TECHNICAL REPORT MASTERING ROS2 BOOK WITH WEBOT

Faisal Anwar

1103204096

TK44G7

Abstrak:

Pada laporan ini saya akan memaparkan penjelasan terkait praktek yang saya lakukan mengikuti buku Mastering ROS, laporan ini berisi screenshoots hasil praktek, dan sedikit penjelasan terkait proses yang saya lakukan dalam melkaukan running. Berikut merupakan runtutan pengerjaan yang akan saya paparkan dibawah ini

ROS

ROS adalah singkatan dari "Robot Operating System." Meskipun namanya mungkin menyarankan bahwa itu adalah sistem operasi yang mirip dengan sistem operasi komputer tradisional seperti Windows atau Linux, ROS sebenarnya adalah sebuah kerangka kerja (framework) yang digunakan untuk pengembangan perangkat lunak robotik. ROS menyediakan berbagai library, alat, dan konvensi yang memudahkan pengembangan aplikasi robotik.

ROS dirancang untuk memudahkan pengembangan, pengujian, dan integrasi perangkat lunak robotik. Dengan menggunakan ROS, pengembang dapat fokus pada pemrograman fungsionalitas spesifik robot tanpa harus khawatir tentang detail implementasi komunikasi antar modul, sensor, atau aktuator.

Salah satu kelebihan besar dari ROS adalah komunitas yang aktif dan luas, yang telah mengembangkan berbagai pustaka dan alat yang dapat digunakan oleh pengembang untuk mempercepat proses pengembangan robotik. ROS mendukung berbagai bahasa pemrograman seperti C++, Python, dan lainnya, sehingga memungkinkan fleksibilitas dalam pengembangan.

Secara keseluruhan, ROS telah menjadi standar de facto dalam pengembangan perangkat lunak robotik di banyak aplikasi, dari robot penelitian di laboratorium akademik hingga robot industri di lingkungan produksi.

MASTERING ROS BOOK

"Mastering ROS" adalah judul buku yang berfokus pada pemahaman dan penguasaan Robot Operating System (ROS). Buku ini ditujukan untuk membantu pembaca memahami berbagai aspek ROS dari dasar hingga lanjutan. Berikut adalah rangkuman singkat terkait dengan buku "Mastering ROS":

Pendahuluan ke ROS: Buku ini biasanya dimulai dengan pendahuluan mengenai apa itu ROS, sejarahnya, dan mengapa menjadi penting dalam pengembangan robotik.

Pengenalan Konsep Dasar: Pembaca diperkenalkan dengan konsep dasar seperti node, topik (topics), layanan (services), dan tindakan (actions) dalam ROS.

Pengembangan Aplikasi ROS: Buku ini memberikan panduan langkah demi langkah tentang bagaimana mengembangkan aplikasi ROS dari awal. Ini mencakup pembuatan node, komunikasi antar node, dan integrasi dengan sensor dan aktuator.

Navigasi dan Pemetaan: Salah satu topik penting dalam robotik adalah navigasi dan pemetaan lingkungan. Buku ini mungkin mencakup bagaimana menggunakan ROS untuk pemetaan dan navigasi robot di lingkungan yang tidak dikenal.

Pengembangan Algoritma Lanjutan: Selain konsep dasar, buku ini mungkin juga mencakup topik-topik lanjutan seperti pengolahan citra, pengenalan objek, kontrol gerakan, dan lainnya menggunakan ROS.

Best Practices dan Tips: "Mastering ROS" juga mungkin memberikan best practices, tips, dan trik untuk mengembangkan aplikasi ROS yang efisien dan andal.

Studi Kasus dan Proyek: Buku ini mungkin juga mencakup studi kasus dan proyek nyata yang menggunakan ROS, memberikan pembaca pemahaman praktis tentang bagaimana ROS digunakan dalam konteks nyata.

Secara keseluruhan, "Mastering ROS" dirancang untuk menjadi sumber yang komprehensif bagi siapa saja yang ingin memahami dan menguasai ROS dalam pengembangan robotik. Dengan mengeksplorasi berbagai konsep, alat, dan teknik, pembaca dapat memperdalam pengetahuannya dan mengembangkan aplikasi yang kompleks dengan menggunakan ROS.

PRAKTEK BAB 2

Getting Started with ROS Programming

Setelah kita mengenal apa itu ROS, server parameter, dan roscore, kita sekarang dapat mulai membuat dan membangun paket ROS. Dalam bab ini, kita akan membuat beberapa ROS node yang berbeda dengan menerapkan system komunikasi ROS.

Untuk menjalankan ROS nodes, pertama tama kita harus membuat workspace dengan cara;

```
mkdir -p ~/catkin_ws/src
```

Setelah membuat workspacec, kita harus melakukan compiling workspace, dengan cara

```
source /opt/ros/noetic/setup.bash
```

Kemudian kita perlu melakukan initialize a new catkin workspace:

```
catkin_init_workspace
```

dan lakukan catkin_make untuk build direktori

```
catkin_make
```

Setelah melakukan beberapa proses diatas, sekarang kita akan membuat catkin package, dengan cara;

```
catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs

actionlib actionlib_msgs
```

gambar seperti dibawah akan muncul ketika anda berhasil melakakukan create pkg

```
Created file mastering_ros_v2_pkg/package.xml
Created file mastering_ros_v2_pkg/CMakeLists.txt
Created folder mastering_ros_v2_pkg/include/mastering_ros_v2_pkg
Created folder mastering_ros_v2_pkg/src
Successfully created files in /home/jcacace/mastering_ros_v2_pkg. Pleas
e adjust the values in package.xml.
```

dan lakukan catkin_make

```
[  0%] Built target _webots_ros_generate_messages_check_deps_field_set_float
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target _webots_ros_generate_messages_check_deps_field_set_int32
[  0%] Built target sensor_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target sensor_msgs_generate_messages_lisp
[  0%] Built target mastering_ros_robot_description_pkg_xacro_generated_to_devel_space_
[  0%] Built target _webots_ros_generate_messages_check_deps_field_set_vec3f
[  0%] Built target _webots_ros_generate_messages_check_deps_node_get_field
[  0%] Built target _webots_ros_generate_messages_check_deps_get_float
[  1%] Built target e_puck_manager
[ 64%] Built target webots_ros_generate_messages_cpp
[ 76%] Built target webots_ros_generate_messages_py
[ 76%] Built target webots_ros_generate_messages_eus
[ 79%] Built target webots_ros_generate_messages_lisp
[ 98%] Built target webots_ros_generate_messages_nodejs
[ 98%] Built target webots_ros_generate_messages
[ 98%] Built target keyboard_teleop
[ 98%] Built target catch_the_bird
[ 98%] Built target e_puck_line
[100%] Built target panoramic_view_recorder
[100%] Built target robot_information_parser
[100%] Built target complete_test
root@024a5ddf0613:~/catkin_ws#
```

Creating ROS nodes

Simpul pertama yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini akan Menerbitkan nilai bilangan bulat pada topik yang disebut /numbers. Salin kode saat ini ke kode baru atau gunakan file yang ada ini dari repositori kode buku ini.

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>

int main(int argc, char **argv) {
        ros::init(argc, argv,"demo_topic_publisher");
        ros::NodeHandle node_obj;
        ros::Publisher number_publisher = node_obj.advertise<std_msgs::Int32>("/numbers", 10);
        ros::Rate loop_rate(10);
        int number_count = 0;
        while ( ros::ok() ) {
                std_msgs::Int32 msg;
                msg.data = number_count;
                ROS_INFO("%d",msg.data);
                number_publisher.publish(msg);
                loop_rate.sleep();
                ++number_count;
        }
        return 0;
}
~
~
~
```

Sekarang setelah kita membahas node penerbit, kita dapat membahas node pelanggan, yaitu demo_topic_subscriber.cpp. Berikut adalah definisi dari node pelanggan:

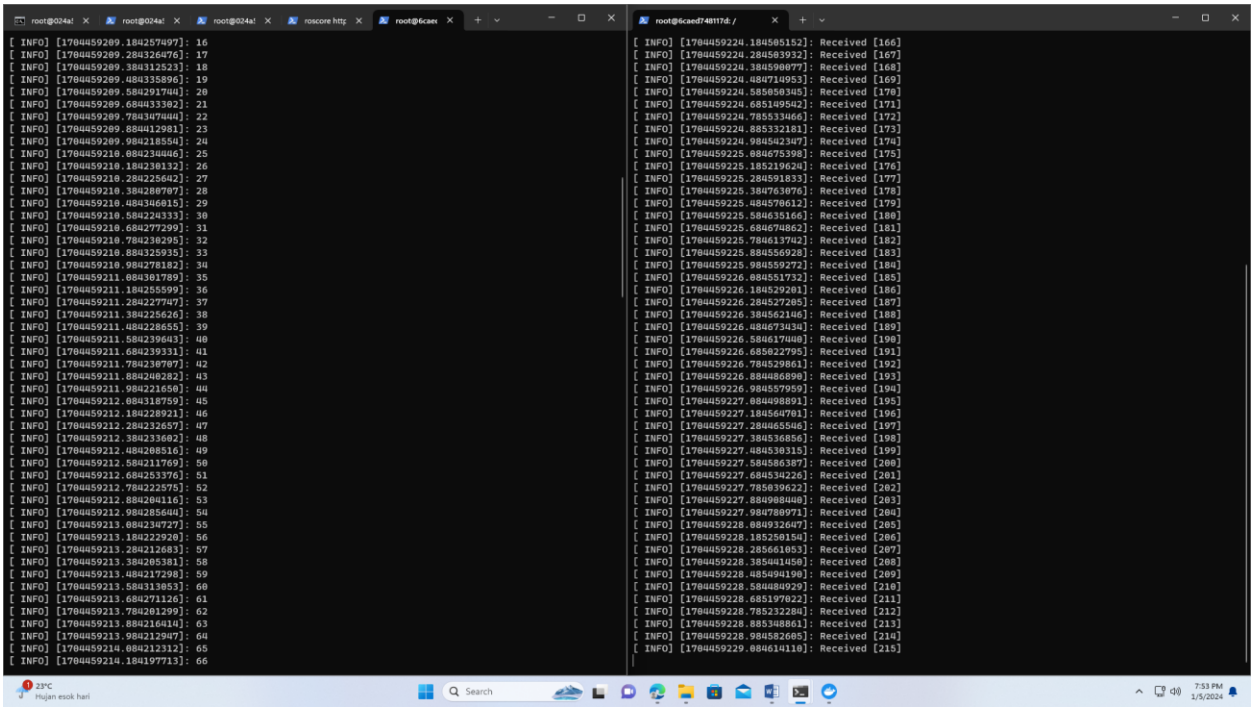
```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>

void number_callback(const std_msgs::Int32::ConstPtr& msg) {
        ROS_INFO("Received [%d]",msg->data);
}

int main(int argc, char **argv) {
        ros::init(argc, argv,"demo_topic_subscriber");
        ros::NodeHandle node_obj;
        ros::Subscriber number_subscriber = node_obj.subscribe("/numbers",10,number_callback);
        ros::spin();
        return 0;
}
~
~
~
~
~
~
~
~
```

Kita harus mengedit file CMakeLists.txt dalam paket untuk mengkompilasi dan membangun sumber kode. Buka mastering_ros_demo_pkg untuk melihat CMakeLists yang ada.txt arsip. Cuplikan kode berikut dalam file ini bertanggung jawab untuk membangun dua node ini:

