

UAS ROBOTIKA 2023/2024

Nama : NURDIN

NIM : 1103204006

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

Chapter 1: Introduction to ROS Programming Essentials

Persyaratan Teknis:

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

Pengantar ROS:

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

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

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

Mengapa Menggunakan ROS?

Kemampuan kelas atas:

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

Sangat dapat dikonfigurasi dengan berbagai parameter.

Banyak sekali alat:

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

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

Dukungan untuk sensor dan aktuator:

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

Antarmuka yang mulus dengan ROS, menghilangkan kerumitan.

Pengoperasian antar platform:

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

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

Modularitas:

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

Jika satu node rusak, sistem masih dapat berfungsi.

Penanganan sumber daya secara bersamaan:

ROS menyederhanakan penanganan sumber daya perangkat keras dengan beberapa proses.

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

Komunitas ROS:

Komunitas yang berkembang pesat dengan pengguna dan pengembang secara global.

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

Tingkat Sistem Berkas ROS:

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

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

Struktur Paket ROS:

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

Perintah-perintah ROS untuk Paket:

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

ROS Metapackages:

Paket khusus hanya dengan file package.xml.

Mengelompokkan beberapa paket terkait secara logis.

Pesan ROS:

Tipe data yang dideskripsikan menggunakan bahasa deskripsi pesan.

Contohnya termasuk int32, string, float32.

ROS menyediakan tipe pesan bawaan untuk aplikasi umum.

Layanan ROS:

Komunikasi permintaan/respon antara node ROS.

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

Grafik Komputasi ROS:

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

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

Master ROS memfasilitasi pendaftaran dan pencarian node.

Node ROS:

Proses dengan komputasi menggunakan pustaka klien ROS.

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

Diidentifikasi dengan nama seperti /camera_node.

Topik ROS:

Bus yang diberi nama yang memfasilitasi transportasi pesan antar node.

Node yang menerbitkan dan berlangganan dipisahkan.

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

ROS Logging:

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

Lapisan Grafik ROS:

Paket middleware komunikasi inti dalam tumpukan ros_comm.

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

Memahami Grafik Komputasi ROS:

Representasi grafis yang menunjukkan komunikasi antar node menggunakan topik.

Alat rqt_graph menghasilkan grafik tersebut.

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

Node ROS:

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

Penamaan Node:

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

Perintah ROSnode:

- a. Gunakan alat rosnode untuk mengumpulkan informasi:

roscpp info [nama_node]

roscpp kill [nama_node]

daftar roscpp

mesin roscpp [nama_mesin]

roscpp ping

pembersihan roscpp

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

Pesan adalah struktur data sederhana dengan tipe field.

Akses definisi pesan menggunakan alat rosmg.

rosmg show [message_type]

daftar rosmg

rosmg md5 [tipe_pesan]

rosmg package [nama_paket]

rosmg packages [package_1] [package_2]

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

Komunikasi searah menggunakan topik.

Gunakan alat rostopic untuk mengumpulkan informasi:

rostopic bw /topic

rostopic echo /topic

rostopic find /jenis_pesanan

rostopic hz /topic

rostopic info /topic

daftar rostopik

rostopic pub /topic message_type args

tipe rostopic /topic

d. Layanan ROS:

Komunikasi permintaan/respon.

Akses definisi layanan menggunakan alat rossrv dan rosservice.

ROS Bagfiles:

Gunakan perintah rosbag untuk merekam dan memutar data pesan ROS.

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

rosbag play [bag_name]

ROS Master:

Bertindak sebagai server DNS, mengasosiasikan nama dengan elemen ROS.

Node berkomunikasi dengan ROS Master menggunakan API berbasis XMLRPC.

Server Parameter ROS:

Translated with DeepL.com (free version)

Chapter 2: Getting Started with ROS Programming

Persyaratan Teknis:

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

```
chillin@ANITAMXWYN:~$ sudo apt-get install ros-noetic-urdf
[sudo] password for chillin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-urdf is already the newest version (1.13.2-1focal.20230620.185459).
ros-noetic-urdf set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 220 not upgraded.
chillin@ANITAMXWYN:~$ sudo apt-get install ros-noetic-xacro
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-xacro is already the newest version (1.14.16-1focal.20230620.185428).
ros-noetic-xacro set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 220 not upgraded.
```

Membuat Paket ROS:

- ROS packages adalah unit dasar dari program ROS.
- Dapat membuat, membangun, dan merilis paket ROS.
- Menggunakan sistem build catkin pada distribusi ROS Noetic.

Membuat Workspace Catkin:

- Buat workspace catkin dengan perintah **mkdir -p ~/catkin_ws/src**.
- Sumber lingkungan ROS perlu diaktifkan dengan perintah **source /opt/ros/noetic/setup.bash**.
- Inisialisasi workspace catkin dengan perintah **catkin_init_workspace**.

Membangun Workspace:

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

- Jalankan **catkin_make** untuk membangun workspace.

```
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws
taccoess@taccoess-VirtualBox: ~
taccoess@taccoess-VirtualBox: /opt/ros/noetic$ ls
bin      include  local_setup.sh  setup.bash      setup.zsh
env.sh   lib      local_setup.zsh setup.sh         share
etc      local_setup.bash  ros_ws          _setup_util.py
taccoess@taccoess-VirtualBox: /opt/ros/noetic$ cd ros_ws
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws$ ls
src
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws$ sudo mkdir src
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws$ ls
src
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws$ cd ..
taccoess@taccoess-VirtualBox: /opt/ros/noetic$ sudo chmod 777 ros_ws -R
taccoess@taccoess-VirtualBox: /opt/ros/noetic$ cd ros_ws
taccoess@taccoess-VirtualBox: /opt/ros/noetic/ros_ws$ catkin_make
Base path: /opt/ros/noetic/ros_ws
Source space: /opt/ros/noetic/ros_ws/src
Build space: /opt/ros/noetic/ros_ws/build
Devel space: /opt/ros/noetic/ros_ws/devel
Install space: /opt/ros/noetic/ros_ws/install
Creating symlink "/opt/ros/noetic/ros_ws/src/CMakeLists.txt" pointing to "/opt/
ros/noetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /opt/ros/noetic/ros_ws/src -DCATKIN_DEVEL_PREFIX=/
opt/ros/noetic/ros_ws/devel -DCMAKE_INSTALL_PREFIX=/opt/ros/noetic/ros_ws/insta
ll -G Unix Makefiles" in "/opt/ros/noetic/ros_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
```

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

Membuat Paket ROS:

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

Penggunaan roscore.xml:

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

Memahami Output roscore:

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

Pengerjaan ROS Nodes:

- Gunakan perintah **catkin_make** untuk membangun paket setelah membuatnya.

