Received: 12 October 2023 | Revised: 21 January 2024 | Accepted: 6 March 2024

IET Cyber-Systems and Robotics

DOI: 10.1049/csy2.12113

ORIGINAL RESEARCH

Learning to bag with a simulation-free reinforcement learning framework for robots

Francisco Munguia-Galeano[1] | **Jihong Zhu**[2] | **Juan David Hernández**[3] | **Ze Ji**[4]

[1]Cooper Group, University of Liverpool, Liverpool, UK

[2]School of Physics Engineering and Technology, University of York, York, UK

[3]School of Computer Science and Informatics, Cardiff University, Cardiff, UK

[4]School of Engineering, Cardiff University, Cardiff, UK

Correspondence
Ze Ji.
Email: jiz1@cardiff.ac.uk

Funding information
Engineering and Physical Sciences Research Council, Grant/Award Number: EP/X018962/1; Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCyT)

Abstract
Bagging is an essential skill that humans perform in their daily activities. However, deformable objects, such as bags, are complex for robots to manipulate. A learning-based framework that enables robots to learn bagging is presented. The novelty of this framework is its ability to learn and perform bagging without relying on simulations. The learning process is accomplished through a reinforcement learning (RL) algorithm introduced and designed to find the best grasping points of the bag based on a set of compact state representations. The framework utilises a set of primitive actions and represents the task in five states. In our experiments, the framework reached 60% and 80% success rates after around 3 h of training in the real world when starting the bagging task from folded and unfolded states, respectively. Finally, the authors test the trained RL model with eight more bags of different sizes to evaluate its generalisability.

KEYWORDS
reinforcement learning, robot learning, robotics

1 | INTRODUCTION

Robots with human-level dexterity that can handle deformable objects may encourage smoother integration of robots in daily and industrial activities [1]. In practice, daily activities depend on more than manipulating rigid objects. In this context, the robots' capacity to manipulate deformable objects to operate in human environments is a necessity [2]. Among deformable objects, bags are used in several relevant tasks, such as transporting objects, packing, and shopping. Even though there have been studies on how to manipulate deformable objects, such as paper [3–6], fabrics [7–9], ropes [10–12], cables [13, 14] and meat [15], the problem of learning to bag with robots is still under-explored.

Bagging is a complex task for robots because there exist challenges related to perception, occlusions, modelling of the bag's dynamics, ambiguity related to finding the opening, and how to grasp one or two layers of the bag depending on the current state of the task. Reinforcement learning (RL) has the potential to deal with the problems mentioned above. However, RL agents are commonly trained in simulation, which brings more challenges when implementing the agent in real-world tasks [16]. One challenge is that before simulating the bag, the model must be as similar as possible to the real object [17, 18]. On the other hand, when switching from simulation to real world, the agent must deal with occlusions, incomplete information, and noises. These are vital factors that make the generalisation of RL difficult [19].

This paper presents a learning framework for robot-bagging tasks with compact state representations and primitive actions, aiming to efficiently train a robot to learn bagging in the real world [20]. The framework identifies five possible states and utilises eight primitive actions related to several grasping points on the bag. The task is solved in four steps (Figure 1): unfolding, opening, placing the piece, and carrying. The main contributions of this paper are as follows: (i) an RL algorithm is introduced, allowing robots to efficiently learn how to bag in the real world (simulation-free), (ii) a versatile state representation for the bagging task and (iii) a framework that provides a reliable perception of the bag state for learning.

The problem domain to empirically validate the framework encompasses textile bags and a red cube (Figure 1). The framework first learns to perform the task through a different number of steps, as explained in Section 5. Then, the trained model is used to perform the task using eight more bags with different sizes and positions to test the framework's generalisation capabilities.

The rest of this article is structured as follows. First, Section 2 summarises related works in RL and manipulation of deformable objects. The problem formulation is described in Section 3. Then, Section 4 formally introduces the framework, followed by Section 5, which describes the experimental setup. Section 6 presents the results, while Section 7 discusses them. Finally, Section 8 concludes this paper.

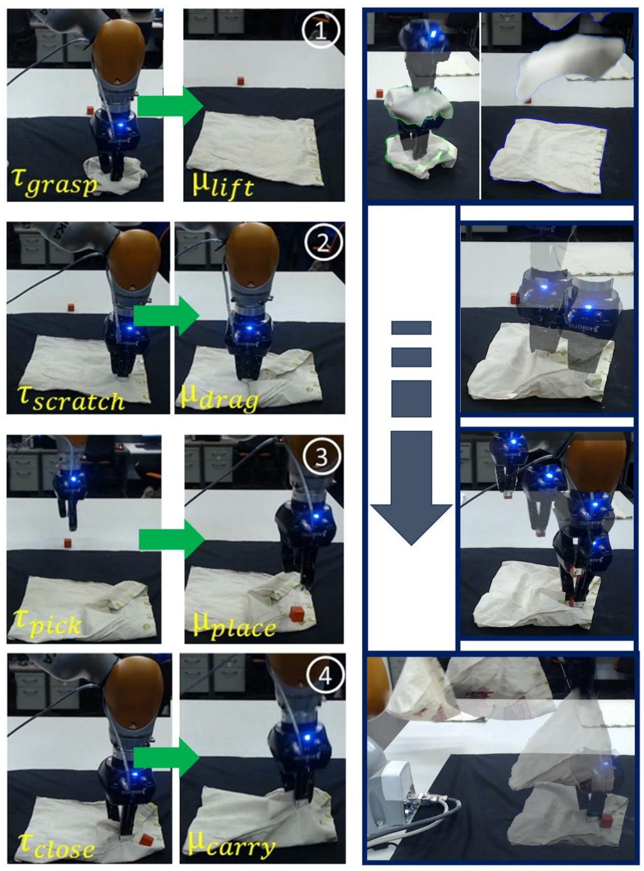


FIGURE 1 The robot, in four steps performs the bagging task. In the first step, the robot unfolds the bag. In the second step, the bag is opened by the robot. The robot places the red cube in the bag's opening in the third step. In the fourth step, the robot carries the bag completing the task.

2 | RELATED WORK

One way to tackle the challenge of handling deformable objects with robots is by using simulated environments that are valuable resources for training agents to learn how to solve a given task. For example, Seita et al. [21] used transporter networks to learn how to rearrange deformable objects, where 3D deformable structures such as bags are included. Bahety et al. [22] proposed a method to rearrange, which is based on two policies learned in simulation. The first policy rearranges the objects, and the second policy learns to lift them. A disadvantage of this approach is that it assumes that the bag is always open. Therefore, in a simulated environment, all the information regarding the opening of the bag and the objects is available, which is useful when training an agent requires a large number of episodes to learn the task. However, when running the agent in the real world, the differences between the simulation and the real-world tasks may lead to undesirable and dangerous behaviours.

In the literature, methods exist to transfer the knowledge obtained during simulation to the real world (sim-to-real). Ma et al. [23] utilised a method that sets several grasping points on a cloth surface. A graph neural network uses these points to learn their dynamics. Subsequently, when the task is transferred to the real world, it is easier to track the points than to track the whole cloth. Wang et al. [24] introduced a method in which an agent is trained to learn how to wrap boxes in a simulated environment. The actual texture of the deformable object is taken from the real object such that the transition from sim-to-real is smoother than when taken from the pure simulation.

In the context of bagging, Iterative Interactive Modelling for Knotting Plastic Bags [25] is an approach that focuses on the bag's handles, learns from demonstrations, and uses a set of primitive actions to knot the handles. A disadvantage of this approach is that the detection of the handles relies on a large dataset. The approach proposed by Chen et al. [26] is of particular relevance to this paper. Their algorithm, AutoBag, is built upon a set of primitive actions that involve reorienting plastic bags until the opening becomes visible, allowing objects to be placed inside. The authors' research primarily focuses on plastic bags. Plastic bags tend to keep their shape after manipulation, where the mechanical characteristics are utilised in favour of the manipulation task. For example, when the opening is visible, the bag tends to maintain its shape. In contrast, with a textile-based bag, once the layer is no longer gripped, it falls, necessitating modifications to the AutoBag algorithm to accommodate this behaviour. In this context, this work aims to investigate a bagging process involving textile-based bags and a learning-based solution performed in the real world.

Despite the vast literature exploring several RL approaches in simulation, less attention has been given to RL in the real world due to several problems [27], such as costly robot time, motion constraints, or stochastic behaviours of objects surrounding the robot. Although training RL in simulation is effective, transferring the trained policies to the real world is a challenge that must be taken seriously, often resulting in significant performance degradation due to the simulation-to-reality gap [28]. Therefore, tackling complex problems such as bagging in the real world may open the door to addressing further challenges in robotics and RL.

