CSE 291G Final Project
Pick and Place Boxes using Robot Arms with Spades

Shuo Cheng Qiaojun Feng

***Abstract*— The ability of manipulating objects makes the robot a useful co-worker in various scenarios. To enhance the working efficiency, we want the robot to learn the skill of using different tools as our humans do. In this project, we design a policy for two robot arms to collaborate to pick and place boxes using spades as the end effector. We first detect the object boxes and the desired goal bin pose. Then we use PD control to move the robot arms. Experiments in simulation are conducted and the quantatitive results on different algorithms are reported. Our final version was ranked the 1st on the leaderboard with a success rate of 90.76% on June 7th, 2020.**

I. Problem Formulation

The manipulation task we tackle with is to pick up 10 boxes and place them into a bin. The novel setting is that the end effector of the robot arm is a spade and we are using two identical robot arms to collaborate. In different runs, there are some random factors that change the initial location and the size of the boxes, the size of the robot spades and the location and the size of the bin. A screenshot of the simulation experiment is shown in Fig. 1.

There are several sensors providing the observations of the environment as well as the robot state. Four fixed cameras provide the RGBD images as well as the instance segmentation including the boxes, the bin and the spade. We use $\mathcal{I} = \{I^i\}_{i=1}^4$ to represent them where I^i includes color, depth and instance ID. The sensors on the robot arms measure the angle and velocity of the robot joints and the pose of the links $\mathcal{J} = \{J^l, J^r\}$ of both arms. $J^l = (\theta^l, \dot{\theta}^l, s^l)$. $\theta^l, \dot{\theta}^l \in \mathbb{R}^7$ has the same dimension as the number of joints (7). $s^l = \{s_i^l\}_{i=1}^9$ includes the pose of 9 links and each of them is in SE(3). For most of the case we care about the last link which is the spade. The other arm is the same.

Assuming the model of the robot arm is known except the size of the spade, we want to control both of the robot arms to fulfill the task. Assume we have a discrete-time world model. At time $\mathbf{T} = t$, the policy is

$$\pi(\mathcal{I}_t, \mathcal{J}_t) = a_t \tag{1}$$

where the action a_t has the same dimension as the number of joints of the two robot arms. In each run, we generate a sequence of action $\{a_1, \ldots, a_t, \ldots\}$ until we decide the task is finished or the time limit for one single run is reached $t = \mathbf{T}_{max\_run}$. We also have a total time limit $\mathbf{T}_{max}$. When the accumulated time from all past runs has reach $\mathbf{T}_{max}$ we stop the whole process and start final evaluation.

The authors are with University of California, San Diego, La Jolla, CA 92093, USA. {scheng, qjfeng}@ucsd.edu. Authors listed in alphabetical order.

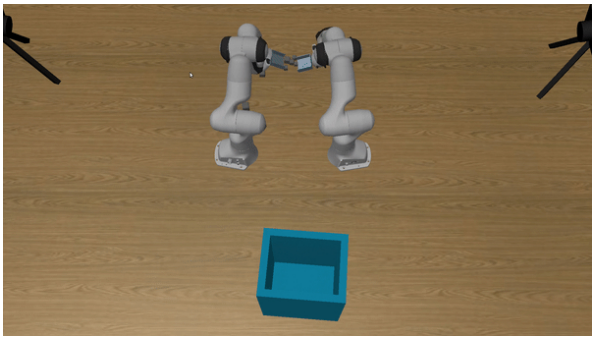

Fig. 1: The Simulation environment.

Define the pose of 10 boxes as $\{X^i\}_{i=1}^{10}$. Define the state of the bin as B. We can have a indicator function that check whether a box is in the bin

$$f_B(X^i) = \mathbb{1}(X^i \text{ is in } B) \tag{2}$$

And this is checked at the end of each run. Define the f^k as the indicator function in the k-th run, we can add up the total number of boxes that are placed in the desired location.

$$N = \sum_{k=1}^{K} \sum_{i=1}^{10} f_B^k(X^i) \tag{3}$$

assuming there are K runs in total. N should be a integer in $[0, 10K]$. Our evaluation metrics include success rate s and efficiency e

$$s = \frac{N}{10K} \tag{4}$$

$$e = \frac{N}{\mathbf{T}_{max}} \tag{5}$$

Problem. Given the observations $\{\mathcal{I}, \mathcal{J}\}$, we want to find a control policy $\pi(\mathcal{I}, \mathcal{J})$ to maximize the task success rate s and efficiency c.

