Optimizing Parcels Sorting Through Reinforcement Learning for Intralogistics

Loris Roveda[a,b,*], **Marco Maccarini**[a], **Filippo Pura**[a], **Fabio Reiso**[c] **and Blerina Spahiu**[d]

[a]Department of Innovative Technologies, University of Applied Science and Arts of Southern Switzerland (SUPSI), Istituto Dalle Molle di studi sull'intelligenza artificiale (IDSIA)
[b]Politecnico di Milano
[c]Aidea srl
[d]Department of Computer Science, Systems and Communication of the University of Milano-Bicocca
ORCID (Loris Roveda): https://orcid.org/0000-0002-4427-536X, ORCID (Blerina Spahiu): https://orcid.org/0000-0002-6958-8215

Abstract. Sorting of parcels is a critical process in intralogistics for the proper processing and dispatching of packages. Commonly, such a process is manually executed by operators along the plant, without any added value, and might result in musculoskeletal injuries due to the non-ergonomic working conditions. Automation solutions are also present in the market and scientific literature. However, available solutions are usually implemented with pre-defined, simplified sorting rules/finite state machines capable of managing only a limited number of parcel types/sorting scenarios. To generalize and fully automate the sorting process in intralogistics, we propose to employ Reinforcement Learning (RL) for the derivation of sorting policies in combination with machine vision for the online tracking of the parcels, used as the state of the RL. More in detail, the on-policy Proximal Policy Optimization (PPO) algorithm is used for RL, and Yolo is chosen as the machine vision algorithm for parcel recognition and tracking. Based on the *AMS* sorting module of the SAIET Engineering company, a modular kinematic model (with parcels collision modeling) of the sorting system (an *n* by *m* AMS - *i.e.*, 2-action actuators - matrix) is derived, and used as the environment for the PPO. Offline sorting policy training is performed by randomizing the parcel number, size, and entry positions. The trained policy is then deployed to the sorting module, which is equipped with cameras for machine vision implementation and performance evaluation. In-distribution and out-of-distribution (*i.e.*, with parcel types not considered in the off-line training) tests achieved the target performance of 96.5% and 94% sorting accuracy, respectively.

1 Introduction

Intralogistics (*i.e.*, the management of material handling and information flow within the boundaries of a facility) [8] has witnessed significant transformation with the advent of advanced technologies. Among these, Artificial Intelligence (AI) and Machine Learning (ML) have emerged as important tools in optimizing operational efficiency [26]. In this context, sorting parcels is a critical application [13]. Sorting refers to the automated or semi-automated process of identifying, classifying, and directing items, such as parcels, containers, or components, based on predefined criteria such as destination, size, weight, SKU (stock keeping unit), or priority level. Traditional rule-based systems (*i.e.*, based on manually-tuned heuristics [32, 2]) often fall short in handling the dynamic and complex environments of modern intralogistics, where high volumes, product variability, and real-time decision-making are the norm. AI and ML offer adaptive, data-driven approaches that can learn from historical data, recognize patterns, and make intelligent decisions without explicit programming [31]. These technologies can enhance automated sorting systems through improved object segmentation, localization, and classification [25], anomaly detection [14], predictive maintenance [3], and autonomous control [24], enabling greater scalability and responsiveness.

In this paper, the combination of RL and machine vision is exploited to optimize the operation of a sorting system (*i.e.*, the *AMS*[1] sorting module of the SAIET Engineering company), where parcels are input as a batch, and a separation space between them is needed as the output, allowing for further processing. Specifically, Proximal Policy Optimization (PPO) [20] is used to train a policy that is then deployed to operate the sorting module. `YOLOv8-n` [25] is chosen as the machine vision algorithm for parcel segmentation, localization, and classification, composing the state in input to the policy. More in detail, the main contributions of this work are:

- the definition of a modular kinematic model (with parcels collision modeling) of the sorting system (an *n* by *m* AMSs - *i.e.*, 2-action actuators - matrix), which is used as the environment for the PPO. Offline sorting policy training is performed by randomizing the parcel number, size, and entry positions;
- the trained policy is deployed to the sorting module, which is equipped with cameras for machine vision implementation and performance evaluation. `YOLOv8-n` is employed to detect and localize the parcels, and such information is used as the state of the policy;
- in-distribution and out-of-distribution (*i.e.*, with parcel types not considered in the off-line training) tests are provided, achieving the target performance of 96.5% and 94% sorting accuracy (*i.e.*, in terms of success rate of the sorting task), respectively.

* Corresponding Author. Email: loris.roveda@supsi.ch; loris.roveda@polimi.it.

[1] https://www.saiet.engineering/ams/

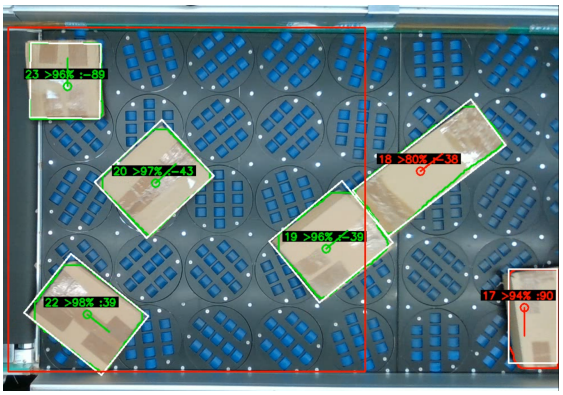
This paper presents a comprehensive implementation, testing, and evaluation of a reinforcement learning and machine vision-based sorting framework. It offers insights into both the development and deployment stages, with a particular focus on training environment modeling, reward design, PPO configuration, real-time deployment, and `YOLOv8-n` fine-tuning.

