IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 October 2024, accepted 14 November 2024, date of publication 20 November 2024,
date of current version 29 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3504354

RESEARCH ARTICLE

Evaluating Task Optimization and Reinforcement Learning Models in Robotic Task Parameterization

MICHELE DELLEDONNE[1,2], ENRICO VILLAGROSSI[1], MANUEL BESCHI[1,2], (Member, IEEE), AND ALIREZA RASTEGARPANAH[3]
[1]Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, National Research Council of Italy, 20133 Milan, Italy
[2]Department of Mechanical and Industrial Engineering, University of Brescia, 25123 Brescia, Italy
[3]School of Metallurgy and Materials, University of Birmingham, B15 2TT Birmingham, U.K.

Corresponding author: Michele Delledonne (michele.delledonne@stiima.cnr.it)

This work was supported in part by the REBELION Project under Grant 101104241; and in part by the Lombardy, Italy Regional Project EcoCirc (deliberation XI/4730 of the 17/05/2021).

ABSTRACT The rapid evolution of industrial robot hardware has created a technological gap with software, limiting its adoption. The software solutions proposed in recent years have yet to meet the industrial sector's requirements, as they focus more on the definition of task structure than the definition and tuning of its execution parameters. A framework for task parameter optimization was developed to address this gap. It breaks down the task using a modular structure, allowing the task optimization piece by piece. The optimization is performed with a dedicated hill-climbing algorithm. This paper revisits the framework by proposing an alternative approach that replaces the algorithmic component with reinforcement learning (RL) models. Five RL models are proposed with increasing complexity and efficiency. A comparative analysis of the traditional algorithm and RL models is presented, highlighting efficiency, flexibility, and usability. The results demonstrate that although RL models improve task optimization efficiency by 95%, they still need more flexibility. However, the nature of these models provides significant opportunities for future advancements.

INDEX TERMS Reinforcement learning, robotic task optimization, task-oriented programming, intuitive robot programming.

I. INTRODUCTION

The ability of robots to perform increasingly complex tasks with unprecedented precision and speed has redefined the boundaries of industrial production [1]. However, outdated and inefficient programming methodologies often hinder the progress in hardware and control strategies [2]. Current programming interfaces are primarily based on textual programming, with each manufacturer developing its language [3]. This results in slow and tedious programming processes, requiring specialized operators with extensive knowledge and experience in robotics programming. Moreover, there is low flexibility and high reconfiguration times when the robot's task changes.

These limitations have spurred the research of more intuitive and user-friendly programming approaches. Intuitive methods, such as programming by demonstration, aim to convert human behaviors or messages into traditional software programs. These methods provide significant improvements in terms of intuitiveness and ease of programming. However, this simplification has specific issues, such as reduced precision or difficulty defining complex tasks. Additionally, the industrial environment could pose challenges for these methods, for example, due to noise and lighting conditions.

The Task Optimization Framework (TOF) [4] was developed to address these challenges, designed to self-tune the parameters necessary for task execution. TOF operates

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

through a trial-and-error process, employing a set of parameters to execute tasks and subsequently receiving a reward based on performance evaluation. This process is guided by an optimization algorithm that updates probability variables based on the rewards from previous trials. Although TOF has demonstrated effectiveness, it still has limitations, primarily due to the lack of detailed task knowledge that hinders more targeted parameter modifications.

Given these limitations, the integration of artificial intelligence (AI), specifically reinforcement learning (RL), has been explored to enhance the efficiency and adaptability of robotic systems. Over recent years, AI has significantly advanced robotics, enabling the execution of intricate tasks with remarkable autonomy and precision. Five RL models of increasing complexity were trained to replace the algorithmic component of TOF to improve its performance.

This paper details the methodology for selecting, integrating, and training RL models within the TOF framework, presenting the outcomes of various learning processes and a comparative analysis focusing on process efficiency and ease of use. The article is organized as follows: Section II analyzes related works, Section III provides an overview of TOF and the integration of RL, the case study description, and the methodology used for the learning process and tests. Section IV presents and discusses the results. Section V finally reports the conclusions and future work.

II. RELATED WORKS

Current robot programming methods involve textual programming, which means operators must manually write all the instructions for the robot's movements and actions, requiring a debugging and optimization phase; the operator must test and correct the code, which is a time-spending activity.

Robot actions are often defined sequentially and rigidly; robots follow a predefined movement pattern without the ability to adapt dynamically to environmental variations or job demands. This type of structure also makes it difficult to make changes or adjust the program in response to new needs or operational requirements [5].

This situation calls for developing new software technologies to make programming more intuitive, flexible, and adaptable to any robot and process [6]. Graphical programming works through a graphical representation of data flows and control [7], [8]. These use graphical elements, including blocks that identify different operations and functionalities, visual connections to interconnect the various blocks, and implement diverse logic, all through drag-and-drop. Methods that ensure modularity, reuse, and simplification of complexity. Programming by demonstration allows the operator to program a robotic task by personally performing it [9] or manipulating the robot to perform it [10]. The robot can generate a model from these demonstrations to autonomously replicate the task. Furthermore, during executions, the system can receive operator feedback to optimize the execution further. Natural language and voice control programming methods decode human language, written or spoken [11], [12], into specific operations that the robot can perform to accomplish the desired task. This aims to make robot programming a regular dialogue between the operator and the robot. Augmented and virtual reality are used in robotic programming to create an immersive and intuitive experience for developers [13]. Developers can interact directly with the robot and its work environment through headsets, displaying programming elements overlaid on the real world [14] or exploring dedicated virtual environments. Using gestures or voice commands, they can manipulate objects, add instructions to the robot's program, and test changes in real time. This approach offers a smoother and more collaborative programming experience, enabling a better understanding of the robot's behavior and facilitating the development of complex applications.

Despite these methods bringing notable innovation in intuitiveness and simplification of programming, they still present defects that prevent them from being used in an industrial context [15]. The task of simplifying programming often leaves gaps in terms of execution precision [16]. Simplifying the programming method tends to remove as much burden as possible from the operator, leaving the robot responsible for defining most of the process to be carried out. However, these techniques usually focus more on transforming the information provided by the operator into the structure of the task; the quality of execution is often overlooked. For example, pick-and-place programming focuses more on which object to pick up and which position it should take without worrying about the accuracy of picking and releasing. This is unacceptable in an industrial environment, considering the high standards required by production. Furthermore, industrial processing usually requires very complex tasks, and the simplicity of programming makes this goal very difficult [17] because the programming process requires many details that intuitive programming cannot define. There is also an adaptability problem; techniques that may be effective in a laboratory environment may not be in an industrial environment [18]. Vision systems may struggle in a non-perfectly lit environment or with variable brightness throughout the day. Vibrations due to processes may make it challenging to use sensors. The excessive noise characteristic of industries does not allow adequate detection of human voices. An industrial environment must also comply with safety regulations. Furthermore, these methods often require special equipment or a significant overhaul of industrial equipment, requiring time, effort, and additional costs.

These issues have led to the search for new solutions, including artificial intelligence (AI), which is becoming increasingly popular in robotics [19]. The ability of AI to learn and adapt to different situations and environments, generating intelligent models, provides powerful tools with capabilities superior to traditional algorithms [20]. AI is being applied across various fields of robotics, including path planning and replanning [21], object detection and recognition [22], and human activity recognition [23]. These advancements have generated tools that can be useful to integrate into robotic programming. Providing the robot with intelligence removes some of the operator burdens, and high-level programming could be possible thanks to task planning.

Task planning and adaptability are crucial for a robot to perform a task autonomously. The robot must define which operations are necessary to achieve a goal and in what order to carry them out. An important aspect is the ability to modify this structure to adapt to various unforeseen events that a work environment may present. AI algorithms are often used for this purpose [24]; specifically, reinforcement learning finds considerable use [25]. Equally important are the control algorithms that enable adaptive movement.

Robot control algorithms allow the robot to adjust movements based on external inputs, such as force sensors, enabling it to adapt to various task requirements, especially when simply defining a trajectory is insufficient, as in the case of interaction with the environment. In an assembly task, path planning alone is not enough to perform the task correctly, as it requires exchanging forces with different components, and the robot must react appropriately to them. Neural network approaches are ideal for solving these problems [26], [27]. Advancements in Large Language Models further enhance these capabilities.

Large Language Models significantly expand robots' interactive and cognitive capabilities. These are trained to understand and generate human language, making them ideal tools for improving human-machine interfaces. These models make it easier to program new robotic tasks through a common language [28]. They also allow a dialogue with the operator to define the task [29]. Among the various AI methods, reinforcement learning stands out.

RL is notably prominent among the methods proposed by AI. It aims to train a model or agent to interact optimally with an environment to achieve a specific objective. The agent interacts with the environment through certain actions, which alter the state of the environment, resulting in a reward that indicates the benefit of the action towards the final goal. Through the continuous execution of actions, the agent learns which actions optimize the reward, specifically its cumulative value. RL finds use in numerous areas of robotics [30], highlighting a notable use in manipulating objects [31].

