

**TECHNICAL REPORT ROBOTICS**  
**MASTERING ROS FOR ROBOTICS PROGRAMMING**



**Disusun Oleh:**

Ratika Dwi Anggraini (NIM: 1103201250)

**PROGRAM STUDI S1 TEKNIK KOMPUTER**  
**FAKULTAS TEKNIK ELEKTRO**  
**UNIVERSITAS TELKOM**  
**2023/2024**

# **BAB I**

## **PENDAHULUAN**

### **1.1 Latar Belakang**

Robotika, sebagai cabang ilmu yang berkembang pesat, telah menjadi fondasi penting dalam memahami, mengembangkan, dan menerapkan teknologi canggih di berbagai sektor kehidupan manusia. Salah satu faktor kunci yang mendukung kemajuan robotika adalah kehadiran kerangka kerja perangkat lunak yang dapat memfasilitasi pengembangan, integrasi, dan pengujian perangkat keras dan perangkat lunak robot dengan efisien. Dalam hal ini, Robot Operating System (ROS) telah menjadi pilihan utama bagi para peneliti dan pengembang dalam menangani kompleksitas dan skalabilitas sistem robotika.

ROS, yang awalnya dikembangkan oleh Willow Garage dan saat ini dikelola oleh Open Robotics, menyediakan kerangka kerja yang kuat dan modular untuk mengelola komunikasi antar komponen robot, memungkinkan pengembangan sistem robotika yang terdistribusi dan berbasis layanan. Kelebihan ROS tidak hanya terletak pada kemampuannya untuk memfasilitasi interaksi antar node dalam sistem robotika, tetapi juga dalam memberikan alat dan lingkungan simulasi yang kuat.

Simulasi robotika menjadi aspek kritis dalam pengembangan robot, memungkinkan para pengembang untuk menguji dan memvalidasi algoritma, sensor, dan tindakan secara virtual sebelum menerapkannya pada robot fisik yang sebenarnya. Dengan menggunakan simulasi, waktu dan biaya pengembangan dapat dikurangi, dan risiko kesalahan pada tingkat fisik dapat diminimalkan. Selain itu, integrasi antara ROS dan simulator seperti Gazebo, CoppeliaSim, dan Webots membuka peluang baru dalam pengembangan robotika. ROS memungkinkan komunikasi yang mulus antara robot fisik dan lingkungan simulasi, memberikan para pengembang fleksibilitas untuk menciptakan dan menguji model robotika dengan berbagai tingkat kompleksitas.

## **1.2 Tujuan**

Tujuan dalam technical report ini adalah sebagai berikut:

1. Untuk dapat mengetahui dan mengimplementasikan ROS
2. Untuk dapat mengetahui dan mengimplementasikan ROS Programming
3. Untuk dapat mengetahui dan mengimplementasikan ROS 3D Modeling
4. Untuk dapat mengetahui dan mengimplementasikan ROS dan Gazebo
5. Untuk dapat mengetahui dan mengimplementasikan ROS, CoppeliaSim dan Webots

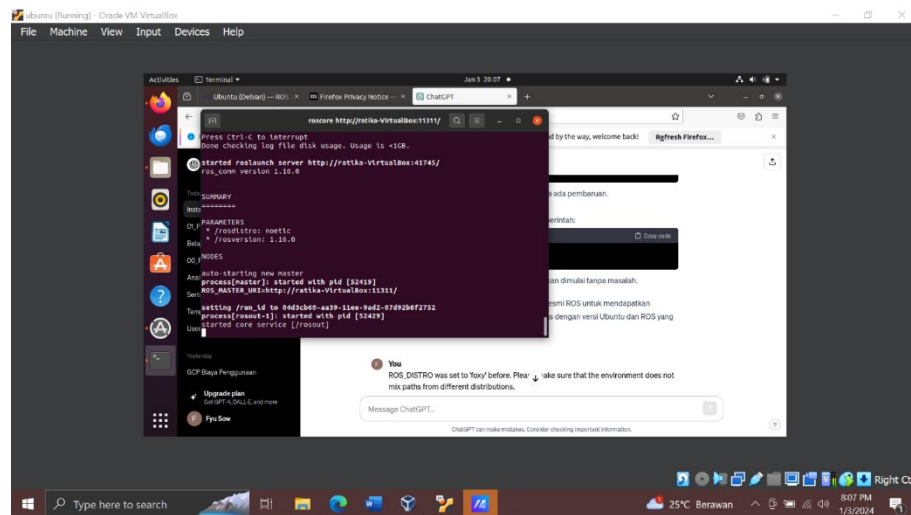
## BAB II

### PEMBAHASAN

#### 2.1 Introduction to ROS

##### 1. Running the ROS master and the ROS parameter server

Sebelum menjalankan node ROS, perlu untuk memulai ROS master dan parameter server ROS. Ini dapat dilakukan dengan perintah `roscore`, yang akan memulai ROS master, parameter server ROS, dan node logging `rosout`. Node `rosout` mengumpulkan pesan log dari node ROS lainnya dan menyimpannya dalam file log, serta menyiarkan ulang pesan log yang terkumpul ke topik lain. Topik `/rosout` dipublikasikan oleh node ROS menggunakan pustaka klien ROS seperti `roscpp` dan `rospy`, dan di-subscribe oleh node `rosout`, yang menyiarkan ulang pesan ke topik lain bernama `/rosout_agg`. Perintah `roscore` harus dijalankan sebelum menjalankan node ROS. Pesan log dan informasi server parameter akan muncul di terminal saat `roscore` dijalankan.

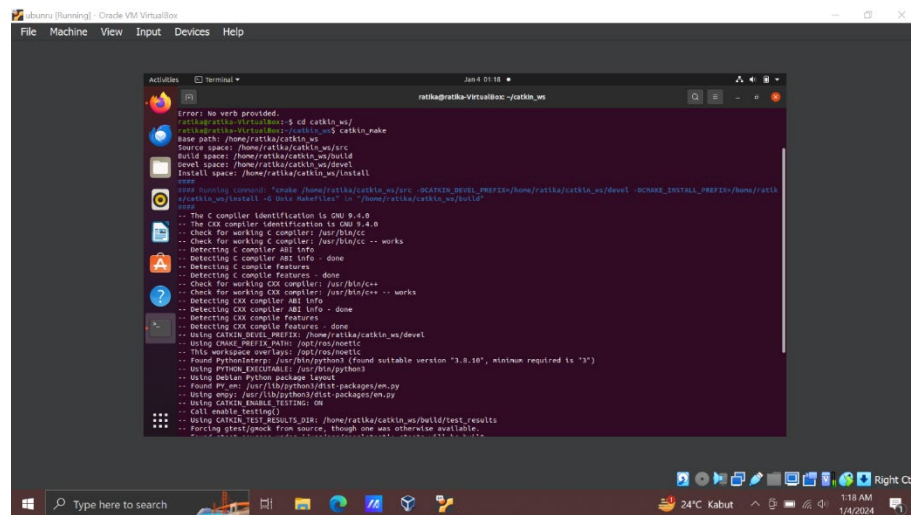


#### 2.2 Getting Started with ROS Programming

##### 1. Creating a ROS package

Paket ROS merupakan unit dasar dari program-program ROS. Paket ini dapat dibuat, dikompilasi, dan dirilis ke publik. Distribusi ROS yang digunakan saat ini adalah Noetic Ninjemys, dengan sistem pembangunan `catkin`. Sistem ini bertanggung jawab untuk menghasilkan target (eksekutabel/perpustakaan) dari kode sumber teks yang dapat

digunakan oleh pengguna akhir. Pada distribusi ROS sebelumnya seperti Electric dan Fuerte, sistem pembangunan yang digunakan adalah rosbuilt. Namun, karena berbagai kelemahan rosbuilt, dikembangkanlah catkin. Hal ini juga memungkinkan kita untuk mendekatkan sistem kompilasi ROS dengan Cross Platform Make (CMake), memberikan keuntungan seperti kemampuan porting paket ke OS lain, contohnya Windows, asalkan OS tersebut mendukung CMake dan Python. Langkah pertama untuk bekerja dengan paket ROS adalah membuat ruang kerja catkin. Setelah menginstal ROS, kita dapat membuat dan mengompilasi ruang kerja catkin bernama catkin\_ws, dimulai dengan membuat folder src dan melakukan inisialisasi ruang kerja catkin. Proses ini dapat dilakukan dengan menggunakan perintah catkin\_make setelah beralih ke direktori ruang kerja.



