PRO HOMINUM
REVISTA DE CIENCIAS SOCIALES Y HUMANAS

Artículo original

Fundamentals of applied robotics: an integrated guide on control, programming and training

Fundamentos de robótica aplicada: Una guía integrada sobre control, programación y entrenamiento

https://doi.org/10.47606/ACVEN/PH0383

Yuri Merizalde Zamora[1*]
https://orcid.org/0000-0002-9845-8073
yuri.merizaldez@ug.edu.ec

Miroslav Alulema Cuesta[1]
https://orcid.org//0009-0001-6527-7567
miroslav.gonzaloa@ug.edu.ec

Tyron Alcivar Reyna[2]
https://orcid.org/0000-0003-1841-2161
tyrone.alcivarr@ug.edu.ec

Joel Barba Salazar[1]
https://orcid.org/0000-0002-0816-7978
joel.barbas@ug.edu.ec

Recibido: 15/07/2025 **Aceptado:** 29/09/2025

ABSTRACT

Applied robotics has become a transversal tool in research, manufacturing, and automation. Yet, its adoption by non-engineering professionals is limited by technical complexity and rapid paradigm shifts. This review provides an accessible framework that enables non-specialists to understand, evaluate, and apply robotic systems. The discussion is organized into three pillars: taxonomy of platforms (manipulators, mobile robots, humanoids), control methods (kinematics, dynamics, trajectory planning), and programming and training paradigms (from ROS architectures to imitation and reinforcement learning). Emphasis is placed on integrating classical control, structured programming, and machine learning into hybrid architectures that achieve robust and adaptable behavior. The article thus serves as both educational and strategic guidance, helping readers to select, integrate, and operate robots for real-world problem solving and innovation across multiple domains.

Keywords: Learning, Robot Types, Control Methods, Programming, Training, Robotic Manipulation, Autonomous Navigation.

1. Universidad de Guayaquil
2. Universidad de Guayaquil / ESPOL
- Autor de correspondencia: yuri.merizaldez@ug.edu.ec

RESUMEN

La robótica aplicada se ha convertido en una herramienta transversal en la investigación, la manufactura y la automatización. Sin embargo, su adopción por parte de profesionales no especializados en ingeniería se ve limitada por la complejidad técnica y la rápida evolución de sus paradigmas. Este artículo de revisión ofrece un marco conceptual accesible que permite a los no especialistas comprender, evaluar y aplicar sistemas robóticos. La discusión se organiza en tres pilares: la taxonomía de plataformas (manipuladores, robots móviles y humanoides), los métodos de control (cinemática, dinámica y planificación de trayectorias) y los paradigmas de programación y entrenamiento (desde arquitecturas como ROS hasta técnicas de imitación y aprendizaje por refuerzo). Se enfatiza la integración del control clásico, la programación estructurada y el aprendizaje automático en arquitecturas híbridas que logran comportamientos robustos y adaptables. De este modo, el artículo funciona como una guía educativa y estratégica que orienta en la selección, integración y operación de robots para resolver problemas reales e impulsar la innovación en múltiples dominios.

Palabras claves: Aprendizaje, Tipos de robots, Métodos de control, Programación, Entrenamiento, Manipulación robótica, Navegación autónoma.

INTRODUCTION

Robotics has evolved from 20th-century industrial automation into a multidisciplinary field driven by artificial intelligence and advanced perception. Early manipulators such as Unimate (1961) were confined to structured, repetitive tasks with little adaptability. Today, robots not only assemble cars but also navigate dynamic warehouses, assist in hospitals, and learn new skills, marking a shift from rigid machines to adaptive systems (Ceccarelli, 2001). As one of the pillars of the Fourth Industrial Revolution, robotics underpins advanced manufacturing, autonomous logistics, scientific research, and exploration of extreme environments. In laboratories, high-precision manipulators automate tasks in genomics or drug discovery, boosting throughput and reproducibility. These applications illustrate how robotics accelerates innovation across diverse domains. Modern robotics is no longer restricted to mechanical engineering and classical control. It now integrates kinematics, dynamics, perception, and machine learning. For scientists, engineers from other fields, or students, this interdisciplinarity often represents a significant entry barrier. The gap between available robotic technologies and the knowledge needed to apply them effectively remains wide, discouraging non-specialists from engaging with the field.

This review addresses that gap by providing a systematic and accessible framework. Its purpose is not to deliver an exhaustive treatise but to outline the core principles required to understand and apply robotic systems. By clarifying

terminology and concepts, it enables informed decisions on selecting platforms, control strategies, and programming paradigms.

Specifically, the article seeks to answer four guiding questions:

- What are the fundamental multidisciplinary knowledge pillars in robotics?
- What taxonomy of robotic platforms should professionals know, and what are their operational principles and control challenges?
- What mathematical and control frameworks, from kinematics and dynamics to feedback loops—are essential for commanding and stabilizing robots?
- What programming and training paradigms, from explicit frameworks such as ROS to imitation and reinforcement learning, support the development of complex behaviors, and how does simulation bridge research and application?

This framework is intended as both an educational introduction and a strategic guide. It demystifies robotics for readers without formal specialization while highlighting the integration of classical control, structured programming, and learning-based approaches into robust hybrid architectures. Ultimately, it equips non-specialists to evaluate, integrate, and operate robots effectively, accelerating innovation in research, industry, and society.

DEVELOPMENT

1. Taxonomy of Applied Robotic Platforms

In robotics, a platform refers to the physical and functional configuration of a robot - its “body” and fundamental capabilities to interact with the environment. A platform combines sensors, actuators, and a mechanical structure, and is selected according to the task rather than technological novelty. Examples range from fixed arms in factories to mobile robots in warehouses, aerial drones, or humanoids designed for human environments. To guide this choice, this work proposes a functional taxonomy centered on three dominant categories: robotic manipulators, mobile robots, and humanoid robots. Although not exhaustive, this classification covers the most relevant architectures in industrial, research, and laboratory contexts, particularly for non-specialized professionals (*Siciliano & Khatib, 2016*).

Each type responds to distinct needs: manipulators provide precision in structured spaces, mobile robots autonomy in expansive environments, and humanoids versatility in human-centered settings. This section examines their architecture, main applications, control challenges, and leading manufacturers, offering a comparative framework that prepares the reader for subsequent discussion on control, programming, and training.

1.1 Robotic Manipulators

Robotic arms are among the most widespread and versatile robotic platforms. Since their industrial debut, they have become essential for tasks requiring

precision, repeatability, and controlled force (Siciliano & Khatib, 2016). Mechanically, they consist of a kinematic chain of links connected by rotary or linear joints, anchored to a fixed base and ending in an end-effector (e.g., gripper, tool, or sensor). Their mobility is defined by degrees of freedom (DOF), typically equal to the number of joints. A 6-DOF arm can achieve any position and orientation in its workspace, enabling complex manipulation (Lynch & Park, 2023).

Common configurations include Cartesian, cylindrical, SCARA, articulated, and Delta architectures, each optimized for specific applications in terms of precision, speed, rigidity, or workspace (Lynch & Park, 2023). Subsection 2.1.1 will discuss these configurations in greater detail.

Industrially, robotic arms perform pick-and-place, assembly, welding, painting, and palletizing. In labs, they automate sample handling, reagent preparation, and screening, reducing human error and freeing researchers for higher-level tasks (Siciliano & Khatib, 2016). A key evolution is the collaborative robot (cobot): designed for safe human proximity, equipped with force sensing and ergonomic interfaces, cobots have democratized automation, even in low-resource settings.

Despite their utility, precise control remains challenging, chiefly due to the inverse kinematics problem: calculating joint angles to achieve a desired end-effector pose. Solving it requires mathematical modeling, programming, and real-time control (Licardo et al., 2024). Table 1 summarizes the main robot types discussed here. Subsections 2.1.1 and 2.1.2 will analyze arm architectures and cobots in detail, providing foundations for selection, programming, and application.

Table 1.
Comparison of robotic arm architecture

Type	DOF	Speed	Accuracy	Payload	Applications
Cartesian	3–5	Medium	Very High	High	Heavy manipulation, 3D printing
SCARA	4	High	High	Medium	Assembly, PCB
Articulated	6+	Medium	High	High	Welding, painting, manipulation
Delta	3–4	Very High	High	Low	High-speed pick-and-place

1.1.1 Architecture and Classification by Configuration

The architecture of a robotic manipulator determines its workspace, rigidity, and suitability for specific tasks. Four dominant configurations are used in industry and labs: Cartesian, SCARA, articulated (anthropomorphic), and Delta, each with distinct strengths. Cartesian robots (gantry type) move along three linear axes (X, Y, Z), offering high rigidity and precision over large workspaces. Ideal for heavy-load tasks, large-scale assembly, or 3D printing (Siciliano & Khatib, 2016). Leading suppliers include Parker Hannifin, Bosch Rexroth, and Yaskawa. SCARA robots (Selective Compliance Assembly Robot Arm) feature two parallel rotary joints, enabling fast, precise horizontal movements with vertical rigidity. Optimized for high-speed insertion, PCB assembly, and packaging. Key suppliers: Epson, Omron, Yaskawa. Articulated robots (6-DOF, anthropomorphic) mimic human arm motion, offering full spatial flexibility. Widely used in welding, painting, and machine

tending. Market leaders: FANUC (R-2000iC), KUKA (KR QUANTEC), ABB (IRB), Yaskawa (Motoman). Finally, Delta robots use a parallel structure with three arms from an overhead base, enabling ultra-fast, lightweight pick-and-place. Common in food, pharma, and electronics for speed and hygiene. Leading models: FANUC Genkotsu, ABB FlexPicker (*Craig*, 2006). Companies such as FANUC (Genkotsu) and ABB (FlexPicker) are among the prominent suppliers.

