

UAS ROBOTIKA 2023/2024

Nama : Nabilah Salwa

NIM : 1103204060

Untuk mengikuti bab ini hingga akhir, satu-satunya persyaratan yang diperlukan adalah sebuah komputer standar yang menggunakan sistem operasi Ubuntu 20.04 LTS atau distribusi Debian 10 GNU/Linux.

Chapter 1: Introduction to ROS Programming Essentials

Persyaratan Teknis:

Komputer standar yang menjalankan distribusi Ubuntu 20.04 LTS atau Debian 10 GNU/Linux.

Pengantar ROS:

ROS (Robot Operating System) adalah kerangka kerja yang fleksibel untuk menulis perangkat lunak robotik.

Dikembangkan pada tahun 2007 oleh Morgan Quigley di Willow Garage, sebuah laboratorium penelitian robotika.

Tujuan: Menetapkan cara standar untuk memprogram robot dan menawarkan komponen perangkat lunak siap pakai untuk integrasi yang mudah ke dalam aplikasi robotik khusus.

Mengapa Menggunakan ROS?

Kemampuan kelas atas:

ROS menyediakan fungsionalitas yang siap digunakan seperti SLAM dan AMCL untuk navigasi otonom dan MoveIt untuk perencanaan gerakan.

Sangat dapat dikonfigurasi dengan berbagai parameter.

Banyak sekali alat:

Ekosistem yang kaya dengan alat seperti rqt_gui, RViz, dan Gazebo untuk debugging, visualisasi, dan simulasi.

Jarang ada kerangka kerja perangkat lunak yang menawarkan seperangkat alat yang begitu banyak.

Dukungan untuk sensor dan aktuator:

Memungkinkan integrasi sensor dan aktuator kelas atas seperti LIDAR 3D, pemindai laser, sensor kedalaman, dll.

Antarmuka yang mulus dengan ROS, menghilangkan kerumitan.

Pengoperasian antar platform:

Middleware pengirim pesan ROS memungkinkan komunikasi antara berbagai program (node).

Node dapat diprogram dalam berbagai bahasa seperti C, C++, Python, atau Java.

Modularitas:

ROS mengimplementasikan pendekatan modular dengan node yang berbeda untuk berbagai proses.

Jika satu node rusak, sistem masih dapat berfungsi.

Penanganan sumber daya secara bersamaan:

ROS menyederhanakan penanganan sumber daya perangkat keras dengan beberapa proses.

Memungkinkan pemrosesan paralel, mengurangi kompleksitas, dan meningkatkan debugging sistem.

Komunitas ROS:

Komunitas yang berkembang pesat dengan pengguna dan pengembang secara global.

Perusahaan robotika besar mengadopsi ROS, bahkan dalam robotika industri, beralih dari aplikasi berpemilik ke ROS.

Tingkat Sistem Berkas ROS:

- Paket: Elemen-elemen sentral yang berisi program ROS, pustaka, file konfigurasi, dll.
- Manifes Paket: Informasi tentang paket, penulis, lisensi, dependensi, dll. (package.xml).

- Metapackages: Mengelompokkan paket-paket terkait tanpa mengandung kode sumber.
- Pesan (.msg) dan Layanan (.srv): Menetapkan jenis pesan dan layanan khusus untuk komunikasi.
- Repositori: Paket ROS yang dikelola menggunakan Sistem Kontrol Versi (VCS) seperti Git, SVN, atau Mercurial.

Struktur Paket ROS:

Struktur paket ROS C++ yang umum meliputi folder untuk config, include, script, src, launch, msg, srv, action, package.xml, dan CMakeLists.txt.

Perintah-perintah ROS untuk Paket:

- catkin_create_pkg: Membuat paket baru.
- rospack: Dapatkan informasi paket.
- catkin_make: Membuat paket.
- rosdep: Menginstal ketergantungan sistem.

ROS Metapackages:

Paket khusus hanya dengan file package.xml.

Mengelompokkan beberapa paket terkait secara logis.

Pesan ROS:

Tipe data yang dideskripsikan menggunakan bahasa deskripsi pesan.

Contohnya termasuk int32, string, float32.

ROS menyediakan tipe pesan bawaan untuk aplikasi umum.

Layanan ROS:

Komunikasi permintaan/respon antara node ROS.

Didefinisikan dalam file .srv, menentukan jenis pesan permintaan dan respons.

Grafik Komputasi ROS:

Komputasi dalam ROS diatur dalam jaringan node yang membentuk grafik komputasi.

Elemen-elemen kunci: Node, Master, Server Parameter, Topik, Layanan, dan Tas.

Master ROS memfasilitasi pendaftaran dan pencarian node.

Node ROS:

Proses dengan komputasi menggunakan pustaka klien ROS.

Toleran terhadap kesalahan, struktur sederhana, dan mengurangi kompleksitas dibandingkan dengan kode monolitik.

Diidentifikasi dengan nama seperti /camera_node.

Topik ROS:

Bus yang diberi nama yang memfasilitasi transportasi pesan antar node.

Node yang menerbitkan dan berlangganan dipisahkan.

Nama-nama unik untuk topik, memungkinkan setiap node untuk mengakses dan mengirim data.

ROS Logging:

Sistem pencatatan untuk menyimpan data (bagfiles) yang penting untuk mengembangkan dan menguji algoritma robot.

Lapisan Grafik ROS:

Paket middleware komunikasi inti dalam tumpukan ros_comm.

Termasuk alat seperti rostopic, rosparam, rosservice, dan rosnod untuk introspeksi.

Memahami Grafik Komputasi ROS:

Representasi grafis yang menunjukkan komunikasi antar node menggunakan topik.

Alat rqt_graph menghasilkan grafik tersebut.

Gambaran umum ini menjelaskan persyaratan teknis, memperkenalkan ROS, menyoroti keunggulannya, mencakup struktur sistem berkas, pembuatan paket, metapaket, pesan, layanan, grafik komputasi, node, topik, dan banyak lagi.

Node ROS:

Node melakukan komputasi menggunakan pustaka klien ROS seperti roscpp dan rospy. Beberapa node dalam sistem robot menangani tugas yang berbeda, sehingga meningkatkan toleransi kesalahan. Node menyederhanakan debugging dan mengurangi kompleksitas dibandingkan dengan kode monolitik.

Penamaan Node:

Tetapkan nama yang berarti untuk node yang sedang berjalan untuk identifikasi, misalnya, /camera_node.

Perintah ROSnode:

- a. Gunakan alat rosnode untuk mengumpulkan informasi:

roscpp info [nama_node]

roscpp kill [nama_node]

daftar roscpp

mesin roscpp [nama_mesin]

roscpp ping

pembersihan roscpp

- b. ***Pesan ROS:***

Pesan adalah struktur data sederhana dengan tipe field.

Akses definisi pesan menggunakan alat rosmg.

rosmg show [message_type]

daftar rosmg

rosmg md5 [tipe_pesan]

rosmg package [nama_paket]

rosmg packages [package_1] [package_2]

- c. ***Topik ROS:***

Komunikasi searah menggunakan topik.

Gunakan alat rostopic untuk mengumpulkan informasi:

rostopic bw /topic

rostopic echo /topic

rostopic find /jenis_pesanan

rostopic hz /topic

rostopic info /topic

daftar rostopik

rostopic pub /topic message_type args

tipe rostopic /topic

d. Layanan ROS:

Komunikasi permintaan/respon.

Akses definisi layanan menggunakan alat rossrv dan rosservice.

ROS Bagfiles:

Gunakan perintah rosbag untuk merekam dan memutar data pesan ROS.

rosbag record [topic_1] [topic_2] -o [bag_name]

rosbag play [bag_name]

ROS Master:

Bertindak sebagai server DNS, mengasosiasikan nama dengan elemen ROS.

Node berkomunikasi dengan ROS Master menggunakan API berbasis XMLRPC.

