The International Journal of Advanced Manufacturing Technology (2023) 125:1965–1985
https://doi.org/10.1007/s00170-022-10601-9

APPLICATION

Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring

Ander Iriondo[1,2] · Elena Lazkano[2] · Ander Ansuategi[1] · Ane Fernandez[1] · Iñaki Maurtua[1]

Received: 15 June 2022 / Accepted: 24 November 2022 / Published online: 12 January 2023

Abstract
This paper describes the dynamic mosaic planning method developed in the context of the PICKPLACE European project. The dynamic planner has allowed the development of a robotic system capable of packing a wide variety of objects without having to adjust to each reference. The mosaic planning system consists of three modules: First, the picked item monitoring module monitors the grabbed item to find out how the robot has picked it. At the same time, the destination container is monitored online to obtain the actual status of the packaging. To this end, we present a novel heuristic algorithm that, based on the point cloud of the scene, estimates the empty volume inside the container as empty maximal spaces (EMS). Finally, we present the development of the dynamic IK-PAL mosaic planner that allows us to dynamically estimate the optimal packing pose considering both the status of the picked part and the estimated EMSs. The developed method has been successfully integrated in a real robotic picking and packing system and validated with 7 tests of increasing complexity. In these tests, we demonstrate the flexibility of the presented system in handling a wide range of objects in a real dynamic packaging environment. To our knowledge, this is the first time that a complete online picking and packing system is deployed in a real robotic scenario allowing to create mosaics with arbitrary objects and to consider the dynamics of a real robotic packing system.

Keywords Bin-packing · Logistics · Manipulation · Online mosaic planning · Pick and place

1 Introduction

Object placing is one of the most challenging tasks in robotic manipulation and contrary to simple pick-and-drop approaches, object placing methods try to find the best location to leave the picked part. One of the well-known applications of object placing algorithms is bin-packing, that tries to fit the picked parts in tight spaces with the goal of optimising the space inside the bin. In industrial environments such as warehouses or distribution centres, efficient and automated bin-packing methods are highly demanded, due to the cost of storage and transportation of oversized containers. Currently, in most of those centres, the packing operations are carried out only by humans, leading to error-prone and inefficient packing solutions.

This work is focused on the packing system developed in the context of the PICKPLACE[1] European project which seeks to develop a flexible, safe and dependable picking and packing system considering the real use-cases of *ULMA Handling Systems* (UHS) and *Türk Otomobil Fabrikasi A.Ş.*(TOFAŞ) end-users.

In the distribution centres installed by UHS, the order pick-and-packaging (the process of collecting items to create a package for shipment) is the last and most challenging task in the warehouse automation. Depending on the type of installation, the warehouse operator takes

✉ Ander Iriondo
ander.iriondo@tekniker.es

Elena Lazkano
e.lazkano@ehu.eus

Ander Ansuategi
ander.ansuategi@tekniker.es

Ane Fernandez
ane.fernandez@tekniker.es

Iñaki Maurtua
inaki.maurtua@tekniker.es

1 Department of Autonomous and Intelligent Systems, Tekniker-Basque Research and Technology Alliance (BRTA), Iñaki Goenaga, 5, Eibar, 20600, Gipuzkoa, Spain

2 Robotics and Autonomous Systems group (RSAIT), Department of Computer Science and Artificial Intelligence, University of the Basque Country (UPV/EHU), Pº Manuel Lardizabal,1, Donostia-San Sebastián, 20018, Gipuzkoa, Spain

[1]PICKPLACE:https://pick-place.eu/

multiple types of goods at the end of the line and places them in target boxes. In the automotive sector, TOFAŞ deals with more than 63.000 different spare parts that are sent to different dealers in manually prepared packages. This monotonous packing process is currently held in 90% by humans due to the difficulty of replacing human dexterity to handle parts with high variability in size, shape, weight and material.

The feasibility of fully automated systems is currently limited by many factors such as the huge variability of items, system robustness and industrial throughput requirements. Human-robot collaboration seeks to combine the robot's efficiency and the human dexterity to optimise the packing process.

Bin-packing solutions are mainly differentiated depending on the level of knowledge available in the application. Traditionally, bin-packing methods deal with ideal conditions in simulation and the identity and the set of items to be packed, as well as their properties are known. These ad hoc solutions assume that the parts can be packed in any orientation and omit the presence of automatic picking/packing systems such as robots. In other approaches that consider automatic picking/packing systems, they assume that the objects are known and that are always grasped in the same way. Thus, in both cases, the packing order of the items and the mosaic can be calculated offline. The main disadvantage of this kind of methods is that they ignore the uncertainties introduced by a real picking/packing system and assume that the process is ideal. In fact, in dynamic scenarios such as big warehouses where thousands of known and even unknown references need to be managed, those solutions fail, and more flexible systems are required.

Taking into account the high variability of the objects considered in the project as well as the need to handle unknown objects, an affordance-based flexible grasping system was developed in the context of the project [1]. Following this approach, the system predicts grasping points without relying on the identity of the parts but only focusing on the shape, colour and texture of them. Having such a flexible grasping system implies that items are not identified before being picked, and thus the grasping point prediction varies in each execution of the algorithm. Indeed, the way that each object is picked has a big effect in the place operation and, therefore, the grasped item has to be monitored online. Additionally, when dealing with a huge amount of references with highly variable shapes and textures, small movements in the outbound box produced by the human or the warehouse system itself may cause the products to drop. Therefore, online monitoring of the outbound box becomes necessary to avoid collisions.

The patented palletising solution of UHS, dubbed IK-PAL, is composed of a palletising robot and the algorithm that calculates the palletisation mosaic. This system has been successfully deployed in warehouses of big companies such as EROSKI.[2] However, this first version of the mosaic planning algorithm was thought to work with regular items and assuming that the parts would remain static after being packed in the mosaic. Nevertheless, the increasing market of the e-commerce nowadays demands flexible solutions to handle a high variety of items with highly variable shapes. Introducing irregular items to the system makes it impossible to assume that the items will remain in the ideal position once they have been packed, and arises the need for monitoring the outbound box.

In this work, several novelties are introduced to evolve IK-PAL's mosaic planning algorithm into a flexible and dynamic planner. As the grasping point detection system developed in the project is agnostic to the objects' identity and the predicted grasping points are not predefined, we first developed a system to estimate online the dimensions and the orientation of the picked parts. Second, we designed a polynomial-order heuristic method to monitor online the empty volume inside an outbound box. This method computes the free volume in the form of empty maximal spaces (EMSs), which serves as input to the dynamic IK-PAL mosaic planner. With this new method, the grabbed items and outbound containers are monitored online, and the packing pose for each picked part is estimated considering the actual status of the packaging. Therefore, introducing these novelties to the system, we are able to dynamically handle dropped objects and to avoid collisions when items move around inside the bin.

The developed dynamic mosaic planner was finally integrated with an automatic picking system in a real robotic scenario. To the best of our knowledge, this is the first time that a complete online picking and packing robotic system is developed that allows creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system.

2 Literature review

Object placing, along with picking, have been core problems since early days of robotics. Particularly, the problem of fitting a set of objects inside a destination bin optimising the used space has traditionally attracted a lot of interest. Most of the proposed solutions try to pack regular shaped objects such as boxes with non-overlapping constraints.

Traditionally, the majority of the approaches assume that the set of items to be packed is known, and therefore, the packing mosaic is calculated offline. Currently, there are exact solutions that optimally solve the 3D bin-packing problem offline. A popular method was presented by

[2]EROSKI: https://www.eroski.es/

Martello et al. in [2] and [3]. It uses a branch-and-bound algorithm to optimally solve the problem. This algorithm was subsequently optimised by Boef et al. in [4]. The bin-packing problem belongs to the class of NP-hard methods and, therefore, it cannot be solved in polynomial time. Consequently, much research has been done in approximate methods to solve the problem to near optimality [5]. To mention some, Jakobs et al. first introduced the well known *bottom-left* heuristic in [6]. The *best-fit decreasing* algorithm was developed and analysed by Johnson et al. in [7]. However, these methods were designed to work in 1D or 2D.

The use of metaheuristic-based algorithms led to good approximate solutions for the orthogonal 3D bin-packing problem [8]. In [9], Crainic et al. efficiently solved the orthogonal 3D bin-packing problem applying a two-level *tabu-search* method. Additionally, Faroe et al. applied the *guided local search* (GLS) to solve the same problem [10]. Furthermore, genetic algorithms were also successfully applied in [11] to solve orthogonal 2D and 3D bin-packing problems. State-of-the-art learning-based approximate strategies used *deep reinforcement learning* (DRL) to optimise the packing of orthogonal parts [12]. Nevertheless, these methods were only designed for the orthogonal bin-packing and are only practical when dealing with boxes. Moreover, in these works, it was assumed that the parts could be packed without restrictions in the orientation. Nonetheless, when the items are packed by a robot, the picking location of the parts limits the placing and not all orientations are feasible.

More recently, the bin-packing of both regular and irregular multi-reference parts has become a need in intralogistic applications, particularly caused by the growth of the e-commerce. When the shape of the items is irregular, it is not feasible to find an optimal solution to the problem as the search space is infinite, and only can be solved with approximate solutions. For instance, Zhao et al. used a metaheuristic algorithm to estimate a near-optimal packing mosaic for irregular objects [13]. In this work, the items were represented as meshes and the empty space as a bounding box of the bin. However, the mosaic was calculated offline and it was not considered that objects could fall, which is likely to occur in the real system using irregular parts. Besides, the method was time demanding and could not be used in real time. In [14], the authors proposed an heuristic-based method that considered the offline packing of geometrically complex 3D objects with stability constraints. On the one hand, the picked part was represented as a 2D heightmap that was generated using ray-casting. On the other hand, the availability of the outbound box was monitored using a second 2D heightmap that was updated whenever a new object was placed. Although the packing method checked the status of the outbound box heightmap to look for a stable packing pose, falls were not considered since this process was executed offline, before picking any item. Besides, the system was only assessed in simulation.

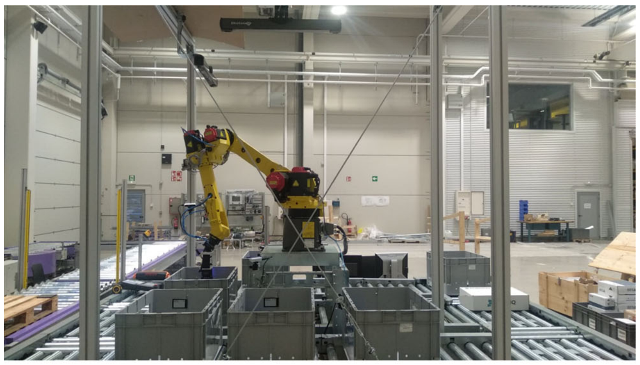
The stability of the packed mosaic is guaranteed dealing with regular mono-reference items under controlled scenarios. However, this is not the case when it comes to multi-reference applications, particularly when irregular parts are also considered. Therefore, when each object is packed, it cannot be assumed that the constructed tote will remain constant and hence, the mosaic must be calculated online. Indeed, it is required to analyse the status of both the picked part and the packaging at each iteration due to the following reasons:

1. When picking a huge number of multi-reference items ad hoc grasping solutions usually fail and flexible solutions are needed. Flexible grasping systems are not able to guarantee that the items will always be picked in the same way. Therefore, the packing position cannot be calculated until the object is picked. The picking pose limits the possible packing orientations.
2. When dealing with irregular parts, objects are likely to fall off or move around after being packed. Thus, the theoretical and real mosaics differ and the destination boxes need to be monitored online whenever a new product is packed.

