Mixed Palletizing and Task Completion for Virtual Warehouses

Girum Demisse, Razvan Mihalyi, Billy Okal, Dev Poudel, Johannes Schauer, and Andreas Nüchter

Abstract— **Palletizing, or packing rectangular boxes of various sizes onto pallets, is a frequently encountered task in many commercial scenarios, e.g., shipment of goods. An extension of this problem is the automated placement of boxes on pallets, henceforth task completion, by means of industrial robot arms. The palletizing and task completion problems are treated in the context of the *IEEE ICRA 2012 Virtual Manufacturing Automation Competition*. We approach the palletizing challenge by a winner-takes-all strategy, where multiple heuristics are evaluated against the given datasets. Our results show a performance comparable to that of the commercial software benchmark from the previous competitions. We solve task completion in USARSim by picking boxes and placing them on a pallet. Our inverse kinematics algorithm consists of a geometric and numeric component.**

I. Introduction

Palletizing items of different sizes, i.e., mixed palletizing, is a common challenge that occurs in many commercial scenarios. Examples are shipping goods from one part of the warehouse to another, or loading containers and delivery vehicles. The shape of goods varies in many manufacturing setups but rectangular cuboids appear most often and hence most of the research in this area focuses on cuboid box packing. Related problems also occurs in other domains, such as VLSI circuit design for packing elements on circuit boards or in paper mills for minimizing the waste when cutting rolls of paper.

Mobility, on the other hand, is the task of moving a pallet, on which items have been packed, from one part of the warehouse to another, usually using a fleet of autonomous robots. As a preliminary step to the mobility task, items need to be placed on a pallet. This is referred to as task completion and is done by using industrial robot arms with robust manipulation capabilities. Task completion involves picking items that arrive on a conveyor belt, or are placed at some location in the scene, and placing them onto the pallet.

In this paper, we describe the approach of the Jacobs University Bremen team for the VMAC palletizing challenge and present the task completion component. In solving the palletizing problem, we implement a decision algorithm that evaluates multiple heuristics for solving the 3D bin packing problem and applies the solution with the highest score. We ensure a minimal execution time through a multi-threaded implementation and we facilitate the extension of the algorithm with new heuristics through a modular software architecture. For task completion, we use an USARSim setup, where we simulate a conveyor belt and allow the KUKA KR60 robot to grip boxes and place them on a pallet. To solve the forward and inverse kinematics for the 6 degrees of freedom (DOF) robot, we implement a geometric and a numeric approach, that we use alternatively, to compensate for each other's weaknesses.

All authors are with the Jacobs University Bremen gGmbH, Germany {a.nuechter@jacobs-university.de}

II. Related Work

Extensive research has been done on the geometric version of the palletizing problem as manifest by the numerous literature occurrences that address both the 2-D and 3-D cases. In most cases, the problem is formulated as a cutting stock problem and it is well known that either such formulation or an explicit packing problem formulation give rise to an NP-hard task [9]. Despite the fact that the problem occurs in various domains, most of the existing approaches are applicable across many domains. For instance, algorithms developed for VLSI component arrangement are easily adaptable and applicable to the palletizing problem [9].

A large number of the existing approaches have been focussed on finding an ideal representation scheme for the problem mainly in two-dimensional cases [5] [7] and [8]. Most of these algorithms showed decent results of standard benchmark datasets such as the Beasley1 as shown in [9].

More recent work, from a previous VMAC competition in 2010, compared the performance of commercial software with the contributions from competitors. The issue of metrics used in design of the algorithms for solving the problem was also addressed [1].

III. Palletizing

Our assumption is, that the order will contain many articles of equal height. More specifically we assume that in most cases there are enough articles of equal height to fill one or more layers of same height. Using this simplification we reduce the 3D knapsack problem to a 2D knapsack problem in each individual layer of same article height. Arranging objects of same height in one layer will create a nearly seamless surface on the top. Hence another layer can be easily placed on it. If our initial assumption is right, i.e., there are many articles of same height, then this heuristic avoids the problem of stacking freely in three dimensions. This prevents articles from overlapping or tumbling down and allows for efficient stacking.

