

Nama : Rizqullah Imamuddin Habibi
NIM : 1103204139

UAS Robotika

Chapter 1 Introduction to ROS

Pada bab pertama ini, kita akan membahas konsep-konsep ROS seperti ROS master, ROS nodes, ROS parameter server, dan pesan dan layanan ROS, sambil membahas instalasi ROS dan cara memulai dengan ROS master. Dalam bab ini, kita akan membahas topik-topik berikut:

- Mengapa kita perlu mempelajari ROS?
- Memahami tingkat sistem file ROS.
- Memahami tingkat grafik komputasi ROS.
- Tingkat komunitas ROS.

Technical requirements:

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

Why should we use ROS?

ROS adalah kerangka kerja fleksibel untuk menulis perangkat lunak robotik. Ini menyediakan fitur-fitur kuat seperti pertukaran pesan, komputasi terdistribusi, dan implementasi algoritma canggih. Alasan memilih ROS:

- Kemampuan tinggi dengan fungsionalitas siap pakai.
- Banyak alat untuk debugging, visualisasi, dan simulasi.
- Dukungan untuk sensor dan aktuator canggih.
- Kompatibilitas antar-platform dan modularitas.

Understanding the ROS filesystem level

Level filesystem pada ROS mencakup konsep-konsep yang terdapat pada disk, seperti:

Paket: Paket adalah unit utama untuk mengatur perangkat lunak dalam ROS. Sebuah paket dapat berisi proses runtime ROS (node), pustaka yang bergantung pada ROS, kumpulan data, file konfigurasi, atau apa pun yang diatur bersama. Paket adalah item build dan item rilis yang paling kecil di ROS. Artinya, hal yang paling terperinci yang dapat Anda bangun dan rilis adalah sebuah paket.

Metapackages: Metapackages adalah Paket khusus yang hanya berfungsi untuk mewakili sekelompok paket lain yang terkait. Umumnya metapackages digunakan sebagai tempat yang kompatibel dengan rosbuilt Stacks yang telah dikonversi.

Manifes Paket: Manifes (package.xml) menyediakan metadata tentang sebuah paket, termasuk nama, versi, deskripsi, informasi lisensi, ketergantungan, dan informasi meta lainnya seperti paket yang diekspor. Manifes paket package.xml didefinisikan dalam REP-0127.

Repositori: Kumpulan paket yang berbagi sistem VCS yang sama. Paket-paket yang berbagi VCS memiliki versi yang sama dan dapat dirilis bersama menggunakan alat otomatisasi rilis catkin, bloom. Seringkali repositori ini akan memetakan ke rosbuilt Stacks yang telah dikonversi. Repositori juga dapat berisi hanya satu paket.

Jenis pesan (msg): Deskripsi pesan, disimpan di `my_package/msg/MyMessageType.msg`, mendefinisikan struktur data untuk pesan yang dikirim dalam ROS.

Jenis layanan (srv): Deskripsi layanan, disimpan di `my_package/srv/MyServiceType.srv`, mendefinisikan struktur data permintaan dan respons untuk layanan di ROS.

ROS computation graph level

Computation Graph adalah jaringan peer-to-peer dari proses ROS yang memproses data bersama. Konsep dasar Computation Graph ROS melibatkan node, Master, Parameter Server, pesan, layanan, topik, dan bags, yang semuanya menyediakan data ke Grafik dengan cara yang berbeda.

1. **Node:** Proses komputasi modular pada tingkat yang halus, seperti pengendalian sensor atau perencanaan jalur.
2. **Master:** Menyediakan registrasi dan pencarian nama untuk node dalam Graf Komputasi. Tanpa Master, node tidak dapat berkomunikasi.
3. **Parameter Server:** Memungkinkan penyimpanan data terpusat berdasarkan kunci, saat ini bagian dari Master.
4. **Pesan:** Node berkomunikasi dengan mengirim pesan, struktur data sederhana dengan bidang tertentu.
5. **Topik:** Pesan diarahkan melalui sistem publish/subscribe menggunakan topik untuk mengidentifikasi kontennya.
6. **Layanan:** Interaksi permintaan/jawab diimplementasikan melalui layanan dengan struktur pesan untuk permintaan dan jawaban.
7. **Bags:** Format penyimpanan dan pemutaran data pesan ROS, penting untuk menyimpan data sensor.

ROS Master bertindak sebagai layanan nama dalam Graf Komputasi ROS, menyimpan informasi registrasi node dan memfasilitasi koneksi dinamis.

Arsitektur ini memungkinkan operasi terurai, di mana nama adalah cara utama dalam membangun sistem yang lebih besar dan kompleks. Nama node, topik, layanan, dan parameter adalah elemen kunci dalam ROS.