Regarding online bin-packing methods, Ha et al. proposed a bin-packing heuristic algorithm to pack orthogonal objects online at arrival time without any prior knowledge about the items to be packed [15]. This work, however, did not consider a picking system that would restrict the packing orientations. In fact, it was only implemented for ideal setups in simulation. For instance, the framework developed by Wang et al. [16] dealt with the packing of a known set of objects arriving in a non-deterministic order. The proposed solution was composed of an offline planner and a verification algorithm that checked the feasibility of all permutations of arrival orders. Although the system handled both regular and irregular objects, in that work object falls were not considered, and the status of the destination box was not monitored. Additionally, authors claim that their work was only practical with a small subset of objects. Hong et al. went one step further and solved the irregular 3D object bin-packing in an online manner [17]. In that approach, a depth camera was used to estimate the area (highest contour) of the picked object. To that end, the depth camera was located below the picked part and it was aligned with the end effector of the robot. Concerning the destination box monitoring, the status of the bin was represented as a 2D depth-map and an analysis of the available contact-free 2D positions was performed, using the contour information of the picked part. Additionally, some other criteria such as placing the objects in the lowest possible position or bringing things together were applied. However, the authors did not use a real picking and packing system (only simulated some manipulation error) which could have a large effect on the structure of the final mosaic. Recently, learning algorithms such as DRL have been successfully applied to solve the online bin-packing problem with orthogonal parts, for instance, in [18]. This approach, however, only dealt with orthogonal items, it lacked an automatic picking system and it was only tested under ideal simulated conditions. Zhao et al. also propose a DRL-based packing system for orthogonal objects that has been tested in a real robotic system [19]. In that work, once the robot left the picked object in the target box, they did not monitor it online and assumed that the object would be ideally placed.

Summing up, most of the works in the literature used orthogonal items, assumed that the packing process was ideal and, therefore, they did not provide online monitoring of the status of both the picked part and the outbound box. However, this is crucial particularly in real unstructured scenarios where the packing conditions are variable and when irregular multi-reference parts are considered in the system. Additionally, the fact that a huge amount or even unknown items need to be handled in intralogistic centres requires to have a flexible grasping system. Indeed, the grasping pose has a large effect in how the part can be packed, but none of the analysed works uses a real picking system. In this work, we present the online packing solution developed in the context of the PICKPLACE European project that is able to pack both regular and irregular items considering the packing status at each iteration. The contributions of the paper are as follows:

- Firstly, we develop an online monitoring system for the picked part. In PICKPLACE, a flexible grasping system is used to handle a wide variety of parts. Therefore, the grasping points are not predefined, and thus, it is necessary to monitor the object in order to know how the robot has picked it up.
- We also implement an algorithm to dynamically monitor the destination box and heuristically estimate the free volume as EMSs. When packing a wide variety of different shaped objects, it is necessary to dynamically monitor the destination box to take into account the uncertainties (e.g. drops, movements) that may occur once placed in the box.
- Finally, we develop a dynamic version of IK-PAL mosaic planner that considering the picked part and the destination box monitoring heuristically calculates the packing pose for the picked part.
- The grasping point detection algorithm developed in [1], the dynamic picked part and destination box monitoring methods and the dynamic IK-PAL planner are integrated in a real pick-and-place system.


(a) UHS pilot.

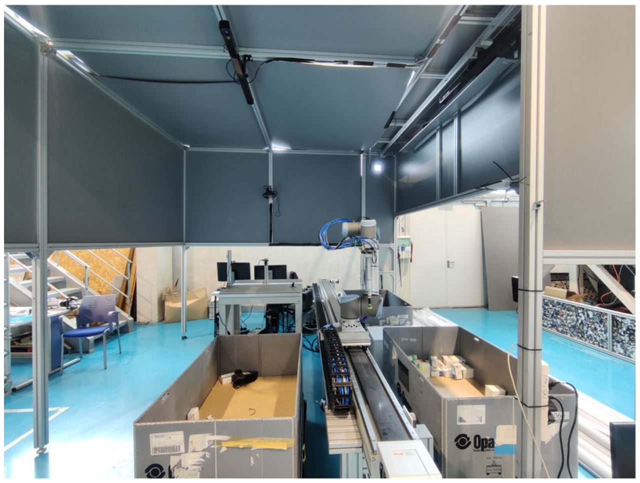

(b) TOFAŞ pilot.

Fig. 1 Pilots in the PICKPLACE project

In this manuscript, we show that the developed packing system is practical to generate mosaics in an online manner using a real robot, with a wide variety of items and being able to also consider uncertainties such as falls in the outbound box.

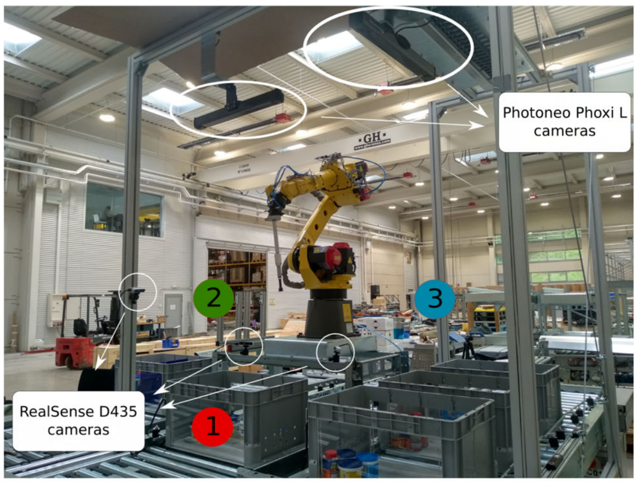
3 System overview

In PICKPLACE, three pick-and-packing scenarios are considered, which are implemented both in the UHS and TOFAŞ pilots (Fig. 1):

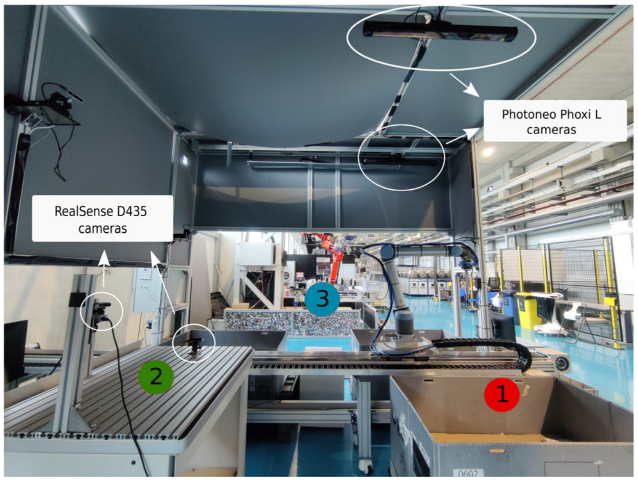
1. Order preparation in UHS: Mono-reference to multi-reference.
2. Order return in UHS: Multi-reference to mono-reference.
3. Order preparation in TOFAŞ: Multi-reference to multi-reference.

In the first case, after the system receives an order from the warehouse management system (WMS), it determines the arrival order of the mono-reference bins to the picking area. The WMS generates the sequence for a new order preparation or for an order return process, and is used in both prototypes. Once the robot picks the required object, it packs the item in the bin of the corresponding order, creating a multi-reference mosaic. In the second use case, the returned multi-reference bin goes directly to the picking area of the UHS pilot. Items are picked according to the order determined by the flexible picking system, until no more items are left inside the bin. Once each object has been picked, its dimensions are estimated and, at the same time, the product is identified. Finally, each product is returned to the corresponding bin of the product's reference, creating a mono-reference mosaic. The third and last use case is implemented in the TOFAŞ pilot and tackles the multi-reference to multi-reference picking and packing problem. Once each product is picked from the picking box, it is identified with a bar-code reader and it is finally placed in the corresponding multi-reference mosaic.

In practice, almost the same workflow is followed in all the use-cases due the following reasons: (1) The flexible grasping system lets us abstract from the type of inbound box (mono/multi-reference). (2) With both known and unknown items, the dimensions and orientation of the picked items need to be estimated online, since those are dependent on how the parts have been picked. Furthermore, the online mosaic generation system abstracts from the category and is able to handle novel parts. The main difference, indeed, is that in the use case 1, the items do not need to be identified after being grasped, as the system knows the category of the mono-reference bin. However, in the use cases 2 and 3, the parts are grasped from a multi-reference bin using a flexible grasping system and therefore, an identification step is required to decide the target bin for the grasped item.



(a) Components of the UHS pilot.



(b) Components of the TOFAŞ pilot.

Fig. 2 Main components of UHS and TOFAŞ pilots. (1) Inbound area, (2) part monitoring area, (3) outbound area

3.1 UHS and TOFAŞ pilots

Both UHS and TOFAŞ pilots are composed of 3 main areas (see Fig. 2): (1) The inbound area, (2) the picked part monitoring area and (3) the outbound area. In both cases, the inbound area contains a single input bin from which the objects are picked. However, the outbound area contains up to three containers in UHS and two containers in TOFAŞ.

The robot used in the UHS prototype is the 6-DoF Fanuc m10ia/10M industrial robot equipped with a suction end effector. In the case of TOFAŞ, the 7-DoF UR10 collaborative robot is used, also with a suction end effector. In the first case, the robot is fixed on a base, from which it is able to reach the 3 areas. In the latter, however, the manipulator is attached to a mobile track which enables the robotic arm to move from one area to another.

In both pilots, over the inbound and outbound areas, there are two *Photoneo Phoxi L* cameras that are used to estimate the grasping points and to monitor the outbound container respectively. In the first case, the camera is fixed as there is only a single bin in the inbound area. In the second case, however, the camera is attached to a linear track that enables the monitoring of up to two/three outbound bins with a single camera, depending on the prototype. As far as the picked part monitoring area is concerned, three extrinsically calibrated *RealSense D435* cameras are used, both in UHS and TOFAŞ, all of them with a fixed location in the layout. This enables the system to have a full view of the item leading to a better estimation of the dimensions and the orientation. The TOFAŞ layout is covered with a housing to reduce the impact of external light on the cameras. All the software runs in an Intel i7-8700@3.2GHz × 12 CPU, with 32GB of RAM and a Nvidia GeForce RTX 2080 Ti GPU with 11GB of memory.

3.2 Software architecture

The application and all its submodules are developed on the *Robot Operating System* (ROS) framework [20]. Indeed, our implementations are based on the layered control architecture dubbed *Robotframework*, which lets us easily develop robotic applications reusing functionalities that are common in many robotic applications [21]. This framework is divided into the following four layers: user interface, application, abilities and drivers (see Fig. 3).

- **Driver layer:** This layer implements all the communication interfaces for data exchange between robots, tools and sensors in simulated environments and real setups.
- **Abilities layer:** It includes the modules that provide the system with high-level functionalities such as manipulation, grasping or perception. It is the middleware between the application level and the drivers.
- **Application layer:** This level contains the robotic applications that provide a solution to the multi-reference random bin picking problem, defined as one of the main goals in PICKPLACE. In our case, each use case is implemented as an operation mode. The robot manager is the module in charge of controlling the overall system, maintaining its status all the time. In addition, it offers all the functionalities through a REST API.
- **User interface layer:** It includes the modules in charge of the interaction between the human and the robotic system, such as the WMS or the Graphical User Interface (GUI).

The diagnostics module collects information from the drivers and the abilities layers and is used to check the status of all the modules, to collect historical data or to make decisions based on the type or errors. The Logger module is responsible for offering log capabilities to all the layers in the architecture.