The rest of the paper is structured as follows: Section 2 provides an overview of the state-of-the-art in the field of machine learning for sorting systems, Section 3 introduces the sorting task and its modeling, Section 4 defines the employed methodology, Section 5 states the achieved results, Section 6 discusses the limitations of the method, and Section 7 summarizes the paper and the future work.

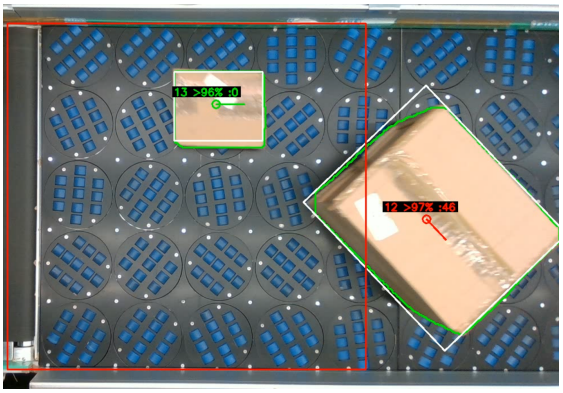
2 Related works

Machine vision and machine learning algorithms have been applied to sorting systems to improve their performance, enhancing the detection, classification, and localization of parcels, as well as the efficiency and rate of success of the sorting system itself. On the one hand, machine vision approaches for the sorting task can be analyzed. [4] proposes a fast, adaptive binarization method for QR code images in logistics systems under uneven lighting. By detecting position patterns to define window size and using integral images to speed up local thresholding, the approach achieves high efficiency and accuracy. While showing a good recognition rate, it relies on clear detection of position patterns, limiting robustness under severe distortion. [17] proposes a visual sorting method combining Mask R-CNN with 3D point cloud data to improve express package sorting in complex logistics scenes. It achieves a 97.2% success rate by accurately detecting packages and computing grasp points. However, its performance depends on high-quality RGB-D data and may be affected by occlusion or poor lighting. [28] introduces a YOLOv4-based method for express parcel sorting using barcode and three-segment code fusion. It improves accuracy (reaching 98.5%) via optimized detection and OCR. However, recognition speed is moderate, and performance may drop under severe occlusion. [7] presents a logistics sorting algorithm using improved YOLOv3 and Mask R-CNN for object localization, combined with sampling evaluation for grasp planning. It achieves high accuracy and robust performance, but it depends on the quality RGB-D input and may struggle with occlusion. On the other hand, machine learning approaches for the sorting task can be discussed. Q-learning is applied in [30] to path planning and sorting on omnidirectional-wheel conveyors. It achieves efficient, collision-aware routing without traditional control logic. However, it deals only with a few parcels at a time (5, with a low hourly rate), directing them to specific locations and without requiring the creation of a gap between them, as needed in this paper. Omniveyor is introduced in [29], a modular logistics sorting system using reinforcement learning to control conveyor modules for efficient, high-density sorting. Its main limitations are related to scaling the policy to different scenarios and real-world deployment, and not being able to manage a high volume of parcels.

In this paper, the following state of the art challenges are tackled: i) fast online parcels segmentation, localization, and classification, ii) scalable and deployable (to real-world sorting modules) RL-based policies, and iii) sorting tasks requiring the handling of high parcel rates for gap creation (required for further processing).



(a) Sorting task not correctly executed.



(b) Sorting task correctly executed.

Figure 1: In (a), an incorrect sorting task is shown. There is no gap at the exit of the sorting module. In (b), a correct sorting task is shown, with a gap between packages at the exit of the sorting module. The gap at the sorting module exit refers only to spacing between packages in the Y direction; spacing in the X direction is not relevant.

3 Sorting task description and modeling

3.1 Sorting task description

The sorting task considered in this paper aims to create a gap (*i.e.*, space between the parcels leaving the sorting module) between the parcels at the end of the sorting module. The parcels enter the sorting system in batches, and the sorting system must be operated to sort them. Figure 1b shows a correct execution of the sorting task (on the right side) vs a wrong execution (on the left side). The size of the considered parcels ranges between 5 cm to 45 cm (considering the maximum length of the parcel), with a weight between 0.05 kg and 2 kg. As a task specification, 6000 parcels/hour (pph) have to be processed.

3.2 AMS module

To perform the sorting task, the SAIET's AMS module is employed. Such a sorting module is composed of an n by m matrix of AMS actuation systems. Each AMS actuation system, with a square shape with a width of 0.2 m, is characterized by 2 actuators, *i.e.*, 2 actions (one forward velocity to move the parcels a_{fv}, one rotation to direct the parcel motion a_r, see Figure 2). In this paper, the sorting module is composed of a 5x5 matrix of AMS actuation systems, with a total width of 1 m. Indeed, the AMS module is characterized by a total of 50 actions. For the

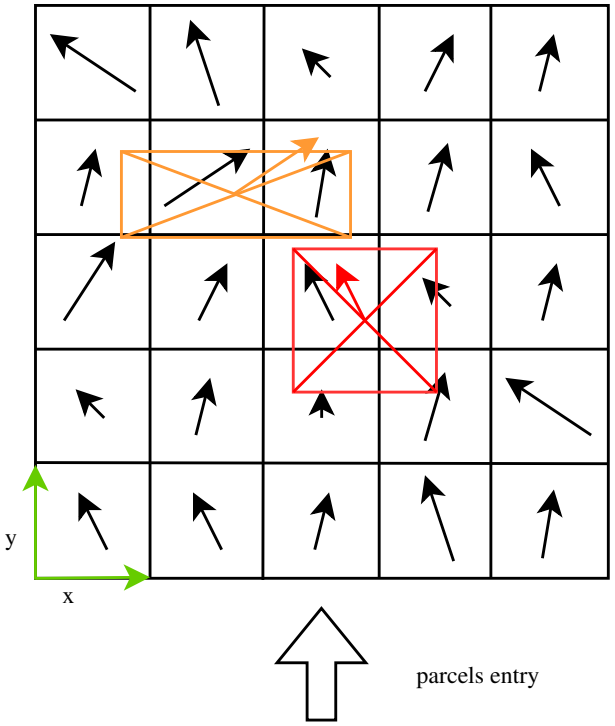


Figure 2: The kinematic model of the AMS module is derived from the schema in this Figure. Each AMS actuation system is characterized by two actions. $a_{fv}(i,j)$ defines the module of the forward velocity applied by the AMS actuation system to the package, while $a_r(i,j)$ defines the direction of the applied velocity, where i and j define the AMS actuation system within the AMS module. For each package, its geometrical baricenter is considered to define which actuation system forward velocity and direction applies.

