Grasping with Chopsticks: Fine Grained Manipulation using Inexpensive Hardware by Imitation Learning

Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots and Siddhartha Srinivasa

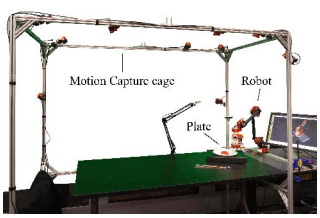
***Abstract*— Billions of people use chopsticks, a simple yet versatile tool, to pick up a wide variety of food items in their daily lives. We hope to leverage human demonstrations to develop autonomous chopsticks-equipped robot manipulation strategies for hard manipulation problems. The small, curved, and slippery tips of chopsticks require fine-grained control, which pose a challenge for picking up small objects. In this preliminary work, we explored imitation learning methods to learn to pick up small cube and ball-shaped objects from expert's teleoperation demonstrations. We trained a behavior cloning agent, a k-Nearest Neighbors agent, and a blending of both in robot-centric and object-centric representations. We found that blending of the two agents showed some promise in teaching the chopsticks robot to pick up small objects in the object-centric frame. However, there is still a need to incorporate adaptive real-time feedback in the learner to improve and generalize the manipulation performance, which points us to some plausible directions for future work.**

I. Introduction

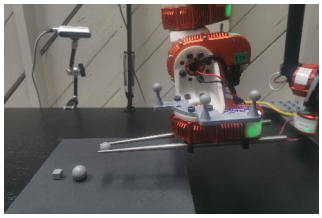
Humans have used chopsticks for thousands of years to eat a wide variety of food items with varying physical characteristics. The simplicity and efficacy of chopsticks' design has inspired researchers to adapt them for various applications such as meal assistance, surgery [1], and micro-manipulation, which has made it a versatile tool. However, chopsticks pose some challenges: their small, curved, and slippery tips require fine-grained manipulation and can make grasping a challenging task. But humans have achieved great success using chopsticks: every day, billions of people use chopsticks to pick up objects with varying shape, size, and deformability, including sushi, tofu, or noodles. Noticeably, humans also demonstrated impressive adaptability teleoperating a robot equipped with chopsticks to pick up everyday life objects [2]. In this work, we hope to leverage human expertise with chopsticks to develop autonomous chopsticks-based manipulation for challenging grasping tasks.

Researchers have successfully applied imitation learning to real-world grasping tasks. However, we are not aware of any prior work on applying imitation learning to tasks requiring fine-grained manipulation. The challenge in doing that is two-fold. Imitation learning learns a policy π that maps from state s to action a. Model-free imitation learning methods like behavior cloning and DAGGER learn the policy function by matching its action distribution to the expert action distribution. In doing so, they define a *surrogate* loss. However, minimizing the trajectory divergence between an expert and the learner does not necessarily lead to succeeding at the task that the expert was demonstrating. To remedy this, one might introduce a model (e.g., learning a dynamics model from interacting with a real robot system) and leverage rollouts from the model to further match the distribution (GAIL) [3] or use reinforcement learning methods to optimize a task reward while using imitation learning as a bootstrapping method [4]. This way, the quality of the model inevitably bounds the quality of the learned policy. But getting a model accurate enough for fine-grained manipulation tasks with contacts using inexpensive hardware can be challenging.

University of Washington, Seattle WA 98105 USA. {kayke, jwq123, tapo, bboots, siddh}@uw.edu



(a) Expert use teleoperation system to collect demonstrations.



(b) Picking up small objects with chopsticks

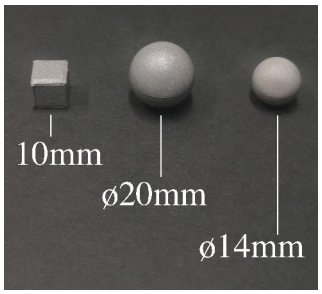
II. Problem

We built a 6-DOF robot equipped with a pair of chopsticks as its end-effector. The goal was to develop algorithms to control this robot with chopsticks to pick up challenging objects.