Among the different functionalities developed in the context of the project, this paper focuses on the packing system. The core of that system is composed of three main modules that are part of the *perception* and *packing mosaic planning* abilities:

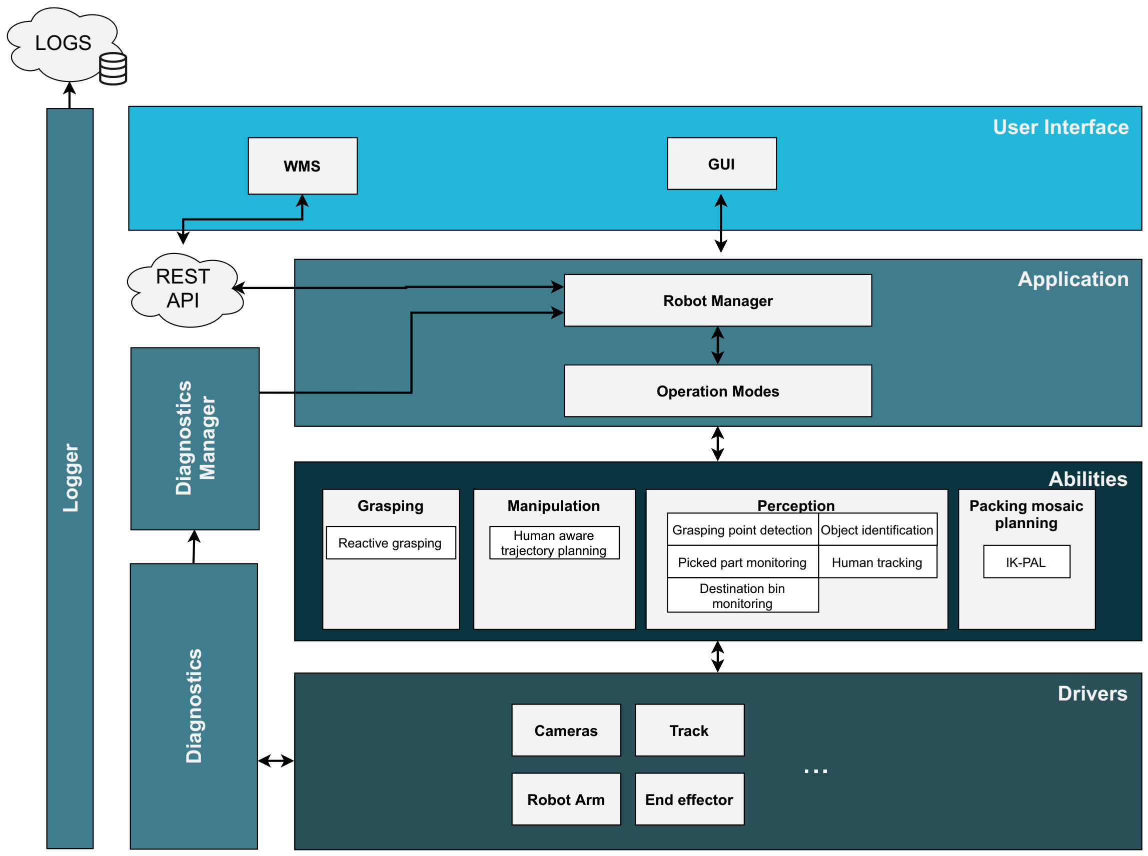


Fig. 3 Robotframework

1. Picked part monitoring.
2. Destination bin monitoring.
3. Dynamic IK-PAL mosaic planning.

These modules are explained in more detail in Sections 3.3, 3.4 and 3.5 respectively.

3.3 Picked part monitoring

The flexible grasping system is capable of handling a large variety of items, and thus the way objects are picked is not predefined. Therefore, the dimensions and the orientation of each picked item need to be estimated online. In this work, the object's bounding box is used as an estimate of the dimensions of the picked part. The bounding box is defined by its origin $bbox_o = [orig_x, orig_y, orig_z, orig_{yaw}]$ (the xyz coordinates and the rotation in z axis of a specific corner of the bounding box with respect to the end effector) and the dimensions $bbox_d = [dim_x, dim_y, dim_z]$.

The bounding box is estimated using three extrinsically calibrated *RealSense D435* cameras. The use of three viewpoints lets us have a full view of the picked item, vital to properly estimate the dimensions of the part. Then, the following steps are performed before estimating the bounding box:

1. Combine the point clouds of the 3 *RealSense* cameras using extrinsic calibrations.
2. Downsample the combined point cloud with a voxel-downsampling algorithm with a voxel-size of V_b^3mm^3.
3. Define a monitoring volume of dimensions M_d with respect to a predefined pose in the space. The robot's end effector moves to a fixed position inside the volume to show the picked part to the cameras.
4. Remove the end effector from the point cloud. Since its dimensions and its position in the monitoring volume are known, a simple point cloud crop is done.
5. Apply a radius filter to remove the noise from the point cloud, with a filtering radius F_r and considering F_n points.

At this point, it is assumed that all the remaining points inside the monitoring volume belong to the item.

The calculated bounding box is finally optimised in the vertical z axis, as it is the most critical orientation during the packing process. The bounding box is not optimised in the x and y axes, as it is assumed that the error in these axes disappears due to gravity when the item is packed.

To optimise the bounding box in the z axis, the origin of the point cloud is translated to its centroid and the cloud is rotated N discrete times with respect to the z axis, checking at which rotation the bounding box is smaller. The bounding box is always estimated following the axes of the origin of the cloud. Finally, the estimated bounding box with the smallest volume is transformed again from the translated coordinate frame into the end effector's coordinate frame (see Fig. 4).

3.4 Heuristic method for destination bin monitoring

The heuristic algorithm to monitor the empty space in the destination box is based on the EMS concept that has been previously used in multiple works [11, 15, 22, 23]. Generally speaking, this concept was applied to 2D and 3D bin-packing problems, always with orthogonal objects assuming ideal conditions in simulated worlds. Moreover, these works assumed that the generated mosaics were static and, therefore, the EMSs were estimated in the theoretical representation of the mosaic. However, this cannot be assumed when irregular multi-reference elements are considered in the system as the uncertainty this introduces to the system generally causes the theoretical and real mosaics to differ. Thereby, our heuristic method tries to estimate the EMSs after each new item has been packed with the aim of having a faithful representation of the real status of the bin.

The system gets as input the point cloud of the scene (CL) and the localisation of the bin (bin_p), and outputs the empty usable volume represented as a list of 3D empty cubes (S) of maximum possible size. Figure 5 depicts a sample scene and its point cloud. Each solution cube $s \in S$ is defined by its origin coordinates with respect to the origin of the bin $s_o = [orig_x, orig_y, orig_z]$ and its dimensions $s_d = [dim_x, dim_y, dim_z]$. The origin of the bin is located in a lower corner of it. A representation of the generated EMSs is depicted in Fig. 6.

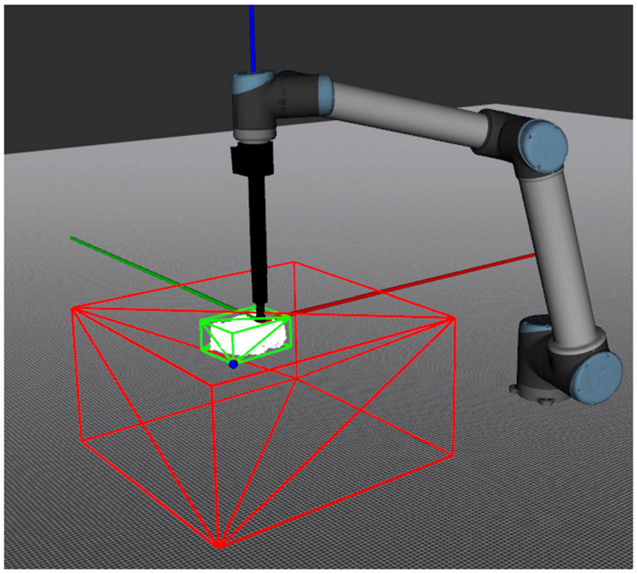
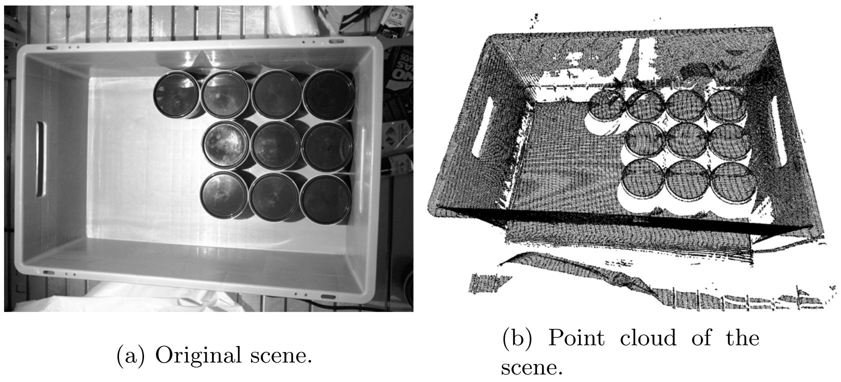


Fig. 4 Bounding box estimation of the picked part. The monitoring volume and the bounding box are represented as red and green cubes respectively. The blue sphere indicates the origin of the bounding box

Fig. 5 Original scene image and point cloud



The process runs as follows. First, the bin is localised in the point cloud of the scene with the aim of discarding the points that lay outside of it. To that end, a previously developed 3D model matching algorithm is used [24]. As the dimensions of the bin (bin_d) are known, the bin localisation algorithm creates the 3D model of the upper edges of the bin and matches it with the point cloud of the scene, finally to obtain its pose (bin_p). In both pilots, a *Photoneo Phoxi L* 3D industrial camera is used to monitor the destination bins.

Once the bin is localised, the point cloud of the scene is transformed into the bin's coordinate frame and the points that are outside of it are discarded. This process is followed by a filtering step in which two methods are applied to reduce the noise in the cloud, both of them available in the *PCL* library [25]. First, a clustering-based filtering helps to remove small groups of points, not attached to the main cloud, that are due to the sensor noise and light reflections. Then, a statistical outlier filter removes the noise attached to the main cloud that prevents regular surfaces. The designed heuristic method uses a 3D occupancy grid $\mathbb{O}$ as a representation of the occupied space inside the outbound box. $\mathbb{O}$ is represented as a 3D array, where each position belongs to a voxel in the space. To fill it, we divide the space inside the bin in voxels of size ${V_s}^3\text{mm}^3$ using *octrees*. This data structure lets us efficiently check the status of each voxel and fill $\mathbb{O}$. An example of the generated

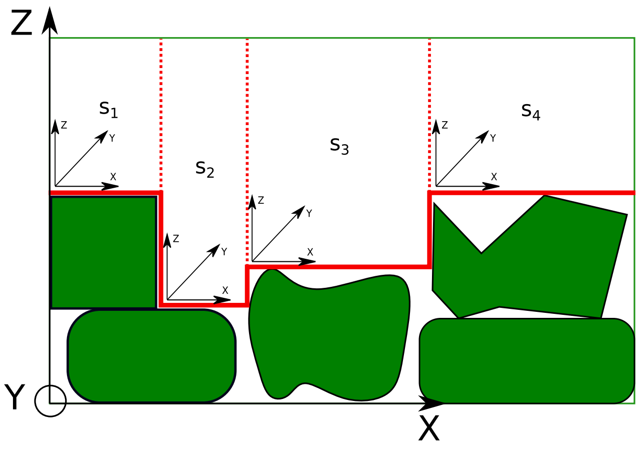
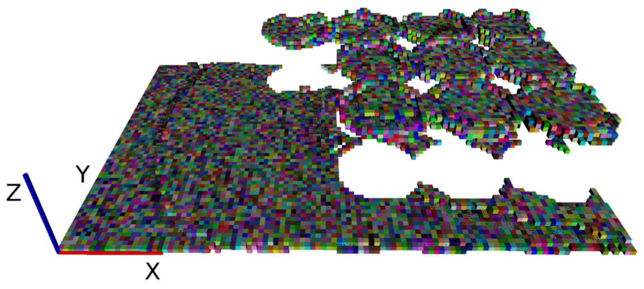
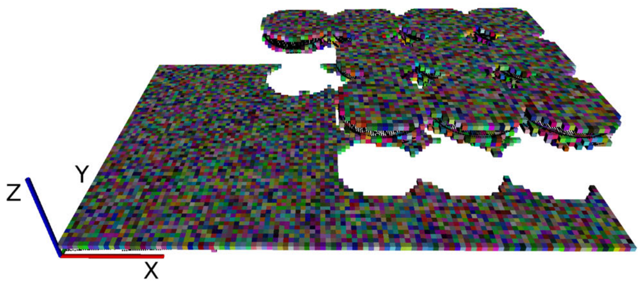


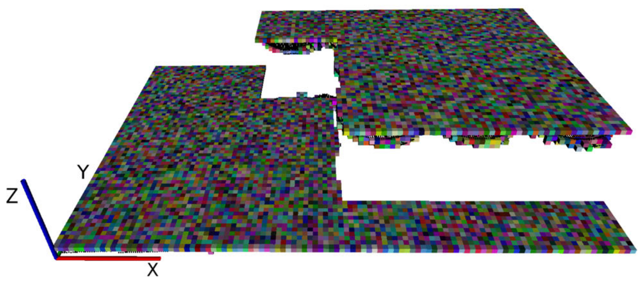
Fig. 6 Side view of the empty cubes generated in an example scene. Green items represent already packed objects and s_1, s_2, s_3, s_4 are the generated EMSs for that scene



(a) Initial occupancy grid with $V_s = 5mm$.



