

Chapter 1 Introduction to ROS

Nama : Al Ghifary Akmal Nasheer

NIM : 1103201242

Kelas : TK-44-06

Why should we use ROS?

Sistem Operasi Robot (ROS) adalah kerangka kerja yang fleksibel yang menyediakan berbagai alat dan pustaka untuk menulis perangkat lunak robot. ROS menawarkan beberapa fitur kuat untuk membantu pengembang dalam tugas seperti pertukaran pesan, komputasi terdistribusi, penggunaan kode ulang, dan implementasi algoritma terkini untuk aplikasi robotika. Proyek ROS dimulai pada tahun 2007 oleh Morgan Quigley dan pengembangannya dilanjutkan di Willow Garage, laboratorium penelitian robotika untuk mengembangkan perangkat keras dan perangkat lunak sumber terbuka untuk robot. Tujuan ROS adalah membentuk cara standar untuk memprogram robot sambil menawarkan komponen perangkat lunak siap pakai yang dapat dengan mudah diintegrasikan dengan aplikasi robotika kustom.

Ada banyak alasan untuk memilih ROS sebagai kerangka pemrograman, dan beberapa di antaranya adalah sebagai berikut:

- Kemampuan tinggi: ROS dilengkapi dengan fungsionalitas siap pakai, seperti paket Simultaneous Localization and Mapping (SLAM) dan Adaptive Monte Carlo Localization (AMCL) yang dapat digunakan untuk navigasi otonom pada robot mobile. Selain itu, paket MoveIt dapat digunakan untuk perencanaan gerakan pada manipulator robot. Fungsionalitas ini dapat langsung digunakan dalam perangkat lunak robot tanpa kesulitan.
- Banyaknya alat: Ekosistem ROS penuh dengan berbagai alat untuk debugging, visualisasi, dan simulasi. Alat-alat seperti rqt_gui, RViz, dan Gazebo merupakan beberapa alat sumber terbuka yang kuat untuk debugging, visualisasi, dan simulasi.
- Dukungan untuk sensor dan aktuator canggih: ROS memungkinkan penggunaan berbagai driver perangkat dan paket antarmuka berbagai sensor dan aktuator dalam robotika, termasuk sensor 3D LIDAR, pemindai laser, sensor kedalaman, aktuator, dan lainnya.
- Interoperabilitas antar platform: Middleware pertukaran pesan ROS memungkinkan komunikasi antara program-program berbeda melalui apa yang disebut sebagai "nodes." Nodes ini dapat diprogram dalam bahasa apa pun yang memiliki pustaka klien ROS.
- Modularitas: ROS mendukung pendekatan modular dengan menulis nodes terpisah untuk setiap proses. Jika satu node mengalami kegagalan, sistem masih dapat berfungsi.

- Penanganan sumber daya konkuren: Dalam ROS, penanganan sumber daya perangkat keras melalui lebih dari dua proses dapat dilakukan dengan mudah melalui penggunaan ROS topics. Setiap node dapat memiliki fungsionalitas yang berbeda, mengurangi kompleksitas komputasi dan meningkatkan kemampuan debug sistem.

Komunitas ROS berkembang pesat, dengan banyak pengguna dan pengembang di seluruh dunia. Banyak perusahaan robotika terkemuka kini memporting perangkat lunak mereka ke ROS, dan tren ini juga terlihat dalam robotika industri, di mana perusahaan beralih dari aplikasi robotika propietari ke ROS. Dengan pemahaman mengapa ROS menjadi pilihan yang nyaman untuk dipelajari, kita dapat memulai dengan memperkenalkan konsep inti ROS, yang terbagi menjadi tiga tingkat: tingkat sistem file, tingkat grafik komputasi, dan tingkat komunitas.

Understanding the ROS filesystem level

ROS bukan hanya kerangka pengembangan. ROS dapat dianggap sebagai meta-OS karena tidak hanya menyediakan alat dan pustaka, tetapi juga fungsi mirip OS, seperti abstraksi perangkat keras, manajemen paket, dan rangkaian alat pengembang. Seperti sistem operasi sungguhan, file ROS diorganisir di hard disk dengan cara tertentu, seperti yang ditunjukkan dalam diagram tingkat sistem file ROS.

1. **Packages (Paket):** Paket ROS adalah elemen sentral perangkat lunak ROS. Paket berisi satu atau lebih program ROS (node), pustaka, file konfigurasi, dan sebagainya, yang diorganisir bersama sebagai satu unit. Paket adalah item build and release atom dalam perangkat lunak ROS.
2. **Package Manifest (Manifest Paket):** File manifest paket berisi informasi tentang paket, penulis, lisensi, dependensi, dan sebagainya. File package.xml di dalam paket ROS adalah file manifest dari paket tersebut.
3. **Metapackages (Metapaket):** Metapaket merujuk pada satu atau lebih paket terkait yang dapat dikelompokkan secara longgar. Metapaket pada dasarnya adalah paket virtual yang tidak berisi kode sumber atau file biasa yang biasanya ditemukan dalam paket.
4. **Messages (.msg):** Pesan kustom dapat didefinisikan di dalam folder msg dalam paket. File pesan memiliki ekstensi .msg.
5. **Services (.srv):** Tipe data permintaan dan respons dapat didefinisikan di dalam folder srv dalam paket. File layanan memiliki ekstensi .srv.
6. **Repositories (Repository):** Sebagian besar paket ROS dikelola menggunakan Sistem Kontrol Versi (VCS) seperti Git, Subversion (SVN), atau Mercurial (hg). Sebuah set file di VCS mewakili repositori.

ROS packages

Paket ROS memiliki struktur khas, termasuk folder seperti config, include, script, src, launch, msg, srv, action, package.xml, dan CMakeLists.txt. Perintah-perintah seperti catkin_create_pkg, rospack, catkin_make, dan rosdep digunakan untuk bekerja dengan paket ROS.

ROS metapackages

Metapaket adalah paket khusus yang hanya membutuhkan satu file, yaitu file package.xml. Metapaket mengelompokkan sejumlah paket menjadi satu paket logis tunggal. Pada file package.xml, metapaket mengandung tag eksport yang menunjukkan bahwa itu adalah metapaket.

ROS messages

ROS menggunakan bahasa deskripsi pesan sederhana untuk mendefinisikan tipe data yang dapat dibaca atau ditulis oleh node ROS. File pesan dapat terdiri dari dua jenis: bidang dan konstan. Bidang memiliki tipe dan nama, dan ada beberapa tipe data bawaan yang dapat digunakan.

The ROS services

ROS Services adalah bentuk komunikasi permintaan/responden antara node ROS. Layanan didefinisikan dalam file .srv dan memuat jenis pesan permintaan dan respons.

Understanding the ROS computation graph level

- Komputasi dalam ROS dilakukan menggunakan jaringan dari node-node ROS, dikenal sebagai grafik komputasi.
- Konsep utama dalam grafik komputasi melibatkan node-node ROS, master, parameter server, pesan, topik, layanan, dan bag.
- Paket komunikasi ROS, termasuk pustaka klien inti seperti roscpp dan rospython, termasuk dalam tumpukan bernama ros_comm.
- Stack ini juga mencakup alat-alat seperti rostopic, rosparam, rosservice, dan rosnode untuk introspeksi konsep-konsep sebelumnya.
- Tumpukan ros_comm berisi paket middleware komunikasi ROS, yang secara kolektif disebut sebagai lapisan grafik ROS.

Elemen-elemen Baru Grafik ROS:

1. **Nodes (Node):** Proses yang melakukan komputasi dalam ROS. Setiap node ditulis menggunakan pustaka klien ROS.

2. **Master (Master):** Menyediakan proses pendaftaran dan pencarian nama untuk node-node lainnya. Nodes tidak dapat saling menemukan, bertukar pesan, atau memanggil layanan tanpa master ROS.
3. **Parameter Server (Server Parameter):** Memungkinkan penyimpanan data secara terpusat yang dapat diakses dan dimodifikasi oleh semua node. Bagian dari ROS master.
4. **Topics (Topik):** Setiap pesan di ROS diangkut menggunakan jalur bernama topik. Nodes dapat mempublikasikan atau berlangganan ke topik.
5. **Logging:** Sistem logging ROS untuk menyimpan data seperti data sensor. Digunakan untuk pengembangan dan pengujian algoritma robot.

ROS nodes

- Nodes melakukan komputasi menggunakan pustaka klien ROS seperti roscpp dan rospy.
- Menjalankan banyak node membuat sistem tahan kesalahan dan memudahkan debug karena setiap node menangani fungsi tunggal.
- ROS menyediakan alat introspeksi untuk nodes, termasuk perintah rosnode untuk mendapatkan informasi tentang dan mengelola nodes.

ROS messages

- Pesan dalam ROS adalah struktur data sederhana yang mengandung tipe bidang.
- ROS messages mendukung tipe data primitif standar dan array dari tipe primitif.
- Informasi tentang messages dapat diperoleh menggunakan perintah rosmsg.

ROS topics

- Komunikasi ROS melalui topik bersifat unidireksional.
- Penggunaan topik memungkinkan pertukaran data antar nodes. Dapat diakses menggunakan perintah rostopic.

ROS services

- Layanan ROS melibatkan komunikasi permintaan/respon antara node-node ROS.
- Informasi tentang services dapat diperoleh menggunakan perintah rossrv dan rosservice.

ROS bagfiles

- Bagfiles digunakan untuk menyimpan data pesan ROS yang dikirim melalui topik.
- Rekam dan mainkan bagfiles menggunakan perintah rosbag.

The ROS master

- ROS master adalah server yang mengasosiasikan nama dan ID unik untuk elemen-elemen ROS aktif.
- Nodes berkomunikasi dengan master untuk pertukaran informasi.

Using the ROS parameter

- Parameter server memungkinkan penyimpanan data terpusat dan dapat diakses oleh semua nodes.
- Menggunakan perintah rosparam untuk bekerja dengan parameter dari baris perintah.

ROS community level

1. **Distribusi (Distributions):** Mirip dengan distribusi Linux, distribusi ROS adalah kumpulan metapaket yang terverifikasi yang dapat diinstal. Memungkinkan instalasi dan pengumpulan perangkat lunak ROS dengan mudah dan mempertahankan versi yang konsisten.
2. **Repositori (Repositories):** ROS mengandalkan jaringan repositori kode terfederasi, di mana berbagai institusi dapat mengembangkan dan merilis komponen perangkat lunak robot mereka sendiri.
3. **ROS Wiki:** Forum utama untuk mendokumentasikan informasi tentang ROS. Dapat diakses oleh siapa saja untuk berkontribusi, memberikan koreksi atau pembaruan, menulis tutorial, dan lainnya.
4. **Sistem Tiket Bug:** Digunakan untuk melaporkan bug atau menambah fitur baru pada perangkat lunak yang ada.
5. **Milis (Mailing Lists):** Milis ROS-users digunakan untuk bertanya tentang perangkat lunak ROS dan berbagi masalah program dengan komunitas.
6. **ROS Answers:** Sumber daya situs web untuk bertanya tentang permasalahan terkait ROS dan menerima solusi dari pengguna ROS lainnya.
7. **Blog ROS:** Blog ROS yang diperbarui dengan berita, foto, dan video terkait komunitas ROS.

Prerequisites for starting with ROS

1. **Ubuntu 20.04 LTS/Debian 10:** ROS didukung secara resmi oleh sistem operasi Ubuntu dan Debian. Disarankan menggunakan versi LTS Ubuntu 20.04.
2. **ROS Noetic Desktop Full Installation:** Instalasi lengkap desktop ROS. Versi yang disarankan adalah ROS Noetic, yang merupakan versi stabil terbaru.

Distribusi ROS:

- ROS mengikuti siklus rilis yang sama dengan distribusi Linux Ubuntu, dengan versi baru dirilis setiap 6 bulan.
- Versi LTS dari ROS dirilis untuk setiap versi Ubuntu LTS dan disertai dukungan jangka panjang (5 tahun).

Running the ROS master and the ROS parameter server

- Sebelum menjalankan node ROS, ROS master dan parameter server harus dijalankan menggunakan perintah **roscore**.
- **roscore** memulai ROS master, parameter server, dan node logging (**rosout**).

Periksa Keluaran Perintah roscore:

- Menjelaskan setiap bagian dari keluaran perintah **roscore** termasuk pembuatan file log, peluncuran file roscore.xml, parameter ROS, peluncuran rosmaster, dan node rosout.

Pemeriksaan ROS Topics, Parameters, dan Services:

- **rostopic list** menampilkan topik-topik ROS yang aktif, termasuk /rosout dan /rosout_agg.
- **rosparam list** menampilkan parameter ROS yang aktif, seperti /rosdistro, /roslaunch/uris/host_robot_virtualbox_51189, /rosversion, dan /run_id.
- **rosservice list** menampilkan layanan ROS yang aktif, seperti /rosout/get_loggers dan /rosout/set_logger_level.

Chapter 2 Getting Started with ROS Programming

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Chapter 2 dari buku "Mastering ROS for Robotics Programming" membahas tentang mulai pemrograman ROS. Setelah membahas dasar-dasar ROS master, parameter server, dan roscore, kita dapat mulai membuat dan membangun ros packages. Pada bab ini, kita akan membuat berbagai node ROS dengan mengimplementasikan sistem komunikasi ROS. Selama bekerja dengan ros packages, kita juga akan memperbarui pengetahuan dasar tentang node ROS, topik, pesan, layanan, dan actionlib.

Creating ROS Packages

Ros packages merupakan unit dasar dari program ROS. Paket ini dapat dibuat, dikompilasi, dan dirilis ke publik. ROS saat ini yang digunakan adalah Noetic Ninjemys dengan sistem pembangunan catkin. Sistem ini bertanggung jawab untuk menghasilkan target (eksekutable/libraries) dari kode sumber teks yang dapat digunakan oleh pengguna akhir. Dalam distribusi ROS sebelumnya seperti Electric dan Fuerte, menggunakan sistem pembangunan rosbld. Namun, karena kelemahan rosbld, catkin muncul dan memungkinkan sistem kompilasi ROS mendekati Cross Platform Make (CMake). Langkah-langkah mencakup pembuatan workspace catkin, inisialisasi workspace, dan kompilasi workspace menggunakan perintah catkin_make.

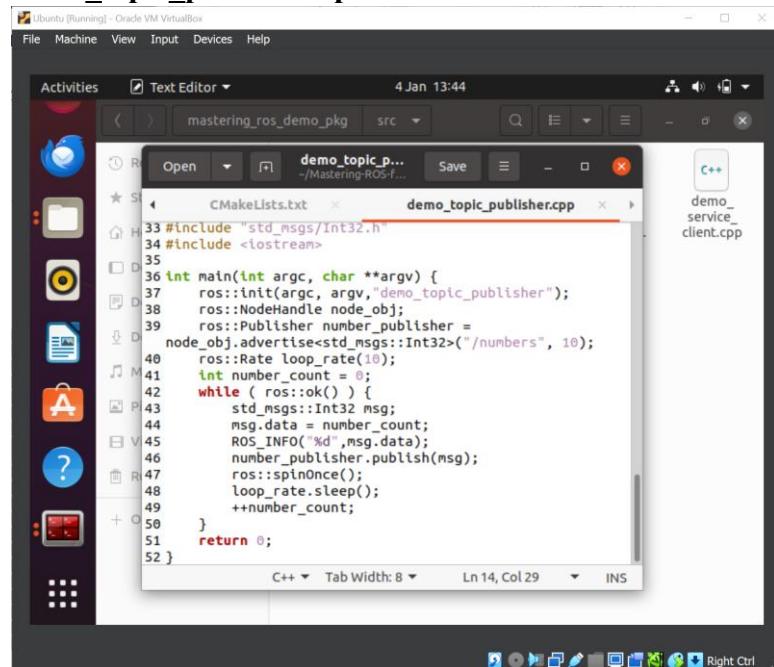
Working with ROS Topics

Topik ROS digunakan sebagai metode komunikasi antara node ROS, memungkinkan mereka berbagi aliran informasi yang dapat diterima oleh node lain. Pembahasan di bagian ini fokus pada pembuatan dua node ROS untuk mempublikasikan dan berlangganan topik. Setelah membuat workspace catkin, kita dapat membuat ros packages menggunakan perintah catkin_create_pkg. Dalam contoh ini, paket mastering_ros_demo_pkg dibuat dengan dependensi roscpp, std_msgs, actionlib, dan actionlib_msgs. Setelah pembuatan paket, dependencies tambahan dapat ditambahkan manual melalui pengeditan file CMakeLists.txt dan package.xml. Dijelaskan pula cara membangun paket dan mulai menambahkan node ke folder src dalam paket tersebut. Setelah berhasil dikompilasi, pembahasan beralih ke cara bekerja dengan ROS topics menggunakan dua file kode sumber demo_topic_publisher.cpp dan demo_topic_subscriber.cpp.

Creating ROS nodes

Node pertama yang dibahas adalah demo_topic_publisher.cpp. Node ini akan mempublikasikan nilai integer pada topik bernama /numbers. Kode ini menggambarkan penggunaan header files, inisialisasi ROS node, pembuatan node handle, pembuatan publisher untuk topik /numbers, dan perulangan tak terbatas yang mengirimkan pesan integer ke topik tersebut dengan kecepatan 10 Hz. Setelah itu, dibahas juga node subscriber demo_topic_subscriber.cpp yang berlangganan topik /numbers dan memiliki callback function untuk menangani pesan yang diterima.

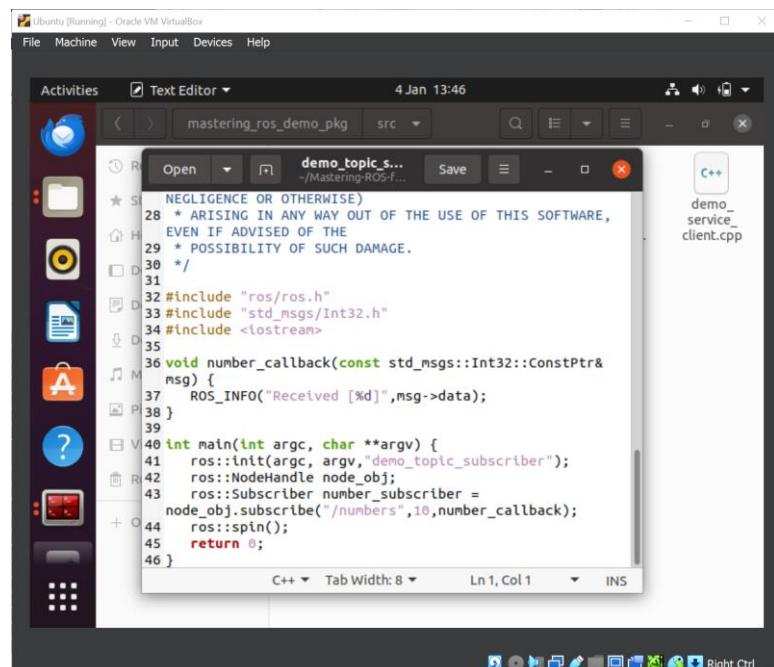
demo_topic_publisher.cpp



```
#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);
        ros::spinOnce();
        loop_rate.sleep();
        ++number_count;
    }
    return 0;
}
```

demo_topic_subscriber.cpp

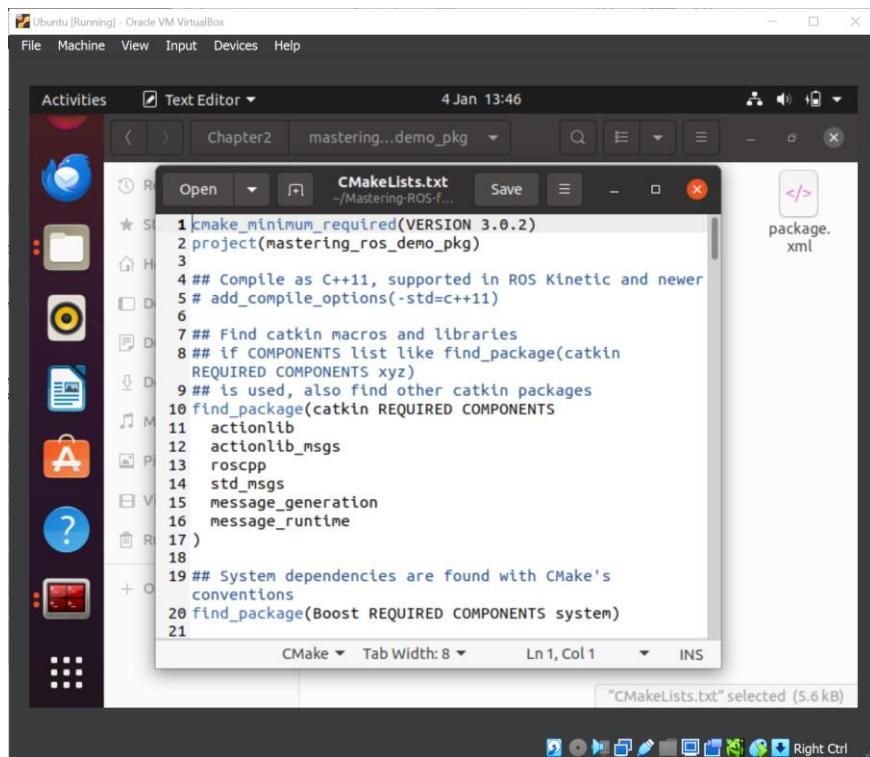


```
NEGLIGENCE OR OTHERWISE)
28 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
29 * EVEN IF ADVISED OF THE
29 * POSSIBILITY OF SUCH DAMAGE.
30 */
31
32 #include "ros/ros.h"
33 #include "std_msgs/Int32.h"
34 #include <iostream>
35
36 void number_callback(const std_msgs::Int32::ConstPtr&
37     msg) {
38     ROS_INFO("Received [%d]", msg->data);
39 }
40
41 int main(int argc, char **argv) {
42     ros::init(argc, argv, "demo_topic_subscriber");
43     ros::NodeHandle node_obj;
44     ros::Subscriber number_subscriber =
45         node_obj.subscribe("/numbers", 10, number_callback);
46     ros::spin();
47     return 0;
48 }
```

Building the nodes

Proses ini melibatkan penyuntingan file CMakeLists.txt dalam paket untuk mengkompilasi dan membangun kode sumber. Kode CMakeLists.txt mengarahkan pada file-file yang perlu dikompilasi dan dihubungkan. Setelah penyuntingan, perintah catkin_make digunakan untuk membangun workspace ROS dan menjalankan kedua node tersebut. Diagram komunikasi antara node penerbit dan pelanggan ditunjukkan, dan alat ROS seperti rosnode dan rostopic digunakan untuk debug dan pemahaman kerja kedua node.

