URO TEST ITB

SOAL NOMOR 1

Muhammad Faiz Alfada Dharma - 19624244

a) ROS (Robot Operating System) adalah sebuah *framework* atau kerangka fleksibel yang dirancang sedemikian rupa untuk pengembangan perangkat lunak / *software* untuk robotika. ROS adalah sebuah *middleware*, yaitu perangkat lunak yang berada di antara sistem operasi dan aplikasi yang berjalan di dalamnya, yang bekerja di atas sebuah *conventional operating system* / sistem operasi konvensional. ROS menyediakan serangkaian pustaka / *libraries* dan alat untuk membantu membuat robot yang kompleks.

   Dalam pengembangan robotik modern, ROS (Robot Operating System) telah merevolusi bagaimana cara robot dikembangkan, dikendalikan, dan diintegrasikan ke dalam berbagai aplikasi. ROS memungkinkan *developers*, peneliti, dan ahli robotika untuk tidak lagi terikat dengan detail perangkat keras / *hardware* yang kompleks, sehingga mereka bisa lebih fokus pada inovasi dan kolaborasi.

   Setiap sistem robotik pada dasarnya terdiri dari berbagai elemen perangkat keras yang berbeda yang hanya dapat bekerja sama ketika mereka saling terhubung. Dengan adanya ROS (Robot Operating System), hal tersebut memungkinkan untuk mengembangkan perangkat lunak / *software* robotik yang diperlukan sehingga elemen perangkat keras dapat terhubung dan saling berkomunikasi agar dapat menyelesaikan tugas yang telah dirancang.

   Sensor dan actuators adalah sebuah alat yang memungkinkan robot untuk mendeteksi dan berinteraksi dengan lingkungan sekitarnya. Dengan adanya ROS (Robot Operating System), ROS memungkinkan integrasi sensor dan aktuator robot melalui komunikasi, *abstraction*/ simplifikasi, dan independensi perangkat keras / *hardware independence, reusable packages and tools*, simulasi, serta pengujian *testing.* ROS (Robot Operating System) *abstracts* / menyederhanakan komponen perangkat keras / *hardware* dan perangkat lunak / *software* sehingga mempermudah interaksi dengan sensor, aktuator dan perangkat robotik. Ini memungkinkan kepada *developers* untuk fokus pada pemrograman tingkat tinggi daripada kerumitan perangkat keras tingkat rendah / *low-level hardware intricacies* dan mempercepat proses pengembangan robot.

ROS (Robot Operating System) memfasilitasi integrasi sensor dan aktuator dalam pengembangan robotika dengan menyediakan kerangka kerja modular / *modular framework.* Kerangka atau *framework* ini memungkinkan koneksi dan koordinasi antara berbagai sensor dan aktuator sehingga *developers* dapat membangun sistem robotika yang kompleks dengan fungsi dan efisiensi yang lebih baik

Oleh karena itu perlu adanya ROS (Robot Operating System) sebagai peran utama dalam pengembangan robot modern.

b) Apa perbedaan utama antara ROS dan ROS2

Perbedaan antara ROS1 dan ROS2 dilihat dari 3 kategori utama, yaitu dalam hal performa, keamanan, dan pemeliharaan jangka panjang.

