# MSc. Thesis

## Robotic Packaging Optimization with Reinforcement Learning and Real-World Data

E.A. Drijver

Delft University of Technology

TU Delft

# Robotic Packaging Optimization with Reinforcement Learning and Real-World Data

Thesis paper

by

Eveline Drijver

to obtain the degree of Master of Science
in Mechanical Engineering
at the Delft University of Technology
to be defended publicly on March 9, 2023 at 14:00

| *Thesis committee*: | | |
|---|---|---|
| Chair: | Dr. C. Della Santina | TU Delft |
| Supervisors: | PhD. Z. Ajanović | TU Delft |
| | MSc. R.J. Perez Dattari | TU Delft |
| | MSc. M. Hulscher | BluePrint Automation |
| External examiner: | Dr. J. Kober | TU Delft |
| Place: | Faculty of Mechanical, Maritime and Materials Engineering (3mE), Delft | |
| Project Duration: | May, 2022 - March, 2023 | |
| Master specialisation | BioRobotics | |
| Student number: | 4467590 | |


# Robotic Packaging Optimization with Reinforcement Learning and Real-World Data

Eveline Drijver
*Department of Cognitive Robotics*
*Delft University of Technology*
Delft, The Netherlands
E.A.Drijver@student.tudelft.nl

***Abstract*—Intelligent manufacturing has become increasingly important in the food packaging industry due to the growing demand for enhanced productivity and flexibility while minimizing waste and lead times. This work explores the integration of such manufacturing in automated secondary robotic food packaging solutions that transfer food products into containers using pick-and-place task scheduling. A major problem in these solutions is varying product supply caused by prior machinery. As a result productivity drops drastically when conveyor belt speeds are not optimally controlled. Conventional heuristic-based engineered approaches are used to address this issue but are inadequate, leading to noncompliance with industry's requirements. Reinforcement learning, on the other hand, has the potential of solving this problem by learning quick and predictive decision-making behavior based on experience. However, the lack of research in reinforcement learning for complex industrial robotic problems, limits its adoption in industry. Therefore, this work aims to investigate the feasibility of reinforcement learning in the robotic packaging industry. We propose a reinforcement learning framework, with policy inference in a highly complex control scheme, designed to optimize the conveyor belt speed of the secondary robotic packaging solution using real-world product supply data. The framework exceeds the 99.8 percent performance requirement and maintains quality at the required 100 percent when tested on real-world data. Compared to the current heuristic-based solution, our proposed framework improves productivity, has smoother control and reduces code execution time. Video of robotic food packaging solution can be found here (youtube.com/watch?v=AdiaRRsN7QQ).**

***Index Terms*—Conveyor optimization, food packaging, intelligent manufacturing, intelligent control, reinforcement learning, task scheduling.**

## I. Introduction

With the manufacturing industry aiming towards flexibility, productivity, quality and mass customization [1], the adoption of intelligent manufacturing becomes increasingly important. This is especially relevant for the highly competitive fields such as the food packaging industry [2]. One of the primary challenges in the food packaging industry is to enhance productivity while simultaneously reducing lead times. Achieving this objective entails developing packaging solutions, that minimize waste and maximize food production rates, in the shortest time possible.

*BluePrint Automation* is a business-to-business company in this sector, that designs, develops, and manufactures automated robotic packaging solutions for a wide range of food products typically found in retail stores [3]. The objective of these machines is to provide food supply companies with a highly efficient and continuous food processing system. BluePrint Automation provides various packaging solutions, including both primary and secondary options. The primary packaging solutions are designed to handle naked and un-packaged products, such as cookies and chocolates. The secondary packaging solutions are designed to handle already packaged products that require placement into a secondary container, such as bags of chips that need to be transported in boxes to supermarkets. [3].

Fig. 1: BluePrint Automation's SPIDER Q series robotic packaging solution [4]. A configuration with three Delta robots with rotational end-effectors and vacuum grippers is demonstrated, picking up pre-packaged food products and placing them into secondary containers.

A major problem in such secondary robotic packaging solutions is the inability to achieve a 100%-container fill rate, which is a critical quality requirement, under varying product supply caused by prior machinery i.e., product feeders and primary packaging solutions. As a result productivity drops significantly which leads to revenue drops, waste and critical problems later in packaging process e.g., sealing of the secondary container. In Fig. 1, the internal components of such a secondary packaging solution are depicted. The system comprises three pick-and-place Delta robots, each equipped with a rotational end-effector and two vacuum grippers. The

Delta robots pick up incoming products, specifically bags of chips in this instance, and placing them into outgoing containers i.e., boxes. In order to optimize the the pick-and-place sequence a task scheduler is used to allocate sets of products to containers. The aim of this scheduler is to maximize the throughput in packaged products per minute. However, as previously mentioned, certain food supply companies require a $100\%$ container fill rate in addition to this objective. To achieve this requirement under varying product supply conditions, it is crucial to optimize the conveyor belt speed with containers during the operation of the robotic packaging machine.

Commonly, this problem is addressed with heuristic-based engineered solutions. However, such solutions tend to compromise the performance of the task scheduler as they attempt to meet the aforementioned container fill rate requirement. Especially in the edge case scenarios that appear in practice, such as the varying product supply. As a result, requirements regarding the productivity of the machine are violated.

Intelligent manufacturing has the potential of solving this issue. Hence, Reinforcement Learning (RL) is used in this work to learn the optimal box conveyor belt speed in order to optimize productivity, minimize waste and reduce lead times. When properly implemented, deep model-fee RL offers the following features making it well-suited for this complex real-world problem:

- **Flexible**: No re-engineering of heuristics is necessary for different robotic packaging machines, provided that the problem's objective remains the same. Only retraining is required.
- **Fast**: Time-demanding calculations are performed during training prior to deployment of the learned policy, assuming the neural network is not too large.
- **Generalization**: Learned policies can generalize to unseen situations, which is especially useful for edge case scenarios that appear in practice.
- **Data-based predictions**: By training an RL agent using interactions with the robotic packaging machine (online) or fixed logged data (offline), it can learn to prevent problems caused by varying product supply, rather than simply correcting them.

Although, RL methods have demonstrated success in various simulated environments, their adoption in real-world applications has been relatively slow. This can be attributed, in large part, to a significant gap between the way current experimental RL setups are designed and the often ill-defined nature of real-world systems [5] [6]. According to [7], nine unique challenges must be addressed in order to deploy RL in a real-world setting. These challenges include dealing with large system delays, satisfying safety and performance constraints and enabling inference at the control frequency of the system. The work of [8], [9] and [10] supports this by outlining recent advances in safe RL, particularly in robotics, that are limited to academic environments, resulting in a focus on simulated or simple real-world tasks. Consequently, potential new challenges that arise in industry on complex robotic systems remain unsolved.

Therefore, this research aims to investigate the feasibility of RL in the industry by solving the earlier introduced robotic packaging optimization problem using BluePrint Automation's simulator and real-world product supply data. We propose a RL approach designed to optimize the box conveyor belt speed in order to maximize the performance of the task scheduler under varying product supply. The proposed method is able to deal with control delays, sparse delayed rewards and policy inference in the highly complex control scheme of the task scheduler. With the proposed method predictive behavior for varying product supply is learned, while satisfying the machines performance constraints. The design encourages smooth control of the conveyor belt and handles the limited availability of real-world product supply data well. Although the proposed framework has not been tested on a physical machine, the combination of RL applied to a real-world problem using real-world data makes this work a valuable contribution to the field.

In this paper we describe the proposed method, present an ablation study with simulated data and validate our method with real-world data. This work contributes:

- a RL solution for a real robotic optimization problem in the food packaging industry.
- a RL solution with policy inference in a highly complex control scheme.
- validation of the RL solution with real-world data from fixed logs.

## II. Related Work

### A. Real-World challenges

One of the significant works addressing the challenges of real-world reinforcement learning is by Google's Brain and Deepmind in [7]. They state that in order to deploy RL in a real-world setting, nine unique challenges must be addressed. These challenges include dealing with large and unknown system delays, partial observability, multi-objective reward functions, creating explainable policies, satisfying safety and performance constraints, handling high-dimensional continuous state and action spaces, enabling real-time inference, dealing with limited samples from a live system, and offline learning from fixed logs (Appendix I). The main conclusion of this work is that existing research only focuses on a subset of these challenges, and few studies have attempted to tackle all of them simultaneously. The latter, being necessary for fast adoption of RL in the real-world. Similarly, the work of [6] explores the challenges that arise when learning autonomously and consistently on a robotic learning system with use of real-world data and reinforcement learning. However, the promising real-world results of this work are limited to a designed experimental setup. In this paper, we aim to build upon these works and explore the challenges that arise when applying RL to a robotic problem in the food packaging industry using real-world data.

### B. Real-World Applications

Examples of successful applications of RL in industry are limited, however cases that have succeeded include Google DeepMind's data-center electricity optimization project and the data-center cooling project in [11]. The former being the most impressive achievement, where the electricity usage of Google's data center was reduced with 40% [12]. However, such success stories are not seen in the robotics packaging industry yet. On the other hand, the work of [13] closely examines the feasibility of RL in real-world settings by demonstrating promising results on multiple reference tracking tasks using a physical industrial robotic manipulator. Our work is related to this research, as we also aim to explore the feasibility of RL in industrial robotic systems. However, our focus is not on the physical implementation in an experimental setup, but rather on evaluating the feasibility in a real-world robotic problem within an industrial setting.

### C. Safe Reinforcement Learning

A popular field of research in RL for robotics is safe RL, which is concerned with learning policies that ensure reasonable system performance and/or safety constraints during the learning and/or deployment process. This is particularly important for the transfer of RL towards industry, where strict safety and performance requirements must be met. The work of [8] and [9] provide an comprehensive overview of the research in this field. Various methods can be used to address the performance and safety requirements in RL. For instance, [10] is devoted to model-free RL for policy learning using constrained Markov decision processes. Additionally, Safety Layers [14] can be used to safely explore Markov decision processes, while Risk-Averse Robust Adversarial RL [15] is useful for learning robust risk-aware probabilistic models. Another approach is Bayesian Controller Fusion [16], which allows for accelerated safe learning with suboptimal controllers. However, current research in the field of safe RL is largely limited to simulated or simplistic real-world robotic tasks. Moreover, most approaches do not offer formal guarantees for the satisfaction of hard or probabilistic constraints during the policy's learning and deployment process [8]. Inspired by these works, we aim to test the feasibility of RL in a more complex real-world robotic problem subjected to strict performance constraints.

### D. Task Scheduling

Despite multiple attempts for using RL for task scheduling [17] - [21], applicability of such work in practice remains limited. The work of [18] models the Job Shop Scheduling Problem (JSSP), which is a widely studied optimization problem for optimal scheduling of resources over time, as a sequential decision making problem and uses deep RL to solve it. Despite, the promising results in static JSSP benchmark problems, the RL-based solution is not able to beat the traditional based heuristic solutions in dynamic environments. In addition, our work shares many similarities with [21], where the authors aim to optimize production scheduling using a deep RL method (i.e., DQN) in the context of intelligent manufacturing. However, they did not surpass the performance of heuristic solutions in their study. Therefore, we argue that using RL for task scheduling in complex industrial robotic systems is currently infeasible and also impractical. This is because the task scheduling solutions that are already fully integrated into machines require mature and trustworthy replacements, making it challenging for RL-based task scheduling solutions to be successfully adopted. As such, we propose a different approach: optimizing an environmental condition rather than replacing an existing task scheduling solution.

## III. Problem Statement

First, we introduce the robotic packaging machine for which the reinforcement learning framework is created, followed by the definition of the belt speed optimization problem and the constrained Markov decision process. Finally, we introduce the high-level interaction framework between the machine's simulator and our proposed method.

### A. Robotic Packaging Machine

This research was conducted in collaboration with the company BluePrint Automation, hereinafter referred to as *the company*. A configuration of the company's secondary robotic packaging solution is illustrated in Fig. 2. The figure includes a complete machine at the bottom left and a simplified view of the machine's interior at the upper left, where a simplified vision system (1) detects the class and location of pre-packaged food products (2) on the conveyor belt. The packaging process involves multiple *Delta robots* (3), which are parallel manipulators that consist, in this case, of a four degrees of freedom end-effector (4) with three parallel arms

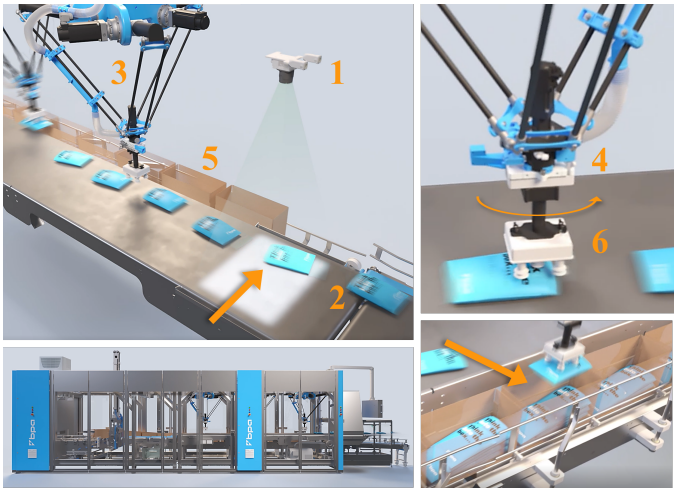



Fig. 2: Collection of renders of BluePrint Automation's SPIDER 300v robotic packaging solution showing the complete machine (bottom left), a simplified vision system with Delta robot (upper left), the Delta robot with rotational end-effector and vacuum gripper (upper right) and the containers containing multiple layers of products (bottom right) [4].

[22]. These Delta robots pick up the packaged products and place them in a container (5) i.e., boxes or pockets. The upper right corner of the figure shows a close-up of a Delta robot with a rotational end-effector (4) and *vacuum gripper* (6), which leverages the difference between its internal pressure and the surrounding atmospheric pressure to lift, hold and move the product. The bottom right corner of the figure provides a close up of a container, a box in this case, which can hold multiple *layers* of pre-packaged products. The contents of a container can either be a single class of product, or it can consist of a mix of different product classes in a predefined pattern. The packaging solutions can be extended with a *task scheduler*, which is a branch-and-bound algorithm that optimizes the pick-and-place sequence to maximize the throughput in products per minute.

This research is specifically focused on a secondary robotic packaging machine that contains four Delta robots, one conveyor belt with two product lanes, and one conveyor belt with a single lane of boxes. Fig. 4 depicts a schematic overview of this specific configuration. Both the conveyor belts move in the same direction, which is also referred to as a *co-flow configuration*. Moreover, the speed of the conveyor belt with products, denoted by $v_P$, cannot be controlled. Additionally, each box needs two layers with respectively two and three single class products. In this specific configuration, the task scheduler is designed to assign two Delta robots to a designated box, where each robot is responsible for filling one of the two layers in that box. The two Delta robots, five products and box together form a single schedule as visualized in Fig. 3. During deployment, the machine needs to satisfy the following requirements:

- No empty boxes may leave the machine.
- No partly filled boxes may leave the machine.
- At least 99.8 % of the supplied products must be placed in a box.

From now on, the term *robotic packaging machine* refers to this specific packaging machine that is subjected to the aforementioned requirements. To test the effectiveness of the task scheduler with respect to the these requirements we use a simulator of the machine that simulates e.g., system delays and product infeed. For measuring the overall effectiveness of the machine we use the *Overall Equipment Effectiveness (OEE)*

Fig. 3: A complete schedule contains two Delta robots, one box and five products divided into two layers. The task scheduler assigns the layers of a box to the Delta robots with the available products on the conveyor belt, aiming to maximize the number of packed products per minute. Each layer is placed in a single operation by a Delta robot.

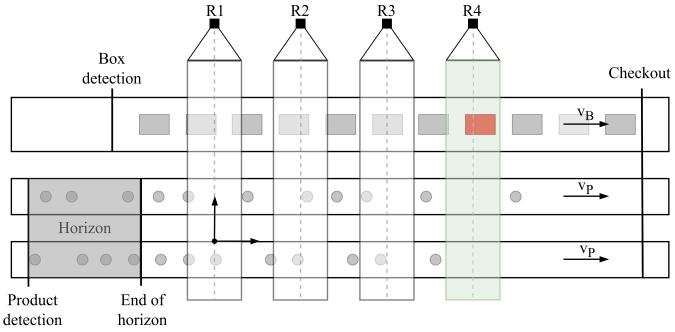



Fig. 4: Schematic top view of the robotic packaging machine with one box belt at the top and one product belt with two lanes at the bottom. The four Delta robots are shown with their 2D Cartesian work envelopes. The products and boxes enter the machine at respectively the box and product detection point and leave the machine at checkout. The variable box belt speed is denoted by $v_B$ and the constant product belt speed by $v_P$. The red box will soon be out of reach of robot 4 without any products available in the workspace of this robot.

industry standard [23], which is widely used in the manufacturing industry. The OEE metric determines the effectiveness of the machine as a ratio between actual and theoretical output [23] and is defined as:

$$\text{Performance}(\%) \times \text{Quality}(\%) \times \text{Availability}(\%) = \text{OEE}(\%). \quad (1)$$

We assume an *availability*, which is the ratio between actual and planned production time, of 100%, due to limited access to the factors affecting this index. The *performance* and *quality* are defined as:

$$\text{Performance} = \text{packed products / supplied products} \times 100\%, \quad (2)$$

$$\text{Quality} = \text{packed boxes / supplied boxes} \times 100\%. \quad (3)$$

The term *packed products* denotes the individual products that exit the robotic packaging machine enclosed in a box, and *packed boxes* refers to the boxes that leave the machine containing the specified number of products i.e., five.

### B. Belt Speed Optimization Problem

In Fig. 4, the schematic top view of the robotic packaging machine is shown with the 2D Cartesian work envelopes of the Delta robots and the box and product belt speeds $v_B$ and $v_P$ respectively. The task scheduler creates the schedules, as defined in Fig. 3, with the products in the horizon. For a constant product inflow and constant product and box belt speeds, the task scheduler is able to comply the requirements stated in Subsection III-A. However, in the case of varying product inflow and constant belt speeds, the Delta robot may not be able to reach the necessary products when an empty or partially filled box is about to leave the machine, which results in a violation of the requirements. This issue is illustrated by the red box in Fig. 4 and can lead to waste and critical problems later in the packaging process, such as during box sealing.

