A Flexible Robotic Depalletizing System for Supermarket Logistics

Riccardo Caccavale, Pierluigi Arpenti, Gianmarco Paduano, Andrea Fontanellli, Vincenzo Lippiello, Luigi Villani, and Bruno Siciliano

***Abstract*—Depalletizing robotic systems are commonly deployed to automatize and speed-up parts of logistic processes. Despite this, the necessity to adapt the preexisting logistic processes to the automatic systems often impairs the application of such robotic solutions to small business realities like supermarkets. In this work we propose a robotic depalletizing system designed to be easily integrated into supermarket logistic processes. The system has to schedule, monitor and adapt the depalletizing process considering both on-line perceptual information given by non-invasive sensors and constraints provided by the high-level management system or by a supervising user. We describe the overall system discussing two case studies in the context of a supermarket logistic process. We show how the proposed system can manage multiple depalletizing strategies and multiple logistic requests.**

***Index Terms*—Control architectures and programming, logistics, intelligent and flexible manufacturing, AI-based methods, behavior-based systems.**

I. Introduction

DEPALLETIZING is the warehouse process where products are removed from the original shipping pallet and organized for storage. This activity is common in logistics, since goods of different types, dimensions and weight, are stocked on pallets in order to be suitably transported and delivered for distribution. The operation is typically performed manually by clerks or workers that have to remove a huge number of weighty products usually one by one. To optimize such fatiguing task, robotic depalletizing systems are usually deployed in medium or big companies [1]–[3] where the environment is structured and products to be manipulated are often homogeneous or standardized. On the other hand, in small realities like supermarkets, those robotic solutions are more challenging because products are typically stocked into non-standardized cases and structural changes in the logistic process are less convenient [4]. For instance, in [2], [3] the authors proposed an autonomous robot endowed with an innovative suction system that is able to pick from above boxes that are stacked in various poses and to place them on a conveyor belt. This method is effective in industrial scenarios but is harder to be integrated into less structured environments like supermarkets. Additional AI-enabled depalletizing systems have been proposed to address problems of motion planning [5] and safety [6]. Complementary, in this work, we are mainly focused on the issues of flexibility and adaptability. Analogously, other approaches have been proposed for palletizing tasks instead. In [7] a mobile manipulator has been proposed for autonomous picking, transporting and palletizing objects. This approach ensures more flexibility but only a specific type of objects is considered. Similarly, also in [8] authors proposed a flexible and easy-to-program robotic palletizer which is mainly designed for structured industrial environments. In this work our aim is to design a robotic system that is fully adaptable to different environmental conditions and depalletizing strategies in order to be easily integrated into preexisting logistic processes, minimizing cost and effort. In particular, our main contribution in this work can be summarized as follows:

- we design a sensorized robotic cell capable of picking and placing cases in different ways;
- we endow our framework with an executive system that is able to easily define hierarchically structured depalletizing tasks depending on the logistic context;
- we propose a depalletizing strategy for on-line scheduling of picking/placing sequences considering the configuration of the products on the pallet and multiple logistic requests.

II. System Design

In this section, we illustrate the overall architecture of the depalletizing system (see Fig. 1) describing its main components and functionalities. The system includes a sensorized robotic cell where a robot manipulator operates and different cases have to be moved or stored from the pallet to several target locations depending on logistic requests. The entire process is monitored and scheduled by the executive system that deploys a hierarchical representation of tasks to orchestrate both the robotic actions and the communications between the cell and the logistic management. This architecture is designed to be decoupled form the logistic flow to make the executive system able to suitably adapt the task execution and the depalletizing strategy to different logistic scenarios.

Manuscript received February 24, 2020; accepted May 24, 2020. Date of publication June 5, 2020; date of current version June 18, 2020. This work was supported by EU H2020 REFILLS Project under Grant Agreement 731590. This letter was recommended for publication by Associate Editor Xiang Zhong and Editor Jingang Yi upon evaluation of the reviewers' comments. *(Corresponding author: Riccardo Caccavale.)*

The authors are with the Department of Electrical Engineering and Information Technologies, Universitá di Napoli Federico II, 80138 Napoli, Italy (e-mail: ri.caccavale@gmail.com; pierluigi.arpenti@unina.it; gianmarcopaduano@gmail.com; g.a.fontanelli@gmail.com; vincenzo.lippiello@unina.it; lvillani@unina.it; siciliano@ieee.org).

