IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 May 2024, accepted 5 July 2024, date of publication 23 July 2024, date of current version 5 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3432741

RESEARCH ARTICLE

A Deep Reinforcement Learning Framework for Control of Robotic Manipulators in Simulated Environments

CARLOS CALDERÓN-CORDOVA[1], (Senior Member, IEEE), ROGER SARANGO[1], DARWIN CASTILLO[2,3], AND VASUDEVAN LAKSHMINARAYANAN[3,4], (Senior Member, IEEE)

[1]Department of Computer Science and Electronics, Universidad Técnica Particular de Loja, Loja 1101608, Ecuador
[2]Department of Chemistry, Universidad Técnica Particular de Loja, Loja 1101608, Ecuador
[3]Theoretical and Experimental Epistemology Laboratory, School of Optometry and Vision Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
[4]Departments of Physics, Electrical and Computer Engineering and Systems Design Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Carlos Calderón-Cordova (cacalderon@utpl.edu.ec)

This work was supported by the Universidad Técnica Particular de Loja under Grant PROY_ARTIC_CE_2022_3667.

ABSTRACT Industrial robots play a crucial role in a wide range of industrial processes. Because of the complexity of the work environment in which these systems are deployed, more robust and accurate control methods are required. Deep reinforcement learning (DRL) is a comprehensive approach that does not require an initial source of structured data for its learning process. Instead, DRL generates its own data based on its experiences within a work environment. To generate its own data, DRL requires integration with virtualized environments provided by simulators. These tools must include scenarios in industrial contexts and allow integration with machine learning tools, among other capabilities. Currently, several platforms support the simulation of various scenarios and generation of synthetic data, thus facilitating the development of end-to-end systems based on artificial intelligence, such as DRL. This article presents an extensive review of the software tools applied to DRL-based control systems for robotic manipulators. The selection of these tools is based on their efficiency, scalability, and compatibility with contemporary industrial standards and offers insights into their practical application in real-world scenarios. This study established a complete framework for designing and developing control systems for robotic manipulators using end-to-end DRL. This framework outlines the tools in detail, including simulators, APIs, libraries, and methods, and their interactions with each other. Additionally, it discusses the practical implications of this framework, highlighting its potential applications in industry and addressing some of the challenges and limitations encountered in applying DRL to complex robotic systems.

INDEX TERMS Robotics manipulator, control, deep reinforcement learning, automatic control, framework, simulation environments, simulation, industry 4.0.

I. INTRODUCTION

Robots are designed to assist or replace human operators in repetitive/dangerous tasks in which human physical limitations and/or extreme environments are present. Continuous developments in mechanics, sensing technology, intelligent control, improved sensors, latest generation processors, and other aspects have enabled improvements in the performance and autonomous capabilities of robotic systems [1]. Robotic manipulators are extensively used in fields such as industrial manufacturing, assembly, packaging, transportation, surgery, medical treatment, military, space exploration, amongst others. In addition, these robotic systems guarantee speed, efficiency, and cost reduction in industrial production lines. Although they are widely used in industry, robots require adaptability to different dynamic changes

The associate editor coordinating the review of this manuscript and approving it for publication was Hongli Dong.

in industrial processes; thus, a self-learning capability is required [2].

Currently, robots can be grouped into several categories: degrees of freedom (DOF), function, generation, structure and mobility. The use of robotic manipulators is increasing in various industrial applications, which introduces new challenges that must be overcome to fulfill the tasks assigned in each case study [3]. A robot manipulator is a nonlinear coupled multiple-input multiple-output (MIMO) system. The uncertainty of the system has two aspects: structural and nonstructural. The first type of uncertainty is due to the unmodeled dynamics of the system, dynamic and static friction, perturbation of system parameters, etc. The second type of uncertainty is caused by external environmental noise, measurement errors, and signal sampling delays, among others.

Control of a robotic manipulator can be performed in two ways: an open loop, which depends on structured environments that are calibrated (no external sensors); the robot is programmed to follow a series of positions, and if a component moves a little, the system must be recalibrated. The closed loop with feedback uses sensors, such as force, vision, and depth sensors, that monitor the robot axes, end-effector, and components, such as velocity or position. Feedback allows the comparison of the obtained information with the desired information to achieve optimal behavior [4]. With advances in artificial intelligence (AI), software-based end-to-end control solutions have been developed. The aim of this control is to select the best hardware that allows the integration of robust software and the optimal algorithms that enable the best performance of the robotic system.

In [5], researchers designed and compared two types of controllers for trajectory tracking of a rigid robotic manipulator (PUMA560). The first controller is a Fractional Order Proportional Integral Derivative (FOPID) tuned using a genetic algorithm (GA), and the second controller is a classical PID. The objective of this control was to track a quintic polynomial trajectory. The results of the comparisons highlight the better performance of FOPID over PID when both control systems were simulated using Simulink/MATLAB 2013.

In [6], the authors proposed an Actor-Critic-based Deep Reinforcement Learning method to solve the classical trajectory-planning problem of the UR5 robotic arm. The proposed DRL method is not based on any model, which guarantees that the joint angle is within the allowed range each time it reaches the target point. A standard path-planning method was implemented in the ROS, and the simulations were validated using CoppeliaSim.

The applicability of DRL has also been extended to construction tasks [7]. The researcher explored the possibility of automating excavation tasks using the Qt-Opt algorithm, which is a variant of Q-learning for continuous spaces. The input to the network were depth images. The experiments were implemented using a simulator developed by Komatsu Ltd. The results of this study demonstrate that the policy obtained by Qt-Opt outperforms SAC and TD3 in the proposed task.

This study provides a guide for the selection of software tools for the application of traditional control methods and artificial intelligence in robotic manipulators implemented in simulated environments. The tools presented are categorized according to the stage of development of the project, and tables of characteristics that summarize the information on each tool are provided. The goal of this study is to provide researchers and students with a practical reference for the selection of methods, algorithms, and software for the simulation and evaluation of robotic manipulators.

The contribution of this paper is as follows:

- Establish a framework for applying deep reinforcement learning techniques in semi-photorealistic environment simulators for software control of robotic manipulators.
- Present and describe the main software tools: Simulators, libraries, algorithms, control methods and DRL-based frameworks for robotic manipulators.
- From this paper, researchers in the robotics field can obtain an idea of how to approach their case study from the perspective of using DRL, simulators and specific applications.
- Recent research provides outstanding results and innovative perspectives for the field, and includes a sufficient technical mathematical level to allow readers to understand the essence of the research. Robotics research publications were selected from the years 2016-2023.
- Several examples of the integration of selected software tools from the selected research papers are provided.

Several research projects have been carried out in the field of robotics, which compile information about simulators, DRL frameworks, and the evaluation of algorithms for the performance of specific tasks in industry and academia.

In this paper, a practical guide for the development of DRL control applications focused on robotic manipulators is presented, and a compilation of software tools that allow the evaluation of several RL algorithms in different simulations on the platforms is provided. Detailed information on the platforms and libraries is presented, as well as a description of the integration of these tools into a single control architecture with a modern DRL-based approach. In addition, related research on DRL in robotics is presented in different sections. Finally, an example of software implementation using the tools and methods described in this paper is provided.

This paper is organized as follows. In Section II, the framework developed for control systems in robotic manipulators is presented, which is based on a series of sequential phases that permit the establishment of a classification method and selection of software tools. Section III discusses the requirements and needs of robotic manipulator applications. Evidently, the use of robotic software depends on the research area, environment, and applicability for implementation in physical systems. In Section IV, a systematic literature review (SLR) is presented to identify the most relevant information for robotic manipulators. In Section V, some traditional

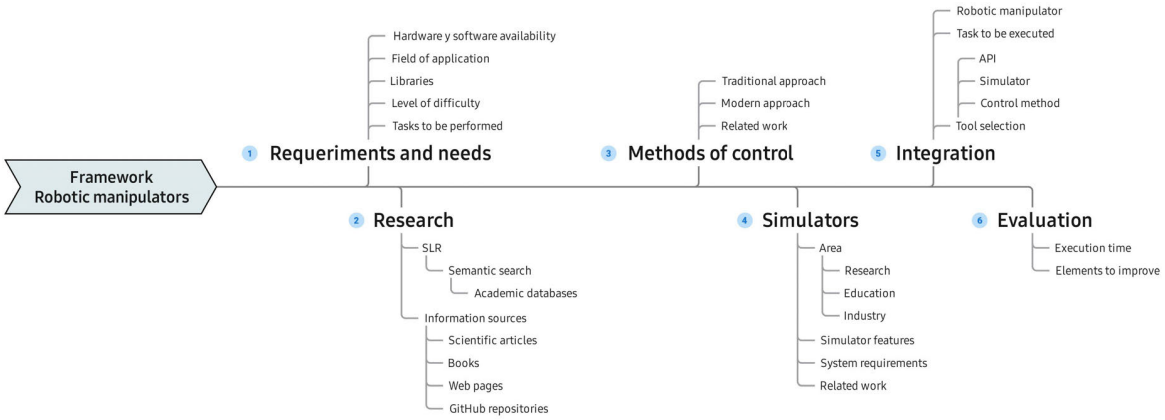

FIGURE 1. Sequential diagram of the framework.

control methods for robotic manipulators, such as SMC control, PID, FOPID, among others, are detailed. Various DRL frameworks with RL libraries, DRL taxonomy, cloud platforms, and dedicated hardware for the training and evaluation of DRL models are also presented. Section VI provides details on the characteristics of simulators used for robotic manipulators that integrate traditional control functionalities, DRL algorithms, APIs, and connection functions with other software tools. Section VII details the base architecture for the integration of the tools (simulator, DRL library, and robot) into a complete software-control system. Section VIII presents an example application for robotic manipulators and describes the integration of each software tool described in previous sections.

II. WORK SCHEME FOR ROBOTIC MANIPULATORS

A scheme that defines the main development phases for the research, development, and software implementation of robotic manipulator control is proposed. Each phase provides details of the tools, methods, and libraries available in robotics literature. Figure 1 presents the framework of this process.

The framework was specifically developed to integrate DRL software tools into a control system for robotic manipulators, with implementation in a simulation environment. This framework comprises of several phases: Requirements and Needs, Research, Control Methods, Simulators, Integration, and Evaluation. Each development phase describes various elements that must be considered for implementation in the robotic system, provides a detailed description of the tools used for robotic manipulators, and provides summary tables presenting relevant information from related work using the tools described in the corresponding section. The objective of this framework is to present a sequential method that provides recommendations and examples of how to implement control architecture in software using a modern DRL approach in robotic manipulators, whose applications range from academia to industry.

This DRL framework for robotic manipulation provides an organized structure and a set of software tools, such as libraries, simulators, methods, and application examples, for developing and applying DRL algorithms in robotic manipulation environments. There are several advantages and disadvantages to consider when developing and applying the technologies mentioned in this framework. The advantages of the proposed framework are as follows:

- Establishment of a clear structure for the phases of DRL application development in robotics (research and implementation). This offers detailed information on the software tools according to each phase (methods, simulators, integration, etc.).
- Offers features and technical details of software tools with modern approaches for motion control in robotics.
- Present and describe elements of DRL control architectures using these software tools; these details are not mentioned in most similar studies.
- Provide details on basic concepts to the application examples implemented in the simulation. This allows the user to have a broad view of the path to follow to develop a DRL project in robotics.

Some disadvantages exist, as follows:

- There is a generalization of its use for situations and application environments, and it should be considered that each task to be performed by a robot has different requirements, and its design must be based on the problem posed, which is not addressed in this framework.
- Although a guide for the development of DRL control for robotics is offered, researchers must consider that DRL models can learn unexpected or undesirable behaviors if the reward function is not well designed, or if the training environment does not capture all possible situations that the robot may encounter in the real world.
- This framework incorporates current software (2023-2024). Upgrading software is a common and necessary practice for improving the performance and adding new features to applications. However, these upgrades can also have significant disadvantages for robotics DRL research, such as incompatibility with

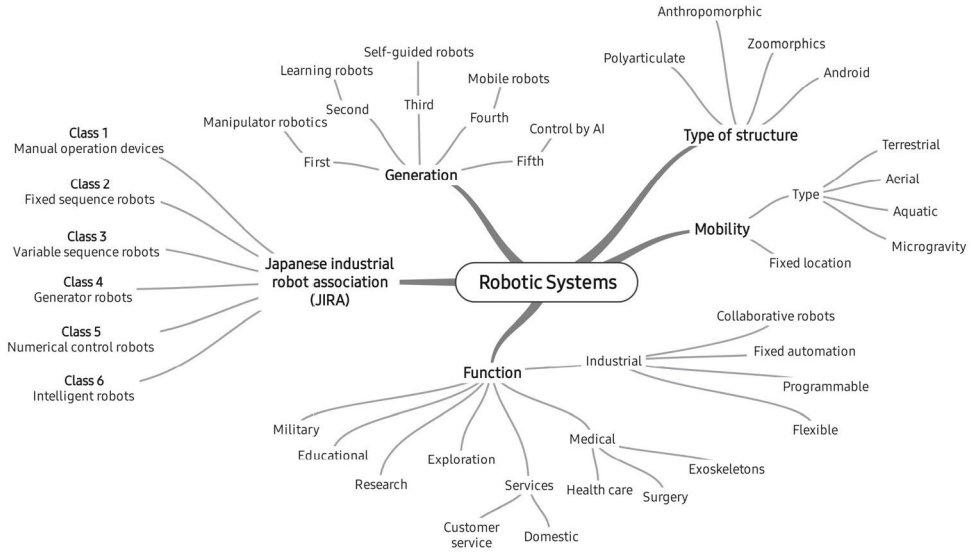

FIGURE 2. Classification of robotic systems.

previous versions, risk of bugs and errors, cost of training and learning new versions of the software, system stability issues, and loss of customization of work performed. Any software upgrade should be considered when reviewing the content of this framework.

In the following sections, each phase of the proposed framework for robotic manipulator control is presented and described, providing details of the techniques, libraries, and related work.

III. REQUIREMENTS AND NEEDS

Several options exist for simulation, control methodologies, and software tools that offer ease of integration, implementation, and deployment of robotic systems in simulated environments. These can then be physically implemented with the help of APIs and multilanguage platforms. The selection of each system component depends on the constraints, limitations, and scope of the research project. To understand the field of application of a robotic system to be implemented, several aspects must be considered, such as the type of structure, functions to be performed and mobility of the robot. Figure 2 summarizes the classification of robotic systems considering some of the approaches described below.

There are several professional associations dedicated to advances in industrial robotics, which establish classifications, definitions, and standards for robotic systems, some of which are as follows:

- Robotics Industries Association (RIA): The North American association is dedicated to promoting industrial robotics and automation through education, research and development of international standards [8].
- Japan Industrial Robot Association (JIRA): This association is dedicated to the development of industrial robotics. JARA is one of the oldest associations dedicated to research and development of robotics in collaboration with governments and other organizations to promote industrial robotics [9].
- European Robotics Association (euRobotics): Founded in 2012, this association represents the robotics and automation industry in Europe. It is dedicated to the research, development and standardization of robotics in industry [10].

Robotics has developed over several generations. Each generation is driven by the technological advances that have enabled new functionalities. These are described below.

- First generation: Developed in the 1950s, it was characterized by the use of simple mechanical and electrical systems, and the tasks were programmed sequentially in open-loop control systems. These robots have been used in material manipulation applications.
- Second generation: Starting in the 1960s, electronics and computers were introduced. Robots were more specialized for welding, assembly, and object manipulation tasks.
- Third generation: Developed in the 1980s, artificial vision systems and programmable controllers have been introduced. The tasks of these robots were more complex.
- Fourth generation: Developed in the 2000s, artificial intelligence systems and the application of ML techniques were introduced in robotics. Robots learn from their environment, interact with humans, and are capable of autonomous navigation.
- Fifth Generation: These are collaborative robots that work safely and efficiently with humans. Robots adjust their behavior to avoid collisions and accidents in the presence of humans in the environment.