CMakeLists.txt



The screenshot shows a Linux desktop environment with a text editor window open. The window title is "CMakeLists.txt". The code in the editor is as follows:

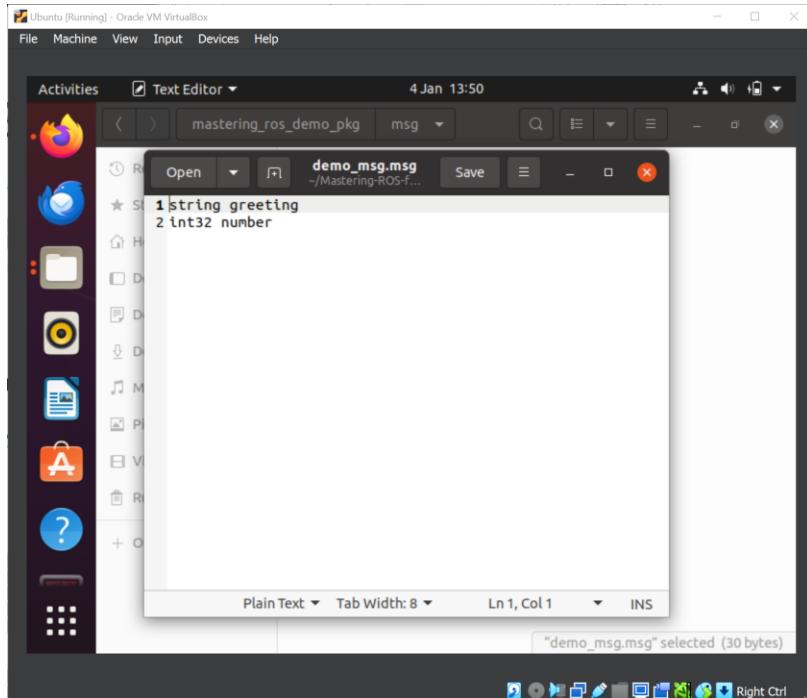
```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(mastering_ros_demo_pkg)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, 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 )
18
19 ## System dependencies are found with CMake's
20 ## conventions
21 find_package(Boost REQUIRED COMPONENTS system)
```

The status bar at the bottom of the editor window indicates "CMakeLists.txt selected (5.6 kB)".

Adding custom .msg and .srv files

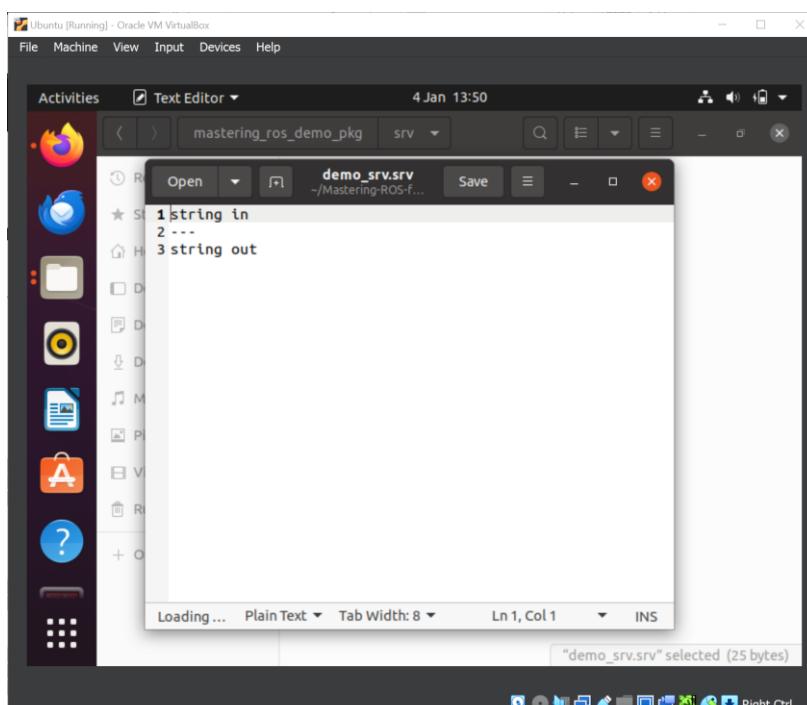
Bagian ini membahas pembuatan pesan (message) dan definisi layanan (service) kustom dalam ros packages. Pesan disimpan dalam berkas .msg, sementara definisi layanan disimpan dalam berkas .srv. Kedua definisi ini memberitahu ROS tentang jenis data dan nama data yang akan dikirim dari sebuah node ROS. Saat pesan kustom ditambahkan, ROS akan mengonversi definisi tersebut menjadi kode C++ yang dapat diikutsertakan dalam node-node kita.

demo_msg.msg



```
1 string greeting
2 int32 number
```

demo_srv.srv

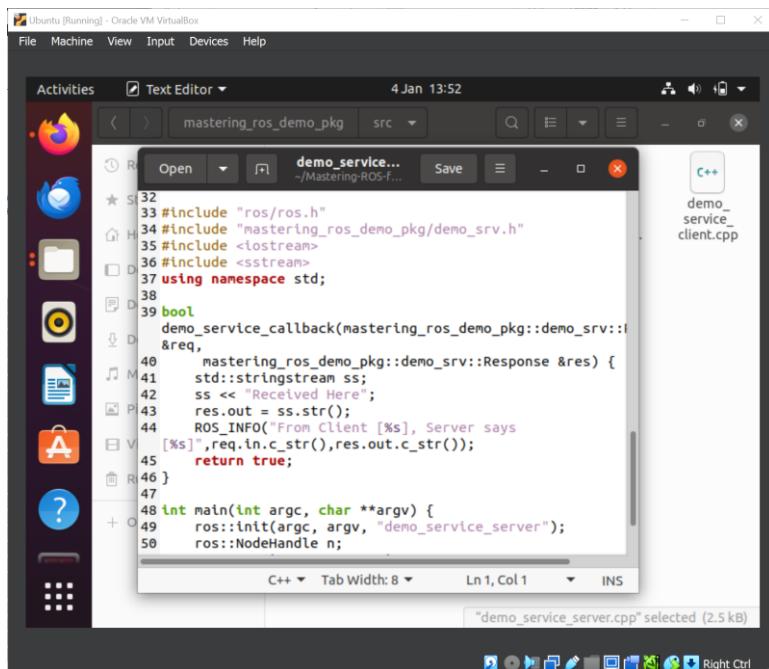


```
1 string in
2 ---
3 string out
```

Working with ROS services

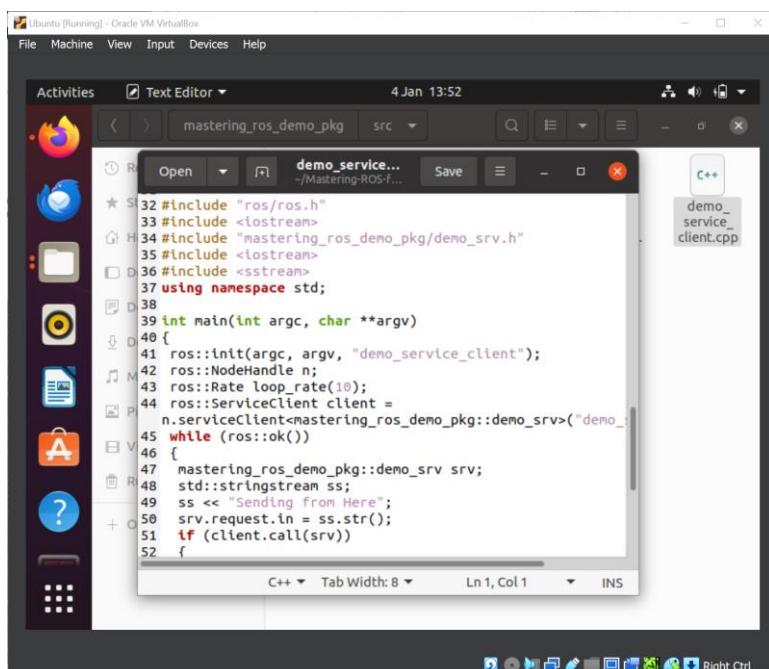
- Membuat node ROS yang menggunakan definisi layanan yang sudah ditentukan sebelumnya.
- Node layanan dapat mengirimkan pesan string sebagai permintaan ke server, dan server akan mengirimkan pesan lain sebagai respons.
- Dua node yang dibuat adalah demo_service_server.cpp (server) dan demo_service_client.cpp (klien).

demo_service_server.cpp



```
32 #include "ros/ros.h"
33 #include "mastering_ros_demo_pkg/demo_srv.h"
34 #include <iostream>
35 #include <sstream>
36 using namespace std;
37
38 bool
39 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;
```

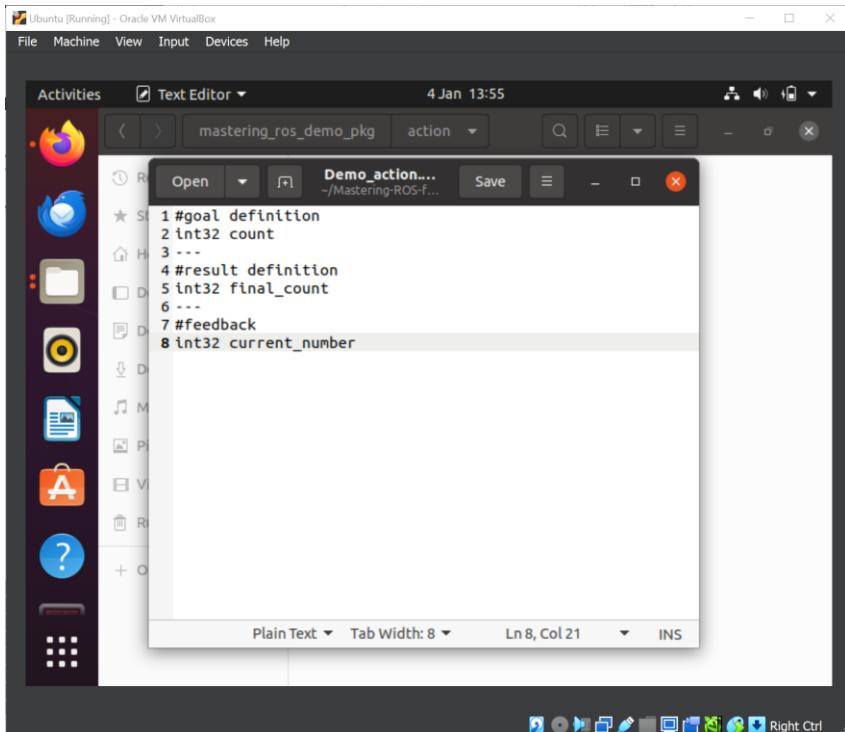
demo_service_client.cpp



```
32 #include "ros/ros.h"
33 #include <iostream>
34 #include "mastering_ros_demo_pkg/demo_srv.h"
35 #include <sstream>
36 using namespace std;
37
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 =
45         n.serviceClient<mastering_ros_demo_pkg::demo_srv>("demo_");
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     {
```

Working with ROS actionlib

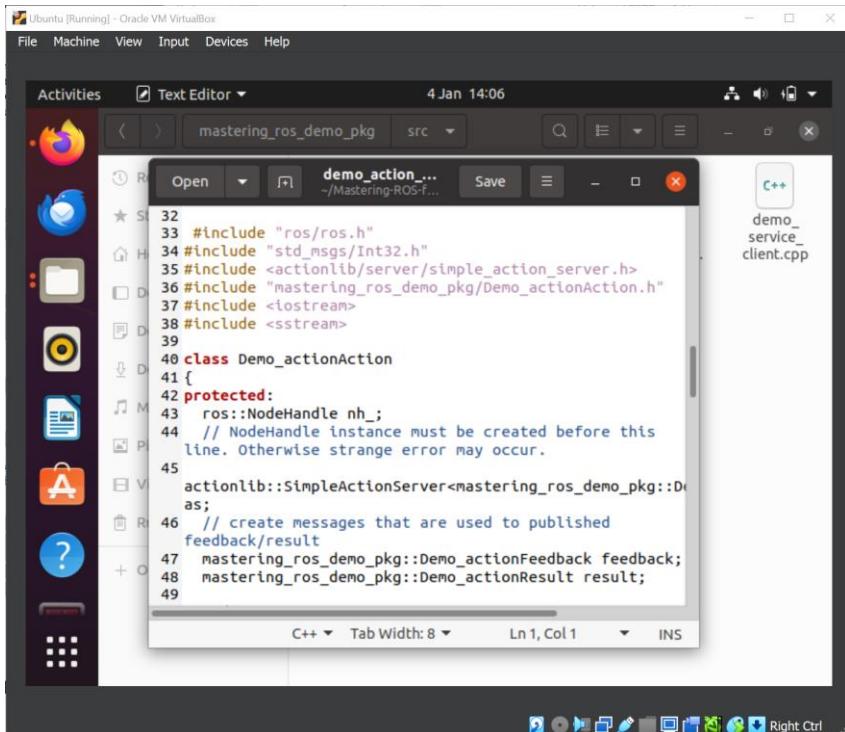
Demo_action.action



```
1 #goal definition
2 int32 count
3 ---
4 #result definition
5 int32 final_count
6 ---
7 #feedback
8 int32 current_number
```

Creating the ROS action server

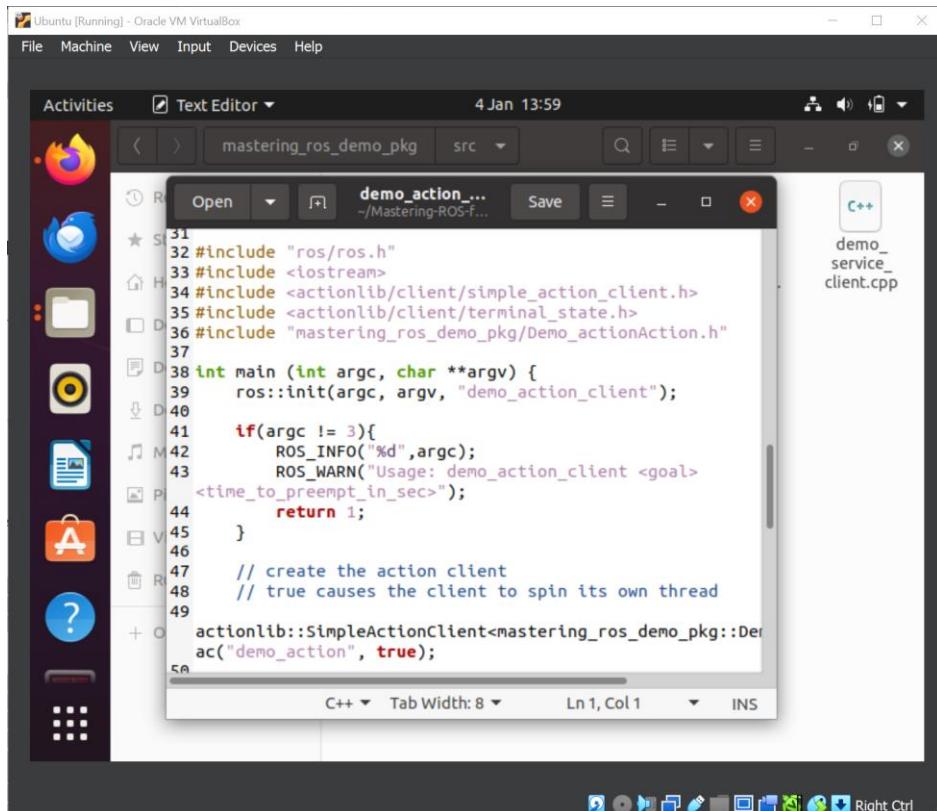
demo_action_server.cpp



```
32
33 #include "ros/ros.h"
34 #include "std_msgs/Int32.h"
35 #include <actionlib/server/simple_action_server.h>
36 #include "mastering_ros_demo_pkg/Demo_actionAction.h"
37 #include <iostream>
38 #include <sstream>
39
40 class Demo_actionAction
41 {
42 protected:
43   ros::NodeHandle nh_;
44   // NodeHandle instance must be created before this
45   // line. Otherwise strange error may occur.
46   actionlib::SimpleActionServer<mastering_ros_demo_pkg::Demo_
47   as;
48   // create messages that are used to published
49   // feedback/result
50   mastering_ros_demo_pkg::Demo_actionFeedback feedback;
51   mastering_ros_demo_pkg::DemoActionResult result;
52 }
```

- Action server menerima nilai tujuan dalam bentuk angka dan akan menghitung dari 0 hingga nilai tersebut.
- Jika perhitungan selesai, server mengirimkan hasilnya; jika tidak, tugas dapat dicabut oleh klien.
- Implementasi menggunakan actionlib untuk dapat membatalkan tugas yang berjalan dan memulai tugas baru jika diperlukan.
- Contoh tugas action server adalah `demo_action_server.cpp`.

demo_action_client.Cpp



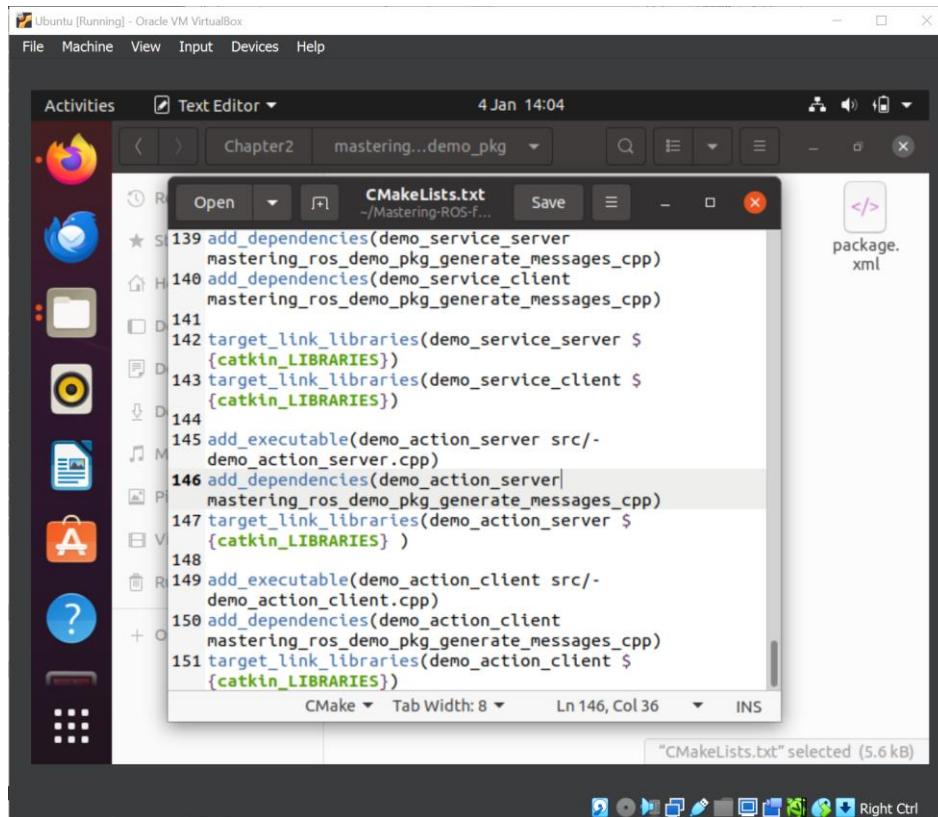
```

31 #include "ros/ros.h"
32 #include <iostream>
33 #include <actionlib/client/simple_action_client.h>
34 #include <actionlib/client/terminal_state.h>
35 #include "mastering_ros_demo_pkg/DemoAction.h"
36
37 int main (int argc, char **argv) {
38   ros::init(argc, argv, "demo_action_client");
39
40   if(argc != 3){
41     ROS_INFO("%d",argc);
42     ROS_WARN("Usage: demo_action_client <goal>
<time_to_preempt_in_sec>");
43     return 1;
44   }
45
46   // create the action client
47   // true causes the client to spin its own thread
48   actionlib::SimpleActionClient<mastering_ros_demo_pkg::DemoAction> ac("demo_action", true);
49
50

```

- Action client mengirim nilai tujuan ke server dan menunggu hingga batas waktu yang ditentukan.
- Jika tugas selesai, client menerima hasilnya; jika tidak, client dapat membatalkan tugas.
- Contoh implementasi adalah `demo_action_client.cpp`.

