Jawaban 1

Oleh Muhammad Akmal
NIM 19624235 | STEI-K

A TENTANG ROBOT OPERATING SYSTEM (ROS)

ROS (*Robot Operating System*) adalah perangkat pengembangan perangkat lunak sumber terbuka (*open source software development kit*) untuk penerapan robotika. ROS menawarkan platform perangkat lunak standar bagi para pengembang di berbagai industri yang akan membantu mereka mulai dari penelitian dan pembuatan prototipe hingga penerapan dan produksi.

ROS bukanlah OS seperti yang tertera pada namanya, namun berupa kumpulan *library* dan *tools* yang berguna bagi developer dalam mengembangkan robot. ROS dapat mempermudah *developer* mengembangkan robot dengan mengintegrasikan berbagai komponen robot, seperti sensor, motor, kamera, dan lain-lain tanpa perlu menghubungkannya secara manual.

ROS menggunakan *hardware abstraction layer* untuk memisahkan *driver* perangkat keras dari komponen perangkat lunak tingkat tinggi seperti sistem navigasi. Setiap komponen robot memiliki ROS *package* yang bertugas seperti *interpreter* antara ROS dengan komponen fisik tersebut. Sementara itu, untuk sistem yang kompleks seperti *drone* atau tangan robot, ROS menggunakan *framework control* yang memisahkan logika kontrol dengan gerakan kasar.

B PERBEDAAN ROS DAN ROS 2

ROS 2, seperti namanya, merupakan kelanjutan dari ROS orisinal (ROS 1) yang didesain untuk meningkatkan kemampuan ROS 1 yang masih terbatas dalam skalabilitas, performa, kontrol *real-time* dan dukungan antar *platform*. Secara teknis, perbedaan utama antara ROS 1 dan ROS 2 adalah sebagai berikut

Fitur	**ROS 1**	**ROS 2**
Arsitektur	ROS Master-Slave Architecture & XML-RPC	Data Distribution Service (DDS)
Bahasa Pemrograman	C++ dan Python 2	C, C++, C#, Java, Python 3.5
Dukungan *Platform*	Linux (Ubuntu)	MacOS, Windows, Linux
Real-Time Support	Terbatas	Tersedia *Built-in*
Modularitas	Monolitik	Modular dan fleksibel
Quality of Services Policy	Tidak tersedia	Dapat dikustomisasi
Development Space	Tersedia	Tidak tersedia

Terlihat bahwa salah satu kekurangan terbesar ROS 1 adalah tidak adanya dukungan *real-time control*. Banyak proyek robotika yang memerlukan kemampuan *real-time* dan responsivitas yang cepat. Untungnya, ROS 2 telah memungkinkan developer mengonfigurasi *real-time Quality of Service (QoS) Policy* karena menggunakan *middleware* DDS. Selain itu, ROS 2 juga lebih unggul dalam komunikasi dan skalabilitas. Developer bisa memprioritaskan pesan tertentu melalui jaringan agar dapat dikirimkan dengan baik dan cepat. Sementara dari segi skalabilitas, ROS 2 lebih unggul karena arsitekturnya yang menggunakan DDS sebagai *middleware* layer.

Dari segi keamanan, ROS 1 jelas kurang karena tujuan awal pengembangannya adalah untuk penelitian dan eksperimen. ROS 2 sudah menerapkan sistem keamanan dengan auntentikasi dan enkripsi sehingga komunikasi antar *node* dalam suatu jaringan dapat dipercaya. Selain itu, terdapat kontrol akses yang membatasi kemampuan tiap *node* untuk mencegah penyerangan terhadap sistem.

Terakhir, ROS 2 lebih mudah untuk dipelihara dalam jangka panjang karena modularitasnya. Selain itu, ROS 2 juga terus dikembangkan sehingga lebih terjamin mendapatkan pembaruan daripada ROS 1 yang hanya didukung hingga 2025. Terlebih lagi, ROS 2 mendukung berbagai platform seperti Windows, MacOS, dan Linux Ubuntu sehingga memudahkan developer mengakses kit dari berbagai platform. Alasan-alasan inilah mengapa developer cenderung memilih ROS 2 daripada ROS 1 untuk proyek robotika baru.