In addition, there are several types of robot structures that vary according to the tasks to be performed in various fields

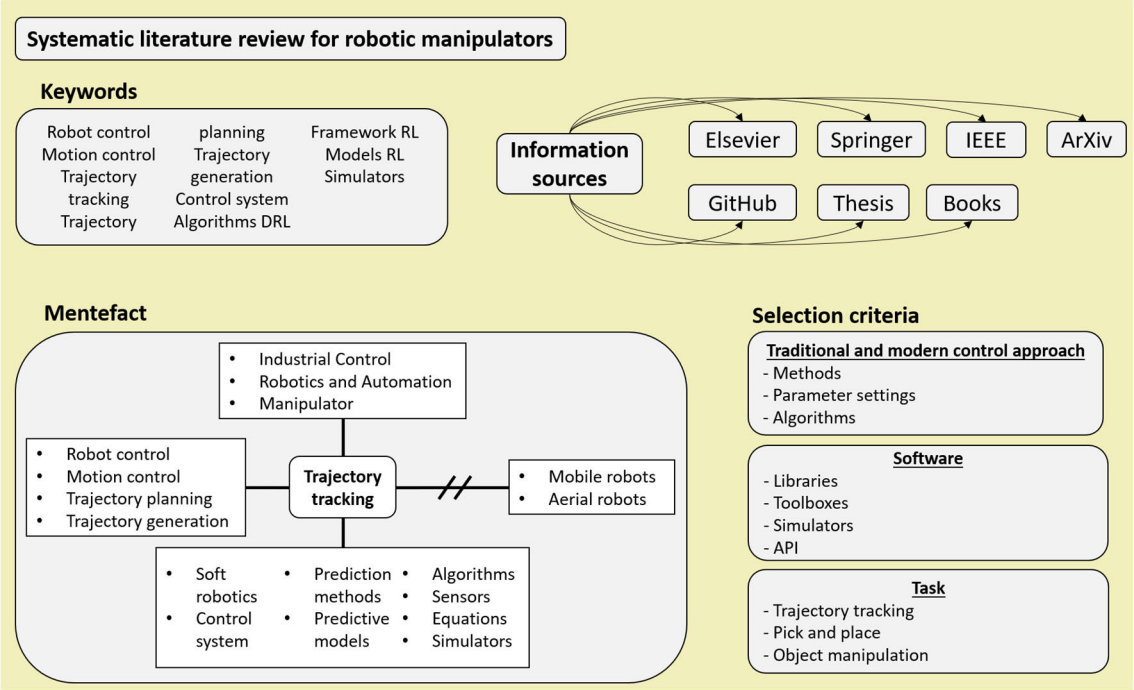

FIGURE 3. SLR data sheet for robotic manipulators.

of application, and have different characteristics in terms of flexibility, precision, speed, and load capacity. Some of the most common robot structures are Cartesian or rectangular coordinate robots, SCARA (Selective Compliance Assembly Robot Arm) robots, articulated robots, cylindrical robots, parallel robots, hybrid robots, polyarticulated robots, zoomorphic robots, and androids.

There are different robot designs; therefore, its mobility is a function of the application to be performed. Two subgroups were described: mobile and fixed. The first consists of robots with structures such as limbs or wheels to move in different environments. The second is located in fixed workstations, and here are the robotic manipulators perform repetitive tasks in industrial processes systems, which have also been developed for application of robotic manipulators in mobile systems in aerial environments [11], aquatic [12], terrestrial [13], and tasks performed in space [14].

In robotics, the area of manipulation is extensive, and different methodologies and algorithms have been explored for the design of motion planning and control, as well as the fabrication of robotic arms and components. There are some relevant points for robotic manipulation, including picking and placing objects [15], assembly, handling deformable objects, trajectory tracking, path planning, human-robot interaction, and collision prevention.

IV. SYSTEMATIC LITERATURE REVIEW

The systematic literature review process permitted the search and filtering of information in the databases relevant to the field of study. In the initial phase, a conceptual map is created, which facilitates the development of the thesaurus for the advanced search criteria of inclusion and exclusion of parameters specific to the topic. This method produces a list of documents of interest or great impact related to the research questions and information relevant to the case study [16].

Now, the semantic structure for the search of articles, obtained from the conceptual map and the scientific thesaurus [17] in the SCOPUS academic database, is presented: TITLE-ABS-KEY (“robotics” AND “arm” AND “manipulator” AND “trajectory” AND “tracking”) AND (“industrial”) AND (“model” OR “control”) AND NOT (“mobile” OR “aerial”). This search resulted in a list of 229 significant articles related to (robotic manipulators) published between 2016-2023. Figure 3 shows the research structure of the robotic manipulators.

Some of the prominent publishers in this field are: IEEE, Elsevier, MDPI, Springer, and some articles were also selected from ArXiv. Additionally, academic theses and GitHub repositories containing studies related to robotic manipulation using simulators and deep reinforcement learning were considered. This data sheet provides details on the conceptual map setup for semantic searches in an academic database. It also shows the keywords used to create the conceptual map.

Figure 3 also shows the main selection criteria of the articles, focusing on the categorization of key elements for the implementation of robotic manipulators: control methods (traditional and modern approaches), use of simulators for research, DRL in robotic applications and industrial applications (trajectory tracking), and development of API for connection between tools. Finally, the main

sources of the information obtained in this study are discussed.

V. CONTROL METHODS

Owing to the physical nature of robotic manipulators, which are nonlinear multivariable mechanical systems with time-varying constraints and uncertainties, modeling of robust controllers is required. In the industrial field, several studies have been conducted on the traditional and modern control methods for robotic manipulators. This section describes the main control methods used, along with examples of the applications and software tools used to design the proposed controllers.

A. TRADITIONAL APPROACH

Although modern control theory has made great progress, classical algorithms such as PID are still used in robotic manipulator controllers because of the simplicity of the design process. Some tuning methods for these types of controllers are based on trial and error based on assumptions about the plant and desired output, obtaining some analytical or graphical characteristics of the process, which are then used to configure the controller [18]. Controllers such as PID are suitable for most setpoint regulation problems, that is, in robotics applications, such as path-following, welding, and laser cutting.

Controllers must consider the dynamic model of the manipulator [19]. To control a physical system using a digital controller, it is necessary to receive the measured signals from the system, process the signals, and then send control signals to the actuator that performs the control action, these control signals must be translated using a digital-to-analog converter (DAC). Most physical systems have process and analog sensors; therefore, it is necessary to use an analog-to-digital converter (ADC) that allows the translation of analog signals to the language of the digital controller. In Figure 4, a classical closed-loop control system with negative feedback is presented. Figure 5 illustrates the physical implementation of a discrete controller in various devices (PC, PLC, and microcontroller) that control the process of the plant of a physical system. The sensor serves as feedback for the control system to monitor the controlled variables [20].

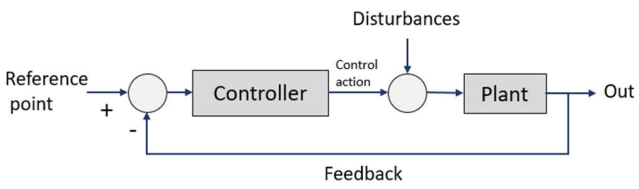

FIGURE 4. Closed loop control system.

Depending on the plant (continuous or discrete), converters were used to couple the input (analog) and output (digital) signals of the controller.

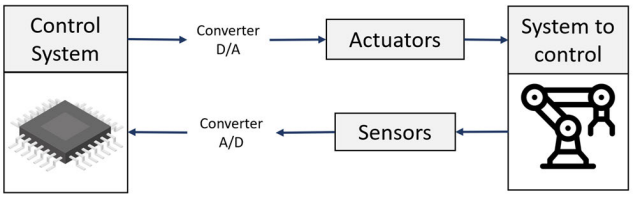

FIGURE 5. Closed loop control system.

1) INTEGER ORDER CONTROLLERS

Integer order controllers (P, PI, PD, and PID) are the most commonly used control strategies in control engineering. The most common controller is the three-term controller (PID), in which the proportional term incorporates appropriate proportional changes for the error. The integral term examines the process variable over time, which corrects the output by reducing the process variable offset. The derivative term monitors the rate of change of the process variable and therefore changes the output when there are unusual variations. The user must adjust each parameter of the three control functions to obtain the desired performance of the process to be controlled [18]. A comprehensive and summarized set of tuning rules for PI and PID controllers was presented in [21]. Some studies related to integer-order controllers for robotic manipulators are presented in Table 1.

2) FRACTIONAL ORDER CONTROLLERS

The use of fractional-order derivatives/integrals presents an alternative strategy for solving robust control problems, applicable to robotic manipulators. Modeling and controlling systems using fractional-order derivatives and integrals is an alternative strategy for effectively solving many control problems. In particular, it has been shown that the use of fractional-order concepts in the modeling and control of robotic manipulators can improve control accuracy and efficiency [22].

In [22], a review of control strategies and fractional-order modeling applied to robotic manipulators is presented, which includes modeling, fractional-order definitions, objective functions, type of fractional-order modeling, fractional-order control, controller parameters, approximation techniques, controller comparison, tuning techniques, simulation tools, and types of validation.

In addition to the software that facilitates the design of fractional-order controllers, there are sets of tuning rules for FOPID controllers. These rules are quadratic and require the same plant response time data as the Ziegler-Nichols tuning rules for integer-order PID controllers [23].

Fractional control applied to robotic manipulators offers advantages, such as increased robustness and adaptability to changes in system parameters and improved trajectory tracking capability, among others. However, it also presents challenges in terms of the complexity of the controller design and implementation, as well as the need for the use of specialized tools and techniques for its development

TABLE 1. Papers related to integer order control.

Reference	Manipulator robot	Task	Controller	Comparison	Software
[24]	CNC Machine	Trajectory tracking	PD-SMC	PD, SMC	MATLAB
[25]	3-DoF	Route planning	PD-SMC	PI, SMC	MATLAB
[26]	6-DoF	Trajectory tracking	PD-SMC	PD, SMC	-
[27]	6-DoF	Trajectory tracking	PD-PSO	SMC, PSO-IK	MATLAB/Robotics Toolbox

and validation. For the control of robotic manipulators, there are several control strategies that are combined with other methods to improve the robustness, performance, and stability of the system. In the control literature, several fractional control strategies are modified versions of these methods, as shown in Tables 1 and 2.

Toolboxes permit the modeling and design of fractional-order systems such as FOMCON [34]. This tool is integrated into MATLAB and presents the following modules with important functions for fractional-order systems [35]:

- Core module: FO system analysis (FOTF kernel and utilities).
- Identification module (system identification based on experimental input/output data in the time and frequency domains).
- Control module (design, tuning and optimization tools for the FOPID controller as additional features).
- Implementation module (continuous and discrete time approximations and the implementation of the corresponding analog and digital filters).

3) SLIDING MODE CONTROL

Sliding mode control is a type of variable structure control. This system was designed to drive and constrain the system state to be within the vicinity of the switching function. This control approach has two advantages: first, the dynamic behavior of the system is adapted by the switching function; second, the closed control loop becomes insensitive to uncertainty, and this feature converts it into a robust control [36].

This control structure has been extensively studied in the field of robotics owing to its characteristics such as robustness, insensitivity to changes in the physical model parameters, and external disturbances (see Table 3). The sliding-mode design approach is divided into two stages: the first reaching stage, in which the system state tends toward the sliding surface in a finite time from any initial state. In the second sliding stage the system state maintains the sliding mode along the surface under control action [37].

Key features and components of Sliding Mode Control are as follows: (1) Sliding Surface: The sliding surface is a hyperplane or a subspace in the state space. (2) Switching control law: Sliding mode control involves the use of a discontinuous control law. The control input was switched between the different modes to maintain the system on the sliding surface. (3) Chattering: This term refers to the high-frequency switching behavior of the control signal near the sliding surface. Although chattering is inherent to sliding mode control, efforts have been made to minimize its effects. (4) Robustness: The control system is designed to operate effectively even in the presence of modeling errors or external disturbances [37].

An example implementation of this control structure is detailed in [36], which provides several examples of this control method and some variations using MATLAB.

B. MODERN APPROACH

In Industry 4.0 [40], several industrial processes include tasks such as production scheduling, assembly, support systems, production line operations, and path planning. Several of these tasks have been realized using digital systems and robots. The application of artificial intelligence has increased the efficiency of industrial process automation. Machine learning (ML) offers the possibility of automata learning and executing tasks. However, these learning algorithms require a significant amount of data.

Several learning methods can be applied to robotics, and each field has its own characteristics that must be considered to determine the best methodology. Deep reinforcement learning (DRL) is a learning option for robots in which the algorithm learns through interaction between the agent and the surrounding environment [41]. One of the most used libraries is the Gymnasium toolkit, which allows the training and evaluation of RL algorithms and is used with several simulators such as PyBullet, Webots, Nvidia Isaac, CARLA, Gazebo, and AirSim, among others, which allows agents to be trained with some reinforcement learning methods [42].

1) DEEP REINFORCEMENT LEARNING

This section provides the fundamental concepts of DRL control architecture and an overview of the different components that are part of modern approach methods. DRL is a field of machine learning whose goal is to create intelligent agents that can interact with the environment. When executing an action, a reward and/or penalty is given back to them depending on the agent's performance. In the process of trial and error, where different actions are explored and exploited, the agent improves learning to perform a task based on the accumulated reward.

- Environment: The environment is defined as the place where the agent interacts with elements in its area and acquires knowledge from its experience. In this environment, the agent receives information regarding its current situation, represented as the current state St,

TABLE 2. Papers related to fractional order control.

Reference	Manipulator robot	Task	Controller	Comparison	Software
[28]	3-DoF	Trajectory tracking	FOPID	IOPID	MSC-ADAMS/MATLAB
[29]	3-DoF	Trajectory tracking	FOPI, FOPID	-	MATLAB
[30]	3-DoF	Trajectory tracking	FOPID	IOPID	MATLAB
[31]	3-DoF	Trajectory tracking	FOPID-SMC	HFOPDSTSMC, GWO-HFOPDSMC, EGWO-HFOPDSMC	MATLAB
[5]	3-DoF	Trajectory tracking	FO-Fuzzy-PID	IOPID, Fuzzy-PID	Simulink/MATLAB
[32]	3-DoF	Trajectory tracking	FOPID-Fuzzy-PI+D	FPI+D	MATLAB
[33]	3-DoF	Trajectory tracking	FOPID-Fuzzy-PID	IO-Fuzzy-PID	MATLAB

TABLE 3. Papers related to sliding mode control.

Reference	Manipulator robot	Task	Controller	Comparison	Software
[37]	6-DoF	Trajectory tracking	SMC	CTC, SMC-ERL, SMC-IERL	MATLAB-CoppeliaSim
[38]	2-DoF	Trajectory tracking	SMC	SMC+RBF+EKF, SMC+DO, SPC+FFC	-
[39]	2-DoF	Trajectory tracking	SMC	-	Simulink-MATLAB

and receives a reward at each step of the interaction. This information was obtained through what are known as observations.

- Agent: Agent is an actor in the deep reinforcement learning problem. The agent continuously interacted with various environmental elements. Depending on the observation and reward received, the agent executes an action at each time step. According to the implemented algorithm, the policy is updated as a function of the accumulated reward.
- Policy: Policy is a function that determines an agent's behavior in a given environment. This function specifies the action that the agent should perform in each state of the environment to maximize its cumulative reward over time. This policy can be deterministic or stochastic. In deterministic policy, there is no uncertainty regarding the choice of action. In a stochastic policy, the action taken may be random with a probability distribution associated with the possible actions in each state.
- Value function: This is a function that assigns a value to each state or state-action pair, indicating the long-term value the agent expects to obtain if it is in that state or if it takes that action in that state. There are two types of value functions: (a) State Value Function: Assigns a value to a specific state, representing the expected utility if the agent starts in that state and follows a given policy. (b) Action Value Function: Assigns a value to a specific action performed in a state. During training, the agent adjusts these value functions to improve its ability to make optimal decisions in the environment.
- Reward: The reward function assigns a numerical value to each state-action transition; this represents a feedback signal indicating the desired outcome of an action in a given state. This function provides the agent with information on how much ''reward'' or ''penalty'' it expects to receive as a result of its action in a specific state of the environment. The reward function can be designed in different ways depending on the specific task and the agent's goals.
- Episode: An episode refers to a complete sequence of interactions between an agent and the environment, from the initiation to the completion of a specific task. During an episode, the agent makes sequential decisions, performs actions, and receives feedback from the environment through the rewards. During an episode, the agent learns and improves its policy based on feedback received from the environment through rewards and observations. After completing multiple episodes, the agent gradually adjusts its policy to maximize the accumulated reward over time.

With the fusion of RL and DL, Deep Reinforcement Learning (DRL) methods can handle perception and planning problems. With the application of DRL, no predefined training dataset is required, and there is no focus on encoding directions for the coordination of robotic manipulator joints. Experimental data are dynamically generated in an environment and used to train a deep neural network, establishing a control policy by learning through iterative parameter updates [1].

In this framework, the Deep Learning architecture (convolutional networks, recurrent networks, etc.) acts as a function approximator to handle high-dimensional data (images, video, time-series data) and approximates large states and action spaces. Some tasks involved in DRL processes include [43]:

- Exploration/exploitation: Trying new actions or performing new actions based on learned knowledge.
- Generalization: Capacity of the agent to adapt to a new environment and execute a task in a simulation or real environment.
- Policy search: Identification of states and actions for the agent to learn an optimal policy in decision making.

