemys. Kami menggunakan sistem pembangunan catkin untuk membangun paket ROS. Sistem pembangunan bertanggung jawab untuk menghasilkan target (eksekutabel/pustaka) dari kode sumber teks yang dapat digunakan oleh pengguna akhir. Pada distribusi-distribusi sebelumnya, seperti Electric dan Fuerte, rosbuilt adalah sistem pembangunan yang digunakan. Karena berbagai kelemahan rosbuilt, catkin kemudian muncul. Hal ini juga memungkinkan kita untuk mendekatkan sistem kompilasi ROS kepada Cross Platform Make (CMake). Ini memiliki banyak keunggulan, seperti memindahkan paket ke sistem operasi lain, seperti Windows. Jika suatu sistem operasi mendukung CMake dan Python, paket berbasis catkin dapat dipindahkan ke dalamnya. Persyaratan pertama untuk bekerja dengan paket ROS adalah membuat ruang kerja catkin ROS. Setelah menginstal ROS, kita dapat membuat dan membangun ruang kerja catkin yang disebut 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 ini akan membuat direktori devel dan build di dalam ruang kerja catkin Anda. Berbagai file pengaturan berada di dalam folder devel. Untuk menambahkan ruang kerja ROS yang telah dibuat ke lingkungan ROS, kita harus menjalankan salah satu dari file-file ini. Selain itu, kita dapat menjalankan file pengaturan 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 berisi node-node contoh untuk mendemonstrasikan cara kerja topik, pesan, layanan, dan actionlib ROS. Perlu diperhatikan bahwa jika Anda belum mengatur ruang kerja dengan benar, maka Anda tidak akan dapat menggunakan perintah ROS apa pun. Perintah catkin_create_pkg adalah cara paling nyaman untuk membuat paket ROS. Perintah ini digunakan untuk membuat paket kita, di mana kita akan membuat demo dari berbagai konsep ROS.

Beralih ke folder src ruang kerja catkin dan buat paket dengan menggunakan perintah berikut:

```
catkin_create_pkg package_name [dependency1] [dependency2]
catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs
actionlib actionlib_msgs
```

Setelah membuat paket ini, bangun paket tanpa menambahkan node apapun dengan menggunakan perintah `catkin_make`. Perintah ini harus dijalankan dari jalur ruang kerja catkin. Perintah berikut menunjukkan cara membangun paket ROS kosong kita:

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

Node pertama yang akan kita bahas adalah `demo_topic_publisher.cpp`. Node ini akan menerbitkan nilai integer pada topik bernama `/numbers`. Salin kode saat ini ke dalam paket baru atau gunakan file 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;
}
```

Di sini, kita mengirimkan nilai integer melalui sebuah topik. Oleh karena itu, kita memerlukan tipe pesan untuk menangani data integer. `std_msgs` berisi definisi pesan standar

untuk tipe data primitif, sedangkan std_msgs/Int32.h berisi definisi pesan integer. Sekarang, kita dapat menginisialisasi sebuah node ROS dengan sebuah nama. Harus diingat bahwa nama node ROS harus bersifat unik:

```
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);
while ( ros::ok()
) {
std_msgs::Int32
msg;
msg.data =
number_count;
ROS_INFO("%d",msg.d
ata);
number_publisher.publis
h(msg); loop_rate.sleep();
```

Sekarang setelah kita membahas node penerbit, kita dapat membahas node pelanggan, yaitu demo_topic_subscriber.cpp.

Berikut adalah definisi dari node subscriber:

```
#include "ros/ros.h"
#include
"std_msgs/Int32.h"
#include <iostream>
void number_callback(const
std_msgs::Int32::ConstPtr& msg) {
ROS_INFO("Received [%d]",msg->data);
```

```

}

int main(int argc, char **argv) {

    ros::init(argc,
    argv,"demo_topic_subscriber");

    ros::NodeHandle node_obj;

    ros::Subscriber number_subscriber =
node_obj.subscribe("/ numbers",10,number_callback);

    ros::spin();
    return 0;

}

```

Kita harus mengedit file CMakeLists.txt di dalam paket untuk mengompilasi dan membangun kode sumber. Buka masterling_ros_demo_pkg untuk melihat file CMakeLists.txt yang ada.