C PENTINGNYA SIMULASI ROBOTIKA

Simulasi robotika perlu dilakukan sebelum mengembangkan robot secara fisik. Alasan yang paling jelas adalah keuntungan biaya. Daripada langsung membangun robot fisik yang memerlukan biaya besar, simulasi robotika menghilangkan biaya karena hanya menguji desain di lingkungan virtual. Selain itu, pengujian sistem robotika baru langsung pada robot fisik bisa meningkatkan risiko keamanan ketika terjadi malafungsi. Simulasi memungkinkan pengembang untuk menguji tanpa khawatir akan kecelakaan, kerusakan, atau bahaya keselamatan yang bisa membahayakan orang atau merusak peralatan.

Keuntungan lainnya dari simulasi adalah dari segi waktu. Developer dapat merancang, menguji, dan memodifikasi perilaku robot secara *real-time* tanpa perlu modifikasi fisik. Hal ini memungkinkan prototipe yang cepat dan peningkatan desain yang lebih cepat daripada mengubah desain robot fisik. Selain itu, simulasi robot juga memungkinkan pengembang menemukan cacat desain lebih awal dalam proses pengembangan. Sistem simulasi bisa mendeteksi masalah seperti gerakan yang tidak seimbang, daya yang tidak cukup, atau penempatan sensor yang tidak tepat.

Poin lain yang menarik adalah simulasi robotika bisa direplikasi secara persis yang sering kali sulit dilakukan dengan robot fisik karena variasi dalam lingkungan. Poin ini penting untuk *debugging* karena dapat memberikan akurasi tinggi bagaimana robot berperilaku tanpa memperhatikan faktor lingkungan. Sebaliknya, dalam simulasi, pengujian robot juga bisa dilakukan di lingkungan yang bervariasi , seperti medan yang berbeda, kondisi cuaca, atau konfigurasi rintangan, tanpa perlu menciptakan kondisi ini secara fisik. Keragaman ini sangat penting untuk robot yang dirancang untuk beroperasi di lingkungan yang menantang seperti luar angkasa, bawah air, atau zona bencana.

Contoh kasus yang menarik dari keuntungan di atas adalah dalam pengujian robot *micro-mouse maze solver*. Terkadang, ukuran robot yang sangat kecil membuatnya mudah selip karena debu dan berisiko rusak ketika menabrak dinding dengan kecepatan tinggi. Simulasi bisa dilakukan untuk memastikan algoritma robot benar sembari mengabaikan kondisi lingkungan. Di sisi lain, ketika membuat *drone* atau robot kapal selam, pengujian di udara atau air langsung berisiko ketika developer belum bisa memprediksi perilaku robot. Simulasi akan penting untuk memastikan robot telah didesain dengan benar dan akan menekan biaya dengan menekan angka kecelakaan.

D TENTANG GAZEBO

Gazebo adalah simulator *open source* robotika 3D yang menyediakan mesin fisika tangguh, grafik berkualitas tinggi, dan kemampuan simulasi sensor. Dengan software ini, developer mampu menyimulasikan lingkungan dunia nyata dan menguji robot di ruang virtual sebelum dibuat secara fisik. Gazebo menghadirkan pendekatan baru untuk simulasi dengan perangkat lengkap berupa *library* pengembangan dan *cloud service* untuk mempermudah simulasi. Selain itu *software* ini memungkinkan iterasi cepat pada desain fisik baru di lingkungan yang realistis dengan aliran sensor fidelitas tinggi.

Secara umum, Gazebo mampu menyimulasikan lingkungan fisik bagi robot dengan menggabungkan mesin fisika (*physics engine*), *3D rendering*, serta simulasi sensor dan kontrol secara *real-time*. Mesin fisika yang digunakan mampu menyimulasikan interaksi fisik seperti tabrakan, gravitasi, gesekan, dan inersia. Kemudian, Gazebo akan me-*render* grafis dalam 3D sehingga developer dapat memvisualisasikan robot dan lingkungan sekitarnya secara real time, sehingga memudahkan *debugging*. Gazebo juga menyimulasikan sensor robot seperti kamera, LiDAR, IMU, ultrasonik, dll. Robot yang disimulasikan di Gazebo juga dapat dikontrol menggunakan motor, *servo*, atau aktuator secara *real-time*.

Berikut ini adalah cara mengintegrasikan ROS dengan Gazebo melalui terminal (Bash) di platform Ubuntu Linux:

1. Instal ROS 2

```
sudo apt update
sudo apt install ros-foxy-desktop
```

2. Instal Gazebo

```
sudo apt install ros-foxy-gazebo-ros-pkgs ros-foxy-gazebo-ros-
control
```

3. Buat ROS *Workspace*

```
mkdir -p ~/ros2_ws/src cd ~/ros2_ws colcon build source
install/setup.bash

source ~/ros2_ws/install/setup.bash
```

4. Luncurkan Gazebo dengan ROS 2

```
ros2 launch gazebo_ros gazebo.launch.py
```

E CARA NAVIGASI ROBOT DALAM SIMULASI

Navigasi robot di dunia simulasi adalah proses yang memungkinkan robot untuk bergerak dari satu titik ke titik lainnya secara otonom dengan memanfaatkan informasi tentang lingkungannya. Navigasi robot dibagi berdasarkan topiknya menjadi *mapping* (pemetaan), lokalisasi, serta algoritma perencanaan jalur.

