Master's Thesis

AN EFFICIENT DEEP REINFORCEMENT LEARNING MODEL FOR ONLINE 3D BIN PACKING COMBINING OBJECT REARRANGEMENT AND STABLE PLACEMENT

2310071 ZHOU PEIWEN

Supervisor CHONG Nak-Young

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March 2025

Abstract

This paper presents an efficient deep reinforcement learning (DRL) framework for online 3D bin packing (3D-BPP). The 3D-BPP is an NP-hard problem significant in logistics, warehousing, and transportation, involving the optimal arrangement of objects inside a bin. Traditional heuristic algorithms often fail to address dynamic and physical constraints in real-time scenarios.

We introduce a novel DRL framework that integrates a reliable physics heuristic algorithm and object rearrangement and stable placement. Our experiment show that the proposed framework achieves higher space utilization rates effectively minimizing the amount of wasted space with fewer training epochs.

The physics heuristic algorithm aims to ensure the stability of packed objects. Three variants of this algorithm are introduced: convexHull-1, convexHull-k, and convexHull–α. The convexHull-1 algorithm checks stability by evaluating whether the center of a sliding window, matching the dimensions of the incoming object, lies within the convex hull formed by the highest points within that window. However, it was found to be insufficient in complex multi-stack scenarios. The convexHull-k algorithm extends this concept by considering the intersection of convex hulls across multiple layers, but it still underestimates stability in certain cases. To address these limitations, the convexHull-α algorithm is proposed, which ensures that the supporting force originates vertically from the ground, using an empty map to track wasted voxels along the z-axis. This algorithm significantly improves the accuracy of stability checks, as demonstrated through simulations.

To further enhance packing efficiency, the framework incorporates an object rearrangement process. This allows the robot manipulator to change the orientation of incoming objects, an action that directly improves space utilization without requiring additional time costs. The DRL model is formulated as a Markov Decision Process (MDP), with two separate agents, each responsible for predicting the optimal orientation and position for the packing object. The reward function is designed to balance the increase in space utilization and the minimization of wasted space.

The proposed DRL framework is trained and evaluated through both simulation experiments. In the simulations, two vision sensors capture visual information of the bin and incoming objects, enabling the model to predict orientations, stable positions, and placement scores. The results show that the proposed framework achieves higher space utilization rates compared to baseline methods, while requiring fewer training epochs.

The real-robot experiments involve connecting a uFactory850 robotic arm to the PC, performing hand-eye calibration using ArUco QR codes, and recognizing and suctioning target objects. The target objects' dimensions and poses are accurately estimated using a semantic segmentation model and OpenCV's pose estimation functions. The robotic arm then adjusts the object's orientation and places it in the bin, as predicted by the DRL model.

The experimental results shows that: 1.The physics heuristic algorithm, especially convexHull-α, significantly outperforms previous methods in terms of stability check accuracy.2. the proposed DRL framework achieves higher space utilization rates effectively minimizing the amount of wasted space with fewer training epochs.

Keywords: 3D Bin Packing, Object Rearrangement, Placement Stability, Deep Reinforcement Learning

Acknowledgment

My studying and working experience in Professor Chong's lab at JAIST is deeply unforgettable. I am grateful to Professor Chong, Dr. Gao, and Senior Li for their help in my research, which also guided me into a new research field—robotics. I also want to thank my brother Ye, Senior Song, Senior Zhang, and Canh Thanh for their support in daily life, helping me survive the Cold Winter in Ishikawa.

I am not an intelligent person, having a solitary personality and a terrible temper. I am thankful to my parents for their education and financial support since childhood. Thank for the time and effort they speed on me,that allowed me to grow up healthily, and enabled me to achieve the current degree. Additionally, I want to express my gratitude to Princess Yang, my wife, for her willingness to always be by my side, encouraging me, supporting me, and love me. I also will do my best to protect and love my Princess.

My 20-year journey of academic trip has now come to an end, and I am about to start my career, integrating into society with a new identity. I am profoundly grateful to everyone who has helped me. I wish everyone achieve success in their studies and have a good future.

List of Symbols

${}^{cam}T_{ArUco}$	The transformation matrix from real ArUco frame to camera frame
${}^{cam}R_{ArUco}$	The rotation matrix from real ArUco frame to camera frame
${}^{ee}T_{ArUco}$	The transformation matrix from real ArUco frame to robot frame
${}^{ee}R_{ArUco}$	The rotation matrix from real ArUco frame to robot frame
${}^{ee}T_{cam}$	The transformation matrix from camera frame to robot frame
CoM	Object's center of mass
(O_t, P_t)	The action tuple for time step t
(P_f)	The upward support force in convex hull
γ	The discount for the cumulative reward in Markov Decision Processes
r_t	The reward for time step t
$R_z(\theta)$	Rotation around x axis about θ
$R_y(\theta)$	Rotation around y axis about θ
$R_z(\theta)$	Rotation around z axis about θ
s_t	The environment state for time step t
t	Markov Decision Processes time step t

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

Robotic bin packing has many applications in the fields of logistics, warehousing, and transportation. The 3D Bin Packing Problem (3D-BPP) [1], a well-known NP-hard problems [2], is referred to as an optimization problem of packing multiple objects into a bin(s), while satisfying the bin capacity constraint [3]. The 3D-BPP can be tackled offline or online depending on whether all objects can be accessible or not. In terms of offline bin packing task, this setting assumes the prior knowledge of all objects, usually, finding the optimal packing sequence and optimal placement are involved in this setting. Typically, meta-heuristic algorithms [4] [5] [6] have been employed to determine the optimal order sequence in previous studies [7], thereafter, heuristic algorithms, such as DBLF proposed by Korha and Mustaf [8] or HM proposed by Wang and Kris [9], are leveraged to determine where to place the object into the bin.

Compared with offline bin packing, online bin packing is more challenging [10]. Basically, the packing order is random, and the agent can only observe the upcoming objects (either single or multiple objects) as illustrated in Fig. 1.1. In this context, relying exclusively on heuristics results in a considerable decline in bin utilization [7]. Under these constraints, Yang *et al.* [11] employed unpacking-heuristics to improve the utilization. Nonetheless, this method raises the time cost, thereby diminishing the overall efficiency of the packing process.

Recent progress in DRL has shown promising results in various domains by enabling models to learn optimal policies through trial and error [12]. Compared with heuristic algorithms, DRL excels in addressing optimization problems effectiveness in complex environments. However, real-world physical law damages the training efficiency as learning the physics in complex environment takes many trial-and-error iterations, and the stable placement cannot be guanranteed. Zhao *et al.* [13] and Yang *et al.* [11] leveraged neural network to predict the physical feasibility map, enabling the agent to learn

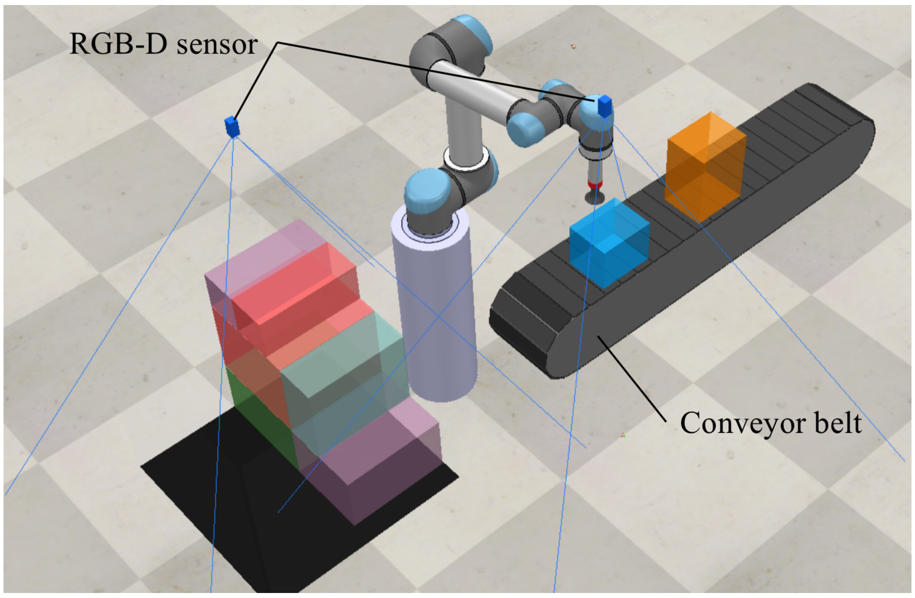


Figure 1.1: Online 3D-BPP, where the agent can only observe an upcoming object and pack it on-the-fly.

feasible packing strategies. Although these methods have achieved promising results in 3D-BPP, object stability is not guaranteed. To address these challenges, we propose an efficient and effective DRL framework [14] [15] [16] using a highly reliable physics heuristic algorithm for online 3D-BPP. The main contributions of this paper are as follows.

- We proposed a highly reliable physics heuristic algorithm that guarantees the stability of object placement in complex multi-stack environments, while retaining as many placement positions as possible.
- We incorporated an object rearrangement process into the proposed framework which allows the robot manipulator to change the orientation of the upcoming object. It is also an efficient action that directly enhances space utilization without requiring additional time costs.

1.2 Relative work

1.2.1 Heuristics in Bin Packing Problem

The bin packing problem is a key challenge in combinatorial optimization [17], aiming to arrange multiple objects efficiently within the larger container. However, 3D-BPP become unsolvable within a reasonable time frame using exact algorithms [18] when involving a large number of objects. Over the years, various heuristic and meta-heuristic methods have been developed to address this problem [19] [20] [8] [21]. Heuristic algorithms critically depend on the sequence of object placement, and current research often employs meta-heuristic techniques such as simulated annealing [22] and genetic algorithms [8].

Consequently, if complete information on all objects to be packed is unavailable, the effectiveness of heuristic algorithms drops significantly. Moreover, in real-world logistics warehouse, gathering detailed information about all objects can be challenging and time-consuming, reducing operational efficiency. Therefore, We propose using the object rearrangement method to change the orientation of objects in order to improve bin utilization, under the constraints of unchangeable order sequence.

1.2.2 DRL in 3D-BPP

DRL combines the decision-making capabilities of reinforcement learning with the powerful representation learning of deep neural networks. Furthermore, it can be adaptable to changing conditions and provide feasible solutions with highly efficient [12] [23], where traditional methods may struggle to find efficient solutions [**?**]. DRL has recently demonstrated strong performance across various robotics tasks [24] [25], showcasing its ability to handle complex spatial and dynamic challenges effectively.

Thus applying DRL to the 3D-BPP could indeed be a highly efficient approach. For example, Zhao *et al.* [13] introduced a prediction-and-projection scheme where the agent first generates a physical stability mask for placement actions as an auxiliary task, then using this mask to adjust the action probabilities output by the actor during training. However, DRL models can suffer from instability and sensitivity to hyperparameters [14], making them difficult to tune and sometimes resulting in unpredictable performance. Moreover, most work focuses only on sample constraints, without considering real-world physical properties of objects, including the object CoM [26] and its deviation in a complete stack. These factors can result in solutions that are impractical for real-world applications where physical stability and balance are essential.

Thus we propose the DRL framework integrated with a physics heuristics. This not only guarantees the stability of object placement but also enhances the training efficiency of the model, allowing for faster convergence [27] [28].

