Robot Software Architecture: Unpacking Mobile Robot Design

Fiona Opiyo | August 31, 2025

Introduction: Crafting Robust Robot Systems

Software architecture defines a system's high-level structure and ensures it meets specific needs. For mobile robotics, this means building in:

1

Real-Time Capabilities

Immediate processing for dynamic environments.

2

Asynchronous Data Processing

Handling concurrent sensor inputs and commands.

3

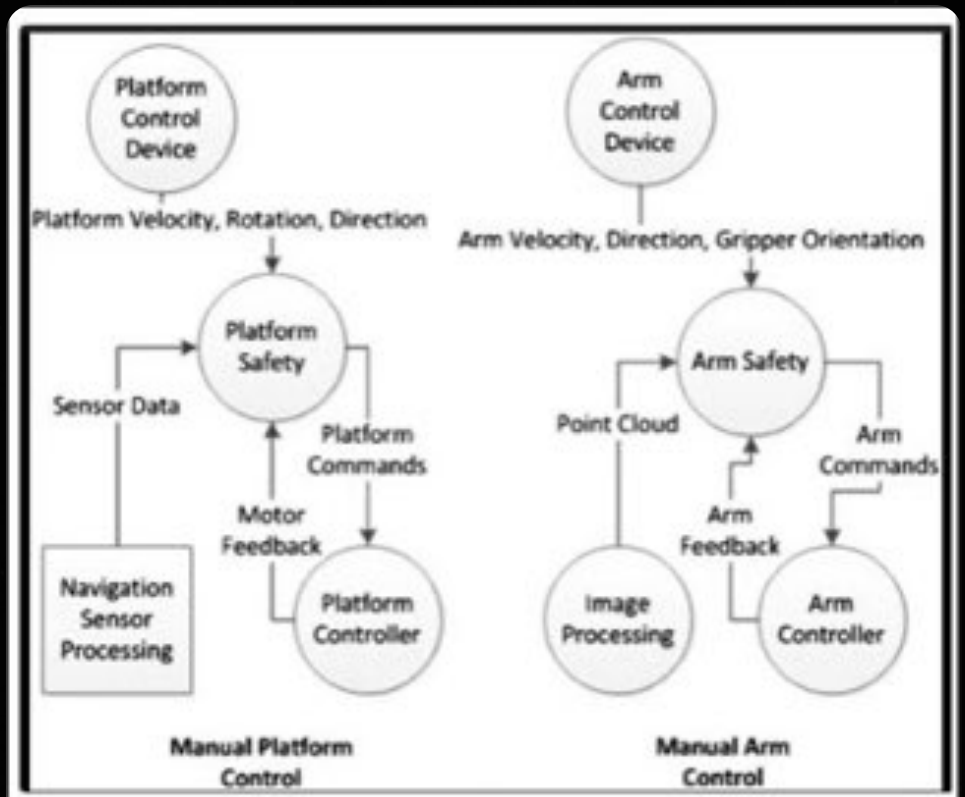
Distributed Functionality

Components operating across different hardware.

Architectural Patterns in Robotics

Component-Based

Platform and arm control devices operate as separate components with their own subsystems while exchanging specific data types