Server Parameter ROS:

Translated with DeepL.com (free version)

Chapter 2: Getting Started with ROS Programming

Persyaratan Teknis:

- Memerlukan laptop standar dengan sistem operasi Ubuntu 20.04 dan ROS Noetic terinstal.
- Kode referensi untuk bab ini dapat diunduh dari repositori GitHub: <https://github.com/PacktPublishing/Mastering-ROS-for-Robotics-Programming-Third-edition.git>.

```
vboxuser@ZYZZ:~$ sudo apt install ros-noetic-desktop-full
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  autoconf automake autopoint autotools-dev binfmt-support blt bzip2-doc cmake
  cmake-data comerr-dev cpp-8 cython3 debhelper default-libmysqlclient-dev
  dh-autoreconf dh-strip-nondeterminism docutils-common dwz fltk1.3-doc fluid
  fonts-lato fonts-lyx freeglut3 freeglut3-dev gazebo11 gazebo11-common
  gazebo11-plugin-base gcc-8 gcc-8-base gdal-data gettext gfortran gfortran-8
  gfortran-9 gir1.2-gtk-2.0 gir1.2-harfBuzz-0.0 google-mock googletest
  graphviz hddtemp hdf5-helpers i965-va-driver ibverbs-providers icu-devtools
  ignition-tools intel-media-va-driver intltool-debian javascript-common
  krb5-multidev libaacs0 libaec-dev libaec0 libann0 libao0 libapr1
  libapr1-dev libaprutil1 libaprutil1-dev libarchive-cpio-perl
  libarchive-zip-perl libarmadillo-dev libarmadillo9 libarpack2 libarpack2-dev
  libass9 libassimp-dev libassimp5 libassuan-dev libatk1.0-dev libavcodec-dev
  libavcodec58 libavdevice-dev libavdevice58 libavfilter-dev libavfilter7
  libavformat-dev libavformat58 libavresample-dev libavresample4 libavutil-dev
  libavutil56 libbdtplus0 libblas-dev libblas3 libblkid-dev libblkid1
  libbluray2 libboost-all-dev libboost-atomic-dev libboost-atomic1.71-dev
  libboost-atomic1.71.0 libboost-chrono-dev libboost-chrono1.71-dev
  libboost-chrono1.71.0 libboost-container-dev libboost-container1.71-dev
  libboost-container1.71.0 libboost-context-dev libboost-context1.71-dev
  libboost-context1.71.0 libboost-coroutine-dev libboost-coroutine1.71-dev
  libboost-coroutine1.71.0 libboost-date-time-dev libboost-date-time1.71-dev
  libboost-dev libboost-exception-dev libboost-exception1.71-dev
  libboost-fiber-dev libboost-fiber1.71-dev libboost-fiber1.71.0
  libboost-filesystem-dev libboost-filesystem1.71-dev libboost-graph-dev
  libboost-graph-parallel-dev libboost-graph-parallel1.71-dev
  libboost-graph-parallel1.71.0 libboost-graph1.71-dev libboost-graph1.71.0
vboxuser@ZYZZ:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
vboxuser@ZYZZ:~$ sudo apt install curl # if you haven't already installed curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libcurl4
The following NEW packages will be installed:
  curl
The following packages will be upgraded:
  libcurl4
1 upgraded, 1 newly installed, 0 to remove and 250 not upgraded.
Need to get 396 kB of archives.
After this operation, 417 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://id.archive.ubuntu.com/ubuntu focal-updates/main amd64 libcurl4 amd64 7.68.0-1ubuntu2.21 [235 kB]
Get:2 http://id.archive.ubuntu.com/ubuntu focal-updates/main amd64 curl amd64 7.68.0-1ubuntu2.21 [161 kB]
Fetched 396 kB in 1s (738 kB/s)
(Reading database ... 184560 files and directories currently installed.)
Preparing to unpack .../libcurl4_7.68.0-1ubuntu2.21_amd64.deb ...
Unpacking libcurl4:amd64 (7.68.0-1ubuntu2.21) over (7.68.0-1ubuntu2.16) ...
Selecting previously unselected package curl.
Preparing to unpack .../curl_7.68.0-1ubuntu2.21_amd64.deb ...
Unpacking curl (7.68.0-1ubuntu2.21) ...
Setting up libcurl4:amd64 (7.68.0-1ubuntu2.21) ...
Setting up curl (7.68.0-1ubuntu2.21) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
vboxuser@ZYZZ:~$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
OK
vboxuser@ZYZZ:~$ sudo apt update
Hit:1 http://id.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://id.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://id.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:5 http://packages.ros.org/ros/ubuntu focal InRelease [4,679 B]
Get:6 http://packages.ros.org/ros/ubuntu focal/main amd64 Packages [884 kB]
Get:7 http://packages.ros.org/ros/ubuntu focal/main i386 Packages [58.9 kB]
Fetched 867 kB in 5s (168 kB/s)
Reading package lists... Done
Building dependency tree
```

Membuat Paket ROS:

- ROS packages adalah unit dasar dari program ROS.

- Dapat membuat, membangun, dan merilis paket ROS.
- Menggunakan sistem build catkin pada distribusi ROS Noetic.

Membuat Workspace Catkin:

- Buat workspace catkin dengan perintah **mkdir -p ~/catkin_ws/src**.

```
Processing triggers for libc-bin (2.31-0ubuntu9.1) ...
vboxuser@ZZZZ:~$ source /opt/ros/noetic/setup.bash
vboxuser@ZZZZ:~$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
vboxuser@ZZZZ:~$ source ~/.bashrc
vboxuser@ZZZZ:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
vboxuser@ZZZZ:~$ mkdir -p ~/catkin_ws/src
vboxuser@ZZZZ:~$ ls
catkin_ws Desktop Documents Downloads Music Pictures Public Templates Videos
vboxuser@ZZZZ:~$
```

- Sumber lingkungan ROS perlu diaktifkan dengan perintah **source /opt/ros/noetic/setup.bash**.
- Inisialisasi workspace catkin dengan perintah **catkin_init_workspace**.

```
Desktop Documents Downloads Music Pictures Public Templates Videos
vboxuser@ZZZZ:~$ mkdir -p ~/catkin_ws/src
vboxuser@ZZZZ:~$ ls
catkin_ws Desktop Documents Downloads Music Pictures Public Templates Videos
vboxuser@ZZZZ:~$ source /opt/ros/noetic/setup.bash
vboxuser@ZZZZ:~$ cd ~/catkin_ws/src
vboxuser@ZZZZ:~/catkin_ws/src$ catkin_init_workspace
Creating symlink "/home/vboxuser/catkin_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
vboxuser@ZZZZ:~/catkin_ws/src$
```

Membangun Workspace:

- Pindah ke folder workspace src dengan perintah **cd ~/catkin_ws/src**.

```
vboxuser@ZZZZ:~/catkin_ws/src$ cd ~/catkin_ws
vboxuser@ZZZZ:~/catkin_ws$ catkin_make
Base path: /home/vboxuser/catkin_ws
Source space: /home/vboxuser/catkin_ws/src
Build space: /home/vboxuser/catkin_ws/build
Devel space: /home/vboxuser/catkin_ws/devel
Install space: /home/vboxuser/catkin_ws/install
####
#### Running command: "cmake /home/vboxuser/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/vboxuser/catkin_ws/devel -G Unix Makefiles" in "/home/vboxuser/catkin_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/vboxuser/catkin_ws/devel
-- CMAKE_PREFIX_PATH: /home/vboxuser/catkin_ws/devel
-- catkin_make: done
```


- Jalankan **catkin_make** untuk membangun workspace.

```
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/vboxuser/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Found Python3: /usr/lib/python3/dist-packages/cmake
-- Using cmake: /usr/lib/python3/dist-packages/em.py
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Using CATKIN_TEST_RESULTS_DIR: /home/vboxuser/catkin_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Found gmock sources under '/usr/src/gmock': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Found Threads: TRUE
-- Using Python3: /usr/bin/python3
-- CATKIN_ENABLE_TESTING: ON
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/vboxuser/catkin_ws/build
### Running command: "make -j12 -l12" in "/home/vboxuser/catkin_ws/build"
### Generating done
vboxuser@ZZZZ:~/catkin_ws$
```

- Sumberkan file **setup.bash** setiap kali sesi bash baru dimulai.

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

```
vboxuser@ZZZZ:~/catkin_ws$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
vboxuser@ZZZZ:~/catkin_ws$ source ~/.bashrc
vboxuser@ZZZZ:~/catkin_ws$
```

- Setelah membuat paket ini, buat paket tanpa menambahkan node apa pun dengan menggunakan perintah **catkin_make**. Perintah ini harus dijalankan dari ruang kerja catkin jalur. Perintah berikut menunjukkan cara membuat paket ROS kosong kami:

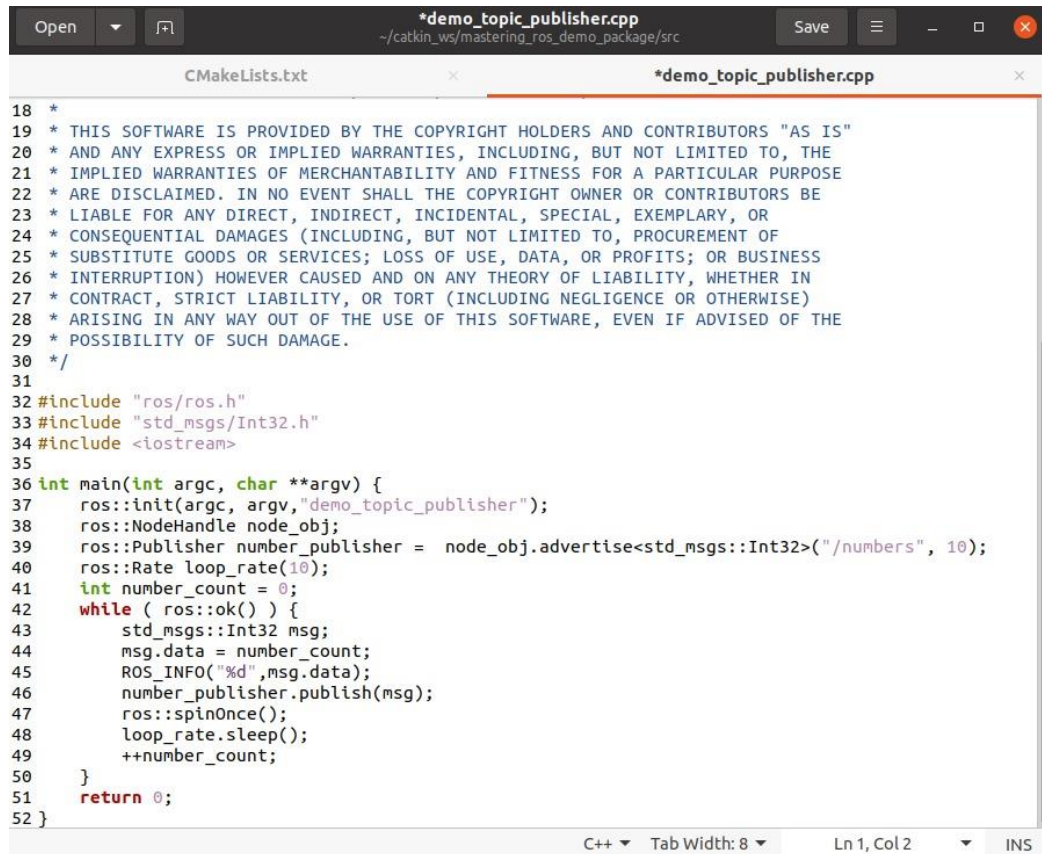
```
cd ~/catkin_ws && catkin_make
```

```
vboxuser@ZZZZ:~/catkin_ws$ cd ~/catkin_ws && catkin_make
Base path: /home/vboxuser/catkin_ws
Source space: /home/vboxuser/catkin_ws/src
Build space: /home/vboxuser/catkin_ws/build
Devel space: /home/vboxuser/catkin_ws/devel
Install space: /home/vboxuser/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/vboxuser/catkin_ws/build"
####
####
#### Running command: "make -j12 -l12" in "/home/vboxuser/catkin_ws/build"
####
vboxuser@ZZZZ:~/catkin_ws$
```

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:



```
18 *
19 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 * POSSIBILITY OF SUCH DAMAGE.
30 */
31
32 #include "ros/ros.h"
33 #include "std_msgs/Int32.h"
34 #include <iostream>
35
36 int main(int argc, char **argv) {
37     ros::init(argc, argv, "demo_topic_publisher");
38     ros::NodeHandle node_obj;
39     ros::Publisher number_publisher = node_obj.advertise<std_msgs::Int32>("/numbers", 10);
40     ros::Rate loop_rate(10);
41     int number_count = 0;
42     while ( ros::ok() ) {
43         std_msgs::Int32 msg;
44         msg.data = number_count;
45         ROS_INFO("%d", msg.data);
46         number_publisher.publish(msg);
47         ros::spinOnce();
48         loop_rate.sleep();
49         ++number_count;
50     }
51     return 0;
52 }
```

Membuat Paket ROS:

- Gunakan perintah **`catkin_create_pkg`** untuk membuat paket ROS.
- Contoh: **`catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs`**.
- Tambahkan dependensi sesuai kebutuhan.

Penggunaan `roscore.xml`:

- File `roscore.xml` mengonfigurasi roscore dan menyimpan parameter serta node dalam grup dengan namespace `/`.

Memahami Output roscore:

- Periksa topik, parameter, dan layanan ROS setelah menjalankan roscore dengan perintah **rostopic list**, **rosparam list**, dan **rosservice list**.

Pengerjaan ROS Nodes:

- Gunakan perintah **catkin_make** untuk membangun paket setelah membuatnya.
- Tambahkan ROS nodes ke folder src dalam paket.

Mengerjakan ROS Topics:

- Topik digunakan sebagai metode komunikasi antara node ROS.
- Gunakan **demo_topic_publisher.cpp** untuk mempublikasikan topik dan **demo_topic_subscriber.cpp** untuk berlangganan.

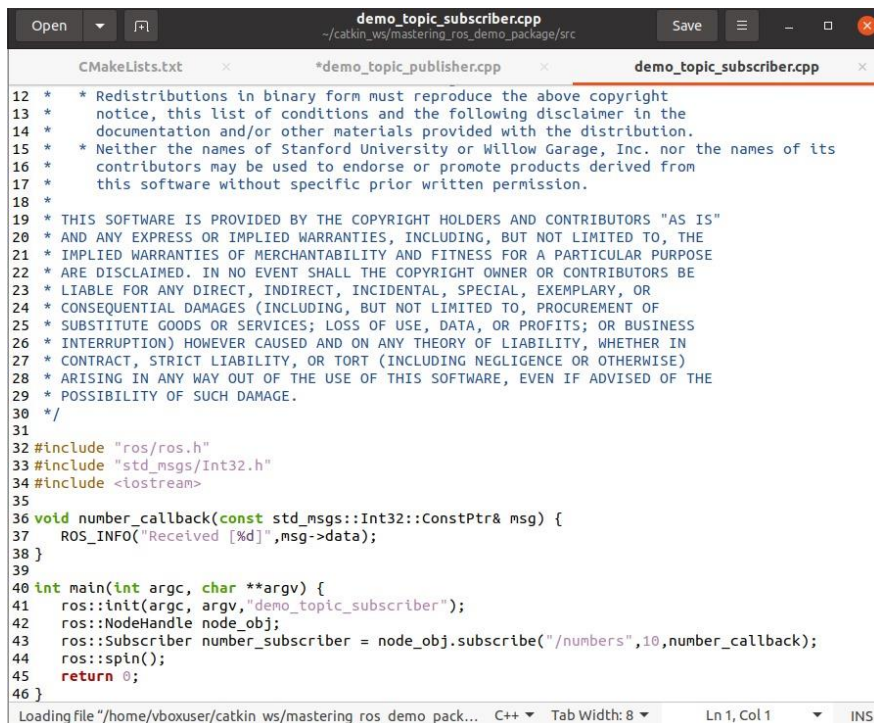
Node demo_topic_publisher.cpp:

- Menginisialisasi node dan Nodehandle.
- Membuat publisher untuk topik `"/numbers"` dengan tipe pesan `std_msgs::Int32`.
- Mengatur frekuensi utama dan loop untuk mempublikasikan nilai integer ke topik `"/numbers"`.
- Menggunakan Ctrl + C untuk menghentikan loop.

Node Publisher (demo_topic_publisher.cpp):

- Memanfaatkan **ROS_INFO** untuk mencetak data pesan.
- Menggunakan **number_publisher.publish(msg)** untuk mempublikasikan pesan ke jaringan ROS.
- Menggunakan **loop_rate.sleep()** untuk memberikan penundaan dan mencapai frekuensi 10 Hz.
- Membahas publisher node yang mempublikasikan nilai integer ke topik `"/numbers"`.

Node Subscriber (demo_topic_subscriber.cpp):

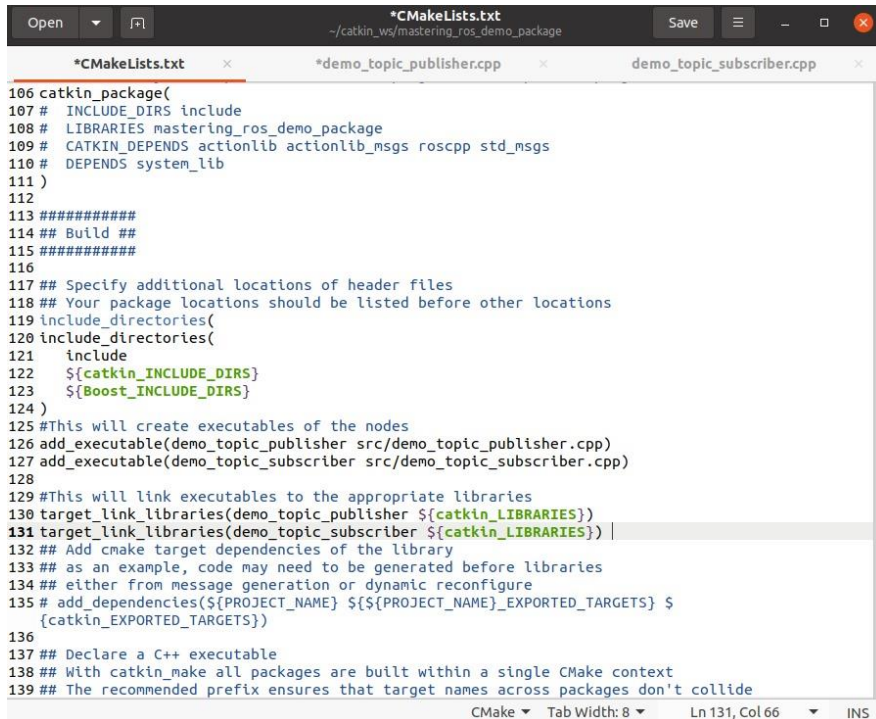


```
12 * * Redistributions in binary form must reproduce the above copyright
13 * notice, this list of conditions and the following disclaimer in the
14 * documentation and/or other materials provided with the distribution.
15 * * Neither the names of Stanford University or Willow Garage, Inc. nor the names of its
16 * contributors may be used to endorse or promote products derived from
17 * this software without specific prior written permission.
18 *
19 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 * POSSIBILITY OF SUCH DAMAGE.
30 */
31
32 #include "ros/ros.h"
33 #include "std_msgs/Int32.h"
34 #include <iostream>
35
36 void number_callback(const std_msgs::Int32::ConstPtr& msg) {
37     ROS_INFO("Received [%d]", msg->data);
38 }
39
40 int main(int argc, char **argv) {
41     ros::init(argc, argv, "demo_topic_subscriber");
42     ros::NodeHandle node_obj;
43     ros::Subscriber number_subscriber = node_obj.subscribe("/numbers", 10, number_callback);
44     ros::spin();
45     return 0;
46 }
```

