6 Oktober 2024

SOAL 1

Nama : Dietrich Pepalem Tarigan
NIM : 10223037

a) Apa itu ROS (Robot Operating System), dan bagaimana peran utamanya dalam pengembangan robotik modern?

Jawab:
Robot Operating System (ROS) adalah kerangka kerja perangkat lunak yang dirancang khusus untuk mengembangkan aplikasi robotika. ROS menyediakan berbagai fungsi dan alat yang memungkinkan pengembang untuk mengendalikan robot dengan mudah. Dengan menggunakan ROS, pengguna dapat menghubungkan, mengontrol, dan mengintegrasikan komponen robot secara efisien [1].

Saat melakukan penelitian untuk topik proyek PhD mereka di Stanford, Eric Berger dan Keenan Wyrobek menemukan masalah yang berulang: Para ahli robotika menghabiskan sebagian besar waktu dan energi mereka untuk menulis ulang kode, dan hanya sebagian kecil yang dihabiskan untuk inovasi sebenarnya. Untuk mengatasi hal ini, keduanya berusaha menciptakan sebuah kerangka kerja open-source yang mudah digunakan dan menyediakan alat serta pustaka yang dapat digunakan kembali oleh pengembang untuk membangun perangkat lunak robotika. Akhirnya, pada tahun 2007, lahirlah Robot Operating System (ROS).

ROS sangat *adaptable* dan *scalable*, serta dapat digunakan untuk membuat banyak hal, mulai dari peralatan rumah tangga seperti penyedot debu hingga drone, robot industri, dan kendaraan otonom. Karena sifatnya yang kolaboratif, ROS sering digunakan dalam proyek penelitian maupun komersial di bidang robotika.

"ROS has become the lingua franca of the robotics world for connecting sensors, actuators and processing elements," ujar Anthony Rowe, profesor teknik elektro dan komputer di Universitas Carnegie Mellon, kepada Built In. *"Without ROS, it would take a tremendous amount of effort to get basic robotics projects off the ground."*

Dengan menggunakan *distributed computing architecture*, ROS memfasilitasi pengembangan aplikasi robotik kompleks pada berbagai platform perangkat keras, termasuk abstraksi perangkat keras, driver perangkat, komunikasi antar proses, dan manajemen paket. ROS juga mencakup berbagai alat khusus robot, seperti visualisasi 3D, kontrol gerakan, dan navigasi [2].

Sejumlah manfaat yang ditawarkan ROS bagi pembelajaran robot interaktif, antara lain:

Modularitas: ROS memungkinkan pengguna untuk mengembangkan dan menguji modul-modul robot secara terpisah sebelum mengintegrasikannya. Hal ini memudahkan proses pembelajaran dan eksperimen dengan komponen robotik yang berbeda.

Komunitas yang Aktif: ROS memiliki komunitas yang luas dan aktif, yang memungkinkan pengguna untuk berbagi pengetahuan, pengalaman, dan kode sumber. Ini menciptakan lingkungan kolaboratif yang mendorong pembelajaran dan inovasi.

Simulasi: ROS menyediakan fasilitas simulasi yang kuat melalui Gazebo. Dengan simulasi ini, pengguna dapat menguji robot dan kode programnya tanpa memerlukan perangkat keras fisik. Hal ini memungkinkan eksperimen yang aman, efisien, dan berulang-ulang [1].

b) Apa perbedaan utama antara ROS dan ROS2, dan mengapa pengembang cenderung memilih ROS2 untuk proyek baru?

Jawab:

ROS 1

- Launched in 2007
- Single-threaded design
- Custom-messaging-system middleware
- Primarily uses C++ and Python
- Reliable, stable user base with extensive community resources
- Used for academic research and prototyping
- Runs on Linux

ROS 2

- Launched in 2017
- Multi-threaded design
- Data-distribution-service middleware
- Also uses C++ and Python, but can support other languages
- Processes requests concurrently to handle time-sensitive tasks
- Native security features, such as encryption and authentication
- Used in industrial and commercial applications
- Runs on Windows, macOS and Linux [2]

Penjelasan:

Desain Multi-threaded vs. Single-threaded
ROS 2 memiliki desain multi-threaded, memungkinkan pemrosesan permintaan secara bersamaan (concurrent). Ini membuat ROS 2 lebih efisien dalam menangani tugas-tugas yang sensitif terhadap waktu, terutama pada sistem yang memerlukan respons cepat dan real-time. Sementara itu, ROS 1 menggunakan desain single-threaded, yang membatasi kemampuan untuk memproses banyak tugas sekaligus.

