A

ROS atau Robot Operating System adalah sebuah kerangka perangkat lunak untuk mengembangkan dan mengelola aplikasi robot. Seluruh perangkat robot akan pasti memerlukan pemrograman agar dapat berjalan. Namun tanpa perantara untuk mempermudahkan komunikasi dalam waktu nyata antara bagian-bagian robot pasti akan mempersulit pemrograman karena terdapat banyak coding yang harus saling terhubung satu sama lain. ROS mempermudahkan ini dengan memecah robot menjadi “nodes”, dimana nodes tersebut memiliki fungsi untuk mengendalikan bagian khusus robot, seperti sensor, kamera, dan aktuator. Pemisahan robot menjadi node-node membantu programmer untuk memperbaiki bagian-bagian robot tanpa mengacaukan keseluruhan program. Hal ini dikarenakan ketika tidak menggunakan ROS, semua kode robot harus dimasukkan ke dalam satu file yang dimana jika satu baris kode diubah maka keseluruhan kode dapat berubah juga sehingga dapat terjadi error.

ROS memiliki peran penting dalam pengembangan robotika modern. Hal ini dikarenakan oleh beberapa alasan, yaitu :

1. ROS mempermudah pengembangan aplikasi robot; Alasan telah dijelaskan pada paragraf sebelumnya.
2. ROS itu open source; Open source artinya source code yang terbuka untuk publik, dimana publik dapat melihat dan menggunakan kembali source code yang dibuat oleh orang lain. ROS yang bersifat open source mempermudahkan programmer untuk membuat sebuah robot karena coding untuk sebuah bagian atau bahkan seluruh robot sudah terdapat di website-website umum, seperti github dan dapat langsung digunakan sehingga menghemat waktu. Penghematan waktu ini menyebabkan programmer dapat memfokuskan waktu mereka lebih banyak ke pembuatan robot yang lebih kompleks lagi.
3. ROS terdapat plugins; Plugins merupakan koneksi antara software yang digunakan dengan software yang lain untuk meningkatkan fungsionalitas sebuah bagian robot. Misalnya, software gazebo dimana software tersebut dapat simulasikan robot secara 3D sebelum tes dengan robot asli sehingga dapat menghemat waktu juga.
4. ROS merupakan media standar; ROS merupakan media standar dimana seluruh pengguna akan peraturan yang terdapat di ROS untuk mengembangkan aplikasi robot. Hal ini mempermudahkan programmer untuk membagi coding untuk robot dengan programmer lain sehingga seluruh komunitas pengembang aplikasi robot dapat bekerja sama untuk membangun robot yang lebih kompleks.
5. ROS dapat menggunakan berbagai bahasa coding; ROS untuk per node dapat menggunakan berbagai bahasa coding, contohnya Python, C++, Java, dan JavaScript. Hal ini mempermudah programmer untuk memulai membuat aplikasi robot karena programmer dengan pengetahuan bahasa coding yang bervariasi tidak harus mempelajari bahasa coding baru. Oleh karena itu, kolaborasi antar programmer untuk membuat aplikasi robot yang lebih kompleks lebih mudah.

Dari alasan-alasan yang disebutkan, masuk akal untuk mengatakan ROS memiliki peran yang penting atau utama dalam pengembangan robotika modern. ROS memiliki peran utama

dalam mempermudah dan menghemat waktu programmer untuk mengembangkan aplikasi robot yang lebih kompleks dengan fitur-fiturnya, yaitu mempermudah manajemen coding, open source, terdapat plugins, media standar, dan menggunakan berbagai bahasa coding.

B

Banyak pengembang aplikasi robot yang lebih memilih ROS 2 daripada ROS 1. Terdapat beberapa alasan untuk hal tersebut, salah satunya adalah ROS 2 sudah menggunakan Data Distribution Service dibandingkan dengan ROS 1 yang menggunakan ROS Master. ROS master berperan sebagai medium untuk node-node robot saling berkomunikasi antara satu sama lain. Namun apabila satu node tidak dapat berkomunikasi dengan ROS Master, maka aplikasi robot tidak dapat berjalan. Kekurangan ini dibetulkan di ROS 2 dengan DDS, dengan DDS komunikasi antara node-node robot dapat dilakukan secara independen tanpa medium. Hasilnya apabila satu node tidak dapat berkomunikasi dengan node lain, robot masih dapat berjalan namun tidak optimal. Selain itu, DDS membuat aplikasi robot menjadi lebih efisien karena komunikasi antar node diperpendek tanpa medium ROS master.

