The Future of Retail: Intelligent Visual and Spatial Robotic System for Next-Gen Shopping Checkout

Ting Hsiang, Chiu
dept. of Information Management
National Taiwan University
Taipei, Taiwan
louiechiu@ntu.im

Shao Yu, Chu
dept. of Information Management
National Taiwan University
Taipei, Taiwan
b06705028@ntu.edu.tw

Christopher Giagoudakis
dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan
T08901101@ntu.edu.tw

***Abstract*—In this paper, we propose a highly dynamic robotic system intended to be used for shopping checkout applications. This robot utilizes the ITRI robot arm and a camera to detect, scan, and intelligently pack objects into a given container. To achieve as minimal latency as possible, as would be expected in a real-world checkout scenario, we present our own highly optimized packing algorithm using iterative 2D layer-based dimension tracing. Detection and scanning capabilities have been integrated into a single, inexpensive webcam to reduce fixed costs and the amount of required equipment. Testing on a variety of multi-sized objects has demonstrated compelling evidence that our solution is a strong candidate for successful commercialization.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Packing objects is, at first glance, a seemingly mundane task that we do not think twice about in our daily lives. However it is in fact a critical component of various industries such as retail, warehousing, and postal services. Unfortunately, packing is also laborious, repetitive, time-consuming, and even physically difficult for certain people such as the elderly. Though this has been recognized for some time, no real solution has been proposed and widely accepted by society. Thus, this still remains an open and relevant problem.

We have chosen to focus our project on solving this problem in the retail industry. Currently, all kinds of shops - from grocery stores to big department stores - employ people to scan and pack customers' items. This means higher costs to the stores through wages and insurance, higher risk of mistakes and consequently a loss in customer trust, and lost productivity when employees call in sick or use the bathroom. This in itself is poor enough already - but some stores make an even poorer decision and make the customers do the work themselves, using self-checkout systems which are unreliable and physically inconvenience the customer.

Instead, we have designed a dynamic checkout robot that can act as a complete shopping checkout solution. No such robot exists at the time of writing. The robotic system consists of four stages:

- Object detection: It is crucial that the robot is able to correctly locate the objects to be packed. This forms the foundation and sets up the success for the remainder of the steps. State-of-the-art edge detection and contour calculations were implemented in order to achieve this.
- Object manipulation: In both real-life packing and robotic packing, being able to manipulate the objects accurately is key. In robotic systems, this has been traditionally done using image-perspective matrix transformations, which are infamous for their complexity. Unlike these traditional applications, we in this project have derived a simplified approach which performs flawlessly and efficiently, without the need for elaborate and convoluted pose transforms.
- Scanning: Another significant issue that arises is that of scanning. Retail stores around the world almost exclusively use some form of barcode on their items to track and manage inventory. When an item is to be packed at checkout, it must first be scanned so that the store knows which objects have been sold. In our case, we use real QR codes which contain a serial number as an analogy of barcodes in real-world scenarios.
- Packing: The overall system then culminates in the packing procedure, by which the robot intelligently packs the objects into a given container. By considering the way human usually to pack, we propose a packing algorithm that minimizes the total height of packed objects. The computation time is also guaranteed to be reasonable for retail shopping scenarios.

The complete packing process is demonstrated in the video https://youtu.be/XeWJ8aPUBX8. Also, our source code is released at https://github.com/shaoyu0966/CheckoutBot.

II. Related Work

Several existing algorithms for packing have been implemented by other researchers, who have also recognized the packing problem as not being satisfactorily solved as of yet. However, these solutions are mainly geared towards an industrial setting, such as warehouse applications, instead of more customer-focused settings such as in a retail shop. Nevertheless, we present various works which served as interesting points of reference and inspiration in designing our own algorithm.