3 | PROBLEM FORMULATION

We formulated the bagging task as a Markov decision process (MDP) [29] and aimed to find a solution using a learning policy π. In an MDP, the agent executes a valid action a from the set of actions A in the current state s and transitions to a valid state s', where s and s' belong to the set of states S, according to the unknown dynamics of the bag. The environment provides a reward according to the reward function $R(s, a)$ upon transitioning to a new state. The set of states S is given using the following equation:

$$S = \{s_0, s_1, s_2, s_3, s_4, s_5\}, \tag{1}$$

where s_0 represents the folded bag, s_1 is the representation of the bag expanded, s_2 represents the bag opened, s_3 shows when the red object is on the bag, s_4 is the success state, and s_5 is the fail state. More specifically, the states are denoted as follows:

- For s_0: Folded bag.
  Condition: the bag's area is small and the opening is not visible (Figure 2a).
- For s_1: Expanded bag.
  Condition: The bag is unfolded, the opening is visible, and the green labels can be seen (Figure 2b).
- For s_2: Opened bag.
  Condition: The opening area of the bag is large enough to accommodate an object (Figure 2c).
- For s_3: Red object on the bag.
  Condition: The object is in the opening of the bag, which distinguishes this state from the others (Figure 2d).
- For s_4: Success state.
  Condition: No visible objects were left on the table, indicating that the robot carried both the bag and the object (Figure 2e).
- For s_5: Fail state.
  Condition: The robot took the bag, but the red cube was still on the table (Figure 2f).

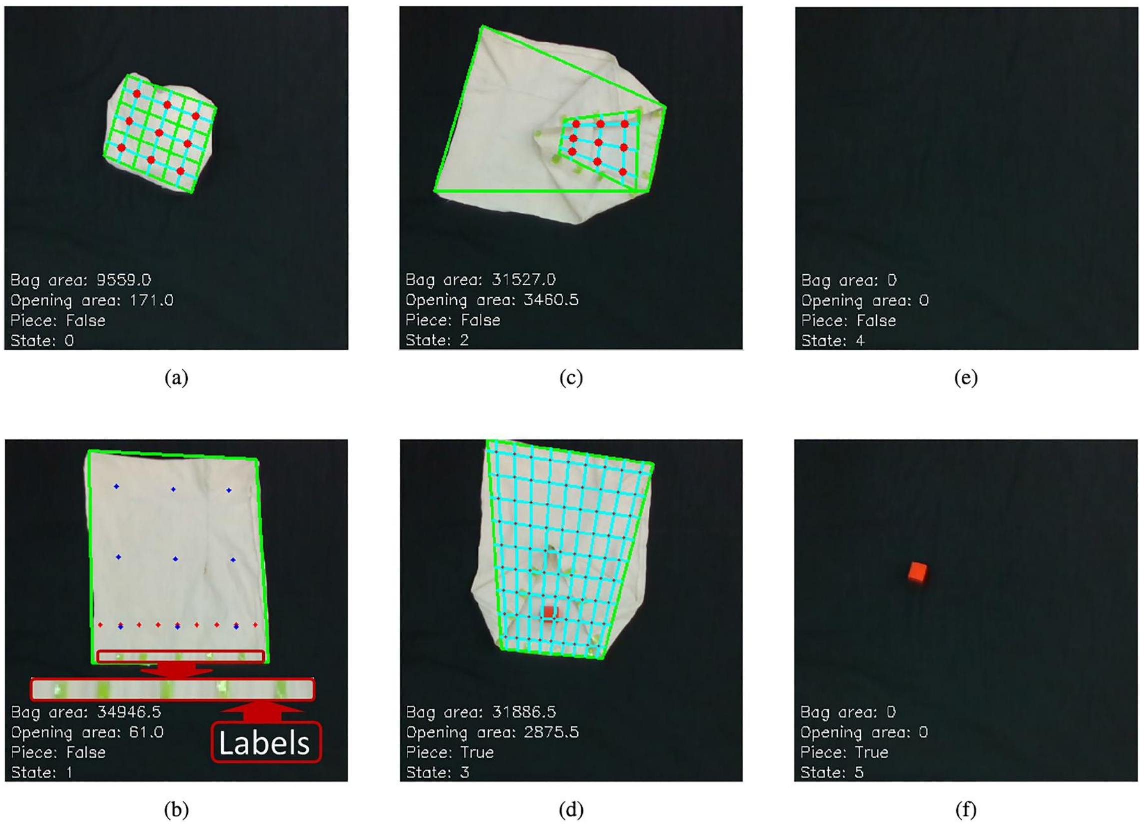


FIGURE 2 This figure illustrates the five states that comprise the bagging task, where the red and blue dots represent the grasping points the robot can select. In (a), the bag is folded such that its area is small, and the opening is not visible. In (b), the bag is unfolded, and the opening is visible. In (c), the bag's opening area is large enough to put an object inside. In (d), the object is in the bag's opening, distinguishing this state from the others. In (e), the task succeeded because no visible objects were left on the table, meaning that the robot carried both the bag and the object. Lastly, (f) shows a failure case when the robot took the bag, but the red cube was still on the table.

Hence, the current state can be calculated as follows:

$$s = \begin{cases} s_0, & (A_{bag} < A_{th}) \wedge (A_o = 0) \wedge (A_{cube} = 0) \\ s_1, & (A_{bag} > A_{th}) \wedge (A_o > 0) \wedge (A_{cube} = 0) \\ s_2, & (A_{bag} > A_{th}) \wedge (A_o > A_{oth}) \wedge (A_{cube} = 0) \\ s_3, & (A_{bag} > A_{th}) \wedge (A_o > A_{oth}) \wedge (A_{cube} > 0) \\ s_4, & (A_{bag} = 0) \wedge (A_o = 0) \wedge (A_{cube} = 0) \\ s_5, & (A_{bag} = 0) \wedge (A_o = 0) \wedge (A_{cube} > 0), \end{cases} \tag{2}$$

where A_{bag} is the bag's area, A_{th} is a threshold value indicating a small area, A_o is the area of the opening, A_{oth} is the threshold value indicating a large enough opening and A_{cube} stands for the area of the cube. The robot can select among a set of primitive actions executed in pairs. Let

$$\Phi = \{\langle \tau_{grasp}, \mu_{lift} \rangle, \langle \tau_{scract}, \mu_{drag} \rangle, \langle \tau_{pick}, \mu_{place} \rangle, \langle \tau_{close}, \mu_{carry} \rangle\}, \tag{3}$$

be the set of tuples that contains the primitive actions, τ that represents the primary primitive action executed by the robot and μ the complementary primitive actions (Figure 3). The primitive action τ_{grasp} is the robot action to grasp two layers of the bag, and μ_{lift} is when the bag is raised above the table and drops the bag aiming to unfold it. $\tau_{scratch}$ is the action to grasp only one layer of the bag, and the primitive action μ_{drag} takes place when the robot moves a grasped layer point to a placing point. The τ_{pick} and μ_{place} actions refer to picking and placing the red object. τ_{close} grasps one layer of the bag when the object is placed in the opening, and μ_{carry} lifts the bag to grab the object. This paper aims to find an optimal policy π^* that computes the correct sequence of primitive actions to transition through the states S until solving the bagging task.

4 | REAL-WORLD LEARNING ROBOT-BAGGING FRAMEWORK

This section presents a framework that aims to solve the problem of learning to bag in the real world with RL (Figure 4). The framework comprises three modules: perception, learning and robot controller.

4.1 | Perception

The perception module plays a crucial role in extracting relevant information from the bag, including its area, opening area, grasping points, and current state. It receives data from an Intel® RealSense^{TM} camera, which provides RGB and depth images. To process these data, the module utilises the OpenCV library. By filtering colours from the black background and

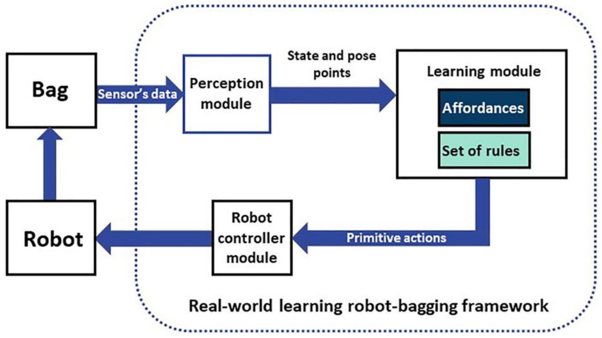


FIGURE 4 Proposed framework for learning to bag using reinforcement learning.