Pemindahan dari ROS master menjadi DDS di ROS 2 membuat aplikasi robot lebih aman. Hal tersebut karena DDS merupakan protokol yang ditetapkan dan diuji yang memberikan jaminan keamanan sehingga data yang dikirimkan antar node sudah terjamin lebih aman dari hacker. ROS 1 tidak seaman ROS 2 karena ROS 1 pertama dirancang untuk keperluan pengembangan robot di kampus bukan tujuan komersial sehingga keamanan tidak diperhatikan.

Alasan lain pengembang aplikasi robot lebih memilih ROS 2 dibanding dengan ROS 1 adalah pemeliharaan jangka panjangnya. ROS 2 merupakan versi terbaru dari ROS, maka masuk akal ROS 2 memiliki pemeliharaan jangka panjang. Sedangkan ROS 1 secara resmi didukung hanya hingga tahun 2025. ROS 2 karena dirancang dari awal untuk memenuhi kekurangan ROS 1 akan memiliki kemampuan untuk menjadi kompatibel dengan aplikasi industrial sehingga mendukung pembangunan robot yang lebih kompleks.

Selain alasan yang diberikan diatas, ROS 2 mempunyai library dasar yang ditulis dalam C - rcl (perpustakaan klien ROS) - dengan perpustakaan yang dibangun di atasnya. Hal ini menyebabkan ROS 2 mampu menawarkan lebih banyak dukungan bahasa coding. Dibandingkan dengan ROS 1 yang hanya memiliki dua library yang terpisah, yaitu roscpp untuk C++ dan rospy untuk Python yang fiturnya tidak setara satu sama lain.

C

Keuntuntungan dari melakukan simulasi robot sebelum perancangan robot adalah penghematan waktu dan biaya. Hal ini karena dengan simulasi robot apabila terdapat eror atau sesuatu yang tidak inginkan, maka kita dapat memecahkan masalah tersebut baik dari masalah coding maupun kerangka fisik robot. Jika tidak melakukan tahap simulasi, terdapat kemungkinan pengembang robot terpaksa membongkar robot untuk mencari masalah dan merangkit ulang robot. Contohnya, saya sedang membuat sebuah robot yang dapat bergerak dan mengambil barang. Namun, saya tidak melakukan simulasi lebih dahulu terhadap robot dan langsung membeli bagian-bagian robot serta merangkit robot. Ternyata saat saya memulaikan

robot, robot tersebut mengalami masalah dalam mengambil sebuah benda. Saat ditelusuri masalahnya, ternyata motor yang bertanggung jawab untuk menggerakkan tangan robot salah spesifikasi. Selain itu, coding untuk bagian kamera terdapat kesalahan yang dimana robot sulit untuk mengukur seberapa besar sesuatu benda terdapat di depannya. Hal tersebut menyebabkan saya membeli motor baru dengan spesifikasi yang benar dan merangkit ulang robot, serta memperbaiki coding untuk kamera robot agar mengukur sesuatu benda yang berada di depannya dengan benar. Pemborosan waktu dan uang dapat dicegah apabila saya melakukan simulasi robot terlebih dahulu.

D

Gazebo merupakan simulator 3D untuk robot dengan lingkungan dan fisika yang realistis, serta dapat visualisasikan perilaku robot dalam lingkungan tersebut. Aplikasi Gazebo sendiri sudah terintegrasi dengan ROS. Gazebo dapat mensimulasikan lingkungan fisik bagi robot karena simulator ini memberi kebebasan kepada pengguna untuk membangun lingkungan yang di dalam simulator, serta Gazebo sendiri menyediakan berbagai sensor, aktuator, dan objek untuk melakukan uji coba terhadap robot dalam berbagai skenario.

Langkah-langkah untuk mengintegrasikan ROS dengan Gazebo untuk mengontrol Robot dalam simulasi, sebagai berikut(dengan anggapan ROS telah di install) :

1. Mengunduh ROS dan Gazebo. Gazebo dapat di unduh dengan ketik `sudo apt install ros-foxy-gazebo-ros-pkgs` di command prompt apabila ROS sudah di unduh terlebih dahulu
2. Membuat paket ROS. Beri nama paket ROS, contohnya : my_simulations
3. Buat folder untuk peluncuran dan dunia dalam paket my_simulations tersebut
4. Buat file (dunia.launch) dalam folder peluncuran
5. Masukkan konten pada file tersebut (klik alat lalu IDE)
6. Buat file (empty_world.world) dalam folder dunia
7. Tambahkan konten pada file tersebut hingga paket ROS terbuat
8. Menjalankan simulasi Gazebo menggunakan ROS.
   Klik Simulasi lalu pilih File dan pilih my_world.launch. (Ini akan secara otomatis memuat versi web gazebo, yang disebut gzweb)

E

Di dalam di dunia nyata maupun simulasi, kemampuan robot dalam bernavigasi akan menjadi sangat amat penting dalam mencapai tujuannya. Dalam melakukan navigasi, robot akan melakukan pemetaan, lokalisasi, dan Koreksi.