1.1.2 The Collaborative Robot Paradigm

A major evolution in manipulation robotics is the collaborative robot (cobot), designed to work safely alongside humans, unlike traditional robots that require isolation. Safety is ensured through integrated force/torque sensors, speed/power limits, and ergonomic, rounded designs (*ISO, 2025*). Cobots enable human-robot synergy: humans contribute dexterity and decision-making; robots provide precision, strength, and repeatability. Typical applications include delicate assembly, machine tending, and inspection tasks. Platforms from Universal Robots, FANUC, and ABB have made cobots accessible, especially for SMEs. Table 2 summarizes key differences between industrial robots and cobots.

Table 2.
Main characteristics and differences between industrial robots and cobots.

Feature	Industrial Robot	Cobot
Safety	Requires isolated cell	Shared operation
Programming	Specialized languages	Manual guidance, GUI
Cost	High	Medium-low
Flexibility	Low	High
Ideal Application	Mass production	Customized tasks, laboratories
Cartesian	3–5	Medium
SCARA	4	High
Articulated	6+	Medium
Delta	3–4	Very High

1.2. Mobile Robots

Unlike fixed-base manipulators, mobile robots, or autonomous ground vehicles (AGVs/AMRs), operate in dynamic, unstructured environments, greatly expanding automation's reach (Thrun et al., 2006). Their value lies not just in movement, but in autonomous, safe, purpose-driven navigation. This requires solving two core problems: "*Where am I*?" (localization) and "*What is around me*?" (mapping). This section examines the algorithmic foundations, physical architectures, and transformative applications of mobile robotics (Durrant et al., 2006).

1.2.1 Physical Architectures: Locomotion and Perception Platforms

The algorithmic capability of SLAM is materialized in physical systems whose locomotion and perception architectures must align with operational demands. The choice of movement platform and sensor suite directly determines a robot's

applicability and robustness in real-world scenarios. Regarding locomotion, wheeled systems dominate for indoor or prepared surfaces due to their efficiency and simplicity. Within this group, Autonomous Mobile Robots (AMRs) -unlike traditional AGVs that follow fixed paths- use SLAM for flexible, adaptive navigation in dynamic environments, making them ideal for logistics in warehouses and factories. Leading providers include Mobile Industrial Robots (MiR) and OTTO Motors (Murphy, 2019).

Tracked systems offer superior traction on uneven or low-adhesion terrain, making them indispensable in defense, construction, and planetary exploration. NASA's Curiosity rover, designed for Martian dust, rocks, and slopes, exemplifies this architecture. Legged systems, particularly quadrupeds, are engineered for complex, unstructured environments like stairs or rubble. Though dynamic balance control remains a major challenge, their obstacle negotiation capability is transformative. Robots such as Boston Dynamics' Spot and ANYbotics' ANYmal are now setting standards for industrial inspection in petrochemical plants or mines (Rankin et al., 2021).

Regarding perception, the sensor suite underpins pose estimation and mapping. Current practice relies on sensor fusion to combine complementary modalities. LIDAR provides high-precision, high-frequency distance measurements, generating geometric point clouds robust to lighting changes — making it the classic choice for high-fidelity indoor and outdoor mapping. Cameras, both visual and depth (RGB-D), offer low-cost, semantically rich data and form the basis of Visual SLAM; models like the Intel RealSense series have democratized 3D perception. The Inertial Measurement Unit (IMU), functioning as the robot's inner ear, delivers high-frequency acceleration and angular velocity data. Its fusion -especially in tightly coupled schemes with LIDAR or cameras- is critical for robust, continuous pose estimation under low visibility or high dynamics (Thrun et al., 2006) and (Newcombe et al., 2011).

In specific applications, additional sensors like GPS, contact sensors, or microphones may be integrated. Sensor fusion, combining data from multiple sources, remains key to ensuring robust perception in dynamic or changing environme (Kelly & Sukhatme, 2011). The physical architecture of a mobile robot is not a set of isolated components but an integrated system where locomotion and perception cooperate to enable autonomy. Configuration must be guided by environment, task, and requirements for precision, speed, and safety-allowing operation from industrial warehouses to university laboratories.

1.2.2 Application Fields

Mobile robots have moved beyond research to become strategic tools across sectors. Solving SLAM has unlocked vast commercial and scientific potential, redefining industries through autonomous navigation, robust perception, and fleet coordination. In logistics, the sector with the greatest commercial impact, AGVs long served as automation backbones, following fixed routes via magnetic or visual guides. Though efficient, their rigidity limited adaptability. This has been superseded by AMRs, which use SLAM to navigate dynamically, avoiding

obstacles and replanning routes in real time. In modern distribution centers, fleets of AMRs implement “goods-to-person” strategies -popularized by Amazon Robotics- bringing shelves to workstations instead of requiring operators to walk vast distances. This not only boosts efficiency and reduces physical strain but also enables real-time inventory tracking and integration with ERP systems for more agile supply chains (Javaid et al., 2022).

In agriculture, mobile robots are transforming crop management. Autonomous tractors and drones with precision sensors monitor soil moisture, plant health, and pests, enabling targeted application of inputs to reduce environmental impact and increase yields. Specialized robots are also emerging for planting, harvesting, and pruning high-value crops, adapting to unstructured, variable terrain -a core requirement of precision agriculture (Bechar & Vigneault, 2016). In mining and construction, mobile robots enhance safety and productivity. Autonomous haulers operate in open-pit mines under hazardous or remote conditions, running continuously regardless of shifts or weather. In construction, robots inspect progress, verify blueprint compliance, and monitor material quality -increasingly integrated with BIM for smarter project management.

In transportation, “last-mile” delivery is being realized through sidewalk and street-navigating robots from companies like Starship Technologies and Nuro. Operating at low speeds with multi-sensor safety systems, they deliver packages, food, and medicine while interacting with urban environments, stopping at lights, avoiding pedestrians. In service sectors, robots assist with telepresence, disinfection, and logistics: in hospitals, they transport medicines, perform UV disinfection, and enable remote consultations; in public spaces, they patrol, monitor conditions, and guide visitors (Srinivas et al., 2022). In academia, mobile robots serve as key didactic and research platforms. Systems like TurtleBot, Jackal, or MiR100 are widely adopted for teaching robotics, AI, and autonomy, valued for their modularity and accessibility in student projects and competitions. In research labs, they are increasingly paired with robotic arms to create mobile manipulation systems capable of transporting and handling samples in high-throughput environments (Zhang et al., 2024).

1.3. Humanoid Robots

Humanoid robots embody one of robotics’ most ambitious goals: integrating precise manipulation and dynamic locomotion into a human-like platform designed to operate in human-built environments - navigating stairs, doors, tools, and unstructured spaces without requiring infrastructure changes. Physically, they feature a torso, two arms, two legs, and a sensor-equipped head, enabling them to interact with the world as humans do: walking, bending, reaching, opening doors, or handling tools. Their high degrees of freedom (typically over 30) grant exceptional versatility - and unprecedented control complexity.

While commercial deployment remains limited, humanoids are increasingly vital as research platforms in control, AI, and general-purpose robotics. Initiatives like the DARPA Robotics Challenge have showcased their potential in disaster response, while companies such as Boston Dynamics, Tesla, and Agility Robotics

are advancing their use in logistics, manufacturing, and assistance. This section explores their core principles - beginning with dynamic balance and whole-body control (Pratt & Manzo, 2013), followed by their physical architecture, distributed control systems, and emerging manufacturer ecosystem. The goal is to clarify why humanoids represent a distinct paradigm, what technical hurdles they face, and how they may reshape automation in human-centric environments.

1.3.1 Conceptual Paradigm and Operating Principles

The core motivation behind humanoid design is versatility. Unlike specialized robots -such as fixed arms for welding or warehouse vehicles- humanoids can operate in human-designed environments without infrastructure modifications: climbing stairs, opening doors, handling tools, or navigating cluttered spaces. This adaptability makes them ideal for disaster response, elder care, space missions, or complex logistics. Yet, this versatility comes at a high technical cost. Unlike wheeled robots, bipedal humanoids are inherently unstable, requiring continuous, active joint control to maintain balance, one of robotics' most complex challenges. This hinges on two pillars: dynamic balance via the Zero Moment Point (ZMP) and Whole-Body Control (Hirai et al., 1998).

The ZMP is the point on the ground where gravitational and inertial moments sum to zero. To avoid falling, the ZMP must remain within the support polygon defined by the feet's contact area. As a robot leans forward, the ZMP shifts toward the foot's edge; if it exits the polygon, the robot falls. The control system must continuously compute torso, leg, and arm trajectories to keep the ZMP within safe bounds -a computationally demanding task requiring high-precision sensors like the IMU for real-time orientation and motion estimation (Vukobratović y Stepanenko, 1972). Pioneered in robots like Honda's ASIMO, ZMP prioritizes static stability over dynamic agility.

