# Robot Hardware Architecture

**Objective**: By the end of this session, you will have a functional understanding of a ROS2 robot's physical architecture. You will set up key hardware components, establish communication links between them, and integrate them into a unified ROS2 network. The final deliverable is **a communication diagram of your system.**

## Table of Contents



## 1. Introduction to the Physical Architecture

A robot is more than a set of motors and sensors. It is a layered system composed of compute units, microcontrollers, actuators, communication buses, and safety electronics that work together to **sense**, **process**, and **act** on the environment.
Understanding each block is essential before integrating ROS2 with real hardware.

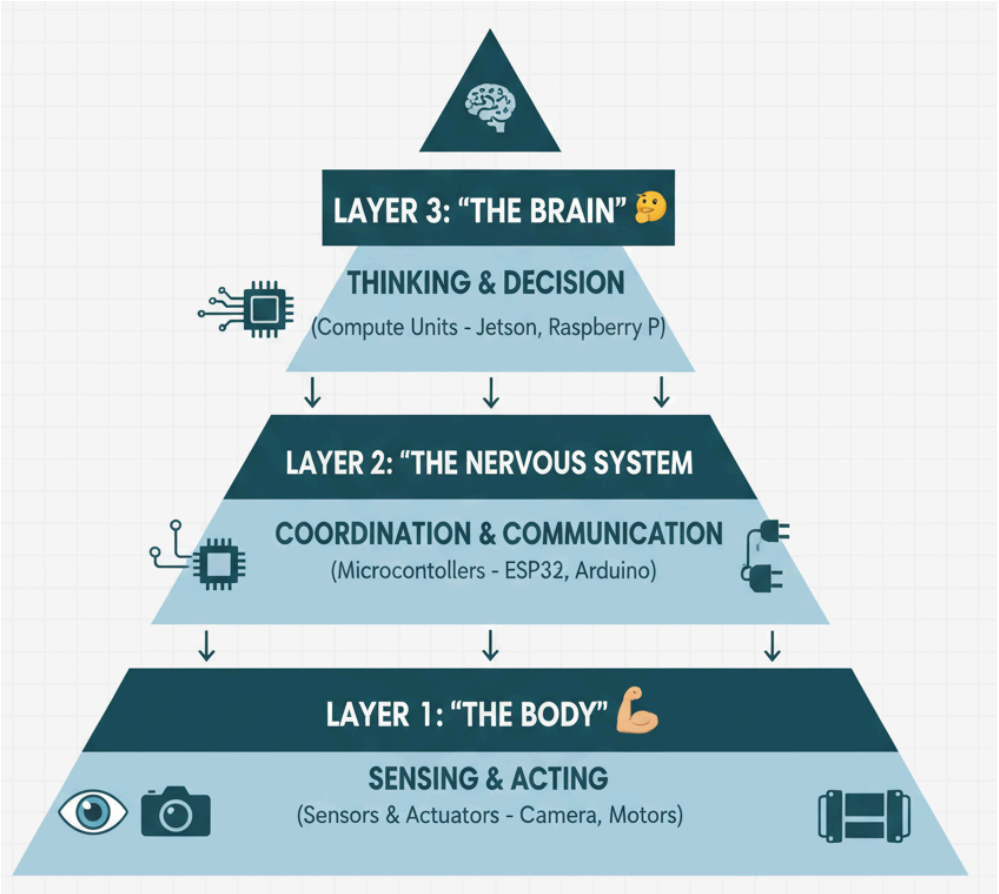



Figure 1: pyramid of robot

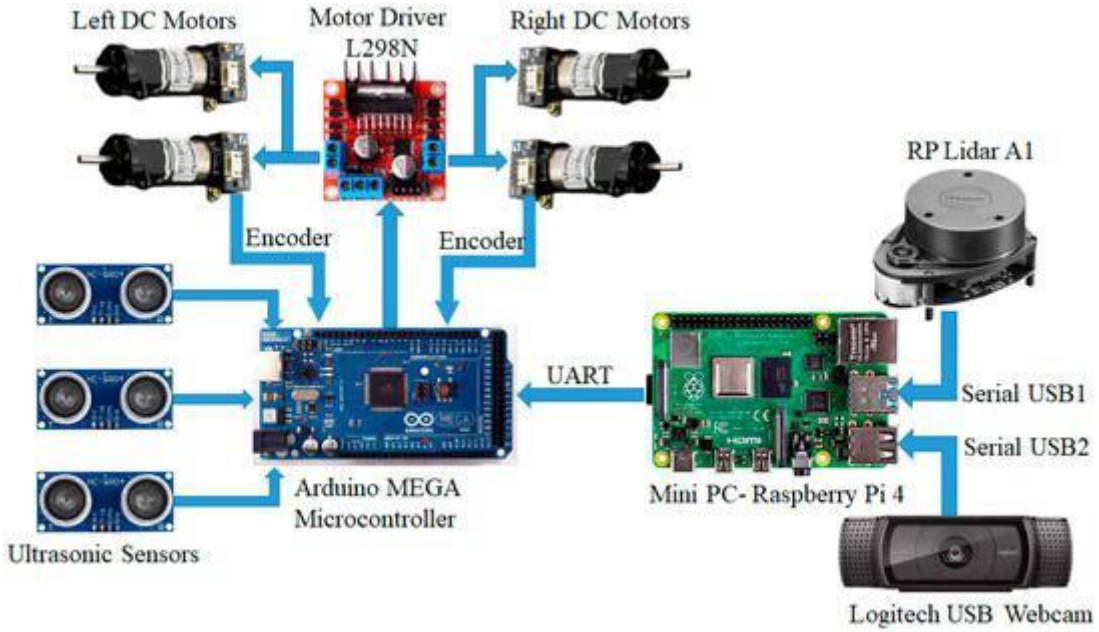