- Menggunakan **ros::Subscriber** untuk berlangganan ke topik `"/numbers"`.
- Mendefinisikan fungsi **number_callback** yang dijalankan saat pesan datang.
- Menampilkan nilai data dari pesan yang diterima.
- Menggunakan **ros::spin()** untuk menjaga agar node tetap berjalan.

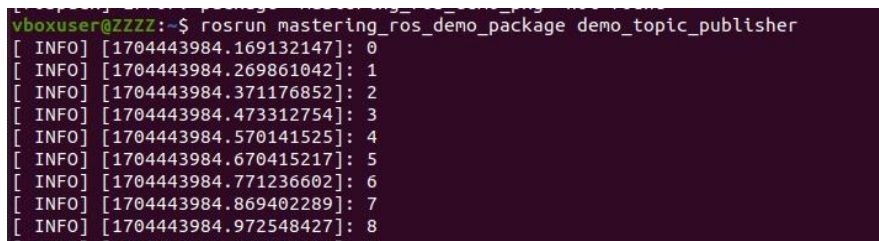
Membangun Nodes:

- Mengedit file **CMakeLists.txt** di dalam paket untuk membangun dan mengompilasi kode sumber.

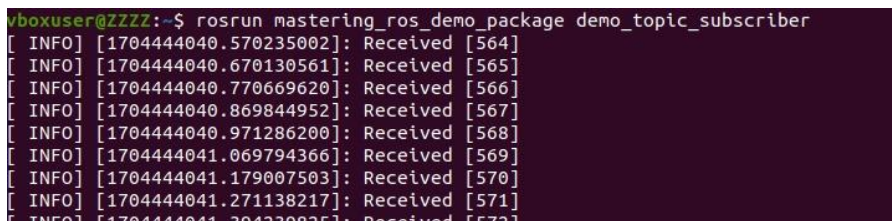


- Menggunakan perintah **catkin_make** di dalam workspace.
- Sekarang, jalankan kedua perintah dalam dua shell. Di penerbit yang sedang berjalan, jalankan yang berikut ini memerintah:

roslaunch mastering_ros_demo_package demo_topic_publisher



Di pelanggan yang sedang berjalan, jalankan perintah berikut:



Menambahkan file .msg dan .srv khusus

Edit file CMakeLists.txt saat ini dan tambahkan baris message_generasi, sebagai berikut:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  message_generation
)
```

Batalkan komentar pada baris berikut dan tambahkan file pesan khusus:

```
--
53 add_message_files(
54   FILES
55   demo_msg.msg
56 )

71 ## Generate added messages and services with any dependencies listed here
72 generate_messages(
```

Untuk memeriksa apakah pesan telah dibuat dengan benar, kita dapat menggunakan perintah rosmmsg:

```
vboxuser@ZZZZ:~$ rosmmsg show mastering_ros_demo_pkg/demo_msg
string greeting
int32 number
```

Mari buat paket menggunakan catkin_make dan uji node dengan mengikuti langkah-langkah berikut:

- Run roscore:
roscore
- Mulai simpul penerbit pesan khusus:

```
vboxuser@ZZZZ:~$ rosrn mastering_ros_demo_pkg demo_msg_publisher
[ INFO] [1704444715.562814120]: 0
[ INFO] [1704444715.563793513]: hello world
[ INFO] [1704444715.665087019]: 1
[ INFO] [1704444715.665213576]: hello world
[ INFO] [1704444715.763894870]: 2
[ INFO] [1704444715.764099006]: hello world
[ INFO] [1704444715.863399092]: 3
[ INFO] [1704444715.863525824]: hello world
[ INFO] [1704444715.964168284]: 4
[ INFO] [1704444715.964649105]: hello world
[ INFO] [1704444716.064010349]: 5
[ INFO] [1704444716.064141334]: hello world
```

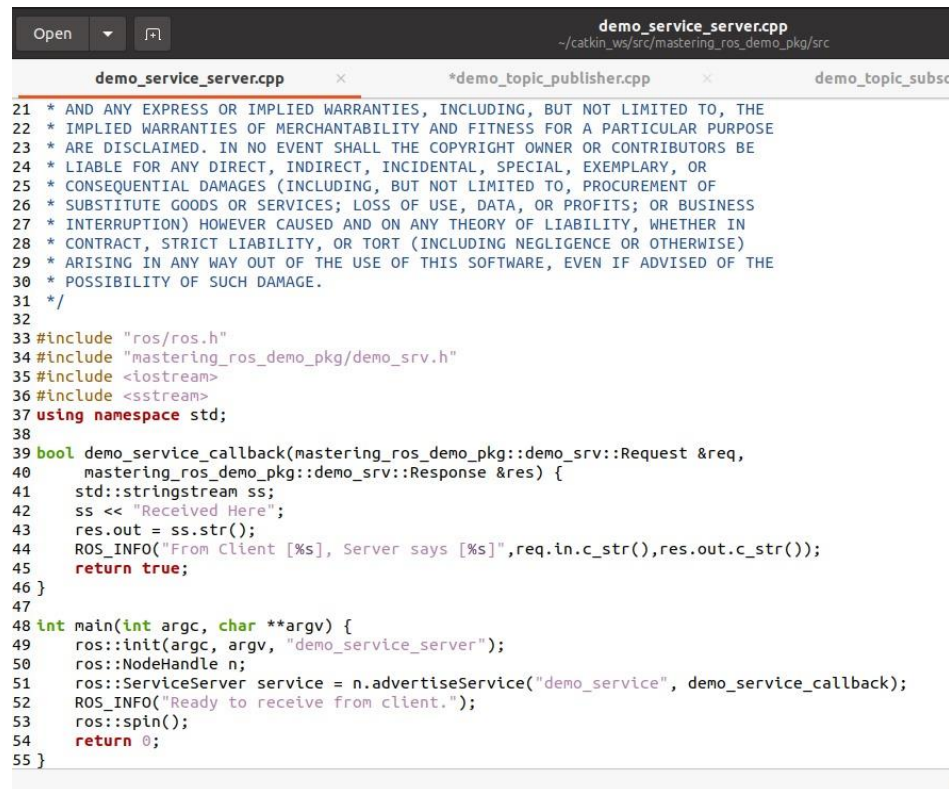
- Mulai node pelanggan pesan khusus:

```
vboxuser@ZZZZ:~$ rosrn mastering_ros_demo_pkg demo_msg_subscriber
[ INFO] [1704444740.762475329]: Recieved greeting [hello world ]
[ INFO] [1704444740.763900943]: Recieved [252]
[ INFO] [1704444740.862952403]: Recieved greeting [hello world ]
[ INFO] [1704444740.863089878]: Recieved [253]
[ INFO] [1704444740.964854220]: Recieved greeting [hello world ]
[ INFO] [1704444740.964970398]: Recieved [254]
[ INFO] [1704444741.063558692]: Recieved greeting [hello world ]
[ INFO] [1704444741.063686851]: Recieved [255]
```

Bekerja dengan Layanan ROS

Arahkan ke `mastering_ros_demo_pkg/src` dan temukan `demo_service_node` `server.cpp` dan `demo_service_client.cpp`.

`demo_service_server.cpp` adalah server, dan definisinya adalah sebagai berikut:



```
demo_service_server.cpp
~/catkin_ws/src/mastering_ros_demo_pkg/src

demo_service_server.cpp x *demo_topic_publisher.cpp x demo_topic_subsc

21 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 * POSSIBILITY OF SUCH DAMAGE.
31 */
32
33 #include "ros/ros.h"
34 #include "mastering_ros_demo_pkg/demo_srv.h"
35 #include <iostream>
36 #include <sstream>
37 using namespace std;
38
39 bool demo_service_callback(mastering_ros_demo_pkg::demo_srv::Request &req,
40 mastering_ros_demo_pkg::demo_srv::Response &res) {
41     std::stringstream ss;
42     ss << "Received Here";
43     res.out = ss.str();
44     ROS_INFO("From Client [%s], Server says [%s]", req.in.c_str(), res.out.c_str());
45     return true;
46 }
47
48 int main(int argc, char **argv) {
49     ros::init(argc, argv, "demo_service_server");
50     ros::NodeHandle n;
51     ros::ServiceServer service = n.advertiseService("demo_service", demo_service_callback);
52     ROS_INFO("Ready to receive from client.");
53     ros::spin();
54     return 0;
55 }
```

Mari kita jelaskan kode ini. Pertama, kami menyertakan file header untuk mendefinisikan layanan yang kami ingin digunakan dalam kode:

```
demo_service_client.cpp x demo_service_server.cpp x
34 #include "mastering_ros_demo_pkg/demo_srv.h"
35 #include <iostream>
36 #include <sstream>
37 using namespace std;
38
39 int main(int argc, char **argv)
40 {
41   ros::init(argc, argv, "demo_service_client");
42   ros::NodeHandle n;
43   ros::Rate loop_rate(10);
44   ros::ServiceClient client = n.serviceClient<mastering_ros_demo_pkg::demo_srv>("demo_service");
45   while (ros::ok())
46   {
47     mastering_ros_demo_pkg::demo_srv srv;
48     std::stringstream ss;
49     ss << "Sending from Here";
50     srv.request.in = ss.str();
51     if (client.call(srv))
52     {
53       ROS_INFO("From Client [%s], Server says [%s]",srv.request.in.c_str(),srv.response.out.c_str());
54     }
55     else
56     {
57       ROS_ERROR("Failed to call service");
58       return 1;
59     }
60   }
61   ros::spinOnce();
62   loop_rate.sleep();
63 }
64 return 0;
65 }
```

Untuk memulai node, pertama-tama jalankan roscore dan gunakan perintah berikut:

```
vboxuser@ZZZZ:~$ roslaunch mastering_ros_demo_pkg demo_service_server
[ INFO ] [1704444910.327215320]: Ready to receive from client.
[ INFO ] [1704444934.197483590]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.298292304]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.398133230]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.498418473]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.598426306]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.699876116]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.798238456]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.89828103]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.995911515]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.096928326]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.197812651]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.298551518]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.398969337]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.499056652]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.597659442]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.697503803]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.799393134]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.898097495]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.997399139]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444936.096304108]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444936.198115340]: From Client [Sending from Here], Server says [Received Here]
```

```
vboxuser@ZZZZ:~$ roslaunch mastering_ros_demo_pkg demo_service_client
[ INFO ] [1704444934.198016799]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.299172695]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.398758106]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.506625015]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.599007201]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.711030949]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.798804983]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.919412382]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444934.998608881]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.097443152]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.198368128]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.299823712]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.403916243]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.499883404]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.598206269]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.698026740]: From Client [Sending from Here], Server says [Received Here]
[ INFO ] [1704444935.802518272]: From Client [Sending from Here], Server says [Received Here]
```


Membuat server tindakan ROS

```
demo_service_client.cpp  demo_service_server.cpp  CMakeLists.txt
32 #include "ros/ros.h"
33 #include <iostream>
34 #include <actionlib/client/simple_action_client.h>
35 #include <actionlib/client/terminal_state.h>
36 #include "mastering_ros_demo_pkg/Demo_actionAction.h"
37
38 int main (int argc, char **argv) {
39     ros::init(argc, argv, "demo_action_client");
40
41     if(argc != 3){
42         ROS_INFO("%d",argc);
43         ROS_WARN("Usage: demo_action_client <goal> <time_to_preempt_in_sec>");
44         return 1;
45     }
46
47     // create the action client
48     // true causes the client to spin its own thread
49     actionlib::SimpleActionClient<mastering_ros_demo_pkg::Demo_actionAction> ac("demo_action",
50
51     ROS_INFO("Waiting for action server to start.");
52
53     // wait for the action server to start
54     ac.waitForServer(); //will wait for infinite time
55
56     ROS_INFO("Action server started, sending goal.");
57
58     // send a goal to the action
59     mastering_ros_demo_pkg::Demo_actionGoal goal;
60     goal.count = atoi(argv[1]);
61
62     ROS_INFO("Sending Goal [%d] and Preempt time of [%d]",goal.count, atoi(argv[2]));
63     ac.sendGoal(goal);
64
65     //wait for the action to return
66     bool finished_before_timeout = ac.waitForResult(ros::Duration(atoi(argv[2])));
67     //Preempting task
```

Buat instance sasaran dan kirimkan nilai sasaran dari baris perintah pertama argumen:

```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(mastering_ros_demo_pkg)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   actionlib
12   actionlib_msgs
13   roscpp
14   std_msgs
15   message_generation
16   message_runtime
17 )
```

Membangun server dan klien tindakan ROS

Kita juga harus menambahkan Boost sebagai ketergantungan sistem:

```
19 ## System dependencies are found with CMake's conventions
20 find_package(Boost REQUIRED COMPONENTS system)
21
```

Kemudian, kita harus menambahkan actionlib_msgs ke generate_messages():

```

65 ## Generate actions in the 'action' folder
66 add_action_files(
67     FILES
68     Demo_action.action
69 )

71 ## Generate added messages and services with any dependencies listed here
72 generate_messages(
73     DEPENDENCIES
74     std_msgs
75     actionlib_msgs
76 )

115 include_directories(
116     include
117     ${catkin_INCLUDE_DIRS}
118     ${Boost_INCLUDE_DIRS}
119 )

```

Terakhir, kita dapat menentukan executable yang dihasilkan setelah kompilasi node ini, beserta dependensi dan pustaka tertautnya:

```

142 target_link_libraries(demo_service_server ${catkin_LIBRARIES})
143 target_link_libraries(demo_service_client ${catkin_LIBRARIES})
144
145 add_executable(demo_action_server src/demo_action_server.cpp)
146 add_dependencies(demo_action_server mastering_ros_demo_pkg_generate_messages_cpp)
147 target_link_libraries(demo_action_server ${catkin_LIBRARIES} )
148
149 add_executable(demo_action_client src/demo_action_client.cpp)
150 add_dependencies(demo_action_client mastering_ros_demo_pkg_generate_messages_cpp)
151 target_link_libraries(demo_action_client ${catkin_LIBRARIES})

```

Setelah catkin_make, kita dapat menjalankan node ini menggunakan perintah berikut:

```

vboxuser@ZZZZ:~$ roslaunch mastering_ros_demo_pkg demo_action_server
[ INFO] [1704445230.547898226]: Starting Demo Action Server
[ INFO] [1704445254.282838955]: /demo_action is processing the goal 10
[ INFO] [1704445254.284753732]: Setting to goal 0 / 10
[ INFO] [1704445254.483327082]: Setting to goal 1 / 10
[ INFO] [1704445254.683034291]: Setting to goal 2 / 10
[ INFO] [1704445254.883563168]: Setting to goal 3 / 10
[ INFO] [1704445255.083016627]: Setting to goal 4 / 10
[ INFO] [1704445255.283044789]: Setting to goal 5 / 10
[ WARN] [1704445255.283373091]: /demo_action got preempted!

```

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:

```
vboxuser@ZZZZ:~$ roscd mastering_ros_demo_pkg/  
vboxuser@ZZZZ:~/catkin_ws/src/mastering_ros_demo_pkg$ ls  
action CMakeLists.txt include launch msg package.xml src srv  
vboxuser@ZZZZ:~/catkin_ws/src/mastering_ros_demo_pkg$ cd launch  
vboxuser@ZZZZ:~/catkin_ws/src/mastering_ros_demo_pkg/launch$ gedit demo_topic.launch  
vboxuser@ZZZZ:~/catkin_ws/src/mastering_ros_demo_pkg/launch$
```

Rekatkan konten berikut ke dalam file

```
demo_topic.launch  
~/catkin_ws/src/mastering_ros_demo_pkg/launch  
1 <?xml version="1.0" ?>  
2 <launch>  
3   <node name="publisher_node" pkg="mastering_ros_demo_pkg" type="demo_topic_publisher" output="screen" />  
4   <node name="subscriber_node" pkg="mastering_ros_demo_pkg" type="demo_topic_subscriber" output="screen" />  
5 </launch>
```

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

```
vboxuser@ZZZZ:~$ roslaunch mastering_ros_demo_pkg demo_topic.launch  
... logging to /home/vboxuser/.ros/log/9921bdda-aba7-11ee-8f84-a7295b31cd14/roslaunch-ZZZZ-31563.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://ZZZZ:44757/  
  
SUMMARY  
=====  
  
PARAMETERS  
* /rostdistro: noetic  
* /rosversion: 1.16.0  
  
NODES  
/  
  publisher_node (mastering_ros_demo_pkg/demo_topic_publisher)  
  subscriber_node (mastering_ros_demo_pkg/demo_topic_subscriber)  
  