- Tambahkan ROS nodes ke folder src dalam paket.

Mengerjakan ROS Topics:

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

Node demo_topic_publisher.cpp:

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

Node Publisher (demo_topic_publisher.cpp):

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

Node Subscriber (demo_topic_subscriber.cpp):

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

Membangun Nodes:

- Mengedit file **CMakeLists.txt** di dalam paket untuk membangun dan mengompilasi kode sumber.
- Menggunakan perintah **catkin_make** di dalam workspace.
- Sekarang, jalankan kedua perintah dalam dua shell. Di penerbit yang sedang berjalan, jalankan yang berikut ini memerintah:

`roslaunch mastering_ros_demo_package demo_topic_publisher`

```

chillin@ANITAMXWYN: ~/catkin_ws 65x32
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- 
-- traversing 1 packages in topological order:
--   - mastering_ros_demo_pkg
-- 
-- +++ processing catkin package: 'mastering_ros_demo_pkg'
-- ==> add_subdirectory(mastering_ros_demo_pkg)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- Configuring done
-- Generating done
-- Build files have been written to: /home/chillin/catkin_ws/build
####
#### Running command: "make -j12 -l12" in "/home/chillin/catkin_ws/build"
####
Scanning dependencies of target demo_topic_publisher
Scanning dependencies of target demo_topic_subscriber
[ 25%] Building CXX object mastering_ros_demo_pkg/CMakeFiles/demo_topic_subscriber.dir/src/demo_topic_subscriber.cpp.o
[ 50%] Building CXX object mastering_ros_demo_pkg/CMakeFiles/demo_topic_publisher.dir/src/demo_topic_publisher.cpp.o
[ 75%] Linking CXX executable /home/chillin/catkin_ws/devel/lib/mastering_ros_demo_pkg/demo_topic_publisher
[100%] Linking CXX executable /home/chillin/catkin_ws/devel/lib/mastering_ros_demo_pkg/demo_topic_subscriber
[100%] Built target demo_topic_publisher
[100%] Built target demo_topic_subscriber
chillin@ANITAMXWYN:~/catkin_ws$ 

```

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

`roslaunch mastering_ros_demo_pkg demo_msg_publisher`

```
chillin@ANITAMXWYN: ~/catkin_ws
roscore http://ANITAMXWYN:11311/
[ INFO] [1704397716.684311157]: hello world
[ INFO] [1704397716.785188380]: 615
[ INFO] [1704397716.785329958]: hello world
[ INFO] [1704397716.885163784]: 616
[ INFO] [1704397716.885304813]: hello world
[ INFO] [1704397716.985337091]: 617
[ INFO] [1704397716.983629208]: hello world
[ INFO] [1704397717.085164923]: 618
[ INFO] [1704397717.085262854]: hello world
[ INFO] [1704397717.183510347]: 619
[ INFO] [1704397717.183643712]: hello world
^C[ INFO] [1704397717.283783001]: 620
[ INFO] [1704397717.283924872]: hello world
chillin@ANITAMXWYN:~/catkin_ws$ roslaunch mastering_ros_demo_pkg demo_msg_publisher
[ INFO] [1704397731.025316846]: 0
[ INFO] [1704397731.026673795]: hello world
[ INFO] [1704397731.132954798]: 1
[ INFO] [1704397731.133091899]: hello world
[ INFO] [1704397731.230372079]: 2
[ INFO] [1704397731.230563650]: hello world
[ INFO] [1704397731.328009417]: 3
[ INFO] [1704397731.328178558]: hello world
[ INFO] [1704397731.426593966]: 4
[ INFO] [1704397731.426947431]: hello world
[ INFO] [1704397731.528702458]: 5
[ INFO] [1704397731.528848612]: hello world
[ INFO] [1704397731.635628264]: 6
[ INFO] [1704397731.635770660]: hello world
[ INFO] [1704397731.726194527]: 7
[ INFO] [1704397731.726436116]: hello world
[ INFO] [1704397731.826709131]: 8
[ INFO] [1704397731.826858559]: hello world
[ INFO] [1704397731.928074342]: 9
```

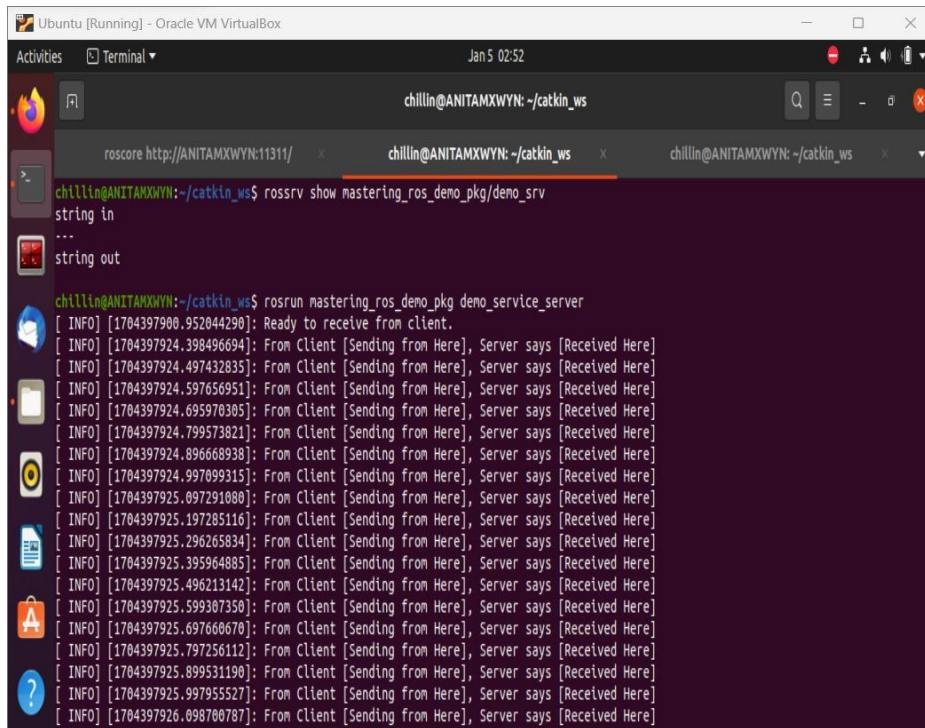
Di pelanggan yang sedang berjalan, jalankan perintah berikut:

The screenshot shows a terminal window titled "chillin@ANITAMXWYN: ~/catkin_ws". The terminal displays a series of log messages from the ROS master, indicating that it has received "greeting [hello world]" messages from various IP addresses. The messages are timestamped and include the IP address, the word "Recieved", and the message content. The terminal also shows the command `roslaunch mastering_ros_demo_pkg demo_srv` being executed, which results in the same log messages being displayed. The terminal window is part of a desktop environment with a sidebar on the left containing various application icons.

```
chillin@ANITAMXWYN: ~/catkin_ws
roslaunch mastering_ros_demo_pkg demo_srv
[INFO] [1704397714.485431670]: Recieved [592]
[INFO] [1704397714.586991684]: Recieved greeting [hello world ]
[INFO] [1704397714.587109887]: Recieved [593]
[INFO] [1704397714.684375375]: Recieved greeting [hello world ]
[INFO] [1704397714.684543841]: Recieved [594]
[INFO] [1704397714.790626798]: Recieved greeting [hello world ]
[INFO] [1704397714.791499792]: Recieved [595]
[INFO] [1704397714.885951824]: Recieved greeting [hello world ]
[INFO] [1704397714.886080775]: Recieved [596]
[INFO] [1704397714.984388772]: Recieved greeting [hello world ]
[INFO] [1704397714.984520302]: Recieved [597]
[INFO] [1704397715.087540425]: Recieved greeting [hello world ]
[INFO] [1704397715.087719003]: Recieved [598]
^Cchillin@ANITAMXWYN: ~/catkin_ws$ roslaunch mastering_ros_demo_pkg demo_srv
[INFO] [1704397739.927023565]: Recieved greeting [hello world ]
[INFO] [1704397739.929218632]: Recieved [89]
[INFO] [1704397740.038642894]: Recieved greeting [hello world ]
[INFO] [1704397740.038723888]: Recieved [90]
[INFO] [1704397740.131908703]: Recieved greeting [hello world ]
[INFO] [1704397740.132035461]: Recieved [91]
[INFO] [1704397740.231299481]: Recieved greeting [hello world ]
[INFO] [1704397740.231495445]: Recieved [92]
[INFO] [1704397740.345488471]: Recieved greeting [hello world ]
[INFO] [1704397740.345565788]: Recieved [93]
[INFO] [1704397740.429463517]: Recieved greeting [hello world ]
[INFO] [1704397740.429559851]: Recieved [94]
[INFO] [1704397740.532202574]: Recieved greeting [hello world ]
[INFO] [1704397740.532277026]: Recieved [95]
[INFO] [1704397740.628570396]: Recieved greeting [hello world ]
[INFO] [1704397740.628671261]: Recieved [96]
[INFO] [1704397740.726704898]: Recieved greeting [hello world ]
[INFO] [1704397740.726777294]: Recieved [97]
[INFO] [1704397740.834593171]: Recieved greeting [hello world ]
```

Setelah melakukan perubahan ini, kita dapat membangun paket menggunakan `catkin_make`. Kemudian, menggunakan perintah berikut, kita dapat memverifikasi prosedurnya:

`rossrv` tampilkan `mastering_ros_demo_pkg/demo_srv`



The screenshot shows a terminal window titled "Ubuntu [Running] - Oracle VM VirtualBox" with a timestamp of "Jan 5 02:52". The user is logged in as "chillin@ANITAMXWYN" in the directory "~/catkin_ws". The terminal shows the following commands and output:

```
chillin@ANITAMXWYN:~/catkin_ws$ roscore http://ANITAMXWYN:11311/
chillin@ANITAMXWYN:~/catkin_ws$ rossrv show mastering_ros_demo_pkg/demo_srv
string in
...
string out

chillin@ANITAMXWYN:~/catkin_ws$ rosrn mastering_ros_demo_pkg demo_service_server
[ INFO] [1704397900.952044290]: Ready to receive from client.
[ INFO] [1704397924.398496694]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.497432835]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.597656951]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.695970305]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.799573821]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.896668938]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397924.997099315]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.097291080]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.197285116]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.296265834]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.395964885]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.496213142]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.599307350]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.697660670]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.797256112]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.899531190]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397925.997955527]: From Client [Sending from Here], Server says [Received Here]
[ INFO] [1704397926.098700787]: From Client [Sending from Here], Server says [Received Here]
```