III. MATERIALS AND METHODS

A. TASK OPTIMIZATION FRAMEWORK (BACKGROUND)

In [4], the authors proposed the Task Optimization Framework (TOF). The framework aims to autonomously define a robotic task's parameters through a trial and error process, ensuring a correct task execution. The framework's core is modular, providing an intuitive task definition structure for easy implementation. This structure defines tasks using three different levels: *TOF skills*, *TOF actions*, and the *TOF task*. *TOF skills* represent the primitive operations a robot can perform, such as actuating an end effector, linear movement, or force application. These *TOF skills* alone cannot generate any result; however, an ordered set of *TOF skills* creates a *TOF action*. *TOF actions*, which compose the second level of the structure, can be seen as the phases into which a task can be decomposed. For instance, in an assembly task, the manipulation of each component represents a *TOF action*. A characteristic of these *TOF actions* is their independence from one another; for example, object manipulation cannot be divided into separate *TOF actions* of picking and releasing because the position with which the object is picked influences its release position. The only dependency is the order in which they are performed, as certain assembly processes must follow a specific sequence (e.g., the final piece cannot be added before the preceding parts are assembled). Finally, an ordered set of *TOF actions* constitutes a task. Fig. 1 shows the *TOF task* structure.

The modularity of this structure is essential not only for intuitive definition but also for parameter definition. The simultaneous parameter definition can be highly complicated. Breaking the *TOF task* into *TOF actions* makes working on parameter subsets possible, simplifying the process.

Behavior trees were used to focus TOF's work on defining required parameters. Their modular tree-like structure is ideal for describing this type of structure, making it easy to program by an operator.

TOF uses a hill-climbing algorithm, presented in [4], to navigate the parameter space, searching for parameters that make the task effective. This process involves executing the task with different parameter values and receiving a reward that defines the quality of execution. Intelligent selection of future sets allows for effective reward improvement by trying to maximize it. It is summarized as follows.

The parameters are modified individually by choosing one of the three possible actions: increase, decrease, or maintain the same value. Equation (1) shows these actions, where p_k represents the parameter to use at the k-th iteration, p_b represents the parameter given by the best reward, δ_{max} represents the maximum allowable variation (it depends on the work area considered), rand[0:1] samples from a uniform distribution with a value between 0 and 1, v represents the action chosen from the three available actions v_i, v_d, and v_{eq}

$$p_k = \begin{cases} p_b + \delta_{max} \cdot \text{rand}[0{:}1], & \text{if } v = v_i \\ p_b - \delta_{max} \cdot \text{rand}[0{:}1], & \text{if } v = v_d \\ p_b, & \text{if } v = v_{eq}. \end{cases} \quad (1)$$

The action to be performed is selected using probability variables associated with each action. Equation (2) shows the principle of this choice. $P(v_i)$, $P(v_{eq})$, and $P(v_d)$ represent, respectively, the probability variables associated with the relative actions v_i, v_{eq}, and v_d.

$$v = \begin{cases} v_i, & \text{if } r_v < P(v_i) \\ v_{eq}, & \text{if } P(v_i) < r_v < (P(v_{eq}) + P(v_i)) \\ v_d, & \text{if } r_v > P(v_{eq}) + P(v_i), \end{cases} \quad (2)$$

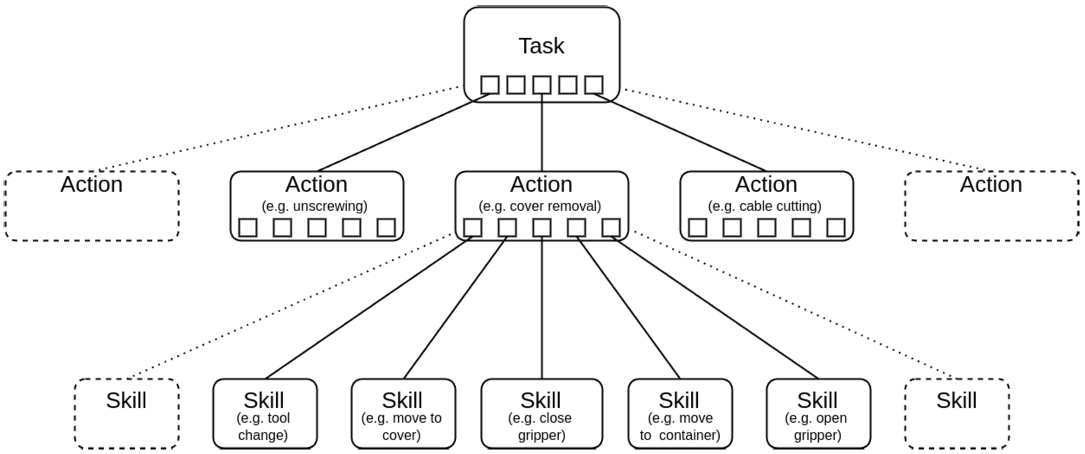


FIGURE 1. Hierarchical structure of TOF task, actions, and skills.

where

$$r_v = \text{rand}[0{:}P_s] \ , \tag{3}$$

represents a random value ranging from 0 to the sum of the probability variables

$$P_s = P(v_i) + P(v_{eq}) + P(v_d) \ . \tag{4}$$

Based on the obtained reward, probability variables related to the performed actions are updated, increasing the probability of actions that led to a reward increase and vice versa. Equation (5) shows the evolution of these variables, where k is the iteration number.

$$P(v)|_k = \begin{cases} P(v)|_{k-1} \cdot \left(\dfrac{k-1}{k}\right) + \dfrac{1}{k}, & \text{if } R > R_o \\ P(v)|_{k-1} \cdot \left(\dfrac{k-1}{k}\right), & \text{if } R \leq R_o. \end{cases} \tag{5}$$

Notably, the search remains near the best current solution, only changing the search area if the reward improves. This feature prevents the algorithm from diverging, which is a significant risk given the usually high number of parameters to modify.

Algorithm 1 shows the TOF process. `task_execution` is a function that performs the robotic task using the current task parameters (p_{curr} and p_{init} for the first execution), r indicates the reward obtained from the evaluation of the task (`task_evaluation`), r_{best} represents the best reward obtained during all the process and p_{best} the relative parameters set, k is the iteration number. It is required for the probability variables (P) update. v is the vector containing the normalized parameter variation. It is obtained by the function `actions_selection` using (2), `params_modification` is the function that uses (1) to update the parameters, `P_increment` and `P_decrement` implement (5).

The hill-climbing process can be problematic if conducted directly in a real environment. Using unknown task parameters can make the robot's behavior unpredictable, potentially leading to harmful actions for the robot or operators. Therefore, the initial parameter definition process is performed in a simulated environment. The proposed methods use PyBullet [32], a physics engine and simulation library widely used in robotics, physics-based simulations, and reinforcement learning.

Algorithm 1 TOF Algorithm

```
Input: $p_{init}$                         ▷ Initial task parameter set
Output: $p_{best}$                   ▷ Optimized task parameter set
  task_execution($p_{init}$)
  $r$, $success$ ← task_evaluation()
  $r_{best} \leftarrow r$
  $p_{curr} \leftarrow p_{init}$
  $p_{best} \leftarrow p_{init}$
  $k \leftarrow 0$
  while not $success$ do
      $k \leftarrow k + 1$
      $v$ ← actions_selection($P$)
      params_modification($p_{curr}$, $v$)
      task_execution($p_{curr}$)
      $r$, $success$ ← task_evaluation()
      if $r > r_{best}$ then
          $p_{best} \leftarrow p_{curr}$
          P_increment($P$, $k$)
      else
          $p_{curr} \leftarrow p_{best}$
          P_decrement($P$, $k$)
      end if
  end while
```

B. REINFORCEMENT LEARNING INTEGRATION

The task decomposition in *TOF actions* makes RL a suitable tool. Theoretically, a well-trained agent could compute the optimal parameters in a limited computation time without needing hill-climbing optimization. In a more realistic scenario, the agent output may allow the optimizer to

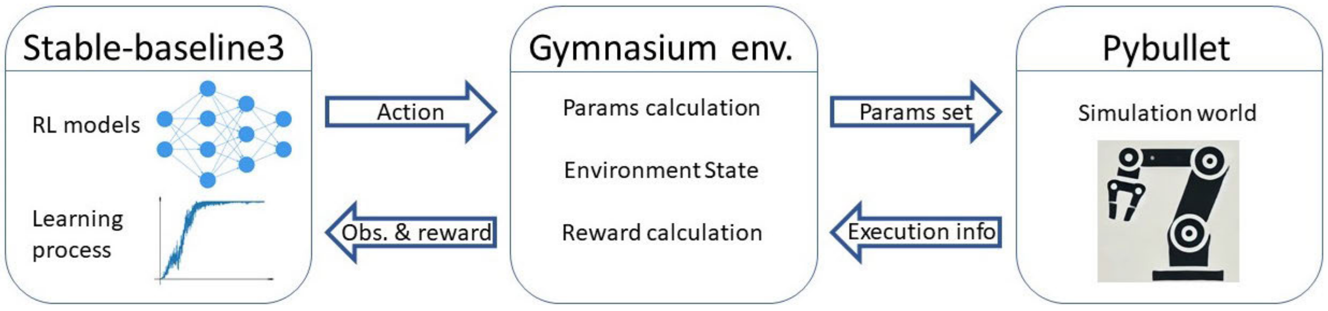


FIGURE 2. **Software structure for RL models training.**

converge in a few generations. However, using RL shifts the design effort from optimizing the parameters space to training the agent to generalize the solution of real-life *TOF actions*.