To ensure compliance with the requirements even under varying product inflow, the box belt speed must be optimized according to the changing product inflow. To achieve this, we propose the following minimization problem:

$$\min_{v_B[k]} \sum_{k=1}^{N} P_l[v_B[k], k], \tag{4a}$$

$$v_B[k] \in [v_{B,min}, v_{B,max}] \tag{4b}$$

where $P_l$ is an unknown function describing the behavior of the robotic packaging machine, which outputs the number of lost products at each time step, $k$, for a given box belt speed, $v_B$. We define a product as lost when it has passed the checkout point, which is denoted in Fig. 4. The objective function minimizes the number of lost products during an operating time of $N$ seconds of the machine by optimizing the box belt speed at each time step. By definition, this results in a maximization of placed products per minute, because the product belt speed, $v_P$, nor the product inflow can be controlled. The requirement regarding the placed products stated in Subsection III-A is modelled as:

$$\sum_{k=1}^{N} P_l[v_B[k], k] / \sum_{k=1}^{N} P_s[k] \leq 1 - 0.998 \tag{5a}$$

$$P_s[k] \neq 0 \tag{5b}$$

where $P_s$ is an unknown function describing the behavior of the robotic packaging machine, which outputs the number supplied products at each time step. This constraint limits the number of lost products, represented by $P_l$, to be lower than $0.2\%$ during an operating time of $N$ seconds of the machine. The two requirements regarding the boxes stated in Subsection III-A are modelled as:

$$B_e\left[v_B[k], k\right] = 0 \tag{6}$$

$$B_p\left[v_B[k], k\right] = 0 \tag{7}$$

where $B_e$ and $B_p$ are unknown functions describing the behavior of the robotic packaging machine, which output respectively the number of empty boxes and partly filled boxes at each time step for a given box belt speed. An additional requirement is necessary to limit the acceleration of the box belt, which is modelled as:

$$\sqrt{(v_B[k] - v_B[k-1])^2} / \Delta k \leq a_{B,max} \tag{8}$$

which calculates the Euclidean distance between two successive speeds and divides this by the time step length $\Delta k$. The maximum box belt acceleration is denoted by $a_{B,max}$. Eq. (9) shows the full constrained optimization problem. Solving (9) yields the box belt speed profile, $v_B[k]$, for an operating time of $N$ seconds of the robotic packaging machine, maximizing the number of placed products above 99.8% while ensuring that no empty or partially filled boxes are lost and the box belt acceleration remains in the feasible range.

$$\begin{aligned} \min_{v_B[k]} & \sum_{k=1}^{N} P_l[v_B[k], k] \\ \text{s.t.} & \sum_{k=1}^{N} P_l[v_B[k], k] / \sum_{k=1}^{N} P_s[k] \leq 1 - 0.998 \\ & \sqrt{(v_B[k] - v_B[k-1])^2} / \Delta k \leq a_{B,max} \\ & B_e\left[v_B[k], k\right] = 0 \\ & B_p\left[v_B[k], k\right] = 0 \\ & P_s[k] \neq 0 \\ & v_B[k] \in [v_{B,min}, v_{B,max}] \end{aligned} \tag{9}$$

### C. Constrained Markov Decision Process

We use a Constrained Markov Decision Process (CMDP) to model the constrained optimization problem (9) as a sequential decision making problem with discrete time steps, $k = 0, 1, .. \in \mathbb{Z}$ and time step length, $\Delta k$. We adopt this mathematical framework, because the robotic packaging machine with task scheduler and variable product inflow form a sequential, stochastic and dynamic environment. A CMDP is defined as a 6-tuple $< S, A, T, R, \gamma, C >$, where $S$ is the set of possible states and $A$ the set of possible actions. $T$ is the transition probability function defined as $T : S \times A \times S \rightarrow [0, 1]$ and denoted by $T(s_k, a_k, s_{k+1})$, it represents the transitioning to state $s_{k+1} \in S$ given state $s_k \in S$ and action $a_k \in A$. $R$ is the reward function defined as $R : S \times A \times S \rightarrow \mathbb{R}$ and denoted by $R(s_k, a_k, s_{k+1})$ representing the reward for a state transition. $\gamma \in [0, 1]$ is the discount factor which trades off immediate reward and delayed reward. The last term, $C$, is the cost function defined as $C : S \times A \times S \rightarrow \mathbb{R}$ and denoted by $C(s_k, a_k, s_{k+1})$ representing the constraints. [24] [25] The objective of the learning agent is to maximize cumulative reward while satisfying the constraints by learning the optimal control policy, $\pi*$, which is a mapping from states $s_k \in S$ to actions $a_k \in A$. This results in the following maximization problem [10]:

$$\begin{aligned} \max_{\theta} J_R^{\pi_\theta} = \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} & \left[ \sum_{k=1}^{N} \gamma^k R\left(s_k, a_k, s_{k+1}\right) \right] \\ \text{s.t.} \quad & J_{C_i}^{\pi_\theta} \leq \epsilon_i \\ & C_i\left(s_k, a_k, s_{k+1}\right) \leq \omega_i \end{aligned} \tag{10}$$

in which in which $J_R^{\pi_\theta}$ is the objective function as function of the reward $R$ when following policy $\pi_\theta$ parameterized by $\theta$. It is important to emphasize that while (9) is a minimization problem, (10) is a maximization problem. To address this discrepancy, we leverage the property that $min$ can be expressed as $-max$, which results in a reward $R$ that is equal to negative number of lost products, denoted as $-P_l[v_B[k], k]$. Furthermore, in (10) we divide the constraints in cumulative constraints, $J_{C_i}^{\pi_\theta} \leq \epsilon_i$, which need to be satisfied from the beginning to the current time step and instantaneous constraints,

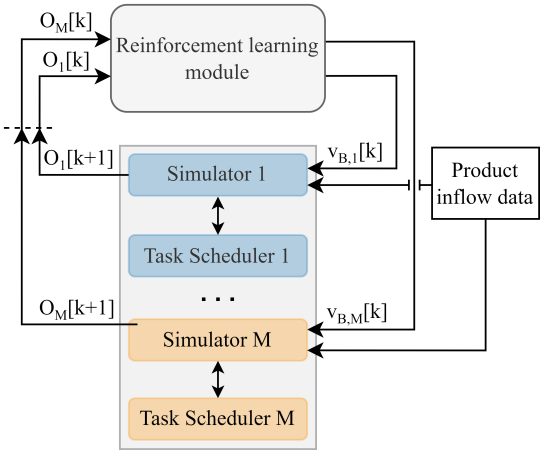



Fig. 5: High-level interaction framework between reinforcement learning module and the M parallelized simulators and task schedulers.

$C_i\left(s_k, a_k, s_{k+1}\right) \leq \omega_i$, which need to be satisfied in each step [10]. Where $i$ is the total number of constraints and $\epsilon_i$ and $\omega_i$ the corresponding threshold values. The cumulative constraint represents the placed products constraint (5a) and the instantaneous constraint is used to model the remaining constraints of optimization problem (9).

### D. High-level Interaction Framework

To optimize the box belt speed under varying product inflow, we introduce the high-level interaction framework that is modeled as shown in Fig. 5. We use multiple *parallelized simulators* of the robotic packaging machine, which simulate e.g., system delays and product inflow, to speed up the training process of the learning agent. Each simulator communicates with a separate task scheduler. The box belt speeds proposed by the RL module together with the product inflow data form the inputs to the $M$ parallelized simulators. For each time step $k$, $M$ box belt speeds $v_B$ are proposed corresponding to the $M$ observations $O$, obtained from the simulators.

## IV. Methodology

In order to solve the CMDP (10) with RL, real-world data and the high-level interaction framework, multiple important design decisions need to be made. To successfully learn a RL policy with accurate inference in the packaging machine's complex control scheme, which has many inter-dependencies, we introduce a framework that takes into account penalty functions, control delays, observation matching, state representation design, sparse and delayed rewards, smooth control and the limited availability of real-world data. In the following subsections, we provide details regarding our approach for addressing each one of these challenges.

### A. Markov Decision Process and Penalty Functions

The first challenge is to convert the CMDP (10) for the belt speed optimization problem (9) into a unconstrained Markov Decision Process (MDP). First, we simplify the problem by removing constraint (5a) regarding the maximum number of lost products. This constraint is cumulative and forms the lower bound of the objective function (4a). Therefore, we state that if a global optimum exists for optimization problem (9), it is equal to the global optimum of the same optimization problem without this cumulative constraint. Therefore, we can neglect this constraint, which also makes constraint (5b) redundant. Second, constraints (6), (7) and (8) are instantaneous hard constraints which can be converted into instantaneous soft constraints, which encourage but not guarantee constraint satisfaction, with use of penalty functions [8] [26]. The objective function of the unconstrained MDP becomes:

$$\begin{aligned}\mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{k=1}^{N}\gamma^k\Big(&R(s_k,a_k,s_{k+1})\\&-\sum_{i=1}^{l}\mu_i\left|C_i\left(s_k,a_k,s_{k+1}\right)-\omega_i\right|^q\\&-\sum_{i=l+1}^{l+m}\mu_i[max(0,C_i(s_k,a_k,s_{k+1})-\omega_i)]^q\Big)\Bigg],\\&\text{where } q\geq 1\end{aligned} \tag{11}$$

in which $\mu_i$ represents the weight for each constraint violation, $l$ the number of equality constraints, $m$ the number of inequality constraints and $N$ the operational time of the robotic packaging machine.

### B. Action Delay and Observation Matching

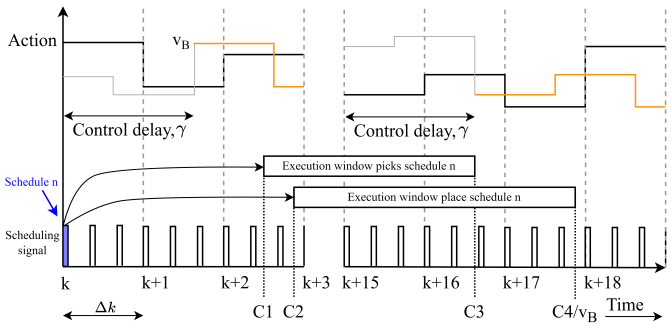
System delays and policy inference are fundamental challenges of real-world RL. The existence of system delays, which is almost always the case in real robotic systems, degrades the performance of RL methods due to the increased complexity of the credit assignment [27]. Delays in RL can be divided in three categories: action, observation and reward delays [7]. This subsection covers the first category. For the robotic packaging machine two types of action delays exist to which we refer as *control delay* and *planned delay*. Control delay is the time interval between action selection by the learning agent and the actual execution of that action by the packaging machine. Such delays negatively influence both the learning process and final solution [25]. In contrast, planned delay is introduced by us to enable proper policy inference in the packaging machine's control scheme. Before discussing how those action delays are defined and how they can be accounted for, the action space must be introduced. We use a continuous, normalized and symmetric action space, $A$, on the interval $[-1, 1]$, which represents the box belt speed $v_B$. Subsequently, we apply one-dimensional linear interpolation to scale the actions to the range $[v_{B,min}, v_{B,max}]$ to comply with the machine's allowed control inputs.

Fig. 6a shows the action signal, box belt speed $v_B$, and the control delay $\gamma$ i.e., the time between action signal and the resulting change in box belt speed. Moreover, the *scheduling signal* of the task scheduler is displayed. Multiple schedule cycles are performed during a single RL time step ($\Delta k$). Each

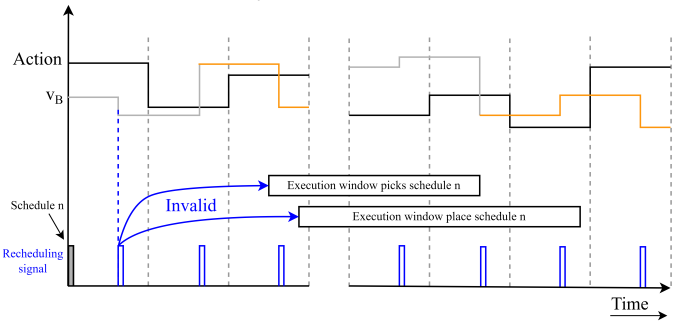
schedule cycle, triggered by the scheduling signal, produces a single schedule containing five products, two robots and a box, as defined in Fig. 3. In blue, a schedule $n$ is displayed with its *pick and place execution windows*. The pick execution window is defined as the time interval between the moment a product enters the workspace of the first robot, denoted as $C1$, and the moment a product leaves the workspace of the last robot, denoted as $C3$. This time interval represents the boundaries within which all products in schedule $n$ are picked up by the two delta robots. Additionally, the place execution window is defined as the time interval between the earliest possible time the first layer of products of schedule $n$ can be placed into a box, denoted as $C2$, and the latest possible time the second layer of products of schedule $n$ can be placed in a box, denoted as $C4/v_B$. This time interval represents the boundaries within which each of the two layers of products of schedule $n$ is placed by the corresponding delta robot in a single operation. $C1$, $C2$ and $C3$ are constant (future) time values relative to the creation time of the corresponding schedule $n$, because they only depend (in)directly on the constant product belt speed $v_P$ and the fixed configuration of the machine. $C4$ denotes the the latest possible box position for which the second layer of products of schedule $n$ can be placed. To make $C4$ time based, it is divided by the variable box belt speed $v_B$. With these definitions the planned delay can be introduced.

Fig. 6b illustrates the *rescheduling problem* of the robotic packaging machine, where schedule $n$ is generated under the assumption of a constant box belt speed, as future changes in speed are not yet known. Therefore, the first speed change after schedule $n$ makes schedule $n$ invalid. A *reschedule*, in which all the schedules are cleared and scheduled again that were computed with the incorrect speed, is necessary to create new schedules with the updated speed, as denoted by the *rescheduling signal*. This results in rescheduling of all schedules created between each speed change which makes regular scheduling redundant. Consequently, for small time steps $\Delta k$, no single schedule created in the horizon can be executed in the future, as each schedule is based on an inaccurate future speed. The task scheduler will try to schedule the products that already have left the horizon in order to limit the damage, however this will not save many products. Therefore, a constant box belt speed is deemed the optimal solution. This rescheduling problem emphasizes the importance of right timing and frequency of policy inference. By adding an additional delay, also called planned delay, between the action signal and the execution of that action this problem is solved.

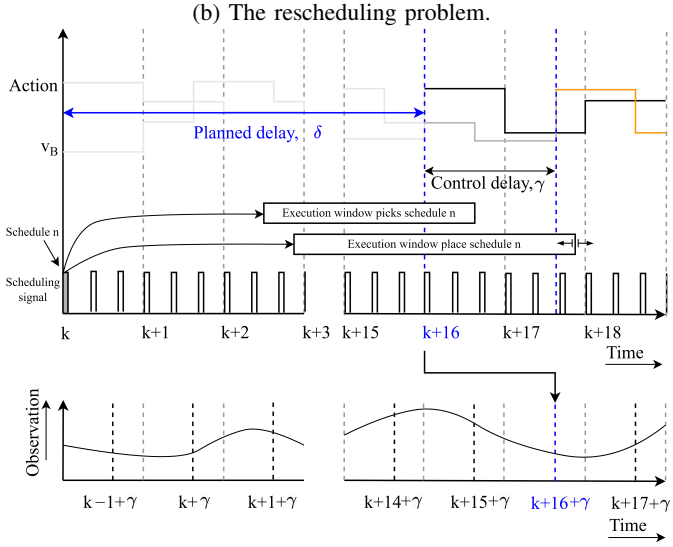
In Fig. 6c the planned delay $\delta$ is added to the control delay such that the execution of the action takes place after the execution window of all picks of schedule $n$. Now, the speed profile of the box belt is known during normal scheduling and rescheduling is not necessary anymore. As shown in Fig. 6c, the placement of products in a box can still happen after the combined control and planned delay; however, this will only happen for low product inflow. As stated earlier, the length of the execution window for a place is not fixed, but dependent




(a) Control delay and schedule execution window.




(b) The rescheduling problem.




(c) Shift action to $k+\delta$ and match observation by shifting to $k+\delta+\gamma$.

Fig. 6: Overview of the control delay, planned delay and observation matching, in which C1 = (Start workspace R1 - horizon end) / $v_P$, C2 = C1 + earliest finish time pick last product($v_P$), C3 = (End workspace R4 - product detection) / $v_P$, C4 = End workspace R4 - box detection and $\Delta k = 0.75s$.

on the box belt speed $v_B$. For low product inflow the box belt speed must be low in order to minimize the number of unfilled boxes, which increases the length of place execution window. In contrast, the constant product belt speed is relatively high, therefore, the Delta robots must hold the products for a while before placing in a box. Hence, the pick part of the schedules remain valid and the place part of the schedule is updated according to the changing box belt speed until a box enters

the workspace of the corresponding Delta robot.

The combined delays create a time mismatch between action execution and received observation. Therefore, the observation at time $k+\delta+\gamma$ is fed back to the agent at time $k$, which is shown in Fig. 6c. This future observation matching method is valid, because the scheduling of the products in the horizon at time $k$ takes place at $k$; therefore, all the information of a schedule created at time, $k$, i.e., the future start and end times, can be used to construct the observation at time $k+\delta+\gamma$.

### C. State Representation Design

To design a state that captures all the relevant information for selecting an appropriate action, multiple experiments were performed in this research regarding feature selection and history, see Appendix G. The final features are:

- Current box belt speed, $v_B$ [m/s],
- Previous action [m/s],
- Product inflow lane 1 measured at product detection [products/min],
- Product inflow lane 2 measured at product detection [products/min],
- Distance of closest non-scheduled empty box relative to the checkout point, $x_{box}$ [m],
- Distance of closest non-scheduled product relative to checkout point, $x_{prod1}$ [m],
- Distance of second closest non-scheduled product relative to checkout point, $x_{prod2}$ [m].

Where each feature is represented by a continuous normalized and symmetric feature space on the interval $[-1, 1]$ obtained by scaling with one-dimensional linear interpolation with respect to each feature's minimum and maximum observed numerical value.

However, the complexity of the environment with lots of inter-dependencies and limited amount of information available (the features) makes the problem inherently partially observable and, therefore, the state non-Markov [28]. For example, no information is available about the product feeders' performance which appears as non-stationarity in the state or about the division of tasks among robots which appears as stochasticity. To handle the partial observability, 30 time steps ($\Delta k$=0.75s) of history is added to each feature, which captures a complete throughput of products from detection to checkout (24 time steps) including margin. By incorporating the added history per feature, the features can effectively describe the final state at $k+\delta+\gamma$. The resulting high-dimensional continuous state space and earlier defined one-dimensional continuous action space increase the complexity of the problem, which is also known as *the curse of dimensionality* [29]. We tackle this by using Neural Networks (NN) for function approximation in the RL agent, which have obtained well-performing results when dealing with high-dimensional spaces [30].

In addition, the observation matching introduced in IV-B, makes the interpretation of the features $x_{box}$, $x_{prod1}$ and $x_{prod2}$ a bit more complex. When the observation is shifted beyond the execution window of schedules (Fig. 6c), the box and product features no longer represent the actual distances

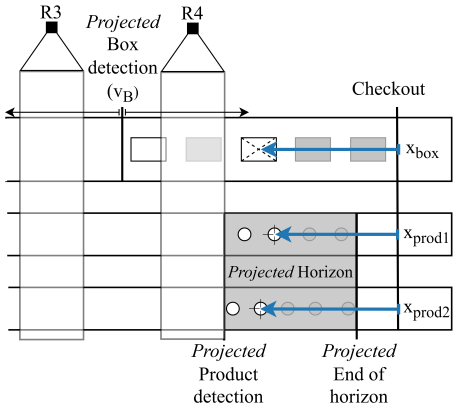



Fig. 7: Schematic top view of robotic packaging machine with projection of horizon and the three state features $x_{box}$, $x_{prod1}$ and $x_{prod2}$.

from the checkout point, but serve as future projections. This comes down to the situation shown in Fig. 7, in which the horizon is projected right after the workspace of robot four ($v_P \cdot (\delta+\gamma)$) and the box detection point is projected with a variable position of $v_B \cdot (\delta+\gamma)$. The positions $x_{box}$, $x_{prod1}$ and $x_{prod2}$ are decoupled from their real positions on the belts, but give the relative position to the horizon a product and box should have, in order to be part of a schedule. The projected end of horizon is not aligned with the checkout point to prevent false positive empty boxes or lost products. For example, in edge case scenarios in which a product that just has left the horizon, could still be part of a schedule.

### D. Sparse Delayed Rewards and Smooth Control

With the previously discussed action delays, observation matching method and state features, one simplification and two potential problems can already be uncovered for the objective function stated in (11).

To simplify the objective function, we eliminate the penalty function associated with the constraint on partly filled boxes (7). As discussed in Section IV-B, the introduction of planned delay eliminates the need for rescheduling and, by definition, ensures compliance with the constraint on partly filled boxes. Specifically, the task scheduler prioritizes the creation of complete schedules, as defined in Fig. 3, thereby precluding the intentional departure of partly filled boxes from the machine. Without planned delay, variations in the box belt speed during schedule execution could lead to the unintentional departure of partly filled boxes. With this simplification the combined reward and constraint function for (11) with $q=1$ becomes:

$$\begin{aligned} r = &- \mu_{prod} \cdot P_l[v_B[k], k] - \mu_{box} \cdot B_e[v_B[k], k], \\ &- \mu_{acc}[max(0, \sqrt{(v_B[k] - v_B[k-1])^2} - a_{B,max} \cdot \Delta k)], \end{aligned} \tag{12}$$

where $\mu_{prod}$ denotes the weight for the number of lost products (4a), $\mu_{box}$ denotes the weight for the constraint on empty

boxes (6) and $\mu_{acc}$ denotes the weight for the constraint on acceleration (8).