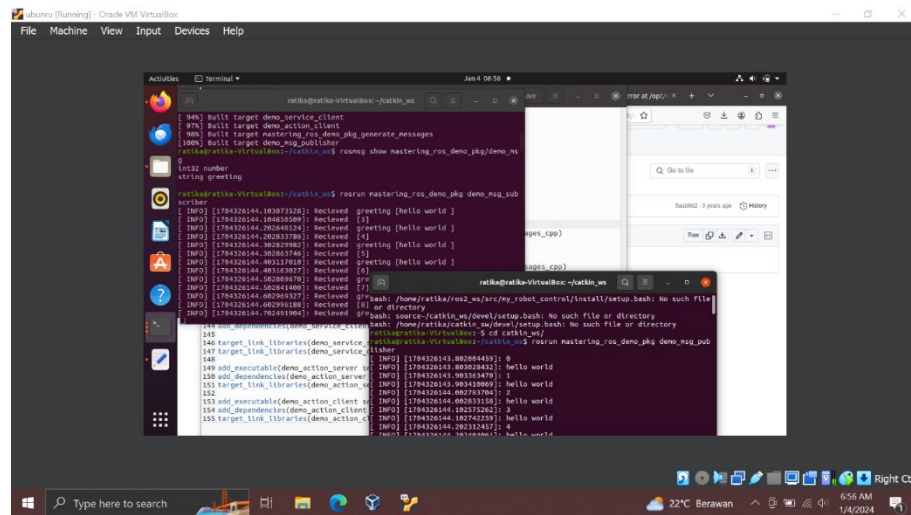
```
error: no verb provided.
ratika@ratika-VirtualBox:~$ cd catkin_ws/
ratika@ratika-VirtualBox:~/catkin_ws$ catkin_make
base path: /home/ratika/catkin_ws
Source space: /home/ratika/catkin_ws/src
Build space: /home/ratika/catkin_ws/build
Devel space: /home/ratika/catkin_ws/devel
Install space: /home/ratika/catkin_ws/install
====
Now running command: 'make /home/ratika/catkin_ws/src -DCATKIN_BUILD_PREFIX=/home/ratika/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=/home/ratika/catkin_ws/install -d this Makefile' in "/home/ratika/catkin_ws/build"
====
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc -- works
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/ratika/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Found Py.sr: /usr/lib/python3/dist-packages/setuptools
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/ratika/catkin_ws/build/test_results
-- Forcing stack/roscpp from source, though one was otherwise available.
-- Found roscpp: /usr/lib/cmake/roscpp/roscppConfig.cmake (found version 1.12.12)
-- Found std_msgs: /usr/lib/cmake/std_msgs/std_msgsConfig.cmake (found version 1.12.12)
-- Found actionlib: /usr/lib/cmake/actionlib/actionlibConfig.cmake (found version 1.12.12)
-- Found actionlib_msgs: /usr/lib/cmake/actionlib_msgs/actionlib_msgsConfig.cmake (found version 1.12.12)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ratika/catkin_ws/build
```

Perintah tersebut Perintah di atas menciptakan ruang kerja catkin dengan nama catkin\_ws, yang kemudian dikompilasi menggunakan perintah catkin\_make. Setelah mengatur ruang kerja, dibuat paket ROS dengan nama masterling\_ros\_demo\_pkg. Paket ini memanfaatkan dependensi seperti roscpp, std\_msgs, actionlib, dan actionlib\_msgs, yang diperlukan untuk implementasi C++ ROS, tipe data dasar ROS, dan pembuatan node berbasis actionlib. Proses ini memerlukan inisialisasi workspace, pembuatan paket, dan penambahan dependensi secara manual ke file CMakeLists.txt dan package.xml.



### 3. Adding custom .msg and .srv files

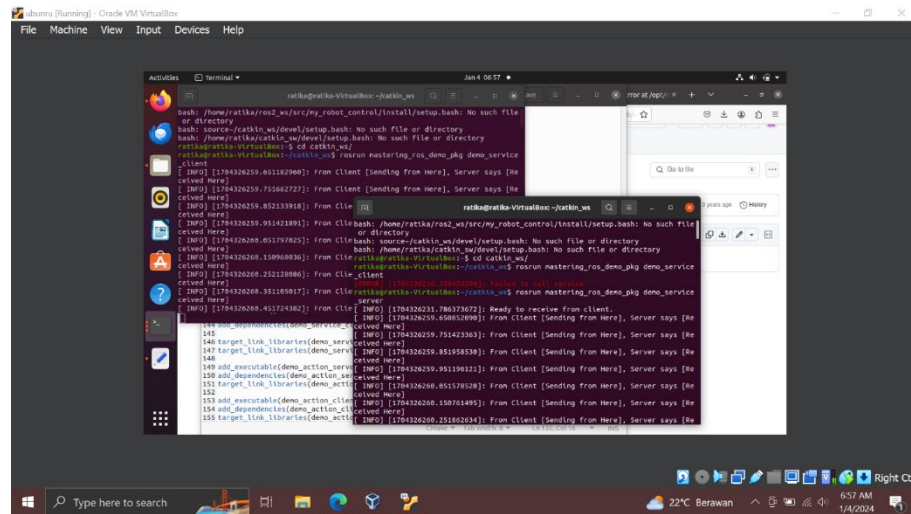
Dalam bagian ini, dijelaskan cara membuat definisi pesan dan layanan kustom dalam paket saat ini. Definisi pesan disimpan dalam file .msg, sedangkan definisi layanan disimpan dalam file .srv. Definisi ini memberi informasi kepada ROS tentang jenis data dan nama data yang akan dikirimkan dari sebuah node ROS. Saat pesan kustom ditambahkan, ROS akan mengonversi definisi tersebut menjadi kode C++ yang setara, yang dapat dimasukkan ke dalam node. Proses dimulai dengan penulisan definisi pesan dalam file .msg di dalam folder msg pada paket. Selanjutnya, perlu mengedit file package.xml dan CMakeLists.txt paket untuk memasukkan dependensi dan menghasilkan pesan yang diperlukan. Setelah langkah-langkah ini, dapat mengompilasi dan membangun paket untuk memastikan pesan telah dibangun dengan benar. Selanjutnya, dapat menguji pesan kustom tersebut dengan membangun penerbit dan pelanggan menggunakan tipe pesan kustom. Dengan langkah-langkah ini, memastikan integrasi pesan kustom dalam pengembangan ROS.



### 4. Working with ROS services

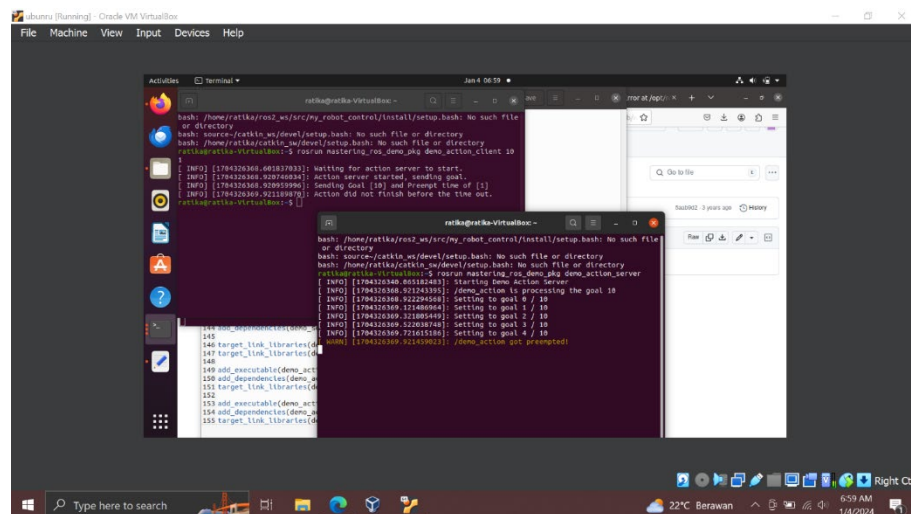
Pada bagian ini, dibahas pembuatan node ROS yang menggunakan definisi layanan yang sudah ditetapkan sebelumnya. Node layanan yang dibuat dapat mengirimkan pesan string sebagai permintaan ke server, dan selanjutnya, server akan mengirimkan pesan lain sebagai respons. Terdapat dua node yang dibuat, yaitu `demo_service_server.cpp` sebagai server dan `demo_service_client.cpp` sebagai client. Pada `demo_service_server.cpp`, terdapat fungsi callback yang dijalankan saat menerima permintaan dari client, sedangkan `demo_service_client.cpp` berfungsi sebagai client yang mengirimkan permintaan ke server.

Kedua node ini dapat dikompilasi dan dibangun menggunakan perintah `catkin_make` pada terminal. Setelah itu, nodes dapat dijalankan dengan menggunakan perintah `roslaunch`.