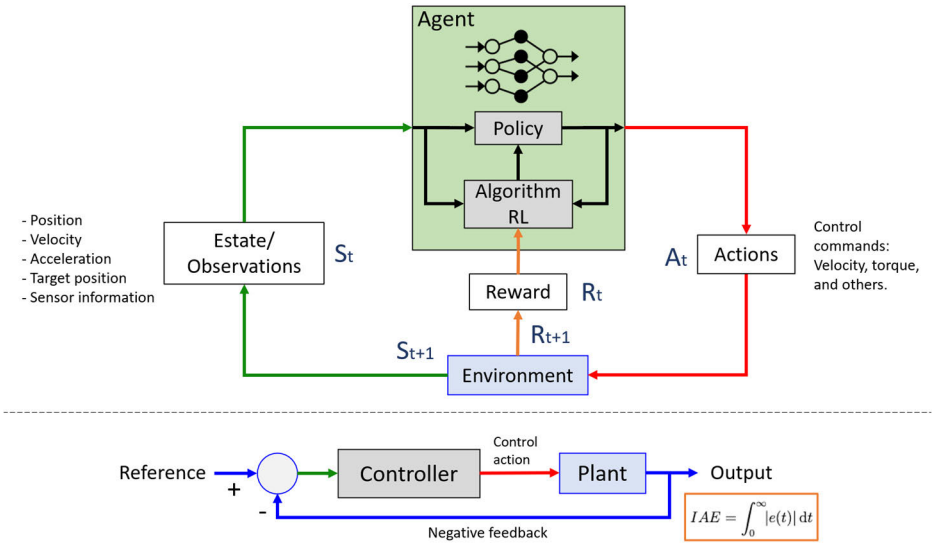

FIGURE 6. Schematic mapping of reinforcement learning for control systems.

- Finding catastrophic events: Finding and preventing events that can cause damage to optimize the policy.
- Handling overestimation: Using a maximal operation in learning Q produces overestimated values.
- Sample size reduction: In DRL, a large amount of data is required to perform training; therefore, it is necessary to reduce the sample size.
- Detection and prevention of overfitting: Overfitting occurs when the agent is sensitive to perturbations in the environment because of limited training data.
- Robust learning: Incorporating robustness into the DRL system provides better results for tasks assigned to agents.

The application areas of Deep Reinforcement Learning are extensive, ranging from education to autonomous vehicle navigations. The goal of RL is to train an agent that performs an optimal sequence of commands to perform the assigned task within an unknown environment. This agent is composed of two elements: the policy, which maps and selects the actions based on observations of the environment (deep neural network), and the RL algorithm, which updates the policy parameters based on the actions performed in the environment, observations, and the calculated reward [44].

In Figure 6, a comparison between traditional control systems and the DRL scheme is illustrated, and Table 4 summarizes the main elements of each scheme. In this case study, the agent receives observations that include information from the sensors, joint angles, and end-effector positions. The output of the control policy network and the RL algorithm (agent) is a set of actions, such as speed or torque, that control the actuators. When the robotic manipulator executes an action, a reward is generated and the algorithm is expected to find the best control strategy (policy) for the realization of a manipulation task.

TABLE 4. Comparison between RL schemes and control systems.

Deep Reinforcement Learning	Traditional control systems
Action	Control law/actions.
Environment	Plant, reference signals, disturbance, filters, DAC, DCA.
Observation	Any measurable value from the environment such as error signal.
Policy/RL Algorithm	PID controller, FOPID, SMC.
Reward	Performance metrics: cost functions.

Traditional control systems and deep learning approaches serve to solve control engineering problems but differ in their methodologies and approaches for applying them. DRL methods involve training agents to learn from their own experiences through trial and error without the need for explicit supervision or knowledge of the environment.

On the other hand, the classical approach to solving control problems uses methodologies consisting of designing controllers that regulate or follow a desired output in a system while minimizing performance criteria, such as steady-state error, overshoot, settling time, and performance indices (IAE, ISE, ITAE, and ITSE).

In summary, deep reinforcement learning (DRL) is appropriate for tackling situations where the system dynamics are complicated and difficult to represent by models, or when the environment is uncertain or variable. However, the use of DRL can be costly in terms of computing resources and requires a large volume of data to train the agent that can be provided by simulators. System control is a more suitable alternative for solving problems in which the system can be accurately modeled, and the control objectives are properly defined.

However, traditional control can be limited by the precision of the model and requires manual tuning of controller

parameters. Currently, there are libraries and toolboxes that allow software tuning of controller parameters.

The field of deep reinforcement learning includes many areas ranging from decision-making in games to robotic manipulation. DRL research focuses on complex interactions between an agent and the environment. DRL agents are based on complex architectures, in addition to the existence of a large number of available algorithms, and the increasing diversity in DRL research results in a difficult selection of software for subsequent projects that require these tools [45].

The effectiveness of control in any system depends largely on the method used to design the controller. Traditional methods, such as PI control and PID control, have been widely used in a variety of applications owing to their simple designs and implementations. FOPID controllers are extensions of the conventional PID controllers. These controllers offer the ability to adjust the derivative and integral gains through fractional terms, which provides a greater degree of freedom in the controller design. However, their implementation and tuning may be complex. It can be difficult to adapt these methods to systems with nonlinear or uncertain dynamics, leading to the development of more advanced approaches. Among these modern approaches, RL and DRL control have gained attention because of their ability to learn optimal control policies directly from interactions with the environment. These methods can handle highly complex and nonlinear systems, although they often require large amounts of data and present challenges in terms of interpretation and generalization. Currently, DRL-focused frameworks and simulation platforms allow the models to be trained and evaluated quickly and efficiently.

Table 5 provides a comparison between various control methods, covering both the traditional and modern approaches. Each control method is discussed in terms of its advantages, disadvantages, and the most appropriate use scenarios. The key features and situations in which they are the most appropriate for implementation are highlighted. This comparison provides an overview of the scope and limitations of each method, which can assist in selecting the most appropriate approach for various applications and control environments.

Deep Reinforcement Learning combines the perception capability of deep learning with the decision-making capability of reinforcement learning, resulting in agent learning to perform actions using images [46]. The core of almost all reinforcement learning is the Markov decision process (MDP), in contrast to unsupervised supervised learning, RL uses a feedback (reward) signal instead of a number of labeled samples [47].

This mathematical framework is used to model decision-making problems in which the outcomes are partially random and under the control of an agent. This model contains five main elements represented in the form of a tuple (S, A, R, P, γ), where S is the finite set of agent states and the environment. A, is a finite set of actions. P, is the model of the system, that is, the probability of transition from state s to state s' after performing an action. R, is the reward function obtained after performing an action; and $\gamma \epsilon [0, 1]$, is the discount rate for the cumulative reward, and the discount factor determines the importance of future rewards, if $\gamma = 0$ only considers the immediate (current) reward, on the contrary if $\gamma = 1$, the agent will try to obtain a higher reward value in the long period [48].

As shown in Figure 6, the agent observes its current state, performs an action, and receives its immediate reward along with its new state. The observed information (immediate reward and new state) is used to adjust the agent's policy, and this process is repeated until the agent's policy approaches optimality. Within the MDP process, some terms are defined as follows: (a) environment, an encapsulation of a set of rules as actions that an agent can realize, states of the environment, rewards, and penalties. (b) States: Is a set of states, places, and positions of the environment where the agent interacts with objects in the world. (c) Actions: The set of actions that the agent can perform in a given environment. (d) Reward is the value obtained by the agent during a state after making an action, indicating whether the current state is useful or not. It acts as feedback for the agent, and can be positive or negative [49].

During the deep reinforcement learning process, at each time step t, the agent is in state S_t, then performs an action in A_t according to the policy π mapping from states to actions. Each action affects the environment, and as a result, changes the state to S_{t+1}. The agent receives reward R_{t+1} at time step $t + 1$. The agent's goal is to determine the optimal policy π^* that maximizes the expected total reward G_t [50].

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

The optimal policy π^* can be expressed as:

$$\pi^* = arg \max_{\pi} E \left[\sum_{t} \gamma^t r\left(s_t, a_t\right) \right] \quad (2)$$

The policy can be deterministic or stochastic, depending on the task to be performed. In the deterministic policy, the agent performs an action determined by the policy of the gripper as the end-effector of the robotic manipulator. If there are two actions (open and close) in state, the policy chooses the 'close' action. In a statistical policy, the agent assigns a certain probability to each action and selects the action in a given state based on that probability. The policy chooses a probability of 0.75 for 'open' and 0.25 for 'close', so the agent is more likely to perform the action of 'open' the gripper.

The reward accumulated after policy π has two state forms: the state function $V_\pi(s)$ which defines the goodness of the state and the state-action value function $Q_\pi(s, a)$ which indicates how good it is to choose a certain action in a particular state.

$$V_\pi(s) = E\left[G_t | s_t = s\right] \quad (3)$$

$$Q_\pi(s) = E\left[G_t | s_t = s, a_t = a\right] \quad (4)$$

TABLE 5. Main characteristics of control methods.

Control Method	Advantages	Disadvantages	Usage scenarios
PI	It is simple to implement and offers a robust and stable response.	It does not include the overshoot in the response.	Systems of low complexity and moderate accuracy requirements.
PID	It improves transient response and offers better precision and controllability compared with PI control.	The controller is sensitive to noise and disturbances in the system.	Systems with slow and stable dynamics.
FOPID	Introduces fractional terms in the derivative gain and integral gain. Integral gain allows better adaptation of the controller to systems with non-linear or time-varying dynamics. With nonlinear or time-varying dynamics.	The implementation of a FOPID controller is more complex than that of a PID controller, which increases costs and computational resource requirements.	Applications where precision control and rapid response to changes in system conditions are required
SMC	It offers robustness to perturbations and model parameter variations.	It is sensitive to model uncertainties and high complexity in the slider design.	Non-linear systems are prone to perturbations.
RL	Adaptability to variable and complex environments.	It requires large amounts of data for training and does not guarantee convergence and stability.	It is used in systems with expensive or difficult-to-obtain feedback, as in robotics.
DRL	Suitable for complex and nonlinear problems, and handles high dimensionality problems and learning optimal control policies.	High computational complexity and need for hardware resources.	Complex tasks requiring adaptability and generalization to different environments, such as robotics and process control.

The action value for state $Q(s, a)$ is updated using the Bellman equation. This equation considers the immediate reward and the future estimate of the action value of the next state.

$$Q^*(s, a) = E_{s'}\left[r + \gamma max Q^*_{a'}(s', a')|s, a\right] \tag{5}$$

This equation is used to calculate the value function as a function of current policy. The Bellman equation represents the recursive relationship between the value of one state and the values of subsequent states, and is used to update and refine the value function as new samples of experience are received [51].

2) TAXONOMY OF DRL ALGORITHMS FOR ROBOTIC MANIPULATION

In this section, a summary of the taxonomy of the main Reinforcement Learning algorithms is presented. The classification is based on certain criteria: model-based, model-free, value-based, policy-based (in-policy and out-of-policy), and the combination of both (value-policy), Monte Carlo-based, and temporal differences. Some of these algorithms can be classified into different categories [52]. The graphs and tables in [53] summarize the properties of most commonly used reinforcement algorithms. In addition, details of the algorithms used in the execution of tasks related to robotic manipulation are provided.

1. Model-based Methods. In reinforcement learning, a ''model'' refers to a representation of the environment that allows an agent to predict the consequences of its actions. Model-based methods are characterized by predicting the elements of the environment by performing certain actions during a trial-and-error process to obtain samples of the state space, action space, and reward function through supervised learning. They can be divided into two categories: methods that work with a given model, in which the agent accesses the dynamics of the environment to perform the transition process, and methods that calculate the reward function (see Figure 7).

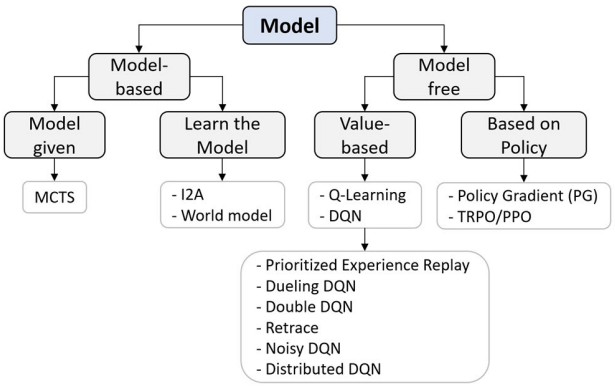

FIGURE 7. Model-based RL methods.

Model-based methods refer to approaches in which an agent constructs a representation of the environment and exploits this model to make decisions and learn the optimal policies. The advantage of model-based methods is that future states and rewards can be anticipated based on environmental dynamics. The disadvantage is that the model of the environment is usually not available, and the dynamics can be complex; therefore, its representation could be wrong [54]. Another approach to the learning process is not to model the environment but to determine the optimal policy based on the highest reward that can be obtained in the environment. The agent interacts directly with the environment and improves the performance based on the explored samples [55]. One of the drawbacks of these methods is the cost of exploring real environments, which is usually high owing to time consumption, equipment deterioration, and security risks.

2. Model-free Methods. From the ''Model-free'' category for policy optimization in deep reinforcement learning arise the value- and policy-based methods. Value-based methods involve optimization of the action-value function $Q\pi(s, a)$.

After optimization, the optimal value function is $Q\pi*(s, a) = max\alpha Q\pi(s, a)$, and the optimal policy can be derived from $\pi \approx argmax\pi Q\pi$. One disadvantage is that they cannot handle continuous action- space problems. The most common value-based algorithms are Q-learning, DQN, and its variants (see Figure 8) [56].

- Prioritized Experience Replay: Weights data based on TD error to improve learning efficiency.
- Dueling DQN: Improves network structure. The action-value function Q is decomposed into the state-value function V and advantage function A to improve the approximation capability.
- Double DQN: Chooses and evaluates actions using different parameters to solve the overestimation problem.
- Retrace: Revises the Q-value calculation method and reduces the variance in value estimation.
- Noisy DQN: Adds noise to the network parameters to increase the scanning increase exploration.
- Distributed DQN: Refines the Q-value estimate in the distribution estimation.

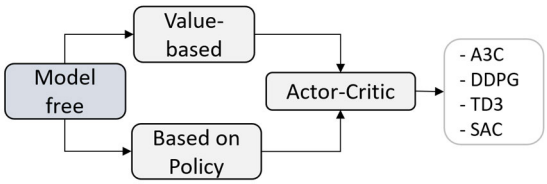

FIGURE 8. Model-free-based RL methods.

Policy-based methods have the advantages of simpler policy parameterization and better convergence, and are suitable for continuous or high-dimensional action spaces. Some common policy-based algorithms include Policy Gradient (PG), TRPO, and PPO. TRPO, and PPO restrict the update step based on PG to avoid policy collapse and to make the algorithm more stable [57].

3. Value-Policy-based Methods. This is a combination of Value and Policy-based methods, and this new method gives rise to an actor-critic framework. The actor-critical method combines the advantages of both methods, using value-based methods to learn a Q-function or a value function to improve sample efficiency, and using policy-based methods to learn the policy function. This method is appropriate for discrete or continuous action space [58]. Some Actor-Critical (AC) algorithms contribute to some improvements (see Figure 9), such as the following.

- A3C: Extends AC to asynchronous and parallel learning, perturbs the correlation between data, and improves the speed of data collection and training.
- DDPG: The target network is inherited from the DQN, and the actor is a deterministic policy.
- TD3: Introduces a trimmed double Q-learning mode and delayed policy update strategy to solve the overestimation problem.
- SAC: Entropy regularization is introduced in Q-value estimation to improve exploration.

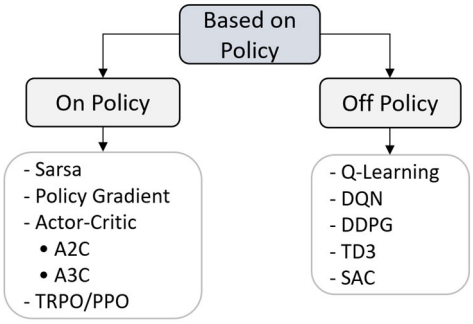

FIGURE 9. Policy-based RL methods.

4. On and Off RL Methods. In-policy methods evaluate and improve the policy used for decision making, requiring the agent to interact with the environment. The policy that interacts with the environment and that to be improved must be the same. Methods outside the policy evaluate or improve a policy other than that used to generate data, and this method is adjusted to the environment. The experiences of other agents interacting in the environment can be used to improve policies [46].

It is important to note that each reinforcement learning algorithm has its own strengths and weaknesses and that the choice of algorithm depends on the problem being addressed and the resources available.

There are a great variety of approaches based on RL for the control of robotic manipulators, and according to research conducted in the field of robotics, it has been shown that this approach is very effective in the control of robotic manipulators because of the complex and nonlinear nature of these systems. Table 6 lists the most relevant RL algorithms used in robotics research. The vast majority of these RL approaches are algorithms that determine the optimal policy based on the highest reward that the agent can obtain in the environment during its training; they do not require modeling the dynamics of the environment.

In several robotics studies, some researchers have evaluated robotic systems in the real world, but owing to the existing difficulties in the physical implementation of this type of system, many implementations have opted to perform tests in simulated environments. This prevents problems such as real-world noise, uncertainty of parameters of the physical systems, high costs, damage to the equipment, and the need to integrate safety measures for physical evaluation [59].

Although simulation environments provide a large volume of data for training RL agents, it is important to consider that they also present challenges. One of these challenges is the simulation gap between the simulated environment and the real world, as most simulators offer only semi-photorealistic scenes, making the input data from vision sensors not one hundred percent unreliable. This limitation can affect the performance of robotic systems in the real world. However, there are simulators such as Isaac Sim, which provide a more photorealistic environment and offer enhanced features in terms of the physics and dynamics of simulation elements,

TABLE 6. DRL algorithms.