1.2.3 Stability check in 3D-BPP

Stable stacking is a critical factor when designing an online 3D bin packing pipeline. Learning the rules of real-world physics is a very difficult process for DRL [29]. This not only lengthens the training time for the model but also causes fluctuations in model convergence.

Therefore, for 3D-BPP, it is necessary to design a reliable and efficient physics heuristics for feasible action detection to quickly rule out incorrect actions in the current state. Zhao *et al.* [13] and Yang *et al.* [11] use the similar scheme that combines the ratio of overlapping parts between the placed item and its contact items with a neural network model for prediction. But this is not a reliable method, since the model is a black box, there are always parts that are inexplicable and unpredictable. On the other hand, Wang and Kris [9] proposed a mathematical model that using a linear programming method solves for the torque balance and force balance of the object for all contact forces. Although this is a very reliable method,it is too complex for regular objects and usually takes a long time to evaluate all the candidates actions.

Chapter 2

Method

We describe our method in two parts. First, we present our stability check method, which is a highlight of our work. Second, we introduce a DRL framework that integrates physical heuristics and object rearrangement.

2.1 Stability Checking via Physics Heuristics

In our research, we assume that the object $i \in I$ for bin packing are rigid body and have a uniform mass distribution, so that the center of mass (CoM) is the geometric center of the object. But our method is not limited by the mass distribution, here just for simplify questions. For uneven objects, we can use Gao *et al.* [30] [31] to estimate as close as possible the CoM.

For the current state bin (W, D, H), we generate a bottom-to-top depth heightmap with a resolution of 0.005, where each voxel represents a 5 mm^3 vertical column of 3D space in the agent's workspace. The object to be placed i is defined by its dimensions (w_i, d_i, h_i). We employ a $w_i \times d_i$ sliding window to traverse the height map to check the stability of each placement.

2.1.1 convexHull-1

Based on physics' principles, we introduce the convexHull-1 method. The main idea of convexHull-1 is shown in Fig. 2.1. The left image depicts a sliding window that matches the size of the incoming object, along with portions of the scene objects contained within the sliding window. The right figure shows the zoom-in version of the content inside the sliding window [32]. To determine the stability of the object, we calculate the largest convex hull of the highest points within the window. Next, we verify whether the center of the window lies within the convex hull [33]. The object is deemed stable when positioned at the center of the sliding window if the convex hull includes the window's center.

The upward support force, denoted as $p_f^i = \{p_1, p_2, ...\}$ is defined as the set of highest points in the window, obtained by object i under currently

Algorithm 2.1 convexHull-α

```
Input: Bin depth heightmap $B$, empty map $E$,object$_i$ $(w_i, d_i, h_i)$, bin height $H$
Output: Stable action map $A$
for $b_{x,y} \in B$ do
    $window \leftarrow B$[x: x + $w_i$][y: y + $d_i$]
    if $window$ out of boundary or $window.max + h_i > H$ then
        $A_{x,y} \leftarrow 0$
        $continue$
    end if
    if $window.max == 0$ then
        $A_{x,y} \leftarrow 1$
        $continue$
    end if
    Find $p_f$ in $window$
    Use $E$ update $p_f$ to $p'_f$
    $center \leftarrow Point(\frac{w_i}{2}, \frac{d_i}{2})$
    $hull \leftarrow convexHull(p'_f)$
    $inside \leftarrow$ Boolean $center.within(hull)$
    $A_{x,y} \leftarrow inside$
end for
return $A$
```

Algorithm 2.2 Empty map update

```
Input: Bin depth heightmap $B$, empty map $E$, action point$(X, Y)$, object $i = (w_i, d_i, h_i)$
Output: Updated empty map $E$
$window^B \leftarrow B[X : X + w_i, Y : Y + d_i]$
$window^E \leftarrow E[X : X + w_i, Y : Y + d_i]$
$h \leftarrow window^B.max$
for $e_{x,y}$ in $window^E$ do
    $window^E_{x,y} \leftarrow e_{x,y} + h - window^B_{x,y}$
end for
$E[X : X + w_i, Y : Y + d_i] \leftarrow window^E$
return $E$
```

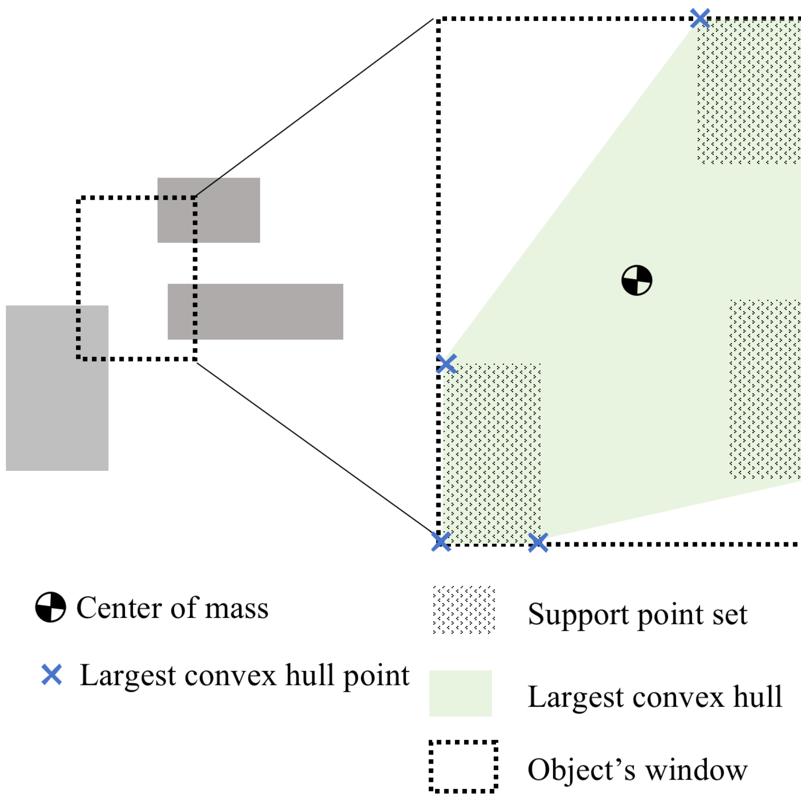


Figure 2.1: The example for convexHull-1.

placement. We utilize OpenCV [34] to calculate the largest convex hull formed by p_f. Then, we evaluate the placement stability by verifying if the center of the sliding window is within the convex hull or not. During our experiments, we observed that relying only on a single layer of the convex hull cannot ensure the stability of object placement. Fig. 2.3 shows an example using convexHull-1 for stability check and fail.

2.1.2 convexHull-k

Consequently, we extend this method to convexHull-k, which considers the intersection of convex hull across multiple layers. However, in more complex scenarios including multiple stacks within the bin, convexHull-k would underestimate the placement of stability, as shown in Fig. 2.2. The blue cube represents the upcoming object, while the gray cubes depict the scenario in the bin. The yellow area shows the layer's convex hull calculated by convexHull-1 algorithm. (a),(b) and (c) illustrate the convex hulls of

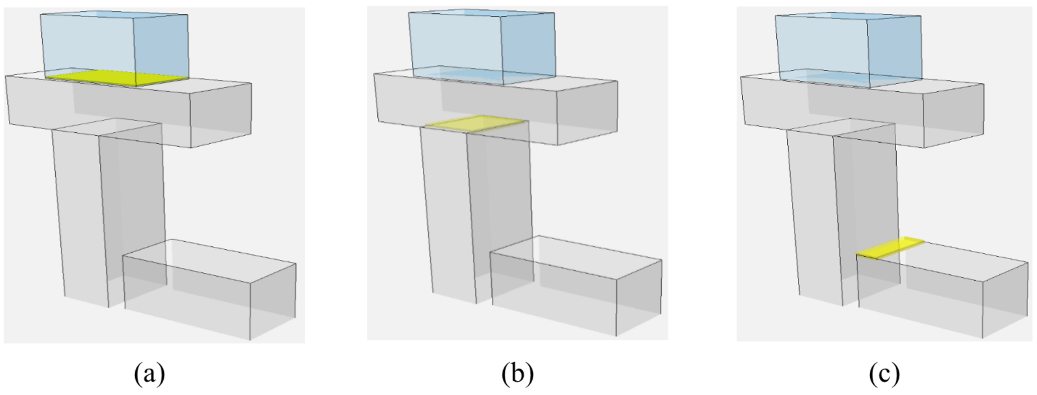


Figure 2.2: The example for convexHull-k

different layers for the upcoming object. Although the intersection of these convex hulls results in NULL, indicating an unstable placement, in practice, the placement is stable.

2.1.3 convexHull-α

To address the aforementioned issue, we introduce convexHull-α, for managing multiple stacks of objects in complex environments. Throughout the object packing procedure, we maintain an empty map with the same size as the action map. The main concept of covexHull-α is that the supporting force must be vertical and originate from the ground. Basically, for each position inside the sliding window, we check the number of wasted voxels along the z axis. We consider that only no wasted voxels can be the reliable support force, which corresponds to the empty map value is zero, denoted as p'_f. After each placement, we update the empty map outlines in Algorithm 2.2. Similarly, we use the new set of points p'_f to calculate the convex hull and determine whether the window's CoM is within it or not.

The Fig. 2.3 illustrates an example of stability check using convexHull-α. Multi-layer packing scenarios showcasing the difference between convexHull-1 and convexHull-α algorithms for checking the stability of the placement. (1) Both convexHull-1 and convexHull-α consider the arrangement to be stable. (2) Conversely, convexHull-1 might incorrectly assess the stability if the incoming object is significantly heavier than the object in the middle layer, as detailed in (3).Algorithm 2.1 outlines our algorithm in detail.