(b) Occupancy grid after the stabilization phase with $H = 5$.



(c) Occupancy grid after square generation phase with $Q = 20$.

Fig. 7 Occupancy grid after each processing phase

occupancy grid is shown in Fig. 7a. The method *occupied* indicates the occupancy of a voxel $v_{i,j,k}$ on the occupancy grid and is defined in Eq. (1).

$$occupied(\mathbb{O}, i, j, k) = \begin{cases} \text{true}, & \text{if } \mathbb{O}[i, j, k] == 1 \\ \text{false}, & \text{if } \mathbb{O}[i, j, k] == 0 \end{cases} \quad (1)$$

When the items are cubic and the mosaic representations used are ideal and not based on real observations, the candidate EMSs are finite. However, the possible EMSs are infinite in real 3D scans of the scene, particularly with irregular and rounded items. Thus, the problem is simplified by representing the space as voxels. The smaller the voxels, the higher the resolution but also the system load.

At this point, the algorithm first processes $\mathbb{O}$ and then calculates the empty cubes with the remaining free space on it. The idea behind this method is to smooth and to fill holes in the occupancy grid with the goal of creating regular surfaces and removing not usable space. The processing done at occupancy grid level is parameterised with two thresholds, H and Q that are explained in detail later on. At the end of the voxel-level processing of $\mathbb{O}$, the empty cubes are generated using all the remaining space inside the box. Indeed, each point in the empty space must belong to one and only one cubic solution. Finally, if J is enabled, a cube-level processing is applied to join neighbour cubes that are at the same height, to maximise their size.

The proposed heuristic method is shown in Algorithm 1 and is composed of four main steps. As explained before, the first processing steps are performed at voxel-level, directly modifying the occupancy grid. However, the final step is performed at the cube level.

The second step of the proposed heuristic method, dubbed the *heightmap stabilisation* phase, makes use of a height threshold H to smooth $\mathbb{O}$ (see Algorithm 2).This method starts from the highest occupied voxel $v_{i,j,k}$ and finds the values $\alpha, \beta, \gamma, \delta \geq 0$ to update the occupancy of the neighbour voxels from $v_{i-\alpha,j-\beta,k}$ to $v_{i+\gamma,j+\delta,k}$ and set them as occupied if:

```
Input: $CL$ , $V_s$,$H$, $Q$,$J$, $bin_d$, $bin_p$
Result: List of empty cubic volumes $S$
initialisation;
Function Main($CL$ , $V_s$, $H$, $Q$, $J$, $bin_d$, $bin_p$):
    // Step 1: Preprocessing
    $\mathbb{O} \leftarrow$
    preprocessPointCloud($CL$, $V_s$, $bin_d$, $bin_p$);

    // Step 2: Heightmap stabilisation
    $\mathbb{O} \leftarrow$
    stabiliseOccupancyGrid($\mathbb{O}$, $H$, $V_s$, $bin_d$);

    // Step 3: Square generation
    $\mathbb{O} \leftarrow$
    squareOccupancyGrid($\mathbb{O}$, $Q$, $V_s$, $bin_d$);
    // Step 4: Generate solutions
    $S \leftarrow$ generateSolutions($\mathbb{O}$, $V_s$, $bin_d$);
    if $J$ then
        $S \leftarrow$ joinSolutions($S$);
    end
    return $S$;
```

Algorithm 1 Calculate the list of empty cubes S.

$$D_z(a, b, k) \leq H \text{where,} a = \{i - \alpha, ..., i + \beta\}, \\ b = \{j - \gamma, ..., j + \delta\} \quad (2)$$

The function D_z measures the distance in voxels from a voxel $v_{i,j,k}$ to the closest occupied voxel below it and in the

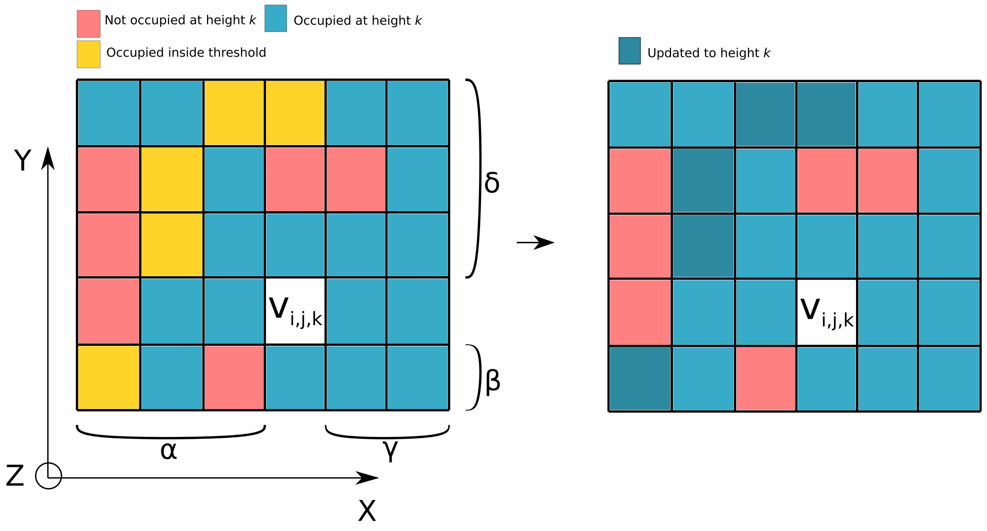


Fig. 8 Square formed by the voxels $v_{i-\alpha,j-\beta,k}$ and $v_{i+\gamma,j+\delta,k}$ at height k of the occupancy grid. In blue, the occupied voxels at height k. In yellow, the voxels where $D_z \leq H$, which are updated to height k. In red, the voxels that are not occupied at height k and $D_z > H$

same column i, j, and is defined in Eq. (3).

$$D_z(i,j,k) = 1 + \sum_{w=1}^{k} 1 \text{ if} \neg\, occupied(\mathbb{O}, i, j, k-w) \quad (3)$$

The values $\alpha, \beta, \gamma, \delta$ are calculated in such a way that in the square formed by the corner voxels $v_{i-\alpha,j-\beta,k}$ and $v_{i+\gamma,j+\delta,k}$, in each row and column at least one voxel is at height k or can be updated to height k (see Fig. 8).

The voxel columns whose occupancy has been changed are marked as modified and this process is executed iteratively until all the voxel columns are analysed. An example of the status of the occupancy grid after the stabilisation phase is depicted in Fig. 7b.

```
Input: $\mathbb{O}, H, V_s, bin_d$
Result: Stabilised occupancy grid $\mathbb{O}$.
Function stabiliseOccupancyGrid($\mathbb{O}, H, V_s, bin_d$):
    $x\_voxels \leftarrow bin_d[0]/V_s$;
    $y\_voxels \leftarrow bin_d[1]/V_s$;
    $z\_voxels \leftarrow bin_d[2]/V_s$;
    $modified \leftarrow false$;
    // $bool\_array(x\_voxels, y\_voxels)$
    for $k \in \{z\_voxels - 1, ..., 0\}$ do
        for $j \in \{0, ..., y\_voxels - 1\}$ do
            for $i \in \{0, ..., x\_voxels - 1\}$ do
                if $occupied(\mathbb{O}, i, j, k)$ & $\neg modified[i][j]$ then
                    Get $\alpha, \beta, \gamma, \delta$ ;
                    for $a \in \{i - \alpha, ..., i + \beta\}, b \in \{j - \gamma, ..., j + \delta\}$ do
                        if $D_z(a, b, k) \leq H$ then
                            // Update the height of the column $a, b$ to $k$
                            updateHeight($\mathbb{O}, a, b, k$) ;
                            $modified[a, b] \leftarrow true$ ;
                        end
                    end
                end
            end
        end
    end
    return $\mathbb{O}$;
```

Algorithm 2 Occupancy grid stabilisation method.

The third processing step, called the *square generation* phase, is in charge of filling holes and giving squared shape to the planes generated in the stabilisation step with the intuition of removing not usable space. As the final empty cubes have a square floor plan and are generated only over occupied planes using all the remaining free space, it is important to fill the occupancy grid creating planes with regular shape. To that end, a second threshold Q is used. This threshold is applied in the x and y axes and is used to determine the maximum size of the holes that will be filled in each plane, without considering the vertical distance D_z to the closest occupied voxel. Therefore, the maximum number of consecutive empty voxels that are filled in each axis is Q, which could lead to filling holes up to an area of Q^2 voxels, considering both x and y axes. The main goal of filling small holes in the planes generated in the stabilisation phase is to reduce unusable small volumes and to generate bigger planes that will contribute to the generation of bigger empty cubic volumes.

This method also starts in the highest occupied voxel $v_{i,j,k}$ in $\mathbb{O}$, and finds the values $\epsilon, \zeta, \eta, \theta \geq 0$ in such a way that from $v_{i-\epsilon,j-\zeta,k}$ to $v_{i+\eta,j+\theta,k}$ there is no hole with a larger dimension (x or y) than Q (Fig. 9). The analysed voxels are marked as visited and the same process is executed until all the voxels are processed (see Algorithm 3). An example of the result of this process is shown in Fig. 7c.

After executing the voxel-level operations, all the remaining free space above the occupied zone is assumed to be usable space. At this point, cubic solutions are generated and those are further processed to create solutions as big as possible.

The solution generation method starts in the highest empty voxel that is over an occupied plane, and computes the dimensions of the plane that it belongs to. The height of the cube is determined by the distance between each plane and the bin's top. The solutions are generated in such a way that (1) all the voxels under the floor of each cubic solution must be occupied and, (2) each empty voxel inside the bin must belong to one solution only. The procedure to generate the cubic solutions is described in Algorithm 4.

As the cubic solutions are generated in a non-optimal way, joining them can lead to maximised size solutions. Therefore, in a final step, the neighbour solutions with the origin at the same height in the z axis are combined in pairs by following these criteria:

1. If the cubic solutions have the same origin in axes x or y and have the same dimensions in the corresponding axis, the cubes are joined to generate a bigger cube (see Fig. 10a).
2. If the cubic solutions have the same origin in axes x or y but not the same dimension in the corresponding axis, then two new cubes are generated if the union of both cubes results in a bigger cube than every single one (see Fig. 10b).