Middleware yang Lebih Kuat:
ROS 1 menggunakan sistem pesan khusus (custom messaging system), sedangkan ROS 2 menggunakan Data Distribution Service (DDS), sebuah middleware yang lebih efisien dan terstandarisasi untuk komunikasi antar komponen. DDS memungkinkan ROS 2 mendukung skala besar, komunikasi yang lebih andal, dan manajemen data yang lebih baik dalam aplikasi yang kompleks, seperti kendaraan otonom dan robot industri.

Keamanan:
Salah satu keunggulan signifikan ROS 2 adalah fitur keamanan bawaannya, seperti enkripsi dan autentikasi, yang tidak dimiliki oleh ROS 1. Fitur-fitur ini sangat penting untuk aplikasi komersial dan industri di mana

keamanan data dan integritas sistem adalah prioritas utama. Dalam lingkungan industri yang sensitif, ROS 2 memungkinkan perlindungan data dan komunikasi yang lebih aman.

Dukungan untuk Berbagai Sistem Operasi:
ROS 1 hanya berjalan di Linux, sementara ROS 2 mendukung Linux, Windows, dan macOS. Fleksibilitas ini membuat ROS 2 lebih mudah diintegrasikan ke berbagai platform, memberikan pilihan yang lebih luas bagi pengembang dalam memilih sistem operasi yang sesuai dengan kebutuhan proyek mereka.

Pemeliharaan Jangka Panjang dan Performa yang Stabil:
ROS 2 dirancang untuk mendukung proyek jangka panjang dengan peningkatan performa dan skalabilitas yang lebih baik. Pengembang yang mencari solusi jangka panjang cenderung memilih ROS 2 karena kemampuan skalabilitasnya, dukungan platform yang lebih luas, dan komunitas yang terus berkembang.

c) Mengapa simulasi robotik penting dalam pengembangan robot, dan apa keuntungan menggunakan simulasi sebelum membangun robot fisik?

Jawab:
Simulasi robotik sangat penting dalam pengembangan robot karena memungkinkan pengujian dan peningkatan algoritma robotik sebelum sistem fisik dibangun. Hal ini tidak hanya menghemat waktu dan biaya, tetapi juga memberikan lingkungan yang aman untuk eksperimen. Misalnya, simulasi dapat digunakan untuk menguji algoritma navigasi dan pemetaan robot dalam lingkungan yang kompleks, seperti medan berbatu atau ruang tertutup, tanpa risiko kerusakan pada perangkat keras nyata [3].

Contoh nyata:
Boston Dynamics menggunakan simulasi ekstensif dalam pengembangan robot seperti *Spot* dan *Atlas*. Sebelum menguji prototipe fisik, mereka melakukan simulasi untuk memastikan stabilitas dan keseimbangan robot di berbagai medan. Ini memungkinkan tim untuk mengidentifikasi dan memperbaiki masalah lebih awal, sehingga mengurangi waktu pengembangan di lapangan dan menghindari risiko kerusakan pada robot fisik [4].

Tesla, dalam pengembangan kendaraan otonom, menggunakan simulasi dunia virtual yang sangat realistis untuk melatih dan menguji algoritma self-driving. Dengan menggunakan simulasi ini, Tesla dapat menguji berbagai skenario jalan raya yang berpotensi berbahaya, yang akan sulit atau mahal dilakukan dalam kondisi nyata. Hal ini membantu Tesla menghemat biaya dan waktu dalam mengembangkan teknologi mobil otonom yang lebih aman [5].

d) Apa itu Gazebo, dan bagaimana Gazebo digunakan untuk mensimulasikan lingkungan fisik bagi robot?