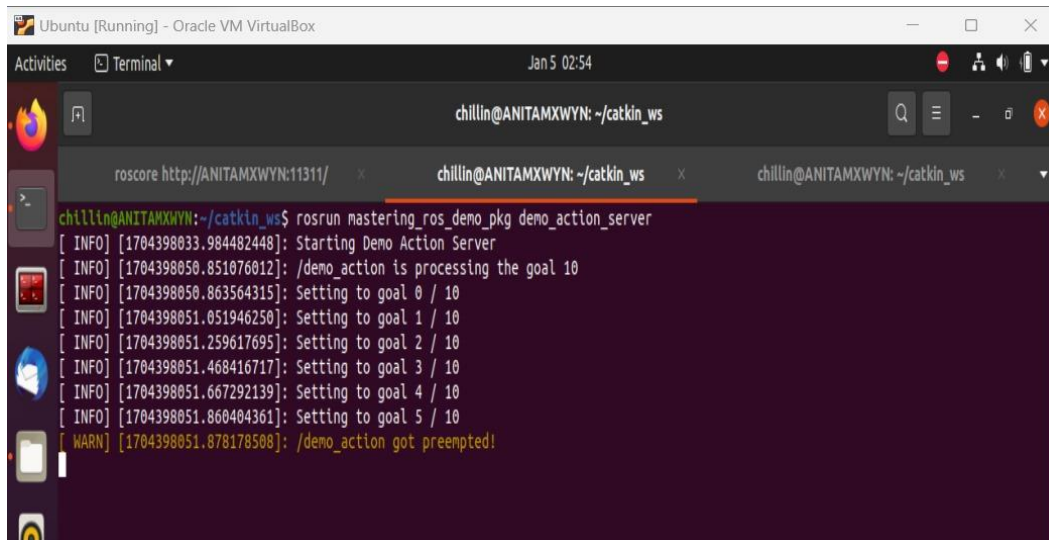
Komunikasi antara Nodes:

- Jalankan **roscore** untuk memulai ROS.
- Gunakan **rosrn** untuk menjalankan publisher dan subscriber nodes secara terpisah.
- Graph menunjukkan bagaimana **demo_topic_publisher** mempublikasikan topik **"/numbers"** yang diambil oleh **demo_topic_subscriber**.

Membangun Server Dan Klien Tindakan ROS

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

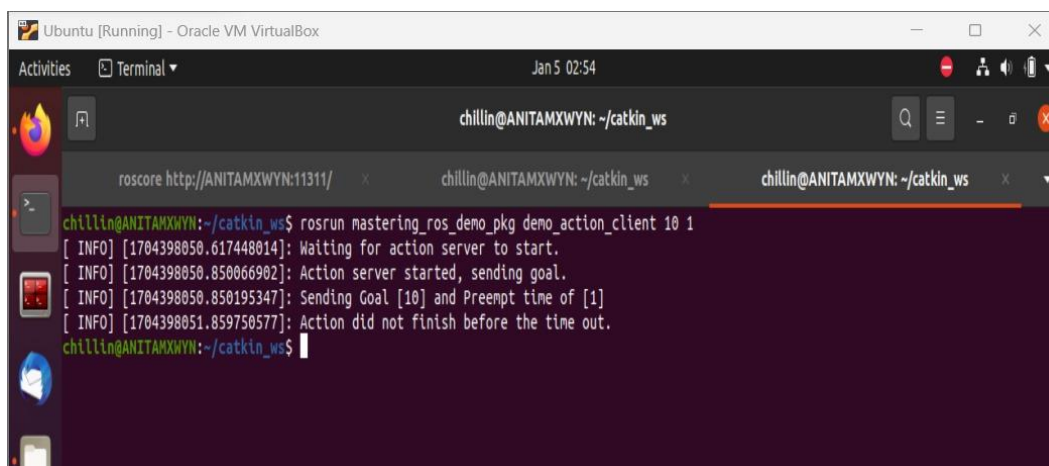
1. Jalankan skor ros:
Roscore
2. Luncurkan node server tindakan:
rosrn mastering_ros_demo_pkg demo_action_server



```
chillin@ANITAMXWYN: ~/catkin_ws
roscore http://ANITAMXWYN:11311/
chillin@ANITAMXWYN: ~/catkin_ws
chillin@ANITAMXWYN: ~/catkin_ws$ rosrn mastering_ros_demo_pkg demo_action_server
[ INFO] [1704398033.984482448]: Starting Demo Action Server
[ INFO] [1704398050.851076012]: /demo_action is processing the goal 10
[ INFO] [1704398050.863564315]: Setting to goal 0 / 10
[ INFO] [1704398051.051946250]: Setting to goal 1 / 10
[ INFO] [1704398051.259617695]: Setting to goal 2 / 10
[ INFO] [1704398051.468416717]: Setting to goal 3 / 10
[ INFO] [1704398051.667292139]: Setting to goal 4 / 10
[ INFO] [1704398051.860404361]: Setting to goal 5 / 10
[ WARN] [1704398051.878178508]: /demo_action got preempted!
```

3. Luncurkan simpul klien tindakan:

`roslaunch mastering_ros_demo_pkg demo_action_client 10 1`



```
chillin@ANITAMXWYN: ~/catkin_ws
roscore http://ANITAMXWYN:11311/
chillin@ANITAMXWYN: ~/catkin_ws
chillin@ANITAMXWYN: ~/catkin_ws$ roslaunch mastering_ros_demo_pkg demo_action_client 10 1
[ INFO] [1704398050.617448014]: Waiting for action server to start.
[ INFO] [1704398050.85066902]: Action server started, sending goal.
[ INFO] [1704398050.850195347]: Sending Goal [10] and Preempt time of [1]
[ INFO] [1704398051.859750577]: Action did not finish before the time out.
chillin@ANITAMXWYN: ~/catkin_ws$
```

Membuat File Peluncuran

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

`roslaunch mastering_ros_demo_pkg demo_topic.launch`


```
untu [Running] - Oracle VM VirtualBox
es Terminal Jan 5 02:55

/home/chillin/catkin_ws/src/mastering_ros_demo_pkg/launch/demo_topic.launch http://localhost:11311

roscore http://ANITAMXWYN:11311/ /home/chillin/catkin_ws/src/mastering_ros... chillin@ANITAMXWYN: ~/catkin_ws

[roslaunch] Couldn't find executable named demo_topic.launch below /home/chillin/catkin_ws/src/mastering_ros_demo_pkg
[roslaunch] Found the following, but they're either not files,
[roslaunch] or not executable:
[roslaunch] /home/chillin/catkin_ws/src/mastering_ros_demo_pkg/launch/demo_topic.launch
chillin@ANITAMXWYN:~/catkin_ws$ roslaunch mastering_ros_demo_pkg demo_topic.launch
... logging to /home/chillin/.ros/log/055e8b04-ab3a-11ee-b4cf-d34c19a3c743/roslaunch-ANITAMXWYN-7629.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ANITAMXWYN:37993/

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.16.0

NODES
/
  publisher_node (mastering_ros_demo_pkg/demo_topic_publisher)
  subscriber_node (mastering_ros_demo_pkg/demo_topic_subscriber)

ROS_MASTER_URI=http://localhost:11311

process[publisher_node-1]: started with pid [7643]
process[subscriber_node-2]: started with pid [7644]
[ INFO] [1704398152.672614568]: 0
[ INFO] [1704398152.908019111]: 1
[ INFO] [1704398152.920650865]: 2
[ INFO] [1704398152.937706347]: 3
```

