Leidi Zhao[1]
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, NJ 07102
e-mail: lz328@njit.edu

Lu Lu
Assistant Professor
Department of Mechanical and Industrial Engineering,
New Jersey Institute of Technology,
Newark, NJ 07102
e-mail: lulu@njit.edu

Cong Wang
Assistant Professor
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, NJ 07102
e-mail: wangcong@njit.edu

Data-Oriented State Space Discretization for Crowdsourced Robot Learning of Physical Skills

This work discusses a crowdsourced learning scheme for robot physical intelligence. Using a large amount of data from crowdsourced mentors, the scheme allows robots to synthesize new physical skills that are never demonstrated or only partially demonstrated without heavy re-training. The learning scheme features a data management method to sustainably manage continuously collected data and a growing knowledge library. The method is validated using a simulated challenge of solving a bottle puzzle. The learning scheme aims at realizing ubiquitous robot learning of physical skills and has the potential of automating many demanding tasks that are currently hard to robotize. [DOI: 10.1115/1.4047961]

Keywords: robot learning, robot physical intelligence, machine learning, human computation, crowdsourcing, data management

1 Introduction

Physical intelligence is of fundamental importance for robots to interact with the real world. Compared to cognitive intelligence for tasks such as language processing and image recognition, robot physical intelligence for tasks such as dexterous manipulation and dynamic mobility is still at an early stage. Methods for enabling robot physical intelligence generally sit in two categories. The first category is motion planning based on analytical physics models. Representative work includes the running MIT cheetah robot [1], the dynamic vision guided baseball [2], and so on. Despite the impressive motion capabilities, such methods require extensive case-specific engineering that relies heavily on ad hoc and complex models, which limit ubiquity.

Meanwhile, learning-based methods, such as learning from demonstration [3] and reinforcement learning [4] allow robots to acquire physical skills through mimicking a mentor or self-directed trying. Robot learning based on data-driven methods greatly reduces the reliance on analytical models derived from laws of physics. Nevertheless, certain limitations exist. Most state-of-the-art methods for robot learning physical skills follow a "policy search" framework, which instead of recording and replaying demonstrated/self-explored successful motion, aims at producing a control policy that can handle the variations when fulfilling the skills. Many of these control policies are in the form of parameter-heavy structures such as a deep neural network (DNN) (e.g., Ref. [5]). Such control policies require lengthy training of a massive amount of parameters, which hampers them from handling dynamic manipulation and realizing online real-time feedback control. The latter is especially necessary to open-loop unstable tasks such as in-hand manipulation. Another popular practice is to formulate control policies as combinations of a few base elements such as the "motion primitives" (e.g., Refs. [6,7]). Training such control policies requires significantly less computation, but the choice of the base elements and the design of the reward function require very task-specific engineering and jeopardize ubiquity. Recently, deep reinforcement learning has gained great attention. It provides better ubiquity by approximating the reward function using a DNN (e.g., Refs. [8,9]). Limitation of such approaches is brought by the lengthy training curves required by every new task. A representative example is the recent work by OpenAI on using a Shadow robot hand to learn in-hand rotation of a cube. Both the heavy training process and the need of task-specific engineering have been marked by experts [10].

One potential breakthrough comes from the concepts of Crowdsourcing and Human Computation. Crowdsourcing is summarized as "taking a job traditionally performed by a designated agent and outsourcing it to a … large group of people'" [11]. Human computation is defined as "utilizing human processing power to solve problems that computers cannot yet solve'" [12]. Great success has been achieved by Google Images by training AI systems with a massive amount of images reviewed and labeled by human participants from over the world. Other successes of crowdsourced human computation include translation, website ranking, and product reviews [13–16]. These applications, however, all belong to cognitive intelligence and have little to do with robot physical intelligence. Meanwhile, the few achievements related to robot physical skills mostly focus on designing the robot control user interface (e.g., MIT Baxter's Homunculus [17], Sarcos Robotics' Guardian GT [18]) or the search for similar or available solutions in the demonstration dataset (e.g., picking and grasping objects [19,20], machine operation [21], and assembly [22]). Major remaining challenges include the synthesis of new skills and the sustainable management of the constantly collected massive amount of data. Starting from the previous publication [23], this project makes contributions by addressing these two challenges.