Figure2 : example of communication diagram of robot

**Key Layers:**

1. **Sensing/Actuation Layer**: Sensors (Camera, IMU) and actuators (Motors) at the edge.
2. **Interface/Control Layer:** Microcontrollers (ESP32, Arduino) that read sensors and drive actuators with low-level code.
3. **Compute Layer**: Single-Board Computers (SBCs like Jetson, RPi) that run ROS2 nodes for processing data and making decisions.
4. **Communication Fabric**: The physical (USB, Serial, Wi-Fi) and logical (ROS2/DDS) network that connects everything.

# 2. Compute Unit (Brain of the Robot)

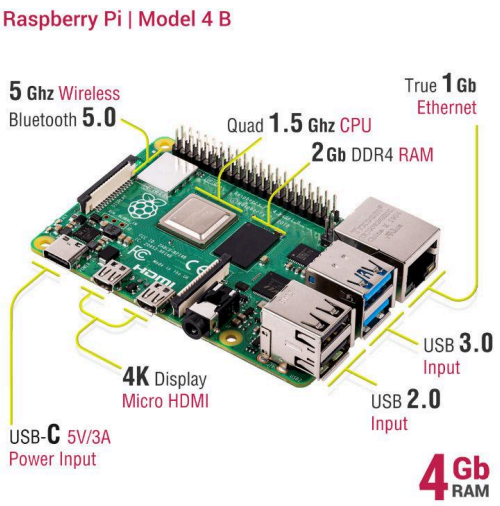
## Why the Compute Unit Matters

In a modern robot, the compute unit is the **central intelligence layer**. It is the system that executes ROS2 nodes, handles perception, makes decisions, performs localization, runs navigation algorithms, and manages communication with all sensors and actuators.