Pemecahan Masalah dan Pemahaman Nodes:

- Gunakan **roslaunch** **list** dan **roslaunch** **info** untuk mendapatkan informasi tentang nodes aktif.
- Gunakan **rostopic** **echo** dan **rostopic** **type** untuk memahami nilai dan jenis pesan yang dikirim melalui topik `"/numbers"`.

Pesan dan Layanan Khusus:

- Membuat pesan khusus dalam file **.msg** dan menyimpannya dalam folder **msg**.

- Mengedit **package.xml** dan **CMakeLists.txt** untuk mengakui dan mengonfigurasi pesan khusus.
- Menggunakan **rosmmsg show** untuk memeriksa definisi pesan.

Penggunaan Layanan Khusus:

- Membuat file **.srv** untuk mendefinisikan layanan khusus dengan Request dan Response.
- Mengonfigurasi **package.xml** dan **CMakeLists.txt** untuk mendukung layanan khusus.
- Menggunakan **rossrv show** untuk memeriksa definisi layanan khusus.

Membangun Nodes dengan Pesan dan Layanan Khusus:

- Mengedit **CMakeLists.txt** untuk menyertakan dependensi pada pesan dan layanan yang dihasilkan.
- Membangun dan menjalankan nodes dengan pesan dan layanan khusus.

Catatan: Pastikan ROS sudah terinstal dengan benar untuk menjalankan perintah-perintah ROS.

Secara ringkas, bab ini memperkenalkan dan mendemonstrasikan penggunaan fitur-fitur ROS seperti topik, layanan, actionlib, dan file peluncuran. Berikut ini adalah poin-poin utamanya:

- Topik:

Digunakan untuk mengalirkan aliran data yang berkelanjutan, seperti data sensor.

Contoh: Teleoperasi menggunakan data joypad, menerbitkan odometri robot, atau streaming video dari kamera.

- Layanan:

Digunakan untuk menjalankan prosedur yang diakhiri dengan cepat.

Contoh: Menyimpan parameter kalibrasi sensor, menyimpan peta robot yang dihasilkan, atau memuat file parameter.

- Actionlib:

Digunakan untuk mengeksekusi tindakan yang panjang dan kompleks sambil mengelola umpan balik.

Contoh: Menavigasi ke arah target, merencanakan jalur gerak.

- File Peluncuran:

File berbasis XML untuk meluncurkan beberapa node ROS secara bersamaan.

Menyederhanakan proses memulai node dan dapat digunakan untuk mengotomatiskan peluncuran beberapa node.

- Demonstrasi:

Memberikan contoh kode untuk topik (node penerbit dan pelanggan), layanan (node server dan klien), dan actionlib (server aksi dan node klien aksi).

Menunjukkan cara membuat file peluncuran untuk meluncurkan beberapa node dengan satu perintah.

Fitur-fitur ROS ini sangat penting untuk mengembangkan aplikasi robotik, yang memungkinkan komunikasi dan koordinasi yang efisien di antara berbagai komponen sistem robotik. Memahami kapan harus menggunakan topik, layanan, atau actionlib berdasarkan sifat tugas sangat penting untuk pemrograman ROS yang efektif. Kode sumber yang disediakan dapat dikloning dari repositori GitHub untuk praktik langsung.

Chapter 3: Working with ROS for 3D Modeling

Persyaratan Teknis:

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

Kode tersebut terdapat di dalam folder Chapter3/mastering_ros_robot_description_pkg/.

Paket-paket ROS untuk Pemodelan Robot:

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

Memahami Pemodelan Robot menggunakan URDF:

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

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

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

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

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

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

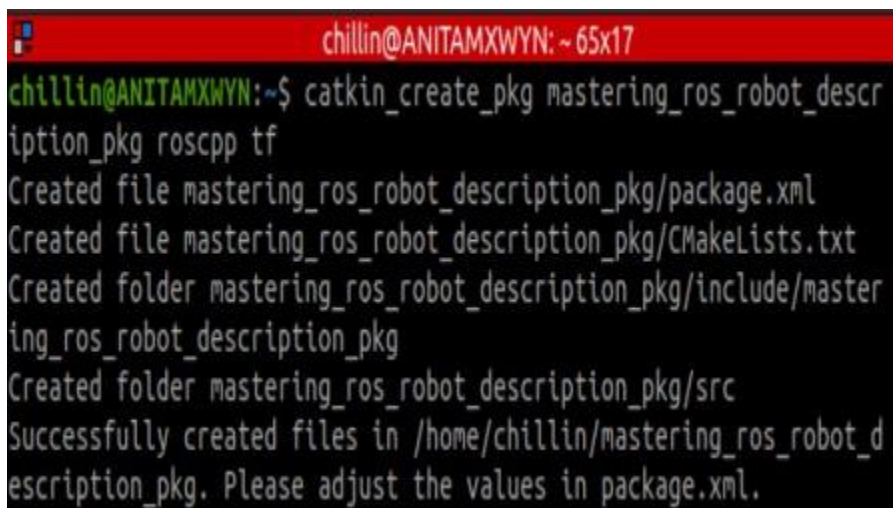
Langkah selanjutnya:

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

Membuat Paket ROS untuk Robot:

- Sebelum membuat file URDF untuk robot, buatlah paket ROS dalam workspace catkin menggunakan perintah berikut:

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

A terminal window with a red title bar showing the command 'catkin_create_pkg mastering_ros_robot_description_pkg roscpp tf geometry_msgs urdf rviz xacro' being executed. The output shows the creation of package.xml, CMakeLists.txt, and the necessary include and src folders, followed by a success message.

```
chillin@ANITAMXWYN: ~ 65x17  
chillin@ANITAMXWYN:~$ catkin_create_pkg mastering_ros_robot_desc  
ription_pkg roscpp tf  
Created file mastering_ros_robot_description_pkg/package.xml  
Created file mastering_ros_robot_description_pkg/CMakeLists.txt  
Created folder mastering_ros_robot_description_pkg/include/master  
ing_ros_robot_description_pkg  
Created folder mastering_ros_robot_description_pkg/src  
Successfully created files in /home/chillin/mastering_ros_robot_d  
escription_pkg. Please adjust the values in package.xml.
```

Ketergantungan Paket:

- Paket terutama bergantung pada paket urdf dan xacro. Jika belum terpasang, dapat diinstal menggunakan manajer paket:

```
sudo apt-get install ros-noetic-urdf sudo apt-get install ros-noetic-xacro
```

Struktur Folder:

- Sebelum membuat file URDF, buat tiga folder di dalam folder paket, yaitu urdf, meshes, dan launch.
- Folder urdf untuk menyimpan file URDF dan xacro yang akan dibuat.
- Folder meshes untuk menyimpan mesh yang akan dimasukkan dalam file URDF.
- Folder launch untuk menyimpan file peluncuran ROS.

Model URDF Pertama:

- Setelah memahami tag-tag penting dalam URDF, kita dapat mulai membuat model dasar menggunakan URDF.
- Model yang pertama kali akan kita desain adalah mekanisme pan-and-tilt dengan tiga link dan dua joint tipe revolute.

Penggunaan Xacro:

- Xacro digunakan untuk menyederhanakan URDF dengan membuat makro dan menggunakan properti.
- Properti seperti panjang dan jari-jari link dapat didefinisikan sebagai konstanta dan digunakan di seluruh kode.
- Ekspresi matematika dapat digunakan untuk menghitung nilai properti, misalnya, $\${pan_link_radius+0.02}$.

Menambahkan Properti Fisik dan Kollision:

- Sebelum mensimulasikan robot, kita perlu mendefinisikan properti fisik dan kollision untuk setiap link.
- Properti seperti geometri, warna, massa, dan inersia link perlu didefinisikan dengan hati-hati untuk mendapatkan simulasi robot yang baik.

Menggunakan Macro:

- Dalam xacro, makro digunakan untuk menyusun potongan kode yang dapat digunakan kembali.
- Ini membantu dalam membuat kode lebih sederhana, lebih mudah dibaca, dan lebih mudah dimodifikasi.

Menggunakan Makro:

Pengantar ke Fitur xacro:

- xacro mendukung makro.

- Makro membantu mengurangi panjang definisi yang rumit.
- Contoh makro: `inertial_matrix` dengan parameter massa.

Mendefinisikan Makro `inertial_matrix`:

- Makro bernama `inertial_matrix` dengan parameter massa.
- Parameter massa digunakan di dalam definisi inersia menggunakan `#{mass}`.
- Contoh penggunaan: `<xacro:inertial_matrix mass = "1" />`.

Mengkonversi `xacro` ke URDF:

- Gunakan perintah `roscpp xacro pan_tilt.xacro > pan_tilt_generated.urdf`.
- Dalam file peluncuran ROS: `<param name="robot_description" command="$(find xacro)/xacro $(find mastering_ros_robot_description_pkg)/urdf/pan_tilt.xacro" />`.
- Visualisasikan `xacro` dengan `roslaunch mastering_ros_robot_description_pkg view_pan_tilt_xacro.launch`.

Membuat Manipulator Robot Tujuh-DOF:

- Tentukan lengan robot tujuh-DOF menggunakan URDF dan `xacro`.
- Karakteristik: 7 DOF, panjang lengan 50 cm, jangkauan lengan 35 cm, 12 tautan, 11 sendi.
- Jenis sendi: 1 sendi tetap, 7 sendi berputar, 2 sendi prismatic untuk gripper.

Menjelaskan Model `xacro`:

- Tentukan konstanta untuk konversi derajat ke radian, nilai PI, dan dimensi link.
- Gunakan makro untuk matriks inersia dan definisi blok transmisi.
- Sertakan file `xacro` lain untuk definisi sensor.
- Gunakan mesh untuk representasi visual.

Melihat Lengan Tujuh-DOF di RViz:

- Luncurkan RViz dengan model lengan menggunakan file `view_arm.launch`.
- Memanfaatkan `joint_state_publisher` dan `robot_state_publisher` untuk interaksi dan visualisasi.

Memahami Joint State Publisher:

- Penerbit keadaan bersama berinteraksi dengan setiap sendi robot.
- Menerbitkan nilai status gabungan menggunakan format pesan `sensor_msgs/JointState`.
- Digunakan untuk menguji gerakan sendi.

Memahami Penerbit Status Robot:

- Penerbit status robot menerbitkan status robot ke `tf`.
- Berlangganan ke status gabungan, menerbitkan pose 3D menggunakan representasi kinematik.
- Diimplementasikan menggunakan node `robot_state_publisher`.

Membuat Robot Bergerak Beroda Diferensial:

- Mendefinisikan robot beroda diferensial dengan dua roda utama dan roda kastor.
- Gunakan file `xacro diff_wheeled_robot.xacro`.
- Sertakan definisi roda dari `wheel.urdf.xacro`.

Melihat Mobile Robot di RViz:

- Luncurkan RViz dengan model robot bergerak menggunakan file `view_mobile_robot.launch`.
- Berinteraksi dengan slider sendi untuk mensimulasikan gerakan.