As humanoids evolved toward more agile motion, Whole-Body Control emerged as a more advanced paradigm. Rather than controlling joints individually, it formulates motion as real-time hierarchical optimization. With over 30 degrees of freedom, multiple joint configurations can achieve the same end-effector position -a property known as kinematic redundancy. Whole-Body Control resolves this by prioritizing tasks: balance (highest), primary task (e.g., reaching), and secondary objectives (e.g., obstacle avoidance, posture comfort, energy efficiency (Khatib, 1987). This enables natural, adaptive movements — as seen in Boston Dynamics' Atlas, capable of running, jumping, or recovering from pushes. The controller solves this hierarchy instantaneously, generating coordinated joint commands. Whole-Body Control marks a qualitative leap, allowing humanoids to function in dynamic, unstructured environments where adaptability outweighs absolute precision (Sentis y Khatib, 2005).

1.3.2 Physical Architecture and Key Components

The physical architecture of a humanoid robot integrates mechanical, electrical, and control systems to replicate human functionality. Unlike fixed arms or mobile robots, humanoids must coordinate dozens of degrees of freedom across

their entire body, demanding a highly integrated design. This subsection examines their three core components: skeleton, actuators, and sensory system. The skeleton comprises a torso, two arms, two legs, and a head, with joints mimicking human anatomy -shoulders, elbows, wrists, hips, knees, and ankles- enabling operation in human-designed environments like stairs or doorways. Material choice is critical: it must balance rigidity with low weight to minimize inertia and energy use. Lightweight alloys (e.g., aluminum) or composites (e.g., carbon fiber) are common. The design must also ensure balanced mass distribution to support dynamic balance control (Hirai et al., 1998).

Actuators -the robot's "muscles"- drive joint motion and are typically electric or hydraulic. Electric actuators dominate modern designs, using high-torque, high-speed motors with optimized power-to-weight ratios. Many incorporate Series Elastic Actuators (SEAs), which include springs between motor and joint to provide compliance, shock absorption, and precise torque sensing -essential for safe interaction with humans or fragile objects (Pratt & Williamson, 1995). Hydraulic actuators, used in high-power platforms like Boston Dynamics' Atlas, offer superior force but are heavier, more complex, and energy-intensive, limiting them to research or high-risk applications.

Humanoids are densely instrumented with sensors for proprioception (internal state) and exteroception (environment). For proprioception: the torso-mounted IMU provides acceleration and angular velocity data, vital for orientation and balance; force/torque sensors at ankles, wrists, and waist measure contact forces for stability and force control; joint encoders track precise joint positions for kinematics and control. For exteroception: RGB and RGB-D cameras (typically in the head) enable object recognition, depth perception, and visual tracking; microphones support voice interaction; LIDAR generates 3D environmental maps for navigation (De Santis et al., 2008).

Humanoid's effectiveness lies not in isolated components but in their coordinated integration. For example: terrain changes detected by LIDAR/cameras trigger real-time balance adjustments via IMU and force sensors, while actuators execute stabilizing motions. This synergy enables autonomous operation in dynamic environments -from warehouses to university labs (Asada et al., 2009).

1.3.3 Computation and Control System

Controlling a humanoid robot cannot rely on a single processor, as it must handle tasks with vastly different timing demands, from high-level planning to real-time motor execution. Thus, its control system is distributed and hierarchical, coordinating multiple processors at different frequencies and priorities. This architecture comprises three levels: high-level computation, real-time control, and low-level control. Each plays a critical role, and their integration enables stable, coordinated, and adaptive motion (Hirai et al., 1998). Below, each part of the control unit is briefly described.

a) High-Level Computation -the robot's "Cognitive Brain"- handles non-urgent but computationally heavy tasks: perception, planning, and interaction. It fuses sensor data (LIDAR, cameras, IMU) to model the environment, decides actions

("walk to door," "grasp object"), and interprets voice or visual cues. Hardware typically includes multi-core CPUs (e.g., Intel i9, AMD Ryzen) and high-end GPUs (e.g., NVIDIA RTX) to accelerate neural networks. It runs on standard OS like Linux or Windows, without real-time constraints (Tobin et al., 2017).

b) Real-Time Control -the "spinal cord"- drives motion at high frequency (typically 1 kHz) with strict determinism: every operation must complete within a fixed time window. It executes Whole-Body Control and ZMP calculations, translating high-level commands into precise joint torques or positions. To guarantee timing, it uses a Real-Time Operating System (RTOS) or Linux with real-time patches (e.g., PREEMPT_RT). This layer is essential for balance and stability during locomotion and manipulation (Macchelli y Melchiorri, 2002).

c) Low-Level Control and Communication -Each joint has a local microcontroller (MCU) running fast loops to ensure motor compliance with real-time commands. To synchronize dozens of distributed controllers, robots use real-time fieldbus protocols. The standard in high-performance robotics is EtherCAT- an Ethernet-based protocol ensuring ultra-low latency and deterministic communication, critical for coordinated motion (Hu et al., 2022).

1.3.4 Key Humanoid Robot Manufacturers and Providers

Humanoid robotics has evolved from academic labs into a dynamic ecosystem blending pioneers, tech giants, and startups, signaling its transition toward commercial viability in logistics, manufacturing, and assistance.

Honda's ASIMO was the first humanoid to demonstrate stable bipedal locomotion and manipulation in structured settings, setting the benchmark for ZMP-based control for over two decades before its 2018 discontinuation. Its legacy remains foundational in robotics history (Hirai et al., 1998). NASA's Valkyrie R5, with 44 degrees of freedom and advanced sensors, was designed for space and hazardous environments. Though not commercial, it served as a key platform in the DARPA Robotics Challenge and general-purpose robotics research. Boston Dynamics' Atlas, meanwhile, represents the pinnacle of robotic agility, combining hydraulics, precision sensing, and Whole-Body Control to execute jumps, spins, and complex manipulation. While not sold commercially, its demonstrations have driven breakthroughs in dynamics and motion planning (Devin et al., 2018).

In the last decade, startups and tech firms have surged, aiming to commercialize humanoids powered by advances in AI, sensors, and electric actuators. Agility Robotics' Digit is among the first deployed commercially, designed for warehouse logistics and "last-mile" integration with mobile robots. Tesla's Optimus leverages the company's AI and manufacturing scale, targeting factory and home use, still in development but with vast potential. Figure AI's Figure 01 focuses on automating labor tasks, integrating LLMs for complex instruction-following. Apptronik's Apollo offers modular design for industrial settings, while Sanctuary AI's Phoenix uses telepresence for human-guided task learning. UBTECH's Walker S, among China's growing contributions, is already applied in automotive assembly. In academia, humanoids remain vital research platforms, accelerated by initiatives like the DARPA Robotics Challenge and

simulators such as NVIDIA Isaac Sim, which enable experimentation without physical hardware, democratizing access. Table 3 summarizes key humanoid robots mentioned in this section.

Table 3.
Main Humanoid Robot Platforms

Manufacturer	Humanoid robot	Country of Origin	Actuator type	Primary Focus
Boston Dynamics	Atlas	EE. UU.	Hidraulic	Advanced Research
Agility Robotics	Digit	EE. UU.	Electric	Logistics and Supply Chain
Tesla, Inc.	Optimus (Gen 2)	EE. UU.	Electric	Manufacturing and General Purpose
Figure AI	Figure 01	EE. UU.	Electric	General Workforce
Apptronik	Apollo	EE. UU.	Electric	Supply Chain and Manufacturing
Sanctuary AI	Phoenix	Canadá	Electric	General Purpose (Telepresence-based)
1X Technologies	NEO	Noruega	Electric	Safety and Consumer Affairs
UBTECH Robotics	Walker S	China	Electric	Smart Manufacturing (Automotive Industry)
NASA JSC	Valkyrie (R5)	EE. UU.	Electric (Rotary/Linear)	Space Exploration and Hazardous Environments

2. Robotic Control Methods: From Kinematics to Dynamics

Once a robotic platform, arm, mobile robot, or humanoid, is selected, the next critical step is enabling precise, controlled movement. While physical architecture defines what a robot can do, the control system determines how it does it. Robotic control transforms high-level tasks, like reaching an object, following a path, or maintaining balance, into precise motor and actuator commands. Technically, control relies on mathematical models describing the robot's behavior and its interaction with the environment. These models enable algorithms to generate the forces and torques needed for accurate motion, even under disturbances. This section introduces the foundational principles of robotic control, preparing the reader for the programming and training topics covered later.

2.1. Representation of Position and Orientation

Before precise control is possible, a robot's configuration in 3D space must be described rigorously. Pose -the combination of position and orientation relative to a reference frame- is the foundation for all motion planning and task execution. Without it, accurate control is impossible (Lynch & Park, 2023).