This letter has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.3000427

2377-3766 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.

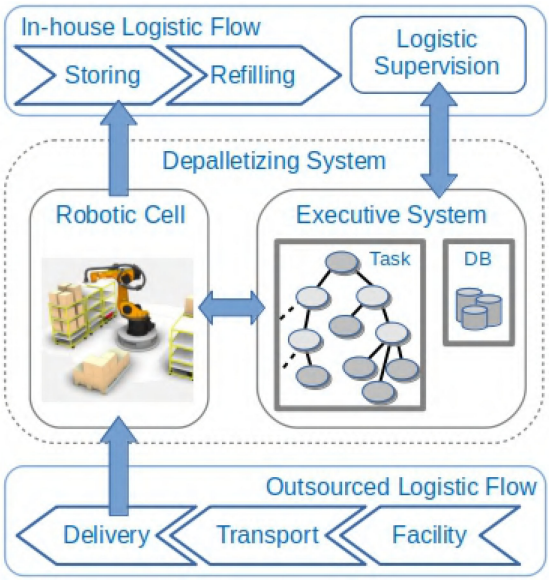


Fig. 1. Overview of the proposed architecture. The executive system supervises robotic task execution and connects the cell with the in-house logistic flow.

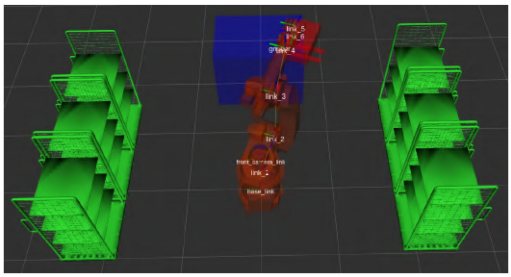


Fig. 2. CAD representation of depalletizing cell. The configuration includes 6 trolleys (green color) used to store the boxes taken from the pallet (blue color) by an industrial manipulator (red color).

A. Design of the Robotic Cell

In order to facilitate the integration in the logistic flow of supermarkets, the proposed robotic cell is designed to be flexible and adaptable to different logistic contexts. In the considered application typical of supermarkets' backroom, the products are collected into cases that have to be stored from the original pallet to a set of generic target locations (like trolleys or shelves). The cell includes a robot manipulator endowed with a suction-based gripping end-effector and an RGB-D camera allowing detection/recognition of the cases and the estimation of their pose. Similar perception systems can be found in [9], [10]. The robotic cell can be split in two areas: an inner area (the inner part of the robot workspace) where the robot operates and an outer area (the outer part of the robot workspace) where products are stored to be eventually taken from external agents and carried to the next steps of the logistic flow. A representation of the proposed cell is depicted in Fig. 2. In our setting, the robot manipulator is placed in the center of the inner area and multiple storing locations can be placed all around it, in the outer area. The pallet is positioned in front of the robot, allowing the manipulator to reach the cases and to move them toward the target locations, or to move cases from one location to another, while the RGB-D camera is placed between the robot and the pallet, front-facing the latter. Notice that the cell configuration allows the cases to be grasped in different ways: either vertically (from the above) or horizontally (from the sides). This feature is particularly relevant in supermarket logistics because several types of cases must be taken from the sides, and storing repositories like shelves or trolleys are usually filled horizontally.

B. Executive System

To control the robotic cell and to on-line adapt picking/placing strategies depending on the observations and the logistic context, we deploy an executive system similar to the one proposed in [11]. Our executive system exploits an HTN-like representation of robotic tasks and operators including symbolic constraints and effects [12]. More specifically, each task is hierarchically defined by a set of predicates $\mathbf{schema}(m, l, p)$, where m is the name of the task, l is a list of subtasks associated with preconditions, while p represents a postcondition used to check the accomplishment of the task. An example of predicate is given below:

$$\begin{aligned} &\boldsymbol{s}chema(task(x_1, x_2 \ldots, x_n), \langle \\ &\quad (subtask_1(x_{11}, x_{12}, \ldots), precond_1), \\ &\quad \ldots, \\ &\quad (subtask_k(x_{k1}, x_{k2}, \ldots), precond_k), \\ &\quad \rangle, postcond). \end{aligned}$$