The Gymnasium [33] library has been used to generate the working environment and the Stable-Baselines3 [34] library for the agent or model to be trained. Both are based on libraries provided by OpenAI, making them suitable for working together. Gymnasium offers pre-developed environment models and a base class from which anyone can generate a custom environment. Stable-Baselines3 provides ready-to-train reinforcement learning algorithms, which can be customized through parameters.

As previously described, TOF focuses on defining the parameters that characterize a task. The algorithmic optimization component navigates the parameter space to find a set that effectively accomplishes a task. To optimize a complex task, all *TOF actions* must be optimized as follows from the Bellman principle of optimality.

A reinforcement learning process requires the definition of specific components. The *environment* is the component to interact to achieve a specific goal. In TOF, this is the simulation or real environment. Specifically, at the software level, the gymnasium environment is the interface that communicates with the simulator or the real robotic cell. It allows for the reception and transmission of the data necessary for training. The data provided to the robot are the parameters that characterize the task. The information received can vary depending on the task to be performed. This information defines the environment *state*, such as the positions of objects in the scene or the robot configuration. Part of this information is used as input for the model and is called *observation*. The *agent* is the model to train, which has the task of learning how to interact best with the environment to achieve the set goal. These interactions are called *actions*; in TOF, these represent the variation of the robotic task parameters. Finally, the *reward* is the feedback provided to train the model that informs it of how much the previously executed action has benefited or not in achieving the goal. In the case study, the reward is evaluated based on the quality of the execution performed. It is provided only during the learning process; it is not part of the model input. Models are usually trained in environments where it is easy to obtain the reward, unlike the usage environment such as the real world. Fig. 2 shows how the learning components are connected.

Given that the agent must work in a highly complex environment with a high risk of divergence, the TD3 was the model choice. This model has characteristics that provide high stability and the capability to work with very complex environments.

C. TASK DESCRIPTION

The task chosen to validate the proposed method and train the RL models is a peg-in-hole operation, where an object must be inserted into a designated hole. This task was selected for its simplicity in definition, as it requires few operations, but it is also relevant for assembly activities that require force interactions, the ones on which our research focuses. However, it presents challenges due to the dependency of grasping positions on the inserting position and the non-trivial insertion process involving multiple interaction forces.

The execution of a peg-in-hole process consists of interdependent operations, where the grasp position influences the insertion position. This requires incorporating the process into a single *TOF action* so it can be analyzed as a whole; this *TOF action* is called *object_peg_in_hole*. At a high level, it may only require defining which object to manipulate and which location to occupy.

In the case study, the task is represented by a single *object_peg_in_hole* action. The *object_peg_in_hole* action can be divided into (i) moving to the grip position, (ii) closing the gripper, (iii) moving to the insertion position, (iv) inserting, and (v) opening the gripper for release. This subdivision allows us to define this action using only three *TOF skills*. *Move_to* skill was used for moving the robot from position A to B, generating and executing a free-collision trajectory. The final position reached by the gripper is defined by providing the name of the desired reference frame and the relative translation and rotation. *Gripper_operation* skill was used for managing the gripper, and defining how the gripper should operate. The main parameter of this skill is whether the gripper should open or close its fingers. If the robotic gripper allows, the force applied during the action or the finger positioning width can also be specified. *Tactile_movement* was used for the insertion operation, allowing movements at a constant speed until a force limit is reached.[1] The movement speed can be defined in both translation and rotation. The target force can be defined as a total force or by setting limits for each direction and rotation.

The task involves symmetrical objects around their axis, making them approximate simple geometric shapes like cylinders, boxes, or cones. It is assumed that the object is placed vertically on the working surface, and the axis and position of the insertion hole are known. Assuming that the grasping position is perpendicular to the working surface and the reference position is parallel to the hole axis simplifies the parameter definition process. This way, relative rotations were removed from the possible parameters to set. Working with non-delicate objects, the simple function of opening and closing the gripper was used without worrying about specific forces and positions. The remaining parameters are the grasping and insertion positions, focusing only on translation.

In particular, the task used in the learning phase involves a cylinder with a height of 10 cm and a base radius of 3 cm. The hole, instead, has a radius of 3.2 cm and a depth of 7 cm. Fig. 3 shows the working environment in the task's three fundamental moments: the initial state, the grasping phase, and complete insertion.

The parameters optimization for this task has two main objectives. The first is to correct any errors made during the programming phase, whether performed by an operator or by taking default parameters. The second objective is to address potential vision-system errors. The grasping and insertion positions are relative to the detected object position and must adapt to any detection inaccuracies.

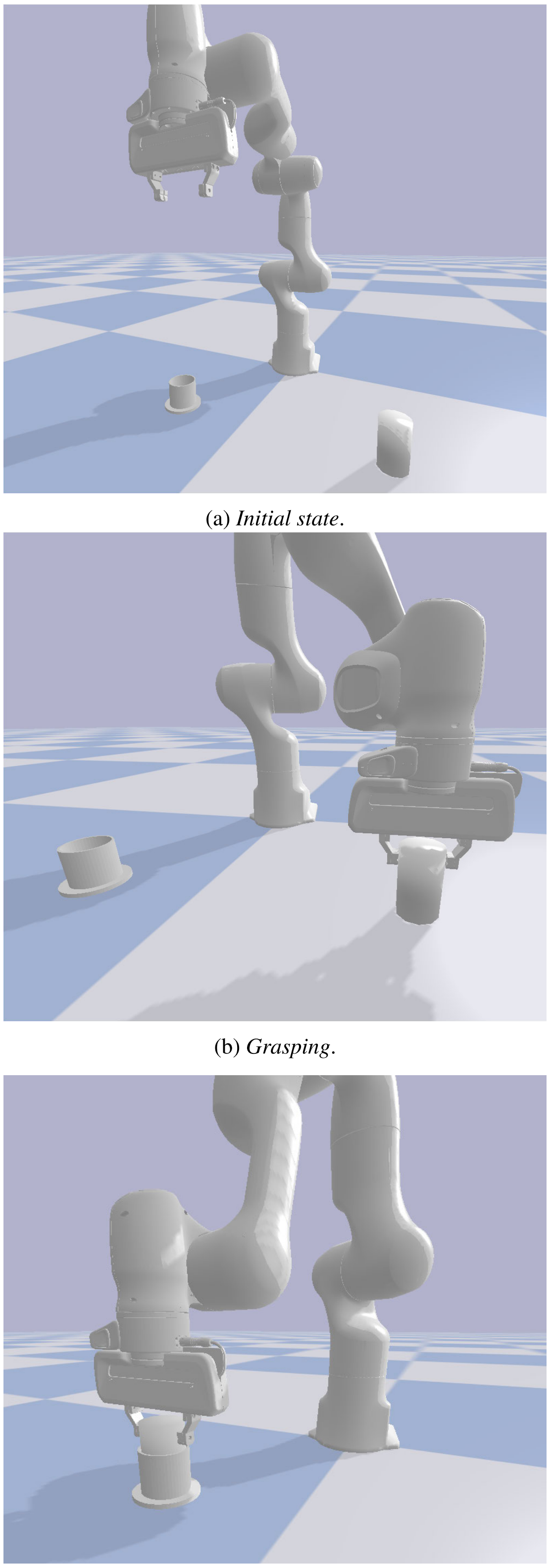
1) MULTI-ACTION TASK

The optimization of individual actions composing a task corresponds to the optimization of the entire task. Experiments were conducted on a more complex assembly scenario involving a series of consecutive peg-in-hole insertions to validate that. Fig. 4 illustrates the objects being handled and how they should be assembled. The task is divided into the following actions: inserting the bottom part of a container into a holder to restrict its movements, placing the desired object into the container, and closing the container. Each pairing between the objects under examination has a clearance of 2 millimeters, making it comparable to the previously described task. Separate models for grasping and insertion were selected to make the data acquisition process lighter and faster.

D. REINFORCEMENT LEARNING MODELS

The objective of the learning process is to obtain an agent capable of navigating within the parameter space to provide a set that generates the successful task. This model must cope with uncertainties that can arise when working with a real environment, ranging from object detection errors to programming errors by the operator. In the case study, possible errors are represented by incorrect settings of

[1] the force measure involves using a Force/Torque sensor.