A. Packing Algorithms

Martello et al. also developed a solution to the packing problem in the paper 'The Three-Dimensional Bin Packing Problem'[3]. The problem constraints include orthogonal packing into rectangular shaped bins, which presents a similar formulation to our project. The authors designed a branch-and-bound algorithm, but stated that the problem proved extremely difficult to solve in practice. Even though the algorithm is theoretically capable of determining the optimal solution if enough time is spent, it often does not converge to optimality in a reasonable time limit. As a result, researchers have looked to other heuristic-based methods to guarantee a more computationally feasible solution. One such method that has proven popular is the Bottom-Left heuristic, proposed by Baker et al. in their paper 'Orthogonal Packings in Two Dimensions' [1], which seeks to minimize the height of any single object in the solution. This was a heuristic that was also adopted in our own algorithm.

B. Robot arm packing applications

Wang and Hauser in their paper 'Stable bin packing of non-convex 3D objects with a robot manipulator' [2], propose a new method of 3D packing by introducing a heightmap-minimization heuristic. Using these heightmap heuristics, the authors were able to achieve more stable and high-quality packing plans than other comparable 3D methods. We believe this to be a novel approach to packing that formed the base for our own 3D solution.

III. Methodology

Our robotic system can be separated into four distinct phases: object detection, object manipulation, scanning, and packing. We detail our system design for each phase below.

A. Object Detection

Consider an object placed in the field of view of a camera, in a random position and orientation. How is the camera able to tell not only where the object is in the image, but also to distinguish it from the background and indeed from other objects? If the camera cannot locate each of the objects to be packed, then the whole system will fail. As such this is a critical system component which must be designed and implemented to an accurate and repeatable standard.

Our implementation relies heavily on the OpenCV framework in Python. Given an input frame taken from the camera feed, Gaussian blurring is applied to reduce noise around the edges and improve the thresholding result. Binary thresholding with a value of 150 then prepares the image for use with OpenCV functions.

The OpenCV functions were applied in the following order: Canny, findContours, moments. Respectively, these provide edge detection, contour generation, and moment calculations. Puzzlingly, sometimes 'findContours' returned duplicates of the same contour. We eliminate this issue by designing a duplicate detection algorithm, whereby the Euclidean distance between the centroids of every contour is calculated. If the distance between centroids is greater than a determined threshold, then we conclude that the two contours are duplicates and one of them is removed. Then using the returned structure of image moments, we calculate the principal angles and centroids of each contour, which together completely define the location and orientation of each object in the image.

B. Camera Calibration

Object manipulation is a core functionality of our system. To ensure that the ITRI arm can grip objects precisely, reliably, and in real time, we design a linear-driven matrix mapping that will transform coordinates in the image frame to coordinates in the gripper frame.

Understanding the meaning of pose variables is a catalyst that allows us to now identify the relationship between the real-world coordinates of an object, and the corresponding pixels of the object in an image. Our custom transform procedure utilizes the linear relationship between pixels and real-world points, since no distortion was present in the camera. Visible distortion should be removed before running the algorithm, using camera calibration procedures described in section 2 related work. The following procedure illustrates the idea of calibration.

1) Three rectangular, rigid, white-taped blocks are placed in the workspace, at the edges of the camera field of view.
2) Using our object detection procedure, the centroids of the blocks x_{ci} are found in terms of pixel coordinates for:
$$\begin{bmatrix} x_{c_1} & x_{c_2} & x_{c_3} \\ y_{c_1} & y_{c_2} & y_{c_3} \\ 1 & 1 & 1 \end{bmatrix}$$
3) The robot end-effector is manually moved to the centre of each of these three blocks x_{ri}, y_{ri}. The real-world coordinates of the end-effector are then taken from the ITRI arm interface as:
$$\begin{bmatrix} x_{r_1} & x_{r_2} & x_{r_3} \\ y_{r_1} & y_{r_2} & y_{r_3} \\ 1 & 1 & 1 \end{bmatrix}$$
4) The coordinate pairs are mapped to each other by solving the following equation:
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{c_1} & x_{c_2} & x_{c_3} \\ y_{c_1} & y_{c_2} & y_{c_3} \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x_{r_1} & x_{r_2} & x_{r_3} \\ y_{r_1} & y_{r_2} & y_{r_3} \\ 1 & 1 & 1 \end{bmatrix} \quad (1)$$

Now, given any arbitrary pixel coordinate (x, y) in the image, we insert this into a vector $K_p = \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}$ and multiply by matrix A above. This returns a vector $K_r = \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix}$, which represents the x, y position of the coordinate in terms of the position of

the end-effector. This can then be sent to the robot arm with the *MOVP* command.