implementation of the sorting task, two consecutive AMS modules are used to being able to properly perform the sorting task, each of them controlled by the trained PPO agent, as discussed in Section 5.

Remark 1. It has to be noted that the size of the AMS module (*i.e.*, the number of employed AMS actuation systems) can be configured based on the actual needs, even though this affects the environment modeling, no modifications are required for training the RL policy, thanks to a robust code implementation.

3.3 *Sorting task modeling*

Based on the sorting task and the AMS module description above, it is possible to derive a model for the sorting system to be employed as the environment for the RL. In this paper, the derived model is purely kinematic. This choice is justified by the reduced size/weight of the considered parcels (*i.e.*, the dynamic effects of the parcel, including, *e.g.*, inertia and friction, are negligible for the operation of the sorting module).

The schematization of the AMS module and parcels' motion is shown in Figure 2. The parcels enter from below with a given constant velocity along the y axis (*i.e.*, simulating a treadmill feeding the sorting module). Each square in the matrix represents the (i,j) AMS actuation system. The control actions $a_{fv}(i,j)$ and $a_r(i,j)$ are visualized on each AMS actuation system by the length (*i.e.*, the bigger the arrow, the faster the velocity action $a_{fv}(i,j)$ is) and orientation of its arrow, respectively. For each parcel k, its geometrical baricenter b_k is considered to define which actuation system $AMS(i,j)$ forward

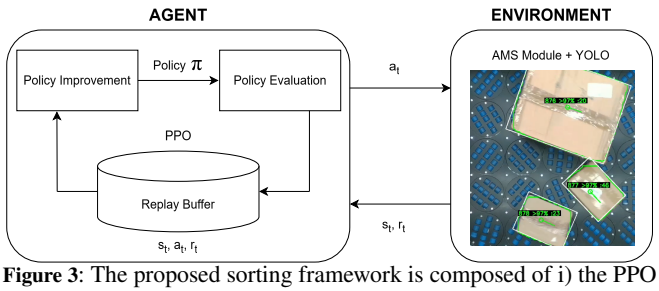


Figure 3: The proposed sorting framework is composed of i) the PPO agent and ii) the environment (*i.e.*, the AMS module equipped with YOLO for parcels segmentation, localization, and classification). The RL policy computes in real-time the actions a_t to be provided to the sorting module based on the segmentation, localization, and classification of the parcels performed by the YOLO algorithm (composing the RL state s_t and used to compute the reward r_t).

velocity and direction applies for the definition of its velocity v_k, so that $a_{v,k} = a_v(i,j)$ and $a_{r,k} = a_r(i,j)$. Therefore, the x and y position coordinates $p_{k,x}(t)$ and $p_{k,y}(t)$ of each parcel $\mathbf{p}_k = [p_{k,x}(t), p_{k,y}(t)]^T$ at timestep t is computed as follows:

$$\begin{cases} p_{k,x}(t) = p_{k,x}(t-1) + \Delta t \cdot a_{v,k}(t) \cdot \cos(a_{r,k}(t)), \\ p_{k,y}(t) = p_{k,y}(t-1) + \Delta t \cdot a_{v,k}(t) \cdot \sin(a_{r,k}(t)), \end{cases} \tag{1}$$

where Δt is the simulation timestep.

We model the lateral collisions of a parcel with another and/or with the lateral bounds of the AMS module. Indeed, the computed position in (1) is updated if collisions are detected to avoid lateral overlapping and the parcel to exit from the AMS module laterally. Frontal overlapping is instead allowed to simulate envelopes-parcels overlapping.

The pseudocode of the proposed modeling is provided in Algorithm 1.

Remark 2. In the proposed model, each parcel motion is affected only by the AMS actuation system where the parcel's geometrical baricenter is on. No rotational modeling is defined as it is negligible for the target performance.

Algorithm 1: Parcel Position Update

```
for k = 0 to K − 1 do
    for i = 0 to n − 1 do
        for j = 0 to m − 1 do
            if b_k on AMS(i, j) then
                a_{v,k} = a_v(i, j)
                a_{r,k} = a_r(i, j)
                p_{k,x}(t) = p_{k,x}(t − 1) + Δt · a_{v,k}(t) · cos(a_{r,k}(t))
                p_{k,y}(t) = p_{k,y}(t − 1) + Δt · a_{v,k}(t) · sin(a_{r,k}(t))
                if lateral_collision then
                    update p_k(t)
```

4 Methodology

As RL has been demonstrated to be powerful for continuous action control purposes, even for complex scenarios [21, 15, 22], it is employed in this paper to perform the above-described sorting task. Specifically, the PPO algorithm trains a policy offline and is then

deployed to the sorting module for its real-time control. The model described in Section 3.3 is employed as the RL environment. Machine vision (*i.e.*, `YOLOv8-n`), instead, is adopted to segment, localize, and classify the parcels, to define the state of the RL algorithm. The designed framework is shown in Figure 3. The following details of the RL (*i.e.*, state, action, reward, etc.) and YOLO algorithms are given.