```
root@6caed748117d:~/catkin_ws/src/mastering_ros_demo_pkg# ls
CMakeLists.txt  action  include  msg  package.xml  src  srv
root@6caed748117d:~/catkin_ws/src/mastering_ros_demo_pkg#
```

```
  ${Boost_INCLUDE_DIRS}
)
#This will create executables of the nodes
add_executable(demo_topic_publisher src/demo_topic_publisher.cpp)
add_executable(demo_topic_subscriber src/demo_topic_subscriber.cpp)
#This will link executables to the appropriate libraries
add_executable(demo_msg_publisher src/demo_msg_publisher.cpp)
add_executable(demo_msg_subscriber src/demo_msg_subscriber.cpp)

add_dependencies(demo_msg_publisher mastering_ros_demo_pkg_generate_messages_cpp)
add_dependencies(demo_msg_subscriber mastering_ros_demo_pkg_generate_messages_cpp)
target_link_libraries(demo_msg_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_msg_subscriber ${catkin_LIBRARIES})
target_link_libraries(demo_topic_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_topic_subscriber ${catkin_LIBRARIES})

add_executable(demo_service_server src/demo_service_server.cpp)
add_executable(demo_service_client src/demo_service_client.cpp)
add_dependencies(demo_service_server mastering_ros_demo_pkg_generate_messages_cpp)
add_dependencies(demo_service_client mastering_ros_demo_pkg_generate_messages_cpp)
```

Kemudian kita bisa melakukan running, berikut merupakan output yang ditampilkan;



Adding custom .msg and .srv files

Pertamatama kita dapat membuat folder msg, dan membuat file demo_msg.msg

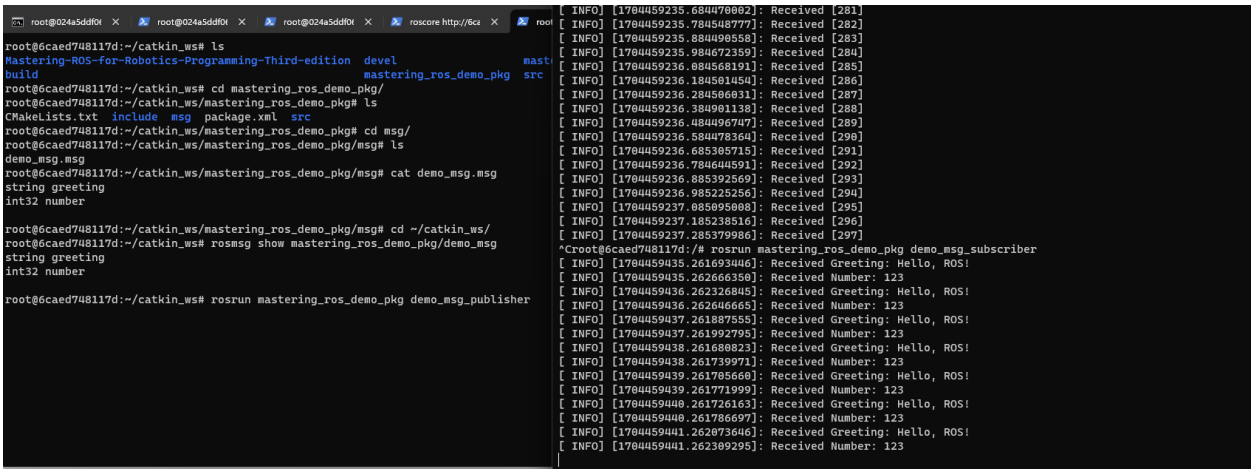
```
root@6caed748117d:~/catkin_ws/mastering_ros_demo_pkg# cd msg/
root@6caed748117d:~/catkin_ws/mastering_ros_demo_pkg/msg# ls
demo_msg.msg
root@6caed748117d:~/catkin_ws/mastering_ros_demo_pkg/msg#
```

```
root@6caed748117d:~/catkin_ws/mastering_ros_demo_pkg/msg# cat demo_msg.msg
string greeting
int32 number

root@6caed748117d:~/catkin_ws/mastering_ros_demo_pkg/msg#
```

Untuk memeriksa apakah pesan telah dibangun dengan benar, kita dapat menggunakan perintah rosmsg:

```
root@6caed748117d:~/catkin_ws# rosmsg show mastering_ros_demo_pkg/demo_msg
string greeting
int32 number

root@6caed748117d:~/catkin_ws#
```

Setelah melakukan proses diatas kita dapat melakukan running sesuai perntah buku, berikut merupakan output programmnya



Working with ROS services

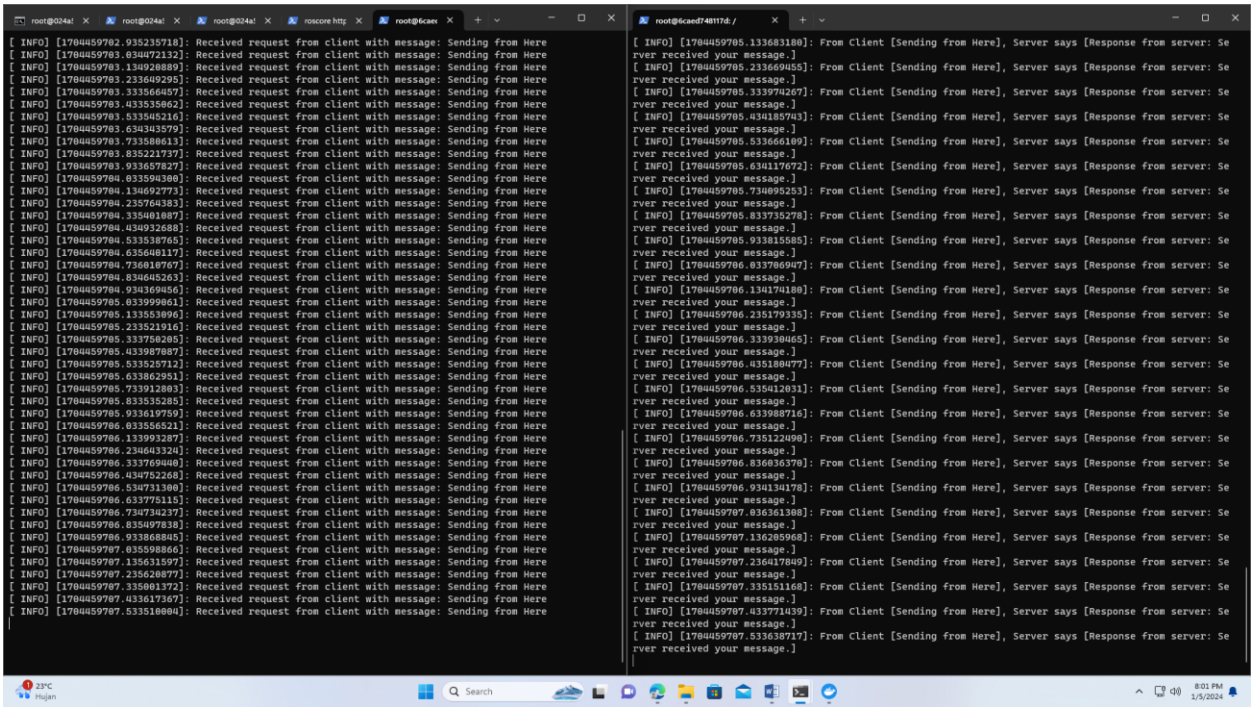
Pada bagian ini, kita akan membuat node ROS, yang dapat menggunakan definisi layanan yang Kami sudah mendefinisikannya. Node layanan yang akan kita buat dapat mengirim pesan string sebagai permintaan ke server; Kemudian, node server akan mengirim pesan lain sebagai respons. Navigasikan ke mastering_ros_demo_pkg/src dan temukan demo_service_server.cpp dan demo_service_client.cpp node. demo_service_server.cpp adalah server, dan definisinya adalah sebagai berikut:

```cpp
#include "ros/ros.h"
#include <iostream>
#include "mastering_ros_demo_pkg/demo_srv.h"
#include <sstream>

int main(int argc, char **argv) {
    ros::init(argc, argv, "demo_service_client");
    ros::NodeHandle n;
    ros::Rate loop_rate(10);
    ros::ServiceClient client = n.serviceClient<mastering_ros_demo_pkg::demo_srv>("demo_service"); // Perhatikan pengg
aan namespace disini

    std::stringstream ss; // Deklarasi stringstream
    while (ros::ok()) {
        mastering_ros_demo_pkg::demo_srv srv;
        ss << "Sending from Here";
        srv.request.in = ss.str();

        if (client.call(srv)) {
            ROS_INFO("From Client [%s], Server says [%s]", srv.request.in.c_str(), srv.response.out.c_str());
        } else {
            ROS_ERROR("Failed to call service");
            return 1;
        }

        ros::spinOnce();
        loop_rate.sleep();
        ss.str("");  // Bersihkan stringstream setelah penggunaan
    }
"demo_service_client.cpp" 32L, 980C                                                    1,1           To
```

Selanjutnya, mari kita lihat cara kerja demo_service_client.cpp. Berikut adalah definisi dari ini kode:

```
#include "ros/ros.h"
#include "mastering_ros_demo_pkg/demo_srv.h"

bool demo_service_callback(mastering_ros_demo_pkg::demo_srv::Request &req,
                           mastering_ros_demo_pkg::demo_srv::Response &res)
{
    ROS_INFO("Received request from client with message: %s", req.in.c_str());