In most orders there will be a number of articles that are left over from being arranged into separate layers or are not enough to form their own layer to start with. Those articles will be put on top of the previously arranged layer stack. The idea is to build incomplete layers of same height, i.e., layers that do not cover the whole area of the pallet. However, these layers are sorted by the area they cover in descending order.

Thus, the layer covering the most area goes to the bottom of the top pile. Once all remaining articles fit into the available area next to each other, they are arranged in a final layer. This last layer can consist of articles of different height, as no other layer will be stacked on top.

As left over articles are most certainly of very different height, our approach evaluates other algorithms than the layer heuristic. One candidate for such an algorithm is the three dimensional corner block list (CBL) algorithm [9].

A. Implementation

We are using Python to implement our approach due to fast development. Computationally expensive subtasks are to be subsequently converted to a compiled language like C/C++ for faster execution.

The input XML file `order.xml` as well as the output XML file `packlist.xml` do not use any attributes but arrange information by XML tags only. This makes it possible to easily serialize the input as well as the output XML into a Python data structure only using the built-in dict and list types as containers. Normal XML parsers need their own datatypes as they need to differentiate between a node's attributes and child elements. Parsing from XML to a Python dictionary and the other way round is accomplished using the xml.etree module. For quick visual verification a minimal SVG library was written as well. It mostly serves for debugging purposes and allows to render arbitrary shapes onto a 2D canvas.

B. Arranging articles in layers

As explained above, our heuristic will arrange articles of the same height into individual layers and then stack those layers on top of each other. An illustration of this layered structure can be seen in Figure 1. As all articles in one layer are of the same height, stacking is not expected to create any problems. Therefore, the performance of our algorithm is also dependent on how well we arrange the articles in each layer. This subtask is a two dimensional knapsack problem and is therefore NP-hard, as its three dimensional counterpart. So another heuristic will be used here.

For the arrangement in one layer, the three dimensional cuboid articles will be simplified to the two dimensional rectangles representing the base rectangle of the cuboid. This simplification is possible since every article in one layer is of the same height. So after sorting the articles into bins of articles with equal height, each bin will be processed and its content sorted into as many full layers as possible, potentially leaving a rest. A layer is considered full if the sum of the base rectangles of the articles that are arranged in it, is larger than 70% of the area the pallet provides. This threshold is chosen arbitrarily and is adjusted depending on the scenario.

Considering one bin containing articles of equal height that have been reduced to their base rectangles, the algorithm will sort the rectangles by area. Afterwards, another algorithm will arrange them into one layer. This step is repeated until no new layer can be created. The possible leftover articles are kept in a list for later processing. The heuristic for arranging this list of rectangles into a bigger rectangle of pallet size is as follows: The empty pallet is represented by an empty node of the size of the pallet. When the first rectangle is inserted, it is placed in the upper left corner of the unused space and partitions the remaining space into an empty rectangle to the right of the inserted rectangle and an empty rectangle below the inserted rectangle. This insertion operation is stored in a tree like structure. The root node is the pallet area itself and upon insertion of the first article, the node becomes occupied and two empty child nodes are created of the size of the empty rectangles to the right and bottom of the inserted one. On every insertion of a new rectangle, the tree is traversed in breadth-first order to find an empty leaf node. If a fitting one is found, then this node is marked as full and two new leaves are added.

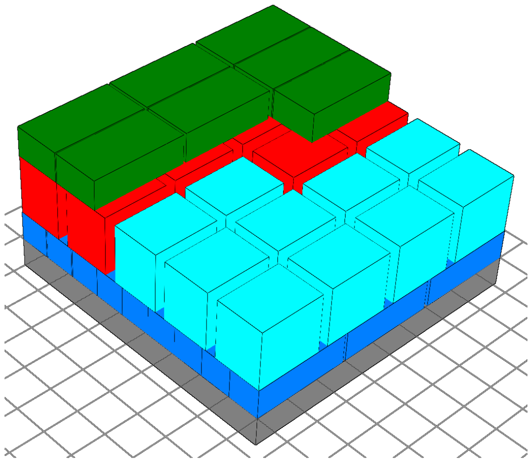


Fig. 1: partial article stack as rendered by the palletViewer evaluation software

Since new rectangles are always inserted in the most upper left position available, it is likely that there is some space between the most right rectangles and the right border and the lowest rectangles and the lower border. This is problematic when stacking several layers on top of each other as an overhang might be created. Hence, it is beneficial to spread out the placed rectangles so that they are most equally distributed over the available space. This is achieved by post-processing the tree that was created by the method above and spreading out each horizontal line to the available height and each rectangle in a horizontal line to the available width. An illustration of this processing step can be found in Figure 2b.