4.1 Reinforcement Learning

4.1.1 Preliminaries

In this study, we examine a reinforcement learning setting in which an agent interacts with its environment in discrete timesteps Δt. The problem is formulated as a discrete-time continuous-space Markov decision process (MDP) [23], defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ is the continuous state space, $\mathcal{A}$ the continuous action space, $\mathcal{P}$ the transition kernel that assigns the probability $p(s_{t+1} \mid s_t, a_t)$ of moving to state s_{t+1} after taking action a_t in state s_t, $\mathcal{R}$ the reward function with $\mathcal{R}(s_t, a_t) = r_t \in \mathbb{R}$, and $\gamma \in [0,1]$ the discount factor that weights future rewards relative to immediate ones.

A *policy* π is a stochastic mapping from states to probability distributions over actions, $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The agent's goal is to find a policy $\pi(a_t \mid s_t)$ that maximizes the expected discounted return $\mathbb{E}\big[\sum_{t=0}^{H} \gamma^t r_t\big]$, where H denotes the horizon of the episode. In deep RL, the policy is instantiated as a neural network parameterized by θ; therefore, learning amounts to optimizing θ so that the resulting policy exhibits optimal behavior.

4.1.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [19] replaces the KL restricted trust region of Trust Region Policy Optimization [18](TRPO) with a clipped surrogate objective, keeping updates inside a first-order "trust region" while allowing multiple Stochastic Gradient Descent (SGD) passes over the same data. Large-scale ablation studies [1, 10] have shown that weight orthogonalisation, advantage normalization, and value-function clipping are critical for stability. Variants such as the Phasic Policy Gradient [5] decouple the policy and value learning phases, and PPO remains the dominant backbone for Reinforcement Learning from Human Feedback (RLHF) pipelines that align large language models with preference data [27].

4.1.3 RL state and actions

At every discrete time $t \in \mathbb{N}$, the agent observes a *state vector* $\mathbf{s}_t \in \mathbb{R}^{4N}$ defined as:

$$\mathbf{s}_t \;=\; \big[\mathbf{s}_t^{(0)},\, \mathbf{s}_t^{(1)},\, \ldots,\, \mathbf{s}_t^{(N-1)}\big]^T, \qquad \textit{with } \mathbf{s}_t^{(k)} \in \mathbb{R}^4, \tag{2}$$

where $N = n \cdot m$ is the number of *AMS* actuation systems as described in Section 3.1. Considering $n = m = 5$, $N = 25$ and $\mathbf{s}_t$ has a dimension of 100.

Each AMS actuation system is identified by its row–column pair (i, j) with $i \in \{0, \ldots, n-1\}$ (rows) and $j \in \{0, \ldots, m-1\}$ (columns). We employ the row–major linear index:

$$k \;=\; i + jn, \qquad k \in \{0, \ldots, N-1\}, \tag{3}$$

so that the ordering of the blocks in (2) matches the physical layout of the matrix.

The four entries composing the RL state $\mathbf{s}_t^{(k)} = [x_k,\, y_k,\, w_k,\, h_k]^T$ describe (at most) one parcel whose baricenter currently lies on the AMS actuation system k:

- x_k [m]: global (*i.e.*, w.r.t. the coordinate system in Figure 2) x-coordinate of the parcel baricenter, used to localize the parcel along the x-axis;
- y_k [m]: global (*i.e.*, w.r.t. the coordinate system in Figure 2) y-coordinate of the parcel baricenter, used to localize the parcel along the x-axis;
- w_k [m]: width of the parcel along the x-axis, used to estimate the parcel encumbrance along the x-axis;
- h_k [m]: width of the parcel along the y-axis, used to estimate the parcel encumbrance along the y-axis;.

If no parcel baricenter occupies the AMS actuation system k at time t, all four entries of $\mathbf{s}_t^{(k)}$ are set to 0.

The proposed state representation keeps its size fixed, as the number of AMS actuation systems is pre-defined. In fact, if a parcel-based state is defined, it would change its size based on the number of parcels on the AMS module, which varies and is not known a priori.

Based on the observed state $\mathbf{s}_t$, the agent issues an *action vector* $\mathbf{a}_t \in \mathbb{R}^{2N}$ defined as:

$$\mathbf{a}_t \;=\; \big[\mathbf{a}_t^{(0)},\, \mathbf{a}_t^{(1)},\, \ldots,\, \mathbf{a}_t^{(N-1)}\big]^T, \qquad \text{with } \mathbf{a}_t^{(k)} \in \mathbb{R}^2, \tag{4}$$

where each $\mathbf{a}_t^{(k)} = [a_{r,k},\, a_{v,k}]^T$ encodes the control commands sent to the k-th AMS actuation system:

- $a_{r,k}$ [rad]: rotation command, controlling the orientation of the k-th actuation module;
- $a_{v,k}$ [m/s]: linear velocity command, controlling the speed of the k-th actuation module along its driving direction.

The action vector structure mirrors the AMS actuation system layout by applying control commands in row–major order according to the indexing in (3). As for the state, the action vector has a fixed size dependent only on the predefined number of actuation systems, enabling consistent agent–environment interactions across time.