The first problem concerns the agent's actions. Since the agent's actions are continuous, it has the ability to control the box belt speed with high precision. However, this could lead to the emergence of multiple (weak) global optima and an increase in the number of weak local optima; small speed changes that do not impact the return, but do negatively affect the box belt's maintenance. While smooth control is not a strict requirement, it is preferable for stable training, real-world applicability, and enhanced interpretability of the resulting policy [7]. Therefore, we remove inequality constraint (8) from the reward function (12) and add penalty function (13) that penalizes speed changes with an appropriate small amount $\zeta$, this way smooth control is encouraged.

$$p(v_B) = -\zeta \sqrt{(v_B[k] - v_B[k-1])^2} \tag{13}$$

The second problem concerns the objective function. It consists of a linear combination of delayed sparse and non-delayed dense rewards, which arise respectively from the implicit constraints governing the number of lost products and empty boxes, and the penalty function for smooth control encouragement. As noted in [31], effectively learning policies for sparse objectives poses a fundamental challenge in deep RL. Moreover, the existence of non-delayed dense reward could increase the risk of the agent getting stuck in a local maximum generated by the non-delayed dense reward which is easier to learn [31]. To prevent this, we propose the following exponential function which converts the sparse rewards to dense ones:

$$\begin{aligned} f(x) &= \sum_{i \in I} -\mu_i \frac{1}{e^{x_i[k+1]}}, \\ \text{where } I &= \{box, prod1, prod2\}, \\ \mu_{prod} &= \mu_{prod1} = \mu_{prod2} \\ x_{prod1}, x_{prod1} &\in [0, v_P \cdot (\delta + \gamma)], \\ x_{box} &\in [0, v_B \cdot (\delta + \gamma)], \end{aligned} \tag{14}$$

where the three state features $x_{box}$, $x_{prod1}$ and $x_{prod2}$ are defined in Fig. 7 and the $\mu_i$ represents the weight for each exponential function. These weights are the same as in (12), such that the original sparse reward is smoothed out exponentially along the belts. The aim of this additional reward is to guide the exploration behaviour of the agent with our heuristic domain knowledge away from constraint violations, also known as reward shaping [32]. The total reward, $r$, obtained at each time step is defined as:

$$\begin{aligned} r = f(x) + p(v_B) - \mu_{box} \quad & \cdot B_e[v_B[k], k] \\ - \mu_{prod} \quad & \cdot P_l[v_B[k], k], \end{aligned} \tag{15}$$

where $B_e$ is the number of lost empty boxes and $P_l$ the number of lost products. It is worth mentioning that certain components of the term $f(x)$ disappear when a product or an empty box exits the machine. For instance, when an empty box leaves the machine, the task scheduler stops monitoring it. As a result, $B_e$ becomes one, and the exponential function in $f(x)$ for this box becomes zero. As a consequence, the magnitude of the original sparse reward remains unchanged when the constraints are violated.

### *E. Robustness in Realistic Scenarios*

Data-efficiency is a core challenge for RL in real-world systems as most of those systems are expensive to operate, relatively slow-moving or fragile [7]. Resulting in a training dataset that does not capture a large part of the state and action space. This challenge is also applicable to the robotic packaging machine, as it is customized for a specific customer located in a different part of the world, making it impractical to directly test our framework on this machine. Consequently, online learning on the physical machine is not feasible. However, we do have access to a limited amount of prerecorded data from the robotic packaging machine. This data comprises information such as product and box detection times, as well as the start and end times of schedules, but after filtering out the noise (Appendix A), it only amounts to 75.5 minutes of operational time, insufficient for offline learning. To address these limitations, we employ parallelized simulators of the machine, as described in Section 5, and replay the real product detection times from the prerecorded data during simulation. This enables us to speed up training with the simulators significantly and utilize actual varying product inflow data, which plays a crucial role in the belt speed optimization problem. However, also for online learning with a simulator, the available amount of prerecorded data is not sufficient. Therefore, we use self-generated simulated product inflow data for training and validate the learned policy on the small set of prerecorded real-world data.

In order to prevent *covariate shift*, a common problem in machine learning [33], between the product inflow distribution of the self-generated training data and the real-world validation data, we use *scenario randomization*. Here, a *scenario* is defined as the profile of the product inflow during one episode, which is a single run of the robotic packaging machine lasting $N$ seconds (9). Although the machine operates continuously in the real world, we can consider it episodic, as the same situations repeat over and over again. By highly randomizing the simulated product inflow in each scenario on a realistic range during training, the agent is exposed to a sufficiently large part of all possible environment states [34]. This improves generalization of the learned policy to *realistic scenarios* i.e., scenario's from the real-world, and increases its robustness against variations in product inflow within those scenarios. The scenarios are designed for normal operation of the robotic packaging machine, which corresponds to a product inflow per lane between 120 and 135 products per minute according to the company's standards. With this method we aim to bridge the sim-to-real gap [35] related to the variability in product inflow data, which is a key issue when optimizing the belt speed. By replaying the prerecorded real-world data in simulation, we can validate the effectiveness of the scenario randomization method.

### F. Medium-Level Interaction Framework

In order to train, tune, evaluate and validate the RL solution for the belt speed optimization problem, we propose the medium-level interaction framework shown in Fig. 8. This framework shows the inside of the RL module of the high-level interaction framework (Fig. 5). The medium-level interaction framework starts with the definition of the simulator, experiment, Optuna and goal settings. The goal settings determine if the RL module is used for tuning of hyperparameters with Optuna, training the RL agent, evaluating the learned policy on the randomized scenarios or validating the policy on the prerecorded real-world data. Tuning is a time-intensive task, because a complete training (when not pruned) must be executed for each sampled set of hyperparameters (i.e., a trial). By multiprocessing the training, the training time is highly reduced. The lower-level modules, represented by the blue blocks, all have their own low-level interaction framework with pseudo code in Appendix C. The goal of each module is characterized as follows:

- **Reinforcement learning module:** Starts a hyperparameter tuning, RL training, policy evaluation with randomized scenarios or policy validation with real-world data study.
- **Information writer module:** Saves all experiment settings and compute final metrics and constraint violations.
- **Experiment design module:** Selects a state, action and reward function depending on experiment.
- **Custom gym environment module:** Provides the training environment.
- **Socket communication module:** Communicates with the simulator of the packaging machine.
- **Callback module:** Provides callbacks for best training, evaluation of a tuning trial, visualization and saving hyperparameters.

Besides, hyperparameter tuning, RL training and policy validation, the developed interaction framework is designed to facilitate easy modifications to each experiment, such as selecting multiple model-free reinforcement learning methods or incorporating new combinations of state, actions, and reward functions.

## V. Experimental Results

In this section, we first describe the experimental setup, followed by the ablation study conducted with our randomized scenarios. Finally, we present the experimental validation of the learned policies, computed with our proposed method and interaction framework, on real-world product inflow data and compare its results with the heuristic-based engineered baseline solution.

### A. Experimental Setup

The results of this research are conducted with Proximal Policy Optimization (PPO) due to its combination of sample efficiency and performance [36]. Furthermore, inspired by the work of [37], [38] and [39] we use a combination of the adaptive learning rate optimizer called Adam and a linear decaying learning rate schedule. Additionally, a hyperparameter study using Optuna is conducted to optimize training performance. The tuned hyperparameters and the findings of this study are presented in in Appendix F.

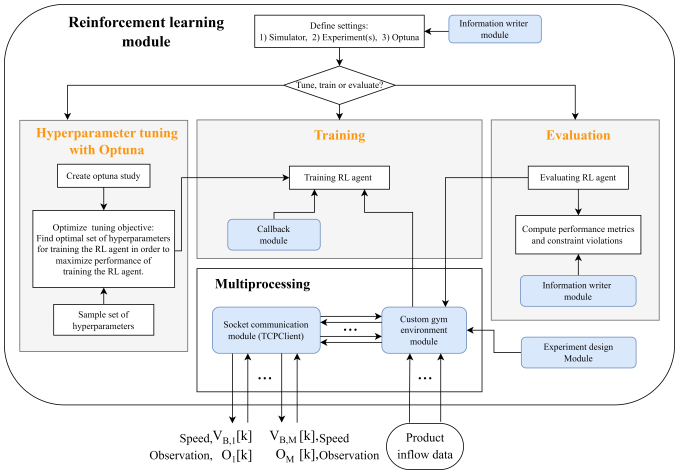



Fig. 8: Medium-level interaction framework of RL module shown in Fig. 5

### B. Ablation Study

To test the effectiveness of the proposed methodology outlined in Section IV, we conduct an ablation study that compares three policies trained using different versions of the reward function. Specifically, this study investigates the impact of sparsity in rewards and the encouragement of smooth control within our model. The ablation study is performed using our randomized scenarios, as defined in Section IV-E, on the following three cases.

- **Case 1: Sparse rewards**
  In this case, we use the reward function as defined in (12). It can be described as sparse, because the corresponding rewards are computed from the number of lost products, denoted by $P_l[v_B]$, the number of empty boxes exiting the machine, denoted by $B_e[v_B]$, and the maximum allowed acceleration, denoted by $a_{B,max}$. As the learning agent improves its performance, the corresponding rewards become sparser, making it more challenging to assign credit due to the need to connect long sequences of speed changes to their respective future rewards.

- **Case 2: Sparse rewards and smooth control**
  In this case, the reward function is slightly changed to encourage smooth control. By setting the maximum acceleration, $a_{B,max}$, in (12) to zero, every speed change is penalized. The reward function becomes:

$$r = -\mu_{prod} \cdot P_l[v_B[k], k] - \mu_{box} \cdot B_e[v_B[k], k], \\ + p(v_B). \tag{16}$$

  with $p(v_B)$ defined in (13).

- **Case 3: Dense rewards and smooth control**
  In this case, the reward function is the same as in Case 2, plus the exponential function $f(x)$ (14), which guides the agent away form constraint violations. The reward function becomes:

$$r = -\mu_{prod} \cdot P_l[v_B[k], k] - \mu_{box} \cdot B_e[v_B[k], k], \\ + p(v_B) + f(x). \tag{17}$$

Table I presents the results of the ablation study for the three cases. We use the mean and standard deviation of several indicators as metric for comparison. First of all, two indexes of the OEE industry standard metric i.e., performance (2) and quality (3) are used. Performance reflects the quantity of supplied products that the machine successfully processes by packing them into boxes. Quality represents the quality of the outgoing boxes in terms of filling rate. According to the requirements stated in Section III-A performance must be greater than or equal to $99.8\%$ and quality must be equal to $100\%$ i.e., no partially filled or empty box may leave the machine. In addition, we include three capacity metrics to provide further context on the machine's productivity i.e., number of lost products, supplied products and packed boxes. Next, two control input metrics are used: the mean box belt acceleration, which indicates the smoothness of the control input, and the scaled code execution time, which represents the time required for computing the control input expressed in code execution time per simulated second of the machine's operation. In order to prevent any bias introduced by the different complexities of the problem during a simulation, the code execution time is defined as the time between start and end of the simulation divided by the simulated operating time of the machine. Last, the three cases are compared on their degree of constraint satisfaction regarding performance, the number of empty and partly boxes leaving the machine and maximum box belt acceleration.

For each case the corresponding policy is trained on 6827 simulations of 1800 simulated seconds, equivalent to approximately 142 days of machine operation, using the randomized scenarios as explained in Section IV-E. The training time for each learned policy is approximately 9 hours, when using maximum parallelized simulations on a Intel Core 11th Generation i7-11850H Processor. After training, each policy is evaluated on 48 simulations of 1800 simulated seconds and a step size of 0.75s, equivalent to a 24-hour machine operation, to determine the means and standard deviations presented in Table I. Although the time step for controlling the belt speed can be considered relatively large, our experiments demonstrated that this step size yielded good performance, and decreasing the step size to increase the policy inference frequency would only result in longer simulation calculation times.

From Table I we can conclude that the mean performance is comparable and above the required performance for each case, according to the company's standard. The quality of the output is equal to 100 %, which means that all boxes are packed with the required number of products. As a result, all three cases obtain full constraint satisfaction in each simulation with the simulated randomized scenarios for the performance, empty box and partially filled box constraint in Table I. Analysis of the capacity metrics reveals that Case 3 exhibits a slightly higher mean number of packed boxes and supplied products, with slightly lower lost products compared to Case 2. However, the difference between the two cases is negligible, and thus, not reflected in the mean performance metric. Additionally, the high standard deviations of the capacity indicators in relation to their means limit their interpretability.

Furthermore, the mean box belt acceleration is the lowest for Case 2, with smooth control encouragement by use of the penalty function, and the highest for Case 1, without smooth control encouragement. Case 3, with the addition of an exponential function to convert sparse rewards to dense ones, falls in between. The low standard deviations for this indicator in relation to their means, suggests that case 2 increases smooth control the most. Additionally, all cases satisfy the maximum acceleration constraint, for each simulated scenario. Moreover, the second control input metric i.e., the code execution time, shows an increase for Case 2 with respect to cases 1 and 3. Given the low standard deviations of this indicator in relation to their means, it is likely that the reward function for Case 2 is slightly more difficult than the other cases. In addition, code execution time is highly influenced by the underlying hardware while evaluating; therefore, the metric is meant for comparison only.

Besides, encouraging smooth control with use of a penalty

TABLE I: Comparison of policies trained with sparse rewards, dense rewards and smooth control encouragement on simulated randomized inflow data

| | **Metrics:** *Mean (Std.)* | **Case** *1: Sparse reward* | *2: Sparse reward & smooth control* | *3: Dense reward & smooth control* |
|---|---|---|---|---|
| Productivity | *Performance*[a] *[%]* | 99.97 (0.019) | 99.96 (0.028) | 99.96 (0.022) |
| | *Quality*[b] *[%]* | 100.00 (0.00) | 100.00 (0.00) | 100.00 (0.00) |
| Capacity | *Lost products* | 2.46 (1.34) | 2.88 (1.99) | 2.52 (1.65) |
| | *Supplied products* | 7126.79 (34.20) | 7127.22 (46.54) | 7135.71 (44.16) |
| | *Packed boxes* | 1424.33 (6.67) | 1424.69 (9.09) | 1426.42 (8.74) |
| Control input | *Mean* $a_B^c[m/s^2]$ | $23.27 \cdot 10^{-3}$ ($0.79 \cdot 10^{-3}$) | $2.48 \cdot 10^{-3}$ ($0.11 \cdot 10^{-3}$) | $8.59 \cdot 10^{-3}$ ($0.48 \cdot 10^{-3}$) |
| | *Code execution time*[d] *[ms/s]* | 11.11 (0.09) | 13.53 (0.50) | 11.36 (0.42) |
| Constraints[e] | *Performance* $\geq$*99.8%* | ✓ | ✓ | ✓ |
| | *Empty boxes = 0* | ✓ | ✓ | ✓ |
| | *Partly filled boxes = 0* | ✓ | ✓ | ✓ |
| | $a_B^c \leq a_{B,max}$ | ✓ | ✓ | ✓ |

[a] (Number of packed products / total supplied products)*100%.

[b] (Number of packed boxes / total supplied boxes)*100%

[c] Box belt acceleration

[d] Code execution time between start and end of single simulation [ms] / simulated operating time of machine [s].

[e] If zero constraint violation is obtained for each of the 48 simulations.

function, other experiments were performed to obtain smooth policies. Appendix E discusses the effects of a low pass Butterworth filter.

Given that the ablation study results do not show any improvement in performance or quality for Case 3 compared to Case 2, and considering that Case 2 exhibits the lowest mean acceleration, we focus on validating only Cases 1 and 2 on real-world data in the following section.

### C. Validation with Real-World Data

In this section, we present the validation of the learned policies from cases 1 and 2 in realistic scenarios by using real-world product inflow data. We compare the RL solutions with the company's *heuristic-based engineered solution*. As outlined in Section IV-E, the availability of real-world product inflow data is limited. Therefore, we perform training using the simulated randomized scenarios, as discussed in the ablation study, and validate performance of the learned policies in realistic scenarios leveraging real-world data.

We employ the same metrics as in the ablation study and refer to Section V-B for the definitions of cases 1 and 2. The baseline solution is defined as follows:

- **Heuristic-Based Engineered baseline:**
  The company's current robotic packaging machine uses a heuristic-based engineered method to regulate the box belt's speed. This method has been employed for several years, making it a suitable benchmark for this research. A notable benefit of this baseline is its ability to use additional rescheduling during operation, as explained in section IV-B. In contrast, the RL solution does not possess this capability. Consequently, our proposed method has a different and potentially smaller solution space than the baseline.

Table II presents the results of the experimental validation of the learned policies simulated with our proposed method using real-world product inflow data. The metrics are computed using seven realistic scenarios of variable machine operating time containing 75.5 minutes of representative real-world data. On all metrics, except the mean box belt acceleration, both RL solutions show similar results. Both solutions have zero constraint violations, for each simulation with real-world scenarios, regarding performance, partly filled boxes, empty boxes and maximum box belt acceleration. However, the mean box belt acceleration has strongly decreased by encouraging smooth control. This effect was seen in the ablation study as well. Fig. 9b shows the speed profile with (orange line, Case 2) and without (gray line, Case 1) smooth control encouragement for the product inflow data shown in Fig. 9a. In the marked regions 1, 2 and 3, the learned policy slowly decreases the box belt speed for drops in product inflow.

Since the smoothness of the second policy (Case 2) makes it more appropriate for real-world scenarios, we continue the analysis comparing the heuristic-based engineered baseline with this RL solution only.

Contrary to the RL solution, the baseline violates the constraint on performance and maximum box belt acceleration

TABLE II: Validation of two policies trained with reinforcement learning and the heuristic-based engineered baseline solution using real-world inflow data

| | **Metrics:** *Mean (Std.)* | **Case** *1: RL with sparse reward* | *2: RL with sparse reward, smooth control* | *Heuristic baseline* | *Case 2 vs. Baseline*[f] |
|---|---|---|---|---|---|
| Productivity | *Performance*[a] *[%]* | 99.94 (0.030) | 99.94 (0.077) | 99.31 (0.22) | +0.63% |
| | *Quality*[b] *[%]* | 100.00 (0.00) | 100.00 (0.00) | 100.00 (0.00) | - |
| Capacity | *Lost products* | 1.86 (1.73) | 1.28 (1.58) | 19.00 (15.64) | -93.26% |
| | *Supplied products* | 2792.71 (2039.84) | 2792.71 (2039.84) | 2792.71 (2039.84) | - |
| | *Packed boxes* | 558.14 (407.75) | 558.29 (407.91) | 553.29 (403.55) | - |
| Control input | *Mean* $a_B^c$ $[m/s^2]$ | $28.28 \cdot 10^{-3}$ $(0.85 \cdot 10^{-3})$ | $3.21 \cdot 10^{-3}$ $(0.25 \cdot 10^{-3})$ | $18.55 \cdot 10^{-3}$ $(1.07 \cdot 10^{-3})$ | -82.70% |
| | *Code execution*[d] *time [ms/s]* | 10.09 (0.45) | 9.92 (0.49) | 18.02 (0.60) | -55.05% |
| Constraints[e] | *Performance* $\geq 99.8\%$ | ✓ | ✓ | × | |
| | *Empty boxes = 0* | ✓ | ✓ | ✓ | |
| | *Partly filled boxes = 0* | ✓ | ✓ | ✓ | |
| | $a_B^c \leq a_{B,\max}$ | ✓ | ✓ | × | |

[a] (Number of packed products / total supplied products)*100%.

[b] (Number of packed boxes / total supplied boxes)*100%

[c] Box belt acceleration

[d] Code execution time between start and end of single simulation [ms] / simulated operating time of machine [s].

[e] If zero constraint violation is obtained for each of the seven simulations.

