Nama : R. Pandu Davabimo Witama
NIM : 13223033

SOAL 1 URO TEST PROGRAMMING

a. Robot Operating System (ROS) adalah satu set *software framework* yang dirancang untuk memfasilitasi pengembangan software robotika. Meskipun namanya mengandung kata "*operating system*", ROS sebenarnya adalah *middleware* yang menyediakan tools, pustaka, dan konvensi untuk memungkinkan komponen-komponen robot, seperti sensor, aktuator, dan algoritma kontrol, berkomunikasi dan bekerja secara harmonis. ROS mendukung pemrosesan terdistribusi melalui konsep *nodes*, di mana setiap fungsi robot berjalan dalam proses yang terpisah namun tetap terhubung. Selain itu, ROS memiliki ekosistem *libraries* yang luas, komunitas yang besar, dan dukungan untuk simulasi, yang mempercepat pengembangan dan pengujian robot ROS.

   ROS penting dalam integrasi komponen robotik karena menyederhanakan komunikasi antar perangkat keras yang beragam seperti sensor, aktuator, dan kamera. Dengan menyediakan antarmuka standar dan driver yang kompatibel untuk berbagai perangkat, ROS memungkinkan setiap komponen robot untuk berinteraksi tanpa harus memperhatikan detail teknis perangkat keras. Hal ini sangat memudahkan pengembang dalam membangun robot yang kompleks, di mana berbagai perangkat keras dan perangkat lunak dapat bekerja secara sinkron untuk menjalankan tugas robotik, seperti navigasi, pengolahan citra, dan kontrol gerakan.

b. Perbedaan utama antara ROS dan ROS2 terletak pada peningkatan performa, arsitektur, dan fitur-fitur baru yang ditujukan untuk mengatasi keterbatasan ROS pada versi pertama. ROS2 dikembangkan untuk mendukung kebutuhan robotika modern, yang mencakup distribusi yang lebih fleksibel, komunikasi yang lebih aman, dan *scalability* yang lebih baik. ROS2 didasarkan pada *DDS (Data Distribution Service)*, standar industri *middleware* yang memungkinkan komunikasi antar proses lebih efisien dan mendukung arsitektur yang terdistribusi dengan lebih baik. Selain itu, ROS2 mendukung sistem operasi *real-time* dan *multithreading*, yang membuatnya lebih *capable* dalam aplikasi yang memerlukan ketepatan waktu yang akurat.

Pengembang cenderung memilih ROS2 untuk proyek baru karena memiliki keunggulan dalam performa, keamanan, dan pemeliharaan jangka panjang. ROS2 lebih efisien dalam menangani komunikasi antar *nodes*, terutama pada jaringan yang terdistribusi atau pada robot multi-komponen. ROS2 juga menawarkan keamanan yang lebih baik, dengan dukungan bawaan untuk autentikasi, enkripsi, dan kontrol akses, yang tidak ada di ROS1. Selain itu, ROS2 didesain untuk pemeliharaan jangka panjang, dengan arsitektur modular yang lebih siap untuk pengembangan lebih lanjut di masa depan, mendukung lebih banyak platform, termasuk sistem *embedded* dan perangkat dengan sumber daya terbatas. Karena ROS1 tidak lagi dikembangkan aktif untuk fitur-fitur baru, ROS2 menjadi pilihan yang lebih menarik bagi proyek robotik modern yang membutuhkan *capability* dan *scalability*.

c. Simulasi robotik sangat penting dalam pengembangan robot karena memungkinkan pengujian dan evaluasi desain serta fungsionalitas robot dalam lingkungan virtual sebelum membangun robot sebenarnya. Dengan simulasi, pengembang dapat mengevaluasi bagaimana robot akan berinteraksi dengan lingkungan, memvalidasi algoritma kontrol, navigasi, atau pengolahan sensor, dan mengidentifikasi masalah potensial tanpa risiko kerusakan pada *hardware* sebenarnya. Simulasi juga memungkinkan eksperimen pada berbagai skenario tanpa batasan di dunia nyata, seperti keterbatasan waktu ataupun ketersediaan *hardware*.