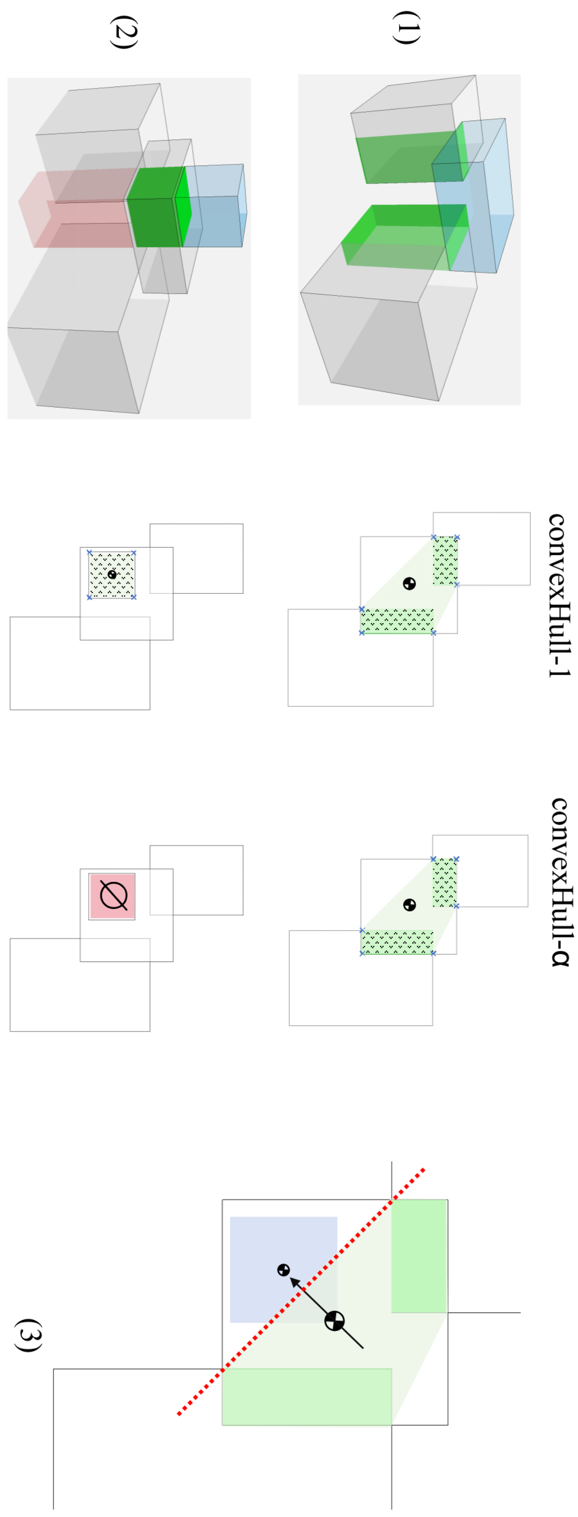


Figure 2.3: Comparison of convexHull-1 and convexHull-α

2.2 DRL for Bin Packing

2.2.1 Problem Formulation

Formally, online 3D bin packing involves packing a number of object $i \in I$, each with arbitrary dimensions (w_i, d_i, h_i) and cuboid shapes, into a bin of arbitrary dimensions (W, D, H). The process is constrained by the visibility of only the immediately upcoming object could be packed into the bin. Once the bin is filled or can not pack upcoming object the process will stop.

To solve this task, we formulate it as a Markov Decision Processes (MDPs) [35], which can be denoted as a tuple $M = \langle S, A, P, R, \gamma \rangle$. Specifically, we employ two agents with polices π_o and π_p to independently predict placement orientation and position.

The whole process is descried as follow: At the time step t, the agent observes the environment and takes a state representation, denoted as s_t. Then the agent π_o predicts the action o and pass to agent π_p to predict action p. Execute the action tuple(o_t, p_t), causing the environment to transition to s_{t+1}, then immediately obtains a reward r_t. The process aims to achieve the maximal cumulative rewards with discount γ, as shown in Eq. (2.1) and (2.2), by jointly optimizing two policies.

$$J_{\pi_o^*} = \max E_{\pi_o, s_t, o_t = \pi_o(s_t)} [\sum_t \gamma^t r_t] \tag{2.1}$$

$$J_{\pi_p^*} = \max E_{\pi_p, s_t, p_t = \pi_p(s_t | o_t)} [\sum_t \gamma^t r_t] \tag{2.2}$$

2.2.2 State Definition

We define state s_t as the configuration of the bin along with the object that is about to be packed. Use the depth image of the bin to generate a bottom-to-top $W \times D$ depth heightmap dh_t [13]. Following the work conducted by Yang *et al.* [11], given the object i with dimensions (w_i, d_i, h_i), we create a three channel map with the dimension $W \times D \times 3$. Each channel corresponds to one of the object's dimensions and is fully populated with the respective dimension values. Then combine them as $(dh_i, w_i^*, d_i^*, h_i^*)$ to represent the State.

2.2.3 Action Definition

In this work, we propose to arrange object orientation in order to achieve better placement. Therefore, the action is defined as the conjunction of

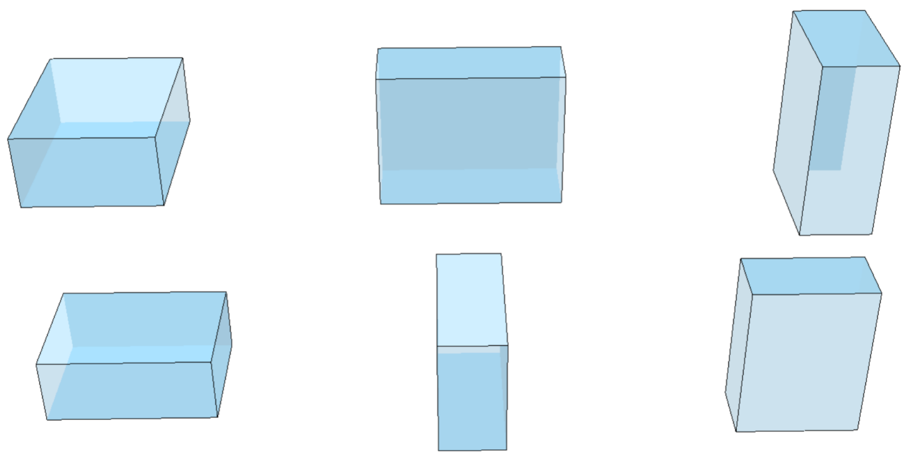


Figure 2.4: Six possible orientations of the packing object.

object rearrangement and placement, which is represented by (o, p), where o represents the target object orientation and p represents a specific position on top layer of the bin. To simplify the packing procedure, both o and p are discretized.

As illustrated in Fig. 2.4, there are six different orientations. The number of positions for possible placement is the same as the number of pixels inside the heightmap. Given (o, p), the agent firstly uses object rearrangement operation to achieve the object orientation o, and then place the object to the position p.

2.2.4 Reward Function

Following the idea mentioned in [13], at the time step t, the immediate reward is the weighted subtraction of increased utilization space and the wasted space given by Eq. (2.3) to (2.5). Please note that the wasted space can be calculated efficiently by comparing the summation of the empty map before and after the placement. In addition, both α and β are set to be one in our experiment.

$$R^i = \alpha \cdot r_v^i - \beta \cdot r_{waste}^i \tag{2.3}$$

$$r_v^i = \frac{w_i \cdot d_i \cdot h_i}{W \cdot D \cdot H} \tag{2.4}$$

$$r_{waste}^i = \frac{v_{waste}^i}{W \cdot D \cdot H} \tag{2.5}$$

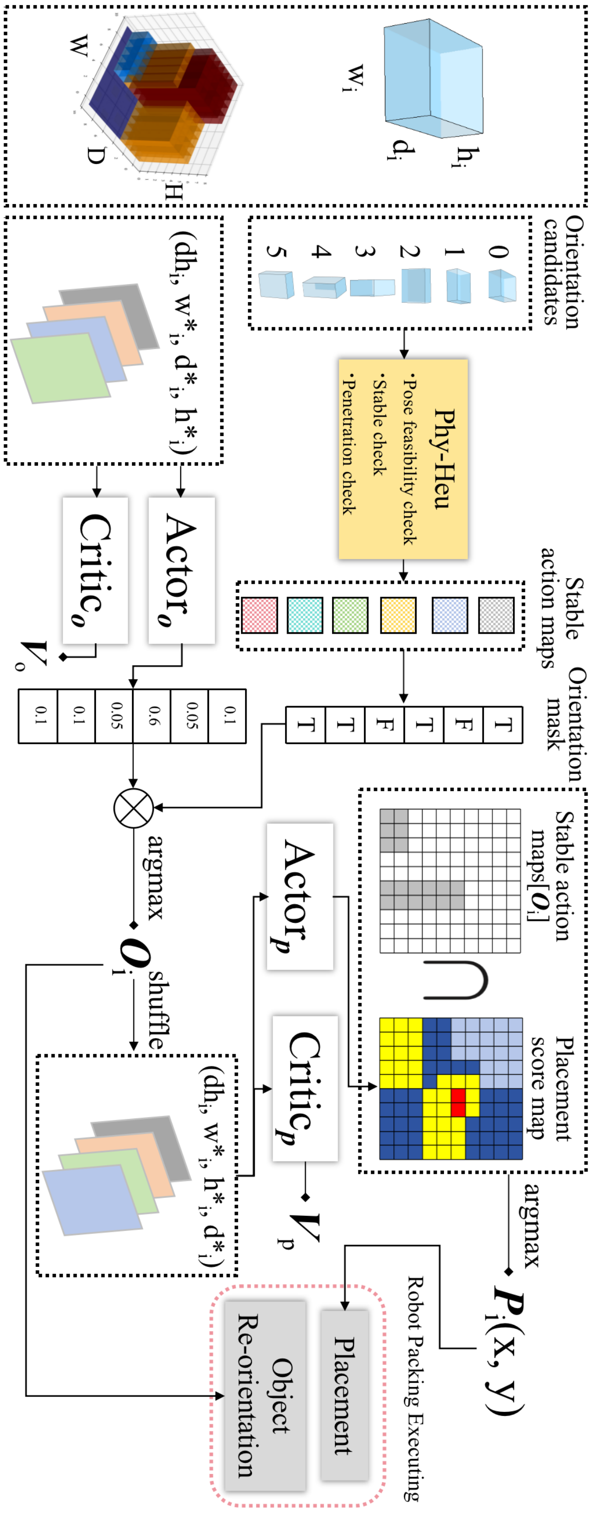


Figure 2.5: The pipeline of the DRL framework.

2.2.5 Physics Heuristics DRL Framework

Distinct from other works [11], we proposed a two-agents DRL framework integrated with physics heuristics as shown in Fig. 2.5. Based on Proximal Policy Optimization (PPO) [36], we develop two actor networks [37]: $Actor_o$ dedicated to predicting the object's orientation and $Actor_p$ to determining the packing position. Both actor networks takes input as the 4-channels maps, the output of $Actor_o$ is a six-dimensional vector where each element dedicates one specific object orientation, the output of $Actor_p$ is the action map for placement with the same size as the heightmap.

The training pipeline is as follows: Given the object i and configuration of bin dh_i, firstly, the Phy-Heu module generates stable action maps for all potential object orientations. Using these stable action maps, we construct an orientation mask to exclude orientations that do not allow for any feasible stable placement. Meanwhile, $Actor_o$ will predict the probability distribution of the object orientations. Using the orientation mask and the predicted distribution of orientations, the orientation is sampled. Next, based on the sampled orientation, the agent $Actor_p$ takes the dh_i and shuffled (w_i^*, d_i^*, h_i^*) to predict the placement score map. Lastly, we sample the action P_i from the intersected map of the corresponding stable action map and the predicted action score map to ensure the placement stability.

Chapter 3

Robot Experiment

3.1 Simulation experiment

3.1.1 Simulation environment

Our simulation experiments were performed with CoppeliaSim [38] and the physical engine [39] using Newton.

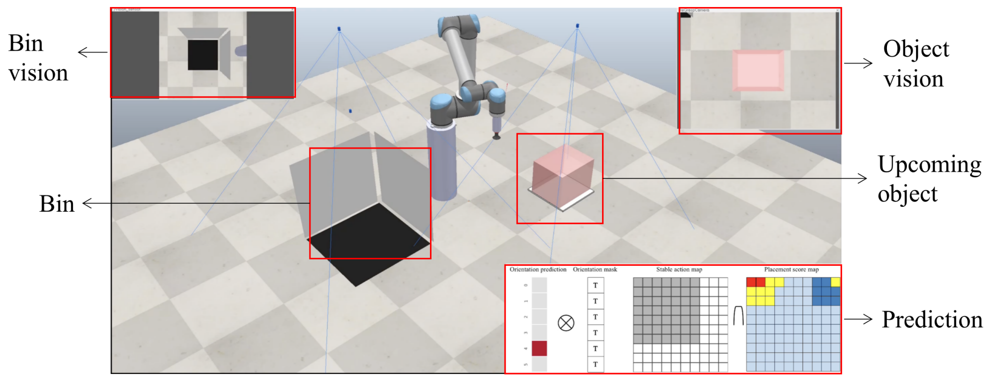


Figure 3.1: Simulation experiment.