Chapter 4: Simulating Robots Using ROS and Gazebo

Berikut ini adalah perincian yang disederhanakan dari teks yang diberikan:

Persyaratan Teknis:

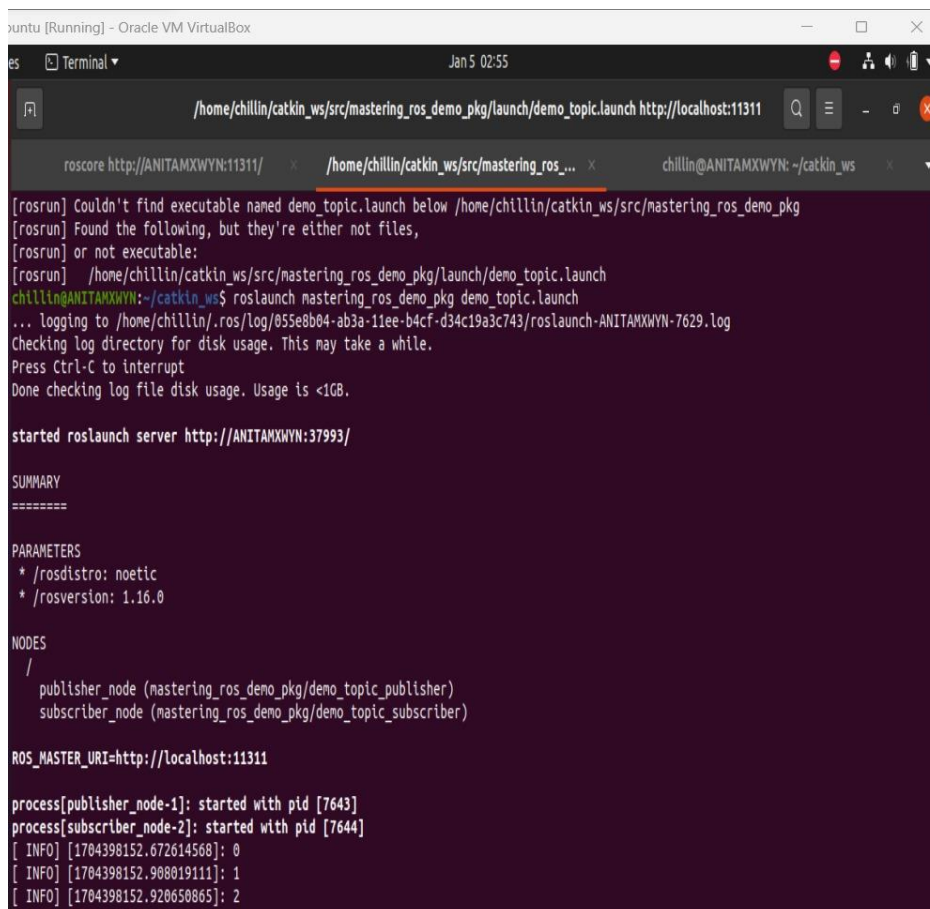
- Laptop standar dengan Ubuntu 20.04 dan ROS Noetic.
- Kode tersedia di Git: tautan.
- Model simulasi dalam folder Bab4/seven_dof_arm_gazebo.
- Lihat kode yang sedang bekerja: tautan.

Simulasi Lengan Robot di Gazebo dan ROS:

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

perintah

berikut:



```
ubuntu [Running] - Oracle VM VirtualBox
Terminal Jan 5 02:55
/home/chillin/catkin_ws/src/mastering_ros_demo_pkg/launch/demo_topic.launch http://localhost:11311
roscore http://ANITAMXWYN:11311/ /home/chillin/catkin_ws/src/mastering_ros_demo_pkg
[roslaunch] Couldn't find executable named demo_topic.launch below /home/chillin/catkin_ws/src/mastering_ros_demo_pkg
[roslaunch] Found the following, but they're either not files,
[roslaunch] or not executable:
[roslaunch] /home/chillin/catkin_ws/src/mastering_ros_demo_pkg/launch/demo_topic.launch
chillin@ANITAMXWYN:~/catkin_ws$ roslaunch mastering_ros_demo_pkg demo_topic.launch
... logging to /home/chillin/.ros/log/855e8b04-ab3a-11ee-b4cf-d34c19a3c743/roslaunch-ANITAMXWYN-7629.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ANITAMXWYN:37993/

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.16.0

NODES
/
  publisher_node (mastering_ros_demo_pkg/demo_topic_publisher)
  subscriber_node (mastering_ros_demo_pkg/demo_topic_subscriber)

ROS_MASTER_URI=http://localhost:11311

process[publisher_node-1]: started with pid [7643]
process[subscriber_node-2]: started with pid [7644]
[ INFO] [1704398152.672614568]: 0
[ INFO] [1704398152.908019111]: 1
[ INFO] [1704398152.920650865]: 2
[ INFO] [1704398152.937665173]: 3
```

Membuat Model Simulasi Lengan Robot untuk Gazebo:

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

Menambahkan Warna dan Tekstur pada Model Robot Gazebo:

- Tentukan warna dan tekstur dalam file `.xacro`.

Menambahkan Tag Transmisi untuk Menggerakkan Model:

- Tentukan elemen transmisi untuk menghubungkan aktuator ke sendi.

Menambahkan Plugin Gazebo_ros_control:

- Tambahkan plugin `gazebo_ros_control` untuk mengurai tag transmisi.

Menambahkan Sensor Visi 3D ke Gazebo:

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

Mensimulasikan Lengan Robot dengan Xtion Pro:

- Luncurkan simulasi lengkap dengan sensor Xtion Pro.