```
cd ~/catkin_ws
```

```
catkin_make
```

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

```
catkin_make -
```

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

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

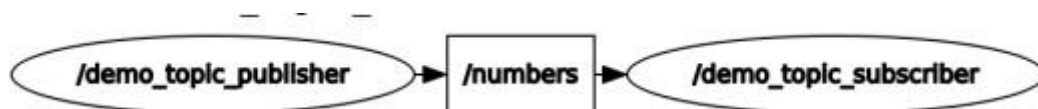
```
roscore
```

```
roslaunch masterling_ros_demo_package
```

```
demo_topic_publisher roslaunch
```

```
masterling_ros_demo_package demo_topic_subscriber
```

Diagram berikut menunjukkan bagaimana node berkomunikasi satu sama lain. Kita dapat melihat bahwa node demo_topic_publisher menerbitkan topik /numbers dan kemudian berlangganan ke node demo_topic_subscriber:



Chapter 3: Working with ROS for 3D Modeling

ROS menyediakan beberapa paket yang bagus yang dapat digunakan untuk membangun model robot 3D. Dalam bagian ini, kita akan membahas beberapa paket ROS penting yang umum digunakan untuk membangun dan memodelkan robot:

1. urdf: Paket ROS paling penting untuk memodelkan robot adalah paket urdf. Paket ini berisi parser C++ untuk URDF, yang merupakan file XML yang merepresentasikan model robot. Komponen-komponen berbeda lainnya membentuk urdf, seperti berikut:
 - a. urdf_parser_plugin: Paket ini mengimplementasikan metode untuk mengisi struktur data URDF.
 - b. urdfdom_headers: Komponen ini menyediakan header struktur data inti untuk menggunakan parser urdf.
 - c. collada_parser: Paket ini mengisi struktur data dengan mem-parsing file Collada.
 - d. urdfdom: Komponen ini mengisi struktur data dengan mem-parsing file URDF.

Kita dapat mendefinisikan model robot, sensor, dan lingkungan kerja menggunakan URDF. Kita juga dapat mem-parse mereka menggunakan parser URDF. Kita hanya dapat menggambarkan robot dalam URDF yang memiliki struktur berpohon pada link-linknya, yaitu, robot akan memiliki link yang kaku dan akan terhubung menggunakan sendi. Link yang fleksibel tidak dapat diwakili menggunakan URDF. URDF dibuat menggunakan tag XML khusus, dan kita dapat mem-parse tag XML ini menggunakan program parser untuk pengolahan lebih lanjut. Sebelum bekerja pada pemodelan URDF, mari tentukan beberapa paket ROS yang menggunakan file model robot:

1. joint_state_publisher: Alat ini sangat berguna saat merancang model robot menggunakan URDF. Paket ini berisi sebuah node yang disebut joint_state_publisher, yang membaca deskripsi model robot, menemukan semua sendi, dan menerbitkan nilai sendi ke semua sendi yang tidak tetap. Sumber-sumber berbeda untuk nilai setiap sendi juga tersedia. Kami akan membahas paket ini dan penggunaannya secara lebih detail pada bagian-bagian selanjutnya.
2. joint_state_publisher_gui: Alat ini sangat mirip dengan paket joint_state_publisher. Ini menawarkan fungsionalitas yang sama seperti paket joint_state_publisher dan, selain itu, mengimplementasikan serangkaian slider yang dapat digunakan oleh pengguna untuk berinteraksi dengan setiap sendi robot dan memvisualisasikan keluaran menggunakan RViz. Dalam hal ini, sumber nilai sendi adalah GUI slider. Saat merancang URDF, pengguna dapat memverifikasi rotasi dan translasi dari setiap sendi menggunakan alat ini.

Sebelum membuat file URDF untuk robot, mari buat paket ROS di ruang kerja catkin agar model robot terus dapat 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-paket ini belum terinstal di 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 launch untuk menampilkan file urdf yang telah dibuat di RViz. Paket lengkap tersedia di repositori Git berikut; Anda dapat mengklon repositori tersebut sebagai referensi untuk mengimplementasikan paket ini, atau Anda dapat mendapatkan paket tersebut dari kode sumber buku ini:

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