Position is represented by a vector in $\mathbb{R}^3$, indicating coordinates (X, Y, Z) relative to a chosen origin, which may be the robot's base, lab floor, or an object, depending on the task (Spong et al., 2006). For example, (0.5, 0.3, 0.2) means 50 cm in X, 30 cm in Y, 20 cm in Z. Orientation, more complex, describes how the body's axes (X', Y', Z') are rotated relative to the reference frame. It is most

commonly represented by a 3×3 rotation matrix R, belonging to the Special Orthogonal Group SO(3): its columns are orthonormal unit vectors and its determinant is +1 (ensuring pure rotation, no reflection) (Spong et al., 2006). This matrix encodes the spatial alignment of the body's axes, e.g., if X' aligns with the reference Y-axis, this is reflected in R's elements. While Euler angles or quaternions are alternatives, rotation matrices are preferred in robotics for their precision and ease of composition. To unify calculations, position and orientation are combined into a 4×4 homogeneous transformation matrix, belonging to the Special Euclidean Group SE(3), with the form:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

where:
R is the rotation matrix (3×3),
p is the position vector (3×1),

2.2. Kinematics: The Geometry of Motion

Kinematics studies motion without considering forces -in robotics, it describes the geometric relationship between joint positions and the end-effector's pose. It is the first step in motion planning: before applying forces, one must understand how the robot moves. Two core problems define it: forward and inverse kinematics-both essential for control and programming. Forward Kinematics (FK) is the "easier" problem. Given joint values (e.g., rotation angles), it computes the end-effector's pose in space. For serial manipulators, this is done by composing homogeneous transformations from base to end-effector, each depending on link geometry and joint value (Lynch y Park, 2023). The solution is always unique and efficient. For example, with known joint angles, FK determines exactly where a 6-DOF arm's wrist is and how it's oriented, crucial for simulation, collision checking, and trajectory validation.

Inverse Kinematics (IK) is the "harder" and more critical problem. Given a desired end-effector pose (e.g., "grasp a glass"), it finds the joint values to achieve it. Unlike FK, IK may have no solution (if the pose is unreachable) or multiple solutions due to kinematic redundancy, when the robot has more DOF than needed (Spong et al., 2006). For instance, a 7-DOF arm can reach the same point with different elbow configurations, useful for obstacle avoidance or posture optimization, though it complicates control.

Two main approaches solve IK:

- Analytical (closed-form) solutions, fast and precise, but limited to simple architectures (e.g., symmetric 6-DOF arms).
- Numerical methods, required for complex or redundant robots, using iterative techniques like the Jacobian method to approximate solutions, slower but universally applicable.

The Jacobian matrix is central to numerical IK: it maps joint velocities to end-effector linear and angular velocities, describing how small joint changes affect

motion. It enables movement optimization, singularity analysis (configurations where DOF are lost, e.g., aligned joint axes), and the design of velocity and force controllers(*Craig*, 2005).

2.3. Dynamics: The Study of Forces and Motion

If kinematics describes how a robot moves, dynamics explains why, studying the relationship between actuator forces/torques and resulting motion, accounting for mass, inertia, and gravity. In practice, dynamics is essential for designing controllers that are precise, efficient, and safe. For example, a robotic arm lifting a heavy load must apply enough torque to overcome inertia and gravity, but not so much as to overload motors or cause jerky motion. In robotics, the fundamental problems of forward and inverse dynamics are distinguished. Forward dynamics answers the question: What acceleration is produced by a given set of joint torques? Given a vector of applied torques τ, forward dynamics calculates the resulting acceleration q̈ of each joint, according to the equation (Lynch y Park, 2023):

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q)$$

where:
M(q) =mass and inertia matrix,
$C(q,\dot{q})$=Coriolis and centrifugal forces, and
G(q) =gravitational torque vector.

This calculation is fundamental for robotic simulation, as it allows predicting how the robot will move under different load or control conditions.

Inverse dynamics is the inverse problem: What torques must be applied for the robot to follow a desired trajectory? Given a specific trajectory, defined by positions q, velocities q̇, and accelerations q̈ over time, inverse dynamics calculates the torques τ that the motors must generate at each instant to execute that trajectory with precision. This is the central problem of advanced robot control, especially in tasks requiring high speed, precision, or manipulation of variable loads (Featherstone, 2008). The most common method for solving inverse dynamics is the recursive Newton-Euler Algorithm, which is computationally efficient and allows calculating torques in real-time. This algorithm propagates forces and moments from the base to the end-effector (forward pass) and then traces back to calculate the torques at each joint. Inverse dynamics is the basis of feedforward control (*Craig,* 2005). Instead of waiting for a tracking error to occur to correct it (as feedback control does), the system anticipates the necessary torques and applies them proactively. This approach drastically improves the accuracy and speed of trajectory tracking, especially in fast, heavy, or variably loaded robots. In practice, feedforward control is combined with feedback control to create a hybrid system that is both proactive and reactive.

2.4 Architecture of a Modern Control System: The Feedback Loop

Kinematic and dynamic models offer an idealized view of robot behavior. In practice, however, real-world uncertainties, model inaccuracies, unmodeled

disturbances (e.g., collisions or variable payloads), and actuator errors, demand more robust control. Thus, robotic systems combine feedforward with a feedback control loop, the core architecture linking sensors, computers, and actuators in a continuous cycle. The information flow in a typical trajectory tracking loop is as follows (Spong et al., 2006):

a) **Desired State**. The high-level system (the task planner) defines a desired trajectory $q_d, \dot{q}_d, \ddot{q}_d$ for the robot's joints over time.

b) **Measurement of Real State** (Sensors). At each instant of the control loop (typically every millisecond), sensors measure the robot's real state. High-resolution encoders on each joint provide the real position q_r and, through differentiation, the real velocity $\dot{q}_r$.

c) **Error Calculation**. The real-time control unit (the computer) compares the desired state with the real state to calculate the tracking error: $e = q_d - q_r$

d) **Control Law** (Software). The heart of the loop is the control law, an algorithm that calculates the torque or force command needed to correct the error. The most ubiquitous controller in industry and robotics is the PID (Proportional-Integral-Derivative) controller. The proportional (P) component reacts to the magnitude of the current error. The larger the error, the greater the corrective force. The integral (I) term accumulates the error over time. Its function is to eliminate steady-state error, correcting small but persistent deviations. The derivative (D) part reacts to the rate of change of the error. It acts as a damper, predicting and attenuating oscillations to stabilize the system.

e) Actuator Command Generation. The output of the PID control law is often combined with the feedforward term calculated by inverse dynamics. The final command $\tau = \tau_{feedforward} - \tau_{feedback}$ is a much more robust torque value. This value is sent via a real-time communication bus (such as EtherCAT) to the motor control boards.

f) **Action** (Actuators). The motor electronic boards receive the torque command and translate it into a precise electrical current that is sent to the electric motors. These motors generate physical torque at the joint, producing movement.

g) **Repetition Cycle**. The resulting movement alters the robot's real state, which is measured again by the encoders in the next cycle, starting the process once more.

The cyclical interaction between perceiving (encoders), processing (PID controller in the computer), and acting (motors) is what allows a robot not only to follow a trajectory but also to actively reject disturbances and maintain precision and stability in the face of real-world imperfections. It is the true essence of applied robotic control (Åström y Hägglund, 2005).

3. Programming Paradigms and Software Architectures

Having explored how robot motion is modeled (kinematics), driven (dynamics), and robustly controlled (feedback), the next step is programming, enabling the robot to perform useful tasks like assembly, navigation, or lab manipulation. If control defines how a robot moves, programming determines what

it does, making movement intelligent, scalable, and safe. Robotic programming goes beyond writing code: it involves designing behaviors that allow the robot to interact meaningfully with its environment (Quigley, 2009). In practice, modern robots are not programmed like traditional applications, but as distributed systems of parallel processes -perception, planning, control, and decision-making- built on tools and frameworks that abstract hardware complexity, letting developers focus on application logic (Andreasson et al., 2023). To help the reader understand the tools available for robot programming, foundational for automation in academic, industrial, or research settings, this section presents the main paradigms, organized by abstraction level.

3.1. The Software Ecosystem: From Direct Programming to Frameworks

Robotic software development spans multiple abstraction levels, each suited to different tasks, customization needs, and team expertise. This section outlines the main paradigms. Direct programming targets the robot's microcontroller or main controller, typically in C/C++ for maximum speed and hardware control. While offering fine-grained performance, it demands deep electronics and hardware knowledge, and is impractical for full-system coordination -reserved for manufacturers or specialized control research (Liu et al., 2022). To simplify development, most manufacturers provide Software Development Kits (SDKs). libraries with APIs in C++ or Python. Functions like *robot.move_to_pose()* abstract hardware complexity, letting developers focus on kinematics and task logic. This is the standard middle ground for industrial applications requiring precision without low-level reinvention (Ahmad y Babar, 2016).

As robots grew more complex -with multiple sensors, actuators, and concurrent processes- dedicated software frameworks (middleware) became essential. These are not OSs like Linux, but layers running atop them, offering standardized tools for distributed development. They enable transparent communication between components (e.g., camera and planner nodes), hardware abstraction (same algorithm works across sensor types), and integrated tools for visualization, simulation, debugging, and package reuse - supported by community repositories (Andreasson et al., 2023)

Framework adoption represents the highest abstraction level, enabling modular, scalable collaboration. While ROS (detailed in Section 4.3) dominates research and startups with its node-based architecture, vast package library, and tools like RViz/Gazebo, alternatives exist depending on project priorities (Ahmad y Babar, 2016):

- MATLAB/Simulink + Robotics System Toolbox excels in model-based design (aerospace, automotive). Its graphical interface enables control system simulation, performance analysis, and automatic C/C++ code generation for hardware deployment — ideal for rapid prototyping and "design-to-hardware" workflow (Falconi et al., 2006).
- NVIDIA Isaac targets AI-driven robotics. It integrates Isaac Sim (photorealistic simulator), Isaac ROS (GPU-optimized packages), and libraries like Isaac Manipulator, all focused on GPU acceleration and Sim-to-

Real transfer for training AI models. A leading choice for intelligent, perception-heavy robot (Zhou et al., 2023).