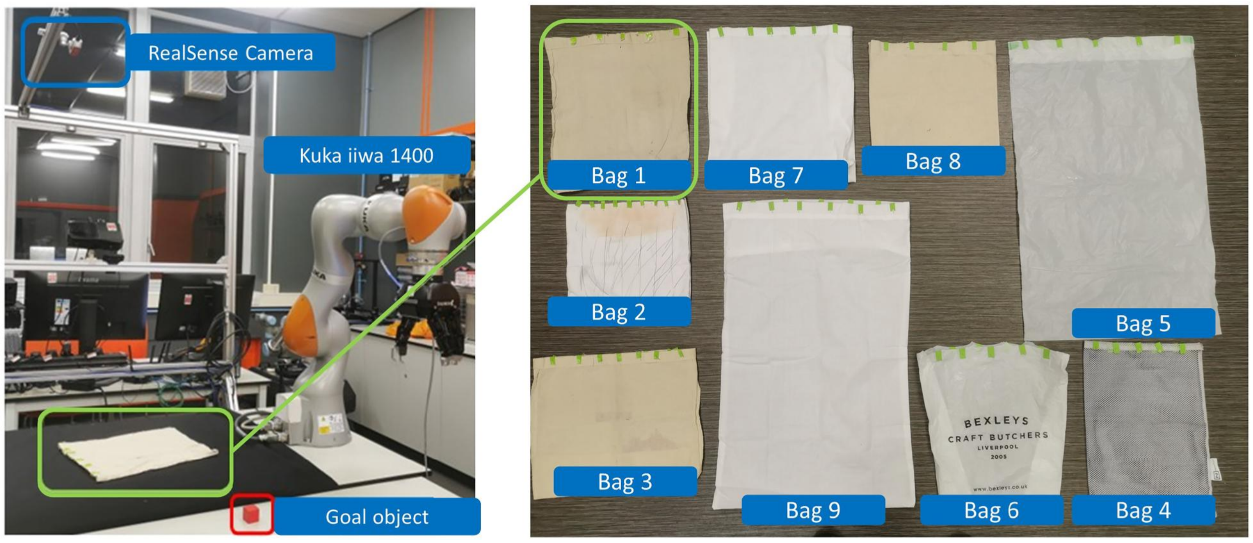


FIGURE 3 The left side of the figure illustrates the experimental setup comprising an object to be bagged (red cube), a Kuka® iiwa 1400^{TM} robot, and an Intel® RealSense^{TM}. On the left side are the nine bags used during the experiments.

obtaining the bag's contours, the module can calculate the A_{bag} value.

Algorithm 1 Opening area's calculator.

```
Require: points: A list of 2D points
Ensure: $A_o$: The area of the opening
 1: if COUNTPOINTS(points) < 3 then
 2:     $A_o \leftarrow 0$
 3:     return $A_o$
 4: end if
 5: triangles ←
 6: nodes ← COUNTPOINTS(points)
 7: centre ← GETCENTER(points)
 8: while True do
 9:     pts ← points[]
10:     i, n ← FINDCLOSESTNODE(centre, pts)
11:     j, m ← FINDCLOSESTNODE(n, pts)
12:     triangles.append([centre, n, m])
13:     pts.pop(j)
14:     k, o ← FINDCLOSESTNODE(n, pts)
15:     triangles.append([centre, n, o])
16:     points.pop(i)
17:     ifCOUNTPOINTS (points) < 3 then
18:         break
19:     end if
20: end while
21: $A_o \leftarrow 0$
22: for each triangle in triangles do
23:     $A_o = A_o +$ GETAREA(triangle)
24: end for
25: return $A_o$
```

In order to assist with the automatic classification of bag states, green labels have been added around the bag's opening (refer to Figure 2b). It is worth mentioning that perception is not the main focus of this work. Hence, we simplify the experimental configurations in order to reliably obtain the states from observations. Specifically, green markers were used to facilitate the detection of the bag opening. Algorithm 1 is utilised to calculate the opening area. This is achieved by providing an array of points containing the pair of coordinates obtained from the labels. Then, through the generation of triangles, it is possible to sum all their areas and, in this way, obtain the opening area A_o.

This information enables the module to determine whether the bag is on the table, folded, or unfolded. To differentiate the bag from the background, we use a black canvas as the background to make it easier to extract the bag, which is white. Similarly, the object for bagging is a red cube. In brief, the primary functions of the perception module are as follows:

- To determine the current state of the task;
- To provide pose points depending on the state;
- To measure the bag's current area; and
- To measure the current opening's area.

The perception module generates multiple pose points denoted by $g = \zeta^2$, where g represents the number of pose points and ζ is the griding parameter. For instance, Figure 2 illustrates a configuration with $g = 9$ and $\zeta = 3$. Increasing the value of g results in more points available for grasping and lifting the bag, consequently leading to a larger set of actions to explore. Thus, the position of the grasping points can be determined.

The s_0 state (Figure 2a) can be distinguished from the others because the opening is not visible, and the bag area is small compared to when it is unfolded. In this state, there are $g = 9$ grasping points (red dots in Figure 2a) whose position can be obtained with the RealSense camera. These points are stored in the set P_{s_0}. Before unfolding the bag and reaching the next state, the robot must explore which of these grasping points is the best to grasp the bag, lifting it at a given height stored in the D_{s_0} set and dropping it.

In the s_1 state (Figure 2b), the opening is visible, and the bag's area has reached a feasible value for opening the bag. Besides, since the opening area is small, s_1 can be distinguished from the other states based on the criteria of the opening area. In this state, the perception module provides $g = 9$ grasping points close to the opening (red dots in Figure 2b) which are stored in the set P_{s_1}, and g placing points (blue dots in Figure 2b) stored in the set D_{s_1}. Consequently, the robot's goal in this state is to find which combination of actions over the grasping point maximises the opening area.

In the s_2 state (Figure 2c), the opening's area serves as a trigger to identify this state. First, the perception module stores the object's position to be bagged in P_{s_2}. Then, the perception module sets $g = 9$ placing points (red dots in Figure 2c) and stores them in the set D_{s_2}. The robot can place the object to be bagged on one of those placing points and get a reward depending on the closeness with the centre of the bag's opening.

In the s_3 state (Figure 2d), the red object is placed in the opening. In this state, the robot is required to explore more grasping points. For this reason, the number of pose points is $g = 81$ (red dots in Figure 2d). Then, the pose points are stored in the set P_{s_4} while the lifting poses corresponding to the complementary primitive action in this state are stored in the set D_{s_4}. Therefore, the robot can interact with the bag and decipher which grasping point allows it to finish the task. When the robot lifts the bag and the module cannot detect any object left, as shown in Figure 2d, state 4 s_4 is identified, and the task is finished. On the other hand, when there are still objects on the table, the module identifies state 5 s_5, which is a failed attempt to bag the object (Figure 2f).

4.2 | Learning

The learning module seeks to find the optimal combination of grasping points on the bag and primitive actions from the robot. Additionally, this subsection aims to motivate the problems with current RL approaches and explain the functionality principle of our algorithm Π-learning.

In RL, the Bellman equation [30] is a fundamental concept that allows the computation of the value function $V(s)$. In this context, the Bellman equation can be expressed as follows:

$$V(s) = R(s,a,s') + \gamma V(s') \tag{4}$$

here, $V(s)$ is the value of the state s, $R(s, a, s')$ is the reward for transitioning from s to s' while taking action a, γ is a discount factor and $V(s')$ is the value of the next state. When a policy π is given, then the value function can be expressed as follows:

$$V^{\pi}(s) = R(s,a,s') + \gamma V^{\pi}(s') \tag{5}$$

This form of the Bellman equation works when the environment is deterministic. However, when the environment presents stochastic behaviours, such as the behaviour presented by the bag, it is necessary to add the probability function $P(s' \mid s, a)$:

$$V^{\pi}(s) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{\pi}(s')) \tag{6}$$

Despite the above equation, which includes the stochasticity of the environment, it still needs to be considered when the policy is stochastic and not deterministic. Hence, to include a stochastic policy,

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{\pi}(s')) \tag{7}$$

The equation above is known as the Bellman expectation equation, and it can be rewritten in its expectation form:

$$V^{\pi}(s) = \mathbb{E}_{\substack{s' \sim P \\ a \sim \pi}} [R(s,a,s') + \gamma V^{\pi}(s')] \tag{8}$$

According to Equation (8), it can be observed that $V(s)$ evaluates the whole value of the state. However, it does not evaluate each action independently, and this is a limitation that can be solved by calculating the quality $Q(s, a)$ of every action–state pair. Then, the Bellman equation of the Q function can be expressed as follows:

$$Q(s) = R(s,a,s') + \gamma Q(s') \tag{9}$$

here, $Q(s, a)$ is the Q value of an action–state pair, $R(s, a, s')$ is the reward for transitioning from s to s' while taking action a, γ is a discount factor and $Q(s', a')$ is the Q value of the next state's action a'. When a policy π is given, the Q-function can be expressed as follows:

$$Q^{\pi}(s,a) = R(s,a,s') + \gamma Q^{\pi}(s',a') \tag{10}$$

This form of the Bellman equation works when the environment is deterministic. However, when the environment presents stochastic behaviours such as the one presented by the bag while being manipulated, it is necessary to add the probability function $P(s' \mid s, a)$:

$$Q^{\pi}(s) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma Q^{\pi}(s',a')) \tag{11}$$