3.1.2 Simulation process

We set up our simulation experiment as shown in Fig. 3.1. We utilize two vision sensors to capture visual information. One is placed directly above the Bin, and the other is positioned directly above the upcoming object. We set the size of the bin to 0.6 m $\times$ 0.6 m $\times$ 0.6 m. Additionally, we also visualize the predictions made by the object placement process model, including orientation prediction, orientation mask, stable action map and placement score map. The visualization of the prediction process facilitates our examination of the model's stability and reliability.

To streamline the entire process, we have deleted the conveyor belt and instead utilized a platform with dimensions of 0.4 m $\times$ 0.4 m. Each upcoming object will appear above the platform, where a vision sensor will observe the object and capture its (W,D,H) to prepare for subsequent predictions.

3.2 Real Robot Experiment

After completing the training in simulations, we implemented the entire packing process on the real robot arm. The experimental procedure encompassed the following steps: hand-eye calibration, target object pose recognition, target object suction, target object pose adjustment, and target object placement(planned but not executed due to time and space constraints).

Our experiment setup, as shown in the Fig. 3.2, a robot arm, a robotic arm controller box, a vision sensor, and a PC. Detailed specifications of these components are provided in the Table 3.2. Control of the robotic arm and driving of the model were implemented on a PC using Python, with the specific environment configurations outlined in the table.

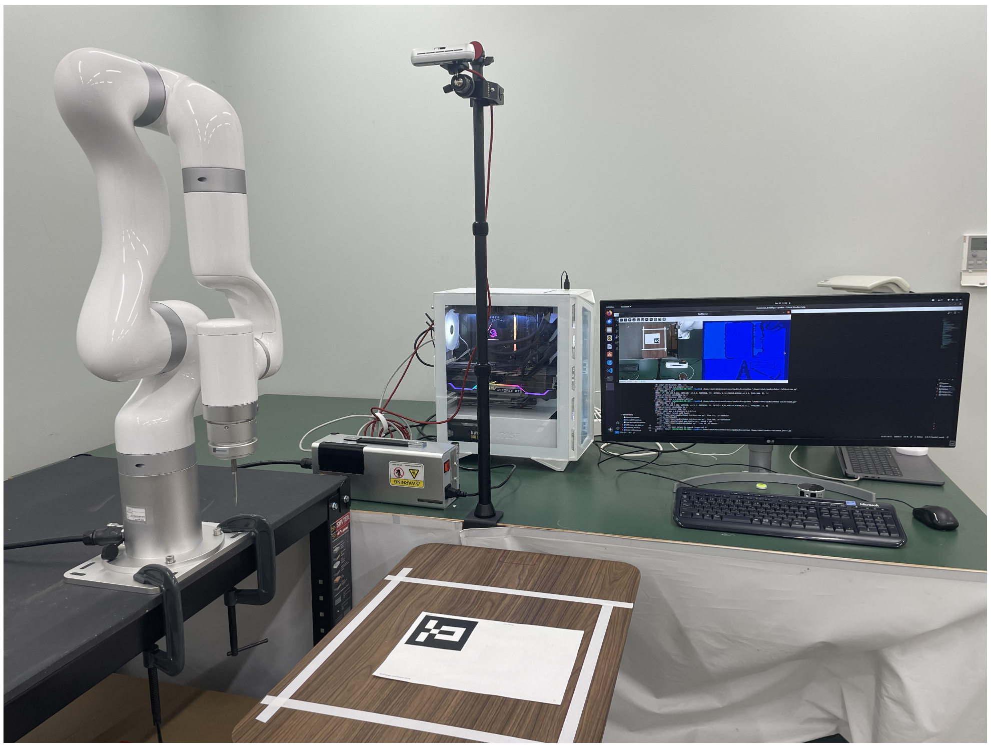


Figure 3.2: Real robot experiment setup.

Robot arm	uFactory 850
Vision sensor	D455i
PC GPU	3090ti

Table 3.1: Real robot experiment setup

System	ubuntu20.04
Environment management	mini Conda
Python [40]	3.10
openCV [34]	3.4.10
PyTorch [41]	2.5
cuda [42]	12.1
realsense [43]	Pyrealsense2

Table 3.2: PC environment setup

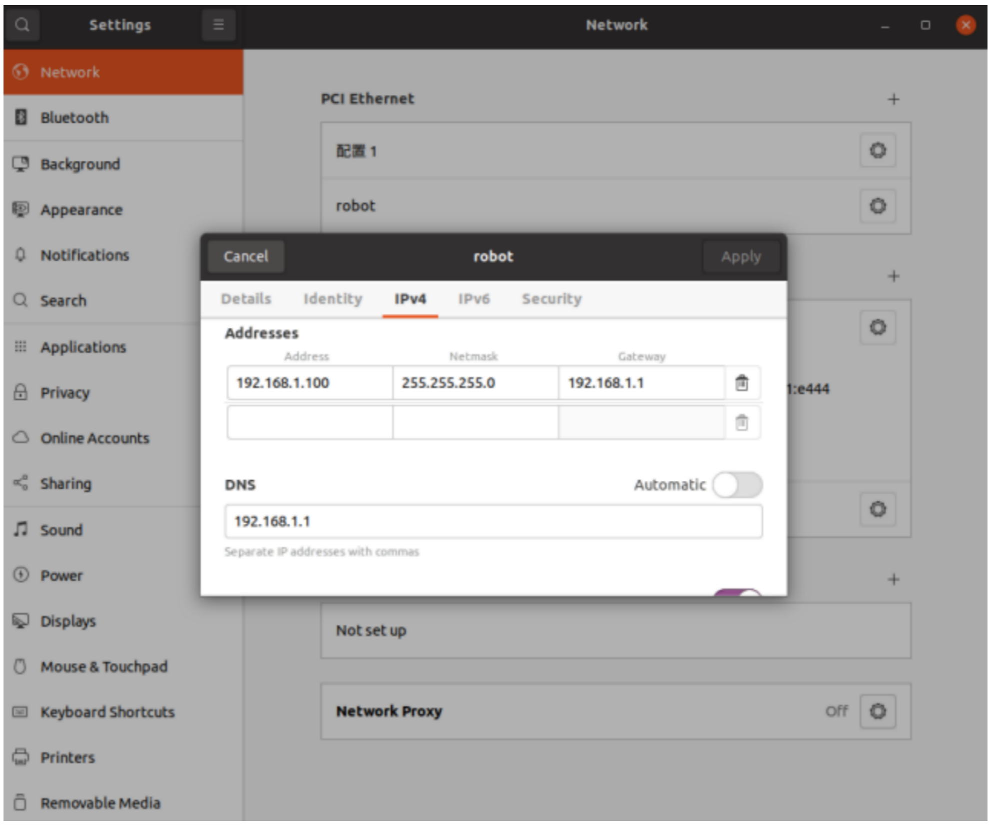
3.2.1 Real robot arm connection

The robotic arm we utilize is the uFactory 850, with specific parameters detailed in the Table 3.3. This robotic arm is connected and controlled via Ethernet mode. Initially, we connect an Ethernet cable to the robotic arm's control box and then manually configure the IPv4 network interface, as shown in the Fig. 3.3. We set the address to 192.168.100, the Net mask to 255.255.255.0, the gateway to 192.168.1.1, and finally, the DNS to 192.168.1.1. After completing the configuration, we ping 192.168.1.100 from the terminal; if data is returned, it indicates successful configuration.

The SDK provided on the official website of this robotic arm is integrated into the "robot" class, with the motion control interface maintaining consistent input and output. This facilitates the subsequent use of different interfaces and classes.

	uFactory 850
DoF	6 Axis
Reach	850 mm
Repeatability	0.02 mm
Max Speed	1 m/s

Table 3.3: uFactory 850 parameters.



Addresses

Address	Netmask	Gateway
192.168.1.100	255.255.255.0	192.168.1.1

DNS Automatic: off

192.168.1.1

Separate IP addresses with commas

Figure 3.3: Example of Real robot connection.

3.2.2 Hand-eye calibration process

The hand-eye calibration process primarily involves the transformation between the base coordinates of the robot and the camera coordinates. We employ the ArUco QR code recognition method [44] for hand-eye calibration [45]. Through the recognition algorithm, ArUco QR codes can instantaneously provide the position and orientation of the QR code's center in the camera coordinate system.

This calibration method has high efficiency, accuracy, and low cost. The calibration tools used are shown in the Fig. 3.4, including the a vision sensor, a spirit Level, end-effect tool, ArUco QR codes, and calibration tool(self-made).

The installation requirements for the vision sensor are as follows:1.The sensor needs to be installed parallel to the X-axis of the robot arm's base coordinate system. 2.Use a spirit level to adjust the vision sensor to a horizontal position, as shown in the Fig. 3.5.

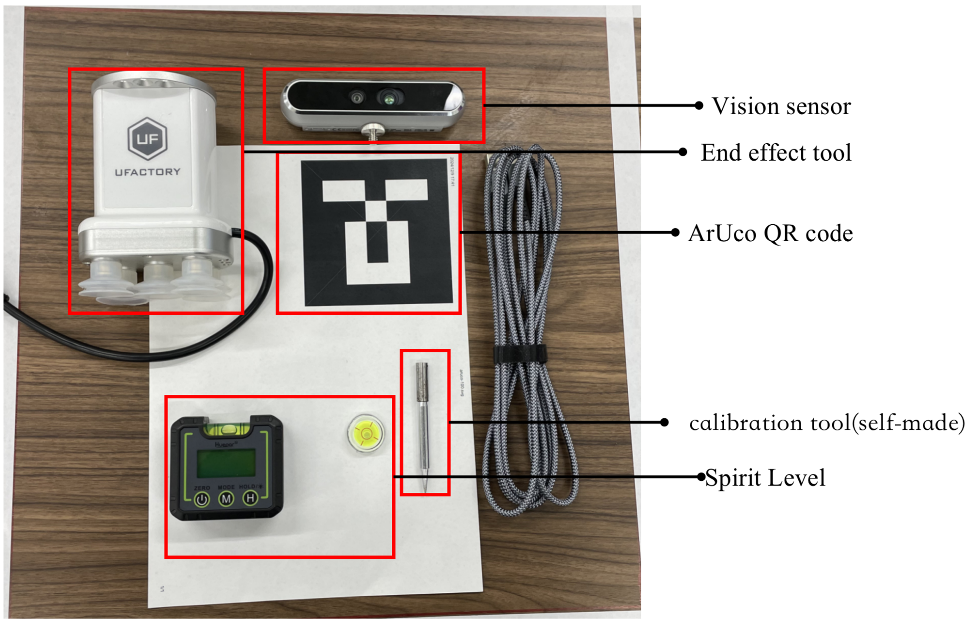


Figure 3.4: Show of calibration tools.

These requirements minimizes deviations caused by camera distortion and maximizes the accuracy of subsequent calibration. It is important to note that a USB 3.0 cable should be used for data exchange with the vision sensor to ensure efficiency.