[f] Effectiveness of RL policy with smooth control (case 2) with respect to baseline (case 3).

in each simulation with real-world data, as presented in Table II. The former becomes clear in the metric of the mean performance, which is $0.49\%$ lower than required. The RL solution increases the mean performance by $0.63\%$ and decreases the mean number of lost products by $93.26\%$ with respect to the baseline solution. Note that the standard deviation for the latter is quite high, which is caused by the high variability in the machine's operating time in each recorded real-world scenario. The same holds for the supplied products and packed boxes. Those capacity metrics are displayed to provide context for the productivity metrics.

Furthermore, the mean box belt acceleration is decreased by $82.70\%$ with respect to the baseline solution, as presented in Table II. Fig. 9c shows the speed profile of the RL solution with smooth control (orange line, case 2) and the baseline solution (blue line) for the product inflow data shown in Fig. 9a. In the marked fragments 1, 2, 4, and 5 the box belt speed decreases drastically for the baseline solution, while the RL solution shows a much smaller and smoother decrease when product inflow drops. Moreover, those drastic speed fluctuations of the baseline are accompanied by an increase in product loss, which is denoted by the dashed lines (see markers 1 and 4). In total, 30 products are lost in the presented fragment for the baseline, and zero products for the RL solution. Even

for out-of-distribution product inflows ($< 120$) no products are lost for the RL solution (see marker 2 Fig. 9a). Furthermore, the RL solution provides not only speed changes with smaller amplitude compared to the baseline, but also with a lower frequency, which together smoothens control (see marker 5). Finally, we observe an intriguing distinction between the RL solution and the baseline; the latter accelerates to higher speeds than the former during the intervals between the sudden speed fluctuations.

In addition, the mean code execution time is decreased by $55.05\%$ compared to the baseline solution, as demonstrated in Table II. Given the low standard deviations of this indicator in relation to their means, it is likely that the RL solution computes its control inputs more efficiently than the baseline solution.

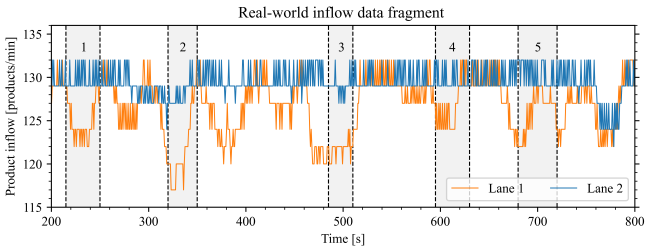
## VI. Discussion

First, we discuss the results of the ablation study, followed by the results of the validation study using real-world data. Finally, we present the limitations of our work and recommendations for future work.

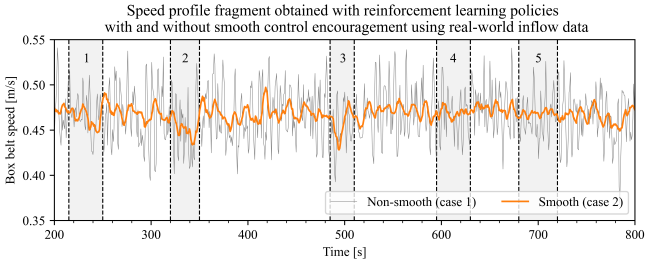
### A. Ablation study

From the results of the ablation study we conclude that with our proposed method, RL can be used to learn appropriate box belt speeds for the simulated robotic packaging machine using our randomized scenarios as introduced in Section IV-E, while having zero performance and quality constraint violations. Moreover, in line with our expectations penalizing speed changes with an appropriately small amount encourages smooth control without violating constraints. Although [7], [31], [32], [40] - [42] state that learning policies effectively for sparse and/ or delayed objectives is difficult for learning agents, there is no empirical evidence to suggest that the non-delayed dense rewards, enforced by the penalty function for smooth control, outperform delayed sparse rewards arising from the lost products and boxes.

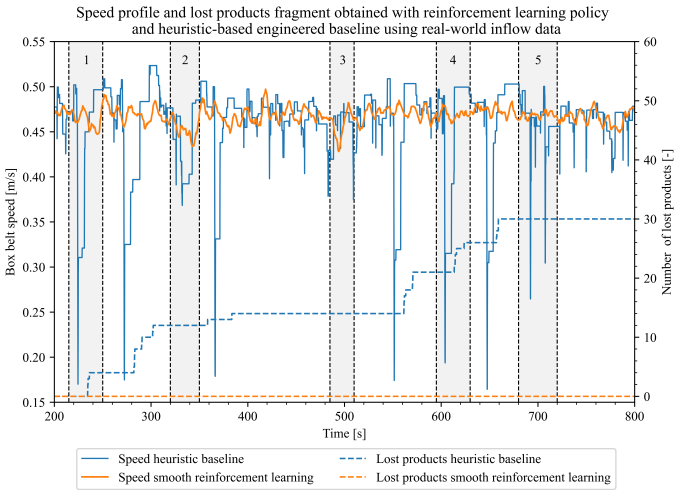
Furthermore, the conversion of the sparse rewards, imposed by the lost products and boxes (Case 2, Table I), into dense rewards (Case 3, Table I) has not improved the performance but does have degraded control smoothness. We potentially altered the optimality of the policies, which is a common problem in reward shaping [43] and explains the observed degradation. On the other hand, the code execution time has improved which suggests that guiding the agent away from constraint violations helps to decrease complexity. In addition, having an reward function with a weighted linear combination of sparse and dense rewards complicates the tuning of the weights for the system designer. We expected that learning from delayed sparse rewards when subjected to non-delayed dense rewards is challenging for the learning agent. However, the results of the ablation study show that the agent is able to improve control smoothness without reward shaping, while maintaining zero constraint violations and high performance. It should be noted that a slight increase in packed boxes is observed for Case 3.




(a)




(b)




(c)

Fig. 9: Fragment of real-world inflow validation data with corresponding speed plots and number of lost products for the three cases listed in Table II.

However, this can be explained by the stochasticity of the product supply: more supplied products result in more packed boxes, when assuming a performance of $100\%$. This limits the interpretation of these metrics. Moreover, the standard deviations of the lost products, supplied products and packed boxes indicate a strong overlap between the distributions of these indicators, therefore no clear difference can be observed.

### B. Validation study

The validation study conducted using real-world product inflow data shows that our proposed framework effectively learns appropriate box belt speeds for a simulated robotic packaging machine in realistic scenarios. This is achieved

without any violation of performance, quality, or acceleration constraints, as evident from Case 2 in Table II. We can, therefore, state that the proposed scenario randomization method discussed in Section IV-E successfully overcomes the covariate shift between the simulated scenarios used for training and the realistic scenarios used for validation. Furthermore, the effects seen in the ablation study are observed with real-world data as well: the RL agent learns to reduce the noise caused by unnecessary large speed changes while maintaining full constraint satisfaction and high performance (Case 2, Table II and Fig. 9b). We can, therefore, state that the smooth control encouragement using the penalty function (13), successfully damps the unnecessary oscillations.

Furthermore, the absence of these unnecessary oscillations, facilitates a more interpretable control input, which is a common challenge in real-world RL [7], and reduces unnecessary maintenance of the box belt. In [44], [45], and [46] a similar strategy for limiting control effort is used to avoid oscillatory control behavior when using continuous actions and PPO.

The improved interpretability of the learned policy allows us to validate our initial expectations. We had expected the policy to oscillate to some degree because lower product inflows necessitate a lower constant box inflow, leading to lower box belt speeds, and vice versa. As seen from the product inflow data presented in Fig. 9a, there are oscillations to some extent, and this is indeed reflected in the final learned policy. Specifically, the decrease in box belt speed observed for drops in product inflow, marked as regions 1, 2, and 3 in Fig. 9a and Fig. 9b, confirms this expectation. The RL solution even generalizes well to out-of-distribution product inflows (marked region 2 Fig. 9). Policies were trained for inflows between 120 and 135 products per minute per lane, while the filtered real-world data captured inflows between 115 and 140. In Appendix A, we discuss the filtering method used to remove the noise caused by machine startups and product inflow stops resulting in product inflows lower than 75 products per minute. These scenarios are not considered as normal operation of the machine according to the company's standards. While this filtering method effectively removes these edge case scenarios, it is important to note that such scenarios can occur in practical settings. Therefore, additional training with real-world data, when available, would be necessary in order to deal with such situations.

From Table II we conclude that the RL solution with smooth control (Case 2) increases performance by $0.63\%$, decreases the mean box belt acceleration by $82.70\%$ and decreases code execution time by $55.05\%$ while having zero constraint violations compared to the baseline solution. The baseline solution violates the constraints on performance and maximum box belt acceleration on each simulation using real-world scenarios. This makes the RL solution not only better performing, but also faster. Fig. 9c demonstrates the behaviour of the RL solution with smooth control compared to the baseline solution. In the marked fragments of Fig. 9b, the box belt speed decreases drastically for the baseline solution in order to save boxes from leaving partly or unfilled i.e., a critical situation, resulting in increased product loss which is denoted by the dashed line. This is caused by the *corrective behaviour* of the baseline solution; it adapts the box belt speed when a critical situation already occurs, resulting in inevitable product loss. In contrary, the RL solution shows a much smaller and smoother response to variations in product inflow. Due to its learned predictive behavior regarding those critical situations, it is able to smoothly adapt the speed with the use of product inflow data and the relative box and product locations on the conveyor belts, prior to the occurrence of critical situations. Therefore, we state that the RL solution prevents instead of corrects critical situations caused by the varying product inflow present in the real-world data. Furthermore, the baseline speeds up to higher values than the RL solution in between critical situations, which is in line with our conclusions; the RL solution keeps the speed lower also in between critical situations to prevent these from happening. This emphasizes the complexity of engineering stable and generalizable heuristics for such highly complex real-world systems.

### C. Limitations and recommendations

This research has uncovered some interesting industry-specific challenge of real-world RL. Control delays [25] and other types of system delays [7] which make the adoption of RL in the real-world challenging, have been widely explored in research during the past 10 years. All, with the aim of reducing the increased complexity of the credit assignment caused by those delays. However, when interfering our RL control with the rest of the machine's complex control scheme (e.g., task scheduler), we even introduced additional planned action delays to overcome a much bigger problem imposed by policy inference as explained in section IV-B. This emphasizes the importance of RL research in actual industrial applications. Although our proposed framework effectively addresses the issues caused by the combined control and planned delay with observation matching, the presence of these factors still poses limitations on our research. Specifically, because the agent's control actions must occur in the future (i.e., the planned delay) to maintain a fixed speed profile during schedule execution, any deviations between the agent's estimated future observations and their actual values cannot be corrected. Possible solutions are adding Gaussian noise to the observations as done in [47] or adapting the rescheduling procedure to allow external controlled speed changes in real-time. The latter solution can be applied with the currently proposed method, even with a single retraining iteration, because the learning agent is not aware that its actions and matching observations occur in the future.

Moreover, the proposed methodology obtains zero constraint violations by encouraging constraint satisfaction. Nevertheless, no formal guarantees can be given. It is worth noting that the company's current heuristic-based solution for regulating the box belt speed does not provide formal guarantees either.

Additionally, this research is conducted with the use of a simulator and prerecorded real-world product inflow data. The real-world data only partially bridges the sim-to-real gap [5] by offering replayed realistic scenarios in simulation. By addressing other aspects of the sim-to-real gap, such as system delay variability or issues with executing schedules due to communication problems with the Delta robots, the policies learned in simulation can be transferred to the physical robotic packaging machine. Domain randomization is a common approach to deal with such variability between simulation and reality. By providing enough variation in the simulator's parameters the real world appears as just another variant [35]. The aim is to enable the learning agent to generalize to real-world data, while fully trained in simulation. Other approaches for bridging this gap are meta-learning, domain adaptation, knowledge distillation, imitation learning and robust RL [5].

Although, successful training is obtained with scenario randomization to prevent covariate shift as explained in Section IV-E. We recommend gathering more real-world product inflow data in order to train with a combination of scenario randomization and real-world data, as done in [48]. Novel situations that are hard to capture with scenario randomization solely for reasonable training times, such as product inflow stops or machine startup issues, are then added to the training data set. On the other hand, collecting enough real data from the machine takes time. In fact, the learned policies were trained on approximately 142 days of machine operating time. Therefore, scenario randomization can be used on top to further increase generalisation for quick adoption of the proposed data-based predictive RL solution. In addition, randomized perturbations can be added to the training data to increase robustness of the policies [49] [50].

As discussed earlier, the results of the ablation study suggest the learning agent did not benefit from the additional exponential function $f(x)$ (Case 3, Table I), which guides the agent away from constraint violations. Possible explanations are: the exponential function alters the optimality of the original MDP or the products and boxes are scheduled sooner, but this does not affect the performance of the machine. The former can be solved by using potential-based reward shaping [43], [32]. The potential-based reward function can be of the form:

$$F\left(x\right) = \gamma f\left(x[k+1]\right) - f(x[k]), \tag{18}$$

with $f(x)$ defined in (14).

Furthermore, by optimising the underlying hardware for training in simulation e.g., by using GPU-offloading or server grade CPUs with a larger number of cores/threads, training time can be drastically improved. As a result, more extensive tuning studies can be performed to further increase performance.

At last, future research should investigate if this robotic optimization problem benefits from Multi-Objective RL (MORL) using sets of Pareto dominating policies [51]. In which the overall idea is to learn a policy for each objective and find the Pareto front of the combined policies. This way no information loss occurs that comes with chaining multiple objectives into one objective function. Moreover, this approach allows adapting the agent's cautious behavior near constraint violations. The agent can for example comprise performance for increased certainty in constraint satisfaction.

## VII. Conclusions

In this work, we investigated the feasibility of reinforcement learning in the robotic industry by solving a complex robotic optimization problem from the food packaging industry.

As the manufacturing industry moves towards greater flexibility, productivity, quality, and mass customization, heuristic-based solutions for solving complex optimization problems in real-world industrial robotic machines often fall short of meeting these demanding requirements. This is particularly relevant in challenging scenarios, such as varying product inflow, which result in revenue drops, waste and critical problems in the following stages of the manufacturing process. Furthermore, such solutions often need to be customized and pre-engineered making them inflexible and the development time-consuming. Additionally, highly complex optimization problems often require advanced and time-consuming calculations that must be performed in real-time during machine operation, which can negatively impact productivity. In contrast, RL has the potential to address these challenges by providing flexibility, real-time performance, generalization, and the ability to learn data-based predictive behavior, as outlined in the introduction. However, scientific research in this field is confined to the academic environment, as a result, challenges that arise on complex industrial robotic systems remain unaddressed. Therefore, we propose an RL framework designed to optimize the box conveyor belt speed of a secondary robotic packaging machine in simulation, using real-world product inflow data, in order to maximize the machine's productivity.

We demonstrated that the proposed framework achieves zero constraint violations in each simulation using real-world product inflow data. Contrary to the heuristic-based baseline, the proposed framework complies with all the given requirements set by the industry. In the validation study, performance surpasses the required $99.8\%$, the quality maintains at $100\%$ and the accelerations are kept in the feasible range. This means that $99.8\%$ of the supplied products are packed in a box, and no boxes leave the machine empty or partially filled. Compared to the heuristic baseline, our proposed solution improves performance by $0.63\%$, while also decreasing the mean acceleration and code execution time by $82.70\%$ and $55.05\%$, respectively.

We conclude that the framework is able to deal with control delays, sparse delayed rewards and policy inference in the complex interdependent control scheme of the task scheduler. With the proposed method predictive behavior for varying product supply is learned, while satisfying the machine's performance and quality constraints. The design encourages smooth control of the conveyor belt, reducing maintenance and increasing interpretability of learned policy. Additionally, it requires fewer time-demanding decisions during the operating

time of the machine. Moreover, it handles the limited availability of real-world product inflow data well by using solely simulated product inflow data and scenario randomization for training.

Future work should investigate how the learned policy computed with the proposed framework can be transferred to a physical robotic packaging machine. While this research has partially bridged the sim-to-real gap through the use of scenario randomization and real-world product inflow data, further efforts are necessary to address the remaining aspects of this gap.

## ACKNOWLEDGMENT

This project is conducted in collaboration with BluePrint Automation. They gave full access to the machine's simulator and task scheduler's source code and provided practical information regarding the food packaging industry. Their cooperation is hereby gratefully acknowledged.

## REFERENCES

[1] Ghobakhloo, Morteza. "The future of manufacturing industry: a strategic roadmap toward Industry 4.0." Journal of manufacturing technology management 29.6 (2018): 910-936.

[2] Wyrwa, Joanna, and Anetta Barska. "Innovations in the food packaging market: Active packaging." European Food Research and Technology 243 (2017): 1681-1692.

[3] BluePrint Automation, "Packaging Machines Manufacturer — Packaging Automation — Blueprint Automation". blueprintautomation.com. https://blueprintautomation.com/en/ (accessed Jan 19, 2023).

[4] BluePrint Automation, "00510 SPIDER 300v BPA animation", Unpublished internal company document, 2022.

[5] Zhao, Wenshuai, Jorge Peña Queralta, and Tomi Westerlund. "Sim-to-real transfer in deep reinforcement learning for robotics: a survey." 2020 IEEE symposium series on computational intelligence (SSCI). IEEE, 2020.

[6] Zhu, Henry, et al. "The ingredients of real-world robotic reinforcement learning." arXiv preprint arXiv:2004.12570 (2020).

[7] Dulac-Arnold, Gabriel, et al. "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis." Machine Learning 110.9 (2021): 2419-2468.

[8] Brunke, Lukas, et al. "Safe learning in robotics: From learning-based control to safe reinforcement learning." Annual Review of Control, Robotics, and Autonomous Systems 5 (2022): 411-444.

[9] Garcıa, Javier, and Fernando Fernández. "A comprehensive survey on safe reinforcement learning." Journal of Machine Learning Research 16.1 (2015): 1437-1480.

[10] Liu, Yongshuai, Avishai Halev, and Xin Liu. "Policy learning with constraints in model-free reinforcement learning: A survey." The 30th International Joint Conference on Artificial Intelligence (IJCAI). 2021.

[11] Moriyama, Takao, et al. "Reinforcement learning testbed for power-consumption optimization." Methods and Applications for Modeling and Simulation of Complex Systems: 18th Asia Simulation Conference, AsiaSim 2018, Kyoto, Japan, October 27–29, 2018, Proceedings 18. Springer Singapore, 2018.

[12] Nian, Rui, Jinfeng Liu, and Biao Huang. "A review on reinforcement learning: Introduction and applications in industrial process control." Computers & Chemical Engineering 139 (2020): 106886.

[13] Pane, Yudha P., et al. "Reinforcement learning based compensation methods for robot manipulators." Engineering Applications of Artificial Intelligence 78 (2019): 236-247.

[14] Dalal, Gal, et al. "Safe exploration in continuous action spaces." arXiv preprint arXiv:1801.08757 (2018).

[15] Pan, Xinlei, et al. "Risk averse robust adversarial reinforcement learning." 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019.

[16] Rana, Krishan, et al. "Bayesian controller fusion: Leveraging control priors in deep reinforcement learning for robotics." arXiv preprint arXiv:2107.09822 (2021).

[17] Shyalika, Chathurangi, Thushari Silva, and Asoka Karunananda. "Reinforcement learning in dynamic task scheduling: A review." SN Computer Science 1 (2020): 1-17.

[18] Liu, Chien-Liang, Chuan-Chin Chang, and Chun-Jan Tseng. "Actor-critic deep reinforcement learning for solving job shop scheduling problems." Ieee Access 8 (2020): 71752-71762.

[19] Zhang, Cong, et al. "Learning to dispatch for job shop scheduling via deep reinforcement learning." Advances in Neural Information Processing Systems 33 (2020): 1621-1632.

[20] Kim, Daewoo, et al. "Learning to schedule communication in multi-agent reinforcement learning." arXiv preprint arXiv:1902.01554 (2019).