4.1.4 Hyper-parameters

For hyper-paramenters setup, we follow the Stable-Baselines3 [16] defaults:

- **roll-out length and optimisation schedule:** each policy update is based on a roll-out of $n_{\text{steps}} = 2048$ on-policy transitions per actor. This length balances the *variance* of the gradient estimate (which falls with longer roll-outs) against the *non-stationarity* introduced by delaying policy updates (which rises with longer roll-outs) [20, 9]. The aggregated batch (2048 steps $\times$ N_{env} parallel environments) is shuffled and split into mini-batches of size 64; the surrogate objective is optimised for 50 epochs, so that every sample is reused at most ten times, remaining within the reuse budget recommended in [20]. A mini-batch of 64 keeps GPU utilisation high while limiting correlation between successive gradient estimates;
- **learning-rate schedule and optimiser:** we employ the Adam optimiser [11] with an initial learning rate of $\alpha_0 = 3 \times 10^{-4}$ and linearly anneal α to 0 over the entire training horizon. Linear decay permits aggressive exploratory updates early on and progressively smaller steps near convergence, implicitly mimicking a shrinking trust region.
- **clipping coefficients:** PPO constrains updates through separate clipping terms for the policy and the value function. We keep the canonical values $\varepsilon_\pi = \varepsilon_V = 0.2$, which bound the policy-ratio surrogate and the squared-error value loss, respectively. In practice,

Table 1: Hyperparameters used for PPO.

Hyperparameter	Value
Hardware configuration	1 NVIDIA GPU + 12 CPU cores
Discount factor γ	0.99
Generalized Advantages Estimation λ	0.95
PPO clipping parameter $\varepsilon_\pi = \varepsilon_V$	0.2
Optimizer	Adam [11]
Actor's learning rate	$3e-4$
Critic's learning rate	$3e-4$
Mini-batch size	64
Number of epochs	50
Value loss weight	0.5
Entropy loss coefficient c_2	1e-4
Parcel for each episode	20
Parcel generation frequency	[40-100 time-steps]

$\varepsilon_\pi = 0.2$ limits the empirical Kullback–Leibler divergence between successive policies to $O(10^{-2})$, preventing destructive shifts while still enabling rapid early learning [20].

Training parameters are also summarized in Table 1.

4.1.5 Reward design

After several simulation studies aiming for reward design, we propose the immediate reward formulation presented in Algorithm 2, described in detail in the following. At simulation time t, let $\mathbf{P}_t = \{p_1, \ldots, p_K\}$ denote the set of K parcels located within a designated rewarding area of length 1 meter along the travel axis, situated after the AMS system, where parcels advance linearly at a fixed speed upon exiting the system. Without loss of generality, we sort the parcels in $\mathbf{P}_t$ by their centroid y-coordinate so that $y(p_1) \leq \cdots \leq y(p_K)$. The immediate reward r_t is computed according to the following steps:

- **step 1 – pair-wise gap:** for each consecutive pair (p_k, p_{k+1}), $k = 1, \ldots, K-1$, we define the *signed* gap along the y-axis as:

$$d_k := y_{\min}(p_{k+1}) \; - \; y_{\max}(p_k),$$

  where $y_{\min}(\cdot)$ and $y_{\max}(\cdot)$ are the lower and upper y-coordinates of the parcel's bounding box. If the two boxes do not overlap in the y-direction, d_k is positive; vice versa, it is negative. In such a way, we can reward gaps and penalize overlaps;
- **step 2 – local score:** each gap is converted into a clipped local score:

$$ls_k \; = \; \min\big\{\, 10 \cdot K \cdot \arctan(d_k) + K,\; 6 \big\}.$$

  Such a formulation is based on an iterative empirical design procedure;
- **step 3 – worst-case reward:** the minimum of the local scores is used to compute the reward r_t:

$$r_t \; = \; \min_{1 \leq k \leq K-1} ls_k;$$

- **step 4 – additional reward:** if there is a positive gap between *all* parcels (*i.e.*, the sorting task has an accuracy of 100%), *i.e.*, $d_k > 0 \; \forall k$, and at least three parcels are present in the current RL step, *i.e.*, $K \geq 3$, an additional bonus is given:

$$r_t \; += 20.$$

The cumulative reward G_t for the episode is finally computed as:

$$G_t = \sum_{\varepsilon=0}^{T-t} \gamma^{\varepsilon} \, r_{t+\varepsilon+1},$$

where the index ε is used to iterate the sum of the immediate rewards, and T is the terminal step of the episode.

Algorithm 2: Immediate reward

```
flag = True
scores = [ ]
K = len(P_t)
for k = 0 to K − 1 do
    p_k = P_t[idx]
    p_{k+1} = P_t[idx + 1]
    d_k = distance(p_k, p_{k+1})
    ls_k = min(10 · N · arctan(d_k) + N, 6)
    scores.append(ls_k)
    if d_k < 0 then
        flag = False
r_t = min(scores)
if flag then
    r_t += 20
```

4.1.6 Training results

The learning capabilities of the PPO agent are summarised in Figures 4 and 5. During the very first training iterations, the policy behaves nearly at random, yielding a strongly *negative* average episodic return of -246.2 at step 0. After only 68 updates, the return crosses the zero line and keeps increasing almost monotonically until convergence. The agent attains its peak performance of 1 313.4 at update 2 368 and finishes with a stable average return of 864.7. Over the last 100 updates, the return settles around $\mu = 768.2$ with a modest variance, indicating that the learned policy has consolidated and exhibits consistent behaviour. These results confirm that the clipping mechanism of PPO does not impede long-term task improvement.

The trend in performance is mirrored by the surrogate objective (Figure 5). The average loss starts at -2.53×10^{-2} and steadily descends to approximately -4.39×10^{-2}, reaching its minimum value of -5.10×10^{-2} at step 2 350, just before the reward attains its maximum. A Pearson correlation of $r = -0.72$ between loss and return highlights their strong inverse relationship: as the policy update becomes more profitable (the loss becomes more negative), the environment reward increases accordingly. Beyond $\sim$ 1 500 updates the loss plateaus, signalling that the KL-divergence constraint is successfully preventing destabilising policy shifts. Taken together, these curves demonstrate that the optimizer strikes a favourable balance between exploration and exploitation, allowing the agent to reach a high-performing policy without over-fitting its surrogate objective.

