UAS ROBOTIKA 2023/2024

Nama : Mochamad Phillia Wibowo

NIM : 1103204191

Section 1 – ROS Programming Essentials

Chapter 1 – Introduction to ROS

- Technical Requirements

  Untuk mengikuti chapter ini, satu-satunya yang Anda perlukan adalah komputer standar yang menjalankan Ubuntu 20.04 LTS atau distribusi Debian 10 GNU/Linux.

- ROS Distribution

  Pembaruan ROS baru-baru ini telah dikeluarkan dengan distribusi ROS terbaru. Distribusi ini mencakup versi terkini dari inti perangkat lunak, serta satu set ROS yang baru atau diperbarui. Siklus rilis ROS mengikuti pola yang sama dengan distribusi Ubuntu Linux, di mana versi ROS terbaru dirilis setiap 6 bulan. Secara umum, untuk setiap versi Ubuntu Long-Term Support (LTS), juga ada versi ROS LTS yang dikeluarkan. Dengan label Dukungan Jangka Panjang (LTS), hal ini menandakan bahwa perangkat lunak yang dirilis akan mendapatkan dukungan untuk periode waktu yang lama, yakni 5 tahun, sejalan dengan kebijakan LTS baik untuk ROS maupun Ubuntu.

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)

- Running the ROS master and the ROS parameter

  Sebelum menjalankan node ROS mana pun, langkah awal yang perlu dilakukan adalah memulai ROS master dan parameter server ROS. Proses ini dapat dilakukan dengan menggunakan perintah bernama "roscore," yang akan menginisialisasi program-program berikut:

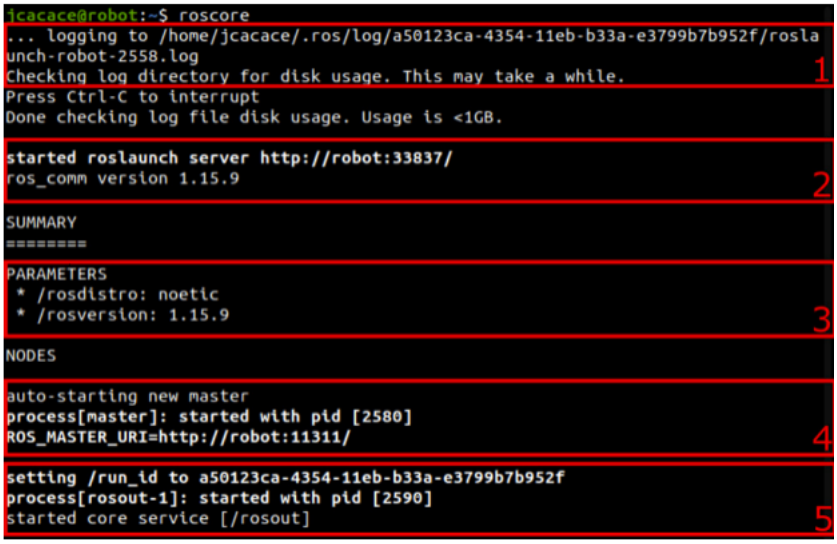
- ROS master
- ROS parameter server
- Node logging rosout

  Node rosout bertanggung jawab untuk mengumpulkan pesan log dari node ROS lainnya, menyimpannya dalam file log, dan menyebarkan pesan log tersebut ke topik lain. Topik /rosout dipublikasikan oleh node ROS menggunakan pustaka klien ROS seperti roscpp dan rospy. Node rosout kemudian berlangganan topik ini dan menyebarkan ulang pesan log ke dalam topik lain yang disebut /rosout_agg. Topik ini berisi kumpulan pesan log yang telah diagregasi.

  Untuk menjalankan roscore sebagai persyaratan sebelum menjalankan node ROS, gunakan perintah berikut di Terminal Linux. Tangkapan layar pada informasi tersebut menunjukkan pesan yang tercetak ketika perintah roscore dijalankan.

```
roscore
```

  Setelah menjalankan perintah ini, kita akan melihat teks berikut di Terminal Linux:



Berikut ini adalah isi dari roscore.xml:

```
<launch>
  <group ns="/">
    <param name="rosversion" command="rosversion roslaunch" />
    <param name="rosdistro" command="rosversion -d" />
    <node pkg="rosout" type="rosout" name="rosout"
respawn="true"/>
  </group>
</launch>
```

Saat menjalankan perintah roscore, pada tahap awal, perintah tersebut melakukan pemeriksaan argumen baris perintah untuk mendapatkan nomor port baru untuk rosmaster. Jika ada nomor port yang ditemukan, roscore akan mulai mendengarkan pada nomor port tersebut; jika tidak, port default akan digunakan. Port ini bersama dengan berkas peluncuran roscore.xml akan diteruskan ke sistem roslaunch. Sistem roslaunch diimplementasikan sebagai modul Python, yang akan mem-parsing nomor port dan menjalankan berkas roscore.xml.

