Robotics Hardware and Software

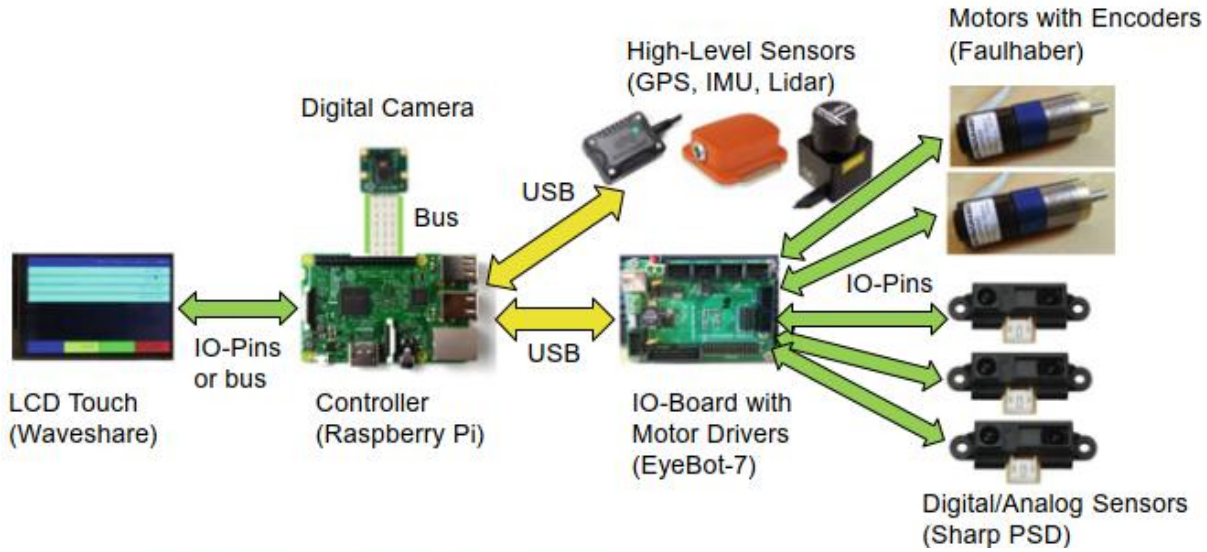
DASE7503 Robotic Systems Integration

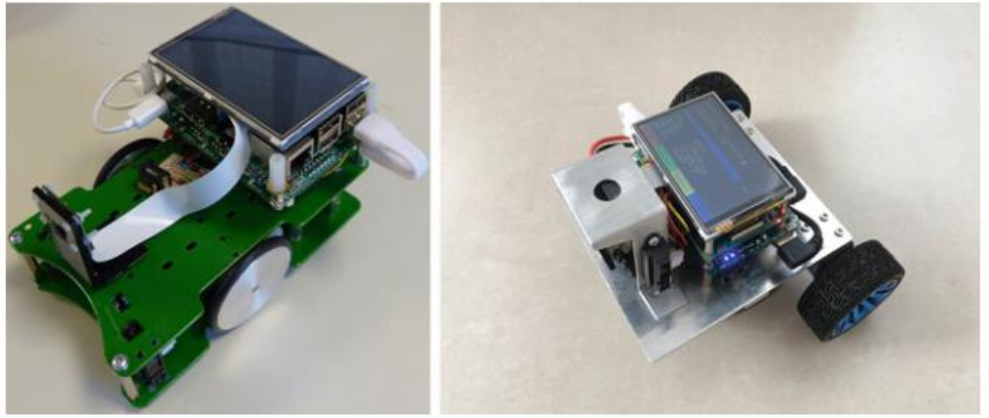
Contents

- Robotics Hardware Platforms
- Controllers
- Software, Software Architecture & Frameworks
- Communication Protocols
- Design and Manufacturing for Robotics

Robotics Hardware Platforms

- A robotics hardware platform refers to **the physical components** and **systems** that form the foundation of a robot, enabling it to perform tasks, sense its environment, and interact with its surroundings. It includes the **mechanical structure**, **sensors**, **actuators**, **control systems**, **power sources**, and **communication interfaces** that work together to realise the robot's functionality.





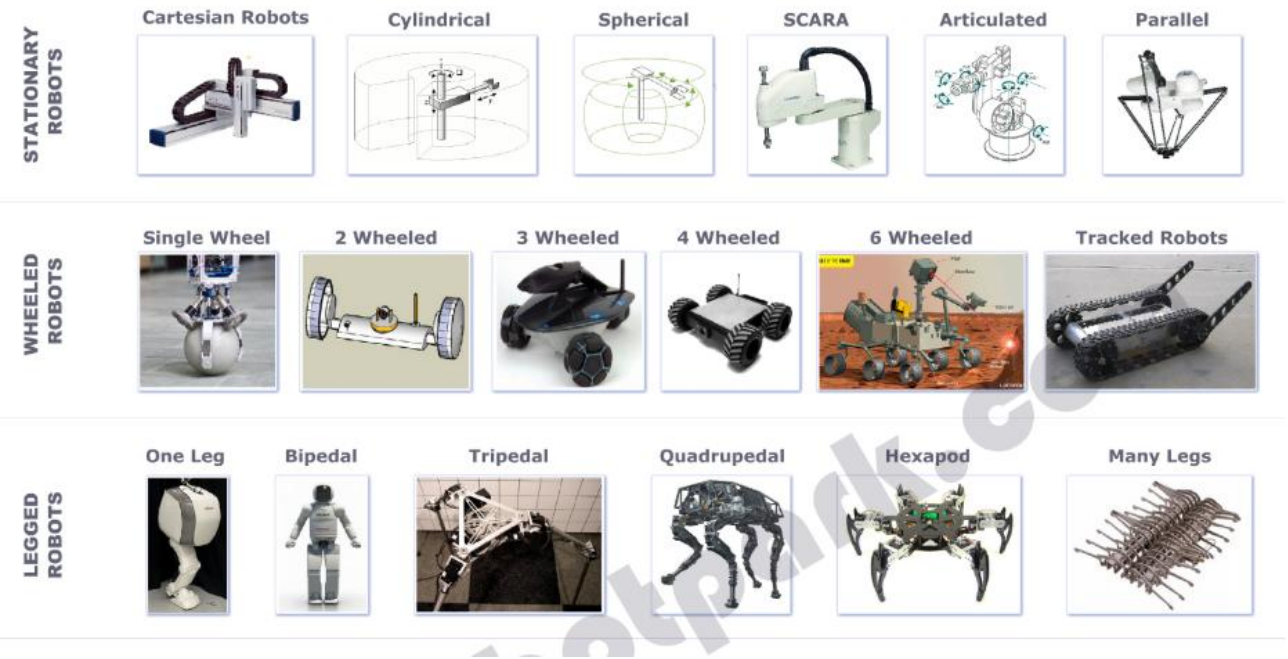
An example of a robotics hardware platform

Robotics Hardware Platforms

- The robotics hardware platforms are designed and fabricated in following common types:
  - Wheeled (Driving) Robots
  - Manipulators
  - Legged/Walking Robots
  - Drones



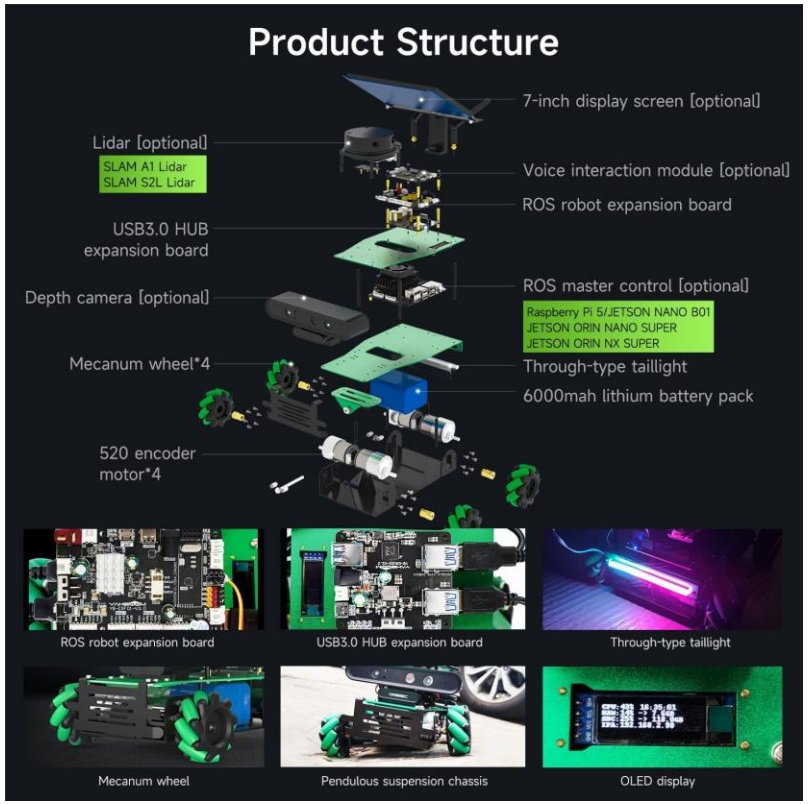




https://www.robotpark.com/All-Types-Of-Robots

Robotics Hardware Platforms – Wheeled Robots

- The Wheeled Robots can be built as different designs:
  - Single Wheel Drive
  - Differential Drive
  - Tracked Robots
  - Synchro-Drive Robots
  - Omni-Directional Robots

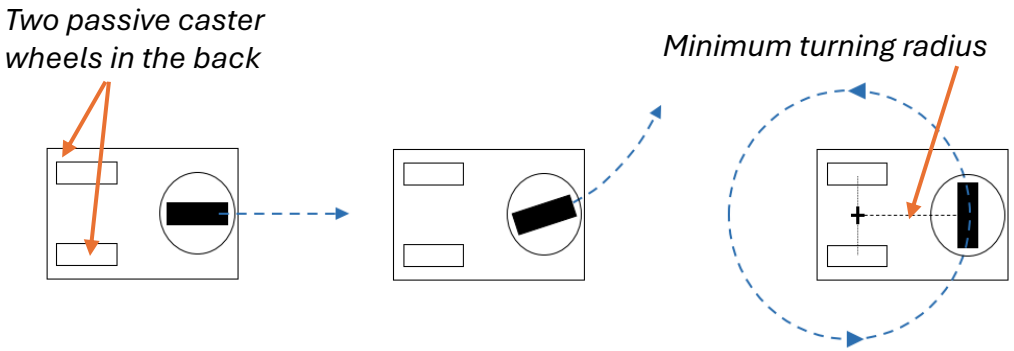


https://www.amazon.com/Yahboom-Raspberry-Navigation-Programming-Ver-without/dp/B0B1JB32TP?th=1

Robotics Hardware Platforms – Wheeled Robots

Single Wheel Drive

- A single wheel that is both driven and steered is the simplest conceptual design for a mobile robot.
- This design requires two passive caster wheels in the back.
- Linear velocity and angular velocity of the robot are completely decoupled.
- For driving straight, the front wheel is positioned in the middle position and driven at the desired speed.
- For driving in a curve, the wheel is positioned at an angle matching the desired curve.
- Curve driving is following the arc of a circle; however, this robot design cannot turn on the spot.
- With the front wheel set to 90°, the robot will rotate about the midpoint between the two caster wheels.

