a.)

ROS (Robot Operating System) adalah framework (struktur yang dapat digunakan berulang kali untuk membangun sesuatu) yang dapat digunakan untuk membantu pengembangan robot. ROS berisi driver hardware, algoritma robotik, alat-alat GUI dan CLI, yang akan memudahkan pengembangan dan simulasi robotik. ROS berada di “atas” operating system (Linux, Windows, dll) sehingga dapat berinteraksi langsung dengan hardware.

ROS sangat berpengaruh dalam robotika modern karena membuat proses pengembangan robotik menjadi lebih fleksibel dan adaptif (proyek A dapat diintegrasikan ke proyek B dengan mudah). Selain itu ROS juga open source dan memiliki komunitas besar serta dokumentasi lengkap sehingga banyak digunakan dalam pembelajaran robotika, riset, industri, dan bisnis.

ROS sangat penting untuk integrasi komponen hardware (sensor, aktuator, kamera, dan lain-lain) dengan software (kode dan simulasi) karena ROS dapat mengatur hubungan keduanya dalam satu-kesatuan yang bekerja secara teratur dengan menggunakan nodes, topics, services, parameters, dan actions. ROS juga meningkatkan adaptabilitas dan kompatibilitas, tanpa ROS sebuah kode dari proyek A mungkin tidak akan kompatibel dengan proyek B.

b.)

Perbedaan utama antara keduanya yaitu, ROS diciptakan untuk riset dan kegiatan akademika tanpa memikirkan aspek industri dan komersialisasi, sedangkan ROS2 diciptakan dengan memikirkan berbagai aspek industri seperti keamanan dan kesiapan robot sebagai produk komersial.

Oleh karena itu, pengembang robotika modern lebih memilih ROS2 karena aspek keamanan, performa, pemeliharaan, kompatibilitas, dan lain-lain secara keseluruhan lebih unggul dibanding ROS. Selain itu ROS2 juga *backward compatible*, yang artinya proyek yang kompatibel dengan ROS pasti kompatibel dan dapat diimplementasikan pada ROS2. Berikut keunggulan ROS2:

- **Keamanan:** ROS menggunakan protokol komunikasi TCP dan UDP, protokol ini kurang menjamin keamanan. Sedangkan ROS2 menggunakan protokol DDS yang lebih menjamin keamanan melalui autentikasi dan kriptografi. Keamanan ini dibutuhkan karena robot mungkin perlu berinteraksi melalui internet.

- **Performa:** Alasan lain digunakannya protokol DDS yaitu DDS tidak melakukan re-transmit data sehingga memiliki performa yang lebih baik pada jaringan (koneksi *Wi-fi* atau satelit) yang jelek, berbeda dengan ROS1. Selain itu, ROS2 juga memiliki performa lebih baik

dalam *real-time computing* dibanding dengan ROS1, hal ini tentunya sangat penting dalam robotika yang membutuhkan latensi dan *delay* yang rendah.

- **Pemeliharaan:** ROS1 menggunakan sistem ROS master dimana sebuah node akan tersentralisasi kepada sebuah “master” agar bisa terhubung dengan node lainnya jika node “master” tidak dapat dijangkau maka seluruh nodes akan tidak terhubung, sedangkan ROS2 tidak menggunakannya dan setiap node cenderung *peer-to-peer* tanpa ada sentralisasi. Dalam hal ini ROS2 lebih unggul dalam pemeliharaan karena lebih terjaga dalam jangka panjang. Selain itu, ROS2 lebih *multi-platform* (Linux, Windows, MacOs) dibanding dengan ROS (Linux saja). ROS2 juga mendukung versi bahasa pemrograman yang lebih baru (misal, c++14 dan python 3.5), selain itu ROS juga akan mencapai *end-of-life* pada 2025 sehingga sangat logis untuk memilih ROS2.

c.)

Dalam proses pengembangan robot tentunya diperlukan banyak ide dan gagasan untuk setiap detail sekecil mungkin yang ada, mulai dari aspek elektronika, mekanika dan pemrograman. Kesalahan kecil dalam pengembangan mungkin akan berakibat fatal dan menelan waktu serta biaya. Oleh karena itu, simulasi robotik sangat penting untuk memvalidasi segala ide dan melakukan verifikasi terkait segala detail implementasi.

Keuntungan yang bisa didapat dari melakukan verifikasi terlebih dahulu sebelum membangun robot fisik yaitu akan meminimalisir kesalahan yang terjadi pada proses pembangunan, sehingga mulai dari proses pengumpulan komponen, membuat prototype, hingga hasil akhir akan sangat kecil kemungkinan terjadi kesalahan. Hal ini tentunya akan menghemat waktu dan biaya.