N	DRL Algorithm	Action space	Model	Policy	Value
1	Q-Learning	Discrete	✗	✗	✓
2	Deep Q-Network (DQN)	Discrete	✗	✗	✓
3	Dueling DQN	Discrete	✗	✗	✓
4	Double DQN	Discrete	✗	✗	✓
5	Distributed DQN	Discrete	✗	✗	✓
6	Sarsa	Discrete	✗	✗	✓
7	Reinforce	Discrete/continue	✗	✓	✗
8	Actor-Critico	Discrete/continue	✗	✓	✗
9	Advantage Actor-Critic (A2C)	Discrete/continue	✗	✓	✓
10	Asynchronous Advantage Actor-Critic (A3C)	Discrete/continue	✗	✓	✓
11	Deep Deterministic Policy Gradient (DDPG)	Continue	✗	✗	✓
12	Twin Delayed DDPG (TD3)	Continue	✗	✗	✓
13	Soft Actor-Critic (SAC)	Discrete/continue	✗	✗	✓
14	Trust Region Policy Optimization (TRPO)	Discrete/continue	✗	✓	✗
15	Proximal Policy Optimization (PPO)	Discrete/continue	✗	✓	✗

TABLE 7. DRL algorithms.

Reference	Robot	Task	Algorithm	RL Approach	Train/Test	Hardware
[60]	ABB Yumi, Franka Emika	Drawer opening and pin in hole	PPO	On-Policy	Sim-to-Real	64 GPU Nvidia
[61]	7-DoF Kuka	Object grasping	QT-Opt	Off-Policy	End-to-End	ImageNet baseline
[62]	Kuka youBot	Object grasping	DQN	Value Function	Simulation	GPU Nvidia GTX 550
[63]	UR5, Kuka KR 16	Motion planning	PPO	On-Policy	Simulation	GPU NVIDIA RTX 3090
[64]	7-DoF Baxter	Trajectory tracking	SAC	Value Function	Simulation	CPU Intel i7-6700HQ
[65]	UR3	Trajectory tracking	DPPO, A3C, DDPG	Value-Policy	Simulation	GPU Nvidia Tesla K40

which are crucial for training AI models that exhibit better performance in executing real-world tasks. Therefore, it is important to carefully consider the simulator used for AI model training and to identify options that offer more realistic environments for optimal results.

In Table 7, a comparison of research papers reviewed in terms of reinforcement learning strategies used to perform some of the tasks executed by robotic manipulators is presented, and information about the hardware used to train or implement the RL algorithm in simulation or physically is also added.

In addition to the literature review performed in this study, the RL algorithms with the highest usage in robotic manipulator research are PPO (Proximal Policy Optimization), DQN (Deep Q-Networks), DDQN (Dueling DQN), PPO (Proximal Policy Optimization), TRPO (Trust Region Policy Optimization), AC (Actor, Critic), SAC (Soft Actor-Critic), TD3 (Twin Delayed DDPG), DDPG (Deep Deterministic Policy Gradient), and QT-Opt for tasks such as pin insertion, object picking and placing, trajectory tracking, motion control, manipulation and grasping of rigid and deformable objects, motion control, and mostly use RGB and LIDAR camera sensors. In [53], the authors provided a broader view of RL algorithms applied to robotic manipulations.

3) DRL FRAMEWORKS

Currently, a large variety of deep reinforcement learning frameworks are available online, each with its own documentation and support for its use. However, one of the main challenges in the application of these frameworks in robotics is the lack of standards for problem solving. The integration of all software elements in the customization of viable solutions for each case study is challenging.

Although there are several libraries for implementing the testing of DRL models, it is crucial to establish criteria for the selection of the library that best suits the needs and requirements of the project. Some important criteria to consider are as follows.

- State of the art (SOTA) in terms of the quantity and quality of RL algorithms implemented in similar robotics projects.
- Availability of documentation, tutorials, and application examples to facilitate the learning process and implementation of DRL models.
- Environments that can be used for training and evaluation of RL algorithms, and whether these environments are compatible with available hardware and software.
- Support for visualization and event logging tools during model creation, such as the ability to integrate with

TABLE 8. DRL frameworks.

Library	Developer	Open source	Repository Github	Version	License	TensorBoard Support
Keras RL	Matthias Plappert	Yes	[66]	0.4.2	MIT	Yes
TensorForce	Kuhnle, Alexander, Schaarschmidt, Michael and Fricke, Kai	Yes	[67]	0.6.5	Apache 2.0	Yes
Stable Baselines3	Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann	Yes	[68]	1.7.0	MIT	Yes
TensorLayer	Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu y Yike Guo	Yes	[69]	2.2.4	Apache	Yes
TF-Agents	Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, Eugene Brevdo	Yes	[70]	0.16.0	Apache 2.0	Yes
TorchRL	Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, Vincent Moens	Yes	[71]	0.1.1	MIT	Yes

TensorBoard for the efficient monitoring of training metrics.

- Update and maintain software packages to ensure compatibility with other tools and improve model performance in the future.

Table 8 shows the libraries used for agent creation using deep reinforcement learning. For each of these libraries, key features are presented, such as the link to the repository and documentation, the current version of the library, and the availability of support for TensorBoard, an open-source project that allows the visualization of graphs of machine learning experiments. With the integration of TensorBoard, it is possible to monitor the training metrics of our DRL models and thus improve their performance.

1. KerasRL: This framework implements several deep reinforcement learning algorithms in Python and integrates them with the Keras DL library. It also works with Gymnasium, which offers environments in which different RL algorithms can be evaluated. Some callbacks and metric monitoring can be implemented during the model training.

2. TensorForce: This is a Python-based open-source DRL framework that is based on TensorFlow. Its main feature is a modular design based on components, thus allowing the configuration of various elements of the system to be implemented. The algorithms are independent of the type and structure of the inputs and outputs of the system, and the interaction of the environment used.

3. Stable Baselines 3 (SB3): An open-source framework that integrates seven model-free, the single-agent DRL algorithms are based on OpenAI Baselines and uses TensorFlow. The API of this framework is inspired by the scikit-learn API. Some features of this framework include: API simplicities for agent training, extensive code documentation, and a user guide with a tutorial in Colab. Algorithm implementations contrast with the published results in the SB3 repository. It also contains the following experimental baselines: A2C, PPO, DDPG, SAC, TD3, HER, and DQN. CSV, and TensorBoard file logs.

4. TensorLayer: This modular Python library for deep learning and reinforcement learning built on top of TensorFlow was designed with the following modules: layered model for neural network creation, a model lifecycle management model, training data management module, and workflow management module. It includes several pre-built algorithms, including PPO, DDPG, and TRPO, as well as complete documentation for implementation.

5. TF-Agents: This library facilitates the design, implementation, and testing of RL algorithms using modular components that can be modified and extended for various application-specific usage requirements [72].

6. TorchRL: This is an open-source RL library for PyTorch and is based on Python. It has low- and high-level abstractions that are intended to be modular and to simplify its use for various applications using methods such as reset() and step() [73].

Table 9 presents the algorithms that are implemented in each of the reinforcement learning frameworks considered here and highlights the algorithms that are available for implementation in simulations or physics.

One point to consider when implementing deep reinforcement learning algorithms in robotics applications is the integration of software that allows the abstraction of features of the simulation environments so that the agent can interact in these scenarios. Diverse studies have used different frameworks to train and evaluate DRL models. Gymnasium is one of the most widely used tools for such implementations.

7. OpenAI Gym/Gymnasium. As a Python-based toolkit for research in the field of reinforcement learning, this software package provides an abstraction of the environment, but not of the agent. It was developed to provide users with different styles of implementing agent interfaces in various environments. This environment interface is one of the most widely used for RL applications because of the standardization of its API for different preset and customized environments, which allows for the reproducibility of research in different areas and the integration of different software such as simulation platforms and RL libraries. Some

TABLE 9. Algorithms of DRL frameworks.

Algorithms	KerasRL	Tensorforce	Stable Baselines3	TensorLayer	TF-Agents	TorchRL
AC		✔		✔		
A2C			✔			✔
A3C	✔	✔		✔		
Asynchronous Policy Gradient				✔		
Continues DQN	✔					
CEM	✔					
DDPG	✔		✔		✔	✔
DPG		✔				
DQN	✔			✔	✔	✔
DAGGER				✔		
Deep Q-Learning		✔				
Deep SARSA	✔					
Double DQN	✔	✔			✔	
Dreamer						
Dueling DQN	✔	✔	✔			
HER			✔			
IQL						✔
n-step DQN		✔				
NAF		✔				
PPO	✔	✔	✔		✔	✔
Policy Gradient				✔		
RedQ						✔
REINFORCE		✔		✔	✔	✔
SAC			✔		✔	✔
TD3			✔		✔	✔
TRPO		✔		✔		

of the environmental abstraction methods for this package are as follows.

- Declaration and Initialization: metadata settings, rendering mode, frame rate, etc.
- Observations: Method that translates the states of the environment into observations.
- Steps: Contains the main logic of the environment, processes actions, calculates the states of the environment, and calculates reward values.
- Reset: A function that initializes a new episode and returns an observation of the initial state or auxiliary information.
- Close: A function that closes any resource used in an environment that is not required.

Another aspect of OpenAI Gym is that it offers an exchange of codes and ideas to evaluate the different algorithms developed by users. In addition to providing users with the facility to implement their own integration methods in various environments, OpenAI Gym offers a collection of environments (POMDP) with which it can be run and evaluated, including Atari, Board Games, and 2D/3D robots [74].

OpenAI has no plans to continue the development of Gym and has given control over the Gym repository to the Farama Foundation in early 2021. Currently, this Foundation is designed to host leading open-source reinforcement learning libraries whose goal is to provide standardization and long-term maintenance of RL projects [75]. Future maintenance of OpenAI Gym will be carried out in the new Gymnasium library, from any code simply replacing the import gym with import gymnasium, the required documentation for users is given in [76].

Figure 10 shows that the architecture consists of three main blocks: the first block corresponds to the different control algorithms, such as RL and DRL; the second block consists of an API that allows interaction between the agent and the environment; and the last block consists of a simulator for robotic systems, which must provide an engine for the physics of the system, graphic interfaces, photorealistic graphics, and programming. This toolkit allows the integration of the Gym API with robotic systems and evaluates reinforcement learning algorithms in different simulated environments.

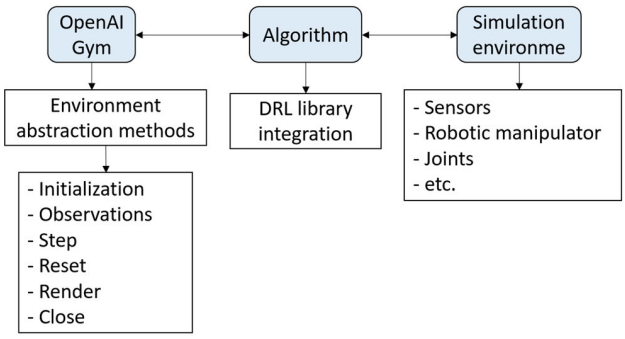

FIGURE 10. Simplified software architecture using the OpenAI Gym API.

There are numerous reviews in the literature on the application of DRL in various contexts. In Table 10, studies related to DL, RL, and DRL with robotic manipulator applications are presented.

TABLE 10. Papers related to DRL control.

Reference	Robot	Task	Control	Comparison	Software
[79]	3-DoF	Trajectory tracking	DNN	-	MATLAB
[80]	6-DoF	Trajectory tracking	Actor-Critic	PD	-
[81]	SCARA	Trajectory tracking	DDPG	DDPG distributed	-
[82]	3-DoF	Trajectory tracking	DDPG	MPC	MATLAB
[83]	7-DoF	Grasping task	PPO, SAC	Setup Real	MuJoCo
[84]	7-DoF	Motion planning	DDPG, HER, TD3	RAMDP	MATLAB, Gazebo
[50]	7-DoF	Path planning	GDRL	SAC	VRep
[85]	7-DoF	Reach, Grasp, Pick and Place	PPO, SAC	Setup Real	PyBullet, ROS
[86]	7-DoF	Peg-in-hole	LSTM+RL	Setup Real	-
[87]	6, 7-DoF	Random-target reaching, door pushing, door pulling, and pick & place	DDPG	Linear-NAF, NAF	MuJoCo
[88]	3, 6, 7-DoF	Contact rich manipulation	TRPO	Simulation, Setup Real	-
[89]	Multi-joint	Trajectory tracking	Adaptive neural control	Fuzzy, PID, SMC	Simscape/MATLAB

4) HYPERPARAMETERS IN DRL

The success of the DRL model training process involves developing an appropriate system design and correctly tuning various hyperparameters to achieve an optimal agent performance when performing specific tasks [77]. In several DRL research papers, the authors did not report details of the design, training, and implementation parameter settings. This complicates the reproducibility of the work [78], and the contribution of knowledge to improving the development of DRL control systems is low or nonexistent. There is no *a priori* method to determine the best values for the hyperparameters in DRL; therefore, several training runs are performed to improve the total rewards, which translates into a high consumption of the agent learning time and computational cost of the hardware during the training stage. Some of the key hyperparameters for DRL training that we consider important are as follows.

- Discount factor: This indicates the relative importance of the future rewards. A discount factor closest to 1 will give more weight to future rewards, but the agent may have difficulty learning behaviors that require short-term optimization. On the other hand, a value closest to zero will prioritize immediate rewards; however, the agent may have difficulty learning long-term behaviors. Therefore, the discount factor value depends on environmental characteristics.
- Batch size: Specifies the number of time steps used in each model training update. A larger batch size may speed up the training but may also require more memory. However, the number of time steps required for training depends on the complexity of the task to be performed by the agent. For tasks such as reaching the target using the end-effector of a robotic manipulator, many training steps are not required; however, it should be noted that there is a minimum number of time steps for the agent to learn a desired policy.
- Neural Network Architecture: The architecture of the neural network used to represent the policy can vary its configuration in terms of layers, units, and type of layers, which can affect the speed and convergence of the training. In the case of images, preprocessing of the input data of the network should be performed to prevent the agent from learning unnecessary data and decrease the training time. The aggregation of dense layers in the neural network output allows the agent to perform actions corresponding to the characteristics learned from the network during the interaction of the agent with the environment. Each layer of the neural network must be subjected to a correct activation function that allows it to obtain at the output of the network a set of actions that represent the values of the desired dimensions (angles, distance, etc.).
- Exploration vs. Exploitation: DRL algorithms often require a balance between exploration (trying new actions to discover their impact on the environment) and exploitation (selecting the best-known action). These parameters must be adjusted to control the amount of time the agent spends exploring a certain part of the environment and exploiting previous knowledge to reinforce the actions during task execution. Some DRL libraries offer the configuration of these hyperparameters in their training routines, depending on the RL algorithm to be used.

The training parameters must be adjusted according to the task to be performed, environment with which the agent must interact, and characteristics of the agent. DRL libraries provide default settings for DRL algorithms; however, researchers must adjust these parameters to achieve a good agent performance.

5) CLOUD SERVICES AND HARDWARE FOR TRAINING AND EVALUATION

Robotics has expanded significantly in a variety of applications. With recent advances in machine learning, more intelligent robots have emerged; however, these upgraded robotic systems require high-performance computational capabilities. Cloud services and resources are used to overcome the limitations of computational resource access to

robotic systems. Cloud resources are used under the premise that a robotic system has insufficient local computational resources and power [90].

Robotic implementations have some restricted computing capabilities, such as data processing, memory capacity, and limited power. Cloud computing allows these robotics problems to be solved by making them less dependent on the robot's native resources. With the use of cloud services, processing and storage can be outsourced because these platforms have virtually unlimited facilities. However, there are other issues to consider, such as the cost of rent for hosting, processing of input data, or deployment of AI models. Paying for the use of these platforms consists of the use of services such as infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), and resources such as CPU, GPU, disk capacity, and network bandwidth. DRL and cloud computing technologies do not yet have a sufficient intersection because the combination of these two components has emerged in recent years [91].

Some platforms enable the development of intelligent systems based on Deep Reinforcement Learning, with applications in autonomous vehicles [92], industrial systems.([93], and robotics [94]. In the following section, two of the most significant cloud services that implement RL are discussed.

Azure Machine Learning: This cloud service enables the training, model deployment, and management of MLOps. The models can be created on this platform or from open-source frameworks such as Pytorch, TensorFlow, or Scikit-learn. Reinforcement Learning techniques are also applied in industrial applications, to train these models are used tools such as: Azure Machine Learning Studio, Python SDK (v2), CLI (v2), Azure Resource Manager REST API that allow performing tasks such as sharing and searching resources, monitoring metrics and managing the training process [95].

Amazon SageMaker RL: This is a fully managed service that allows the creation, training, and deployment of machine learning models in different applications, including robotics and industrial control. One of the features of this service is the support for reinforcement learning. The following components were used to train the RL models:

- Deep learning framework such as Pytorch, Tensorflow and Apache MXNet.
- The RL toolset provides algorithms and agent-environment interaction management, which is compatible with Intel Coach, Open AI Baselines, and Ray RLLib, among others.
- Integration of AWS, custom open source, or commercial environments such as AWS RoboMaker, RoboSchool, PyBullet, MATLAB, and Simulink. In addition, it allows the use of OpenAI Gym environments with API elements.

