Jampacker: An efficient and reliable robotic bin packing system for cuboid objects

Marichi Agarwal, Swagata Biswas, Chayan Sarkar, Sayan Paul, and Himadri Sekhar Paul

***Abstract*—Bin packing using a robotic arm is an important problem in the context of Industry 4.0. In this work, we present a reliable and efficient bin packing system, called *Jampacker*. We propose a new offline 3D bin packing algorithm, called *Jampack* that achieves higher packing efficiency in comparison to the state-of-the-art algorithms. Jampack computes placement points, called *internal corner points* that tries to maximize the utilization of free spaces in-between objects, which are generally ignored by existing algorithms. Additionally, we introduce a fault recovery module (FRM) for the robotic manipulator that helps to achieve a more reliable and efficient packing in a physical container. The FRM module monitors the object placement by the robotic arm, calculates a fault score for the placement, adjusts the placement if required, and also learns & adjusts the offset for the placement procedure on-the-go. We show that this system achieves faster completion of the overall packing process.**

***Index Terms*—Factory automation, Three-dimensional bin packing, Logistics, Industrial robots, Fault tolerant manipulation.**

I. Introduction

AUTOMATION in the warehouse has various aspects where *pick and pack* automation are the most imperative ones. Modern warehouses use robotic fleets to fetch objects from the storage racks spread across a large area. Many existing studies focus on efficient goods movement (*picking problem*) using a team of robots in typical fulfillment warehouses where the number of customers is large, and the order volume per customer is very small [1]. On the other hand, warehouses that serve as replenishment centers usually have a smaller customer base and large order volume per customer. In such scenarios, the *packing problem* has a higher emphasis. In this article, we focus on efficient and autonomous object packing using a robotic arm.

Motivation. Each order from customers at a replenishment warehouse contains a large variety of objects (of different quantities and sizes), and they are packed in larger boxes (bins) before being transported to customers. Packing efficiency involves both space and time. Better space utilization not only reduces the required number of bins to pack all the objects, but it can also reduce the transportation cost. On the other hand, faster completion of packing impacts the overall order processing time significantly. 3D bin-packing problem (3D-BPP) is known to be NP-hard [2], [3] and several heuristics have been proposed to achieve a feasible solution [4], [5]. However, even if a 3D-BPP achieves a high space utilization, the physical process of packing can be impacted by the imperfection of the robotic arm or manipulator.

Problem description. Fig. 1 depicts a schematic of the bin packing system along with overall system architecture. Since the objective of any 3D-BPP algorithm is to maximize space utilization, the placement strategy (coordinate) it generates may leave no room between objects. The planner assumes the manipulator to be perfect and can place any object at the exact coordinate and with perfect orientation. However, physical manipulators are never perfect. A slight displacement of one object may disrupt the rest of the physical packing. To mitigate imperfection in a manipulator, it can be aided by employing a sensor-based feedback mechanism. Based on the sensor feedback, the manipulator can readjust the placement of the object at the desired location within a tolerable error. Obviously, the higher the tolerable error, the lesser the readjustment effort. This leads to the scenario where either not all objects are packed, or extra space has to be allocated to accommodate the higher tolerable error. On the other hand, the lower tolerable error leads to a large number of readjustment effort. However, if the manipulator is adjusted on-the-go, it can save a lot of this effort and reduce the overall packing time while achieving a high space utilization.

The major contributions of this work are as follows.

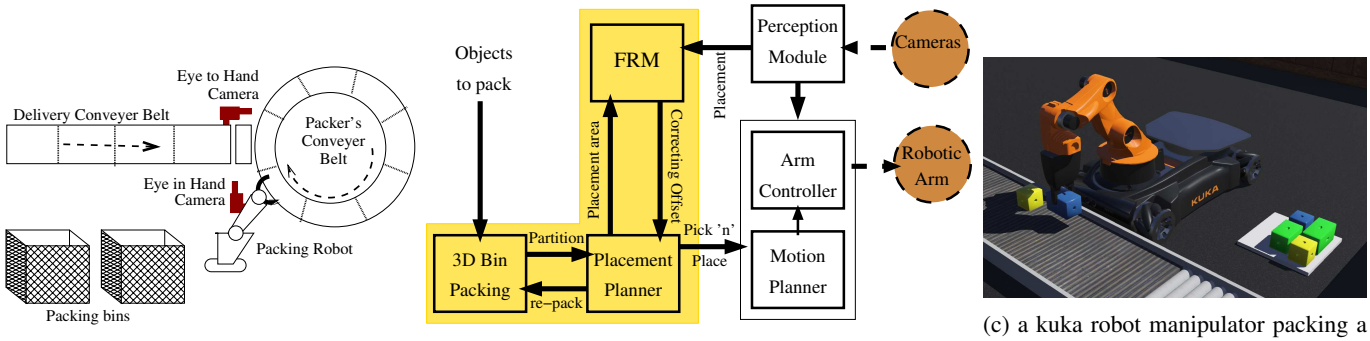
- We present an application-agnostic 3D bin-packing algorithm (Jampack) for cuboid objects that achieves higher space efficiency compared to the state-of-the-art. Most of the 3D-BPP heuristics first sort the objects based on criteria and then pack the objects one-by-one. They maintain the set of packing points and find the most suitable point among the feasible packing points for the current object to be packed. We propose a new method called *internal corner points* that finds and utilizes the packing points coherently, which leads to higher packing efficiency.
- We also present a fault recovery module (FRM) for robotic arms that can mitigate error in manipulation by dynamic learning and adjustment. An independent sensory system (e.g., vision-based) observes, analyzes, and provides feedback on the placement of an object to FRM. Based on this feedback, FRM learns and estimates an effective error in placement by the arm and computes an appropriate offset correction for the manipulator. Then, the manipulator can readjust the placement position of subsequent objects resulting in more accurate placement. As a result, the FRM reduces readjustment effort for wrong object placement, which in turn improves the overall packing time.