Despite the equation above, which includes the stochasticity of the environment, it still needs to be considered when the policy is stochastic and not deterministic. Hence, to include a stochastic policy:

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \sum_{a} \pi(a|s) Q^{\pi}(s',a')) \tag{12}$$

The equation above is known as the Bellman expectation equation of the Q-function, and it can be rewritten in its expectation form:

$$Q^{\pi}(s) = \mathbb{E}_{s' \sim P}[R(s,a,s') + \gamma \mathbb{E}_{a' \sim \pi} Q^{\pi}(s',a')] \tag{13}$$

In general, the value and Q function Bellman equations are used for calculating the value of the state and the quality of the Q value's state–action pairs, respectively. These equations are based on the environment dynamics given by $P(s' \mid s, a)$, a policy π, and the reward function $R(s, a, s')$. The Bellman optimality equation expresses the expected maximum or total reward that can be achieved from a given state based on the value function (Equation 7). The equation defines the relationship between the value of a state and the values of its neighbouring states. The Bellman optimality equation of the value function is given using the following equation:

$$V^{*}(s) = \max_{a} \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{*}(s')) \tag{14}$$

The preceding equation states that the optimal value of a state is the maximum expected value obtained by taking the best action in the current state. Moreover, this equation considers the expected values of the resulting states. On the other hand, the Bellman optimality equation of the Q function can also be calculated under the same logic and is denoted as follows:

$$Q^{*}(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^{*}(s',a')) \tag{15}$$

Additionally, the relationship between the value and Q functions is that the value function can be derived from the Q function by selecting the maximum Q value for each state. In order to calculate the value function based on the Q function, the maximum Q value over all possible actions in a given state is selected. Consequently, for each state, the action that maximises the Q value is selected, which becomes the value of that state. Therefore, $V(s)$ equals the maximum Q value for a given state s. Then, the relationship can be expressed as follows:

$$V^*(s) = \max_a Q^*(s,a) \tag{16}$$

Following the logic from the previous sentence, if the maximum value of $V^*(s)$ corresponds to the maximum value of $Q^*(s, a)$, then the Q function can be derived from the value function by substituting Equation (16) with Equation (15), then:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^*(s)) \tag{17}$$

In general, the relationship between the value and Q functions is that the value function can be derived from the Q function by selecting the maximum Q value for each state. At the same time, the Q function can be derived from the value function by using the Bellman equation. However, as mentioned earlier, the model of the bag is unknown. Aiming to solve these sorts of problems, Mote Carlo (MC) methods are algorithms designed to estimate the value function and find the dynamics of the environment through interaction (model-free) [31]. The expected value of the value function of a given state can be approximated by visiting that state N times and obtaining the total return:

$$V(s_t) \approx \frac{1}{N_t} \sum_i^{N_t} R_i \tag{18}$$

here, R_i is the total return, s_t corresponds to the step t when that state was visited, and N_t denotes the number of times the s_t state in a given step t has been visited. In order to establish a balance between computational efficiency and memory requirements, instead of using the arithmetic mean as in Equation (18), the incremental mean is more commonly used, and it is expressed as follows:

$$V(s_t) = V(s_t) + \alpha(R_t - V(s_t)), \tag{19}$$

where $\alpha = \frac{1}{N_t}$. MC methods also focus on solving the problem of estimating the Q function for a given policy. Hence, the following expression can be deduced:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t)) \tag{20}$$

Despite the potential of MC methods, their working principle design relies on reaching a terminal state to approximate a value function. A drawback is that if the episode is too long, it means costly computation. Consequently, any MC method would not be a proper fit for the bagging task. In this context, Sutton [29] proposed an alternative that balances Bellman equation methods and MC methods. This approach is known as temporal difference (TD). These approaches learn by bootstrapping, meaning that instead of waiting till the end of an episode, they updated their value estimation based on the following:

$$V(s) \approx R(s,a) + \gamma V(s'), \tag{21}$$

where $R(s, a)$ is the immediate reward obtained after performing action a from state s. The reward function used in this work is given using the following equation:

$$R(s,a) = \begin{cases} \frac{A_{b_{max}}}{A_{bag}}, & s = s_0 \text{ or } s = s_1 \\ \frac{Ao_{max}}{A_o}, & s = s_2 \\ 1, & s = s_3, \text{ and the object is} \\ & \text{at the center of the opening} \\ 1, & s = s_4 \\ -0.1 & s = s_5 \\ 0, & \text{otherwise}, \end{cases} \tag{22}$$

where $A_{b_{max}}$ is the maximum area of the bag when it is unfolded, A_{bag} is the area of the bag after executing an action, Ao_{max} is the maximum area that the bag's opening can reach, and A_o is the bag's opening area after executing an action.

In a similar manner in which the average mean was replaced with the average mean from Equation (19), then Equation (21) can be expressed as follows:

$$V(s) = V(s) + \alpha(R(s,a) + \gamma V(s') - V(s)) \tag{23}$$

The last expression, known as the TD estimation rule, can be used to estimate the value of the state. It is also possible to apply the same logic to the Q function in order to have an optimal policy based on that estimate. The TD estimate rule of the Q function is given using the following equation:

$$\begin{aligned} Q(s,a) = Q(s,a) + \alpha(R(s,a) \\ + \gamma Q(s',a) - Q(s,a)) \end{aligned} \tag{24}$$

As previously described, this work proposes using several primitive actions, meaning that the action space is discrete. Among several approaches, Q-learning (QL) is a popular off-policy RL algorithm that learns an MDP in environments with discrete action and state spaces [32]. This algorithm is also model-free because it does not require any previous model of the environment. QL is designed to update the quality values Q of a state–action combination, given using the following equation:

$$Q : A \times S \rightarrow \mathbb{R}, \tag{25}$$

where Q are the Q-values (usually stored in a table), and they represent the quality of the state–action pair. In other words, the higher the Q-value, the better the action for that state.

Then, a Q-table is necessary to represent each Q-value such that QL updates the state–action pair by using the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{26}$$

where α is the learning rate and γ is the discount factor. The discount factor is usually a value between 0 and 1 ($0 \leq \gamma \leq 1$) that balances the importance the agent puts on future rewards rather than immediate rewards. According to Equation (26), the state–action pair is updated based on the next state s' even when that state has not been explored, which is why QL is considered an off-policy method.

Despite the popularity of QL and its successful implementation in multiple fields, in regard to the problem defined in Section 3, there exist several drawbacks. The first one is related to the size of the exploration space, where considering eight primitive actions, 81 pose points, and four possible states in the case of Figure 2, it would be necessary to explore a total of 2592 primitive action–pose point pairs. This approach lacks practical significance because it involves a long training time in the real world.

Additionally, the dependency of QL on the next state value would also involve significant exploration to achieve stable convergence. This is because of the dynamics of the bag, which, despite executing the best action, would take repeating the same action until the bag's state transitions. More specifically, while for the best action a given the state s_t, the bag may transition to s_{t+1}, it may also stay in the same state s_t due to the manner in which the problem was defined (see Equation 2) and for the characteristics of the environment, which also include failures in the real world that could contaminate the training.

Aiming to solve the drawbacks, the authors propose the following. First, the exploration space is reduced by implementing affordances that define what primitive actions are suitable given the current state. The use of affordances has proven to be an efficient method for reducing the exploration space [33, 34]. In this work, the affordances are obtained from a manually defined set of rules denoted with Ψ. Then, the robot executes actions with probability ϵ, also known as ϵ-greedy policy (the value of ϵ controls the exploration of the environment), and gets a reward. The process is repeated for n steps till the training is completed. Let

$$\Psi = \Big\{ \langle s_0, \tau_{grasp}, \mu_{lift} \rangle, \langle s_1, \tau_{scratch}, \mu_{drag} \rangle, \langle s_2, \tau_{pick}, \mu_{place} \rangle, \langle s_3, \tau_{grasp}, \mu_{carry} \rangle \Big\}, \tag{27}$$

be the set of rules that contains the tuples in which each state s is related to its valid actions. Then, the primary affordable actions of the states are given as follows:

$$\Lambda_{s_j,primary} = \Psi^2 \times P_{s_j}, \tag{28}$$

where $s_j \in \Psi^1$ and $j \in (0, |S|)$. The affordable complementary actions are given using the following equation:

$$\Lambda_{s_j,complementary} = \Psi^3 \times D_{s_j}, \tag{29}$$

where $s_j \in \Psi^1$. The set of affordable action pairs is given using the following equation:

$$A = \Lambda_{s_j,primary} \times \Lambda s_j, complementary \tag{30}$$

Aiming to avoid confusion with the QL notation algorithm, we utilise Π. Algorithm 2 has a function that calculates the state–action value Π, representing the quality value of an action a given a state s:

$$\Pi : S \times A \rightarrow \mathbb{R} \tag{31}$$

At the beginning of the learning, all Π values are initialised to zero and stored in a Π-table. During the training process, it is updated with the following equation:

$$\Pi(s,a) \leftarrow \frac{\Pi(s,a) + R(s,a)}{m}, \tag{32}$$

where m is the number of times that action–state pair $\Pi(s, a)$ has been selected by the agent, $m > 0$ and $a \in A$. Here, it is not necessary to utilise α as in Equation (20) because the compact state representation combined with the Π-table allows us to know the number of times m that state has been visited. Hence, the bootstrapping technique is not necessary. In contrast to classical QL, which requires exploring a total of 2592 primitive action–pose point pairs (considering 8 primitive actions, 81 pose points, and 4 possible states in the case of Figure 2), Π-learning is tailored to handle these conditions, particularly in the context of the bagging task. This is because many actions do not result in state changes due to factors such as incomplete unfolding in state 0 or failure to open the bag in state 1. As a result, the robot must repeat the same action until a transition occurs. To this end, the optimal policy is extracted from $\Pi(s, a)$ with the following equation:

$$\pi^*(s) = \arg\max_a [\Pi(s,a)] \tag{33}$$

Unlike QL, Π-learning incorporates Equation (32), which is independent of the next state. It also reduces the exploration space by defining rules and pairs of primitive actions using Equation (30). For instance, in the scenario depicted in Figure 2, QL would require exploring 2592 actions, while Π-learning would only necessitate exploring 81 actions. This significant reduction in exploration space aims to reduce training time for Π-learning. Furthermore, the Π-table remains unaffected by the next state. This is crucial because the bag assumes different shapes after each action and may or may not transition into the next state. This behaviour may cause instability for QL because of its dependence on the next state information. Consequently, the perception module detects state transitions and allows Π-learning to concentrate on maximising the reward solely based on the current state.

Algorithm 2 takes the number of training steps n and the set of actions A as the input. First, a Π-table is generated. Then, after n steps of training, the algorithm returns the optimal policy $\pi^*(s)$. The Π-learning algorithm is specifically designed to enable learning with a reduced number of states while dealing with a wide range of actions.

Algorithm 2 Π-learning.

```
Require: Training steps $n$, set of actions $A$
Ensure: Optimal policy $\pi^*(s)$
  1: Initialise a Π-table with zeros
  2: for $n$ steps do
  3:     With probability $\epsilon$, select a valid
         action $a$ from $A$
  4:     Perform $a$ and calculate the reward
         with Equation (22)
  5:     Update Π-table with Equation (32)
  6: end for
  7: Extract the optimal policy from the
     Π-table with Equation (33)
```

4.3 | Robot controller

The robot controller module coordinates the continuous actions of the robot in a precise manner. To accomplish this, the module inputs any primitive action and translates it into continuous actions that the robot can handle. Since the perception module provides the grasping points of the bag and the learning module generates a sequence of primitive actions given a state, the robot controller module can make the robot interact with the environment. This information is managed through the robot-operating system (ROS).

4.4 | Bagging task implementation

This subsection aims to provide a more exhaustive description of the bagging task, focusing on how primitive actions are implemented and how positions are determined within the perception module. The following is a point-by-point description of the processes that allow the robot to perform the bagging task:

- **The primitive actions** are steps taken by the robot to interact with the bag and achieve the overall task. The primitive actions are hard-coded commands and depend on the perception module's output, which interprets data from an Intel® RealSense™ camera and provides a set of possible grasping points. These actions are available as services running under ROS.
- **The grasping points** are necessary for the robot to securely hold the bag, depending on the current state. The perception module generates multiple pose points denoted by $g = \zeta^2$, where g represents the number of pose points and ζ is the griding parameter. These points are automatically positioned around the bag's opening and body. If more grasping points are required, the value of ζ must be increased. Depending on the state of the bag given by the perception module, the framework calls an ROS service corresponding to the current state and feeds the grasping points as parameters.
- **The position determination of grasping points** is important for the execution of primitive actions. The algorithm presented in Algorithm 1 calculates the opening area A_o by generating triangles from an array of points obtained from green labels around the bag's opening. This information is used to determine the positions for grasping and placing points. When it comes to the body of the bag, the perception module retrieves the bag's body as a 4-sided geometry with sides F, G, H, and I (refer to Figure 5). The objective is to evenly distribute ζ^2 points inside the described geometry based on the reference point (x_0, y_0), corresponding to the upper right corner of the 4-sided geometry surrounding the bag. Each side of the geometry can be expressed as follows:
  1. Side F represented by the coordinates (p_{11}, p_{21}) and (p_{31}, p_{41}).
  2. Side G represented by the coordinates (p_{12}, p_{22}) and (p_{32}, p_{42}).
  3. Side H represented by the coordinates (p_{13}, p_{23}) and (p_{33}, p_{43}).
  4. Side I represented by the coordinates (p_{14}, p_{24}) and (p_{34}, p_{44}).

The reference point (x_0, y_0) corresponds to the upper right corner of the 4-sided geometry. The objective is to distribute ζ^2 points inside this geometry evenly. Hence, the ith point on sides F, G, H, and I can be represented as follows:

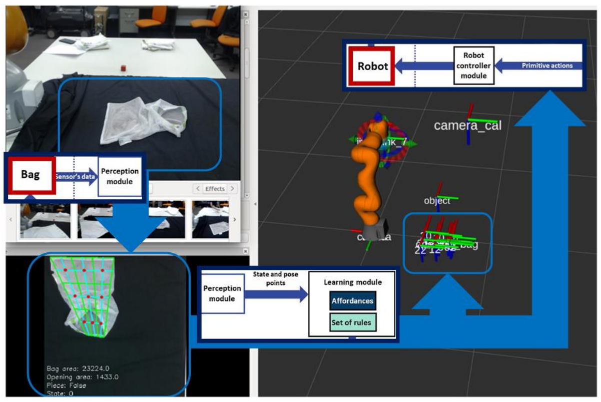


FIGURE 5 Implementation of the real-world learning robot-bagging framework.

TABLE 1 Characteristics and parameters of the bags used in the experiments.

Name	Opening length	Bag width	Material	A_{th}	A_{oth}	Ab_{max}	Ao_{max}
Bag 1	30 cm	35 cm	Cotton	25,000	150	34,000	3900
Bag 2	25 cm	25 cm	Polyester	18,000	50	28,000	3200
Bag 3	33 cm	26 cm	Cotton	25,000	150	34,000	3900
Bag 4	27 cm	40 cm	Polyester mesh	20,000	50	25,000	3900
Bag 5	47 cm	75 cm	Plastic	25,000	150	34,000	3900
Bag 6	36 cm	26 cm	Plastic	25,000	150	30,000	3900
Bag 7	31 cm	40 cm	Cotton	25,000	150	34,000	3900
Bag 8	29 cm	32 cm	Cotton	22,000	90	25,000	3900
Bag 9	46 cm	76 cm	Linen	25,000	150	34,000	3900

$$(x_i, y_i) = \left(x_0 + \frac{i}{\zeta}\cdot(p_{n1} - p_{n-3}), y_0 + \frac{i}{\zeta}\cdot(p_{n+1} - p_{n-1})\right) \tag{34}$$

here, i ranges from 0 to $\zeta - 1$, (x_i, y_i) represents the evenly distributed points and n represents the side of the 4-sided figure. If n is equal to 1, it corresponds to side F. If n equals 2, it corresponds to side G, and so on. With these points, a grid can be generated inside the geometry. Hence, the grasping points are at the centre of each grid cell.

The implementation of primitive actions relies on the perception module's analysis of RGB and depth images from the RealSense camera. Algorithm 1 contributes to determining the bag's opening area, and Equation (34) determines the evenly distributed points on the sides of the 4-sided geometry for the body of the bag. With these points, a grid can be generated inside the geometry. Hence, the grasping points are at the centre of each grid cell. This integration of perception and primitive actions defined as services in ROS allows the robot to execute the bagging task defined in this paper.[1]

5 | EXPERIMENTAL SETUP

The experimental setup to empirically validate our framework includes three bags (Table 1), the object to be bagged (red cube), a Kuka® iiwa 1400™ robot, and an Intel® RealSense™ camera (Figure 3). The task's goal is to train the robot to learn to bag the object (red cube). The experiments start by training several agents to learn how to handle 'Bag 1' in the real world for 10 (Ours (10)), 30 (Ours (30)), 50 (Ours (50)), and 100 (Ours (100)) training steps for each state of the task (unfolding, opening, placing the piece, and carrying), totaling 40, 120, 200 and 400 total training steps, respectively. Then, we compare the performance of our framework against the following state-of-the-art algorithms using the implementations from the stable baselines [35]: duelling deep Q-network (DQN) [36] and asynchronous actor–critic (A2C) [37], two robust and well-tested algorithms for discrete action spaces. When the training is finished, we run 10 attempts for each agent and count the number of times the bagging task is successfully finished for each step. Additionally, 10 attempts are performed to calculate the success rate of all the algorithms when starting from the unfolding and opening steps. With this experiment, we aim to discover if our framework can learn better than the baseline algorithms.