A schema stands for a parametric task and can be either abstract or concrete, where abstract schemata are tasks that need to be further decomposed into sub-tasks, while concrete schemata are atomic primitives to be executed. Notice that, in our task representation, both robotic actions and the communications (like sending/receiving messages to/from external systems or users) are associated with primitives that can be scheduled and executed by the executive system. Schemata are also endowed with context-specific preconditions and postconditions that are continuously evaluated during execution. An example of a storing task is proposed below:

$$\begin{aligned} &\mathbf{schema}(store(Box), \langle \\ &\quad (take(Box), true), \\ &\quad (check(Box), Box.taken \wedge \neg Box.known), \\ &\quad (query(Box), Box.known \wedge \neg Box.target), \\ &\quad (leave(Box), Box.taken \wedge Box.target) \\ &\quad \rangle, Box.stored). \end{aligned}$$

The task $store(Box)$ represents the process of picking and placing a box from the pallet to a target location. The task is decomposed into 4 sub-tasks: the box is firstly grasped from the pallet ($take(Box)$, always enabled) and moved to a barcode reader ($check(Box)$) if not recognized by the vision system ($Box.taken \wedge \neg Box.known$). When the box is correctly recognized ($Box.known$), if the storing position is unknown ($\neg Box.target$) the system asks for it ($query(Box)$) and the box is finally placed in the storing position ($leave(Box)$). The whole task is considered accomplished when the box is stored into the target place ($Box.stored$). In order to be monitored and

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.

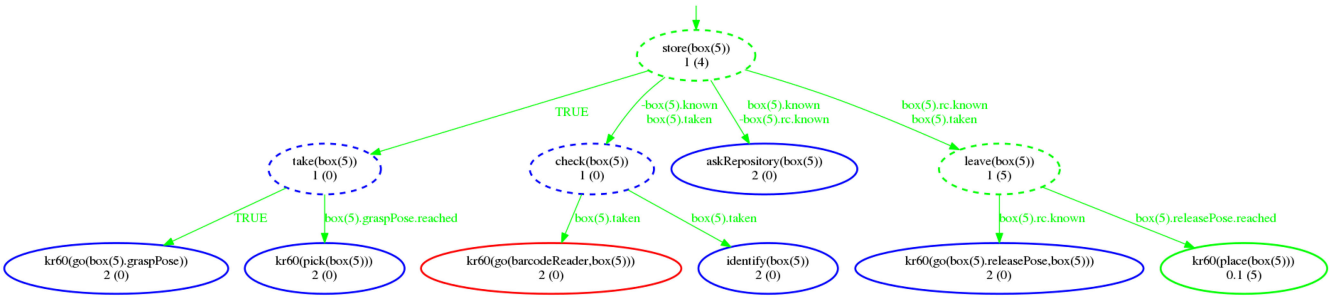


Fig. 3. Running example of a $store$ task. Preconditions are attached to the edges while dashed and continuous ovals are for abstract and concrete tasks respectively. In this representation green nodes/preconditions are enabled/satisfied, blue nodes are accomplished and red nodes are disabled.

executed, tasks are allocated on-line from the repository to the executive system. This process generates for each task an annotated tree whose nodes are grounded schemata (i.e. the parameter Box of the previous example is now replaced with a box id) and edges are parental relation between them. During the execution, we associate to each node of the tree a state that can be: $enabled$ if all precondition along the branch are satisfied, $disabled$ if at least one precondition is not satisfied and $accomplished$ if the postcondition is satisfied. In particular, enabled nodes of the tree that are associated to motion or communication patterns (concrete nodes) are directly executed by means of robot movements or sending/receiving messages. An example of allocated $store$ task is proposed in Fig. 3. The task includes 4 abstract subtasks (dashed ovals), which are further decomposed into 7 concrete actions (continuous ovals). Nodes that are identified by $kr60$ functors are associated with robotic actions, while the others are communication actions. Notice that this structured and hierarchical representation of tasks can be easily adapted to different logistic contexts: robotic primitives can be exploited as building blocks [13] to compose more complex tasks or to adjust task execution.