    // Di sini Anda bisa menangani permintaan dan menyiapkan respons
    res.out = "Response from server: Server received your message.";

    return true; // Mengembalikan true jika layanan berhasil diproses
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "demo_service_server");
    ros::NodeHandle n;

    // Membuat server layanan dengan nama "demo_service"
    ros::ServiceServer service = n.advertiseService("demo_service", demo_service_callback);

    ROS_INFO("Ready to receive client requests.");

    ros::spin(); // Tetap berjalan dan menunggu permintaan layanan

    return 0;
}

"demo_service_server.cpp" 29L, 911C                                    1,1           All
```

Setelah melakukan beberapa configurasi, selanjutnya kita lakukan running pada program yang baru kita buat diatas, berikut merupakan output programnya

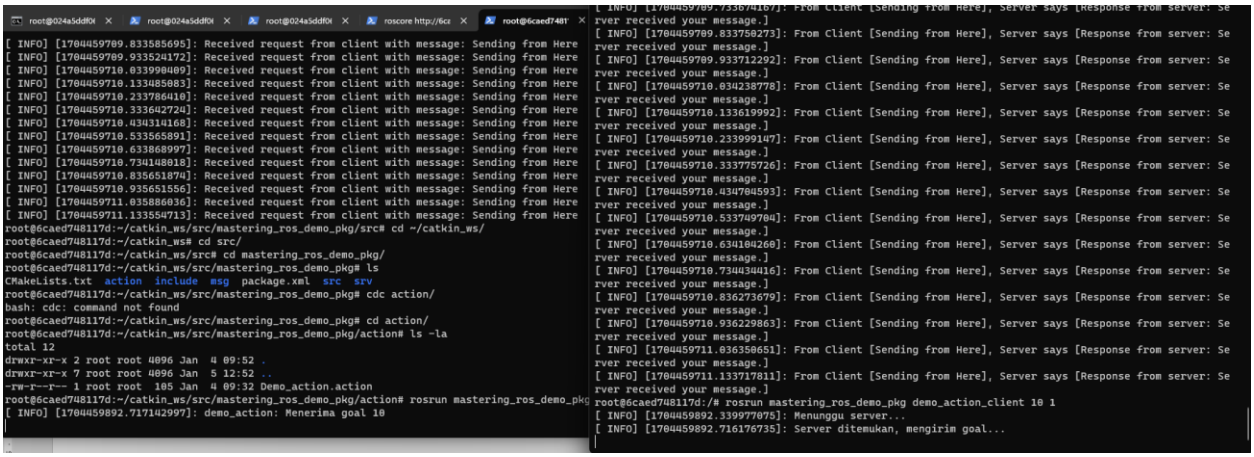


Working with ROS actionlib

Sekarang, mari kita bahas server aksi demo dan klien aksi. Klien aksi demo akan mengirim angka sebagai tujuan. Ketika server tindakan menerima tujuan ini, itu akan dihitung dari 0 hingga nomor gol dengan ukuran langkah 1 dan dengan penundaan 1 detik. Jika selesai sebelum Mengingat waktu, itu akan mengirimkan hasilnya; Jika tidak, tugas akan didahului oleh klien. Si Umpan balik di sini adalah kemajuan penghitungan. File tindakan dari tugas ini adalah sebagai berikut dan adalah disebut Demo_action.action:

```
root@6caed748117d:~/catkin_ws/src/mastering_ros_demo_pkg/action# ls -la
total 12
drwxr-xr-x 2 root root 4096 Jan  4 09:52 .
drwxr-xr-x 7 root root 4096 Jan  5 12:52 ..
-rw-r--r-- 1 root root  105 Jan  4 09:32 Demo_action.action
root@6caed748117d:~/catkin_ws/src/mastering_ros_demo_pkg/action#
```

Setelah melakuakan beberapa configurasi yang diperlikan berikut merupakan hasil running dari beberapa program yang baru kita buat sebelumnya



Creating launch files

File peluncuran di ROS sangat berguna untuk meluncurkan lebih dari satu node. Dalam contoh sebelumnya, kami melihat maksimal dua node ROS, tetapi bayangkan sebuah skenario di yang kita harus meluncurkan 10 atau 20 node untuk robot. Akan sulit jika kita harus melakukannya Jalankan setiap node di terminal satu per satu. Sebagai gantinya, kita dapat menulis semua node di dalam File berbasis XML yang disebut file peluncuran dan, menggunakan perintah yang disebut roslaunch, kami mengurai ini dan luncurkan node.

PRAKTEK BAB 3

Working with ROS for 3D Modeling

ROS menyediakan beberapa paket untuk merancang dan membuat model robot, seperti urdf, kdl_parser, robot_state_publisher, dan collada_urdf. Paket-paket ini akan membantukami membuat deskripsi model robot 3D dengan karakteristik yang tepat dari yang sebenarnya perangkat keras.

Untuk mengikuti contoh dalam bab ini, Anda memerlukan laptop standar yang berjalan Ubuntu 20.04 dengan ROS Noetic diinstal. Kode referensi untuk bab ini dapat berupa: diunduh dari repositori Git di https://github.com/PacktPublishing/ Mastering-ROS-untuk-Robotika-Pemrograman-Ketiga-edition.git. Itu kode terdapat di dalam Bab3/mastering_ros_robot_description_ pkg/ folder. Anda dapat melihat kode bab ini beraksi di sini: https://bit.ly/2W5jief.

Creating the ROS package for the robot description

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

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

Creating our first URDF model

Berikut merupakan beberapa robot model yang langsung saya gitclone dari github pemilik vidio

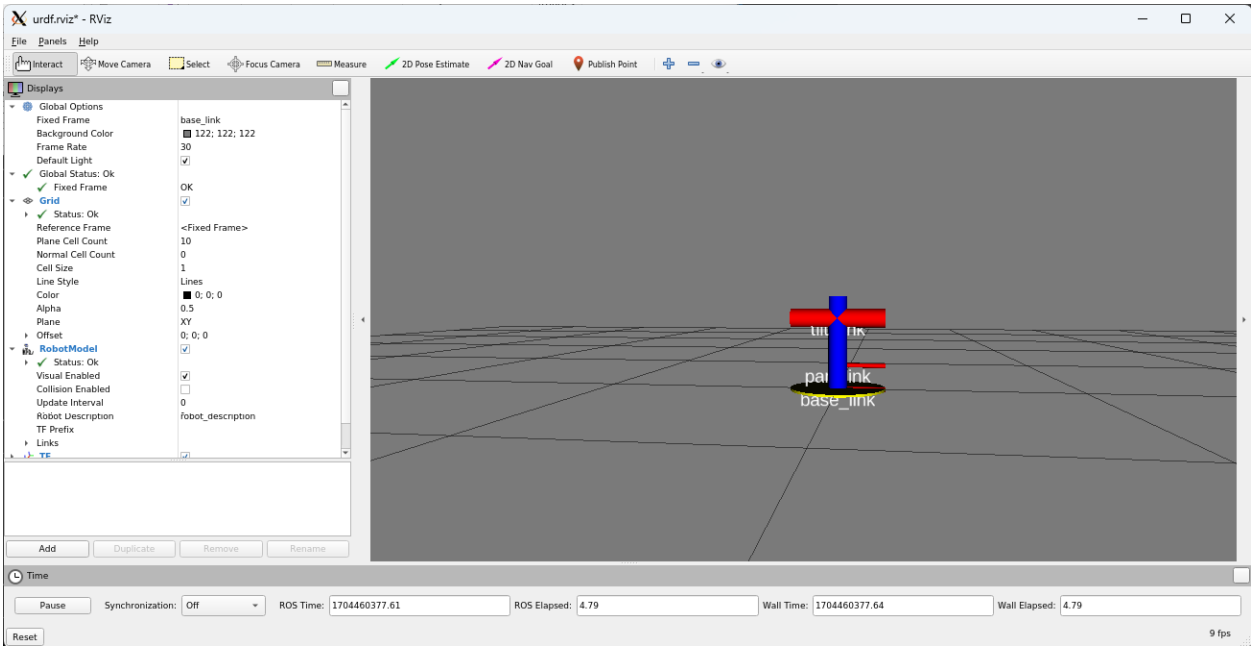
```
root@024a5ddf0613:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf# ls
diff_wheeled_robot.urdf              pan_tilt.pdf             sensors                    seven_dof_arm_with_rgbd.xacro
diff_wheeled_robot.xacro             pan_tilt.urdf            seven_dof_arm.urdf         wheel.urdf.xacro
diff_wheeled_robot_with_laser.xacro  pan_tilt.xacro           seven_dof_arm.xacro
pan_tilt.gv                          pan_tilt_generated.urdf  seven_dof_arm_moveit.urdf
root@024a5ddf0613:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf#
```

Jika kita ingin melihat struktur tautan dan sambungan robot secara grafis, kita dapat menggunakan Alat perintah yang disebut urdf_to_graphiz:

```
root@024a5ddf0613:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf# urdf_to_graphiz pan_tilt.urdf
Created file pan_tilt.gv
Created file pan_tilt.pdf
root@024a5ddf0613:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf#
```

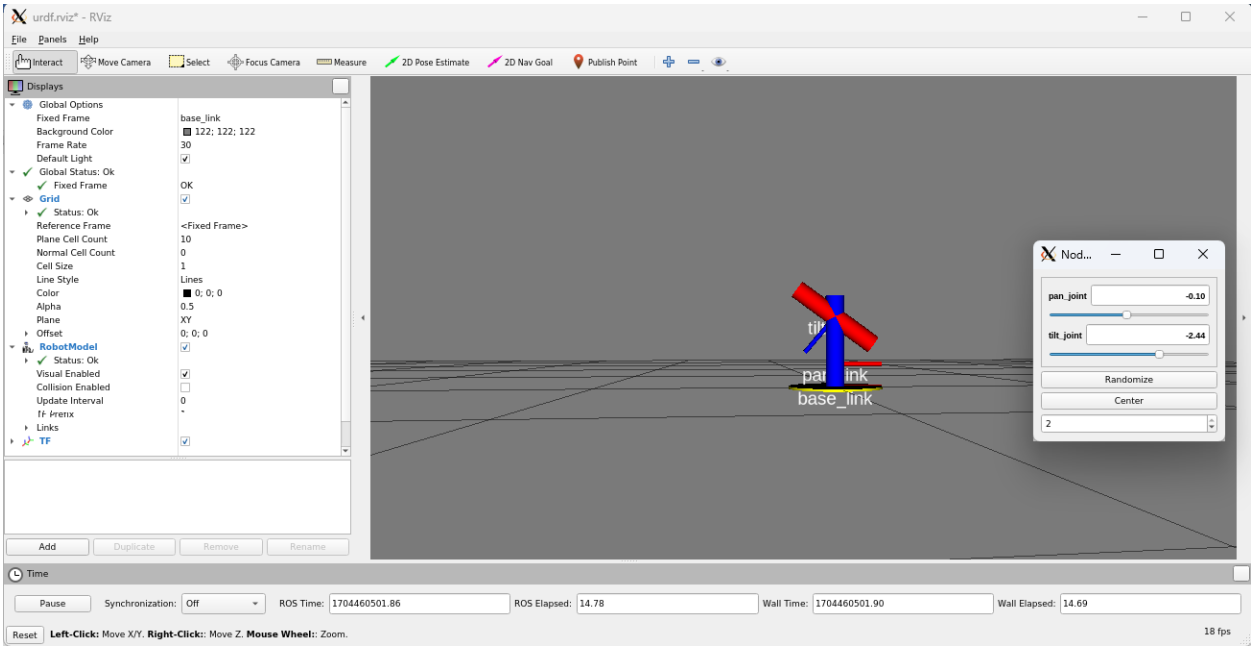
Visualizing the 3D robot model in RViz

Setelah menyesuaikan semua configurasi yang ada dibuku, jikasemuanya berjalan dengan baik maka akan muhncul tampilan seperti dibawah ini.



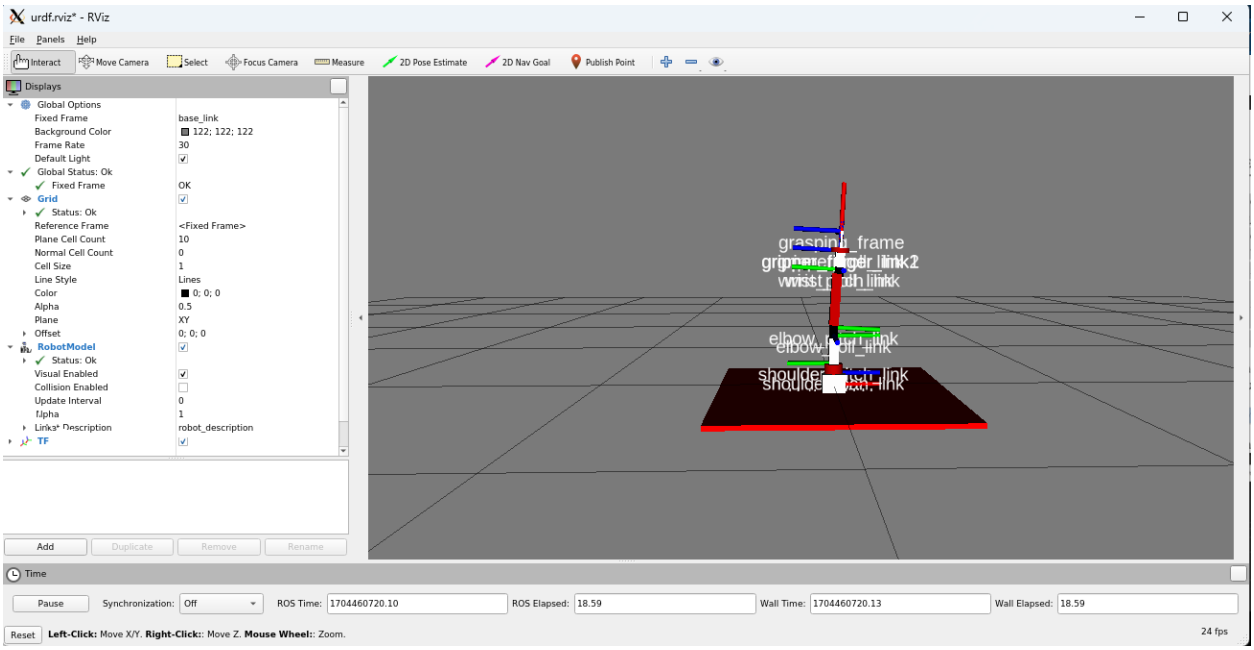
Interacting with pan-and-tilt joints

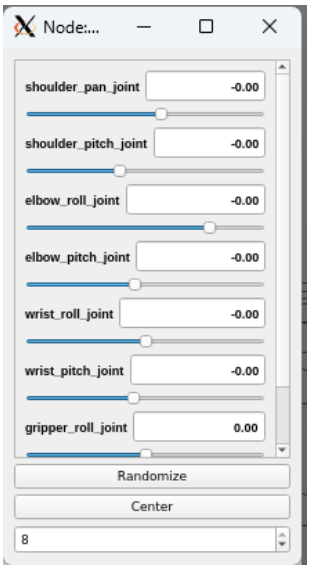
Kita juga dapat berintraksi dengan tilt joints, dengan GUI yang tersedia, berikut merupakan tampilannya



Creating the robot description for a seven-DOF robot manipulator

Sekarang, kita dapat membuat beberapa robot kompleks menggunakan URDF dan xacro. Robot pertama kami akan berurusan dengan adalah lengan robot tujuh-DOF, yang merupakan manipulator tautan serial dengan Beberapa tautan serial. Lengan tujuh-DOF secara kinematis berlebihan, yang berarti memiliki lebih banyak sendi dan DOF daripada yang dibutuhkan untuk mencapai posisi dan orientasi tujuannya. Si Keuntungan dari manipulator berlebihan adalah kita dapat memiliki lebih banyak konfigurasi bersama untuk posisi dan orientasi tujuan yang diinginkan. Ini akan meningkatkan flexibil dan keserbagunaan gerakan robot dan dapat menerapkan gerakan bebas tabrakan yang efektif dalam robot Workspace.





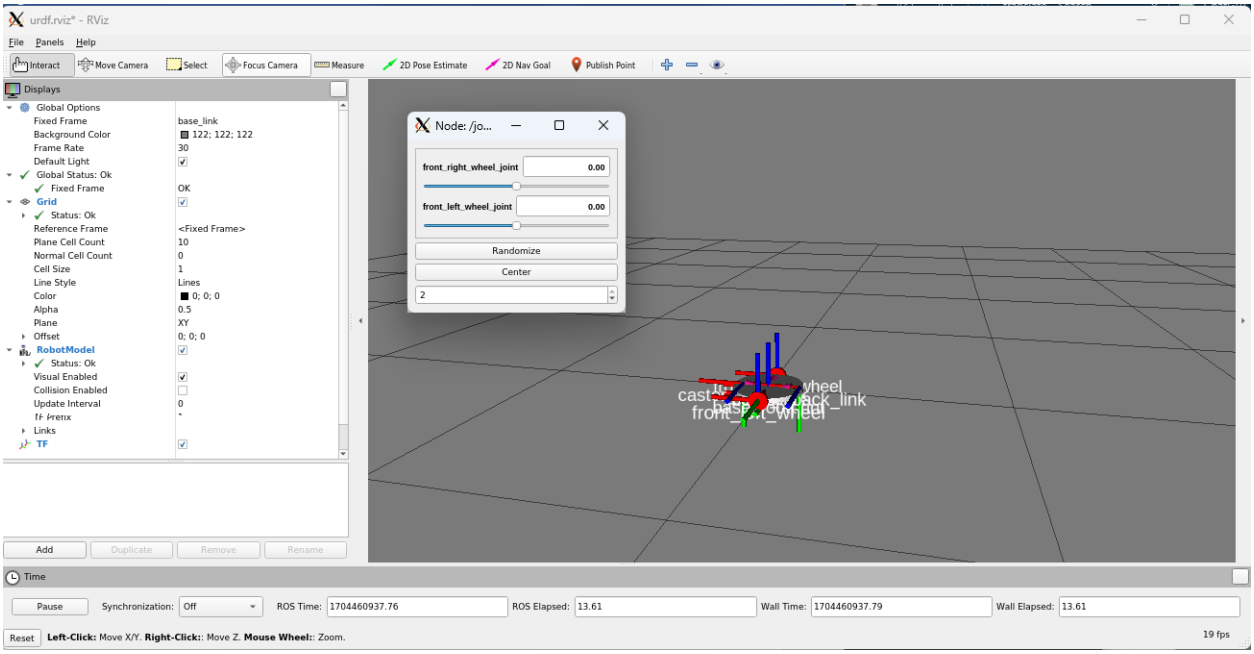
Gambar diatas adalah hasil running dari robot seven Doff arm, kita juga mendapatkan GUI untuk menggerakan robot tersebut.

Paket penerbit negara bersama adalah salah satu paket ROS yang umum digunakan untuk berinteraksi dengan setiap sendi robot. Paket berisi joint_state_ node penerbit, yang menemukan sambungan tidak tetap dari model URDF dan menerbitkan nilai status gabungan dari setiap sambungan dalam format pesan sensor_msgs/JointState. Paket ini juga dapat digunakan bersamaan dengan robot_state_publisher paket untuk mempublikasikan posisi semua sendi. Sumber yang berbeda dapat digunakan untuk mengatur nilai setiap sambungan. Seperti yang telah kita lihat, salah satu caranya adalah dengan menggunakan slider GUI. Cara ini adalah terutama digunakan untuk pengujian. Jika tidak, topik JointState yang node berlangganan dapat digunakan.

Creating a robot model for the differential drive mobile robot

Robot beroda diferensial akan memiliki dua roda yang terhubung ke sisi berlawanan dari robot sasis, yang didukung oleh satu atau dua roda kastor. Roda akan mengontrol kecepatan robot dengan mengatur kecepatan roda tunggal. Jika kedua motor berjalan Pada kecepatan yang sama, roda akan bergerak maju atau mundur. Jika satu roda berjalan Lebih lambat dari yang lain, robot akan berbelok ke sisi kecepatan yang lebih rendah. Jika kita ingin berbalik robot ke sisi kiri, kami mengurangi kecepatan roda kiri dan sebaliknya.

Setelah menjalankan beberapa konfigurasi yang sudah dijelaskan dibuku, berikut merupakan tampilan dari robotnya



PRAKTEK BAB 4

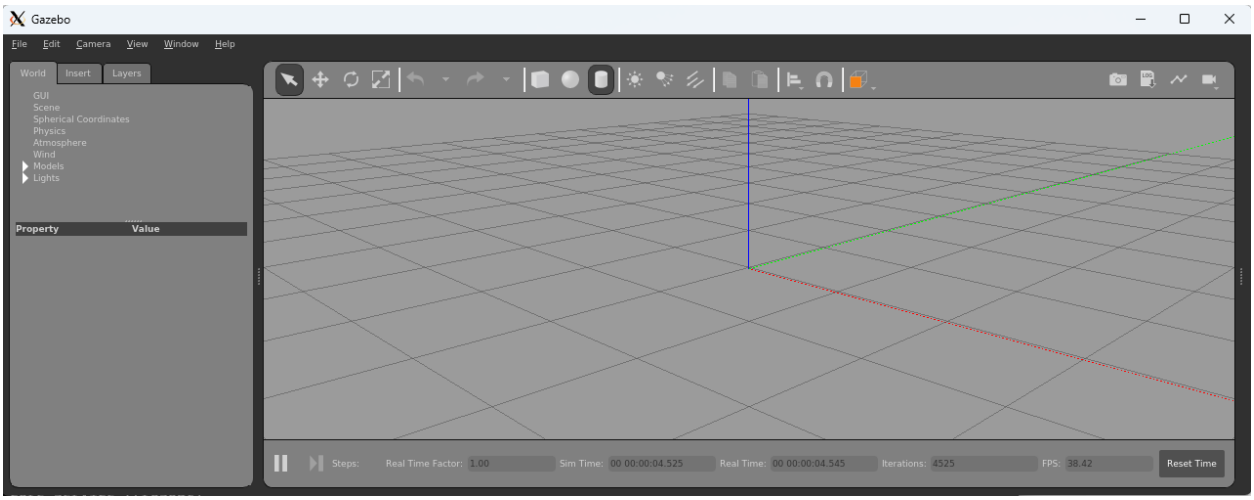
Simulating Robots Using ROS and Gazebo

Simulating the robotic arm using Gazebo and ROS

Pada bab sebelumnya, kami merancang lengan tujuh DOF. Di bagian ini, kami akan mensimulasikan robot di Gazebo menggunakan ROS. Sebelum memulai dengan Gazebo dan ROS, kita harus menginstal paket berikut untuk bekerja dengan Gazebo dan ROS:

```
sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic- gazebo-msgs
ros-noetic-gazebo-plugins ros-noetic-gazebo-ros- control
```

kita coba run gazebo untuk cek apakah gazebo bisa di run dengan baik



Creating the robotic arm simulation model for Gazebo

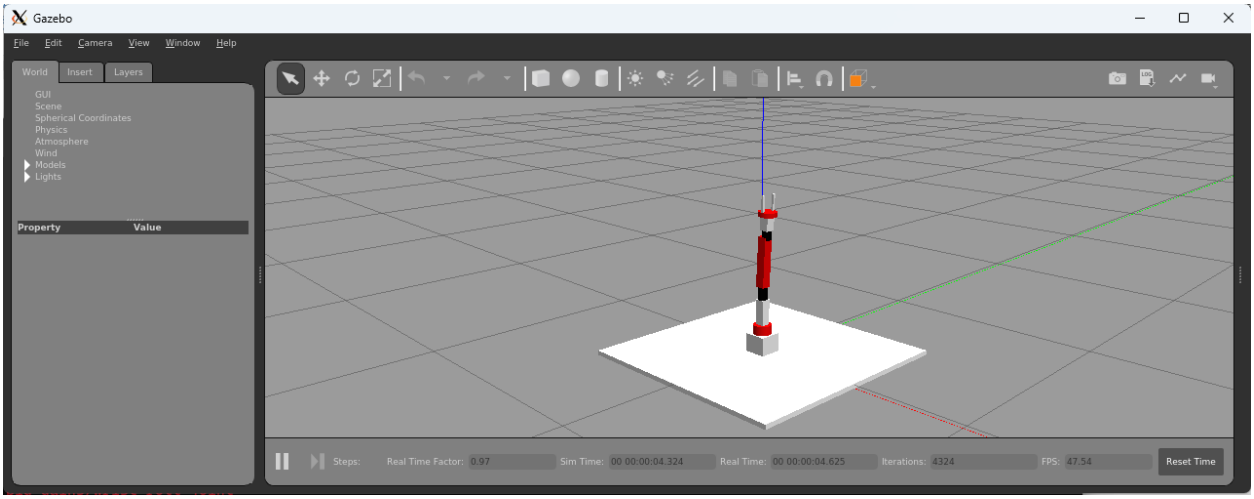
Kita dapat membuat model simulasi untuk lengan robot dengan memperbarui robot yang ada deskripsi dengan menambahkan parameter simulasi. Kita dapat membuat paket yang diperlukan untuk mensimulasikan lengan robot menggunakan perintah berikut:

```
catkin_create_pkg seven_dof_arm_gazebo gazebo_msgs gazebo_ plugins
gazebo_ros gazebo_ros_control mastering_ros_robot_ description_pkg
```

Berikut merupakan beberapa file .launch yang saya download di github reps

```
root@024a5ddf0613:~/catkin_ws/src/seven_dof_arm_gazebo/launch# ls
seven_dof_arm_bringup_moveit.launch          seven_dof_arm_gazebo_control.launch  seven_dof_arm_with_rgbd_world.launch
seven_dof_arm_bringup_obstacle_moveit.launch  seven_dof_arm_obstacle_world.launch  seven_dof_arm_world.launch
root@024a5ddf0613:~/catkin_ws/src/seven_dof_arm_gazebo/launch#
```

Setelah melakukan beberapa configurasi, kita dapat melakukan running pada robot yang baru kita buat tadi dengan cara



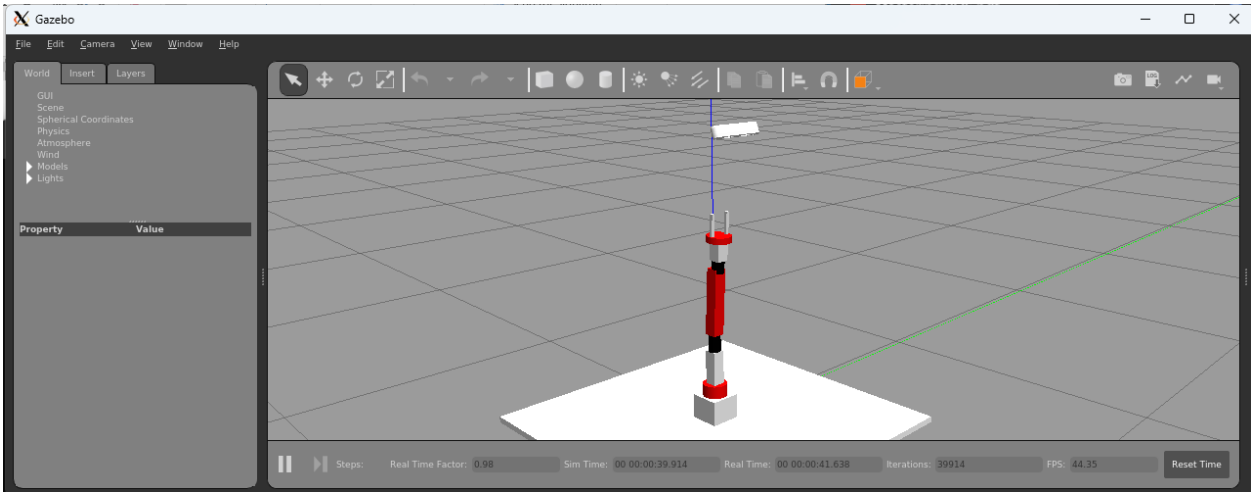
Adding colors and textures to the Gazebo robot model

Kita dapat melihat pada robot simulasi bahwa setiap tautan memiliki warna dan tekstur yang berbeda. Si Tag berikut di dalam file .xacro memberikan tekstur dan warna ke tautan robot:

```
<gazebo reference="bottom_link">

<material>Gazebo/White</material>

</gazebo>

<gazebo reference="base_link">

<material>Gazebo/White</material>

</gazebo>

<gazebo reference="shoulder_pan_link">

<material>Gazebo/Red</material>

</gazebo>
```

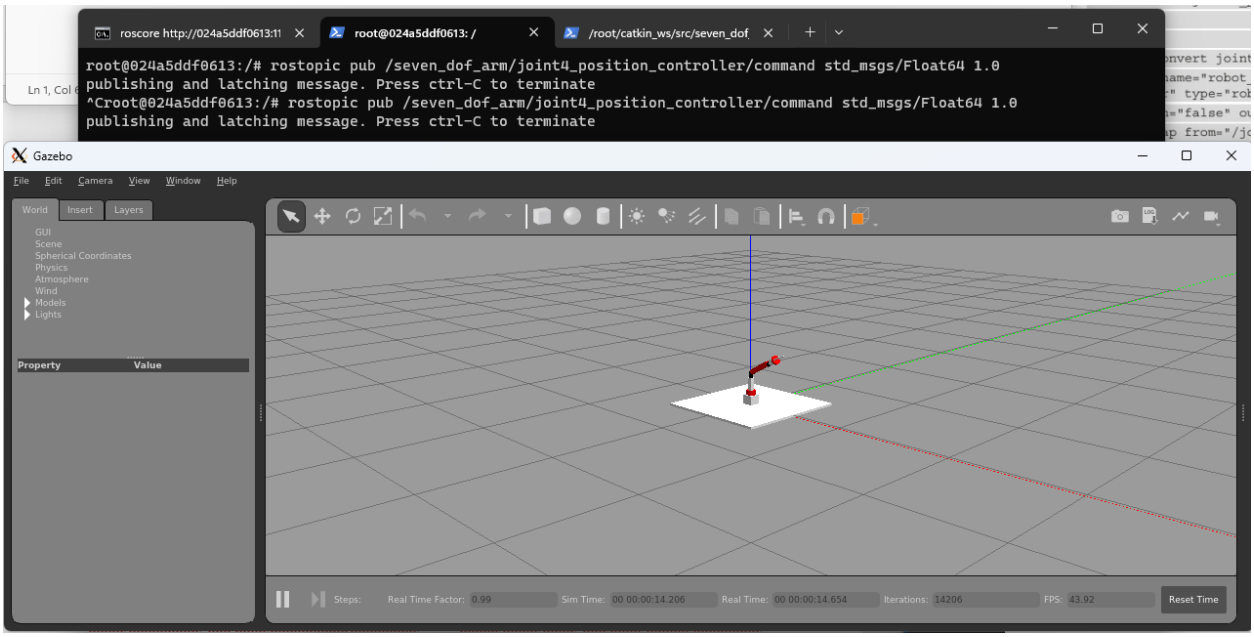
Simulating the robotic arm with Xtion Pro

Sekarang kita telah belajar tentang definisi plugin kamera di Gazebo, kita dapat meluncurkan Simulasi lengkap kami menggunakan perintah berikut:



Moving the robot joints using ROS controllers in Gazebo

Pada bagian ini, kita akan membahas cara memindahkan setiap sendi robot di Gazebo. Untuk memindahkan setiap sambungan, kita perlu menetapkan pengontrol ROS. Untuk setiap sendi, kita perlu pasang pengontrol yang kompatibel dengan antarmuka perangkat keras yang disebutkan di dalam tag transmisi.



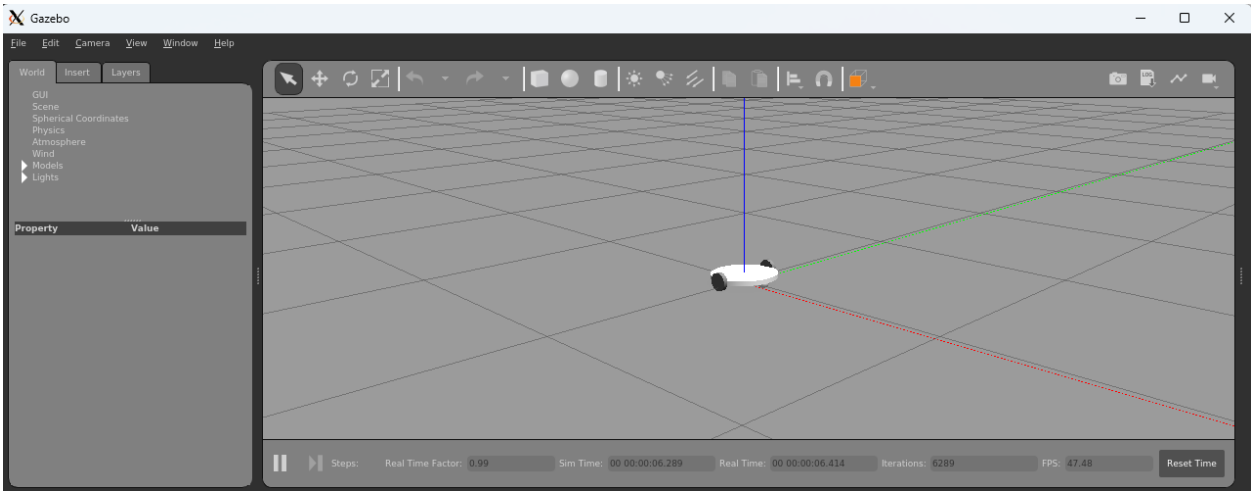
Kita dapat mempublish suatu topic dan menggerakan robot dedngan itu

Simulating a differential wheeled robot in Gazebo

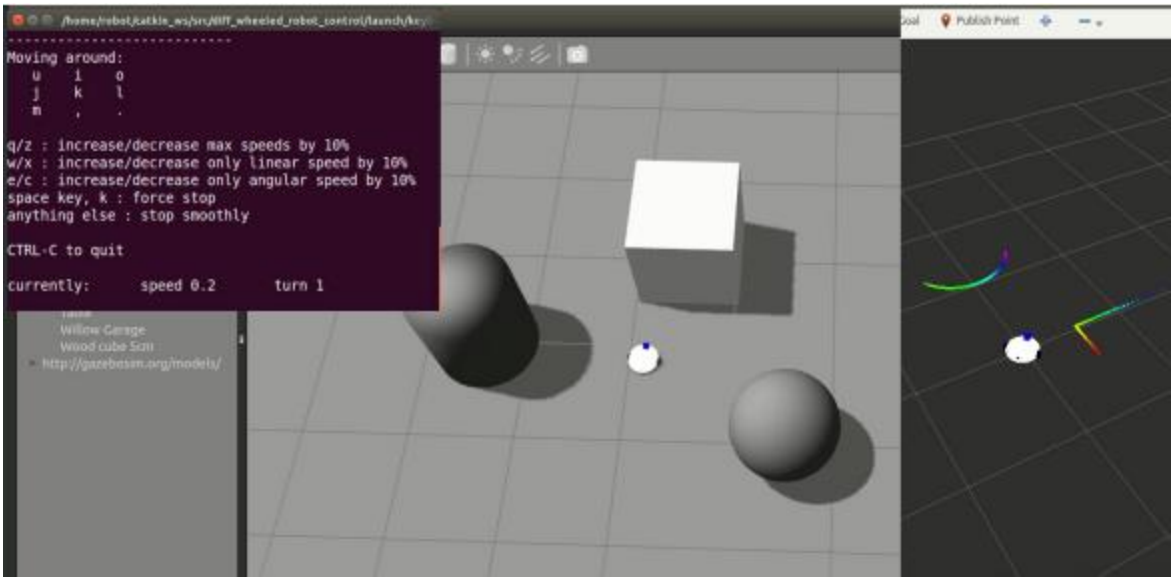
Kita telah melihat simulasi lengan robot. Di bagian ini, kita dapat mengatur Simulasi untuk robot beroda diferensial yang kami rancang di bab sebelumnya. Anda akan menemukan deskripsi robot seluler diff_wheeled_robot.xacro di mastering_ros_robot_description_pkg/URDF folder.

Mari buat file peluncuran untuk menelurkan model simulasi di Gazebo. Seperti yang kami lakukan untuk lengan robot, kita dapat membuat paket ROS untuk meluncurkan simulasi Gazebo menggunakan yang sama dependensi paket seven_dof_arm_gazebo. Jika Anda telah mengkloning repositori kode, Anda sudah memiliki paket ini, jika tidak, kloning seluruh kode dari Git repositori, atau dapatkan paket dari kode sumber buku:

Setelah melakukan beberapa konfigurasi setelahnya kita lakukan percobaan run, berikut merupakan output dari porgram



Node teleop ROS menerbitkan perintah ROS Twist dengan mengambil input keyboard. Dari simpul ini, kita dapat menghasilkan kecepatan linier dan sudut, dan sudah ada implementasi node teleop standar tersedia; Kita cukup menggunakan kembali simpul. Teleop diimplementasikan dalam paket diff_wheeled_robot_control. Naskah Folder berisi node diff_wheeled_robot_key, yang merupakan node teleop. Sesuai biasanya, Anda dapat mengunduh paket ini dari repositori Git sebelumnya. Pada titik ini, Anda Capai paket ini dengan perintah berikut:


Moving around:
u i o
j k l
m , .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
currently: speed 0.2 turn 1
Willow Garage
Wood cube 5cm
http://gazebosim.org/models/
Publish Point

PRAKTEK BAB 5

Simulating Robots Using ROS, CoppeliaSim, and Webots

Dalam bab ini, kita akan belajar cara mengatur simulator ini dan menghubungkannya dengan Jaringan ROS. Kami akan membahas beberapa kode awal untuk memahami bagaimana mereka bekerja dengan keduanya sebagai perangkat lunak mandiri dan bagaimana mereka dapat digunakan dengan layanan dan topik ROS. Kami akan membahas topik-topik berikut dalam bab ini:

• Menyiapkan CoppeliaSim dengan ROS

• Mensimulasikan lengan robot menggunakan CoppeliaSim dan ROS

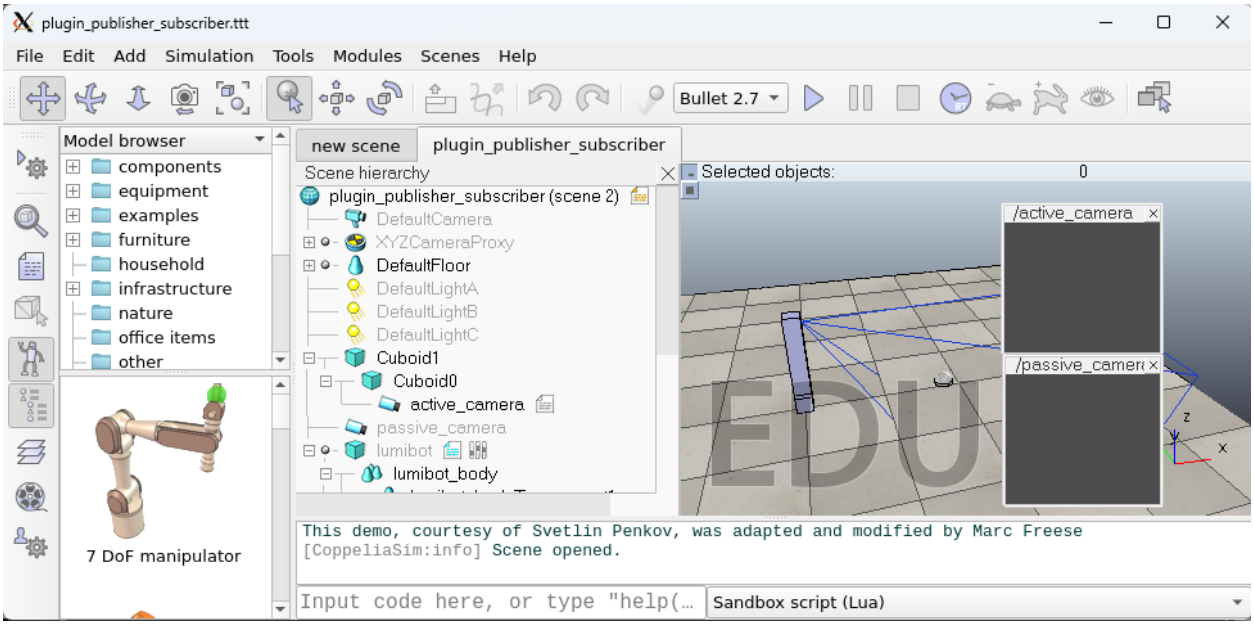
• Menyiapkan Webots dengan ROS

• Menulis pengontrol pertama Anda

• Menulis node teleoperation (teleop) menggunakan web

Setting up CoppeliaSim with ROS

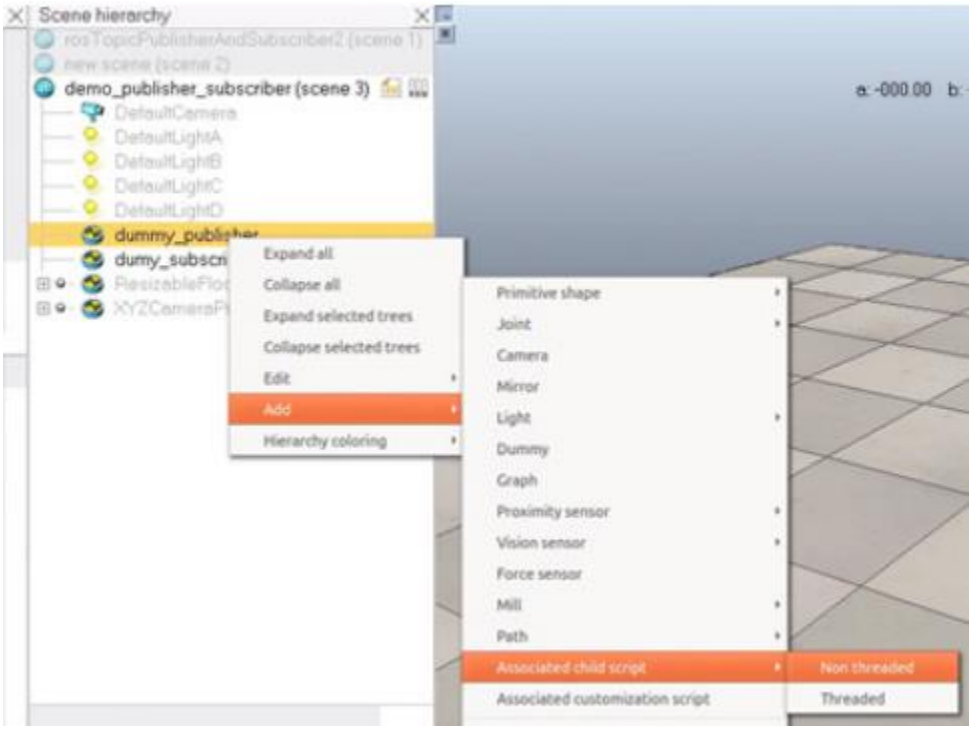
Sebelum mulai bekerja dengan CoppeliaSim, kita perlu menginstalnya di sistem kita dan mengkonfigurasi lingkungan kita untuk memulai jembatan komunikasi antara ROS dan adegan simulasi. CoppeliaSim adalah perangkat lunak lintas platform, tersedia untuk operasi yang berbeda sistem seperti Windows, macOS, dan Linux. Ini dikembangkan oleh Coppelia Robotics GmbH dan didistribusikan dengan lisensi pendidikan dan komersial gratis.



Berikut merupakan tampilan dari copliasim setelah selesai instalasi, jika copeliasim berjalan dengan baik, maka tampilan akan seperti diatas ini

Interacting with CoppeliaSim using ROS topics

Sekarang kita akan membahas cara menggunakan topik ROS untuk berkomunikasi dengan CoppeliaSim. Ini adalah berguna ketika kita ingin mengirim informasi ke objek simulasi, atau mengambil data dihasilkan oleh sensor robot atau aktuator. Cara paling umum untuk memprogram adegan simulasi simulator ini adalah dengan menggunakan Lua Skrip. Setiap objek adegan dapat dikaitkan dengan skrip yang dipanggil secara otomatis ketika simulasi dimulai dan dijalankan secara siklis selama waktu simulasi.



Working with ROS messages

Untuk mempublikasikan pesan ROS baru dalam skrip Lua, kita perlu membungkusnya dalam struktur data berisi bidang yang sama dari pesan asli. Prosedur terbalik harus dilakukanuntuk mengumpulkan informasi yang dipublikasikan tentang topik ROS. Mari kita menganalisis pekerjaan yang dilakukan di Contoh sebelumnya sebelum kita pindah ke sesuatu yang lebih rumit. Di dummy_ contoh penerbit, tujuannya adalah untuk mempublikasikan data bilangan bulat pada topik ROS. Kita bisa memeriksa struktur pesan bilangan bulat menggunakan perintah ROS ini:

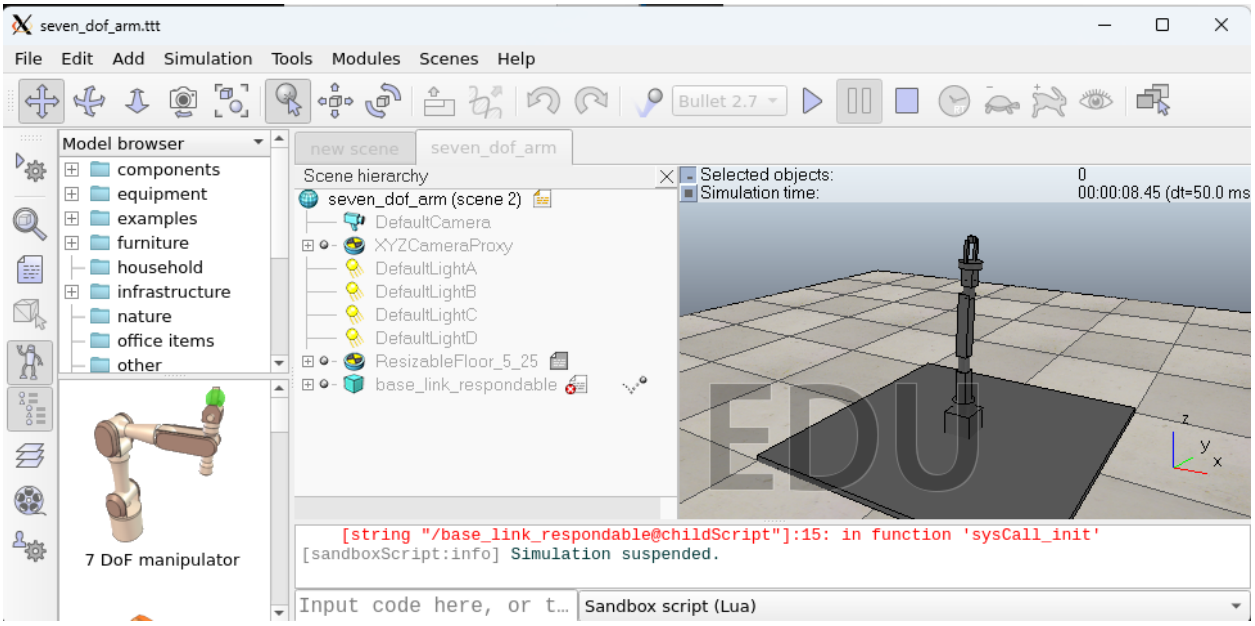
```
root@024a5ddf0613:/# rosmsg show std_msgs/Int32
int32 data