(a) *Initial state*.

(b) *Grasping*.

(c) *Insertion*.

FIGURE 3. Task execution in the simulation environment.

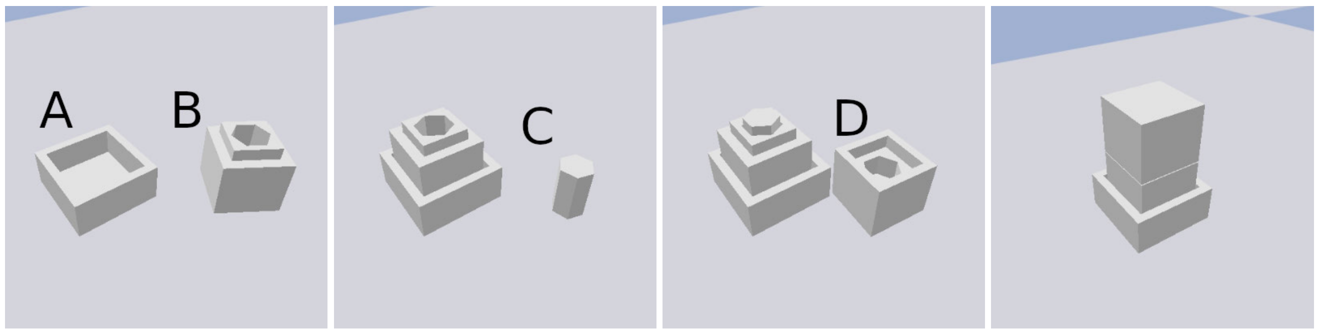


FIGURE 4. **Multi-action task objects and desired final configuration. Objects: holder (A)(fixed), container bottom (B), internal object (C), container lid (D). Operation: insert B in A, insert C in B, and place D on C and B.**

the grasping and insertion positions or by incorrect object identification, therefore not knowing its exact position. This section presents the training approach of five models designed with increasing complexity but also with increasing efficiency.

1) POSITION MODEL

The first model was trained using information as similar as possible to that provided by the optimization algorithm integrated into TOF. The input to the algorithm consists of the set of parameters to be optimized and the reward obtained from the executions. By design, the Reinforcement Learning model does not use the reward as an input; the reward is only utilized during the training phase to evaluate the model's action goodness in response to the input received. Therefore, only the current set of skill parameters remains from the algorithm input. However, using only the set of parameters does not provide enough information for training. These parameters represent the variation of the physical quantities that define the robot task under study, but they have no physical meaning. Therefore, the model is also provided with the influenced physical quantities as input, in this case, the relative position of the gripper and the insertion position.

The RL structure is, therefore, based on an *observation* composed of the current set of parameters and the respective grasping and insertion positions. The *action* represents the modification of the parameters; it is a vector of float values that range between -1 and 1. These values are then multiplied by the maximum possible variation, the same as that provided by the hill-climbing algorithm. In this way, the parameters can be increased or decreased within a variation contained in a known range. The *reward* is the evaluation of the task execution. It takes values between -1 and 1. Its value is defined by analyzing the different conditions in which the environment can be found. The reward can take on the following values:

- from -1 to -0.3 if the object is not grasped. The reward increases as the distance between the grasping and detected object positions decreases. It wants to bring the grasping position closer to the object;
- from -0.3 to 0.3 if the grasp position collides with the object. The reward increases the more aligned the grasp position is with the object detection concerning the vertical axis, and the further it is from the floor, it must move centrally and above the object;
- from 0.3 to 1 if the object is grasped increases as the final distance between the object and the desired position decreases.

The defined thresholds have values that are convenient for dividing the reward into three areas. The physical quantities are processed so that the reward falls within these thresholds for each of their values.

Equation (6) is used to generate the reward, where d_{g_o} indicates the distance between the grasp position and the detected object, $d_{h_{g\_o}}$ is the same distance but only along the horizontal plane, d_{f_o} is the distance between the object and floor along the vertical axis, and d_{o_t} the distance between the detected object and the target position.

$$R = \begin{cases} 1 - d_{o\_t}, & \textit{if grasped} \\ 0.0 - d_{h_{g\_o}} + d_{f\_o}, & \textit{if collision} \\ -0.3 - d_{g\_o}, & \textit{else}, \end{cases} \tag{6}$$

with $d_{o\_t} \in [0, 0.7]$, $d_{h_{g\_o}} \in [0, 0.3]$, $d_{f\_o} \in [0, 0.3]$ and $d_{g\_o} \in [0, 0.7]$. The distances are saturated with the reported value, the distances are measured in meters, and the definition of the simulation environment makes it challenging to saturate these limits.

A single trial corresponds to a learning step of the entire task execution; therefore, the training process requiring many steps implies unacceptable learning times, leading to ineffective training in a simulated environment. Overcoming this problem, the *Position model* was trained in a fictitious environment that does not use simulation, designed to reproduce the reward's behavior concerning the parameter values, drastically reducing the times. This fictitious environment has the same characteristics as the previous one.

2) GENERIC POSITION MODEL

To expand the capability of the *Position model*, a model was trained on multiple types of objects, providing the object's

TABLE 1. **RL model summary.**

Model	Input	Description
Position	Parameter set Grasping position Insertion position	Focused on a single object, it modifies parameters based on the distance of the detected object position to the grasp and hole positions.
Generic position	Parameter set Grasp. & Insert. positions Object info	Generalized on symmetric objects, it modifies parameters based on the distance of the detected object position to the grasp and hole positions. It also considers the object type and its dimension.
Grasping forces	Parameter set Grasping position Grasping forces history	It modifies grasping parameters based on the interaction forces collected during the grasping phase.
Insertion forces	Parameter set Insertion position Insertion forces history	It modifies insertion parameters based on the interaction forces collected during the insertion phase.
Peg-in-hole forces	Parameter set Grasp. & Insert. positions Grasp. & Insert. forces history	It modifies parameters based on the forces collected during the grasping and insertion phase.

dimensions and type considered as information. Selected objects have symmetric geometries concerning their axis to be attributable to simple geometries such as box, cylinder, cone, or sphere. The model name is *Generic position model*.

3) GRASPING FORCES MODEL AND INSERTION FORCES MODEL

As a further improvement, the history of interaction forces was observed during the grasping and insertion phases. The force history should help the model understand where the object is located. Due to the fictitious environment, recording the interaction forces during each execution is impossible. Therefore, a recording process was carried out using the simulation environment, modifying the grasping and insertion positions within a grid to record the interaction forces in each situation. During the learning phase, the fictitious model takes the obtained histories and performs interpolation to generate information even for all intermediate positions, and for all ones that do not fall within the grid, the forces are considered zero. The grid of positions used to obtain the data is composed of x and y values between $\pm 2cm$ with an interval of $1mm$ to obtain 1600 positions. These positions are relative to the correct position to record in the affected area. The precomputed histories were carried out by separating the grasping phase from the insertion phase; specifically, the insertion phase was performed using a perfect grasping position. This approach does not consider the dependency between grasping and insertion, so it was impossible to train a model that simultaneously worked on both phases. However, two separate models were retained, one for grasping and one for insertion.

This RL structure presents an *observation* composed of the current set of parameters, the respective considered positions, and the history of the measured forces. The *action* remains to modify the parameters. The reward takes values between 1 and 0 when the considered positions fall within the known range; if the object is not touched or the insertion is entirely outside the hole, it becomes impossible to modify the position sensibly. The reward value decreases as the position moves away from the correct one. If the position does not fall within the known range, the reward takes values from 0 to -1. It decreases as the distance between the current and theoretical correct positions increases. Theoretical correct position means the position that would be correct if the object was actually in the detected pose. It is, therefore, considered that the vision-system error is not larger than the record area, so the model can always bring itself back to a position that falls within the known range. Equation (7) shows the formulas used to generate the reward R, where $d_{p_{co}\_p_{cu}}$ indicates the distance between the current and correct positions, and $d_{p_{cu}\_p_{th}}$ is the distance between the current and theoretical positions. As in (6), distances are saturated and measured in meters.

$$R = \begin{cases} 1 - d_{p_{co}\_p_{cu}}, & \textit{if in known range} \\ 0.0 - d_{p_{cu}\_p_{th}}, & \textit{else}. \end{cases} \tag{7}$$

with $d_{p_{co}\_p_{cu}} \in [0, 1]$ and $d_{p_{cu}\_p_{th}} \in [0, 1]$. The model that uses the grasping force history is called *Grasping forces model*, the other one *Insertion forces model*.

4) PEG-IN-HOLE FORCES MODEL

New force histories were collected to obtain a model that considers the dependency of the two phases and works simultaneously on both. In this phase, the forces were recorded considering the complete task, so each grasping position was associated with all the insertion positions and vice versa. By maintaining the previous grid, the number of recordings would have become too burdensome. It was, therefore, decided to modify the grid, maintaining the same range but with an interval of $5mm$, to obtain a grid of 64 points for each position and, therefore, a set of 4096 combinations. However, the structure of the environment remains unchanged, except that the single grasping or insertion position is no longer considered, and both are considered simultaneously. This model was finally called *Peg-in-hole forces model*. Table 1 shows a quick summary of the models proposed in Section III-D.