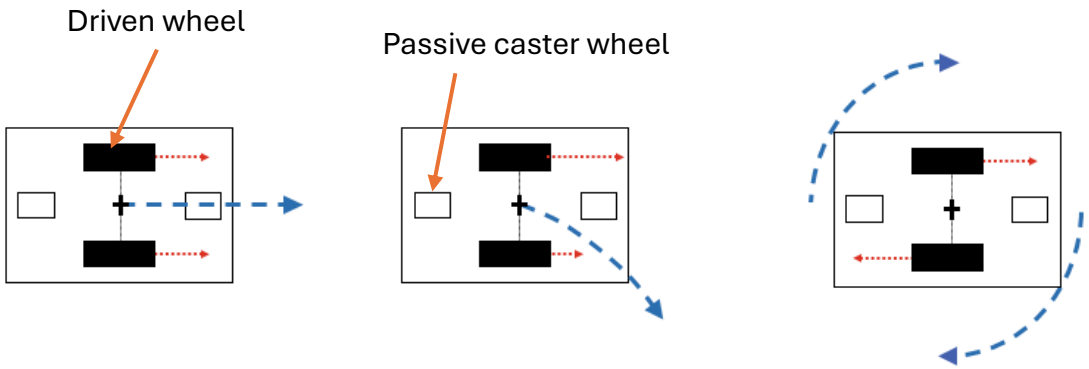


Driving and rotation of single wheel drive

Robotics Hardware Platforms – Wheeled Robots

Differential Drive

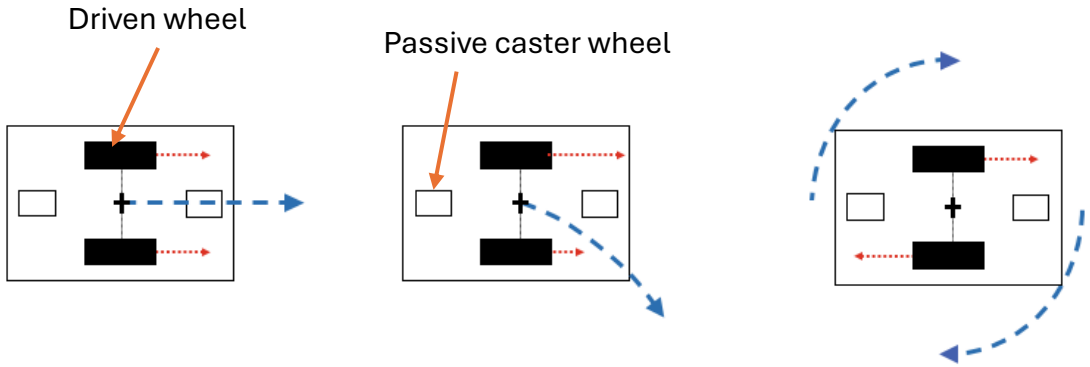
- Its design has two motors mounted in fixed positions on the left and right sides of the robot, independently driving one each.
- Since at least **three ground contact points** are necessary, this design requires **one** or **two additional passive caster wheels** or **sliders**, depending on the location of the driven wheels.
- Differential drive is mechanically simpler than the single wheel drive, because it does not require rotation of a driving motor.
- However, directional control for differential drive is more complex than for single wheel drive, because it requires the coordination of two driven wheels.



Robotics Hardware Platforms – Wheeled Robots

Differential Drive

- The minimal differential drive design with **only a single passive wheel** cannot have the driving wheels in the middle of the robot for stability reasons.
- When turning on the spot, the robot will rotate about the off-center midpoint between the two driven wheels.
- The design with **two passive wheels or sliders**, one at the front and one at the back of the robot, allows rotation about the center of the robot.
- However, this design can introduce surface contact problems, because it is using four contact points instead of three.



Driving and rotation of differential drive

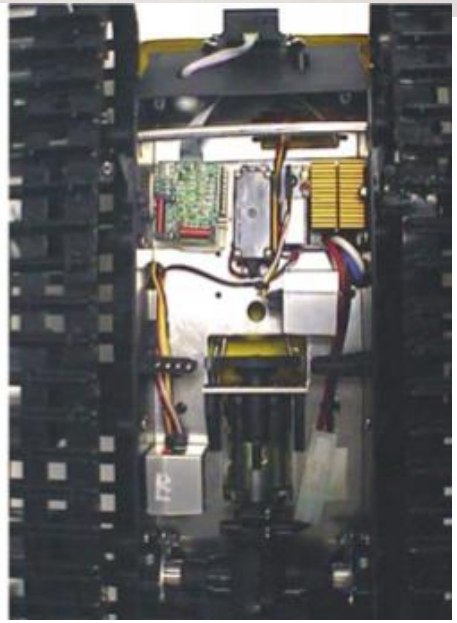
- Driving straight forward: $v_L = v_R$, $v_L > 0$
- Driving in a right curve: $v_L > v_R$
- Turning clockwise on the spot: $v_L = -v_R$, $v_L > 0$

Robotics Hardware Platforms – Wheeled Robots

Tracked Robots

- A tracked mobile robot can be seen as a special case of a wheeled robot with differential drive.
- In fact, the only difference is the robot's better maneuver-ability in **rough terrain** and its higher friction in turns, due to its tracks and multiple points of contact with the surface.



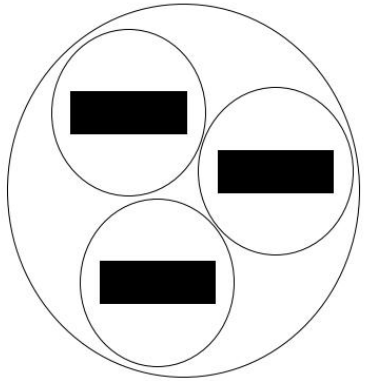


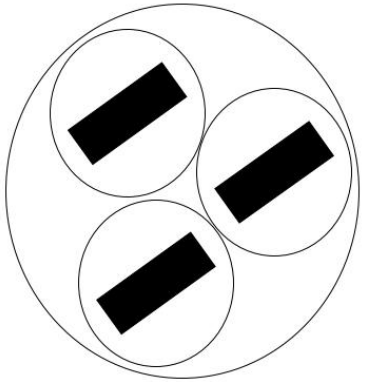
Robotics Hardware Platforms – Wheeled Robots

Synchro-Drive

- An extension to the robot design with a single driven.
- Three wheels, which are driven and all being steered, are rotated together so they always point in the same driving direction.
- Can be accomplished, for example, by using a single motor and a chain for steering and a single motor for driving all three wheels.
- Overall, a synchro-drive robot still has only two degrees of freedom.



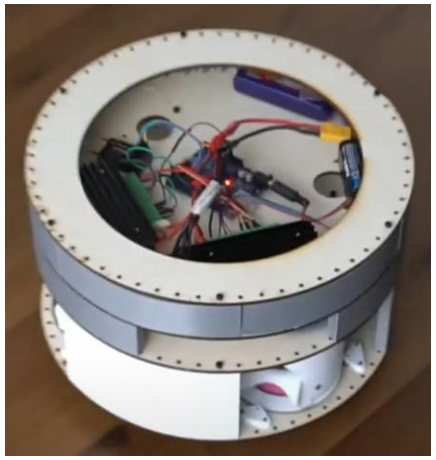




Robotics Hardware Platforms – Wheeled Robots

Synchro-Drive

- almost an omni-directional (holonomic) vehicle, in the sense that it can drive in any desire direction (for this reason, it usually has a cylindrical body shape).
- Limitations of a synchro-drive robot are
  - Stop and realign its wheels when going from driving forward to driving sideways.
  - Cannot drive and rotate it at the same time
- An example task that demonstrates the advantages of a synchro-drive is "complete area coverage" of a robot in a given environment. The real-world equivalent of this task is cleaning or vacuuming floors