Mapping adalah proses di mana robot membuat peta dari lingkungannya. Dalam simulasi, peta ini bisa berupa peta 2D atau 3D. Tujuan dari mapping adalah memungkinkan robot untuk melihat dunia di sekitarnya dan menghindari hambatan selama bergerak. Sementara itu, lokalisasi

adalah proses untuk menentukan posisi dan orientasi robot dalam suatu peta yang sudah ada. Tanpa lokalisasi yang akurat, robot tidak akan tahu di mana dia berada dan sulit untuk merencanakan jalur yang benar. Lokalisasi yang baik memerlukan data sensor yang kuat, seperti dari LiDAR, kamera, atau sensor ultrasonik, yang memberikan robot informasi lingkungan sekitar.

Setelah robot mengetahui posisi dan peta lingkungannya, robot perlu menentukan jalur terbaik dari titik awal ke tujuan, sambil menghindari rintangan. Beberapa algoritma perencanaan jalur yang sering digunakan di dunia simulasi meliputi *A** (A-star), Dijkstra, atau RRT (*Rapidly-exploring Random Tree*). Setelah jalur ditemukan, robot harus dapat bergerak dengan tepat sesuai jalur yang telah direncanakan.

F TRANSFORM DALAM ROS

TF (Transform) dalam konteks ROS adalah sistem koordinat yang digunakan untuk melacak posisi dan orientasi objek (seperti robot, sensor, atau komponen robot) dalam 3D. TF memungkinkan robot untuk memahami bagaimana satu bagian dari sistem terkait dengan bagian lainnya dan lingkungan sekitarnya dalam hal posisi (translasi) dan orientasi (rotasi). Dalam navigasi robot, TF memainkan peran penting dalam menyelaraskan berbagai data sensor, memandu gerakan, serta memastikan robot bergerak dengan benar sesuai posisi dan orientasinya di dunia simulasi atau nyata.

TF digunakan untuk membuat dan memelihara *tree* dari berbagai *frames* koordinat yang menggambarkan hubungan antara komponen-komponen robot dan lingkungan. Setiap *frame* merupakan sistem koordinat lokal yang menunjukkan posisi dan orientasi relatif dari suatu elemen. Dengan sistem TF, robot bisa menggabungkan berbagai data dari sensor berbeda, mengetahui posisi sensor terhadap robot, dan bahkan memahami bagaimana robot berorientasi terhadap dunia luar. TF menyediakan transformasi yang memungkinkan sistem untuk mengonversi data dari satu *frame* ke *frame* lain, baik secara langsung maupun melalui rantai *frame* (TF *tree*).

Secara umum, elemen dalam TF dibagi 3, *frames*, *transforms*, dan TF Tree. *Frames* adalah sistem koordinat untuk berbagai elemen robot, seperti roda, kamera, atau basis robot. Sementara *transforms* merupakan matriks yang mendefinisikan posisi (translasi) dan orientasi (rotasi) dari satu *frame* relatif terhadap *frame* lain. TF menangani transformasi tersebut secara otomatis dalam ROS. Hubungan seluruh *frames* tersebut disimpan dalam struktur *tree* dan semua *frame* lainnya terhubung sebagai cabang, hal inilah yang diberi nama TF Tree.

Cara kerja TF secara umum dibagi menjadi 2, yakni *broadcasting* dan *listening*. Dalam *broadcasting*, setiap elemen TF menyiarkan transformasi frame relatif terhadap *frame* lainnya secara terus-menerus yang disebut TF *broadcaster*. Sementara dalam *listening*, TF *listener* komponen lain mendengarkan hubungan antara *frame* tersebut dan menggunakan informasi tersebut untuk mengetahui posisi dan orientasi elemen-elemen yang terhubung

Contohnya penggunaan TF di sini adalah jika developer ingin mengetahui posisi kamera (*camera_frame*) relatif terhadap basis robot (*base_link*), TF akan secara otomatis memberikan transformasi antara kedua frame tersebut berdasarkan hubungan yang ditentukan.