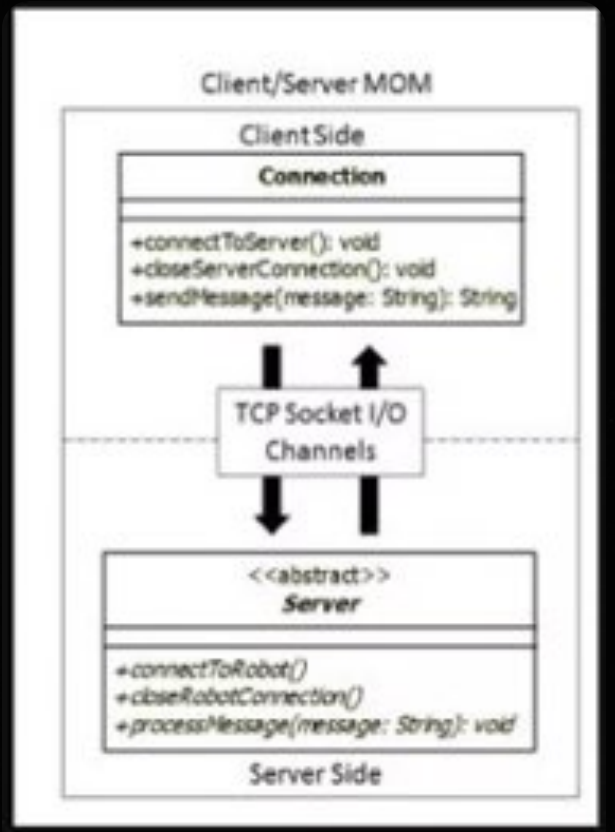


An example of component based architecture

Architectural Patterns in Robotics

Client-Server

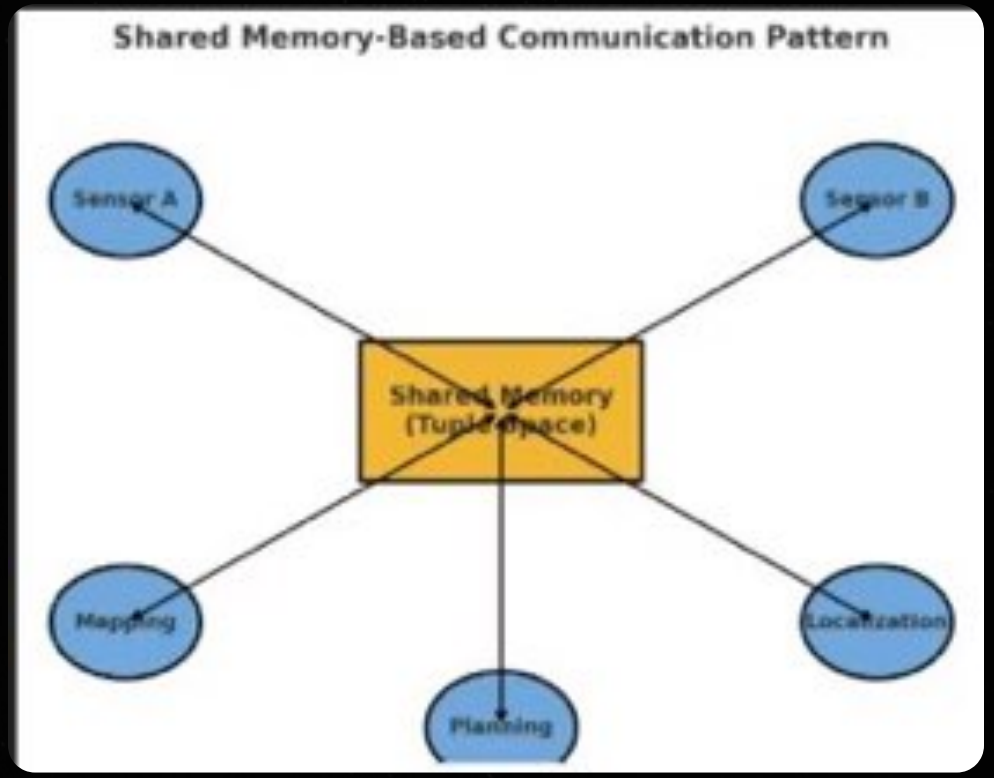
The server defines abstract operations for different robots, while the client establishes the connection and sends high-level commands (e.g., “ROTATE 90”, “MOVE 20”) that control the robot’s actions.