2 Basic Framework

A two-step skill synthesis scheme has been developed based on state space discretization [23], which discretizes the state space into a finite number of cells. Figure 1 shows a functional block diagram of the scheme. A physical skill is considered as a controlled state transition through the cells in the discretized state space. A skill is specified by a given initial state and a desired final state, with optional intermediate key states. Demonstration trajectories can be obtained from human demonstration as well as the self-practice of the robot. Such trajectories can be constantly collected, segmented, and registered to the cells of the discretized state space. Instead of replaying the collected trajectories, new skills can be synthesized by first routing through trajectory segments registered to the cells and then applying statistical inference tools such as

[1]Corresponding author.

Manuscript received December 12, 2019; final manuscript received June 21, 2020; published online August 24, 2020. Assoc. Editor: Alexander Leonessa.

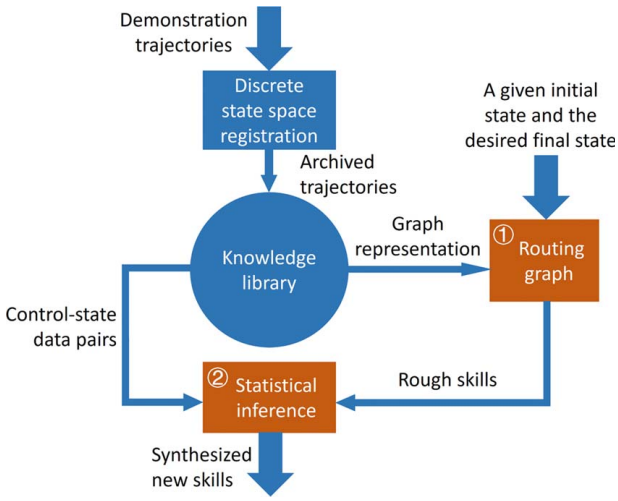

Fig. 1 Two-step skill synthesis

Gaussian Process Regression to handle the difference between the actual states and the archived data.

Figure 2 explains the scheme with a simple example. Consider a simple 4×4 full factorial discretization applied over a two-dimensional state space, with position and velocity as two state variables of a force controlled linearly moving block. Suppose three demonstration trajectories (in the form of timed data point sequences) are archived to the cells that they go through in the state space. A corresponding routing graph can be formed with the cells as nodes and the archived trajectories as edges. Routing algorithms as simple as the classic Dijkstra's algorithm can be applied to the routing graph and synthesize a control sequence to realize the skill. Our previous publication [23] presents several techniques to practically realize the proposed scheme, including

(1) The use of pseudo-random sequences to allocate a relatively small total number of cells while assuring inter-dimensional uniformity.
(2) The use of Gaussian process regression to connect trajectory segments and handle the difference between the actual states and the archived data.
(3) Merging collected trajectories to the achieved ones so that the size of the knowledge library is upper bounded and can be managed sustainably regardless of the continuous collection of new data.

However, efficiency of the state space discretization strategy using pseudo-random sequences turns out to be still a bottleneck.

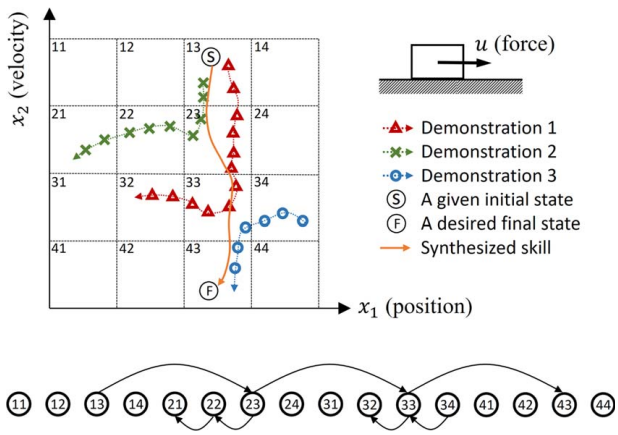

Fig. 2 A simple example of state space discretization and the corresponding routing graph

The collected trajectories usually distribute highly non-uniformly over the state space, leaving a large portion of cells empty. The issue is especially problematic when the state space is of high dimension, which is common for multi-degrees-of-freedom systems. The unnecessary computation caused by the inefficiency makes systems of high-dimensional state spaces intractable, as well as hampers running the skill synthesis in real-time. The latter is much needed for open-loop unstable systems such as the "bottle puzzle" introduced later in Sec. 5. In regard to this issue, this paper advances from Ref. [23] and presents a data-oriented discretization strategy.