After installing the visual sensor, print an ArUco QR code and attach it to a horizontal desktop (the experimental desktop should first be leveled using a level instrument). By using the OpenCV interface cv2.aruco.detectMarkers(), one can obtain the position and orientation of the center point of the QR code in the camera coordinate system, as shown in the Fig. 3.6. Different ID specify different QR codes. The recognition ID used in the code should match the one used in the experiment.

Our core objective is to obtain the transformation matrix that converts coordinates from the camera frame to the robot frame. The specific derivation formula is shown in the Eq. 3.1, 3.2.

$$^{ee}R_{ArUco} =^{ee} T_{cam} *^{cam} R_{ArUco} \tag{3.1}$$

$$^{ee}P_{ArUco} =^{ee} T_{cam} *^{cam} P_{ArUco} \tag{3.2}$$

The transformation matrix $^{ee}T_{cam}$ is a 4x4 matrix composed of a rotation matrix $^{ee}R_{cam}$ and a translation matrix $^{ee}P_{cam}$, as illustrated in the Eq. 3.3.

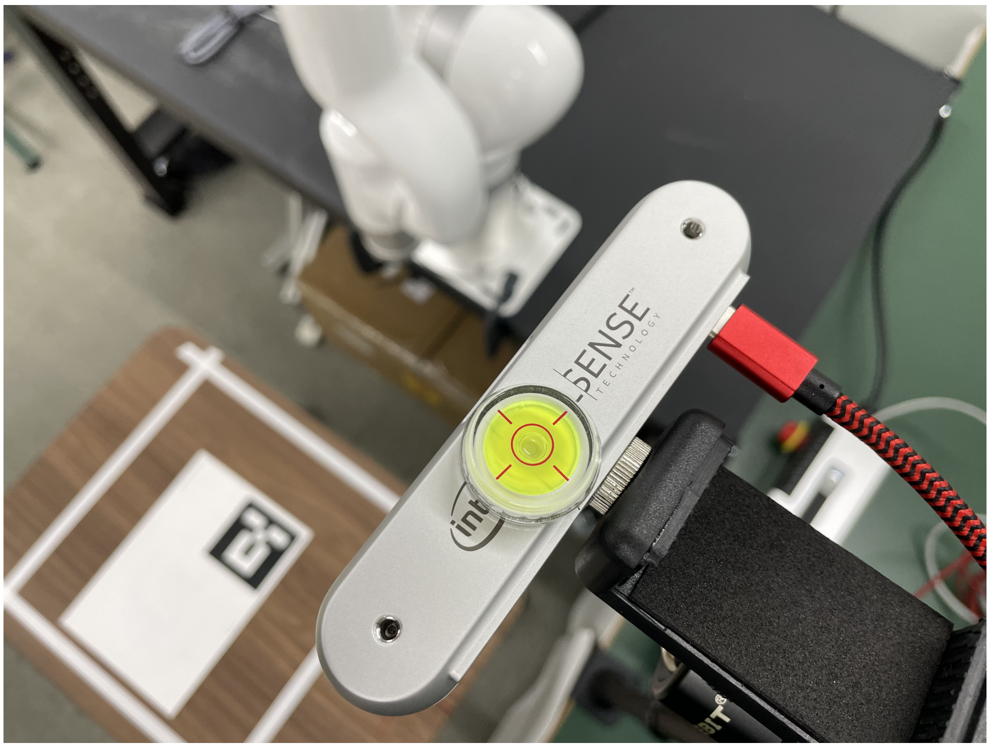


Figure 3.5: Installation requirements for the vision sensor.

$$ {}^{ee}T_{cam} = \begin{pmatrix} {}^{ee}R_{cam} & {}^{ee}P_{cam} \\ 0 & 1 \end{pmatrix} \tag{3.3} $$

Based on the prior installation of the vision sensor, we can ascertain the attitude change from the camera coordinates to the robot's base coordinates, which involves rotation around the y axis -90 degree, represented as $R_y(-90)$, as illustrated in the Eq. 3.4.

$$ R_y(-90) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tag{3.4} $$

However, the unknown component is the translation matrix [x, y, z]. Determining this translation matrix is the focus of our current calibration process. Firstly, we set a rough and safe value P_{e1} to ensure that the calibration tool does not collide with the table surface. After the robotic arm operates, the results are obtained as shown in the Fig. 3.7, which reveals a significant gap between the rough estimates of x, y, z and their true values.

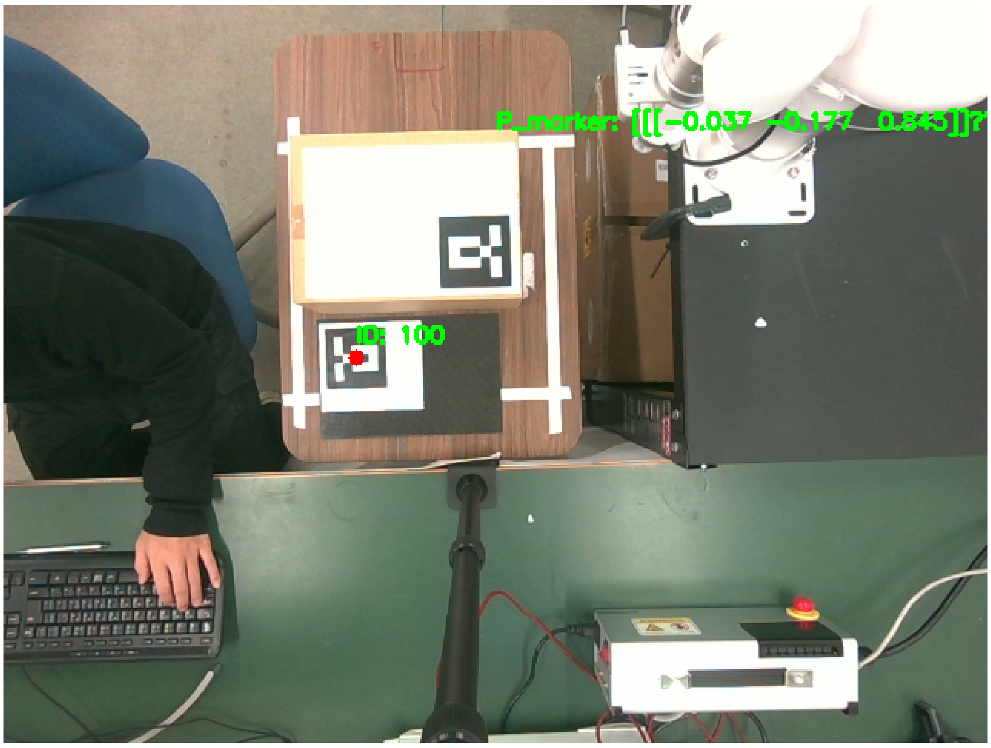


Figure 3.6: ArUco QR code detection example.

To achieve more precise calibration, an accurate correction of the z-axis discrepancy is first conducted. By using a ruler to measure the distance between the end of the calibrator and the desktop, repeated corrections are made to the z-value, ensuring safety throughout the process, until the result as shown in the Fig. 3.8 is achieved. This indicates that the z-value approaches the true value. After obtaining the true z-value, it facilitates the calibration of (x, y).

However, the calibration process for (x, y) is not as straightforward as that for z value. It is based on the derivation of a transformation formula, as shown in the Fig. 3.9.

Therefore, to correct the values of (x, y), measurements of the calibration tool and the center of the QR code are required, and these need to be referenced against the calculation of the transformation matrix. By repeating the above process, we obtain (x, y) values that converge towards the true values and achieve the results as shown in the figure Fig. 3.10.The end effector calibration tool accurately points to the center of the QR code. This indicates the completion of the calibration process and the result shown in the Eq. 3.5.

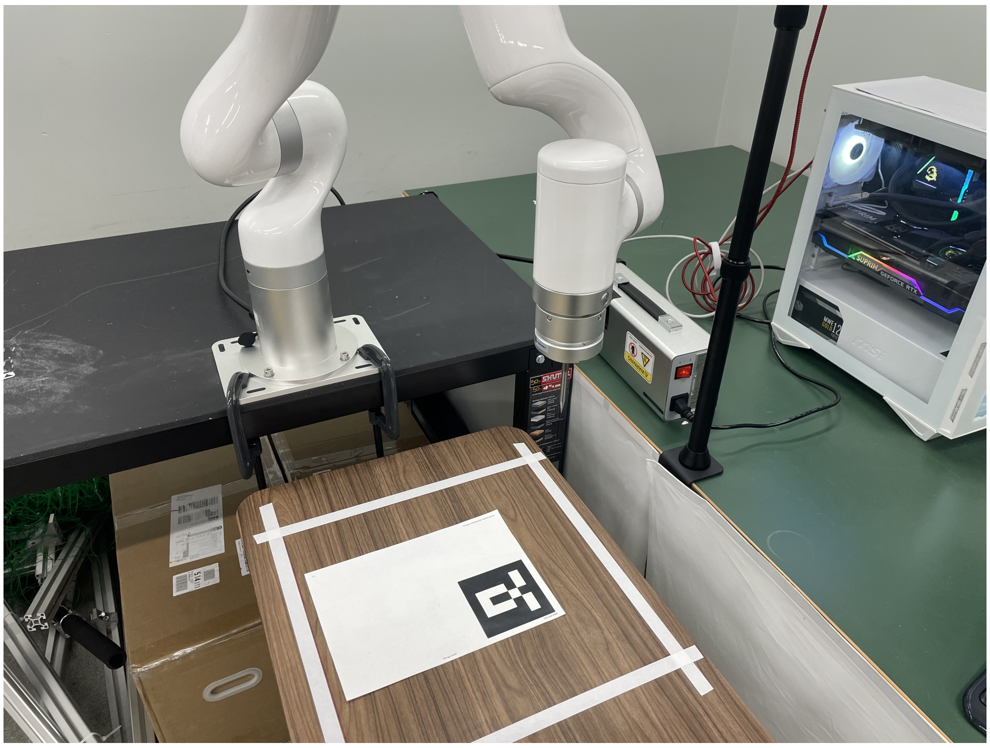


Figure 3.7: Significant gap between the rough estimates.

$$
{}^{ee}T_{cam} = \begin{pmatrix} -1 & 0 & 0 & 0.389 \\ 0 & 1 & 0 & 0.399 \\ 0 & 0 & -1 & 0.512 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}
$$

To ensure the accuracy of calibration, the calibration target is evenly positioned at nine different locations relative to the vision sensor. A dataset comprising translation matrices is obtained from these positions, and the average of this dataset is then calculated. This method ensures the validity and authenticity of the data, preventing errors that may arise from a single or limited number of measurements. Furthermore, it minimizes the impact of camera distortion. Consequently, the reliability and stability of subsequent experiments are improved.