Architectural Patterns in Robotics

Blackboard

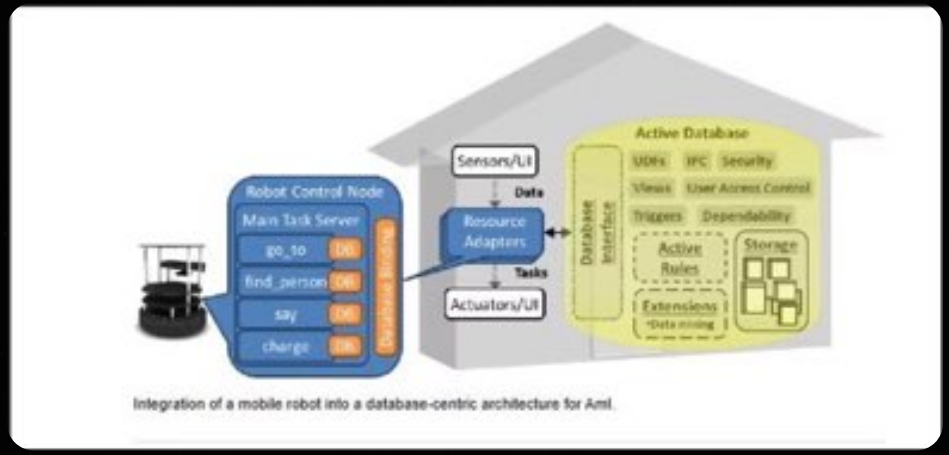
Each component (e.g., sensors, mapping, localization, planning) interacts asynchronously with the central **shared memory (tuple space)**, posting and retrieving data.



Architectural Patterns in Robotics

Database-Centric

A database-centric robotics architecture uses in-database processing and resource adapters to securely handle real-time events and integrate diverse robotic components like sensors, actuators, and interfaces.

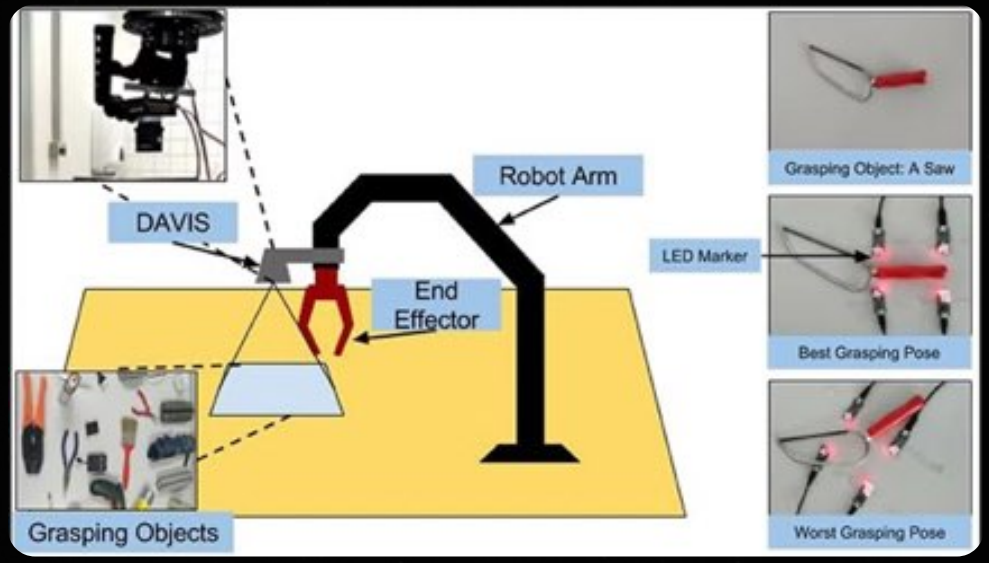


Integration of a mobile robot into a database-centric architecture for AmI.