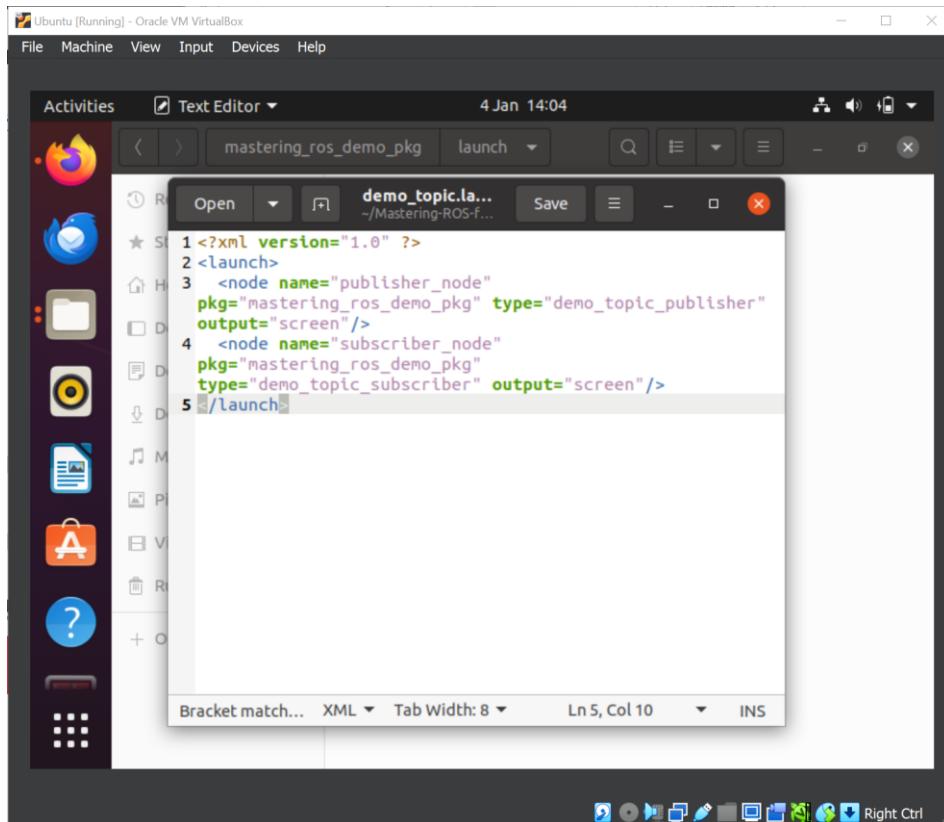
Building the ROS action server and client



```
139 add_dependencies(demo_service_server
mastering_ros_demo_pkg_generate_messages_cpp)
140 add_dependencies(demo_service_client
mastering_ros_demo_pkg_generate_messages_cpp)
141
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})
```

- Package.xml dan CMakeLists.txt diubah untuk memasukkan dependensi dan mengkonfigurasi pembangunan tiga paket actionlib, actionlib_msgs, dan message_generation.
- Penggunaan Boost sebagai dependensi sistem.
- Dua file action ditambahkan ke folder action untuk mendefinisikan spesifikasi aksi.
- Dua executable (action server dan action client) ditambahkan ke CMakeLists.txt dengan dependensi yang sesuai.
- Setelah catkin_make, nodes dapat dijalankan menggunakan perintah rosrun dan diuji dengan meroservice dan rostopic.

Creating launch files



- Launch files (launch files) di ROS sangat berguna untuk menjalankan lebih dari satu node sekaligus.
- Dengan launch files, kita dapat menjalankan beberapa node ROS secara bersamaan tanpa harus membuka terminal untuk setiap node.
- Launch files adalah berkas berbasis XML yang berisi definisi node-node yang akan dijalankan.
- Perintah roslaunch secara otomatis memulai ROS master dan parameter server, menghilangkan kebutuhan untuk menjalankan roscore dan node-node secara terpisah.
- Launch files dapat memudahkan pengelolaan node-node yang kompleks, seperti pada kasus robot dengan puluhan node.
- Launch files ditempatkan dalam folder "launch" di dalam ros packages.
- Isi launch files (contoh: demo_topic.launch) terdiri dari elemen-elemen seperti <node> yang mendefinisikan node-node yang akan dijalankan.
- Setelah membuat launch files, dapat diluncurkan dengan perintah roslaunch dan memuat semua node yang dijelaskan di dalamnya.
- Informasi debug dan log dapat dilihat menggunakan perintah rosnode list dan rqt_console.
- Dengan menggunakan launch files, proses menjalankan dan mengelola node-node ROS menjadi lebih efisien dan terorganisir.

Chapter 3 Working with ROS for 3D Modeling

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Pada tahap awal pembuatan robot, terlibat dalam proses desain dan pemodelan. Desain dan pemodelan robot dapat dilakukan menggunakan alat CAD seperti Autodesk Fusion 360, SolidWorks, Blender, dan lainnya. Salah satu tujuan utama dari pemodelan robot adalah simulasi.

Alat simulasi robot dapat memeriksa cacat kritis dalam desain robot dan memastikan bahwa robot akan berfungsi sebelum memasuki fase manufaktur. Dalam bab ini, kita akan membahas proses desain dua robot, yaitu manipulator dengan tujuh Derajat Kebebasan (DOF) dan robot penggerak roda differential.

Di bab-bab berikutnya, kita akan melihat simulasi, mempelajari cara membangun perangkat keras nyata, dan membahas antarmuka dengan ROS. Jika Anda berencana membuat model 3D robot dan mensimulasikannya menggunakan ROS, Anda perlu mempelajari beberapa package ROS yang dapat membantu dalam desain robot.

Menciptakan model untuk robot dalam ROS penting karena berbagai alasan. Misalnya, model ini dapat digunakan untuk mensimulasikan dan mengendalikan robot, memvisualisasikannya, atau menggunakan alat ROS untuk mendapatkan informasi tentang struktur kinematika robot.

ROS menyediakan beberapa package untuk merancang dan membuat model robot, seperti urdf, kdl_parser, robot_state_publisher, dan collada_urdf. Package-package ini akan membantu kita membuat deskripsi model robot 3D dengan karakteristik yang tepat dari perangkat keras nyata.

ROS packages for robot modelling

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

- **urdf**: Package ROS paling penting untuk memodelkan robot adalah package urdf. Package ini berisi parser C++ untuk URDF, yang merupakan file XML yang mewakili model robot. Beberapa komponen berbeda membentuk urdf, seperti urdf_parser_plugin, urdfdom_headers, collada_parser, dan urdfdom.
- **joint_state_publisher**: Alat ini sangat berguna saat merancang model robot menggunakan URDF. Package 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.

- **joint_state_publisher_gui**: Alat ini mirip dengan package joint_state_publisher. Ini menawarkan fungsionalitas yang sama dan, selain itu, mengimplementasikan serangkaian slider yang dapat digunakan oleh pengguna untuk berinteraksi dengan setiap sendi robot dan memvisualisasikan output menggunakan RViz.
- **kdl_parser**: Package ini berisi alat parser untuk membangun pohon Kinematika dan Dinamika (KDL) dari model URDF robot. KDL adalah perpustakaan yang digunakan untuk menyelesaikan masalah kinematika dan dinamika.
- **robot_state_publisher**: Package ini membaca status sendi robot saat ini dan menerbitkan posisi 3D dari setiap tautan robot menggunakan pohon kinematika yang dibangun dari URDF.
- **xacro**: Xacro adalah singkatan dari XML Macros, dan kita dapat menganggap file xacro sebagai file URDF dengan beberapa tambahan. Ini berisi beberapa tambahan untuk membuat URDF lebih pendek dan lebih mudah dibaca dan dapat digunakan untuk membangun deskripsi robot yang kompleks. Kita dapat mengonversi xacro menjadi URDF kapan saja menggunakan alat ROS.

Understanding robot modeling using URDF

Dalam bagian sebelumnya, kita telah mencantumkan beberapa package penting yang menggunakan format file urdf. Pada bagian ini, kita akan melihat lebih jauh tentang tag XML URDF, yang membantu dalam memodelkan robot. Kita perlu membuat file dan menuliskan hubungan antara setiap tautan dan sendi dalam robot serta menyimpan file dengan ekstensi .urdf.

URDF dapat merepresentasikan deskripsi kinematika dan dinamika robot, representasi visual robot, dan model tumbukan robot. Tag-tag URDF yang umum digunakan untuk menyusun model robot URDF adalah:

- **link**: Tag ini merepresentasikan satu tautan robot. Dengan tag ini, kita dapat memodelkan tautan robot beserta propertinya, seperti ukuran, bentuk, dan warna; bahkan dapat mengimpor mesh 3D untuk merepresentasikan tautan robot. Tag ini juga dapat menyediakan properti dinamis tautan, seperti matriks inersia dan properti tumbukan.
- **joint**: Tag ini merepresentasikan sendi robot. Kita dapat menentukan kinematika dan dinamika sendi serta mengatur batasan pergerakan dan kecepatannya. Tag ini mendukung berbagai jenis sendi, seperti revolute, continuous, prismatic, fixed, floating, dan planar.
- **robot**: Tag ini menggabungkan seluruh model robot yang dapat direpresentasikan menggunakan URDF. Di dalam tag robot, kita dapat mendefinisikan nama robot, tautan, dan sendi robot.
- **gazebo**: Tag ini digunakan ketika kita menyertakan parameter simulasi simulator Gazebo di dalam URDF. Kita dapat menggunakan tag ini untuk menyertakan plugin gazebo, properti material gazebo, dan lainnya.

Creating the ROS package for the robot description

Sebelum membuat file URDF untuk robot, kita perlu membuat package ROS di dalam workspace Catkin. Ini dapat dilakukan dengan perintah `catkin_create_pkg` dengan dependensi pada package-package seperti `roscpp`, `tf`, `geometry_msgs`, `urdf`, `rviz`, dan `xacro`. Package ini terutama bergantung pada package `urdf` dan `xacro`, yang dapat diinstal menggunakan manajer package.

Creating our first URDF model

Setelah memahami tag-tag URDF, kita dapat mulai membuat model dasar menggunakan URDF. Robot pertama yang akan kita rancang adalah mekanisme pan-and-tilt dengan tiga tautan dan dua sendi revolute. Kita dapat melihat dan menjelaskan kode URDF untuk mekanisme ini. File URDF disimpan dalam folder **urdf**, dengan tambahan folder **meshes** dan **launch** untuk menyimpan file mesh dan file peluncuran ROS.

pan_tilt.urdf

The screenshot shows a Linux desktop environment with a dark theme. A text editor window titled "pan_tilt.urdf" is open, displaying the following XML code:

```
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"/>
```

The status bar at the bottom indicates "pan_tilt.urdf" selected (2.2 kB). The desktop background features a grid icon in the bottom right corner.

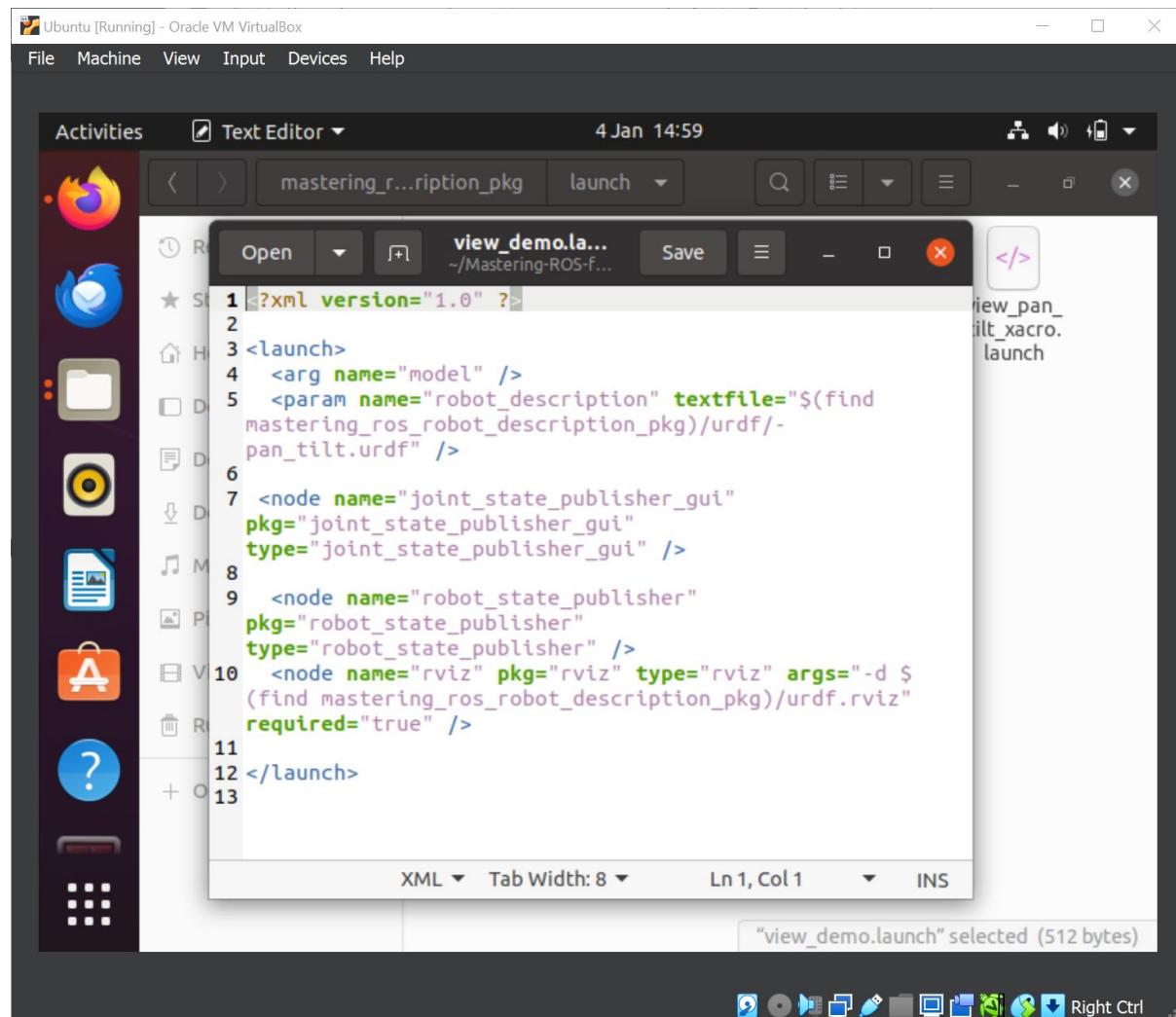
Explaining the URDF file

File URDF memiliki struktur yang terdiri dari tag-tag XML seperti `<link>`, `<joint>`, dan `<robot>`. Setiap tag mendefinisikan tautan, sendi, atau model robot secara keseluruhan. Contoh mekanisme pan-and-tilt dijelaskan dengan menggunakan tag `<link>` untuk mendefinisikan properti visual dan dinamis tautan, dan tag `<joint>` untuk mendefinisikan tautan antara dua link. Pengecekan file URDF dilakukan dengan menggunakan perintah `check_urdf`, dan visualisasi grafik menggunakan perintah `urdf_to_graphviz`. Visualisasi 3D dari model di RViz membantu pemahaman posisi dan hubungan setiap sendi robot.

Visualizing the 3D robot model in Rviz

Setelah merancang URDF, kita dapat melihatnya di RViz. Kita dapat membuat file peluncuran `view_demo.launch` dan menempatkannya di folder `launch`. File ini berisi kode XML yang menyertakan parameter deskripsi robot, node pengelolaan status sendi, dan node pengelolaan status robot. Jika semuanya berfungsi dengan benar, kita dapat meluncurkan model menggunakan perintah `rosrun` dan melihat mekanisme pan-and-tilt di RViz.

`view_demo.launch`



```
<?xml version="1.0" ?>
<launch>
  <arg name="model" />
  <param name="robot_description" textfile="$(find mastering_ros_robot_description_pkg)/urdf/pan_tilt.urdf" />
  <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find mastering_ros_robot_description_pkg)/urdf.rviz" required="true" />
</launch>
```

Interacting with pan-and-tilt joints

RViz dilengkapi dengan GUI yang memungkinkan pengguna mengontrol sendi pan dan tilt menggunakan slider. Node `joint_state_publisher_gui` digunakan untuk memasukkan GUI ini ke dalam file peluncuran. Batasan dari setiap sendi juga dapat diatur dalam tag `<limit>` di dalam tag `<joint>`.

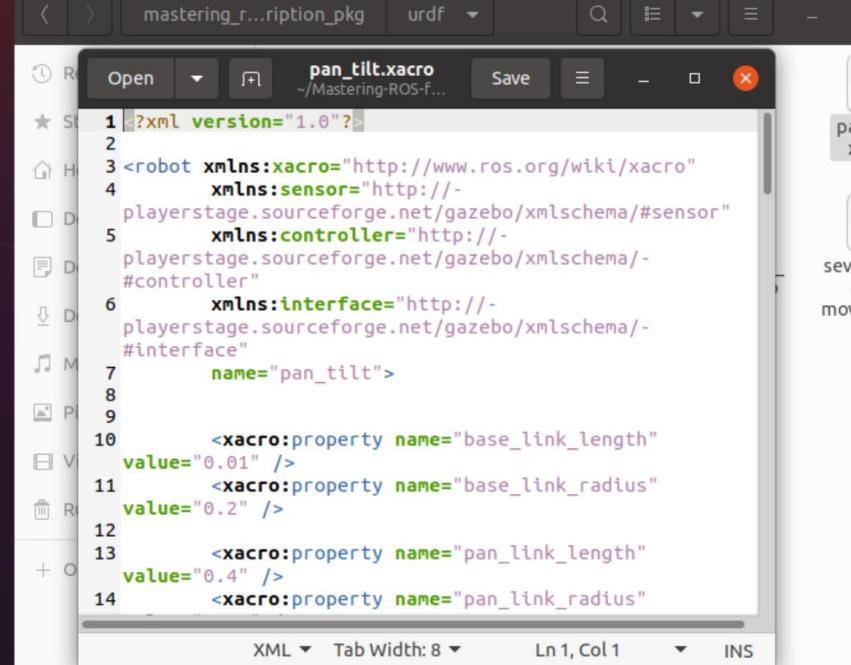
Adding physical and collision properties to a URDF model

Sebelum mensimulasikan robot di simulator seperti Gazebo atau CoppeliaSim, properti fisik dan kollision dari setiap tautan robot perlu ditentukan dalam file URDF. Informasi seperti geometri, warna, massa, dan inersia didefinisikan dalam tag `<collision>` dan `<inertial>`.

Understanding robot modeling using xacro

URDF memiliki keterbatasan dalam hal kejelasan, kegunaan kembali, modularitas, dan pemrograman. Pemodelan menggunakan xacro memenuhi kebutuhan ini dengan menyederhanakan URDF, memberikan dukungan pemrograman, mendeklarasikan properti, menggunakan ekspresi matematika, dan menggunakan makro untuk mengurangi panjang kode. File xacro dapat dikonversi menjadi URDF untuk digunakan dalam sistem ROS.

pan_tilt.xacro



The screenshot shows a Linux desktop environment with the Unity interface. A text editor window titled "pan_tilt.xacro" is open, displaying ROS XML code for a robot. The code defines a robot with sensor and controller properties, and two xacro properties for base link length and radius. The text editor has tabs for "XML" and "urdf". To the right of the editor, there are two floating windows showing snippets of XML code: one for "pan_tilt.xacro" and another for "seven_dof_arm_moveit.urdf". The desktop background features a grid icon in the bottom right corner.

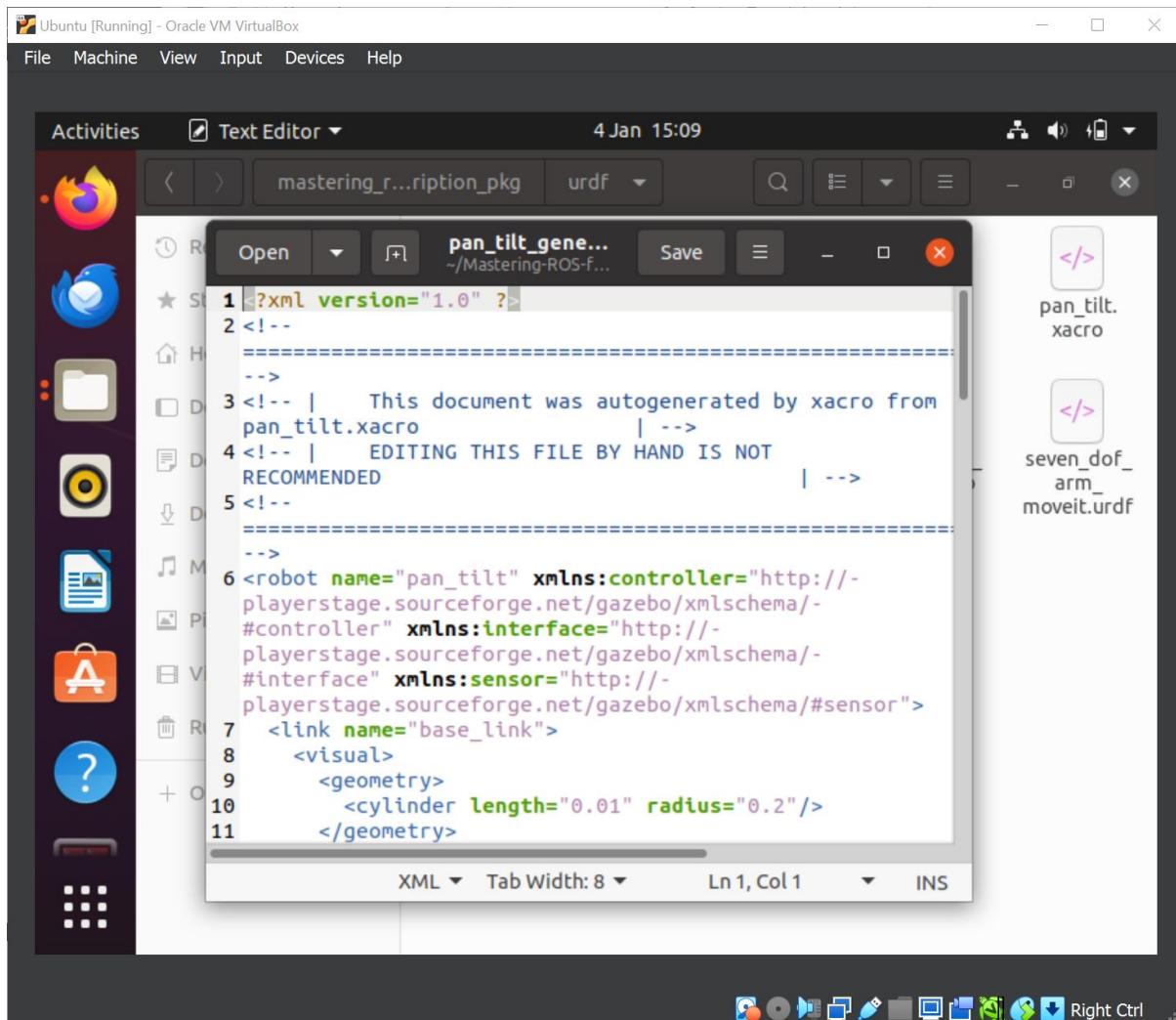
```
?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro"
       xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       name="pan_tilt">
<xacro:property name="base_link_length" value="0.01" />
<xacro:property name="base_link_radius" value="0.2" />
<xacro:property name="pan_link_length" value="0.4" />
<xacro:property name="pan_link_radius" value="0.1" />
```

Converting xacro to URDF

File xacro dapat dikonversi menjadi file URDF setiap saat. Setelah merancang file xacro, kita dapat menggunakan perintah `rosrun xacro` untuk mengonversinya menjadi file URDF. Perintah ini digunakan untuk file `pan_tilt.xacro`, menghasilkan `pan_tilt_generated.urdf`. Dalam file peluncuran ROS, kita dapat menggunakan perintah `<param>` untuk mengonversi xacro ke URDF dan menggunakan parameter `robot_description`.