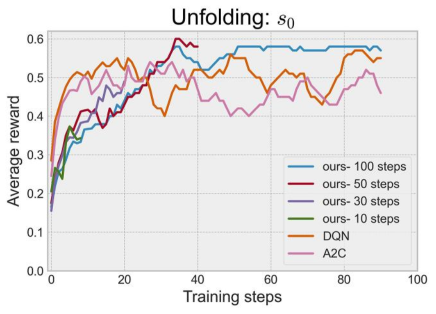
The trained agent with the highest success rate is used to test the generalisation capacities of the framework. To evaluate the framework's proficiency in handling the task from a different starting position, we change the position and orientation of 'Bag 1' twice. Then, the framework is tested on the rest of the bags for 10 attempts.

6 | RESULTS

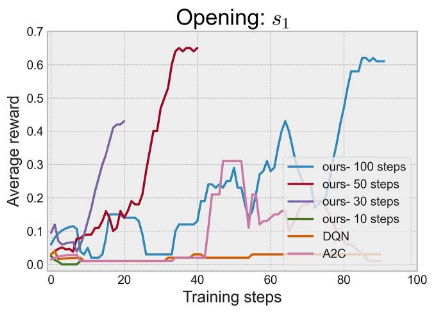
This section presents the results of the experiments previously described. Firstly, the learning progress of the framework and the state-of-the-art algorithms for each step of the task (unfolding, opening, placing the piece, and carrying) are illustrated in Figure 6. Secondly, Table 2 shows the total reward obtained by each algorithm, followed by Table 3, which contains the reward obtained by all the approaches for each step of the bagging task. Table 4 shows the success rates from performing the bagging task 10 times with 'Bag 1' for each step of the task as well as the success rates starting from step 1 (opening) and from step 2 (unfolding). Then, Figure 7 shows the robot performing the task in different initial positions with 'Bag 1'. Lastly, the success rates of handling all the bags are summarised in Table 5.

The learning curves in Figure 6a show the progress of the agent learning to unfold the bag, which consists of selecting a grasping point, lifting the bag, and dropping it till it is unfolded. Our framework running for 100 training steps converged in around 50 training steps, while DQN and A2C demonstrated unstable learning behaviour because they struggled to converge. The framework running 10 and 30 training steps shows that the agent requires more exploration. On the other hand, our

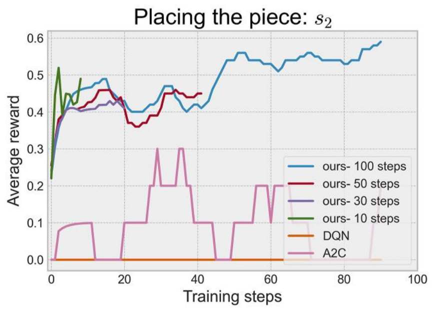
[1]For further technical details, the reader is encouraged to visit: https://github.com/FranciscoMunguiaGaleano/LearningToBag.



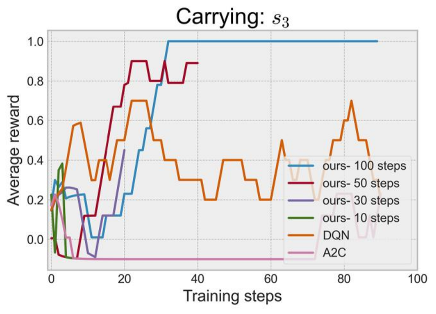
(a)



(b)



(c)



(d)

FIGURE 6 The learning curves above display the results of our experiments. In (a), the learning curve progress of the unfolding step demonstrates that our approach converges after 100 training steps while duelling DQN and A2C struggle to do so. In (b), our approach was the only one to converge after training for 100 and 50 training steps. In (c), A2C and DQN failed to find a solution while our approach converged. Lastly, in (d), our approach trained for 100 steps and converged to the highest value. A2C, asynchronous actor-critic; DQN, deep Q-network.

framework learning for 50 training steps indicates that 50 is the minimum number of training steps required to find the best grasping and lifting positions for our approach.

The learning curves in Figure 6b illustrate the learning progress of the agents for the opening step, which involves grasping one layer of the bag and dragging it to another point of the bag. The reward during this step is equal to the total area of the opening. The framework that ran for 100 training steps converged in approximately 80 training steps, while the one that ran for 50 training steps found the best solution in around 40 training steps. This is due to the stochastic nature of the exploration, which randomly found a better action in an early stage of the learning process for the 50-training steps experiment. DQN and A2C fell into a local minimum, and their learning could not progress. Our framework running for 10 and 30 training steps could not explore the environment enough to find an optimal policy.

The learning curves in Figure 6c show the progress of the agent learning to place the goal object (red cube) in the bag's opening such that the closer the robot places the goal object, the higher the reward. Our framework running for 10, 30, 50 and 100 training steps converged faster than the previous steps because this step is more straightforward than the previous steps by including only nine placing points in the bag's opening. The frameworks running for 10, 30, 50 and 100 could converge, while DQN and A2C could not find a solution.

The learning curves in Figure 6d show the learning progress of the agents for the carrying task. In this step, if the robot executes the carrying action and the red cube is not on the table after that, the reward is equal to 1 and -0.1 otherwise. The framework that runs for 50 and 100 training steps could converge to a stable plateau, while running for 10 and 30 could not result in the right grasping point for carrying the bag. DQN fell into a local minimum and could not find a solution. A2C could find a better grasping point to carry the bag. However, the learning curve shows that A2C struggles to converge.

The success rates of all the approaches are summarised in Table 5, in which 10 attempts were performed for each step of the bagging task. For step 1, Ours (100) and Ours (50) reached the highest success rate (70%), followed by DQN (40%). The lowest success rates were achieved by Ours (30) and A2C with 0% and 20%, respectively. For step 2, A2C, DQN, and Ours (10) presented the lowest success rates, which demonstrates that step 2 is the most difficult to learn. For step 3, all the approaches reached a success rate equal to or superior to 90%, which demonstrates that this step is the easiest to learn. In the last step, Ours (10) had the lowest performance with 60% while the rest of the approaches reached a success rate equal to or superior to 90%. Additionally, 10 attempts were carried out from step 1 (unfolding) and step 2 (opening), which for ours (100) resulted in 60% and 80% of success rates, respectively. It can be observed that the low success rates of ours (10), DQN, and A2C are because of getting stuck on step 2. Moreover, the difficulty increases when the task starts from step 1, which is reflected in the performance of all the agents.

The total average reward obtained by all the approaches is summarised in Table 2, where the three approaches that collected the highest rewards are ours (100), ours (50), and

DQN with 1.9608, 1.7 and 1.03, respectively. When it comes to the total average reward collected for each step, Table 3 shows that for step 1, DQN collected the highest reward of 0.51, followed by ours (100), which collected 0.5. For step 2, ours (50) collected the highest reward of 0.28, followed by ours (100) with 0.241. In step 3, the highest reward was obtained by ours (100). For step 4, the highest reward was obtained by ours (100), followed by ours (50) and DQN. In general, the stable baselines DQN and A2C presented problems learning step 2 (opening), while the rewards collected for all the approaches in step 1 are almost the same.

The last experiment tested the generalisation capabilities of the framework. The success rates are summarised in Table 4. Figure 7 illustrates the robot performing the bagging task with 'Bag 1' starting from two different positions and orientations. Despite the change in the initial position, the framework was capable of finishing the task. This is because our approach focuses on the state and grasping points with respect to the bag, which is independent of the pose of the bag in the global workspace frame. The robot performed the bagging task with 'Bag 2' with a success rate of 20% when starting from step 1% and 50% when starting from step 2. The robot performing the bagging task with 'Bag 3' had a success rates of 30% and 70% when starting from step 1 and step 2, respectively. Most of the failures are caused by the robot not being able to open or unfold the bag.

For 'Bag 4', the framework fails to complete the task when starting from steps 1 and 2. The main reason is that the gridded fabric does not allow air to unfold the bag as it falls, and it also causes the gripper to grasp two layers instead of one. Nevertheless, when starting from step 3, the robot solves that state with an 80% success rate and a 20% success rate when attempting step 4. In the case of 'Bag 5', a large plastic bag, the unfolding strategy is ineffective since the large area of the bag does not allow it to be completely unfolded. Consequently, when starting the task from step 1, the framework performed with a 0% success rate. However, when starting the task from step 2, the framework reached a success rate of 50%.