C. Selection and Grasping

When a new pallet is transported into the cell, the system is provided with a set of boxes/products B_{pall} that are placed on the pallet, while each element $b \in B_{\text{pall}}$ is associated with box-specific information like barcode, weight and dimensions. Since the configuration and the pose of each element is not known in advance, the sequence in which boxes are taken have to be decided on-line depending on perceptual information. An effective heuristic for depalletizing is to take boxes from the upper part to the base (top-to-down) and from the sides to the center of the pallet (sides-to-center). This way, the base of the pallet is always larger then the summit and the stability of the structure is not compromised. Moreover, since the pallet is positioned in front of the robot, we slightly prioritize frontal boxes to support manipulability. Following the above criteria, we define a suitable function $h(b)$ that associates a priority to boxes considering their positions on the pallet, defined as:

$$h(b) = \frac{1}{7}|y_b| + \frac{2}{7}(x_{\max} - x_b) + \frac{4}{7}z_b. \qquad (1)$$

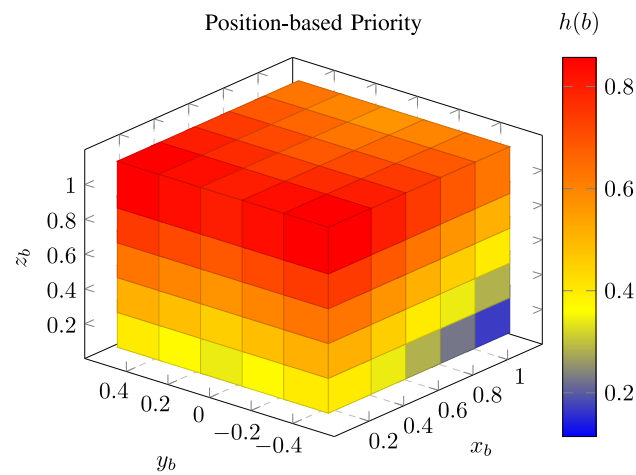


Fig. 4. Color-based representation of the priority given the 3D position of the boxes. Assuming the robot in the origin, colors from red to blue represent decreasing priority.

In Eq. (1), the components of the center-of-mass of the box $[x_b, y_b, z_b]$ are weighted to prioritize boxes that are higher positioned ($\frac{4}{7}z_b$), closer to the robot ($\frac{2}{7}(x_{\max} - x_b)$) and closer to the edges ($\frac{1}{7}|y_b|$). Here we strongly prioritize higher positioned boxes to reduce the size of the higher layers of the pallet, then we consider boxes from the sides to the center giving more emphasis to the most accessible ones (frontal boxes).

A graphical representation of the function $h(b)$ is illustrated in Fig. 4. Here, boxes that are placed in red areas are prioritized by the robotic system, while the others (yellow/blue areas) are taken at a later time. Besides their position, the priority of the boxes is also affected by the requests that can be asynchronously provided to the system in response to specific logistic necessities. In this case, if a requested box is recognized by the perceptual system, the score of all perceived boxes is adjusted to facilitate the selection of the requested one. We can define the output of the perceptual system as a subset of boxes $B_{rec} \cup B_{det} \subseteq B_{\text{pall}}$ that is partitioned into a set of recognized boxes (B_{rec}) with their associated barcodes and a set of detected boxes (B_{det}) that have been perceived by the system but not recognized (e.g. boxes with no textures and occluded barcode). Moreover, we assume to have in input a set of requested boxes $R \subseteq B_{\text{pall}}$ were each box $b \in R$ is associated to a priority $p(b) \in [0, 1]$ denoting the importance

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.

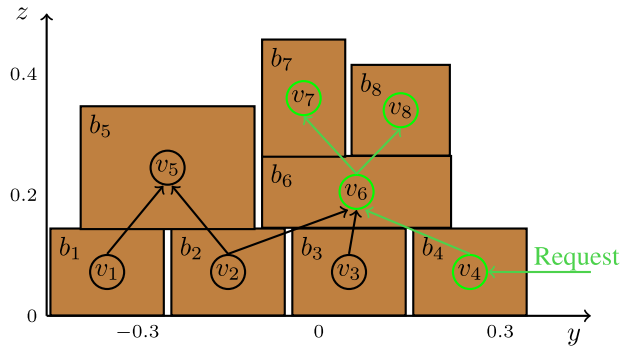


Fig. 5. Example of the graph G (arrows and circles) overlapping a group of boxes (brown rectangles). Each box b_i is associated to the corresponding vertex v_i of the graph (inner circles) while arrows are relation between them. When the box b_4 is requested the score of the associated sub-graph $T(b_4)$ (green arrows/nodes) is updated.