Architectural Patterns in Robotics

Event-Driven

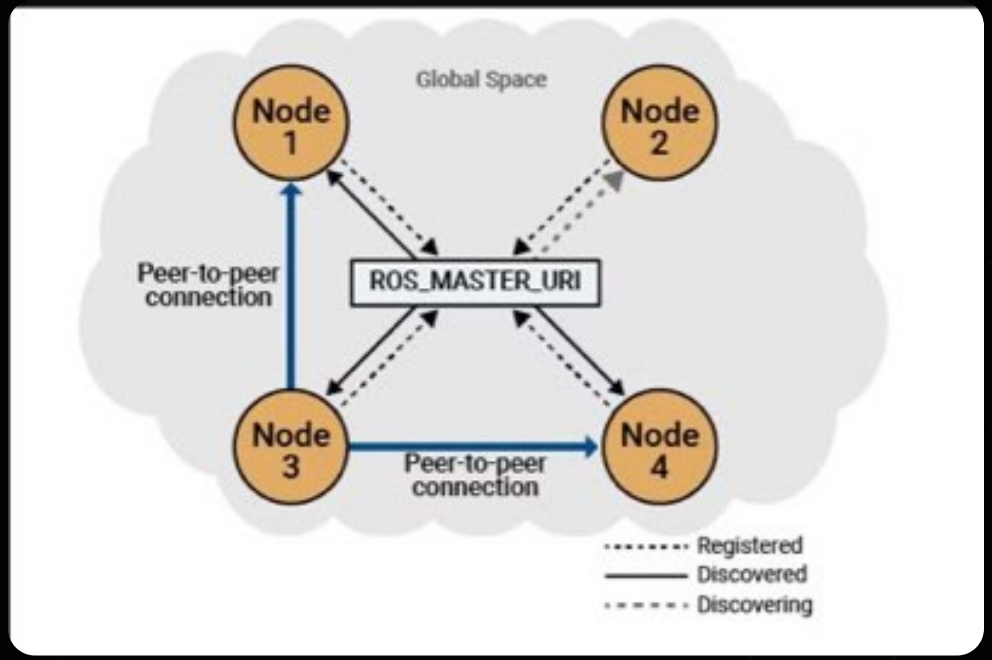
This diagram shows an **event-driven architecture** where a neuromorphic vision sensor (DAVIS) on the robot arm detects asynchronous visual events from objects, triggering real-time grasp detection and immediate adjustment of the arm's end effector for the best grasping pose.



Architectural Patterns in Robotics

Peer-to-Peer

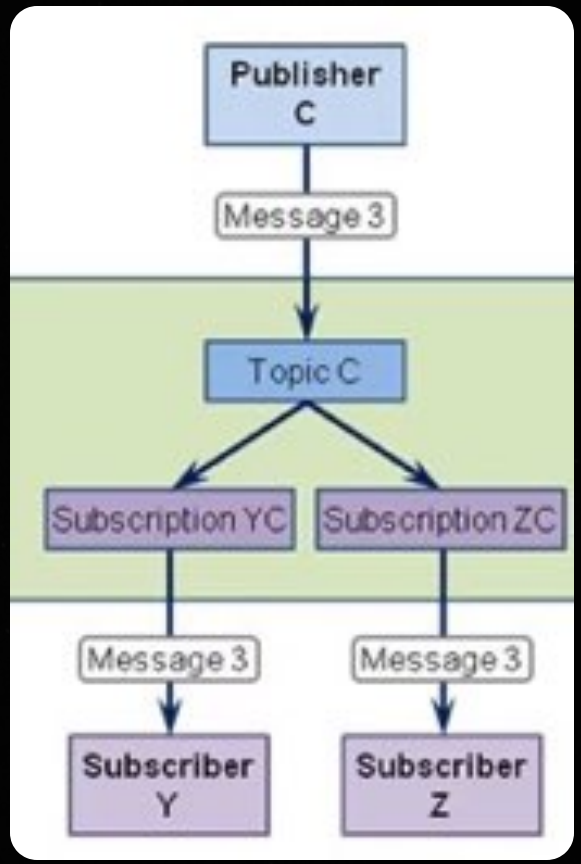
Nodes communicate directly with each other after discovery, enabling distributed coordination without relying solely on a central controller.



Architectural Patterns in Robotics

Publish-Subscribe

A **decentralized, asynchronous** model where publishers emit data on topics and subscribers listen



The Two Pillars of Robust Robot Architecture

Component-Based Architecture

Modularity & Reusability for flexible, interchangeable parts.