C. Optimizing the result

One feature of our approach is that many properties are quickly adaptable to different scenarios to increase performance:

- when sorting items for arranging them into layers, sort by base area, maximum of length and width or article weight
- process the articles rotated by 90 degrees
- arrange the items as if the pallet was rotated by 90, 180 or 270 degrees

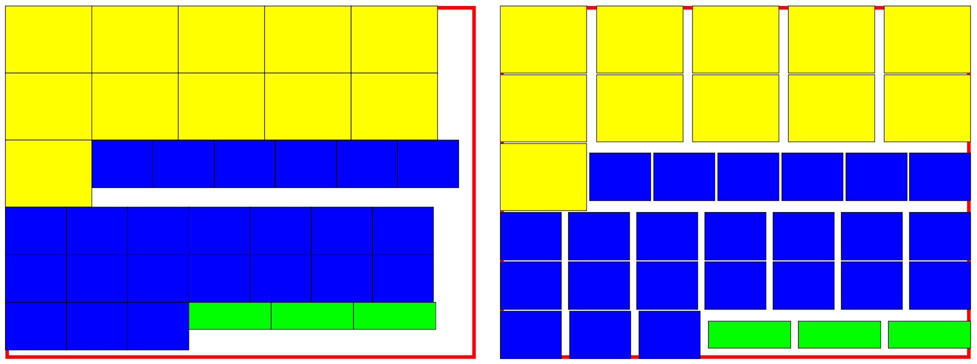

(a) rectangles arranged on pallet area (b) rectangles spread out over pallet area

Fig. 2: 2D placement heuristic for arbitrary layer configuration

- run algorithm on half or quarters of a pallet and join the individual results afterward
- try 3D-CBL for the leftover articles
- try 3D-CBL for the whole order

Articles come with very different densities and the evaluating algorithm checks for a good distribution of mass and a low placement of the center of mass. Hence it is reasonable to place heavy articles in the center of each layer. Our heuristic for building a layer works by starting at one of the corners and is hence not optimal for arranging articles around the middle. Two approaches to remedy the situation are listed above. Firstly, layers could be rotated so that while heavy packages are still at the corners, the mass is equally distributed around all four corners. Secondly, it is possible to run the algorithm on a pallet area that is divided into two or four parts and then started at the center of an edge or the middle of the pallet area respectively.

Since it is hard to decide on a one-size-fits-all strategy without sacrificing performance in special cases, all possible permutations of the options above are enumerated by the algorithm and compared against the evaluation software. The configuration that produces the highest score will be the final output of the algorithm. This brute-force method currently takes about 8 to 10 hours to complete, so in further research it will be used to find settings that never produce good output; these settings are then be removed from the tried options so that the overall runtime reduces to an acceptable value.

IV. Task Completion

In task completion we use an USARSim simulated KUKA KR60 robot arm with an attached vacuum gripper end effector, as given in Figure 3a. We focus on the simplified task of moving the end effector to a given pose, e.g., on the conveyor belt, gripping the object, moving to another given pose, e.g., on the pallet, releasing the object and repeating. Solving this task requires solving the inverse kinematics (IK) problem, for this 6 DOF robot. To this end, we implement two approaches: yielding a geometric solution, to be used for far-reaching poses, and a numeric solution, to be used for close-reaching poses.

Using the previously computed pack list from the palletizing task, we spawn boxes of sizes as specified in the simulator one at a time at a specific pickup location from which the robot grabs them and puts them onto the pallet. The processes of picking up the boxes and spawning new ones are synchronized using message passing.