1. Performa
    - Pada ROS1, ROS selalu mulai dengan menjalankan master, Master-Slave ROS, sebelum menjalankan sebuah *node.* Pada ROS2, ROS 2 menggunakan Data Distribution Service (DDS), yang dirancang untuk memberikan efisiensi dan keandalan yang lebih tinggi dimana setiap *node* memiliki kemampuan untuk menemukan *node* lainnya sehingga dapat langsung memulai sebuah *node* tanpa harus mengkhawatirkan apakah master sudah berjalan atau belum.
    - ROS2 juga memperkenalkan konsep *Quality of Service* (QoS) yang memungkinkan pengaturan yang lebih mendetail terhadap komunikasi antar-*node* untuk aplikasi robotik kompleks. ROS 2 memungkinkan konfigurasi aliran data yang memengaruhi cara data dikirim dan diterima. Ini mencakup pengaturan untuk keandalan pesan, tenggat waktu, dan prioritas, yang dapat memastikan pesan-pesan penting dikirim tepat waktu.
2. Keamanan
    - Aplikasi robotik sangatlah beragam dan dalam beberapa kasus robot mungkin perlu berkomunikasi melalui jaringan yang tidak aman. Hal ini menjadi masalah besar. ROS2 menggunakan Data Distribution Service (DDS) sebagai protokol jaringan utama untuk komunikasi internal di ROS2. Protokol DDS sudah teruji dan terbukti memberikan jaminan keamanan dan juga keandalan, tidak seperti pada ROS1. Hal ini berbeda dengan protokol TCP/UDP (*TCP/UDP custom protocol)* yang dikembangkan di ROS1. Protokol tersebut tidak memberikan

jaminan keamanan yang banyak seperti yang disediakan oleh DDS pada ROS2.

3. Pemeliharaan jangka panjang
    - ROS2, dengan adanya performa yang berkelanjutan dan memiliki sistem keamanan, ROS2 akan bermanfaat untuk pemeliharaan jangka panjang robot. DDS *communication* meningkatkan ketahanan jaringan dan keandalan yang tinggi, terutama dalam kondisi yang sedang tidak stabil. Dengan adanya dukungan dari berbagai multi-platform, seperti Linux, Windows, macOS dan integrasinya yang lebih mudah dengan *cloud resources*, ROS2 memungkinkan pengelolaan dan pengawasan sistem robotik yang lebih mudah.

Pengembang robotik cenderung memilih ROS2 untuk proyek baru karena ROS2 menawarkan performa yang lebih baik, dukungan untuk aplikasi real-time, dan sistem komunikasi yang lebih aman melalui DDS (Data Distribution Service). Selain itu, ROS2 memiliki fitur *Quality of Service* (QoS) yang memberikan kontrol lebih terhadap komunikasi antar-*node*.

c) Simulasi robotik penting dalam pengembangan robot. Simulasi robotik memungkinkan *developers* untuk melakukan sebuah pengujian dan validasi desain robot, *control algorithm,* dan interaksi dengan berbagai elemen dalam lingkungan simulasi yang aman / *risk-free setting.* Dengan menggunakan perangkat lunak simulasi / *simulation software,* robot dapat dianalisis dalam berbagai kondisi tanpa memerlukan prototype fisik / *physical prototypes* yang akan memakan waktu dan biaya.

Simulasi robot adalah kunci dalam memajukan pengembangan dan penerapan sistem robotik. Simulasi robot memungkinkan robot untuk dianalisis secara mendetail dari mekanik, elektronik, serta perangkat lunak / *software* secara virtual sehingga *developers* mampu mengidentifikasi desain yang bermasalah dan mengoptimalkan kinerja tanpa risiko di dunia nyata. Contoh simulasi yang dilakukan, seperti *environmental modeling* dan *virtual commissioning.*

1. *Environmental modelling* adalah simulasi dunia fisik dimana robot akan beroperasi. Ketika menguji robot menggunakan *environmental modelling*, robot akan diuji proses navigasi dan pelaksanaan tugas lainnya. Simulasi ini menjadi efisien untuk menguji robot dalam beragam kondisi tanpa memakan biaya untuk pengembangan robot dengan melakukan pengujian di dunia nyata

2. *Virtual commissioning*. Dalam pengembangan robot, simulasi *virtual commissioning* dapat mengurangi waktu pengembangan karena memungkinkan untuk menguji dan melakukan validasi *control system* dalam *virtual world* sebelum menerapkannya di dunia nyata. Dengan simulasi ini, dapat mengidentifikasi dan mengeliminasi kesalahan desain lebih awal sehingga mengurangi kesalahan di dunia nyata yang kemungkinan akan memakan banyak waktu.

d) Gazebo adalah sebuah *open-source robot simulation environment,* alat yang digunakan untuk mensimulasikan robot dengan memungkinkan para *developers* membuat *virtual world* dan mensimulasikan robot di dalamnya. Meskipun terpisah dari ROS, Gazebo terintegrasi dengan baik. Dengan Gazebo, *developers* dapat membuat sebuah *virtual world* dan memuat versi simulasi dari robot ke dalamnya. Hal ini memungkinkan pengujian algoritma / *algorithm* dan gerakan robot sebelum diterapkan pada perangkat keras / *hardware*.