Crucially, major industrial players (KUKA, FANUC, ABB) rely on proprietary frameworks (e.g., KUKA.OS), designed for certified safety, real-time determinism, and integration with field buses like EtherCAT -non-negotiable in production. Less flexible than open-source options, they guarantee the robustness and support industry demands. Thus, framework choice is strategic- balancing flexibility (research), mathematical rigor (control), computational power (AI), or reliability (industry).

3.2. Simulators

Developing software directly on a physical robot is slow, costly, and risky, a single error can cause collisions damaging hardware, environment, or personnel. Thus, simulators have become indispensable in modern robotics. A robotic simulator is a virtual environment modeling both world physics and the robot's kinematics, dynamics, and sensors. Simulation serves multiple critical roles (Kober et al., 2013). First, it enables safe, repeatable algorithm development and debugging-testing motion planning, SLAM, or control in varied virtual scenarios, with instant restarts upon failure, accelerating iteration. Second, it's essential for generating synthetic data to train deep learning models in perception and reinforcement learning. Real-world data collection is a bottleneck; simulators generate thousands of environment variations, lighting, textures and object poses with perfect labels, drastically speeding AI training.

Third, in professional workflows (CI/CD), simulators enable automated testing: every code change triggers simulation tests, preventing regressions and improving software robustness (Tobin et al., 2017). Finally, in professional development workflows (CI/CD-Continuous Integration/Continuous Deployment), simulators enable automated testing. Each code modification can trigger a battery of tests in simulation, ensuring that no regressions or unexpected behaviors are introduced into the system, thereby improving the robustness and reliability of the robotic software.

A high-quality simulator requires key components: a robust physics engine (e.g., MuJoCo, PhysX, Bullet) to model gravity, collisions, and friction; realistic sensor modeling -including noise- for RGB-D, LIDAR, force/torque sensors, and IMUs; and seamless integration with frameworks like ROS or MATLAB, allowing code developed in simulation to deploy on real robots with minimal changes. The ecosystem is diverse, with tools tailored to different needs:

- Gazebo, ROS's flagship open-source simulator, versatile for robot and sensor modeling, though not photorealistic.
- NVIDIA Isaac Sim, built on Omniverse, leader in photorealism and physical accuracy, with real-time ray tracing and tight AI integration, ideal for Sim-to-Real.
- CoppeliaSim (V-REP), flexible, user-friendly, popular in education and prototyping, with multi-language support.

- Webots, open-source, cross-platform, widely adopted in robotics competitions.
-

In summary, simulation is no longer optional, it is central to the robotics workflow, enabling faster, safer, and more cost-effective development, and serving as the key enabler for large-scale AI deployment in robotics.

3.3. Robot Operating System (ROS): A De Facto Standard

Within robotic software frameworks, the Robot Operating System (ROS) deserves special attention, it has become a de facto standard in research and an expanding pillar in commercial applications. Despite its name, ROS is not an OS but a flexible middleware: a collection of libraries, tools, and conventions designed to simplify building complex robotic behaviors across diverse platforms (Quigley, 2009). Its adoption stems from core design principles. Foremost is modularity via a node-based architecture: independent processes (nodes) handle specific tasks, e.g., sensor reading, path planning, motor control. Nodes communicate via a publish/subscribe model: a laser node publishes data to a “topic”; any subscribing node (e.g., planner) receives it. This decoupling allows nodes to be developed, tested, and restarted independently, in different languages (C++/Python), even across networked machines.

ROS’s true power lies in its global open-source ecosystem. Thousands of packages implement nearly any robotic function: navigation2 offers full SLAM, localization, and planning stacks; MoveIt provides motion planning, IK, and collision avoidance for arms (Şucan et al., 2012). Hundreds of sensor drivers (LIDAR, cameras, IMUs), plus libraries like PCL and OpenCV, integrate natively. Development is accelerated by tools like RViz (3D visualization of sensors, transforms, robot state) and Gazebo (realistic simulation) (Koenig y Howard, 2004). Others like *rqt* and *rosbag* aid debugging and data recording.

Critically, ROS has evolved. ROS 1, while revolutionary, lacked features for commercial deployment: no real-time guarantees, weak security, poor network resilience. ROS 2, a full rewrite on the industrial DDS middleware, addresses these with:

- Multi-platform and Embedded Systems Support: It can run on operating systems like Windows and macOS, and on microcontrollers,
- Real-time Communication: It offers Quality of Service (QoS) guarantees for more reliable and deterministic communication, and
- Enhanced Security: It includes a security framework for authentication and encryption of communication between nodes.

Currently, ROS 2 is the recommended standard for all new projects. While migration from ROS 1 continues, the community has consolidated around ROS 2 as the future platform, spanning academia to industry (Steven Macenski et al., 2022).

3.4. High-Level Programming and Control Interfaces

While frameworks like ROS provide infrastructure for communication and process management, task-specific programming often occurs through even higher-level interfaces. These abstract motion planning and low-level control, allowing developers to specify what to do - not how to do it, making robot programming more intuitive, accessible, and less error-prone (van den Berg et al., 2009). Several dominant paradigms for high-level programming exist, which are briefly analyzed below.

For many industrial and collaborative robots, the main interface is a teaching pendant with GUI. Operators without coding skills can “teach” waypoints by manually guiding the robot or using a joystick. The system records and replays the sequence, ideal for repetitive, structured tasks, but inflexible to environmental changes. Skill-based programming offers greater flexibility. Instead of low-level motions, developers use predefined “skills”, behavioral modules encapsulating complex actions. For example, calling skill_pick_object(location) internally handles perception, grasp planning, collision avoidance, and force control. This modular approach enables reuse across applications (Kofer et al., 2021).

The current frontier is natural language control, powered by LLMs (e.g., GPT series). These interfaces translate ambiguous instructions (“pick up the object on the table and bring it to the shelf”) into executable actions. The core challenge is symbol grounding, linking words (“red bottle”) to perceptual data (e.g., a colored point cloud). LLMs decompose instructions into logical plans executed by the robot’s control stack. Though still evolving, early demonstrations show great potential for non-expert accessibility(Kofer et al., 2021).

Another method is Programming by Demonstration (PbD), where users teach tasks by physically guiding the robot or using teleoperation. The system observes the demonstration and uses machine learning to generalize it into a policy that adapts to new, similar situations (Argall et al., 2009). PbD will be explored in detail in the next section.

4. Robot Training: The Machine Learning Paradigm

Traditional control and programming excel in structured environments but struggle with real-world variability. Explicitly coding every contingency is infeasible. To overcome this, modern robotics increasingly adopts machine learning: instead of being programmed, robots are trained, learning control policies from data to generalize and master complex skills like manipulating deformable objects or operating in clutter. This section explores the two main training approaches: Imitation Learning and Reinforcement Learning (Kober et al., 2013).

4.1. Imitation Learning

Imitation Learning -or Programming by Demonstration (PbD)- is the most intuitive training method: if a human can perform a task, the robot learns by observing them. Instead of code, an expert demonstrates the skill; the robot generalizes from these examples. It’s especially powerful for dexterous, contact-

rich manipulation. PbD involves three stages: demonstration collection, policy learning, and reproduction with refinement.

Demonstration collection gathers data from human performance. Key methods include (Schaal, 1999):

- Kinesthetic teaching: the operator physically guides the robot while its encoders record joint trajectories, intuitive but limited by human strength.
- Teleoperation: remote control via joysticks, VR, or exoskeletons, enables high-fidelity force and orientation control, even at a distance.
- Passive observation: the robot watches a human via cameras, most challenging, requiring vision-based pose estimation and human-to-robot motion mapping

Policy learning transforms demonstrations (state-action sequences) into a policy, a function mapping observation to actions. The simplest method is behavioral cloning (BC), treating it as supervised learning: predicting expert action from observed state. Easy to implement, but fragile, small errors lead to unseen states where the policy fails. More robust are generative models like Gaussian Mixture Models (GMMs) or Dynamic Movement Primitives (DMPs), which model the distribution of demonstrations. They generate smooth motions, adapt to variations (e.g., object position changes), and offer greater robustness to perturbations (Ijspeert et al., 2002).

Reproduction and refinement deploy the policy for task execution. Success depends on generalization beyond original demonstrations. Performance often improves by combining PbD with reinforcement learning, allowing the robot to refine the policy through its own experience (Argall et al., 2009). Imitation Learning reduces programming burden and transfers human dexterity to machines, enabling automation of tasks once thought exclusively human.

4.2. Reinforcement Learning

While Imitation Learning teaches a robot what to do based on expert demonstrations, Reinforcement Learning (RL) is a paradigm where the robot learns autonomously through direct interaction with its environment. Rather than being shown correct behavior, it is given a goal encoded in a reward function and must discover - through trial and error- a sequence of actions that maximizes the total accumulated reward over time. This approach is particularly suited to tasks where the optimal solution is non-obvious or too complex for a human to demonstrate effectively (Sutton y Barto, 2018).