Jawab:
Gazebo adalah simulator robot 3D yang terintegrasi dengan ROS. Dengan Gazebo, pengguna dapat membuat lingkungan simulasi yang realistis dan memvisualisasikan perilaku robot dalam lingkungan tersebut. Gazebo menyediakan berbagai sensor, aktuator, dan objek yang dapat digunakan untuk melatih dan menguji robot dalam berbagai skenario [1].

Keunggulan Pembelajaran dengan ROS dan Gazebo:

a. **Pembelajaran Interaktif:** Pengguna dapat secara interaktif mengembangkan dan menguji program robot menggunakan ROS dan Gazebo. Mereka dapat melihat hasil secara langsung, menganalisis, dan memperbaiki kode program dengan cepat.
b. **Pengenalan Konsep Robotika**: ROS dan Gazebo membantu siswa memahami konsep dasar robotika, seperti pergerakan, sensor, pengolahan data, dan tindakan responsif.
c. **Pengembangan Keterampilan Praktis:** Melalui praktik langsung dengan ROS dan Gazebo, siswa dapat mengembangkan keterampilan pemrograman, pengaturan sensor, kontrol robot, dan integrasi komputer.

langkah-langkah dasar mengintegrasikan ROS dengan Gazebo untuk mengontrol robot dalam simulasi:

1. **Siapkan ROS dan Gazebo:** Instal ROS (ROS 1 atau ROS 2) serta Gazebo di kompute. Pastikan versi ROS dan Gazebo kompatibel, karena beberapa fitur Gazebo mungkin bergantung pada versi ROS yang digunakan.

2. **Buat Paket ROS:** Buat paket baru di dalam workspace ROS untuk menampung deskripsi robot dan file kontrol. Pada ROS 1.

3. **Definisikan Model Robot:** Gunakan URDF (Unified Robot Description Format) atau XACRO (XML Macros) untuk membuat deskripsi rinci tentang robot yang dibuat, termasuk properti fisik seperti *joints*, *link*s, dan *sensors*. Model ini akan digunakan untuk mensimulasikan gerakan dan interaksi robot dengan lingkungan di Gazebo.

4. **Integrasikan Plugin Gazebo:** Di dalam file URDF atau XACRO, tambahkan tag dan plugin khusus untuk Gazebo. Plugin ini memungkinkan simulasi sensor, fisika, dan kontrol dalam lingkungan Gazebo:

5. **Jalankan Simulasi:** Buat file launch yang menjalankan Gazebo dan ROS secara bersamaan, yang akan menginisiasi lingkungan simulasi dan model robot. File ini biasanya berupa *".launch"* di ROS 1 atau *".py"* di ROS 2, yang berfungsi untuk memanggil robot dalam Gazebo

6. **Kontrol Robot:** Gunakan controller ROS seperti *joint trajectory controllers* atau *differential drive controllers* untuk mengontrol pergerakan robot. Kita dapat menerbitkan perintah ke topik yang relevan dari node ROS, misalnya menggunakan "*position_controllers/JointTrajectoryController"* untuk menggerakkan sendi robot. Contohnya, di ROS 2 kita dapat menggunakan *"ros2_control"* untuk menghubungkan keadaan robot dengan *"physics engine"* di Gazebo [6, 7].

e) Bagaimana cara kerja navigasi robot di dunia simulasi?

Jawab:
Navigasi robot adalah proses di mana robot menentukan posisinya dan arah pergerakannya dalam suatu lingkungan sambil merencanakan dan melaksanakan pergerakan menuju tujuan tertentu. Ini melibatkan pemahaman terhadap lingkungan sekitar, penghindaran rintangan, serta penyesuaian jalur secara real-time. Dalam navigasi robot, teknik-teknik seperti visual odometry dan Simultaneous Localization and Mapping

(SLAM) sangat penting untuk meningkatkan akurasi dan efisiensi dalam pengoperasian robot di lingkungan yang dinamis. Dengan menggunakan sensor seperti LiDAR, sensor ultrasonik, dan kamera, robot dapat mengumpulkan data spasial yang diperlukan untuk menavigasi dengan baik

Navigasi robot dalam dunia simulasi merupakan proses yang melibatkan beberapa konsep dasar seperti **pemetaan** (mapping) dan **lokalisasi** (localization). Berikut penjelasan mengenai bagaimana kedua konsep tersebut berfungsi dan dapat diimplementasikan pada robot Anda:

Pemetaan (Mapping): Pemetaan adalah proses menciptakan representasi visual dari lingkungan yang tidak dikenal. Robot menggunakan sensor seperti LiDAR atau kamera untuk mengumpulkan data tentang sekitarnya. Data ini digunakan untuk membuat peta yang akurat dari area yang sedang dijelajahi. Pemetaan sering dilakukan bersamaan dengan proses lokalisasi, sehingga robot dapat mengidentifikasi tempat-tempat tertentu dalam peta yang telah dibuat

Lokalisasi (Localization): Lokalisasi adalah proses di mana robot menentukan posisinya sendiri dalam peta yang telah dibuat. Robot mengidentifikasi landmark atau ciri-ciri khas di lingkungan dan menghitung posisinya relatif terhadap ciri tersebut. Dengan cara ini, robot dapat "mengetahui" di mana ia berada dalam ruang dan bagaimana melanjutkan ke tujuan yang diinginkan

Simultaneous Localization and Mapping (SLAM): Ini adalah teknik yang menggabungkan pemetaan dan lokalisasi dalam satu proses. Robot yang dilengkapi dengan teknologi SLAM dapat membuat peta sekaligus mengetahui posisi dirinya dalam peta tersebut secara bersamaan. Misalnya, saat robot bergerak di dalam ruangan yang baru, ia akan mencatat landmark yang ditemui, menghitung jaraknya dari landmark tersebut, dan memperbarui peta sesuai dengan informasi baru yang didapat

Implementasi pada Robot: Untuk mengintegrasikan fitur navigasi pada robot, langkah pertama yang harus dilakukan adalah mengumpulkan data dan memilih sensor yang sesuai, seperti LiDAR atau kamera stereo. Selanjutnya, kita dapat menggunakan algoritma SLAM yang sudah ada atau mengembangkan algoritma pemetaan dan lokalisasi sendiri untuk memfasilitasi proses tersebut. Terakhir, penting untuk mengintegrasikan perangkat lunak pemrosesan data dengan robot agar dapat memetakan lingkungan dengan akurat dan mengetahui posisinya di dalam peta yang dibuat. Dengan pendekatan ini, robot dapat beroperasi dengan efisien dalam navigasi baik di dunia simulasi maupun nyata [8, 9, 10]

f) Apa itu TF (Transform) dalam konteks ROS, dan bagaimana TF membantu robot memahami posisi dan orientasinya dalam ruang tiga dimensi?

Jawab:
Dalam konteks ROS (Robot Operating System), TF (Transform) adalah sistem yang memungkinkan robot untuk melacak dan memahami berbagai kerangka acuan (coordinate frames) dalam ruang tiga dimensi. TF menyimpan hubungan antara berbagai kerangka acuan dalam struktur pohon, yang memungkinkan pengguna untuk mentransformasi titik, vektor, dan objek lain antara dua kerangka acuan pada waktu tertentu. Setiap kerangka acuan dapat mengalami perubahan posisi dan orientasi seiring berjalannya waktu, dan TF memudahkan pengelolaan data ini sehingga robot dapat menentukan posisinya dan berinteraksi dengan objek di sekitarnya dengan tepat [11].

Dalam robotika, *frame* mengacu pada sistem koordinat yang menggambarkan posisi dan orientasi objek, biasanya di sepanjang sumbu x, y, dan z. ROS mengharuskan setiap frame memiliki "*frame_id"* yang unik, dan sebuah adegan terdiri dari beberapa frame untuk setiap komponen robot (misalnya *body* dan sensor), objek yang terdeteksi, dan pemain dalam dunia robot.

Contoh Kasus: Robot Lengan dengan Sensor
Bayangkan sebuah robot lengan yang dilengkapi dengan kamera dan menavigasi melalui labirin. Dalam kasus ini, setiap komponen robot dan lingkungannya direpresentasikan dalam sistem koordinat tertentu, sehingga memungkinkan robot memahami posisi dan orientasinya relatif terhadap objek lain.