## 5. Building the ROS action server and client

Setelah membuat dua file dalam folder src, langkah selanjutnya adalah mengedit file package.xml dan CMakeLists.txt untuk membangun kedua node tersebut. Pada file package.xml, perlu ditambahkan dependensi terkait pembangkitan dan runtime pesan, seiring dengan layanan ROS. Selain itu, library Boost harus disertakan dalam CMakeLists.txt untuk pembangunan node-node ini. Action files yang telah ditulis juga perlu ditambahkan, bersama dengan dependensi actionlib, actionlib\_msgs, dan message\_generation. Setelahnya, eksekusi catkin\_make dapat dilakukan, dan nodes dapat dijalankan dengan perintah rosrn setelah menjalankan roscore. Proses ini mencakup pembangunan dan eksekusi action server dan action client, dengan output yang sesuai.





## 6. Creating launch files

File peluncuran (launch files) pada ROS sangat berguna untuk menjalankan lebih dari satu node sekaligus. Pada contoh sebelumnya, terdapat maksimal dua node ROS, namun bayangkan dalam skenario di mana perlu meluncurkan 10 atau 20 node untuk sebuah robot. Proses ini akan sulit jika harus menjalankan setiap node satu per satu di terminal. Sebagai alternatif, dapat menuliskan semua node di dalam sebuah file berbasis XML yang disebut file peluncuran (launch file), dan menggunakan perintah `roslaunch`, file ini diuraikan untuk menjalankan node-node tersebut. Perintah `roslaunch` akan secara otomatis memulai ROS master dan parameter server, sehingga tidak perlu menjalankan perintah `roscore` atau node-node secara individual; jika meluncurkan file peluncuran, semua operasi dilakukan dalam satu perintah. Perlu diingat bahwa jika memulai sebuah node menggunakan perintah `roslaunch`, mengakhiri atau me-restart perintah ini akan memiliki efek yang sama seperti me-restart `roscore`. Untuk membuat file peluncuran, dapat membuat file baru bernama `demo_topic.launch` yang akan meluncurkan dua node ROS untuk memublikasikan dan berlangganan nilai integer. Setelah membuat file peluncuran ini, dapat dijalankan menggunakan perintah `roslaunch`.

The screenshot displays a Windows 11 desktop environment. The active application is a terminal window titled "Jan 4 07:07". The terminal shows the execution of the following command:

```
./home/rttkk/rttkk_ws/src/mastering_ros_demo_pkg/launch/demo_topic_launch http://localhost:11311
```

The output of the command is as follows:

```
[rosrun] Found the following, but they're either not files,
[rosrun] or not executable:
[rosrun] ./home/rttkk/rttkk_ws/src/mastering_ros_demo_pkg/launch/demo_topic_launch
[rosrun] ./home/rttkk/rttkk_ws/src/mastering_ros_demo_pkg/launch/demo_topic_launch
[rosrun] Logging to /home/rttkk/.ros/log/2024-01-04/07:07:00-rttkk-rttkk-10253.log
[rosrun] Checking log directory for disk usage. This may take a while.
[rosrun] Press Ctrl+C to interrupt
[rosrun] Done checking log file disk usage. Usage is 1GB.

started ros2launch server http://rttkk-VirtualBox:11311/

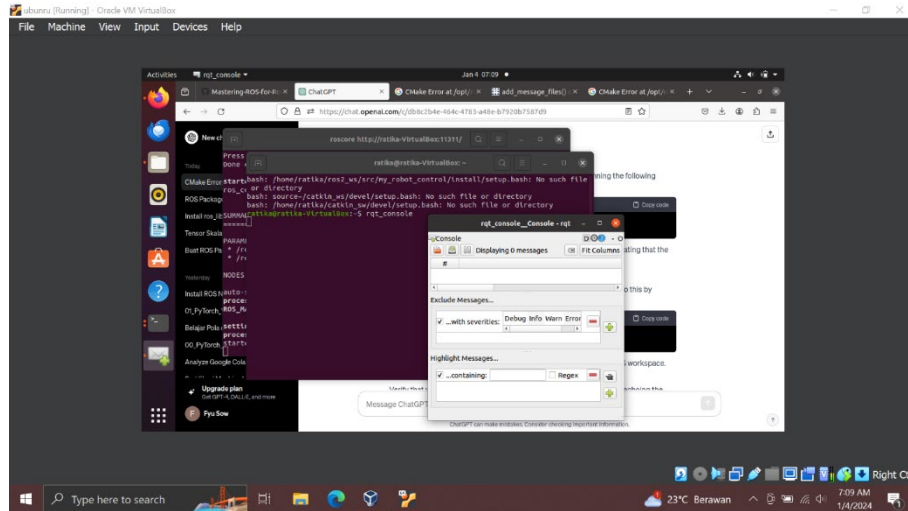
PARAMETERS
*****
PARAMETERS
* /stdout: none
* /rvsersion: 1.16.0
NODES
/
  publisher_node (mastering_ros_demo_pkg/demo_publisher)
  subscriber_node (mastering_ros_demo_pkg/demo_subscriber)

node: starting new master
process[roster]: started with pid [10271]
ROS_MASTER_URI=http://localhost:11311

setting [run_id to 3ff5f8e6-a95f-line-a629-612ebec12090]
process[roster-1]: started with pid [10282]
started core service [/roster]
process[publisher_node-1]: started with pid [10289]
process[subscriber_node-1]: started with pid [10299]
[INFO] [1784320846.99627464]: 0
[INFO] [1784320846.89678643]: 1
[INFO] [1784320846.89627596]: 2
[INFO] [1784320846.89627596]: 3
[INFO] [1784320846.89722674]: Received [1]
[INFO] [1784320846.19655537]: 4
[INFO] [1784320846.19655537]: 5
```

The Windows taskbar at the bottom shows the Start button, a search bar with the text "Type here to search", and several pinned application icons. The system tray on the right shows the date and time as "7:07 AM 1/4/2024", the weather for "Berawan" at "23°C", and icons for network, volume, and power.

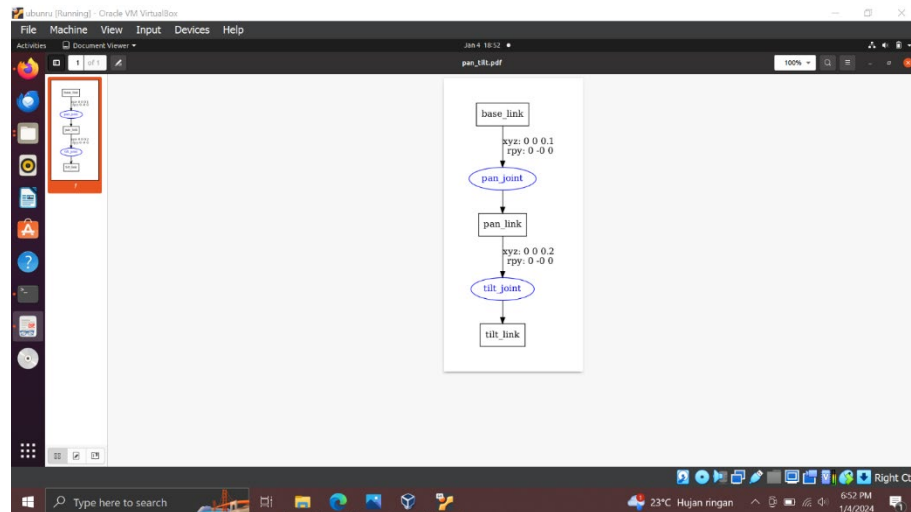
Untuk memeriksa daftar node, gunakan perintah 'rostopic list'. Untuk melihat pesan log dan melakukan debug pada node, gunakan alat GUI bernama 'rqt\_console'. Dengan ini, dapat melihat log yang dihasilkan oleh dua node di dalam alat ini.