The RL framework is defined by the continuous interaction between the agent, the robot, and its environment, which encompasses physical dynamics, objects, and task constraints. At each moment, the system is described by a state variable s, encoding the robot's configuration and environmental context, such as joint angles, object poses, or sensor readings. From this state, the agent selects an action a, such as applying torque to a joint or initiating a movement. The environment then responds with a scalar reward r, signaling how well the action

aligns with the task objective. Designing this reward function is one of the most critical and delicate aspects of applied RL; a poorly specified reward can lead the agent to exploit unintended loopholes, a phenomenon known as reward hacking, producing behaviors that maximize reward without fulfilling the true intent of the task

Central to the learning process is the policy π, a function that maps states to actions and define the agent's decision-making strategy. As in imitation learning, the objective is to find an effective policy, but in RL, no demonstrations guide the agent. Instead, it must discover the optimal policy π* through iterative exploration: observing the state, selecting an action, receiving a reward, transitioning to a new state, and updating its policy accordingly. Over time, this loop refines the agent's behavior toward maximizing long-term reward.

Applying RL directly to physical robots, however, presents significant practical challenges that limit its real-world viability (Kober et al., 2013). One major obstacle is sample inefficiency: most RL algorithms require millions of interactions to converge, which is infeasible on physical hardware due to mechanical wear, operational costs, and time constraints. Equally critical is the issue of safety during early exploration, the robot may execute erratic or unstable actions that risk damaging itself, its surroundings, or nearby personnel. Compounding these difficulties is the inherent complexity of reward design: crafting a function that accurately captures task intent while avoiding unintended shortcuts remains an art as much as a science. To address these limitations, modern robotics increasingly relies on Deep Reinforcement Learning, which leverages deep neural networks to approximate policies or value functions. Algorithms such as Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) have become standard tools for continuous control in robotics, valued for their training stability and capacity to generalize across tasks (Haarnoja et al., 2018).

Several strategies have emerged to make RL practical. First, training predominantly in simulation allows the agent to accumulate millions of safe, low-cost experiences before deployment -though transferring the learned policy to the real world, known as Sim-to-Real, remains a core challenge addressed in the next section. Second, Reinforcement Learning from Demonstrations incorporates expert trajectories to initialize the policy, accelerating convergence and steering exploration toward useful behaviors. Finally, careful reward engineering -replacing sparse rewards with dense, informative signals- and the use of learned world models that simulate action outcomes before execution, both contribute to more efficient and reliable learning. Reinforcement Learning represents the promise of truly autonomous robotics, systems capable of acquiring complex motor skills such as agile locomotion or dexterous manipulation, potentially surpassing human performance in specific domains. While its widespread practical application remains an active area of research, RL is undeniably one of the most transformative and forward-looking directions in the field (Levine, 2018).

4.3. Simulation to Reality

As previously discussed, training a reinforcement learning agent directly on a physical robot is typically infeasible due to sample inefficiency and safety risks. The prevailing solution is to train the control policy extensively in simulation and then transfer it to the real robot, a process known as Simulation-to-Reality (Sim-to-Real) transfer. This has become a cornerstone paradigm and active research area in modern robotics (Kober et al., 2013). Its success hinges on overcoming the reality gap, the unavoidable mismatch between simulated and physical environments (Tobin et al., 2017).

To bridge this gap, several key strategies have emerged. One is increased simulation fidelity: calibrating the simulator with real-world measurements, such as masses, inertias, or friction coefficients, obtained through system identification. Platforms like NVIDIA Isaac Sim prioritize physical accuracy and photorealistic rendering to minimize discrepancies from the outset. A more robust, counterintuitive approach is domain randomization: instead of pursuing a single perfect simulation, the policy is trained across thousands of randomized environments, varying object mass, friction, lighting, or textures. By exposing the agent to extreme diversity, it learns policies that generalize to real-world conditions, even if those exact conditions were never simulated (Tobin et al., 2017).

Domain adaptation offers another path: aligning real-world sensor data with the simulator's expected format. Techniques like Generative Adversarial Networks (GANs) can transform real camera images into styles resembling simulated visuals, allowing the policy to process them without retraining (Bousmalis et al., 2016). Finally, real-world fine-tuning enables final adaptation: once transferred, the policy undergoes limited additional training using real data. Because simulation provides a functional starting point, only minimal real interactions are needed to adjust for subtle physical nuances (Peng et al., 2018).

Finally, real-world fine-tuning enables final adaptation: once transferred, the policy undergoes limited additional training using real data. Because simulation provides a functional starting point, only minimal real interactions are needed to adjust for subtle physical nuances.

5. A Roadmap for Getting Started with Hands on Practice

Having explored the theoretical foundations of applied robotics, from manipulator kinematics to reinforcement learning in simulation, a natural question emerges: how do you take the first practical step? While this article offers a unified conceptual framework, its true value lies in applying these ideas to build, program, and experiment with real systems. This section serves as a practical, accessible guide for professionals, educators, and students seeking to begin hands-on robotics, without requiring an industrial lab or large budget. It proposes a phased roadmap, grounded in real platforms, available tools, and best practices, enabling readers to transition smoothly from theory to action, using the very principles and technologies discussed throughout this paper.

5.1 Learning Resources: Simulators and Development Platforms

The first practical step in learning modern robotics doesn't require physical hardware. Accessible simulators and development platforms let non-specialists explore control, programming, and perception in safe, repeatable environments, ideal for grasping software architecture, inter-process communication, and complex task execution, as discussed earlier. Among the most widely used is Gazebo, an open-source physics simulator enabling realistic modeling of robots, sensors, and dynamic environments. Historically tied to ROS 1 and now fully compatible with ROS 2, it's extensively used to test navigation, manipulation, and SLAM algorithms (Koenig y Howard, 2004). A strong alternative is Webots, developed by CSEM, offering an intuitive interface, multi-language support (Python, C++), and preconfigured robot models, especially suited for education and research.

Other notable platforms include CoppeliaSim (formerly V-REP), valued for its flexibility and modularity, and NVIDIA Isaac Sim, a next-generation GPU-accelerated simulator that generates photorealistic synthetic data for training AI models with physical accuracy. On the software side, ROS 2 has become the de facto standard for robotic development in research and industry (Macenski et al., 2022). Designed from the ground up for real-time performance, safety, and distributed communication via DDS, its ecosystem includes Navigation2 (for mobile robots), MoveIt 2 (for manipulators), and countless open-source packages enabling complex behavior development.

For beginners, combining a simulator like Gazebo or Webots with ROS 2 creates a complete, hardware-free learning environment, perfect for implementing and testing trajectory planning, feedback control, and sensor fusion.

5.2. Accessible Hardware Kits and Platforms

After mastering control, programming, and perception in simulation, the next natural step is transitioning to physical hardware. Fortunately, a growing range of accessible platforms, commercial and open-source, enables students, educators, and professionals to experiment with real robots without industrial budgets. These can be grouped by the three main robot types introduced in Section 2: manipulators, mobile robots, and humanoids.

For manipulators, the OpenMANIPULATOR-X (ROBOTIS) is a 4-DOF arm based on Dynamixel servos, fully ROS 2 compatible. Its modularity and open documentation make it ideal for learning inverse kinematics and pick-and-place tasks. The Dobot Magician/MG400 offers 4–6 DOF with intuitive interfaces and ROS, vision, and force sensor integration. For advanced applications, the Franka Emika Panda, a 7-DOF cobot, is widely used in labs to explore force control and human-robot interaction.

In mobile robotics, the TurtleBot 4, now ROS 2 native, comes in two versions: Raspberry Pi for basic use, and NVIDIA Jetson for GPU-accelerated vision tasks. It's perfect for SLAM, navigation, and sensor fusion. The JetBot (Jetson Nano-based) focuses on AI, enabling real-time neural network deployment for tracking or obstacle avoidance. For field applications, platforms like Ardupilot Rover support building larger autonomous vehicles for agriculture or logistics.

Among humanoids, though costlier and more complex, options exist for education and research. The Poppy Humanoid is a fully 3D-printable, open-source platform designed for control and cognition research, with ROS compatibility and modular design. The ROBOTIS OP3 is a mid-sized humanoid used in labs and competitions, also ROS-integrated. Platform choice should align with project goals, budget, and experience level. Many integrate directly with simulators like Gazebo or NVIDIA Isaac Sim (Zhou et al., 2023), enabling Sim-to-Real workflows from day one. This synergy of affordable hardware and open software is democratizing applied robotics, empowering more people to bridge theory and practice.

5.3. From Simulation to the Real World

To transform theory into practice, a progressive learning path is recommended, minimizing complexity while maximizing tangible progress. This structured approach helps beginners consolidate fundamentals before advancing, reducing the frustration often associated with robotics. The key stages are outlined below by learning level.

Level 0 (Familiarization) focuses on conceptual understanding: kinematics, dynamics, feedback control, software architecture, and perception. At this stage, exploring official documentation (e.g., ROS 2), online tutorials, and introductory courses (e.g., Coursera) is ideal (*Coursera*, 2025). The goal is not coding mastery, but grasping the vocabulary and logic underlying robotic systems. Level 1 (Simulation Practice) shifts focus to hands-on control in virtual environments. Using Gazebo or Webots with ROS 2, learners can command virtual robots, moving a mobile platform to a target, implementing PID for trajectory tracking, or guiding a manipulator to a spatial point. This level enables safe experimentation with SLAM, navigation, and inverse kinematics, where errors carry no physical cost (Macenski et al., 2023). Level 2 (Hardware Transition) bridges simulation and reality by deploying code on accessible platforms like TurtleBot 4 or Dobot Magician (Ghazal et al., 2024). Here, real-world challenges emerge: sensor noise, latency, mechanical imperfections, and environmental variability. Debugging and adapting software to overcome these issues becomes a core skill. Level 3 (Integrated Projects) combines multiple domains to solve functional tasks, e.g., programming a robot to navigate, locate, and deliver an object, or designing a vision-guided manipulation system using OpenCV or YOLO for detection and pose estimation. This stage may incorporate task planning, and eventually, imitation or reinforcement learning in simulation (Zhou et al., 2023). Level 4 (Documentation & Scaling) closes the learning cycle. Documenting the project on GitHub, writing a technical report, or presenting results at a forum or competition reinforces knowledge and prepares learners for collaborative work, where clarity, reproducibility, and communication are as vital as technical ability.