Berikut adalah langkah-langkah dasar mengintegrasikan ROS dengan Gazebo untuk mensimulasikan robot:

1. Menginstalasi Gazebo
    - Cara termudah untuk menginstal Gazebo dengan *dependencies* dan *plugin* yang benar agar dapat berfungsi dengan ROS adalah dengan menggunakan perintah berikut:

      sudo apt install ros-foxy-gazebo-ros-pkgs
2. Membuat URDF/SDF/World Files
    - Gazebo menggunakan file SDF untuk menggambarkan robot, lingkungan, dan model dalam simulasi. Sementara ROS menggunakan URDF untuk mendeskripsikan robot. Gazebo dapat secara otomatis mengonversi URDF ke SDF menggunakan perintah *gz sdf -p.* Namun, hasil konversi ini sering memerlukan modifikasi tambahan agar dapat berfungsi sepenuhnya dalam simulasi.
3. Tambahkan *plugin*
    - Setiap kali Gazebo perlu berinteraksi dengan ROS, Gazebo harus menggunakan *plugin*. *Plugin* adalah *extra code libraries* / kode tambahan yang bisa diinstal, lalu dirujuk dalam file URDF dan akan dijalankan oleh Gazebo.
4. *Launching* Gazebo
    - Menggunakan file *launch* ROS / ROS *launch file* untuk meluncurkan Gazebo dan membuka sebuah *simulation environment.*
5. *Spawning* Robot

- Jalankan *robot_state_publisher* untuk mempublikasikan file URDF / *URDF file* dan memunculkan robot di Gazebo
- Setelah robot dimunculkan di dalam Gazebo, ada dua hal utama yang perlu dilakukan oleh untuk bisa berintegrasi dengan ROS yang lebih luas:

  – Mensimulasikan aktuator virtual / *virtual actuators* berdasarkan beberapa jenis input perintah / *command input*

  – Mempublikasikan *resulting joint positions / angles* ke topik *joint_states*

e) Dalam dunia simulasi, navigasi robot melibatkan dua konsep dasar, yaitu *mapping* dan *localization. Mapping* dan *localization* biasa menggunakan teknologi bernama SLAM (Simultaneous Localization and Mapping). Simultaneous Localization and Mapping berupaya membuat robot atau *autonomous vehicle* memetakan area yang tidak dikenal secara bersamaan menentukan di mana posisi robot tersebut berada di dalam area tersebut.

1. *Localization*

   Ketika *autonomous robot* dinyalakan, hal pertama yang dilakukan adalah mengidentifikasi posisi / lokasinya. Dalam skenario *localization*, robot dilengkapi dengan sensor untuk mengidentifikasi lingkungan sekitarnya dan memantau juga pergerakannya. Dengan menggunakan data dari sensor tersebut, robot dapat menentukan di mana posisinya pada peta yang diberikan.

2. *Mapping*

   Bagi robot, membuat peta bukanlah tugas yang mudah. Untuk memulainya, jenis detektor visual / *visual detector*, seperti kamera atau sensor lidar digunakan untuk merekam lingkungan sekitar. Saat robot bergerak, ia menangkap lebih banyak informasi visual dan membuat koneksi, serta *extract features* untuk menandai beberapa titik yang dapat dikenali.