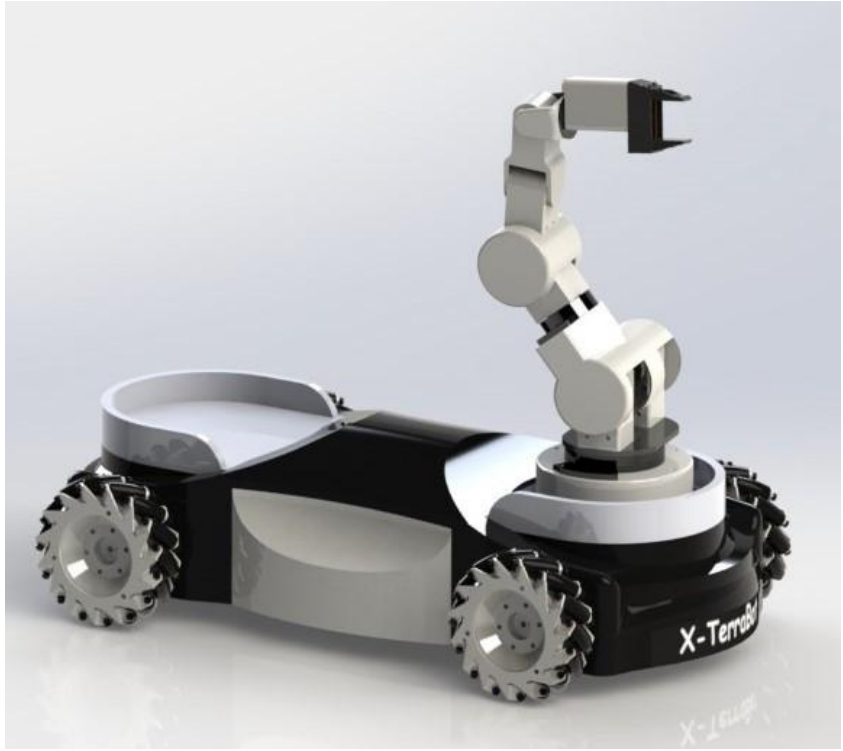




Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots

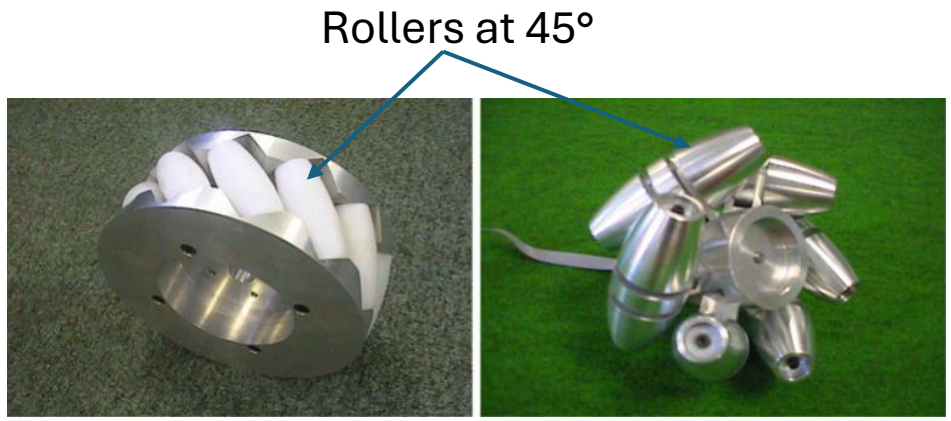
- All the robots discussed so far, with the exception of synchro drive vehicles, have the same deficiency - they cannot drive in all possible directions. For this reason, these robots are called **non-holonomic**.
- In contrast, a **holonomic** or **omni-directional** robot is capable of driving in any direction.
- Most non-holonomic robots cannot drive in a direction perpendicular to their driven wheels. For example, a **differential drive robot** can drive forward/backward, in a curve, or turn on the spot, but it **cannot drive sideways**.
- The omni-directional robots introduced are capable of driving in any direction in a 2D plane.



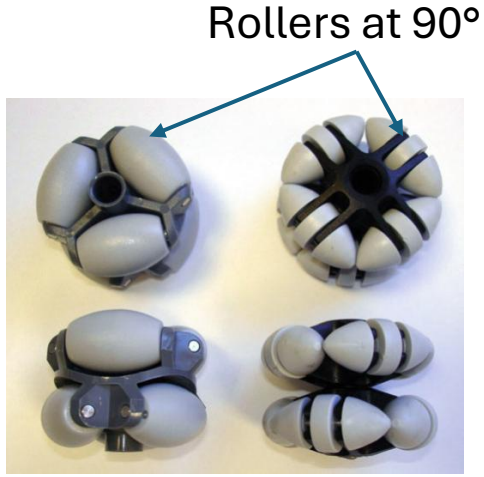
Robotics Hardware Platforms – Wheeled Robots

Omni Directional Robots – *Mecanum Wheels*

- Developed and patented by the Swedish company in 1973.
- A number of different Mecanum wheel variations.
- Each wheel's surface is covered with **a number of free rolling cylinders**. It is important to stress that the wheel hub is driven by a motor, but the rollers on the wheel surface are not.
- The wheels have the rollers at ±45° and there is a left-hand and a right-hand version of this wheel type; there are also Mecanum wheels with rollers set at 90° , and these do not require left-hand/right-hand versions.



Mecanum wheel designs with rollers at 45°

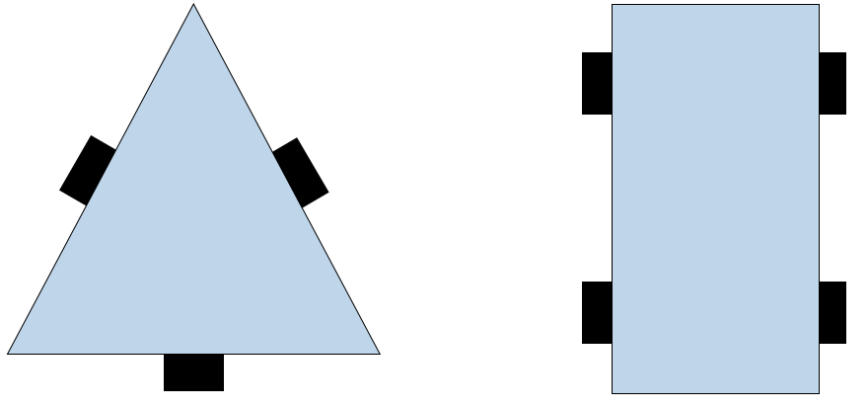


Mecanum wheel designs with rollers at 90°

Robotics Hardware Platforms – Wheeled Robots

Omni Directional Robots – *Mecanum Wheels*

- **A Mecanum-based robot** can be constructed with either **three** or **four** independently driven Mecanum wheels.
  - Robot designs with three Mecanum wheels require rollers set at 90° to the wheel axis.
  - Robot designs with four Mecanum wheels require rollers set at an angle of ±45° to the wheel axis.
- For the construction of a robot with four Mecanum wheels, two **left-handed** wheels (rollers at + 45° to the wheel axis) and two **right-handed** wheels (rollers at –45° to the wheel axis) are required.

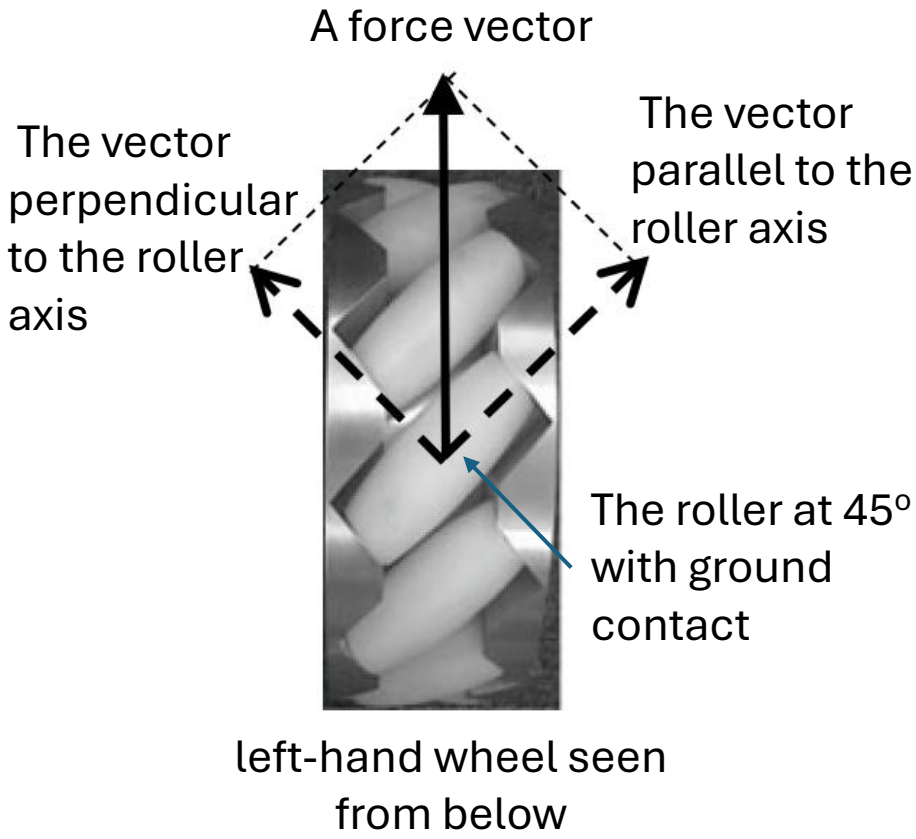


Three-wheel and four-wheel omni-directional robots

Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots – *Mecanum Wheels*

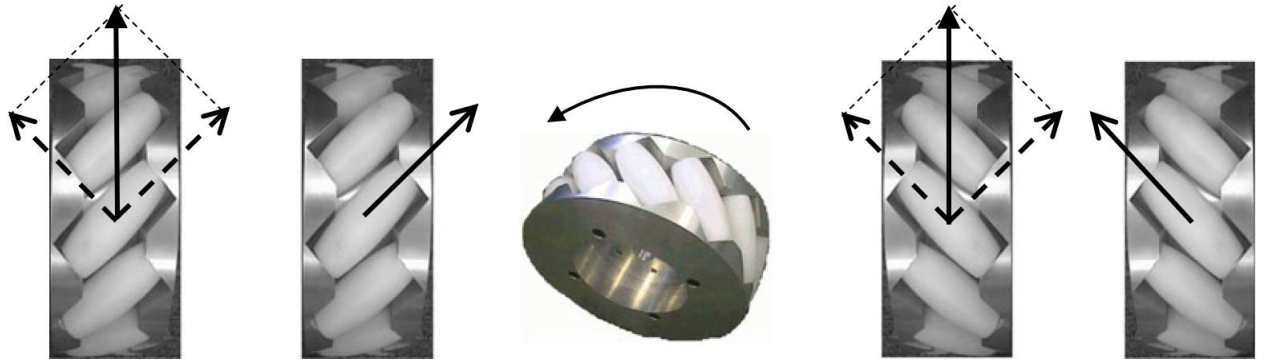
- Although the rollers are freely rotating, this does not mean the robot can spin its wheels without moving. This would only be the case if the rollers were placed parallel to the wheel axis.
- Looking at an individual wheel from the bottom view, the force generated by the wheel rotation acts on the ground through the one roller that has ground contact.
  - At this roller, the force can be split in a vector parallel to the roller axis and a vector perpendicular to the roller axis.
  - The force perpendicular to the roller axis will result in a small roller rotation, while **the force parallel to the roller axis will exert a force on the wheel and thereby on the robot**.



Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots – *Mecanum Wheels*

- In a four-wheel assembly, **the resulting wheel forces at 45°** from each wheel have to be combined to determine the overall robot motion.
- If the two wheels, as shown in Figure, are the robot's front wheels and both are rotated forward, then each of the two resulting 45° force vectors can be split into a forward and a sideways force.
- The two forward forces add up, while the two sideways forces (one to the left and one to the right) cancel each other out.



left-hand wheel seen from below

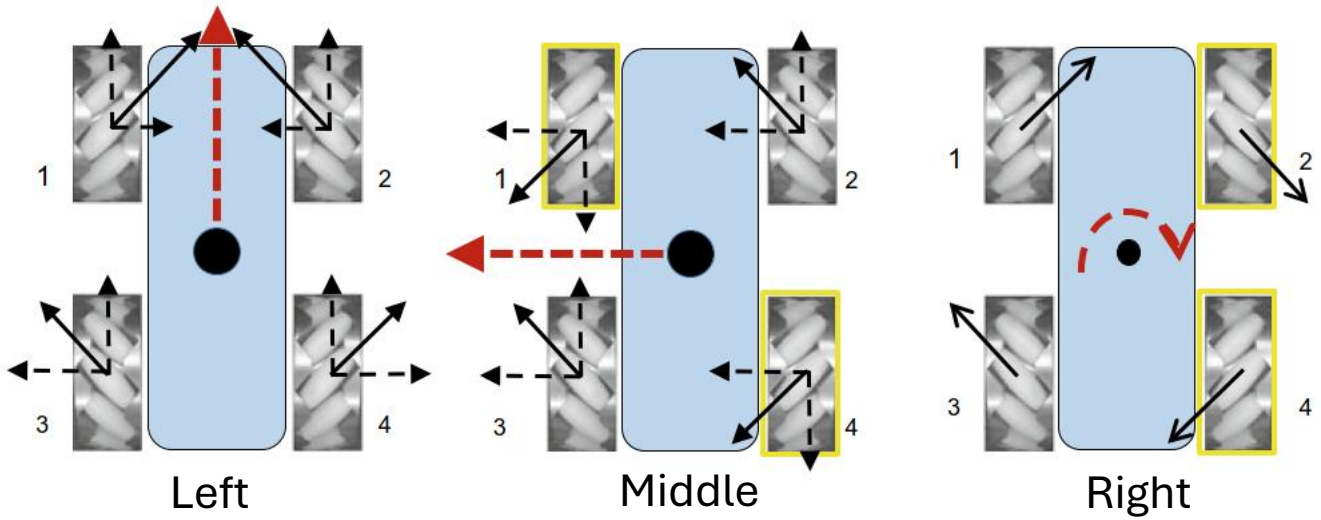
right-hand wheel seen from below

Mecanum principle, vector decomposition

Robotics Hardware Platforms – Wheeled Robots

Omni Directional Robots – *Omni-Directional Drive*

- The **left** in the figure shows the situation for the full robot with four independently driven Mecanum wheels.
- All four wheels being driven forward, it means four vectors pointing forward that are added up and four vectors pointing sideways, two to the left and two to the right, that cancel each other out.
- Therefore, although the robot's chassis is subjected to additional perpendicular forces, the vehicle will simply drive straight forward.

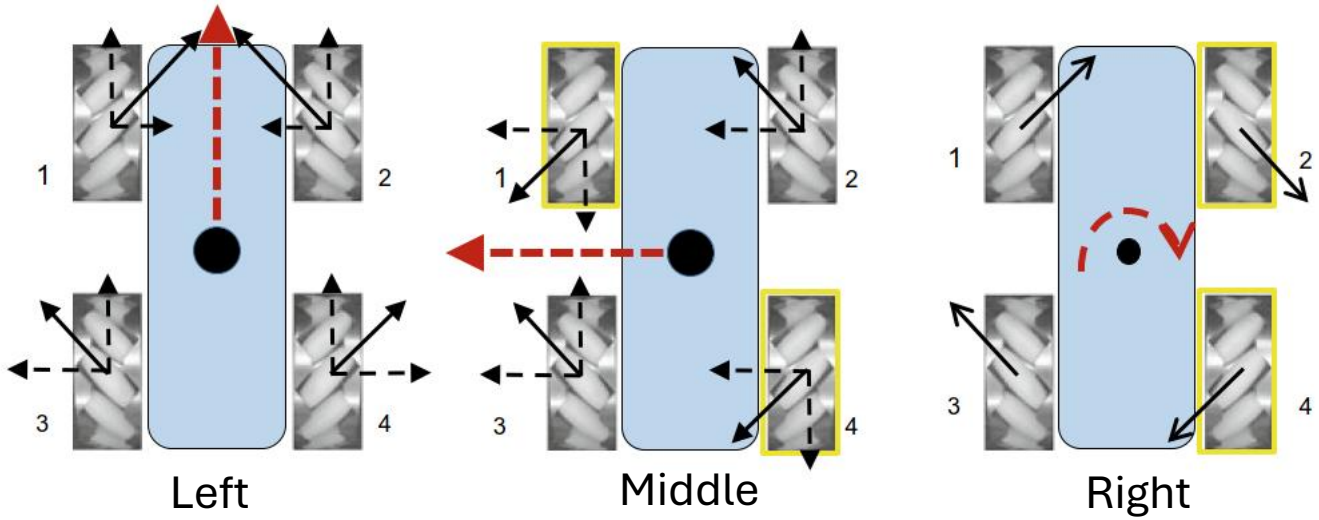


Mecanum principle for driving forward, sliding sideways and rotating; wheels without outline rotate forward, wheels with yellow outline rotate backward

Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots – *Omni-Directional Drive*

- The **middle** in the figure assumes wheels 1 and 4 are driven backward, and wheels 2 and 3 are driven forward.
- In this case, all forward/backward velocities cancel each other out, but the four vector components to the left add up and let the robot slide to the left.

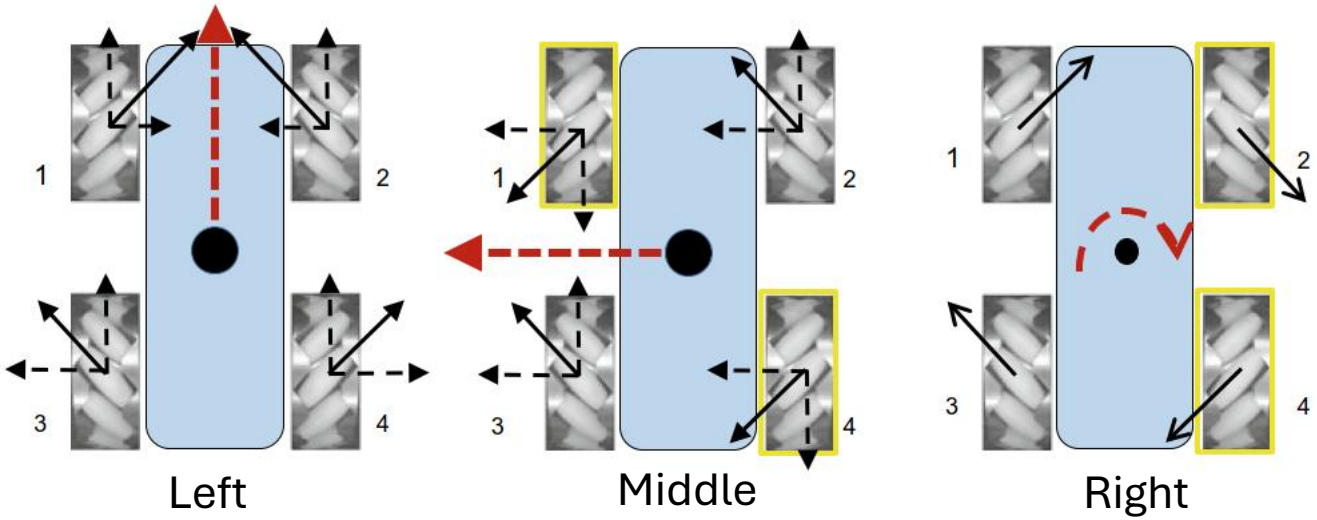


Mecanum principle for driving forward, sliding sideways and rotating; wheels without outline rotate forward, wheels with yellow outline rotate backward

Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots – *Omni-Directional Drive*

- The **right** in the figure shows that no vector decomposition is necessary in this case to reveal the overall vehicle motion.
- It can be clearly seen that the robot motion will be a clockwise rotation about its center.



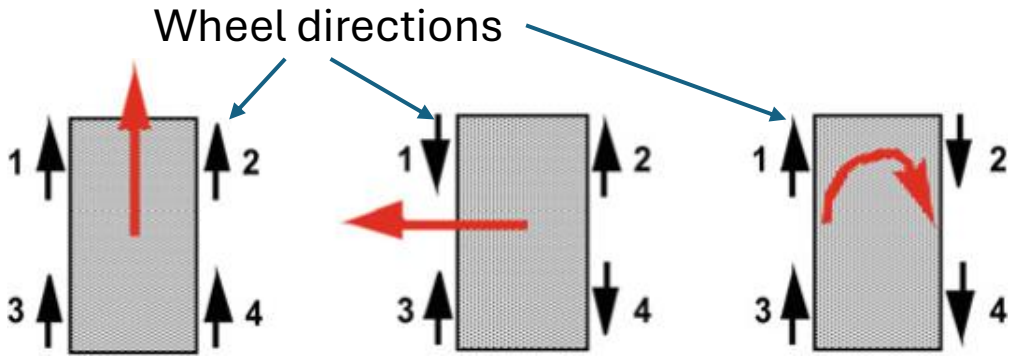
Mecanum principle for driving forward, sliding sideways and rotating; wheels without outline rotate forward, wheels with yellow outline rotate backward

Robotics Hardware Platforms – Wheeled Robots

Omni-Directional Robots – *Omni-Directional Drive*

- The following list shows the **basic motions**, *driving forward*, *driving sideways* and *turning on the spot*, with their corresponding wheel directions, as shown in Figure.