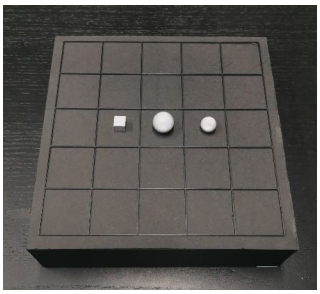
We have access to the tracked location of the objects, the robot's joint positions, and a kinematic model of the robot. However, the model for our inexpensive hardware is not highly accurate. On the one hand, the robot is assembled from parts and there are mounting inaccuracies. On the other hand, our robot arm's parts are not strictly rigid: at different configurations, the robot's joints and links can bend by a varying amount of small angles (e.g., 0.007 rad). With the best calibration to our knowledge, such errors still accumulate along the robot joints and result in errors ranging from 1 mm to 6 mm at the robot's end-effector. This implies that the difference between the model-estimated chopstick tip position and its actual position can be equal to the radius of the small objects that we want to pick up in our experiments.

Nevertheless, humans can use our imperfect hardware to accomplish very challenging grasping tasks. For example, human experts achieved more than 60% success rate teleoperating the robot to pick up a slippery glass ball [2]. This opens up the possibility to learn from human demonstrations.

We did experiments on three objects: a cube with 1 cm edge length, a ball with 2 cm diameter and another ball with 1.4 cm diameter, as shown in Fig. 2a. We define success as grasping the objects using chopsticks, lifting them above the workstation, and holding them in the air for 1 s. We evaluated



(a) Objects to pick up



(b) 5×5 grid for evaluation

the performance of each method on each object by counting the success rate over 25 trials. During evaluation, we divide the square workstation plate to 5×5 grid and place the object around the center of each grid to ensure a good coverage over the complete workspace, see Fig. 2b.

III. Data Collection

For each object, we collected 500 trajectories of an expert teleoperating the robot and picking up the object. During each trajectory, we initiated the robot around a *fixed* home configuration and placed the object at a *random* location across the workstation. All trajectories were collected by one expert user to minimize any multi-modal behavior that might interfere with classic imitation learning methods like behavior cloning. Note that we filtered out failed trajectories (when the chopsticks moved the object without lifting it up), and kept only the 500 successful trajectories.

Each trajectory recorded a series of triplets of the current end-effector pose, the object location, and a target end-effector pose. The target pose was generated from expert-held leader chopsticks. We used the current end-effector pose and the object location to represent the state and the target-end effector pose as the action. The expert demonstrations contain state-action pairs, denoted by (s^*, a^*).

On average, each trajectory lasted about 6 s. Our arm runs at a high-level control frequency of 100Hz (i.e., how often we update the target end-effector pose). To ensure similar update rates, we recorded the trajectory and tested our policy at 100Hz. For each object we collected about 300K state-action pairs.

IV. Methods

A. `Replay`: *Replaying demonstrations*

To test the *repeatability* of our hardware controller, we chose 25 demonstrations that picked up the object from each of the 5×5 grid, placed the object at *exactly the same locations* used during data collection and replayed the demonstrations to see if the robot could pick up the object.

B. `BC`: *Behavior cloning and surrogate loss*

Hoping to learn a generalized agent that can pick up the objects from any given location, we used behavior cloning (`BC`) to optimize a neural network policy π parameterized by θ. It takes the state as input and outputs the target end effector pose. The output from the neural network, $\pi(s^*)$, is an 8D vector containing 3D for XYZ-positions, 4D for quaternions, and 1D for the opening angle between chopsticks. Minimizing the difference between $\pi(s^*)$ and expert action a^* requires defining a surrogate loss function that transforms the 8D vector to the 1D loss. We divided the 8D vectors to position, rotation, and opening angle, and computed each loss using either mean squared error or rotation difference. We then used a weighted linear combination, denoted as $\mathbf{w}$:

$$\min_{\theta} \sum_{i} \left[\pi_\theta(s_i^*) - a_i^*\right] \cdot \mathbf{w} \tag{1}$$

where $\mathbf{w}$ is a tunable parameter that can assign different weights to position error, rotation error, and end-effector opening error. Note the choice of $\mathbf{w}$ and *where* to measure the position error was arbitrarily defined. Therefore, we merely optimized the surrogate loss as our objective. Implementation details and tuning of parameters are described in Sec. VII-B.