[21] Waschneck, Bernd, et al. "Optimization of global production scheduling with deep reinforcement learning." Procedia Cirp 72 (2018): 1264-1269.

[22] Castañeda, Luis Angel, Alberto Luviano-Juárez, and Isaac Chairez. "Robust trajectory tracking of a delta robot through adaptive active disturbance rejection control." IEEE Transactions on control systems technology 23.4 (2014): 1387-1398.

[23] OEE Industry Standard, "OEE – Overall Equipment Effectiveness: The OEE Industry Standard," oeeindustrystandard.org. https://www.oeeindustrystandard.org/v2011/ (accessed Feb. 10, 2023).

[24] Otterlo, Martijn van, and Marco Wiering. "Reinforcement learning and markov decision processes." Reinforcement learning. Springer, Berlin, Heidelberg, 2012. 3-42.

[25] Schuitema, Erik, et al. "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach." 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010.

[26] Freund, Robert M. "Penalty and barrier methods for constrained optimization." Lecture Notes, Massachusetts Institute of Technology (2004).

[27] Bouteiller, Yann, et al. "Reinforcement learning with random delays." International conference on learning representations. 2020.

[28] Frydenberg, Morten. "The chain graph Markov property." Scandinavian Journal of Statistics (1990): 333-353.

[29] Köppen, Mario. "The curse of dimensionality." 5th online world conference on soft computing in industrial applications (WSC5). Vol. 1. 2000.

[30] Mousavi, Seyed Sajad, Michael Schukat, and Enda Howley. "Deep reinforcement learning: an overview." Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2. Springer International Publishing, 2018.

[31] Plappert, Matthias, et al. "Multi-goal reinforcement learning: Challenging robotics environments and request for research." arXiv preprint arXiv:1802.09464 (2018).

[32] Brys, Tim, et al. "Multi-objectivization of reinforcement learning problems by reward shaping." 2014 international joint conference on neural networks (IJCNN). IEEE, 2014.

[33] Sugiyama, Masashi, and Motoaki Kawanabe. Machine learning in non-stationary environments: Introduction to covariate shift adaptation. MIT press, 2012.

[34] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[35] Tobin, Josh, et al. "Domain randomization for transferring deep neural networks from simulation to the real world." 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017.

[36] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[37] Liu, Zechun, et al. "How do adam and training strategies help bnns optimization." International Conference on Machine Learning. PMLR, 2021.

[38] International Conference on Machine Learning. PMLR, 2021. Hendrycks, Dan, and Kevin Gimpel. "Bridging nonlinearities and stochastic regularizers with gaussian error linear units." CoRR, abs/1606.08415 3 (2016).

[39] Xiong, Ruibin, et al. "On layer normalization in the transformer architecture." International Conference on Machine Learning. PMLR, 2020.

[40] Charlesworth, Henry, and Giovanni Montana. "Plangan: Model-based planning with sparse rewards and multiple goals." Advances in Neural Information Processing Systems 33 (2020): 8532-8542.

[41] Goecks, Vinicius G., et al. "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments." arXiv preprint arXiv:1910.04281 (2019).

[42] Mohtasib, Abdalkarim, Gerhard Neumann, and Heriberto Cuayáhuitl. "A study on dense and sparse (visual) rewards in robot policy learning." Towards Autonomous Robotic Systems: 22nd Annual Conference,

TAROS 2021, Lincoln, UK, September 8–10, 2021, Proceedings 22. Springer International Publishing, 2021.
[43] Ng, Andrew Y., Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping." Icml. Vol. 99. 1999.
[44] Bøhn, Eivind, et al. "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization." 2019 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2019.
[45] Li, Lun, et al. "Basic flight maneuver generation of fixed-wing plane based on proximal policy optimization." Neural Computing and Applications (2023): 1-17.
[46] Jiang, Zifei, and Alan F. Lynch. "Quadrotor motion control using deep reinforcement learning." Journal of Unmanned Vehicle Systems 9.4 (2021): 234-251.
[47] Wang, Dawei, et al. "A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing." IEEE Robotics and Automation Letters 5.2 (2020): 3098-3105.
[48] Kang, Katie, et al. "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight." 2019 international conference on robotics and automation (ICRA). IEEE, 2019.
[49] Zhang, Huan, et al. "Robust deep reinforcement learning against adversarial perturbations on state observations." Advances in Neural Information Processing Systems 33 (2020): 21024-21037.
[50] Moos, Janosch, et al. "Robust reinforcement learning: A review of foundations and recent advances." Machine Learning and Knowledge Extraction 4.1 (2022): 276-315.
[51] Van Moffaert, Kristof, and Ann Nowé. "Multi-objective reinforcement learning using sets of pareto dominating policies." The Journal of Machine Learning Research 15.1 (2014): 3483-3512.

## APPENDIX

### A. Filtering of Real-World Data

The filtering of the real-world data is performed in two steps: prior to the evaluation of the trained policy and after evaluation prior to the data analysis. In the first step low inflows ($< 75$ [products/min]) corresponding to feeder startup issues or stops in production appearing as noise in the real-world data are filtered out. Algorithm 1 shows this filtering method. Moreover, in the first 100s the simulator and task scheduler are prone to startup issues, therefore the first 100s of input data are duplicated. Which is cut off in the second filtering step on the data gathered by evaluating the trained policy on the filtered real-world input data. After filtering 75.5 minutes of real-world input data is left for data analysis.

**Algorithm 1** Filtering real-world product inflow data

**Input:** Vector of detected product structs
**Output:** Vector of filtered detected products structs
*Initialisation* : minimum product inflow, $\phi$
**for each** product $\in$ Detected products **do**
  $t_{detect} \leftarrow$ product detect time
  $t_{detectprevious} \leftarrow$ previous product detect time
  $\Delta t \leftarrow t_{detect} - t_{detectprevious}$
  **if** ( $\Delta t < 60/\phi$) **then**
    Filtered detected products $\leftarrow$ product($t_{detect}$ - delay)
  **else**
    delay $\leftarrow$ delay + $\Delta t$
  **end if**
**return** Filtered detected products

### B. Parameters

Table III provides the parameters used in this research, except for the PPO hyperparameters, which are discussed in Appendix F.

TABLE III: Parameters

| Name | Value | Unit |
|---|---|---|
| **General:** | | |
| RL method | PPO | - |
| **RL timings:** | | |
| RL time step ($\Delta k$) | 0.75 | s |
| Action delay | 1.2 | s |
| Planned delay | 12 | s |
| Scheduling time step[a] | 0.15 | s |
| History length | 30 | RL time steps |
| **Speeds:** | | |
| Product belt speed $v_P$ | 0.64 | m/s |
| Minimum box belt speed $v_{B,min}$ | 0.096 | m/s |
| Maximum box belt speed $v_{B,max}$ | 0.82 | m/s |
| Control precision | $10^{-4}$ | m/s |
| **Machine configuration:** | | |
| C1 | 1.250/$v_P$ | s |
| C2 | C1 + 1.540 | s |
| C3 | 7.789/$v_P$ | s |
| C4 | 6.963 | m |
| Product detection (x-axis) | -2.600 | m |
| End of horizon (x-axis) | -1.600 | m |
| Box detection (x-axis) | -1.774 | m |
| Checkout (x-axis) | 8.500 | m |
| Center of workspace robot 1 (x-axis) | 0.000 | m |
| Center of workspace robot 2 (x-axis) | 1.638 | m |
| Center of workspace robot 3 (x-axis) | 3.276 | m |
| Center of workspace robot 4 (x-axis) | 4.914 | m |
| Length of workspace robot 1 and 2 (x-axis) | 0.700 | m |
| Length of workspace robot 3 (x-axis) | 0.650 | m |
| Length of workspace robot 4 (x-axis) | 0.550 | m |
| **RL features:** | | |
| Maximum value $x_{box}$ | 0.632 | m |
| Maximum value $x_{prod1}$ | 0.266 | m |
| Maximum value $x_{prod2}$ | 0.266 | m |
| **Training:** | | |
| Number of simulators | 16 | - |
| Random seed range for environments | [1015-1030] | - |
| Number of simulations (training) | 6827 | - |
| Simulation duration (training) | 1800 | s |
| Maximum product inflow | 120 | products / minute |
| Minimum product inflow | 135 | products / minute |
| **Ablation study:** | | |
| Number of simulations | 48 | - |
| Simulation duration | 1800 | s |
| Maximum product inflow | 120 | products / minute |
| Minimum product inflow | 135 | products / minute |
| **Validation study:** | | |
| Number of simulations | 7 | - |
| Simulation duration of seven scenarios | [747, 1023, 1755, 576, 465, 396, 267] | s |
| Maximum product inflow | 115 | products / minute |
| Minimum product inflow | 140 | products / minute |
| Threshold minimum product inflow filtering $\phi$ | 75 | products / minute |
| **Constraints:** | | |
| $q$ | 1 | - |
| $\zeta$ | $\frac{1}{4\cdot(v_{B,max}-v_{B,min})}$ | - |
| $\mu_{\text{prod}} = \mu_{\text{prod 1}} = \mu_{\text{prod 2}}$ | 1 | - |
| $\mu_{\text{box}}$ | 10 | - |
| $\mu_{\text{acc}}$ | $\frac{1}{4}$ | - |

[a] The time between scheduling cycles, each of which computes a single schedule as defined in 3.

### C. Low-Level Modules with Pseudocode

The lower-level modules, represented by the blue blocks in Fig. 8, all have their own low-level interaction framework with pseudo code in this appendix. The experiment design, information writer, reinforcement learning, custom gym environment, socket communication and callback module are shown in respectively Fig. 10, 11, 12, 13 & 14, 15 and 16.

**Experiment design**

```
def init_observation(experiment number):
        Select an observation space depending on the experiment number, for example:
            spaces.Box(low,high,shape,dtype)
            spaces.Dict(dict(spaces.Box(low,high,shape,dtype)))
        return observation space


def init_action(experiment number):
        Select an action space depending on the experiment number, for example:
            spaces.Discrete(number_of_discrete_actions)
            spaces.Box(high, low, shape, dtype)
        return action space

def select_state(experiment number, observation, product inflows):
        Select state features depending on the experiment number.
        Normalize each state feature extracted from the observation and product inflows
        in range [-1,1] with linear interpolation
        If history > current time:
            Add default values until feature length is equal to initialized shape of feature
        state = dict(all features with history)
        return state

def select_reward(experiment number, observation):
        Extract state and constraint violations regarding missed products, empty boxes
        and partly filled boxes from the observation
        Select reward function depending on the experiment number.
        reward = reward_function(state, constraint violations)
        return reward

def select_action(experiment number, observation)
        Select type of action depending on the experiment number.
        if discrete actions:
            speed = f(action)
        return speed

        else:
            speed = denormalized and discretized action to range
            [minimum speed, maximum speed] with linear interpolation
        return speed

def save_state_action_reward(experiment number, save path):
        Save the state, action and reward of the selected experiment
        by writing them to a file in the folder of the current experiment
```

Fig. 10: Experiment design

**Information writer**

```
def check_and_write_simulator_settings()
        Check feasibility
        Write settings to simulator file

def write_experiment_info()
        Write training/evaluation/validation info to file in specified log folder
        Write State, Action, reward of selected experiment to same file

def calculate_performance_metrics():
        Calculate the mean and std for each performance metric of tabel I and II

def constraints_satisfied()
        Calculate the number of constraint violations for each constraint:
        partly boxes, empty boxes, product efficiency and maximum acceleration.

def save_final_results()
        calculate_performance_metrics()
        constraints_satisfied()
        Write performance metrics to file
        Write data for plots in paper to file

def write_training_time()
        Write training time to file
```

Fig. 11: Information writer

**RL module**

```
from information writer import check_and_write_simulator_settings(),
write_experiment_info(), save_final_results(), write_training_time()

from Callbacks import SaveBestTraining(), EvaluateTrial(), Tensorboard()

def make_env()
        Register the custom gym environment
        Make the custom gym environment
        Add random seed to environment
        Wrap environment in monitor wrapper

def sample_params()
        Sample a set of hyperparameters for current trial with:
        trail.suggest_int()/_float()/_categorical()

def objective()
        Sample and update hyperparameters with sample_params()
        train(hyperparameters)

def train ()
        env = SubprocVecEnv([make_env() for number of environments to train or tune], 'spawn')
        callbacks = [SaveBestTraining(settings), EvaluateTrial(settings), Tensorboard(settings)]
        model = method.load('MultiInputPolicy', env, hyperparameters, settings)

        try:
            model.learn(training_budget, callbacks, settings)
        except:
            if simulator and/or scheduler executable is still running:
                psutil.Process(processID).kill()

        env.close()
        model.save(model_save_name)
        write_training_time()

        if trial_is_pruned:
            raise exception

        return last_mean_reward


if __name__ = '__main__':
        # Define simulator settings
            Simulation time
            Scheduled delay
            Scheduling type: if the baseline or RL solution is used.
            Validate policy: if simulated data is used or real-world data
            Communication settings: HOST and PORT

        # Define train/evaluate/validate for experiment(s)
            Method: define which model-free RL method to use (e.g. PPO)
            TRAINING: if training or evaluation of of policy
            Experiment: list of experiment number to train, evaluate or validate
            Number of episodes to train, evaluate or validate
            Number of environments to train, evaluate or validate with (= # logical processors)

        # Define OPTUNA settings
            OPTUNA: if hyperparameters should be tuned
            Study name
            Number of trials
            Number of jobs
            Number of startup trials
            Number of warmup steps
            Evaluation frequency during trial
            Number of environments used for evaluations during trial
            Number of episodes used for evaluations during trial

        # Define default hyperparameters
            Learning rate, batch size, activation function, ...


        check_and_write_simulator_setting()
        write_experiment_info()

        if OPTUNA:
            study = optuna.create_study(sampler(), pruner(), settings)

            try:
                study.optimize(objective(), settings)
            except:
                study.trials_dataframe().to_csv()  #write report

            study.trials_dataframe().to_csv() #write report

            plot_optimization_history()
            plot_importance_of_hyperparameters()

        elseif TRAINING:
            train(default hyperparameters)

        else:
            remove registered custom gym environment

            env = SubprocVecEnv([make_env() for number of environments to evaluate or validate]
                                , 'spawn')
            model = method.load(name_of_saved_model)

            try:
                mean_reward, std_reward = evaluate_policy(model, env, settings)
                final_results = env.env_method('print_evaluation_results')
                env.close()
                save_final_results()
            except:
                env.close()
                print the error
```

Fig. 12: The RL module

```
class: custom_gym_environment(gym.Env)

from experiment design import init_observation, init_action

# Define class variables
    Tracker array to keep track of state, actions and rewards at each time step during episode
    Tracker array to keep track of performance metrics and constraint violations per episode

def __init__(setting provided at registry):
    super(custom_gym_environment,self).__init__() #Acces inherited methods
                                                  from the main OpenAi Gym class
    # Declare instance variables, for example:
        Experiment number
        Environment ID
        Communication port and host
        Time step
        Save path for files
        Observation space = init_observation(experiment number)
        Action space = init_action(experiment number)
        VALIDATION
        EVALUATION
        reset_class_variables()



def reset_class_variables()
    Reset the class variables that keep track of the state, action and reward
    when a new custom gym environment is registered or when a new episode is started.


def get_product_inflow():
    if VALIDATION:
        Open log files with real-world inflow data
        Create array of product inflow data per lane from log file
    else:
        Open settings file for simulator
        Compute product inflow data with product influx per lane and influx
        time intervals from simulator settings file.

def extract_limits_for_plots():
    Calculate limits for speeds (action) and state features for interpretation
    of speed and state plots after training.

def reset():   #Overrides reset() function from main OpenAi Gym class
    if previous simulation is still running:
        process.kill() the simulator and scheduler executable

    while processes not yet killed properly:
        time.sleep(0.1)

    reset_class_variables()

    Start new simulation for given experiment number and communication port with:
    subprocess.Popen()

    Create a TCPClient() communication object with the host and port with:
    RLClient = TCPClient(Host, Port)
    Reset the class variables with:
    RLClient = RLClient.reset_class_variables()
    Make connection with the simulator with:
    RLClient.make_connection()

    try:
        Receive observation from server of simulator with RLCLient.recv_data()

    except socket.error:
        Print the error


    if observation received:
        Fill tracker arrays to keep track of state, action, rewards

    else:
        Terminate all child processes

    get_product_inflow()

    state = select_state(experiment number, observation, product inflows)

    return state
```

Fig. 13: Custom gym environment class (part 1)

```
def step(action):  #Overrides step() function from main OpenAi Gym class
    Convert the proposed action to speed with:
    speed = select_action(number of experiment, observation, action)

    Send message with proposed speed to the simulator with:
    RLClient.send_data(speed, length of message in bytes)

    Receive observation from server of simulator with:
    RLClient.recv_data()

    if disconnect message is received from simulator:
        DONE = True

    else:
        Fill tracker arrays to keep track of state and action

    reward = select_reward(number of experiment, observation)

    Fill tracker array to keep track of reward

    if DONE and EVALUATION:
        Calculate acceleration during episode
        Fill tracker arrays to keep track of of performance metrics and
        constraint violations per episode

    state = select_state(experiment number, observation, product inflows)

    return state, reward, DONE

def print_evaluation_results():
    extract_limits_for_plots()
    Plot actions, state features and constraint violations
    Save all plots
    return all tracker arrays

def close(): # Overrides close function from OpenAi Gym class.
    if simulation is still running:
        Kill the simulator and scheduler executable with:
        process.kill()

    while processes not yet killed properly:
        time.sleep(0.1)
```

Fig. 14: Custom gym environment class (part 2)

**class: TCPClient():**

```
# Define class variables
    Number of send messages

def __init__(host, port):
    # Declare instance variables:
        address = {host, port)
        connected = false
        Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        observation_msg = 240
        action_msg = 241
        disconnect_msg = 242


def reset_class_variables()
    Reset the class variables when a new TCPClient object is created


def make_connection():
    while not connected:
        try:
            socket.connect(address)
        except socket.error as exc:
            print("Failed to make connection with simulator. Socket.error: %s" % exc)

def disconnect(env_ID):
    socket.close()
    print(f" Gym environment with {env_ID} ID is disconnected from simulator server")

def recv_data(printing=False):
    Receive message from socket with:
    data = socket.recv(buffer)
    return __handle_recv_msg(data, printing)

def recv_data_and_print():
    return recv_data(True)


def send_data(msg, msg_length, printing=False):
    Keep track of number of send messages

    Construct message and send with:
    return socket.sendall(__handle_send_msg(msg, msg_lenght,
                                            number_of_send_msg, printing)


def send_data_and_print(msg, msg_length):
    return send_data(msg, msg_length, True)

def __extract_header_recv_data(data, printing):
    Unpack the header data in message with:
    struct.unpack(data)

    if printing:
        Print the header with:
        __print_msg_header(header)

    return msg_type from header data

def __handle_recv_msg(data, printing)
    Extract the msg_type from the header data with:
    msg_type = __extract_header_recv_data(data, printing)

    if msg_type == observation_msg:
        Unpack the observation data with:
        observation = struct.unpack(data)

        if printing:
            Print observation message

    return observation

    elif msg_type == disconnect_msg:
        Disconnect form simulator server with:
        disconnect()

        if printing:
            Print disconnect message
    return 0

    else:
        print("Unknown message received")
    return -1

def __handle_send_msg(msg, msg_length, number_of_send_msg, printing):
    Construct header containing:
        - Version info (necessary for simulator)
        - Message sequence (number of sent messages should be increasing)
        - Length of message
        - Command (number corresponding to the type of data in message)

    if printing:
        __print_msg(header, msg)

    Convert header and message to bytes and return with:
    return struct.pack(header, msg)
```