Contoh kasus yang bisa kita lihat yaitu, misal kita akan membuat sebuah *line follower robot*. Dalam hal ini kita bingung untuk menggunakan sensor apa dalam deteksi garis hitam. Kita dapat memilih antara *infrared sensor*, *color sensor,* dan *camera based sensor*. Dapat kita simulasikan penggunaan ketiga sensor dan dilihat mana hasil yang lebih cocok. Tanpa simulasi mungkin kita harus mencoba ketiga jenis sensor secara fisik dengan membangun tiga prototype sehingga akan memakan waktu, tenaga, dan biaya.

d.)

Gazebo adalah software open source simulasi robotik 3D dan 2D yang dapat diintegrasikan dengan ROS2. Gazebo dapat mensimulasikan lingkungan fisik dengan cara membuat *virtual environment* yang dapat berisi objek dan dapat diisi dengan berbagai robot hasil pengembangan. Sensor pada simulasi bekerja seperti sensor di dunia nyata, sensor dapat menerima data dan mengirimnya pada *topic* dan *service* pada ROS2. Aktuator simulasi juga dapat di tes dengan properti fisika seperti dunia nyata (misal gesekan, gravitasi, dan lain-lain).

Langkah-langkah dasar mengintegrasikan ROS2 dengan Gazebo yaitu:

- Menyiapkan OS Ubuntu 24.04 agar dapat dengan mudah menggunakan ROS2
- Menginstall ROS2 dengan distro Jazzy (dipilih karena merupakan versi stabil terbaru)
- Menginstall Gazebo Harmonic 8.6.0 (paling kompatibel dengan ROS2 Jazzy)
- Menginstall package ROS2 untuk Gazebo yang bernama “ros-jazzy-ros-gz”
- Membuka Gazebo dengan plugin/script dari package yang telah diinstall (misal, “ros2 launch ros_gz_sim gz_sim.launch.py”)
- Gazebo sudah terintegrasi dengan ROS2, untuk langkah-langkah pengembangan selanjutnya dapat dilakukan dengan melihat dokumentasi.

e.)

Untuk mengetahui cara kerja navigasi robot di dunia simulasi, terdapat beberapa konsep dasar yang perlu dipahami. pertama, simulasi dilakukan pada *virtual environment* yang merupakan simulasi fisika yang berisi objek dan lingkungan yang berbentuk semirip mungkin dengan lingkungan nyata. Pada lingkungan virtual ini, terdapat *global frame* yang merupakan titik posisi yang dipilih sehingga semua pengukuran dilakukan relatif terhadap titik ini.

Kemudian *mapping* adalah kegiatan melakukan pembuatan *virtual environment* yang berisi layout spasial, objek, dan properti fisika (misal, gravitasi). Sedangkan lokalisasi adalah pengambilan posisi (menggunakan aksis x, y, z) dan rotasi dari robot di simulasi secara kontinu berdasarkan *actions* yang dilakukan.

Navigasi robot di dunia simulasi dapat diimplementasikan menggunakan *gazebo* dan framework ROS2 (khususnya, SLAM). untuk simulasi 2D dapat digunakan package gmapping, hector_slam, cartographer, ohm_tsd_slam. Sedangkan untuk simulasi 3D dapat digunakan package rgbdslam, ccny_rgbd, lsd_slam, rtabmap.

f.)

TF (Transform) mengekspresikan posisi dan orientasi suatu objek relatif terhadap objek lain (biasanya, relatif terhadap *global frame* yang tidak bergerak*)*. Setiap objek pada simulasi akan memiliki TF masing-masing dan akan terdapat hubungan antara beberapa objek dengan objek lainnya (misal, lengan robot akan bergerak relatif terhadap badan robot). TF membantu robot memahami posisi dan orientasi secara akurat karena posisi dan orientasinya dapat digambarkan dan diubah menggunakan operasi matematis, sebagaimana adanya di dunia nyata.

Misalnya dalam pengembangan robot berkaki empat, ketika di simulasi robot mendeteksi objek pada jarak beberapa meter di depan relatif terhadap posisi robot, misal robot harus berbelok ke kanan. Pada saat itu, robot akan mulai mengubah TF yang dimiliki

mulai dari TF badan robot, kemudian TF sendi kaki robot akan berubah relatif terhadap posisi TF badan robot yang sudah berubah, terakhir TF telapak kaki robot akan berubah relatif terhadap posisi TF sendi kaki robot yang sudah berubah. Perubahan semua komponen TF dapat dilakukan dengan matriks transformasi yang berbeda-beda, namun menghasilkan *goals* yang sama (berbelok kanan).