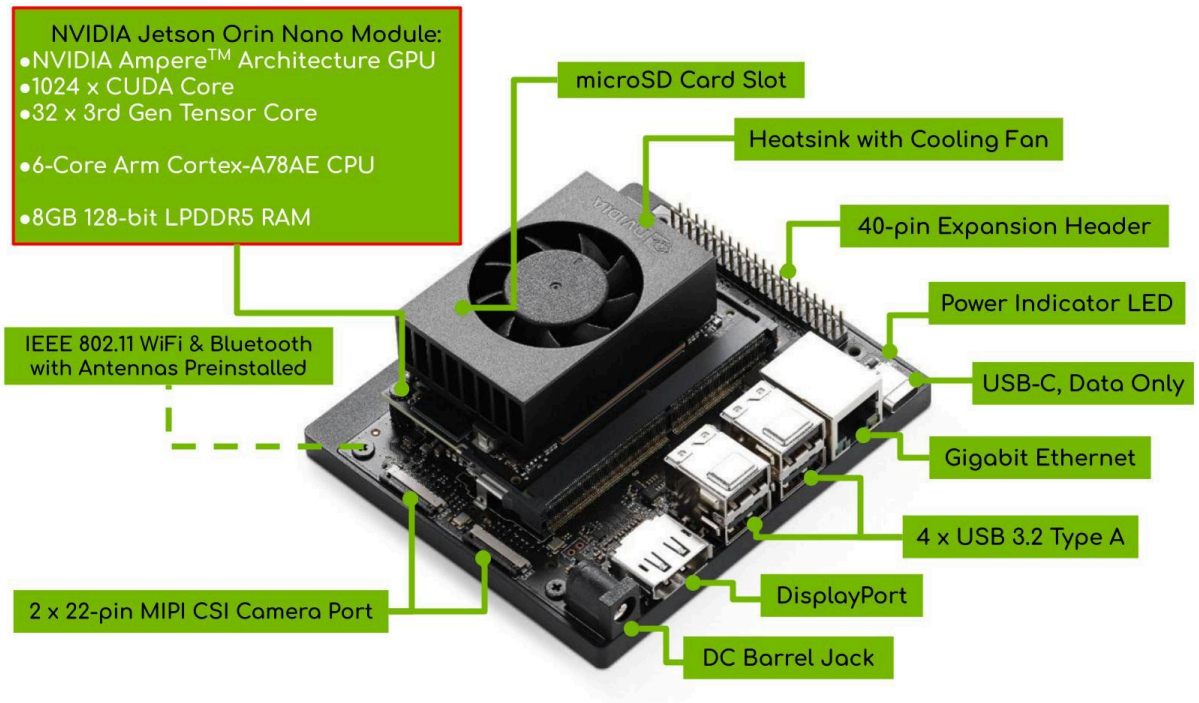
A robotics engineer must understand that **the compute unit is not just a computer** , it is the **core orchestrator of the entire robot ecosystem**.

## Common Compute Units

- **Raspberry Pi 4 / 5**
  Affordable, good for small robots, supports ROS2, runs Ubuntu 22.04.



- **NVIDIA Jetson Nano Orin**
  Designed for AI, computer vision, deep learning.




- **Intel NUC**
  High-performance, suitable for SLAM, 3D perception, industrial robotics.

- **Arduino UNO Q**

## Responsibilities

- Running ROS2 nodes
- Handling SLAM, navigation, sensor fusion
- Communicating with microcontrollers
- Running Python/C++ algorithms
- Recording data (rosbag)

The compute unit is the **high-level brain**.

## What a Robotics Engineer Must Know in This Step

You must understand:

### ✔ CPU & GPU Capabilities

- Jetson boards include CUDA-enabled GPUs → needed for AI, SLAM, depth processing.
- Raspberry Pi uses ARM CPU only → good for lightweight tasks.

Why this matters:
It determines **which algorithms you can run** (e.g., YOLO, ORB-SLAM3, RTAB-Map).

### ✔ Memory and Storage

- How much RAM is available (important for SLAM & mapping)
- Type of storage (SD card vs. NVMe vs. eMMC)
- Read/write speed (affects sensor logging and bag recording)

### ✔ Power Requirements

Robots often fail because the compute unit does not receive **clean and stable voltage**.

You must know:

- Required input voltage/amperage
- Peak current draw
- Power filtering and DC-DC conversion
- Overcurrent protection (fuses)

### ✔ I/O Interfaces

Essential for connecting sensors:

- USB 2.0 / USB 3.0 for cameras, LiDARs
- CSI for camera modules

- UART for microcontrollers
- I2C/SPI for IMUs
- Ethernet/Wi-Fi for ROS2 networking

A robotics engineer must know:
➡ which sensors need high bandwidth
➡ which ports support those sensors
➡ which interface is best based on latency + reliability

## ✔ OS installation (Ubuntu 20.04 / 22.04)

Robotics engineers must be able to:

- flash an image
- configure bootloader
- set hostname
- secure SSH access
- prepare networking (Wi-Fi, static IP, mDNS)

## ✔ ROS2 installation (Humble / Iron)

You must deeply understand:

- Which ROS2 version is compatible with your board
- How to source environments (bashrc setup)
- How to build workspaces with colcon
- How to install sensor drivers