Fig. 15: TCPClient class

**Callbacks.py:**

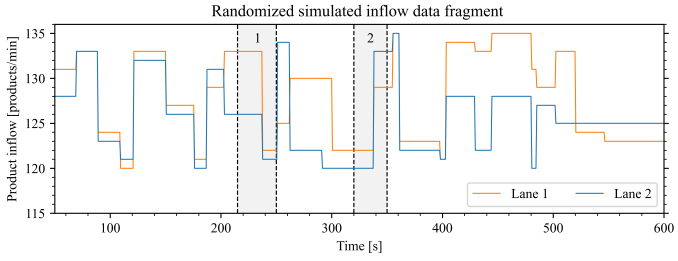
```
# Import the base callbacks
    from stable_baselines3.common.callbacks import BaseCallback, EvalCallback
    from stable_baselines3.common.logger import HParam


class SaveBestTraining()
# Callback for saving a model based on training reward. Reward is checked with a prior
defined frequency. Callback adapted from:
https://stable-baselines.readthedocs.io/en/master/guide/examples.html

    def __init__(setting):
        # Declare instance variables:
        super(SaveBestTraining,self).__init__()
        check_frequency
        save_frequency
        log_directory
        episode_mean_reward
        save_path
        best_mean_reward


    def _on_step():
        Calculate the moving average of the reward over the last X episodes
        Save intermediate model every save_frequency

    if mean_reward >= best_mean_reward:
        Save model with:
        model.save(save_path)

    return True


class TrialEvalCallback()
# Callback used for evaluating and reporting a trial. Necessary for hyperparameter tuning
with OPTUNA. Callback adopted from:
https://colab.research.google.com/github/araffin/tools-for-robotic-rl-icra2022/
blob/main/notebooks/optuna_lab.ipynb

    def __init__(setting):
        # Declare instance variables:
        super().__init__()
        Evaluation_environment
        Trial
        Number_of_episodes_to_evaluate
        Evaluation_frequency

    def _on_step():
        Evaluate policy with Evaluation_frequency
        Send report to Optuna
        Prune trial if needed
        return True


class TensorboardCallback()
# Callback for plotting additional training values in tensorboard. Callback adapted from:
https://stable-baselines3.readthedocs.io/en/master/guide/tensorboard.html

    def __init__(setting):
        # Declare instance variables:
        super(TensorboardCallback,self).__init__()
        evaluation_frequency
        number_of_missed_products
        number_of_empty_boxes


    def _on_step():
        Retrieve number of missed products and empty boxes from the custom gym
        environment with:
        number_of_missed_products += training_env.get_attr("missed_products")
        number_of_empty_boxes += training_env.get_attr("empty_boxes")

        if evaluation_frequency > 0 and n_calls % evaluattion_frequency ==0:
            Evaluate policy in parent class

            Record the retrieved scalar values from the training environment with:
            logger.record("episode/number_of_missed_products, number_of_missed_products )
            logger.record("episode/number_of_empty_boxes , number_of_empty_boxes )

            Reset the retrieved scalar values once recorded

        return True


class HParamCallback()
# Callback that saves the hyperparameters and metrics at the start of the training,
and logs them to TensorBoard. Callback adapted from:
https://stable-baselines.readthedocs.io/en/master/guide/tensorboard.html

    def __init__(setting):
        # Declare instance variables:
        super().__init__()

    def _training_start():
        Initialize the hyperparameters to be logged to TensorBoard with:
        hparam_dict = {
        "gae_lambda": model.gae_lambda,
        "batch_size": model.batch_size,
        ...}

        Initialize the metrics to be logged to TensorBoard with:
        metric_dict = {
        "eval/mean_reward": 0,
        "rollout/ep_rew_mean":0,
        "train/value_loss":0,
        ...
        }

        Record the hyperparameters and metrics at the start of the training with:
        logger.record("hparams", HParam(hparam_dict, metric_dict), exclude=(),)
```

Fig. 16: Callback classes

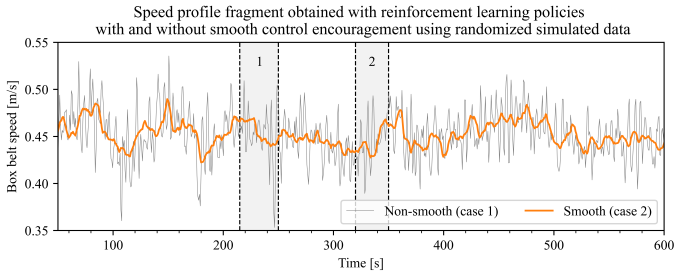
### D. Ablation Study figures

We provide additional figures in support of our ablation study. Figure 17b displays the speed profile for a segment of simulated product inflow data, generated using our scenario randomization method, as shown in Figure 17a. The orange line (Case 2) represents the speed profile with smooth control encouragement, while the gray line (Case 1) depicts the speed profile without such encouragement. In marked region 1, the learned policy reduces the box belt speed for decreases in product inflow, whereas in marked region 2, it increases the speed for increases in product inflow. This trend is evident across multiple product inflow variations, as shown in Figure 17. Notably, no products are lost in the presented fragment.

In addition, the RL agent learns to minimize the noise caused by unnecessary large speed changes in the randomized scenarios while satisfying all constraints and maintaining high performance (as shown by Case 2 in Table I and Figure 17b). Therefore, we conclude that smooth control encouragement, facilitated by the penalty function given in Equation (13), effectively mitigates unnecessary oscillations when using simulated product inflow data.




(a)




(b)

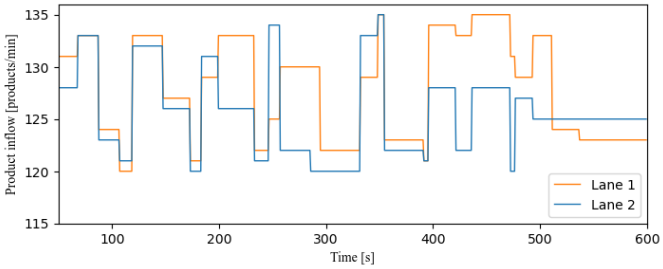
Fig. 17: Fragment of simulated product inflow data, generated with scenario randomization, and corresponding speed plot for case 1 and 2 listed in Table I.

### E. Low Pass Butterworth filter

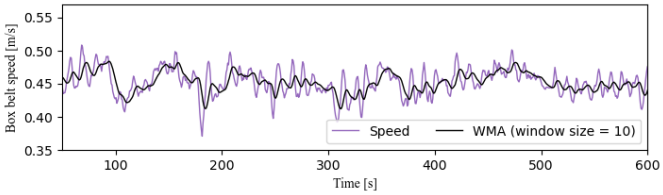
Penalty functions were employed in this research to promote smooth control, but we also conducted additional experiments to explore methods for mitigating unnecessary oscillations in the RL policies. This appendix presents the results of an experiment in which a low pass Butterworth filter was applied for post-filtering to reduce noise in the learned policy. The filter was designed with an order of 8, sample frequency of $2 \cdot \Delta k$ hertz, and cutoff frequency of 0.6 hertz.

Figure 18b displays the speed profile and its weighted moving average when randomized simulated product inflow data was used. The filter effectively decreased the noise in comparison to the policy without the filter (as shown by the gray line in Figure 17b). However, the policy was not as smooth as the learned policy obtained using our proposed method in this research (as indicated by the orange line in Figure 17b). The same conclusion was drawn for real-world product inflow data, as shown in Fig. 18d: the filter reduced the noise, but it did not exceed the performance of the learned policy obtained through our proposed method (as shown by the orange line in Figure 9b).

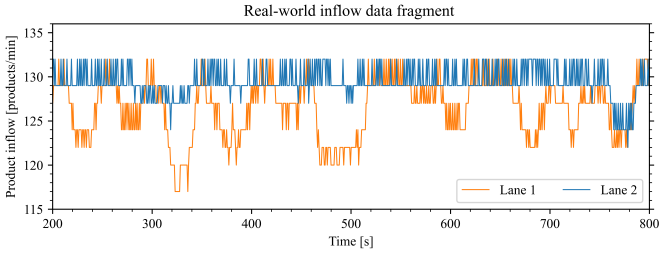
Furthermore, we observed that applying the post-filtering technique resulted in violations of the constraints. Consequently, we did not pursue this approach further in our research.



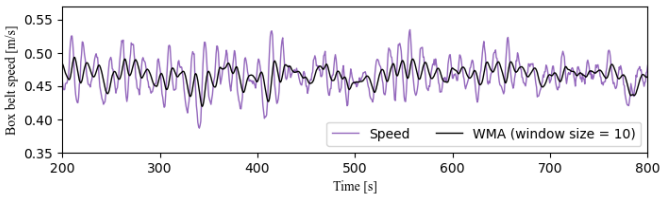

(a) Fragment of simulated product inflow data.




(b) Fragment of speed profile using simulated data




(c) Fragment of real-world product inflow data.




(d) Fragment of speed profile using real-world data

Fig. 18: Fragments of simulated (a, b) and real-world (c, d) inflow data with corresponding speed plots, generated for Case 1 (see tables I and II) using a low pass Butterworth filter.

### F. Hyperparameters Tuning Study with Optuna

TABLE IV: Hyperparamter tuning study

| Hyperparameter | Tuned Value [a] | Default Value [b] |
|---|---|---|
| Learning rate ($\alpha$) | $3.91{\cdot}10^{-4}$ | $3{\cdot}10^{-4}$ |
| Learning rate optimizer, schedule | Adam, Linear decay | Adam, Constant |
| Number of Epochs | 6 | 10 |
| Clip range | 0.3 | 0.2 |
| Clip range schedule | Linear decay | None |
| Entropy coefficient | $6.71{\cdot}10^{-3}$ | 0.0 |
| GAE Lambda ($\lambda$) | 0.91 | 0.95 |
| Number of nodes | 96 | 64 |
| Number of layers | | 2 |
| Activation function | | tanh |
| Minibatch size | | 64 |
| Horizon (T) | | 2048 |
| Discount factor ($\gamma$) | | 0.99 |
| Maximum gradient clipping | | 0.5 |
| Exploration | | Action noise |
| Limit to KL divergence | | None |

[a] If tuned value differs from default it is displayed.
[b] Default values from Stable Baselines3's PPO

The results of the hyperparameter tuning study conducted with Optuna are presented in Table IV. Our analysis revealed that increasing the size of the neural network beyond the default values did not lead to a clear improvement in training performance, but significantly increased the training time. We experimented with various numbers of layers (between 2 and 8) and nodes (between 64 and 256). However, due to limited computational resources, we were unable to perform an extensive tuning study to determine the relative importance of each hyperparameter and their degree of optimality. We did observe a slight increase in performance when the number of nodes was increased to 96, although this could be attributed to the randomness in exploration, the seed, or the stochasticity of the environment.

Our results indicate that the learning rate and clip range had the most significant influence on the training performance during the hyperparameter tuning study. We achieved stable training by linearly decaying both hyperparameters. Large learning rates led to drastic unlearning during the experiments, while small learning rates resulted in slow training. In Fig. 19, additional plots are presented for the mean reward, approximate KL-divergence, clip-fraction, clip-range, entropy loss, explained variance, learning rate, policy gradient loss, and value loss for Case 2 (sparse rewards and smooth control) and Case 3 (dense rewards and smooth control) of Table I. The higher mean reward in Case 2 compared to Case 3 was due to the different reward functions used. Furthermore, the higher explained variance in Case 3 compared to Case 2 explains the better performance of Case 3, as the learned policy predictions were better.

### G. Experiments for State, Action and Reward Design

A total of 71 experiments were conducted in order to develop the proposed methodology presented in this research.

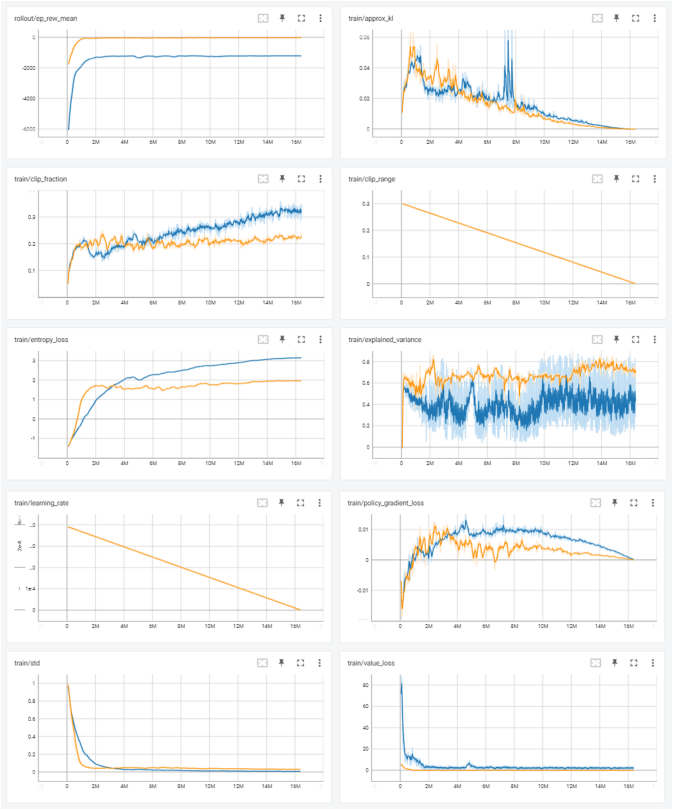



Fig. 19: TensorBoard plots from training policy Case 2: *sparse rewards with smooth control* (orange line) and Case 3: *dense rewards with smooth control* (blue line) Table II.

Various combinations and variations in state, actions, and reward functions were tested. Additionally, several RL methods (A2C, DQN, TRPO, and PPO) were experimented with, and only PPO demonstrated satisfactory performance for the belt speed optimization problem. Furthermore, a hyperparameter study was conducted in several experiments, as discussed in Appendix F. Due to the large number of experiments, it is not feasible to discuss each one in detail, therefore this appendix presents the main insights derived from the experiments.

We experimented with the following actions:

- **Continuous speeds:** These are the actions of our proposed methodology. The agent's action involves selecting speeds within the allowable range of minimum and maximum speeds. The primary challenge lies in ensuring that the maximum box belt acceleration is not exceeded.
- **Discrete speeds:** The continuous speeds are discretized for several discretization step sizes.
- **Discrete speed changes:** The agent chooses if the speed must be increased, decreased or kept constant. By varying the size of the speed change and the number of possible speed changes in combination with the RL time step, different acceleration profiles can be learned. The key challenge is to keep the resulting speeds between their minimum and maximum value.

We concluded that discretizing the continuous speeds led to

rapid learning, but the resulting policy did not generalize well for varying product inflows. Furthermore, the discrete speed changes were found to be sensitive to local maxima, as the learning agent often became trapped at the minimum and maximum box belt speeds. On the other hand, continuous actions were found to be prone to oscillatory behavior, yet yielded the best performance.

In addition to the state features listed in Section IV-C, we conducted experiments to investigate the impact of including the following features:

- Number of products and/or full/partly/empty boxes present in the machine.
- Number of products and/or full/partly/empty boxes out of reach of robots.
- Number of lost products, empty boxes and partly filled boxes.
- Number of supplied products
- Ratio between scheduled and non-scheduled products.
- Ratio between full/partly/empty boxes and present boxes in the machine.
- Ratio between lost products and supplied products (which is equivalent to the performance requirement given by (2)).
- Last position of products on conveyor belt before being picked up by robot.
- Last position of full/partly/empty boxes on conveyor belt before being filled with products.

Furthermore, we conducted experiments to determine the optimal length of the feature history. However, none of the aforementioned additional features yielded similar performance as the state features proposed in Section IV-C. This highlights the challenge in designing appropriate state representations.

In this study, we observed several interesting behaviors of the learning agent during the experiments. Specifically, we found that the two goals of saving boxes and saving products are contradictory, as high box belt speeds lead to saving all products, which correspond to a performance of 100 %, while low speeds lead to filling all boxes, which correspond to a quality 100 %. Balancing these two goals proved to be a challenge for the agent, resulting in large oscillatory behavior between speeds that achieved one of the two goals. To mitigate these oscillations, we introduced a penalty function in Equation (13) to encourage smooth control.

Another major turning point in our research was the introduction of planned delay and observation matching. Accurate policy inference with the correct frequency was found to be necessary to overcome the rescheduling problem described in Section IV-B. Without planned delay, the agent learned to keep the box belt speed constant to avoid rescheduling, which led to additional product loss. Interestingly, the constant box belt speed was indeed the best possible speed under the unfavorable circumstances. However, observation matching needed to be carefully applied to prevent aliasing in observing the current box belt speed due to the relatively large RL time steps.

### *H. Theoretical background: Reinforcement Learning*

This appendix provides additional theoretical background information on reinforcement learning. We cover the following topics: Markov decision process, Markov property, optimality criteria, Bellman expectation and optimality, the exploration-exploitation dilemma, temporal difference learning, RL taxonomy, proximal policy optimization and neural networks.

*1) **Markov decision process**:* A Markov decision process(MDP) is a mathematical framework which is a 5-tuple of states $\mathcal{S}$, actions $\mathcal{A}$, rewards $\boldsymbol{r}$, transition probabilities $p\left(s' \mid s, a\right)$ and the discount factor $\gamma$. With a MDP decisions are made taking into account immediate and future rewards depending on subsequent situations/states. The components of an MDP are defined according to Van Hasselt (2021) [1] as:

- $\mathcal{S}$ is a finite set of possible states.
- $\mathcal{A}$ is a finite set of possible actions.
- $p\left(s' \mid s, a\right)$ is the transition probability to $s'$ given state $s$ and action $a$.
- $\boldsymbol{r}:\ \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the expected reward when transitioning from $(s, a)$.
$$\boldsymbol{r} = \mathbb{E}[\mathcal{R} | s, a]$$
- $\gamma \in [0, 1]$ is a discount factor that trades off delayed rewards to immediate ones.

In fig. 20 the formalisation of the agent-environment interaction into a MDP is shown. The agent is continuously interacting with its environment by taking an action at time step $t$ using the information from the state and reward at that same time step. As a response, the environment produces the next state and reward at time step $t + 1$.

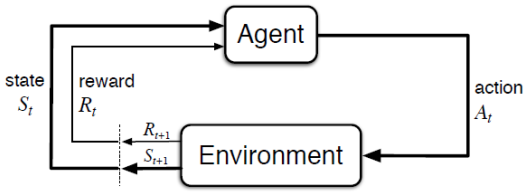



Fig. 20: Markov decision process with agent-environment interaction. Figure adopted from [2]

*2) **Markov Property**:* The agent state is *Markov* when it satisfies the *Markov Property*. Which means that the current agent state captures all the relevant information from history for selecting an appropriate action, in such manner that the full history can be thrown away. In a Markov decision process it is assumed that all agent states are Markov. [3]

**Definition Markov Property:** *"Consider a sequence of random variables, $\{S_t\}_{t \in \mathbb{N}}$, indexed by time. A state $s$ has the Markov property when for states $\forall s' \in \mathcal{S}$*

$$p\left(S_{t+1} = s' \mid S_t = s\right) = p\left(S_{t+1} = s' \mid h_{t-1}, S_t = s\right)$$

*for all possible histories*

$$h_{t-1} = \{S_1, \ldots, S_{t-1}, A_1, \ldots, A_{t-1}, R_1, \ldots, R_{t-1}\}.$$"

[1]

*3) **Optimality criteria**:* The goal of learning agent is to maximize cumulative reward. The optimal policy captures the strategy of the agent to reach this goal. Therefore, each RL algorithm contains an optimality criterion which evaluates the optimality of a policy. For a Markov Design Process three basic optimality models exist: the finite horizon reward model (19), the discounted, infinite horizon reward model (20) and the average reward model (21) [5].

$$J = \mathbb{E}\left[\sum_{t=0}^{N} R_t\right] \tag{19}$$

$$J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] \tag{20}$$

$$J = \lim_{N\to\infty} \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N} R_t\right] \tag{21}$$

In the three models the agent should maximize the expected reward.