Kita dapat melihat isi file xacro dari robot pan-and-tilt dengan membuat file peluncuran khusus dan menggunakan perintah `roslaunch`. Hasil visualisasi dari file xacro ini seharusnya sama dengan hasil visualisasi dari file URDF.

`pan_tilt_generated.urdf`



The screenshot shows a Linux desktop environment with a text editor window open. The window title is "pan_tilt_gene... ~/Mastering-ROS-F... urdf". The code displayed is:

```
1 <?xml version="1.0" ?>
2 <!--
=====
-->
3 <!-- | This document was autogenerated by xacro from
pan_tilt.xacro | -->
4 <!-- | EDITING THIS FILE BY HAND IS NOT
RECOMMENDED | -->
5 <!--
=====
-->
6 <robot name="pan_tilt" xmlns:controller="http://-
playerstage.sourceforge.net/gazebo/xmlschema/-#
controller" xmlns:interface="http://-
playerstage.sourceforge.net/gazebo/xmlschema/-#
interface" xmlns:sensor="http://-
playerstage.sourceforge.net/gazebo/xmlschema/#sensor">
7   <link name="base_link">
8     <visual>
9       <geometry>
10         <cylinder length="0.01" radius="0.2"/>
11       </geometry>

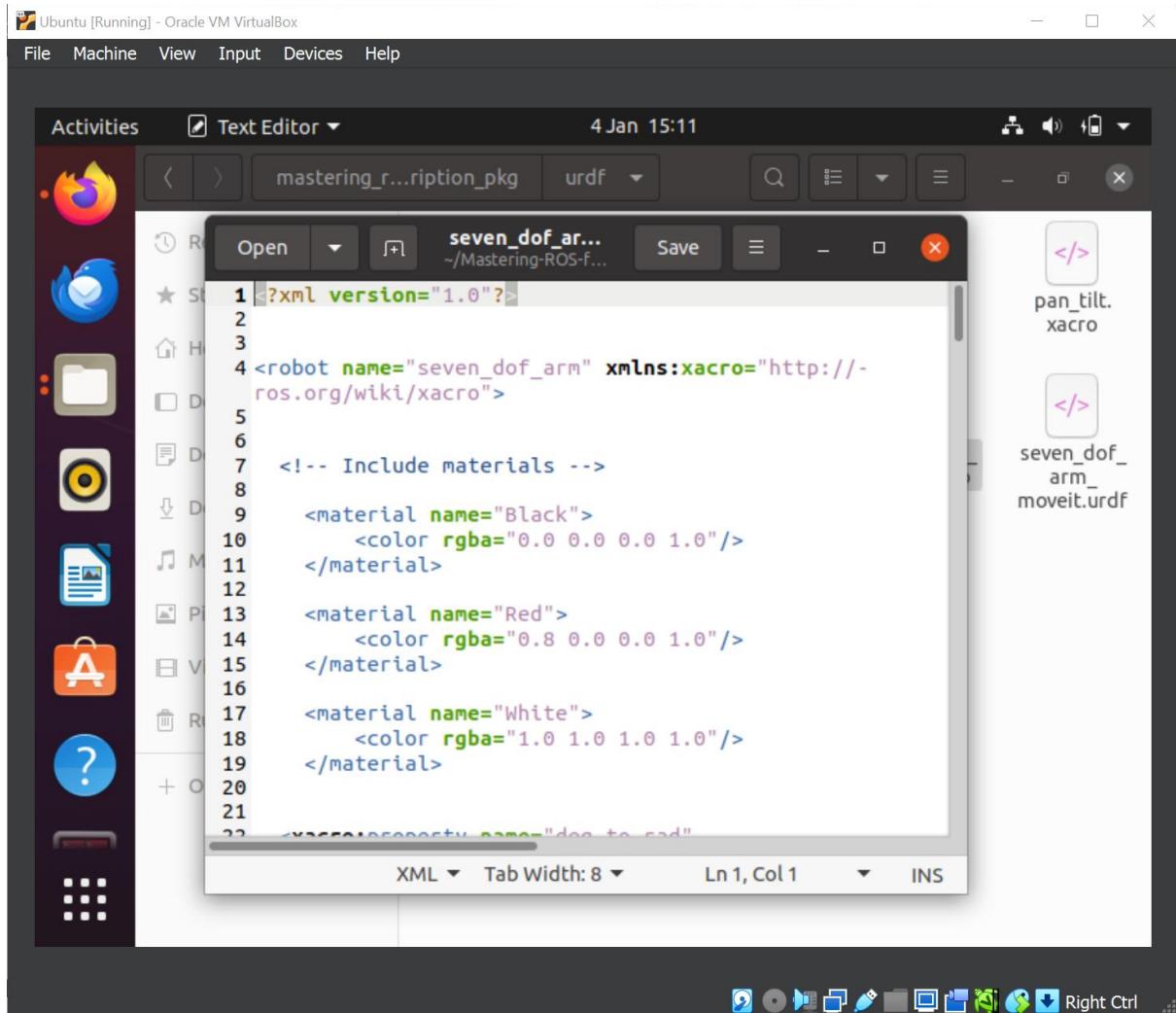
```

Creating the robot description for a seven-DOF robot manipulator

Selanjutnya, kita dapat membuat robot yang lebih kompleks menggunakan URDF dan xacro. Robot pertama yang akan dibahas adalah manipulator 7-DOF, yang memiliki tujuh sendi. Manipulator ini memiliki kelebihan sendi dibanding yang diperlukan untuk mencapai posisi

dan orientasi tujuan. Keuntungan manipulator yang redundan adalah fleksibilitas dan keberagaman konfigurasi sendi untuk mencapai posisi dan orientasi tujuan tertentu.

seven_dof_arm.xacro



```
1 <?xml version="1.0"?>
2
3
4 <robot name="seven_dof_arm" xmlns:xacro="http://
  ros.org/wiki/xacro">
5
6
7   <!-- Include materials -->
8
9   <material name="Black">
10    <color rgba="0.0 0.0 0.0 1.0"/>
11  </material>
12
13  <material name="Red">
14    <color rgba="0.8 0.0 0.0 1.0"/>
15  </material>
16
17  <material name="White">
18    <color rgba="1.0 1.0 1.0 1.0"/>
19  </material>
20
21
22 <xacro:property name="dof_to_end">
```

Arm specification

Manipulator 7-DOF memiliki spesifikasi tertentu, termasuk jumlah derajat kebebasan, panjang lengan, jangkauan, jumlah tautan, dan jumlah sendi. Terdapat berbagai jenis sendi, seperti sendi revolute dan prismatic, yang digunakan dalam deskripsi manipulator ini. Deskripsi manipulator ini ditulis dalam file xacro, yang akan dijelaskan pada bagian selanjutnya.

Explaining the xacro model of the seven-DOF arm

Setelah mendefinisikan elemen-elemen yang harus dimasukkan dalam file model robot, kita siap menyertakan 10 tautan dan 9 sendi (7 untuk lengan dan 2 untuk gripper) pada robot ini, serta 2 tautan dan 2 sendi pada gripper robot.

Using constants

Konstanta digunakan dalam file xacro untuk membuat deskripsi robot lebih pendek dan mudah dibaca. Konstanta seperti faktor konversi derajat ke radian, nilai PI, panjang, lebar, dan tinggi setiap tautan didefinisikan di sini.

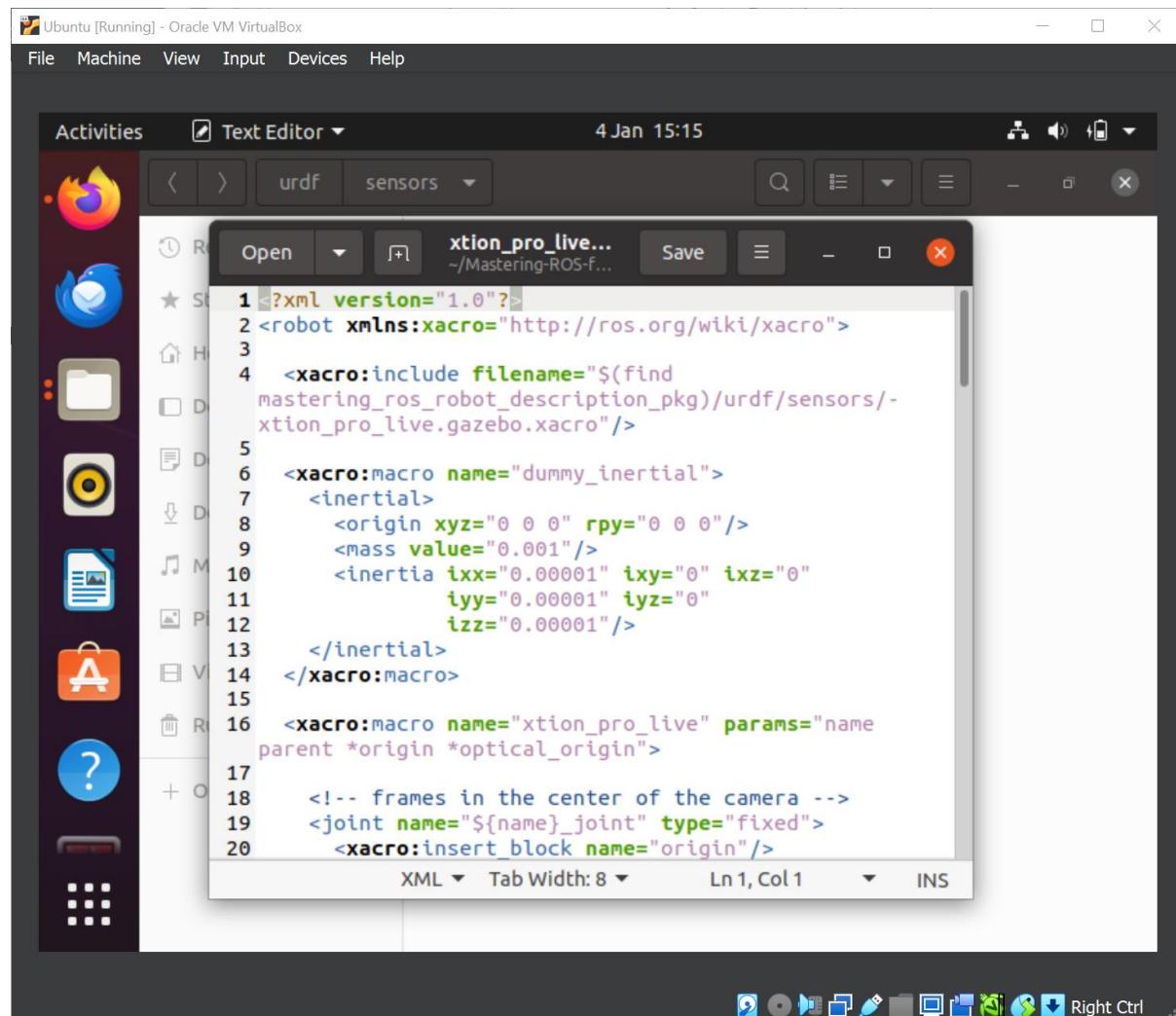
Using macro

Macro digunakan untuk menghindari repetisi dan membuat kode menjadi lebih singkat. Dua macronya adalah "inertial_matrix" untuk matriks inersia dan "transmission_block" untuk blok transmisi. Ini membantu menggambarkan elemen yang sama berkali-kali dengan lebih singkat.

Including other xacro files

Kemampuan xacro robot dapat diperluas dengan menyertakan definisi xacro sensor. Contoh penggunaan tag <xacro:include> untuk menyertakan definisi sensor visi.

xtion_pro_live.urdf.xacro



```
1 <?xml version="1.0"?>
2 <robot xmlns:xacro="http://ros.org/wiki/xacro">
3
4   <xacro:include filename="$(find
      mastering_ros_robot_description_pkg)/urdf/sensors/-_
    xtion_pro_live.gazebo.xacro"/>
5
6   <xacro:macro name="dummy_inertial">
7     <inertial>
8       <origin xyz="0 0 0" rpy="0 0 0"/>
9       <mass value="0.001"/>
10      <inertia ixx="0.00001" ixy="0" ixz="0"
11          iyy="0.00001" iyz="0"
12          izz="0.00001"/>
13    </inertial>
14  </xacro:macro>
15
16  <xacro:macro name="xtion_pro_live" params="name
      parent *origin *optical_origin">
17
18    <!-- frames in the center of the camera -->
19    <joint name="${name}_joint" type="fixed">
20      <xacro:insert_block name="origin"/>
```

Using meshes in the link

Dalam deskripsi link, kita dapat menyisipkan bentuk primitif atau file mesh menggunakan tag `<mesh>`. Dengan contoh penggunaan mesh pada sensor vision, tampilan visual robot menjadi lebih realistik dan akurat. Hal ini mempermudah simulasi robot dalam lingkungan 3D.

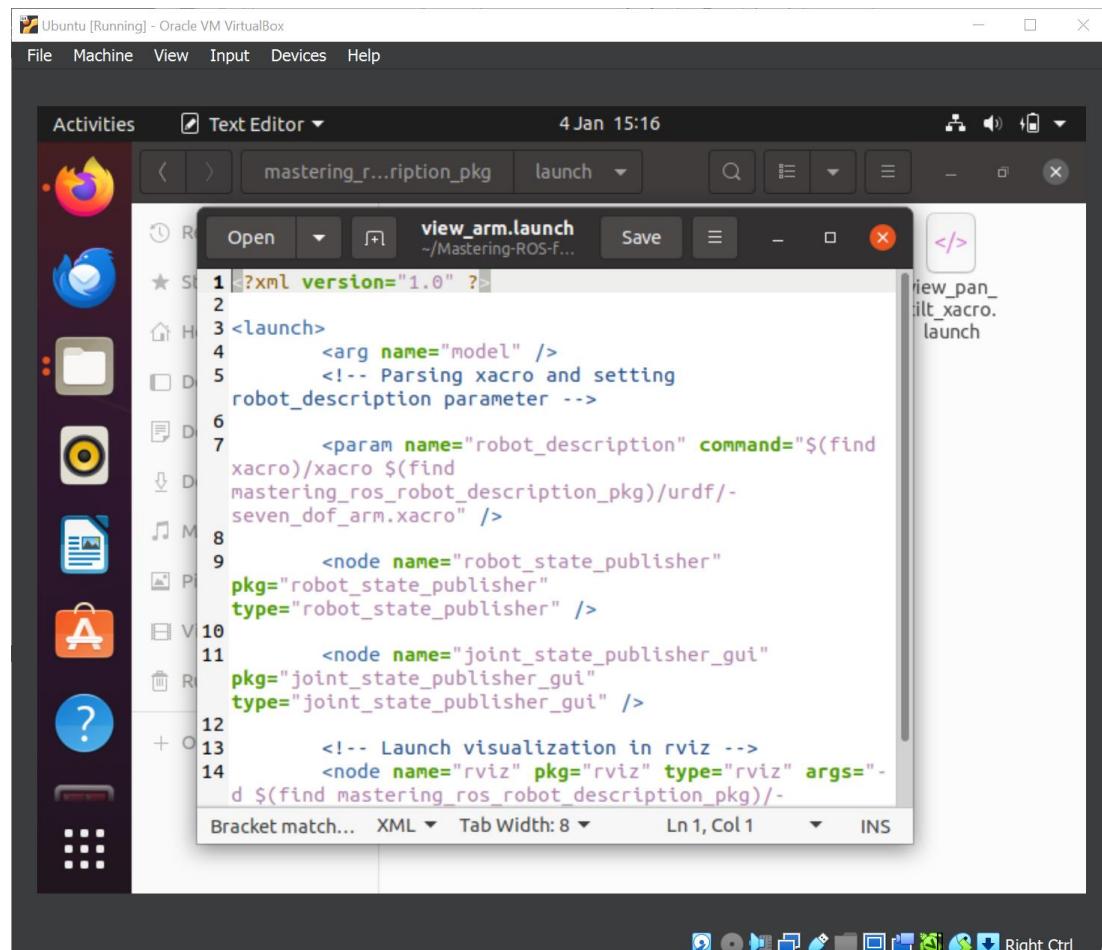
Working with the robot gripper

Gripper robot dirancang untuk mengambil dan meletakkan blok, memiliki dua sendi prismatic pada gripper. Definisi sendi mencakup parameter seperti batasan, kontrol keselamatan, dan karakteristik dinamika. Konfigurasi sendi-sendi ini memberikan fleksibilitas dalam mengendalikan gripper sesuai dengan kebutuhan aplikasi.

Viewing the seven-DOF arm in Rviz

Setelah mendefinisikan semua elemen, termasuk konstanta, makro, dan penggunaan mesh atau sensor vision, model lengan robot 7-DOF ini siap untuk divisualisasikan menggunakan RViz. RViz memungkinkan untuk memeriksa dan memvalidasi model robot secara grafis, memastikan bahwa elemen-elemen yang didefinisikan berfungsi dengan baik dalam representasi visual 3D.

`view_arm.launch`



```
?xml version="1.0" ?>
<launch>
    <arg name="model" />
    <!-- Parsing xacro and setting
robot_description parameter -->
    <param name="robot_description" command="$(find
xacro)/xacro $(find
mastering_ros_robot_description_pkg)/urdf/-
seven_dof_arm.xacro" />
    <node name="robot_state_publisher"
pkg="robot_state_publisher"
type="robot_state_publisher" />
    <node name="joint_state_publisher_gui"
pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui" />
    <!-- Launch visualization in rviz -->
    <node name="rviz" pkg="rviz" type="rviz" args="-
d $(find mastering_ros_robot_description_pkg)/-
```

Understanding joint state publisher

Package joint state publisher adalah salah satu package ROS yang umumnya digunakan untuk berinteraksi dengan setiap sendi robot. Package ini mencakup node joint_state_publisher yang mengidentifikasi sendi yang tidak tetap dari model URDF dan menerbitkan nilai keadaan sendi setiap sendi dalam format pesan sensor_msgs/JointState. Package ini dapat digunakan bersama dengan package robot_state_publisher untuk menerbitkan posisi semua sendi. Berbagai sumber dapat digunakan untuk mengatur nilai setiap sendi, termasuk penggunaan GUI slider untuk pengujian atau menggunakan topik JointState yang di-subscribe oleh node.

Understanding robot state publisher

Package robot state publisher membantu dalam menerbitkan keadaan robot ke tf (transform frame). Package ini berlangganan pada keadaan sendi robot dan menerbitkan pose 3D dari setiap tautan menggunakan representasi kinematika dari model URDF. Node robot state publisher dapat diimplementasikan dalam file peluncuran dengan menambahkan baris yang sesuai. Saat package ini dijalankan, kita dapat memvisualisasikan transformasi robot dengan memilih opsi tf di RViz. Joint state publisher dan robot state publisher umumnya sudah terpasang bersama instalasi ROS desktop.

Creating a robot model for the differential drive mobile robot

Robot roda differential drive memiliki dua roda yang terhubung ke sisi berlawanan dari sasis robot, didukung oleh satu atau dua roda caster. Roda mengendalikan kecepatan robot dengan mengatur kecepatan roda tunggal. Jika dua motor berjalan dengan kecepatan yang sama, roda akan bergerak maju atau mundur. Jika satu roda berjalan lebih lambat dari yang lain, robot akan berbelok ke sisi roda yang berkecepatan lebih rendah.

Model URDF dari robot ini hadir dalam package ROS yang telah di-clone. Robot ini memiliki lima sendi dan tautan, dengan dua sendi utama yang menghubungkan roda dengan basis robot. Tiga sendi lainnya adalah sendi tetap yang menghubungkan roda caster dan footprint basis ke badan robot.