**How-To: Flashing an OS onto a Compute Unit (Jetson Nano / Raspberry Pi)**

**Objective**: Prepare your compute unit with a fresh Ubuntu Server 22.04 LTS installation.

Steps for Raspberry Pi:

1. Download the Image: Download the pre-configured Ubuntu 22.04 LTS Server for Raspberry Pi image from the official Ubuntu website.
2. Flash the SD Card:
    - Insert a microSD card (32GB or larger recommended) into your computer.
    - Use a tool like Raspberry Pi Imager or Balena Etcher.
    - Select the downloaded `.img.xz` file and flash it to the SD card.
3. Enable SSH (Headless Setup):
    - After flashing, you will see a `system-boot` partition on your computer.
    - Create an empty file named `ssh` (no extension) in the root of this partition. This enables SSH on first boot.
4. First Boot & Network Setup:
    - Insert the SD card into the Raspberry Pi, connect power, and plug in an Ethernet cable or configure Wi-Fi by creating a `network-config` file as per Ubuntu's documentation.
    - Find your Pi's IP address from your router and connect via SSH: `ssh ubuntu@<pi_ip_address>`. The default password is `ubuntu`. You will be forced to change it.

Steps for Jetson Nano (Developer Kit):

1. Download the Image: Download the NVIDIA SDK Manager on a host computer
2. Prepare the Jetson:
    - Connect the Jetson Nano to the host computer via a USB-C cable.
    - Put the Jetson into Force Recovery Mode: Power off, hold the `FORCE RECOVERY` button, press the `POWER` button, wait 2 seconds, and release `FORCE RECOVERY`.
3. Flash with SDK Manager:
    - Open SDK Manager. It should detect the Jetson in recovery mode.
    - Select the target hardware (`Jetson Nano`) and choose Jetson Linux and Jetson SDK Components.
    - Proceed with the flashing process. This can take 20-40 minutes.

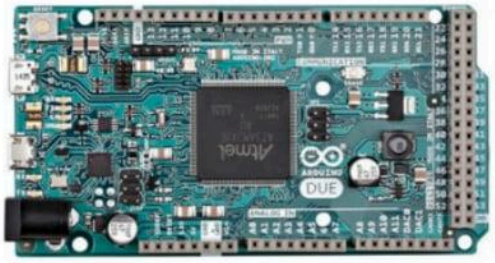
# 3. Microcontroller (Low-Level Control Layer)

Microcontrollers handle **real-time, low-latency tasks** that computers cannot reliably process.

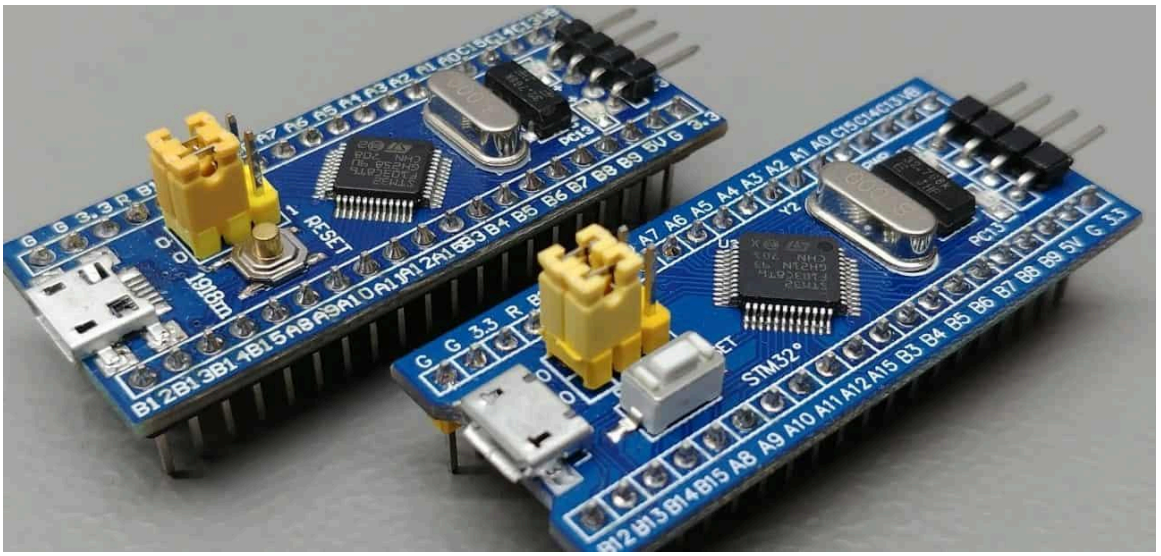
## Common Microcontrollers

- **Arduino (Uno, Mega)** → simple robots




- **STM32** → industrial-grade, fast, real-time



- **ESP32** → WiFi & Bluetooth built-in, micro-ROS compatible

### Responsibilities

- Reading sensors (encoders, IMU, ultrasonic)
- Generating PWM for motors
- Implementing PID speed control
- Ensuring deterministic timing
- Sending data to ROS2