Contoh penggunaannya: untuk mengendalikan sensor laser Hokuyo, kita memulai node `hokuyo_node` yang mempublikasikan data pada topik `scan`. Node lain, seperti filter, dapat berlangganan topik ini untuk memproses data tanpa saling mengetahui.

Kemudahan konfigurasi di ROS memungkinkan perubahan nama pada waktu runtime, mendukung fleksibilitas dalam topologi Graf Komputasi.

ROS community level

ROS Community Level mencakup konsep-konsep dan sumber daya yang memungkinkan komunitas terpisah untuk bertukar perangkat lunak dan pengetahuan. Sumber daya ini melibatkan:

1. **Distribusi:** Distribusi ROS adalah kumpulan dari tumpukan yang telah diberi versi yang dapat Anda instal. Distribusi memiliki peran serupa dengan distribusi Linux: memudahkan instalasi koleksi perangkat lunak, serta menjaga konsistensi versi di seluruh perangkat lunak.

2. **Repositori:** ROS bergantung pada jaringan repositori kode federatif, di mana institusi berbeda dapat mengembangkan dan merilis komponen perangkat lunak robot mereka sendiri.
3. **ROS Wiki:** Wiki komunitas ROS adalah forum utama untuk mendokumentasikan informasi tentang ROS. Siapa pun dapat mendaftar akun dan berkontribusi dalam dokumentasi, memberikan koreksi atau pembaruan, menulis tutorial, dan lainnya.
4. **Sistem Tiket Bug:** Lihat Tiket untuk informasi tentang cara membuat tiket.
5. **Mailing List:** Mailing list ros-users adalah saluran komunikasi utama tentang pembaruan baru untuk ROS, serta forum untuk bertanya tentang perangkat lunak ROS.
6. **ROS Answers:** Situs Tanya Jawab untuk menjawab pertanyaan terkait ROS.
7. **Blog:** Blog ros.org memberikan pembaruan reguler, termasuk foto dan video.

Chapter 2

Dalam bab ini, fokusnya adalah pada pembuatan dan pengembangan paket ROS sambil mengimplementasikan sistem komunikasi ROS. Topik utama yang dibahas meliputi pembuatan node ROS, bekerja dengan topik, pesan, layanan, dan actionlib.

Untuk mengikuti bab ini, diperlukan laptop standar dengan sistem operasi Ubuntu 20.04 yang telah diinstal ROS Noetic. Kode referensi dapat diunduh dari GitHub. Workspace catkin ROS, 'catkin_ws,' dibuat untuk mengompilasi workspace dan mengakses fungsi ROS.

Bab ini menekankan pentingnya paket ROS sebagai unit dasar program ROS. Pembuatan workspace catkin ROS diilustrasikan, diikuti dengan penggunaan perintah catkin_make untuk membangun workspace. Proses tersebut melibatkan pembuatan paket, inisialisasi workspace catkin baru, dan penyiapan variabel lingkungan yang diperlukan.

Topik ROS, yang memfasilitasi komunikasi asinkron antar node, diperkenalkan. Bab ini membimbing langkah-langkah pembuatan dua node ROS untuk mempublikasikan dan berlangganan topik. Demonstrasi melibatkan penggunaan roscpp untuk implementasi C++, std_msgs untuk tipe data dasar, dan actionlib untuk membuat tugas yang dapat dihentikan.

Kode dan penjelasan ini menggambarkan pembuatan sebuah paket ROS dengan dua node: sebuah penerbit (demo_topic_publisher.cpp) dan seorang pelanggan (demo_topic_subscriber.cpp). Node penerbit menerbitkan nilai integer pada topik bernama /numbers, sementara node pelanggan berlangganan pada topik yang sama dan mencetak data yang diterima.

- Node Penerbit (demo_topic_publisher.cpp)

1. Inisialisasi:

```
ros::init(argc, argv, "demo_topic_publisher");

ros::NodeHandle node_obj;
```

Menginisialisasi node ROS dengan nama unik (demo_topic_publisher) dan membuat objek NodeHandle untuk berkomunikasi dengan sistem ROS.

2. Inisialisasi Penerbit:

```
ros::Publisher number_publisher = node_obj.advertise<std_msgs::Int32>("/numbers", 10);
```

Membuat penerbit topik bernama /numbers dengan tipe std_msgs::Int32 dan ukuran buffer 10.

3. Loop untuk Menerbitkan:

```
while (ros::ok()) {  
    // Membuat dan mengisi pesan  
    std_msgs::Int32 msg;  
    msg.data = number_count;  
  
    // Melog dan menerbitkan pesanROS_INFO("%d",  
    msg.data); number_publisher.publish(msg);  
  
    // Tidur untuk mencapai laju penerbitan 10 Hz  
    loop_rate.sleep();  
    ++number_count;  
}
```

Terus-menerus menerbitkan nilai integer yang diinkrementasi pada topik /numbers.