## 2.3 Working with ROS for 3D Modeling

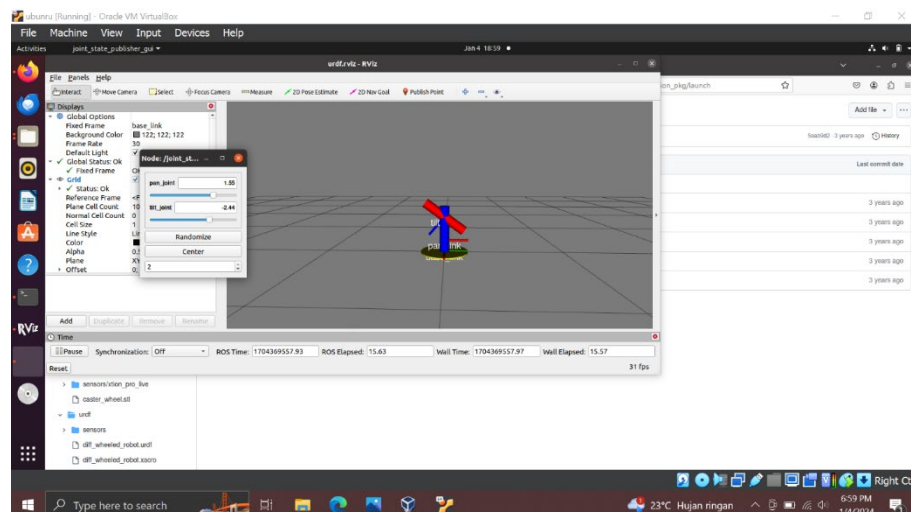
### 1. Explaining the URDF file

Dalam pemeriksaan kode URDF untuk model robot pan-and-tilt, perlu menambahkan tag `<robot>` di bagian atas deskripsi agar sistem mengenali file ini sebagai bahasa markup dan memungkinkan penyorotan kata kunci oleh editor teks. Tag `<robot>` mendefinisikan nama robot (diberi nama `pan_tilt`). Kode URDF kemudian mengandung definisi link dan joint untuk mekanisme pan-and-tilt. Definisi `base_link` mencakup tag `<visual>` yang menggambarkan penampilan visual link, seperti geometri dan materi. Selanjutnya, ada definisi joint dengan tipe dan link parent-child yang spesifik, serta origin dan sumbu joint. Setelah menyimpan kode URDF sebagai `pan_tilt.urdf`, dapat diperiksa apakah file URDF mengandung kesalahan dengan perintah `check_urdf`. Untuk visualisasi grafis struktur link dan joint robot, dapat digunakan perintah `urdf_to_graphviz` yang menghasilkan file gambar dan PDF. Struktur robot dapat dilihat menggunakan `evince pan_tilt.pdf`. Visualisasi grafis membantu memahami posisi dan hubungan setiap joint pada robot, sementara untuk melihat model secara tiga dimensi, dapat menggunakan RViz.



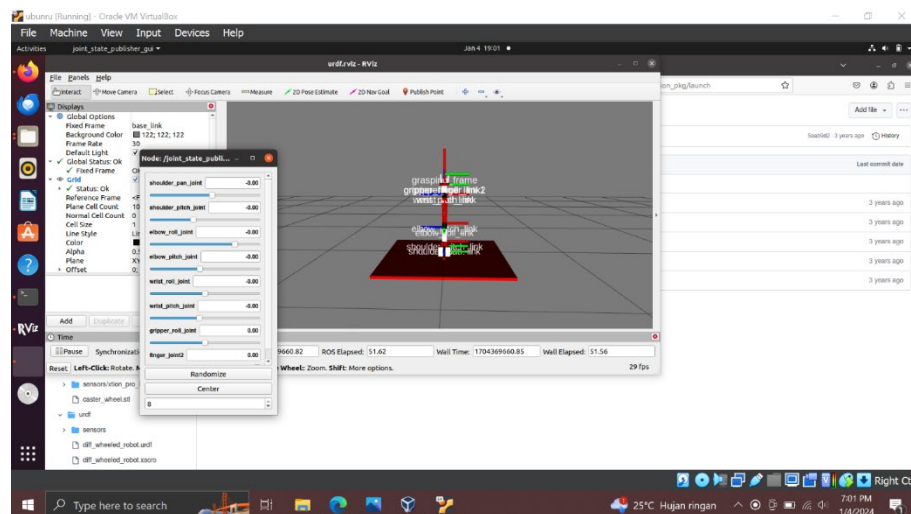
## 2. Visualizing the 3D robot model in Rviz

Setelah merancang URDF, dapat dilihat pada RViz. Dapat membuat file peluncuran `view_demo.launch` dan memasukkan kode berikut ke dalam folder peluncuran. Buka direktori `mastering_ros_robot_description_pkg/launch` untuk melihat kode tersebut. File launch tersebut mencakup parameter untuk menyimpan deskripsi robot dari file URDF yang dirancang. Meluncurkan model menggunakan perintah `roslaunch` akan menampilkan mekanisme pan-and-tilt pada RViz, seperti yang ditunjukkan pada gambar. Pada versi ROS sebelumnya, GUI dari `joint_state_publisher` diaktifkan dengan parameter ROS bernama `use_gui`. Namun, pada versi ROS saat ini, file peluncuran perlu diperbarui untuk meluncurkan `joint_state_publisher_gui` daripada menggunakan `joint_state_publisher` dengan parameter `use_gui`.



## 3. Viewing the seven-DOF arm in RViz

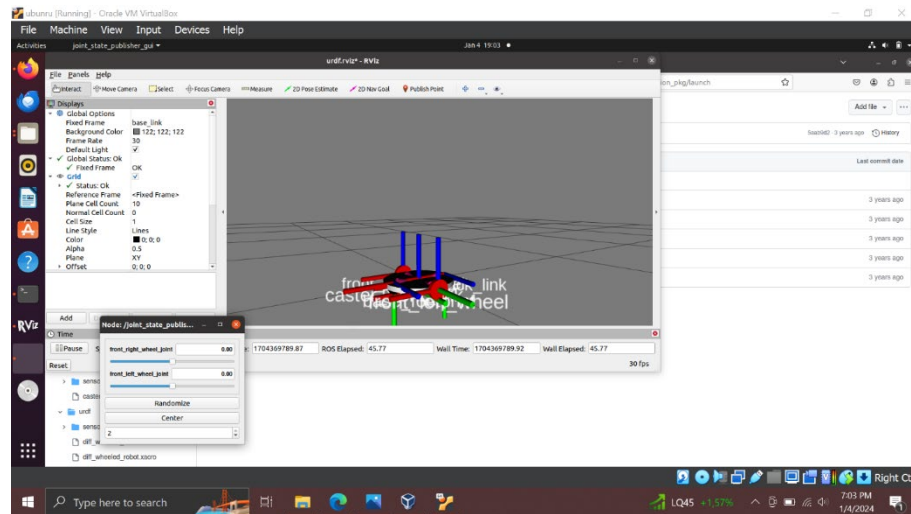
Setelah membahas model robot, sekarang dapat melihat file xacro yang dirancang di RViz, mengendalikan setiap sendi menggunakan node joint state publisher, dan mempublikasikan status robot menggunakan robot state publisher. Tugas tersebut dapat dilakukan menggunakan file peluncuran view\_arm.launch yang berada di dalam folder launch paket ini. Dalam file ini, parameter robot\_description diisi dengan perintah xacro untuk menghasilkan deskripsi robot dari file xacro yang telah dirancang. Dengan meluncurkan file tersebut, robot akan ditampilkan di RViz dengan GUI joint\_state\_publisher, memungkinkan interaksi dengan slider sendi dan pergerakan sendi robot. Selanjutnya, akan menjelajahi fungsi-fungsi dari joint state publisher dan robot state publisher, membantu mempublikasikan status robot ke tf, dan memvisualisasikan transformasi robot di RViz, serta memahami cara membuat model robot mobile dengan mekanisme roda berdiferensiasi.



#### 4. Creating a robot model for the differential drive mobile robot

Sebuah robot dengan mekanisme roda berdiferensiasi memiliki dua roda yang terhubung ke sisi berlawanan dari kerangka robot, yang didukung oleh satu atau dua roda castor. Kecepatan robot dikendalikan melalui regulasi kecepatan masing-masing roda. Jika dua 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 dengan kecepatan lebih rendah. Dalam model URDF robot ini, terdapat lima sendi dan link. Dua sendi utama menghubungkan roda dengan basis robot, sedangkan sendi lainnya adalah sendi tetap yang menghubungkan roda castor dan footprint dasar ke tubuh robot. File URDF yang digunakan disebut `diff_wheeled_robot.xacro`, yang memanfaatkan definisi xacro roda

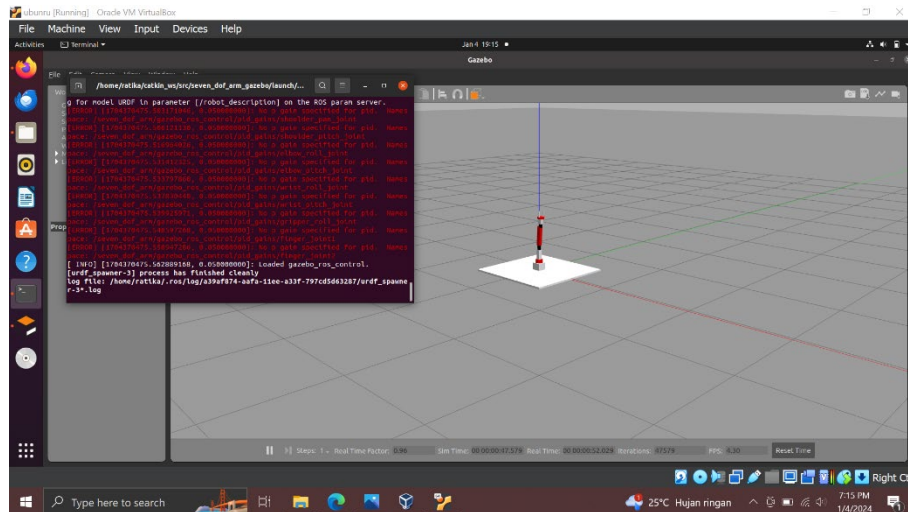
untuk mencegah duplikasi. File peluncuran `view_mobile_robot.launch` memungkinkan visualisasi robot pada RViz, dengan menginisialisasi parameter `robot_description` dan menjalankan joint state publisher dan robot state publisher. Melalui perintah `roslaunch`, robot dapat dilihat di RViz, meskipun tidak dapat diaktifkan untuk bergerak secara nyata.



