NAMA : Albizhar Zidane Budi Laksana

NIM : 1103202116

KELAS : TK-44-G7

UAS ROBOTIKA

TECHNICAL REPORT CHAPTER 1 – 6

Chapter 1 : introduction to ROS

Technical requirements

Diperlukan penggunaan komputer dengan OS Ubuntu 20.04 LTS atau varian distribusi Debian 10 GNU/Linux. Disertakan sedikitnya spesifikasi minimum untuk kinerja optimal.

Why should we learn ROS?

Sistem Pengendalian Robot (SPR) merupakan platform yang lentur, menyediakan alat serta perpustakaan bagi pengembangan perangkat lunak robot. Tujuannya adalah untuk membentuk standar pemrograman robotik, menyediakan komponen perangkat lunak yang siap pakai, mudah diintegrasikan ke aplikasi kustom robotika, serta mendukung pengembangan fitur canggih dan ketersediaan sensor dan aktuator yang tinggi tanpa kesulitan. Dalam SPR, tersedia sejumlah perangkat sumber terbuka untuk debugging, visualisasi, dan simulasi. Ekosistemnya terus berkembang pesat dengan kontribusi dari pengguna dan pengembang di seluruh dunia. Adopsi perangkat lunak SPR juga menjadi tren di industri robotika, di mana perusahaan beralih dari solusi propietary ke SPR.

Proyek SPR diawali pada tahun 2007 oleh Morgan Quigley dan berlanjut di Willow Garage, sebuah laboratorium yang fokus pada pengembangan perangkat keras dan perangkat lunak sumber terbuka untuk robot. ROS bertujuan untuk menetapkan standar dalam pemrograman robot sambil menyediakan perangkat lunak yang mudah diintegrasikan ke aplikasi khusus robotika. Ada sejumlah alasan untuk memilih SPR sebagai kerangka kerja pemrograman, di antaranya adalah sebagai berikut:

- Kapasitas Kelas Atas: ROS menyediakan fungsionalitas yang telah siap pakai. Misalnya, dalam ROS, Pelokalan dan Pemetaan Simultan (SLAM) serta Lokalisasi Monte Carlo Adaptif (AMCL) dapat digunakan untuk navigasi mandiri pada robot

mobile. Paket seperti MoveIt juga digunakan untuk perencanaan gerakan pada manipulator robot. Fungsionalitas ini dapat langsung diintegrasikan ke dalam perangkat lunak robot tanpa kesulitan. Di beberapa kasus, paket-paket ini sudah cukup untuk menjalankan tugas-tugas inti dalam robotika pada berbagai platform. Selain itu, kemampuan ini dapat disesuaikan secara tinggi; berbagai parameter dapat disetel untuk menyempurnakan masing-masing fungsi.

- Beragamnya Alat: Ekosistem ROS dilengkapi dengan beragam peralatan untuk melakukan debugging, visualisasi, dan simulasi. Contohnya, rqt_gui, RViz, dan Gazebo merupakan beberapa alat sumber terbuka terkemuka yang digunakan untuk tiga tujuan tersebut. Keberagaman alat dalam kerangka kerja perangkat lunak seperti ini cukup langka.
- Dukungan untuk Sensor dan Aktuator Kelas Atas: ROS memungkinkan integrasi driver dan paket antarmuka yang berbeda untuk berbagai sensor dan aktuator dalam robotika. Sensor kelas atas seperti LIDAR 3D, pemindai laser, sensor kedalaman, aktuator, dan sebagainya dapat dihubungkan dengan ROS tanpa kesulitan.
- Manajemen Sumber Daya Secara Simultan: Mengelola sumber daya perangkat keras melalui lebih dari dua proses seringkali rumit. Bayangkan jika ingin memproses gambar dari sebuah kamera untuk deteksi wajah dan gerakan. Kita bisa menulis kode sebagai entitas tunggal yang mampu melakukan keduanya atau menulis kode yang bersifat konkuren. Di ROS, kita dapat mengakses perangkat menggunakan topik ROS dari driver ROS. Banyak node ROS bisa berlangganan pesan gambar dari driver kamera ROS, dan setiap node dapat memiliki fungsi yang berbeda. Ini mengurangi kompleksitas komputasi dan meningkatkan kemampuan debugging seluruh sistem.