Dalam berkas roscore.xml, parameter dan node ROS disusun dalam tag XML grup dengan ruang nama /. Tag XML grup menandakan bahwa semua node di dalamnya memiliki pengaturan yang serupa. Parameter rosversion dan rosdistro menyimpan output dari perintah rosversion dan rosversion-d menggunakan tag perintah, yang terdapat dalam tag param ROS. Tag perintah bertanggung jawab menjalankan perintah yang dicantumkan di dalamnya dan menyimpan hasilnya dalam dua parameter ini.

Pelaksanaan rosmaster dan server parameter terjadi melalui modul roslaunch dengan menggunakan alamat ROS_MASTER_URI. Proses ini dijalankan dalam modul Python roslaunch. ROS_MASTER_URI adalah gabungan dari alamat IP dan nomor port yang akan didengarkan oleh rosmaster. Nomor port dapat diubah sesuai dengan nomor port yang disediakan dalam perintah roscore.

- Checking the Roscore Command's Output

Mari kita lihat topik ROS dan parameter ROS yang dibuat setelah menjalankan roscore. Perintah berikut ini akan menampilkan daftar topik yang aktif di Terminal:

rostopic list

Daftar topiknya adalah sebagai berikut, sesuai dengan diskusi kita tentang langganan simpul rosout / topik rosout. Ini berisi semua pesan log dari node ROS. /rosout_agg akan menyiarkan ulang pesan log:

/rosout

/rosout_agg

Perintah berikut mencantumkan daftar parameter yang tersedia saat menjalankan roscore. Perintah berikut ini digunakan untuk mencantumkan parameter ROS yang aktif:

rosparam list

Parameter-parameter ini disebutkan di sini; parameter-parameter ini menyediakan nama distribusi ROS, versi, alamat server roslaunch, dan run_id, di mana run_id adalah ID unik unik yang terkait dengan proses tertentu dari roscore:

/rosdistro

/roslaunch

/uris/host_robot_virtualbox__51189

/rosversion /run_id

Daftar layanan ROS yang dihasilkan ketika menjalankan roscore dapat diperiksa dengan menggunakan perintah berikut: rosservice list Daftar layanan yang sedang berjalan adalah sebagai berikut:

/rosout/get_loggers /rosout/set_logger_level Layanan ROS ini dibuat untuk setiap node ROS, dan digunakan untuk mengatur tingkat.

Chapter 2 - Getting Started with ROS Programming

- Creating ROS Package

  Paket ROS adalah unit dasar dalam program ROS, dapat dibuat, dibangun, dan dirilis ke publik. Noetic Ninjemys adalah distribusi ROS yang digunakan, dengan sistem build catkin untuk membangun paket ROS. Sistem build bertanggung jawab untuk menghasilkan target dari sumber kode. Rosbuild digunakan pada distribusi lama seperti Electric dan Fuerte, tetapi catkin muncul untuk mengatasi kekurangan rosbuild dan mendekatkan kompilasi ROS ke CMake. Keuntungannya termasuk kemampuan untuk mem-porting paket ke OS lain, seperti Windows, selama OS tersebut mendukung CMake dan Python. Persyaratan pertama untuk bekerja dengan paket ROS adalah membuat workspace bernama catkin_ws setelah menginstal ROS.