ROS_MASTER_URI=http://localhost:11311  
  
process[publisher_node-1]: started with pid [31577]  
process[subscriber_node-2]: started with pid [31578]  
[ INFO] [1704445382.050832743]: 0  
[ INFO] [1704445382.152601799]: 1  
[ INFO] [1704445382.253402768]: 2  
[ INFO] [1704445382.351550166]: 3
```

Chapter 3: Working with ROS for 3D Modeling

Persyaratan Teknis:

Untuk mengikuti contoh-contoh pada bab ini, diperlukan laptop standar yang menjalankan Ubuntu 20.04 dengan ROS Noetic terinstal. Kode referensi untuk bab ini dapat diunduh dari repositori Git di <https://github.com/PacktPublishing/Mastering-ROS-for-Robotics-Programming-Third-edition.git>.

Kode tersebut terdapat di dalam folder Chapter3/mastering_ros_robot_description_pkg/.

Paket-paket ROS untuk Pemodelan Robot:

Paket-paket ROS yang penting untuk membangun dan memodelkan robot termasuk urdf, joint_state_publisher, joint_state_publisher_gui, kdl_parser, robot_state_publisher, dan xacro. Paket-paket ini sangat penting untuk membuat, memvisualisasikan, dan berinteraksi dengan model robot.

Memahami Pemodelan Robot menggunakan URDF:

tag tautan: Mewakili satu tautan robot, termasuk properti seperti ukuran, bentuk, warna, dan properti dinamis. Terdiri dari bagian inersia, visual, dan tabrakan.

tag sambungan: Merepresentasikan sendi robot yang menghubungkan dua tautan. Mendukung berbagai jenis sambungan (berputar, kontinu, prismatic, tetap, mengambang, planar). Mendefinisikan kinematika, dinamika, dan batas gerakan.

tag robot: Mengenkapsulasi seluruh model robot, yang berisi link dan sendi.

tag gazebo: Digunakan untuk menyertakan parameter simulasi Gazebo di dalam URDF, termasuk plugin gazebo dan properti material.

Visualisasi elemen URDF termasuk tautan, sambungan, dan model robot.

Tag URDF dan detail lebih lanjut dapat ditemukan di <http://wiki.ros.org/urdf/XML>.

Langkah selanjutnya:

Bagian selanjutnya akan melibatkan pembuatan paket ROS baru yang berisi deskripsi robot yang berbeda.

Membuat paket ROS untuk robot keterangan:

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

```
vboxuser@ZZZZ:~$ cd catkin_ws/  
vboxuser@ZZZZ:~/catkin_ws$ catkin_create_pkg mastering ros_robot_description_pkg roscpp tf geometry_msgs urdf rviz xacro  
Created file mastering/package.xml  
Created file mastering/CMakeLists.txt  
Created folder mastering/include/mastering  
Created folder mastering/src  
Successfully created files in /home/vboxuser/catkin_ws/mastering. Please adjust the values in package.xml.  
vboxuser@ZZZZ:~/catkin_ws$
```

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

```
vboxuser@ZZZZ:~/catkin_ws$ sudo apt-get install ros-noetic-xacro  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
ros-noetic-xacro is already the newest version (1.14.16-1focal.20230620.185428).  
ros-noetic-xacro set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 220 not upgraded.
```

```
vboxuser@ZZZZ:~/catkin_ws$ sudo apt-get install ros-noetic-urdf  
[sudo] password for vboxuser:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
ros-noetic-urdf is already the newest version (1.13.2-1focal.20230620.185459).  
ros-noetic-urdf set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 220 not upgraded.
```

Membuat model URDF pertama kami

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:

```
Open pan_tilt.urdf Save
~/catkin_ws/mastering_ros_robot_description_pkg/urdf

1 <?xml version="1.0"?>
2 <robot name="pan_tilt">
3
4   <link name="base_link">
5
6     <visual>
7       <geometry>
8         <cylinder length="0.01" radius="0.2"/>
9       </geometry>
10      <origin rpy="0 0 0" xyz="0 0 0"/>
11      <material name="yellow">
12        <color rgba="1 1 0 1"/>
13      </material>
14    </visual>
15
16    <collision>
17      <geometry>
18        <cylinder length="0.03" radius="0.2"/>
19      </geometry>
20      <origin rpy="0 0 0" xyz="0 0 0"/>
21    </collision>
22    <inertial>
23      <mass value="1"/>
24      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
25    </inertial>
26  </link>
27
28  <joint name="pan_joint" type="revolute">
29    <parent link="base_link"/>
30    <child link="pan_link"/>
31    <origin xyz="0 0 0.1"/>
32    <axis xyz="0 0 1" />
33    <limit effort="300" velocity="0.1" lower="-3.14" upper="3.14"/>
34    <dynamics damping="50" friction="1"/>
35  </joint>
36
37  <link name="pan_link">
```

Menjelaskan file URDF

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

```
vboxuser@ZZZZ:~/catkin_ws$ git clone https://github.com/qboticslabs/mastering_ros_3rd_edition
Command 'git' not found, but can be installed with:

apt install git
Please ask your administrator.

vboxuser@ZZZZ:~/catkin_ws$ check_urdf pan_tilt.urdf
Command 'check_urdf' not found, but can be installed with:

apt install liburdfdom-tools
Please ask your administrator.

vboxuser@ZZZZ:~/catkin_ws$ sudo apt-get install liburdfdom-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
 liburdfdom-tools
0 upgraded, 1 newly installed, 0 to remove and 220 not upgraded.
Need to get 12.7 kB of archives.
After this operation, 62.5 kB of additional disk space will be used.
Get:1 http://id.archive.ubuntu.com/ubuntu focal/universe amd64 liburdfdom-tools amd64 1.0.4+ds-2 [12.7 kB]
Fetched 12.7 kB in 1s (17.3 kB/s)
Selecting previously unselected package liburdfdom-tools.
(Reading database ... 265117 files and directories currently installed.)
Preparing to unpack .../liburdfdom-tools_1.0.4+ds-2_amd64.deb ...
Unpacking liburdfdom-tools (1.0.4+ds-2) ...
Setting up liburdfdom-tools (1.0.4+ds-2) ...
Processing triggers for man-db (2.9.1-1) ...
vboxuser@ZZZZ:~/catkin_ws$
```

Berinteraksi dengan sambungan pan-and-tilt

Kita dapat memasukkan node ini ke dalam file peluncuran menggunakan pernyataan berikut. Batasan dari pan-and-tilt harus disebutkan di dalam tag gabungan:

```

28 <joint name="pan_joint" type="revolute">
29   <parent link="base_link"/>
30   <child link="pan_link"/>
31   <origin xyz="0 0 0.1"/>
32   <axis xyz="0 0 1"/>
33   <limit effort="300" velocity="0.1" lower="-3.14" upper="3.14"/>
34   <dynamics damping="50" friction="1"/>
35 </joint>
36
37 <link name="pan_link">
38   <visual>
39     <geometry>
40       <cylinder length="0.4" radius="0.04"/>
41     </geometry>
42     <origin rpy="0 0 0" xyz="0 0 0.09"/>
43     <material name="red">
44       <color rgba="0 0 1 1"/>
45     </material>
46   </visual>
47   <collision>
48     <geometry>
49       <cylinder length="0.4" radius="0.06"/>
50     </geometry>
51     <origin rpy="0 0 0" xyz="0 0 0.09"/>
52   </collision>
53   <inertial>
54     <mass value="1"/>
55     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
56   </inertial>
57 </link>
58
59 <joint name="tilt_joint" type="revolute">
60   <parent link="pan_link"/>
61   <child link="tilt_link"/>
62   <origin xyz="0 0 0.2"/>
63   <axis xyz="0 1 0"/>
64   <limit effort="300" velocity="0.1" lower="-4.64" upper="-1.5"/>
65   <dynamics damping="50" friction="1"/>
66 </joint>
67
68 <link name="tilt_link">
69   <visual>
70     <geometry>
71       <cylinder length="0.4" radius="0.04"/>
72     </geometry>
73     <origin rpy="0 1.5 0" xyz="0 0 0"/>
74     <material name="green">
75       <color rgba="1 0 0 1"/>
76     </material>
77   </visual>
78   <collision>
79     <geometry>
80       <cylinder length="0.4" radius="0.06"/>
81     </geometry>
82     <origin rpy="0 1.5 0" xyz="0 0 0"/>
83   </collision>
84   <inertial>
85     <mass value="1"/>
86     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
87   </inertial>

```

Chapter 4: Simulating Robots Using ROS and Gazebo

Berikut ini adalah perincian yang disederhanakan dari teks yang diberikan:

Persyaratan Teknis:

- Laptop standar dengan Ubuntu 20.04 dan ROS Noetic.
- Kode tersedia di Git: [tautan](#).
- Model simulasi dalam folder Bab4/seven_dof_arm_gazebo.
- Lihat kode yang sedang bekerja: [tautan](#).

Simulasi Lengan Robot di Gazebo dan ROS:

- Merancang lengan tujuh DOF pada bab sebelumnya.
- Simulasi di Gazebo menggunakan ROS.
- Menginstal paket-paket yang diperlukan untuk Gazebo dan ROS.
- Setelah instalasi, periksa apakah Gazebo sudah terpasang dengan benar menggunakan perintah berikut:

```
ubuntu@vboxuser:~$ roscore & roslaunch gazebo_ros gazebo
[1] 32835
... logging to /home/vboxuser/.ros/log/58726a36-abad-11ee-8f84-a7295b31cd14/roslaunch-ZZZZ-32835.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://ZZZZ:44263/
ros_comm version 1.16.0

SUMMARY
=====
PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [32877]
ROS_MASTER_URI=http://ZZZZ:11311/

setting /run_id to 58726a36-abad-11ee-8f84-a7295b31cd14
process[rosout-1]: started with pid [32890]
started core service [/rosout]
[INFO] [1704447150.676787140]: Finished loading Gazebo ROS API Plugin.
[INFO] [1704447150.679638252]: waitForService: Service [/gazebo/set_physics_properties] has not been advertised, waiting...
[INFO] [1704447151.352580448]: waitForService: Service [/gazebo/set_physics_properties] is now available.
[INFO] [1704447152.161573268]: Physics dynamic reconfigure ready.
```

Membuat Model Simulasi Lengan Robot untuk Gazebo:

- Membuat paket untuk mensimulasikan lengan robot.
- Model simulasi dalam file seven_dof_arm.xacro.

Menambahkan Warna dan Tekstur pada Model Robot Gazebo:

- Tentukan warna dan tekstur dalam file .xacro.

Menambahkan Tag Transmisi untuk Menggerakkan Model:

- Tentukan elemen transmisi untuk menghubungkan aktuator ke sendi.

Menambahkan Plugin Gazebo_ros_control:

- Tambahkan plugin gazebo_ros_control untuk mengurai tag transmisi.

Menambahkan Sensor Visi 3D ke Gazebo:

- Mengintegrasikan sensor penglihatan 3D (Asus Xtion Pro) di Gazebo.

Mensimulasikan Lengan Robot dengan Xtion Pro:

- Luncurkan simulasi lengkap dengan sensor Xtion Pro.

Memvisualisasikan Data Sensor 3D:

- Melihat gambar RGB, IR, dan kedalaman.
- Memvisualisasikan data point cloud di RViz.

Menggerakkan Sendi Robot menggunakan Pengontrol ROS di Gazebo:

- Mengonfigurasi pengontrol ROS untuk status dan posisi sendi.
- Gambaran umum tentang pengontrol ROS dan antarmuka perangkat keras.
- Interaksi pengontrol ROS dengan Gazebo.

Menghubungkan Pengontrol Status Gabungan dan Pengontrol Posisi Gabungan:

- File konfigurasi untuk pengontrol status dan posisi bersama.
- Definisi pengontrol untuk setiap sambungan dengan penguatan PID.

```

seven_dof_arm_gazebo_control.yaml
~/Downloads/Mastering-ROS-for-Rob...ter4/seven_dof_arm_gazebo/config

1 seven_dof_arm:
2   # Publish all joint states -----
3   joint_state_controller:
4     type: joint_state_controller/JointStateController
5     publish_rate: 50
6
7   # Position Controllers -----
8   joint1_position_controller:
9     type: position_controllers/JointPositionController
10    joint: shoulder_pan_joint
11    pid: {p: 100.0, i: 0.01, d: 10.0}
12   joint2_position_controller:
13     type: position_controllers/JointPositionController
14     joint: shoulder_pitch_joint
15     pid: {p: 100.0, i: 0.01, d: 10.0}
16   joint3_position_controller:
17     type: position_controllers/JointPositionController
18     joint: elbow_roll_joint
19     pid: {p: 100.0, i: 0.01, d: 10.0}
20   joint4_position_controller:
21     type: position_controllers/JointPositionController
22     joint: elbow_pitch_joint
23     pid: {p: 100.0, i: 0.01, d: 10.0}
24   joint5_position_controller:
25     type: position_controllers/JointPositionController
26     joint: wrist_roll_joint
27     pid: {p: 100.0, i: 0.01, d: 10.0}
28   joint6_position_controller:
29     type: position_controllers/JointPositionController
30     joint: wrist_pitch_joint
31     pid: {p: 100.0, i: 0.01, d: 10.0}
32   joint7_position_controller:
33     type: position_controllers/JointPositionController
34     joint: gripper_roll_joint
35     pid: {p: 100.0, i: 0.01, d: 10.0}
36

```

Meluncurkan pengendali ROS dengan Gazebo:

- Buat berkas peluncuran di direktori seven_dof_arm_gazebo/launch.
- Sertakan peluncuran Gazebo dan muat konfigurasi pengontrol bersama dari file YAML.
- Muat pengontrol menggunakan paket controller_manager.
- Jalankan penerbit status robot untuk status gabungan dan transformasi.

```

seven_dof_arm_gazebo_control.launch
~/Downloads/Mastering-ROS-for-Rob...ter4/seven_dof_arm_gazebo/launch