```
cd mastering_ros_robot_description_pkg/
```

Setelah merancang URDF, kita dapat melihatnya di RViz. Kita dapat membuat file peluncuran view_demo.launch dan memasukkan kode berikut ke dalam folder launch. Buka direktori mastering_ros_robot_description_pkg/launch untuk mendapatkan kode tersebut:

```
roslaunch mastering_ros_robot_description_pkg view_demo.launch
```

Sesuai yang telah disebutkan sebelumnya, file xacro dapat dikonversi menjadi file urdf setiap saat. Setelah merancang file xacro, kita dapat menggunakan perintah berikut untuk mengonversinya menjadi file URDF:

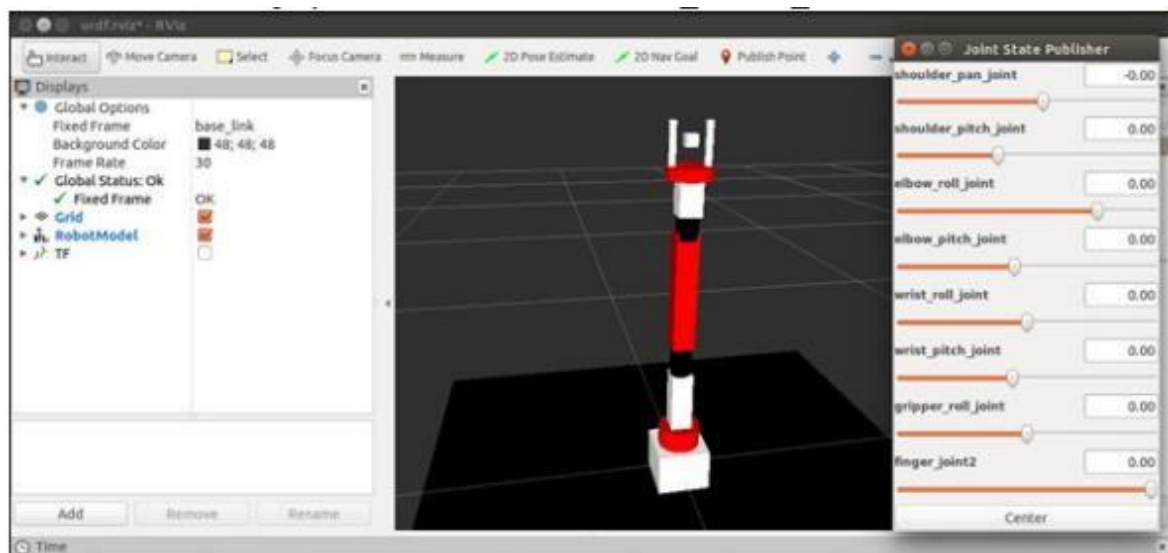
```
roslaunch xacro pan_tilt.xacro > pan_tilt_generated.urdf
```

```
roslaunch mastering_ros_robot_description_pkg view_pan_tilt_xacro.launch
```

Meluncurkan urdf menggunakan perintah berikut:

```
roslaunch mastering_ros_robot_description_pkg view_arm.launch
```

Robot akan ditampilkan di RViz dengan node GUI joint_state_publisher:



Chapter 4: Simulating Robots Using ROS and Gazebo

Pada bab sebelumnya, kita merancang sebuah lengan dengan tujuh derajat kebebasan. Pada bagian ini, kita akan mensimulasikan robot tersebut di Gazebo menggunakan ROS. Sebelum memulai dengan Gazebo dan ROS, kita sebaiknya menginstal paket-paket berikut untuk bekerja 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
```

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

```
roscore & rosrun gazebo_ros gazebo
```

Kita dapat membuat model simulasi untuk lengan robot dengan memperbarui deskripsi robot yang ada dengan menambahkan parameter simulasi. Kita dapat membuat paket yang diperlukan untuk mensimulasikan lengan robot dengan menggunakan perintah berikut:

```
catkin_create_pkg seven_dof_arm_gazebo gazebo_msgs gazebo_plugins gazebo_ros gazebo_ros_control mastering_ros_robot_description_pkg
```

Sebagai alternatif, paket lengkap tersedia di repositori Git berikut; Anda dapat mengklon repositori tersebut sebagai referensi untuk mengimplementasikan paket ini, atau Anda dapat mendapatkan paket tersebut dari kode sumber buku ini:

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

```
cd Chapter4/seven_dof_arm_gazebo
```

Jalankan perintah berikut dan periksa apa yang Anda dapatkan:

```
roslaunch seven_dof_arm_gazebo seven_dof_arm_world.launch
```

Anda dapat melihat lengan robot di Gazebo, seperti yang ditunjukkan dalam gambar berikut; jika Anda mendapatkan keluaran ini tanpa kesalahan, Anda sudah selesai.