Manuscript received: July 5, 2020; Revised: October 7, 2020; Accepted: November 9, 2020.

This paper was recommended for publication by Editor Youngjin Choi upon evaluation of the Associate Editor and Reviewers' comments.

M. Agarwal, S. Biswas, C. Sarkar, S. Paul, and H. S. Paul are with the Embedded Systems & Robotics Lab, TCS Research & Innovation, India.

Digital Object Identifier (DOI):



(a) schematic of a 3D-bin packing system (b) modular components of a 3D-bin packing system (c) a kuka robot manipulator packing a bin in simulation

Fig. 1: Jampacker – (a) an overview of the 3D-bin packing system, (b) highlighted modules showing scope of this article, (c) a simulation setup depicting the system where sides of the bin are removed for better visualization.

- Finally, we propose an efficient bin packing system, called *Jampacker*, which combines both Jampack and FRM with a robotic arm motion planner that results in near-perfect packing with minimal overall packing time.

II. A Brief Literature Survey

The problem of robot packing involves two sub-problems – bin packing problem and robotic manipulation. Bin packing problem is a classical combinatorial optimization problem with NP-Hard complexity. According to the typology proposed by Wäscher *et al.* [6], bin packing problems can be classified into three types based on the properties of the bins – (i) single bin packing problem (BPP) with strictly homogeneous bins, (ii) multiple bin packing problem (MBPP) with weakly heterogeneous bins, and (iii) residual bin packing problem (RBPP) with strongly heterogeneous bins. For all three types, the objective of minimizing the number of bins remain the same. The problem has been adapted for 1D [7], [8], 2D [9], [10], 3D rectilinear [11] and 3D non-convex [12] type objects or packing by features such as weight or volume.

The state-of-the-art exact algorithms for the offline 3D bin packing use a branch-and-bound approach [13], [14]. Exact methods have exponential complexity and fail to generate the results within an acceptable time. Thus, various heuristic and meta-heuristic approaches have been developed that work close to the exact solution [15]. Some of the well-known heuristics are first-fit decreasing [16], deepest-bottom-left-fill (DBLF) [17], etc. Martello *et al.* [5] proposed a heuristic that packs objects in the form of layers and combined the layers. The objects are packed using the concept of corner points (CPs) where a CP is a coordinate in the residual space of the bin where an object can be accommodated. Recently, Mahvash *et al.* [11] proposed a technique to pack 3D objects in identical bins. The objects can be rotated in all directions and placed in the bins in six orientations. They utilized the concept of extreme points (proposed by Crainic *et al.* [4]), an extension of the corner point method, and fuse it with linear programming to find an efficient solution. We propose a better approach to find the corner points, called *internal corner point* and achieves a higher packing efficiency.

Some evolutionary methods are also proposed by researchers, such as simulated annealing [18], particle swarm optimization [19], [20], ant colony optimization [21], nature-inspired meta heuristic [22], etc. However, these methods usually take a long time to find a solution, and often the solutions are not as good as the classical techniques.

Elhedhli *et al.* [23] and Gzara *et al.* [24] propose a 3-dimensional bin-packing using Layer Based Column Generation approach. Layers are generated heuristically using a relaxed version of the pricing problem as well as using the Maxrects heuristic [25]. At the end of the column generation procedure, a set of layers of objects are generated. The bin construction heuristic selects layers ordered in descending order of density and places them sequentially in a set of open bins. However, it does not guarantee proper utilization of space between the layers.

Accuracy of a robot is the measure of its ability to reach a specified task-space and orientation [26]. Inaccuracies in robotic tasks generally occur due to a combination of joint error, manufacturing error, and arm deflections. The error analyses lead to the development of a feasible joint-tolerance domain and authors in [27] present a mathematical treatment of the subject. A more application-oriented study of the field is presented by Zou *et al.* [28] where they study placement error by robotic arms using vision-based sensing. A machine vision-based fault tolerance mechanism has been proposed in [29]. Their proposed technique involves continuous monitoring of arm angles, estimating deviations, and they control the ongoing movement of a joint. A generic study of fault tolerance in robotics can be found in [30]. The basic architecture of the FRM is inspired by those presented in the study.

III. Problem Definition

We address the problem of packing a set of rectilinear objects of varied dimensions into bins of fixed dimensions. The objects are to be packed by a robotic arm to achieve operational efficiency. The overview of our system is shown in Fig. 1. Since all the objects are known before-hand, we need an offline 3D bin-packing method that can pack all the objects in a minimum number of bins. Additionally, we provide the location of each object inside the bins and the placement sequence such that the placement satisfies the constraints associated with robotic arm movements. As the robotic arm can inject some error while executing, i.e., off-location placing

Algorithm 1: *Jampack*: A heuristic for generating packing pattern for a given set of objects.

Input: set of sorted objects to be packed (O), bin dimension (W, L, D).

Output: object list (O) updated with the bin id in which it is packed and packing location.

```
1  Procedure Jampack(O, W, L, D)
2      B ← φ;
3      newBin = openNewBin(W, L, D);
4      newBin.icp ← [[0, 0, 0], [W, L, D]];
5      B ← B + newBin;
6      for (o ∈ O) do
7          for b ∈ B do
8              for c ∈ b.icp do
9                  for each orientation do
10                     if canAccommodate(o, c) then
11                         assignPackingPoint(o, c);
12                         b.icp ← updateICP(o, b.icp);
13                         removeDuplicateICP(b.icp);
14                         goto line 6;
15             if B.index(b) == len(B) − 1 then
16                 newBin = openNewBin(W, L, D);
17                 newBin.icp ← [[0, 0, 0], [W, L, D]] ;
18                 B ← B + newBin;
19     return O;
```

of the objects; correct and reliable packing is another essential component of our system.