of such request. If a requested box is reachable and graspable by the robot (namely, in the frontal-upper part of the pallet) it can be taken directly, otherwise, if the box is in the middle of the pallet or partially occluded by other boxes, the system has to adapt the way boxes are taken in order to free the requested one. We define an additional priority function $l(b)$ that drives the system to unstack and store the requested boxes. To this end, given the pallet configuration, we create a graph structure where each vertex is a box while edges are the dependencies between boxes. More formally, we define $G = (E, V)$ along with a mapping function $v : B_{rec} \cup B_{det} \rightarrow V$ associating boxes and vertices, where for each couple of boxes b_1, b_2 such that b_2 is placed on or occluding b_1 there exists an edge $(v(b_1), v(b_2)) \in E$. In G we can define the subset of vertices $T(b) \subseteq V$ for a perceived box b which contains all the vertices that are reachable from $v(b)$. This way, given a requested box b_r and the subset $T(b_r)$ the priority of the request $p(b_r)$ can be propagated to all the boxes b that are blocking b_r (i.e. $v(b) \in T(b_r)$). An example of propagation is shown in Fig. 5, while the function $l(b)$ for a generic perceived box is defined in the following equation:

$$l(b) = \max_{x:v(b)\in T(x)} (p(x)). \tag{2}$$

In this way, if a box b is blocking one or more requested boxes, it inherits the priority of the most important one. Finally, the score of a box $s(b)$ can be defined as a weighted sum of both priority functions, namely:

$$s(b) = wl(b) + (1 - w)h(b), \tag{3}$$

where the weight w can be suitably regulated to emphasize logistic-based or position-based depalletizing strategies. Notice that learning approaches, similar to [13], [14], can be easily deployed to balance the two functions following human demonstrations. The whole process of box selection is described in Algorithm 1. In every row the system collects the lists of requested (line 1) and perceived (line 2) boxes and initialize a queue that will be used to store the boxes sorted by score (line 3). The list of perceived boxes is also exploited to generate the graph of dependencies G (line 4). Each perceived box b is then associated to its position-based priority $h(b)$ (lines 6–8) and to its

Algorithm 1: The Selection Process is Invoked When no Storing Tasks are Allocated.

1: **Procedure** BOX SELECTION
2: get requested $R = \{r_1, r_2, \ldots, r_m\}$
3: get boxes $B = \{b_1, b_2, \ldots b_n\}$
4: init queue $Q \leftarrow \emptyset$
5: create graph G from boxes B
6: **for** $b \in B$ **do**
7: get b pose $[x_b, y_b, z_b]$
8: $h(b) = \frac{1}{7}|y_b| + \frac{2}{7}(x_{\max} - x_b) + \frac{4}{7}z_b$
9: get vertex $v(b)$ from G
10: $l(b) = \max_{x:v(b)\in T(x)}(p(r))$
11: $s(b) = wl(b) + (1 - w)h(b)$
12: insert b into Q ordered by $s(b)$
13: **end for**
14: $found \leftarrow false$
15: **while** $\neg found$ **do**
16: pop box b with max $s(b)$ from queue Q
17: compute grasping poses Gp from b
18: **if** exists a reachable pose $p \in Gp$ **then**
19: set target $t \leftarrow b$
20: set $found \leftarrow true$
21: **end if**
22: **end while**
23: allocate $store(t)$ task
24: **end procedure**

request-based priority $l(b)$ (lines 9–10). Finally, both priorities are fused into the score $s(b)$ (line 11) and the box is inserted into the queue Q sorted by s (line 12). In the second part of the algorithm, until a graspable box is found (lines 14–15), boxes are taken form the queue (line 16) and if a valid grasping point exists for it (lines 17–18) the box is selected to be depalletized (lines 19–20) and the associated storing-task is allocated to be monitored and executed by the executive system (line 23).

