UAS Robotika 2023/2024

Nama : Daffa Asyqar Ahmad Khalisheka
NIM : 1103200034

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.

Why should we use ROS?

Robot Operating System (ROS) adalah kerangka kerja fleksibel yang menyediakan berbagai alat dan pustaka untuk menulis perangkat lunak robotik. ROS menawarkan beberapa fitur canggih untuk membantu pengembang dalam tugas-tugas seperti pengiriman pesan, komputasi terdistribusi, penggunaan kembali kode, dan implementasi algoritme canggih untuk aplikasi robotik. Proyek ROS dimulai pada tahun 2007 oleh Morgan Quigley dan pengembangannya dilanjutkan di Willow Garage, sebuah laboratorium untuk mengembangkan perangkat keras dan perangkat lunak sumber terbuka untuk robot. Tujuan dari ROS adalah untuk menetapkan cara standar untuk memprogram robot sambil menawarkan perangkat lunak siap pakai yang dapat dengan mudah diintegrasikan dengan aplikasi robotik khusus. Ada banyak alasan untuk memilih ROS sebagai kerangka kerja pemrograman, dan beberapa di antaranya adalah sebagai berikut:

- Kemampuan kelas atas: ROS hadir dengan fungsi yang siap digunakan. Sebagai contoh, Pelokalan dan Pemetaan Simultan (SLAM) dan Adaptive Monte Carlo Localization (AMCL) di ROS dapat digunakan untuk memiliki otonom navigasi di robot seluler, sedangkan paket MoveIt dapat digunakan untuk gerakan perencanaan untuk manipulator robot. Kemampuan ini dapat langsung digunakan dalam perangkat lunak robot tanpa kerumitan. Dalam beberapa kasus, paket-paket ini cukup untuk memiliki tugas robotika inti pada platform yang berbeda. Juga, kemampuan ini sangat tinggi dapat dikonfigurasi; kita dapat menyempurnakan masing-masing menggunakan berbagai parameter.
- Banyak sekali alat: Ekosistem ROS dilengkapi dengan banyak sekali alat untuk melakukan debugging, memvisualisasikan, dan melakukan simulasi. Alat-alat tersebut,

seperti rqt_gui, RViz, dan Gazebo, adalah beberapa alat open source terkuat untuk debugging, visualisasi, dan simulasi. Kerangka kerja perangkat lunak yang memiliki banyak alat ini sangat jarang.

- Dukungan untuk sensor dan aktuator kelas atas: ROS memungkinkan kita untuk menggunakan perangkat yang berbeda driver dan paket antarmuka berbagai sensor dan aktuator dalam robotika. Seperti sensor kelas atas termasuk LIDAR 3D, pemindai laser, sensor kedalaman, aktuator, dan banyak lagi. Kita dapat menghubungkan komponen-komponen ini dengan ROS tanpa kerumitan.
- Penanganan sumber daya secara bersamaan: Menangani sumber daya perangkat keras melalui lebih dari dua proses selalu memusingkan. Bayangkan kita ingin memproses sebuah gambar dari sebuah kamera untuk deteksi wajah dan deteksi gerakan; kita dapat menulis kode sebagai entitas tunggal yang dapat melakukan keduanya, atau kita dapat menulis kode berulir tunggal untuk konkurensi. Jika kita ingin menambahkan lebih dari dua fitur ke dalam sebuah thread, aplikasi-aplikasi akan menjadi kompleks dan sulit untuk di-debug. Tetapi di ROS, kita dapat mengakses perangkat menggunakan topik ROS dari driver ROS. Sejumlah node ROS dapat berlangganan pesan gambar dari driver kamera ROS, dan setiap node bisa memiliki fungsi yang berbeda. Hal ini dapat mengurangi kompleksitas dalam komputasi dan juga meningkatkan kemampuan debugging seluruh sistem.

ROS metapackages

Metapackages adalah paket khusus yang hanya membutuhkan satu file; yaitu file package.xml file.