Given a set of n heterogeneous objects with dimension $w_i \times h_i \times d_i, (i \in 1, 2, ..., n)$ to be packed into homogeneous bins of dimension $W \times H \times D$. The objects are placed in the bins subject to the following constraints – (i) each object is placed inside only one bin, (ii) object lies within the limits of the bin, (iii) the objects are not rotated, and (iv) there is no overlap of objects.

If the placing point of the i^{th} object (back-left-bottom) is decided as $< x_i, y_i, z_i >$ and the actual placing point is $< x_i^{'}, y_i^{'}, z_i^{'} >$, we define two threshold values δ and δTH to decide the tolerance bounds for error in execution. Let T_i denote the volume occupied by item i when its positioning is transformed due to translational and rotational error.

IV. JAMPACK: AN EFFICIENT ALGORITHM FOR 3D-BPP

In this section, we describe *Jampack* that packs a set of cuboid objects in a minimum number of bins. Jampack comprises of three aspects – (a) object sorting criteria, (b) generating the set of packing points, and (c) finding the most suitable point among the feasible packing points for the next object to be packed. The efficiency of Jampack stems from our packing point generation technique, called *internal corners point* (ICP), which generates all the extreme points (EPs) [4] along with some additional feasible points.

A. Algorithm Description

Jampack initially sorts all the objects by non-increasing base area (with ties broken by height in non-decreasing order). Other sorting techniques as proposed by Crainic *et al.* [4] can also be used, but they do not seem to provide any distinct advantage over this one.

Given a list of sorted objects O and bins with dimension $W \times L \times D$, Algorithm 1 packs them using the concept of ICP. Jampack maintains a list of ICPs (x, y, z) (back-left-bottom corner) and their corresponding *bounding corner point* or BCP (x', y', z') (front-right-top corner) for each bin, where an ICP-BCP pair represents a contiguous free space along all the three axes. For an empty bin, its ICP-BCP pair is initialized with $(0, 0, 0)$ and (W, L, D).

For each object $o \in O$, Jampack scans through all the ICPs ($c \in b.icp$) for each bin ($b \in B$) and identifies the ICP, c, that can accommodate the object using procedure *canAccommodate* (line 10 in Algorithm 1). An object o with dimension (w_o, l_o, d_o) can be accommodated at an ICP if and only if, $(x' - x \geq w_o)$ and $(y' - y \geq l_o)$ and $(z' - z \geq d_o)$. If the object cannot fit at an ICP with the current orientation, it is rotated (line 9). Altogether an object can be placed in six different orientations. The highest priority is given to the ICP that places the object closer to the base and the left and back wall of the bin, i.e., the ICP with the lowest values of *z, y, x* (in this order). The chosen ICP is assigned as the packing point for the object (line 11). In case no feasible corner is found in any of the open bins, a new bin is opened, and $(0, 0, 0)$ is assigned as the packing point for the object.

Given the list of ICPs and the packing coordinate of the newly placed object, procedure *updateICP* (line 12) updates the existing ICPs (if required) and generates new ICP-BCP pairs (if any). In a nutshell, the idea is to split each of the overlapping vacant spaces by the plane drawn along each side of the object. The back-left-bottom corner of each of the new free volumes is the new ICPs. Details are in Algorithm 2. Finally, the duplicate and coinciding ICPs are removed by the procedure *removeDuplicateICP* (line 13). If no feasible packing point is found in any of the open bins, a new bin is opened (lines 15-18). The algorithm continues until all the objects are assigned a packing point in any of the bins.

B. Time Complexity Analysis

The outer loop in Algorithm 1 runs for all the objects to be packed, i.e., n iterations, where n is the total number of objects to be packed. The next two inner loops (line 7 and 8) runs for all the open bins in the system and the number of ICPs in the bin, respectively. The innermost loop (line 9) runs for the number of possible orientations of a cuboid object, i.e., six. Since it is a constant, it does not contribute to the complexity of the overall algorithm. Packing a new object adds at most three new ICPs to the bin's list. When there are k objects per bin, these two loops will run for $(n/k)*(k*3)$ iterations, which is $\mathcal{O}(n)$. Now, Algorithm 2 has the worst-case run-time of n when all the objects are packed in a single bin and there are $n*3$ corner points. This is called by Algorithm 1 for n^2 times in the worst case. Thus, the overall complexity of Jampack is $\mathcal{O}(n^3)$.

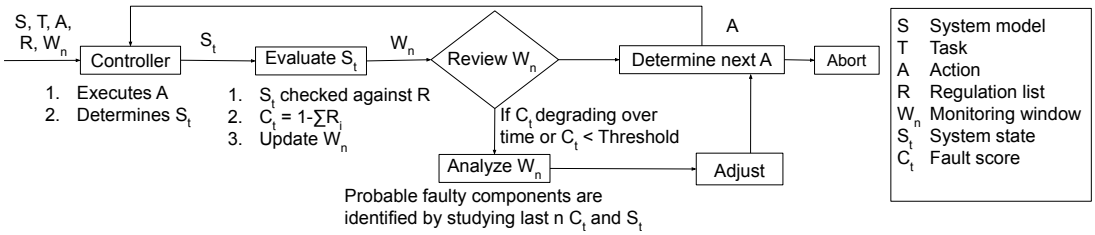


Fig. 2: Fault recovery module comprised of safety score estimation and adjustment on-the-go.

Algorithm 2: *Pseudo code for generating new ICPs and updating the existing ones.*

Input: object (o) with dimension (w_o, l_o, d_o) added to the bin at coordinate (x_o, y_o, z_o) , list of existing ICPs ($b.icp$) in bin (b).
Output: updated list of ICP-BCP pairs of bin b.