Certain machine learning frameworks can be used with Python and R, among these we have: Apache MXNet, Apache Spark, Chainer, Hugging Face, PyTorch, Scikit-learn, SparkML Serving, TensorFlow, and Triton Inference Server, these platforms also provide tools for visualization and analysis of RL model training data [96].

It is necessary to emphasize that many of the tasks to be performed for model training and evaluation require a large computational load owing to the complexity of the samples by exploring high-dimensional action spaces and states [97].

There are several computational complexities of DRL for robotics: training data is generated during learning, limited simulation speed, and computational requirements for running libraries and APIs. In addition, simulators play an important role in robot training, as they provide an efficient and scalable platform without safety issues for robot training [98].

The hardware in modern PC's has undergone considerable technological evolution. GPUs are commonly used for gaming or video editing; however, they play a relevant role in AI applications using Deep Learning and Reinforcement Learning. GPUs have a major advantage over CPUs because of their ability to process data in parallel with a reduction in power consumption, which allows more calculations to be performed simultaneously. For machine learning applications, the manufacturer NVIDIA is the leader in GPU technologies. NVIDIA specializes in the development of hardware and software for training and evaluating AI systems. Over time, NVIDIA has developed many generations of GPU hardware, each of which has improved the use of processing resources and new functionalities.

Two of the main hardware systems for AI implementation are the GeForce and A100 Tensor Core. The former, although used to run high-end PC games, is also used in AI. The second is a relatively new line of NVIDIA hardware that offers high data-processing and AI capabilities. In addition, NVIDIA offers software support that provides frameworks and libraries such as CUDA, cuDNN, and TensorRT, which optimize AI modeling workloads on GPU hardware. The following table presents the features of some of the available models of the four NVIDIA GPU series. The information presented in Table 11 is based on a previous study [99].

TABLE 11. NVIDIA GeForce GPU hardware.

GPU	RAM (GB)	CUDA	Tensor Core	Power (W)
RTX 4090	24	16384	4th	850
RTX 3090 Ti	24	10752	3rd	850
RTX 2080 Ti	11	4352	2nd	650
GTX 1660 Ti	6	1536	-	450

High-performance NVIDIA GPU ranges can be divided into two types: laptops and desktops. It should be noted that the hardware for laptops is lite versions because of the limited space for cooling systems, which would decrease the performance of the entire system, unlike a desktop computer that has space and more robust hardware for GPU, storage, RAM, cooling, and power supply, among others.

Some of the GPU elements mentioned in the table above are (1) CUDA Core, which are cores that perform operations simultaneously and in parallel. They are used for tasks such as graphics processing, simulations, and machine-learning model training. (2) The Tensor Core is a hardware component that accelerates mathematical operations related to AI, specifically to perform operations using low-precision matrices (FP16 and FP32). (3) Video RAM (VRAM), which has a higher capacity allows more data processing during training, thereby improving the performance of AI applications. Depending on the GPU model, power consumption (W) varies according to the architecture, workload, and hardware range.

VI. SIMULATORS

Currently, advances in the field of robotics are based on the use of simulators, and because there is great availability and variety of simulators, determining which one is relevant depends on the field of research and the scenario to be simulated. The use of simulation platforms in robotics offers many advantages, such as the creation of synthetic data for AI models, testing new theoretical methods, access to a variety of environments and robots without the need to put at risk any physical platform, and faster execution of control methods and robots in simulation [42]. In this study, we focus on the presentation and description of simulators to research applications using robotic manipulators.

Simulators are very useful for the creation of DRL agents, as they allow the emulation of real-world scenarios, help generate a large volume of data for DRL applications, and avoid any hardware damage during the training process. Tools allow robots to be connected and trained using DRL approaches. As in the case of Gymnasium, this tool can customize the simulation environments [43].

Transfer learning is an important process in robotic applications because most robotic systems are trained and evaluated in simulators, and transferring this learning to real robots is necessary. Simulation allows maximizing the learning process of the robot by collecting large amounts of data, adapting to various configurations of environmental conditions, and so on [100].

There is a wide range of simulator options between commercial and open sources, which are constantly changing as new functionalities to keep up with the robotics research trends. In the following paragraphs, we present a review of simulators used for robotic manipulator applications.

1. CoppeliaSim: Formerly called V-Rep, released in 2010, a closed source with a free educational license is used in research for testing and debugging complex robotic systems, such as the navigation of biped robots and visual trajectory tracking of differential drive robots. Each object/model in the simulation can be individually controlled using secondary commands, ROS nodes, and external client API integration. The elements of this simulator consist of scene objects (sensors, paths, robots), calculation modules (inverse kinematics, collision detection), and control mechanisms (scripts and plugins) [101].

2. Isaac Sim: This is a virtual robotics laboratory and high-fidelity 3D simulator whose functionality is based on five extensions: main (core functionality and control), sensors (interface creation), asset conversion (robot tool import), robots (classes), and others (debugging and motion program generation). It has a built-in URDF loader that allows importing URDF models of robots into a simulator (simulating joints and motions) [102]. For reinforcement learning, there is an extension called Isaac Gym that allows vectorization of a customized environment, and the training processes are accelerated through an end-to-end GPU [98]. Documentation for the use of this physical robotics simulator is available in [103].

3. Gazebo: One of the most popular simulators for research on mobile ground robots, with limbs and wheels, developed in 2004, is an open-source simulator that offers a library of models for sensors such as cameras, GPS, and IMU. It runs on Linux, although several versions of Windows exist. This allows the import of various types of environments and robot models using URDF files. This simulator does not provide motion planning functionality; however, owing to ROS integration, it allows path planning [104].

4. RoboDK: This simulation and programming software was used for the industrial robots. Programming can be performed, simulated, and generated offline (from a computer). It contains an extensive library of CAD models based on industrial robots and tools. The interface is intuitive and has a module designed for calibrating simulated robots [105].

5. MATLAB/Robotics System Toolbox: Provides tools and algorithms for designing, simulating, and testing manipulators and mobile and humanoid robots. The toolbox includes algorithms for collision checking, trajectory generation, direct and inverse kinematics, and dynamics using a rigid-body tree representation. Mobile robots include algorithms for mapping, localization, path planning, path tracking, and motion control. It has its own library of industrial robot models that can be imported, visualized, and simulated. Simulation applications (Simulink) can add sensor models and environments and can be connected to Gazebo [106].

Table 12 compiles relevant information on these simulators that are applicable to the field of robotics, specifically for robotic manipulators, and compares the main features of each to establish a clear picture of the use and applications of robotic manipulators. Some of these elements allow the selection of a simulator for the required case study. These include the type of license, support for ROS, languages used for control programming and functionalities of robots and objects in simulation, availability of APIs for connection with other software or hardware tools, types of components such as sensors, end effectors, dynamic and static objects available for the simulation scene, libraries of robotic models, and the facility to create/import various types of robot models.

TABLE 12. Comparison of simulators for robotic manipulators.

	CoppeliaSim	Isaac Sim	Gazebo	RoboDK	MATLAB
Web Site	[101]	[107]	[104]	[105]	[106]
Latest version	4.6 (10, 2023)	2022.2	11.0.0 (01, 2020)	5.5.4 (02, 2023)	R2023a
License	Free version (students and research), Pro version (purchase)	Individual free of cost, Subscription for some products	Open-source	Trial version (1 month), Pro version (purchase)	Standard, academic, (purchase)
ROS support	ROS1, ROS2	ROS1, ROS2	ROS1, ROS2	No	ROS1, ROS2
Programming language	Lua	Python	C++	Python	M
API	C, C++, Python, Java, Urbi, MATLAB, Octave	Python, ROS, sensors, Robots, etc.	C++	C, C++, Python, MATLAB	C/C++, Fortran, Java, Python, COM components and applications
Robot tools	Pens, paint spray gun, welding torch, gripper	Customizable	Gripper	Grippers, soldering tools, cutting tools, finishing tools, tool changer	Gripper
Sensors	Vision, force, proximity, gyroscopic accelerometers, lasers, lidar	Camera (RGB, depth), LIDAR, IMU, segmentation	Camera, depth, distance, proximity, proximity, force, laser camera	Lasers, cameras, laser trackers	Camera, sonar, LIDAR, GPS, IMU and custom model integration
Robot types	Mobile, Humanoid, Industrial	Humanoid, industrial, mobile	Mobile, Humanoid, Industrial	Industrial robots (commercial robots)	Humanoid, industrial, mobile, customized robots

TABLE 13. System requirements.

Hardware/ Software	CoppeliaSim	Isaac Sim	Gazebo	RoboDK	MATLAB
SO	GNU Ubuntu, Mac OS, Windows	GNU Ubuntu, Windows	GNU Ubuntu	Linux, Mac OS, Windows, Android	Mac, Windows, Linux
CPU	Intel Core i7	Intel Core i7 (9th Gen), AMD Ryzen 7	Intel Core i5	Intel Core i7	Intel o AMD x86–64
RAM	12 GB	64 GB	4 GB	12 GB	8 GB
Storage	500 GB SSD	500 GB SSD	500 MB	500 GB	23 GB
GPU	AMD	GeForce RTX 3080	Nvidia 1 GB	AMD	OpenGL 3.3, 1GB
VRAM	6 GB	10 GB	-	6 GB	6 GB

Table 13 presents information regarding the basic requirements for a desktop or laptop computer to install and use each of the simulation platforms for the development of robotic systems. The selection of the simulator depends on factors such as the scenario, application area, tasks to be performed, available components, robot models, APIs, and programming language. This is only one example, and the specific functions and features of these simulators can change over time. It is recommended to consult the official documentation of each simulator to ensure that the most updated information is used.

More detailed information about commercial, open-source, industrial, and educational simulators can be found in the literature. For the best description of simulators used in robotics, some relevant works are emphasized; in [42], a detailed review of the fields of use and capabilities of several simulators is presented, as well as a description of the functionalities of each of the platforms. In .) [108], an SLR on realistic simulators oriented to educational robotics, which can simulate robots, sensors, and actuators, was presented, and in [109], educational robotics simulators with graphical user interfaces that allow students to interact with virtual robots and robotic mechanisms were presented.

One of the main obstacles in selecting a simulation platform is hardware system requirements. It must be clear which scenarios will be used in the simulation and which elements will be used (sensors, robotic systems, actuators, tools, and software) for the project to be carried out. The available hardware limits the use of simulators because of their specifications. The following table summarizes some hardware requirements that a PC must install on any simulation platform.

VII. SYSTEM INTEGRATION

The use of simulators to train robots with the Deep Reinforcement Learning paradigm can be time consuming because of the difficulty of developing customized experiments and the integration of the different libraries and APIs required for the robotic system. This section presents the general structure of the proposed architecture for the implementation

of robotic manipulator control and simulations using two different approaches. The first approach uses three modules: the CoppeliaSim, Gymnasium, and DRL algorithms. The second approach does not use the Gymnasium toolset to demonstrate different ways of integrating various tools for DRL in robotics. The main objective was to develop a customized configuration of scenarios for the training and testing of robotic manipulators.

Many simulators have tools that permit the addition of features for training and testing DRL algorithms on different types of robots in a fast and safe manner in semi/photorealistic environments [110]. Owing to the difficulty of merging different deep reinforcement learning frameworks with customized robots and environments, a training and testing environment for robotic manipulators was created in this study.

CoppeliaSim was chosen because it is one of the most widely used robotic simulators in research, education, and industry. It can generate semi-photorealistic scenarios in which different types of fixed or mobile station robots can be developed, tested, and managed, and the control can be based on artificial intelligence techniques. It offers an API in Python that allows the integration of all the necessary packages in a single programming language. On the other hand, Gymnasium is a library that addresses the approaches to the definition, structure, and interaction functions of the environment with external elements necessary for our robot to interact in the simulation environment under AI-based control laws.

A. CREATING THE CUSTOM ENVIRONMENT WITH GYMNASIUM

This section provides details of the creation of a customs environment using Gymnasium (v0.26.0) and the semi-photorealistic simulator CoppeliaSim (v 4.6). The minimum organization required to build our custom environment package in Python using Gymnasium and CoppeliaSim is shown in Figure 11. Depending on the application and functions to be implemented, the necessary files will be created in the envs/ directory, allowing the development of a complete software system.

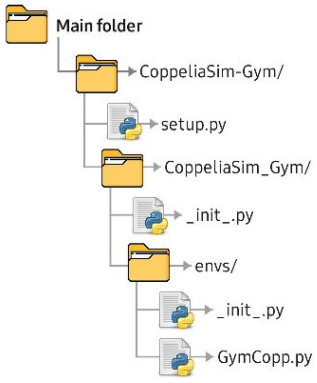

FIGURE 11. Initializing environment with Gymnasium.

At the bottom level, there are two files: (1) GymCopp.py contains the necessary code to manage the customized environment, the declaration, and initialization of the environment that inherits from the abstract class gymnasiumEnv. The metadata attributes are added to the class (GymCoppManR), the rendering modes supported by the environment are specified, and in the __init__ method, some variables are configured to define the environment: self.observation_space, and self.action_space. (2) The _init_.py file must contain the import of the "GymCoppManR" class. From the GymCopp.py file, a connection to the simulator was created using native Python API.

At the middle level, for each custom environment to be detected by Gymnasium, it must be registered in CoppeliaSim-Gym/CoppeliaSim_Gym/_init_.py. The environment ID consists of three components: namespace, mandatory name, and version CoppeliaSim _Gym/GymCoppManR-v0.

The last task is the creation of a package that allows structuring the code written in the Python package. At the top level, a setup.py file must be configured with the minimum code for the package name, version, and installation requirements.

Figure 12 shows the basic structure of the corresponding .py files to create a customized environment for the training and evaluation of DRL agents using CoppeliaSim simulator. After configuring the different basic files for the creation of our customized environment, we installed the package locally on the PC. In the Anaconda environment, open a terminal, go to the CoppeliaSimGym folder, and enter the following instruction: pip install -e CoppeliaSim-Gym.

In the GymCopp.py file, four elements are created that allow working with the simulator: 1) configuration for the connection of the client with the CoppeliaSim server; 2) obtaining the robot joint handlers, sensors, and objects of the simulation scene; 3) Gymnasium methods that allow the calculation of observations, rewards, actions, restarting, and closing the connection with the environment; and 4) functions for the acquisition, data processing, and sending control signals to the robot. Depending on the task to be performed, functions were created to control the actions of the robot in the scene.

Finally, to verify that the custom environment is working, it is necessary to create an instance of the environment through the constructor in the client script: gymnasium.make (Figure 13). The observations were constructed from the states and calculated in both reset and step. The information returned by the reset and step methods also contained data in terms of rewards. The reset method is used to start a new episode and return a tuple of the initial observations and auxiliary information. The step method contains most of the environmental logic, receives the action, calculates the state of the environment after applying the action, and returns the tuple of the four data (observation, reward, done, and information). The closed method must close all open resources after the environment is completed.

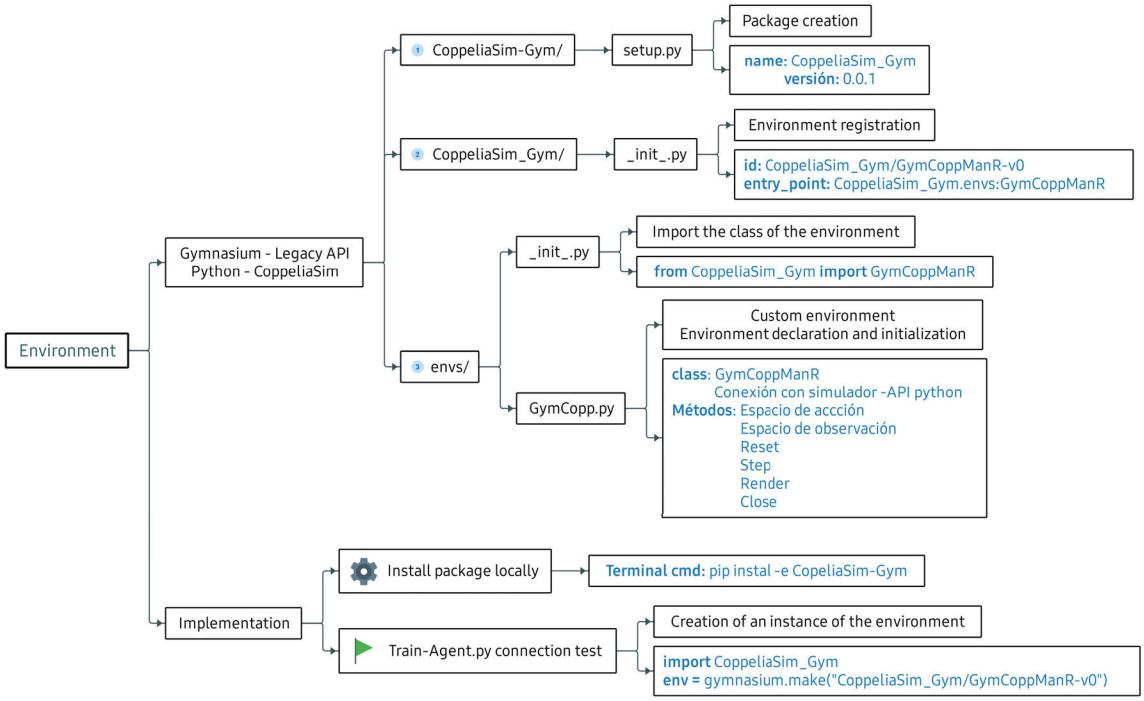

FIGURE 12. Structure of the environment package.