A. Overview and Implementation

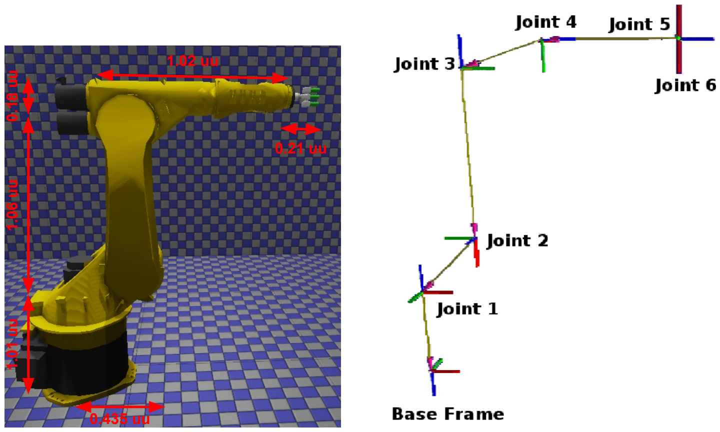
We are using ROS [10] to implement our inverse kinematics algorithms and communicate with the USARSim environment. Once the robot is spawned in USARSim, our ROS node, i.e., client, listens to messages about joint angles, gripper status, etc. The client node then receives a goal pose, either from a configuration file or ROS launch file, and based on a Euclidean distance threshold decides whether to apply the geometric or the numeric IK solution. Once the end effector has reached the desired pose, the object is gripped and the goal pose is changed. Once the end effector reaches the new pose, the object is released and the arm is reset to its resting position. This cycle repeats, until the connection to USARSim is closed.

A preliminary step in our implementation is the description of the robot model in ROS, given by the Unified Robot Description Format (URDF), which is an XML file representing the coordinate systems of the robot joints and the positions of the joint links. The visualization of our KUKA KR60 robot model is presented in Figure 3b.

Given the URDF file, we attempted to use an off-the-shelf IK solver [3], [4]. However, these either yielded no solution or required unreasonably long execution times. Consequently, we built our own IK solvers, based on textbooks [11], [2], as detailed in subsections IV-B, IV-C.

B. Geometric IK solution

This approach computes the closed form solution of the 6 DOF inverse kinematics, by solving a system of geometric equations and thus computing the joint angles required for a specified goal pose. The computation is done by first deriving joint values 1, 2 and 3 from Figure 3b, which are responsible for the translation component of the end effector pose. The remaining joint angles, which account for the orientation component in the end effector pose, are then computed based on the previous joint angles. Here, we briefly illustrate how the first three angles, i.e., $\theta_1, \theta_2, \theta_3$,



(a) KUKA KR60 in USARSim (b) URDF Robot Model visualized in rViz

Fig. 3: Visualization of the KUKA KR60 robot in simulation

are calculated. We denote the initial position and the goal position, as (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively. Please note that all the constants are expressed in UDK/Unreal units (uu), using the $1uu \leftrightarrow 804mm$ conversion.

1) First angle calculation

$$\text{Let } A = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
$$B = \sqrt{{x_2}^2 + (y_1 - y_2)^2}$$
$$\text{then } \theta_1 = \cos^{-1}\left(\frac{B^2 + {x_1}^2 - A^2}{2Bx_1}\right)$$

Note that the goal position has to be rotated by θ_1, after the first angle calculation.

2) Second angle calculation

Let $R = \sqrt{0.18^2 + 1.23^2}$. We distinguish between the following cases.

case 1: Goal position does not lie on a circle with radius R

$$C = \sqrt{(z_2 - 1.01)^2 + (x_2 - 0.435)^2}$$
$$D = \sqrt{(x_2 - 0.435)^2 + (z_2 - 2.07)^2}$$
$$\psi = \cos^{-1}\left(\frac{1.06^2 + c^2 - R^2}{2 * 1.06C}\right)$$
$$\gamma = \cos^{-1}\left(\frac{1.06^2 + c^2 - D^2}{2 * 1.06C}\right)$$
$$\theta_2 = \begin{cases} \psi - \gamma & \text{, if } x_2 < x_1 \\ \gamma - \psi & \text{, if } x_2 > x_1 \end{cases}$$

case 2: Goal position lies on a circle with radius R

$$\theta_2 = 0$$

3) Third angle calculation We first calculate the goal position with respect to the new coordinate axis of the third joint.

$$x_2 = x_2 - 0.435 \pm (1.06 * \cos(\theta_2))$$
$$z_2 = z2 - 1.01 - (1.06 * \sin(\theta_2))$$

The newly acquired goal position also has to be rotated to the opposite direction of θ_2.