III. Case Study

In this section, we propose two case studies in a supermarket scenario. In the first case study, we perform a realistic simulation where multiple pallet configurations and logistic requests are randomly generated and the proposed framework has to suitably adapt the depalletizing sequence to them. In the second case, we show the system at work in a realistic supermarket depalletizing task, where cases are stored from the pallet to compartments of trolleys taking into account of the unknown pallet configuration and of the on-line request for a prioritized case. In both experiments, we assume that storing positions of all cases on the pallet are already available into the supermarket database, while asynchronous requests randomly arises during the execution.

A. Experimental Set-up

The testing environment is depicted in Fig. 6. The depalletizing cell covers a 5 × 5 meters area and includes:

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.

Fig. 6. Depalletizing cell.

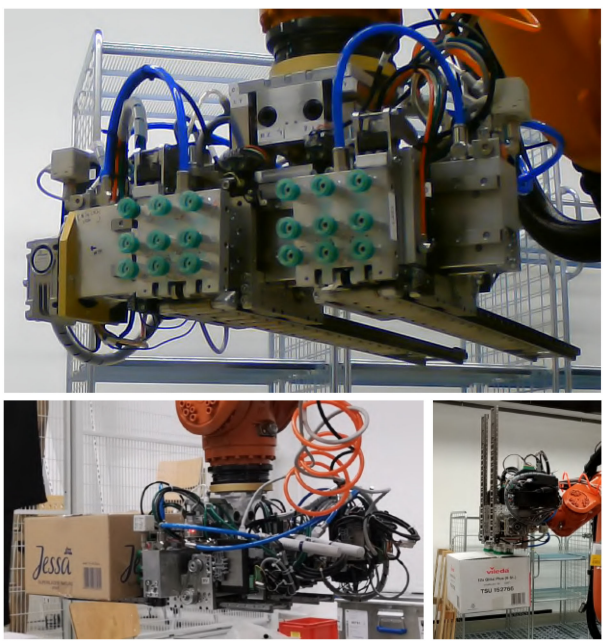


Fig. 7. Snapshots of the gripper including a frontal view (up), and horizontal (down-left) and vertical (down-right) grasping procedures.

- 7 locations for box storing: 6 movable trolleys on the sides and 1 fixed shelf on the rear part of the cell;
- a 6-DOF Kuka kr60 robotic arm with 60 kg of payload and a 2 meters radius workspace;
- a reconfigurable suction-based gripping system endowed with 2 independent modules, allowing vertical and horizontal grasping of boxes of variable size (see Fig. 7);
- a vision system for boxes recognition/detection and their pose estimation exploiting an Intel Realsense RGBD camera.

The software architecture is integrated in ROS-kinetic, while robot motion-planning is performed by means of the OMPL library [15].

B. Case Study 1: Simulation

The aim of this case study is to evaluate the effectiveness of the selection method proposed Section II-C. In particular, we set-up a RViz simulation where both the pallet configuration and the sequence of requests are randomly generated. The system has to depalletize cases following their scores, while avoiding collisions due to occlusions. The total volume of each random configuration is fixed to 1.6 $\times$ 0.8 $\times$ 1 meters, while the number, position and dimensions of cases are randomly generated. For each configuration, the system is deployed in two modalities: the first one is a baseline, where $s(x) = h(x)$, then the selection process only depends on the position of boxes on the pallet; instead, in the second modality, both components are considered as in Eq. (3). Here, we are interested in studying how the average number of actions needed to fulfill a request changes, considering different settings of w.

To this end, we selected 5 values for the weights [0.1, 0.3, 0.5, 0.7, 0.9] and, for each setting, we generated 50 random configurations of cases/requests. Considering both modalities, we performed a total of 500 simulated executions. Table I illustrates the average and the deviation of the number of cases stored before the requested one is selected. Here, it is possible to notice that, as expected, in the second modality (where both $h(x)$ and $l(x)$ are deployed) the number of actions needed to accomplish a request is always below the baseline modality (where only $h(x)$ is considered). Moreover, as depicted in Fig. 8, the average action-reduction grows with the increment of the weight, becoming stable around 5.5 actions for $w \geq 0.5$. On the other hand, the request-oriented behavior of the system may lead to unstable configurations of the cases. In Fig. 9, we show an example of a random configuration (a) that is partially depalletized following the two modalities. After 20 steps (b-c) the $h(x)$-only modality lead to a compact configuration, while in the $h(x)$ + $l(x)$ modality there are some unstable pillars of boxes that are more likely to collapse.