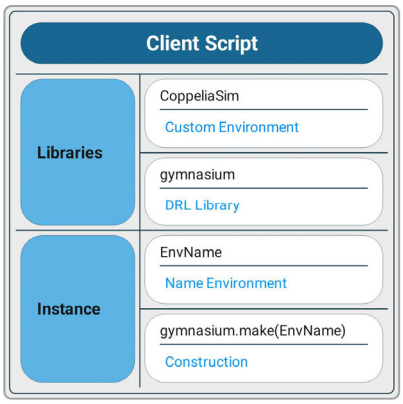

FIGURE 13. Integration of Gymnasium with CoppeliaSim.

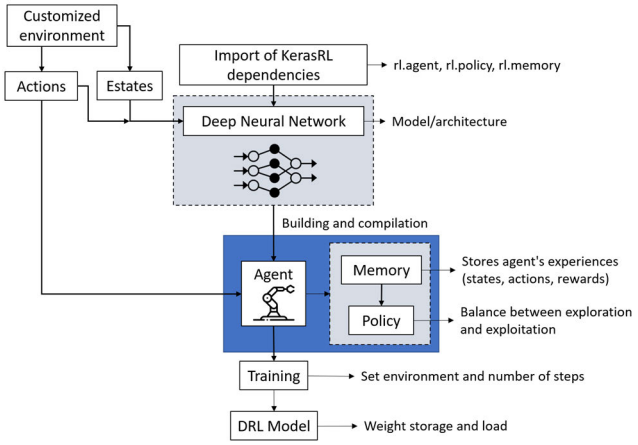

FIGURE 14. General structure of KerasRL training.

In the client script, it is only required to import two libraries: the first is the Gymnasium tool library and the second corresponds to the customized environment created by the user. Then, a call instance is created that allows the user to interact with the created environment and to be able to control the agent and acquire data from the simulation scene.

1) DRL LIBRARY

Various RL frameworks are simple to understand and contain several algorithms used in robotics research. An example application of the DRL model building block based on the DQN algorithm is presented in Figure 14. A DQN (Deep Q-Network) is a reinforcement learning algorithm used to train agents to make decisions in a simulation environment such as CoppeliaSim. These agents can be trained to perform various tasks, such as obstacle avoidance and route planning.

To implement DQN in CoppeliaSim, first, the environment is defined, which includes the task to be performed, the agent and state space, and the action space. This can be achieved using CoppeliaSim's native API, which can define objects and properties and control behavior via commands.

Once the environment is defined, a reinforcement learning library, such as KerasRL or any of those available in RL literature, can be used to implement the DQN algorithm. The agent interacts with the environment by observing the state and taking actions based on a policy, which is learned through trial and error using a Q-learning algorithm. The Q-values are estimated by a neural network trained using a combination of experience reproduction and objective network techniques to improve stability and convergence. The training process

consists of iteratively updating the neural network weights as a function of the TD error (temporal difference) between the predicted Q-values and the actual rewards obtained from the environment.

This framework can implement several deep reinforcement learning algorithms developed in Python. Most libraries that implement reinforcement algorithms work with the Gymnasium toolset, so their integration and evaluation are easy to perform.

2) SEMI-PHOTOREALISTIC SCENE AND ROBOTIC MANIPULATOR

The general structure of the proposed framework comprises of three modules: CoppeliaSim, Gymnasium, and DRL. The .py files necessary for the integration of the three modules were created inside the ''envs'' folder. In the first file, the necessary methods are created to add, define, create, and configure the features of the simulator scenes and sensors. The second file contains all functions required to import, control, and acquire data on the state of the robotic manipulator and its joints. The third file inherits all functions of the classes of the previous files, which follow the basic structure of the Gymnasium API. The management of the data flow between the server components (CoppeliaSim) and the client (Jupyter Notebook) is performed using the simulator's native Python-based remote Legacy API.

In addition to programming the main functions of using sensors, executing actions, controlling data flow between the server and client, etc., are also necessary. However, all these functions can be performed in a single file. To simplify their use in the GymCopp.py file, the methods and functions required to manage the data flow of the different components of the scene (sensors, objects, robotic manipulator, trajectory, marks, etc.) are integrated using the API (see Figure 15).

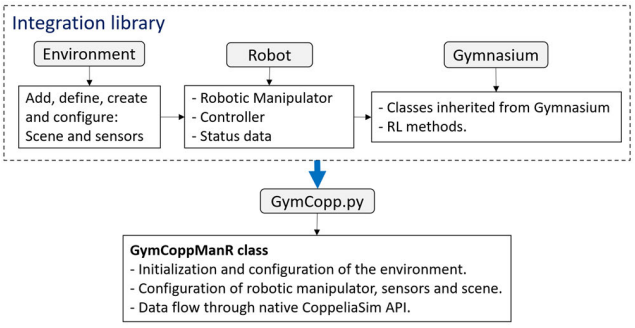

FIGURE 15. Library simplification for the general structure.

3) GENERAL SCHEME OF THE SYSTEM

In our case study, we summarize and explain how the components that integrate our software system work. The general configuration is constituted by a robot with N-degrees of freedom that fulfills the role of ''Agent'', and the customized scene in CoppeliaSim is used as the environment in which our agent will interact.

The environment comprises five elements, the first three are Gymnasium Configuration, Robot Configuration, and Environment Configuration. These elements contain the necessary configurations for using the CoppeliaSim simulator as a training and testing environment for the DRL agent. The two remaining elements are the Robotic Manipulator and Scene, these elements contain the initialization of sensors, scene objects, and data concerning the robot and motion control, all of which are updated when a new step is established. When the environment is reset, both the robotic manipulator and the scene objects return to their initial positions. These components are shown in Figure 16.

Several DRL frameworks can be integrated with Gymnasium, and the inputs of the DRL model are reward calculation, observations of the robot, scene, and Gymnasium's initial setup data. The output of the model corresponds to the action commands that allow motion control of the robotic manipulator. In this case, the action state was set as the angle data, which is a vector of N elements corresponding to the robotic manipulator joints.

Table 14 presents some robotic application projects that use the elements of the framework described in this study. OpenAI Gym is a tool that is mostly used to create environments using the CoppeliaSim simulator.

B. TYPES OF GRAPHICS-DRL SYSTEM WITHOUT GYMNASIUM

In the field of DRL with robotics applications, it was observed that most studies used the Gymnasium toolset to create customized environments with any simulation platform to train and evaluate a DRL agent. This is because of the standardization of methods for agent interaction with the simulation environment and the ease of integration with various simulation platforms and DRL libraries. However, as demonstrated in the literature, software tools designed for DRL can also be used. Figure 17 shows the proposed software architecture for implementing DRL techniques for robotic manipulation without using Gymnasium.

This scheme consists of three major components:

- The simulator provides the functionality required for interaction with the robot and the simulation scene. A work area is created on a platform that allows different elements to interact with each other to mimic the behavior of a real-world scene. A native API controls and sends the data flow to the other components of the system inside and outside the simulation platform using programming languages, such as LUA and Python. It also provides various sensors, robotic components, and static and dynamic elements for a scene.
- The selected framework provides tools and methods for building and training a DRL agent in a customized environment based on a semi-photorealistic simulator. Various predefined robots can be added to the platform and user-customized models can be imported. The DRL library and API enable handling of the decision and

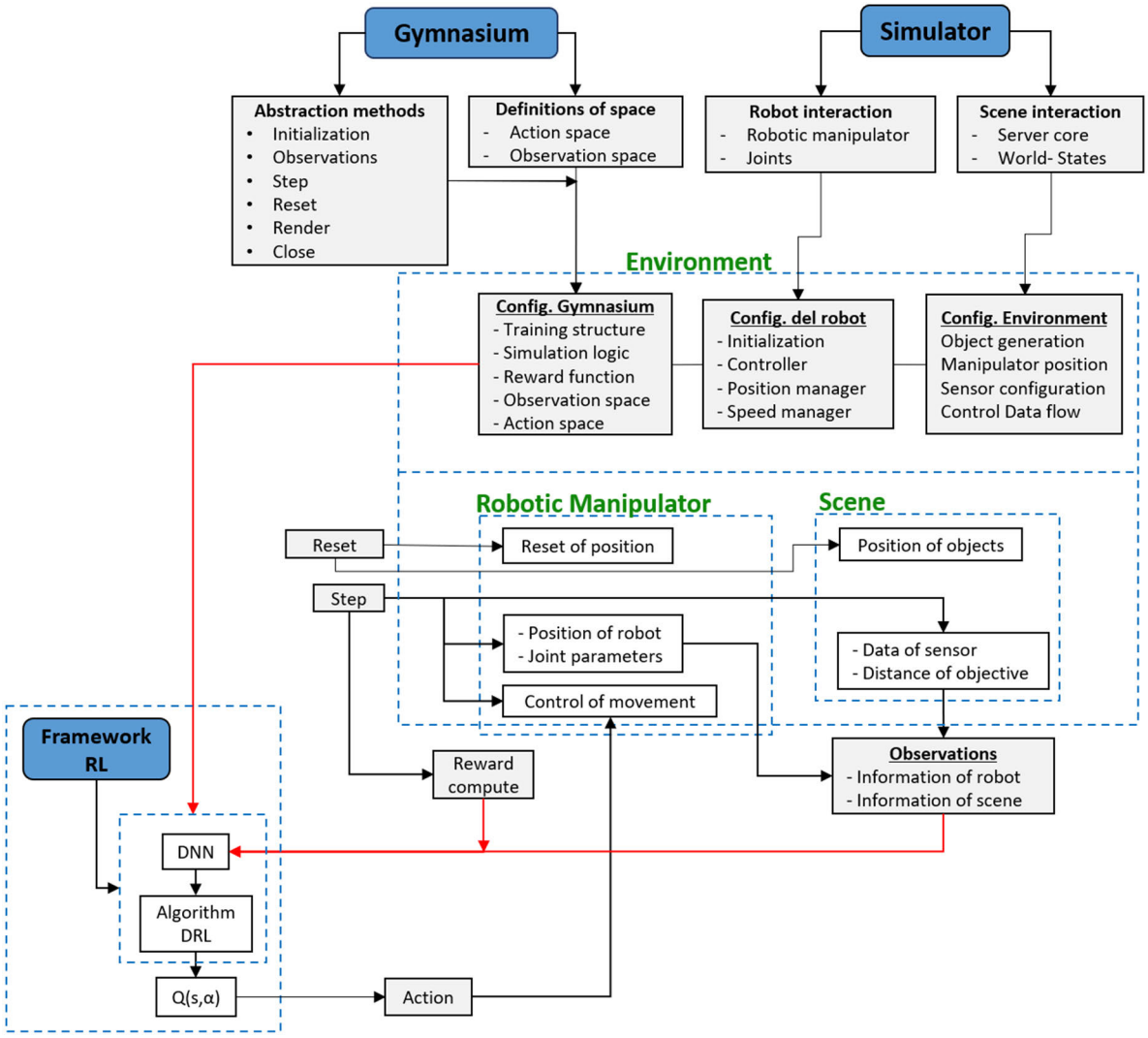

FIGURE 16. Schematic diagram of the software system integration.

TABLE 14. Properties RL projects with robotic applications.

Project	Robot	Type	Task	Library	Algorithm	Simulator
[111]	Kuka 7 DoF	Robotic manipulator	Grasping objects	OpenAI Gym	DDPG	PyBullet
[112]	UR5e	Robotic manipulator	Object insertion	Catalyst RL	DDPG, SAC, TD3	CoppeliaSim
[113]	Kuka IIWA	Robotic manipulator	Grasping objects	TensorFlow	QT-Opt	Bullet Physics
[114]	Khepera IV	Mobile robot	Position control, trajectory tracking, obstacle avoidance, multiple robots	KerasRL	DDPG, DQN	CoppeliaSim

control tasks of the agent and the data generated by the environment. The integration of different components of the simulator with external software tools allows for the creation of robust systems with features that increase their functionality.

- In the set of .py files, scripts integrated in the simulator or from external tools, and the necessary classes and methods are created to obtain the data from the environment, process the information from the sensors, determine control actions for the robotic manipulator, export data related to the training and evaluation of the systems, and create and save the DRL model. The architecture is based on a client-server, and several scripts were created to perform environment configurations for DRL processes, customize functions for robotic tasks, and export data for visualization

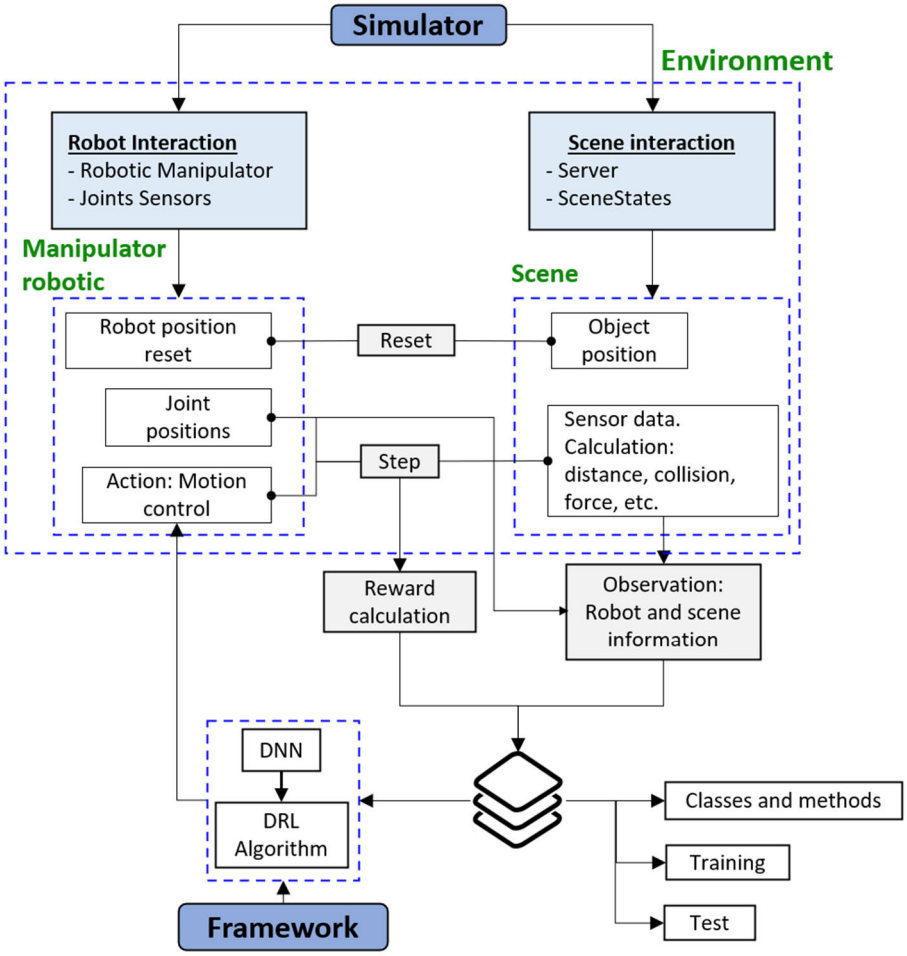

FIGURE 17. General DRL architecture.

in graphs to perform a quantitative analysis of the performance of the system.

Similarly, in the training and evaluation logic for a DRL agent, the reset methods reset the behavior of the robot and objects in the scene to a preset state, and a step that implements the data acquisition functions of the sensors and actions to control the position of the joints of the robotic manipulator is established. Remarkably, the control functions depend on the task to be performed by the robot. The API of the simulator allows the control of some elements of the simulation, because the researcher must solve each task of the system.

VIII. CASE STUDY IMPLEMENTATION

For the development of the modern control approach for robotic manipulators, some of the software tools mentioned in this framework have been used. The experimental setup consisted of a robotic manipulator (KUKA LBR iiwa 14 R820) with seven degrees of freedom (DoF) and a stereo camera. To evaluate the robotic system with DRL, the CoppeliaSim simulator was used to perform the task of reach a target (cube) within the simulation scene. The software architecture is based on a client-server. The simulator was connected to the client via CoppeliaSim's remote API in Python programming language. TensorFlow was used as a baseline for DRL implementation (DDPG algorithm).

Task: Reach target. The agent must approach the blue object located in the working area. The observations obtained from the environment were captured using a stereo camera located in the robotic manipulator, and two images were obtained: RGB and a deep map. From this input, the parameters of the rectangle that frame the target object of the image are extracted using OpenCV.

The parameters of the neural network were adjusted using the DDPG algorithm, which plans the trajectory toward the object located in the visible area captured by the stereo camera. The input of this network consisted of the parameters of the position of the joints of the manipulator and the rectangular area of the object in the image, and the output was the calculated coordinates of the manipulator. The Reward Function consists of calculating the current and next state potentials and the penalty for the duration of the motion, thus establishing a smooth multipoint trajectory. The state potential is the computation of the weighted sum of the

weighted sum of the area of the rectangle of the localized object, its displacement from the center of the image, and the average distance of the depth map. In Figure 18, the robotic manipulator is presented within the CoppeliaSim scene.