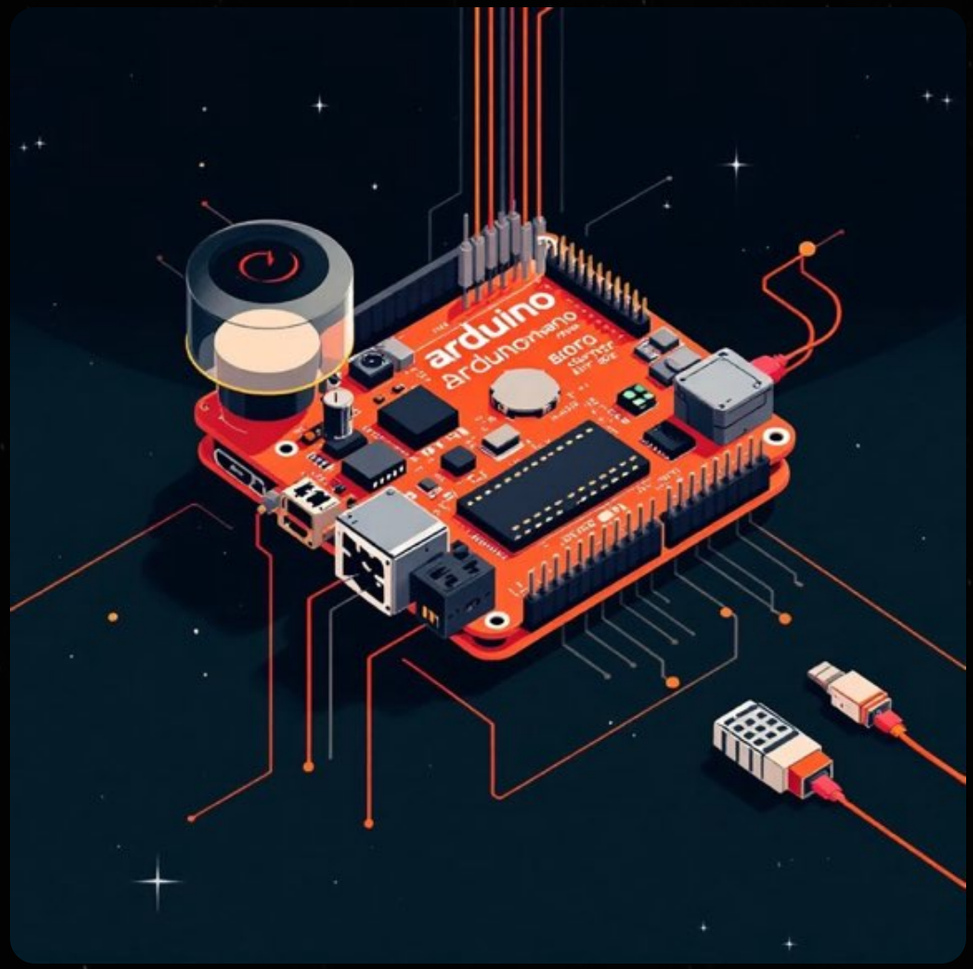
Publish-Subscribe Pattern

A **decentralized, asynchronous** model where publishers emit data on topics and subscribers listen

Detailed Software Architecture for Mobile Robots

Hardware Abstraction Layer (HAL)

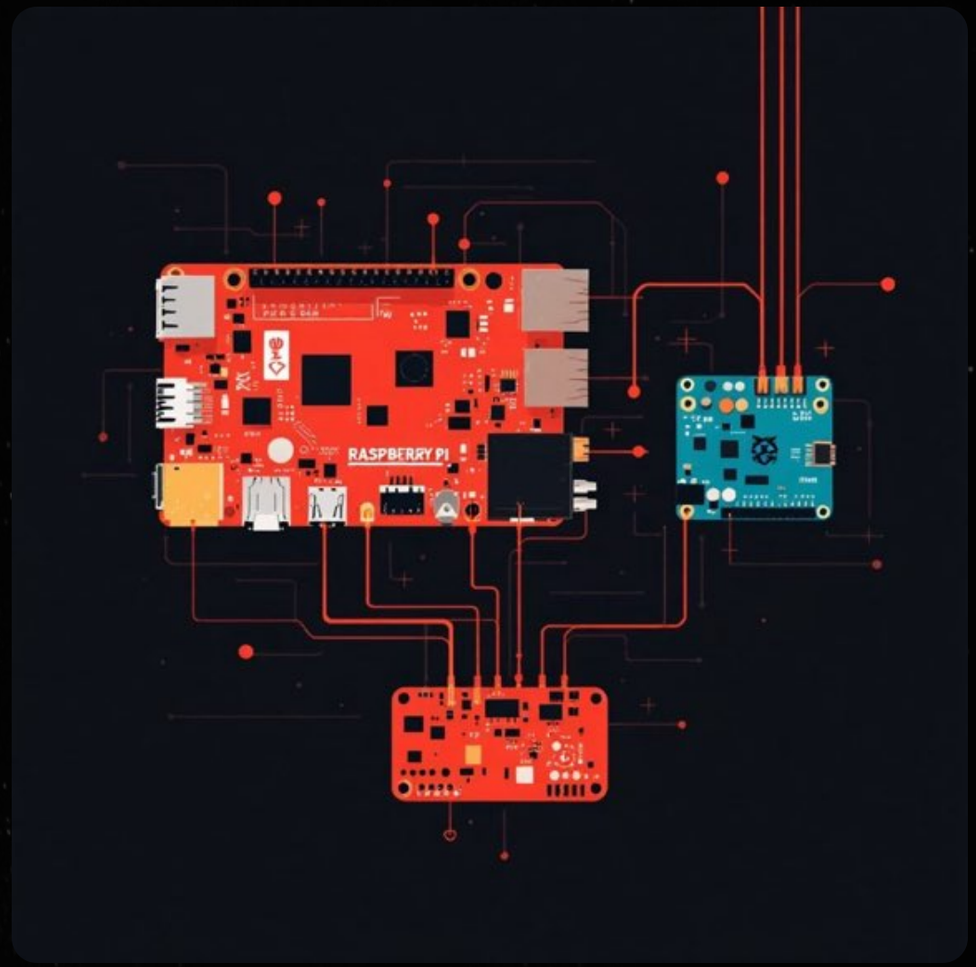
Directly interfaces with hardware (e.g., Arduino Nano for motor control, encoder readings).