Procedure `updateICP`$(o, b.icp)$
  $newICP \leftarrow \phi$;
  for $(c \in b.icp)$ **do**
    if `isOverlapping`(o, c) **then**
      if $x_o > x_c$ **then**
        $icpBack = [[x_c, y_c, z_c], [x_o, y'_c, z'_c]]$;
        $newICP \leftarrow newICP + icpBack$;
      if $x_o + w_o < x'_c$ **then**
        $icpFront = [[x_o + w_o, y_c, z_c], [x'_c, y'_c, z'_c]]$;
        $newICP \leftarrow newICP + icpFront$;
      if $y_o > y_c$ **then**
        $icpLeft = [[x_c, y_c, z_c], [x'_c, y_o, z'_c]]$;
        $newICP \leftarrow newICP + icpLeft$;
      if $y_o + l_o < y'_c$ **then**
        $icpRight = [[x_c, y_o + l_o, z_c], [x'_c, y'_c, z'_c]]$;
        $newICP \leftarrow newICP + icpRight$;
      if $z_o > z_c$ **then**
        $icpBottom = [[x_c, y_c, z_c], [x'_c, y'_c, z_o]]$;
        $newICP \leftarrow newICP + icpBottom$;
      if $z_o + d_o < z'_c$ **then**
        $icpTop = [[x_c, y_c, z_o + d_o], [x'_c, y'_c, z'_c]]$;
        $newICP \leftarrow newICP + icpTop$;
      $b.icp \leftarrow b.icp \backslash c$
  $b.icp \leftarrow b.icp + newICP$;
  return $b.icp$;

V. Fault Recovery Module (FRM)

The objective of this module is to detect and measure deviations in object placement and then estimate a placement offset value to mitigate deviations in future placements. The general workflow of the fault recovery process for an on-line fault rectification is shown in Fig. 2 .

The FRM module is a software system (S) that works in tandem with the perception module, placement planner, and motion planner to identify the faults and attempt recovery. While executing a given task T, the system is required to abide by a set of regulations. Each of the regulations (say R_i) is associated with a measurement and two threshold and can be defined as a tuple $r =< r^i_m, r^i_{ff}, r^i_{fs} >$. The value r^i_m is a measure computed from single or multiple sensor readings available with the system S. The other two quantities are the thresholds to determine the state of the system, w.r.t. r^i_m. If $r^i_m \leq r^i_{nf}$ the system is in a non-faulty state (NF). When $r^i_m \leq r^i_{fs}$ the system is in faulty-but safe state (NS). Otherwise, the system is in a fault state (FT). Each of these states is associated with a penalty value and is denoted as p^i_s, where s indicates the state. Typically, $p^i_{NF} = 0$ and $p^i_{FS} < p^i_{FT}$.

A system's extent of fault is computed as an accumulation of penalty value for all defined regulations. The *Fault Score* of the system at instance t is computed as $C_t = \sum_{i=1}^{m} \alpha_i P_i$, where m is the number of regulations, P_i is the penalty of the system due to regulation R_i, and α_i is the relative importance of the regulation for the overall failure of the system. The FRM also maintains a *Monitoring Window*, $W_n = [< C_t, S_T >: t_i \leq t \leq t_j]$, where the system keeps a record of its fault scores and system states of last n steps. The sensor readings, software states, hardware states, etc., determine the system state.

Fig. 2 depicts the system model *S*, the assigned task *T*, the immediate next action *A*, regulation list *R*, and monitoring window size n that are given as input to the controller. The controller initiates action *A*, and records & checks the system state against the regulation list *R*. The fault score C_t is computed and evaluated against the latest $n-1$ score values. If a consistent degradation of the safety score over (C_t, C_{t-n}) is observed or the fault score C_t is below a given threshold C_{th}, the system state of last n steps are analyzed to identify the probable faulty component. Once the faulty component is identified, the controller re-adjusts the given system accordingly and determines the next action A. If the system is beyond repair then the controller aborts the task.

VI. Jampacker: The Bin-packing Framework

The overall bin-packing framework, whose architecture is shown in Fig. 1b, utilizes Jampack, a 3D-BPP solution, and FRM to achieve fault-tolerant compact packing. After, Jampack determines the sets of objects along with their packing points for each of the bins, the *placement planner* first, generates the packing sequence for the robotic arm. Then, the placement planner works in tandem with FRM and the motion planner, which takes feedback on the actual placement from the perception module, to determine the rectified placement location of the next object.

A. Placement Planner

This module generates a dependency plan amongst objects to be packed in a bin. Placement sequence is represented by a directed acyclic graph (DAG), $G(V, E)$ with vertices $V \in I$, where I is the set of packed objects and edge $e = (u, v)$

Algorithm 3: *FRM*: Placement of object with retry

Input: S : System Model, T : Assigned Task, R : Regulation List, W : Monitoring window
Output: Pos : Actual placement position of the object

Position robotic arm at sensing position;
Object Recognition;
Position arm at picking position;
Pick object and move to target location given in T;
Place Object;
Sense position of the placed object ;
SC = Evaluate R ;
if SC *indicates system faulty* **then**
    Compute position error E;
    Apply negated offset for E to target location;
    Pick object and place at new location;
Perform Steps 6-8 for up to k iterations. Manual intervention required if system is in faulty state after k attempts;
Pos = Observe bounding box of the object;
Update W with SC;
Return P;

connecting the vertices u and v if object u must be placed before object v. Before placing an object v, all its predecessor objects (the objects below it and on its left) must be placed. A buffer of 10% (determined in Section VII-C) is provided around each object (along the $X-Y$ plane) before generating the packing points. So the computed placement points for each object may need to be adjusted depending on the consumption of buffer space by the robotic arm for an object already packed. This planner module gives one object to pack to the arm and then receives feedback from the FRM on the actual placement position of an object. Based on this information, the placement position of the rest of the objects is adjusted by the planner.