TABLE I
NUMBER OF ACTIONS NEEDED TO SATISFY THE REQUESTS FOR BOTH MODALITIES (250 RUNS FOR EACH MODALITY)

weight		$h(x)$-only	$h(x)$+$l(x)$
$w = 0.1$	avg	12.74	11.14
	std	9.91	9.89
$w = 0.3$	avg	13.10	9.18
	std	10.02	8.92
$w = 0.5$	avg	12.81	7.81
	std	9.19	7.34
$w = 0.7$	avg	13.03	7.69
	std	9.73	7.45
$w = 0.9$	avg	12.84	7.34
	std	9.47	7.13

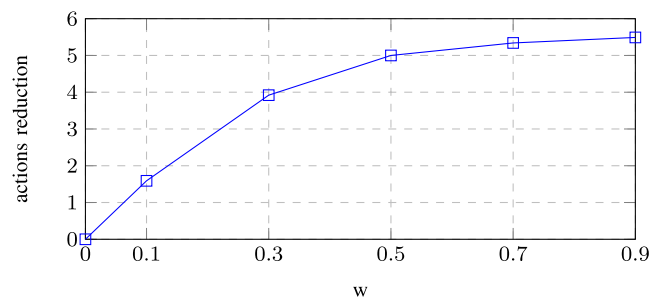
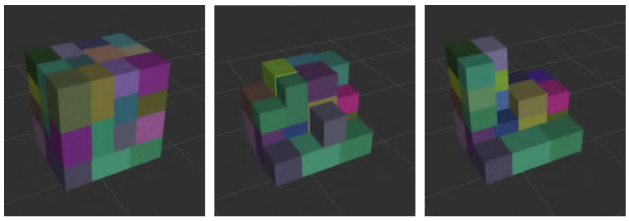


Fig. 8. Average improvement on request accomplishment. The increment of w reduces the average number of actions needed to fulfill a request.

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.



(a) Random (b) $h(x)$-only (c) $h(x)$+$l(x)$

Fig. 9. Random configuration of cases (a) along with the state of the execution using $h(x)$ only (b) and the full version of $s(x)$ (c) after 20 steps.

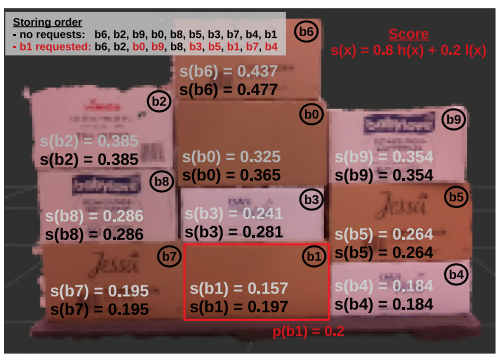


Fig. 10. Pallet configuration during real-world experiment: The labels on each box are the values of the scores before (gray equations) and after (black equations) that box $b1$ is requested; in the left upper part of the figure, the differences in the storing sequence are highlighted in red.

C. Case Study 2: Real Scenario

In this case study, the system is tested at work in a real supermarket scenario. A simplified configuration of the pallet is considered, with 10 cases placed in a single frontal line (see Fig. 6). The task is to store all the cases on a trolley (first on the right in figure) by using the full version of our scoring function in Eq. (3). For this experiment we set $w = 0.2$ to induce a conservative strategy (unstable configurations are rare with this setting) but still considering requests. In particular, $b1$ is designed as the requested box with a low priority ($p(b1) = 0.2$). This is intended to show how even less prioritized requests affect the behavior of the system. Fig. 10 shows the pallet configuration seen from the RGB-D frontal camera along with the priorities before and after the request. As shown in the figure (up), by following the position-based priority only (no requests), the box $b1$ would be the last one to be stored, while this box is stored two actions before with the request-based priority enabled. In this set-up, for safety reasons, we have limited the robot velocity to the 4% of the maximum allowed velocity. The system takes $19'56''$ to store the requested box, and each storing task takes an average of $2'28''$ per box.