Basic motions	Corresponding Wheel Directions
Drive forward:	all four wheels forward
Driving backward:	all four wheels backward
Sliding left:	1, 4: backward; 2, 3: forward
Sliding right:	1, 4: forward; 2, 3: backward
Turning clockwise on the spot:	1, 3: forward; 2, 4: backward
Turning counterclockwise :	1, 3: backward; 2, 4: forward



Kinematics of an omni-directional robot

Robotics Hardware Platforms – Wheeled Robots

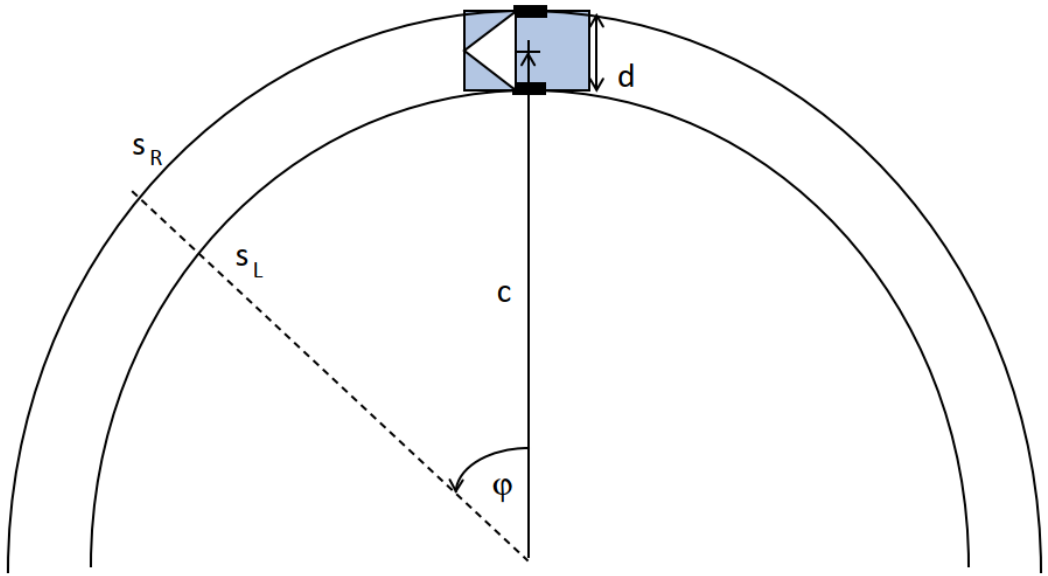
Drive Kinematics

- Two fundamental questions:
  a) Given **each wheel's movements**, what is the robot's change in position and orientation (pose)?
  b) Which individual wheel speeds will be required, in order to achieve a desired change in robot pose?
- The solution to question a) is called **forward kinematics**, the solution to question b) is called **inverse kinematics**.
- The two most prominent drive systems: Differential drive & Omni-directional drive.

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- For obtaining a differential drive robot's current trajectory, its shaft encoders should be constantly monitored.



Trajectory calculation for differential drive

- r — wheel radius
- d — distance between driven wheels
- ticks_per_rev — number of encoder ticks for one full wheel revolution
- $ticks_L$ — number of ticks during measurement in left encoder
- $ticks_R$ — number of ticks during measurement in right encoder
- s_L — the distance in meter traveled by the left wheel
- s_R — the distance in meter traveled by the right wheel

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- Determine the values of s_L and s_R in meters, which are the distances traveled by the left and right wheels, respectively.
- Dividing the measured ticks by the number of ticks per revolution gives us the number of wheel revolutions.
- Multiplying this by the wheel circumference gives the traveled distance in meters:

$$s_L = 2\pi \cdot r \cdot \text{ticks}_L / \text{ticks_per_rev}$$

$$s_R = 2\pi \cdot r \cdot \text{ticks}_R / \text{ticks_per_rev}$$

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- The distance the robot has traveled, i.e., $s=(s_L + s_R)/2$.
- This formula works for a robot driving forward, backward or turning on the spot.
- Still need to know the robot's rotation φ over the distance traveled.
- Assuming the robot follows a circular segment, s_L and s_R can be defined as the traveled part of a full circle (φ in radians) multiplied by each wheel's turning radius.
- If the turning radius of the robot's center is c, then during a left turn, the turning radius of the right wheel is c + d/2, while the turning radius of the left wheel is c – d/2. Both circles have the same center.

$$s_R = \varphi \cdot (c + d/2)$$

$$s_L = \varphi \cdot (c - d/2)$$

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- Subtracting both equations eliminates c: $s_R - s_L = \varphi \cdot d$
- And finally solving for φ : $\varphi = (s_R - s_L)/d$
- Using wheel velocities $v_{L,R}$ instead of driving distances $s_{L,R}$ and using $\dot{\theta}_{L\ R}$ as wheel rotations per second with radius r for left and right wheels, we get:

$$v_{\mathrm{L}} = 2\pi r \cdot \dot{\theta}_L$$

$$v_{\mathrm{R}} = 2\pi r \cdot \dot{\theta}_R$$

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- The formula specifying the velocities of a differential drive robot can now be expressed as a matrix. This is called the forward kinematics:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = 2\pi r \begin{bmatrix} 1/2 & 1/2 \\ -1/d & 1/d \end{bmatrix} \begin{bmatrix} \dot{\theta}_L \\ \dot{\theta}_R \end{bmatrix}$$

where

v	is the robot's linear speed (equals ds/dt or $\dot{s}$),
ω	is the robot's rotational speed (equals dθ/dt or $\dot{\theta}$),
$\dot{\theta}_{L,R}$	are the individual wheel speeds in revolutions per second,
r	is the wheel radius,
d	is the distance between the two driven wheels.

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Differential Drive Kinematics

- The **inverse kinematics** is derived from the previous formula, solving for the individual wheel speeds.
- It tells us the required wheel speeds for a desired vehicle motion (linear and rotational speed).
- The inverse kinematics by inverting the 2 x 2 matrix of the forward kinematics can be found:

$$\begin{bmatrix} \dot{\theta}_L \\ \dot{\theta}_R \end{bmatrix} = \frac{1}{2\pi r} \begin{bmatrix} 1 & -d/2 \\ 1 & d/2 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Omni-Drive Kinematics

- The **forward kinematics** for our four-wheel omni-drive system is a matrix formula that specifies which direction the robot will drive in (linear velocity v_x along the robot's center axis, v_y perpendicular to it) and what its rotational velocity ω will be for the given individual wheel speeds $\dot{\theta}_{FL}$, ..., $\dot{\theta}_{BR}$ and wheel distances d (left/right) and e (front/back):

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = 2\pi r \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ -1/4 & 1/4 & -1/4 & 1/4 \\ \frac{-1}{2(d+e)} & \frac{1}{2(d+e)} & \frac{-1}{2(d+e)} & \frac{1}{2(d+e)} \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_{FL} \\ \dot{\theta}_{FR} \\ \dot{\theta}_{BL} \\ \dot{\theta}_{BR} \end{bmatrix}$$

with:

$\dot{\theta}_{FL}$, etc.	four individual wheel speeds in revolutions per second,
r	wheel radius,
d	distance between left and right wheel pairs,
e	distance between front and back wheel pairs,
v_x	robot velocity in forward direction,
v_y	robot velocity in sideways direction,
ω	robot rotational velocity.

Robotics Hardware Platforms – Wheeled Robots

Drive Kinematics – Omni-Drive Kinematics

- The **inverse kinematics** is a matrix formula that specifies the required individual wheel speeds for given desired linear and angular velocities (v_x, v_y, ω) and can be derived by inverting the matrix of the forward kinematics.

$$\begin{bmatrix} \dot{\theta}_{FL} \\ \dot{\theta}_{FR} \\ \dot{\theta}_{BL} \\ \dot{\theta}_{BR} \end{bmatrix} = \frac{1}{2\pi r} \begin{bmatrix} 1 & -1 & -(d+e)/2 \\ 1 & +1 & +(d+e)/2 \\ 1 & +1 & -(d+e)/2 \\ 1 & -1 & +(d+e)/2 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Controllers for Robotic Systems

- A controller for robotic systems is **a device** or **software** that manages and directs the robot's operations, ensuring coordinated movement, sensor integration, and task execution.

- It acts as the **central processing unit** that interprets inputs, executes control algorithms, and outputs commands to actuators and other peripheral devices.