The microcontroller is the **real-time layer** of the robot.

## 4. Part 3: Sensor Overview & Connection

Before calibration and fusion, a robotics engineer must **understand the physical sensors**, how they work, how to connect them, and how to verify proper communication.

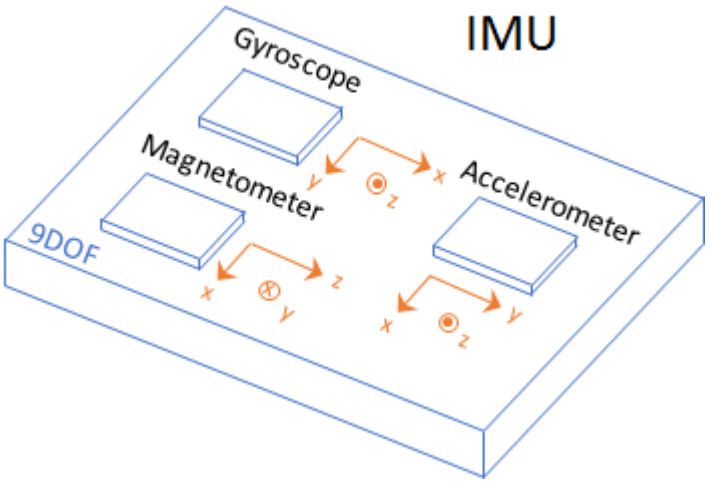
Robots rely on sensors to perceive the environment. The compute unit must receive this data **reliably, continuously, and with correct timing**. This part ensures your trainees understand:

- what each sensor measures
- how it connects electrically
- how to check that the system sees it
- how data flows into ROS2

## A. IMU — Inertial Measurement Unit


An IMU is composed of:

- **Accelerometer** → measures linear acceleration (x, y, z) (m/s2,m/s2,m/s2)
- **Gyroscope** → measures angular velocity (roll, pitch, yaw rates) (rad/s,rad/s,rad/s)
- (sometimes) **Magnetometer** → measures heading relative to Earth's magnetic field (vecteur)



**Why it matters:**

IMUs give the robot its **orientation**, detect movement, and serve as the core of **odometry** and **sensor fusion**.

