a) Apa itu ROS (Robot Operating System), dan bagaimana peran utamanya dalam pengembangan robotik modern? (Jelaskan mengapa ROS penting untuk integrasi berbagai komponen robot seperti sensor, aktuator, dan kamera dalam satu sistem yang bekerja harmonis.)

ROS (*Robot Operating System*) adalah sekumpulan *framework* atau *software library* yang bersifat *open source*, yang dapat digunakan untuk membantu membuat robot. ROS adalah *middleware* yang berperan penting dalam pengintegrasian komponen, karena dapat memungkinkan komponen-komponen robot berkomunikasi (ter-*link*) satu sama lain. ROS bersifat *user-friendly*, karena kita tidak mesti sepenuhnya tahu cara kerja suatu perangkat keras.

---

b) Apa perbedaan utama antara ROS dan ROS2, dan mengapa pengembang cenderung memilih ROS2 untuk proyek baru? (Jelaskan keunggulan ROS2 dibandingkan ROS dalam hal performa, keamanan, dan pemeliharaan jangka panjang.)

	ROS 1	**ROS 2**
Bahasa Pemrograman	C++03, Python 2+	C++11, C++14, C++17, Python 3.5+
Sistem Operasi	Tidak kompatibel dengan Windows 11	Kompatibel dengan Windows 11
Build System	CMake	CMake, Python *packages*
Real-time	Tidak didukung	Didukung/kompatibel
Nodes per process	Hanya satu	Banyak, *real-time*
Middleware	Memakai format *serialization* dan *transport protocol* yang bersifat *custom*.	Memakai *abstract middleware interface*
Cara *build* di Windows	Hanya bisa di-*build* dari *source*, dan hanya mendukung beberapa ROS *packages*	Juga dapat di-*build* dengan *binary package* Chocolatey.

Banyak *developer* yang lebih memilih ROS2 karena dapat bekerja secara *real-time*, dan juga kompatibel dengan Windows 11, sehingga tidak perlu *dual-boot* (ganti OS) ke Linux atau Windows 10; selain itu, bahasa C++ yang dipakai ROS 1 juga hanya mendukung versi yang sudah tidak di-*support* lagi, yaitu C++03, sedangkan ROS 2 memakai C++14 (beberapa distro memakai C++17) yang belum di-*drop* dan masih dipakai dalam industri.

ROS 2 lebih efisien dan *scalable*, juga memiliki *low-latency*, daripada ROS 1, karena menggunakan DDS (Data Distribution Service) dan dapat bekerja secara *real-time*. DDS memiliki mekanisme keamanan yang baik karena memberi *authentication*, enkripsi, dan kontrol akses. ROS 2 memiliki distro yang bersifat LTS (*Long Term Support*), yaitu selama 5 tahun, sehingga program-program yang dibuat akan dapat di-*maintain* dalam kurung waktu yang lama.

---

c) Mengapa simulasi robotik penting dalam pengembangan robot, dan apa keuntungan menggunakan simulasi sebelum membangun robot fisik? (Berikan contoh kasus di mana simulasi dapat menghemat waktu dan biaya pengembangan robot.)

Simulasi robotik penting untuk mengetahui apakah suatu robot dan komponen-komponennya bekerja dengan baik, sebelum membangun fisik dari robot tersebut. Keuntungan dari simulasi robotik sebelum membangun fisiknya adalah:

- **Hemat biaya**
  Misalnya, terdapat kesalahan pada robot fisik yang mengakibatkan komponen tersebut rusak. Kita harus membeli komponen tersebut lagi, sehingga tidak hemat biaya.
- **Hemat waktu dan lebih efisien**
  Misal, ternyata ada komponen pada robot fisik yang kurang, maka harus menghabiskan waktu membeli komponen tersebut lagi. Atau, jika ada komponen yang berlebih—yang ternyata robot fisik tersebut tidak perlukan—,maka akan ribet jika harus melepasnya lagi. Selain itu, kita juga dapat memvisualisasikan *design* robot tersebut menggunakan *software*, sehingga tidak perlu merakit berulang-ulang secara fisik.