II. Technical Approach

In this section, we present our system for solving the boxes pick and place problem. Our system contains two parts: the visual perception module and the planning and control module, we will describe each part in detail. In addition, we discuss some optimizations that we use to improve the efficiency of our approach.

A. Visual Perception

Visual perception plays a very important role in robot manipulation tasks. In our settings, the bin and the boxes are randomly placed in each scene, so it is necessary for us

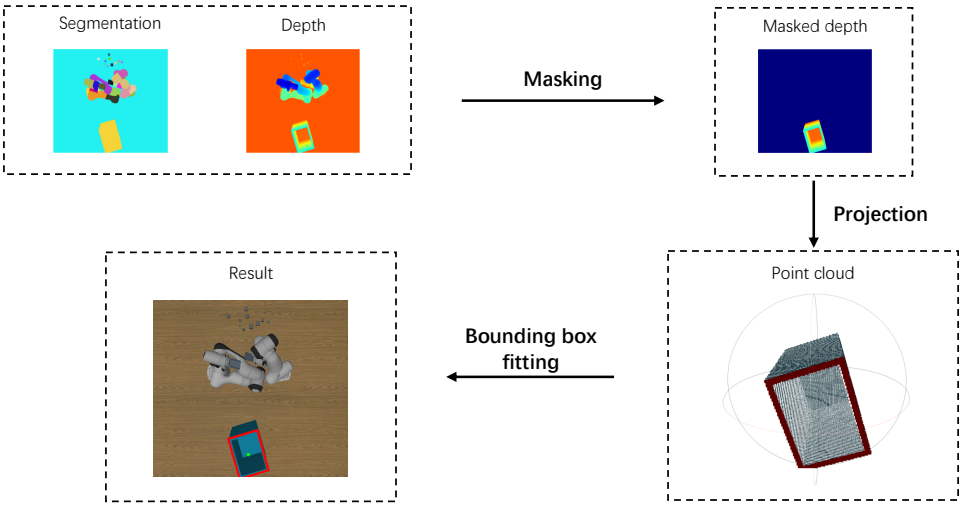

Fig. 2: Overview of our perception algorithm. We first identify the pixels and their depth values for the bin using segmentation results, these points are then projected into the world coordinate using the camera intrinsic and extrinsic matrices. We fit a bounding box for the top regions to determine the position of the top center and the orientation of the bin. Please refer to Sec.II-A for more details.

to estimate the positions (and orientations) for each objects. Our perception algorithm (please refer to Fig. 2 for more details) is built on top of the off-the-shelf perception results. Given ground-truth depth map and instance segmentation map captured by the top camera, we first identify the region belongs to the bin using the segmentation result. For each pixel (x, y) with depth value d that in the bin region, the homogeneous vector p is $(xd, yd, d, 1)$. Let K, T be the camera intrinsic and extrinsic matrix respectively, the projection matrix $H \in \mathbb{R}^{4\times4}$ can be written as:

$$H = T^{-1}K^{-1} \tag{6}$$

We then project the points that associating with the bin into the world coordinate using the project matrix:

$$p^w = Hp \tag{7}$$

Lastly, we determine the position and orientation of the bin by fitting a bounding box for the points at the top regions (which can be identified by comparing the z coordinate of each point), and calculate the center $p_c^w = (x_c, y_c, z_c)$ of the top region of the bin.

To determine the target pose of the end effector, we also need to know the length of the spade. For a specific frame, we read out the joint position p_r of the last link, and we calculate the center of the spade p_s (similar to calculating the center of the bin), the spade length l_s is $2|p_r - p_s|$. With the spade length and the pose of the bin, a delicate design would consider the orientation of the bin for computing the final target pose of the end effector. In this work, we set the translation as $(x_c + 2l_s, y_c, z_c)$, and we set the rotation as $(0, -\pi, 0)$. We found this simple design works very well in our experiments.