*4) **Bellman Expectation and Optimality**:* With the elements of the MDP a value function is computed which can be used within RL algorithms to reach the goal of maximizing cumulative reward. A value function links the optimality criteria of the previous subsection to policies. The *value function* determines how good it is being in a state or in a state-action pair. It calculates the expected cumulative discounted reward from that state or state-action pair onwards following a policy $\pi$. Two types of value functions exist: a state-value function, $v_\pi$, and an state-action-value function, $q_\pi$, both under policy $\pi$. Both can be defined recursively resulting in the *Bellman Expectation Equations* for $v_\pi$ (eq. (22) [2]) and $q_\pi$ (eq. (23) [2]).

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \\ &\qquad\qquad \left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \\ &\quad \text{for all } s \in \mathcal{S}, \end{aligned} \tag{22}$$

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r \mid s, a\right) \\ &\qquad \left[r + \gamma \sum_{a'} \pi\left(a' \mid s'\right) \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'\right]\right] \\ &= \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \sum_{a'} \pi\left(a' \mid s'\right) q_\pi\left(s', a'\right)\right] \\ &\quad \text{for all } s \in \mathcal{S}, \end{aligned} \tag{23}$$

The Bellman expectation equations for $v_\pi$ and $q_\pi$ provide a judgement of respectively the state and state-action pair following a policy $\pi$. It is the expectation of the immediate reward, $r$, and the discounted value of the successor state, $q_\pi\left(s', a'\right)$ or $v_\pi\left(s'\right)$. The recursive behavior of the Bellman expectation equations allow the propagation of information of all the successor states following a policy ,$\pi$, back to the current state or state-action pair. The Bellman expectation equations can only be used for prediction problems, they evaluate the future, under a certain policy and do not provide information about the optimal value function or policy. For a given MDP with known transition probabilities and rewards, the optimal value functions obey the *Bellman Optimality Equations* stated in eq. (24) and (25) [2].

$$v_*(s) = \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right] \tag{24}$$

$$q_*(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right] \tag{25}$$

For a known MDP, the Bellman Optimality Equations can be used for solving control problems.

*5) **Exploration-Exploitation dilemma**:* A fundamental challenge in RL is the trade-off between exploring new states and exploiting the ones that have already resulted in high cumulative (discounted) reward. This challenge is also called the *exploration-exploitation dilemma*. Various strategies exist to trade-off exploration and exploitation, but the one most widely used in research is *$\epsilon$-greedy*. With the $\epsilon$-greedy approach the action with highest value is selected with probability $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$ and a random action is selected with probability $\frac{\varepsilon}{|\mathcal{A}(s)|}$ [6]. With $\epsilon$ the amount of exploration can be adjusted. The optimal action is selected more frequently when the total number of actions $|\mathcal{A}(s)|$ increases. Other types of strategies exist, like greedy, optimistic initialisation, Upper Confidence Bound, Thompson sampling, policy gradients and many more [1].

*6) **Temporal difference learning**:* In model-free RL the MDP is unknown, therefore the Bellman Expectation Equations can not be used directly. $v_\pi(s)$ and $q_\pi(s,a)$ can be estimated from experience using bootstrapped samples, which

is called TD learning. These estimations are denoted by $V_\pi(s)$ and $Q_\pi(s,a)$ respectively. In equation 26 the estimated state-value update of the TD(0) algorithm is shown. The estimate state-value is updated with the TD error multiplied with the learning rate parameter $\alpha$.

$$V_\pi\left(S_t\right) \leftarrow V_\pi\left(S_t\right) + \alpha \underbrace{\underbrace{\left(R_{t+1} + \gamma V_\pi\left(S_{t+1}\right)\right)}_{TD\ target} - V_\pi\left(S_t\right)}_{TD\ error\ (\delta_t)} \quad (26)$$

*7) **RL taxonomy**:* In Fig. 21 a mapping of RL algorithms is shown.

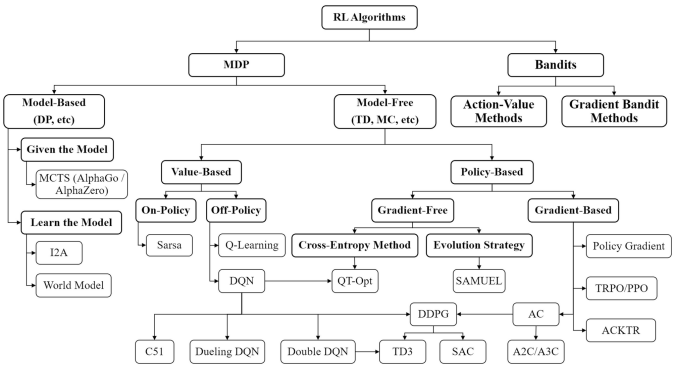



Fig. 21: Taxonomy of reinforcement learning algorithms. Figure adopted from [7]

A RL algorithm can be *value-based*, in which a state value or state-action value is learned from which the optimal policy is derived, *policy-based*, in which the (optimal) policy is learned directly or *actor-critic*, in which both a value and policy are learned. Actor-critics can be seen as the solution for the existence of high variance in policy gradient algorithms, therefore a lot of the state-of-the-art model-fee RL algorithms are based on the actor-critic framework. In table V the advantages and disadvantages of policy-based compared to value-based algorithms are shown. An actor-critic algorithm combines the advantages of value based methods (the critic), such as low variance and sample efficiency, with the advantages of the policy based methods (the actor) as stated in table V, at the cost of some bias [8].

| Advantages | Disadvantages |
|---|---|
| + True objective [1] | - Sensitive to local optima [1] |
| + Well-suited for high-dimensional or continuous action spaces [1] | - Generalisation problems [1] |
| + Can learn stochastic policies [1] | - High variance [8] |
| + Stable under function approximation (for sufficiently small learning rates) [8] | - Sample inefficient [8] |

TABLE V: Advantages and disadvantages of policy-based algorithms compared to value-based algorithms

*8) **Trust Region Policy Optimization**:* Policy gradient methods are subjected to variance and instability problems. Trust Region Policy Optimization (TRPO) solves both the issues. TRPO is an actor-critic algorithm that improves stability by restricting the size of each policy update [9]. A KL-divergence constraint, which constraints the difference between two distributions, is used to ensure each policy update lies within the trust region at each iteration. The TRPO optimization problem is defined as:

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}},a\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} Q_{\theta_{\text{old}}}(s,a)\right] \\ &\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}} \left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \| \pi_\theta(\cdot \mid s)\right)\right] \leq \delta \end{aligned} \quad (27)$$

In (27) the final TRPO optimization problem is shown. One applied simplification is the replacement of the advantage function $A_{\theta_{\text{odd}}}(s,a)$ by the action-value function $Q_{\theta_{\text{odd}}}(s,a)$. The advantage function is the difference between the action-value and state-value function as shown in eq. (28) and is a measure of how good an action in a given state is.

$$A_{\theta_{\text{old}}}(s,a) = Q_{\theta_{\text{old}}}(s,a) - V_{\theta_{\text{old}}}(s) \quad (28)$$

Although the advantage function is not used in the final optimization problem shown in (27), it is important background information as different formulations of the objective function exist in literature. Two other relevant concepts to understand the TRPO optimization problem are importance sampling and KL divergence.

*Importance sampling*

TRPO is considered on-policy, because it learns a policy from experience obtained by the latest version of that same stochastic policy. Importance sampling is used in TRPO to learn a new policy $\pi_\theta(a \mid s)$ by re-weighting the action-value $Q_{\theta_{\text{old}}}(s,a)$ estimated with samples from the old policy $\pi_{\theta_{\text{old}}}(a \mid s)$. Eq. (29) shows how it is incorporated in TRPO.

$$\mathbb{E}_{a\sim\pi_\theta}\left[Q_{\theta_{\text{old}}}(s,a)\right] = \mathbb{E}_{a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} Q_{\theta_{\text{old}}}(s,a)\right] \quad (29)$$

*KL divergence*

The Kullback-Leibler (KL) divergence measures the difference between two probability distributions [9]. The constraint shown in the optimization problem (27) restricts the divergence between the new policy $\pi_\theta$ and the old policy $\pi_{\theta_{\text{old}}}$ to be lower than or equal to the threshold $\delta$. When for example the probability of an action $a$ given state $s$ is large for the new policy but small for the old policy, the divergence between the two policies is probably large. The KL divergence ensures that each policy update is within the trust region, which stabilizes the algorithm.

*9) **Proximal Policy Optimization**:* Proximal Policy Optimizaton (PPO) is derived from the Trust Region Policy Optimization (TPRO) algorithm. While maintaining the benefits from TPRO, PPO is a more general and simpler version with better sample complexity [10]. The combination of good performance and ease of use makes PPO one of the most popular model-free deep RL algorithms at the moment. PPO is a relatively new model-free RL algorithm, it is published

by Schulman et al. in 2017 [10]. PPO simplifies TRPO by using a so called *clipped surrogate objective* which results in an unconstrained optimization problem without the KL divergence restrictions from TRPO. The aim is to maximize $J^{CLIP}(\theta)$ defined as [10]:

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}, \text{clip}\left(r(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{\theta_{\text{old}}}\right)\right]. \tag{30}$$

In which $\hat{A}_{\theta_{\text{old}}}$ is the estimate of the advantage function as defined in equation 28 and the ratio $r(\theta)$ is defined as [10]:

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)}. \tag{31}$$

The clipped surrogate objective function is similar to the objective function of TRPO. However, each policy update is now limited by taking the minimum of the original TRPO objective function and the clipped version, instead of using the KL divergence between the policies. The clip function limits the ratio between the new and old policy, $r(\theta)$, within $[1-\epsilon, 1+\epsilon]$, in which $\epsilon$ is a hyperparameter [10]. This way, too large policy updates are prevented while maximising the objective function. Therefore, PPO is able to stable improve the policy.

*10)* ***Deep RL: neural networks****:* Real-world RL problems often deal with high-dimensional continuous state-action spaces for which tabular solutions are not feasible anymore. Function approximations are used to overcome this problem, which is also referred to as *the curse of dimensionality*. Several function approximators can be used, such as a linear combination of features, nearest neighbour and neural networks. In this research the focus lies on the use of neural networks within RL, which is also referred to as *deep reinforcement learning*.

Neural Networks are a necessary addition to RL when state-action spaces are large or even continuous, which is mostly the case for real-world robotic systems. In figure 22 a simple 3-layer neural network is shown in which each node represents a neuron. The input vector is received trough the input layer, which has as many neurons as dimensions of the input vector. The hidden layers capture the actual learned behavior by use of weights, biases and activation functions. The final output is produced by the output layer. The amount of neurons in a single hidden layer is called the *width* of that layer. The number of layers, except the input layer, of a neural network is called the *depth* of the neural network. When a neural network has many layers, it is referred to as a *deep neural network*.

Each neuron can be modelled as a *perceptron*, which is shown in figure 23. A weighted sum of inputs with some bias is passed to an activation function, which could be of different types, for example ReLU, sigmoid or hyperbolic [12]. It is the activation function that makes the perceptron (or neuron) nonlinear, which is essential for the neural network to approximate any function [13].

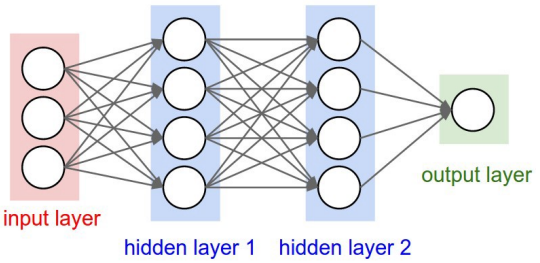



Fig. 22: Visualization of artificial neural network. Figure adopted from [11]

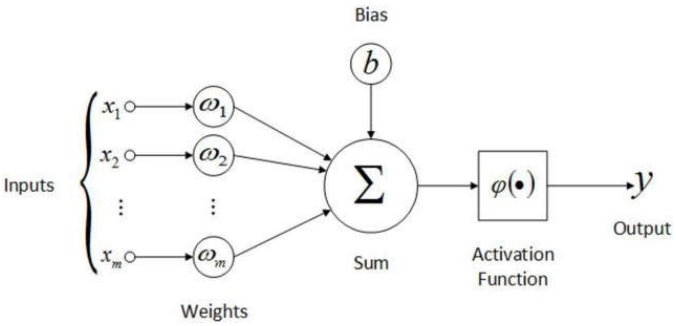



Fig. 23: Visualization of a perceptron. Figure adopted from [14]

The transformation within the neuron is captured by the following nonlinear transformation function $h$:

$$h = \varphi(\omega_{n,m} \cdot x_m + b_n), \tag{32}$$

in which $\omega_{n,m}$ is matrix containing the weights, $x_m$ is a vector containing the inputs, $b_n$ is a bias vector and $\varphi$ is the the activation function.

The simple type of neural network explained so far is called the FNN. In FNN each neuron in the hidden layers is fully connected to every other neuron of the previous layer [15]. The FNN is trained by optimizing the weights and biases to minimize a loss function, for example the mean squared error. Stochastic gradient descent with backpropagation is the most common method for optimizing the parameters of the neural network [15]. In this method the parameters are updated in the opposite direction of the gradient of the loss function.

More types of neural networks exist. The most popular ones are the CNN and the RNN, which are more complex and powerful than the FNN explained before. CNN is especially suited for spatial-data, such as images, and RNN for sequential data [15].

***References appendix L***

[1] H. van Hasselt, Reinforcement Learning Lecture 1: Introduction. Reinforcement Learning Lecture Series 2021, DeepmindxUCL, Sep 2021. https://deepmind.com/ learning-resources/reinforcement-learning-series-2021.

[2] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2020.

[4] D. Silver, Lecture 1: Introduction to Reinforcement Learning.

Introduction to Reinforcement with David Silver, DeepmindxUCL, May 2015. https://deepmind.com/ learning-resources/-introduction-reinforcement-learning-david-silver.
[5] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," Adaptation, learning, and optimization, vol. 12, no. 3, p. 729, 2012.
[6] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. de Freitas, "Reinforcement learning for mobile robotics exploration: A survey," IEEE Transactions on Neural Networks and Learning Systems, 2021.
[7] H. Zhang and T. Yu, "Taxonomy of reinforcement learning algorithms," in Deep Reinforcement Learning, pp. 125–133, Springer, 2020.
[8] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," Advances in neural information processing systems, vol. 30, 2017.
[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in International conference on machine learning, pp. 1889–1897, PMLR, 2015.
[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.
[11] S. University, Convolutional Neural Networks. CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2022. https://cs231n.github. io/convolutional-networks/.
[12] R. J. Pérez Dattari, "Interactive learning with corrective feedback for continuous-action policies based on deep neural networks," 2019.
[13] M. R. Baker and R. B. Patil, "Universal approximation theorem for interval neural networks," Reliable Computing, vol. 4, no. 3, pp. 235–239, 1998.
[14] D. Raj, "Single-layer neural networks in machine learning (perceptrons)," Jul 2020.
[15] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," arXiv preprint arXiv:1811.12560, 2018.

## *I. Literature review: Challenges of Real-world Reinforcement Learning*

This appendix addresses the challenges of real-world RL. The transfer of effective RL applications towards the real-world lags behind due to assumptions that are not met in the real-world. According to [24] the challenges of real-world RL are:

1) Offline learning from fixed logs created with an external behavior policy.
2) Using limited samples for learning on live system.
3) High-dimensional continuous state and action spaces.
4) Safety constraints which may not be violated.
5) Partially observable systems.
6) Learning from unspecified, risk-sensitive or multi-objective reward functions.
7) Computation of explainable policies to system operator.
8) Real-time inference.
9) System delays which are large and/or unknown.

Most of the research is focused on tackling one or a few of the proposed challenges, however all the nine challenges should be taken into account when designing and developing new algorithms. Each of these challenges is explained briefly in the following nine sections, together with possible solution directions.

*1) **Offline learning with fixed logs**:* In real-world RL it is often not feasible to learn from online interaction with the environment as the amount of necessary interactions is too expensive and/or time-consuming to execute [24]. In that case the policy needs to be learned offline from fixed logs of the system behavior. Offline RL, also called batch RL, is related to off-policy learning, in which a target policy, $\pi$ is learned while following another policy, $\mu$. When learning offline usually, the batches of data are obtained while following a different (unknown) policy than the policy that is learned. In off-policy methods also online learning can be applied, meaning that new data, which can be altered, is generated during learning [24].

*2) **Limited samples**:* Data-efficiency is important in most real-world systems as most of those systems are expensive to operate, relatively slow-moving or fragile [24]. Therefore the training data does not capture a large part of the state and action space, which emphasises the need for sample-efficient and quick learning [24]. Multiple attempts have been made to tackle this challenge.

| *Solution name* | *Explanation* | *Who* |
|---|---|---|
| Model agnostic Meta-Learning (MAML) | Train a model that learns how to learn in a new setting by use of other training data. | Finn et al. 2017 [1] |
| Bootstrap DQN | Applies deep exploration by using a bootstrapped neural network, which enables more efficient exploration of a Deep Q-Network. | Osband et al. 2016 [2] |
| Bootstrap DDPG with use of expert demonstration | Implementation of for example human demonstrations to speed up learning. | Vecerík et al. 2019. [3] |
| Model-based deep RL | Incorporating a model improves sample efficiency. | Hafner et al. 2018 [4]; Chua et al. 2018 [25]]; Nagabandi et al. 2019 [26]] |
| Soft Actor-Critics (SAC) | SAC is one of the most efficient model-free RL algorithm. Due to its off-policy entropy regulated framework. | Haarnoia et al. 2018 [5] |

TABLE VI: Collection of solutions found in literature for sample-efficient and quick learning

*3) **High-dimensional continuous state and action spaces**:* In real-world robotic systems the state and action spaces are often large and continuous instead of small and discrete. Function approximation is used to tackle this challenge by estimating the value function, policy, agent state and model depending on which part of the problem need to be approximated [6].

*4) **Safety constraints**:* Dealing with safety constraints are a major challenge for applying RL in real-world systems. In

most physical systems the system's output must satisfy certain constraints for safe operation. A RL agent must explore to find the optimal policy, but may never harm humans or destroy systems when doing so. *Safe* is a broad term in the context of RL; it also applies to less obvious safe constraints, such as constraining the minimum performance level of a system or the maximum monetary costs [24].

*5)* ***Partially observable****:* Real-world systems are partially observable as most of the time not every single part of the environment influencing the the system's state is visible to the agent. Examples of partial observations are malfunctioning sensors or actuators or a system containing multiple similar robots behaving slightly different. Those observations appear as noise, non-stationarity or as stochasticity [24]. Also, differences between simulators and real systems can be considered as partial observable. Table VII shows a collection of solutions to tackle this challenge.

| *Solution description* | *Explanation* | *Who* |
|---|---|---|
| History in agent's observation | Add history to observation to convert POMDP into MDP. | Mnih et al. 2015 [7] |
| Agent with recurrent network | Track and recover hidden state by use of recurrent network. | Hausknecht and Stone 2015 [8] |
| Sim-to-real transfer | Learn to transfer policies from simulation to real system. | Andrychowicz et al. 2018 [9]; Peng et al. 2018 [10] |

TABLE VII: Collection of solutions found in literature for dealing with partial observability.

*6)* ***Unspecified or multi-objective reward function****:* Often real-world systems have multiple objectives that need to be minimized, therefore the agent must trade off between different objectives [24]. It could be possible that the objective can not be clearly formulated at all. For both cases solutions exists, a couple are shown in table VIII.

| *Solution description* | *Explanation* | *Who* |
|---|---|---|
| Value function for each objective | - | Van Seijen et al. 2017 [11] |
| Policy optimizing of each sub-problem | - | Li et al. 2019 [12] |
| Pareto-dominating mixture of objectives | Finding the Pareto optimality for set of objectives | Moffaert and Nowé 2014 [13] |
| Conditional Value at Risk (CVaR) objective | Used as a risk measure for tasks with negative outcomes. Optimize for worst-case performance. | Tamar et al. 2015 [14]; |
| Inverse RL | When reward function is not fully specified | Ross et al. 2011 [15] |

TABLE VIII: Collection of solutions found in literature for dealing with unspecified or multi-objective reward functions.