   Keuntungan utama dari simulasi adalah penghematan waktu dan biaya. Misalkan kita sedang mengembangkan robot pembersih lantai otonom. Sebelum membangun versi fisiknya, kita bisa menggunakan simulasi untuk menguji bagaimana robot bergerak di berbagai jenis ruangan, menghindari rintangan seperti meja atau kursi, dan memastikan bahwa robot mampu membersihkan seluruh area tanpa terjebak. Dengan simulasi, kita bisa mencoba berbagai algoritma navigasi tanpa perlu mengatur ulang furnitur atau bahkan memiliki perangkat keras robot yang sesungguhnya. Dengan cara ini, simulasi dapat menghemat waktu karena kita tidak perlu melakukan pengujian fisik berulang kali, dan menghemat biaya karena kita tidak perlu memperbaiki robot jika terjadi kesalahan, seperti robot menabrak dinding atau merusak komponennya. Setelah algoritma bekerja dengan baik di simulasi, baru kita membangunnya secara fisik, dengan keyakinan bahwa desain tersebut sudah matang dan siap diuji di dunia nyata.

d. Gazebo adalah *software* simulasi yang dirancang untuk menciptakan lingkungan fisik realistis bagi robot, memungkinkan pengembang untuk menguji dan mengevaluasi algoritma robotik dalam ruang tiga dimensi. Gazebo mendukung simulasi berbagai efek fisika, seperti gravitasi dan gesekan, serta integrasi dengan sensor dan aktuator. Dengan menggunakan Gazebo, pengembang dapat merancang model robot dan mensimulasikannya ke dalam dunia fisik virtual, sehingga mereka dapat melihat bagaimana robot berinteraksi dengan lingkungan dan mengidentifikasi masalah potensial sebelum membangun prototipe yang sebenarnya.

   Untuk mengintegrasikan ROS dengan Gazebo, langkah pertama adalah memastikan bahwa kedua *software* ini telah ter-*install*. Selanjutnya, buat workspace ROS dan model robot yang mendeskripsikan geometri dan fungsi robot. Kemudian, buat paket ROS yang berisi file konfigurasi dan file launch untuk menjalankan Gazebo dengan model robot yang ingin disimulasikan. Setelah itu, jalankan simulasi dengan perintah ROS di terminal, dan gunakan skrip untuk mengendalikan robot dengan mengirimkan perintah melalui topik ROS. Dengan proses ini, pengembang dapat dengan mudah menguji dan mengoptimalkan algoritma robotik sebelum implementasi di dunia nyata.

e. Navigasi robot di dunia simulasi melibatkan beberapa konsep dasar yang penting, seperti mapping, lokalisasi, dan perencanaan jalur. Mapping adalah proses di mana robot membuat peta dari lingkungan sekitarnya. Dengan menggunakan sensor seperti LiDAR atau kamera, robot mengumpulkan data untuk membangun representasi visual dari area yang akan dilalui. Setelah peta terbentuk, robot perlu mengetahui posisinya dalam peta tersebut, yang disebut lokalisasi. Lokalisasi membantu robot mengetahui di mana ia berada dengan menggunakan algoritma seperti *Monte Carlo Localization*. Ini penting agar robot bisa bergerak dengan akurat di dalam lingkungan yang telah dipetakan.

   Untuk menerapkan fitur-fitur ini dalam simulasi, kita bisa menggunakan paket ROS seperti gmapping untuk mapping dan amcl (Adaptive Monte Carlo Localization) untuk lokalisasi. Jadi, saat robot bergerak di simulasi, ia akan mengumpulkan data dari sensornya dan memperbarui peta secara real-time. Setelah peta jadi, kita bisa menggunakan algoritma perencanaan jalur seperti A* atau Djikstra *algorithm* untuk merencanakan rute ke tujuan tertentu, sambil memastikan robot menghindari rintangan yang terdeteksi. Dengan cara ini,

kita bisa menguji berbagai strategi navigasi di lingkungan simulasi sebelum menerapkannya pada robot fisik.

f. TF (Transform) dalam konteks ROS adalah sistem yang digunakan untuk mengelola dan mentransformasikan koordinat antara berbagai *frame of reference* atau kerangka acuan di dalam ruang tiga dimensi. Dalam robotika, penting untuk memahami posisi dan orientasi objek (seperti robot itu sendiri, sensor, dan objek lain di sekitarnya) relatif satu sama lain. TF memungkinkan robot untuk melacak hubungan ini secara *real-time*, sehingga pengembang bisa mendapatkan informasi yang akurat tentang posisi dan orientasi berbagai komponen robot dalam koordinat yang berbeda.

   Misalnya, saat mengembangkan robot pembersih lantai, kita mungkin memiliki frame untuk robot itu sendiri, satu untuk kamera yang dipasang di robot, dan satu lagi untuk sensor jarak. TF digunakan untuk memastikan bahwa data dari sensor jarak bisa diubah ke dalam frame robot sehingga dapat digunakan untuk navigasi dan penghindaran rintangan. Saat robot bergerak, TF akan terus memperbarui transformasi antara frame-frame ini, sehingga algoritma navigasi dapat dengan tepat menghitung posisi dan orientasi robot dalam ruang tiga dimensi. Dengan menggunakan TF, kita dapat memastikan robot bergerak dengan benar dalam simulasi dan berinteraksi dengan objek lain dengan akurat, sehingga mengurangi kemungkinan terjadinya kesalahan saat robot beroperasi di dunia nyata.