### IMU Data Usage in ROS2

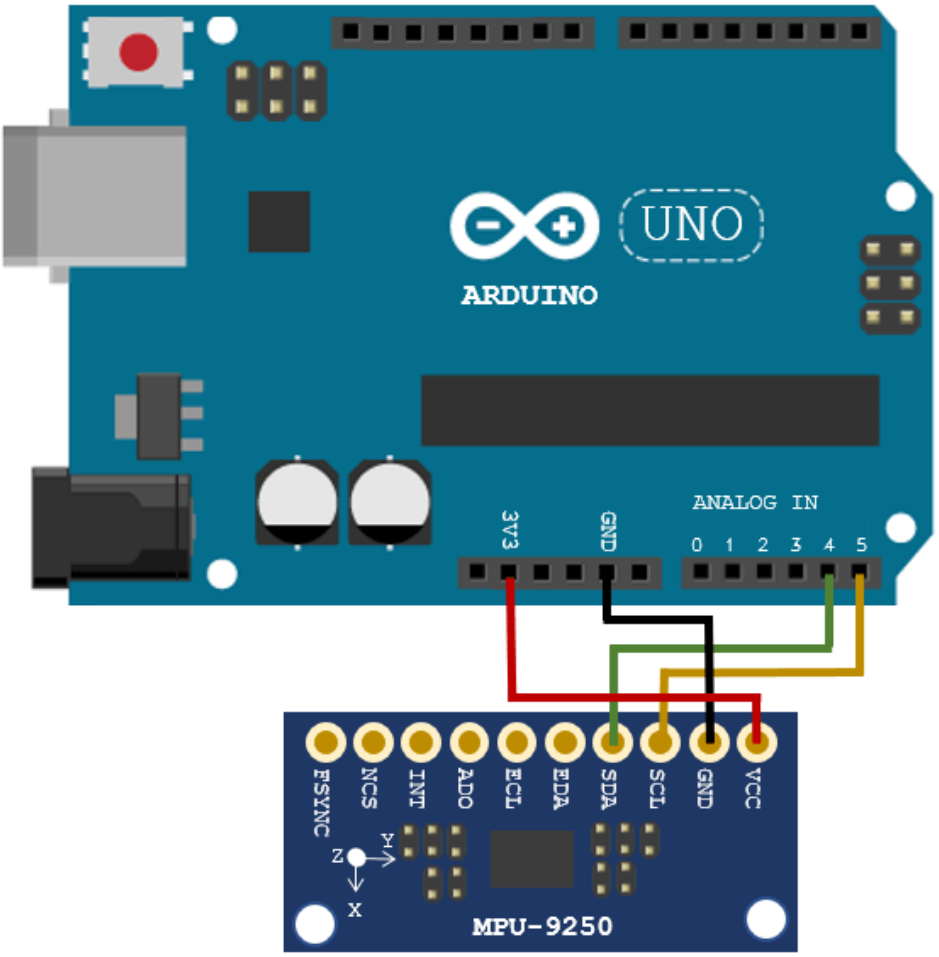
- `/imu/data_raw` → accelerometer + gyro
- `/imu/data` → fused orientation (from Madgwick/Complementary filter)

### Typical IMU Connection Protocols

| Protocol | Pros | Cons | Notes |
|---|---|---|---|
| **I2C** | simple, low wires | limited speed | good for low-rate IMUs |
| **SPI** | fast, low latency | needs more pins | best for high-rate IMUs |
| **UART** | robust | requires driver | used in industrial IMUs |
| **USB** | plug-and-play | higher power draw | common for advanced IMUs |

### Electrical considerations

- IMUs usually require **3.3V or 5V**
- I2C wires must not exceed ~30 cm
- SPI requires clean grounding
- USB IMUs require stable current

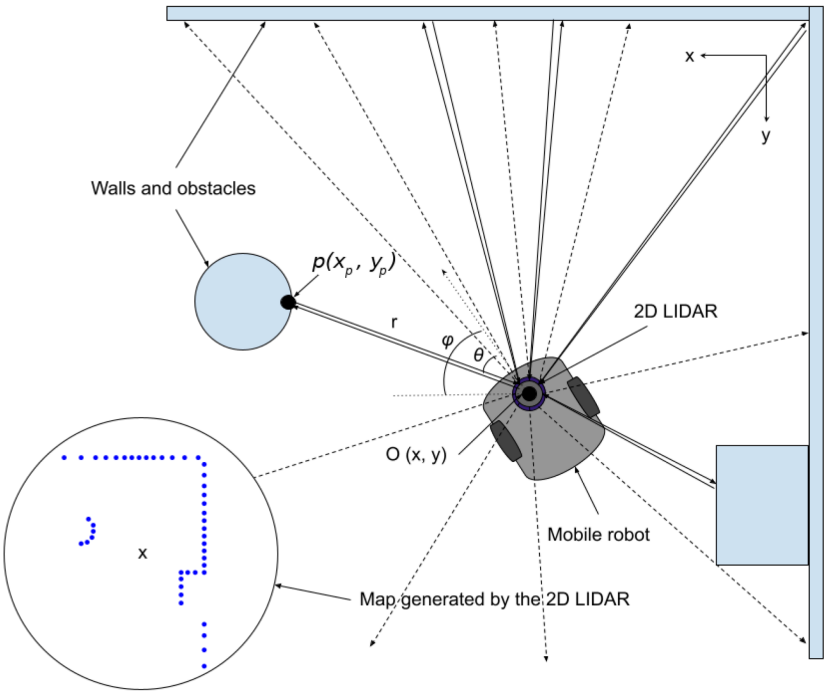



## B. LiDAR — Light Detection and Ranging

LiDARs generate **2D or 3D point clouds** by firing laser pulses and measuring the return time.