## 2.4 Simulating Robots Using ROS and Gazebo

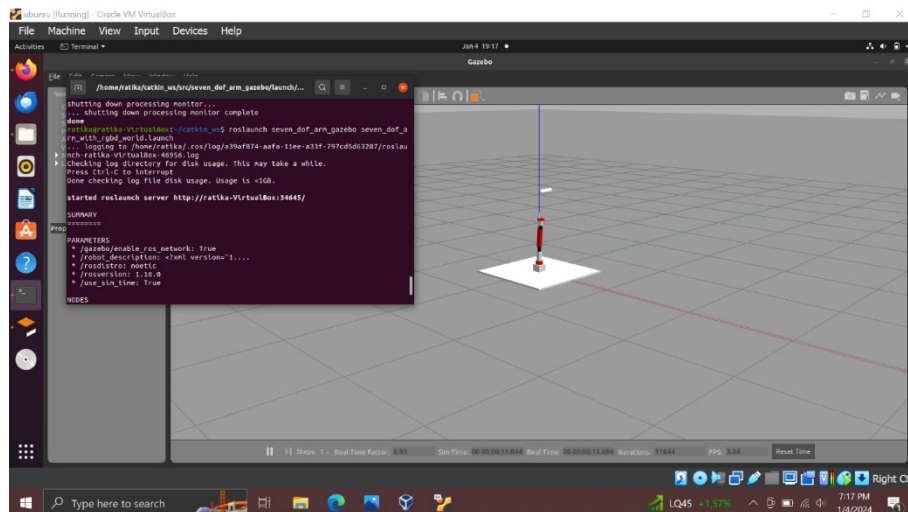
### 1. Creating the robotic arm simulation model for Gazebo

Untuk membuat model simulasi lengan robot dalam lingkungan Gazebo, dapat dilakukan dengan mengupdate deskripsi robot yang ada dan menambahkan parameter simulasi. Langkah awalnya adalah membuat paket simulasi menggunakan perintah `'catkin_create_pkg'`, yang mencakup dependensi seperti `'gazebo_msgs'`, `'gazebo_plugins'`, `'gazebo_ros'`, `'gazebo_ros_control'`, dan `'mastering_ros_robot_description_pkg'`. Selanjutnya, dapat diakses paket lengkap dari repositori Git atau sumber kode buku. Simulasi lengkap robot terdapat dalam file `'seven_dof_arm.xacro'` di direktori `'mastering_ros_robot_description_pkg/urdf/'`, yang diisi dengan tag URDF untuk collision, inertial, transmission, joints, links, dan Gazebo. Untuk meluncurkan model simulasi, digunakan paket `'seven_dof_arm_gazebo'` dengan file `'seven_dof_arm_world.launch'`. File ini memuat argumen seperti `paused`, `use_sim_time`, `gui`, `headless`, dan `debug`. Hasilnya dapat dilihat dengan menjalankan perintah `'roslaunch seven_dof_arm_gazebo seven_dof_arm_world.launch'`, dan jika tidak ada kesalahan, simulasi lingkungan Gazebo dengan lengan robot akan muncul.



## 2. Simulating the robotic arm with Xtion Pro

Setelah mempelajari definisi plugin kamera di Gazebo, dapat dilakukan peluncuran simulasi lengkap menggunakan perintah `'roslaunch seven_dof_arm_gazebo seven_dof_arm_with_rgbd_world.launch'`. Pada simulasi ini, model robot dilengkapi dengan sensor RGB-D yang terletak di bagian atas lengan. Dengan simulasi ini, dapat berinteraksi dengan sensor RGB-D secara virtual seolah-olah terhubung langsung ke komputer. Selanjutnya, dapat memeriksa apakah sensor tersebut memberikan output gambar yang sesuai.

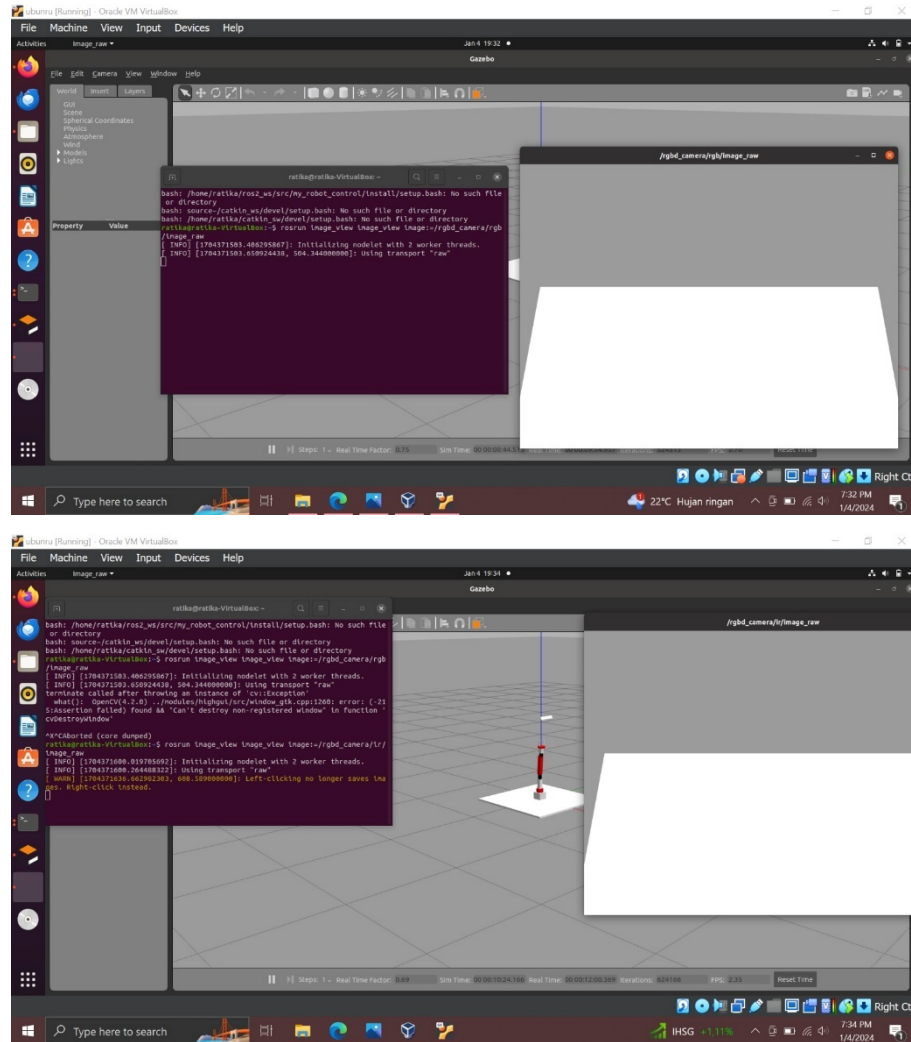


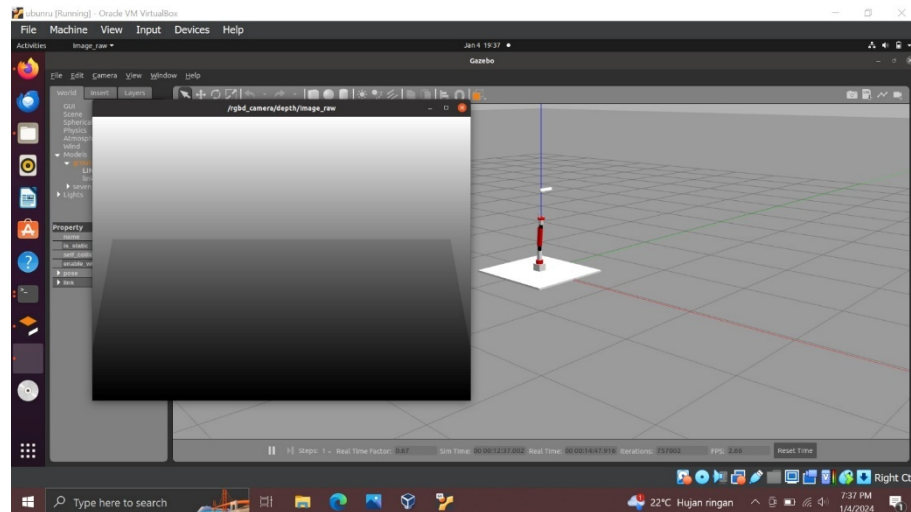
## 3. Visualizing the 3D sensor data

Setelah meluncurkan simulasi menggunakan perintah sebelumnya, dapat memeriksa topik yang dihasilkan oleh plugin sensor. Untuk melihat data gambar dari sensor visi 3D menggunakan alat bernama `image_view`, langkah-langkahnya sebagai berikut: melihat