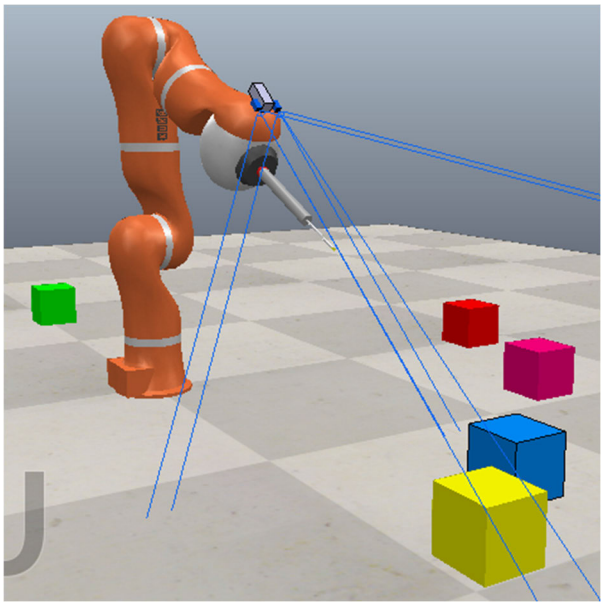

FIGURE 18. General structure of KerasRL training.

The training and evaluation processes of the software system were developed on a machine with an i7 CPU, with 12 GB of RAM, and 1TB of storage. A scene in which the robot was configured in kinematic mode was used to perform the training, whereas a scene in which the robot was configured in dynamic mode was used to evaluate the DRL agent.

A. TRAINING

During the training phase, the agent learned the task of reaching the target (in this case, the blue cube). The agent was trained for 10,000 episodes, which required approximately 2h and 30 minutes to complete. During the training, cubes of different colors were placed at random positions within the working area during each episode. Here, the robot scans this area and detects the blue cube using the neural network, and the RL algorithm calculates the values for each joint of the manipulator so that the end effector can reach the target. ATensorBoard was used to visualize some metrics of the training results. These are actor Q network values, total reward, and penalties during the 10k episodes, as shown in Figures 19, 20, and 21, respectively.

B. EVALUATION

To evaluate the performance of the trained model, a script was created, in which N test episodes were set. Figure 22 shows the steps to evaluate the trained model for the performance of the task of reaching an object.

Table 15 summarizes the control characteristics of the robotic system implemented in the software with DRL to perform the task of reaching a particular object within the simulation scene.

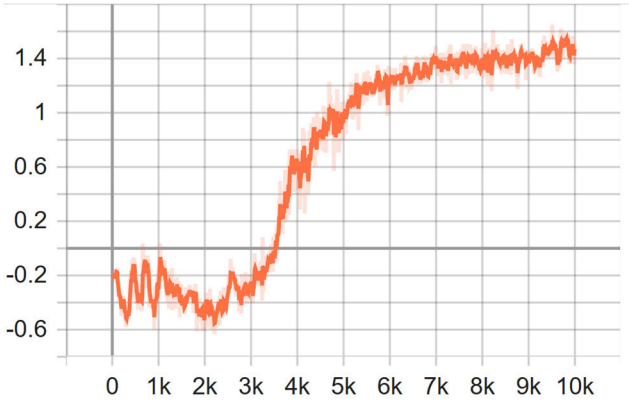

FIGURE 19. Evaluation of the Q-actor network.

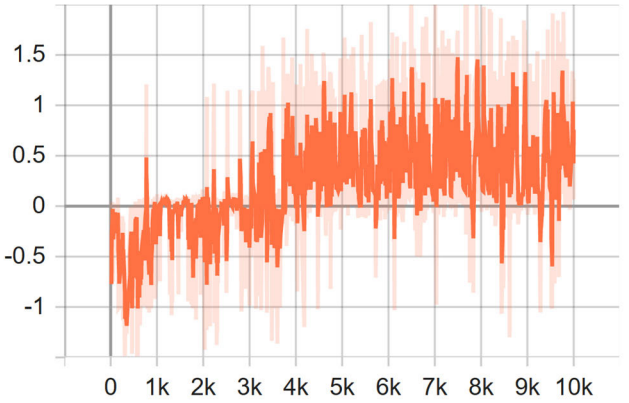

FIGURE 20. Total reward.

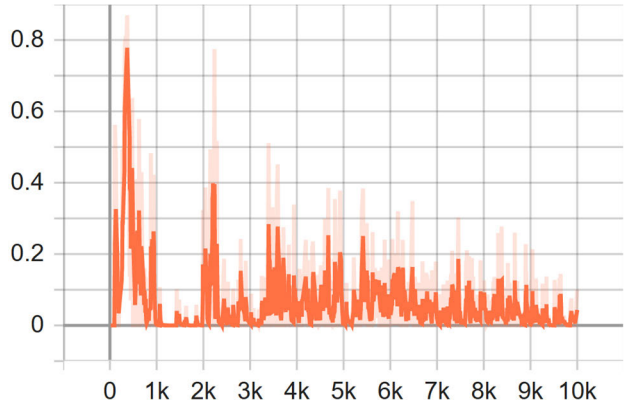

FIGURE 21. Penalties.

TABLE 15. Features of the robotic system.

Experimental setup	Hardware	CPU i7, Windows 10, 12 RAM.
	Software	CoppeliaSim, TensorFlow, OpenCV, TensorBoard.
	Robotic system	KUKA LBR iiwa 14 R820, stereo camera.
Control	Modern approach: DRL (DDPG)	
Task	Reach an object (blue cube)	

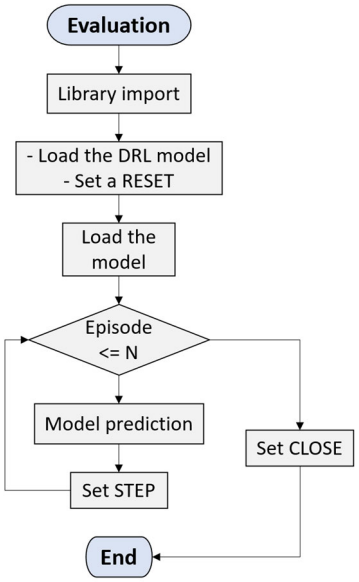

FIGURE 22. Evaluation of the Q-actor network.

The training and evaluation of this system were performed on a computer with a CPU; however, this process could be replicated on a PC equipped with a GPU. To achieve this task, it is necessary to have a GPU that has the drivers and libraries necessary for the configuration of the CUDA [115] and cuDNN [116] systems for TensorFlow installed. Other frameworks, such as PyTorch, can be used. Depending on the operating system of the computer, a set of steps are performed to enable the use of the GPU for tensor operations.

IX. CONCLUSION

This paper presents a framework for performing experiments with robotic manipulators in physical and semi-photorealistic simulation environments using traditional and modern control approaches. A literature review was conducted to present the main software tools available in the robotics field, as well as several related works that allow the extension of the use and functionalities of the mentioned tools.

The main features of existing technology in the field of robotics for DRL research applications in robotic manipulators are presented in detail. Because of the variety of software tools, some of the most used tools in the literature were selected, and the information is summarized in tables to provide a better perspective for the reader.

The use of software tools for robotics research provides multiple benefits, such as cost reduction in the acquisition of equipment, safety, speed in the deployment of robotic systems in complex environments, and verification of the operation and performance of the control system for robotic manipulators using simulators and visualization tools.

For demonstration purposes, the integration of our software system was composed of several selected standard libraries such as TensorFlow, OpenCV, Numpy, and remote API Legacy from CoppeliaSim, which allowed the creation of a DRL agent. Python programming language was used as a base to facilitate the configuration and customization of the robot, vision sensors, environment configuration, and methods for experimentation with Artificial Intelligence-based robotic manipulators.

This article presents a framework for developing robotic applications in different areas with software tools using simulators, deep reinforcement learning libraries, APIs, toolboxes, etc. Thus, a methodology that allows the integration of each software element and facilitates the deployment of a robotic system without the existing restrictions of the real world can be developed. This work can be a starting point for new researchers in the field of robotics, and can serve as a reference guide for creating new lines of research.

APPENDIX

Source code and training and evaluation scenes in CoppeliaSim of the DRL agent available at: https://github.com/RogerSgo/DRLForManipulator

REFERENCES

[1] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, Jan. 2021.
[2] Z. Zheng, M. Yu, P. Guo, and D. Zeng, "Neurodynamics adaptive reward and action for hand-to-eye calibration with deep reinforcement learning," *IEEE Access*, vol. 11, pp. 60292–60304, 2023.
[3] C. Lopez-Franco, D. Diaz, J. Hernandez-Barragan, N. Arana-Daniel, and M. Lopez-Franco, "A metaheuristic optimization approach for trajectory tracking of robot manipulators," *Mathematics*, vol. 10, no. 7, p. 1051, Mar. 2022.
[4] J. J. E. Iqbal, "Modern control laws for an articulated robotic arm," *Eng., Technol. Appl. Sci.*, vol. 9, no. 2, pp. 4057–4061, 2019.
[5] R. Mohammed, F. Bendary, and K. Elserafi, "Trajectory tracking control for robot manipulator using fractional order-fuzzy-PID controller," *Int. J. Comput. Appl.*, vol. 134, no. 15, pp. 22–29, Jan. 2016.
[6] W. Liu, H. Niu, M. N. Mahyuddin, G. Herrmann, and J. Carrasco, "A model-free deep reinforcement learning approach for robotic manipulators path planning," in *Proc. 21st Int. Conf. Control, Autom. Syst. (ICCAS)*, Oct. 2021, pp. 512–517.
[7] T. Osa and M. Aizawa, "Deep reinforcement learning with adversarial training for automated excavation using depth images," *IEEE Access*, vol. 10, pp. 4523–4535, 2022.
[8] (2023). *RIA: Robotic Industries Association*. Accessed: Oct. 10, 2023. [Online]. Available: https://webstore.ansi.org/sdo/ria
[9] (2023). *JARA: Japan Robot Association*. Accessed: Oct. 10, 2023. [Online]. Available: https://www.jara.jp/e/
[10] (2023). *Eurobotics*. Accessed: Oct. 10, 2023. [Online]. Available: https://eu-robotics.net/
[11] Y.-C. Liu and C.-Y. Huang, "DDPG-based adaptive robust tracking control for aerial manipulators with decoupling approach," *IEEE Trans. Cybern.*, vol. 52, no. 8, pp. 8258–8271, Aug. 2022.
[12] K. Cetin, H. Tugal, Y. Petillot, M. Dunnigan, L. Newbrook, and M. S. Erden, "A robotic experimental setup with a stewart platform to emulate underwater vehicle-manipulator systems," *Sensors*, vol. 22, no. 15, p. 5827, Aug. 2022.
[13] M. Rahman, H. Liu, M. Masri, I. Durazo-Cardenas, and A. Starr, "A railway track reconstruction method using robotic vision on a mobile manipulator: A proposed strategy," *Comput. Ind.*, vol. 148, Jun. 2023, Art. no. 103900.
[14] D. S. Carabis and J. T. Wen, "Trajectory generation for flexible-joint space manipulators," *Frontiers Robot. AI*, vol. 9, pp. 1–14, Mar. 2022.
[15] A. S. Morgan, K. Hang, W. G. Bircher, F. M. Alladkani, A. Gandhi, B. Calli, and A. M. Dollar, "Benchmarking cluttered robot pick-and-place manipulation with the box and blocks test," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 454–461, Apr. 2020.

[16] P. V. Torres-Carrión, C. S. González-González, S. Aciar, and G. Rodríguez-Morales, "Methodology for systematic literature review applied to engineering and education," in *Proc. IEEE Global Eng. Educ. Conf.*, Apr. 2018, pp. 1364–1373.

[17] (2023). *Engineers*. Accessed: Oct. 10, 2023. [Online]. Available: https://www.ieee.org/publications/services/thesaurus-thank-you.html

[18] R. P. Borase, D. K. Maghade, S. Y. Sondkar, and S. N. Pawar, "A review of PID control, tuning methods and applications," *Int. J. Dyn. Control*, vol. 9, no. 2, pp. 818–827, Jun. 2021.

[19] B. Siciliano, O. Khatib, and T. Kroger, *Springer Handbook of Robotics*, 2nd ed., Cham, Switzerland: Springer, 2008.

[20] M. Fadali and A. Visioli, "Introduction to digital control," in *Digital Control Engineering: Analysis and Design*, 3rd ed. New York, NY, USA: Academic, 2020, ch. 1, pp. 1–8.

[21] A. O'dwyer, *Handbook of PI and PID Controller Tuning Rules*, 3rd ed. Singapore: World Scientific, 2009.

[22] K. Bingi, B. Rajanarayan Prusty, and A. Pal Singh, "A review on fractional-order modelling and control of robotic manipulators," *Fractal Fractional*, vol. 7, no. 1, p. 77, Jan. 2023.

[23] D. Valério and J. S. da Costa, "Tuning of fractional PID controllers with Ziegler–Nichols-type rules," *Signal Process.*, vol. 86, no. 10, pp. 2771–2784, Oct. 2006.

[24] P. R. Ouyang, J. Acob, and V. Pano, "PD with sliding mode control for trajectory tracking of robotic system," *Robot. Comput.-Integr. Manuf.*, vol. 30, no. 2, pp. 189–200, Apr. 2014.

[25] W. E. Abdul-Lateef, Y. N. I. Alothman, and S. A.-H. Gitaffa, "An optimal motion path planning control of a robotic manipulator based on the hybrid PI-sliding mode controller," *Bull. Electr. Eng. Informat.*, vol. 12, no. 2, pp. 727–737, Apr. 2023.

[26] J. Charaja, E. Muñoz-Panduro, O. E. Ramos, and R. Canahuire, "Trajectory tracking control of UR5 robot: A PD with gravity compensation and sliding mode control comparison," in *Proc. Int. Conf. Control, Autom. Diagnosis*, Oct. 2020, pp. 1–5.

[27] T. A. Tutunji, M. Al-Khawaldeh, and M. Alkayyali, "A three-stage PSO-based methodology for tuning an optimal PD-controller for robotic arm manipulators," *Evol. Intell.*, vol. 15, no. 1, pp. 381–396, Mar. 2022.

[28] L. Angel and J. Viola, "Control performance assessment of fractional-order PID controllers applied to tracking trajectory control of robotic systems," *WSEAS Trans. Syst. Control*, vol. 17, pp. 62–73, Feb. 2022.

[29] J. E. Lavín-Delgado, J. E. Solís-Pérez, J. F. Gómez-Aguilar, and R. F. Escobar-Jiménez, "Trajectory tracking control based on non-singular fractional derivatives for the Puma 560 robot arm," *Multibody Syst. Dyn.*, vol. 50, no. 3, pp. 259–303, Nov. 2020.

[30] B. Ataslar-Ayyildiz and O. Karahan, "Tuning of fractional order PID controller using CS algorithm for trajectory tracking control," in *Proc. 6th Int. Conf. Control Eng. Inf. Technol.*, Oct. 2018, pp. 1–6.

[31] H. Komijani, M. Masoumnezhad, M. M. Zanjireh, and M. Mir, "Robust hybrid fractional order proportional derivative sliding mode controller for robot manipulator based on extended grey wolf optimizer," *Robotica*, vol. 38, no. 4, pp. 605–616, Apr. 2020.

[32] V. Mohan, B. Panjwani, H. Chhabra, A. Rani, and V. Singh, "Self-regulatory fractional fuzzy control for dynamic systems: An analytical approach," *Int. J. Fuzzy Syst.*, vol. 25, no. 2, pp. 794–815, Mar. 2023.

[33] H. I. Abdulameer and M. J. Mohamed, "Fractional order fuzzy PID controller design for 2-Link rigid robot manipulator," *Int. J. Intell. Eng. Syst.*, vol. 15, pp. 103–117, Jul. 2021.

[34] A. Tepljakov, E. Petlenkov, and J. Belikov. (2023). *FOMCON Toolbox for MATLAB*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/extall/fomcon-MATLAB

[35] A. Tepljakov, E. Petlenkov, J. Belikov, and I. Petras, "FOMCON toolbox for modeling, design and implementation of fractional-order control systems," *Appl. Control*, vol. 1, pp. 211–236, Jul. 2019.

[36] J. Liu, *Sliding Mode Control Using MATLAB*. New York, NY, USA: Academic, 2017.

[37] P. Ji, C. Li, and F. Ma, "Sliding mode control of manipulator based on improved reaching law and sliding surface," *Mathematics*, vol. 10, no. 11, p. 1935, Jun. 2022.

[38] X. Cheng, H. Liu, and W. Lu, "Chattering-suppressed sliding mode control for flexible-joint robot manipulators," *Actuators*, vol. 10, no. 11, p. 288, Oct. 2021.

[39] A. Eltayeb, M. F. Rahmat, M. A. M. Eltoum, S. Ibrahim, and M. A. M. Basri, "Adaptive sliding mode control design for the 2-DOF robot arm manipulators," in *Proc. Int. Conf. Comput., Control, Electr., Electron. Eng.*, Sep. 2019, pp. 1–5.