1 ?xml version="1.0" ?
2
3 <launch>
4   <!-- Launch Gazebo -->
5   <include file="$(find seven_dof_arm_gazebo)/launch/seven_dof_arm_world.launch" />
6
7
8   <!-- Load joint controller configurations from YAML file to parameter server -->
9   <rosparam files="$(find seven_dof_arm_gazebo)/config/seven_dof_arm_gazebo_control.yaml" command="load"/>
10
11
12   <!-- load the controllers -->
13   <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
14     output="screen" ns="/seven_dof_arm" args="joint_state_controller
15     joint1_position_controller
16     joint2_position_controller
17     joint3_position_controller
18     joint4_position_controller
19     joint5_position_controller
20     joint6_position_controller
21     joint7_position_controller"/>
22
23
24   <!-- convert joint states to TF transforms for rviz, etc -->
25   <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
26     respawn="false" output="screen">
27     <remap from="/joint_states" to="/seven_dof_arm/joint_states" />
28   </node>
29
30 </launch>

```

Memeriksa Topik Pengontrol:

- Gunakan `roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo_control.launch` untuk memeriksa topik pengontrol.
- Konfirmasikan peluncuran yang berhasil dengan pesan spesifik di terminal.
- Topik yang dihasilkan termasuk perintah pengontrol posisi untuk setiap sendi.

Menggerakkan Sendi Robot:

- Perintahkan setiap sendi dengan menerbitkan nilai yang diinginkan ke topik perintah pengontrol posisi sendi.
- Contoh: `rostopic pub /seven_dof_arm/joint4_position_controller/command std_msgs/Float64 1.0`.
- Lihat status gabungan dengan `rostopic echo /seven_dof_arm/joint_states`.

Mensimulasikan Robot Beroda Diferensial di Gazebo:

- Siapkan simulasi untuk robot beroda diferensial.
- Buat file peluncuran di `diff_wheeled_robot_gazebo/launch`.
- Luncurkan menggunakan `roslaunch diff_wheeled_robot_gazebo diff_wheeled_robot_gazebo.launch`.
- Memvisualisasikan robot di Gazebo.

Menambahkan Pemindai Laser ke Gazebo:

- Ubah `diff_wheeled_robot.xacro` untuk menyertakan pemindai laser.
- Konfigurasi informasi khusus Gazebo untuk plugin pemindai laser.
- Memvisualisasikan data pemindai laser dengan objek yang ditambahkan di Gazebo.

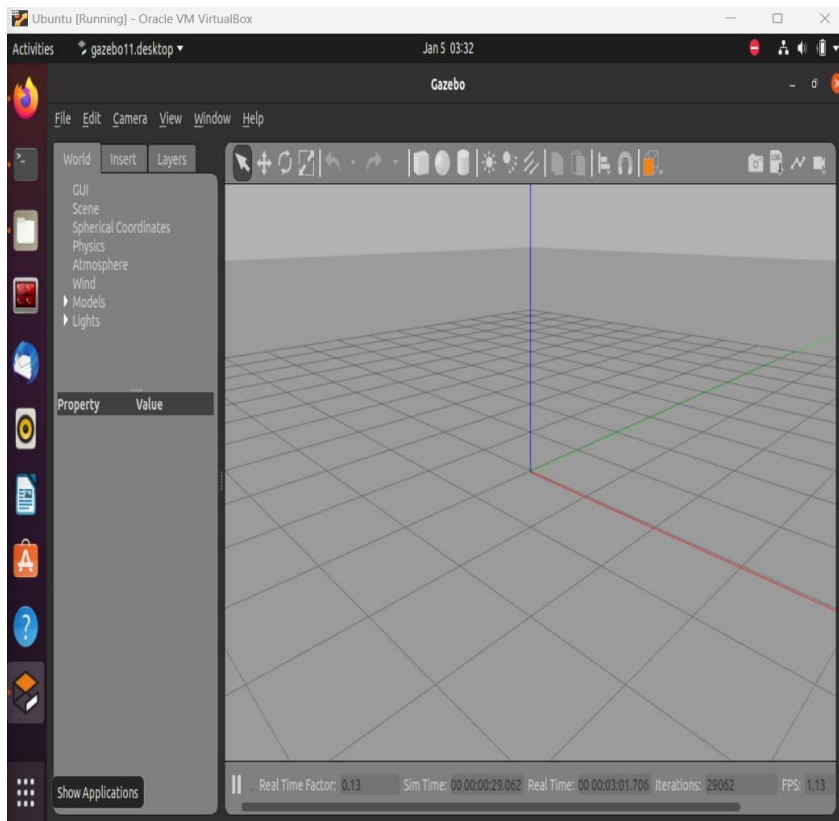
Memindahkan Robot Bergerak di Gazebo:

- Tambahkan plugin `libgazebo_ros_diff_drive.so` untuk perilaku penggerak diferensial.
- Tentukan parameter seperti sambungan roda, pemisahan, diameter, dll.
- Sertakan penerbit status gabungan dalam file peluncuran.

Node Teleop ROS:

- Gunakan node `diff_wheeled_robot_key` untuk teleoperasi.
- Sesuaikan skala linier dan sudut.

- Luncurkan teleop dengan `roslaunch diff_wheeled_robot_control keyboard_teleop.launch`.



Visualisasi di RViz:

- Gunakan RViz untuk memvisualisasikan status robot dan data laser.
- Atur Bingkai Tetap ke /odom dan tambahkan Pemindaian Laser dengan topik /scan.
- Tambahkan model Robot untuk dilihat.

Memindahkan Robot:

- Gunakan tombol di terminal teleop (U, I, O, J, K, L, M, koma, titik) untuk penyesuaian arah.
- Gunakan tombol lain (Q, Z, W, X, E, C, K, spasi) untuk penyesuaian kecepatan.

Menjelajahi Area:

- Robot hanya bergerak jika tombol yang sesuai ditekan di terminal simulasi teleop.
- Jelajahi area menggunakan robot yang dioperasikan secara teleop dan visualisasikan data laser di RViz.

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

Persyaratan Teknis:

- Laptop standar dengan Ubuntu 20.04 dan ROS Noetic.
- Unduh kode dari: Menguasai-ROS-untuk-Pemrograman-Robotika-Edisi-ketiga. Gunakan kode dari folder Bab5/csim_demo_pkg dan Bab5/webost_demo_pkg.
- Melihat kode yang sedang bekerja: Kode Bab 5.

Menyiapkan CoppeliaSim dengan ROS:

- Unduh dan ekstrak CoppeliaSim 4.2.0 dari halaman unduhan Coppelia Robotics, pilih versi edu untuk Linux.
- Pindah ke folder unduhan dan jalankan: `tar vxf CoppeliaSim_Edu_V4_2_0_Ubuntu20_04.tar.xz`.
- Ganti nama folder untuk kenyamanan: `mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim`.
- Atur variabel lingkungan COPPELIASIM_ROOT: `echo "export COPPELIASIM_ROOT=/path/to/CoppeliaSim/folder" >> ~/.bashrc`.

Mode CoppeliaSim untuk Robot Simulasi:

- API jarak jauh: Fungsi yang dapat dipanggil dari aplikasi eksternal (C/C++, Python, Lua, MATLAB). Membutuhkan sisi klien (aplikasi eksternal) dan server (skrip CoppeliaSim).
- RosInterface: Antarmuka saat ini untuk komunikasi ROS, menggantikan plugin ROS yang sudah usang. Mereplikasi fungsi API jarak jauh.

Memulai CoppeliaSim dengan ROS:

- Jalankan roscore sebelum membuka CoppeliaSim.
- Memulai CoppeliaSim: `cd $COPPELIASIM_ROOT && ./coppeliaSim.sh`.

Memeriksa Pengaturan:

- Verifikasi node ROS yang aktif setelah meluncurkan CoppeliaSim.

Berinteraksi dengan CoppeliaSim menggunakan Topik ROS:

- Gunakan topik ROS untuk mengirim/menerima data ke/dari objek simulasi.
- Objek CoppeliaSim dapat dikaitkan dengan skrip Lua untuk dieksekusi selama simulasi.
- Skrip Lua menggunakan simROS untuk berinteraksi dengan ROS.

Memahami Plugin RosInterface:

- Bagian dari kerangka kerja API CoppeliaSim.
- Plugin ROS harus dimuat selama startup CoppeliaSim.
- Jelajahi fungsi plugin RosInterface menggunakan scene plugin_publisher_subscriber.ttt.

Berinteraksi dengan CoppeliaSim menggunakan Topik ROS (lanjutan):

- Gunakan skrip Lua untuk mempublikasikan dan berlangganan topik ROS.
- Contoh: skrip dummy_publisher dan dummy_subscriber menukar data bilangan bulat pada topik /number.

Bekerja dengan Pesan ROS:

- Membungkus pesan ROS dalam skrip Lua untuk menerbitkan dan mengekstrak informasi.
- Contoh: Menerbitkan gambar dari sensor kamera dalam adegan simulasi.

Mensimulasikan Lengan Robotik menggunakan CoppeliaSim dan ROS:

- Mengimpor model URDF lengan tujuh-DOF ke dalam CoppeliaSim.
- Aktifkan motor untuk gerakan sendi. Menyetel penguatan PID untuk kinerja loop kontrol.
- Menguji gerakan sendi dengan mengatur posisi target.

Menambahkan Antarmuka ROS ke Pengontrol Bersama CoppeliaSim:

- Antarmuka lengan tujuh-DOF dengan plugin RosInterface untuk kontrol bersama.
- Gunakan skrip Lua untuk mempublikasikan status gabungan dan berlangganan perintah gabungan melalui topik ROS.
- Contoh: sysCall_init menginisialisasi penanganan sambungan dan mengatur penerbit/pelanggan.
- Mengontrol sambungan menggunakan perintah ROS, misalnya, rostopic pub /csim_demo/seven_dof_arm/elbow_pitch/cmd std_msgs/Float32 "data: 1.0".

- Memantau status sambungan, misalnya, rostopic echo /csim_demo/seven_dof_arm/elbow_pitch/state.

Membuat Webots dengan ROS

1. **Instalasi Webots:** Unduh Webots dari situs web resmi (<http://www.cyberbotics.com/#download>) atau gunakan Debian/Ubuntu APT package manager 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
2. **Mulai Webots:** Ketikkan perintah berikut untuk membuka antarmuka pengguna Webots:

\$ webots
3. **Mengenal Simulasi Webots:**
 - Konfigurasi Dunia: Gunakan file konfigurasi dunia (.wbt) untuk mendefinisikan lingkungan simulasi.
 - Kontroler: Setiap simulasi diatur oleh satu atau lebih program kontroler yang dapat diimplementasikan dalam bahasa seperti C, C++, Python, atau Java.
 - Plugin Fisik: Modifikasi perilaku fisik simulasi menggunakan plugin yang ditulis dalam bahasa yang sama dengan kontroler.

Simulasi Robot Bergerak dengan Webots

1. **Membuat Adegan Simulasi:** Gunakan wizard untuk membuat adegan simulasi baru dengan memilih Wizards | New Project Directory dan mengonfigurasi direktori dan nama proyek.
2. **Menambahkan Objek ke Adegan:**
 - Pilih RectangleArea dari panel hirarki.
 - Klik tombol + untuk menambahkan objek dan pilih PROTO nodes | objects | factory | containers | WoodenBox (Solid).

- Pilih RectangleArea dan tambahkan robot e-puck dengan memilih (Webots Projects) / robots / gctronic / e-puck / E-puck PROTO.

3. Konfigurasi Objek:

- Klik dua kali pada RectangleArea untuk mengubah ukuran lantai.
- Konfigurasi objek, seperti WoodenBox, melalui properti.

Menulis Kontroler Pertama Anda

1. Buat Kontroler Baru:

- Gunakan Wizards | New Robot Controller dan pilih C++.
- Compile kontroler menggunakan tombol Build.

2. Implementasikan Kontroler:

- Gunakan kontroler berikut untuk menggerakkan robot e-puck secara maju-mundur:
- // Kode kontroler C++ di sini

Simulasi Lengan Robot dengan Webots dan ROS

1. Instalasi Webots-ROS Package:

```
sudo apt-get install ros-noetic-webots-ros
```

2. Mengganti Kontroler Webots:

- Ganti kontroler robot Webots dengan kontroler ROS.

Menulis Node Teleop Menggunakan webots_ros

1. Buat Node ROS:

- Implementasikan node ROS untuk mengontrol kecepatan roda e-puck berdasarkan pesan geometry_msgs::Twist.
- Gunakan layanan Webots untuk mengatur kecepatan dan posisi roda.

Memulai Webots dengan Berkas Peluncuran

1. Berkas Peluncuran Webots:

- Gunakan berkas peluncuran yang disertakan di webots_ros untuk memulai Webots dan konfigurasi adegan simulasi.

Pastikan untuk menyesuaikan setiap langkah dengan versi perangkat lunak yang digunakan dan rincian spesifik sistem Anda.