TABLE 2. RL model evaluation indices.

Model	Position	Generic position	Grasping forces	Insertion forces	Peg-in-hole forces
Simulation time [min]	0	0	250	300	1400
Learning steps [n]	100000	2250000	90000	52000	375000
Learning time [min]	7	160	40	30	300
Programming error goal [bool]	True	True	True	True	True
Vision-system error goal [bool]	False	False	True	True	True
Required step [n]	2	3	15	14	5

IV. RESULTS AND DISCUSSION

A. REINFORCEMENT LEARNING RESULTS

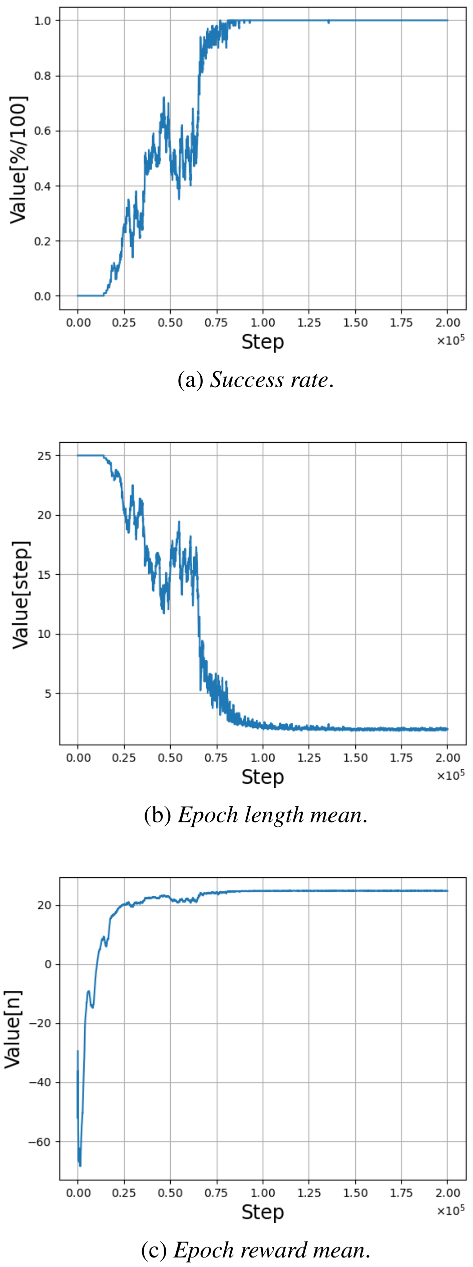
This section presents the results obtained training the models described in Section III-D, analyzing them individually. The following indices report the models' efficiency and computational burdens since the solution needs to be easily adaptable to new requirements.

- *Simulation Time* indicates the time used to collect data using the simulation environment.
- *Learning Steps* represent the steps required to achieve and maintain a full success rate.
- *Learning Time* defines the time needed to complete all the Learning Steps.
- *Programming error goal* shows if a model can fix a possible programming error.
- *Vision-system error goal* shows whether the model can cope with the vision-system inaccuracy.
- *Required step* indicates the steps that the model, after learning, needs to provide a correct set of parameters.

Table 2 reports the indices computed for all the models.

Position model presents a zero-simulation time because it avoids simulation usage by design. Training a model using an environment that included programming and object detection errors was impossible. This model improved reward by reaching the grasping area. Once here, the model does not have enough information to correct the vision-system error. For this reason, the indices are obtained by training the model using an environment without object detection errors. Fig. 5a shows that the model learned to solve the problem successfully after only 100000 steps, corresponding to 7 minutes. Fig. 5b demonstrates how the model reduces epoch length to around two steps, requiring very few steps to provide a successful parameter set. Fig. 5c depicts the evolution of the reward. The drawback of this model is its inability to handle potential vision-system errors and its capability to provide the theoretical grasp position only relative to an object.

Generic position model also presents a zero simulation time, as the only environment modification compared to the previous case is the generalization of the object. Like the previous model, this one can only learn from an environment without object detection errors. Fig. 6a illustrates how this model's learning requires exponentially more steps than the previous one; achieving a success rate of 1 takes around 1250000 steps, but for greater stability, it requires approximately 2250000 steps, totaling 160 minutes. Fig. 6b shows that the model learns to provide the desired solution with an average of 3 steps. Fig. 6c demonstrates a rapid initial increase in reward; the model quickly learns to position itself



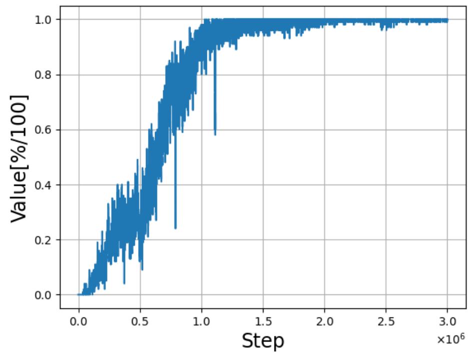
(a) *Success rate.*

(b) *Epoch length mean.*

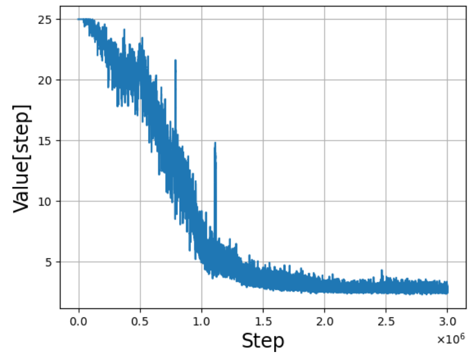
(c) *Epoch reward mean.*

FIGURE 5. ***Position model*** **learning results.**

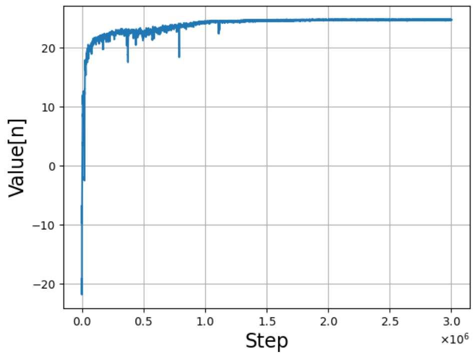
in the grasp area, with most of the learning period dedicated to increasing the success rate percentage. This model addresses object generalization but not detection errors. It can be highly useful in the initial phase of defining positions, providing a good set of parameters to start with and correcting vision-system errors.



(a) *Success rate.*



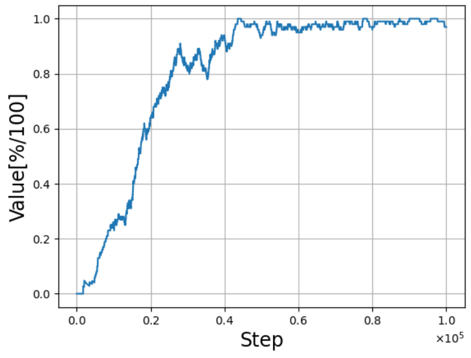
(b) *Epoch length mean.*



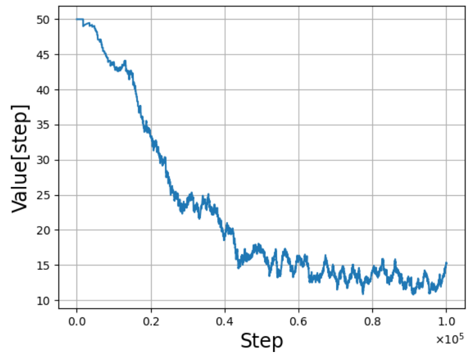
(c) *Epoch reward mean.*

FIGURE 6. ***Generic position model*** **learning results.**

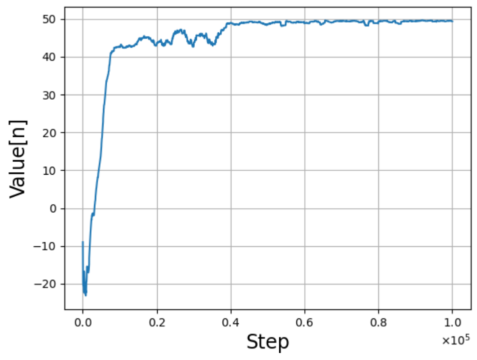
Grasping forces model is trained using an environment that utilizes data obtained from the simulation world. The time required to record the various interactions from different positions is 250 minutes. Fig. 7a shows that the model achieves a success rate of 1 after 45000 steps but reaches stability after 90000 steps. The success rate does not stabilize at 1, which may be because different positions around the correct position can produce similar forces. The flat surface of the gripper fingers ensures that small movements parallel to this surface do not produce variations in force. Fig. 7b illustrates that the model learns to provide a solution in fewer than 15 steps. The number of steps is higher than in previous models but is justified by the significant increase in problem complexity. Fig. 7c shows an initial rapid increase in reward followed by a drastic slowdown, which may again be attributed to the similarity of data around the correct position.