Controllers for Robotic Systems

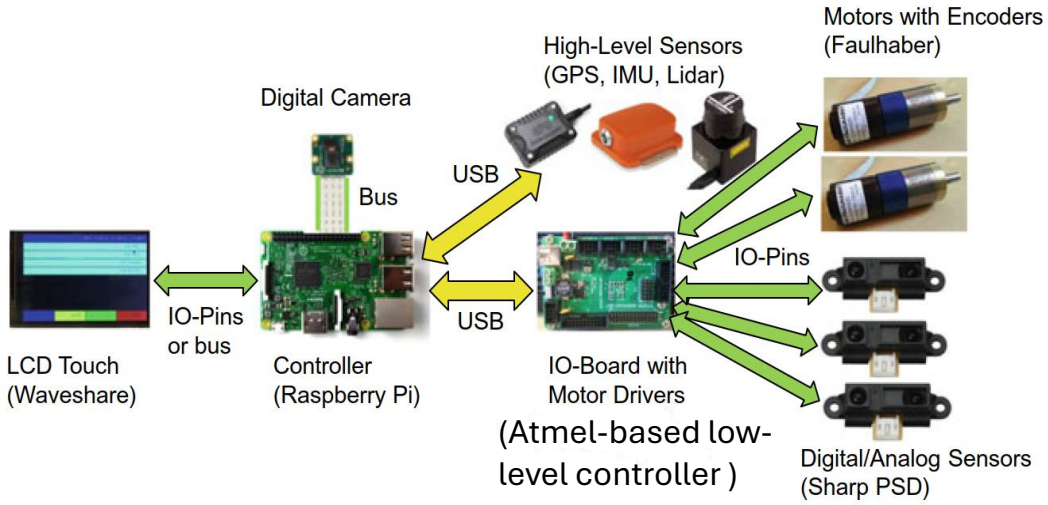
Types of Controllers for Robotic Systems:

- **Robotic Controllers (Dedicated Hardware):**
  - **Industrial Robot Controllers:** Specialized units designed for precise control of industrial robotic arms and automation systems. There are a few common examples, including FANUC Robot Controllers, ABB Robot Controllers, Universal Robots (UR) Controllers, etc.
  - **Embedded Controllers:** Microcontrollers or embedded systems integrated within the robot for real-time control. There are a few common examples, including Arduino, ESP32, Raspberry Pi, etc.
  - **PLC (Programmable Logic Controller):** Used mainly in industrial automation for controlling machinery and processes.
- **Software-Based Controllers:**
  - **ROS (Robot Operating System):** An open-source robotics middleware providing libraries and tools for controlling robots, sensor integration, and navigation.
  - **Custom Control Software:** Developed using programming languages like C++, Python, or MATLAB, running on embedded systems or PCs.
- **Hybrid Controllers:**
  - Combining hardware (microcontrollers or embedded systems) with software (middleware or control algorithms) for complex robotic applications.

Controllers for Robotic Systems

Microcontrollers

- A microcontroller for robotic applications is a compact integrated circuit that serves as the brain of a robot, controlling its movements, sensors, and other peripherals.
- It processes inputs from sensors such as GPS, IMU, Lidar, etc., executes control algorithms, and sends commands to actuators such as motors, servos, and other device

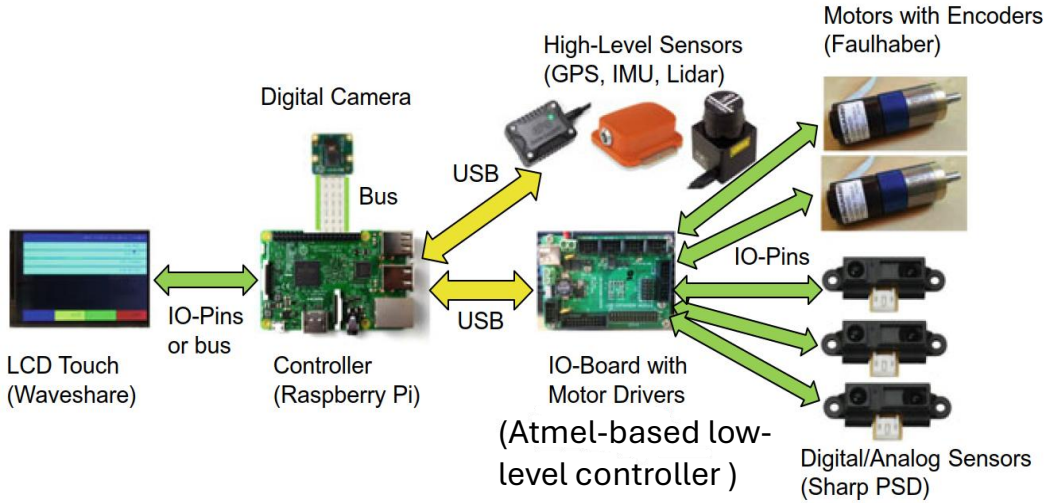


A robotic system design with Raspberry Pi and Atmel

Controllers for Robotic Systems

Key Features of Microcontrollers

- **Multiple I/O Pins:** For connecting sensors, motors, LEDs, and other peripherals. The Raspberry links directly to the camera and the optional touch-screen LCD, as well as to any high-level sensors that require a USB or Ethernet LAN interface, e.g., satellite-based global positioning system (GPS), inertial measurement unit, internal sensor (IMU) or light detection and ranging, laser-based (Lidar).
- **PWM (Pulse Width Modulation):** To control motor speed and position.
- **Analog-to-Digital Converters (ADC):** For reading analog sensor signals like distance, light, or temperature.
- **Timers and Counters:** For precise timing and control.
- **Communication Interfaces:** Such as UART, I2C, SPI, USB for communication with other modules or computers.
- **Real-time operation:** Ability to handle time-sensitive tasks.

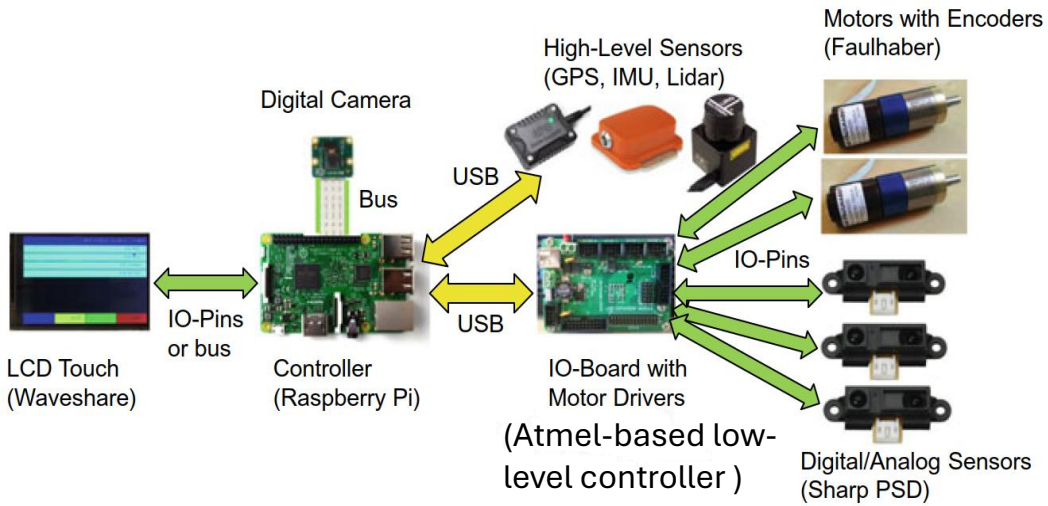


A robotic system design with Raspberry Pi and Atmel

Controllers for Robotic Systems

Common Microcontrollers Used in Robotics

- **Arduino (based on AVR microcontrollers, e.g., ATmega328):**
  - Easy to program with Arduino IDE.
  - Large community and extensive libraries.
  - Suitable for beginners and educational projects.
- **ESP32:**
  - Wi-Fi and Bluetooth capabilities.
  - More processing power and memory than typical Arduino boards.
  - Good for wireless robotics.
- **STM32 (ARM Cortex-M series):**
  - High performance and low power consumption.
  - Suitable for more complex or demanding robotic applications.
- **Raspberry Pi 5:**
  - A faster ARM Cortex-A77 or newer processor, enabling complex computations, real-time processing, and AI tasks.
  - More RAM options (Up to 8GB or more).
  - Improved connectivity (Support for Wi-Fi 6, Bluetooth 5.0, multiple USB ports, and Gigabit Ethernet).
  - Advanced I/O capabilities (Increased GPIO pins, high-speed interfaces like PCIe (via the M.2 slot), and support for high-resolution cameras and sensors.
  - Video and camera support.
  - Supports MicroPython and C/C++.
  - Good for embedded robotics projects.

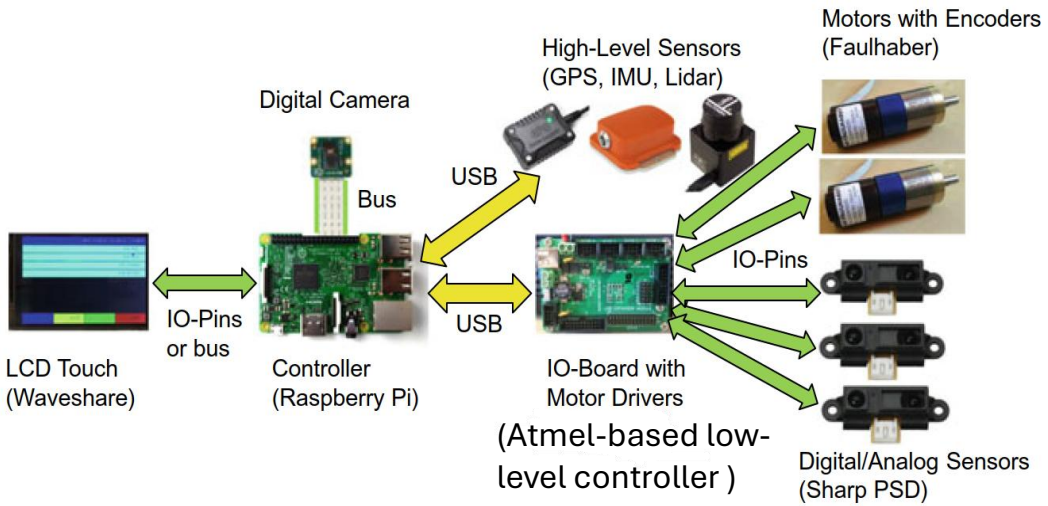


A robotic system design with Raspberry Pi and Atmel

Controllers for Robotic Systems

An Example - A Robotic System Design with Raspberry Pi & Atmel-based Low-Level Controller

- The system design with a **Raspberry Pi** and an **Atmel-based low-level controller**, specialized for input/output, which provides the link between sensors and actuators to the high-level controller.
- The Raspberry links directly to the camera and the optional touch-screen LCD, as well as to any high-level sensors that require a USB or Ethernet LAN interface, e.g., satellite based global positioning system (GPS), inertial measurement unit, internal sensor (IMU) or light detection and ranging, laser-based (Lidar).
- All motors with encoders, servos and all simple sensors are connected to the I/O board, which communicates to the Raspberry Pi via a bidirectional USB link.
- The advantage of this approach is that the high-level controller is freed from time-consuming low-level control.
- Also, since the Raspberry Pi usually does not run a real-time operating system, it cannot always maintain the correct time interval, which is essential for motor and servo control. Instead, this will be done by the low-level controller, which does not have an operating system itself, but can make use of various built-in hardware timers to guarantee real-time signals.