This allows the robot end-effector to move to any desired position as recorded by the camera. The z-coordinate can simply be measured and introduced as a constant in the program, since the vertical distance between the camera and workspace does not change.

Furthermore, the end-effector orientation must be adjusted in order to grip an object. Our object detection algorithm returns the principal angle, indicating the axis from which the pixel masses of the object are evenly distributed. Setting the gripper angle to this value forces the gripper to orient itself parallel to the principal axis and thus be in the correct position to grip the object.

C. Scanning

With object detection and manipulation frameworks constructed, we turn our attention to the scanning phase. The QR code on our objects contains a serial number as it often does in real life, which we can then match to an entry in our product database for the dimensions of the object. Since the objects start in the workspace in any random orientation, we derive a systematic process that guarantees locating the QR code in a maximum of three iterations.

1) The end-effector grips the object from the sides.
2) The object is lifted to the camera, which is running a scanning program implemented in Python. If a QR code is detected on the object side facing the camera, the algorithm exits.
3) If no QR code is detected, the end-effector rotates 180 degrees to expose the opposite side of the object and the camera looks for a code again.
4) If no code is detected, the object is placed in a dedicated *manipulate area*. The end-effector rotate 90 degrees of its grip on the object such that the two faces right next to the original are exposed. Steps 2-3 are then repeated.
5) Finally, if a QR code is still not detected, the end-effector adjusts its grip in the manipulate area once more, such that the final two faces are exposed. Steps 2-3 are repeated again, with the code guaranteed to be found by this point.

After we have our scanning done, we will proceed onto the next phase, which is to place the object in certain orientation setting.

D. Object Manipulation

This section describes our orientation tracking function, which takes two parameters as arguments: the current orientation of an object, and the desired final orientation of that object. The function will then calculate the sequence of manipulations required to obtain the desired orientation, and will send the commands to the robot arm.

This function will in fact be used at two different points in our program. The first time will be immediately after we have scanned an object, which will ensure that the object is placed in the next area (a dedicated intermediate area) in a predefined orientation. The second time will be right before packing begins, so that the object is in the correct orientation to be packed.

o make the representation of this function easier, we first define the following vector:

$$\begin{bmatrix} x & y & z \end{bmatrix} \tag{2}$$

The above is an example of an input to our function, and says that the current object has its longest edge parallel to the x-axis, its second longest edge parallel to the y-axis, and its shortest edge parallel to the z-axis.

$$f(\begin{bmatrix} x & y & z \end{bmatrix}, \begin{bmatrix} z & x & y \end{bmatrix}) \tag{3}$$

This is now an example of a call to our function. In this case the function should generate the following sequence of actions for the current object:

1) Moving the longest edge from being parallel to the x-axis, to now being parallel to the z-axis.
2) Moving the second longest edge from being parallel to the y-axis, to no being parallel to the x-axis
3) Moving the shortest edge from being parallel to the z-axis, to now being parallel to the y-axis.

1) First call: After successfully scanning QR code:

$$f(rand., \begin{bmatrix} x & y & z \end{bmatrix}) \tag{4}$$

Once the object has been scanned, we need to place it in a dedicated intermediate area with with other previously scanned objects. This is necessary because we need to know the dimensions of all objects before we run our packing algorithm by scanning their QR code.

So, the goal is to place the scanned object in the intermediate area in a pre-defined orientation, so that the robot knows the orientation when it comes time to pack. Our pre-defined orientation is as follows, where all directions are with respect to the robot arm base frame:

1) The longest dimension of the object should be parallel to the x-axis.
2) The second longest dimension of the object should be parallel to the y-axis
3) The shortest dimension of the object should be parallel to the z-axis.