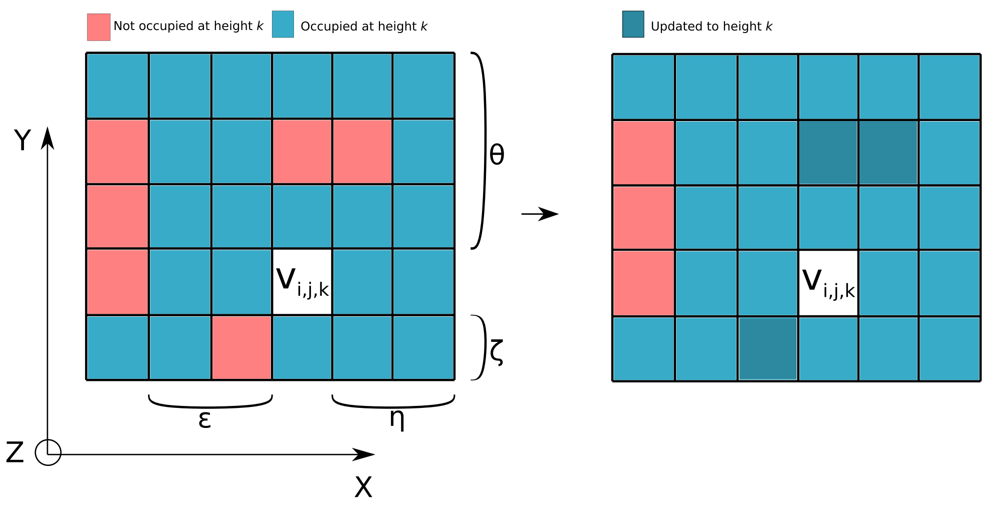
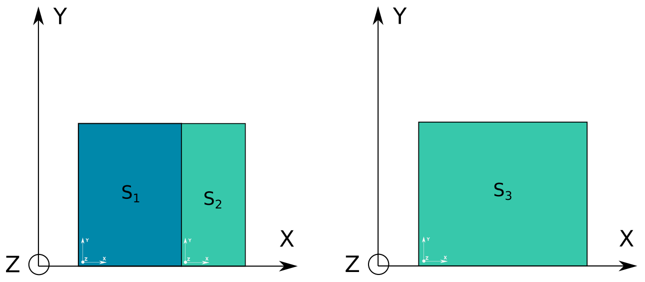


Fig. 9 Example with $Q = 2$. The voxels in the square formed by $v_{i-\epsilon,j-\zeta,k}$ and $v_{i+\eta,j+\theta,k}$ are updated to height k without considering the distance D_z

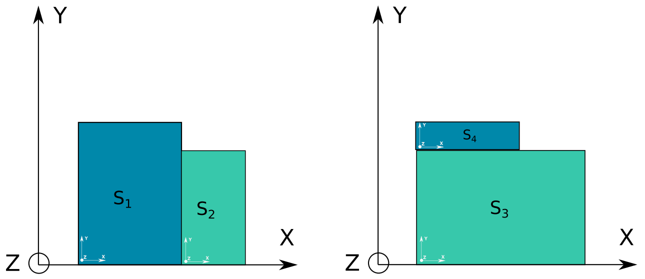
3. If the cubic solutions are neighbours but they do not share a common origin on any axis, then three new cubes are generated if the union of both cubes results in a bigger cube than any original one (see Fig. 10c).

```
Input: $\mathbb{O}$, $Q$, $V_s$, $bin_d$
Result: Squared occupancy grid $\mathbb{O}$.
Function
squareOccupancyGrid($\mathbb{O}$, $Q$, $V_s$, $bin_d$):
    $x\_voxels \leftarrow bin_d[0]/V_s$;
    $y\_voxels \leftarrow bin_d[1]/V_s$;
    $z\_voxels \leftarrow bin_d[2]/V_s$;
    $modified \leftarrow false$;
    // $modified$ : $bool\_array(x\_voxels, y\_voxels)$
    for $k \in \{z\_voxels - 1, ..., 0\}$ do
        for $j \in \{0, ..., y\_voxels - 1\}$ do
            for $i \in \{0, ..., x\_voxels - 1\}$ do
                if $occupied(\mathbb{O}, i, j, k)$ &
                $\neg modified[i][j]$ then
                    Get $\epsilon, \zeta, \eta, \theta$ considering $Q$;
                    for $a \in \{i - \epsilon, ..., i + \zeta\}, b \in$
                    $\{j - \eta, ..., j + \theta\}$ do
                        // Update the height
                           of the column $a, b$
                           to $k$
                        updateHeight($\mathbb{O}, a, b, k$);
                        $modified[a, b] \leftarrow true$ ;
                    end
                end
            end
        end
    end
    return $\mathbb{O}$;
```

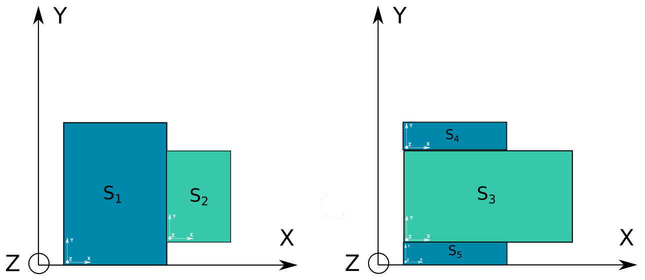
Algorithm 3 Occupancy grid square generation method.



(a) Cubes s_1 and s_2 with common origin and dimension in y axis are joined to generate s_3.



(b) Cubes s_1 and s_2 with same origin in y axis are combined if the combination (s_3) is bigger than s_1 and s_2. As a result s_3 and s_4 are generated.



(c) Cubes s_1 and s_2 are neighbours and are combined if the combination (s_3) is bigger than s_1 and s_2. As a result s_3, s_4 and s_5 are generated.

Fig. 10 Joining criteria for the cubic solutions

```
Input: $\mathbb{O}$, $V_s$, $bin_d$
Result: List of empty cubic volumes $S$.
Function generateSolutions($\mathbb{O}$, $V_s$, $bin_d$):
    // occupied: Whether voxel $v_{i,j,k}$ is occupied
    // hasObstacleAbove: Whether voxel $v_{i,j,k}$ has any higher occupied voxel in column i,j
    // isInSolution: Whether voxel $v_{i,j,k}$ already belongs to a solution
    // isAboveObstacle: Whether voxel $v_{i,j,k}$ has any lower occupied voxel in column i,j
    $x\_voxels \leftarrow bin_d[0]/V_s$;
    $y\_voxels \leftarrow bin_d[1]/V_s$;
    $z\_voxels \leftarrow bin_d[2]/V_s$;
    $S \leftarrow [\,]$;
    $inSolution \leftarrow false$ ;
    // $inSolution$ : $bool\_array(x\_voxels, y\_voxels)$
    for $k \in \{z\_voxels - 1 , ..., 0\}$ do
        for $j \in \{0 , ..., y\_voxels - 1\}$ do
            for $i \in \{0 , ..., x\_voxels - 1\}$ do
                if $\neg occupied(\mathbb{O}, i, j, k)$ & $\neg hasObstacleAbove(\mathbb{O}, i, j, k)$ & $\neg inSolution[i, j]$ & $isAboveObstacle(\mathbb{O}, i, j, k)$ then
                    $s_o \leftarrow [i, j, k]$;
                    // Calculate the dimensions of the cube
                    $s_d \leftarrow$ getSolDims($\mathbb{O}, i, j, k$);
                    $S.insert(s)$;
                    $inSolution[i, j] \leftarrow true$;
                end
            end
        end
    end
    return $S$;
```

Algorithm 4 Cubic solution generation method.

The projection of the generated 3D cubic solutions is depicted in Fig. 11.

3.5 Dynamic IK-PAL mosaic planning

IK-PAL is the proprietary mosaic planner algorithm developed by UHS. The first version of this algorithm was an offline mosaic planner that was able to plan full mosaics with regular parts. This offline planner assumed that the items were picked and packed always in the same way, using the same grasping points and that the set of items to be packed was previously known. Moreover, this method was designed to be used with regular parts only, assuming that after each object was placed in the destination box, the mosaic would remain static. In fact, a manual intervention of the operator was needed whenever an unexpected movement occurred or any item fell off. However, this static mosaic assumption does not hold when irregular items are introduced in the system. Thus, the mosaic has to be created online considering the packing status after each packing operation.

Therefore, in the context of the project, a new dynamic version of the algorithm has been designed and implemented. The new version of the mosaic planner gets as input the list of estimated EMSs in the destination bin, as well as the bounding box of the picked item. As a result, it returns the 3D coordinate inside the destination bin where the origin of the bounding box has to be located, besides to the required rotation of the picked item on the z axis. Since the picked parts are represented as boxes, the rotation in z axis is defined by a binary variable, where a rotation of 90° is applied when it is enabled.

The dynamic IK-PAL is an heuristic-based mosaic planner that only considering the current packing status estimates the next optimal packing pose with the following criteria:

1. First, the EMSs where the bounding box fits in are extracted. In each selected empty volume, there are two possible poses for the bounding box: As it is or with the rotation enabled.
2. A tree data structure is generated with all possible solutions.
3. Criteria such as the degree of filling, compactness or the size of the remaining empty volumes are considered to give a rating to them.
4. The solution with the highest rating is selected.

As this module is a proprietary software of UHS that is under license, the implementation details are not available.

4 Experimentation

This section explains the details of the experimentation carried out to validate the developed algorithms.

4.1 Picked part monitoring

The main objective of the experimentation with the picked part monitoring algorithm was to select the appropriate

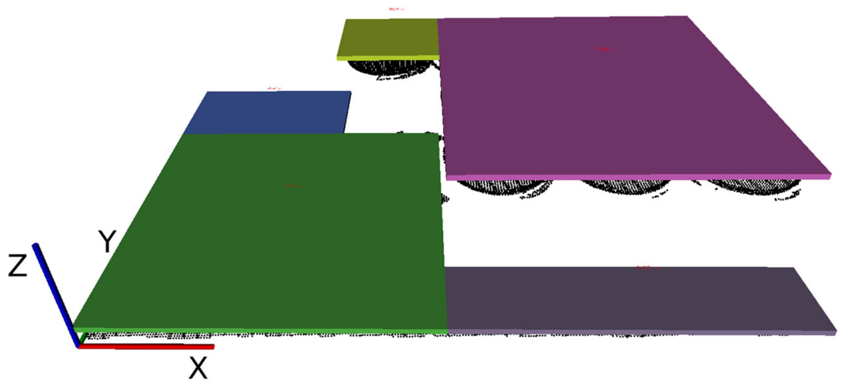


Fig. 11 Projections of the cubic solutions that represent the empty space inside the bin

parameters for the calculation of the dimensions of the picked object. In addition, we measured the accuracy of the proposed system by comparing its estimates with the measured dimensions of several objects.

As previously explained in Section 3.3, three *RealSense-D435* cameras were used to estimate the dimensions of the object picked up. As expected, the quality of the 3D information they provide was not comparable to industrial cameras such as the *Photoneo Phoxi L* which were used both to detect grasping points and to monitor the destination bin. In fact, such cameras were very sensitive to ambient light conditions. Therefore, the selection of the parameters was done experimentally and taking into account the lighting conditions of the real environment, and these can be seen in Table 1.

To measure the error made by the algorithm, 10 representative objects with different materials and shapes were selected. For each object, once the robot picked it up and once at the monitoring station, a measurement of the object's dimensions was performed. In addition, keeping the robot static in the monitoring volume, the algorithm was run 10 times (since the point cloud of the scene changed slightly at each acquisition), and the average dimensions estimated by the algorithm were calculated. Figure 12 depicts the measurements process with one of the objects used.

As it can be seen in Table 2, the error committed on each axis was approximately 1 cm on average. According to the camera specifications, the error that these cameras make in estimating depth information is 2%. In our case, the objects were approximately 50 cm far from each camera, and thus, the error could be up to 1 cm per axis. Although there were other factors that could introduce noise into the estimation, such as the extrinsic calibration of the cameras or reflections caused by certain materials such as shiny metals or semi-transparent objects, it was concluded that the measurements of the algorithm were accurate and in accordance with the camera specifications.