5.4. Educational Resources and Learning Communities

Learning robotics is not a solitary endeavor, it thrives with access to quality resources and active communities of developers, researchers, and enthusiasts. These provide critical support to overcome initial hurdles and sustain progress on

increasingly complex projects. Structured online courses, such as “Robotics” (University of Pennsylvania, Coursera) or “Robotics: Perception” (edX), offer rigorous introductions to mathematical and algorithmic foundations. Platforms like The Construct provide cloud-based ROS 2 simulations with guided tutorials and hands-on challenges.

Free YouTube content from channels like ROS Developers, Robotics Back-End, and The Construct is invaluable for troubleshooting and seeing real implementations. Official documentation, especially the ROS Wiki and ROS 2 docs, is essential for mastering nodes, topics, services, and launch files. GitHub hosts thousands of open-source packages, from SLAM to arm controllers, enabling learners to study, adapt, and reuse real code. Online communities are vital for problem-solving and collaboration. ROS Answers is the go-to forum for technical questions, while Stack Overflow offers additional support. Platforms like Discord and Reddit (e.g., r/ROS, r/robotics) host dynamic discussions on projects, hardware, and trends. Participating in hackathons, competitions (RoboCup, NASA Space Robotics Challenge), or university clubs provides team-based, goal-driven learning, accelerating practical skill development under real-world constraints. Together, these resources form a vibrant learning ecosystem, turning theory into practice, and isolation into collaboration. Leveraging them is key to transforming interest into real skills and tangible robotic projects.

CONCLUSIONS

This article has explored the core pillars of modern applied robotics: platform selection (Section 2), control methods (Section 3), software architectures (Section 4), and machine learning-based training (Section 5). Together, they form a foundation for understanding how to choose, program, and train robots for real-world tasks, from precise manipulation to adaptive behavior in unstructured environments. Though presented sequentially for clarity, these elements do not operate in isolation. Leading robotic systems today integrate model-based control, structured programming, and data-driven learning in hybrid architectures, defining the current frontier of the field. The following sections synthesize how these pillars interact in practice, examine the key technical challenges that remain, and offer guidance for applying this knowledge to real-world project.

This article has outlined the fundamentals of modern applied robotics, from platform selection to control, programming, and training. What emerges is that contemporary robotics is not a single discipline, but a synergistic integration of multiple fields. Model-based control, modular software architectures, and machine learning are not competing paradigms; they are complementary layers that, when combined, enable more capable, robust, and adaptive systems. Model-based control -rooted in kinematics and dynamics- delivers the precision, stability, and safety needed for structured environments, such as industrial assembly or lab automation. Yet it falters under uncertainty or complex physical interaction. Here, software frameworks like ROS step in, providing the modular architecture and inter-process orchestration (perception, planning, control) that make systems scalable, reusable, and maintainable, acting as the “glue” between layers.

Machine learning, via imitation and reinforcement learning, equips robots to perceive, adapt, and act in unstructured settings. It enables skills like grasping deformable objects or walking on rough terrain, tasks nearly impossible to hand-code. The Sim-to-Real paradigm has made this transition feasible, allowing policies trained in simulation to deploy successfully in the physical world. For professionals, scientists, or students, the key insight is not to master one domain, but to understand how these pillars integrate. Practical application begins with a simple question: What problem can I automate? Once defined, whether in a lab, factory, or research setting, platform selection (manipulator, mobile, humanoid) follows logically. The real challenge is combining the right tools: motion planning and feedback control for predictable tasks; machine learning when adaptability, dexterity, or environmental interaction is required.

Robotics is no longer futuristic, it's a present, tangible tool transforming industries, accelerating discovery, and improving lives. This guide aims to demystify its complexity, offering a conceptual framework and clear roadmap to bridge theory and practice.

Based on the conceptual framework presented, the following strategic recommendations guide professionals, educators, and students seeking to apply robotics effectively, not as rigid rules, but as reflective principles for navigating complexity. Begin with the problem, not the robot. It's tempting to start with technology, but the most effective path begins with a clear need: What repetitive, dangerous, or high-precision task can be automated? Whether in a lab, factory, or research setting, defining the task, sample handling, inspection, transport, makes platform selection (manipulator, mobile, humanoid) a logical next step, not an overwhelming starting point. This approach ensures technology serves real needs, not novelty.

Embrace interdisciplinary, progressive learning. Robotics demands convergence: mechanical engineers should learn Python and control basics; computer scientists must grasp kinematics and DOF. Start in simulation (Gazebo, Webots) for safe experimentation, advance to low-cost hardware (TurtleBot, educational arms), then scale to industrial systems. Robotics is a "team sport", collaboration across disciplines often yields the most robust solutions.

Treat simulation as a democratizing tool. Without access to expensive hardware, virtual environments like Gazebo, Webots, or NVIDIA Isaac Sim offer full access to core concepts, kinematics, SLAM, feedback control, ROS, and machine learning, at zero hardware cost and no safety risk. Simulation lowers barriers and accelerates learning through rapid, repeatable iteration. Prioritize robustness over AI, at first. While machine learning is exciting, initial applications should rely on proven methods: PID control, motion planning, structured programming. A robot following a predefined trajectory is often faster, safer, and more reliable than a complex AI model. Master these fundamentals first, view AI as a strategic complement, not a replacement.

Finally, see the robot as a capability multiplier, not a human replacement. In labs, robots handle thousands of repetitive experiments, freeing scientists for design and analysis. In industry, cobots take on heavy or monotonous tasks, letting

humans focus on quality, decisions, and adaptability. The right question is not "*Can a robot do my job*?" but "*What parts can it handle so I can achieve far more*?"

Nomenclature

q = Vector of joint variables (joint positions).
q_d = Vector of desired or commanded joint positions.
q_r = Vector of actual or measured joint positions.
$\dot{q}$ = Vector of joint velocities (first time derivative of q)
$\dot{q}$_d, $\dot{q}$_r = Vectors of desired and actual joint velocities, respectively.
$\ddot{q}$ = Vector of joint accelerations (second time derivative of q).
p = Position vector in Cartesian space ($\mathbb{R}^3$).
R = 3×3 rotation matrix representing orientation, belonging to SO(3)
SO(3) = Special Orthogonal Group; the space of all 3D rotations.
SE(3) = Special Euclidean Group; the space of all rigid-body transformations (poses) in 3D.
J = Jacobian matrix; relates joint velocities to end-effector velocities.
τ = Vector of torques or forces applied at the joints.
e = Error vector
s = State of the agent in the context of Reinforcement Learning.
A = Action taken by the agent in Reinforcement Learning.
R = Reward signal received by the agent in Reinforcement Learning.
Π = Policy; the strategy that maps states to actions in Reinforcement Learning.

Acronyms

AMR = Autonomous Mobile Robot (Robot Móvil Autónomo)
API = Interfaz de Programación de Aplicaciones
DDS = Data Distribution Service
DOF = Degrees of Freedom (Grados de Libertad)
FK = Forward Kinematics (Cinemática Directa)
GPU = Graphics Processing Unit (Unidad de Procesamiento Gráfico)
GUI = Graphical User Interface (Interfaz Gráfica de Usuario)
HRI = Human-Robot Interaction (Interacción Humano-Robot)
IK = Inverse Kinematics (Cinemática Inversa)
IMU = Inertial Measurement Unit (Unidad de Medición Inercial)
LfD = Learning from Demonstration (Aprendizaje por Imitación)
LIDAR = Light Detection and Ranging
LLM = Large Language Model (Gran Modelo de Lenguaje)
PID = Proportional-Integral-Derivative (Proporcional-Integral-Derivativo)
RL = Reinforcement Learning (Aprendizaje por Refuerzo)
ROS = Robot Operating System
RTOS = Real-Time Operating System (Sistema Operativo en Tiempo Real)
SDK = Software Development Kit
SLAM = Simultaneous Localization and Mapping (Localización y Mapeo Simultáneos)
ZMP = Zero Moment Point (Punto de Momento Cero)

REFERENCES

Ahmad, A., & Muhammad Ali, B. (2016). Software architectures for robotic systems: A systematic mapping study. Journal of Systems and Software, 122, 16–39. https://doi.org/10.1016/j.jss.2016.08.039

Andreasson, H., Grisetti, G., Stoyanov, T., & Pretto, A. (2023). Software architectures for mobile robots. In Encyclopedia of Robotics (pp. 1–11). Springer. https://doi.org/10.48550/arXiv.1703.06907

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., & Yoshida, C. (2009). Cognitive developmental robotics: A survey. IEEE Transactions on Autonomous Mental Development, 1(1), 12–34. https://doi.org/10.1109/TAMD.2009.2021702

Argall, B., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. Robotics and Autonomous Systems, 57(5), 469–483. https://doi.org/10.1016/j.robot.2008.10.024

Bechar, A., & Vigneault, C. (2016). Agricultural robots for field operations: Concepts and components. Biosystems Engineering, 149, 94–111. https://doi.org/10.1016/j.biosystemseng.2016.06.014

Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., & Erhan, D. (2017). Domain separation networks. Advances in Neural Information Processing Systems, 30, 343–351. https://doi.org/10.48550/arXiv.1608.06019

Craig, J. (2005). Introduction to robotics: Mechanics and control (3rd ed.). Pearson.