Firmware Communication Interface

Bridges high-level control (Raspberry Pi) and microcontroller (Arduino) via serial communication.

ROS 2 Node: `arduino_bridge_node`



1. Firmware Communication Interface (`arduino_bridge_node`)

- **Role**: The bridge between the ROS software world and the physical hardware.
- **Subscribes To**: `/cmd_vel` (velocity commands). It listens for movement instructions from the navigation stack or a teleop node (like a joystick).
- **Publishes**: `/odom` (odometry data). It broadcasts the robot's estimated position based on raw data from the wheel encoders.

2. Robot Localization (`ekf_localization_node`)

- **Role**: The sensor data fuser, creating a single source of truth for the robot's position.
- **Subscribes To**: Multiple sensor topics, such as `/wheel_odom` (from the Arduino bridge) and `/imu` (from an Inertial Measurement Unit).
- **Publishes**: A fused and more accurate `/odom` topic and the `/tf` transform (the robot's position in the odometry frame).

3. Mapping (`slam_toolbox`)

- **Role**: The environment modeler, building a map while tracking the robot's position.
- Package: `slam_toolbox`
- **Subscribes To**: `/scan` topic(from the LiDAR) and `/odom` (from the localization node).
- **Publishes**: The `/map` topic (a 2D grid of the environment) and a `/tf` transform to place the robot correctly on that map.

4. Navigation Stack (Nav2)

- **Role**: The primary decision-maker for autonomous movement.
- Packages: `nav2_bringup`, `nav2_planner`
- **Subscribes To**: A wide array of data: the `/map` topic(to know the world), `/odom` topic (to know its current position), `/scan` topic(to see immediate obstacles), and `/goal_pose` topic(to know its destination).
- **Publishes**: `/cmd_vel` topic (the final velocity commands to drive the robot).