B. FRM in Action

FRM observes every attempt to place an object inside a bin. When it finds that the system is in a faulty state due to placement error, i.e., the object's placement is not contained within its buffer area, it re-adjusts the placement points for the arm to enable a more accurate placement. The FRM module allows k re-attempts for an object after that manual intervention is required. After placement, it informs the planner module of the actual position of the object. The FRM process is depicted as Algorithm 3.

In this work, we are concerned only with the proper positioning of the object in the bin and consider the rest of the process error-free. Let the target drop location of the object O be (x_1,y_1,z_1) as specified by the *placement planner*. In the $X-Y$ plane, let the bottom co-ordinates of the object be $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ and (x_4,y_4). We ignore the error along the z-axis as it depends on the gravitational force and is beyond manual control. While placing the object in the defined location, the controller may encounter a translation error (ϵ_x,ϵ_y) followed by a rotational error θ along the vertical axis through the center of gravity of the object (p,q) around the origin as shown in Fig. 3.

Let the co-ordinates of the object in $X-Y$ plane after translation error be $(x_1',y_1'),(x_2',y_2'),(x_3',y_3')$ and (x_4',y_4'), and after rotational error be $(x_1'',y_1''),(x_2'',y_2''),(x_3'',y_3'')$ and (x_4'',y_4''). This can be evaluated as given in Eq. 1. As shown in Fig. 3, a tolerance of Δ is given to each of the object placed by the planner.

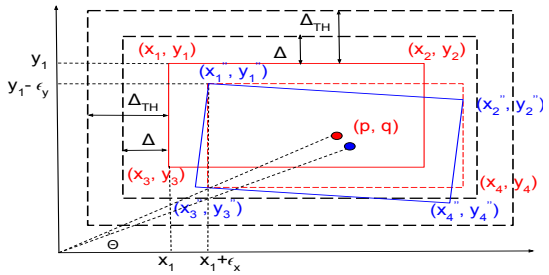


Fig. 3: Translation and rotational error handling.

$$
\begin{aligned}
x_i' &= x_i + \epsilon_x \qquad y_i' = y_i + \epsilon_y \\
p &= \frac{x_1' + x_2'}{2} \qquad q = \frac{y_1'+y_3'}{2} \\
x_i'' &= (x_i' - p)\cos(\theta) - (y_i' - q)\sin(\theta) + p \\
y_i'' &= (x_i' - p)\sin(\theta) + (y_i' - q)\cos(\theta) + q
\end{aligned} \tag{1}
$$

Here, a single rule is defined in the Regulation List. The bottom plane of the object should be placed within the *Tolerance Bounded Region* bounded by $(x_1-\Delta, y_1+\Delta), (x_2+\Delta, y_2+\Delta), (x_3-\Delta, y_3-\Delta)$ and $(x_4+\Delta, y_4-\Delta)$. If this condition is satisfied then *S* is in fault-free state ($s_{ff}=1$) and penalty is p_{ff}. If the object is not placed within the above mentioned bounded region but it is within the *Threshold Bounded Region*, $(x_1-\Delta_{TH}, y_1+\Delta_{TH})$, $(x_2+\Delta_{TH}, y_2+\Delta_{TH})$, $(x_3-\Delta_{TH}, y_3-\Delta_{TH})$ and $(x_4+\Delta_{TH}, y_4-\Delta_{TH})$, then *S* is in fault safe state, $s_{fs}=1$ and the penalty is p_{fs}.

This is the case shown in Fig. 3 and the action needs to be rectified, i.e., the action is re-executed to minimize the error. If the object is placed outside the *Tolerance Bounded Region*, then *S* is in faulty state, $s_f=1$ and the penalty given is p_f. If the first attempt to place the object results in a faulty state, the arm observes the displacement and computes a new location as a target based on the error observed. This is simply a negated error applied to the target location specified by the planner. The arm makes k such retries to place the object within the permitted placement area. The determination of the value k is an adjustment process of the arm and depends on the actual hardware component.

Initially, the fault score list for the monitoring window is assigned with value zero. The controller initiates action *A* and after the execution of *A*, the system state is recorded. The system state, in this scenario, is defined as the actual co-ordinates of the corners of the bottom surface of the object. The fault score is also evaluated as $C_i = 1-\alpha P$ where C_i refers to the fault score after executing i^{th} action, P refers to the penalty given, and α represents the weighting factor. Here $P = p_{ff}\times s_{ff} + p_{fs}\times s_{fs} + p_f \times s_f$ and $\alpha = 1$. The system state and the fault score for the same are then recorded in W_n. W_n records the system state and fault score for the last n actions. W_n is analyzed. If the fault score, C_i is less than the threshold value, C_{th}, then the system state is analyzed to identify the fault. The controller identifies the fault

in the positioning of the objects. According to the analyses, the controller adjusts the system, i.e., the drop location (x_1, y_1, z_1) are redefined as $(x_0+\delta_x, y_0+\delta_y, z_0+\delta_z)$ and proceeds to the next object packing action. The controller initiates a manual intervention if the fault is irreparable(for example, when the object is dropped outside the bin).

C. Motion Planner

We have used an off-the-shelf RRT motion planner (OMPL library) to solve the task at hand. We assume that the maximum reach of the robot and the initial robot position is such that the robot manipulator is capable of placing the objects in any position within the given bin. Now, given the initial and target positions, the planner samples in joint space, checks for collisions in task space, and provides the trajectory plan while taking care of the kinematic restrictions of the manipulator. The entire picking and placing task is divided into two parts, and the trajectory plans are generated for each separately. The first part of the trajectory planning involves picking up the object from the conveyor belt and positioning the arm at the fixed height above the bin and vice-versa after the placement of the current object. The second part involves moving the arm from the fixed height position above the bin to the placing location and vice-versa including the placement readjustment (k retries by FRM). The collision checks and plan generations are performed in between every two manipulations as the newly placed objects bin are also treated as obstacles during the planning of the next task.