In the accompanying video [2], we show the training trajectories of five models trained for 1 minute, 1 hour, 2 hours, 4 hours, and 16 hours, respectively. Consistent with the trend illustrated in Figure 4, model performance improves sharply as the number of training steps—and therefore the total training time—increases.

4.1.7 Simulation test results

The model, trained for 16 h and described in the previous Sections, was evaluated in simulation using an in-distribution configuration. A total of 1000 parcels were processed under an average throughput of $\approx$ 6000 pph, reproducing real-world operating conditions. The resulting mean singulation error was $\approx 3\,\%$ (*i.e.*, sorting accuracy of $\approx 97\,\%$). As shown in the accompanying video[3], the sorting errors are concentrated during short load spikes (*i.e.*, big bulks).

[2] https://drive.switch.ch/index.php/s/ZWEf0NkMMsg0JcM
[3] https://drive.switch.ch/index.php/s/etvGxyxToB2tVgt

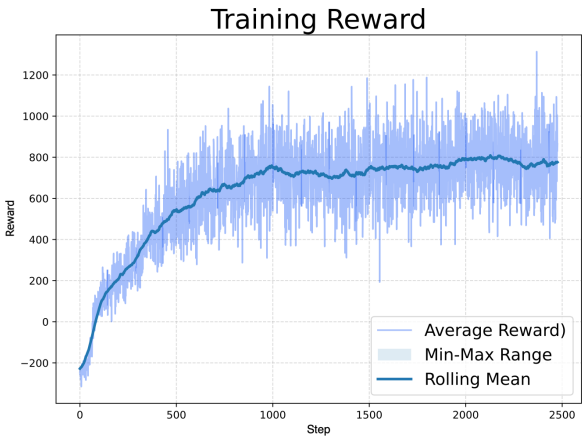


Figure 4: **Learning curve of the PPO agent.** Average episodic return (solid line) as a function of the training update. The shaded band denotes the range (*min–max*) observed over the evaluation roll-outs at each checkpoint. After an initial exploration phase, the return increases rapidly, surpasses zero at update 68, and finally converges to a stable plateau after around 800 steps.

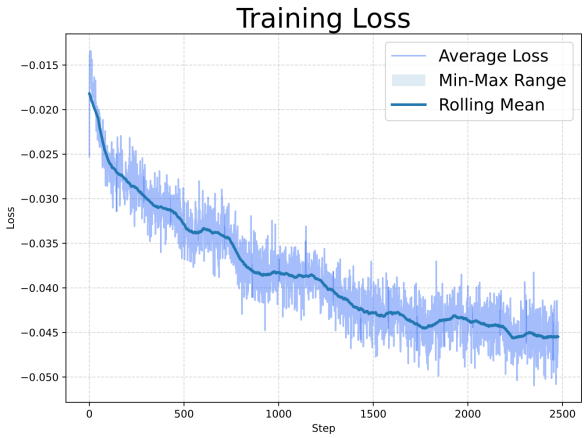


Figure 5: **Surrogate objective loss during training.** Mean clipped-surrogate loss per update. More negative values indicate more profitable policy improvements. The loss decreases sharply in the first 500 updates and then plateaus, reflecting the stabilising effect of the KL-divergence constraint and entropy regularisation.

4.2 *YOLO for parcels segmentation, localization, and classification*

This section details the construction of a machine vision pipeline able to segment, localize, and classify parcels in real time. To this end, `YOLOv8-n` [25] was fine-tuned on a custom dataset of parcel images.

4.2.1 *Dataset collection and pre-processing*

A dataset of 1300 RGB images (resolution $\sim 1280\,\text{pixel} \times 720\,\text{pixel}$) was collected directly on the sorting plant under production lighting conditions. Each image was manually annotated with a single *parcel* bounding box using Roboflow [6]. The dataset was randomly split into 70 % training, 20 % validation, and 10 % test sets.

To enlarge the effective sample size and increase appearance invariance, the following on-the-fly augmentations were applied during training:

- **auto-orientation**: EXIF tags were read and images were rotated to upright;

Table 2: Confusion matrix on the test set.

	Actual package	**Actual background**
Predicted package	97%	2%
Predicted background	1%	0%

- **resizing & letterboxing**: input frames were scaled to $640\,\text{pixel} \times 640\,\text{pixel}$ while preserving aspect ratio;
- **random crops**: up to 15 % of the borders were randomly cropped;
- **rotations**: random in-plane rotations in $[-15^\circ, 15^\circ]$;
- **noise injection**: additive white Gaussian noise with $\sigma \sim \mathcal{U}(0, 0.05)$;
- **photometric jitter**: random brightness, contrast, and saturation were applied to the dataset.

4.2.2 *Model architecture and fine-tuning*

Experiments adopted the `YOLOv8-n` backbone pre-trained on the COCO dataset [12]. The network was fine-tuned for a single-class detection task for 200 epochs using the following hyper-parameters:

- optimiser: AdamW (β_1=0.9, weight decay $= 1 \times 10^{-5}$;
- initial learning rate: 1×10^{-2} with cosine decay;
- early stopping: patience $= 100$ epochs on validation $\text{mAP}_{50\text{–}95}$ (mean Average Precision).

Inference is performed at 16 Hz on an NVIDIA RTX-4090 GPU, meeting the real-time requirement.