  mkdir -p ~/catkin_ws/src

  Untuk mengkompilasi ruang kerja ini, kita harus mencari lingkungan ROS untuk mendapatkan akses ke ROS untuk mendapatkan akses ke fungsi-fungsi ROS:

  source /opt/ros/noetic/setup.bash

  Beralihlah ke folder src sumber yang telah kita buat sebelumnya:

  cd ~/catkin_ws/src

  Menginisialisasi ruang kerja catkin baru:

  catkin_init_workspace

  Kita dapat membangun ruang kerja meskipun tidak ada paket. Kita dapat menggunakan perintah untuk beralih ke folder ruang kerja:

  cd ~/catkin_ws

  Perintah catkin_make akan membangun ruang kerja berikut:

  catkin_make

  Perintah ini akan membuat direktori devel dan build di ruang kerja catkin Anda. File penyiapan yang berbeda terletak di dalam folder devel. Untuk menambahkan ruang kerja ROS yang telah dibuatyang telah dibuat ke lingkungan ROS, kita harus mengambil salah satu dari berkas-berkas ini. Selain itu, kita dapat mengambil berkas setup dari ruang kerja ini setiap kali sesi bash baru dimulai dengan perintah-perintah berikut:

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

source ~/.bashrc

Setelah mengatur ruang kerja catkin, kita dapat membuat paket kita sendiri yang memiliki sampel untuk mendemonstrasikan cara kerja topik, pesan, layanan, dan actionlib ROS. Perhatikan bahwa jika Anda belum menyiapkan ruang kerja dengan benar, maka Anda tidak akan dapat menggunakan Perintah ROS. Perintah catkin_create_pkg adalah cara yang paling mudah untuk membuat paket ROS. Perintah ini digunakan untuk membuat paket yang akan kita gunakan untuk membuat demo berbagai konsep ROS. Beralihlah ke folder src ruang kerja catkin dan buat paket dengan menggunakan perintah berikut: catkin_create_pkg package_name [dependency1] [dependency2] Folder kode sumber: Semua paket ROS, baik yang dibuat dari awal atau diunduh dari repositori kode lain, harus ditempatkan di folder src di ruang kerja ROS; jika tidak, paket tersebut tidak akan dikenali oleh sistem ROS dan dikompilasi.

Berikut ini adalah perintah untuk membuat sampel paket ROS:

catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs

Setelah membuat paket ini, bangun paket tanpa menambahkan node apa pun dengan menggunakan perintah catkin_make. Perintah ini harus dieksekusi dari ruang kerja catkin path. Perintah berikut ini menunjukkan kepada Anda bagaimana membangun paket ROS kosong kita: cd ~/catkin_ws && catkin_make Creating ROS nodes Node pertama yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini akan mempublikasikan sebuah nilai integer pada sebuah topik bernama /numbers. Salin kode yang ada saat ini ke dalam sebuah baru atau gunakan berkas yang sudah ada dari repositori kode buku ini. Berikut adalah kode lengkapnya:

cd ~/catkin_ws && catkin_make

- Creating ROS nodes

Node pertama yang akan kita bahas adalah demo_topic_publisher.cpp. Node ini akan mempublikasikan sebuah nilai integer pada sebuah topik bernama /numbers. Salin kode yang ada saat ini ke dalam sebuah baru atau gunakan berkas yang sudah ada dari repositori kode buku ini. Berikut adalah kode lengkapnya:

- Building the nodes

```
#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;
}
```

- Building the nodes

Kita harus mengedit file CMakeLists.txt di dalam paket untuk mengkompilasi dan membangun sumber kode. Arahkan ke mastering_ros_demo_pkg untuk melihat file CMakeLists.txtyang ada. Potongan kode berikut dalam file ini bertanggung jawab untuk membangun dua node ini

```
include_directories(
   include
   ${catkin_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
target_link_libraries(demo_topic_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_topic_subscriber ${catkin_
LIBRARIES})
```

Kita dapat menambahkan potongan sebelumnya untuk membuat file CMakeLists.txt baru untuk mengkompilasi kedua potongan kode tersebut.

Perintah catkin_make digunakan untuk membangun paket. Pertama, mari kita beralih ke sebuah ruang kerja:

cd ~/catkin_ws

Buatlah ruang kerja ROS, termasuk mastering_ros_demo_package, sebagai berikut:

catkin_make

Kita dapat menggunakan perintah sebelumnya untuk membangun seluruh ruang kerja atau menggunakan opsi Opsi -DCATKIN_WHITELIST_PACKAGES. Dengan opsi ini, Anda dapat mengatur satu atau atau lebih paket untuk dikompilasi:

catkin_make -DCATKIN_WHITELIST_PACKAGES="pkg1,pkg2,..."