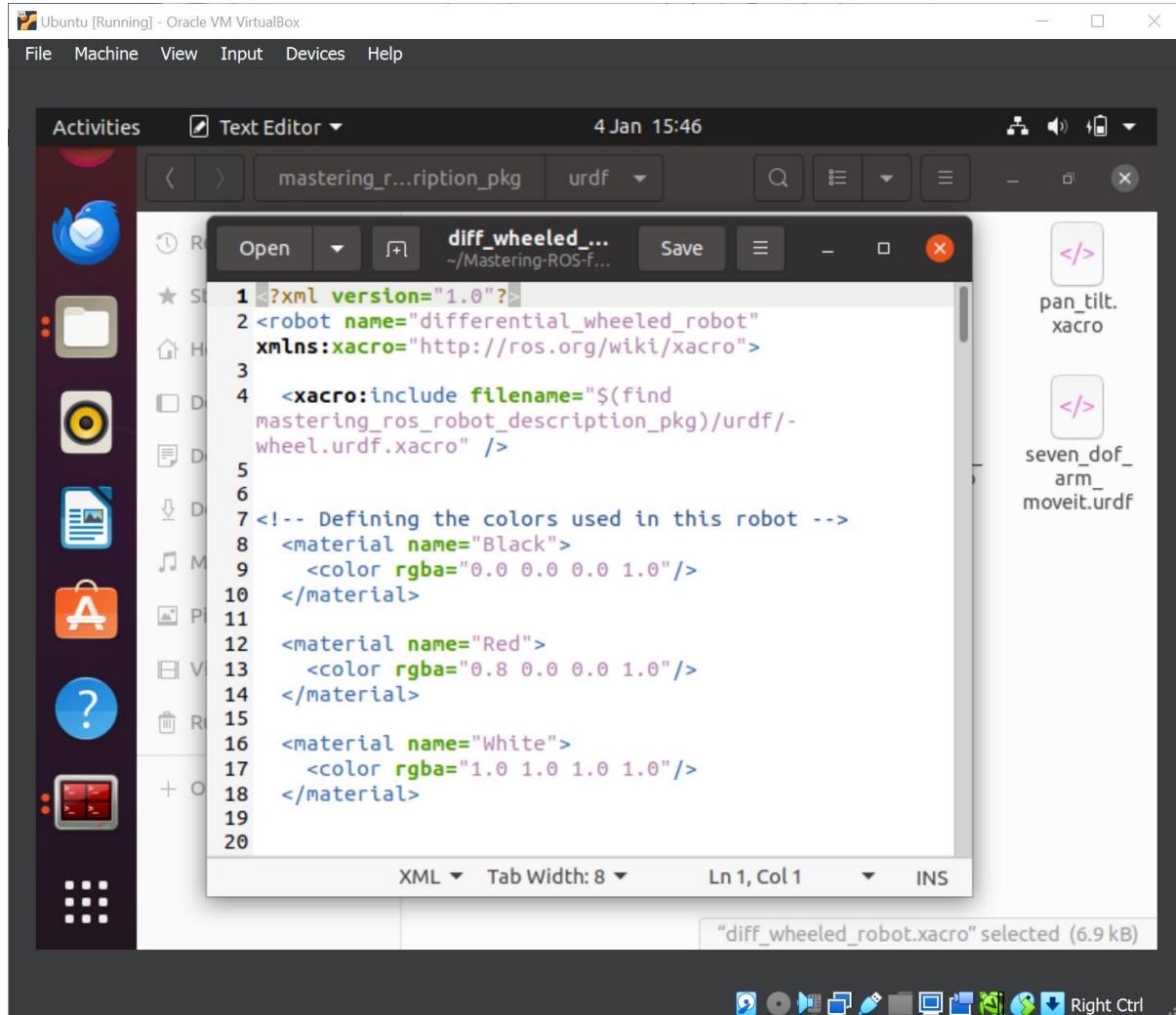
wheel.urdf.xacro

The screenshot shows a Linux desktop environment with a window titled "wheel.urdf.xacro" open in a text editor. The window displays XML code for a URDF (Universal Robot Description Format) file. The code defines a robot named "wheel" with properties for wheel radius, height, and mass, and a macro for the wheel link. The text editor interface includes tabs for "mastering_r...ription_pkg" and "urdf", and a status bar indicating the file is selected (2.5 kB).

```
<?xml version="1.0"?>
<robot name="wheel" xmlns:xacro="http://www.ros.org/~wiki/xacro">
  <!-- Wheels -->
  <xacro:property name="wheel_radius" value="0.04" />
  <xacro:property name="wheel_height" value="0.02" />
  <xacro:property name="wheel_mass" value="2.5" /> <!--
    in kg-->
  <xacro:property name="base_x_origin_to_wheel_origin"
    value="0.25" />
  <xacro:property name="base_y_origin_to_wheel_origin"
    value="0.3" />
  <xacro:property name="base_z_origin_to_wheel_origin"
    value="0.0" />
  <xacro:macro name="wheel" params="fb lr parent
    translateX translateY flipY"> <!--fb : front, back ;
    lr: left, right -->
    <link name="${fb}_${lr}_wheel">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
```

File URDF `diff_wheeled_robot.xacro` berisi definisi untuk roda dan transmisinya, menggunakan file xacro terpisah (`wheel.urdf.xacro`) untuk mencegah duplikasi definisi. Gaya Gazebo, joint continuous, dan transmission diatur untuk masing-masing roda.

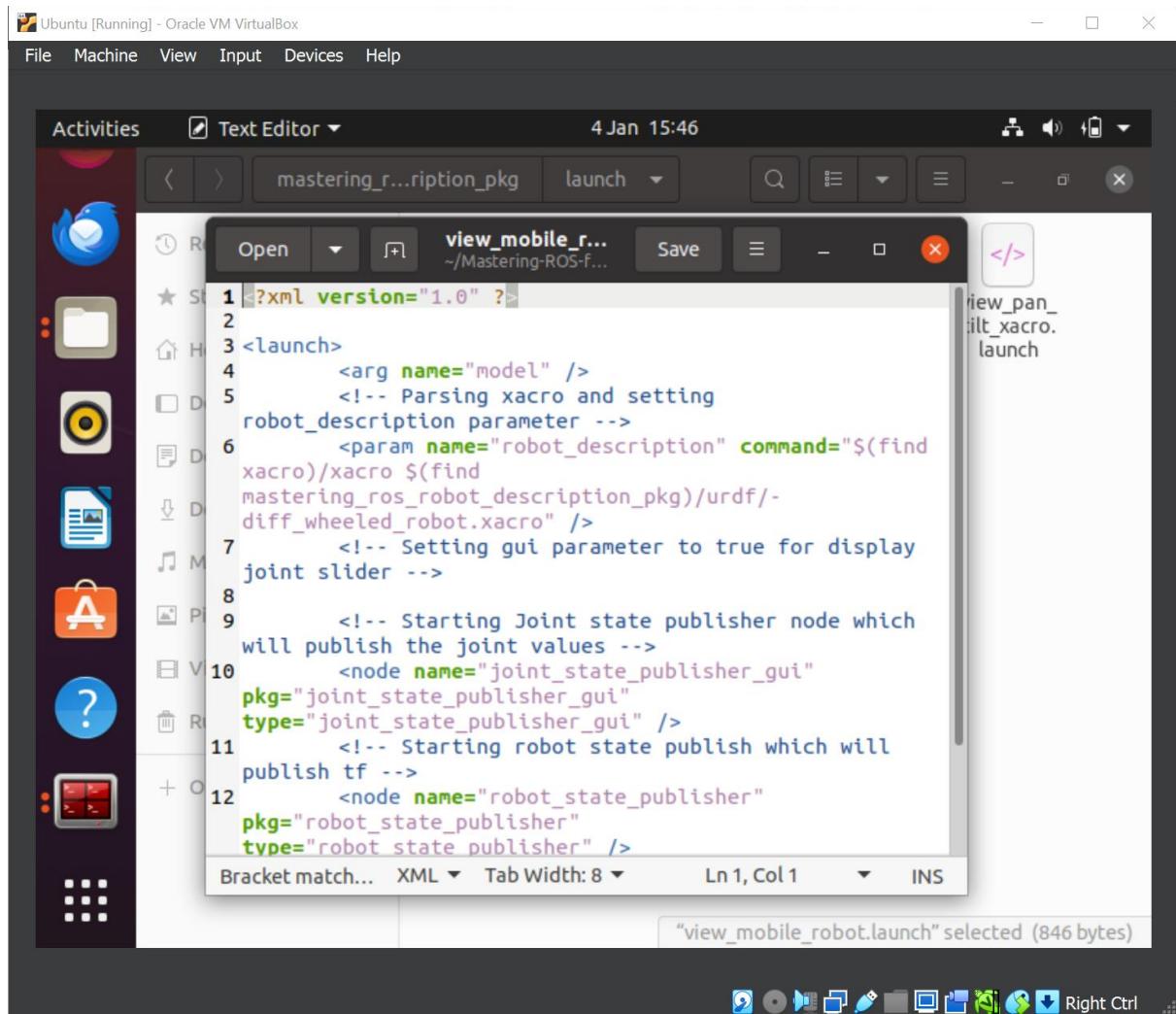
diff_wheeled_robot.xacro



```
1 <?xml version="1.0"?>
2 <robot name="differential_wheeled_robot"
  xmlns:xacro="http://ros.org/wiki/xacro">
3
4   <xacro:include filename="$(find
      mastering_ros_robot_description_pkg)/urdf/
    wheel.urdf.xacro" />
5
6
7 <!-- Defining the colors used in this robot -->
8   <material name="Black">
9     <color rgba="0.0 0.0 0.0 1.0"/>
10  </material>
11
12 <material name="Red">
13   <color rgba="0.8 0.0 0.0 1.0"/>
14 </material>
15
16 <material name="White">
17   <color rgba="1.0 1.0 1.0 1.0"/>
18 </material>
19
20
```

File launch `view_mobile_robot.launch` memuat model robot ke RViz menggunakan xacro dan meluncurkan node joint state publisher, robot state publisher, dan RViz untuk visualisasi.

`view_mobile_robot.launch`



```
1 <?xml version="1.0" ?>
2
3 <launch>
4     <arg name="model" />
5     <!-- Parsing xacro and setting
       robot_description parameter -->
6     <param name="robot_description" command="$(find
      xacro)/xacro $(find
      mastering_ros_robot_description_pkg)/urdf/-
      diff_wheeled_robot.xacro" />
7     <!-- Setting gui parameter to true for display
       joint slider -->
8
9     <!-- Starting Joint state publisher node which
       will publish the joint values -->
10    <node name="joint_state_publisher_gui"
11        pkg="joint_state_publisher_gui"
12        type="joint_state_publisher_gui" />
13    <!-- Starting robot state publish which will
       publish tf -->
14    <node name="robot_state_publisher"
15        pkg="robot_state_publisher"
16        type="robot_state_publisher" />
```

The screenshot shows a Linux desktop environment with a text editor window open. The window title is "view_mobile_robot.launch". The code in the editor is an XML launch file for a mobile robot. It includes sections for parsing xacro files, setting the robot description parameter, starting a joint state publisher node, and starting a robot state publisher node. The code uses XML syntax with tags like <launch>, <arg>, <param>, <node>, and <!--> comments. The text editor has a dark theme and includes standard features like file operations, search, and tabs.

Chapter 4 Simulating Robots Using ROS and Gazebo

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Setelah merancang model 3D sebuah robot, tahap berikutnya adalah mensimulasikannya. Simulasi robot memberikan gambaran tentang bagaimana robot beroperasi dalam lingkungan virtual.

Dalam bab ini, kita akan menggunakan simulator Gazebo (<http://www.gazebosim.org/>) untuk mensimulasikan robot dengan Seven Degrees Of Freedom (DOF) arms dan robot beroda mobile.

Gazebo adalah simulator multi-robot untuk simulasi robot kompleks di dalam dan di luar ruangan. Simulator ini memungkinkan simulasi robot kompleks, sensor robot, dan berbagai objek 3D. Gazebo sudah memiliki model simulasi untuk robot, sensor, dan objek 3D populer di repositorinya (https://bitbucket.org/osrf/gazebo_models/). Model-model ini dapat digunakan langsung tanpa harus membuat yang baru.

Gazebo terintegrasi dengan ROS dengan baik berkat antarmuka ROS yang memungkinkan kendali penuh terhadap Gazebo di ROS. Meskipun Gazebo dapat diinstal tanpa ROS, antarmuka ROS-Gazebo harus diinstal untuk berkomunikasi dari ROS ke Gazebo.

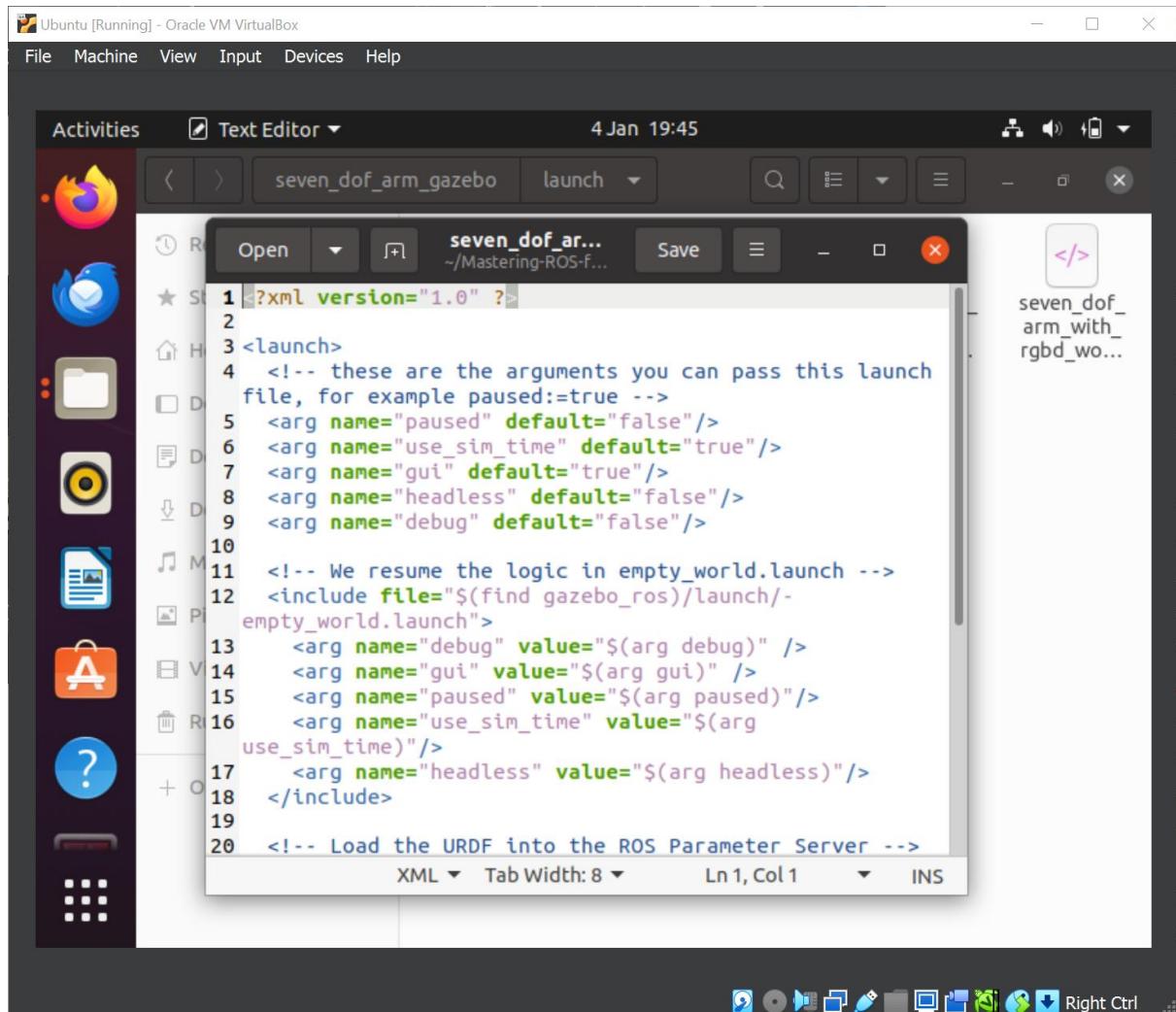
Simulating the robotic arm using Gazebo and ROS

- Setelah merancang lengan robot dengan tujuh derajat kebebasan (DOF), langkah selanjutnya adalah mensimulasikannya di Gazebo menggunakan ROS.
- Instal package-package berikut sebelum memulai dengan Gazebo dan ROS.
- Gazebo memiliki beberapa package seperti `gazebo_ros_pkgs`, `gazebo-msgs`, `gazebo-plugins`, dan `gazebo-ros-control`, yang dibutuhkan untuk berinteraksi dengan ROS.

Creating the robotic arm simulation model for Gazebo

- Pembuatan model simulasi untuk lengan robotik dilakukan dengan memperbarui deskripsi robot yang sudah ada dan menambahkan parameter simulasi.
- Package `seven_dof_arm_gazebo` dapat dibuat menggunakan perintah `catkin_create_pkg`.
- Model simulasi lengkap terdapat dalam file `seven_dof_arm.xacro` di folder `mastering_ros_robot_description_pkgs/urdf/`.

seven_dof_arm_world.launch



```
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor 4 Jan 19:45
seven_dof_arm_gazebo launch
Open Save
seven_dof_ar...
~/Mastering-ROS-F...
1 <?xml version="1.0" ?>
2
3 <launch>
4   <!-- these are the arguments you can pass this launch
      file, for example paused:=true -->
5   <arg name="paused" default="false"/>
6   <arg name="use_sim_time" default="true"/>
7   <arg name="gui" default="true"/>
8   <arg name="headless" default="false"/>
9   <arg name="debug" default="false"/>
10
11  <!-- We resume the logic in empty_world.launch -->
12  <include file="$(find gazebo_ros)/launch/
empty_world.launch">
13    <arg name="debug" value="$(arg debug)" />
14    <arg name="gui" value="$(arg gui)" />
15    <arg name="paused" value="$(arg paused)" />
16    <arg name="use_sim_time" value="$(arg
      use_sim_time)"/>
17    <arg name="headless" value="$(arg headless)" />
18  </include>
19
20  <!-- Load the URDF into the ROS Parameter Server -->
```

Adding colors and textures to the Gazebo robot model

- Tag `gazebo` di dalam file `.xacro` digunakan untuk memberikan tekstur dan warna pada link-link robot.
- Setiap tag `gazebo` merujuk pada link tertentu dari model robot.

Adding transmission tags to actuate the model

- Tag `<transmission>` digunakan untuk mengaktifkan robot menggunakan kontroler ROS.
- Macro `transmission_block` digunakan untuk mendefinisikan elemen `<transmission>` dan menghubungkannya dengan joint (joint) dan aktuator.

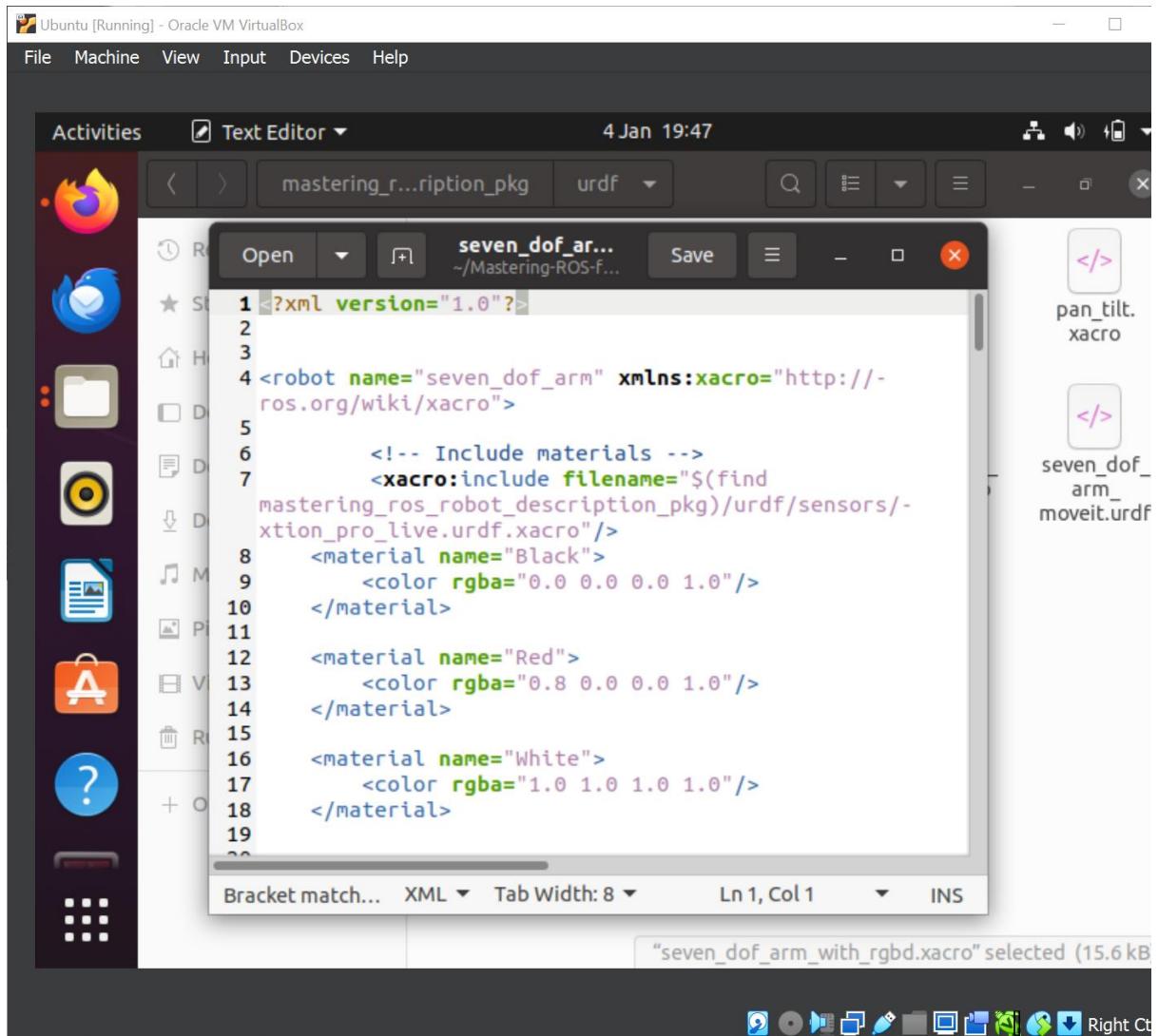
Adding the `gazebo_ros_control` plugin

- Plugin ini diperlukan untuk mengurai tag-tag transmisi dan menetapkan antarmuka perangkat keras yang sesuai serta manajemen kontrol.
- Ditambahkan ke dalam file `.xacro` dengan konfigurasi berupa namespace robot dan interface perangkat keras default.