ROS Metapackages

Metapaket ROS adalah komponen spesifik yang memerlukan satu file saja, yaitu package.xml. Secara esensial, paket meta ini menggabungkan beberapa paket menjadi satu paket logis tunggal. Dalam file package.xml, paket meta ini berisi tag ekspor, seperti yang dapat dilihat di bawah ini:

```
<export>
    <metapackages/>
</export>
```

Pada paket meta, tidak ada ketergantungan <buildtool_depend> untuk catkin; hanya ada ketergantungan <run_depend>, yang merupakan paket-paket yang tercakup dalam paket meta tersebut. Stack navigasi ROS adalah contoh konkret dari lokasi yang memuat paket meta. Setelah instalasi ROS dan paket navigasi selesai, kita dapat mencoba menggunakan perintah berikut setelah berpindah ke direktori stack navigasi: roscd navigation Kemudian, buka file package.xml menggunakan editor teks pilihan Anda. Sebagai contoh, kita bisa menggunakan gedit dengan perintah:

gedit package.xml

Running the ROS master and the ROS parameter

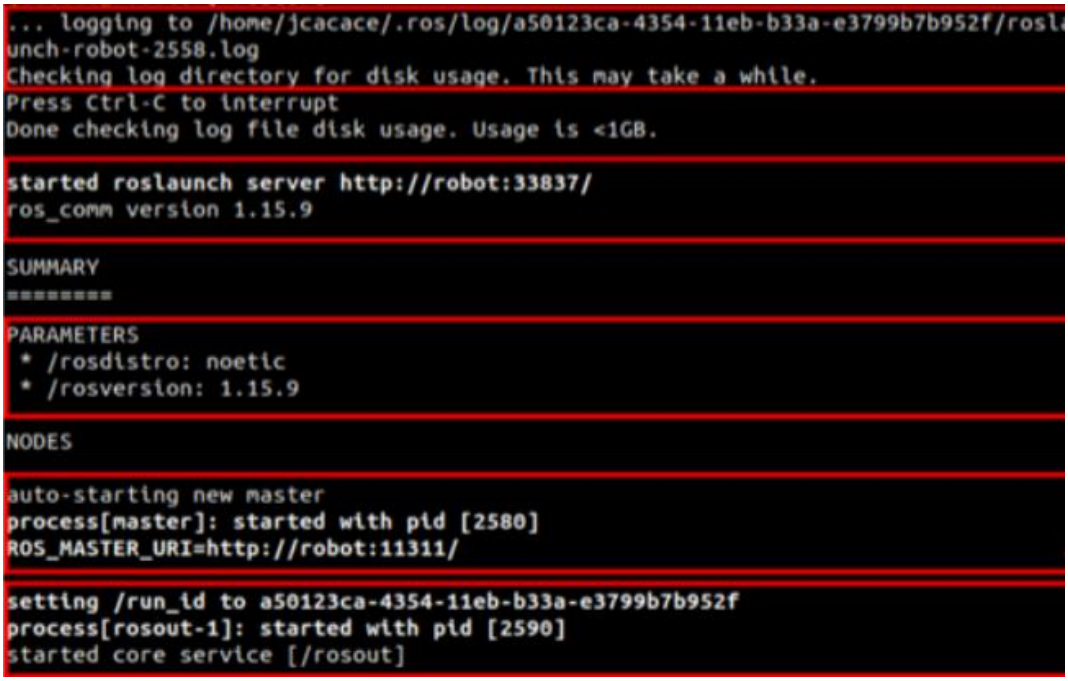
Sebelum memulai eksekusi node ROS apa pun, perlu untuk memulai master ROS dan server parameter ROS. Inisiasi master ROS dan server parameter ROS dapat dilakukan dengan menggunakan perintah bernama roscore, yang akan memulai program-program berikut:

- Master ROS
- Server parameter ROS
- Node-node logging rosout

Node rosout bertugas mengumpulkan pesan log dari node-node ROS lainnya dan menyimpannya dalam file log, serta menyiarkan kembali pesan log yang terkumpul ke topik lain. Topik /rosout dipublikasikan oleh node-node ROS menggunakan pustaka klien ROS seperti roscpp dan rospy, dan topik ini di-subscribe oleh node rosout, yang mengirimkan kembali pesan ke dalam topik lain yang bernama /rosout_agg. Topik ini berisi kumpulan pesan log yang telah digabungkan. roscore harus diterapkan sebagai prasyarat untuk menjalankan node ROS apa pun. Tangkapan layar di bawah menunjukkan pesan-pesan yang muncul saat perintah roscore dijalankan di Terminal. Untuk memulai roscore pada Terminal Linux, gunakan perintah berikut:

❖ 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>
```

Inilah penjelasan setiap segmen saat menjalankan perintah roscore di Terminal:

1. Pada segmen pertama, teramati pembuatan sebuah file log di folder ~/.ros/log yang mengumpulkan log dari node-node ROS. File ini berguna untuk keperluan debugging.

2. Segmen kedua memulai sebuah file peluncuran ROS yang disebut roscore.xml. Ketika file peluncuran dimulai, rosmaster dan server parameter ROS juga akan otomatis diinisiasi. Perintah roslaunch berupa skrip Python yang mampu menginisiasi rosmaster dan server parameter ROS setiap kali menjalankan file peluncuran. Alamat server parameter ROS pada port tertentu ditampilkan pada segmen ini.

3. Segmen ketiga menampilkan parameter seperti rosdistro dan rosversion di Terminal. Parameter-parameter ini muncul saat menjalankan roscore.xml. Detail lebih lanjut tentang roscore.xml akan dibahas pada bagian selanjutnya.
4. Segmen keempat menunjukkan bahwa node rosmaster dimulai menggunakan ROS_MASTER_URI yang telah ditetapkan sebelumnya sebagai variabel lingkungan.
5. Pada segmen kelima, tampak dimulainya node rosout yang berlangganan ke topik /rosout dan meneruskannya ke /rosout_agg.

Chapter 2 : Getting Started with ROS Programming

Creating a ROS package

Unit dasar dalam lingkungan program ROS adalah paket ROS. Dalam ROS, kita dapat membuat, membangun, dan merilis paket-paket ini ke publik. Saat ini, distribusi ROS yang digunakan adalah Noetic Ninjemys. Kami menggunakan sistem pembangunan catkin untuk membangun paket-paket ROS ini. Sistem pembangunan bertanggung jawab untuk menghasilkan target, seperti eksekutabel atau pustaka, dari kode sumber teks yang dapat digunakan oleh pengguna akhir. Pada distribusi sebelumnya, seperti Electric dan Fuerte, digunakan sistem pembangunan bernama rosbuild. Namun, karena berbagai keterbatasan rosbuild, muncullah catkin. Catkin memungkinkan pendekatan yang lebih dekat dengan Cross Platform Make (CMake) dalam kompilasi ROS. Ini memiliki keunggulan, seperti memungkinkan paket untuk beroperasi pada sistem operasi lain, termasuk Windows. Asalkan sistem operasi mendukung CMake dan Python, paket-paket yang berbasis catkin dapat diadaptasi untuk digunakan di dalamnya. Langkah pertama untuk bekerja dengan paket-paket ROS adalah menciptakan ruang kerja catkin ROS. Setelah menginstal ROS, langkah selanjutnya adalah 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 tersebut akan menciptakan direktori devel dan build di dalam ruang kerja catkin Anda. Di dalam folder devel terdapat sejumlah file konfigurasi. Untuk menyertakan ruang kerja ROS yang telah dibuat ke dalam lingkungan ROS, diperlukan jalannya salah satu dari file-file tersebut. Selain itu, Anda juga dapat menjalankan file-file konfigurasi ruang kerja setiap kali sesi bash dimulai dengan menggunakan perintah berikut:

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

Setelah menyiapkan ruang kerja catkin, langkah selanjutnya adalah membuat paket sendiri yang berisi contoh node untuk menunjukkan cara kerja topik, pesan, layanan, dan actionlib dalam lingkungan ROS. Pastikan ruang kerja Anda telah disiapkan dengan benar

sebelumnya, karena tanpa itu, Anda tidak akan dapat menggunakan perintah-perintah ROS. Perintah yang paling nyaman untuk menciptakan paket ROS adalah catkin_create_pkg. Dengan perintah ini, kita dapat membuat paket yang akan berisi demo dari berbagai konsep dalam ROS.

Untuk melanjutkan, navigasikan ke dalam folder src di ruang kerja catkin dan buatlah 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 paket dibuat, kita bisa melakukan pembangunan paket tanpa menambahkan node apapun dengan perintah catkin_make. Pastikan perintah ini dijalankan dari direktori yang ada di ruang kerja catkin. Perintah di bawah ini menunjukkan bagaimana membangun paket ROS yang kosong:

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

Node awal yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini bertugas menerbitkan nilai integer ke dalam topik yang diberi nama /numbers. Anda bisa menyalin kode ini ke dalam paket yang baru dibuat atau menggunakan file yang sudah ada dari repositori kode buku ini.

Code:

```
#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 situ, kita akan mengirimkan nilai integer melalui sebuah topik. Oleh karena itu, diperlukan tipe pesan yang dapat menangani data integer. Paket std_msgs berisi definisi pesan standar untuk tipe data primitif, sementara std_msgs/Int32.h mengandung definisi pesan untuk tipe data integer. Sekarang, kita dapat menginisialisasi sebuah node ROS dengan sebuah nama. Penting untuk diingat bahwa nama node ROS harus bersifat unik:

Code:

```
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 membicarakan node penerbit, mari kita lihat node pelanggan, yaitu demo_topic_subscriber.cpp.

Code:

```
#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;
```

```
}
```

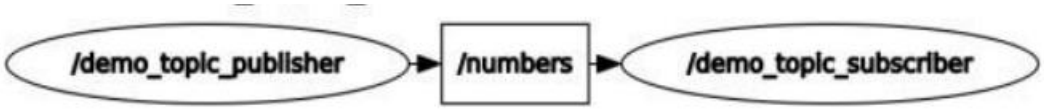
Kita harus mengedit file CMakeLists.txt di dalam paket untuk mengompilasi dan membangun kode sumber. Buka mastering_ros_demo_pkg untuk melihat file CMakeLists.txt yang ada.

```
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

rosrun mastering_ros_demo_package

demo_topic_publisher rosrun

mastering_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 luar biasa untuk membangun model robot 3D. Dalam bagian ini, kita akan melihat beberapa paket ROS penting yang umumnya digunakan untuk membangun dan memodelkan robot:

1. urdf: Paket ROS yang paling penting untuk memodelkan robot adalah paket urdf. Di dalamnya terdapat parser C++ untuk URDF, sebuah file XML yang mewakili model robot. Berbagai komponen lain membentuk urdf, seperti berikut:
    - urdf_parser_plugin: Paket ini mengimplementasikan metode untuk mengisi struktur data URDF.
    - urdfdom_headers: Komponen ini menyediakan header struktur data inti untuk menggunakan parser urdf.
    - collada_parser: Paket ini mengisi struktur data dengan mem-parse file Collada.
    - urdfdom: Komponen ini mengisi struktur data dengan mem-parse file URDF.

Kita dapat mendefinisikan model robot, sensor, dan lingkungan kerja menggunakan URDF. Kita juga dapat mem-parse mereka menggunakan parser URDF. URDF hanya dapat menggambarkan robot dengan struktur berpohon pada link-linknya, dengan link yang kaku dan terhubung melalui sendi. Link yang fleksibel tidak dapat diwakili menggunakan URDF. URDF dibuat menggunakan tag XML khusus, yang bisa di-parse menggunakan program parser untuk pengolahan lebih lanjut.

Sebelum kita mulai bekerja pada pemodelan URDF, mari kita 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 bernama joint_state_publisher, yang membaca deskripsi model robot, menemukan semua sendi, dan menerbitkan nilai sendi ke semua sendi yang tidak tetap. Terdapat sumber nilai yang berbeda untuk setiap sendi. Kita akan membahas paket ini dan penggunaannya secara lebih rinci pada bagian selanjutnya.
2. joint_state_publisher_gui: Alat ini mirip dengan paket joint_state_publisher. Selain menawarkan fungsionalitas yang sama, paket ini juga menyediakan serangkaian slider yang dapat digunakan pengguna untuk berinteraksi dengan setiap sendi robot dan memvisualisasikan keluarannya menggunakan RViz. Di sini, sumber nilai sendi