4.2.3 *Performance evaluation*

Detection performance were assessed on the held-out test split with the *confusion matrix* (Table 2) and the derived *precision*, *recall* and F_1*-score*. For a binary class (package vs. background), let TP, FP, FN and TN denote true positives, false positives, false negatives and true negatives, respectively. The measures are defined as:

$$\begin{aligned} \text{Precision} &= \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \\ \text{F}_1\text{-score} &= 2\,\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}. \end{aligned} \tag{5}$$

Precision quantifies prediction correctness, recall measures the proportion of actual parcels detected, and the harmonic F_1-score balances the two. For object detection, mean Average Precision at IoU ≥ 0.5 (mAP_{50}) is also reported.
With the counts of Table 2 we obtain:

$$\text{Precision} = 0.98, \qquad \text{Recall} = 0.99, \qquad \text{F}_1\text{-score} = 0.98.$$

The detector, therefore, meets both accuracy and runtime constraints for deployment on the distribution line. Figure 6 provides a direct visual assessment of the model's accuracy: as shown, the model segments the parcels with high reliability and no observable errors.

5 Experimental results

The trained policy and the fine-tuned YOLO have been deployed to the real sorting system. Two test scenarios were devised:

(i) **in-distribution:** the system handled *only* parcels (*i.e.*, packages), mirroring the training conditions;

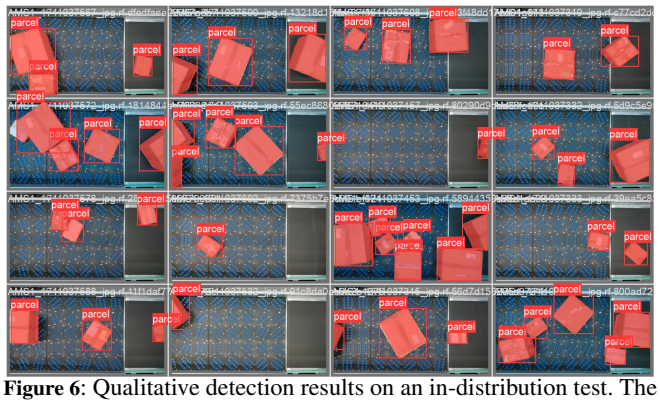


Figure 6: Qualitative detection results on an in-distribution test. The model maintains high segmentation accuracy and a low false-positive rate under real-world lighting and motion blur.

(ii) **out-of-distribution (mixed flow):** the system processed a stream containing both parcels and envelopes. This setting is inherently more challenging because envelopes are lighter and smaller; they interact less with the AMS surface, which reduces controllability and yields more complex dynamics.

These experiments enable a direct comparison between nominal performance and the model's robustness when confronted with previously unseen object types.

Two AMS modules are employed to compose the sorting system. Each of the sorting modules is controlled by an independent policy trained as in Section 4.1.

5.1 In-distribution test

As shown in the accompanying video[4], the model was evaluated in a parcels-only configuration for the in-distribution test, representative of real distribution-center conditions. To ensure a fair comparison with the simulation experiments, the average throughput was fixed at 6000 *pph*. The model shows a mean absolute error of $\approx 0.5\,\%$ higher than in the simulation, achieving a sorting accuracy of $\approx 96.5\,\%$.

5.2 Out-of-distribution test

Beyond the in-distribution configuration, we assessed the model under a deliberately more challenging setting in which *envelopes* were introduced into the scene together with parcels, keeping the processing rate *pph* unchanged with respect to the in-distribution case. After retraining the vision model described in Section 4.2 to recognize both parcels and envelopes, we deployed it to this new scenario.

Because envelopes differ markedly from parcels in terms of size and—crucially—mass, the task is strictly out-of-distribution: the model must now handle items whose physical properties diverge from those seen during training. Envelopes are more difficult for the sorting system to handle: their low mass reduces the normal force (and thus the frictional grip) with the AMS actuation systems, while their smaller footprint makes them more challenging to sort.

As it is shown in the video[5], the sorting accuracy dropped only by $\approx 3\%$, achieving a sorting accuracy of $\approx 94\%$. This decrease in performance can be largely attributable to occasional bursty inflows of mixed items and, as noted above, to the presence of envelopes themselves.

[4] https://drive.switch.ch/index.php/s/q5Lao8UDROQH2YH
[5] https://drive.switch.ch/index.php/s/2OkMfmCbc1blbqt

6 Discussion and limitations

The proposed framework is effectively capable to learn the proposed sorting task in simulation (based on the proposed modeling in Section 3.3). As shown in the in-distribution simulation and experimental tests, a similar sorting accuracy has been achieved ($\approx 97\%$ vs $\approx 96.5\%$, respectively). This indicates that the proposed modeling is reliable and well-represents the real sorting task.

The out-of-distribution test reveals only a modest drop in sorting performance, underscoring the trained policy's robustness in unseen scenarios. Despite unmodeled variations in envelope behavior, the system still achieves high accuracy. While a more sophisticated dynamic model might further enhance offline training, its substantial implementation complexity and uncertain payoff would unduly burden our straightforward, low-compute approach.

The enhanced RL framework equipped with machine vision allows to control in real-time the complex (*i.e.*, 50 actions) sorting system under any parcel scenario. However, the main operative limitation comes with the control frequency, which is limited by the YOLO inference frequency (16 Hz). By increasing the YOLO inference frequency, it would be possible to increase the AMS module control frequency, which might allow achieving increased sorting performance.

The proposed policy is specifically tailored to the task and parcel types defined in Section 3.1 and, consequently, does not directly adapt to alternative objectives, *e.g.*, cross-flow management or new parcels. Nonetheless, the underlying framework is inherently task-agnostic: by redefining the reward function detailed in Section 4.1.5, one can encode arbitrary operational goals and derive policies for entirely new tasks. Additionally, the YOLO algorithm can be easily fine-tuned with new parcel datasets. Furthermore, the methodology naturally extends to a Multi-Agent Reinforcement Learning (MARL) paradigm, enabling coordinated control across larger-scale, functionally heterogeneous interlogistics facilities.