**What LiDAR provides:**

- Distances to obstacles
- 360° scans (2D LiDAR)
- 3D depth layers (3D LiDAR)
- High-frequency scans (5–20 Hz)




**LiDAR ROS2 Topics**

- `/scan` (2D)
- `/points` or `/points_raw` (3D)

**Message type:** `sensor_msgs/msg/LaserScan`

```
std_msgs/Header header          # Standard ROS2 message header

float32 angle_min              # Start angle of the scan [rad]

float32 angle_max              # End angle of the scan [rad]

float32 angle_increment        # Angular distance between measurements [rad]

float32 time_increment         # Time between measurements [seconds]

float32 scan_time              # Time between scans [seconds]

float32 range_min              # Minimum range value [meters]

float32 range_max              # Maximum range value [meters]

float32[] ranges               # Distance data [meters] for each angle

float32[] intensities            # Intensity data (optional, e.g., reflected laser
strength)
```

## Typical LiDAR Communication Interfaces

| Interface | Use Case | Notes |
|---|---|---|
| **USB** | 2D LiDARs (RPLIDAR, YDLIDAR) | plug-and-play<br>serial |
| **Ethernet** | 3D LiDARs (Ouster, Velodyne) | high bandwidth |
| **UART** | small LiDAR modules | lower refresh rate |

**Bandwidth Requirements**

- USB 2.0 → OK for 2D LiDAR
- USB 3.0 / Ethernet → required for 3D LiDAR

**Important:**
LiDARs require stable 5V power. USB ports sometimes cannot supply enough current → use powered hubs.

# C. Depth Camera — Intel RealSense

Depth cameras provide **rich perception data** by capturing color (RGB) images, depth images, and point clouds. They are critical for **obstacle detection, 3D mapping, visual odometry, and autonomous navigation**.

Robotics engineers must understand:

- How depth sensing works
- Connection and data flow
- Integration into ROS2
- Synchronization with other sensors

**Why robots use depth cameras:**

- obstacle detection
- SLAM
- visual odometry
- human/marker tracking
- 3D environment reconstruction

## Output Data

- **Color image (RGB)** → `/color/image_raw`
- **Depth image (distance per pixel)** → `/depth/image_raw`
- **Aligned color-depth image** → `/aligned_depth_to_color`
- **3D point cloud** → `/points`

**Connection Requirements**

- **USB 3.0 mandatory** for RealSense
- **USB 3.1+** recommended for ZED cameras
- High bandwidth → avoid running on USB hubs

**Power Consideration:**

RealSense D435/D455 requires **5V 1A** USB current.

# 5. Part 4: Establishing Communication Links

Create the **physical and logical bridges** between the robot's hardware components so they can reliably communicate with each other and with the ROS2 ecosystem.

Robotics engineers must ensure that **compute units, microcontrollers, and sensors** are all interconnected, **data flows correctly**, and the system can handle real-time requirements.

## Why Communication Links Matter

- Robots are **distributed systems**: compute unit, microcontroller, sensors, and actuators all have separate roles.
- Proper links ensure:
    - Low-latency data delivery
    - Deterministic timing for real-time control
    - Accurate sensor fusion
    - Stable ROS2 topic publishing/subscribing

**Challenges in real hardware vs simulation:**

- Noise, latency, jitter
- Signal loss or disconnections
- Bandwidth limitations
- Electrical interference

# Types of Communication

## A. Physical Links

| Link Type | Components | Notes |
|---|---|---|
| **USB** | IMU, LiDAR, Depth Camera | Easy plug-and-play, high bandwidth, powered USB recommended |
| **UART / Serial** | Microcontrollers, IMU | Low bandwidth, long cable support, robust |
| **SPI / I2C** | IMU, encoders | Short-range, low-latency, shared bus (I2C) |
| **Ethernet** | 3D LiDAR, NUC/Jetson | High throughput, low latency, long cables |

## B. Logical Links (ROS2)

- **ROS2 nodes** communicate using topics, services, actions.
- **DDS Middleware** ensures:
    - Reliable or best-effort delivery
    - QoS (Quality of Service) management
    - Automatic discovery of nodes
- Typical data flows:
    - Microcontroller → Compute Unit → `/imu/data`
    - LiDAR → Compute Unit → `/scan` or `/points`
    - Depth Camera → Compute Unit → `/depth/image_raw`