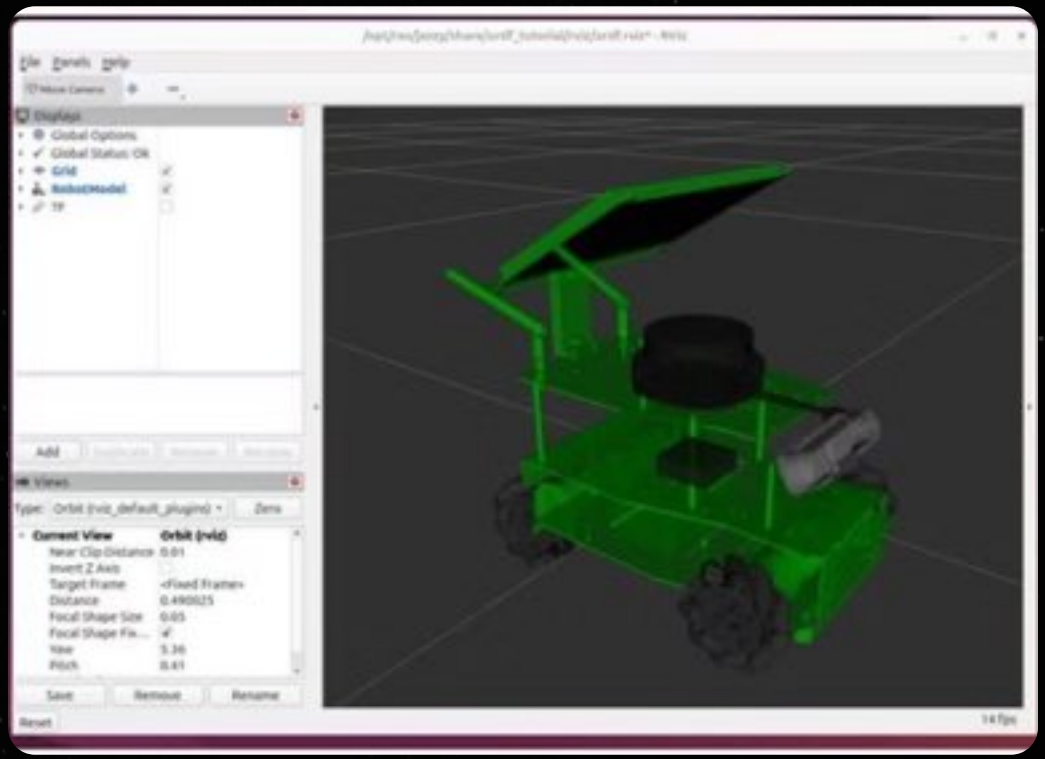
Essential Tools & Data Representation

Perception (LiDAR)

Utilizes 360° 2D LiDAR to provide range measurements for mapping and obstacle detection.

Robot Description (URDF/Xacro)

Provides a complete digital model of the robot, including physical structure and sensors.

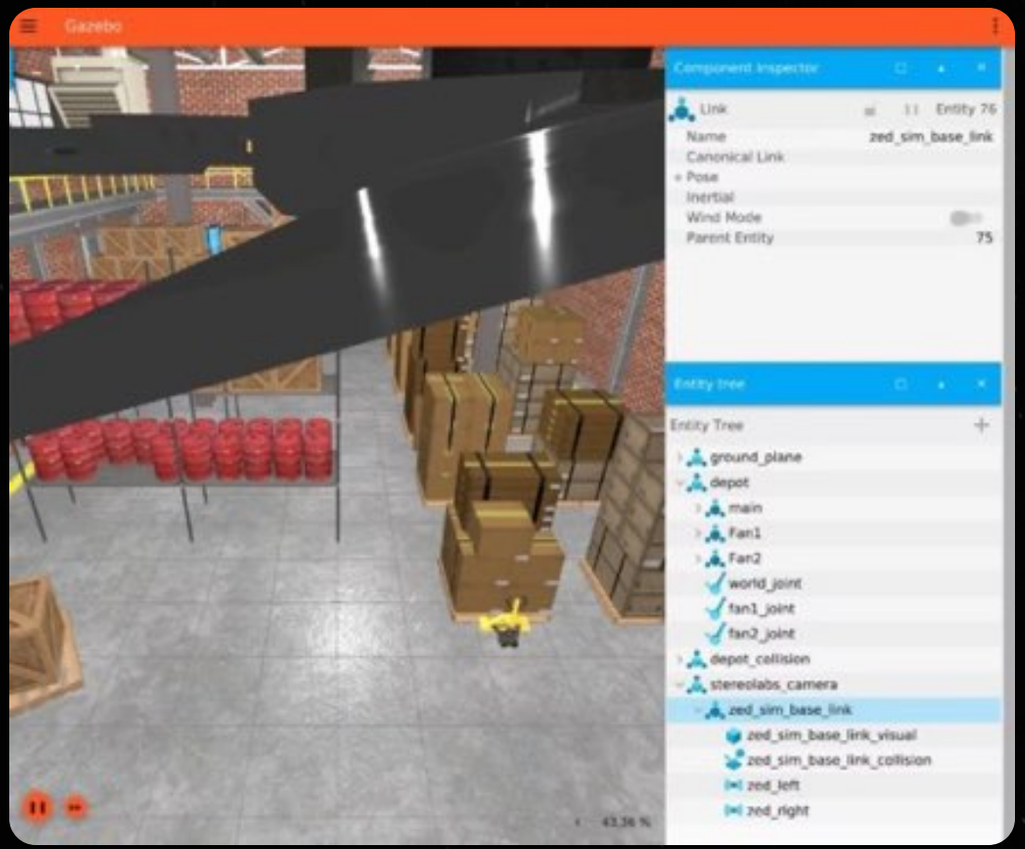


Simulation & Visualization

Simulation (Gazebo)