Metapackages hanya mengelompokkan sekumpulan beberapa paket sebagai satu paket logis. Di dalam file package.xml, metapackage berisi tag ekspor, seperti yang ditunjukkan di sini:

```
  <export>
    <metapackage/>
  </export>
```

Selain itu, dalam metapackages, tidak ada dependensi <buildtool_depend> untuk catkin; hanya ada dependensi <run_depend>, yang merupakan paket-paket yang dikelompokkan yang dikelompokkan di dalam metapackage.

Tumpukan navigasi ROS adalah contoh yang baik untuk suatu tempat yang berisi metapaket. Jika ROS dan paket navigasinya telah terinstal, kita dapat mencoba menggunakan perintah berikut dengan beralih ke folder metapaket navigasi:

roscd navigation

Buka package.xml menggunakan editor teks favorit Anda (gedit, dalam kasus berikut):

gedit package.xml

ROS distributions

Pembaruan ROS dirilis dengan distribusi ROS yang baru. Distribusi ROS yang baru adalah terdiri dari versi terbaru dari perangkat lunak intinya dan satu set ROS baru / yang diperbarui yang baru/terbaru. ROS mengikuti siklus rilis yang sama dengan distribusi Ubuntu Linux: versi baru-baru dari ROS dirilis setiap 6 bulan. Biasanya, untuk setiap versi Ubuntu LTS, sebuah ROS versi LTS dirilis. Dukungan Jangka Panjang (LTS) dan berarti bahwa yang dirilis perangkat lunak yang dirilis akan dipertahankan untuk waktu yang lama (5 tahun untuk ROS dan 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 apa pun, kita harus memulai master ROS dan parameter ROS server. Kita dapat memulai master ROS dan server parameter ROS dengan menggunakan satu yang disebut roscore, yang akan memulai program-program berikut:

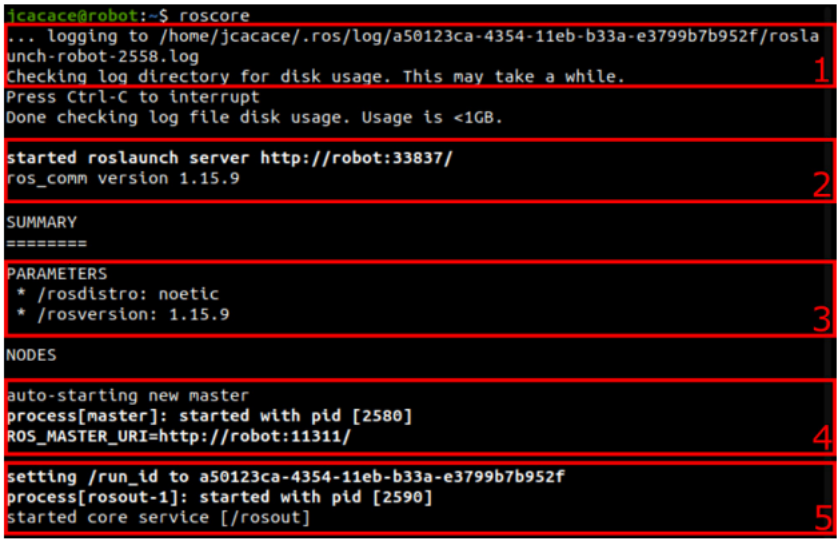
- ROS master
- ROS parameter server
- rosout logging nodes

Node rosout akan mengumpulkan pesan log dari node ROS lain dan menyimpannya dalam file log, dan juga akan menyiarkan ulang pesan log yang dikumpulkan ke topik lain. Topik Topik /rosout dipublikasikan oleh node ROS menggunakan pustaka klien ROS seperti roscpp dan rospy, dan topik ini dilanggan oleh node rosout, yang menyiarkan ulang pesan dalam topik lain yang disebut /rosout_agg. Topik ini berisi aliran agregat log pesan. Perintah roscore harus dijalankan sebagai prasyarat untuk menjalankan ROS node. Tangkapan layar berikut ini menunjukkan pesan-pesan yang dicetak ketika kita menjalankan perintah roscore di Terminal.