IV. Conclusion

In this letter, we proposed a robotic framework for flexible depalletizing in supermarkets, designed to be adapted to different logistic necessities. In particular, our system allows to pick cases or boxes of different dimensions enabling either vertical and horizontal grasping procedures. To facilitate the integration of the system into preexisting logistic processes, we also deployed an executive system for structured task execution that permits a modular definition of hierarchical depalletizing tasks that can be used as building blocks to define complex or novel activities. Finally we defined a heuristic to schedule depalletizing processes that can be suitably regulated to mediate between pallet stability and possible requests provided by external supervisory agents. We tested the proposed framework in a simulated scenario, analyzing its performance with multiple simulated pallets; finally we discussed the system at work in a real robotic depalletizing cell designed for a supermarket environment. The focus of this work was to define a flexible and adaptive architecture suitable for depalletizing tasks in supermarkets; extending the framework to palletizing or to industrial scenarios will be the topic of our future research. Moreover, we plan to investigate more complex environmental conditions along with more sophisticated task structures including safety constraints and fault detection/correction.

References

[1] D. Katsoulas and D. I. Kosmopoulos, "An efficient depalletizing system based on 2d range imagery," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, vol. 1, pp. 305–312.

[2] H. Nakamoto, H. Eto, T. Sonoura, J. Tanaka, and A. Ogawa, "High-speed and compact depalletizing robot capable of handling packages stacked complicatedly," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 344–349.

[3] J. Tanaka, A. Ogawa, H. Nakamoto, T. Sonoura, and H. Eto, "Suction pad unit using a bellows pneumatic actuator as a support mechanism for an end effector of depalletizing robots," *ROBOMECH J.*, vol. 7, no. 1, 2020.

[4] W. Echelmeyer, A. Kirchheim, and E. Wellbrock, "Robotics-logistics: Challenges for automation of logistic processes," in *Proc. IEEE Int. Conf. Autom. Logistics*, 2008, pp. 2099–2103.

[5] T. Sakamoto, K. Harada, and W. Wan, "Real-time planning robotic palletizing tasks using reusable roadmaps," *J. Robot., Netw. Artif. Life*, vol. 6, no. 4, pp. 240–245, 2020.

[6] M. Jocas, P. Kurrek, F. Zoghlami, M. Gianni, and V. Salehi, "Ai-based learning approach with consideration of safety criteria on example of a depalletization robot," in *Proc. Design Soc.: Int. Conf. Eng. Des.*, 2019, vol. 1, pp. 2041–2050.

[7] R. Krug *et al.*, "The next step in robot commissioning: Autonomous picking and palletizing," *IEEE Robot. Autom. Lett.*, vol. 1, no. 1, pp. 546–553, Jan. 2016.

[8] F. M. Moura and M. F. Silva, "Application for automatic programming of palletizing robots," in *Proc. IEEE Int. Conf. Auton. Robot Syst. Competitions*, 2018, pp. 48–53.

[9] D. Holz, A. Topalidou-Kyniazopoulou, J. Stückler, and S. Behnke, "Real-time object detection, localization and verification for fast robotic depalletizing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 1459–1466.

[10] M. Schwarz, A. Milan, A. S. Periyasamy, and S. Behnke, "RGB-D object detection and semantic segmentation for autonomous manipulation in clutter," *Int. J. Robot. Res.*, vol. 37, no. 4-5, pp. 437–451, 2018.

[11] R. Caccavale and A. Finzi, "Flexible task execution and attentional regulations in human-robot interaction," *IEEE Trans. Cognitive Developmental Syst.*, vol. 9, no. 1, pp. 68–79, Mar. 2017.

[12] D. S. Nau *et al.*, "SHOP2: An htn planning system," *J. Artif. Intell. Res.*, vol. 20, pp. 379–404, 2003.

[13] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Auton. Robots*, vol. 43, no. 6, pp. 1291–1307, 2019.

[14] R. Caccavale and A. Finzi, "Learning attentional regulations for structured tasks execution in robotic cognitive control," *Auton. Robots*, vol. 43, no. 8, pp. 2229–2243, 2019.

[15] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

Authorized licensed use limited to: Universita degli Studi di Napoli Federico II. Downloaded on June 20,2020 at 11:46:57 UTC from IEEE Xplore. Restrictions apply.