Perhatikan bahwa Anda perlu mengembalikan konfigurasi ini untuk mengkompilasi paket lain atau seluruh ruang kerja. Hal ini dapat dilakukan dengan menggunakan perintah berikut:

catkin_make -DCATKIN_WHITELIST_PACKAGES=""

Jika pembangunan sudah selesai, kita dapat mengeksekusi node. Pertama, mulai roscore:

Roscore

Sekarang, jalankan kedua perintah tersebut dalam dua shell. Pada penerbit yang sedang berjalan, jalankan perintah berikut berikut ini:

rosrun mastering_ros_demo_package demo_topic_publisher

Pada subscriber yang sedang berjalan, jalankan perintah berikut:

rosrun mastering_ros_demo_package demo_topic_subscriber

- Building the ROS action server and client

  Setelah membuat dua file ini di folder src, kita harus mengedit file package.xml dan CMakeLists.txt untuk membangun node.

  File package.xml harus berisi paket pembuatan pesan dan runtime, serupa mirip dengan layanan dan pesan ROS.

  Kita harus menyertakan pustaka Boost dalam CMakeLists.txt untuk membangun node-node ini. Juga, kita juga harus menambahkan berkas aksi yang kita tulis untuk contoh ini. Kita harus melewatkan actionlib, actionlib_msgs, dan message_generation dalam find_package():

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

  Run roscore:

  Roscore

  Menjalankan code di client node:

  rosrun mastering_ros_demo_pkg demo_action_server

  Menjalankan code di client node:

  rosrun mastering_ros_demo_pkg demo_action_client 10 1

- Creating launch files

  File peluncuran di ROS sangat berguna untuk meluncurkan lebih dari satu node. Dalam contoh sebelumnya, kita telah melihat maksimal dua node ROS, tetapi bayangkan sebuah skenario di mana kita harus meluncurkan 10 atau 20 node untuk sebuah robot. Akan sulit jika kita harus menjalankan setiap node di terminal satu per satu. Sebagai gantinya, kita dapat menulis semua node di dalam file XML yang disebut file peluncuran dan, dengan menggunakan perintah yang disebut roslaunch, kita mem-parsing file file ini dan meluncurkan node. Perintah roslaunch akan secara otomatis memulai master ROS dan parameter server. Jadi, pada dasarnya, tidak perlu memulai perintah roscore dan perintah apa pun node; jika kita meluncurkan berkas, semua operasi akan dilakukan dalam satu perintah. Perhatikan bahwa jika Anda memulai sebuah node menggunakan perintah roslaunch, menghentikan atau memulai ulang ini akan memiliki efek yang sama dengan memulai ulang roscore. Mari kita mulai dengan membuat file peluncuran. Beralihlah ke folder paket dan buat sebuah baru bernama demo_topic.launch untuk meluncurkan dua node ROS untuk menerbitkan dan

berlangganan nilai bilangan bulat. Kita akan menyimpan berkas peluncuran di dalam folder peluncuran, yang yang ada di dalam paket:

roscd mastering_ros_demo_pkg

mkdir launch

cd launch

gedit demo_topic.launch

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

roslaunch mastering_ros_demo_pkg demo_topic.launch

Kita dapat memeriksa daftar node dengan menggunakan perintah berikut:

rosnode list

Kita juga dapat melihat pesan log dan men-debug node menggunakan alat GUI yang disebut rqt_ konsol:

rqt_console

Chapter 3 - Working with ROS for 3D Modeling

- Creating the ROS package for the robot description

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

  catkin_create_pkg mastering_ros_robot_description_pkg roscpp tf

  geometry_msgs urdf rviz xacro

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

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

- Explaining the URDF file

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

  check_urdf pan_tilt.urdf

  Untuk menggunakan perintah ini, paket liburdfdom-tools harus diinstal. Anda dapat menginstalnya menggunakan perintah berikut:

  sudo apt-get install liburdfdom-tools

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

  urdf_to_graphiz pan_tilt.urdf

  Perintah ini akan menghasilkan dua file:

  pan_tilt.gv dan pan_tilt.pdf

  Kita dapat melihat struktur robot ini dengan menggunakan perintah ini:

  evince pan_tilt.pdf

- Visualizing the 3D robot model in Rviz

  Kita dapat meluncurkan model dengan menggunakan perintah berikut

  roslaunch mastering_ros_robot_description_pkg view_demo.launch

- Viewing the seven-DOF arm in Rviz

  Buat berkas peluncuran berikut ini di dalam folder peluncuran, dan bangun paket menggunakan perintah catkin_make. Luncurkan urdf menggunakan perintah berikut:

  roslaunch mastering_ros_robot_description_pkg view_arm.launch

  Kita dapat melihat robot bergerak menggunakan perintah berikut:

  roslaunch mastering_ros_robot_description_pkg view_mobile_ robot.launch

Chapter 4 - Simulating Robots Using ROS and Gazebo

Simulating the robotic arm using Gazebo and ROS Dalam chapter sebelumnya, kita sudah mendesain lengan tujuh DOF. Pada bagian ini, kita akan mensimulasikan robot di Gazebo menggunakan ROS.