7 Conlcusion

In this work, we presented a novel framework for parcel sorting in intralogistics, integrating RL (*i.e.*, PPO) with real-time machine vision (*i.e.*, YOLOv8-n).

The effective learning and deployment of high-performance sorting policies are shown by the provided experimental results, achieving 96.5% sorting accuracy under nominal conditions and maintaining 94% accuracy in more challenging, out-of-distribution scenarios. These results highlight the robustness of the learned policies and the viability of model-based RL for industrial sorting applications. Indeed, the proposed approach constitutes a scalable and flexible solution for autonomous parcel handling, advancing the state of the art in intelligent intralogistics automation.

Future directions include enhancing visual inference speeds, adapting the framework to broader sorting tasks via reward shaping, and extending the methodology to Multi-Agent Reinforcement Learning (MARL) settings for large-scale, distributed intralogistics environments.

Acknowledgment

The authors thank SAIET Engineering srl (especially Alberto Sarto) for their support during the experimental phase.

References

[1] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020. URL https://arxiv.org/abs/2006.05990.

[2] A. Ashok and N. Raja Ravi. Optimizing automated parcel sorting in logistics: Integration of closed-loop overflow management systems in existing sorting machine, 2024.

[3] T. P. Carvalho, F. A. Soares, R. Vita, R. d. P. Francisco, J. P. Basto, and S. G. Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019.

[4] R. Chen, W. Li, K. Lan, J. Xiao, L. Wang, and X. Lu. Fast adaptive binarization of qr code images for automatic sorting in logistics systems. *Electronics*, 12(2):286, 2023.

[5] K. Cobbe, J. Hilton, O. Klimov, and J. Schulman. Phasic policy gradient. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 2020–2027. PMLR, 2021. URL https://arxiv.org/abs/2009.04416.

[6] B. Dwyer, J. Nelson, T. Hansen, et al. Roboflow (version 1.0) [computer software], 2024. URL https://roboflow.com. Computer vision.

[7] C. Feng. Design of logistics sorting algorithm based on deep learning and sampling evaluation. *International Journal of Computational Intelligence Systems*, 17(1):82, 2024.

[8] J. Fottner, D. Clauer, F. Hormes, M. Freitag, T. Beinke, L. Overmeyer, S. N. Gottwald, R. Elbert, T. Sarnow, T. Schmidt, et al. Autonomous systems in intralogistics: state of the art and future research challenges. *Logistics Research*, 14(1):1–41, 2021.

[9] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep Reinforcement Learning that Matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[10] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. ICLR Blog Track, March 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. Blog article.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

[13] K. Muehlbauer, S. Meissner, and S. Baeuml. Current state of process automation in internal logistics: a germany-wide expert survey. *International Journal of Logistics Systems and Management*, 49(3):372–388, 2024.

[14] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab. Machine learning for anomaly detection: A systematic review. *Ieee Access*, 9: 78658–78700, 2021.

[15] G. Onori, A. A. Shahid, F. Braghin, and L. Roveda. Adaptive optimization of hyper-parameters for robotic manipulation through evolutionary reinforcement learning. *Journal of Intelligent & Robotic Systems*, 110 (3):108, 2024.

[16] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[17] C. Ren, H. Ji, X. Liu, J. Teng, and H. Xu. Visual sorting of express packages based on the multi-dimensional fusion method under complex logistics sorting. *Entropy*, 25(2):298, 2023.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[21] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Autonomous Robots*, 46(3):483–498, 2022.

[22] A. A. Shahid, Y. Narang, V. Petrone, E. Ferrentino, A. Handa, D. Fox, M. Pavone, and L. Roveda. Scaling population-based reinforcement learning with gpu accelerated simulation. *arXiv preprint arXiv:2404.03336*, 2024.

[23] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] J. P. Usuga Cadavid, S. Lamouri, B. Grabot, R. Pellerin, and A. Fortin. Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. *Journal of Intelligent Manufacturing*, 31:1531–1558, 2020.

[25] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, et al. Yolov10: Real-time end-to-end object detection. *Advances in Neural Information Processing Systems*, 37:107984–108011, 2024.

[26] M. Woschank, E. Rauch, and H. Zsifkovits. A review of further directions for artificial intelligence, machine learning, and deep learning in smart logistics. *Sustainability*, 12(9):3760, 2020.

[27] T. Wu, B. Zhu, R. Zhang, Z. Wen, K. Ramchandran, and J. Jiao. Pairwise proximal policy optimization: Harnessing relative feedback for llm alignment. *arXiv preprint arXiv:2310.00212*, 2023. URL https://arxiv.org/abs/2310.00212.

[28] X. Xu, Z. Xue, and Y. Zhao. Research on an algorithm of express parcel sorting based on deeper learning and multi-information recognition. *Sensors*, 22(17):6705, 2022.

[29] M. Yin, H. Zhang, C. Cai, M. Liang, and J. Liu. Omniveyor: An assembled logistics sorting system powered by reinforcement learning. *IEEE Transactions on Automation Science and Engineering*, 2025.

[30] W. Zaher, A. W. Youssef, L. A. Shihata, E. Azab, and M. Mashaly. Omnidirectional-wheel conveyor path planning and sorting using reinforcement learning algorithms. *IEEE Access*, 10:27945–27959, 2022.

[31] C. Zhai, Y. Li, H. W. Fu, H. Xu, L. Zhou, and W. Yao. The application of intelligent sorting equipment in warehouse logistics. In *2023 2nd International Conference on Mechanical Engineering and Power Engineering (MEPE)*, pages 15–19. IEEE, 2023.

[32] C. Zhou, A. Stephen, H. Li, L. H. Lee, and E. P. Chew. Information based approach for sort operation in logistic industry. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1356–1361. IEEE, 2017.