For aligning the spade to box locations, we first compute the center of each box. Observing the fact that our method can finish all the boxes in 3∼4 rounds (within 200 ms) in most of the scenes, we randomly select a location during each execution.

B. Trajectory Planning and Control

Without further explanation, we are referring to the spade link pose when we talk about the pose. The task of control is to move the end effector from the current pose T_c to the target pose T_t. There are multiple trajectory to reach that and the planning algorithm can choose the most desirable trajectory which should ideally be collision-free and smooth and stable.

When the current pose T_c and the target pose T_t are relatively far from each other, we do simple linear interpolation in the form of twist to determine the intermediate target. We first calculate the relative transformation of the target pose in the frame of the current pose

$$T_t^c = T_c^{-1}T_t \tag{8}$$

Then we convert the SE(3) representation into the 6-D exponential coordinates or twist

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \theta\xi = \log(T_t^c)^\vee \in \mathbb{R}^6 \tag{9}$$

where ξ_b is a unit twist and θ is a scalar indicating magnitude. We can change the scalar to control the speed of the transformation. Suppose we set the scalar as γ, the body twist is

$$\xi_b = \gamma\xi \tag{10}$$

and the new local target pose will be

$$T_t' = T_c \exp(\gamma\hat{\xi}) \tag{11}$$

To control the robot joint we need to find the connection between the link pose and the robot joint. We use the manipulator Jacobian $J(\theta) \in \mathbb{R}^{6\times7}$ to connect the spatial twist ξ_s with the joint velocity $\dot{\theta}$

$$\xi_s = J(\theta)\dot{\theta} \tag{12}$$

Notice here we need to first convert the body twist to be the spatial twist by timing the adjoint matrix of the spade's

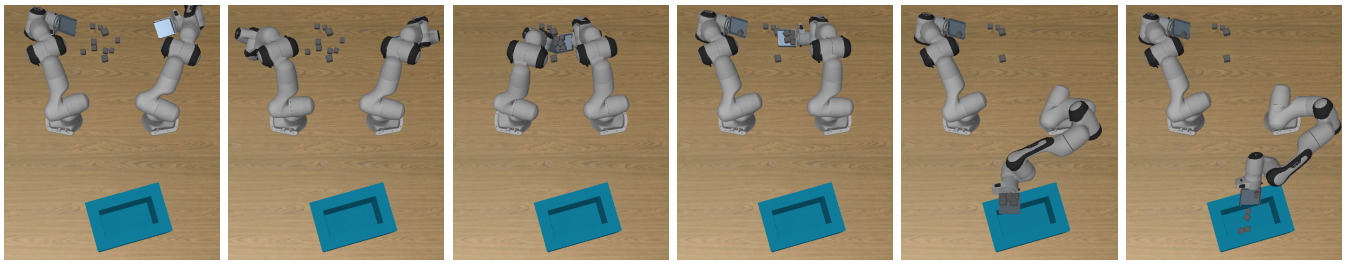

Fig. 3: The complete workflow for addressing the pick and place problem can be described as finite state machine. The figures here show 6 different ending configurations of each state. Please refer to Sec. II-C for more details.

current pose Ad^c

$$\xi_s = \mathrm{Ad}^c \xi_b \tag{13}$$

By taking the pseudo-inverse of manipulator Jacobian matrix, we can get the desired joint velocity

$$\dot{\theta} = J(\theta)^+ \xi_s \tag{14}$$

and the desired joint angle

$$\theta_{t+1} = \theta t + \Delta t \dot{\theta} \tag{15}$$

For further details please refer to [1].

Given the desired angle and the velocity of the joints, we use PD controller implemented by NVIDIA PhysX to control the force applied to the joints. The gravity, Coriolis, and external forces needed to keep the robot at current configuration are calculated by the provided function.

C. Finite State Machine

Inspired by the baseline implementation, our policy follows a simple circle-structured finite state machine model. The states are listed below (please refer to Fig. 3 for more details). The state transition route is $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$ to form a loop. For convenience, let $t(\cdot)$ be the translation, $q(\cdot)$ be the quaternion representation, and $e(\cdot)$ be the Euler angle with xyz order.