Gunakan perintah berikut untuk menjalankan roscore pada Terminal Linux:

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>
```

Ketika perintah roscore dijalankan, pada awalnya, perintah ini akan memeriksa argumen baris perintah untuk mendapatkan nomor port baru untuk rosmaster. Jika mendapatkan nomor port, ia akan akan mulai mendengarkan nomor port yang baru; jika tidak, ia akan menggunakan port default. Port ini port ini dan berkas peluncuran roscore.xml akan diteruskan ke sistem roslaunch. Sistem Sistem roslaunch diimplementasikan dalam sebuah modul Python; sistem ini akan mem-parsing nomor port dan meluncurkan berkas roscore.xml.

Dalam file roscore.xml, kita dapat melihat bahwa parameter dan node ROS dienkapsulasi dalam tag XML grup dengan ruang nama /. Tag XML grup menunjukkan bahwasemua node di dalam tag ini memiliki pengaturan yang sama.

Parameter rosversion dan rosdistro menyimpan output dari perintahperintah rosversionroslaunch dan rosversion-d menggunakan tag perintah, yang merupakan bagian dari tag param ROS. Tag perintah akan menjalankan perintah yang disebutkan di dalamnya dan menyimpan output perintah dalam dua parameter ini.

rosmaster dan server parameter dieksekusi di dalam modul roslaunch melalui alamat ROS_MASTER_URI. Hal ini terjadi di dalam modul Python roslaunch. ROS_MASTER_URI adalah kombinasi dari alamat IP dan port yang akan didengarkan oleh rosmaster untuk mendengarkan. Nomor port dapat diubah sesuai dengan nomor port yang diberikan dalam perintah 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 a ROS package

Paket ROS adalah unit dasar dari program ROS. Kita dapat membuat paket ROS, membangun membangunnya, dan merilisnya ke publik. Distribusi ROS yang kami gunakan saat ini adalah Noetic Ninjemys. Kami menggunakan sistem build catkin untuk membangun paket ROS. Sebuah sistem build bertanggung jawab untuk menghasilkan target (executable/library) dari sumber tekstual kode yang dapat digunakan oleh pengguna akhir. Pada distribusi yang lebih lama, seperti Electric dan Fuerte, rosbuild adalah sistem pembangun. Karena berbagai kekurangan dari rosbuild, catkinmuncul. Hal ini juga memungkinkan kita untuk memindahkan sistem kompilasi ROS lebih dekat ke Cross Platform Make (CMake). Hal ini memiliki banyak keuntungan, seperti mem-porting paket ke OS lain, seperti Windows. Jika sebuah OS mendukung CMake dan Python, maka paket-paket yang berbasis berbasis catkin, maka paket-paket dapat di-porting ke OS tersebut.

Persyaratan pertama untuk bekerja dengan paket ROS adalah membuat catkin ROS ruang kerja ROS. Setelah menginstal ROS, kita dapat membuat dan membangun sebuah catkinworkspace bernama catkin_ws:

```
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:

```
#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

In the running subscriber, run the following command:

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

Launch the action server node:

rosrun mastering_ros_demo_pkg demo_action_server

Launch the action 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-noetic gazebo-msgs ros-noetic-gazebo-plugins ros-noetic-gazebo-ros contro

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

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-moveit plugins 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-joint-trajectory□controller

Step 5 – Debugging the Gazebo-MoveIt! Interface

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

Chapter 7: Exploring the Advanced Capabilities of ROS MoveIt!

Motion planning using the move_group C++ interface

Pada Bab 6, Menggunakan ROS MoveIt! dan Tumpukan Navigasi, kita telah membahas cara berinteraksi dengan lengan robot dan cara merencanakan jalurnya menggunakan MoveIt! Visualisasi ROS (RViz) plugin perencanaan gerakan. Pada bagian ini, kita akan melihat cara memprogram gerakan robot menggunakan API C++ move_group. Perencanaan gerak menggunakan RViz juga dapat dilakukan secara terprogram melalui API C++ move_group.

Langkah pertama untuk mulai bekerja dengan API C++ adalah membuat paket ROS lain yang memiliki paket MoveIt! sebagai dependensi. Kita dapat membuat paket yang sama ini menggunakan perintah perintah berikut:

catkin_create_pkg seven_dof_arm_test catkin cmake_modules interactive_markers moveit_core moveit_ros_perception moveit_ ros_planning_interface pluginlib roscpp std_msgs

Perintah berikut ini akan memulai RViz dengan lengan 7-DOF dengan perencanaan gerak plugin:

roslaunch seven_dof_arm_config demo.launch

Pindahkan end effector untuk memeriksa apakah semuanya bekerja dengan baik di RViz.Jalankan node C++ untuk perencanaan ke posisi acak dengan menggunakan perintah berikut:

rosrun seven_dof_arm_test test_random_node

Checking self-collisions using MoveIt! APIs

Kita dapat meluncurkan demo perencanaan gerak dan menjalankan node ini menggunakan perintah berikut ini:

roslaunch seven_dof_arm_config demo.launch

Kita dapat menjalankan simpul pemeriksaan tabrakan dengan perintah ini:

rosrun seven_dof_arm_test check_collision

Working with perception using MoveIt! and Gazebo

Hingga saat ini, di MoveIt!, kami hanya bekerja dengan satu lengan saja. Pada bagian ini, kita akan melihat bagaimana untuk menghubungkan data sensor penglihatan 3D ke MoveIt!. Sensor dapat

disimulasikan menggunakan Gazebo, atau Anda dapat langsung menghubungkan sensor Red-Green-Blue-Depth (RGB-D), sepertiKinect atau Intel RealSense, dengan menggunakan paket openni_launch. Di sini, kita akan bekerja menggunakan Simulasi Gazebo. Kita akan menambahkan sensor ke MoveIt! untuk membuat peta lingkungan di sekitar robot. Perintah berikut akan meluncurkan lengan robot dan Asus Xtion pro simulasi di Gazebo di dunia dengan rintangan:

roslaunch seven_dof_arm_gazebo seven_dof_arm_obstacle_world. Launch

Periksa topik yang dihasilkan setelah memulai simulasi dengan perintah berikut:

rostopic list

Kita dapat melihat point cloud di RViz dengan menggunakan perintah berikut:

rosrun rviz rviz -f base_link

Chapter 8: ROS for Aerial Robots

Using the PX4 flight control stack

Firmware PX4 memungkinkan pengembang untuk secara langsung mensimulasikan kode yang berjalan pada autopilot pada sistem Linux Anda. Selain itu, dimungkinkan untuk memodifikasi sumber autopilot kode dan memuat ulang versi baru pada papan Pixhawk. Untuk menginstal firmware pada Anda, Anda harus mengunduhnya terlebih dahulu. Meskipun tidak wajib, menautkan ini dengan ROS akan menempatkannya dengan mudah di ruang kerja ROS Anda. Untuk mengunduh autopilot, masuk ke ruang kerja ROS Anda dan gunakan perintah berikut:

git clone https://github.com/PX4/PX4-Autopilot.git --recursive