Sebelum memulai dengan Gazebo dan ROS, kita harus menginstal paket-paket berikut agar dapat bekerja dengan Gazebo dan ROS:

sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noeticgazebo-msgs ros-noetic-gazebo-plugins ros-noetic-gazebo-roscontro

Setelah instalasi, periksa apakah Gazebo sudah terpasang dengan benar menggunakan perintah perintah berikut:

roscore & rosrun gazebo_ros gazebo

- Creating the robotic arm simulation model for Gazebo

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

  catkin_create_pkg seven_dof_arm_gazebo gazebo_msgs gazebo_ plugins gazebo_ros gazebo_ros_control mastering_ros_robot_ description_pkg

- Simulating the robotic arm with Xtion Pro

  Sekarang kita telah mempelajari tentang definisi plugin kamera di Gazebo, kita dapat meluncurkan simulasi lengkap kita dengan menggunakan perintah berikut:

  roslaunch seven_dof_arm_gazebo seven_dof_arm_with_rgbd_world. Launch

  Kita juga dapat melihat data point cloud dari sensor ini di RViz.

  Luncurkan rviz menggunakan perintah berikut:

  rosrun rviz -f /rgbd_camera_optical_frame

  Launching the ROS controllers with Gazebo

Mari kita periksa topik pengontrol yang dihasilkan setelah menjalankan file peluncuran ini:

roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo_control. Launch

Moving the robot joints

Setelah menyelesaikan topik sebelumnya, kita dapat mulai memerintahkan setiap sendi ke posisi yang kita inginkan. Untuk menggerakkan sendi robot di Gazebo, kita harus mempublikasikan nilai sendi yang diinginkan dengan pesan ketik

std_msgs/Float64 pada topik perintah pengontrol posisi sendi. Berikut ini adalah contoh untuk memindahkan sendi keempat ke 1,0 radian:

rostopic pub /seven_dof_arm/joint4_position_controller/command std_msgs/Float64 1.0

Kita juga dapat melihat keadaan sendi robot dengan menggunakan perintah berikut:

rostopic echo /seven_dof_arm/joint_states

- Simulating a differential wheeled robot in Gazebo

  Untuk meluncurkan file ini, kita dapat menggunakan perintah berikut:

  roslaunch diff_wheeled_robot_gazebo diff_wheeled_gazebo.launch

- Adding the ROS teleop node

  Node teleop ROS menerbitkan perintah ROS Twist dengan mengambil input keyboard. Dari node ini, kita dapat menghasilkan kecepatan linier dan sudut, dan sudah ada implementasi node teleop standar yang tersedia; kita cukup menggunakan kembali node tersebut.

  Teleop diimplementasikan dalam paket diff_wheeled_robot_control. Folder script berisi node diff_wheeled_robot_key, yang merupakan node teleop. Seperti biasa biasa, Anda dapat mengunduh paket ini dari repositori Git sebelumnya. Pada titik ini, Anda mencapai paket ini dengan perintah berikut:

  roscd diff_wheeled_robot_control

  Agar berhasil mengkompilasi dan menggunakan paket ini, Anda mungkin perlu menginstal joy_node paket joy_node:

  sudo apt-get install ros-noetic-joy

- Mari kita mulai menggerakkan robot.