VII. Evaluation

We evaluate Jampacker in three phases – (i) we compare the performance of Jampack with two state-of-the-art algorithms, (ii) we demonstrate the efficacy of FRM, and (iii) we showcase how the overall framework can achieve reliable and efficient packing.

A. Dataset and Methodology for 3D-BPP

We present our analysis for two categories of datasets.

(a) *standard datasets*: class 1 and classes 4-8 as described by Martello *et al.* [13] where the instances (object dimension and bin dimension) are generated using the available C code[1]. Bins are cubic in these instances. Following the experimental protocol of [4] and [13], we did not consider classes 2 and 3 because these have properties similar to those of class 1.

(b) *industry dataset*: the three categories of datasets *large, medium and small* that are provided by our client partner and were packed at their sorting center. The objects in this category are of varied dimensions, and the bin is relatively large. The classification of the objects as large ($35-50$ objects/bin), medium ($50-65$ objects/bin), and small ($65-80$ objects/bin); depends on how many objects are required to fill up a bin optimally. The bin size used is $220cm \times 120cm \times 80cm$ (a regular container size used by our client partner).

We randomly sampled 10 instances of each of the 9 different datasets, such that objects of each instance would optimally fill 20 bins. In total, we created 90 instances.

[1]http://www.diku.dk/~pisinger/codes.html

TABLE I: Comparison of average no. of bins used to pack all objects and average bin utilization of first 20 bins for various datasets using EP-FFD (EP), LCGA (LC), and Jampack (JP).

Dataset	**Avg. bin count**			**Avg bin utilization (%)**		
	EP	*LC*	*JP*	*EP*	*LC*	*JP*
Class 1	29.1	28.6	**25.5**	72.47	78.07	**81.76**
Class 4	39.4	39.2	**38.9**	64.16	61.5	**65.78**
Class 5	28	28.1	**26.4**	78.14	77.97	**83.4**
Class 6	25.2	24.2	**22.3**	85.58	86.17	**93.34**
Class 7	28.5	26.8	**24.6**	78.27	80.97	**87.9**
Class 8	28.3	27.2	**25.2**	78.02	79.19	**85.37**
large	26.2	24.9	**23.9**	78.12	82.7	**87.27**
medium	26.8	24	**23**	78.32	84.44	**88.83**
small	25.4	24	**23**	82.57	84.76	**89.47**

TABLE II: Object placement planning using EP-FFD and Jampack for objects shown in Fig. 4a and Fig. 4b.

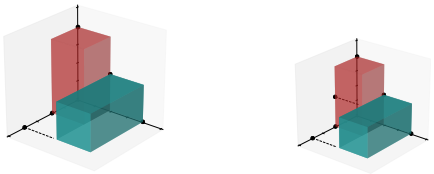
Placement sequence	Object dim. (w,h,d)	Extreme Point - FFD		Jampack	
		Placed at	Updated EP-RS	Placed at	Updated ICP-BCP
empty			(0,0,0)(10,10,10)		(0,0,0)(10,10,10)
1 (red)	(4,4,8)	(0,0,0)	(4,0,0)(10,10,10) (0,4,0)(10,10,10) (0,0,8)(10,10,10)	(0,0,0)	(4,0,0)(10,10,10) (0,4,0)(10,10,10) (0,0,8)(10,10,10)
2 (blue)	(8,4,4)	(0,4,0)	(4,0,0)(10,4,10) (0,0,8)(10,10,10) (8,0,0)(10,10,10) (0,8,0)(10,10,10) (0,4,4)(10,10,10)	(0,4,0)	(4,0,0)(10,4,10) (0,0,8)(10,10,10) (8,0,0)(10,10,10) (0,8,0)(10,10,10) (0,4,4)(10,10,10) **(4,0,4)(10,10,10)**
3	(4,8,4)	no corner found. open new bin		(4,0,4)	(4,0,0)(10,4,4) (0,0,8)(10,10,10) (8,0,0)(10,10,10) (0,8,0)(4,10,10) (0,8,0)(10,10,4) (0,4,4)(4,10,10)

B. Efficiency of Jampack

To scrutinize the efficacy of *Jampack*, we compare it with *extreme-point first-fit-decreasing* (EP-FFD) heuristic proposed by Crainic *et al.* [4] and *layer based column generation approach* (LCGA) proposed by Elhedhli *et al.* [23]. We evaluate Jampack based on two metrics – (i) the total number of bins required to pack all the objects, and (ii) the average bin utilization of the first 20 bins (since we sampled objects that can optimally fill 20 bins). Utilization for each bin is given by,

$$\text{utilization} = \frac{\text{total volume of the packed objects}}{\text{bin volume}} \times 100\%.$$

The results enlisted in TABLE I are averaged over 10 instances of each type of the 9 datasets (the reason for the *Avg. bin count* to be in decimal). Armed with empirical results, we



(a) EPs (b) ICPs

Fig. 4: EPs and ICPs generated by EP-FFD and Jampack respectively by placing two objects.

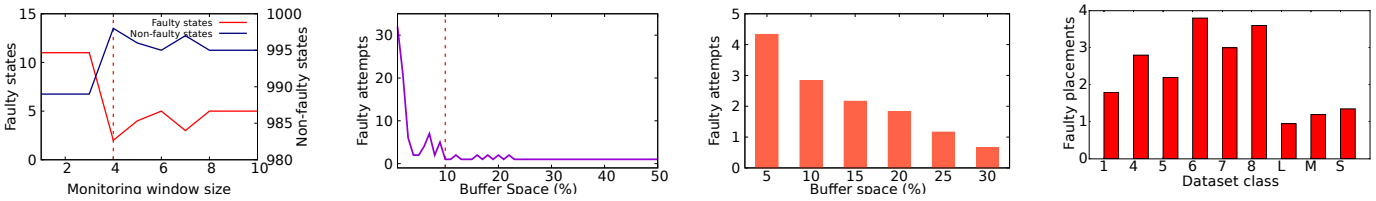


Fig. 5: Performance of FRM – (a) for varying window size, (b) for varying buffer space, (c) for varying buffer space of 1000 objects (window size 4), (d) for varying class of 1000 objects (buffer space 10%, window size 4).