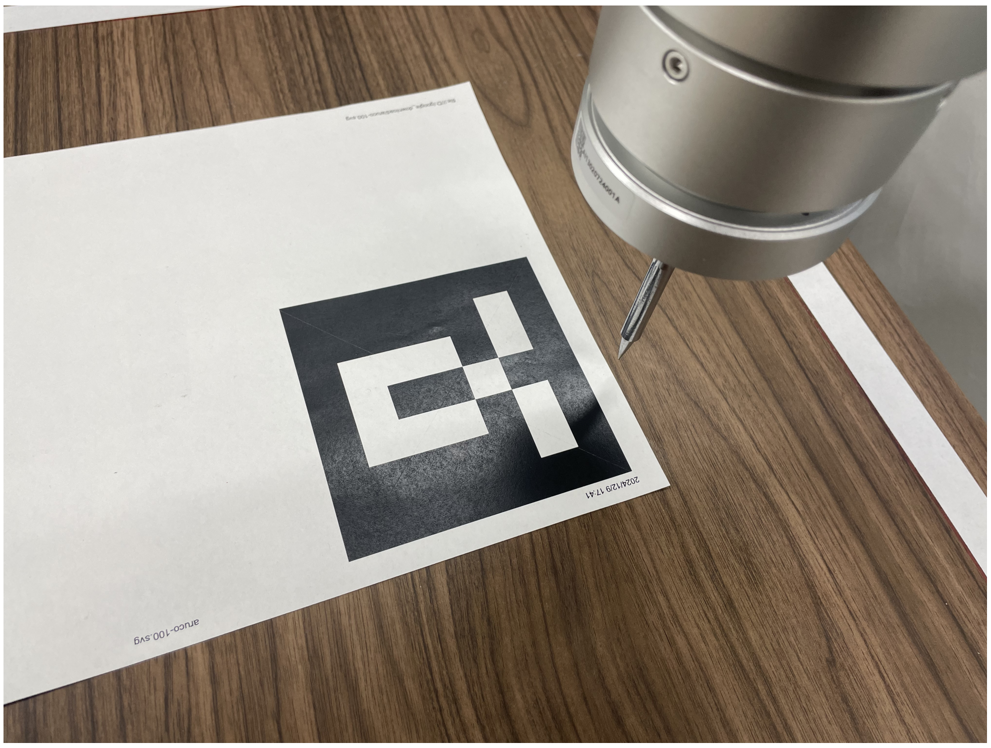


Figure 3.8: z-value approaches the true value.

3.2.3 Target object recognition

The recognition of target objects comprises two parts, with the first part focusing on the identification of object pose. Based on real-world scenarios, such as logistics and transportation, objects are generally manipulated on the flat surface. Therefore, our experiments only consider the 4D pose of the objects.

We attach ArUco QR codes to the target objects, ensuring that the corners align as closely as possible to guarantee the accuracy of the target object's pose estimation. Utilizing the OpenCV interface the cv2.aruco.estimatePoseSingleMarkers() function, we estimate the ArUco pose, as illustrated in the Fig. 3.11. The figure depicts the target object along with the ArUco QR code attached to it, while also visualizing the ArUco coordinates. In this way, we indirectly obtain the pose of the object $(0, 0, \theta)$.

The second part involves the identification of the object's dimensions Width, Depth, Height, denoted as (W, D, H). We employed the semantic segmentation model, Kirillov *et al.* [46], to segment the target object, which

$$^{ee}P_{ArUco} = ^{ee}T_{cam} * ^{cam}P_{ArUco}$$

$$\begin{bmatrix} X_{ee} \\ Y_{ee} \\ Z_{ee} \\ 1 \end{bmatrix} = \begin{pmatrix} -1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & -1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix}$$

$$X_{ee} = -X_{cam} + dx$$
$$Y_{ee} = Y_{cam} + dy$$
$$Z_{ee} = -Z_{cam} + dz$$

Figure 3.9: Derivation of a transformation formula.

provides the fine-grained mask of the target object.

This mask is a binary array with the same resolution as the input image, where pixels belonging to the target object are assigned a value of 1, and those not belonging are assigned a value of 0. By calculating the mean of the mask, specifically averaging the number of value 1 in each row, we can estimate object's width. Similarly, averaging the number of value 1 in each column allows for the estimation of object's depth. To estimate height, we intersect of the mask with a depth image and then compute the average the depths of the intersecting pixels.

Using these methods, we efficiently and accurately estimate the pose of the target object $(0, 0, \theta)$ and its dimensions (W, D, H), facilitating subsequent experiments.

3.2.4 Target object suction

Unlike traditional grasping strategies, we employ suction to manipulate objects. The end effector we use is a suction cup, which enables efficient handling and grasping of large target objects, especially for those that exceeds the maximum opening width of the gripper. Given that our practical applications typically involve factory sorting and delivery sorting, where objects often have a minimum width of approximately 150 mm, the use of

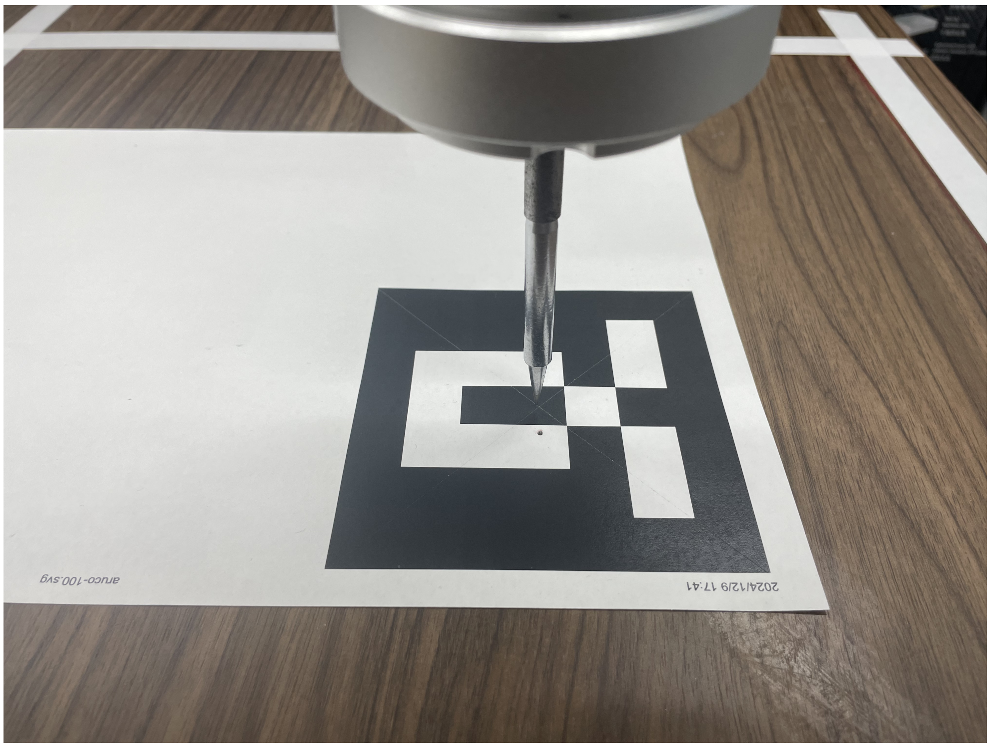


Figure 3.10: x, y, z-value approaches the true value.

large grippers will cost a lot. Therefore, suction cups [47] are highly suitable for both practical scenarios and experimental implementations.

Based on the object mask obtained from the target object recognition module, we get the center point $P_c(x, y)$ of the object, by meaning the value 1 position. And determine the depth d of this point using depth images. Thus, the position for suction is determined as $P_c(x, y, d)$. Combining the position $P_c(x, y, d)$ with the orientation $(0, 0, \theta)$, we convert these into coordinates relative to the robotic arm's base using a transformation matrix ${}^{ee}T_{cam}$. These coordinates are then input into the inverse kinematics control interface of the robotic arm to execute the suction operation.

3.2.5 Target object pose adjustment

Our model adjusts the orientation of objects to enhance the utilization rate during the packing process. However, in the real world, this is a highly complex procedure. To accomplish this, it is necessary to employ rational and efficient motion planning and motion control strategies. The definitions

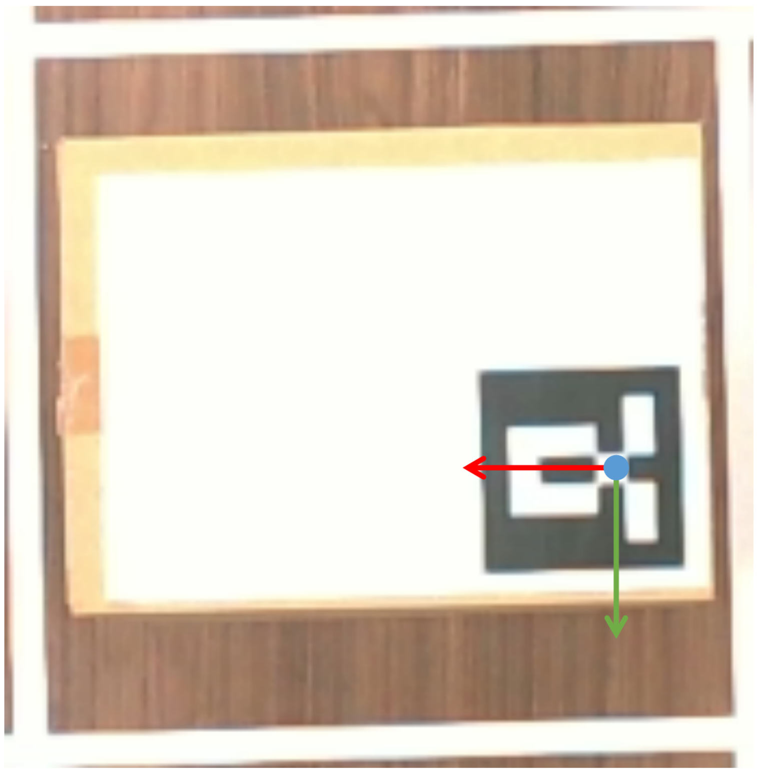


Figure 3.11: Target object along with the ArUco QR code .

of these six orientations are referenced from Fig. 2.4 in the Method model definition. Pose 1 primarily involves the estimation of the target object's pose. Pose 2 to 5 are mainly adjustments and variations based on the orientation 1, as shown in the Table 3.4. Below, we will elaborate on the adjustments to the object's six orientations in detail.

Pose$_1$ allows successful suction based on the target object's pose$P_c(x, y, d, 0, 0, \theta)$, as shown in the Fig. 3.12. After suction successfully in Pose$_1$, rotating 90 degrees around the z-axis can get the Pose$_2$.

However, Pose$_3$ and Pose$_4$ cannot be suctioned directly. We calculate the center point of a side surface based on the pose and dimensions of the target object, adjust the robotic arm to suction at this center point, and then move to the zero pose to achieve Pose$_3$. Pose$_4$ is rotating Pose$_3$ 90 degrees around the z-axis, as shown in the Fig. 3.13

Pose$_5$ and Pose$_6$ require more robotic arm control. Similarly, we calculate the center point of a side surface based on the pose and dimensions of the target object, adjust the robotic arm to suction at this center point, and

Orientation	**Rotation**	**Dimensions**
$Pose_1$	$*$	(W, D, H)
$Pose_2$	$R_z(-90)$	(D, W, H)
$Pose_3$	$R_x(-90)$	(W, H, D)
$Pose_4$	$R_x(-90) * R_z(-90)$	(H, W, D)
$Pose_5$	$R_y(-90) * R_z(-90)$	(H, D, W)
$Pose_6$	$R_y(-90) * R_z(-90)$	(D, H, W)

Table 3.4: Orientation detail for $Pose_1 \sim Pose_6$.

then rotate 90 degrees around the x-axis. Subsequently, the object is placed on the surface and regrasped at the top center point to achieve Pose$_5$. Pose$_6$ is obtained by rotating Pose$_5$ 90 degrees around the z-axis, as shown in the Fig. 3.14.