C. `KNN`: *K-nearest-neighbor*

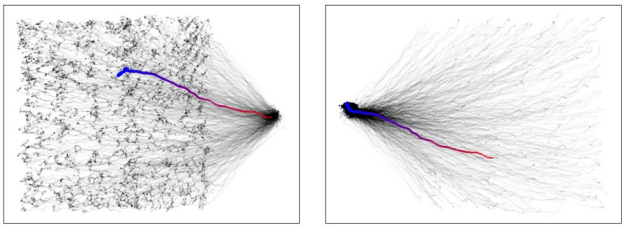
Testing the BC agent is subject to the covariate shift problem: the accumulated error in the agent's output may lead the agent to a previously-unseen state ($s \notin \{s^*\}$). Since the optimization of the neural network did not consider such unseen states as part of its loss, it might generate an unconstrained output. The covariate shift problem is particularly likely for our hardware with inaccurate models and controllers. We witnessed a few cases during testing the behavior cloning agent: the robot started shaking or moved to an opposite direction to intuition.

Non-parametric methods like KNN could alleviate such covariate shift problem. We can feed an input state to the KNN, query for neighbor states, find the corresponding actions for all neighbors and generate a weighted combination of the neighbors' actions as the target action. No matter how far the robot drifted away from the collected demonstrations and yielded a previously-unseen state, the output from the KNN would still be a combination of some actions from the training data. Intuitively, the output from the KNN is constrained.

We use the last 3 end-effector poses and the object location as the input to the KNN, denoted as $\bar{s}$ (in contrast to s^* for BC). We use KNN's output as the target action. KNN has some tunable parameters: the choice of k, the distance function for comparing two states $dist(\bar{s_1}, \bar{s_2})$ and the mix function that combines action labels from multiple neighbors $mix(\dots)$. We picked $k = 10$ and the distance function: $dist(\bar{s_1}, \bar{s_2}) = [\bar{s_1} - \bar{s_2}] \cdot \mathbf{w}'$, where $\mathbf{w}'$ allows us to tune the weight of each dimension. Denoting the distances to the nearest k neighbors as d, we use e^{-d}/Z as weights to linearly combine the actions labels where Z is the normalization factor. Tuning of parameters are described in Sec. VII-C.

D. `ObjC`: *The object-centric frame*

What coordinate frame to represent the data in, can change the distribution of the data. So far, we represented our collected demonstrations in the robot-centric frame, i.e., the robot base is the origin of the coordinate frame. Alternatively, we could represent the data in *the object-centric frame*. The object frame uses the location of the object as the origin point. By the design of our tracking system, this location is at the centroid position of the object. In the robot-centric frame, all our trajectories started from the same initial configuration



(a) Robot-centric frame. (b) Object-centric frame.

Fig. 3: Visualizing the end-effector positions under different coordinate frames for all demonstrations. Each black dot is the xyz-position of the end effector in one step. We highlight one trajectory that starts with red dots and ends in blue.

and reached towards different locations to pick up the object. In the object frame, all trajectories would appear to reach towards the origin point though each trajectory came from different initial poses, as shown in Fig. 3.

The transformation to the object-centric frame would result in more dense distribution of trajectories passing around the origin point, where the object sat. This decreases the probability of encountering unseen states when the chopsticks were around the object to pick up and therefore, has the potential to alleviate some covariate shift problem just before picking up. However, the transformation could lead to a higher chance of encountering unseen states when the robot just starts from its initial configuration and could aggravate the covariate shift problem at the beginning of the trajectory. To understand how the choice of frame might affect the agents' performance, we compared the performance of using object-centric frame (**ObjC**) versus robot-frame (default) for both `BC` and `KNN`.

E. `BCxKNN`*: Behavior cloning and KNN*

We hope to combine the reliability of KNN with behavior cloning. Specifically, we feed the input state first to the KNN and use the KNN's constrained output to be the input of our BC agent, optimizing:

$$\min_{\theta} \sum_{i} [\pi_\theta(\texttt{KNN}_{\mathbf{w}'}(\bar{s_i^*})) - a_i^*] \cdot \mathbf{w} \tag{2}$$

where $\mathbf{w}$ is a tunable parameter for weighing the loss, $\mathbf{w}'$ is a tunable parameter for `KNN`'s distance function and $\bar{s^*}$ is our input to the `KNN`. To match our `KNN` agent as described above, we used the last 3 end-effector poses and the object location as the input to the `KNN`, $\bar{s^*}$.

Since the output from `KNN` is directly fed to a neural network, we are free to specify this output, as long as we choose a mixture of labels associated with neighbor states. For now, we kept using the mixed actions from neighbors (same as the `KNN` method in Section IV-C). We reused all parameters for `KNN` $-$ `ObjC` and `BC` $-$ `ObjC`.