3 Data-Oriented State Space Discretization

Instead of using pre-allocated cells, this paper presents a new method that dynamically allocates cells to discretize the state space based on the changing distribution of the collected data. Despite a changing total number of cells (which is fixed when using the previous method), the creation of new cells tends to converge to a trivial speed as data collection continues. A set of initial cells are allocated along the trajectory of a randomly selected demonstration (in the form a timed sequence of data points) every certain number of data points. For every new incoming demonstration trajectory, the same segmentation is conducted to produce a set of candidate new cells. The candidate new cells are then compared with the existing cells. If a candidate new cell relatively overlaps with certain existing cells, the candidate will be merged to the existing cells. Otherwise, it will be assigned an ID and join as a new cell. All allocated cells have the same size. Most computation of such a strategy is spent on finding similar cells, which can be formulated as a high-dimensional nearest neighbor search problem.

Unlike one-dimensional cases and the multi-dimensional cases with aligned data points where a complete sorting can be done easily, the exact solution of high-dimensional nearest neighbor search for a randomly distributed dataset requires examining the distances between the query point and all other points in the dataset. Due to the computational challenge, many approximating solutions have been developed and can be roughly grouped into four categories: tree-based methods, hashing-based methods, graph-based methods, and quantization-based methods. Some widely used algorithms include the KD-tree [24], HNSW [25], Locality Sensitive Hashing [26], and Product Quantization [27]. The search efficiency and accuracy of each method vary greatly depending on the specific application. Experimental results [28] have indicated that the graph-based methods tend to perform better than others in general.

A graph-based method is developed to provide nearest neighbor search for the proposed robot learning scheme. In addition to high search efficiency, high graph updating efficiency is also important since the cell allocation changes constantly. Meanwhile, a high precision (a.k.a., recall) is only needed when the query point is close to an existing cell in the knowledge library. For a query point far away from all existing cells in the library, a high precision is unnecessary. This is because even the exact nearest neighbor of the query point is found, a quality control signal can hardly be produced using statistical inference since the states are too different.

The proposed search method utilizes two direct graphs whose nodes represent the cells for state space discretization with edges connecting to certain close cells. Note the two graphs are for nearest neighbor search only (i.e., search graphs) and are different from the routing graph introduced earlier for skill synthesis. One of the two search graphs is called an In-dataset Graph (IDG), which locates either an existing cell if the candidate new cell happens to overlap with it (closer than a small threshold), or find a relatively close cell. In the first case, the candidate cell will be merged to the existing cell. Otherwise a second search graph, called diversified graph (DG) will be used to search further from the cell located in the IDG and find a possibly closer existing cell. Again, if eventually an overlapping cell is located, the

candidate cell will be merged. Otherwise, the candidate cell will be assigned an ID and join the discretization of the state space. In addition, every newly adopted cell will be registered to IDG and DG. IDG and DG have the same nodes representing all the allocated cells (indexed using their cell IDs), while their topologies are organized using "neighbor lists"—having a node in the neighbor list of another node means the former is connected to the latter with an edge directing to it.

Algorithm 1 explains the search over IDG. If the candidate cell is eventually adopted to the state space discretization (after the follow-up search using DG), it will be registered to the end of the neighbor list of its closest node in IDG (i.e., the node located by Algorithm 1). Building the IDG using such a strategy is also a (partial) sorting process. Despite the inclusion of new nodes over time, all previous search paths are preserved since the searches all follow the same order of the nodes in the neighbor lists. It also guarantees a monotonic path to locate an existing cell if the candidate cell overlaps with it (closer than a small threshold).

Algorithm 1 Nearest neighbor search in IDG

```
1  IDG_SEARCH(IDG, q, k);
2  Parameters: current IDG (including nodes {p_1, p_2, ...} and the neighbor
     list of each node), node representing a candidate new cell q, ID of the
     initial search node k;
3  Result: ID of the nearest node in the IDG k;
4  While i ≤ number_of_neighbors(p_k) do
5      n = ID of p_k's i'th neighbor;
6      if distance(q, p_n) < distance(q, p_k) then
7          k = n, i = 1;
8      else
9          i = i + 1;
10 Return k.
```

The node found from searching IDG (by Algorithm 1) is used as the initial node to search further for a closer neighbor in the diversified graph (DG) using a greedy algorithm (Algorithm 2). Figure 3 explains the construction of the neighbor lists in DG. Consider a base node p that has two nodes a and b in its neighbor list. A node q can only join the neighbor list of p if and only if $distance(q, p) < min(distance(q, a), distance(q, b))$—i.e., its distance to the base node has to be shorter than its distances to all the nodes previously joined that neighbor list. The node q' in Fig. 3 cannot join the neighbor list of p after a and b since $distance(p, q') > distance(a, q')$. Based on such a rule, pruning can be applied to truncate and limit the sizes of the neighbor lists below an upper bound.