A robotic system design with Raspberry Pi and Atmel

Controllers for Robotic Systems

An Example - A Robotic System Design with Raspberry Pi & Atmel-based Low-Level Controller

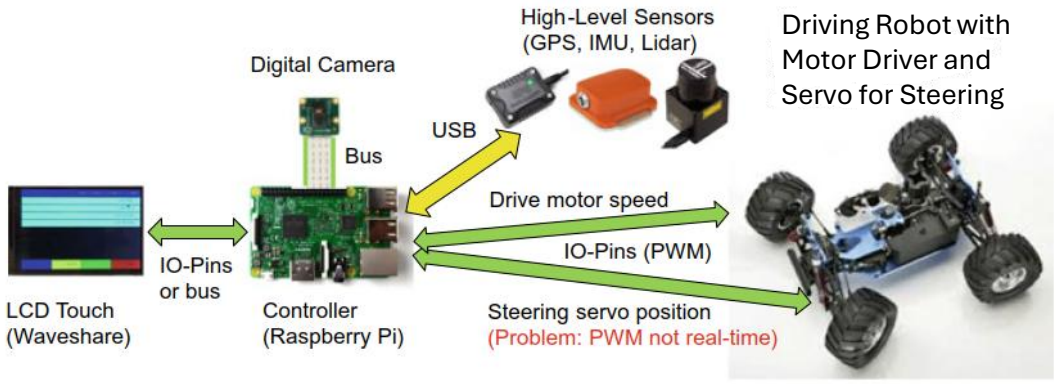
The alternative to using Atmel-based I/O board, or any secondary processor board for that matter, is to use separate extension boards for each of the required functionalities. These are:

- **Voltage controller board** required, e.g., for converting 7.2 V battery supply to 5 V, 3 A for Raspberry Pi and high-level sensors.
- **Dual (or quad) motor driver board** required to drive DC motors.
- **Encoder interface board** required to properly control motor speeds. Without this, a differential drive robot cannot drive straight.
- **Servo controller board** required to generate PWM output signals for model servos, such as the steering in a remote-controlled car. However, these signals can also be generated done in software if a lower PWM accuracy is acceptable.
- **Analog sensor interface board** required to interface to analog sensors, such as distance sensors.
- **Digital sensor interface board** required for sensors that return their data as a timed sequence of digital values. Simple binary sensors (on/off, 1/0) can be directly connected to an I/O pin of the Raspberry Pi.

Controllers for Robotic Systems

An Example - A Simple Robotic System Design with Raspberry Pi Only

- The system design with a **Raspberry Pi** only and without any dedicated I/O board.
- However, motors are linked through an external motor driver to an output pin of the Raspberry Pi, while low-level sensors cannot be connected at all.
- If a driving robot with built-in motor driver and servo for steering is being used instead of a robot driving hardware, then Raspberry-I/O lines can be used directly to drive the robot.
- As the Raspberry Pi does not have any hardware timers, care has to be taken that the software pulse generation for the steering command does not result in a jitter of the servo.

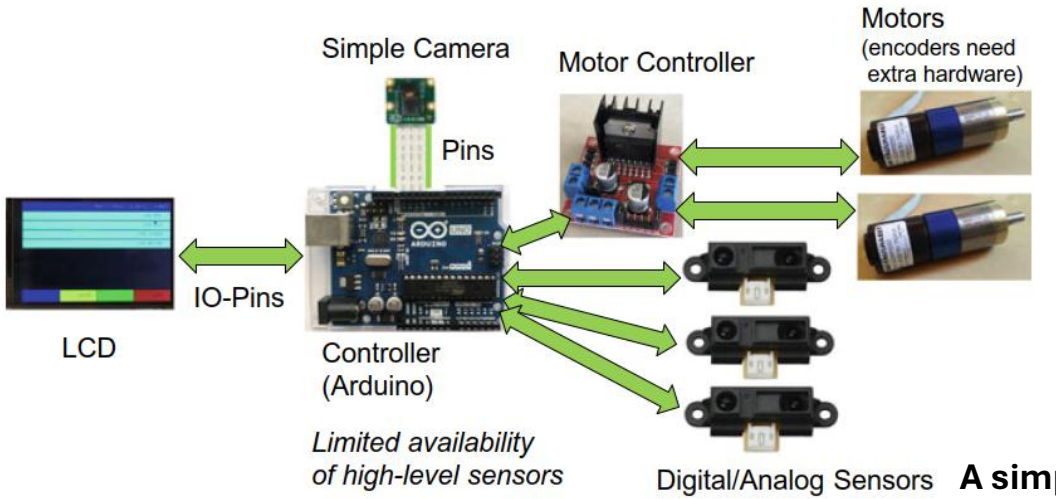


A simple robotic system design with Raspberry Pi only

Controllers for Robotic Systems

An Example - A Simple Robotic System Design with Arduino Only

- The system design with Arduino only is equally the previous (Raspberry Pi only), but also a lot less powerful for robot control.
- Although LCDs, camera modules and some high-level sensors are available, the computing power of the Arduino is not really capable to make the best use of them.
- Connecting motors require an external motor driver board and encoders would need another specialized hardware addition.
- Only low-level sensors can be directly connected to the Arduino pins.



A simple robotic system design with Arduino only

Controllers for Robotic Systems

Key Considerations in Selecting a Controller

- Task complexity and precision requirements.
- Robot type (e.g., manipulator, mobile robot, drone).
- Environmental conditions and uncertainties.
- Real-time processing needs.
- Connectivity requirements.
- Computational resources available.

In conclusion, the **optimal control strategy** often involves combining multiple controllers or designing hierarchical systems to achieve robust, efficient, and adaptable robotic behavior.

For beginners, Arduino-based systems are often the best starting point, while more advanced projects may benefit from STM32, Raspberry Pi, or similar microcontrollers.

Software for Robotic Systems

To encompass a wide range of tools and applications designed to enable robots to perceive, plan, control, and interact with their environment. These software components can be categorized into several key areas:

- **Operating Systems and Middleware:**
  - **ROS (Robot Operating System):** A flexible framework providing libraries and tools for building robot applications, including message-passing, hardware abstraction, and device drivers.
  - **ROS 2:** The successor to ROS, supporting **real-time capabilities**, **improved security**, and **multi-platform support**.
- **Simulation Software:**
  - **Gazebo**: A powerful physics-based simulation environment for testing robot algorithms in a virtual world.
  - **Unity Robotics**: Offers a toolkit for high-fidelity physics and realistic environments, popular for synthetic data generation.
- **Control and Planning Software:**
  - **MoveIt!:** A motion planning framework for robotic arms, integrated with ROS.

Software for Robotic Systems (Continued…)

To encompass a wide range of tools and applications designed to enable robots to perceive, plan, control, and interact with their environment. These software components can be categorized into several key areas:

- **Perception and Sensor Processing:**
  - **OpenCV:** Computer vision library for image processing, object detection, and recognition.
  - **PCL (Point Cloud Library):** Processing 3D point cloud data from sensors like LiDAR or depth cameras.
- **Artificial Intelligence and Machine Learning:**
  - Frameworks like TensorFlow, PyTorch, and scikit-learn integrated for perception, decision-making, and autonomous behavior.
- **Embedded and Real-Time Software:**
  - Real-time operating systems (RTOS) such as FreeRTOS or RTLinux for time-critical applications.
- **Development Environments and Tools:**
  - IDEs like Visual Studio Code or Eclipse.
  - Visualization tools such as RViz (for ROS) to visualize robot sensors, states, and planned paths.

Software Architecture for Robotic Systems

- To refer to the structured design and organization of software components and their interactions to achieve reliable, flexible, and maintainable robot behavior. A well-designed architecture enables efficient integration of perception, planning, control, and communication modules.
- Common Architectural Pattern in Robotics:
  - Layered Architecture:
    - Description: Divides software into hierarchical layers such as hardware abstraction, middleware, perception, planning, and control.
    - Benefits: Modularity, easier debugging, and separation of concerns.
  - **Publish-Subscribe** Architecture:
    - Description: Uses message-passing systems (like ROS) where components publish and subscribe to data topics.
    - Benefits: Decoupling of components, scalable communication.
    - Example: Sensor data published to topics, navigation module subscribes to relevant data.
  - Behavior-Based Architecture:
    - Description: Organizes software around behaviors or skills that can operate concurrently and reactively.
    - Benefits: Robustness, adaptability, suitable for autonomous mobile robots.
    - Example: Behaviors like obstacle avoidance, goal seeking, and exploration.
  - Hybrid Architectures:
    - Combines multiple patterns (e.g., layered and behavior-based) to leverage their advantages.

Software Architecture for Robotic Systems

- Key Considerations in Designing Software Architecture:
  - Modularity: Facilitates testing and maintenance.
  - Real-Time Requirements: Ensures timely responses for control tasks.
  - Scalability: Supports adding new functionalities.
  - Fault Tolerance: Handles failures gracefully.
  - Hardware Abstraction: Provides uniform interfaces across different sensors and actuators.
  - Communication: Efficient data exchange mechanisms.
- Frameworks Supporting Architecture Design:
  - ROS / ROS 2: Implements publish-subscribe patterns, supports modular components.
  - OOP (Object-Oriented Programming): Facilitates modular, reusable code.
- A practical and effective robotic software architecture balances modularity, real-time performance, scalability, and robustness, tailored to the specific robot's application domain

Software Framework for Robotic Systems

- To provide a structured environment to develop, deploy, and manage robotic applications efficiently. Some of the most widely used frameworks include:
  - **ROS (Robot Operating System):**
    - An open-source, flexible framework that offers libraries and tools to help build robot applications.
    - Supports hardware abstraction, device drivers, message-passing, package management, and more.
    - Widely adopted in academia and industry, with a large community and extensive packages.
  - **ROS 2:**
    - The next-generation version of ROS, designed for real-time systems, improved security, and better support for embedded systems.
    - Supports multiple middleware implementations and enhanced communication mechanisms.
  - **Microsoft Robotics Developer Studio (MRDS):**
    - Provides a visual programming environment and supports multiple programming languages.
    - Focuses on simulation and control of robotic systems.
- **Choosing a framework depends on your specific needs**, such as hardware compatibility, real-time requirements, simulation needs, and programming language preferences. ROS (particularly ROS 2) remains the most popular choice due to its extensive ecosystem and community support.