V. Preliminary Results

We tested all methods on picking up three objects: a tracked cube (easiest), a tracked ball with 20 mm diameter and another tracked ball with 14 mm diameter (hardest). For each method we conducted 25 trials, picking up the object from each cell of the 5×5 grid. The results are shown in Table. I and winners for each column are highlighted.

TABLE I: Success rates of imitation learning methods, evaluated over 25 trials, performance written in percentage.

Method	Cube	Ball⌀20 mm	Ball⌀14 mm	All
`Expert`	100	80	68	82.67
`Replay`	100	80	80	86.67
`BC`	84	16	12	37.33
`KNN`	64	28	8	33.33
`BC` $-$ `ObjC`	**92**	16	24	44.00
`KNN` $-$ `ObjC`	84	**64**	12	53.33
`BCxKNN` $-$ `ObjC`	64	**64**	**36**	**54.67**

We tested the `Expert` success rate in teleoperating the robot chopsticks to pick up the objects. Note that we kept only successful trajectories as our `Demonstrations`. We replayed 25 successful demonstrations and documented their success rates as `Replay`. We noted that `Replay` achieved 100% success rate on cube but only 80% success rate on balls. This is probably due to that picking up a cube is less sensitive to positional inaccuracy than picking up a ball, which required grasping at precisely the points across the diameter.

`BC` and `KNN` achieved comparable performance, so did the `BC` $-$ `ObjC` and `KNN` $-$ `ObjC`. Empirically, behavior cloning achieved higher success rate picking up the cube and the small ball whereas KNN achieved higher success rate picking up the big ball.

The object-centric frame agents (`ObjC`) performed better that robot-centric regardless of algorithms on picking up all objects. We then trained a `BCxKNN` $-$ `ObjC` agent, using the same parameters of our best `BC` $-$ `ObjC` and `KNN` $-$ `ObjC` agent. This agent performed the best on picking up the big and small ball but had trouble picking up the cube. During its roll-out, we witnessed that the agent could move towards the cube but occasionally had problems *closing the chopsticks*, keeping the same pose without moving.

It is worth mentioning that KNN agents were more shaky regardless of their performance and could fail to pick up the object as it suddenly stopped moving; BC agents could sometimes pick up the cube, but it occasionally moved to weird poses and hit the workstation, perhaps due to the covariate shift; and BCxKNN agents appeared to have the smoothest motion but occasionally stopped moving, just like the KNN agents.

VI. Discussions

We explored imitation learning methods to pick up small cube and ball objects from expert demonstrations. We observed that representing locations under the object-centric frame made both KNN and BC agents perform better. We combined BC and KNN agents and achieved better success rate picking up the balls but not for the cube.

Why did transformation to the object-centric frame make both KNN and BC agents perform better than using the robot-centric frame? One plausible reason could be that the object-centric frame increases the density of trajectories in the critical zone and makes both BC and KNN agents more

robust to small variations. Our task requires high precision around the object. The grasping success is highly sensitive to small perturbations when the agent is just about to pick up the object. In Fig. 3, we visualized the distribution of demonstrations in both the frames. Having a diverse and dense set of trajectories around the object (Fig. 3b) increases likelihood for the agent to have seen a similar training data point. We also noted that all the object-centric frame agents had an easier time picking up the object that was *closer* to the initial robot configuration, suggesting that `ObjC` agents could potentially suffer from more covariate shift if the agent needed to go through a longer trajectory to pick up the object.

Why did our `BCxKNN` agent perform the best for balls but not for the cube? One possible reason might be the cube demonstrations. Our expert had slightly different styles to pick up the cube and the balls. For example, they sometimes held the chopsticks still in the air before closing around the cube. This might explain some of our `KNN` agent's failures. To pick up the cube, `KNN` and `BCxKNN` agents would sometimes stop moving, when the chopsticks were already at the right position for pick up and the robot only needed to close the chopsticks. Perhaps `BCxKNN` inherited such problematic behavior from `KNN` (since we used the same parameters as that used in the `KNN` algorithm). We are experimenting with the output from `KNN` and its parameters to search for a fix.