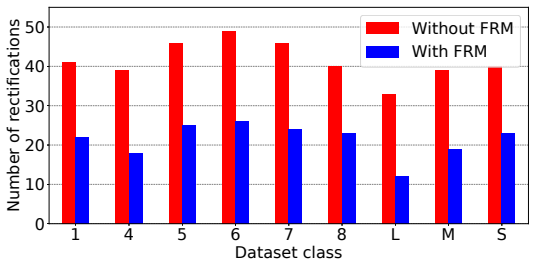


Fig. 6: Number of rectifications required to pack 50 objects.

state that Jampack outperforms both EP-FFD and LCGA for both *standard* and *industry* datasets in terms of the number of bin required and the bin utilization.

Packing point generation and selection proposition favor the overall efficiency of a bin packing algorithm. As EP-FFD does not generate all feasible packing points, which is added by Jampack, as shown in Fig. 4a and Fig. 4b, it requires more bins to pack all the objects. TABLE II elaborates a packing plan for the objects shown in Fig. 4a and Fig. 4b. To pack them, EP-FFD uses 2 bins whereas all three can be packed in a single bin using Jampack. With these additional corner points, Jampack increases the packing efficiency of the state-of-the-art EP-FFD algorithm by $1.62\%-10.51\%$. Additionally, Jampack efficiently utilizes the small vacant spaces in between objects to pack objects with smaller dimensions.

When the objects are of highly varying dimensions, it does not favor layering. In other words, there is always some wasted space when layers are built using heterogeneous types of objects. This is precisely the case in our dataset. As a result, Jampack outperforms LCGA. In particular, Jampack uses $1-3$ bins less and achieves $3.69\% - 7.17\%$ higher utilization per bin in comparison to LCGA.

C. Efficacy of FRM

We assume that the maximum reach of the robot and the initial robot position is such that the robot manipulator is capable of placing objects in any position within the given bin. As mentioned earlier, if an object is placed within the predefined buffer space, then it is considered to be a non-faulty placement; otherwise, it is considered as a faulty placement. In case of a faulty placement, the system re-attempts until the object is placed within the buffer space, and it initiates a manual intervention after k attempts. Also, the system re-tries to rectify the placement error at each attempt using its past state information from the monitoring window.

Monitoring window size and the buffer space are the two parameters that affect the efficacy of FRM. To determine the values of these parameters, we have simulated 1000 objects of variable dimensions and analyzed the effect of Gaussian distributed translation and rotational error while placing each object at a predefined location. The object placement error is due to the final positioning error of the robot end-effector. The sources of this error are varied ranging from manufacturing, assembling, installation, sensing to even temperature changing [31]. Therefore, to study the effect of the final positioning error of the robot end-effector, we have assumed that the translational and rotational error follows Gaussian distribution. Similar studies are also done in [32]. We empirically choose the minimum monitoring window size when the number of faulty states gets minimized and the number of non-faulty (fault-free and fault-safe) states get maximized (Fig. 5a). The minimum monitoring window size is chosen to decrease the computational cost. It is observed that the number of faulty states increases with window size. This is because the system encounters multiple faults prior to its rectification of error using its past state information from the monitoring window.

In Fig. 5b, we observe that the number of faulty placements reduces significantly when buffer space is 10% of the object area. The number of faulty placements further reduces and stabilizes at a buffer space of $\approx 25\%$. However, such a large buffer space will result in inefficient bin space utilization. This is also true for the real-world dataset (class 10) as shown in Fig. 5c. Therefore, for all our experiments, we have considered monitoring window size=4 and buffer space to be 10% of object dimension. We have tested and validated for different class of 1000 objects (Fig. 5d). The average of 10 simulation results is shown in Fig. 5c and Fig. 5d.

D. Overall performance of Jampacker

Jampacker is integrated with a motion planning algorithm and tested using Webots [33] simulation environment. Fig. 1c depicts our simulation setup using a KUKA robotic manipulator. We have tested the system using the datasets as described in Section VII-A. We have also considered the monitoring window size 4 and 10% buffer space as discussed in Section VII-C. The result is outlined in Fig. 6. Here, we have compared the number of rectifications required to pack the bin with and without FRM. The number of rectifications refers to the number of faulty attempts and re-attempts made to

place the objects within their predefined buffer space. Without FRM, the Kuka robot plans to place the object in the target location. If it fails to place it within the predefined buffer space, it re-attempts to place it in the target location ignoring the knowledge gained from previous placement attempts. This process continues until the Kuka robot successfully places the object within the predefined buffer space. On the contrary, the FRM module learns from its previous actions and readjusts as explained in Section V to minimize the number of rectifications required. Fig. 6 clearly depicts that the number of rectifications required is significantly less when we deploy the FRM module. The number of rectifications required is directly proportional to the time required for packing. Therefore, the FRM module reduces the overall bin packing time and also minimizes the number of manual interventions required. After the packing, we evaluate the difference between the planned buffer space and the actual buffer space utilized. This helps in estimating the optimal buffer space required to pack a bin. It is observed that in certain cases the buffer space saved ranges between $2\% - 5\%$. The estimated saved buffer space can be reused to pack another object in the same bin. Also, this state information can be reused to pack the next bin. Hence, ensuring maximal bin space utilization.