  Luncurkan Gazebo dengan pengaturan simulasi yang telah selesai menggunakan perintah berikut:

  roslaunch diff_wheeled_robot_gazebo diff_wheeled_gazebo_full. Launch

  Mulai node teleop:

  roslaunch diff_wheeled_robot_control keyboard_teleop.launch

  Start RViz to visualize the robot state and laser data:

  rosrun rviz

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

- Setting up CoppeliaSim with ROS

  Sebelum mulai bekerja dengan CoppeliaSim, kita perlu menginstalnya pada sistem kita dan mengkonfigurasi lingkungan kita untuk memulai jembatan komunikasi antara ROS dan adegan simulasi. CoppeliaSim adalah perangkat lunak lintas platform, tersedia untuk berbagai sistem operasi sistem seperti Windows, macOS, dan Linux. Ini dikembangkan oleh Coppelia Robotics GmbH dan didistribusikan dengan lisensi pendidikan dan komersial gratis. Unduh versi terbaru dari simulator CoppeliaSim dari halaman unduhan Coppelia Robotics di http://www.coppeliarobotics.com/downloads.html, pilih versi edu untuk Linux. Pada bab ini, kita akan mengacu pada versi CoppeliaSim 4.2.0

  tar vxf CoppeliaSim_Edu_V4_2_0_Ubuntu20_04.tar.xz

  Versi ini didukung oleh Ubuntu versi 20.04. Akan lebih mudah untuk mengganti nama folder ini dengan nama yang lebih intuitif, seperti ini:

  mv CoppeliaSim_Edu_V4_2_0_Ubuntu20_04 CoppeliaSim

  Untuk mengakses sumber daya CoppeliaSim dengan mudah, akan lebih mudah jika Anda mengatur variabel lingkungan COPPELIASIM_ROOT yang mengarah ke folder utama CoppeliaSim, seperti ini:

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

  Sekarang, kita siap untuk memulai simulator. Untuk mengaktifkan antarmuka komunikasi ROS, perintah roscore harus dijalankan pada mesin Anda sebelum membuka simulator, sedangkan untuk membuka CoppeliaSim, kita dapat menggunakan perintah berikut:

  cd $COPPELIASIM_ROOT

  ./coppeliaSim.sh

- Simulating a robotic arm using CoppeliaSim and ROS

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

  sebagai berikut:

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

- Setting up Webots with ROS

  Seperti yang telah dilakukan dengan CoppeliaSim, kita perlu menginstal Webots pada sistem kita sebelum mengaturnya dengan ROS. Webots adalah perangkat lunak simulasi multiplatform yang didukung oleh Windows, Linux, dan macOS. Perangkat lunak ini awalnya dikembangkan oleh Swiss Federal Federal Swiss, Institut Teknologi Lausanne (EPFL). Sekarang, ini dikembangkan oleh Cyberbotics, dan itu dirilis di bawah lisensi Apache 2 yang gratis dan open source. Webots menyediakan yang lengkap lingkungan pengembangan untuk memodelkan, memprogram, dan mensimulasikan robot. Ini telah dirancang untuk penggunaan profesional dan banyak digunakan dalam industri, pendidikan, dan penelitian. mari kita mulai dengan mengotentikasi repositori Cyberbotics, sebagai berikut:

  wget -qO- https://cyberbotics.com/Cyberbotics.asc | sudo apt-key add –

  Kemudian, Anda dapat mengonfigurasi manajer paket APT Anda dengan menambahkan repositori Cyberbotics repositori Cyberbotics, sebagai berikut:

  sudo apt-add-repository 'deb https://cyberbotics.com/debian/ binary-amd64/' sudo apt-get update

  Kemudian, lanjutkan ke instalasi Webots dengan menggunakan perintah berikut:

  sudo apt-get install webots

  Kita sekarang siap untuk memulai Webots dengan menggunakan perintah berikut: $ webots

  $ webots

- Simulating the robotic arm using Webots and ROS

  Integrasi Webots-ROS membutuhkan dua sisi: sisi ROS dan sisi Webots. Sisi ROS diimplementasikan melalui paket webots_ros ROS, sedangkan Webots mendukung ROSsecara 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

- Writing a teleop node using webots_ros

  Pada bagian ini, kita akan mengimplementasikan sebuah node ROS untuk secara langsung mengontrol kecepatan roda dari robot keping elektronik yang dimulai dari pesan geometry_msgs::Twist. Untuk melakukan ini, kita perlu mengeksploitasi webots_ros sebagai sebuah ketergantungan. Mari kita buat paket webots_demo_pkgdengan menetapkan webots_ros sebagai dependensi, sebagai berikut:

  catkin_create_pkg webots_demo_pkg roscpp webots_ros geometry_ msgs

Chapter 6 - Using the ROS MoveIt! and Navigation Stack

- The MoveIt! Architecture

  Sebelum menggunakan MoveIt! di sistem ROS, Anda harus menginstalnya. Prosedur instalasi sangat sederhana dan hanya berupa satu perintah. Dengan menggunakan perintah berikut, kita menginstal MoveIt! core, satu set plugin dan perencana untuk ROS Noetic:

  sudo apt-get install ros-noetic-moveit ros-noetic-moveitplugins ros-noetic-moveit-planners