1. Pemetaan

   Dalam lingkungan yang tidak dikenal oleh robot, robot akan pertama membuat sebuah peta yang biasanya berbentuk 2D (Bisa 3D tergantung aplikasinya) agar tidak mudah tersesat dan mencapai tujuan. Untuk membuat

peta ini, pengembang robot biasanya akan menggunakan konsep SLAM (Simultaneous Localization and Mapping). SLAM adalah metode yang digunakan untuk kendaraan otonom dimana kendaraan tersebut membuat peta dan melokalisasi kendaraan otonom di peta tersebut secara bersamaan.

SLAM tergantung pada sensor-sensor yang terdapat pada robot, dimana informasi yang didapatkan oleh sensor seperti jarak robot terhadap sesuatu benda (landmark) akan digunakan untuk membuat sebuah peta. Sensor yang paling digunakan untuk konsep SLAM ini adalah sensor LiDAR (light detection and ranging). Teknologi Lidar menggunakan energi cahaya untuk mengumpulkan data dari suatu permukaan dengan menembakkan laser ke suatu target dan mengukur berapa lama waktu yang diperlukan agar sinyal tersebut kembali.

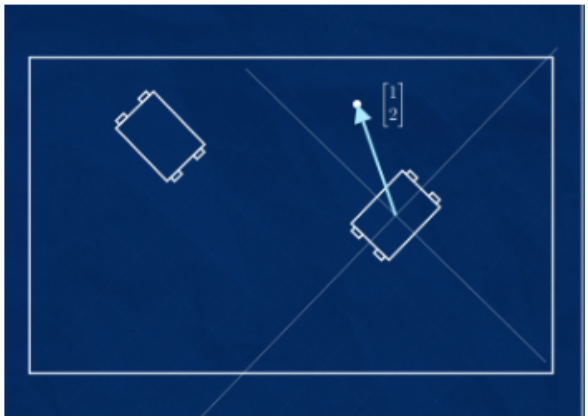
2. Lokalisasi

   Setelah robot membuat peta, robot akan menentukan posisi robot di peta tersebut. Lalu, robot akan lanjut menelusuri lingkungan sambil membangun peta hingga dapat mencapai posisi tujuan. SLAM dapat melakukan pemetaan dan lokalisasi, namun untuk lokalisasi terdapat metode yang lebih terkenal adalah AMCL(Adaptive Monte Carlo Localization). AMCL menggunakan filter partikel untuk memperkirakan posisi robot berdasarkan sensor, seperti LiDAR yang dijelaskan sebelumnya
3. Koreksi

   Saat robot menelusuri lingkungan, robot terkadang dapat tersesat dan/atau robot dapat menemukan lingkungan yang sudah di petakan ternyata berubah, contohnya memindah sebuah benda di ruangan. Dalam kondisi tersebut robot akan menggunakan gabungan data dari sensor, peta, dan algoritma(seperti A.I dan SLAM) untuk mengevaluasi lingkungan dan mengubah rute agar tujuan tercapai.

F

TF(Transform) dalam konteks ROS merupakan library untuk transformasi dari satu frame ke frame lainnya, di ROS sekarang tf diubah namanya menjadi tf2(Transform versi 2). Transform berguna ketika salah satu bagian dari robot atau robot lain ingin berinteraksi.