root@024a5ddf0613:/#
```

Ini berarti bahwa kita perlu mengisi bidang data dari struktur pesan dengan yang diinginkan nilai untuk streaming, seperti yang kami lakukan dalam skrip penerbit. Kode untuk melakukan ini dapat dilihat di sini:

Simulating a robotic arm using CoppeliaSim and ROS

Pada bab sebelumnya, kami menggunakan Gazebo untuk mengimpor dan mensimulasikan tujuh derajat lengan kebebasan (DOF) yang dirancang dalam Bab 3, Bekerja dengan ROS untuk Pemodelan 3D. Sini kami akan melakukan hal yang sama menggunakan CoppeliaSim. Langkah pertama untuk mensimulasikan tujuh-DOF kami ARM adalah untuk mengimpornya dalam adegan simulasi. CoppeliaSim memungkinkan Anda mengimpor robot baru menggunakan file URDF; untuk alasan ini, kita harus mengonversi model xacro lengan dalam URDF , menyimpan file URDF yang dihasilkan di folder urdf csim_demo

```
rosrun xacro seven_dof_arm.xacro > /path/to/csim_demo_pkg/urdf/seven_dof_arm.urdf
```



Setelah melakukan import URDF, kita akan melakukan pengaturan agar lengan robot siap diguankan, seperti mengkonfigurasi Scene Object Properties, dan juga Joint Dynamic Properties,

Di sini, kita menggunakan fungsi setJointTargetPosition untuk mengubah posisi a diberikan sendi. Argumen input dari fungsi-fungsi tersebut adalah penangan objek Bersama dan nilai yang akan ditetapkan. Setelah memulai simulasi, kita dapat memindahkan sendi yang diinginkan, seperti elbow_pitch joint, menerbitkan nilai menggunakan alat baris perintah, seperti yang diilustrasikan dalam cuplikan kode berikut:

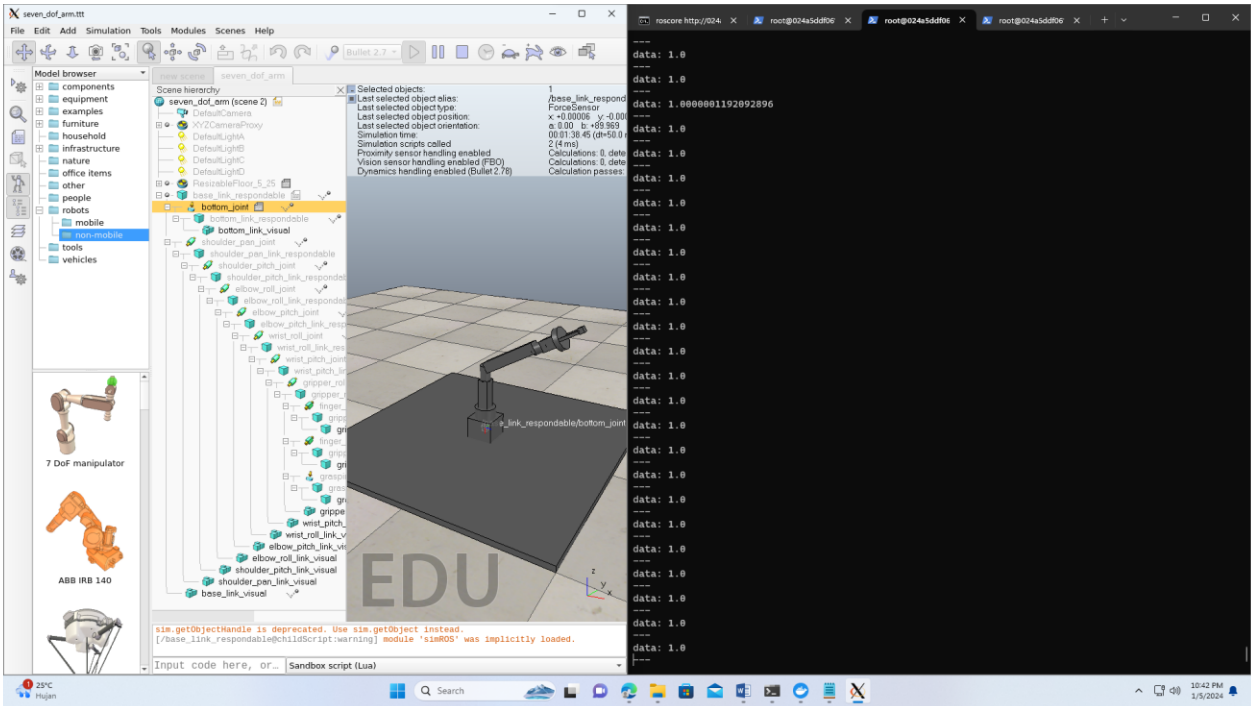
```
rostopic pub /csim_demo/seven_dof_arm/elbow_pitch/cmd std_msgs/ Float32 "data: 1.0"
```

At the same time, we can get the position of the joint listening on the state topic, as

follows:

```
rostopic echo /csim_demo/seven_dof_arm/elbow_pitch/state
```

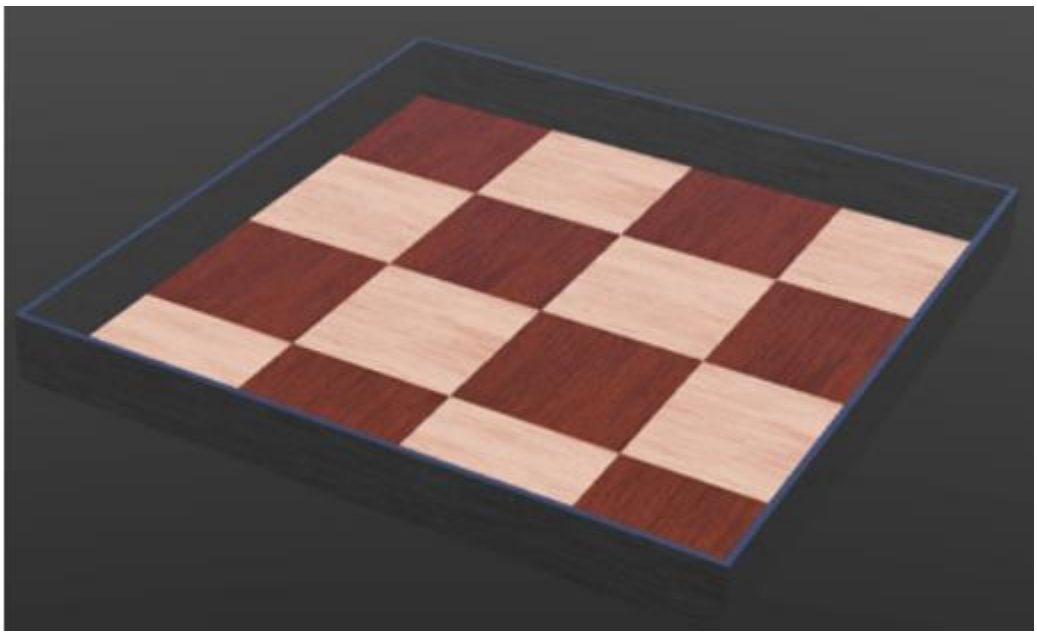
Berikut merupakan output dari kode diatas;



Terlihat lengan robot menjadi miring, itu dikarenakan kita mengirim data float 1.0

Simulating a mobile robot with Webots

Tujuan dari bagian ini adalah untuk membuat adegan simulasi dari awal, yang berisi objek dan robot beroda bergerak. Untuk melakukan ini, kita perlu menciptakan dunia kosong baru. Kita bisa Gunakan opsi wizard untuk membuat adegan simulasi baru. Dunia ini sudah tersedia di Kode sumber buku dalam paket webots_demo_pkg. Untuk membuat simulasi baru, gunakan menu bilah atas dan pilih Wizards | Direktori Proyek Baru. Applet akan membantu Anda untuk mengatur semuanya. Klik Berikutnya untuk memilih direktori proyek. Menyisipkan jalur folder sesuai keinginan Anda, dan pilih mobile_robot sebagai nama folder. Anda juga dapat memilih nama dunia; masukkan robot_motion_controller.wbt dan berhati-hatilah untuk menyematkan Tambahkan opsi area persegi panjang. Kemudian, klik Selesai dan, setelah adegan dimuat, seharusnya muncul seperti yang digambarkan dalam cuplikan layar berikut:



Writing your first controller

Di bagian ini, kita akan menulis controller pertama kita untuk mobile robot. Kami telah melihat Bagaimana pengontrol menangani gerakan robot dan reaksinya terhadap sensor. Mari kita ubah pengontrol robot E-puck untuk memindahkan beberapa arah tetap. Kita bisa memilih bahasa pemrograman yang berbeda untuk pengontrol kami; namun, kami akan menggunakan C ++. Tujuan dari pengontrol baru adalah untuk memerintahkan kecepatan roda robot untuk menunjukkan struktur khas pengontrol Webots.

The code of the controller is listed in the following snippet:

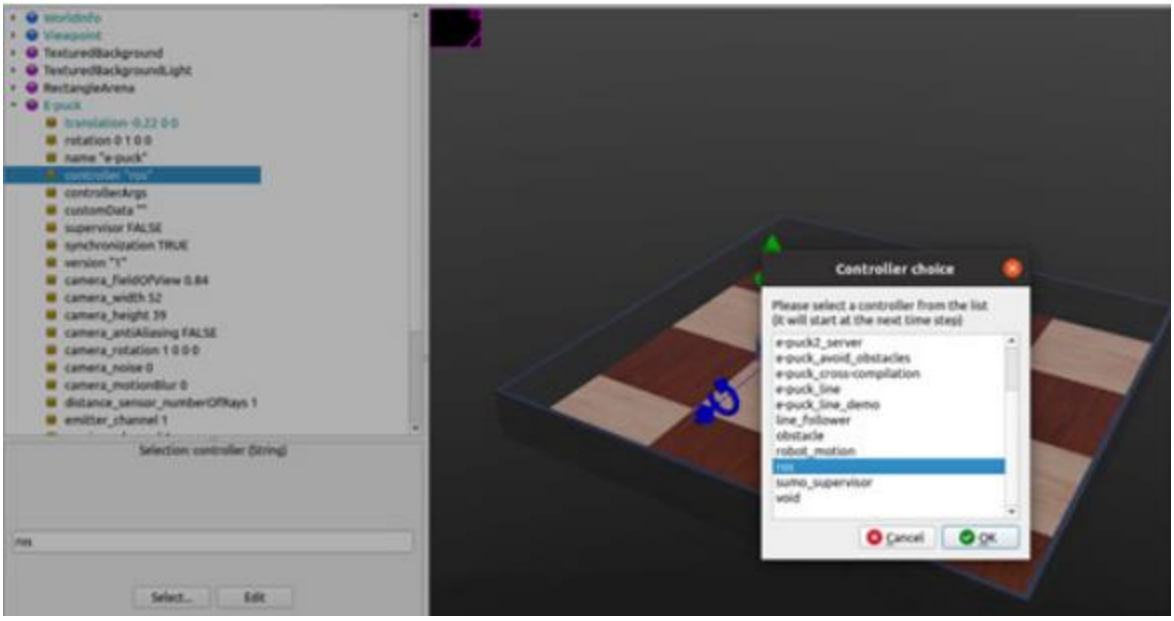
```
#include <webots/Robot.hpp>
#include <webots/Motor.hpp>
#define MAX_SPEED 6.28
//64 Milliseconds
#define TIME_STEP 64
using namespace webots;
```

```
int main(int argc, char **argv) {
Robot *robot = new Robot();
Motor *leftMotor = robot->getMotor("left wheel motor");
Motor *rightMotor = robot->getMotor("right wheel motor");
leftMotor->setPosition(INFINITY);
rightMotor->setPosition(INFINITY);
double t=0.0;
double r_direction=1.0;
while(true) {
leftMotor->setVelocity( MAX_SPEED*0.1);
rightMotor->setVelocity( r_direction*MAX_SPEED*0.1);
robot->step(TIME_STEP) ;
t+= TIME_STEP;
if ( t > 2000 ) {
r_direction*=-1.0;
}
if( t > 4000) {
r_direction = 1.0;
t = 0.0;
}
}
delete robot;
return 0;
}
```

Simulating the robotic arm using Webots and ROS

Integrasi Webots-ROS membutuhkan dua sisi: sisi ROS dan sisi Webots. The ROS sisi diimplementasikan melalui paket ROS webots_ros, sementara Webots mendukung ROS Secara native berkat pengontrol standar yang dapat ditambahkan ke model robot apa pun. Untuk menggunakan Webots dengan ROS, Anda perlu menginstal paket webots_ros. Ini dapat dilakukan dengan menggunakan APT, sebagai berikut:

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



Kesimpulan

Dalam bab ini, kami terutama mereplikasi apa yang telah kami lakukan di bab sebelumnya dengan Gazebo, menggunakan simulator robot lainnya: CoppeliaSim dan Webots. Ini adalah Program perangkat lunak simulasi multiplatform yang mengintegrasikan berbagai teknologi dan sangat serbaguna. Berkat UI intuitif mereka, mereka mungkin lebih mudah digunakan untuk pengguna baru. Kami terutama mensimulasikan dua robot, satu diimpor menggunakan file URDF dari tujuh-DOF lengan dirancang di bab-bab sebelumnya, dengan yang lainnya menjadi roda diferensial popular robot yang disediakan oleh model simulasi Webots. Kami belajar cara berinteraksi dan mengontrol sendi robot model kami dengan ROS dan cara memindahkan robot seluler penggerak diferensial menggunakan topik. Pada bab berikutnya, kita akan melihat bagaimana menghubungkan lengan robot dengan ROS MoveIt paket dan robot seluler dengan tumpukan Navigasi.