$$S = \cos^{-1}\left(\frac{R^2 + 0.18^2 - 1.23^2}{2R * 0.18}\right)$$
$$P = \cos^{-1}\left(\frac{x_2}{R}\right)$$
$$\theta_3 = \begin{cases} S - (\frac{\pi}{2} - P) & \text{, if } z_2 > 0.18 \\ \frac{\pi}{2} - S - P & \text{, otherwise} \end{cases}$$

C. Numeric IK solution

The numeric approach relies on the approximation of the function mapping the joint space to Cartesian space with its Jacobian. The algorithm 1 then uses the Jacobian pseudo-inverse to compute the change in joint angles that bring the end effector closer to the goal position [2]. In doing so, we use the forward kinematics provided by the ROS transform (`tf`) messages between all the joints of our URDF robot model. The Jacobian is computed numerically and the inverse kinematics solution follows from the above description.

Algorithm 1 Iterative Algorithm for solving IK

```
Given: $x_{target}$
Wanted: $q_{target}$, s.t. $x_{target} = FK(q_{target})$
$x_{actual} = FK(q_{actual})$
while $||x_{target} - x_{actual}|| < \epsilon$ do
    Let $\Delta x = x_{target} - x_{actual}$
    Let $J = J(q_{actual})$
    $\Delta q = (J^T J)^{-1} J^T \cdot \Delta x$
    $q_{actual} \leftarrow q_{actual} + \Delta q$
    $x_{actual} = FK(q_{actual})$
end while
return $q_{actual}$
```

TABLE I: Scores for ICRA 2011 `order.xml` files for Georgia Institute of Technology (GT), Drexel University (Drexel) and Jacobs University Bremen (Jacobs). Some `order.xml` files for Drexel were not provided.

	GT	Drexel	Jacobs
Day 1, Round 1	84.36	67.59	90.43
Day 1, Round 2	28.96	32.94	81.96
Day 1, Round 3	9.69	26.45	80.99
Day 1, Round 4	40.63	44.46	87.81
Day 2, Round 1	78.18	39.61	89.87
Day 2, Round 2	21.56	-	80.27
Day 2, Round 3	4.80	-	75.69
Day 2, Round 4	37.88	-	81.61

V. Preliminary Results

A. Palletizing

Table I shows the score our approach currently achieves compared to the results from the Georgia Institute of Technology (GT) and Drexel University (Drexel). As input files, the `order.xml` files from the ICRA 2011 mixed palletizing competition tarball `icra2011files.tar.gz` [1] are used. The evaluation program is the palletViewer software package using the scoring file `scoreAsPlannedConfig1.xml` from the ICRA 2011 tarball. To produce the scores seen in the table, the `packlist.xml` from GT and Drexel that can be found in the ICRA 2011 tarball, as well as our own results were given to palletViewer. The ICRA 2011 website [2] also compares the GT and Drexel results against an undisclosed commercial software. Since we are not able to reproduce the listed GT and Drexel scores ourselves using the provided `order.xml` files and the palletViewer evaluation program, the commercial results are omitted from the comparison. A limitation of the current version of palletViewer is, that it can only process one pallet as an input even though many orders require to be distributed across multiple pallets. As soon as the palletViewer version for 2012 is released, our code will be adjusted to work well with it.

B. Task Completion

For the task completion component we assessed the performance of our approach qualitatively by visual inspection and quantitatively by comparing the accuracy of the reached poses. We set up a scenario, as presented in Figure 4, where the robot picks boxes that successively appear at a fixed location and places them on the pallet at locations indicated by the palletizing algorithm.

Our experiments show that the numeric IK is prone to jerk when reaching far poses, but is quite smooth for close poses. In contrast, the geometric approach is smooth in both cases, but is less accurate than the numeric approach for close poses. The quantitative metric used for accuracy is the norm of the pose error vector:

$$||x_{target} - x_{actual}||_2$$

[1]`http://www.vma-competition.com/files/icra2011files.tar.gz`

[2]`http://www.vma-competition.com/?q=node/13`

TABLE II: Accuracy of IK approaches in Unreal units

	Geometric	Numeric
Close poses	$\approx 0.06uu$	$\approx 0.01uu$
Far poses	$\approx 0.02uu$	–

The results in Table II show that the numeric approach is about an order of magnitude more accurate in the case of close poses. For the case of far poses, we do not report the result, because the jitteriness of the robot arm yields the solution infeasible.