berasal dari 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 sebuah paket ROS di dalam ruang kerja catkin untuk memungkinkan pemodelan terus 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 dapat ditemukan di repositori Git berikut ini; Anda bisa mengkloning repositori tersebut sebagai referensi untuk mengimplementasikan paket ini, atau Anda bisa mendapatkan paket tersebut dari sumber kode 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
```

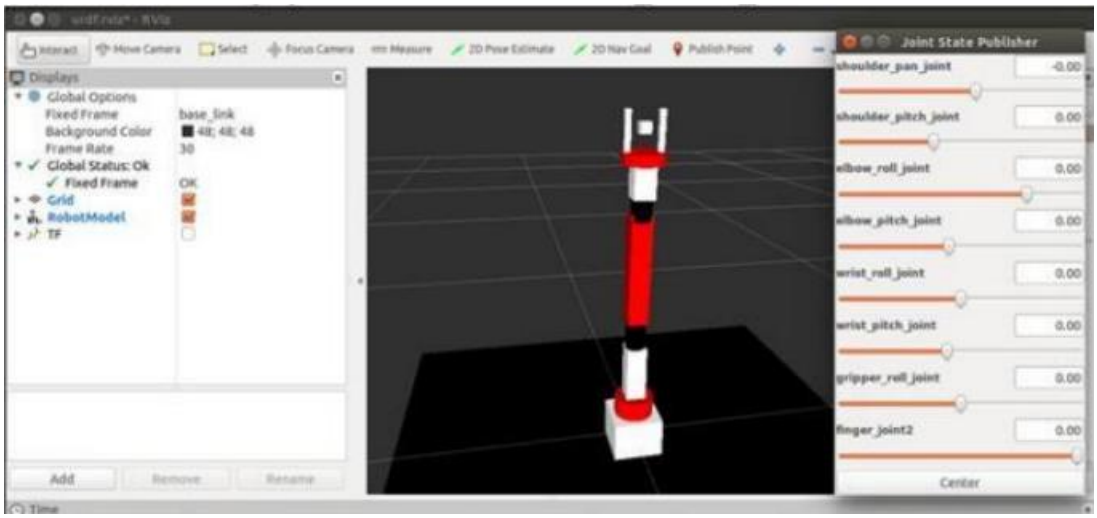
Sesuai yang telah disebutkan sebelumnya, file xacro dapat dikonversi menjadi file urdf setiap saat. Setelah merancang file xacro, kita dapat menggunakan perintah berikut untuk mengonversinya menjadi file URDF:

```
rosrun 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

Di bab sebelumnya, kita telah merancang sebuah lengan dengan tujuh derajat kebebasan. Sekarang, pada bagian ini, kita akan melakukan simulasi robot tersebut di Gazebo dengan menggunakan ROS. Sebelum memulai dengan Gazebo dan ROS, disarankan untuk menginstal beberapa paket berikut agar dapat bekerja dengan Gazebo dan ROS:

```
sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-gazebo-msgs ros-noeticgazebo-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 sudah ada dan menambahkan parameter simulasi. Untuk mensimulasikan lengan robot, kita perlu membuat paket yang sesuai 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
```

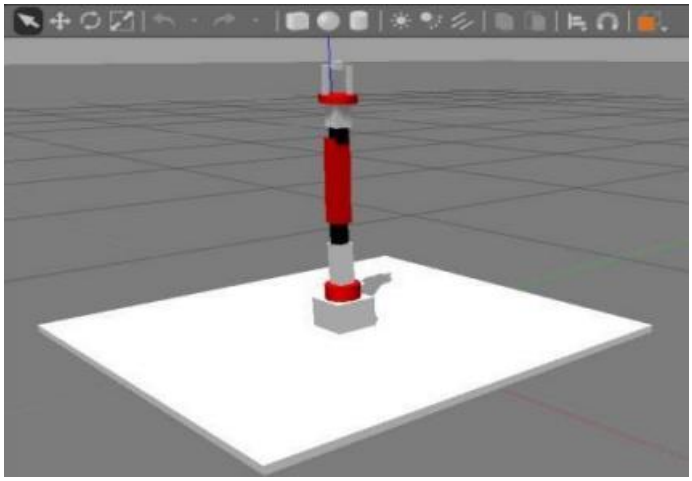
Sebagai opsi lain, paket lengkap tersedia di repositori Git berikut ini; Anda dapat mengklon repositori tersebut sebagai acuan untuk menerapkan paket ini, atau Anda juga bisa memperoleh paket tersebut dari sumber kode buku ini:

```
git clone https://github.com/PacktPublishing/Mastering-ROS-
for-RoboticsProgramming-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 bisa mensimulasikan pergerakan serta fisika robot, dan bahkan mensimulasikan beragam jenis sensor. Untuk membangun sensor di Gazebo, diperlukan pemodelan perilakunya. Ada beberapa model sensor yang telah tersedia di Gazebo yang bisa langsung digunakan dalam kode kita tanpa perlu menulis model baru.

Sekarang, setelah kita memahami tentang definisi plugin kamera di Gazebo, kita dapat memulai simulasi lengkap menggunakan perintah berikut:

```
roslaunch seven_dof_arm_gazebo
```

```
seven_dof_arm_with_rgbd_world. Launch
```

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

```
/rgbd_camera/depth/image_raw
/rgbd_camera/ir/image_raw
/rgbd_camera/rgb/image_raw
```

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

1. Untuk melihat gambar mentah RGB:

   ```
   rosrun image_view image:=/rgbd_camera/rgb/image_raw
   ```

2. Untuk melihat gambar mentah IR:

   ```
   rosrun image_view image:=/rgbd_camera/ir/image_raw
   ```

3. Untuk melihat gambar kedalaman:

```
rosrun image_view image:=/rgbd_camera/depth/image_raw
```

Selain itu, kita juga bisa melihat data awan titik dari sensor ini di RViz. Untuk melakukannya, jalankan rviz dengan perintah berikut:

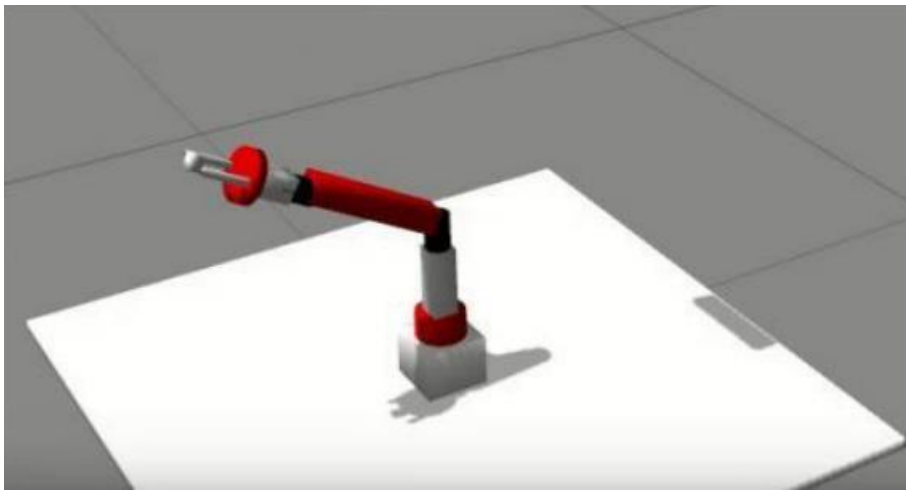
```
rosrun rviz -f /rgbd_camera_optical_frame
```

Setelah menjelajahi topik-topik sebelumnya, langkah selanjutnya adalah mengendalikan posisi setiap sendi. Untuk menggerakkan sendi robot di Gazebo, kita perlu menerbitkan nilai yang diinginkan ke sendi tersebut dalam tipe pesan std_msgs/Float64 ke topik perintah kontrol posisi sendi. Berikut contoh bagaimana menggerakkan sendi keempat ke posisi 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 memulai dengan CoppeliaSim, langkah pertama adalah mengunduhnya dan mengonfigurasi lingkungan agar siap untuk menjalankan simulasi. CoppeliaSim adalah perangkat lunak lintas platform yang tersedia untuk berbagai sistem operasi seperti Windows, macOS, dan Linux. Perangkat ini dikembangkan oleh Coppelia Robotics GmbH dan tersedia dalam versi lisensi edukasi gratis dan lisensi komersial.

Mengunduh versi terbaru dari CoppeliaSim dari halaman unduhan Coppelia Robotics di http://www.coppeliarobotics.com/downloads.html, pilih versi edu yang kompatibel dengan sistem operasi Linux. Dalam bab ini, kita akan menggunakan versi CoppeliaSim 4.2.0. Setelah selesai mengunduh, ekstrak arsipnya. Buka terminal dan navigasikan ke folder unduhan Anda, lalu jalankan 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 oleh kamera aktif, menerima data visi secara langsung dari kerangka kerja ROS. Selain itu, kita bisa memvisualisasikan aliran video yang dihasilkan oleh CoppeliaSim menggunakan paket image_view, dengan cara menjalankan perintah berikut:

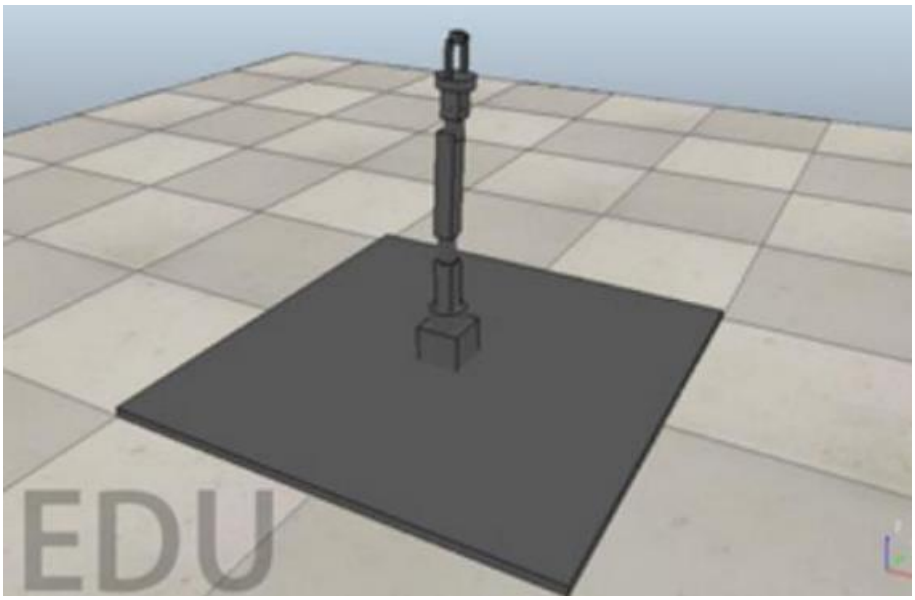
```
rosrun image_view image:=/camera/image_raw
```

Untuk mengirimkan pesan ROS baru dalam skrip Lua, kita harus membungkusnya dalam struktur data yang memiliki bidang yang sama dengan pesan aslinya. Sebaliknya, langkah terbalik diperlukan untuk mengumpulkan informasi yang diterbitkan pada topik ROS. Mari kita tinjau pekerjaan yang telah dilakukan pada contoh sebelumnya sebelum kita beralih ke hal

yang lebih kompleks. Pada contoh dummy_publisher, tujuannya adalah untuk menerbitkan data integer pada topik ROS. Kita bisa memeriksa struktur pesan integer dengan menggunakan perintah ROS ini:

```
rosmsg show std_msgs/Int32 int32 data
```

Pada bab sebelumnya, Gazebo digunakan untuk mengimpor dan mensimulasikan lengan dengan tujuh derajat kebebasan (DOF) yang dirancang dalam Bab 3, Bekerja dengan ROS untuk Pemodelan 3D. Kali ini, kita akan melakukan proses yang serupa dengan menggunakan CoppeliaSim. Langkah pertama untuk mensimulasikan lengan tujuh DOF adalah mengimpornya ke dalam sene simulasi. CoppeliaSim memfasilitasi pengimporan robot baru melalui file URDF; oleh karena itu, kita perlu mengonversi model xacro dari lengan ke dalam format file URDF, kemudian menyimpan file URDF yang dihasilkan di folder urdf dari paket csim_demo_pkg seperti ini:



Untuk mempersiapkan Webots agar dapat diintegrasikan dengan ROS, langkah awal yang perlu dilakukan mirip dengan langkah yang telah dilakukan untuk CoppeliaSim, yaitu instalasi perangkat lunak ini di sistem Anda. Webots merupakan perangkat lunak simulasi yang mendukung berbagai platform seperti Windows, Linux, dan macOS. Awalnya dikembangkan oleh Swiss Federal Institute of Technology, Lausanne (EPFL), kini dikelola oleh Cyberbotics, dan dirilis di bawah lisensi Apache 2 yang gratis dan sumber terbuka. Webots menyediakan lingkungan pengembangan komprehensif untuk memodelkan, memprogram, dan mensimulasikan robot, terutama digunakan dalam konteks profesional, pendidikan, dan penelitian.

Ada beberapa cara untuk melakukan instalasi. Anda dapat mengunduh paket .deb dari halaman unduhan Webots (http://www.cyberbotics.com/#download) atau menggunakan manajer paket Debian/Ubuntu Advanced Packaging Tool (APT). Jika sistem Anda menjalankan Ubuntu, kita dapat memulai dengan melakukan otentikasi repositori Cyberbotics dengan langkah-langkah 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.