Adding a 3D vision sensor to Gazebo

- Robot di Gazebo dapat memiliki sensor yang mensimulasikan pergerakan dan fisika robot serta berbagai sensor.
- Untuk menambahkan sensor penglihatan 3D (seperti sensor RGB-D) seperti Asus Xtion Pro, kita memodelkannya dan menyertakan definisi Gazebo-ROS plugin dalam file **.xacro**.
- Pengaturan plugin termasuk nama kamera, topik gambar, dan pembaruan.

seven_dof_arm_with_rgbd.xacro

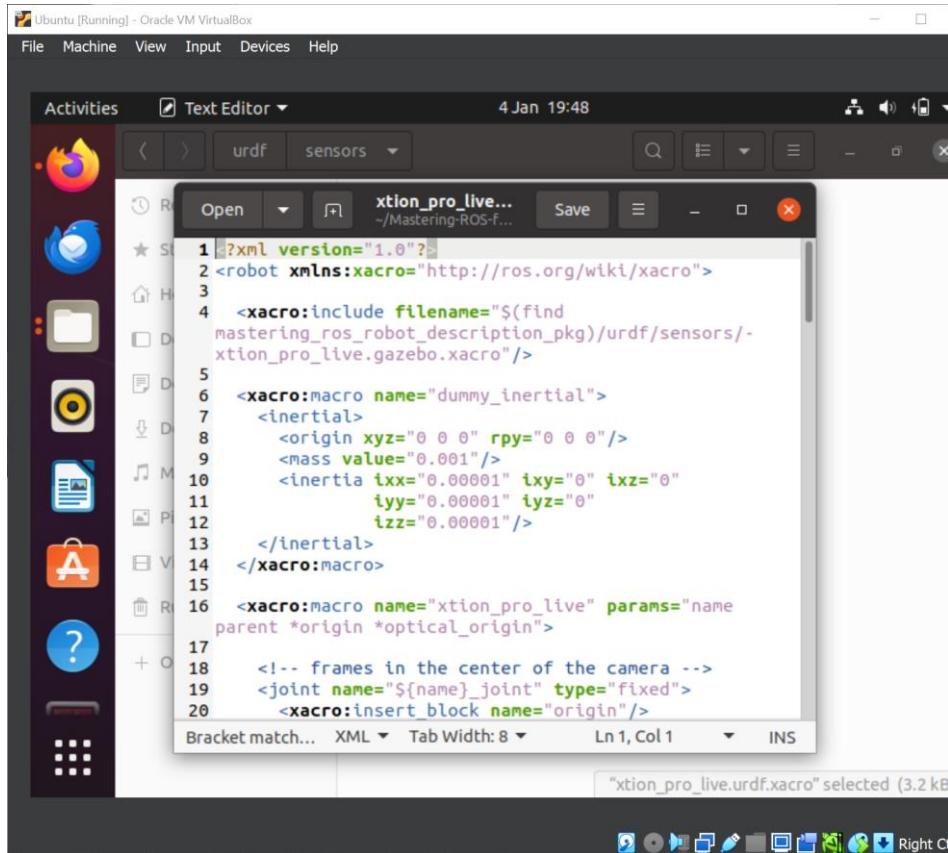


The screenshot shows a Linux desktop environment with a text editor window open. The window title is "seven_dof_ar...". The code in the editor is:

```
1 <?xml version="1.0"?>
2
3
4 <robot name="seven_dof_arm" xmlns:xacro="http://
ros.org/wiki/xacro">
5
6     <!-- Include materials -->
7     <xacro:include filename="$(find
mastering_ros_robot_description_pkg)/urdf/sensors/
xtion_pro_live.urdf.xacro"/>
8     <material name="Black">
9         <color rgba="0.0 0.0 0.0 1.0"/>
10    </material>
11
12    <material name="Red">
13        <color rgba="0.8 0.0 0.0 1.0"/>
14    </material>
15
16    <material name="White">
17        <color rgba="1.0 1.0 1.0 1.0"/>
18    </material>
19
20
```

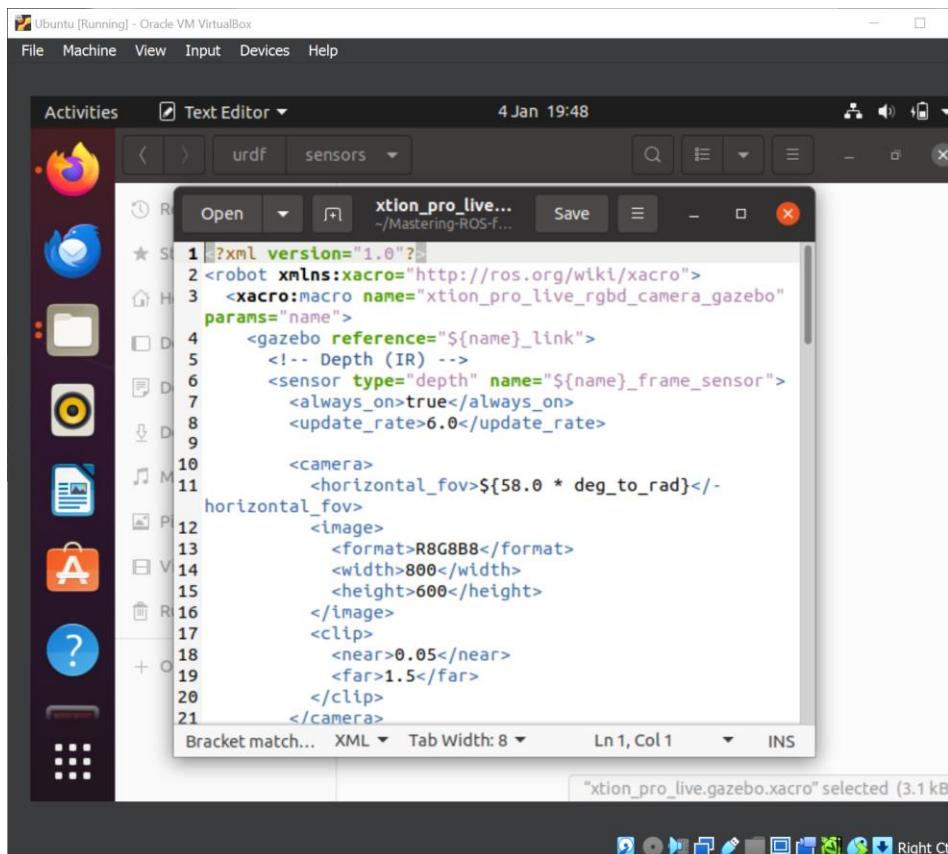
The code defines a robot named "seven_dof_arm" and includes another XML file for sensor definitions. It also defines three materials: Black, Red, and White, each with a specific color value.

xtion_pro_live.urdf.xacro



```
1 <?xml version="1.0"?>
2 <robot xmlns:xacro="http://ros.org/wiki/xacro">
3
4   <xacro:include filename="$(find
5     mastering_ros_robot_description_pkg)/urdf/sensors/
6     xtion_pro_live.gazebo.xacro"/>
7
8   <xacro:macro name="dummy_inertial">
9     <inertial>
10      <origin xyz="0 0 0" rpy="0 0 0"/>
11      <mass value="0.001"/>
12      <inertia ixx="0.00001" ixy="0" ixz="0"
13        iyy="0.00001" tyz="0"
14        tzz="0.00001"/>
15   </inertial>
16   </xacro:macro>
17
18   <xacro:macro name="xtion_pro_live" params="name
19     parent *origin *optical_origin">
20     <!-- frames in the center of the camera -->
21     <joint name="${name}_joint" type="fixed">
22       <xacro:insert_block name="origin"/>
```

xtion_pro_live.gazebo.xacro



```
1 <?xml version="1.0"?>
2 <robot xmlns:xacro="http://ros.org/wiki/xacro">
3
4   <xacro:macro name="xtion_pro_live_rgbd_camera_gazebo"
5     params="name">
6     <gazebo reference="${name}_link">
7       <!-- Depth (IR) -->
8       <sensor type="depth" name="${name}_frame_sensor">
9         <always_on>true</always_on>
10        <update_rate>6.0</update_rate>
11
12        <camera>
13          <horizontal_fov>${58.0 * deg_to_rad}<-
14            horizontal_fov>
15          <image>
16            <format>R8G8B8</format>
17            <width>800</width>
18            <height>600</height>
19          </image>
20          <clip>
21            <near>0.05</near>
22            <far>1.5</far>
23          </clip>
24        </camera>
25
26      </gazebo>
```

Simulating the robotic arm with Xtion Pro

Setelah merancang plugin kamera di Gazebo, kita dapat meluncurkan simulasi lengkap menggunakan perintah `roslaunch seven_dof_arm_gazebo seven_dof_arm_with_rgbd_world.launch`. Dalam simulasi ini, model robot dengan sensor di bagian atas lengannya dapat dilihat di lingkungan Gazebo. Langkah-langkah selanjutnya melibatkan pengujian sensor RGB-D yang mensimulasikan input kamera secara langsung.

Visualizing the 3D sensor data

Setelah mensimulasikan robot dan sensor di Gazebo, kita dapat memeriksa topik-topik yang dihasilkan oleh plugin sensor. Image-view tool dapat digunakan untuk memeriksa gambar RGB, IR, dan kedalaman dari sensor 3D. Selain itu, kita dapat memonitor data awan titik sensor menggunakan perangkat lunak RViz.

Moving the robot joints using ROS controllers in Gazebo

Bagian ini membahas cara menggerakkan setiap joint robot di lingkungan simulasi Gazebo menggunakan kontroler ROS. Kontroler ROS membutuhkan antarmuka perangkat keras yang sesuai dengan parameter yang ditentukan dalam tag transmisi. Kontroler ROS ini menyediakan mekanisme umpan balik untuk menerima titik set dan mengontrol keluaran berdasarkan umpan balik dari aktuator.

Understanding the ros_control packages

Package `ros_control` terdiri dari berbagai modul dan alat yang dapat digunakan oleh kontroler. Ini mencakup `control_toolbox`, `controller_interface`, `controller_manager`, `controller_manager_msgs`, `hardware_interface`, dan `transmission_interface`. Setiap package menyumbang pada fungsionalitas kontroler, manajemen kontroler, dan antarmuka perangkat keras di ROS.

Different types of ROS controllers and hardware interfaces

Kontroler ROS seperti `joint_position_controller`, `joint_state_controller`, dan `joint_effort_controller` memberikan kemampuan kontrol yang berbeda pada robot. Sementara itu, antarmuka perangkat keras seperti Effort Joint Interface, Velocity Joint Interface, dan Position Joint Interface digunakan untuk mengirim perintah ke perangkat keras robot.

How the ROS controller interacts with Gazebo

Interaksi antara kontroler ROS dan Gazebo melibatkan penggunaan antarmuka perangkat keras. Antarmuka perangkat keras bertindak sebagai perantara antara kontroler ROS dan

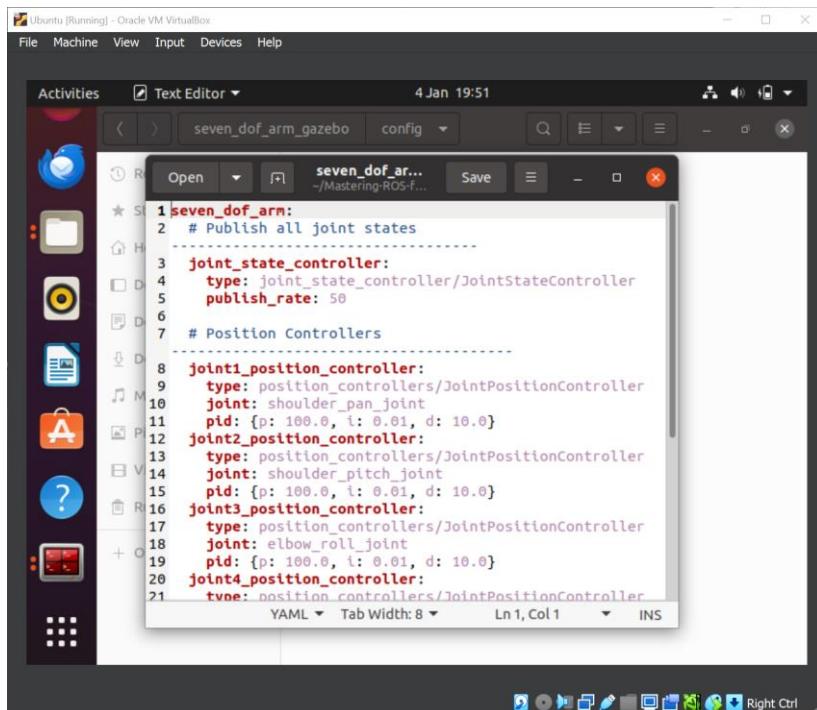
perangkat keras fisik atau simulasi. Informasi mengenai setiap joint dan aktuator disalurkan dari kontroler ke antarmuka perangkat keras, dan nilai-nilai tersebut dapat diteruskan ke simulasi Gazebo atau perangkat keras fisik sesungguhnya.

Interfacing the joint state controllers and joint position controllers with the arm

Menghubungkan kontroler robot dengan setiap joint merupakan tugas yang sederhana. Langkah pertama melibatkan penulisan file konfigurasi untuk dua kontroler. Kontroler status joint akan mempublikasikan status joint lengan, sementara kontroler posisi joint dapat menerima posisi tujuan untuk setiap joint dan dapat menggerakkan masing-masing joint. File konfigurasi untuk kontroler dapat ditemukan di seven_dof_arm_gazebo_control.yaml.

Pertama, kita mendefinisikan kontroler status joint yang mempublikasikan status joint lengan pada tingkat 50 Hz. Selanjutnya, kita perlu mendefinisikan kontroler posisi untuk setiap joint dengan parameter PID tertentu. Semua kontroler ini terletak di dalam namespace seven_dof_arm.

seven_dof_arm_gazebo_control.yaml

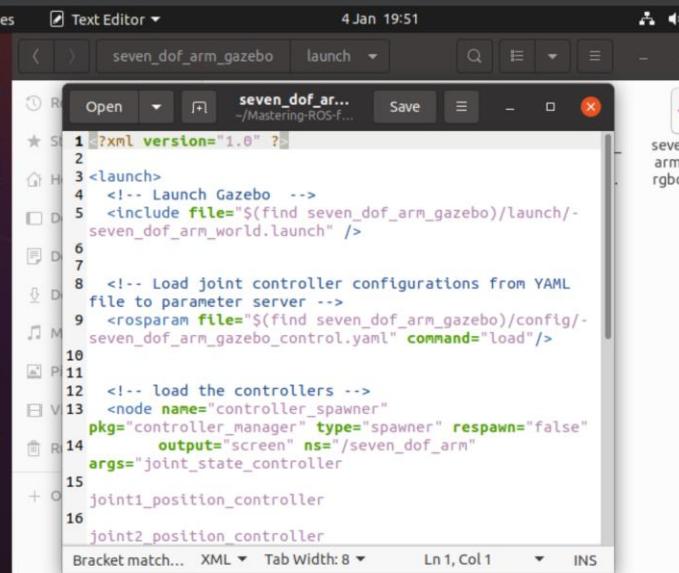


```
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor 4 Jan 19:51
seven_dof_ar... ~/Mastering-ROS-f...
Open Save
seven_dof_arm:
# Publish all joint states
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
# Position Controllers
joint1_position_controller:
  type: position_controllers/JointPositionController
  joint: shoulder_pan_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}
joint2_position_controller:
  type: position_controllers/JointPositionController
  joint: shoulder_pitch_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}
joint3_position_controller:
  type: position_controllers/JointPositionController
  joint: elbow_roll_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}
joint4_position_controller:
  type: position_controllers/JointPositionController
```

Launching the ROS controllers with Gazebo

Setelah konfigurasi kontroler selesai, kita dapat membuat file peluncuran yang memulai semua kontroler bersama dengan simulasi Gazebo. File peluncuran ini mencakup pengaturan simulasi Gazebo, memuat konfigurasi kontroler, dan menjalankan robot state publisher untuk mempublikasikan status dan transformasi joint.

seven_dof_arm_gazebo_control.launch



The screenshot shows a typical Ubuntu desktop environment running in Oracle VM VirtualBox. The terminal window at the bottom has the command "roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo.launch" entered and is waiting for input. A file browser window is open, showing the file "seven_dof_arm_gazebo.launch" located in the "/Mastering-ROS-f..." directory. The code in the file is as follows:

```
1 <?xml version="1.0" ?>
2
3 <launch>
4   <!-- Launch Gazebo -->
5   <include file="$(find seven_dof_arm_gazebo)/launch/->
 seven_dof_arm_world.launch" />
6
7
8   <!-- Load joint controller configurations from YAML
file to parameter server -->
9   <rosparam file="$(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"
14     pkg="controller_manager" type="spawner" respawn="false"
15       output="screen" ns="/seven_dof_arm"
16     args="joint_state_controller
17       joint1_position_controller
18       joint2_position_controller
```

Simulating a differential wheeled robot in Gazebo

Setelah mensimulasikan lengan robot, langkah selanjutnya adalah menyiapkan simulasi untuk robot beroda diferensial yang telah dirancang sebelumnya. Dengan menggunakan file peluncuran `diff_wheeled_gazebo.launch`, kita dapat memuat model robot beroda diferensial ke dalam lingkungan Gazebo.

diff_wheeled_robot.xacro

Ubuntu [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Text Editor 4 Jan 19:52

mastering_r...ption_pkg urdf

diff_wheeled... /-Mastering-ROS-f...

Open Save

```
1 <?xml version="1.0"?>
2 <robot name="differential_wheeled_robot"
  xmlns:xacro="http://ros.org/wtiki/xacro">
3
4   <xacro:include filename="$(find
      mastering_ros_robot_description_pkg)/urdf/
    wheel.urdf.xacro" />
5
6
7 <!-- Defining the colors used in this robot -->
8   <material name="Black">
9     <color rgba="0.0 0.0 0.0 1.0"/>
10  </material>
11
12  <material name="Red">
13    <color rgba="0.8 0.0 0.0 1.0"/>
14  </material>
15
16  <material name="White">
17    <color rgba="1.0 1.0 1.0 1.0"/>
18  </material>
19
20
```

Bracket match... XML Tab Width: 8 Ln 1, Col 1 INS

"diff_wheeled_robot.xacro" selected (6.9 kB)

diff_wheeled_gazebo.launch

A screenshot of a Linux desktop environment (Ubuntu) showing a terminal window titled "diff_wheeled_gazebo.launch". The terminal window displays an XML launch file. The file includes code for launching Gazebo, specifying arguments like paused, use_sim_time, gui, headless, and debug. It also includes an include directive for "empty_world.launch". The terminal status bar shows the file is selected (1.2 kB).

```
1 <?xml version="1.0" ?>
2 <launch>
3
4   <!-- these are the arguments you can pass this launch
file, for example paused:=true -->
5   <arg name="paused" default="false"/>
6   <arg name="use_sim_time" default="true"/>
7   <arg name="gui" default="true"/>
8   <arg name="headless" default="false"/>
9   <arg name="debug" default="false"/>
10
11  <!-- We resume the logic in empty_world.launch -->
12  <include file="$(find gazebo_ros)/launch/
empty_world.launch">
13    <arg name="debug" value="$(arg debug)" />
14    <arg name="gui" value="$(arg gui)" />
15    <arg name="paused" value="$(arg paused)" />
16    <arg name="use_sim_time" value="$(arg
use_sim_time)" />
17    <arg name="headless" value="$(arg headless)" />
18  </include>
19
20
```

Adding the laser scanner to Gazebo

Untuk melengkapi simulasi, kita menambahkan pemindai laser ke robot beroda diferensial. Kode tambahan diperlukan untuk mendefinisikan tautan dan sambungan pemindai laser serta konfigurasi plugin Gazebo untuk mensimulasikan pemindai laser.

diff_wheeled_robot.xacro

A screenshot of a Linux desktop environment (Ubuntu) showing a terminal window titled "diff_wheeled_robot.xacro". The terminal window displays an XACRO file for a differential-wheeled robot. The file includes code for defining base links, wheels, and a laser scanner. It uses macros for calculating cylinder inertia and defines a base footprint link. The terminal status bar shows the file is selected (1.2 kB).

```
value="0.3" />
59  <xacro:property name="base_z_origin_to_wheel_origin"
value="0.0" />
60
61  <!-- Hokuyo Laser scanner -->
62  <xacro:property name="hokuyo_size" value="0.05" />
63
64  <!-- Macro for calculating inertia of cylinder -->
65  <xacro:macro name="cylinder_inertia" params="m r h">
66    <inertia ixx="${m*(3*r*r+h*h)/12}" ixy="0" ixz
= "0"
67      iyy="${m*(3*r*r+h*h)/12}" iyx = "0" iyz
68      izz="${m*r*r/2}" />
69  </xacro:macro>
70
71  <!-- BASE-FOOTPRINT -->
72  <!-- base_footprint is a fictitious link(frame) that
is on the ground right below base_link origin -->
73  <link name="base_footprint">
74    <inertial>
75      <mass value="0.0001" />
76      <origin xyz="0 0 0" />
77      <inertia ixx="0.0001" ixy="0_0" ixz="0_0"
```

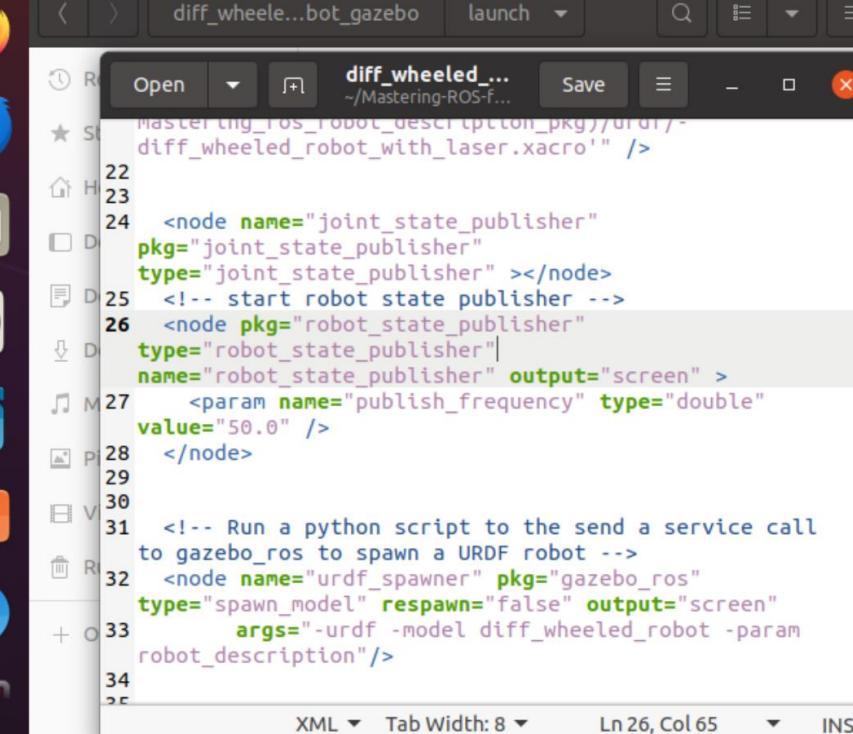
Moving the mobile robot in Gazebo

Robot yang digunakan adalah robot diferensial dengan dua roda dan dua roda penumpu. Untuk membuat robot bergerak di Gazebo, kita memerlukan plugin Gazebo-ROS bernama libgazebo_ros_diff_drive.so. Plugin ini memberikan perilaku kendaraan beroda diferensial ke robot kita dengan menerima perintah kecepatan melalui topik cmd_vel. Perintah kecepatan dapat dipublikasikan menggunakan node ROS teleop untuk menggerakkan robot di lingkungan simulasi Gazebo.