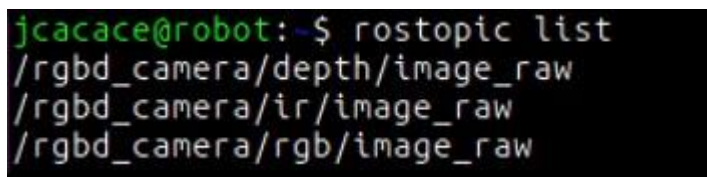
Di Gazebo, kita dapat mensimulasikan pergerakan robot dan fisiknya; kita juga dapat mensimulasikan berbagai jenis sensor. Untuk membangun sensor di Gazebo, kita harus memodelkan perilakunya. Ada beberapa model sensor yang sudah dibangun di Gazebo yang dapat digunakan secara langsung dalam kode kita tanpa menulis model baru.

Sekarang setelah kita telah mempelajari tentang definisi plugin kamera di Gazebo, kita dapat meluncurkan simulasi lengkap kita menggunakan perintah berikut:

roslaunch seven_dof_arm_gazebo

seven_dof_arm_with_rgbd_world.launch

Setelah meluncurkan simulasi menggunakan perintah sebelumnya, kita dapat memeriksa topik-topik yang dihasilkan oleh plugin sensor:

A terminal window with a black background and green text. The prompt is 'jcacace@robot:~\$' followed by the command 'rostopic list'. The output lists three topics: '/rgbd_camera/depth/image_raw', '/rgbd_camera/ir/image_raw', and '/rgbd_camera/rgb/image_raw'.

```
jcacace@robot:~$ rostopic list
/rgbd_camera/depth/image_raw
/rgbd_camera/ir/image_raw
/rgbd_camera/rgb/image_raw
```

Untuk melihat data gambar dari sensor visi 3D menggunakan alat bernama image_view, lakukan langkah- langkah berikut:

1. View the RGB raw image:
roslaunch image_view image:=/rgbd_camera/rgb/image_raw
2. View the IR raw image:
roslaunch image_view image:=/rgbd_camera/ir/image_raw
3. View the depth image:
roslaunch image_view image:=/rgbd_camera/depth/image_raw

Kita juga dapat melihat data awan titik dari sensor ini di RViz. Jalankan rviz menggunakan perintah berikut:

roslaunch rviz -f /rgbd_camera_optical_frame

Setelah menyelesaikan topik-topik sebelumnya, kita dapat mulai mengendalikan setiap sendi ke posisi yang diinginkan. Untuk menggerakkan sebuah sendi robot di Gazebo, kita harus menerbitkan nilai sendi yang diinginkan dengan tipe pesan std_msgs/Float64 ke topik perintah pengendali posisi sendi. Berikut adalah contoh menggerakkan sendi keempat ke 1.0 radian:

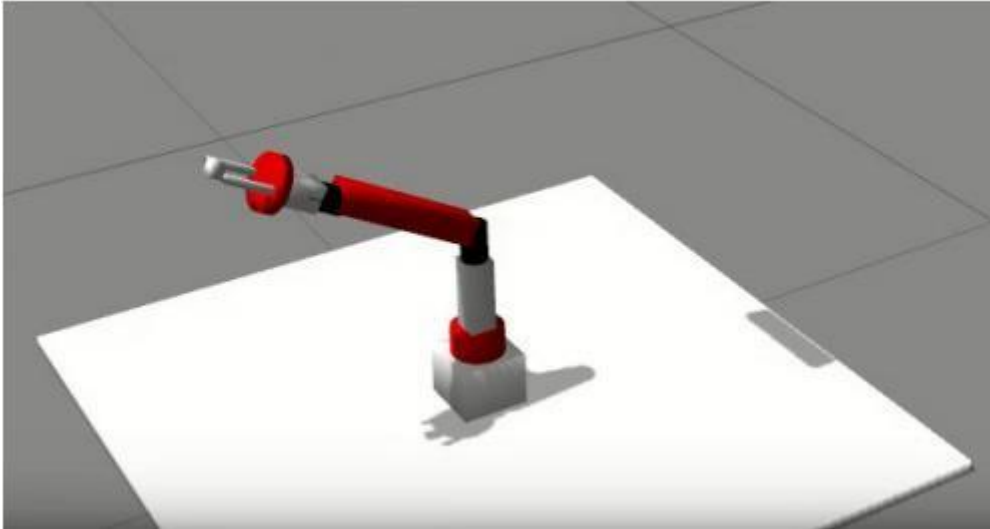
rostopic pub

/seven_dof_arm/joint4_position_controller/command

std_msgs/Float64 1.0

Kita juga dapat melihat status sendi robot dengan menggunakan perintah berikut:

`rostopic echo /seven_dof_arm/joint_states`



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

Sebelum mulai bekerja dengan CoppeliaSim, kita perlu menginstalnya di sistem kita dan mengonfigurasi lingkungan kita untuk memulai jembatan komunikasi antara ROS dan sene simulasi. CoppeliaSim adalah perangkat lunak lintas platform, tersedia untuk berbagai sistem operasi seperti Windows, macOS, dan Linux. Ini dikembangkan oleh Coppelia Robotics GmbH dan didistribusikan dengan lisensi edukasi gratis dan lisensi komersial. Unduh versi terbaru simulator CoppeliaSim dari halaman unduhan Coppelia Robotics di <http://www.coppeliarobotics.com/downloads.html>, memilih versi edu untuk Linux. Dalam bab ini, kita akan merujuk pada versi CoppeliaSim 4.2.0. Setelah menyelesaikan unduhan, ekstrak arsipnya. Pindah ke folder unduhan Anda dan gunakan perintah berikut:

```
tar vxf CoppeliaSim_Edu_V4_2_0_Ubuntu20_04.tar.xz
```

```
mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim
```

```
echo "export COPPELIASIM_ROOT=/path/to/CoppeliaSim/folder >>
```

```
~/bashrc"
```

Sekarang, kita siap untuk memulai simulator. Untuk mengaktifkan antarmuka komunikasi ROS, perintah roscore harus dijalankan di mesin Anda sebelum membuka simulator, sementara untuk membuka CoppeliaSim, kita dapat menggunakan perintah berikut:

```
cd $COPPELIASIM_ROOT
```

```
./coppeliaSim.sh
```

Dalam simulasi ini, kamera pasif menampilkan gambar yang diterbitkan dari kamera aktif, menerima data visi secara langsung dari kerangka kerja ROS. Kita juga dapat memvisualisasikan aliran video yang diterbitkan oleh CoppeliaSim menggunakan paket image_view, dengan menjalankan perintah berikut:

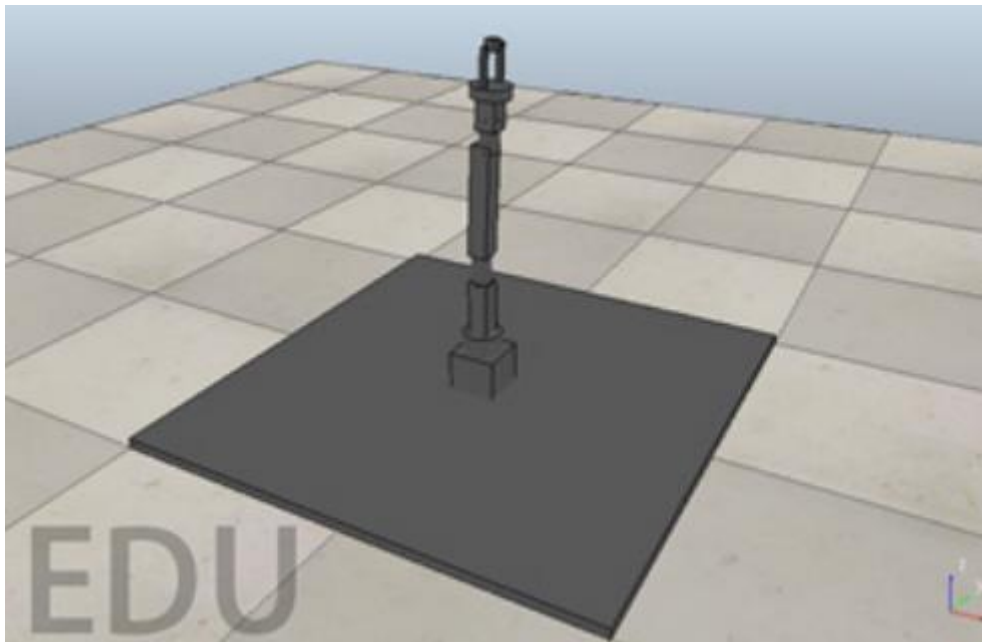
```
roslaunch image_view image:=/camera/image_raw
```