Software Framework for Robotic Systems

Robot Operating System (ROS)

- To a widely used open-source software framework designed to facilitate the development of robotic systems.
- To provide a collection of tools, libraries, and conventions that simplify the process of building, deploying, and managing robot applications.

Software Framework for Robotic Systems

Robot Operating System (ROS)

- The key aspects of ROS as a software framework for robotics:
    1. **Modular Architecture**: **Component-Based**: ROS organizes software into nodes (processes) that perform specific functions. **Communication**: Nodes communicate via message passing through topics, services, and actions. Advantages: Promotes reusability, scalability, and easier debugging.
    2. **Communication Mechanisms**: **Publish-Subscribe Model**: Nodes publish data to topics; other nodes subscribe to relevant topics. **Services**: Synchronous request-response communication for specific operations. **Actions**: Long-running tasks with feedback, suitable for navigation or manipulation tasks.
    3. **Hardware Abstraction**: Provides standardized interfaces to interact with sensors, actuators, and hardware drivers. Enables hardware independence, making it easier to switch or upgrade components.
    4. **Tooling and Ecosystem**: **RViz**: Visualization tool for robot states, sensor data, and planned trajectories. **Gazebo**: Simulation environment integrated with ROS for testing algorithms virtually. **roslaunch**: Tool to start multiple nodes and set configurations easily. **rqt**: GUI tools for visualization and debugging. **rosbag**: Record and playback of sensor data and messages.

Software Framework for Robotic Systems

Robot Operating System (ROS)

- The key aspects of ROS as a software framework for robotics:
  5. **Package Management**: ROS uses **packages** to organize code, which can include nodes, libraries, configuration files, and launch scripts. Extensive ecosystem: Thousands of ready-to-use packages for perception, navigation, manipulation, etc.
  6. **Supported Languages**: Primarily C++ and Python, with bindings available for other languages.
  7. **Community and Support**: Large, active community providing tutorials, documentation, and third-party packages. Widely adopted in academia and industry for research and commercial applications.
  8. **Transition to ROS 2**: ROS 2 introduces improvements such as real-time support, better security, multi-robot support, and improved middleware (DDS). It maintains backward compatibility and expands capabilities for complex robotic systems.

Software Framework for Robotic Systems

ROS2 Core Concepts

- **Nodes**: a ROS 2 node can be any kind of computer program that contains:
  - Instructions to print logs in the terminal
  - Graphical windows (2D, can also be 3D)
  - Hardware drivers and more
- **Topics**:
  - Nodes communicate with each other using ROS communication features.
  - There are three types of communication: topics, services, and actions.
  - One node can publish or subscribe to a topic. When publishing, the node sends some data. When subscribing, it receives the data.
  - A topic is defined by a name and a data type
- **Services**:
  - It is a way of client/server communications in ROS 2.
  - The working principle of a service is that one node that contains a server, then any other node (client) can send a request to that server. The server processes the request and returns a response to the client node.
  - A service is defined by a name and an interface (request, response)
- **Actions:**
  - To be basically the same thing as a service (client/server communication), but designed for longer tasks, and when you might want to also get some feedback during the execution, be able to cancel the execution, and so on.
  - An action is defined by two things: a name and an interface (goal, result, feedback).

Software Framework for Robotic Systems

ROS2 Core Concepts (continued…)

- **Parameters:**
  - Parameters are settings that can be provided at runtime (which means when we run the node). They allow us to easily configure the different nodes that we start, and thus, they make ROS 2 applications more dynamic.
  - A parameter exists within a node. You can find all parameters for a node and get the value for each one. When starting the node, you can give a custom value for the parameters you want to modify.
- **Launch files**:
  - allow you to start several nodes and parameters from just one file, which means that you can start your entire application with just one command line.

Communication Protocols for Robotic Systems

- Robotic systems use communication protocols like ROS, Ethernet/IP, Modbus, MQTT, and TCP/IP to enable interaction between components.
- The choice depends on the application, with industrial settings favoring robust protocols like Ethernet/IP for real-time control, while IoT-enabled robots use lightweight options like MQTT for remote monitoring.
- Protocols are also categorized by their communication method: wired (e.g., Ethernet) or wireless (e.g., Wi-Fi, Bluetooth), with emerging options like 5G and LoRa for long-distance communication.

Design and Manufacturing for Robotic Systems (continued…)

To a multidisciplinary approach that integrates mechanical engineering, electrical engineering, computer science, and systems engineering. An overview of key considerations and steps are as follows:

1. Requirements Analysis

- Define the robot's purpose (e.g., industrial automation, service, medical).
- Specify operational environment (indoor, outdoor, hazardous areas).
- Determine performance criteria (speed, load capacity, precision).

2. Conceptual Design

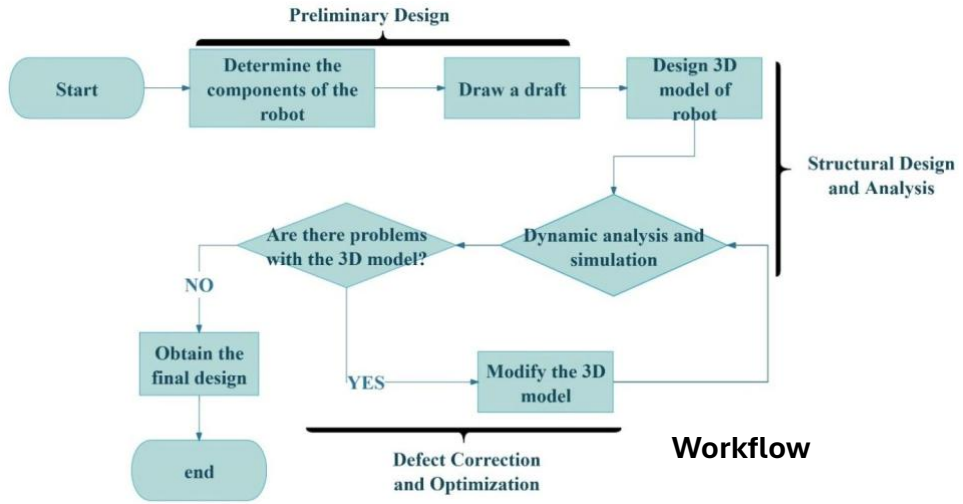
- Decide on the robot type (mobile, manipulator, humanoid, drone).
- Sketch initial concepts and select core components:
  - Actuators (motors, hydraulics, pneumatics)
  - Sensors (vision, proximity, force)
  - Control systems (microcontrollers, PLCs)

3. Mechanical Design

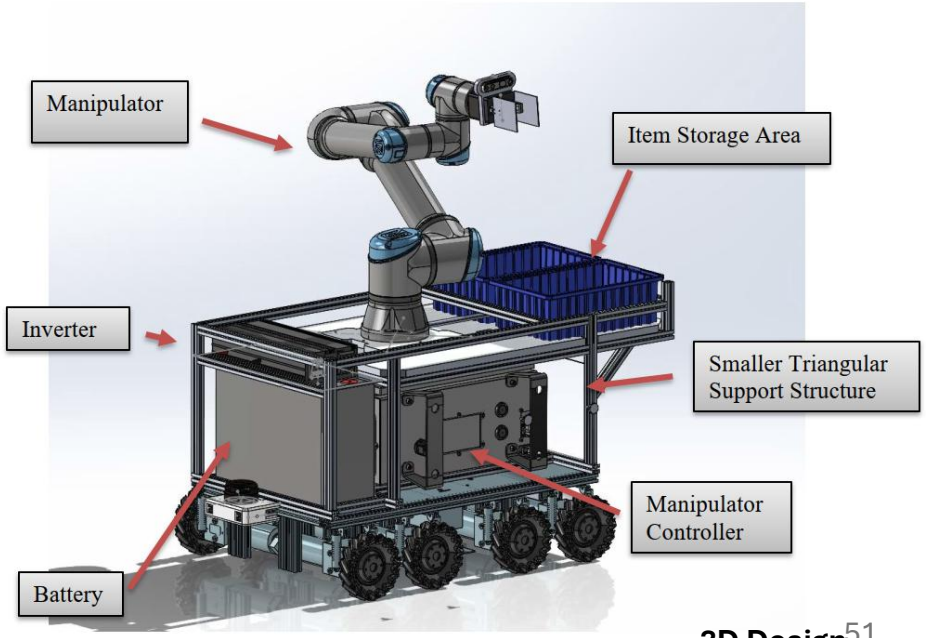
- Create detailed CAD models of the robot's structure.
- Select materials balancing strength, weight, and cost.
- Design for manufacturability and maintenance.

4. Electrical and Control System Design

- Develop circuitry for power distribution and signal processing.
- Design control algorithms and software (e.g., PID, machine learning).
- Integrate sensors and actuators with controllers.



Workflow



3D Design

Design and Manufacturing for Robotic Systems

5. Prototyping

- Build initial prototypes for testing.
- Iterate designs based on performance and testing feedback.

6. Manufacturing Processes

- Choose manufacturing techniques:
  - CNC machining for precision parts.
  - Injection molding for plastic components.
  - 3D printing for rapid prototyping.
- Implement quality control measures.
- Assemble components into functional systems.

7. Testing and Validation

- Conduct functional testing in controlled environments.
- Evaluate performance against specifications.
- Perform durability and reliability testing.

8. Deployment and Maintenance

- Install the robotic system in its intended environment.
- Develop maintenance protocols and spare parts inventory.
- Collect operational data for continuous improvement.



Physical intelligent robot