(a) *Success rate.*



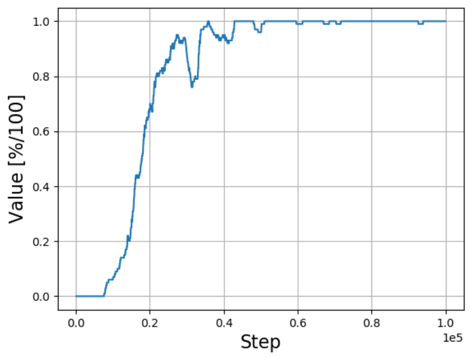
(b) *Epoch length mean.*



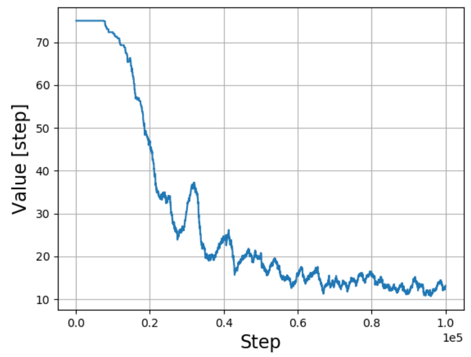
(c) *Epoch reward mean.*

FIGURE 7. ***Grasping forces model*** **learning results.**

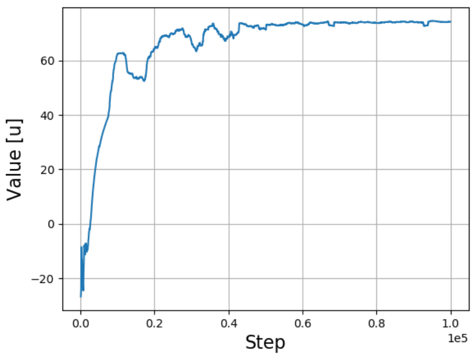
Insertion forces model also utilizes data from the simulation world. The recording time is 300 minutes, which is longer than the previous model because the insertion operation takes slightly more time than grasping. Fig. 8a demonstrates how the model learns to solve the problem successfully after 52000 steps, with the success rate remaining stable. Compared to the previous model, the learning data is more distinctive. Fig. 8b highlights that the model can solve the problem in fewer than 14 steps. Despite the increased data characterization, the resolution speed remains similar to the previous model. Fig. 8c shows a trend without a stagnant zone, as the insertion data is more distinctive.



(a) *Success rate.*



(b) *Epoch length mean.*



(c) *Epoch reward mean.*

FIGURE 8. ***Insertion forces model*** **learning results.**

Peg-in-hole forces model utilizes the simulator for 1400 minutes. The time is significantly higher than previous models because, despite reducing the number of grasping and insertion positions, their combination exponentially increases the number of required recordings. Additionally, each recording involves executing the entire task, requiring more time. Fig. 9a shows a stable success rate of 1 after 375,000 steps. Despite the not extremely high number of steps, the learning time is longer at 300 minutes. Using data as the model's state requires high memory usage, slowing down computation speed; this effect is also observed in the previous two models, grasping and insertion. Fig. 9b demonstrates that this model is more efficient than the two separate models, achieving an average of 5 steps per epoch. This illustrates that training a model considering relationships between phases aids the model. The reward trend in Fig. 9c exhibits a smooth progression without significant issues.

B. TOF RESULTS

This section will briefly show the results presented in the previous work [4]. These results were obtained using the algorithm presented in Section III-A on a pick-and-place task. Table 3 shows the results related only to the optimization phase, as the framework does not require initial phases where a model needs to be trained. It can be noted that the presented results show different algorithms because, in addition to the algorithm shown previously, variants were created to make comparisons. Algorithm 2 modifies the search area in the state space to better adapt to the requirements. Algorithm 3 presents a different assignment of weights to previous rewards, trying to give more value to the most recent ones. Algorithm 4 implements both modifications. The table shows the average iterations needed to reach a goal. 'Centering' is the goal that indicates the number of steps needed to align the peg (in this case, it was a can) with the hole. 'Success' indicates a complete insertion of the can into the hole. The averages were obtained from the results of 20 optimization processes for each algorithm.

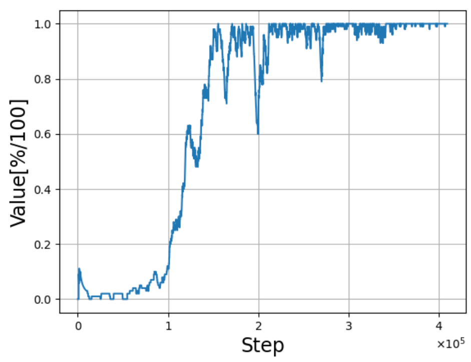
TABLE 3. TOF tests result. Centering: iterations to obtain can centering. Success: iterations to perfect execution (500 is the maximum iteration number in the simulation).

	Alg. 1	Alg. 2	Alg. 3	Alg. 4
Centering	56	73	93	48
Success	156	118	319	145

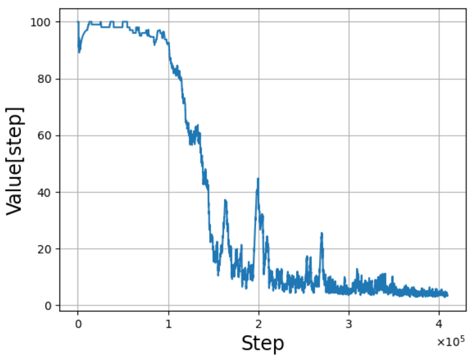
It is immediately noticeable that the iterations required to achieve a correct result are higher than those obtained with RL models. This highlights how a customized tool is more efficient.

C. MODELS AND ALGORITHMS COMPARISON

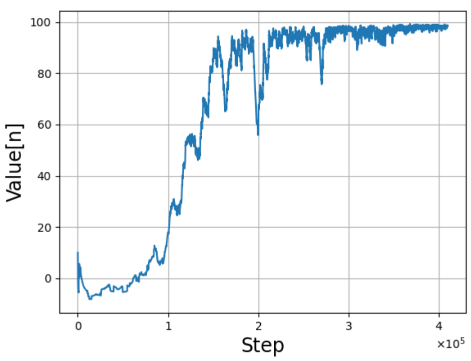
This section compares the TOF approach presented in [4] and the approach, based on RL, presented in the current paper solving the peg-in-hole task. Three aspects are considered for the comparison: *ease of use* and adaptation (*i.e.,* the effort



(a) *Success rate.*

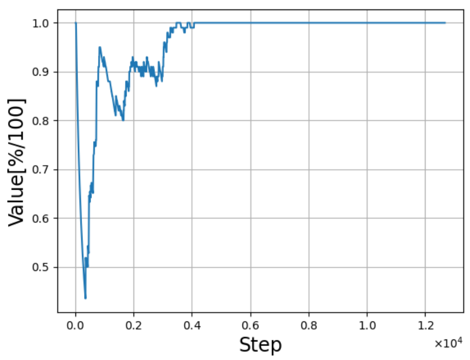


(b) *Epoch length mean.*

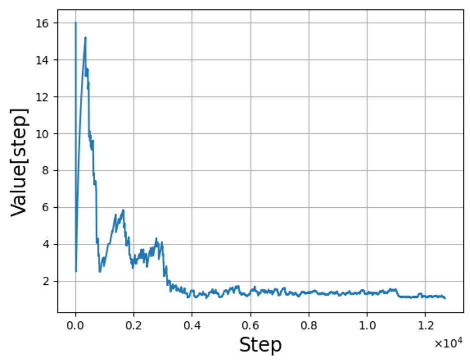


(c) *Epoch reward mean.*

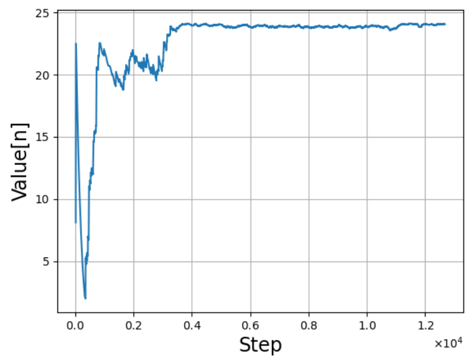
FIGURE 9. ***Peg-in-hole forces model* learning results.**



(a) *Success rate.*



(b) *Epoch length mean.*

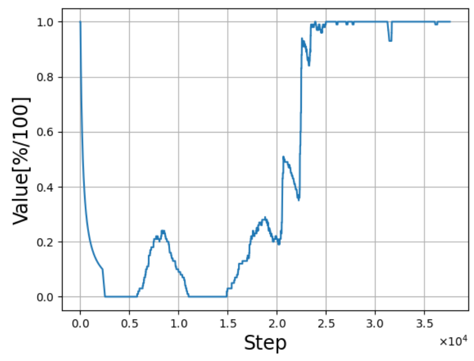


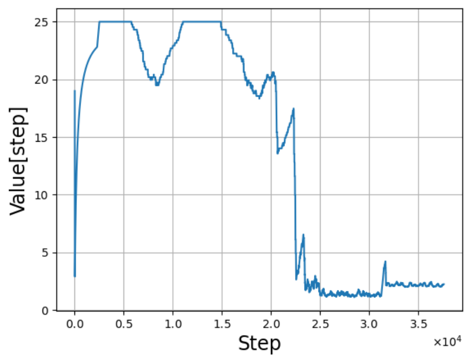
(c) *Epoch reward mean.*

FIGURE 10. **Container bottom *Insertion forces model* learning results.**

required by an operator to use one of these methods when approaching a new task); *process efficiency* (*i.e.,* how efficient and valuable a method is and under what conditions); *future potential* (*i.e.,* how the various methods could evolve and identify the most interesting ones for development).