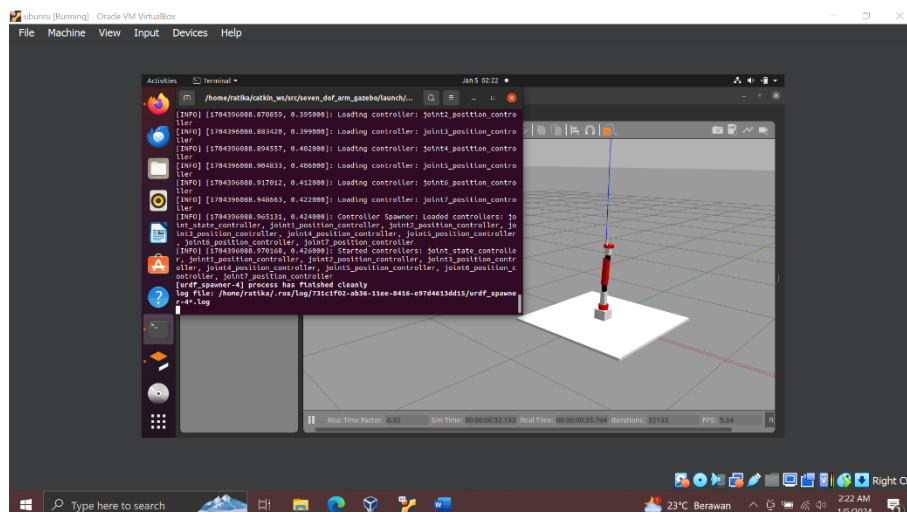
gambar RGB mentah dengan perintah `roslaunch image_view` `image:=/rgbd_camera/rgb/image_raw`, melihat gambar IR mentah dengan perintah `roslaunch image_view` `image:=/rgbd_camera/ir/image_raw`, dan melihat gambar kedalaman dengan perintah `roslaunch image_view` `image:=/rgbd_camera/depth/image_raw`. Hasilnya dapat dilihat dalam tangkapan layar yang menampilkan ketiga jenis gambar tersebut.





#### 4. Launching the ROS controllers with Gazebo

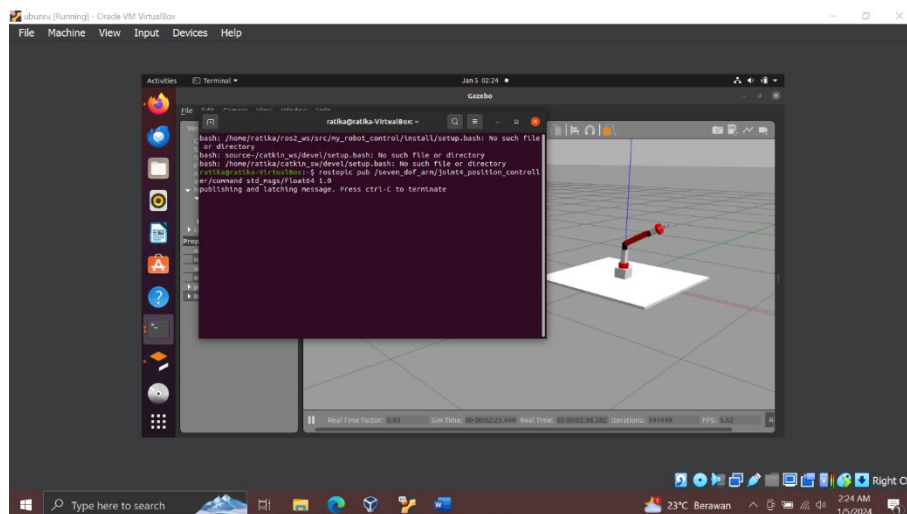
Jika konfigurasi pengontrol sudah siap, dapat membuat file peluncuran yang memulai semua pengontrol bersamaan dengan simulasi Gazebo. Buka file `seven_dof_arm_gazebo_control.launch` di direktori `seven_dof_arm_gazebo/launch`. File peluncuran tersebut memulai simulasi Gazebo untuk lengan robot, memuat konfigurasi pengontrol, memuat pengontrol keadaan sendi dan pengontrol posisi sendi, dan akhirnya menjalankan penerbit keadaan robot, yang menerbitkan keadaan sendi dan transformasi (TF). Setelah menjalankan file peluncuran ini dengan perintah `roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo_control.launch`, dapat memeriksa topik-topik pengontrol yang dihasilkan, dan jika berhasil, akan muncul pesan di terminal serta topik baru untuk setiap sendi guna mengontrol posisinya, seperti yang terlihat dalam tangkapan layar sebelumnya.





## 5. Moving the robot joints

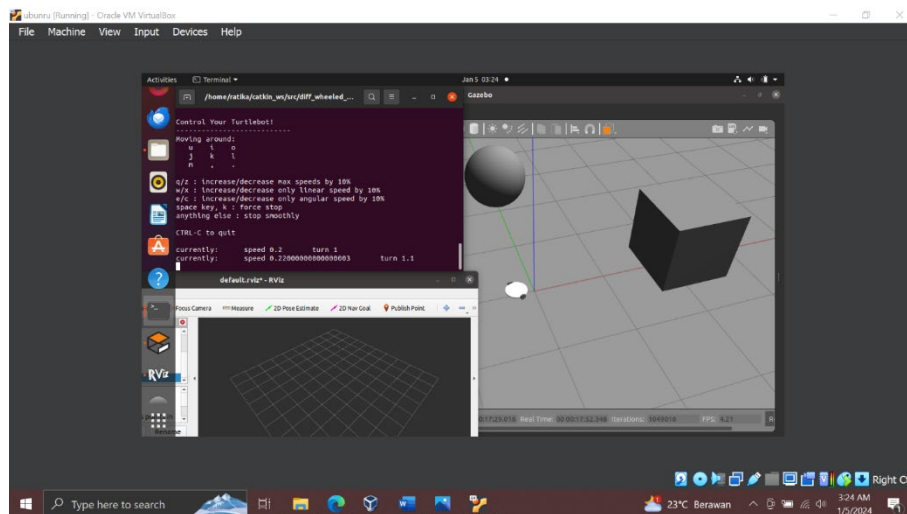
Setelah menyelesaikan topik sebelumnya, dapat mulai menggerakkan setiap sendi ke posisi yang diinginkan dalam simulasi Gazebo. Untuk menggerakkan sendi robot, harus menerbitkan nilai sendi yang diinginkan dengan tipe pesan `std_msgs/Float64` ke topik perintah pengontrol posisi sendi. Contohnya, untuk menggerakkan sendi keempat menjadi 1.0 radian, dapat menggunakan perintah `rostopic pub /seven_dof_arm/joint4_position_controller/command std_msgs/Float64 1.0`. Selain itu, dapat melihat keadaan sendi robot dengan perintah `rostopic echo /seven_dof_arm/joint_states`. Dengan ini, dapat mengendalikan semua sendi lengan robot tujuh derajat kebebasan dan pada saat yang sama membaca nilai-nilai mereka. Dengan cara ini, dapat mengimplementasikan algoritma pengendalian robot kustom. Pada bagian selanjutnya, akan mempelajari cara mensimulasikan robot penggerak differential-drive.



## 6. Adding the ROS teleop node

Node teleop ROS mempublikasikan perintah ROS Twist dengan menggunakan input dari keyboard. Implementasi teleop ini tersedia dalam paket `diff_wheeled_robot_control`, dengan node `diff_wheeled_robot_key` sebagai node teleop. Paket ini dapat diunduh dari repositori Git yang telah disebutkan sebelumnya. Untuk mengakses paket ini, gunakan perintah `'roscd diff_wheeled_robot_control'`. Untuk berhasil mengompilasi dan menggunakan paket ini, mungkin perlu menginstal paket `joy_node` dengan perintah `'sudo apt-get install ros-noetic-joy'`. Terdapat juga file peluncuran `keyboard_teleop.launch` yang memulai node teleop dengan parameter-parameter tertentu. Setelah mengatur simulasi Gazebo menggunakan perintah `'roslaunch diff_wheeled_robot_gazebo`

diff\_wheeled\_gazebo\_full.launch', node teleop dapat dimulai dengan 'roslaunch diff\_wheeled\_robot\_control keyboard\_teleop.launch'. Visualisasi keadaan robot dan data laser dapat dilihat melalui RViz. Pada terminal teleop, dapat menggunakan tombol-tombol untuk mengarahkan dan menyesuaikan kecepatan robot. Robot akan bergerak hanya ketika tombol yang sesuai ditekan di dalam terminal teleop. Jika semuanya berjalan lancar, dapat menjelajahi area menggunakan robot dan visualisasi data laser di RViz.