2) Second call: Upon packing solution generated:

$$f(\begin{bmatrix} x & y & z \end{bmatrix}, rand.) \tag{5}$$

Now we need to take the object from the intermediate area in its known orientation, and manipulate it into the orientation required by the packing solution. Running through all the combinations we know there will be at most 6 different kinds of packing orientations to handle, and we code the sequence of actions for each one. Since the implementation is spatially abstract and difficult to convey, we don't go in-depth and discuss each case.

After these two different phases of manipulation, we are guaranteed to have the object in the correct orientation ready for the next phase, which is packing.

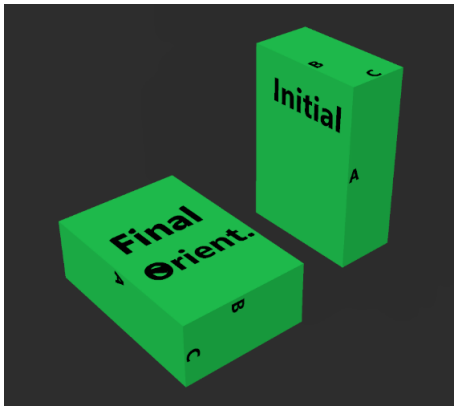

Fig. 1. Example of a orientation settings

E. Packing

During the packing phase, we decide the final position and orientation of each object such that the packing result is similar to those done by human. We observed that people tend to pack as dense as possible while utilizing the whole horizontal plane. Therefore, we set our objective to be minimizing the total height.

To obtain an optimal packing result, we form the problem into a MILP (mixed integer linear program) as follows.

- Parameters
  - A, B, C is the container size in X, Y, and Z dimension.
  - M, N, L is the width, depth, and height for item i.
  - U upper bound of coordinate difference.
- Variables
  - Non-negative Variables
    - (x_i, y_i, z_i) : Coordinates of item i's left-bottom-behind corner.
    - a_i, b_i, c_i : Size of item i in X, Y, and Z dimension.
    - h : Maximum Z coordinate of all corner points
  - Binary Variables

$$ox_{i,j} = \begin{cases} 0 \text{ if } x_j - (x_i + a_i) \geq 0 \\ 1 \text{ otherwise} \end{cases} \quad (6)$$

$$oy_{i,j} = \begin{cases} 0 \text{ if } y_j - (y_i + b_i) \geq 0 \\ 1 \text{ otherwise} \end{cases} \quad (7)$$

$$oz_{i,j} = \begin{cases} 0 \text{ if } z_j - (z_i + c_i) \geq 0 \\ 1 \text{ otherwise} \end{cases} \quad (8)$$

$$eam_i = \begin{cases} 0 \text{ if } a_i = M_i \\ 1 \text{ otherwise} \end{cases} \quad (9)$$

$$ean_i = \begin{cases} 0 \text{ if } a_i = N_i \\ 1 \text{ otherwise} \end{cases} \quad (10)$$

$$eal_i = \begin{cases} 0 \text{ if } a_i = L_i \\ 1 \text{ otherwise} \end{cases} \quad (11)$$

$$ebm_i = \begin{cases} 0 \text{ if } b_i = M_i \\ 1 \text{ otherwise} \end{cases} \quad (12)$$

$$ebn_i = \begin{cases} 0 \text{ if } b_i = N_i \\ 1 \text{ otherwise} \end{cases} \quad (13)$$

$$ebl_i = \begin{cases} 0 \text{ if } b_i = L_i \\ 1 \text{ otherwise} \end{cases} \quad (14)$$

$$ecm_i = \begin{cases} 0 \text{ if } c_i = M_i \\ 1 \text{ otherwise} \end{cases} \quad (15)$$

$$ecn_i = \begin{cases} 0 \text{ if } c_i = N_i \\ 1 \text{ otherwise} \end{cases} \quad (16)$$

$$ecl_i = \begin{cases} 0 \text{ if } c_i = L_i \\ 1 \text{ otherwise} \end{cases} \quad (17)$$