Regarding *ease of use*, both methods share some common features. The operator is burdened to define a new work environment and task and must provide the necessary information for the optimization phase. This specifies which parameters to modify, their range, and which indices to use for the reward calculation. TOF does not require anything more to be operational. On the other hand, RL models require additional steps. For models that do not use data, a new dedicated environment must also be defined each time a new model needs to be trained. This is because these environments simulate the rewards generated by a specific robotic task, so if the task changes, the environment must also change. For models that work with historical force data, the environment can be made generic, able to adapt to a different number of parameters and different data dimensions, customizing which phases of the task to record. The drawback of these models is the initial data collection. Even for this process, it is possible to generate generic software, but this process takes time, which increases proportionally with the length of the task.


(a) *Success rate.*


(b) *Epoch length mean.*

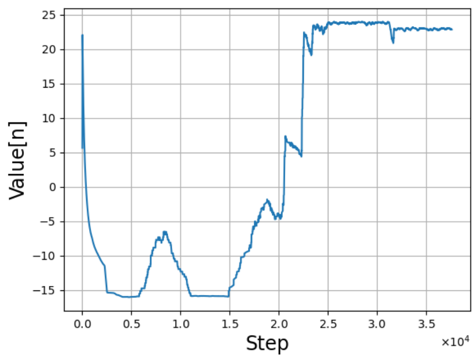
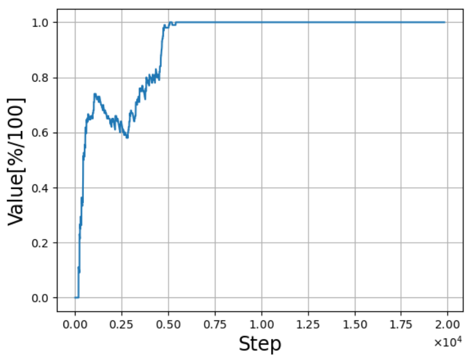
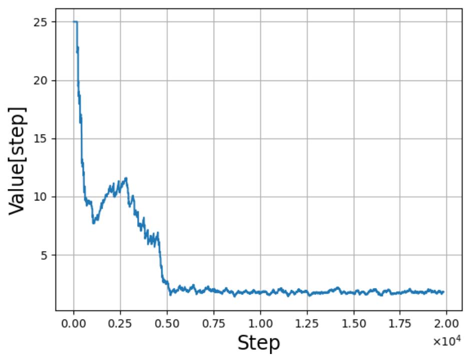

(c) *Epoch reward mean.*

FIGURE 11. Internal object ***Insertion forces model*** learning results.


(a) *Success rate.*


(b) *Epoch length mean.*

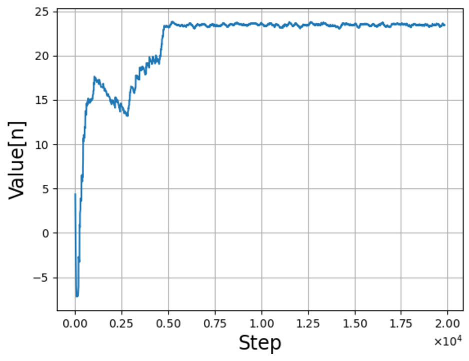

(c) *Epoch reward mean.*

FIGURE 12. Container lid ***Insertion forces model*** learning results.

Speaking about process efficiency, the RL models are significantly better. Aside from the Theoretical Position and Generic Theoretical Position models, which only partially meet the requirements for a successful task, the other models have a much lower number of steps per process than can be achieved using TOF. RL models have an efficiency better than 95%. This shows that a customized model is necessarily better than a generalized algorithm. However, since they require a costly data collection phase, these models are more useful when repeatedly working with the same objects. The high variability of the scene objects greatly complicates the entire adaptation process regarding RL models.

Future potential of the algorithms used by TOF is possible, although their modification would likely not result in significantly better efficiencies than the current ones. Their growth potential is insufficient to allow them to reach the efficiencies achieved with RL models. In contrast, RL models still have enormous growth potential. For example, consider a model that learns from data collected on different types of objects and with different dimensions. Of course, the data acquisition process would be much more burdensome. Still, it would generate a much more flexible model, eliminating the adaptability problem to different tasks while maintaining the same efficiency.

Overall, the TOF framework provides a more flexible tool with the same usage times. For RL models, the time necessary for data collection must be considered. However, RL models present significant development possibilities that could lead to having an equally flexible and more efficient tool. The only drawback of these new models would be a high data collection time, having to record data for various situations, but then compensated by its execution speed. This time requirement makes the method suitable only for producing medium-large batches, where reschedules are rare and training is done offline.

D. MULTI-ACTION TASK RESULT

This section presents the results of training RL models on a multi-action task. The task under examination is introduced in Section III-C1, highlighting the training of separate models for grasping and insertion. Regarding the grasping task, the geometry of the considered objects resulted in the inability to train any model successfully. The gripping surfaces are flat, meaning infinite gripping positions exhibit the same interaction forces. Therefore, assuming a maximum localization error of 0.5 cm for the objects, the training focused on the insertion models.

Figures 10 11 and 12 show the training results of the models. A significant improvement in learning speed is observed; the values range between 4000 and 5000 steps for the two container parts (elements A and B in Figures 4a and 4b). At the same time, for the internal object, they reach 20,000 steps (element C in Figure 4c). This marked improvement can be attributed to the geometry of the parts. The square base of the container allows for a millimeter of mechanical clearance on both sides, allowing more clearance along the diagonals. The same reasoning applies to the hexagon but more strictly explains the increase in learning steps.

This demonstrates the effectiveness of these optimization methods even for tasks composed of multiple actions.

V. CONCLUSION

This paper compares two methods for defining the robotic task parameters using a trial and error approach. The first method uses the Task Optimization Framework, a tool based on a dedicated algorithm developed by authors in [4]. The second presents an alternative solution where the algorithm is replaced by RL models trained for the specific task.

The results are obtained by evaluating the performance of both methods in solving a peg-in-hole problem. These show efficiency better than 95% with the use of RL models. However, this is accompanied by poor flexibility, as the models are trained to solve a specific task. In contrast, TOF shows excellent flexibility but requires considerable time for each optimization process. It is, therefore, evident that for similar tasks where training new RL models is not required, the latter are much more useful. In more general tasks, TOF proves to be the better choice. The potential for method evolution greatly favors RL models, as they could become superior in every aspect with the possibility of greater generalization.

Despite the demonstrated potential, both methods are still limited by the simulation constraints. The optimization process is slowed down significantly by long simulation times. Additionally, not all processes can be accurately simulated. Tasks involving the manipulation of flexible or soft objects, or interactions with very complex geometries, such as the interaction of a thread and a screw, continue to pose challenges for achieving reliable simulations.

In future work, the authors aim to develop new RL models that generalize tasks as much as possible to obtain models capable of adapting to different environments and object types. Experiments will also be conducted on diverse tasks to validate the effectiveness of the proposed methods with generic tasks. Simulators such as Isaac Lab, which allow parallel simulations to reduce process times drastically, will also be considered.

REFERENCES

[1] A. Gasparetto and L. Scalera, "A brief history of industrial robotics in the 20th century," *Adv. Historical Stud.*, vol. 8, no. 1, pp. 24–35, 2019.

[2] M. Bdiwi, "Intuitive robot programming and intelligent tools," *IST Int. Surf. Technol.*, vol. 16, no. 2, pp. 8–9, Jun. 2023, doi: 10.1007/s35724-022-1138-6.

[3] R. Raffaeli, P. Bilancia, F. Neri, M. Peruzzini, and M. Pellicciari, "Engineering method and tool for the complete virtual commissioning of robotic cells," *Appl. Sci.*, vol. 12, no. 6, p. 3164, Mar. 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/6/3164

[4] M. Delledonne, E. Villagrossi, and M. Beschi, "Optimizing parameters of robotic task-oriented programming via a multiphysics simulation," in *Proc. IEEE 28th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2023, pp. 1–4.