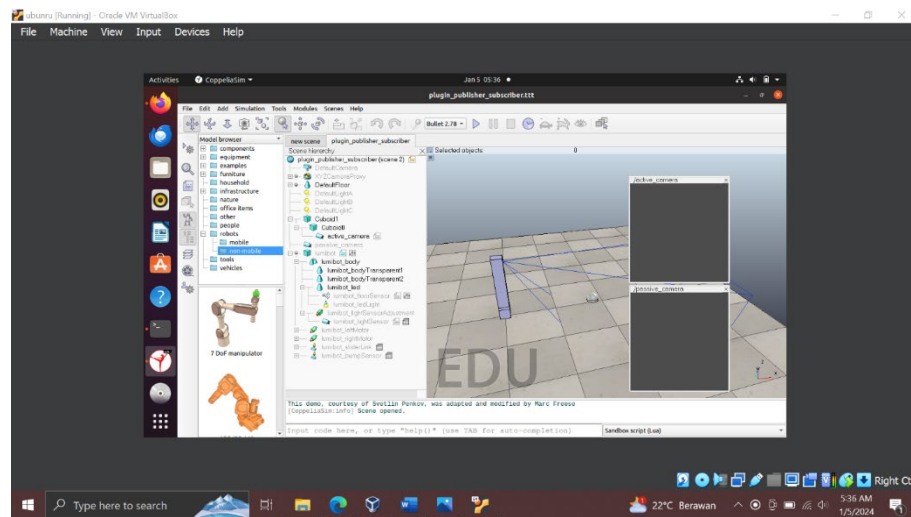


## 2.5 Simulating Robots Using ROS, CoppeliaSim, and Webots

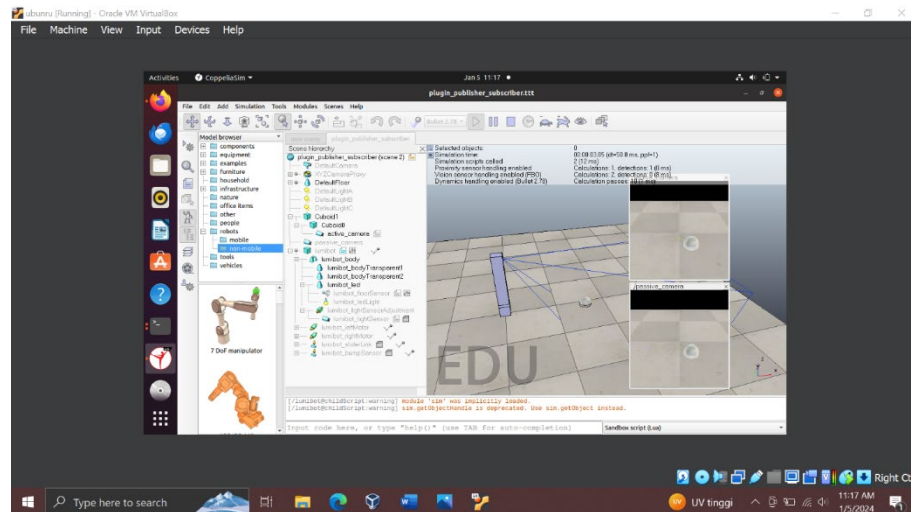
### 1. Setting up CoppeliaSim with ROS

Sebelum memulai bekerja dengan CoppeliaSim, perlu menginstalnya pada sistem dan mengonfigurasi lingkungan untuk memulai jembatan komunikasi antara ROS dan skenario simulasi. CoppeliaSim adalah perangkat lunak lintas platform yang tersedia untuk sistem operasi berbeda seperti Windows, macOS, dan Linux. Setelah mengunduh versi terbaru dari simulator CoppeliaSim dari halaman unduhan Coppelia Robotics, ekstrak arsip tersebut dan ubah nama folder untuk memudahkan akses. Selanjutnya, atur variabel lingkungan COPPELIASIM\_ROOT yang menunjukkan ke folder utama CoppeliaSim. CoppeliaSim mendukung beberapa mode pengendalian robot simulasi, termasuk Remote API yang terdiri dari fungsi-fungsi yang dapat dipanggil dari aplikasi eksternal, serta RosInterface yang berfungsi sebagai antarmuka saat ini untuk komunikasi antara ROS dan CoppeliaSim. Pada tahap ini, perlu konfigurasi lingkungan untuk menjalankan CoppeliaSim, termasuk memaksa sistem operasi untuk memuat pustaka bersama Lua dan

Qt5 dari folder utama CoppeliaSim. Setelah itu, simulasinya dapat dimulai dengan menjalankan roscore dan membuka CoppeliaSim dengan perintah yang sesuai. Selama startup, semua plugin yang terpasang akan dimuat. Untuk memeriksa apakah semuanya berfungsi dengan baik, dapat dilihat daftar node ROS yang berjalan setelah menjalankan CoppeliaSim, dan pastikan bahwa `sim_ros_interface` node telah dimulai bersama dengan program CoppeliaSim. Untuk mengeksplorasi fungsionalitas RosInterface plugin, dapat digunakan skenario simulasi tertentu yang telah disediakan. Pada contoh tertentu, skenario `plugin_publisher_subscriber.ttt` menunjukkan robot dengan dua kamera yang mengakuisisi dan mempublikasikan aliran video ke topik ROS tertentu. Dengan memahami langkah-langkah tersebut, dapat dimulai penggunaan CoppeliaSim dengan ROS.

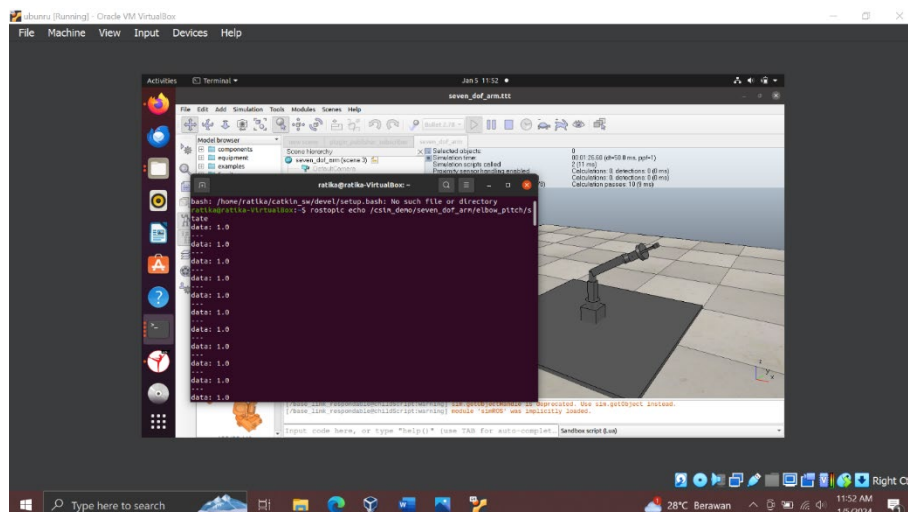


Setelahnya, simulasi dimulai, dan pada contoh tersebut, terdapat dua kamera pada robot. Kamera pasif menampilkan gambar yang dipublikasikan oleh kamera aktif, menerima data visi langsung dari kerangka kerja ROS. Untuk memvisualisasikan aliran video yang dipublikasikan oleh CoppeliaSim, dapat menggunakan paket `image_view` dengan menjalankan perintah `"roslaunch image_view image:=/camera/image_raw"`. Dengan demikian, kita dapat menjelajahi antarmuka antara CoppeliaSim dan ROS menggunakan plugin RosInterface.