Adding joint state publishers to the launch file

Setelah menambahkan plugin differential drive, kita perlu menambahkan joint state publishers ke dalam file peluncuran yang sudah ada, atau kita dapat membuat file peluncuran baru. File peluncuran terakhir yang baru dapat ditemukan dalam diff_wheeled_robot_gazebo/launch dengan nama diff_wheeled_gazebo_full.launch. File peluncuran ini berisi penerbit status joint, yang membantu pengembang untuk memvisualisasikan tf di rviz. Berikut adalah baris tambahan yang perlu ditambahkan ke dalam file peluncuran ini untuk penerbitan status joint:

diff_wheeled_gazebo_full.launch



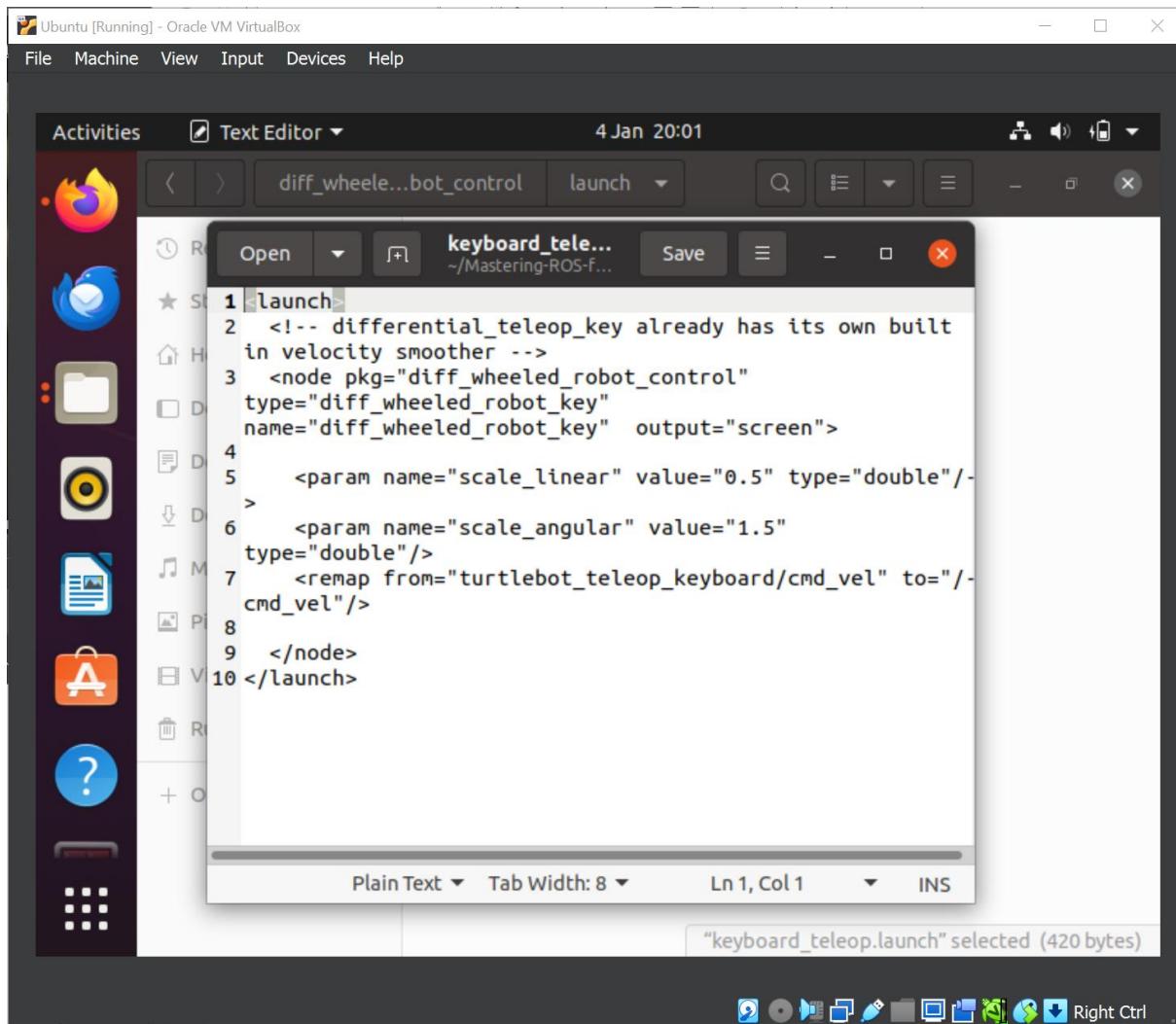
The screenshot shows a terminal window titled "Text Editor" running on an Ubuntu desktop. The window displays a ROS launch file named "diff_wheeled_robot_gazebo.launch". The code in the file is as follows:

```
diff_wheeled_...
~/Mastering-ROS-f...
22
23
24    <node name="joint_state_publisher"
25      pkg="joint_state_publisher"
26      type="joint_state_publisher" ></node>
27      <!-- start robot state publisher -->
28      <node pkg="robot_state_publisher"
29        type="robot_state_publisher"
30        name="robot_state_publisher" output="screen" >
31          <param name="publish_frequency" type="double"
32            value="50.0" />
33          </node>
34
35      <!-- Run a python script to the send a service call
36          to gazebo_ros to spawn a URDF robot -->
37      <node name="urdf_spawner" pkg="gazebo_ros"
38        type="spawn_model" respawn="false" output="screen"
39        args="--urdf -model diff_wheeled_robot -param
40        robot_description"/>
41
42
```

Adding the ROS teleop node

Node teleop ROS mempublikasikan perintah Twist ROS dengan menggunakan input keyboard. Dari node ini, kita dapat menghasilkan kecepatan linear dan angular, dan sudah ada implementasi node teleop standar yang dapat kita gunakan kembali. Teleop diimplementasikan dalam package `diff_wheeled_robot_control`. File skrip `diff_wheeled_robot_key`, yang merupakan node teleop, dapat ditemukan di folder `script`. Untuk menggunakan package ini, Anda perlu mengunduhnya dari repositori Git sebelumnya dan menginstal package `joy_node` jika belum terpasang.

keyboard_teleop.launch



The screenshot shows a Linux desktop environment with a text editor window open. The window title is "keyboard_teleop.launch". The code inside the window is:

```
1 <launch>
2   <!-- differential_teleop_key already has its own built
      in velocity smoother -->
3   <node pkg="diff_wheeled_robot_control"
      type="diff_wheeled_robot_key"
      name="diff_wheeled_robot_key"  output="screen">
4
5     <param name="scale_linear" value="0.5" type="double"/-
>
6     <param name="scale_angular" value="1.5"
      type="double"/>
7     <remap from="turtlebot_teleop_keyboard/cmd_vel" to="/-/
      cmd_vel"/>
8
9   </node>
10 </launch>
```

The text editor interface includes a toolbar at the top, a sidebar with icons on the left, and various status indicators at the bottom. A status bar at the bottom right of the window says "keyboard_teleop.launch selected (420 bytes)".

Chapter 5 Simulating Robots Using ROS, CoppeliaSim, and Webots

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Setelah mempelajari cara mensimulasikan robot dengan Gazebo, pada bab ini kita akan membahas penggunaan dua perangkat lunak simulasi robot yang kuat: CoppeliaSim (<http://www.coppeliarobotics.com>) dan Webots (<https://cyberbotics.com>).

Kedua simulator robot ini bersifat multiplatform. CoppeliaSim dikembangkan oleh Coppelia Robotics dan menawarkan banyak model simulasi robot industri dan mobile yang populer, siap digunakan, serta berbagai fungsionalitas yang dapat dengan mudah diintegrasikan dan dikombinasikan melalui antarmuka pemrograman aplikasi (API) yang didedikasikan. Selain itu, CoppeliaSim dapat beroperasi dengan Robot Operating System (ROS) menggunakan antarmuka komunikasi yang memungkinkan kita mengendalikan adegan simulasi dan robot melalui topik dan layanan. Sama seperti Gazebo, CoppeliaSim dapat digunakan sebagai perangkat lunak mandiri, namun plugin eksternal harus diinstal untuk bekerja dengan ROS. Sedangkan Webots, adalah perangkat lunak sumber terbuka gratis yang digunakan untuk mensimulasikan robot 3D. Webots dikembangkan oleh Cyberbotics Ltd. dan sejak Desember 2018 telah dirilis di bawah lisensi sumber terbuka.

Seperti halnya dengan CoppeliaSim, Webots dapat dengan mudah dihubungkan dengan ROS. Dalam bab ini, kita akan mempelajari cara mengatur simulator ini dan menghubungkannya dengan jaringan ROS. Kami akan membahas beberapa kode awal untuk memahami cara kerja keduanya sebagai perangkat lunak mandiri dan bagaimana mereka dapat digunakan dengan layanan dan topik ROS.

Setting up CoppeliaSim with ROS

Sebelum mulai bekerja dengan CoppeliaSim, kita perlu menginstalnya di sistem kita dan mengonfigurasi lingkungan agar dapat memulai jembatan komunikasi antara ROS dan adegan simulasi. CoppeliaSim adalah perangkat lunak lintas platform, tersedia untuk sistem operasi berbeda seperti Windows, macOS, dan Linux. Dikembangkan oleh Coppelia Robotics GmbH, CoppeliaSim didistribusikan dengan lisensi edukasi dan komersial gratis. Unduh versi terbaru simulator CoppeliaSim dari halaman unduhan Coppelia Robotics di <http://www.coppeliarobotics.com/downloads.html>, pilih versi edu untuk Linux. Dalam bab ini, kami akan merujuk pada versi CoppeliaSim 4.2.0.

Setelah selesai mengunduh, ekstrak arsipnya. Pindah ke folder unduhan Anda dan gunakan perintah berikut:

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

Versi ini didukung oleh Ubuntu versi 20.04. Sebaiknya, ubah nama folder ini menjadi sesuatu yang lebih intuitif, misalnya:

```
mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim
```

Untuk dengan mudah mengakses sumber daya CoppeliaSim, disarankan untuk mengatur variabel lingkungan COPPELIASIM_ROOT yang menunjuk ke folder utama CoppeliaSim, seperti ini:

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

Di sini, **/path/to/CoppeliaSim/folder** adalah jalur absolut ke folder yang diekstrak.

CoppeliaSim menawarkan mode berikut untuk mengendalikan robot yang disimulasikan dari aplikasi eksternal:

- Remote application programming interface (API): API jarak jauh CoppeliaSim terdiri dari beberapa fungsi yang dapat dipanggil dari aplikasi eksternal yang dikembangkan dalam C/C++, Python, Lua, atau MATLAB. API jarak jauh berinteraksi dengan CoppeliaSim melalui jaringan menggunakan komunikasi soket. Dapat mengintegrasikan API jarak jauh pada node C++ atau Python Anda untuk menghubungkan ROS dengan adegan simulasi.
- RosInterface: Antarmuka saat ini untuk mengaktifkan komunikasi antara ROS dan CoppeliaSim. Dalam bab ini, kita akan membahas cara berinteraksi dengan CoppeliaSim menggunakan plugin RosInterface yang mereplikasi fungsionalitas API jarak jauh secara transparan. Melalui antarmuka ini, CoppeliaSim akan bertindak sebagai node ROS yang dapat berkomunikasi dengan node lain melalui layanan ROS, penerbit ROS, dan pelanggan ROS.

Untuk mengaktifkan antarmuka komunikasi ROS, jalankan perintah roscore di mesin Anda sebelum membuka simulator. Selanjutnya, untuk membuka CoppeliaSim, gunakan perintah berikut:

```
cd $COPPELIASIM_ROOT ./coppeliaSim.sh
```

Selama startup, semua plugin yang diinstal di sistem akan dimuat. Untuk memastikan semuanya berfungsi dengan baik, Anda dapat memeriksa daftar node yang berjalan di sistem

Anda setelah meluncurkan CoppeliaSim. Sebagai contoh, jika melihat node-node ROS yang aktif, seperti sim_ros_interface, maka plugin RosInterface telah berhasil dimulai.

Untuk menjelajahi fungsionalitas plugin RosInterface, Anda dapat membuka skenario **plugin_publisher_subscriber.ttt** yang terletak di folder **csim_demo_pkg/scene** dari kode yang disediakan dengan buku ini. Setelah membuka skenario ini, jendela simulasi harus muncul, memperlihatkan robot yang dilengkapi dua kamera.

Dalam skenario ini, kamera pasif menampilkan gambar yang dipublikasikan dari kamera aktif, menerima data visual langsung dari kerangka ROS. Anda juga dapat memvisualisasikan aliran video yang dipublikasikan oleh CoppeliaSim menggunakan paket `image_view` dengan menjalankan perintah:

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

Kemudian, dapat dibahas cara menghubungkan CoppeliaSim dan ROS menggunakan plugin RosInterface.

Understanding the RosInterface plugin

Plugin RosInterface adalah bagian dari kerangka kerja API CoppeliaSim. Meskipun plugin telah terinstal dengan benar di sistem Anda, operasi pemuatan akan gagal jika roscore tidak berjalan pada saat itu. Untuk mencegah perilaku yang tidak terduga, kita akan melihat cara memeriksa apakah plugin RosInterface berfungsi dengan baik. Selanjutnya, kita akan membahas cara berinteraksi dengan CoppeliaSim menggunakan topik ROS.

Interacting with CoppeliaSim using ROS topics

Pada bagian ini, kita akan membahas cara menggunakan topik ROS untuk berkomunikasi dengan CoppeliaSim. Ini berguna ketika kita ingin mengirim informasi ke objek simulasi atau mengambil data yang dihasilkan oleh sensor atau aktuator robot.

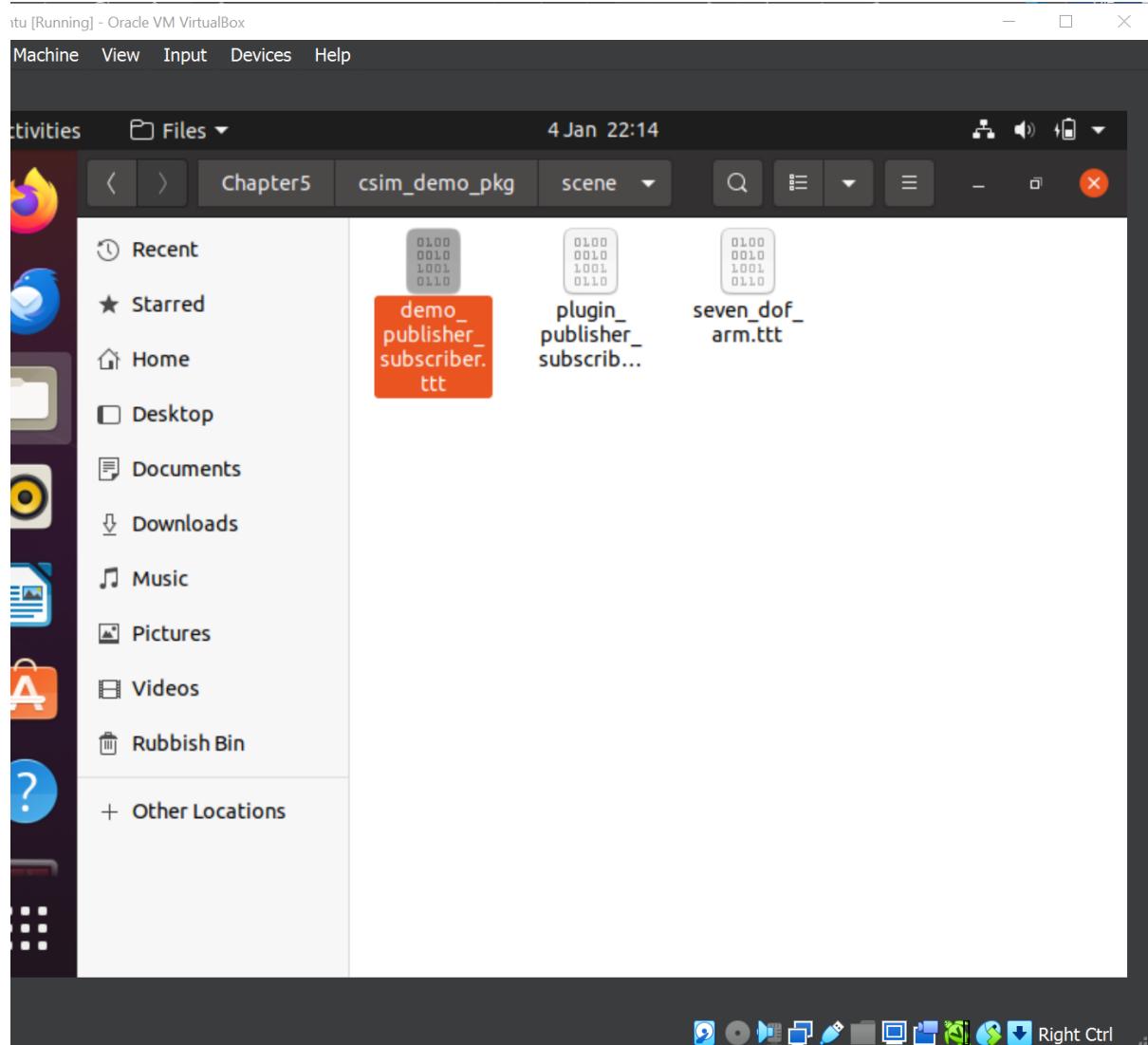
Cara paling umum untuk memprogram adegan simulasi di simulator ini adalah dengan menggunakan skrip Lua. Setiap objek dari adegan dapat dihubungkan dengan skrip yang secara otomatis dipanggil ketika simulasi dimulai dan dijalankan secara siklik selama waktu simulasi.

Sebagai contoh, kita akan membuat adegan dengan dua objek. Satu akan diprogram untuk mempublikasikan data integer dari topik tertentu, sedangkan yang lainnya akan berlangganan topik ini, menampilkan data float di konsol CoppeliaSim.

Gunakan menu tarik-turun pada panel hirarki adegan, pilih Entri Tambah | Dummy. Kita dapat membuat dua objek, objek `dummy_publisher` dan objek `dummy_subscriber`, dan mengasosiasikan skrip dengan masing-masing dari mereka. Gunakan tombol kanan mouse pada objek yang dibuat, dan pilih Entri Tambah | Anak skrip terkait | Non threaded, seperti yang ditunjukkan dalam tangkapan layar.

Selain itu, kita dapat langsung memuat adegan simulasi dengan membuka file **demo_publisher_subscriber.ttt** yang terletak di folder **csim_demo_pkg** dari sumber kode buku ini di direktori scene.

demo_publisher_subscriber.ttt

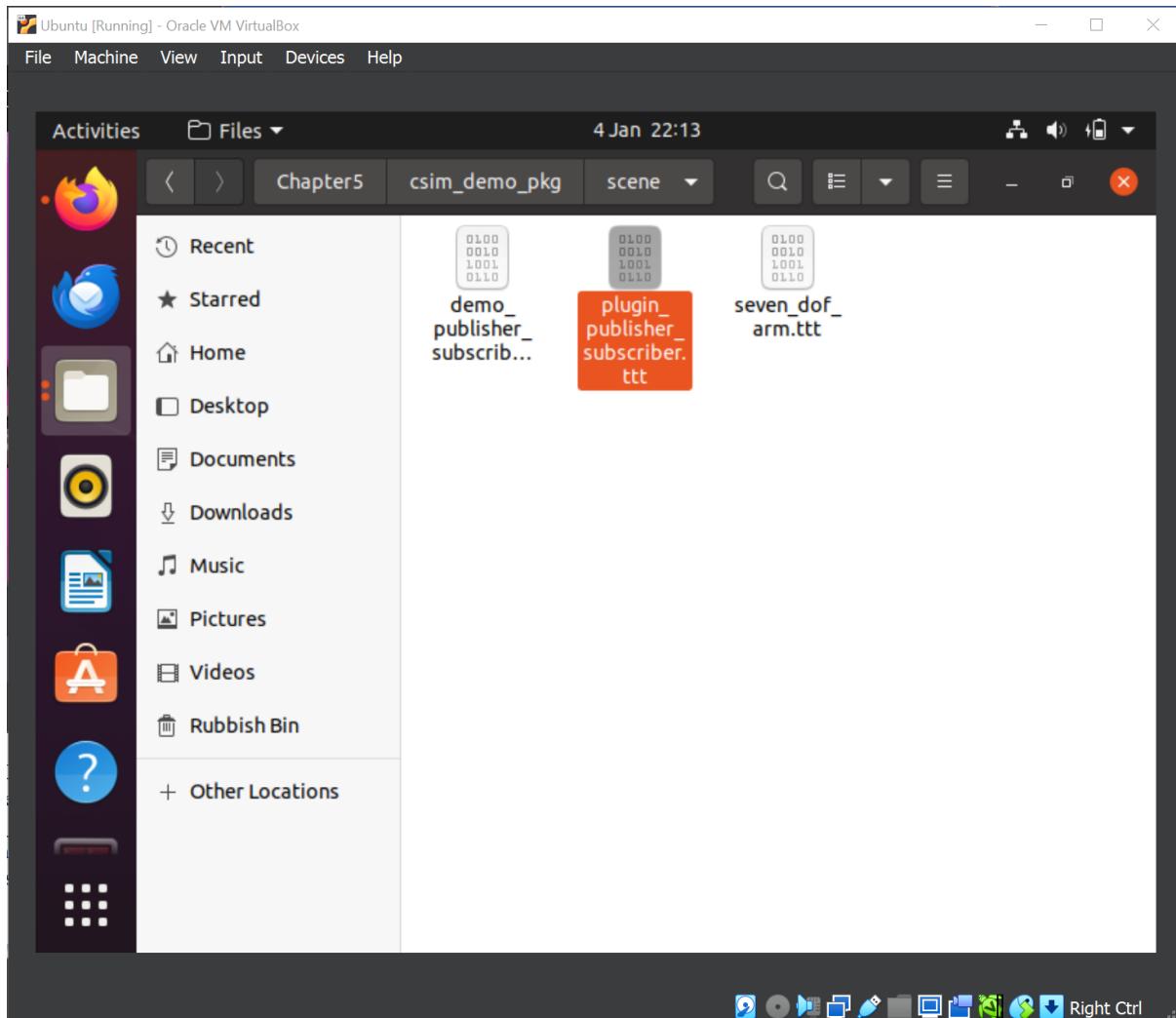


Working with ROS messages

Dalam proses simulasi robot menggunakan CoppeliaSim, penting untuk memahami cara berinteraksi dengan ROS (Robot Operating System). Dalam contoh penerbit dummy_publisher, kita mempublikasikan data integer ke topik ROS. Ini dilakukan dengan memahami struktur pesan yang akan dipublikasikan. Sebagai contoh, jika kita ingin mempublikasikan data integer, kita perlu membungkusnya dalam struktur pesan yang sesuai, sesuai dengan format yang diharapkan oleh ROS. Dalam hal ini, kita perlu memasukkan nilai ke dalam bidang "data" dari pesan.