Table 1 Tuning parameters for the picked part monitoring algorithm

Picked part monitoring	
V_b	$2mm$
F_r	$10mm$
F_n	5
M_d	$600 \times 600 \times 300mm$
N	64

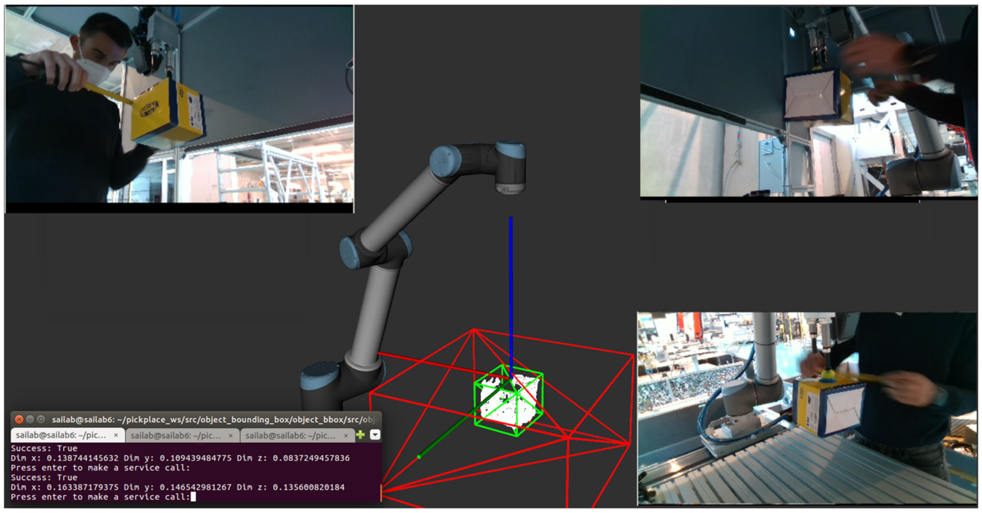
4.2 Destination bin monitoring

At this experimental step, the goal was to find the combination of parameters that best adapted on average to highly variable scenes. For this purpose, the algorithm was evaluated with a set of 168 scenes taken from the dataset published in [1], where the number, type and placement of objects were random. The dataset was composed of scenes, similar to the one showed in Fig. 5, where the localisation of the bin in each scene was also available. To carry out the search for the optimal parameters, we opted for a grid-search approach, where the possible values for each variable were manually defined, and were as follows:

$$V_s = [1, 5, 10] H = [False, \frac{10}{V_s}, \frac{30}{V_s}, \frac{50}{V_s}]$$
$$Q = [False, \frac{10}{V_s}, \frac{50}{V_s}, \frac{100}{V_s}] J = [False, True] \tag{4}$$

The False option in the parameters H, Q and J indicates that the heightmap stabilisation, square generation or/and solution joining processing phases were disabled respectively. In all the tests, both the clustering-based and the statistical outlier filters were enabled to remove the noise from the point cloud scenes. Thus, per each scene 96 combinations were tested, which gave a total of $96 \cdot 168 = 16128$ trials. In each test, the following metrics were used to measure the quality of the generated solutions:

- Usable volume ratio: One of the objectives of the algorithm was to maximise the total usable volume (i.e. the sum of the volume of all the estimated EMSs that represent usable free space). To do so, we measured the



```
sailab@sailab6: ~/pickplace_ws/src/object_bounding_box/object_bbox/src/obj
sailab@sailab6: ~/pic...  sailab@sailab6: ~/pic...  sailab@sailab6: ~/pic...
Success: True
Dim x: 0.138744145632 Dim y: 0.109439484775 Dim z: 0.0837249457836
Press enter to make a service call:
Success: True
Dim x: 0.163387179375 Dim y: 0.146542981267 Dim z: 0.135600820184
Press enter to make a service call:
```

Fig. 12 Bounding box error measurements

ratio of estimated usable space to total space, which told us what percentage of the actual free space was estimated as usable space.

- Execution time: The second objective was to optimise the algorithm's execution time (i.e. given a point cloud of the scene the time needed to estimate the EMSs), which was vital to guarantee the application's cycle time.
- Number of EMSs: The last aspect to optimise was the number of estimated EMSs. Specifically, the aim was to minimise the number of EMSs that represent the usable space inside the box.

Finally, for each of the 96 parameter combinations, the average values of the previously defined metrics were estimated. Since we were interested in optimising the parameters based on three different criteria, following the general approach, we tried to find a balance between them.

Figure 13 represents the mean values of all the combinations available in the grid-search process. In red are represented the non-dominated combinations that belong to the Pareto front, which are considered as equally good solutions. Conversely, the combinations shown in blue are dominated solutions that do not belong to the Pareto front. In our case, we selected the parameter combination that is represented by the green star among the non-dominated candidate solutions. The selected combination of parameters and the estimated mean scores per each criteria are shown in Table 3.

To have a clearer view on the behaviour of the algorithm using the selected combination of parameters, we evaluated it in scenes with different complexity. To that end, the validation scenes were divided into three degrees of complexity in which the heuristic was run. Figure 14 shows the execution times, usable volume ratios and the generated EMSs in scenes with low, medium and high complexities.

Table 2 Errors in the bounding box estimation during the picked part monitorisation

Item	Measured		(cm)	Estimated		(cm)	Error		(cm)
	x	y	z	$\overline{x}$	$\overline{y}$	$\overline{z}$	x	y	z
1	10	12	6	10.5	13.8	6	0.5	1.8	0
2	14	12.5	12	16.3	14.6	13.5	2.3	2.1	1.5
3	20.5	20.5	16.5	22.2	21.8	18.1	1.7	1.3	1.6
4	39	18.5	6	40.3	18.7	7.6	1.3	0.2	1.6
5	14.6	23	14	14.1	22	12	0.5	1	2
6	32	20	24.5	31.5	19.7	23.2	0.5	0.3	1.3
7	16.5	21.5	10	17.3	22.1	9.4	0.8	0.6	0.6
8	10	6.5	5	11.6	7.8	5.7	1.6	1.3	0.7
9	17	18	13	17	17.2	13.9	0	0.8	0.9
10	35.5	11	13	37.5	12	13.5	2	1	0.5
$\overline{x}$							1.12	1.04	1.07

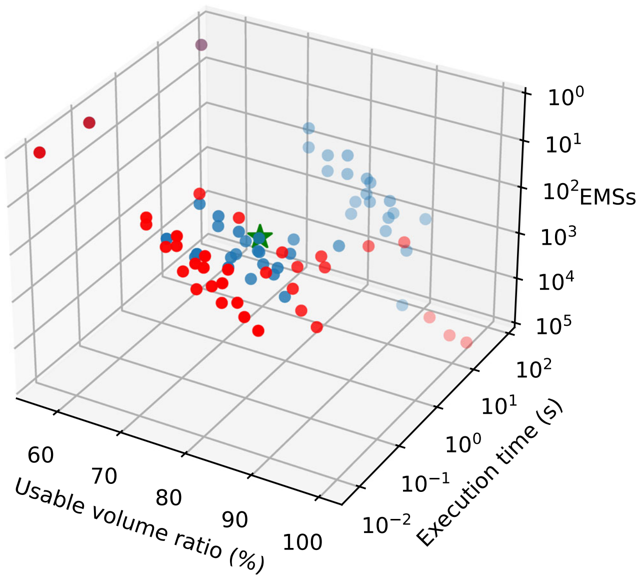


Fig. 13 The scores of the 96 parameter combinations considering the average usable volume ratio, average execution time and the average number of generated EMSs. In red, the non-dominated parameter combinations that belong to the Pareto front. The green star represents the scores of the selected parameter combination

The scenes with less than 10 objects were categorised as of "low complexity", scenes with 10 to 20 objects as of "medium complexity" and the scenes with more than 20 objects as of "high complexity".

As far as execution times are concerned (see Fig. 14a), although it can be seen that on average there was a slight increase as the complexity of the scene increased, we can see that between the best case of "low complexity" and the worst case of "high complexity", there was an increase of approximately three times. The same occurred within the "high complexity" scenes that although there was not much variation between the number of objects, the execution time did vary. This happened because the computational cost was not directly related to the number of objects in the scene but rather to the regularity of the scene. Despite the scene itself, it did not have any effect on the processing steps performed at the occupancy grid level, the more irregular the scene was, the higher was the number of EMSs generated (see Fig. 14c). An increase in the number of EMSs generated had an effect on the solution joining phase where adjacent

Table 3 Tuning parameters and estimated mean scores for the destination bin monitoring heuristic

Selected parameters	
V_s	$5mm$
H	2
Q	20
J	$True$
Estimated mean scores	
Usable volume ratio	83.67%
Execution time	0.074s
EMSs	26.9

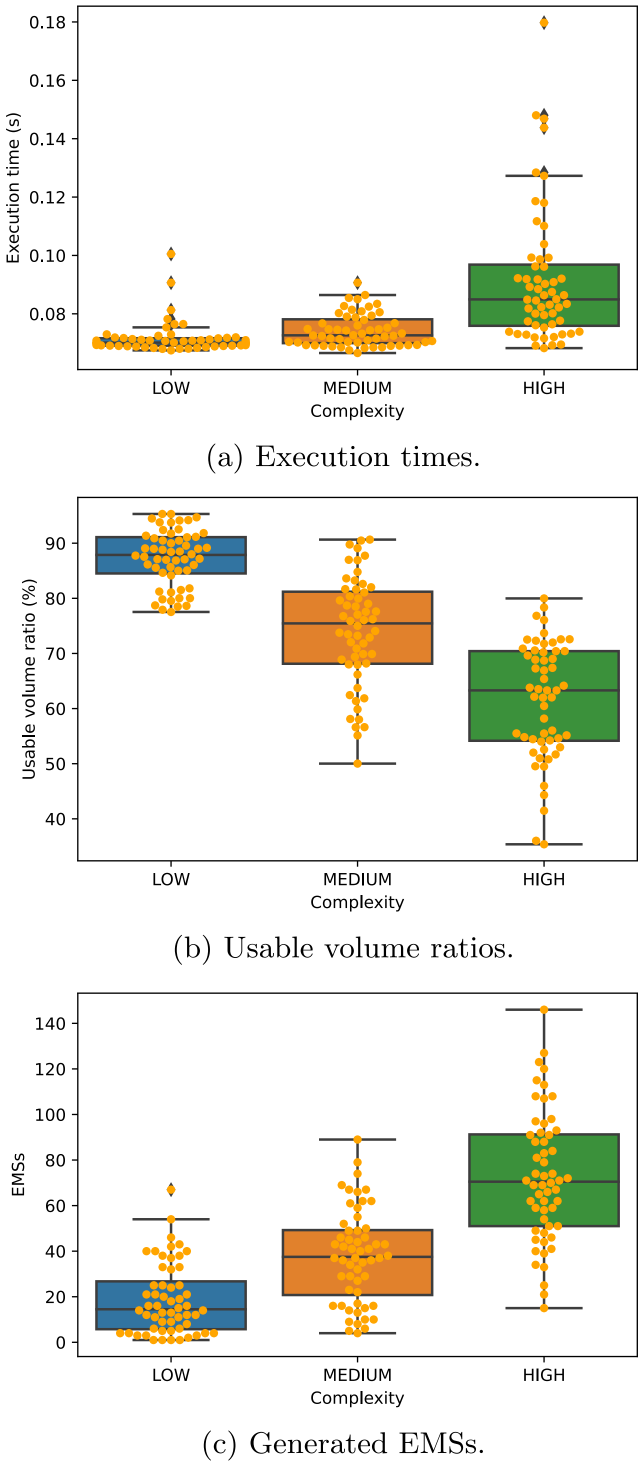


Fig. 14 Execution times, usable volume ratios and generated EMSs with respect to scenes with low, medium and high complexity

solutions were exhaustively searched for in order to join them, causing an increase of the execution time. In general, scenes with more objects tended to be more irregular, which also affected the usable volume ratio. This was due to the fact that in irregular scenes, there were many gaps that were not usable and were eliminated in the stabilisation and square generation phases (see Fig. 14b).

As we have seen, as the complexity of the scenes increased, the number of EMSs generated also increased. Therefore, we also measured the average volume of the generated EMSs among the 168 scenes to see how it changed as the complexity of the scene incremented. In order to get an idea of their size, we compared them with the smallest, the medium-sized and the largest objects among the representative objects selected in Section 4.1, and split them into 4 groups. In addition, we also measured the percentage of the total volume each of the groups represented. Figure 15 shows the percentage of EMSs generated in each of the 4 groups and the percentage of the total volume they represent for scenes with low, medium and high complexity.