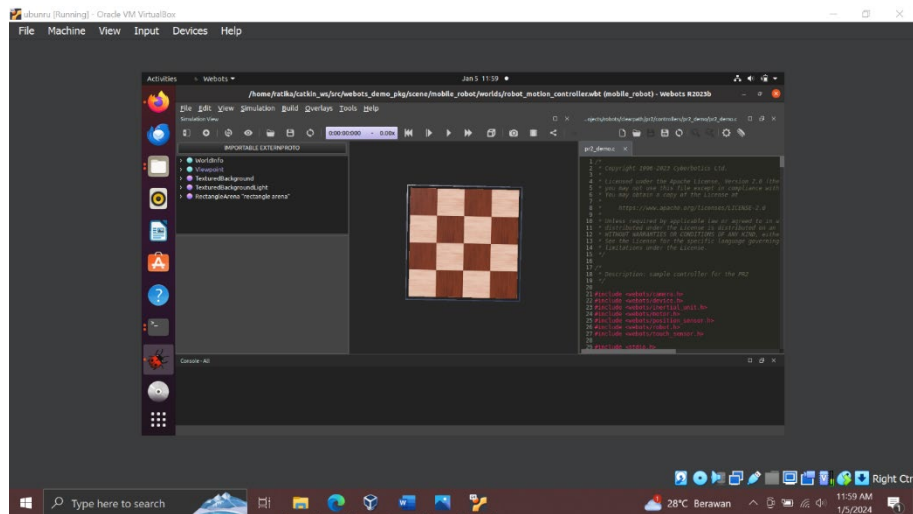
## 2. Simulating a robotic arm using CoppeliaSim and ROS

Pada bagian ini, pendekatan serupa dilakukan dengan menggunakan CoppeliaSim. Langkah awal untuk mensimulasikan lengan dengan tujuh DOF adalah mengimpor model tersebut ke dalam adegan simulasi. CoppeliaSim memungkinkan pengguna untuk mengimpor robot menggunakan file URDF; oleh karena itu, proses dimulai dengan mengonversi model xacro lengan menjadi file URDF dan menyimpannya di dalam folder urdf pada paket csim\_demo\_pkg. Setelah diimpor, pengguna dapat mengendalikan setiap sendi dengan mengaktifkan motor melalui panel Properti Objek Scene. Namun, agar robot dapat dikendalikan menggunakan ROS, diperlukan integrasi dengan plugin RosInterface. Dengan menggunakan plugin ini, robot dapat berfungsi sebagai node ROS yang berkomunikasi melalui topik. Selanjutnya, pembahasan akan fokus pada cara mengintegrasikan pengontrol robot dengan plugin RosInterface pada CoppeliaSim.



### 3. Simulating a mobile robot with Webots

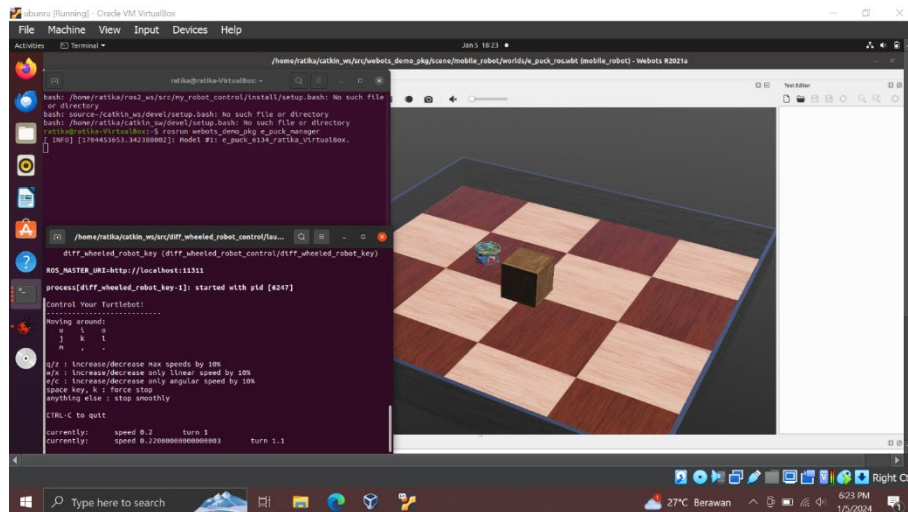
Pada bagian ini, tujuannya adalah membuat adegan simulasi dari awal yang berisi objek dan sebuah robot beroda mobile. Untuk mencapai ini, dibutuhkan pembuatan dunia simulasi kosong yang baru. Wizard dapat digunakan untuk membuat adegan simulasi baru. Dunia ini sudah tersedia dalam kode sumber buku pada paket `webots_demo_pkg`. Untuk membuat simulasi baru, pilih menu bar atas dan pilih Wizards | New Project Directory. Sebuah applet akan membantu dalam mengatur semuanya. Klik Next untuk memilih direktori proyek. Masukkan path folder sesuai keinginan, dan pilih `mobile_robot` sebagai nama folder. Juga pilih nama dunia; masukkan `robot_motion_controller.wbt` dan pastikan untuk memilih opsi Add a rectangle area. Lalu, klik Finish, dan setelah adegan dimuat, seharusnya tampak seperti yang tergambar pada tangkapan layar.



### 4. Writing a teleop node using webots\_ros

Pada bagian ini, diimplementasikan node ROS untuk mengontrol langsung kecepatan roda dari robot e-puck berdasarkan pesan `geometry_msgs::Twist` dengan memanfaatkan `webots_ros` sebagai dependensinya. Paket `webots_demo_pkg` telah dibuat dengan dependensi `roscpp`, `webots_ros`, dan `geometry_msgs`. Node ini memiliki dua panggilan kembali: satu untuk membaca kecepatan linear dan angular yang diinginkan dan menetapkan kecepatan roda, dan satu lagi untuk membaca nama model yang ditetapkan untuk robot e-puck. Dalam fungsi utama, dilakukan inisialisasi node ROS dan `NodeHandle`, menunggu model robot di-stream oleh Webots, mendefinisikan pelanggan untuk topik `/cmd_vel`, mengatur posisi roda menjadi `INFINITY`, mengatur kecepatan menjadi `0.0`, dan menerapkan kecepatan yang dihitung pada panggilan

geometry\_msgs::Twist. Dengan langkah-langkah ini, simulasi dapat dimulai dan robot mobile dapat dikendalikan menggunakan node teleop keyboard dari paket diff\_wheeled\_robot\_control yang dikembangkan sebelumnya. Simulasi dapat dimulai dengan menjalankan node e\_puck\_manager dan meluncurkan keyboard\_teleop.launch dari paket diff\_wheeled\_robot\_control. Dengan menggunakan keyboard, robot dapat dikendalikan di lingkungan simulasi Webots.



## **BAB III**

### **KESIMPULAN**

Pembahasan yang telah diuraikan dalam technical report ini memberikan gambaran mendalam tentang penerapan Robot Operating System (ROS) dalam pengembangan dan simulasi robotika. ROS telah menjadi kerangka kerja yang sangat penting dalam komunitas robotika karena menyediakan alat dan fungsionalitas yang kaya untuk memfasilitasi pengembangan, pengujian, dan simulasi robot secara efisien. Dalam konteks ini, kami membahas beberapa tahapan kunci dalam pengembangan robot menggunakan ROS, serta integrasinya dengan simulasi melalui alat seperti Gazebo, CoppeliaSim, dan Webots.

Salah satu langkah awal yang sangat penting adalah menjalankan ROS master dan parameter server sebelum memulai node ROS. ROS master bertindak sebagai pusat komunikasi di lingkungan ROS, sementara parameter server menyimpan konfigurasi parameter. Proses ini memberikan dasar yang kuat untuk mengaktifkan komunikasi antar node, yang merupakan inti dari pengembangan aplikasi robotika yang kompleks.

Pada tahap selanjutnya, fokus dialihkan ke pembuatan ROS package, yang merupakan unit dasar dari program-program ROS. Melalui langkah-langkah seperti inisialisasi ruang kerja catkin, pembuatan paket, dan penambahan dependensi, pengembang dapat mengorganisir proyek mereka dengan baik. Pembuatan dan pengaturan nodes dalam paket tersebut memungkinkan pengembang untuk memulai dan mengelola komunikasi antar node dengan mudah.

Dalam pengembangan model 3D robot, penggunaan Universal Robot Description Format (URDF) menjadi kunci. Dengan merancang dan memvisualisasikan model secara tiga dimensi, pengembang dapat memahami struktur robot dengan lebih baik. Integrasi sensor tambahan seperti Xtion Pro menambah dimensi fungsionalitas, memperkaya pengalaman simulasi.

Simulasi robotika menggunakan alat seperti Gazebo memungkinkan pengembang untuk menguji dan mengoptimalkan algoritma mereka sebelum mengimplementasikannya pada robot fisik. Melalui pembahasan simulasi robotic arm dan mobile robot, kami menunjukkan cara mengontrol joint pada robot simulasi, membuat file peluncuran (launch files), dan mengintegrasikan pengontrol dengan ROS.

CoppeliaSim menjadi fokus berikutnya, dengan penekanan pada integrasi plugin RosInterface. Melalui CoppeliaSim, pengembang dapat membuat model simulasi dan mengaktifkan komunikasi antara ROS dan simulator. Penerapan ini memungkinkan simulasi robot untuk berfungsi sebagai node ROS yang berkomunikasi melalui topik, membuka peluang untuk pengujian dan pengembangan yang lebih efektif. Webots, sebagai alat simulasi lainnya, memberikan pengembang kemampuan untuk membuat dunia simulasi dari awal. Dengan menciptakan dunia baru, pengembang dapat menyesuaikan kondisi simulasi sesuai kebutuhan proyek mereka. Dalam konteks Webots, pembahasan mencakup pembuatan node teleop dan kemampuan untuk mengontrol robot beroda mobile.