---

d) Apa itu Gazebo, dan bagaimana Gazebo digunakan untuk mensimulasikan lingkungan fisik bagi robot? (Jelaskan langkah-langkah dasar mengintegrasikan ROS dengan Gazebo untuk mengontrol robot dalam simulasi.)

Gazebo adalah program yang digunakan untuk memvisualisasikan atau mensimulasikan robot, baik secara 2 dimensi maupun 3 dimensi, yang bersifat *open-source*. Gazebo dapat membuat *environment*/lingkungan virtual, misalnya, suatu ruang 3 dimensi.

Cara mengintegrasikan ROS dengan Gazbo:

1. *Install* dan jalankan Gazebo
2. Buat *workspace* di ROS
3. Buat *world* baru di Gazebo
4. *Install package* ROS yang dibutuhkan untuk integrasi dengan Gazebo

5. Buat *node* pada ROS agar dapat berinteraksi dengan Gazebo

---

e) Bagaimana cara kerja navigasi robot di dunia simulasi? (Apa saja konsep dasar seperti mapping dan lokalisasi yang perlu dipahami, dan bagaimana fitur ini dapat diimplementasikan pada robot Anda?)

Cara kerja navigasi robot pada simulasi yaitu robot virtual memperhatikan lingkungan di sekitarnya, misalnya ada penghalang (*obstacle*) apa-apa saja. Robot tersebut lalu mencari jalan/arah (*pathfinding*) untuk mencapai suatu titik/tujuan. Agar *pathfinding* ini berhasil, dibutuhkan *mapping* dan lokalisasi.

Mapping adalah proses membuat peta/*environment* virtual. Peta ini dapat berupa 2 dimensi maupun 3 dimensi. Namun, secara dasar/*basic*, biasanya memakai 2 dimensi dengan *grid* agar lebih mudah. Tiap sel/kotak bisa saja kosong (dapat dilewati robot), ada penghalang (tidak dapat dilewati robot), atau tidak diketahui (misalnya ada kabut/asap sehingga sulit dideteksi; bisa dilewati jika tidak ada penghalang). Algoritma ini disebut *Occupancy Grid Mapping*. Pengimplementasian *Mapping* juga dapat menggunakan algoritma SLAM (*Simultaneous Localization and Mapping*)

Lokalisasi adalah proses menentukan lokasi robot. Misalnya posisi, ketinggian (jika 3 dimensi), dan menghadap ke arah mana. Ini dapat diimplementasikan dengan algoritma *Dead Reckoning* atau *Visual Odometry*.

Path Planning adalah proses menentukan jalan/arah yang dapat dilewati oleh robot. Jalan/arah yang akan dilewati adalah yang paling optimal (bukan hanya terpendek, namun aman untuk dilewati). *Path Planning* dapat diimplementasikan menggunakan algoritma Dijkstra ataupun A*.

---

f) Apa itu TF (Transform) dalam konteks ROS, dan bagaimana TF membantu robot memahami posisi dan orientasinya dalam ruang tiga dimensi? (Berikan contoh bagaimana TF digunakan untuk memastikan robot bergerak dengan benar dalam simulasi.)

TF (Transform) adalah *library* pada ROS yang dapat digunakan untuk melacak beberapa *coordinate frame* secara bersamaan. Setiap komponen atau bagian tubuh pada robot memiliki *coordinate frame*-nya tersendiri, TF memungkinkan *frame-frame* tersebut terhubung. Misalkan, saat robot mengambil suatu objek, TF akan men-*transform coordinate frame* dari sensor yang mendeteksi objek ke lengan robot; atau saat robot bergerak ke suatu target/titik, TF dapat menentukan transformasi yang perlu untuk memindahkan *base frame* robot agar sampai ke tujuan.