1) Set to initial pose. Drive the left end effector to pose $t(2, 1, 0)$ and $q(-1.5, -1, 1, -2)$ and drive the right end effector to pose $t(-2, 1, 0)$ and $q(-1.5, 1, 1, -2)$.
2) Align to the targeted box. Drive the left end effector to pose $t(x_b, 0.5, 0.67)$ and $e(\pi, -\pi/6, -\pi/2)$, and drive the left end effector to pose $t(x_b, -0.5, 0.6)$ and $e(\pi, -\pi/4, \pi/2)$ in 20ms, where x_b is the x coordinate of the selected box position.
3) Two spades move to each other to collect boxes. Drive the left end effector to pose $t(x_b, 0.07, 0.67)$ and $e(\pi, -\pi/6, -\pi/2)$, and drive the right end effector to pose $t(x_b, -0.07, 0.6)$ and $e(\pi, -\pi/4, \pi/2)$ in 30ms.
4) Pick up the boxes onto the right spade. First drive the left end effector to pose $t(x_b, 0.07, 0.77)$ and $e(\pi, -\pi/4, -\pi/2)$ in 5ms. At the meantime, drive the right end effector to pose $t(x_b, -0.07, 0.6)$ and $e(\pi, -\pi/1.8, \pi/2)$ in 20ms, and drive the left end effector to pose $t(2, 1, 0)$ and $q(-1.5, -1, 1, 2)$ in the next 15ms.
5) Right spade moves to the location of the bin. First lift up the spade by driving the right end effector to pose $t(x_b, -0.07, 1.1)$ and $e(\pi, -\pi/1.5, \bar{q})$ with our controller in 30ms, where $\bar{q}$ is current z-angle value. Then drive the right end effector to pose $t(-1, 0, 1.2)$ and $e(0, -\pi/3, 0)$ with our controller in 60ms. Finally arrive the bin location by driving the right end effector to pose $t(x_c + 2l_s, y_c, z_c)$ and $e(0, -\pi/1.2, 0)$ in 10ms, where (x_c, y_c, z_c) is the bin location and l_s is the spade length.
6) Right spade releases the boxes into the bin. Tilting the spade by driving the right end effector to pose $t(x_c + 2l_s, y_c, z_c)$ and $e(0, -\pi, 0)$ in 10ms.

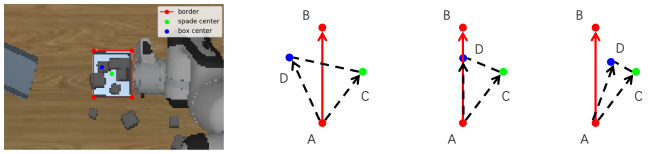

Fig. 4: We determine the spade state by checking the segment between the box center and the spade center and the four borders of the spade.

D. Optimizations

To further improve the efficiency of our method, we propose some extra components to optimize the pipeline.

Pick with greedy Different from sample the box location randomly and execute the picking operation, we develop another strategy for choosing the target box location. More specifically, for each box location, we first count the number of boxes in a given offset $\pm d$ in the x dimension (in our experiment, d is set to 0.05, the approximate half width of the spade), and we prioritize the locations with higher counts. We evaluate two picking strategy in the experiment section.

Check spade state during execution We observe that in some situations, the system executes the place operation with an empty spade, which wastes time and jeopardizes the efficiency. To check whether a box is inside the spade, we need to determine whether the segment between the center of the spade and the center of the box intersects with the four borders of the spade (please see Fig. 4 for more details). The state between two segments $\overline{AB}$ and $\overline{CD}$ can be determined by the following rules, where $\times$ is the cross product of two vectors:

- $(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot (\overrightarrow{AB} \times \overrightarrow{AD}) < 0$: point C and point D are on the different sides of segment $\overline{AB}$, so segment $\overline{AB}$ intersects with segment $\overline{CD}$.