  Generating a MoveIt! configuration package using the Setup Assistant tool

- Step 1 – Launching the Setup Assistant tool

  Untuk memulai alat bantu MoveIt! Setup Assistant, kita dapat menggunakan perintah berikut:

  roslaunch moveit_setup_assistant setup_assistant.launch

  Motion planning of a robot in RViz using the MoveIt! configuration package

  MoveIt! menyediakan sebuah plugin RViz yang memungkinkan para pengembang untuk mengatur masalah perencanaan. Dari plugin ini, pose manipulator yang diinginkan dapat diatur, dan lintasan gerak dapat dibuat untuk menguji kemampuan perencanaan MoveIt! Untuk meluncurkan plugin ini bersama dengan robot kita dapat langsung menggunakan file peluncuran MoveIt! yang disertakan dalam konfigurasi MoveIt! konfigurasi. Paket ini terdiri dari file konfigurasi dan file peluncuran untuk memulai gerakan perencanaan gerakan di RViz. Ada file peluncuran demo di dalam paket untuk menjelajahi semua paket fungsionalitas paket.

  Berikut ini adalah perintah untuk memanggil file peluncuran demo:

  roslaunch seven_dof_arm_config demo.launch

- Interfacing the MoveIt! configuration package to Gazebo

  Kita dapat memulai perencanaan gerakan di dalam RViz dan menjalankannya dalam simulasi Gazebo, dengan menggunakan menggunakan perintah tunggal berikut ini:

  $ roslaunch seven_dof_arm_gazebo seven_dof_arm_bringup_moveit. Launch

  Perhatikan bahwa sebelum meluncurkan scene perencanaan dengan benar, kita harus menggunakan untuk menginstal beberapa paket yang dibutuhkan oleh MoveIt! untuk menggunakan pengendali ROS:

  sudo apt-get install ros-noetc-joint-state-controller ros-noetic-position-controllers ros-noetic-jointtrajectorycontroller

- Step 5 – Debugging the Gazebo-MoveIt!

  Pada bagian ini, kita akan membahas beberapa masalah umum dan teknik debugging dalam antarmuka. Jika lintasan tidak berjalan di Gazebo, pertama-tama buat daftar topik, sebagai berikut:

  rostopic list

- Building a map using SLAM

  Sebelum mulai mengonfigurasi tumpukan Navigasi, kita perlu menginstalnya. Desktop ROS tidak akan menginstal ROS Navigation stack. Kita harus menginstal Navigation secara terpisah, dengan menggunakan perintah berikut:

  sudo apt-get install ros-noetic-navigation

  Sebelum beroperasi dengan gmapping, kita harus menginstalnya dengan menggunakan perintah berikut:

  sudo apt-get install ros-noetic-gmapping

- Running SLAM on the differential drive robot

  Kita dapat membuat paket ROS bernama diff_wheeled_robot_gazebo dan menjalankan file gmapping.launch untuk membangun peta. Cuplikan kode berikut ini menunjukkan yang perlu kita jalankan untuk memulai prosedur pemetaan.

  Mulai simulasi robot dengan menggunakan dunia Willow Garage (ditunjukkan pada Gambar 6.21), sebagai berikut:

  roslaunch diff_wheeled_robot_gazebo diff_wheeled_gazebo_full. Launch

  Jalankan file peluncuran pemetaan dengan perintah berikut ini:

  roslaunch diff_wheeled_robot_gazebo gmapping.launch

  Mulai teleoperasi keyboard untuk menavigasi robot secara manual di sekitar lingkungan. Robot dapat memetakan lingkungannya hanya jika mencakup seluruh area. Kode diilustrasikan di sini:

  roslaunch diff_wheeled_robot_control keyboard_teleop.launch

  Kita dapat menyimpan peta yang telah dibuat dengan menggunakan perintah berikut. Perintah ini akan mendengarkan topik peta dan menghasilkan sebuah gambar yang berisi keseluruhan peta. Paket map_server melakukan operasi ini:

  rosrun map_server map_saver -f willo You need to install the map server, as follows: sudo apt-get install ros-noetic-map-server