- Constraints
  - Position Constraints

$$x_i + a_i \leq A \quad \forall i \in \{0, 1, 2, ..., n\} \quad (18)$$

$$y_i + b_i \leq B \quad \forall i \in \{0, 1, 2, ..., n\} \quad (19)$$

$$z_i + c_i \leq C \quad \forall i \in \{0, 1, 2, ..., n\} \quad (20)$$

– Orientation Selection Constraints

$$a_i - M_i <= U \cdot eam_i \quad \forall i \in \{0,1,2,...,n\} \quad (21)$$
$$M_i - a_i <= U \cdot eam_i \quad \forall i \in \{0,1,2,...,n\} \quad (22)$$
$$a_i - N_i <= U \cdot ean_i \quad \forall i \in \{0,1,2,...,n\} \quad (23)$$
$$N_i - a_i <= U \cdot ean_i \quad \forall i \in \{0,1,2,...,n\} \quad (24)$$
$$a_i - L_i <= U \cdot eal_i \quad \forall i \in \{0,1,2,...,n\} \quad (25)$$
$$L_i - a_i <= U \cdot eal_i \quad \forall i \in \{0,1,2,...,n\} \quad (26)$$
$$b_i - M_i <= U \cdot ebm_i \quad \forall i \in \{0,1,2,...,n\} \quad (27)$$
$$M_i - b_i <= U \cdot ebm_i \quad \forall i \in \{0,1,2,...,n\} \quad (28)$$
$$b_i - N_i <= U \cdot ebn_i \quad \forall i \in \{0,1,2,...,n\} \quad (29)$$
$$N_i - b_i <= U \cdot ebn_i \quad \forall i \in \{0,1,2,...,n\} \quad (30)$$
$$b_i - L_i <= U \cdot ebl_i \quad \forall i \in \{0,1,2,...,n\} \quad (31)$$
$$L_i - b_i <= U \cdot ebl_i \quad \forall i \in \{0,1,2,...,n\} \quad (32)$$
$$c_i - M_i <= U \cdot ecm_i \quad \forall i \in \{0,1,2,...,n\} \quad (33)$$
$$M_i - c_i <= U \cdot ecm_i \quad \forall i \in \{0,1,2,...,n\} \quad (34)$$
$$c_i - N_i <= U \cdot ecn_i \quad \forall i \in \{0,1,2,...,n\} \quad (35)$$
$$N_i - c_i <= U \cdot ecn_i \quad \forall i \in \{0,1,2,...,n\} \quad (36)$$
$$c_i - L_i <= U \cdot ecl_i \quad \forall i \in \{0,1,2,...,n\} \quad (37)$$
$$L_i - c_i <= U \cdot ecl_i \quad \forall i \in \{0,1,2,...,n\} \quad (38)$$

$$eam_i + ean_i + eal_i = 2 \quad \forall i \in \{0,1,2,...,n\} \quad (39)$$
$$ebm_i + ebn_i + ebl_i = 2 \quad \forall i \in \{0,1,2,...,n\} \quad (40)$$
$$ecm_i + ecn_i + ecl_i = 2 \quad \forall i \in \{0,1,2,...,n\} \quad (41)$$

– Non-overlapping Constraints

$$x_j - x_i - a_i >= -U \cdot ox_{i,j} \quad \forall i \in \{0,1,2,...,n\} \quad \forall j \in \{0,1,...,i-1,i+1,...,n\}$$
$$y_j - y_i - b_i >= -U \cdot oy_{i,j} \quad \forall i \in \{0,1,2,...,n\} \quad \forall j \in \{0,1,...,i-1,i+1,...,n\}$$
$$z_j - z_i - c_i >= -U \cdot oz_{i,j} \quad \forall i \in \{0,1,2,...,n\} \quad \forall j \in \{0,1,...,i-1,i+1,...,n\}$$

– Maximum Height Constraint

$$h \geq z_i + c_i \; \forall i \in \{0,1,2,...,n\} \quad (42)$$

- Objective Function

$$\min h \quad (43)$$

- Iterations
With all the constraints above, we are able to obtain object positions and orientations such that the total height is minimized. However, there might be objects placed in some position violating the nature of gravity. Thus, we then run the MILP for several extra iterations. Within each iteration, one constraint is removed and 6 new constraints are added such that the object with the maximum upperside z-coordinate among all unfixed objects is fixed and optimization are performed on all other unfixed objects.
We solve the MILPs with an optimization solver, Gurobi. Though integer programs are considered to be NP-hard, we investigated the solving time for our model and found that the solver is able to solve settings with about 30 objects within 1 second. We conclude that the solving time required is reasonable for real-world packing.