SLAM (Simultaneous Localization and Mapping) umumnya terdiri dari dua komponen utama:

1. *Range Measurement*

   Semua metode SLAM mencakup beberapa jenis *device /* perangkat atau alat yang memungkinkan robot untuk mengamati dan mengukur lingkungan sekitarnya. Hal ini dapat dilakukan dengan menggunakan kamera, sensor, dan LiDar *laser scanner technology*. Pada dasarnya, *device /* perangkat apa

pun yang dapat digunakan untuk mengukur lokasi, jarak, atau kecepatan dapat menjadi bagian dari SLAM.

2. *Data Extraction*

   Setelah melakukan *range measurement*, SLAM memerlukan perangkat lunak / *software* untuk menginterpretasi data tersebut. Proses ini melibatkan berbagai algoritma / *algorithm,* seperti *scan-matching* untuk *extract sensory information* dan mengidentifikasi lingkungan. SLAM yang berfungsi dengan baik mengintegrasikan perangkat *range measurement*, *data extraction software*, robot, serta teknologi lainnya. Semua komponen ini harus bekerja bersama dengan lancar agar robot dapat menavigasi lingkungan dengan akurat

f) TF (*Transform)* memainkan peran penting dalam robotika. Dalam ROS, TF (*Transform)* membantu melakukan transformasi koordinat / *coordinate transformation*, mengubah titik atau pengukuran dari satu *frame* ke *frame* lainnya. TF memungkinkan robot untuk mengetahui posisi dan orientasinya dalam ruang 3D dengan mempertahankan hubungan antar *frame*.

TF (*Transform)* mendefinisikan translasi dan rotasi yang diperlukan untuk berpindah dari satu *frame* sumber ke *frame* target / *target frame*, baik itu dari *parent-to-child* atau *child-to-parent.* Satu set lengkap *transform*, dari *frame* dasar membentuk *transform tree*. Dengan adanya *transform tree*, dapat dengan cepat menemukan posisi dan orientasi *frame* tertentu, tidak memperhatikan seberapa jauh tingkatnya dari *frame* dasar.

Untuk mensimulasikan robot dan mematikan bahwa robot bergerak di ROS menggunakan *transform*, prosesnya dimulai dengan membuat file URDF / URDF *file* yang mendeskripsikan robot, termasuk *links* dan *joints*-nya. *Node robot_state_publisher* digunakan untuk mengirimkan *transform* antara *links* tersebut. Untuk *joints* yang dapat bergerak, *joint_state_publisher_gui* digunakan untuk mempublikasikan status dinamis ke topik / *joint_states* yang memungkinkan kontrol interaktif menggunakan *slider*. Setelah itu, keadaan robot divisualisasikan di RViz, di mana *transform* dapat diamati secara *real-time* dan dapat memanipulasi *joint states* melalui *slider.*

Sumber dan referensi:

https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-robot-operating-system/
https://www.linkedin.com/advice/3/how-can-ros-support-integration-sensors-actuators-robotics-rlihe#:~:text=ROS%20facilitates%20the%20integration%20of,with%20enhanced%20functionality%20and%20efficiency.
https://www.cyberweld.co.uk/robot-operating-systems-an-overview
https://www.amantyatech.com/Robot_Operating_System_Powering_Robotics#:~:text=ROS%20abstracts%20hardware%20and%20software,and%20accelerating%20the%20development%20process.
https://www.mathworks.com/discovery/robot-simulation.html#:~:text=Key%20Benefits%20of%20Robot%20Simulation,Enables%20quick%20evaluation%20and%20iteration.
https://www.linkedin.com/pulse/comparison-scalability-performance-ros1-ros2-systems-fouad-anwar/
https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a
https://www.model-prime.com/blog/ros-1-vs-ros-2-what-are-the-biggest-differences
https://roboticsbackend.com/ros1-vs-ros2-practical-overview/
https://articulatedrobotics.xyz/tutorials/ready-for-ros/gazebo/
https://www.flyability.com/blog/simultaneous-localization-and-mapping
https://ouster.com/insights/blog/introduction-to-slam-simultaneous-localization-and-mapping