[40] R. S. Peres, X. Jia, J. Lee, K. Sun, A. W. Colombo, and J. Barata, "Industrial artificial intelligence in Industry 4.0–systematic review, challenges and outlook," *IEEE Access*, vol. 8, pp. 220121–220139, 2020.

[41] A. del Real Torres, D. S. Andreiana, Á. Ojeda Roldán, A. Hernández Bustos, and L. E. Acevedo Galicia, "A review of deep reinforcement learning approaches for smart manufacturing in Industry 4.0 and 5.0 framework," *Appl. Sci.*, vol. 12, no. 23, p. 12377, Dec. 2022.

[42] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51416–51431, 2021.

[43] S. Gupta, G. Singal, and D. Garg, "Deep reinforcement learning techniques in diversified domains: A survey," *Arch. Comput. Methods Eng.*, vol. 28, no. 7, pp. 4715–4754, Dec. 2021.

[44] MathWorks. (2023). *Reinforcement Learning for Control Systems Applications*. Accessed: Oct. 10, 2023. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-for-control-systems-applications.html

[45] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A research framework for deep reinforcement learning," 2018, *arXiv:1812.06110*.

[46] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning," *Sensors*, vol. 21, no. 4, p. 1278, Feb. 2021.

[47] H. Sun, W. Zhang, R. Yu, and Y. Zhang, "Motion planning for mobile robots—Focusing on deep reinforcement learning: A systematic review," *IEEE Access*, vol. 9, pp. 69061–69081, 2021.

[48] L.-L. Liu, E.-L. Chen, Z.-G. Gao, and Y. Wang, "Research on motion planning of seven degree of freedom manipulator based on DDPG," in *Advanced Manufacturing and Automation VIII*. Cham, Switzerland: Springer, 2019, pp. 356–367.

[49] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209320–209344, 2020.

[50] Y. Shen, Q. Jia, Z. Huang, R. Wang, and G. Chen, "Guided deep reinforcement learning for path planning of robotic manipulators," in *Proc. 5th Int. Conf.*, 2020, pp. 1–23.

[51] J. Su, H. Cheng, H. Guo, R. Huang, and Z. Peng, "An approximate quadratic programming for efficient Bellman equation solution," *IEEE Access*, vol. 7, pp. 126077–126087, 2019.

[52] H. Dong, Z. Ding, S. Zhang, and Z. Chang, *Deep Reinforcement Learning—Fundamentals, Research and Application*. Cham, Switzerland: Springer, 2020.

[53] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, Apr. 2023.

[54] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *Proc. 3rd IEEE Int. Conf. Robotic Comput. (IRC)*, Feb. 2019, pp. 590–595.

[55] A. Lobbezoo, Y. Qian, and H.-J. Kwon, "Reinforcement learning for pick and place operations in robotics: A survey," *Robotics*, vol. 10, no. 3, p. 105, Sep. 2021.

[56] E. F. Morales, R. Murrieta-Cid, I. Becerra, and M. A. Esquivel-Basaldua, "A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning," *Intell. Service Robot.*, vol. 14, no. 5, pp. 773–805, Nov. 2021.

[57] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," in *Proc. AAAI Conf. Artif. Intell.*, 2020, no. 4, pp. 5981–5988.

[58] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 1395–1476, 2021.

[59] A. Afzal, C. L. Goues, M. Hilton, and C. S. Timperley, "A study on challenges of testing robotic systems," in *Proc. IEEE 13th Int. Conf. Softw. Test., Validation Verification (ICST)*, Oct. 2020, pp. 96–107.

[60] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 8973–8979.

[61] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman, "Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning," 2020, *arXiv:2004.10190*.

[62] M. Coskun, O. Yildirim, and Y. Demir, "Robotic grasping in simulation using deep reinforcement learning," in *Proc. 7th Int. Conf. Comput. Sci. Eng. (UBMK)*, Sep. 2022, pp. 131–136.

[63] T. Bhuiyan, L. Kastner, Y. Hu, B. Kutschank, and J. Lambrecht, "Deep-reinforcement-learning-based path planning for industrial robots using distance sensors as observation," in *Proc. 8th Int. Conf. Control Robot.*, 2023, pp. 1–14.
[64] D. Pavlichenko and S. Behnke, "Real-robot deep reinforcement learning: Improving trajectory tracking of flexible-joint manipulator with reference correction," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 2671–2677.
[65] J. Xie, Z. Shao, Y. Li, Y. Guan, and J. Tan, "Deep reinforcement learning with optimized reward functions for robotic trajectory planning," *IEEE Access*, vol. 7, pp. 105669–105679, 2019.
[66] M. Plappert. (2023). *Deep Reinforcement Learning for Keras*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/keras-rl/keras-rl
[67] A. Kuhnle, M. Schaarschmidt, and K. Fricke. (2023). *Tensorforce: A TensorFlow Library for Applied Reinforcement Learning*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/tensorforce/tensorforce
[68] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. (2023). *Stable Baselines3*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/DLR-RM/stable-baselines3
[69] H. Dong, A. Supratak, L. Mai, F. Liu, A. Oehmichen, S. Yu, and Y. Guo. (Oct. 10, 2023). *TensorLayer*. [Online]. Available: https://github.com/tensorlayer/tensorlayer
[70] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-agents: A reliable, scalable and easy to use TensorFlow library for contextual bandits and reinforcement learning," TensorFlow, Google, CA, USA, Tech. Rep., 2018.
[71] A. Bou. (2023). *TorchRL*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/pytorch/rl
[72] TensorFlow. (2023). *Agents Es Una Biblioteca Para El Aprendizaje Por Refuerzo En TensorFlow*. Accessed: Oct. 10, 2023. [Online]. Available: https://www.tensorflow.org/agents?hl=es-419
[73] PyTorch. (2023). *TorchRL Documentation*. Accessed: Oct. 10, 2023. [Online]. Available: https://pytorch.org/rl/
[74] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, "Openai gym," 2016, *arXiv:1606.01540*.
[75] (2023). *Announcing The Farama Foundation: The Future of Open Source Reinforcement Learning*. Accessed: Oct. 10, 2023. [Online]. Available: https://farama.org/Announcing-The-Farama-Foundation
[76] M. Towers. (2023). *Gymnasium*. Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/Farama-Foundation/Gymnasium
[77] R. Liessner, J. Schmitt, A. Dietermann, and B. Bcker, "Hyperparameter optimization for deep reinforcement learning in vehicle energy management," in *Proc. ICAART*, 2019, pp. 134–144.
[78] T. Eimer, M. Lindauer, and R. Raileanu, "Hyperparameters in reinforcement learning and how to tune them," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 1–20.
[79] G. Zhong, Y. Li, and J. Li, "Trajectory tracking control of robotic manipulators by multi-layer neural networks," in *Proc. 15th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, 2020, pp. 1814–1819.
[80] Y. P. Pane, S. P. Nageshrao, and R. Babuska, "Actor-critic reinforcement learning for tracking control in robotics," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, Dec. 2016, pp. 5819–5826.
[81] S. Zhang, C. Sun, Z. Feng, and G. Hu, "Trajectory-tracking control of robotic systems via deep reinforcement learning," in *Proc. IEEE Int. Conf. Cybern. Intell. Syst. (CIS) IEEE Conf. Robot., Autom. Mechatronics (RAM)*, Nov. 2019, pp. 386–391.
[82] C.-H. Tsai, J.-J. Lin, T.-F. Hsieh, and J.-Y. Yen, "Trajectory control of an articulated robot based on direct reinforcement learning," *Robotics*, vol. 11, no. 5, p. 116, Oct. 2022.
[83] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda, "Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning," *Auto. Robots*, vol. 46, no. 3, pp. 483–498, Mar. 2022.
[84] M. Kim, D.-K. Han, J.-H. Park, and J.-S. Kim, "Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay," *Appl. Sci.*, vol. 10, no. 2, p. 575, Jan. 2020.
[85] A. Lobbezoo, *Robotic Reach, Grasp, and Pick-and-Place Using Combined Reinforcement Learning and Traditional Controls*. Waterloo, Canada: University of Waterloo, 2022.
[86] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 819–825.
[87] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.
[88] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 8943–8950.
[89] H. Sai, Z. Xu, and E. Zhang, "Adaptive practical predefined-time neural tracking control for multi-joint uncertain robotic manipulators with input saturation," *Neural Comput. Appl.*, vol. 35, no. 27, pp. 20423–20440, Sep. 2023.
[90] M. Penmetcha and B.-C. Min, "A deep reinforcement learning-based dynamic computational offloading method for cloud robotics," *IEEE Access*, vol. 9, pp. 60265–60279, 2021.
[91] G. Zhou, W. Tian, R. Buyya, R. Xue, and L. Song, "Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions," *Artif. Intell. Rev.*, vol. 57, no. 5, pp. 1–17, Apr. 2024.
[92] M. Spryn, A. Sharma, D. Parkar, and M. Shrimal, "Distributed deep reinforcement learning on the cloud for autonomous driving," in *Proc. IEEE/ACM 1st Int. Workshop Softw. Eng. AI Auto. Syst. (SEFAIAS)*, May 2018, pp. 16–22.
[93] M. Escobar, *Distributed Deep Reinforcement Learning in an HPC System and Deployment to the Cloud*. Barcelona, Spain: Universitat Politecnica de Catalunya, 2021.
[94] R. Alroobaea, A. Binmahfoudh, S. M. Alzahrani, and A. Althobaiti, "Markov decision process with deep reinforcement learning for robotics data offloading in cloud network," *J. Electron. Imag.*, vol. 31, no. 6, May 2022, Art. no. 061809.
[95] Microsoft. (2023). *Azure Machine Learning Documentation*. [Online]. Available: https://learn.microsoft.com/es-es/azure/machine-learning/?view=azureml-api-2
[96] Amazon. (2023). *Amazon SageMaker Documentation*. [Online]. Available: https://docs.aws.amazon.com/sagemaker/index.html
[97] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," in *Proc. Conf. Robot Learn.*, 2018, pp. 270–282.
[98] V. Makoviychuk et al., "Isaac gym: High performance GPU-based physics simulation for robot learning," 2021, *arXiv:2108.10470*.
[99] NVIDIA. (2023). *Compare GeForce Graphics Cards*. [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/compare/
[100] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, "Review of deep reinforcement learning-based object grasping: Techniques, open challenges, and recommendations," *IEEE Access*, vol. 8, pp. 178450–178481, 2020.
[101] C. Robotics. (2023). *Robotics Simulator CoppeliaSim*. [Online]. Available: https://www.coppeliarobotics.com/
[102] F. F. Monteiro, A. L. B. Vieira-e-Silva, J. M. X. N. Teixeira, and V. Teichrieb, "Simulating real robots in virtual environments using NVIDIA's Isaac SDK," in *Proc. Anais Estendidos 21st Simpósio Realidade Virtual Aumentada*, Rio de Janeiro, Brazil, 2019, pp. 47–48, doi: 10.5753/svr_estendido.2019.8471.
[103] N. Omniverse. (2023). *Isaac Sim Documentation*. [Online]. Available: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/overview.html
[104] O. Robotics. (2023). *Gazebo*. Accessed: Oct. 10, 2023. [Online]. Available: https://gazebosim.org/home
[105] RoboDK. (2023). *Simulate Robot Aplications-Program Any Industrial Robot With One Simulation Environment*. Accessed: Oct. 10, 2023. [Online]. Available: https://robodk.com/
[106] MathWorks. (2023). *Robotics System Toolbox*. Accessed: Oct. 10, 2023. [Online]. Available: https://la.mathworks.com/help/robotics/
[107] N. Omniverse. (2023). *NVIDIA Omniverse—Isaac Sim*. [Online]. Available: https://developer.nvidia.com/isaac-sim
[108] C. Camargo, J. Gonçalves, M. Á. Conde, F. J. Rodríguez-Sedano, P. Costa, and F. J. García-Peñalvo, "Systematic literature review of realistic simulators applied in educational robotics context," *Sensors*, vol. 21, no. 12, p. 4031, Jun. 2021.
[109] S. Tselegkaridis and T. Sapounidis, "Simulators in educational robotics: A review," *Educ. Sci.*, vol. 11, no. 1, p. 11, Jan. 2021.

[110] M. Rojas, G. Hermosilla, D. Yunge, and G. Farias, "An easy to use deep reinforcement learning library for AI mobile robots in isaac sim," *Appl. Sci.*, vol. 12, no. 17, p. 8429, Aug. 2022.
[111] W. Liu, L. Peng, J. Cao, X. Fu, Y. Liu, Z. Pan, and J. Yang, "Ensemble bootstrapped deep deterministic policy gradient for vision-based robotic grasping," *IEEE Access*, vol. 9, pp. 19916–19925, 2021.
[112] D. Bogunowicz, A. Rybnikov, K. Vendidandi, and F. Chervinskii, "Sim2real for peg-hole insertion with eye-in-hand camera," 2020, *arXiv:2005.14401*.
[113] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke and S. Levine, "QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. 2nd Annu. Conf. Robot Learn.*, Zürich, Switzerland, 2018, pp. 651–673.
[114] F. Quiroga, G. Hermosilla, G. Farias, E. Fabregas, and G. Montenegro, "Position control of a mobile robot through deep reinforcement learning," *Appl. Sci.*, vol. 12, no. 14, p. 7194, Jul. 2022.
[115] NVIDIA. (2023). *CUDA Installation Guide for Microsoft Windows*. Accessed: Oct. 10, 2023. [Online]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html
[116] N. Omniverse. (2023). *Installation Guide of NVIDIA CUDNN*. Accessed: Oct. 10, 2023. [Online]. Available: https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html

CARLOS CALDERÓN-CORDOVA (Senior Member, IEEE) received the degree in electronics and telecommunications engineering from UTPL, Ecuador, in 2006, and the master's degree in electromechanics from UNL, Ecuador, in 2017.

He is currently the Director of the CONSYS-UTPL Research Group and the Coordinator of the LERAP-UTPL Prototyping and Innovation Laboratory. His areas of expertise are digital transformation of industry, industrial robotics, automatic control, and the Industrial IoT. He is the author of 38 scientific publications indexed in Scopus/Web of Science. He has also generated nine international patent applications. He has given keynote conferences and presentations at 49 national and international scientific events.

Prof. Calderon-Cordova has received important honors and awards, the following are highlighted: "Best Academic Inventor" National Award from CEDIA, Ecuador, in 2022 and 2024, "Honorable Mention of the 2021 SIGHT Volunteer of the Year" Recognition (IEEE, USA), in 2021, and "Ecuador Innova" Honor (Vice Presidency of the Republic of Ecuador), in 2014. He was the Chair of the IEEE Robotics and Automation Society Ecuador, in 2023. He was the Chair of IEEE SIGHT Ecuador (2020–2021) and the Co-Founder and the Executive President of the Technology-based company KRADAC (2010–2021).

ROGER SARANGO was born in Loja, Ecuador, in 1990. He received the B.S. degree in electronics and telecommunications engineering from the Universidad Técnica Particular de Loja and the M.S. degree in industry 4.0 from the Universidad Internacional de La Rioja.

He has worked on research projects with fractional control and simulated world robotics applications, such as the control of robotic manipulators and autonomous vehicles. He is currently the author or co-author of the scientific publications indexed in Scopus. His current research interests include the deployment of machine learning with deep reinforcement learning, computer vision for mobile robots, fixed, and autonomous vehicles in simulated environments.

DARWIN CASTILLO received the bachelor's degree in electronics and telecommunications engineering from the Universidad Técnica Particular de Loja, Ecuador, and the master's degree in biomedical engineering from the Universidad Politécnica de Madrid, Spain. He is currently pursuing the Ph.D. degree in mathematics with the Universitat Politècnica de València, Spain. He was an Associate Professor with the Universidad Técnica Particular de Loja. His research interests include computer vision, biomedical engineering, and math and educational innovation projects. He is a member of SPIE and IEEE EMBS Chapter.

VASUDEVAN LAKSHMINARAYANAN (Senior Member, IEEE) was a KITP Scholar with the Kavli Institute for Theoretical Physics, UC Santa Barbara; an Associate Professor with Michigan Center for Theoretical Physics; and has held research and teaching positions with UC Irvine, UC Berkeley, University of Michigan, and the University of Missouri. Currently, he is a Professor in optometry and vision science, physics, ECE, and systems design engineering with the University of Waterloo. His research interests include optical science and engineering, applied math, biomedical engineering, neuroscience, cognitive science, clinical ophthalmology and optometry, and history of science. He is on the optics advisory board of the International Center for Theoretical Physics, Trieste, Italy, and a Consultant to the Medical Devices Group, U.S. FDA. He has represented the United States at two IUPAP general assemblies. Additionally, he served as the Chair for U.S. Advisory Committee for the International Commission on Optics, the Chair of the Committee on International Scientific Affairs of the APS, a member of the Public Policy Committee of APS, a member of Executive Committee of the Forum on International Physics, an AAAS Science and Technology Policy Fellow, and the Director of the OSA amongst other professional service.