When the robot performed the task with 'Bag 6', it failed to complete the task in steps 1 and 2. This poor performance was due to the robot's inability to hold the bag while opening it. Since the plastic bag is too light, the robot drags it instead of opening it. However, when placing the object in the opening and lifting the bag in steps 3 and 4, respectively, the robot managed to complete the task with a 10% success rate for step 3 and 80% for step 4. For 'Bag 7', despite being of a similar size to 'Bag 1', the layer thickness provoked a reduction in the overall success rate, which was 10% when starting from step 1% and 50% when starting from step 2.

When the robot performed the bagging task with 'Bag 8', it failed to complete it when starting from step 1 due to a similar reason as 'Bag 7', which is that the thickness of the layers did not allow the robot to grasp the bag properly. Lastly, while performing the task with 'Bag 9', a linen-made and large-sized bag, the robot reached a 40% success rate when starting from step 2 but failed when starting from step 1 for the same reason explained for 'Bag 5'.

TABLE 2 Total reward obtained by our framework and stable baselines after training.

Approach	Total reward	Training time	Total training steps
Ours (100)	1.9608	173 min	400 (100 for each step of the task)
Ours (50)	1.7000	95 min	200 (50 for each step of the task)
Ours (30)	0.9900	62 min	120 (30 for each step of the task)
Ours (10)	0.2475	18 min	40 (10 for each step of the task)
DQN	1.0300	198 min	400 (100 for each step of the task)
A2C	0.6488	186 min	400 (100 for each step of the task)

Abbreviations: A2C, asynchronous actor-critic; DQN, deep Q-network.

TABLE 3 Reward obtained by our framework and stable baselines after training per step.

Approach	Reward step 1	Reward step 2	Reward step 3	Reward step 4
Ours (100)	0.500	0.241	0.490	0.730
Ours (50)	0.460	0.280	0.440	0.520
Ours (30)	0.390	0.248	0.430	0.220
Ours (10)	0.366	0.014	0.500	0.110
DQN	0.510	0.020	0.120	0.380
A2C	0.470	0.070	0.100	0.009

Abbreviations: A2C, asynchronous actor-critic; DQN, deep Q-network.

TABLE 4 Success rate of the framework and stable baselines after training.

Experiment 'bag 1'	Step 1	Step 2	Step 3	Step 4	Success rate from step 1	Success rate from step 2
Ours (100)	7/10	9/10	9/10	10/10	6/10	8/10
Ours (50)	7/10	7/10	9/10	10/10	5/10	6/10
Ours (30)	2/10	4/10	10/10	10/10	1/10	4/10
Ours (10)	0/10	0/10	10/10	6/10	0/10	0/10
DQN	4/10	1/10	10/10	9/10	0/10	2/10
A2C	2/10	1/10	10/10	10/10	0/10	1/10

Abbreviations: A2C, asynchronous actor-critic; DQN, deep Q-network.

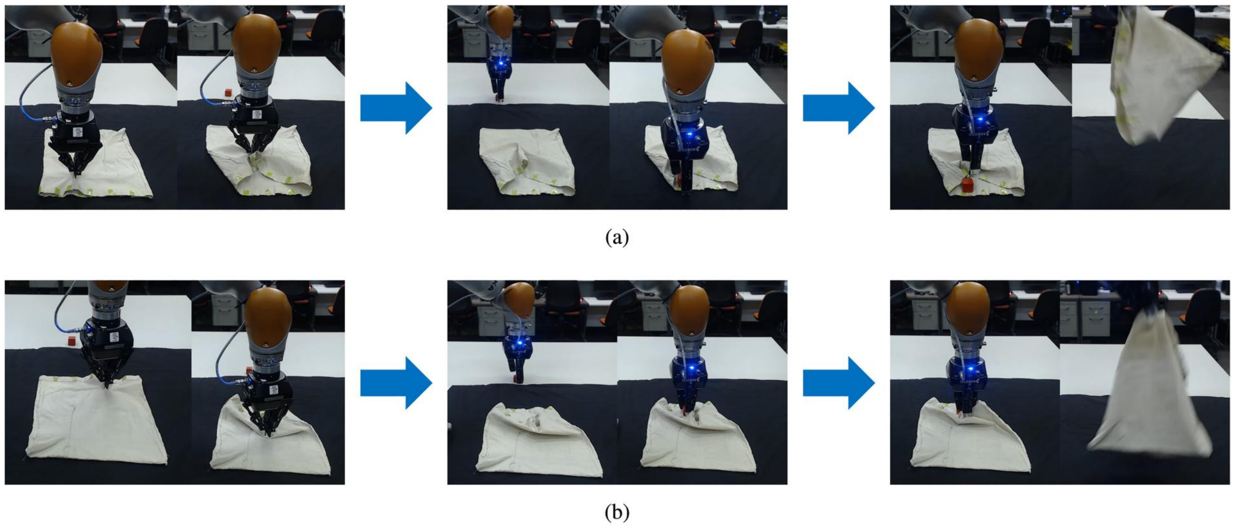


FIGURE 7 The robot performing the bagging task with two different bags. In (a), the bag's opening faces the camera's view. In (b), the bag's opening is facing the opposite direction of the camera's view. The robot successfully completed the tasks in both cases with different bag orientations.

TABLE 5 Success rate of the framework and stable baselines after training per step.

Experiment	Step 1	Step 2	Step 3	Step 4	Success rate from step 1	Success rate from step 2
Bag 1	6/10	8/10	9/10	9/10	6/10	8/10
Bag 2	4/10	6/10	10/10	8/10	2/10	5/10
Bag 3	5/10	8/10	10/10	9/10	3/10	7/10
Bag 4	0/10	0/10	8/10	2/10	0/10	0/10
Bag 5	0/10	5/10	10/10	8/10	0/10	4/10
Bag 6	0/10	0/10	10/10	9/10	0/10	7/10
Bag 7	2/10	7/10	10/10	9/10	1/10	5/10
Bag 8	1/10	8/10	9/10	9/10	0/10	6/10
Bag 9	0/10	5/10	10/10	9/10	0/10	4/10

7 | DISCUSSION

Prior to the current work, we implemented the DQN and A2C algorithms aiming to solve the problem of learning bagging in the real world. However, we observed low performance during the bagging task with DQN and A2C (refer to Table 2), which we attribute to the limited number of training steps (400 training steps in total). Achieving better results with these algorithms would likely require implementing them in a simulated environment. However, this approach presents challenges, such as the sim-to-real problem discussed in Section 2 and the need to accurately simulate the physical properties of the bag. Furthermore, DQN and A2C algorithms typically require thousands to millions of steps to achieve stable learning. For instance, Hester et al. [38] demonstrated that the Deep Q-Network from Demonstrations required 1 million steps to achieve satisfactory scores in their experiments. DQN took 84–85 million steps for similar performance in the same application. This problem led to the design of the Π-learning algorithm.

Additionally, after completing the learning phase with our framework, the robot performed 100 attempts using each approach for the bagging task with 'Bag 1'. The success rates are summarised in Table 2 in which it can be appreciated that the main reason for the failures was the robot's inability to complete the unfolding (step 1) or opening of the bag (step 2). This highlights the need for further improvements, such as a more robust unfolding strategy and enhanced camera measurement accuracy. The improved accuracy of the camera's measurements would allow the robot's gripper to reach the surface of the bag's layer more precisely, as the lack of precise measurements of the camera caused the robot to grasp both layers or failed to grasp any layer at all.

The decision to not use continuous RL algorithms, such as Deep Deterministic Policy Gradient [39] or Soft Actor-Critic [40], is motivated by the aim of preventing dangerous behaviours of the robot in our experimental setup, particularly during the learning stage. Unlike continuous RL algorithms, which may lead to collisions and undesired behaviours due to their inherent exploration process, defining primitive actions and employing a discrete action–selection approach helped to prevent such incidents specifically for the task proposed in this paper. To summarise, the following list encompasses the main limitations of the proposed framework and, therefore, potential challenges that require further research:

- **Unfolding difficulty**: The current approach faces challenges when it comes to unfolding the bag. Problems such as the bag being too large or the opening not being visible after following our approach prevent the framework from generalising to a wider set of bags.
- **Opening difficulty**: The robot encounters issues in opening the bag after the unfolding step. This is due to the resistance of the same bag's material to change its initial configuration such as in the case of plastic bags or because the robot grasps two layers instead of one.
- **Accuracy of the camera**: The camera's accuracy is not sufficient for the robot to distinguish between layers. Even a difference of 1 mm causes the robot to grasp both layers or fails to grasp any layer.
- **Automatically recognise the bag's opening with no markers**: For the cases presented in this paper, the bags had no handles, making it ambiguous and challenging to find the opening using only a camera. This process is difficult even for humans who often need to rotate the bag several times before finding the opening.