Our tasks require high precision in manipulation. It puts a burden on both the controller (e.g. the exact placement of end effector) and on the perception stack (e.g. where exactly the ball is). The inability to pick up the ball during replaying of the demonstrations, even at the same locations as used during expert demonstrations, suggests the need to have the agent close the feedback loop and adjust to its perceived state. Unable to query the expert for such labels, we need to solicit this information from other sources. One possible way is to roll out the learner in the simulator, observe its deviated state, and generate a synthetic demonstration data that guides the learner to "correct" its behavior. We denote the start of a demonstration as $(s_0, a_0)(s_1, a_1)\ldots$, and let us assume that after executing a_0 from s_0, the agent reaches s_1'. We can generate some action a_1' which aims to bring the robot from s_1' back to s_2 and add (s_1', a_1') to the training data. One can use Model Predictive Control to generate a_1'. This would be an extension to the Data as a Demonstrator method [5] and could be a plausible direction for our future work.

VII. Appendix

A. Calibration improved performance

The default model and controller for our hardware was not highly accurate. The average end-effector position error was $10\,$mm. After careful calibration, the average EE error reduced to $4\,$mm. Initially, even carefully-tuned controller had low success rates for picking up cube and small ball during replay (90% and 15%). We implemented custom PID controller and gain-tuning to achieve 100% and 80%.

B. Implementation details for `BC`

We measured the position error at the chopsticks' tip, which ideally could be the same as the object's location after chopsticks picked up the object. To tune the surrogate loss parameter $\mathbf{w}$, we assigned an initial weight vector that put a heavy punishment on the x-y-z loss and open loss. Depending on the behavior of the agent during test time, we increased the x-y-z loss weight if the agent couldn't reach the object or it failed to grasp at the correct spot; we increased the open loss weight if the agent couldn't close the chopsticks at the right time; we increased the rotation loss weight if the agent rotated to unseen poses during data collection. We repeated the tuning process for 2 iterations and ended up assigning the following weights to x-y-z, rotation, and opening losses: $1.9, 0.01, 2.2$.

C. Implementation details for `KNN`

Our parameters $\mathbf{w}'$ for `KNN` distance function assigns a weight to the x-y-z, rotation, opening, and target components for an input state. We tuned them using a similar procedure to how we tuned $\mathbf{w}$ for `BC`. We ended up assigning the following weights to each component: $1.1, 0.1, 1, 4.25$. In additional to the parameterized distance function, we also experimented with transforming our data such that after the transformation, we could use the L2 norm as the distance function. We tried 1) standard normalization of each dimension independently and 2) running principal component analysis to extract orthogonal bases from data. Neither worked well for our application.

We used a different input to `KNN` from `BC`: `BC` accepted the current end effector pose and the object location, s^*, as an 11D input whereas `KNN` accepted the last 3 end effector poses and the current object location as a 27D input, denoted as $\bar{s}$. We wished to keep the input dimension for `BC` small so that the neural network training would require less data. We needed to use the last few end-effector poses to `KNN` because `KNN` had problem closing the chopsticks when we used only the current end effector pose. Denoting the last three end effector poses as p_t, p_{t-1}, p_{t-2} and the current object location as o_t, the input to `BC` would be $s^* = [p_t, o_t]$. We generate $\bar{s^*} = [p_t, \gamma p_{t-1}, \gamma p_{t-2}, o_t]$ where γ is a discount factor that we set to 0.8.

Acknowledgement

This work was (partially) funded by the National Institute of Health R01 (#R01EB019335), National Science Foundation CPS (#1544797), National Science Foundation NRI (#1637748), the Office of Naval Research, the RCTA, Amazon, and Honda Research Institute USA.

References

[1] H. Sakurai, T. Kanno, and K. Kawashima, "Thin-diameter chopsticks robot for laparoscopic surgery," in *ICRA*, 2016, pp. 4122–4127.

[2] L. Ke, A. Kamat, J. Wang, T. Bhattacharjee, C. Mavrogiannis, and S. S. Srinivasa, "Telemanipulation with chopsticks: Analyzing human factors in user demonstrations," in *IROS*. IEEE, 2020.

[3] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *NIPS*, 2016, pp. 4565–4573.

[4] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *AAAI*, 2018.

[5] A. Venkatraman, B. Boots, M. Hebert, and J. A. Bagnell, "Data as demonstrator with applications to system identification," in *ALR Workshop, NIPS*, 2014.