As it can be seen in Fig. 15, the average percentage of EMSs with a volume smaller than the smallest object ranged from 36 to 63% as the complexity of the scenes increased. Although these were relatively high percentages since these volumes were not usable, the total volume they represented ranged from 0.8 to 6% in the worst case. In the opposite case, the percentage of EMSs generated with a higher volume than the largest object ranged from 36.5% in the simplest scenes to 4.3% in complex scenes. This indicates that when the scenes were less complex,

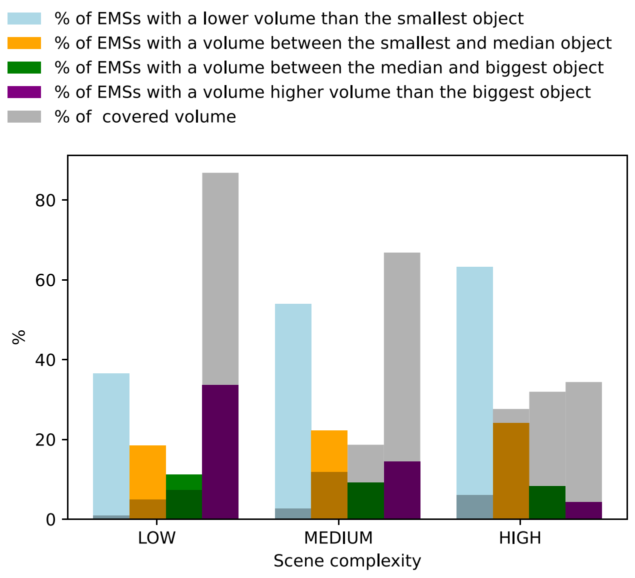


Fig. 15 Size and covered volume of the generated EMSs with respect to the smallest, median and biggest objects in scenes with low, medium and high complexity

the algorithm was able to generate large volumes, which on average represented the 86.8% of the total volume. However, as complexity increased, the number of large EMSs decreased to 4% but still represented 34.3% of the space. In summary, we can say that the size of the EMSs generated was directly related to the complexity of the scene. Although the algorithm had the tendency to generate small solutions, they represented a small percentage of the total volume. In general, these were small unusable spaces that could not be joined with larger EMSs due to the current configuration of the algorithm, but this effect could be reduced by increasing the H and Q thresholds. Figure 16 shows an example where due to the rounded shape of the bottle, a solution with a very small volume was generated.

5 Validation in the real robotic system

The validation of the complete packing system was carried out in the TOFAŞ pilot. To that end, the developed modules were deployed in the real system and those were integrated with the automatic picking system previously mentioned. We selected the multi-reference to multi-reference use-case as it is the most complex and provides a clear view of the strengths and weaknesses of the system. For the sake of simplicity, we only used a single bin in both inbound and outbound areas.

To validate the system, we performed 7 different tests which were divided in three blocks (easy, medium and complex). The main goal of these tests was to show the flexibility and usefulness of the system to create mosaics with several types of objects and configurations inside the inbound bin, without the need of ad-hoc configurations. Since there was no similar online bin-packing system publicly available, we recorded the tests performed to demonstrate the usefulness of the system. The tuning parameters used during the validation of the complete packing system are those that were selected during the experimentation, detailed in Section 4. Note that we used the same configuration in all the tests.

We performed the following tests:

- **Easy:**
  1. *Test 1:* Mono-reference boxes were placed in the inbound box, all of them with the same orientation.
  2. *Test 2:* Multi-reference boxes were manually thrown inside the inbound box, ordered from big to small considering their volume.
- **Medium:**
  1. *Test 3:* Multi-reference boxes were mixed in the inbound box, all of them in the most stable

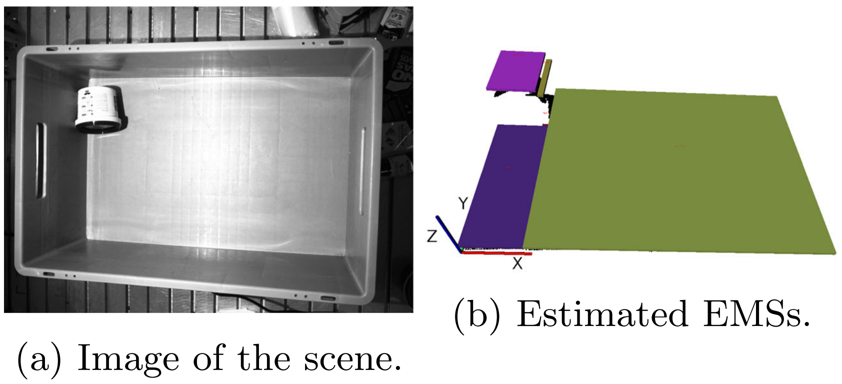


Fig. 16 Example of the generation of a small not usable space due to the rounded shape of the bottle

orientation. Once the algorithm was unable to pack more objects in that orientation, the orientation of the remaining objects was manually changed.
2. *Test 4:* Multi-reference cans and bottles were placed in the inbound box, ordered from big to small considering their volume.

- **Complex:**
  1. *Test 5:* Multi-reference boxes were placed in the inbound box, with random order and orientation.
  2. *Test 6:* A reference of boxes and another reference of bottles were mixed in the inbound box.
  3. *Test 7:* Random objects were randomly thrown into the inbound box.

Figure 17 shows the flow of the bin-packing application during the tests, which were conducted in the following way:

1. The number of objects for each test was selected based on the requirements of the test and the availability of the type of objects required.
2. All the items were put in the inbound box at the beginning of the test except in test 2 and test 4, where items were manually placed (ordered from big to small).
3. At each iteration, the grasping point detection module decided which object to pick. If the grasp failed, the next grasping point was selected and a new grasp was executed. This procedure was repeated $N = 3$ times, and if the system was not able to pick any object, the test was concluded.
4. The objects were packed in the same order that had been picked. We did not perform any object identification since a single destination bin was used and all the picked items were packed in the same bin.
5. The test ended when there were not objects left in the input bin, when the robot was not able to pick more objects or when the system was not able to pack the picked item.

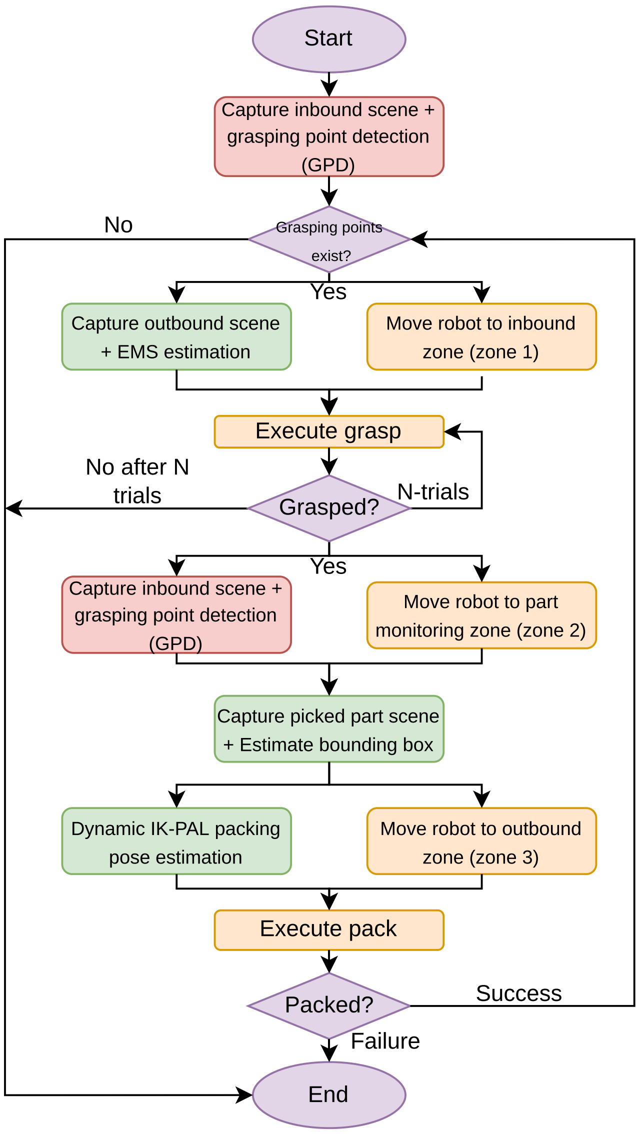

Start
Capture inbound scene + grasping point detection (GPD)
Grasping points exist?
No
Yes
Capture outbound scene + EMS estimation
Move robot to inbound zone (zone 1)
Execute grasp
No after N trials
Grasped?
N-trials
Yes
Capture inbound scene + grasping point detection (GPD)
Move robot to part monitoring zone (zone 2)
Capture picked part scene + Estimate bounding box
Dynamic IK-PAL packing pose estimation
Move robot to outbound zone (zone 3)
Execute pack
Packed?
Success
Failure
End

Fig. 17 Flowchart of the complete bin-packing application. Red boxes indicate tasks related to the grasping of the objects. Green boxes belong to tasks related to the packing of the picked part. In orange the tasks that imply the motion of the robot

The tests were recorded and the videos are publicly available at the following link: https://bit.ly/3aWc0ke.

In the recorded videos, it can be observed that the main strength of the system was the flexibility, being able to pack online a wide variety of items. The analysis of the destination bin at each iteration let us consider fallen objects or unexpected movements inside the bin that happened after releasing the parts.

Concerning the first test, the aim was to see how the system behaves with single reference boxes placed in their most stable orientation. Although due to their low weight some objects moved/rotated slightly during placement, these movements were taken into account in each iteration, thanks to the monitoring of the target box. However, as a consequence of these rotations, the separation between objects increased. The constructed mosaic was stable due to the orientation of the products in the inbound bin.

In the second and third tests, the system was assessed with multi-reference boxes. Although in both cases the system was able to generate a mosaic in a flexible way and taking into account the variations caused by the robotic packing system, as expected the packing order had a great effect on the final result. In both tests, it can be appreciated that, when certain objects were packed, they sometimes touched nearby objects. This happened due to the error introduced in the monitorisation of the picked part, since the dimension estimates were smaller than the real ones. Although this caused the movement of nearby objects inside the box, the system took this uncertainty into account when packing the next object.

Although we have seen that the system generates stable mosaics with multi-reference boxes placed in their most stable pose, this is not the case when the orientation of the boxes is random (test 5). In this case, the generated mosaic was unstable, which caused some objects to fall when they were packed on top of other unstable objects. Although the system successfully considered such movements in the target box, the quality of the final mosaic decreased considerably.

In a similar way to test 2, in test 4, we also tried to evaluate the behaviour of the system when the objects were ordered from bigger to smaller, but this time using cans and bottles. The system worked correctly, and although some small friction occurred when placing an object, which modified the final pose of the object to be left, the system took it into account correctly in the following iteration. As can be guessed, it is also key the cans and bottles to be ordered from larger to smaller the constructed mosaic to be stable.

In test 6, a box reference was mixed with a bottle reference in the inbound box, and the picking system determined the packing order. As can be seen, the generated mosaic was irregular and sometimes unstable, particularly when objects of different sizes were packed. This caused some objects to fall after placing an object, making it even harder to generate a stable mosaic. Although the separation between objects and, in general, the use of space could be improved, it is clear that the strong point of the system is the ability to adapt to unforeseen situations and to plan online taking into account these uncertainties.

Finally, in test 7, we sought to demonstrate the ability of the system to pack a wide variety of objects without the need to configure the system for each reference. It can be seen that the system showed great flexibility and was capable of dynamically packing objects taken in random order. Of course, with such a variety of objects which are totally unknown to the system until the moment they are picked, it is difficult to build a stable and space-efficient mosaic.

5.1 Timeliness analysis

As mentioned above, the system has been developed on the ROS framework, which does not guarantee real-time execution of robotic applications. The aim of this test is to evaluate the timely execution of our system and to demonstrate that although we do not use a real-time framework, the system maintains a cycle time without major variations.

For this, the application was run 10 times following the flow shown in Fig. 17. In each run, 10 objects that were easily picked with the suction tool (boxes and cans) were selected and randomly dropped into the input container. In each picking/packing cycle, the execution times of each main component of the application were recorded, which are shown in Table 4. The measurement of the average total execution time has been made taking into account that not all tasks are executed sequentially. As can be seen in Fig. 17, while the robot is moving from one area to another, other tasks are carried out in parallel, thus reducing the cycle time. The target cycle time was set at 25 s.