*7)* ***Explainable policies****:* According to [24] explainability of policies is essential in real-world systems, because the systems are operated by humans who need to understand the behavior of the system especially in failure cases [24]. Model-based algorithms most of the time give more insight in the behavior of the system. In the work of [16] a set of methods is proposed for interpretation of deep neural networks. Still a lot of research needs to be done to tackle the challenge of explainability.

*8)* ***Real-time inference****:* For online model-free RL algorithms the control frequency of the agent and real-world system must be the same. The order of magnitude in which a system requires a control input could vary from milliseconds to minutes or even longer. So, for online RL, experience can only be acquired at real-time which can lead to too slow or too fast learning. The RL method can for example be too computationally expensive and not able to keep up with the required control speed, or the system reacts too slow for the agent to obtain enough experience to learn from. [24]

| *Solution description* | *Explanation* | *Who* |
|---|---|---|
| Real-Time Markov Reward Process | Incorporates evolving state during action selection. | Ramstedt and Pal 2019 [17] |
| Anytime inference | Return a valid solution at any time. | Vlasselaer et al. 2015 [18] |
| Reactive SARSA | Class of algorithms that deal with continuously changing asynchronous environments. | Travnik et al. 2018 [19] |

TABLE IX: Collection of solutions found in literature for dealing with real-time inference.

*9)* ***System delays****:* In real-world systems actions, observations and the reward signals can have significant delays due to for example low-frequency sensing or safety checks that must be performed before executing a certain action.

| *Solution description* | *Explanation* | *Who* |
|---|---|---|
| History in state | Learn the system delays from the recent history | Hester and Stond 2013 [20] |
| Incorporate intermediate reward | Incorporate intermediate rewards, when the true reward is obtained much later than the actions causing it. | Mann et al. 2018 [21] |
| Memory-based agent | Use memory to better assign rewards to past events. | Hung et al. 2018 [22] |
| RUDDER algorithm | Use a return-equivalent MDP in which delayed rewards are re-distributed. | Arjona-Medina et al. 2018 [23] |

TABLE X: Collection of solutions found in literature for dealing with system delays.

***References appendix L***


[1] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in International conference on machine learning, pp. 1126–1135, PMLR, 2017.

[2] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," Advances in neural information processing systems, vol. 29, 2016.

[3] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz, "A practical ap- proach to insertion with variable socket position using deep reinforcement learning," in 2019 international conference on robotics and automation (ICRA), pp. 754–760, IEEE, 2019.

[4] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learn- ing latent dynamics for planning from pixels," in

International conference on machine learning, pp. 2555–2565, PMLR, 2019.
[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., "Soft actor-critic algorithms and applications," arXiv preprint arXiv:1812.05905, 2018.
[6] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," Adaptation, learning, and optimization, vol. 12, no. 3, p. 729, 2012.
[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.
[8] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in 2015 aaai fall symposium series, 2015.
[9] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., "Learning dexterous in-hand ma- nipulation," The International Journal of Robotics Research, vol. 39, no. 1, pp. 3–20, 2020.
[10] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in 2018 IEEE international conference on robotics and automation (ICRA), pp. 3803–3810, IEEE, 2018.
[11] H. H. Van Seijen, S. M. F. Booshehri, R. M. H. Laroche, and J. S. Romoff, "Hybrid reward architecture for reinforcement learning," Apr. 13 2021. US Patent 10,977,551.
[12] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multiobjective opti- mization," IEEE transactions on cybernetics, vol. 51, no. 6, pp. 3103–3114, 2020.
[13] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 3483–3512, 2014.
[14] A. Tamar, Y. Glassner, and S. Mannor, "Optimizing the cvar via sampling," in Twenty- Ninth AAAI Conference on Artificial Intelligence, 2015.
[15] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 627–635, JMLR Workshop and Conference Proceedings, 2011.
[16] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understand- ing deep neural networks," Digital Signal Processing, vol. 73, pp. 1–15, 2018.
[17] S. Ramstedt and C. Pal, "Real-time reinforcement learning," Advances in neural infor- mation processing systems, vol. 32, 2019.
[18] J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt, "Any- time inference in probabilistic logic programs with tp-compilation," in Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
[19] J. B. Travnik, K. W. Mathewson, R. S. Sutton, and P. M. Pilarski, "Reactive reinforce- ment learning in asynchronous environments," Frontiers in Robotics and AI, vol. 5, p. 79, 2018.
[20] T. Hester and P. Stone, "Texplore: real-time sample-efficient reinforcement learning for robots," Machine learning, vol. 90, no. 3, pp. 385–429, 2013.
[21] T. A. Mann, S. Gowal, A. Gyorgy, H. Hu, R. Jiang, B. Lakshminarayanan, and P. Srini- vasan, "Learning from delayed outcomes via proxies with applications to recommender systems," in International Conference on Machine Learning, pp. 4324–4332, PMLR, 2019.
[22] C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne, "Optimizing agent behavior over long time scales by transporting value," Nature communications, vol. 10, no. 1, pp. 1–12, 2019.
[23] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," Advances in Neural Information Processing Systems, vol. 32, 2019.
[24] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, "An empirical investigation of the challenges of real-world reinforcement learning," arXiv preprint arXiv:2003.11881, 2020
[25] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," Advances in neural information processing systems, vol. 31, 2018.
[26] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in Conference on Robot Learning, pp. 1101–1112, PMLR, 2020.

### *J. Literature review: Safe Reinforcement Learning*

In 2015 García and Fernández proposed the first survey classifying the existing approaches towards safe RL until then [1]. They divided the existing approaches into the transformation of the optimization criterion and the modification of the exploration process. Since then a lot of research is done within the RL community to tackle the problem of safety. Most of the existing safe RL algorithms are a combination of transforming the optimization criterion and modifying the exploration process. Brunke et al. (2021) have proposed an updated overview of safe learning in robotics in which safety is addressed on a broader level by also focusing on safety in control instead of only within reinforcement learning [2]. In figure 24 a summary is shown of the proposed safe learning control approaches, in which the safety guarantees are plotted against the reliance of the approach on available data [2]. The safety guarantee axis is divided into three levels:

- Soft constraints (level I)
- Probabilistic/chance constraints (level II)
- Hard constraints (level III)

A hard constraint may not be violated, a probabilistic/chance constraint must satisfy the constraint with a specific probability and a soft constraint only encourages constraint satisfaction by penalizing constraint violation in the objective function [2]. The other axis is divided into known dynamics, unknown dynamics and variants of imperfect prior knowledge that lies in between.

As can be seen in the figure the approaches can be divided within this framework into *Safely Learning Uncertain Dynamics, Safety Certification* and *RL encouraging Safety and Robustness*. The first two directions must have (imperfect) prior knowledge about the dynamics of the system to guarantee hard and/or probabilistic constraints during learning. The power of RL is the ability to learn from

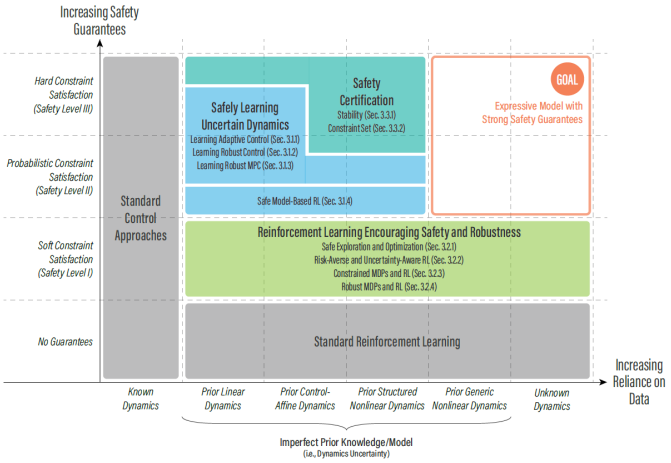



Fig. 24: Summary of the safe learning control approaches addressed the survey paper of Brunke et al. (2021). Figure adopted from [2]

interaction with the environment without having a prior model of the environment dynamics. However, as shown in fig. 24, this power is accompanied with a lack of successful approaches that provide formal guarantees for probabilistic and hard constraint satisfaction. The aim of the survey paper of [2] is to provide an overview of existing approaches that address safety within the control and RL community and by this bridge the gap between them towards increasing safety in RL. The light green block of fig. 24 covers the existing approaches in safe RL. The approaches are shortly introduced in this appendix with a focus on the trend covering constrained MDPs.

*1) **Safe exploration of MDPs**:* In literature safe exploration is addressed in several ways. By preserving ergodicity of the MDP any state is reachable from any other state by following a suitable policy [3], this way the agent never explores unsafe states from which it can not recover. However, finding the optimal policy that preserves ergodicity is a NP-hard problem [2]. Furthermore, the assumption of ergodicity is merely satisfied in real-world RL.

*2) **Risk-aware and cautious RL adaptation**:* In the risk-aware and cautious RL adaptation trend safety is encouraged by using metrics for risk or uncertainty during learning as guidance for the agent to explore safely. The CARL [4] and SAVED [5] algorithm are both model-based algorithms which learn risk-aware probabilistic models during pretraining and use these models to plan in unknown safety-critical environments.

*3) **Robust RL**:* In this trend safety is addressed in terms of robustness of the learned policy against system disturbances and generalization of the policy among different tasks. A robust RL agent can be trained by domain randomization [6] to generalize the learned policy for different domains/tasks or by adversarial training, in which an adversarial agent attacks a protagonist agent to train for system disturbances.

*4) **RL based on constrained MDPs**:* In this trend the constrained MDP (CMDP) framework is used to address safety. However, all these methods are still in their infancy as no promising performance is shown in simulations.
Different divisions of constraints are presented in literature. Kim et al. (2020) divide the constraints in feasibility and safety constraints [7]. Feasibility constraints bound the input signal to only feasible inputs, whereas safety constraints prevent inputs leading to unsafe situations. Liu et al. (2021) divide the constraints for policy learning in instantaneous and cumulative constraints [8], in which instantaneous constraints must be satisfied in each step and cumulative constraints from the beginning to the current time step [8]. In eq. 33 the instantaneous ($C_i\left(s_t, a_t, s_{t+1}\right) \leq \omega_i$) and cumulative constraint ($J_{C_i}^{\pi_\theta} \leq \epsilon_i$) problem of Liu et al. (2021) is shown, in which $J_R^{\pi_\theta}$ is the objective function as function of the reward $R$ when following policy $\pi_\theta$ parameterized by $\theta$ and $i$ being the number of instantaneous or cumulative constraints [8]. None of the proposed safe RL methods by Liu et al. (2021) satisfy both types of constraints.

$$\begin{aligned} &\max_\theta J_R^{\pi_\theta} \\ \text{s.t.} \quad &J_{C_i}^{\pi_\theta} \leq \epsilon_i \\ &C_i\left(s_t, a_t, s_{t+1}\right) \leq \omega_i \end{aligned} \tag{33}$$

Liu et al. (2021) provide an comprehensive overview of existing model-free RL methods for policy learning with constraints. They address Lagrangian relaxation, Constraint Policy optimization (CPO), Projection-based Constrained Policy Optimization (PCPO), Interior-point Policy Optimization (IPO), Lyapunov approaches, Backward Markov Chain (BMC), State Augmentation, Safety layer, Gaussian process approaches and a human in the loop as methods for solving constrained RL problems. In figure 25 a summary of the conclusions of the survey paper by Liu et al. (2021) is shown, in which the methods are divided according to the type of constraints they support and are assessed on the basis of performance, scalability, computational complexity and support by underlying theory [8]. The best performing methods are briefly addressed in this appendix.

| Constraint | Method | References | Performance | Scalability | Computation | Theorem |
|---|---|---|---|---|---|---|
| Cumulative | Lagrangian | [Altman, 1999] | ✓ | ✓ | × | × |
| | CPO | [Achiam *et al.*, 2017] | – | ✓ | × | ✓ |
| | IPO | [Liu *et al.*, 2020b] | ✓ | ✓ | ✓ | – |
| | PCPO | [Yang *et al.*, 2020] | ✓ | ✓ | × | ✓ |
| | Lyapunov | [Chow *et al.*, 2019] | ✓ | ✓ | × | × |
| | BMC | [Satija *et al.*, 2020] | ✓ | ✓ | ✓ | ✓ |
| | State Augmentation | [Xu and Mannor, 2011] | – | × | ✓ | – |
| Instantaneous | Lagrangian | [Bohez *et al.*, 2019] | ✓ | ✓ | × | × |
| | Safety layer | [Dalal *et al.*, 2018] | ✓ | ✓ | ✓ | × |
| | GP | [Wachi and Sui, 2020] | ✓ | ✓ | ✓ | ✓ |
| | Human | [Saunders *et al.*, 2017] | ✓ | × | ✓ | × |

Fig. 25: Comparison of constrained model-free RL approaches with score in descending order: check mark , - , and ×. Figure adopted from [8]

*Lagrangian relaxation*

Lagrangian relaxation is a widely used and powerful technique within classical optimization which relaxes complex constrained optimization problems. The problem is relaxed by removing the complex constraint by adding it to the objective function with a non-negative Lagrangian multiplier $\lambda$ and this way creating an unconstrained problem which penalizes constraint violation. By solving the resulting Lagrangian Dual problem with Lagrangian function L, constrained satisfaction is obtained. An example of a Lagrangian Dual problem for cumulative constraints is shown in equation 34 deduced from the original problem of equation 33 [8].

$$\min_{\lambda_i \leq 0} \max_{\theta} L\left(\theta, \lambda_i\right) = \min_{\lambda_i \leq 0} \max_{\theta} \left( J_R^{\pi_\theta} - \sum_i \lambda_i \left( J_{C_i}^{\pi_\theta} - \epsilon_i \right) \right) \tag{34}$$

The main drawbacks of this approach applied in constrained RL are the sensitivity to the initialization of the Lagrange multipliers and their difficulty to optimize in practice due to computational complexity as the optimization of the multiplier is done for every action taken by the RL agent. Moreover, zero constraint satisfaction is not guaranteed during learning with Lagrangian relaxation [9], however high performance in terms of long-term reward is obtained with the approach [10].

*Constraint Policy optimization*

CPO is based on the trust region method called TRPO, which is a local policy search algorithm that is effective for high-dimensional control. For a comprehensive explanation on TRPO see appendix H. The CPO algorithm calculates for every policy update the predicted change in constraint costs and selects the policy update within the trust region that results in the highest expected return without exceeding the admissible predicted constraint costs [11]. Achiam et al. (2017) claims guarantees for near-constraint satisfaction [11]. The use of conjugate gradients in CPO makes this approach computational expensive.

*Projection-based Constrained Policy Optimization*

PCPO is related to CPO and has the same drawbacks, thus having computationally expensive conjugate gradients. PCPO also uses TRPO, but makes use of projections of the policy on the closest policy that satisfies the constraints. In figure 26 the update procedure over PCPO is shown. [12]

*Interior-point Policy Optimization*

The IPO method combines the policy-gradient method PPO, which is explained in appendix H, with the idea behind interior-point methods, also called barrier methods. Liu et al. (2020) use the clipped surrogate objective of PPO (equation 30) with a logarithmic barrier function to reduce the optimization problem into a unconstrained one, as shown in equation 35. By using this logarithmic barrier function the solution is pushed away from the constraints to a certain extent depending on hyperparameter $t$. The goal is to have a penalty of zero in the objective function when the constraint

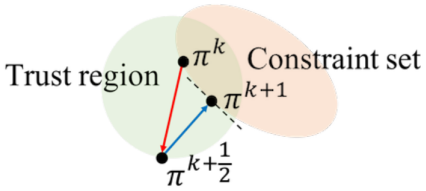



Fig. 26: Update procedures for PCPO. In step one (red arrow), PCPO follows the reward improvement direction in the trust region (light green). In step two (blue arrow), PCPO projects the policy onto the constraint set (light orange). Figure adopted from [12]

is satisfied and a penalty that goes to negative infinity when the constraint is violated [13]. IPO is easy to implement and tune and as showed in table 25 scores well on performance, scalability and computation, however this method does not have theoretical guarantees with respect to performance. [13]

$$\max_{\theta} \left( L^{CLIP}(\theta) + \sum_{i=1}^{m} \frac{1}{t} \log\left(-J_{C_i}^{\pi_\theta} - \epsilon_i\right) \right) \tag{35}$$

*Lyapunov*

In Lyapunov based approaches the constraints of a CMDP are modelled using Lyapunov functions. In control theory Lyapunov functions are widely used for analyzing stability of dynamical systems [14]. Chow et al. (2018) propose an approach in which these Lyapunov functions are constructed using linear programmiing. Under certain conditions the existence of a Lyapunov functions for a system is sufficient for system stability [15]. By identifying Lyapunov functions for the constraints of an CMDP the safety of a policy during training can be guaranteed [14]. A LP need to be solved for every action taken by the RL agent, therefore this approach is computationally expensive.

*Safety Layer*

The safety layer method can be used on top of any unsafe continuous control deep policy based RL algorithm. The safety layer corrects each action of the original policy by solving equation 36 in which the closest action to the original action proposed by a deep policy network is found. In figure 27 a visualisation of this method is shown. According to Dalal et al. (2018) the safety layer is trained with offline logged data of past trajectories of arbitrary actions to train a linear model for the constraints, $c_i(s, a)$ [16]. This method scores well on performance, scalability and computational complexity as shown in table 25.

$$\begin{aligned} &\arg\min_{a} \frac{1}{2} \left\| a - \mu_\theta(s) \right\|^2 \\ &\text{s.t. } c_i(s, a) \leq C_i \forall i \in [K] \end{aligned} \tag{36}$$

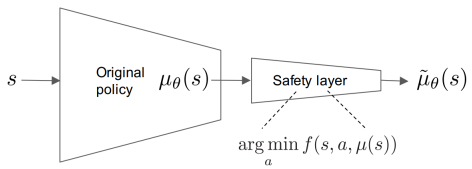



Fig. 27: Visualisation of the safety layer method. Figure adopted from [16]

### *References appendix L*

[1] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," Journal of Machine Learning Research, vol. 16, no. 1, pp. 1437–1480, 2015.

[2] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," Annual Review of Control, Robotics, and Autonomous Systems, vol. 5, 2021.

[3] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," arXiv preprint arXiv:1205.4810, 2012.

[4] J. Zhang, B. Cheung, C. Finn, S. Levine, and D. Jayaraman, "Cautious adaptation for reinforcement learning in safety-critical settings," in International Conference on Machine Learning, pp. 11055–11065, PMLR, 2020.

[5] B. Thananjeyan, A. Balakrishna, U. Rosolia, F. Li, R. McAllister, J. E. Gonzalez, S. Levine, F. Borrelli, and K. Goldberg, "Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks," IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3612–3619, 2020.

[6] W. Zhao, J. P. Queralta, and T.Westerlund, "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.

[7] Y. Kim, R. Allmendinger, and M. López-Ibáñez, "Safe learning and optimization techniques: Towards a survey of the state of the art," in International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning, pp. 123–139, Springer, 2020.

[8] Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, 2021.

[9] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh, "Lyapunov based safe policy optimization for continuous control," arXiv preprint arXiv:1901.10031, 2019.

[10] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," The Journal of Machine Learning Research, vol. 18, no. 1, pp. 6070–6120, 2017.

[11] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in International conference on machine learning, pp. 22–31, PMLR, 2017.

[12] T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge, "Projection-based constrained policy optimization," arXiv preprint arXiv:2010.03152, 2020.

[13] Y. Liu, J. Ding, and X. Liu, "Ipo: Interior-point policy optimization under constraints," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 4940–4947, 2020.

[14] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," Advances in neural information processing systems, vol. 31, 2018.

[15] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," Journal of Machine Learning Research, vol. 3, no. Dec, pp. 803–832, 2002.

[16] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," arXiv preprint arXiv:1801.08757, 2018.