F. Experimental Setup

The setup for the workspace is shown as in Figure 2, and 3.

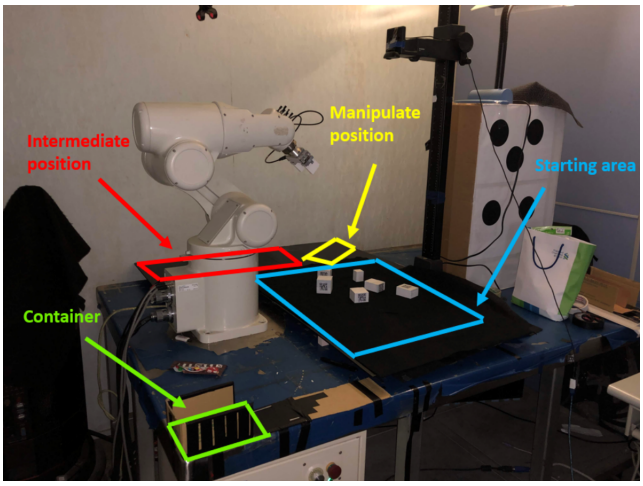

Fig. 2. Experiment setup from side.

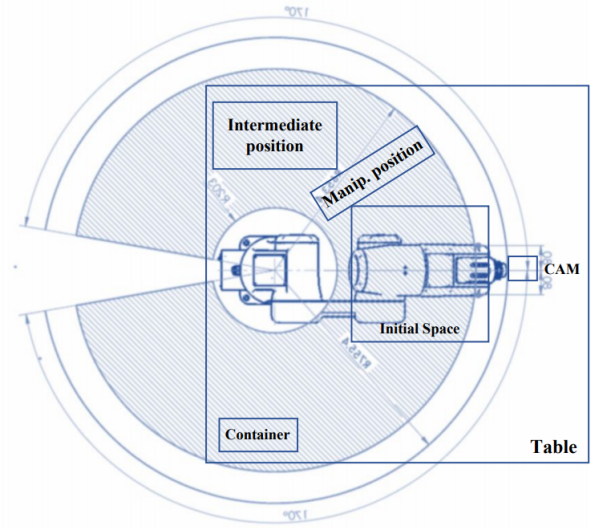

Fig. 3. Experiment setup from top-down

The following are the sizes for the blocks that are kept in our program's internal database. The ordering of dimensions is critical, and we always list the dimensions in order of largest to smallest.

TABLE I
OBJECT SIZE IN DATABASE

Serial No.	M (mm)	N (mm)	L (mm)
19	50	50	50
18	50	50	50
16	60	50	30
14	60	45	30
1	55	50	30
15	50	45	30
9	40	30	30

G. Results

Upon testing the system, we immediately came across several difficulties with the physical environment in the laboratory. The poor lighting made our object detection algorithm unable to pick up object contours, which we tried to overcome by moving the lamp to a better position and by covering the lamp with translucent materials to provide more diffuse light output. Together they made a noticeable improvement, but our object detection was still left a little unreliable. In addition, once we expanded the workspace of the robot we realized that different parts of the workspace were at different heights. We tried to fix this by using large, rigid pieces of plastic to create a constant flat surface across the whole workspace. However it did not work as well as expected, since the lack of support meant that the plastic sagged in places and still ended up creating a non-constant height. This reduced our ability to make precise movements with the robot arm, for example the robot might drop an object from too high because the surface beneath it was too low, which would cause the object to bounce and change its position/orientation.

Furthermore, we realized our implementation was also limited by the robot hardware itself. The small size of the grippers meant we could only pick up objects with a maximum dimension of 6cm in any direction, which limited our ability to show the intelligence of our packing algorithm. In addition, the grippers only had two settings - fully opened or fully closed - which meant that we could not grip and release objects inside the container because there was not enough space. As a result we had to instead push objects into the container, which introduced more object manipulation and therefore more potential for drift and error. Finally, the grippers often did not cleanly release the object, but would instead apply a slight torque causing the object to twist or bounce upon releasing.