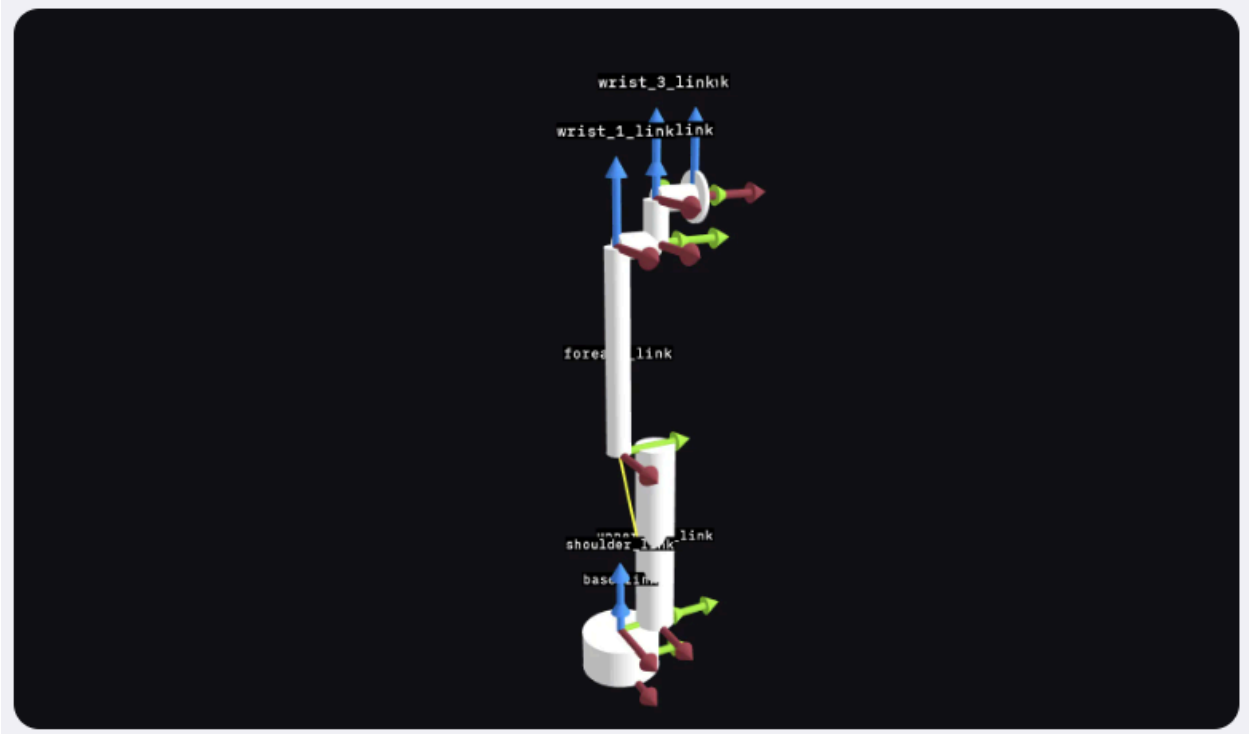


Sumber : https://articulatedrobotics.xyz/tutorials/ready-for-ros/tf/

Misalnya, terdapat dua robot di dalam sebuah ruangan, robot 1 (robot kanan) menemukan benda yang menarik dan diinformasikan kepada robot 2(robot kiri) yang berada di ruangan yang sama. Robot 2 ingin menuju ke benda tersebut, namun letak benda tersebut relatif terhadap robot 2 tidak diketahui. TF dalam situasi ini memainkan peran dalam mengubah posisi benda yang relatif terhadap robot 1 menjadi relatif terhadap robot 2 sehingga robot 2 dapat menuju ke benda menarik itu.

Di dalam robotika, frame merupakan koordinat sistem yang deskripsikan posisi dan orientasi objek pada axis x, y, dan z(apabila dalam 3D). Di ROS mengharuskan setiap frame memiliki frame_id uniknya sendiri, dan terdiri dari beberapa frame untuk setiap komponen robot (yaitu anggota tubuh, sensor), objek yang terdeteksi, dan pemain di dunia robot.

Dalam TF, dunia biasanya memiliki frame yang dimana posisinya selalu konstan. Pada dasar robot biasanya terdapat frame dan komponen dari dasar robot terdapat juga frame yang biasa dipanggil child frame karena frame ini berasal dari frame parent, yaitu dasar robot. Apabila komponen yang disebut child frame memiliki turunan komponen, misalnya tangan robot yang memiliki sensor, maka turunan komponen(sensor) akan menjadi child frame dan komponen itu sendiri menjadi parent frame. Dan apabila turunan komponen memiliki turunan lagi, maka penamaan frame akan sama sehingga membuat pohon transform. Transform akan melakukan translasi dan rotasi satu frame ke frame lainnya di dalam pohon transform sehingga posisi dan orientasi sesuatu frame akan lebih mudah ditentukan.



Sumber : https://foxglove.dev/blog/understanding-ros-transforms