Menetapkan Frame: Basis robot memiliki frame yang disebut *base_link*, yang berfungsi sebagai frame induk. Lengan robot mungkin memiliki frame anak yang disebut *arm_base_link*, dan kamera dapat memiliki frame anak lain yang disebut *camera_link*. Hubungan antara frame-frame ini didefinisikan oleh transformasinya:

- **Transformasi Statis**: Misalnya, transformasi dari *base_link* ke *camera_link* dapat merepresentasikan posisi tetap kamera relatif terhadap dasar robot.
- **Transformasi Mobil**: Transformasi dari *arm_base_link ke end_effector* (ujung lengan robot) berubah seiring dengan gerakan lengan.

Menyebarluaskan Transformasi: Dalam lingkungan ROS (Robot Operating System), sebuah node dapat dibuat untuk menyebarkan transformasi ini. Sebagai contoh, dengan menggunakan *TransformBroadcaste*r, node tersebut terus menerus menerbitkan posisi dan orientasi frame lengan dan kamera relatif terhadap frame dasar. Ini memungkinkan komponen lain dalam sistem untuk mengakses informasi posisi terkini, memastikan bahwa mereka bereaksi dengan benar terhadap perubahan di lingkungan

Menggunakan Transformasi: Node kedua, menggunakan *TransformListener*, dapat mengambil titik yang terdeteksi oleh kamera dan mentransformasikannya dari frame kamera ke frame dasar robot. Ini sangat penting agar robot memahami di mana objek tersebut berada dalam konteks sistem navigasinya. Misalnya, jika kamera mendeteksi objek pada koordinat (1.0, 0.2, 0.0) dalam frame lokalnya, listener dapat mengonversi koordinat tersebut ke frame global yang didefinisikan oleh *base_link*, memungkinkan robot untuk bergerak menuju objek tersebut dengan akurat [12].

Refrensi

[1] *Robot Operating System (ROS) dan Gazebo sebagai Media Pembelajaran Robot Interaktif*. (2021). School of Information Systems. https://sis.binus.ac.id/2023/05/08/robot-operating-system-ros-dan-gazebo-sebagai-media-pembelajaran-robot-interaktif/

[2] *What Is Robot Operating System (ROS)? | Built In*. (2024). Built In. https://builtin.com/articles/what-is-ros

[3] *What Is Robot Simulation?* (2024). Mathworks.com. https://www.mathworks.com/discovery/robot-simulation.html

[4] *Taking Atlas from Sim to Scaffold*. (n.d.). Boston Dynamics. https://bostondynamics.com/video/inside-the-lab-taking-atlas-from-sim-to-scaffold/

[5] Admin Here. (2024, August 31). *Atlas vs Optimus: Boston Dynamics & Tesla's Humanoid Robots*. -. Supply Chain Today - Homepage. https://www.supplychaintoday.com/atlas-vs-optimus-boston-dynamics-teslas-humanoid-robots/

[6] Mazzari, V. (2015, February 26). *Robotic simulation scenarios with Gazebo and ROS*. Génération Robots - Blog. https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/

[7] *How to Control a Robotic Arm Using ROS 2 Control and Gazebo – Automatic Addison*. (2024, April 30). Automaticaddison.com. https://automaticaddison.com/how-to-control-a-robotic-arm-using-ros-2-control-and-gazebo/

[8] *Localization and Mapping in Robotics*. (2024, September 10). Boston Engineering. https://www.boston-engineering.com/solutions/technical-innovation/robotics/robotics-capabilities/localization-and-mapping-in-robotics/

[9] *Understanding SLAM in Robotics and Autonomous Vehicles*. (n.d.). Www.flyability.com. https://www.flyability.com/blog/simultaneous-localization-and-mapping

[10] *Robot navigation - Vocab, Definition, and Must Know Facts | Fiveable*. (2024). Fiveable.me. https://library.fiveable.me/key-terms/autonomous-vehicle-systems/robot-navigation

[11] *tf2 - ROS Wiki*. (2014). Ros.org. https://wiki.ros.org/tf2

[12] *Understanding ROS Transforms*. (2024). Foxglove.dev. https://foxglove.dev/blog/understanding-ros-transforms