Perhatikan bahwa kita menggunakan opsi --rekursif pada perintah clone untuk mengunduh semua yang disertakan dalam repositori utama. Ini berarti bahwa beberapa bagian dari kode sumber autopilot autopilot disimpan di repositori eksternal lain yang ditautkan oleh repositori utama. Perintah klon mungkin memerlukan waktu beberapa menit. Perhatikan bahwa, setelah perintah clone selesai, direktori baru yang disebut PX4-Autopilot telah dibuat. Folder ini berisi semua file yang diperlukan untuk memodifikasi dan mengunggah firmware pada tertanam (autopilot) dan untuk mensimulasikan kode sumber pada simulator yang berbeda. Untuk menautkan semua elemen yang diperlukan, direktori firmware dikenali sebagai paket ROS, bahkan meskipun tidak dikompilasi seperti itu. Sebagai tambahan, nama paket ini adalah px4. Perhatikan bahwa nama direktori tidak selalu mewakili nama paket ROS. Jadi, setelah Anda mengkloning direktori ini ke dalam ruang kerja ROS Anda, Anda dapat bergabung dengan folder firmware dengan perintah berikut:

roscd px4

Anda sekarang siap untuk mengkompilasi paket ini dan memulai simulasi. Sebelum melakukan ini, Anda perlu menginstal set dependensi berikut ini:

sudo apt install python3-pip

pip3 install --user empy

pip3 install --user toml

pip3 install --user numpy

pip3 install --user numpy

sudo apt-get install libgstreamer-plugins-base1.0-dev

pip3 install --user jinja2

Untuk memungkinkan komunikasi ROS/Flight Control Unit (FCU), Anda harus menginstal paket mavros:

sudo apt-get install ros-noeitc-mavros ros-noeitc-mavros-msgs

Sekarang Anda dapat menginstal dataset geografis:

sudo /opt/ros/noetic/lib/mavros/install_geographiclib_datasets. Sh

Terakhir, Anda dapat menjalankan perintah berikut untuk mengkompilasinya

roscd px4 && make px4_sitl_default

Dalam kasus ini, kami menggunakan perintah make dengan target tertentu. Ini adalah Perangkat Lunak dalam

target Software in the Loop (SITL). Hal ini memungkinkan kita untuk mensimulasikan kode sumber firmware. Seperti yang sudah dinyatakan, simulator memungkinkan kode penerbangan PX4 untuk mengontrol kendaraan model komputer dalam dunia simulasi. Setelah kami meluncurkan simulasi, kami dapat berinteraksi dengan dengan kendaraan ini seperti halnya dengan kendaraan sungguhan, menggunakan perangkat lunak stasiun bumi seperti QGroundControl, API offboard, atau gamepad pengontrol radio. Simulator yang berbeda didukung. Daftar lengkapnya dapat ditemukan di tautan berikut: https://docs.px4.io/master/en/simulation/. Namun, kita akan menggunakan Gazebo. Untuk meluncurkan Kode kontrol PX4 dengan Gazebo, jalankan perintah berikut ini dari direktori root sumber firmware:

make px4_sitl_default gazebo

Writing a ROS-PX4 application

Sekarang mari kita buat paket baru di mana kita akan menyimpan semua file sumber dan file peluncuran yang diperlukan untuk mengirim dan menerima data dari UAV simulasi menggunakan ROS. Masuk ke ROS Anda Anda dan gunakan perintah berikut:

catkin_create_project px4_ros_ctrl roscpp mavros_msgs geometry_msgs

Installing RotorS

Mari kita mulai dengan menginstal RotorS pada sistem kita. Untuk menyelesaikan langkah ini, Anda harus menginstal ketergantungan berikut ini:

sudo apt-get install ros-noetic-joy ros-noetic-octomap ros ros-noetic-mavlink protobuf-compiler libgoogle-glog-dev ros-noetic-control-toolbox

Sekarang, kloning repositori RotorS di ruang kerja ROS Anda:

roscd && cd ../src

git clone https://github.com/ethz-asl/rotors_simulator.git

Kemudian, kompilasi ruang kerja menggunakan perintah catkin_make.

Jika kompilasi selesai tanpa terjadi kesalahan, maka Anda siap untuk meluncurkan simulator menggunakan salah satu model yang disediakan oleh RotorS. Selain itu, paket ini mengimplementasikan pengontrol UAV untuk memerintahkan posisinya di dunia simulasi. Untuk Sebagai contoh, untuk mensimulasikan model hexacopter, Anda dapat menggunakan perintah berikut:

roslaunch rotors_gazebo mav_hovering_example.launch mav_ name:=firefly world_name:=basic

Summary

Chapter ini memperkenalkan konsep robot udara dan mendiskusikan elemen-elemen utamanya. Kami juga menjelaskan salah satu papan autopilot paling terkenal yang digunakan untuk mengembangkan aplikasi dengan UAV - papan kontrol Pixhawk yang menjalankan autopilot PX4. Setelah kami mempelajari cara menggunakan platform multirotor nyata dan mengintegrasikannya dengan ROS, kami kemudian kemudian membahas dua modalitas simulasi. Sangat penting untuk mensimulasikan efek dari algoritme kontrol sebelum menjalankannya di UAV sungguhan. Ini untuk mencegah kerusakan pada robot dan orang-orang di sekitarnya.