[5] M. Wojtynek, J. J. Steil, and S. Wrede, "Plug, plan and produce as enabler for easy workcell setup and collaborative robot programming in smart factories," *KI—Künstliche Intelligenz*, vol. 33, no. 2, pp. 151–161, Jun. 2019, doi: 10.1007/s13218-019-00595-0.

[6] P. Bilancia, J. Schmidt, R. Raffaeli, M. Peruzzini, and M. Pellicciari, "An overview of industrial robots control and programming approaches," *Appl. Sci.*, vol. 13, no. 4, p. 2582, Feb. 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/4/2582

[7] T. B. Ionescu, "Leveraging graphical user interface automation for generic robot programming," *Robot.*, vol. 10, no. 1, pp. 1–23, 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/1/3

[8] M. Chemnitz, M. Yordanova, and A. Vick, "Graphische programmierung von industrierobotern," *Zeitschrift für wirtschaftlichen Fabrikbetrieb*, vol. 116, no. 4, pp. 227–231, Apr. 2021, doi: 10.1515/zwf-2021-0044.

[9] J. P. C. de Souza, A. M. Amorim, L. F. Rocha, V. H. Pinto, and A. P. Moreira, "Industrial robot programming by demonstration using stereoscopic vision and inertial sensing," *Ind. Robot, Int. J. Robot. Res. Appl.*, vol. 49, no. 1, pp. 96–107, Jan. 2022, doi: 10.1108/ir-02-2021-0043.

[10] L. Biagiotti, R. Meattini, D. Chiaravalli, G. Palli, and C. Melchiorri, "Robot programming by demonstration: Trajectory learning enhanced by sEMG-based user hand stiffness estimation," *IEEE Trans. Robot.*, vol. 39, no. 4, pp. 3259–3278, Apr. 2023.

[11] H. Hu, J. Chen, H. Liu, Z. Li, and L. Huang, "Natural language-based automatic programming for industrial robots," *J. Grid Comput.*, vol. 20, no. 3, p. 26, Aug. 2022, doi: 10.1007/s10723-022-09618-x.

[12] T. B. Ionescu and S. Schlund, "Programming cobots by voice: A pragmatic, Web-based approach," *Int. J. Comput. Integr. Manuf.*, vol. 36, no. 1, pp. 86–109, Jan. 2023.

[13] W. Yang, Q. Xiao, and Y. Zhang, "HAR^2bot: A human-centered augmented reality robot programming method with the awareness of cognitive load," *J. Intell. Manuf.*, vol. 35, pp. 1985–2003, Mar. 2023, doi: 10.1007/s10845-023-02096-2.

[14] B. Ikeda and D. Szafir, "PRogramAR: Augmented reality end-user robot programming," *ACM Trans. Hum.-Robot Interact.*, vol. 13, no. 1, pp. 1–20, Mar. 2024, doi: 10.1145/3640008.

[15] G. Ajaykumar, M. Steele, and C.-M. Huang, "A survey on end-user robot programming," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–36, Oct. 2021, doi: 10.1145/3466819.

[16] S. Blankemeyer, R. Wiemann, L. Posniak, C. Pregizer, and A. Raatz, "Intuitive robot programming using augmented reality," in *Proc. CIRP*, vol. 76, 2018, pp. 155–160. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827118300933

[17] P. Tsarouchi, S. Makris, G. Michalos, M. Stefos, K. Fourtakas, K. Kaltsoukalas, D. Kontrovrakis, and G. Chryssolouris, "Robotized assembly process using dual arm robot," in *Proc. CIRP Conf. Assem. Technol. Syst.*, vol. 23, 2014, pp. 47–52. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827114011354

[18] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, Nov. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957415818300321

[19] J. Arents and M. Greitans, "Smart industrial robot control trends, challenges and opportunities within manufacturing," *Appl. Sci.*, vol. 12, no. 2, p. 937, Jan. 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/2/937

[20] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognit. Robot.*, vol. 3, pp. 54–70, Jan. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667241323000113

[21] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," *Expert Syst. Appl.*, vol. 227, Oct. 2023, Art. no. 120254. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742300756X

[22] S. M. Marvasti-Zadeh, L. Cheng, H. Ghanei-Yakhdan, and S. Kasaei, "Deep learning for visual tracking: A comprehensive survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 5, pp. 3943–3968, May 2022.

[23] M. M. Islam, S. Nooruddin, F. Karray, and G. Muhammad, "Human activity recognition using tools of convolutional neural networks: A state of the art review, data sets, challenges, and future prospects," *Comput. Biol. Med.*, vol. 149, Oct. 2022, Art. no. 106060. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482522007739

[24] L. Heuss, D. Gebauer, and G. Reinhart, "Concept for the automated adaption of abstract planning domains for specific application cases in skills-based industrial robotics," *J. Intell. Manuf.*, vol. 35, no. 8, pp. 4233–4258, Oct. 2023, doi: 10.1007/s10845-023-02211-3.

[25] K. Jabrane and M. Bousmah, "A new approaach for training cobots from small amount of data in industry 5.0," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 10, pp. 1–13, 2021, doi: 10.14569/ijacsa.2021.0121070.

[26] S. Chen and J. T. Wen, "Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control," *Robotics*, vol. 10, no. 1, p. 50, Mar. 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/1/50

[27] T. N. Truong, A. T. Vo, and H.-J. Kang, "Neural network-based sliding mode controllers applied to robot manipulators: A review," *Neurocomputing*, vol. 562, Dec. 2023, Art. no. 126896. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223010196

[28] H. Fan, X. Liu, J. Y. H. Fuh, W. F. Lu, and B. Li, "Embodied intelligence in manufacturing: Leveraging large language models for autonomous industrial robotics," *J. Intell. Manuf.*, vol. 2024, pp. 1–17, Jan. 2024, doi: 10.1007/s10845-023-02294-y.

[29] K. Hori, K. Suzuki, and T. Ogata, "Interactively robot action planning with uncertainty analysis and active questioning by large language model," in *Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII)*, Jan. 2024, pp. 85–91.

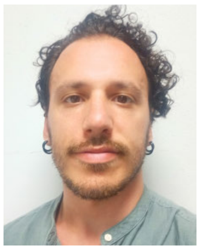
[30] Md. A. Khan, M. R. J. Khan, A. Tooshil, N. Sikder, M. A. P. Mahmud, A. Z. Kouzani, and A.-A. Nahid, "A systematic review on reinforcement learning-based robotics within the last decade," *IEEE Access*, vol. 8, pp. 176598–176623, 2020.

[31] 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. [Online]. Available: https://www.mdpi.com/1424-8220/23/7/3762

[32] E. Coumans and Y. Bai. (2016). *Pybullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. [Online]. Available: http://pybullet.org

[33] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. (Mar. 2023). *Gymnasium*. [Online]. Available: https://zenodo.org/record/8127025

[34] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html



MICHELE DELLEDONNE received the B.S. and M.S. degrees in industrial automation engineering from the University of Brescia, Italy, in 2019 and 2021, respectively. He is currently pursuing the joint Ph.D. degree in mechanical and industrial engineering with the University of Brescia and the Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, National Research Council, Milan, Italy. His research interests include robot task optimization, intuitive robot programming, and control techniques for industrial manipulators.

ENRICO VILLAGROSSI received the B.S. and M.S. degrees in industrial automation engineering and the Ph.D. degree in applied mechanics from the University of Brescia, in 2008, 2011, and March 2017, respectively. His Ph.D. thesis was about the use of industrial robots in machining applications. Since December 2020, he has been a permanent Researcher with the Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, National Research Council (STIIMA-CNR), Italy. He has been involved in several European projects, including FP7, H2020, and HE programs. His research interests include cutting-edge robotic applications in advanced manufacturing, robotics applications for circular economy, and sustainable robotics.

MANUEL BESCHI (Member, IEEE) received the B.S. and M.S. degrees in industrial automation engineering from the University of Brescia, Italy, in 2008 and 2010, respectively, and the Ph.D. degree in computer science, engineering and control systems technologies from the Department of Mechanical and Industrial Engineering. He has been an Associate Professor with the University of Brescia, since 2022.

ALIREZA RASTEGARPANAH received the Ph.D. degree in robotics from the University of Birmingham, in 2016. Continuing his research pursuits, he joined University College London and later the Faraday Institution. He is a Senior Research Fellow with diverse research interests broadly centered on robotics, machine vision, AI, machine learning, and human–robot interaction. He is internationally recognized in robotic disassembly and currently a Co-PI of the REBELION Project, which focuses on robotizing the process of testing, disseminating, and sorting EV batteries.

Open Access funding provided by 'Consiglio Nazionale delle Ricerche-CARI-CARE-ITALY' within the CRUI CARE Agreement