

Nama : Anggi Amalia
NIM : 1103210183
Kelas : TK4503

Bagian 2

Getting Started with ROS Programming

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 pesan
    ROS_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.

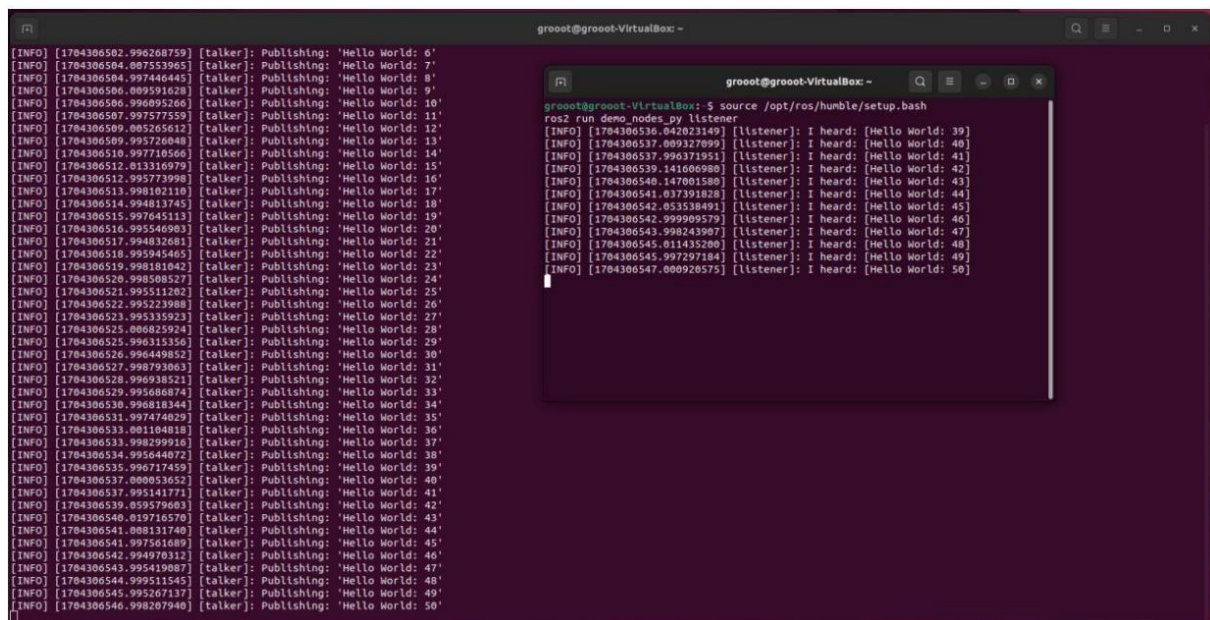
- Spin:

```
ros::spin();
```

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



The screenshot shows a terminal window with two panes. The left pane displays the output of a 'talker' node, which is publishing 'Hello World' messages at regular intervals. The right pane shows the output of a 'listener' node, which is receiving these messages and printing them to the console. The messages are timestamped and include the node's IP address and the message content.

```
grooot@grooot-VirtualBox: -  
[INFO] [1704306502.996268759] [talker]: Publishing: 'Hello World: 0'  
[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] [1704306510.997710566] [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] [1704306516.995546983] [talker]: Publishing: 'Hello World: 20'  
[INFO] [1704306517.994032681] [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] [1704306521.995511282] [talker]: Publishing: 'Hello World: 25'  
[INFO] [1704306522.995223988] [talker]: Publishing: 'Hello World: 26'  
[INFO] [1704306523.995335923] [talker]: Publishing: 'Hello World: 27'  
[INFO] [1704306525.006825924] [talker]: Publishing: 'Hello World: 28'  
[INFO] [1704306525.996315356] [talker]: Publishing: 'Hello World: 29'  
[INFO] [1704306526.996489652] [talker]: Publishing: 'Hello World: 30'  
[INFO] [1704306527.998793063] [talker]: Publishing: 'Hello World: 31'  
[INFO] [1704306528.996938521] [talker]: Publishing: 'Hello World: 32'  
[INFO] [1704306529.995668874] [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] [1704306539.059579603] [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] [1704306542.994970312] [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'  
  
grooot@grooot-VirtualBox: -  
grooot@grooot-VirtualBox: $ source /opt/ros/humble/setup.bash  
ros2 run demo_nodes_py listener  
[INFO] [1704306536.042023149] [listener]: I heard: [Hello World: 39]  
[INFO] [1704306537.009327999] [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] [1704306543.998243907] [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]
```

Talker :