VIII. Conclusions

In this work, we describe Jampacker, a robotic arm based automatic packer. The system comprises of two major components – an efficient 3D bin packing (3D-BPP) algorithm, called Jampack, and a fault recovery module (FRM) for the robotic arm. Using our proposed method of *internal corner points*, Jampack performs better than the state-of-the-art algorithm; thus applicable to any generic 3D-BPP application. On the other hand, we enhance the reliability of a robotic manipulator by estimating and adjusting the possible error of placing an object on-the-go, which is also application-agnostic. Our system reduces the overall time of readjustment after placing an object incorrectly, resulting in overall faster completion of packing. In the future, we would be extending this work with actual sensor-based feedback for motion planning as well as to object identification and displacement detection inside a bin. Also, we shall extend the FRM by considering sensor accuracy.

References

[1] C. Sarkar and M. Agarwal, "Cannot avoid penalty? let's minimize," in *2019 Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1052–1058.

[2] S. Martello *et al.*, "The three-dimensional bin packing problem," *J. of the Operational Research Soc.*, vol. 48, no. 2, pp. 256–267, 2000.

[3] T. Crainic *et al.*, *Recent advances in multi-dimensional packing problems*, 2012.

[4] T. G. Crainic *et al.*, "Extreme point-based heuristics for three-dimensional bin packing," *Informs J. on Computing*, vol. 20, no. 3, pp. 368–384, 2008.

[5] S. Martello *et al.*, "Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 1, p. 7, 2007.

[6] G. Wäscher *et al.*, "An improved typology of cutting and packing problems," *Eur. J. of Operational Research*, vol. 183, no. 3, pp. 1109–1130, 2007.

[7] D. S. Johnson *et al.*, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM J. on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[8] A. C. Alvim *et al.*, "A hybrid improvement heuristic for the one-dimensional bin packing problem," *J. of Heuristics*, vol. 10, no. 2, pp. 205–229, 2004.

[9] Y.-P. Cui *et al.*, "Sequential heuristic for the two-dimensional bin-packing problem," *Euro J of Operational Research*, vol. 240, no. 1, pp. 43–53, 2015.

[10] A. Lodi *et al.*, "Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints," *Discrete Applied Mathematics*, vol. 217, pp. 40–47, 2017.

[11] B. Mahvash *et al.*, "A column generation-based heuristic for the three-dimensional bin packing problem with rotation," *J. of the Operational Research Soc.*, vol. 69, no. 1, pp. 78–90, 2018.

[12] F. Wang and K. Hauser, "Stable bin packing of non-convex 3d objects with a robot manipulator," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8698–8704.

[13] S. Martello *et al.*, "The three-dimensional bin packing problem," *J. of the Operational Research Soc.*, vol. 48, no. 2, pp. 256–267, 2000.

[14] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *Euro J. of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[15] L. Wang *et al.*, "Two natural heuristics for 3d packing with practical loading constraints," in *Pacific Rim Intl Conf. on AI*. Springer, 2010, pp. 256–267.

[16] B. Xia and Z. Tan, "Tighter bounds of the first fit algorithm for the bin-packing problem," *Discrete Applied Mathematics*, vol. 158, no. 15, pp. 1668–1675, 2010.

[17] L. Wang *et al.*, "Two natural heuristics for 3d packing with practical loading constraints," in *Pacific Rim Intl. Conf. on AI*. Springer, 2010, pp. 256–267.

[18] T. Kämpke, "Simulated annealing: use of a new tool in bin packing," *Annals of Operations Research*, vol. 16, no. 1, pp. 327–332, 1988.

[19] D. Liu *et al.*, "On solving multiobjective bin packing problems using evolutionary particle swarm optimization," *Euro J of Operational Research*, vol. 190, no. 2, pp. 357–382, 2008.

[20] T. S. Li, C. Liu, P. Kuo, N. Fang, C. Li, C. Cheng, C. Hsieh, L. Wu, J. Liang, and C. Chen, "A three-dimensional adaptive pso-based packing algorithm for an iot-based automated e-fulfillment packaging system," *IEEE Access*, vol. 5, pp. 9188–9205, 2017.

[21] J. Levine and F. Ducatelle, "Ant colony optimization and local search for bin packing and cutting stock problems," *J. of the Operational Research Soc.*, vol. 55, no. 7, pp. 705–716, 2004.

[22] M. Abdel-Basset *et al.*, "An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems," *Personal and Ubiquitous Computing*, vol. 22, no. 5-6, pp. 1117–1132, 2018.

[23] S. Elhedhli *et al.*, "Three-dimensional bin packing and mixed-case palletization," *Informs J. on Optimization*, vol. 1, no. 4, pp. 323–352, 2019.

[24] F. Gzara *et al.*, "The pallet loading problem: Three-dimensional bin packing with practical constraints," *Euro J. of Operational Research*, 2020.

[25] J. Jylänki, "A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing," *retrived from http://clb.demon.fi/files/RectangleBinPack.pdf*, 2010.

[26] R. G. Fenton, "Accuracy and repeatability of robots," Un. of Toronto, ON, Tech. Rep., 1984.

[27] B. Benhabib *et al.*, "Computer-aided joint error analysis of robots," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 317–322, 1987.

[28] X. Zou *et al.*, "Fault-tolerant design of a limited universal fruit-picking end-effector based on vision-positioning error," *Applied Engg. in Agriculture*, vol. 32, no. 1, pp. 5–18, 2016.

[29] X. Li *et al.*, "Fault-tolerant control method of robotic arm based on machine vision," in *Chinese Control and Decision Conference (CCDC)*, 2018, pp. 484–489.

[30] M. L. Visinsky *et al.*, "Robotic fault detection and fault tolerance: A survey," *Reliability Engg & System Safety*, vol. 46, no. 2, pp. 139–158, 1994.

[31] Z. Wang *et al.*, "Industrial robot trajectory accuracy evaluation maps for hybrid manufacturing process based on joint angle error analysis," 2018.

[32] J. Wu *et al.*, "A computational framework of kinematic accuracy reliability analysis for industrial robots," *Applied Mathematical Modelling*, vol. 82, pp. 189–216, 2020.

[33] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *Intl J of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.