- Node Pelanggan (demo_topic_subscriber.cpp)

1. Inisialisasi:

```
ros::init(argc, argv,  
"demo_topic_subscriber");ros::NodeHandle  
node_obj;
```

Menginisialisasi node pelanggan dengan nama unik (demo_topic_subscriber) dan membuat objek NodeHandle.

2. Inisialisasi Pelanggan:

```
ros::Subscriber number_subscriber = node_obj.subscribe("/numbers", 10,  
number_callback);
```

Membuat pelanggan untuk topik /numbers dengan ukuran buffer 10 dan menentukan fungsi panggilan balik (number_callback) yang akan dieksekusi saat data diterima.

3. Fungsi Panggilan Balik:

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

Mencetak nilai integer yang diterima saat panggilan balik dipicu.

- Spin:

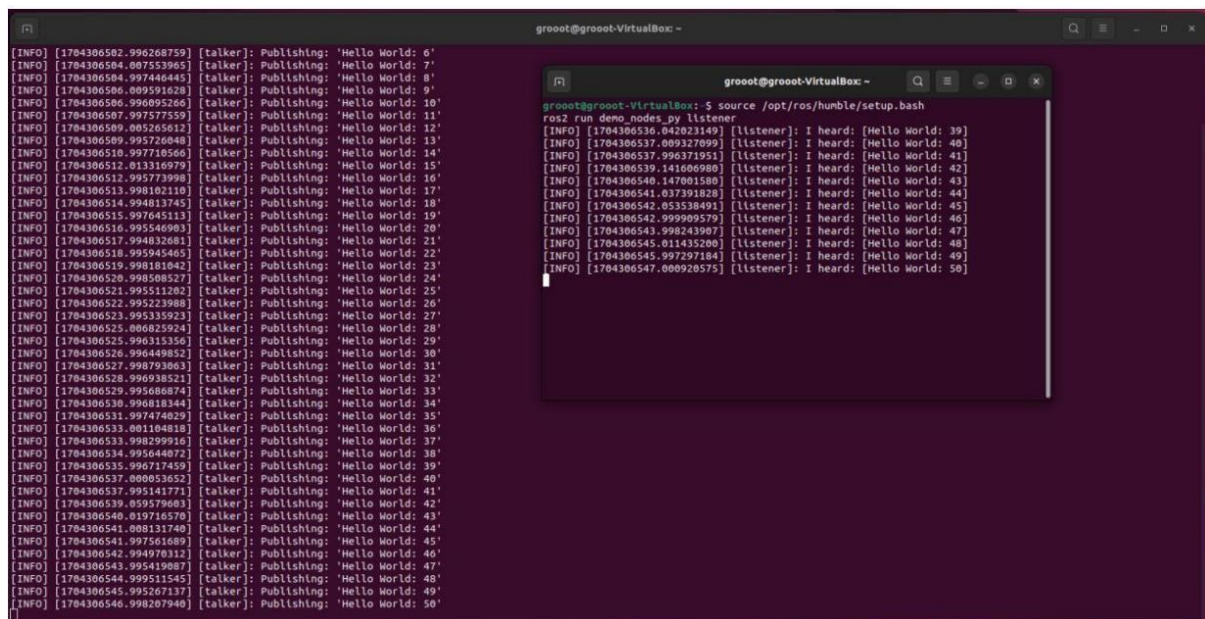
```
ros::spi
```

```
n());
```

Menjaga node tetap aktif, menunggu pesan masuk, dan menjalankan fungsi panggilan balik.

File CMakeLists.txt dikonfigurasi untuk membangun kedua node (demo_topic_publisher dan demo_topic_subscriber). Pesan kustom diperkenalkan menggunakan file .msg (demo_msg.msg), dan kompilasinya dikelola dalam file CMakeLists.txt. Sebuah layanan kustom diperkenalkan menggunakan file .srv (demo_srv.srv), dan kompilasinya dikelola dengan cara yang sama. Artikel memberikan petunjuk langkah demi langkah tentang cara membangun, menjalankan, dan menguji node, serta perintah tambahan untuk debugging dan memahami komunikasi antar node.