Algorithm 2 Nearest neighbor search in DG

```
1  DG_SEARCH(DG, p, q, k, m);
2  Parameters: current DG (including nodes {p_1, p_2, ...} and the neighbor
     list of each node), node representing a candidate new cell q, ID of the
     initial search node k, search memory m, candidate pool S = ∅, flag
     f = 0, search path log L = ∅;
3  Result: ID of the nearest neighbor j, search path L;
4  Add p_k to S, add p_k to L;
5  while f < m do
6      f = the index of the first unchecked node in S;
7      Mark S(f) as checked;
8      for any neighbor p_n of p_k in DG do
9          if p_n ∉ L then
10             Add p_n to S, add p_n to L;
11     Sort the elements of S in ascending order according to their dis-
         tance to q;
12     if size(S) > m then
13         Truncate S to keep its first m elements;
14 Return j = ID of S(1), search path L.
```

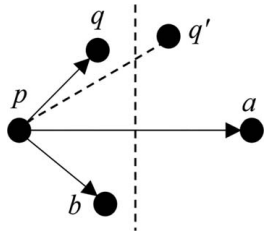

Fig. 3 Neighbor list construction for DG

A complete update of the DG with an incoming node requires a brute force procedure—first put all existing nodes in its neighbor list as well as try to put the incoming node in the neighbor lists of all existing nodes following the rule described earlier. Then, perform pruning. Such an action bears a computational complexity $O(Cn)$, where C is the average out-degree of a node and is affected by the order of the nodes in the neighbor lists, and n is the total number of nodes (cells). For a large set of nodes required by the robot learning scheme, such a method is extremely inefficient and impractical. Thus, instead of a complete DG, a reduced DG is constructed as described in Algorithm 3. Specifically, the search path for the nearest neighbor of an incoming node (representing a candidate new cell for state space discretization) is recorded when performing the greedy search over DG. If the cell is qualified to join the state space discretization, the new node will be added to the neighbor lists of (only) the nodes along the search path. Meanwhile, the nodes along the search path will be put in the new node's neighbor list. If any updated neighbor list's size is larger than a specified maximum out-degree R, graph diversification will be performed using Algorithm 4.

4 Computational Complexity

Based on the nearest neighbor search using IDG and DG, dynamic cell allocation can be realized efficiently for the state space discretization needed by the robot learning scheme. The same nearest neighbor search method is also used to locate the cell that contains a given state. The computational complexity of the search method is crucial to the implementation of the robot learning scheme. This is especially true for open-loop unstable physical skills, which require periodical online re-synthesis of the control sequence to achieve real-time feedback control.

Algorithm 3 Registering a new node

```
1  REG_NEW_NODE(IDG, DG, q, C);
2  Parameters: current IDG and DG (including nodes {p_1, p_2, ...} and their
     neighbor lists in IDG and DG), node representing a candidate new cell q,
     ID of the initial search node k, search memory m, search path log L,
     threshold t, maximum out-degree R;
3  Result: updated IDG and DG;
4  k = IDG_SEARCH(IDG, q, k);
5  if distance(q, p_k) < t/2 then
6      q is considered the same as p_k, no new cell is created;
7  else
8      [j, L] = DG_SEARCH(DG, q, k, m);
9      if distance(q, p_j) > t then
10         Add q to the neighbor list of p_j in IDG;
11         Add all nodes in L to the neighbor list of q in DG;
12         Add q to the neighbor lists of all nodes in L in DG;
13         for any node p_n ∈ L do
14             if size(p_n's neighbor list) > R then
15                 DG_DIV(DG, p_n, R);
16         if size(q's neighbor list) > R then
17             DG_DIV(DG, q, R);
18     else
19         No new cell is created.
```

Algorithm 4 Graph diversification for DG

```
DG_DIV(DG, q, R);
Parameters: current DG (including nodes {p1, p2, ...} and their neighbor lists), base node q, q's neighbor list before diversification C, q's neighbor list after diversification L = ∅, maximum out-degree R;
Result: the selected neighbor list L;
Sort the nodes in C in an ascending order of their distance to q;
for i = 1:size(C) do
    for j = 1:size(L)+1 do
        if j == size(L)+1 then
            Add C(i) to L;
        if distance(q, C(i)) > distance(q, L(j)) then
            Break.
        if size(L) > R then
            Break.
```