Some potential solutions to the challenges listed above are proposed as follows:

- **Improved unfolding and opening strategy**: Adding a second arm or using a human-like gripper would allow the implementation of more robust strategies involving dexterity or simply adding more resources to the robot to hold the bag while attempting more actions.
- **Incorporating tactile sensing**: The camera's low accuracy limitations can be overcome by adding fingers with tactile feedback. This would allow the robot to localise the real position of the layers, and in this manner, know when the gripper has held one or two layers of the bag.
- **Enhanced exploration strategy**: To overcome the ambiguity regarding how to find the opening of the bag without markers, it would be necessary to implement a process that involves robustly unfolding the bag. Then, one side of the bag was explored. If that side does not open, continue with the next side if the bag is still unfolded. These actions are repeated until the robot finds the opening of the bag.

8 | CONCLUSION

This paper presented a learning framework for a robot manipulator to acquire the bagging task. Our real-world learning robot-bagging framework has been empirically validated. Leveraging our novel RL algorithm Π-learning, this framework enables efficient learning of the bagging task on a simulation-free basis. After training for a total of 400 steps, which took approximately 3 h, the framework achieved success rates of 60% and 80% when starting from the unfolding or opening step, respectively. Additionally, the framework demonstrated potential generalisation capabilities across different bags.

However, there are certain limitations to the proposed framework. For instance, the framework was shown to be more reliable with bags made from specific materials, such as textile-based bags, and had a poor performance while handling plastic bags. Moreover, the framework is designed to handle only one object smaller than the bag's opening. Additionally, there is room for improvement in the primitive actions of unfolding the bag (τ_{grasp}) and grasping only one layer of the bag ($\tau_{scratch}$) as these actions were the primary causes of the robot's failure in the bagging tasks. Despite these limitations, our framework exhibits the potential to tackle challenging problems such as bag manipulation.

In future work, we have plans to enhance the unfolding and opening routines by incorporating bi-manual robotic manipulation techniques. This advancement would enable the bagging of multiple objects, expanding the framework's capabilities beyond a single object. Additionally, we aim to explore the integration of supervised learning methods and the integration of tactile sensing, which would facilitate the generalisation of the perception module to a wider range of bag types. This extension would enhance the framework's versatility and applicability.

ACKNOWLEDGEMENTS

This work was partially supported by Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCyT) and the Engineering and Physical Sciences Research Council (grant No. EP/X018962/1).

CONFLICT OF INTEREST STATEMENT

The authors declare no known competing financial interests or personal relationships that could influence the work reported in this paper.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Francisco Munguia-Galeano https://orcid.org/0000-0001-8397-3083

REFERENCES

1. Culleton, M., McGinn, C., Kelly, K.: Framework for assessing robotic dexterity within flexible manufacturing. J. Intell. Rob. Syst. 87, 507–529 (2017)
2. Zhu, J., et al.: Challenges and outlook in robotic manipulation of deformable objects. IEEE Robot. Autom. Mag. 29(3), 67–77 (2022). https://doi.org/10.1109/mra.2022.3147415
3. Balkcom, D.J., Mason, M.T.: Introducing robotic origami folding. In: IEEE International Conference on Robotics and Automation, 2004, vol. 4, pp. 3245–3250. IEEE (2004). https://doi.org/10.1109/ROBOT.2004.1308754
4. Balkcom, D.J., Mason, M.T.: Robotic origami folding. Int. J. Robot Res. 27(5), 613–627 (2008). https://doi.org/10.1177/0278364908090235

5. Elbrechter, C., Haschke, R., Ritter, H.: Bi-manual robotic paper manipulation based on real-time marker tracking and physical modelling. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1427–1432. IEEE (2011)
6. Elbrechter, C., Haschke, R., Ritter, H.: Folding paper with anthropomorphic robot hands using real-time physics-based modeling. In: 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012), pp. 210–215 (2012)
7. Borràs, J., Alenyà, G., Torras, C.: A grasping-centered analysis for cloth manipulation. IEEE Trans. Robot. 36(3), 924–936 (2020). https://doi.org/10.1109/tro.2020.2986921
8. Hoque, R., et al.: Visuospatial foresight for physical sequential fabric manipulation. Aut. Robots 46(1), 175–199 (2022)
9. Jangir, R., Alenya, G., Torras, C.: Dynamic cloth manipulation with deep reinforcement learning. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 4630–4636. IEEE (2020)
10. Nair, A., et al.: Combining self-supervised learning and imitation for vision-based rope manipulation. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 2146–2153. IEEE (2017)
11. Shi, L., et al.: Reactive motion planning for rope manipulation and collision avoidance using aerial robots. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3384–3391. IEEE (2022)
12. Sundaresan, P., et al.: Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 9411–9418. IEEE (2020)
13. Zhou, H., et al.: A practical solution to deformable linear object manipulation: a case study on cable harness connection. In: 2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM), pp. 329–333. IEEE (2020)
14. Zhu, J., et al.: Robotic manipulation planning for shaping deformable linear objects with environmental contacts. IEEE Rob. Autom. Lett. 5(1), 16–23 (2019). https://doi.org/10.1109/lra.2019.2944304
15. Jørgensen, T.Bo, et al.: An adaptive robotic system for doing pick and place operations with deformable objects. J. Intell. Rob. Syst. 94(1), 81–100 (2019). https://doi.org/10.1007/s10846-018-0958-6
16. Sharma, S., et al.: Learning switching criteria for sim2real transfer of robotic fabric manipulation policies. In: 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), pp. 1116–1123. IEEE (2022)
17. Polydoros, A.S., Nalpantidis, L.: Survey of model-based reinforcement learning: applications on robotics. J. Intell. Rob. Syst. 86(2), 153–173 (2017). https://doi.org/10.1007/s10846-017-0468-y
18. Culleton, M., McGinn, C., Kelly, K.: Framework for assessing robotic dexterity within flexible manufacturing. J. Intell. Rob. Syst. 87(3-4), 507–529 (2017). https://doi.org/10.1007/s10846-017-0505-x
19. Nguyen, H., La, H.: Review of deep reinforcement learning for robot manipulation. In: 2019 Third IEEE International Conference on Robotic Computing (IRC), pp. 590–595. IEEE (2019)
20. Francisco Munguia Galeano: Learning to bag. https://github.com/FranciscoMunguiaGaleano/LearningToBag (2024). Accessed 6 March 2024
21. Seita, D., et al.: Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 4568–4575. IEEE (2021)
22. Bahety, A., et al.: Bag all you need: learning a generalizable bagging strategy for heterogeneous objects. *arXiv preprint arXiv:2210.09997* (2022)
23. Ma, X., Hsu, D., Lee, W.S.: Learning latent graph dynamics for visual manipulation of deformable objects. In: 2022 International Conference on Robotics and Automation (ICRA), pp. 8266–8273. IEEE (2022)
24. Wang, X., et al.: Learning-based fabric folding and box wrapping. IEEE Rob. Autom. Lett. 7(2), 5703–5710 (2022). https://doi.org/10.1109/lra.2022.3158434
25. Gao, C., et al.: Iterative interactive modeling for knotting plastic bags. In: Conference on Robot Learning, pp. 571–582. PMLR (2023)
26. Chen, L.Y., et al.: Autobag: learning to open plastic bags and insert objects. In: 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 3918–3925. IEEE (2023)
27. Dulac-Arnold, G., et al.: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Mach. Learn. 110(9), 2419–2468 (2021). https://doi.org/10.1007/s10994-021-05961-4
28. Zhao, W., Queralta, J.P., Westerlund, T.: Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 737–744. IEEE (2020)
29. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
30. Bellman, R.: Dynamic programming. Science 153(3731), 34–37 (1966). https://doi.org/10.1126/science.153.3731.34
31. Browne, C.B., et al.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games 4(1), 1–43 (2012). https://doi.org/10.1109/tciaig.2012.2186810
32. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. 8(3/4), 279–292 (1992). https://doi.org/10.1023/a:1022676722315
33. Munguia-Galeano, F., Tan, A.-H., Ji, Z.: Deep reinforcement learning with explicit context representation. IEEE Transact. Neural Networks Learn. Syst., 1–14 (2023). https://doi.org/10.1109/tnnls.2023.3325633
34. Munguia-Galeano, F., et al.: Affordance-based human–robot interaction with reinforcement learning. IEEE Access 11, 31282–31292 (2023). https://doi.org/10.1109/access.2023.3262450
35. Hill, A., et al.: Stable baselines. https://github.com/hill-a/stable-baselines (2018). Accessed 22 Feb 2023
36. Wang, Z., et al.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1995–2003. PMLR (2016)
37. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937. PMLR (2016)
38. Todd, H., et al.: Deep Q-learning from demonstrations. Proc. AAAI Conf. Artif. Intell. 32(1) (2018). https://doi.org/10.1609/aaai.v32i1.11757
39. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015)
40. Haarnoja, T., et al.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870. PMLR (2018)

How to cite this article: Munguia-Galeano, F., et al.: Learning to bag with a simulation-free reinforcement learning framework for robots. IET Cyber-Syst. Robot. e12113 (2024). https://doi.org/10.1049/csy2.12113