Contoh:



```
groot@groot-VirtualBox: -
[INFO] [1704306502.996268759] [talker]: Publishing: 'Hello World: 6'
[INFO] [1704306504.007553965] [talker]: Publishing: 'Hello World: 7'
[INFO] [1704306504.997446445] [talker]: Publishing: 'Hello World: 8'
[INFO] [1704306506.009591628] [talker]: Publishing: 'Hello World: 9'
[INFO] [1704306506.996095266] [talker]: Publishing: 'Hello World: 10'
[INFO] [1704306507.997577559] [talker]: Publishing: 'Hello World: 11'
[INFO] [1704306509.005265612] [talker]: Publishing: 'Hello World: 12'
[INFO] [1704306509.995726048] [talker]: Publishing: 'Hello World: 13'
[INFO] [1704306510.997716566] [talker]: Publishing: 'Hello World: 14'
[INFO] [1704306512.013316979] [talker]: Publishing: 'Hello World: 15'
[INFO] [1704306512.995773998] [talker]: Publishing: 'Hello World: 16'
[INFO] [1704306513.998102110] [talker]: Publishing: 'Hello World: 17'
[INFO] [1704306514.994813745] [talker]: Publishing: 'Hello World: 18'
[INFO] [1704306515.997645113] [talker]: Publishing: 'Hello World: 19'
[INFO] [1704306516.995546903] [talker]: Publishing: 'Hello World: 20'
[INFO] [1704306517.994832681] [talker]: Publishing: 'Hello World: 21'
[INFO] [1704306518.995945465] [talker]: Publishing: 'Hello World: 22'
[INFO] [1704306519.998181042] [talker]: Publishing: 'Hello World: 23'
[INFO] [1704306520.998508527] [talker]: Publishing: 'Hello World: 24'
[INFO] [1704306521.995511202] [talker]: Publishing: 'Hello World: 25'
[INFO] [1704306522.995223988] [talker]: Publishing: 'Hello World: 26'
[INFO] [1704306523.995335923] [talker]: Publishing: 'Hello World: 27'
[INFO] [1704306525.006025924] [talker]: Publishing: 'Hello World: 28'
[INFO] [1704306525.996315356] [talker]: Publishing: 'Hello World: 29'
[INFO] [1704306526.996449852] [talker]: Publishing: 'Hello World: 30'
[INFO] [1704306527.998793063] [talker]: Publishing: 'Hello World: 31'
[INFO] [1704306528.996938521] [talker]: Publishing: 'Hello World: 32'
[INFO] [1704306529.995686874] [talker]: Publishing: 'Hello World: 33'
[INFO] [1704306530.996818344] [talker]: Publishing: 'Hello World: 34'
[INFO] [1704306531.997474029] [talker]: Publishing: 'Hello World: 35'
[INFO] [1704306533.001104818] [talker]: Publishing: 'Hello World: 36'
[INFO] [1704306533.998299916] [talker]: Publishing: 'Hello World: 37'
[INFO] [1704306534.995644072] [talker]: Publishing: 'Hello World: 38'
[INFO] [1704306535.996717459] [talker]: Publishing: 'Hello World: 39'
[INFO] [1704306537.000853652] [talker]: Publishing: 'Hello World: 40'
[INFO] [1704306537.995141771] [talker]: Publishing: 'Hello World: 41'
[INFO] [1704306539.059379603] [talker]: Publishing: 'Hello World: 42'
[INFO] [1704306540.019716570] [talker]: Publishing: 'Hello World: 43'
[INFO] [1704306541.008131740] [talker]: Publishing: 'Hello World: 44'
[INFO] [1704306541.997561689] [talker]: Publishing: 'Hello World: 45'
[INFO] [1704306542.994978312] [talker]: Publishing: 'Hello World: 46'
[INFO] [1704306543.995419087] [talker]: Publishing: 'Hello World: 47'
[INFO] [1704306544.999511545] [talker]: Publishing: 'Hello World: 48'
[INFO] [1704306545.995267137] [talker]: Publishing: 'Hello World: 49'
[INFO] [1704306546.998207940] [talker]: Publishing: 'Hello World: 50'

groot@groot-VirtualBox: -
groot@groot-VirtualBox: $ source /opt/ros/humble/setup.bash
ros2 run demo_nodes_py listener
[INFO] [1704306536.042023149] [listener]: I heard: [Hello World: 39]
[INFO] [1704306537.009327099] [listener]: I heard: [Hello World: 40]
[INFO] [1704306537.996371951] [listener]: I heard: [Hello World: 41]
[INFO] [1704306539.141606980] [listener]: I heard: [Hello World: 42]
[INFO] [1704306540.147001580] [listener]: I heard: [Hello World: 43]
[INFO] [1704306541.037391828] [listener]: I heard: [Hello World: 44]
[INFO] [1704306542.053538491] [listener]: I heard: [Hello World: 45]
[INFO] [1704306542.999909579] [listener]: I heard: [Hello World: 46]
[INFO] [1704306543.998243967] [listener]: I heard: [Hello World: 47]
[INFO] [1704306545.011435200] [listener]: I heard: [Hello World: 48]
[INFO] [1704306545.997297184] [listener]: I heard: [Hello World: 49]
[INFO] [1704306547.000920575] [listener]: I heard: [Hello World: 50]
```