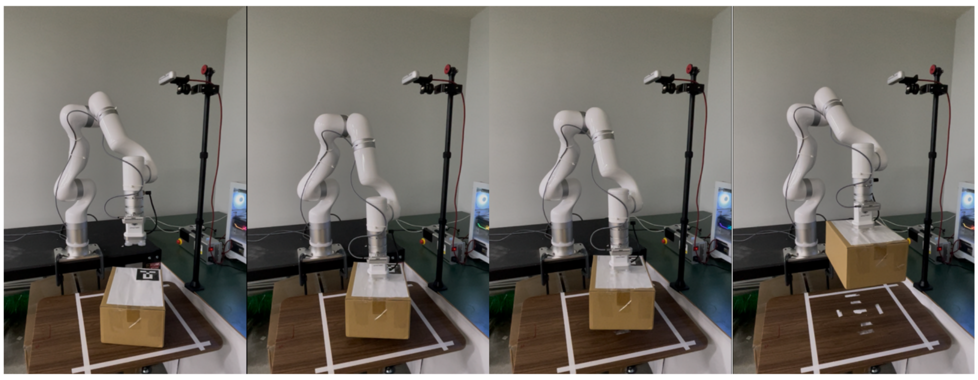


Figure 3.12: Real robot suction target object in the $Pose_1$.

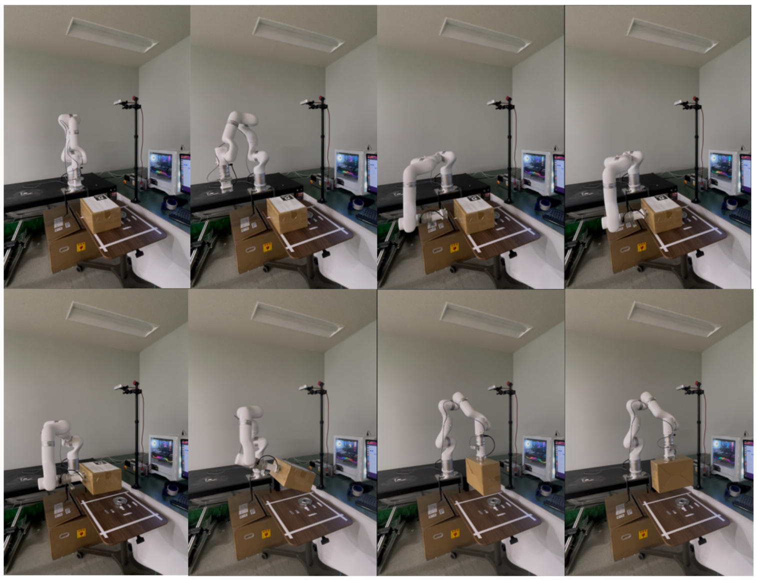


Figure 3.13: Real robot suction target object in the $Pose_3$ and $Pose_4$.

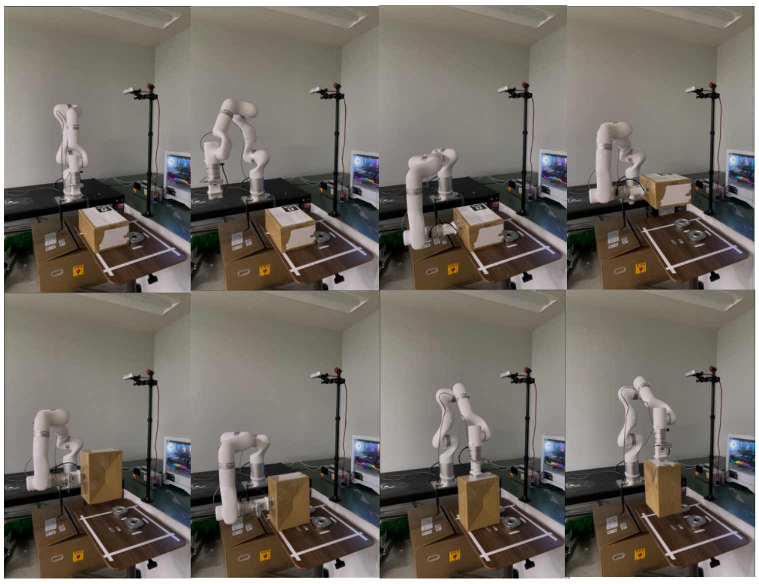


Figure 3.14: Real robot suction target object in the $Pose_5$ and $Pose_6$.

Chapter 4

Result

4.1 Physics Heuristics Validation

We compare the physics heuristic with algorithms convexHull-1 and convexHull-α on CoppeliaSim. The bin dimensions $W = D = H = 0.6m$. Objects are randomly generated with dimensions $w_i, d_i, h_i \in [0.03, 0.3]m$. In this experiment, based on the stable action map computed by convexHull-1 or convexHull-α, a random position considered to be stable for placement is selected at each time step. The stability of the bin objects are checked after each placement. The runtime of the two algorithms and the number of un-stable placement are reported in Table 4.2. Based on Table 4.2, We find that convexHull-α significantly surpasses convexHull-1 *w.r.t.* the accuracy of stability check. There was only one instance where convexHull-α incorrectly assessed the stability. We suspect this is due to the stable issue of the physical engine. In addition, convexHull-1 and convexHull-α have similar runtime which indicate the efficiency of convexHull-α.
Meanwhile, we also compared our approach with Zhao *et al.* [48]. Unlike our proposed method, which only relies on computer vision, Zhao *et al.* [48] method need the knowledge of the physical properties of each placed object, such as the center of mass. This process not only results in slower computation for the physical analysis algorithm but also demands high precision from the sensors. Consequently, Zhao *et al.* [48] cannot guarantee stable placement, exhibits slow computation, and requires more sensors.

	convexHull-1	**convexHull-α**
Object number	3000	3000
Fall number	153	1
Time cost(s)	4203.3	4452.6
Per cost(s)	1.40 - 1.00	1.48 - 1.00
Fall rate	5.1%	0.03%

Table 4.1: Comparison of convexHull-1 and convexHull-α.

	Mass information	**Time complexity**	**Space complexity**
Ours	No need	O(k)	O(m^2)
[48]	Need	O(NlogN)	O(N)

Table 4.2: Comparison of physics heuristics.

We evaluate and compare the efficiency of algorithms through time complexity and space complexity. As shown in 4.2, our algorithm has an highly efficient time complexity, which only depends on the size of the placed objects. In contrast, Zhao *et al.* [48] time complexity increases with the number of placed objects. Regarding space complexity, our algorithm's requirement is only related to the size of the bin, where Zhao *et al.* [48]'s space complexity is associated with the number of placed objects. Our algorithm becomes more efficient as the number of objects increases.

4.2 Simulation result

We use RS [13] bin packing dataset to generate upcoming object, in order to test our model and robot simulation. We have implemented a series of interfaces using Python to control and manipulate the robot arm in simulations to complete the entire packing process. These interfaces include: an inverse kinematics control interface for the robot arm, a vision sensor conversion interface, the RL model interface, and a transformation interface for converting image coordinates to robotic coordinates. These interfaces have been integrated into a Robot class for easy and direct using.

We show some result in the Simulation, as shown in the Fig. 4.1.

4.3 DRL framework result

RS [13] bin packing dataset is leveraged to train and test the proposed DRL framework. To evaluate the effectiveness of our proposed method, the result reported in Zhao *et al.* [13] is our baseline as we share the same setting. Consistent with previous studies, we employed space utilization (Uti.) as the metric to evaluate the bin packing policy, where a higher value indicates better performance. We test on the dataset RS, CUT-1, and CUT-2 [13], which is summarized in Table 4.3.

The results show that our method achieves higher Uti. with fewer training epochs. Additionally, we analyzed the RS test results, comparing each test's

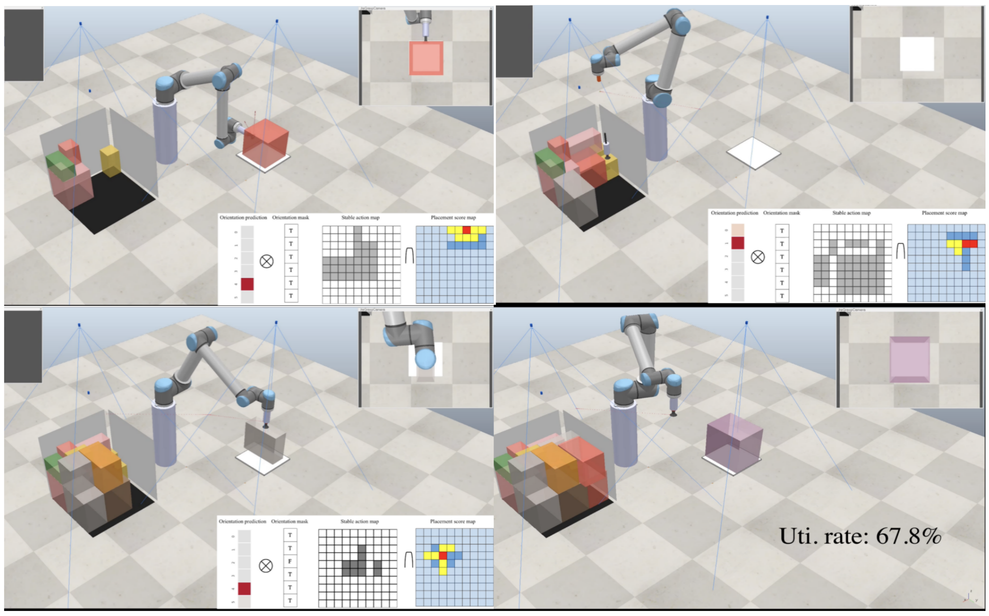


Figure 4.1: Simulation experiment result.

	RS	**CUT-1**	**CUT-2**	**epoch**
Ours	61.2%	63.3%	62.5%	18.6k
[13]	50.5%	60.8%	60.9%	100k

Table 4.3: RL framework comparison and test result.

Uti with the standard deviation of the object volumes in the object sequence. Specifically, larger standard deviation indicates greater volume difference among the objects. As shown in Fig. 4.2, we found that the our model trained on the RS dataset is not affected by the differences in object volume within the sequence.