Coursera. (27 de agosto de 2025). Robotics Foundations I-Robot Modelling. https://www.coursera.org/learn/robotics-foundations-robot-modelling

De Santis, A., Siciliano, B., De Luca, A., & Bicchi, A. (2008). An atlas of physical human–robot interaction. Mechanism and Machine Theory, 43(3), 253–270. https://doi.org/10.1016/j.mechmachtheory.2007.03.003

Devin, C., Abbeel, P., Darrell, T., & Levine, S. (2018). Deep object-centric representations for generalizable robot learning. IEEE International Conference on Robotics and Automation (ICRA). Brisbane, QLD, Australia. https://doi.org/10.1109/ICRA.2018.8461196

Dormido, S. (2006). Advanced PID control. IEEE Control Systems Magazine, 26(1), 98–101. https://doi.org/10.1109/MCS.2006.1580160

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. IEEE Robotics & Automation Magazine, 13(2), 99–110. https://doi.org/10.1109/MRA.2006.1638022

Falconi, R., Melchiorri, C., Biagiotti, L., & Macchelli, A. (2006). Roboticad: A Matlab© toolbox for robot manipulators. IFAC Proceedings Volumes, 39(15), 431–436. https://doi.org/10.3182/20060906-3-IT-1901.00073

Featherstone, R. (2008). Rigid body dynamics algorithms. Springer. https://doi.org/10.1007/978-1-4899-7560-7

Ghazal, M., Al-Ghadhanfari, M. y Waisi, N. (2024). Simulation of autonomous navigation of turtlebot robot system based on robot operating system. Bulletin of Electrical Engineering and Informatics, 13(2), 1238-1244. https://doi.org/10.11591/eei.v13i2.6419

Hirai, K., Hirose, M., Haikawa, Y., & Takenaka, T. (1998). The development of Honda humanoid robot. Proceedings - IEEE International Conference on Robotics and Automation, 2, 1321–1326. Leuven, Belgium. https://doi.org/10.1109/ROBOT.1998.677288

Hu, F., Qu, F., Li, J., & Zhao, H. (2022). Real-time research based on ARM-based robot EtherCAT master system. International Symposium on Control Engineering and Robotics. Changsha, China. https://doi.org/10.1109/ISCER55570.2022.00009

Ijspeert, A., Nakanishi, J., & Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. IEEE International Conference on Robotics and Automation (ICRA). Washington, DC, USA. https://doi.org/10.1109/ROBOT.2002.1014739

ISO. (2025). ISO/TS 15066:2016 robots and robotic devices—Collaborative robots. https://www.iso.org/standard/62996.html

Javaid, M., Haleem, A., Singh, R., Rab, S., & Suman, R. (2022). Significant applications of Cobots in the field of manufacturing, 2, 222–233. https://doi.org/10.1016/j.cogr.2022.10.001

Kelly, J., & Sukhatme, G. (2011). Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. The International Journal of Robotics Research, 30(1), 56–79. https://doi.org/10.1177/0278364910382802

Kober, J., Bagnell, J., & Peters, J. (2001). A historical perspective of robotics toward the future. Journal of Robotics and Mechatronics, 13(3), 299–313. https://doi.org/10.20965/jrm.2001.p0299

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238–1274. https://doi.org/10.1177/0278364913495721

Kofer, D., Bergner, C., Deuerlein, C., Schmidt-Vollus, R., & Heß, P. (2021). Human–robot-collaboration: Innovative processes, from research to series standard. Procedia CIRP, 97, 98–103. https://doi.org/10.1016/j.procir.2020.09.185

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. IEEE/RSJ International Conference on Intelligent Robots and Systems. Sendai, Japan. https://doi.org/10.1109/IROS.2004.1389727

Li, P., An, Z., Abrar, S., & Zhou, L. (2025). Large language models for multi-robot systems: A survey. arXiv. https://arxiv.org/pdf/2502.03814v1

Liu, Z., Liu, Q., Xu, W., Wang, L., & Zhou, Z. (2022). Robot learning towards smart robotic manufacturing: A review. Robotics and Computer-Integrated Manufacturing, 77, 102360. https://doi.org/10.1016/j.rcim.2022.102360

Lynch, K., & Park, F. C. (2017). Modern robotics: Mechanics, planning, and control. Cambridge University Press. https://api.semanticscholar.org/CorpusID:69542521

Macchelli, A., & Melchiorri, C. (2002). A real-time control system for industrial robots and control applications based on real-time Linux. IFAC Proceedings Volumes, 3(1), 55–60. https://doi.org/10.3182/20020721-6-ES-1901.00821

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm607. https://doi.org/10.1126/scirobotics.abm607

Macenski, S., Moore, T., Lu, D., Merzlyakov, A. y Ferguson, M. (2023). From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2. Robotics and Autonomous Systems, 168, 104493. https://doi.org/10.1016/j.robot.2023.104493

Merlet, J. P. (2006). Parallel robots. Springer.

Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., & Davison, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. 10th IEEE International Symposium on Mixed and Augmented Reality. Basel, Switzerland. https://doi.org/10.1109/ISMAR.2011.6092378

Peng, X., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-real transfer of robotic control policies. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Brisbane, QLD, Australia. https://doi.org/10.1109/IROS.2018.8460528

Pratt, G., & Manzo, J. (2013). The DARPA Robotics Challenge [Competitions]. IEEE Robotics & Automation Magazine, 20(2), 7–12. https://doi.org/10.1109/MRA.2013.2255424

Pratt, G., & Williamson, M. (1995). Series elastic actuators. Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1, 399–406. Pittsburgh, PA, USA. https://doi.org/10.1109/IROS.1995.525827

Quigley, M., Conley, K., Gerkey, B., & Faust, J. (2009). ROS: An open-source robot operating system. ICRA Workshop on Open Source Software, 3(3.2), 5. http://lars.mec.ua.pt/public/LAR%20Projects/BinPicking/2016_RodrigoSalgueiro/LIB/ROS/icraoss09-ROS.pdf

Rankin, A., Maimone, M., Biesiadecki, J., Patel, N., Levine, D., & Toupet, O. (2021). Mars curiosity rover mobility trends during the first 7 years. Journal of Field Robotics, 38(5), 759–800. https://doi.org/10.1002/rob.22011

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv. https://doi.org/10.48550/arXiv.1707.06347

Schaal, S. (1999). Is imitation learning the route to humanoid robots? Trends in Cognitive Sciences, 3(6), 233–242. https://doi.org/10.1016/S1364-6613(99)01327-3

Sentis, L., & Khatib, O. (2005). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. International Journal of Humanoid Robotics, 2(4), 505–518. https://doi.org/10.1142/S0219843605000594

Siciliano, B., & Khatib, O. (Eds.). (2016). Springer handbook of robotics. Springer. https://doi.org/10.1007/978-3-319-32552-1

Srinivas, S., Ramachandiran, S., & Rajendran, S. (2022). Autonomous robot-driven deliveries: A review of recent developments and future directions. Transportation Research Part E: Logistics and Transportation Review, 165, 102834. https://doi.org/10.1016/j.tre.2022.102834

Spong, M., Hutchinson, S., & Vidyasagar, M. (2019). Robot modeling and control. Wiley.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press. https://mitpress.mit.edu/9780262039246/reinforcement-learning/

Sucan, I., Moll, M., & Kavraki, L. (2012). The open motion planning library. IEEE Robotics & Automation Magazine, 19(4), 72–82. https://doi.org/10.1109/MRA.2012.2205651

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics. MIT Press. https://mitpress.mit.edu/9780262201629/probabilistic-robotics/

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. arXiv. https://doi.org/10.48550/arXiv.1703.06907

Tomo, J., Domjan, M., & Orehovacki, T. (2024). Intelligent robotics—A systematic review of emerging technologies and trends. Electronics, 13(3), 542. https://doi.org/10.3390/electronics13030542

Van den Berg, J., Snoeyink, J., Lin, M., & Manocha, D. (2009). Centralized path planning for multiple robots: Optimal decoupling into sequential plans. Robotics: Science and Systems V, University of Washington. Seattle, USA. https://doi.org/10.15607/RSS.2009.V.018

Vukobratović, M., & Stepanenko, J. (1972). On the stability of anthropomorphic systems. Mathematical Biosciences, 15(1), 1–37. https://doi.org/10.1016/0025-5564(72)90061-2

Zhang, D., Wang, J., Jing, Y., & Shen, A. (2024). The impact of robotics on STEM education: Facilitating cognitive and interdisciplinary advancements. Applied and Computational Engineering, 69(1), 7–12. https://doi.org/10.54254/2755-2721/69/20241433

Zhou, Z., Song, J., Xie, X., Shu, Z., Ma, L., Liu, D., Yin, J., & See, S. (2024). Towards building AI-CPS with NVIDIA Isaac Sim: An industrial benchmark and case study for robotics manipulation. ICSE-SEIP '24: Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, 263–274. https://doi.org/10.1145/3639477.3639740