Fungsi `talker` dalam ROS 2 adalah salah satu dari demo_nodes_cpp, yang merupakan bagian dari paket demo_nodes_cpp. Ini adalah contoh sederhana dari publisher ROS 2 yang memproduksi data untuk kemudian diteruskan ke topik (topic) tertentu di sistem ROS 2. Secara teknis, `talker` adalah sebuah node yang menggunakan ROS 2 untuk membuat sebuah topik (topic) dengan nama 'chatter' (topik yang sering digunakan dalam contoh) dan mengirimkan pesan-pesan string ke topik tersebut dengan interval waktu tertentu. Pesan-pesan yang dikirim oleh `talker` biasanya berisi informasi atau teks tertentu dan dikirim dalam format pesan ROS 2 yang sesuai. Dalam kasus ini, `ros2 run demo_nodes_cpp talker` digunakan untuk menjalankan node `talker` dari paket `demo_nodes_cpp`. Setelah dijalankan, `talker` akan mulai mempublikasikan pesan-pesan ke topik 'chatter' yang kemungkinan bisa didengar oleh node lain yang berlangganan pada topik yang sama. Ini adalah contoh yang sering digunakan untuk memahami bagaimana sebuah node dalam ROS 2 dapat memproduksi dan mengirimkan data melalui topik, yang dapat menjadi bagian penting dari sistem yang lebih besar dalam pengembangan robotika dengan ROS 2.

Listener :

Fungsi ``listener`` dalam ROS 2 merupakan bagian dari `demo_nodes_py`, sebuah contoh yang juga terdapat dalam paket `demo_nodes_py`. Ini adalah contoh dari subscriber pada ROS 2 yang berlangganan pada topik tertentu untuk menerima pesan-pesan yang dikirimkan oleh publisher atau node lain. Secara teknis, ``listener`` adalah sebuah node yang menggunakan ROS 2 untuk berlangganan pada topik `'chatter'`. Saat sebuah node lain, seperti ``talker``, mempublikasikan pesan-pesan ke topik `'chatter'`, node ``listener`` akan menerima pesan-pesan tersebut dan menjalankan fungsi tertentu sebagai respons terhadap pesan-pesan yang diterima.

Dalam kasus ini, perintah ``ros2 run demo_nodes_py listener`` digunakan untuk menjalankan node ``listener`` dari paket ``demo_nodes_py``. Setelah dijalankan, ``listener`` akan berjalan dalam mode yang siap untuk menerima pesan-pesan yang dipublikasikan ke topik `'chatter'` oleh node lain yang mempublikasikan informasi ke topik yang sama.

Contoh ini berguna untuk memahami bagaimana sebuah node dalam ROS 2 dapat berlangganan pada topik untuk menerima dan merespons pesan-pesan yang dipublikasikan oleh node lain. Hal ini sangat penting dalam pengembangan sistem robotika yang kompleks di ROS 2, karena memungkinkan komunikasi antara berbagai komponen dalam sistem secara asinkron.

Pertanyaan

1. Jenis-jenis Protokol Komunikasi antar Node yang Didukung oleh ROS

ROS mendukung beberapa protokol komunikasi antar node dan dua yang utama adalah

- Publish/Subscribe (Pub/Sub)
Node berkomunikasi melalui topik yang memiliki nama tertentu. Penerbit menerbitkan pesan pada suatu topik, dan pelanggan menerima pesan tersebut dengan berlangganan pada topik yang sesuai. Cocok untuk komunikasi satu-ke-banyak, di mana beberapa node perlu menerima informasi yang sama.
- Service/Client
Node berkomunikasi melalui layanan, di mana satu node menyediakan layanan, dan node lain dapat membuat permintaan kepada layanan tersebut. Cocok untuk komunikasi permintaan-respon, di mana suatu node membutuhkan tindakan atau informasi tertentu dari node lain.

2. Perbedaan antara Perintah `roslaunch` dan `roslaunch`?

`roslaunch`:

- Digunakan untuk langsung menjalankan sebuah node ROS.
- `roslaunch <nama_paket> <nama_node>`
- Contoh: `roslaunch turtlesim turtlesim_node`
Sederhana dan langsung untuk menjalankan node secara individual.

`roslaunch`:

- Digunakan untuk meluncurkan satu atau lebih node, termasuk konfigurasi dan parameter.
- `roslaunch <nama_paket> <nama_file_launch>`
- Contoh: `roslaunch my_package my_launch_file.launch`
Memungkinkan untuk konfigurasi yang lebih kompleks dan meluncurkan beberapa node sekaligus.

3. Perbedaan antara ROS Topics dan Services dalam Operasinya

- ROS Topics:
Menggunakan model publish/subscribe di mana node dapat menerbitkan pesan pada topik atau berlangganan pada topik untuk menerima pesan yang diterbitkan. Komunikasi asinkron dan satu-ke-banyak.
- ROS Services:

Melibatkan pemanggilan layanan oleh satu node untuk meminta layanan dari node lain, yang kemudian memberikan respons. Komunikasi sinkron dan satu-ke-satu.

4. Perbedaan antara ROS Services dan Actionlib dalam Operasinya:

ROS Services:

Layanan berorientasi pada permintaan dan respons tanpa adanya kemajuan terkait waktu. Berguna untuk operasi yang memiliki hasil yang dapat diprediksi dan diukur dengan baik.

Actionlib:

Actionlib menyediakan layanan yang memungkinkan pemantauan kemajuan dan pembatalan tugas yang berjalan untuk operasi yang memerlukan waktu yang lama. Berguna untuk tugas yang memerlukan pemantauan kemajuan dan dapat dihentikan, seperti navigasi atau gerakan robot.