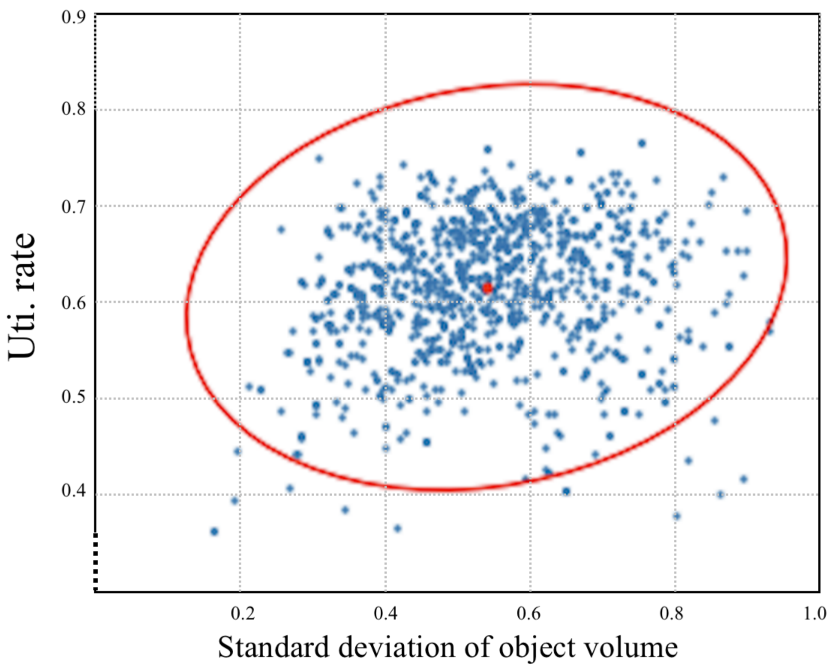


Figure 4.2: Space utilization of our model independent of the standard deviation in object volume.

Chapter 5

Conclusion

One of the key contributions of this work is proposal a highly reliable physics heuristic algorithm that ensures the stability of object placement in complex multi-stack environments. The algorithm generates a bottom-to-top depth heightmap of the bin and employs a sliding window to traverse the heightmap, checking the stability of each potential placement position. To address the limitations of existing convex hull-based methods, we introduce the convexHull-α algorithm, which considers the intersection of convex hulls across multiple layers and maintains an empty map to ensure that the supporting force originates from the ground. Empirical results demonstrate that convexHull-α significantly outperforms convexHull-1 in terms of stability check accuracy, with a fall rate of only 0.03% compared to 5.1% for convexHull-1.

Furthermore, the integration of object rearrangement into the DRL framework allows the robot manipulator to change the orientation of incoming objects, thereby improving space utilization without incurring additional time costs. This is achieved by formulating the problem as a Markov Decision Process (MDP) and employing two independent actor networks to predict object orientation and placement position, respectively. The physics heuristic algorithm generates stable action maps for each potential orientation, guiding the orientation prediction process. Subsequently, the placement agent network utilizes these stable action maps to determine the final placement position, ensuring the stability of the packed objects.

The effectiveness of the proposed DRL framework is evaluated through both simulation and real robot experiments. In simulations, the model achieves higher space utilization rates compared to baseline methods, requiring fewer training epochs for convergence. This suggests that the integration of the physics heuristic algorithm not only ensures stability but also improves the training efficiency of the DRL model. The real robot experiments further validate the practicality of the proposed approach, demonstrating the ability to accurately recognize object poses, suction objects, adjust their orientations.

In conclusion, the strengths of the main arguments presented in this thesis are supported by both theoretical and empirical evidence. The convexHull-α algorithm, as a novel physics heuristic, provides a reliable and efficient way to check the stability of object placements in complex multi-stack environments. The integration of object rearrangement into the DRL framework enhances space utilization without compromising stability or increasing time costs. The experimental results, both in simulations and on a real robot, demonstrate the effectiveness and practicality of the proposed approach, achieving higher space utilization rates with fewer training epochs compared to baseline methods.

In the future, we aim to make our physics heuristic algorithm more accurate by precisely predicting each stable placement position, and to improve the training efficiency of the DRL model. Additionally, we will incorporate the method proposed by Gao *et al* [30] [31] and Li *et al* [49] to grasp and pack irregular objects in the real world and attempt to propose a strategy for packing irregular and uneven objects [50] in complex real-world environments.

References

[1] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, “Solving a new 3d bin packing problem with deep reinforcement learning method,” *arXiv preprint arXiv:1708.05930*, 2017.

[2] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, “Optimal packing and covering in the plane are np-complete,” *Information Processing Letters*, vol. 12, no. 3, pp. 133–137, 1981.

[3] S. Martello, D. Pisinger, and D. Vigo, “The three-dimensional bin packing problem,” *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000.

[4] T. G. Crainic, G. Perboli, and R. Tadei, “Extreme point-based heuristics for three-dimensional bin packing,” *Informs Journal on computing*, vol. 20, no. 3, pp. 368–384, 2008.

[5] K. Karabulut and M. M. İnceoğlu, “A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method,” in *International Conference on Advances in Information Systems.* Springer, 2004, pp. 441–450.

[6] C. T. Ha, T. T. Nguyen, L. T. Bui, and R. Wang, “An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet,” in *Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part II 20.* Springer, 2017, pp. 140–155.

[7] E. Hopper and B. C. Turton, “An empirical investigation of metaheuristic and heuristic algorithms for a 2d packing problem,” *European Journal of Operational Research*, vol. 128, no. 1, pp. 34–57, 2001.

[8] K. Karabulut and M. M. İnceoğlu, “A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method,” in *International Conference on Advances in Information Systems.* Springer, 2004, pp. 441–450.

[9] F. Wang and K. Hauser, "Dense robotic packing of irregular and novel 3d objects," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1160–1173, 2021.

[10] S. Ali, A. G. Ramos, M. A. Carravilla, and J. F. Oliveira, "On-line three-dimensional packing problems: A review of off-line and on-line solution approaches," *Computers & Industrial Engineering*, vol. 168, p. 108122, 2022.

[11] S. Yang, S. Song, S. Chu, R. Song, J. Cheng, Y. Li, and W. Zhang, "Heuristics integrated deep reinforcement learning for online 3d bin packing," *IEEE Transactions on Automation Science and Engineering*, 2023.

[12] K. Pertsch, Y. Lee, Y. Wu, and J. J. Lim, "Guided reinforcement learning with learned skills," *arXiv preprint arXiv:2107.10253*, 2021.

[13] H. Zhao, Q. She, C. Zhu, Y. Yang, and K. Xu, "Online 3d bin packing with constrained deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, 2021, pp. 741–749.

[14] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[15] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *2019 Third IEEE international conference on robotic computing (IRC)*. IEEE, 2019, pp. 590–595.

[16] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation," *arXiv preprint arXiv:1610.00633*, vol. 1, p. 1, 2016.

[17] C. Zhang, Y. Wu, Y. Ma, W. Song, Z. Le, Z. Cao, and J. Zhang, "A review on learning to solve combinatorial optimisation problems in manufacturing," *IET Collaborative Intelligent Manufacturing*, vol. 5, no. 1, p. e12072, 2023.

[18] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Management Science*, vol. 44, no. 3, pp. 388–399, 1998.

[19] B. S. Baker, E. G. Coffman, Jr, and R. L. Rivest, “Orthogonal packings in two dimensions,” *SIAM Journal on computing*, vol. 9, no. 4, pp. 846–855, 1980.

[20] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, “Worst-case performance bounds for simple one-dimensional packing algorithms,” *SIAM Journal on computing*, vol. 3, no. 4, pp. 299–325, 1974.

[21] L. Wang, S. Guo, S. Chen, W. Zhu, and A. Lim, “Two natural heuristics for 3d packing with practical loading constraints,” in *PRICAI 2010: Trends in Artificial Intelligence: 11th Pacific Rim International Conference on Artificial Intelligence, Proceedings 11.* Springer, 2010, pp. 256–267.

[22] D. Zhang and W. Huang, “A simulated annealing algorithm for the circles packing problem,” in *International Conference on Computational Science.* Springer, 2004, pp. 206–214.

[23] Y. Jiang, Z. Cao, and J. Zhang, “Solving 3d bin packing problem via multimodal deep reinforcement learning,” 2021.

[24] S. Song, A. Zeng, J. Lee, and T. Funkhouser, “Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations,” *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4978–4985, 2020.

[25] Z. Rao, Y. Wu, Z. Yang, W. Zhang, S. Lu, W. Lu, and Z. Zha, “Visual navigation with multiple goals based on deep reinforcement learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5445–5455, 2021.

[26] L. Desanghere and J. J. Marotta, “The influence of object shape and center of mass on grasp and gaze,” *Frontiers in psychology*, vol. 6, p. 1537, 2015.

[27] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, “Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost,” in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 3651–3657.

[28] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, “Deep reinforcement learning for robotics: A survey of real-world successes,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 8, 2024.

[29] C. Banerjee, K. Nguyen, C. Fookes, and M. Raissi, "A survey on physics informed reinforcement learning: Review and open problems," *arXiv preprint arXiv:2309.01909*, 2023.

[30] Z. Gao, A. Elibol, and N. Y. Chong, "On the generality and application of mason's voting theorem to center of mass estimation for pure translational motion," *IEEE Transactions on Robotics*, vol. 40, pp. 2656–2671, 2024.

[31] ——, "Zero moment two edge pushing of novel objects with center of mass estimation," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 3, pp. 1487–1499, 2023.

[32] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," in *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2009, pp. 147–156.

[33] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[34] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[35] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

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

[37] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for ai-enabled wireless networks: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1226–1252, 2021.

[38] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 1321–1326.

[39] J. Hummel, R. Wolff, T. Stein, A. Gerndt, and T. Kuhlen, "An evaluation of open source physics engines for use in virtual reality assembly simulations," in *Advances in Visual Computing: 8th International*

Symposium, ISVC 2012, Rethymnon, Crete, Greece, July 16-18, 2012, Revised Selected Papers, Part II 8. Springer, 2012, pp. 346–357.

[40] G. Van Rossum and F. L. Drake Jr, *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995, vol. 620.

[41] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Introduction to pytorch," *Deep learning with python: learn best practices of deep learning models with PyTorch*, pp. 27–91, 2021.

[42] J. Sanders, *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional, 2010.

[43] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 1–10.

[44] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial markers for pose estimation: Overview, applications and experimental comparison of the artag, apriltag, aruco and stag markers," *Journal of Intelligent & Robotic Systems*, vol. 101, pp. 1–26, 2021.

[45] R. Horaud and F. Dornaika, "Hand-eye calibration," *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[46] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.

[47] H. Pham and Q.-C. Pham, "Critically fast pick-and-place with suction cups," in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 3045–3051.

[48] H. Zhao, C. Zhu, X. Xu, H. Huang, and K. Xu, "Learning practically feasible policies for online 3d bin packing," *Science China Information Sciences*, vol. 65, no. 1, p. 112105, 2022.

[49] C. Li, P. Zhou, and N. Y. Chong, "Safety-optimized strategy for grasp detection in high-clutter scenarios," in *2024 21st International Conference on Ubiquitous Robots (UR).* IEEE, 2024, pp. 192–197.

[50] H. Liu, L. Zhou, J. Yang, and J. Zhao, "The 3d bin packing problem for multiple boxes and irregular items based on deep q-network," *Applied Intelligence*, vol. 53, no. 20, pp. 23 398–23 425, 2023.

Publications

[1] Li, C., Zhou, P., Chong, N. Y. 'Safety-optimized Strategy for Grasp Detection in High-clutter Scenarios'. In 2024 21st International Conference on Ubiquitous Robots (UR) (pp. 192-197). IEEE. 2024, June

[2] Zhou, P., Gao, Z., Li, C., Chong, N. Y. 'An Efficient Deep Reinforcement Learning Model for Online 3D Bin Packing Combining Object Rearrangement and Stable Placement'. In 2024 24th International Conference on Control, Automation and Systems (ICCAS) (pp. 964-969). IEEE. 2024, October