A total of $10 \cdot 10 = 100$ picking/packing cycles were executed with an average execution time of $22.32s$. Considering the worst case, the cycle time was $24.70s$, which supposes an increase of $2.38s$ compared to the average. However, we can see that even in the worst case scenario, the system meets the cycle time. It should be noted that the maximum execution times for each phase may not have been obtained in the same run.

As can be seen, the most time-consuming phases were especially those that required the movement of the robot (i.e. phases 3, 4, 5, 8 and 9). Especially, the most time-consuming phase was the execution of grasping, as the robot makes several grasping attempts in case it is not able to grasp an object the first time. The phases of grasping point identification and EMS estimation were executed in parallel while the robot is moving between zones, thus not

Table 4 Average, standard deviation and maximum duration of each phase of the application

Phase	Task	$\bar{t}(s)$	$\sigma(s)$	Max.(s)
1	Capture inbound scene	0.8151	0.0545	0.9437
	GPD	0.3018	0.0210	0.3555
2	Capture outbound scene	0.8248	0.0492	0.9736
	EMS estimation	0.071	0.0302	0.1819
3	Move to zone 1	2.9987	0.0212	3.0680
4	Execute grasp	4.711	1.1455	7.3174
5	Move to zone 2	4.8324	0.0200	4.8899
6	Capture bbox scene	0.3002	0.0435	0.3814
	Estimate bbox	0.1169	0.0365	0.1836
7	Packing pose estimation	0.2089	0.0915	0.4638
8	Move to zone 3	5.4005	0.0198	5.4445
9	Execute pack	3.9558	0.2968	4.7045
Total	–	22.326	1.1689	24.7017

influencing the cycle time. This was possible because the computation time required for these two phases, even in the worst case, was considerably less than the duration of the robot's movement. However, the monitoring of the grasped object did take place online, and in the worst case, it could take up to $0.56s$.

The cycle times obtained for our system show that there is little variation in time. Furthermore, it can be seen that even in the worst case our system meets the target cycle time.

6 Conclusion and further work

This work describes the dynamic bin-packing system that was developed and validated in the context of the PICKPLACE European project. To the best of our knowledge, this is the first time a complete online bin-packing system has been deployed to a real scenario which allows creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system.

The software architecture is based on ROS and follows the layered architecture proposed in *Robotframework*. In this work, three main modules of the packing system are presented that let us pack a wide variety of objects in an online manner and without the need of ad hoc configurations. The developed system has been; successfully deployed and validated in the real robotic environment, where 7 tests were performed and recorded in video.

In the recorded videos, it can be observed that the main strength of the system was the flexibility, being able to pack online a wide variety of items. The analysis of the destination bin at each iteration let us consider fallen objects or unexpected movements inside the bin that happened after releasing the parts. In spite of the fact that the system had shown to be useful and flexible, there were some limitations that led to a sub-optimal space utilisation in the destination bin:

Picking order and part orientation As expected, the picking order had a large effect in the final mosaic. As can be seen in tests 2 and 4, when the products were manually ordered from big to small forcing the robot to pick them in order, the quality of the final mosaic improved. In addition, the orientation of the products in the inbound box had a big impact in the stability of the mosaic. Nevertheless, those are aspects that directly affect both the quality/stability of the mosaic and the space utilisation, but are difficult to improve in a multi-reference to multi-reference setup. The grasping system has the capability to decide to some extent the picking order, but this is limited to the visible and graspable objects in the inbound bin.

Inaccurate bounding box estimation Although very accurate industrial cameras were used for both grasping point estimation and outbound box monitoring, this was not the case for the bounding box estimation, where cheaper *RealSense D435* cameras were used. The depth quality of those cameras was not good enough (up to 1 cm of error in our bounding box estimation region) and usually led to bigger estimations. This caused the object to be released in an inaccurate pose and this had a direct effect in the separation between objects.

Product's weight On the one hand, depending on the weight of the product, the movement of the robot towards the bounding box estimation area sometimes caused the product to oscillate. This oscillation led to inaccuracies in the estimation of the bounding box. On the other hand, if the height of the product was not properly estimated and the product was released further than expected, the weight of the dropped product could cause the object to end in an unexpected final position. Although the online outbound container monitoring considered these movements for the next iterations, the quality of the mosaic was negatively affected.

Occlusions The online mosaic planner decided the optimal release position for the picked part at each iteration, only based on the outbound box monitoring and the bounding box estimation. Due to several criteria that are used to determine the release position, many times object towers were created which increased the likelihood to create occlusions.

External lighting The changing lighting conditions had a large effect on the 3D depth cameras. Although the pilot was covered with a housing and some noise filtering algorithms were used, this caused an increase in the sensor noise. This led to inaccurate depth estimations, particularly with shiny objects, which could cause collisions in the destination box.

Regarding the timeliness of the system, the experimentation carried out shows that there is very little variation in the cycle times. However, we are aware that in order to industrialise such an application and guarantee a fixed cycle time, a real-time implementation in a framework such as ROS2 [26] would have to be used.

In this article, we only considered the suction end effector to pick and place the items; nonetheless, it would be interesting to also consider other end effectors such as grippers. The gripper, compared to suction, needs at least two contact points to manipulate the items and this would introduce more complexity to the system. Furthermore, those contact points are usually located on the perimeter of the parts, which complicates the packing of objects close to each other.

The online mosaic planner plays a key role in the mosaic generation process since it is in charge of deciding the final position for each picked object. This work was conditioned by the UHS's dynamic IK-PAL planner. Although in most cases it offered coherent results, sometimes it was difficult to understand the reason behind some decisions made by the planner. Therefore, the use of an open source planner would improve the traceability of the system. To the best of our knowledge, there is no planner with the requirements of our system, and this would require the implementation of a new planner from scratch.

Author contribution Conceptualisation: A.I., A.F. and I.M.; methodology: A.I., E.L. and A.A.; implementation: A.I. and A.F.; original draft preparation: A.I.; review and editing: E.L. and A.A.; supervision: I.M.

Funding This article has been funded by the European Union's Horizon 2020 research and Innovation Programme under grant agreement No. 780488, and the project "5R- Red Cervera de Tecnologías robóticas en fabricación inteligente", contract number CER-20211007, under "Centros Tecnológicos de Excelencia Cervera" programme funded by "The Centre for the Development of Industrial Technology (CDTI)".

Declarations

Competing interests The authors declare no competing interests.

References

1. Iriondo A, Lazkano E, Ansuategi A (2021) Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications. Sensors 21(3):816. https://doi.org/10.3390/s21030816
2. Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. Manag Sci 44(3):388–399. https://doi.org/10.1287/mnsc.44.3.388
3. Martello S, Pisinger D, Vigo D (2000) The three-dimensional bin packing problem. Oper Res 48(2):256–267. https://doi.org/10.1287/opre.48.2.256.12386
4. Den Boef E, Korst J, Martello S, Pisinger D, Vigo D (2005) Erratum to the three-dimensional bin packing problem: robot-packable and orthogonal variants of packing problems. Oper Res 53(4):735–736. https://doi.org/10.1287/opre.1050.0210
5. Coffman EG, Garey MR, Johnson DS (1984). In: Ausiello G, Lucertini M, Serafini P (eds) Approximation algorithms for bin-packing–an updated survey. Vienna: Springer Vienna; p 49–106. Available from. https://doi.org/10.1007/978-3-7091-4338-4_3
6. Jakobs S. (1996) On genetic algorithms for the packing of polygons. Eur J Oper Res 88(1):165–181. https://doi.org/10.1016/0377-2217(94)00166-9
7. Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. J Comput 3(4):299–325. https://doi.org/10.1137/0203025
8. Ali S, Ramos AG, Carravilla MA, Oliveira JF (2022) On-line three-dimensional packing problems: a review of off-line and on-line solution approaches. Comput Ind Eng. p 108122. https://doi.org/10.1016/j.cie.2022.108122
9. Crainic TG, Perboli G, Tadei R (2009) TS2PACK: a two-level tabu search for the three-dimensional bin packing problem. Eur J Oper Res 195(3):744–760. https://doi.org/10.1016/j.ejor.2007.06.063
10. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for the three-dimensional bin-packing problem. INFORMS J

Comput 15(3):267–283. https://doi.org/10.1287/ijoc.15.3.267.16080

11. Gonċalves JF, Resende MG (2013) A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 145(2):500–510. https://doi.org/10.1016/j.ijpe.2013.04.019
12. Hu H, Zhang X, Yan X, Wang L, Xu Y (2017) Solving a new 3d bin packing problem with deep reinforcement learning method. ArXiv.1708.05930
13. Zhao Y, Rausch C, Haas C (2021) Optimizing 3D irregular object packing from 3D scans using metaheuristics. Adv Eng Inform 47:101234. https://doi.org/10.1016/j.aei.2020.101234
14. Wang F, Hauser K (2019) Stable bin packing of non-convex 3D objects with a robot manipulator. In: International conference on robotics and automation (ICRA). IEEE; p 8698–8704 Available from. https://doi.org/10.1109/ICRA.2019.8794049
15. Ha CT, Nguyen TT, Bui LT, Wang R (2017) An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In: European conference on the applications of evolutionary computation. Springer; p 140–155. Available from. https://doi.org/10.1007/978-3-319-55792-2_10
16. Wang F, Hauser K (2020) Robot packing with known items and nondeterministic arrival order. Trans Autom Sci Eng 18(4):1901–1915. https://doi.org/10.1109/TASE.2020.3024291
17. Hong YD, Kim YJ, Lee KB (2020) Smart pack: online autonomous object-packing system using RGB-D sensor data. Sensors 20(16):4448. https://doi.org/10.3390/s20164448
18. Duan L, Hu H, Qian Y, Gong Y, Zhang X, Wei J et al (2019) A multi-task selected learning approach for solving 3D flexible bin packing problem. In: Proceedings of the international conference on autonomous systems and multiagent systems (AAMAS) Available from. https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1386.pdf
19. Zhao H, Zhu C, Xu X, Huang H, Xu K. (2022) Learning practically feasible policies for online 3D bin packing. Sci China Inf Sci 65(1):1–17. https://doi.org/10.1007/s11432-021-3348-6
20. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J et al (2009) ROS: an open-source robot operating system. In: ICRA workshop on open source software. vol 3. Kobe, Japan. p 5. Available from. http://robotics.stanford.edu/ang/papers/icraoss09-ROS.pdf
21. Martin J, Ansuategi A, Maurtua I, Gutierrez A, Obregón D, Casquero O et al (2021) A generic ROS-based control architecture for pest inspection and treatment in greenhouses using a mobile manipulator. IEEE Access 9:94981–94995. https://doi.org/10.1109/ACCESS.2021.3093978
22. Parreño F, Alvarez-Valdés R, Oliveira JF, Tamarit J. M. (2010) A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. Ann Oper Res 179(1):203–220. https://doi.org/10.1007/s10479-008-0449-4
23. Gonçalves JF, Resende MG (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 39(2):179–190. https://doi.org/10.1016/j.cor.2011.03.009
24. Susperregi L, Fernandez A, Molina J, Iriondo A, Sierra B, Lazkano E, et al. (2020) RSAII: flexible robotized unitary picking in collaborative environments for order preparation in distribution centers. In: Bringing innovative robotic technologies from research labs to industrial end-users. Springer; p 129–151. Available from. https://doi.org/10.1007/978-3-030-34507-5_6
25. Rusu RB, Cousins S (2011) 3d is here: point cloud library (pcl). In: International conference on robotics and automation. IEEE; p 1–4. Available from. https://pointclouds.org/assets/pdf/pcl_icra2011.pdf
26. Macenski S, Foote T, Gerkey B, Lalancette C, Woodall W (2022) Robot Operating System 2: design, architecture, and uses in the wild. Sci Robot 7(66):eabm6074. https://doi.org/10.1126/scirobotics.abm6074