Physics-based 3D simulation environment for safe testing and debugging.

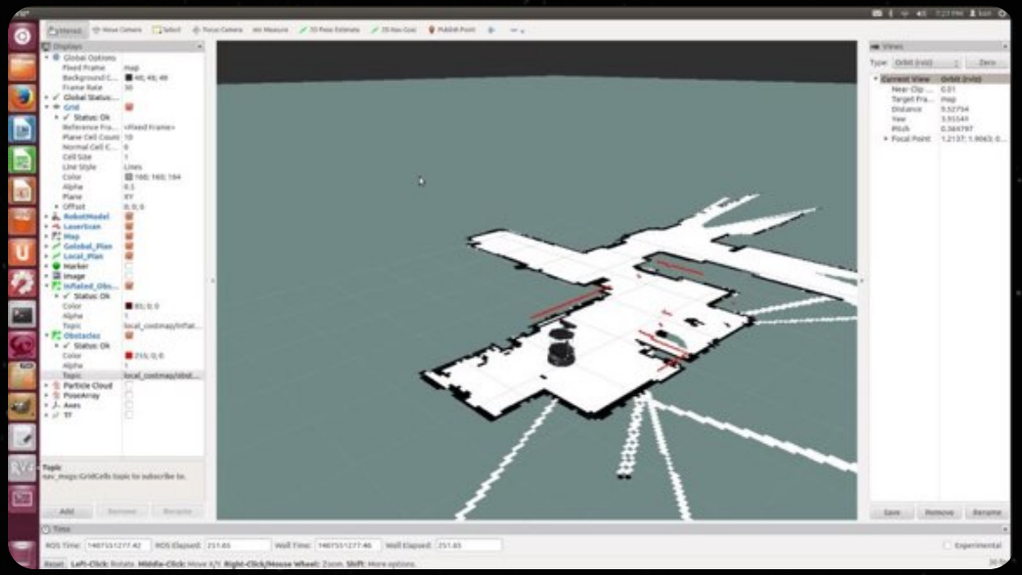
- Features: gravity, collisions, sensor emulation.



Visualization (RViz2)

Main tool for real-time visualization of robot data and interactions.

- Displays: robot model, laser data, maps, TF tree.



Example of how to organize your work

```
ros2_ws/
├── src/
│   ├── robot_description/        # URDF/Xacro
│   ├── arduino_bridge/           # Serial communication
│   ├── navigation_launch/        # Nav2 + slam_toolbox launch
│   ├── slam_toolbox/             # External package or submodule
│   ├── custom_nodes/             # mission control, odometry
│   └── gazebo_sim/               # World, robot model, plugins
├── install/
├── build/
└── launch/
```

Integrated Robot Architecture Layers

01

Physical Hardware Layer

Sensors & Actuators: RPLIDAR, Arduino, DC Motors, Raspberry Pi.

02

Firmware Layer

Arduino Firmware (C++): Motor PID, Serial Protocol, Watchdog Timer.

03

ROS 2 Node Layer

Perception, Localization, Navigation Stacks: `rplidar_ros2_node`, `ekf_filter_node`, `controller_server`, etc.

04

Failure Recovery System

Monitoring (`system_monitor`) & Recovery Behaviors (`recoveries_node`).

05

Debugging Tools

`rqt_graph`, `rosbag2`.

06

Simulation

Gazebo with various plugins.

References

1. https://roboticsdojo.substack.com/p/software-architecture-of-mobile-robot
2. https://drive.google.com/file/d/16vKXEoRQc7crN7w_REoBUZu1l8MaCfrK/view?usp=sharing
3.