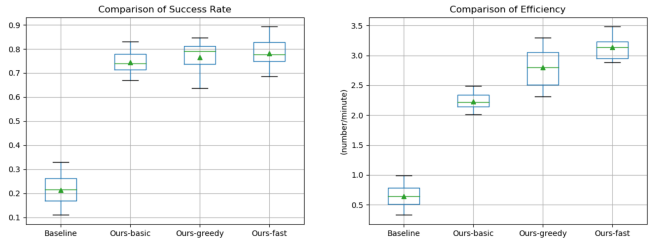

Fig. 5: Comparison of success rate and efficiency.

- $(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot (\overrightarrow{AB} \times \overrightarrow{AD}) = 0$: at least one of point C and point D is on the line of segment $\overline{AB}$.
- $(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot (\overrightarrow{AB} \times \overrightarrow{AD}) > 0$: segment $\overline{AB}$ does not intersect with segment $\overline{CD}$.

A box is inside the spade if and only if the segment between the box center and the spade center does not intersect with either one of the four borders. Once we found all the boxes are outside the spade, we change current state to phase 2 of our finite state machine and start the picking process again.

Skip to next scene before ending We notice that in most cases, our method can finish all the boxes in 3∼4 rounds, which consumes less than 200 ms – the maximal time for each run. In some cases, the boxes are distributed far away from the robot base, which can not be reached by the end effector. Based on these observations, we experimentally set up a valid region $x \in [-0.3, 0.2], y \in [-0.4, 0.4]$. Once we found there is no box in this valid region, we will skip to the next scene. In addition, observing that a complete pick-and-place operation consumes at least 40ms, we let our method skip to the next scene when the remaining time is shorter than 40ms.

III. Experiments

In this section, we evaluate our methods in the SAPIEN [2]–[4] environment. SAPIEN is a realistic and physics-rich simulated environment that hosts a large-scale set for articulated objects. In each run, the environment generate 10 boxes and 1 bin with different sizes and poses, the initial configurations of the robot arms and the sizes of the spades are also changed, but the bases of the robots and the cameras are fixed. We compare our different implemented versions with the provided baseline approach:

- **Baseline**: the baseline method sets up a fixed target pose for place the boxes in every run, which works poorly with the random bin pose. In addition, the controller implemented in the baseline method is also not stable, some boxes are dropped due to the low quality trajectory and perturbation.
- **Ours-basic**: the basic version utilizes the target pose computed by our perception module, which adapts to the dynamic environment better. Benefited from our implemented controller, we achieve more stable and faster control.
- **Ours-fast**: the fast version is built on the basic version, it checks the spade state and the valid region, to determine whether it needs to re-initialize the finite state machine or skip to the next scene.
- **Ours-greedy**: the greedy version is modified from the fast version, where the random picking operation is replaced by the greedy picking operation.

We repeat the experiments for 10 times[1], each time the environment runs for 2000ms in total. We report the results in Fig. 5.

We found that with our perception module and our implemented controller, our basic version can outperform the baseline method by a large margin. Our controller achieves stable and fast control, which reduces the operation time and avoids leaking boxes out. With accurate visual perception, the boxes will be located and placed correctly. We also notice that some optimizations such as checking spade state and valid region could further improve the efficiency and success rate. Adopting the greedy picking strategy will make the system unstable, even though sometimes it achieves a slightly higher success rate. Our final submission is the fast version, which is ranked the 1st on the benchmark with a success rate of 90.76%.

IV. Conclusion

In this project, we implement a simple yet effective system for solving the box pick and place problem. We evaluate our system on the public benchmark and achieve good results.

There are several limitation in current implementation. In the planning and the control phase, there is not a planning tool that generate a smooth trajectory based on the kinematic of the robot arm, but simple linear interpolation in the twist space. Though it appears to work in this simulation among similar configuration spaces, it cannot generalize to more complicated tasks. Also there is no explicit collision avoidance design for the planning phase. Actually there are many well-implemented motion planning library that can be helpful on such task.

Our future work would be to introduce some learning techniques to further improve the system and make it more adaptive and robust to the real and complicated environment.

References

[1] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[2] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su, "SAPIEN: A simulated part-based interactive environment," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[1]Random seeds used in our experiments: 12, 15, 25, 29, 37, 43, 55, 71, 80, 97.