Selanjutnya, kita melihat cara menyiaran gambar melalui ROS yang diambil oleh sensor kamera di adegan simulasi. Proses ini melibatkan pengambilan gambar dan propertinya menggunakan metode tertentu. Selanjutnya, gambar tersebut disusun dalam struktur data yang sesuai dengan format pesan sensor_msgs/Image yang diharapkan oleh ROS. Data ini kemudian dipublikasikan ke topik ROS terkait.

plugin_publisher_subscriber.ttt



Simulating a robotic arm using CoppeliaSim and ROS

Ketika kita ingin mensimulasikan robot lengan tujuh derajat kebebasan (DOF) menggunakan CoppeliaSim, langkah pertama adalah mengimpor model robot ke dalam adegan simulasi. Hal ini membutuhkan konversi model xacro ke dalam file URDF agar dapat diimporkan oleh CoppeliaSim. Setelah diimporkan, komponen robot seperti sendi dan link akan muncul dalam adegan. Namun, agar robot dapat dikendalikan, perlu mengaktifkan motor pada sendi-sendi tersebut. Pengaturan ini memastikan bahwa robot dapat bergerak selama simulasi dan mengikuti perintah kendali.

Setelah impor dan konfigurasi selesai, langkah berikutnya adalah menghubungkan robot dengan ROS menggunakan plugin RosInterface. Ini melibatkan pembuatan skrip Lua yang memungkinkan pertukaran informasi antara CoppeliaSim dan ROS melalui topik-topik khusus. Dengan mengkonfigurasi penerbit dan langganan pada setiap sendi robot, kita dapat mengontrol robot menggunakan perintah ROS. Sebaliknya, kita dapat memantau status sendi melalui topik ROS yang sesuai.

Melalui proses ini, kita berhasil mensimulasikan dan mengontrol robot lengan tujuh DOF menggunakan CoppeliaSim dan terhubung dengan ROS. Dengan ini, kita dapat terus mendalami simulasi robot dan mengeksplorasi penggunaan perangkat lunak simulasi robot lainnya seperti Webots.

Adding the ROS interface to CoppeliaSim joint controllers

Pada bagian ini, kita akan mempelajari cara mengintegrasikan lengan robot tujuh derajat kebebasan (DOF) dengan plugin RosInterface pada CoppeliaSim untuk mentransmisikan status sendi dan menerima masukan kendali melalui topik ROS. Seperti yang telah terlihat pada contoh sebelumnya, kita memilih suatu komponen robot (misalnya, komponen base_link_respondable) dan membuat skrip Lua untuk mengelola komunikasi antara CoppeliaSim dan ROS. Dalam blok inisialisasi, kita mendapatkan handler untuk semua sendi robot, seperti shoulder_pan, shoulder_pitch, elbow_roll, dan lainnya. Selanjutnya, kita menetapkan penerbit sudut sendi menggunakan simROS.advertise() untuk setiap sendi model. Hal ini juga diulangi untuk pelanggan perintah sendi yang mengatur callback untuk memproses data masukan yang diterima. Sebagai contoh, kita menggunakan fungsi setJointTargetPosition untuk mengubah posisi suatu sendi berdasarkan data yang diterima. Setelah simulasi dimulai, kita dapat memindahkan sendi tertentu, misalnya, elbow_pitch, dengan menerbitkan nilai menggunakan perintah baris perintah ROS, dan sekaligus memantau posisi sendi tersebut melalui topik terkait. Dengan ini, kita siap mengimplementasikan algoritma kontrol untuk menggerakkan sendi-sendi lengan robot tujuh DOF. Dengan topik ini, kita menyelesaikan bagian pertama dari bab ini dan melanjutkan untuk membahas perangkat lunak simulasi robot lainnya, yaitu Webots.

Setting up Webots with ROS

Pertama-tama, kita perlu menginstal Webots di sistem kita sebelum mengonfigurasikannya dengan ROS, mirip dengan langkah-langkah yang dilakukan pada CoppeliaSim. Webots adalah perangkat lunak simulasi multiplatform yang didukung oleh Windows, Linux, dan macOS. Pengembangan awal dilakukan oleh Swiss Federal Institute of Technology, Lausanne (EPFL), dan saat ini dikelola oleh Cyberbotics dengan lisensi Apache 2 yang gratis dan sumber terbuka. Setelah instalasi, Webots menyediakan lingkungan pengembangan lengkap untuk memodelkan, memprogram, dan mensimulasikan robot, sering digunakan dalam industri, pendidikan, dan penelitian.

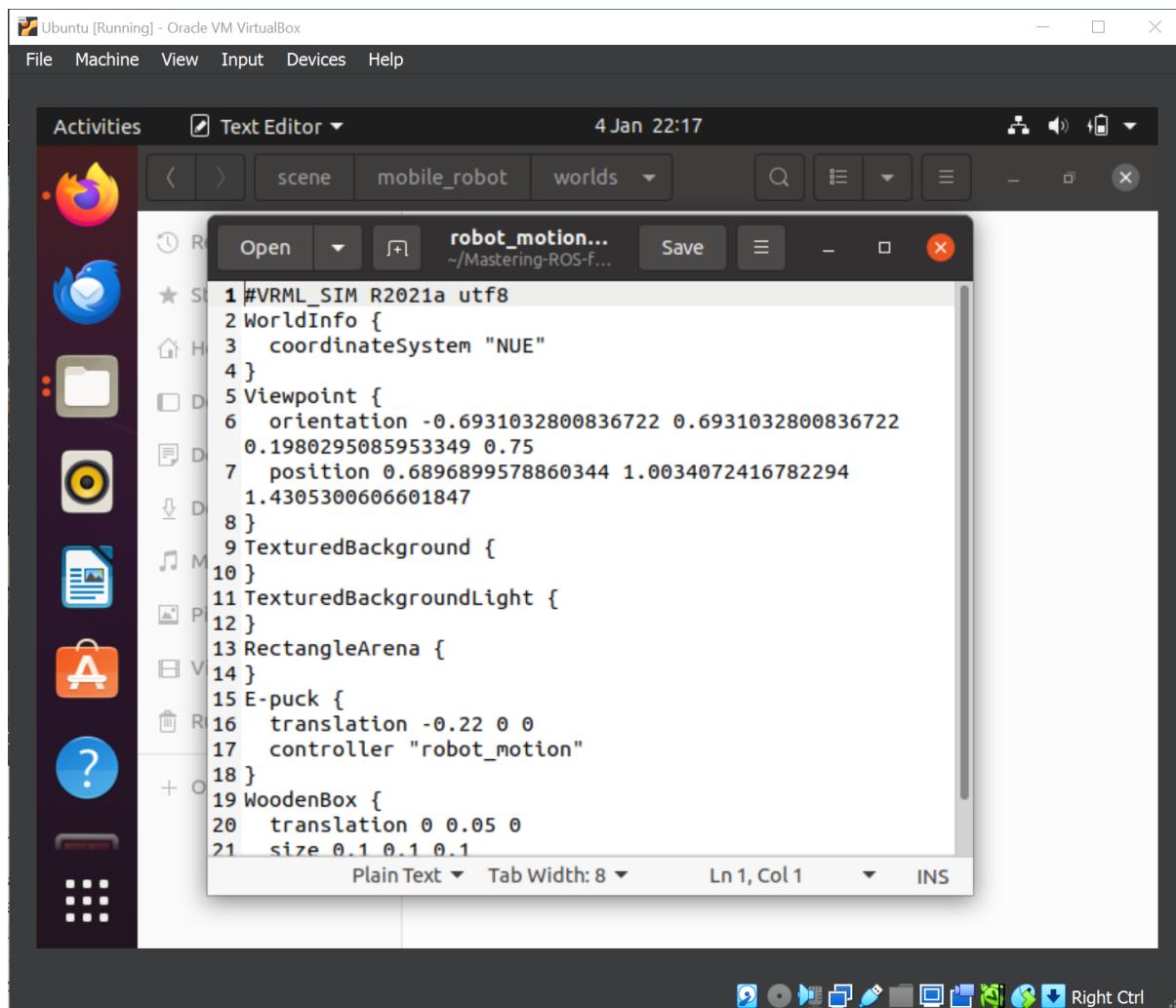
Introduction to the Webots simulator

Simulasi Webots terdiri dari tiga elemen utama: file konfigurasi dunia, pengontrol, dan plugin fisik. File konfigurasi dunia (.wbt) digunakan untuk mengatur lingkungan simulasi. Pengontrol dapat diimplementasikan dalam berbagai bahasa pemrograman seperti C, C++, Python, atau Java. Webots juga mendukung skrip MATLAB. Plugin fisik digunakan untuk memodifikasi perilaku fisik simulasi. Komunikasi antara ROS dan Webots dapat diimplementasikan menggunakan pengontrol yang bertindak sebagai node ROS, menyediakan fungsi Webots sebagai layanan atau topik ke node ROS lainnya.

Simulating a mobile robot with Webots

Tujuan dari bagian ini adalah membuat adegan simulasi dari awal yang berisi objek dan robot beroda. Langkah pertama adalah membuat dunia baru menggunakan opsi wizard. Dunia ini sudah tersedia dalam kode sumber buku di paket webots_demo_pkg. Setiap objek dalam adegan disusun secara hierarkis, seperti yang ditunjukkan pada panel pohon di UI. Sebuah dunia Webots terdiri dari file konfigurasi dunia, pengontrol, dan plugin fisik. Pengontrol dapat diimplementasikan dalam berbagai bahasa pemrograman dan digunakan untuk mengelola simulasi. Pada akhirnya, kita dapat memulai simulasi menggunakan tombol Start.

robot_motion_controller.wbt



```
#VRML_SIM R2021a utf8
WorldInfo {
  coordinateSystem "NUE"
}
Viewpoint {
  orientation -0.6931032800836722 0.6931032800836722
  0.1980295085953349 0.75
  position 0.6896899578860344 1.0034072416782294
  1.4305300606601847
}
TexturedBackground {
}
TexturedBackgroundLight {
}
RectangleArena {
}
E-puck {
  translation -0.22 0 0
  controller "robot_motion"
}
WoodenBox {
  translation 0 0.05 0
  size 0.1 0.1 0.1
}
```

Writing your first controller

Pada bagian ini, kita membuat kontroler pertama untuk robot beroda. Kita melihat bagaimana kontroler mengelola pergerakan robot dan reaksinya terhadap sensor. Mari ubah kontroler robot E-puck untuk bergerak ke arah tertentu. Kita dapat memilih bahasa pemrograman yang berbeda, tetapi kita akan menggunakan C++. Tujuan kontroler baru ini adalah mengendalikan kecepatan roda robot untuk menunjukkan struktur tipikal dari kontroler Webots.

Langkah pertama adalah mengubah kontroler yang terkait dengan robot beroda kita. Perlu dicatat bahwa setiap robot hanya dapat menggunakan satu kontroler pada satu waktu. Sebaliknya, kita dapat mengasosiasikan kontroler yang sama dengan robot yang berbeda. Langkah-langkah untuk menulis kontroler baru melibatkan:

1. Membuat file kontroler baru.
2. Menulis kontroler baru.
3. Mengompilasi kontroler baru.
4. Mengubah kontroler default robot dengan yang baru di panel properti robot.

Seperti yang telah dilakukan untuk pembuatan dunia, kita dapat menggunakan antarmuka wizard untuk menghasilkan kontroler baru. Kita memilih bahasa pemrograman C++ dan nama robot_motion. Sejumlah kode sumber awal akan muncul di editor teks, yang kemudian dapat dikompilasi dengan tombol Build.

Kode kontroler menggunakan Webots API untuk mengendalikan kecepatan roda robot. Dalam loop utama, kecepatan setiap motor diatur sebagai 10 persen dari kecepatan maksimum, dan arah gerak motor kanan ditetapkan sebagai 1.0 untuk maju lurus dan -1.0 untuk berputar. Waktu yang sudah berlalu dipertimbangkan untuk mengatur kecepatan kontrol, dan fungsi step dipanggil untuk mengirim perintah ke motor.

Setelah mengompilasi kontroler, kita dapat menambahkannya ke robot dalam panel hirarki. Simulasi dapat dimulai untuk melihat hasil dari kontroler Webots pertama. Pada bagian berikutnya, akan dijelaskan integrasi ROS dan Webots.

Simulating the robotic arm using Webots and ROS

Integrasi Webots-ROS melibatkan dua sisi: sisi ROS dan sisi Webots. Sisi ROS diimplementasikan melalui paket ROS webots_ros, sementara Webots mendukung ROS secara alami berkat kontroler standar yang dapat ditambahkan ke model robot apa pun. Untuk menggunakan Webots dengan ROS, Anda perlu menginstal paket webots_ros, yang dapat dilakukan menggunakan APT dengan perintah **sudo apt-get install ros-noetic-webots-ros**.

Setelah instalasi, kontroler yang sebelumnya dikembangkan harus diubah dengan kontroler bernama ros, seperti yang ditunjukkan dalam tangkapan layar. Setelah simulasi dimulai, kita dapat berinteraksi langsung dengan robot menggunakan sejumlah layanan yang mengimplementasikan fungsionalitas Webots di jaringan ROS sesuai konfigurasi sensor dan aktuator robot. Pastikan roscore aktif di jaringan ROS; jika tidak, akan muncul kesalahan di konsol Webots.

Webots hanya menerbitkan topik bernama /model_name, yang berisi daftar model yang aktif dalam adegan simulasi. Informasi ini penting untuk menggunakan layanan Webots karena Webots menggunakan sintaksis khusus untuk mendeklarasikan layanannya atau topiknya di jaringan.

Pada tahap ini, layanan baru diterbitkan di jaringan ROS, mewakili gambar yang diambil oleh kamera. Layanan /camera/enable digunakan untuk mengaktifkan kamera, dan data kamera dapat dilihat menggunakan plugin image_view.

Selain itu, kita dapat mengaktifkan dan membaca sensor lain, seperti sensor jarak, dan juga mengatur posisi, kecepatan, dan torsi sendi robot. Dalam kasus ini, kita ingin mengatur kecepatan roda. Integrasi dengan ROS membutuhkan pendekatan dari kedua sisi, dan di sisi ROS, kita dapat mengimplementasikan node baru menggunakan paket webots_ros.

Writing a teleop node using webots_ros

Dalam bagian ini, kita akan mengimplementasikan sebuah node ROS untuk mengendalikan langsung kecepatan roda robot E-Puck berdasarkan pesan geometry_msgs::Twist. Untuk

melakukannya, kita perlu menggunakan webots_ros sebagai dependensi. Langkah-langkahnya adalah sebagai berikut:

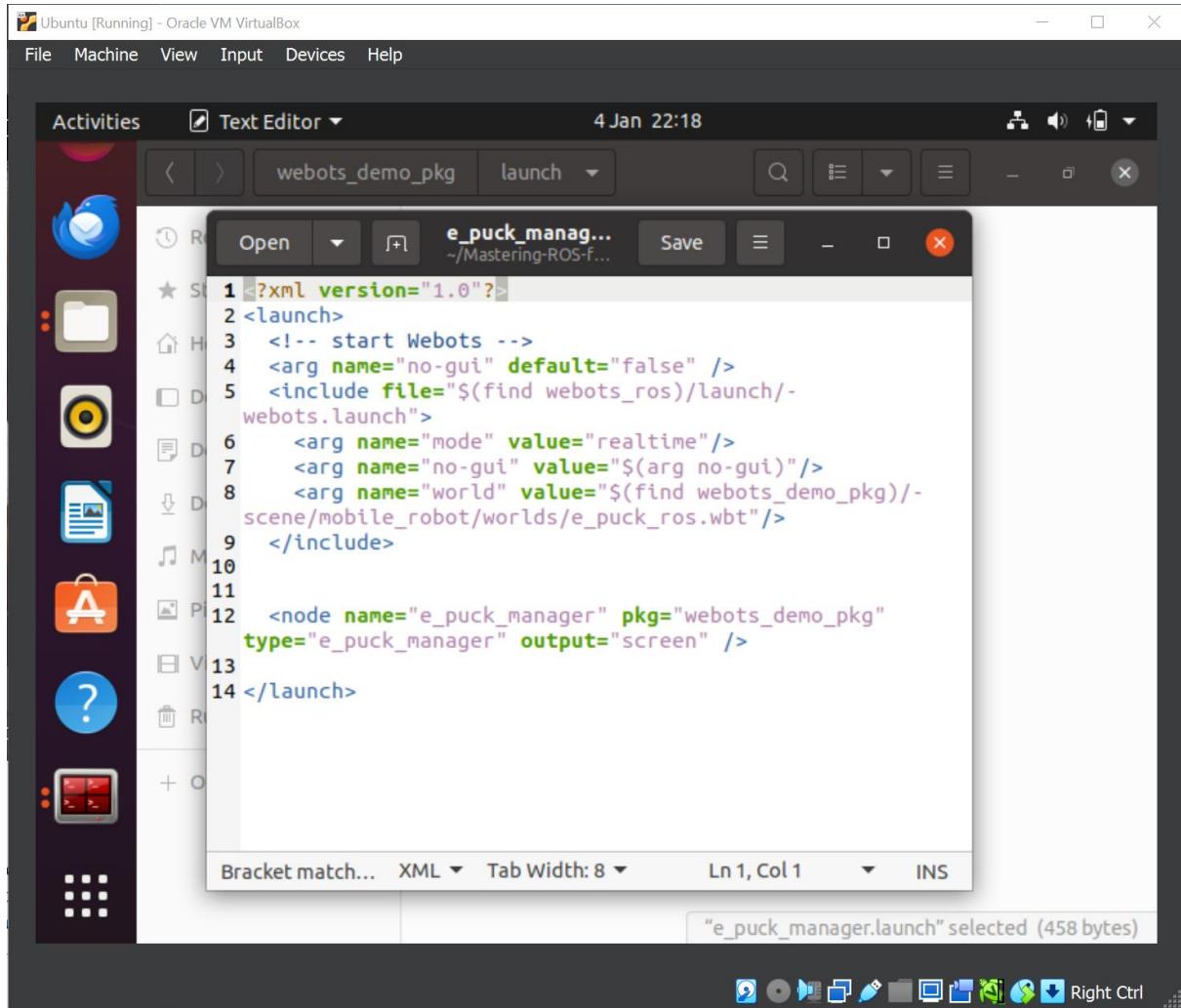
1. Membuat paket webots_demo_pkg dengan menyatakan webots_ros sebagai dependensinya menggunakan perintah **catkin_create_pkg webots_demo_pkg roscpp webots_ros geometry_msgs**.
2. Mendefinisikan beberapa file header yang diperlukan untuk mengimplementasikan pesan yang dibutuhkan untuk menggunakan layanan Webots.
3. Membuat variabel untuk menyimpan data yang diterima oleh panggilan kembali ROS, yaitu informasi tentang mode robot dan kecepatan yang akan diterapkan.
4. Mengimplementasikan dua panggilan kembali (callbacks) dalam node ini, salah satunya untuk membaca kecepatan linier dan angular yang diinginkan dan menetapkan kecepatan roda, dan yang lainnya untuk membaca nama model yang ditugaskan ke robot E-Puck.
5. Mengimplementasikan fungsi utama di mana semua yang diperlukan untuk mengendalikan robot diatur. Langkah-langkah meliputi inisialisasi node ROS, menunggu model robot disiarkan oleh Webots, dan mendefinisikan pelanggan (subscriber) untuk topik /cmd_vel.
6. Menggunakan layanan ROS untuk mengatur posisi roda menjadi INFINITY dan mengatur kecepatan menjadi 0.0.
7. Dalam loop utama, mengaplikasikan kecepatan yang dihitung dalam panggilan geometry_msgs::Twist ke roda kiri dan kanan menggunakan layanan ROS.

Dengan menggunakan node keyboard teleop dari paket diff_wheeled_robot_control, kita dapat mengendalikan robot mobile di Webots menggunakan ROS. Simulasi dapat dimulai dan node dapat diluncurkan menggunakan beberapa perintah yang disebutkan di akhir bagian.

Starting Webots with a launch file

Pada bagian terakhir ini, kita akan melihat bagaimana memulai Webots langsung menggunakan berkas *launch*. Hal ini dapat dilakukan berkat berkas *launch* yang sudah disediakan dalam paket `webots_ros`. Untuk memulai dunia Webots yang diinginkan, kita perlu menyertakan berkas *launch* ini dan mengatur berkas `.wbt` untuk dimulai, seperti yang ditunjukkan dalam berkas `e_puck_manager.launch` di dalam direktori `webots_demo_package/launch`

`e_puck_manager.launch`



```
1 <?xml version="1.0"?>
2 <launch>
3   <!-- start Webots -->
4   <arg name="no-gui" default="false" />
5   <include file="$(find webots_ros)/launch/->
      webots.launch">
6     <arg name="mode" value="realtime"/>
7     <arg name="no-gui" value="$(arg no-gui)"/>
8     <arg name="world" value="$(find webots_demo_pkg)/->
        scene/mobile_robot/worlds/e_puck_ros.wbt"/>
9   </include>
10
11
12   <node name="e_puck_manager" pkg="webots_demo_pkg"
13     type="e_puck_manager" output="screen" />
14 </launch>
```

The screenshot shows a Linux desktop environment with a text editor window open. The window title is "e_puck_manager.launch". The code in the editor is an XML launch file for Webots. It includes arguments for mode, no-gui, and world, and a node for the e_puck_manager. The code is color-coded for syntax highlighting.