VI. Conclusion and Future Work

In this paper we presented the approach of the Jacobs University Bremen team for the *IEEE ICRA 2012 Virtual Manufacturing Automation Competition*. Our approach dynamically chooses the best heuristic to apply for solving the 3D bin packing problem. The results shown in section V are superior to the other university teams' results in [6] and comparable to the commercial software benchmark. The flexibility of our approach allows us to quickly adjust to the given metrics and scenarios.

Additionally, we investigate task completion in a simulated environment, i.e., using the output of the palletizing task to place items on a pallet with a robotic arm. We solve task completion within a ROS framework, where we combine two textbook approaches for solving the inverse kinematics of the simulated KUKA KR60 robot. While our approach may benefit from further improvements, e.g., use via-points for planning the end-effector trajectory so as to avoid object collisions, it lays the foundation for the intermediary step that bridges the palletizing and the mobility components. Given the experimental nature of the task completion component, we only report preliminary results and qualitatively assess the performance of our work.

Needless to say, a lot of work remains to be done. In future work we will focus on the mobility challenge, building on top of our existing components. Furthermore, we aim at selecting heuristics for palletizing using a learning approach.

References

[1] Stephen Balakirsky, Fred Proctor, Tom Kramer, Pushkar Kolhe, and Henrik I. Christensen. Using simulation to assess the effectiveness of pallet stacking methods. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, SIMPAR'10, pages 336–349, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1989.

[3] David Lu. Arm Kinematics ROS package. `http://www.ros.org/wiki/arm_kinematics`.

[4] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[5] Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, Chung Kuan Cheng, and Jun Gu. Corner block list: an effective and efficient topological representation of non-slicing floorplan. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, ICCAD '00, pages 8–12, Piscataway, NJ, USA, 2000. IEEE Press.

[6] ICRA 2011. VMAC Results. `http://www.vma-competition.com/?q=node/13`.

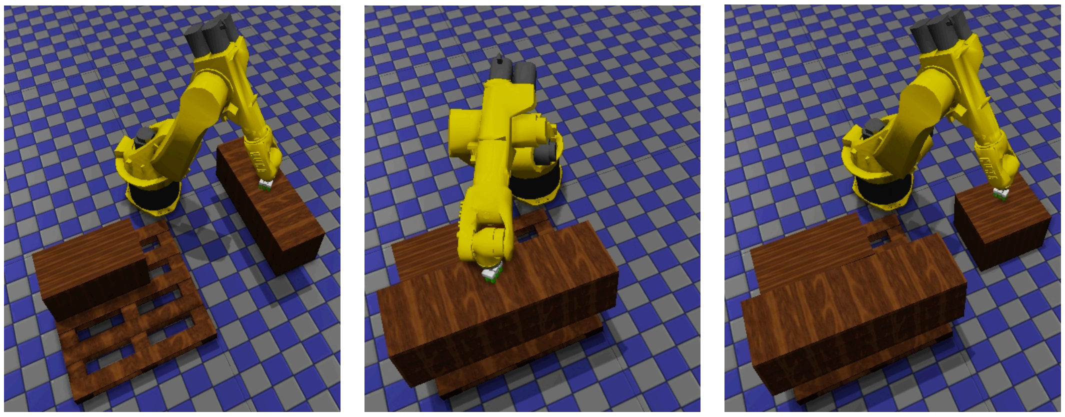


Fig. 4: Task completion scenario. Sequence of robot movements from left to right.

[7] Jai-Ming Lin and Yao-Wen Chang. Tcg: a transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 764–769, New York, NY, USA, 2001. ACM.

[8] Jai-Ming Lin and Yao-Wen Chang. Tcg-s: orthogonal coupling of p*-admissible representations for general floorplans. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 842–847, New York, NY, USA, 2002. ACM.

[9] Yuchun Ma, Xianlong Hong, Sheqin Dong, and C.K. Cheng. 3d cbl: an efficient algorithm for general 3d packing problems. *IEEE Transactions on Circuits and Systems*, 2:1079–1082, 2005.

[10] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[11] Wolfgang Weber. *Industrieroboter – Methoden der Steuerung und Regelung*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2nd edition edition, 2009.