However, with a good initial calibration of camera and real-world coordinates, most of these issues could be overcome to a satisfactory degree. We ran the program with six objects placed in random orientations in the starting space:

The robot was then able to successfully detect all objects, find the QR codes, and manipulate them into the intermediate area in our desired orientation, as in Fig. 5. Finally, the robot successfully generated the optimal packing solution, and packed the objects in the correct order, as in Fig. 6.

Fig. 4. Scattered objects from camera's view

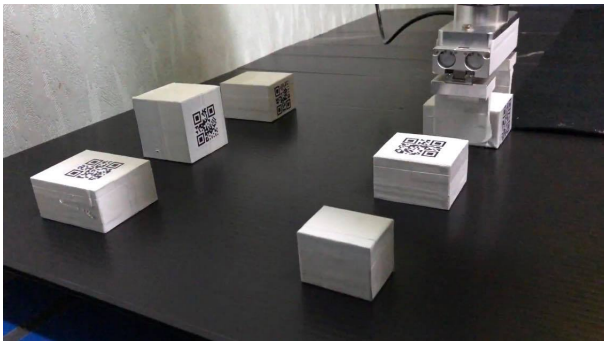

Fig. 5. Objects placed in intermediate position

H. Conclusion

In this paper we have presented a robotic system that is designed for use in shopping checkout scenarios. Although the robot was able to successfully pack objects into a container, we were limited by the physical environment and accuracy of the robot hardware in the performance and level of autonomy that we could attain. In addition, we realized that different optimization goals can produce different packing outcomes. In future work we would focus on tweaking our optimization algorithm such that we achieve more human-like packing behaviour. Despite this, given the success of the project and the number of new innovations in the system - such as the custom-designed orientation tracking process, and the proprietary packing optimization software - we believe our product could generate significant interest from the retail industry for further research and development or commercialisation.

REFERENCES

[1] Ronald L. Rivest Brenda S. Baker Ed Coffman. *Orthogonal Packings in Two Dimensions*. URL: https://www.researchgate . net / publication / 220617342_Orthogonal_Packings_in_Two_Dimensions.

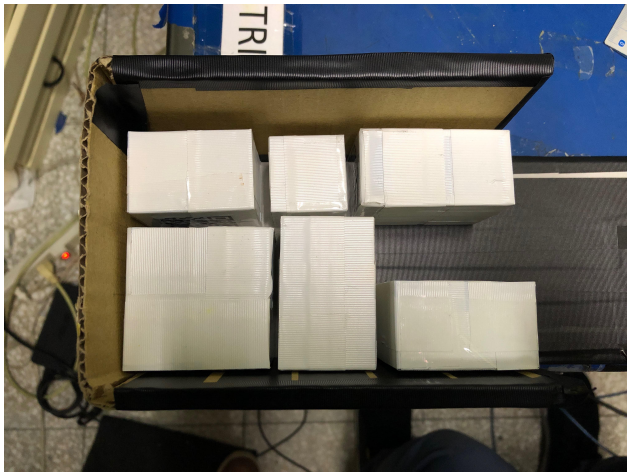

Fig. 6. Packing results

[2] Kris Hauser1 Fan Wang1. *Stable bin packing of non-convex 3D objects with a robot manipulator*. URL: https://arxiv.org/pdf/1812.04093.pdf.

[3] Daniele Vigo Silvano Martello David Pisinger. *The Three-Dimensional Bin Packing Problem*. URL: https://www.researchgate.net/publication/2353632_The_Three-Dimensional_Bin_Packing_Problem.

TABLE II
DIVISION OF WORK

Name	ID	Division of work
Christopher Giagoudakis	T08901101	$\frac{1}{3}$
Shao Yu, Chu	B06705028	$\frac{1}{3}$
Ting Hsiang, Chiu	B06705023	$\frac{1}{3}$