Untuk menerbitkan pesan ROS baru dalam skrip Lua, kita perlu membungkusnya dalam struktur data yang berisi bidang yang sama dengan pesan asli. Prosedur sebaliknya harus dilakukan untuk mengumpulkan informasi yang diterbitkan pada topik ROS. Mari analisis pekerjaan yang dilakukan pada contoh sebelumnya sebelum kita beralih ke sesuatu yang lebih rumit. Pada contoh dummy_publisher, tujuannya adalah untuk menerbitkan data integer pada topik ROS. Kita dapat memeriksa struktur pesan integer menggunakan perintah ROS ini:

```
rosmmsg show std_msgs/Int32 int32 data
```

Pada bab sebelumnya, kita menggunakan Gazebo untuk mengimpor dan mensimulasikan lengan tujuh derajat kebebasan (DOF) yang dirancang di Bab 3, Bekerja dengan ROS untuk Pemodelan 3D. Di sini, kita akan melakukan hal yang sama menggunakan CoppeliaSim. Langkah pertama untuk mensimulasikan lengan tujuh DOF kita adalah mengimpornya ke dalam sene simulasi. CoppeliaSim memungkinkan Anda mengimpor robot baru menggunakan file URDF; karena itu, kita harus mengonversi model xacro dari lengan ke

dalam file URDF, menyimpan file URDF yang dihasilkan di folder urdf dari paket csim_demo_pkg, seperti berikut:



Seperti yang telah dilakukan dengan CoppeliaSim, kita perlu menginstal Webots di sistem kita sebelum mengaturnya dengan ROS. Webots adalah perangkat lunak simulasi multiplatform yang didukung oleh Windows, Linux, dan macOS. Perangkat lunak ini awalnya dikembangkan oleh Swiss Federal Institute of Technology, Lausanne (EPFL). Sekarang, ini dikembangkan oleh Cyberbotics, dan dirilis di bawah lisensi Apache 2 yang gratis dan sumber terbuka. Webots menyediakan lingkungan pengembangan lengkap untuk memodelkan, memprogram, dan mensimulasikan robot. Ini dirancang untuk penggunaan profesional dan banyak digunakan di industri, pendidikan, dan penelitian. Anda dapat memilih berbagai cara untuk menginstal simulator ini. Anda dapat mengunduh paket .deb dari halaman web Webots (<http://www.cyberbotics.com/#download>) atau menggunakan manajer paket Debian/Ubuntu Advanced Packaging Tool (APT). Mengasumsikan bahwa Anda menjalankan Ubuntu, mari mulai dengan mengotentikasi repositori Cyberbotics, seperti berikut:

```
wget -qO- https://cyberbotics.com/Cyberbotics.asc | sudo apt-key add -
```

```
sudo apt-add-repository 'deb
```

```
https://cyberbotics.com/debian/binary-amd64/' sudo apt-get
```

```
update
```

```
sudo apt-get install webots
```

```
$ webots
```

Setelah perintah ini, antarmuka pengguna (UI) Webots akan terbuka. Dengan menggunakan menu simulasi di bagian atas jendela, Anda dapat mengontrol simulasi, memulai atau memberhentikan simulasi, atau mempercepat eksekusinya.