Memvisualisasikan Data Sensor 3D:

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

Menggerakkan Sendi Robot menggunakan Pengontrol ROS di Gazebo:

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

Menghubungkan Pengontrol Status Gabungan dan Pengontrol Posisi Gabungan:

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

Meluncurkan pengendali ROS dengan Gazebo:

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

Memeriksa Topik Pengontrol:

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

Menggerakkan Sendi Robot:

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

Mensimulasikan Robot Beroda Diferensial di Gazebo:

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

Menambahkan Pemindai Laser ke Gazebo:

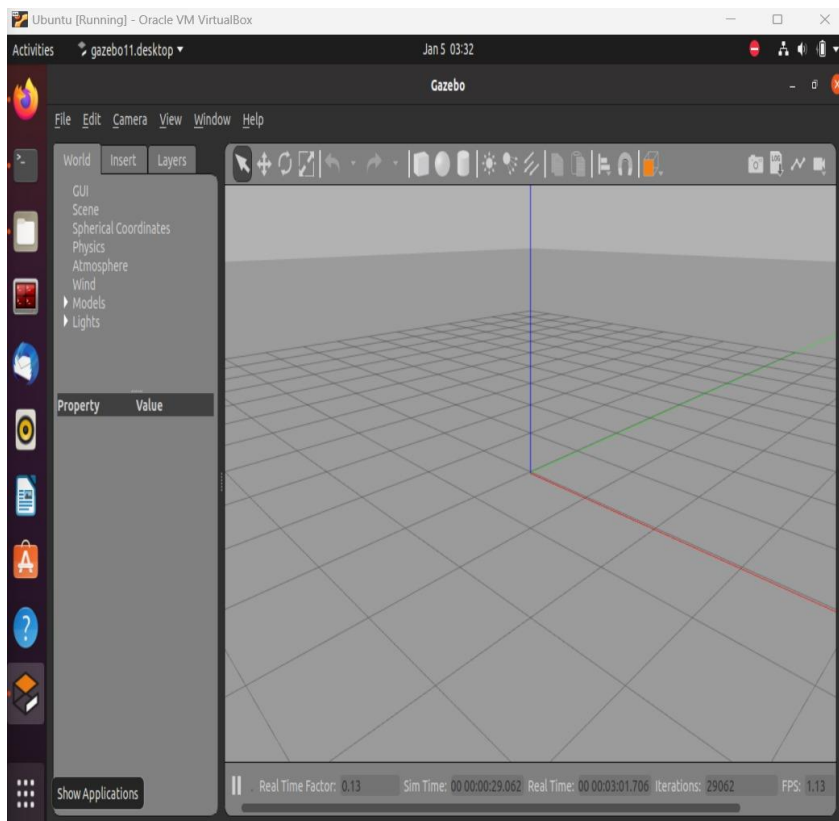
- Ubah `diff_wheeled_robot.xacro` untuk menyertakan pemindai laser.
- Konfigurasi informasi khusus Gazebo untuk plugin pemindai laser.
- Memvisualisasikan data pemindai laser dengan objek yang ditambahkan di Gazebo.

Memindahkan Robot Bergerak di Gazebo:

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

Node Teleop ROS:

- Gunakan node diff_wheeled_robot_key untuk teleoperasi.
- Sesuaikan skala linier dan sudut.
- Luncurkan teleop dengan roslaunch diff_wheeled_robot_control keyboard_teleop.launch.



Visualisasi di RViz:

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

Memindahkan Robot:

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

Menjelajahi Area:

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

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

Persyaratan Teknis:

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

Menyiapkan CoppeliaSim dengan ROS:

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

Mode CoppeliaSim untuk Robot Simulasi:

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

Memulai CoppeliaSim dengan ROS:

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

Memeriksa Pengaturan:

- Verifikasi node ROS yang aktif setelah meluncurkan CoppeliaSim.

Berinteraksi dengan CoppeliaSim menggunakan Topik ROS:

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

Memahami Plugin RosInterface:

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

Berinteraksi dengan CoppeliaSim menggunakan Topik ROS (lanjutan):

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

Bekerja dengan Pesan ROS:

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

Mensimulasikan Lengan Robotik menggunakan CoppeliaSim dan ROS:

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

Menambahkan Antarmuka ROS ke Pengontrol Bersama CoppeliaSim:

- Antarmuka lengan tujuh-DOF dengan plugin RosInterface untuk kontrol bersama.
- Gunakan skrip Lua untuk mempublikasikan status gabungan dan berlangganan perintah gabungan melalui topik ROS.

- Contoh: sysCall_init menginisialisasi penanganan sambungan dan mengatur penerbit/pelanggan.
- Mengontrol sambungan menggunakan perintah ROS, misalnya, rostopic pub /csim_demo/seven_dof_arm/elbow_pitch/cmd std_msgs/Float32 "data: 1.0".
- Memantau status sambungan, misalnya, rostopic echo /csim_demo/seven_dof_arm/elbow_pitch/state.

Membuat Webots dengan ROS

1. **Instalasi Webots:** Unduh Webots dari situs web resmi (<http://www.cyberbotics.com/#download>) atau gunakan Debian/Ubuntu APT package manager dengan langkah-langkah berikut:


```
wget -qO- https://cyberbotics.com/Cyberbotics.asc | sudo apt-key add - sudo apt-add-repository 'deb https://cyberbotics.com/debian/ binary-amd64/' sudo apt-get update sudo apt-get install webots
```
2. **Mulai Webots:** Ketikkan perintah berikut untuk membuka antarmuka pengguna Webots:


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

Simulasi Robot Bergerak dengan Webots

1. **Membuat Adegan Simulasi:** Gunakan wizard untuk membuat adegan simulasi baru dengan memilih Wizards | New Project Directory dan mengonfigurasi direktori dan nama proyek.

2. Menambahkan Objek ke Adegan:

- Pilih RectangleArea dari panel hirarki.
- Klik tombol + untuk menambahkan objek dan pilih PROTO nodes | objects | factory | containers | WoodenBox (Solid).
- Pilih RectangleArea dan tambahkan robot e-puck dengan memilih (Webots Projects) / robots / gctronic / e-puck / E-puck PROTO.

3. Konfigurasi Objek:

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

Menulis Kontroler Pertama Anda

1. Buat Kontroler Baru:

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

2. Implementasikan Kontroler:

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

Simulasi Lengan Robot dengan Webots dan ROS

1. Instalasi Webots-ROS Package:

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

2. Mengganti Kontroler Webots:

- Ganti kontroler robot Webots dengan kontroler ROS.

Menulis Node Teleop Menggunakan webots_ros

1. Buat Node ROS:

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

Memulai Webots dengan Berkas Peluncuran

1. Berkas Peluncuran Webots:

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

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