The search over IDG has monotonic paths, which give a complexity of $O(\log(n))$ [29], where n is the total number of nodes (cells). Updating IDG with a new node takes trivial additional action. The reduced DG preserves the monotonic search character of a complete DG to some extent, with certain variations depending on the search memory and the maximum out-degree. When the search memory and the maximum out-degree are fixed, simulation indicates an empirical average complexity of $O(K \log(n))$ for DG searches, where K is related to the search memory and the maximum out-degree. This indicates that there could be an amount at the order of $K \log(n)$ nodes whose neighbor lists need update. Additional computation is needed for diversification of DG, which is performed only when the size of a neighbor list is larger than the maximum out-degree R and thus requires an $O(R \log(R))$ computation for sorting and an $O(R)$ computation for pruning. The overall complexity related to DG is then bounded by $O(R^2 \log(R) K \log(n))$.

5 Validation

The proposed data-oriented state space discretization is largely motivated against the challenge of handling systems with state spaces of high dimensions. A "bottle puzzle" that has a 12-dimensional state space is designed to validate the upgraded robot learning scheme. Figure 4 explains the setup of the test. The task requires an L-shaped tool to be maneuvered to retrieve a ball from a bottle. The bottle is grounded and cannot be moved. Constrained by the shape and dimensions of the items, the only way to remove the ball from the bottle is to first place the ball on top of the thin part of the tool through a dynamic maneuver and then retrieve the ball while balancing the tool laterally so the ball does not fall off.

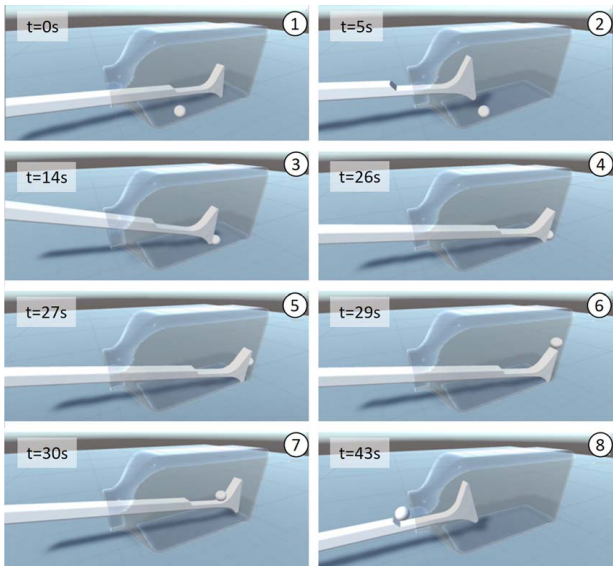

Fig. 4 Successful autonomous solving of the bottle puzzle

In addition to having a high-dimensional state space, the test also bears both "contemporary" and "sequential" characters. A task is considered contemporary if no steps are involved and the control law is relatively time-invariant (e.g., balancing an inverted pendulum). Such tasks often feature strong dynamic maneuvers. A task is considered sequential if nontrivial logical steps of a specific sequence are required while dynamical maneuvers are not necessarily involved (e.g., chess games). Tasks of either of the two characters have already been tackled separately by many learning methods. The unique feature of skill synthesis based on state space discretization of the proposed robot learning scheme is expected to be capable of handling tasks of both contemporary and sequential characters. The test is facilitated using a simulator. Simulation of the physics is based on Unity Physics, while data management and learning are deployed using MATLAB. A virtual reality (VR) interface based on HTC Vive system is used to allow human mentors to provide demonstrations intuitively. UDP connections are used to bridge among Unity, MATLAB, and the VR interface.

The 12 state variables of the system include the position and orientation of the tool and the position and velocity of the ball. The control variables are the six-axis force/torque applied to the tool. Ten non-expert participants provided 50 demonstrations by controlling the L-shaped tool using the VR interface. Their control signals and the response signals of the tool and the ball are recorded as the demonstration data. A total of less than 8000 cells are allocated by the proposed method. As a comparison, the previous method introduced in Ref. [23] using pseudo-random sequences can hardly provide any satisfactory skill synthesis for this test even with more than 10^{20} cells (because of the high dimension of the state space).

Due to the tricky physics involved, majority (43 out of 50) of the demonstrations is unsuccessful, while the few successful ones do not cover all possible situations. In particular, no successful demonstration has been collected with the ball initially beneath the tool, which is set to be the case in the autonomous operation. This is to challenge the capability of the learning scheme to synthesize new skills and handle unexpected situations that are never demonstrated or only partly demonstrated. More demonstrations covering larger areas in the state space would certainly improve the skill synthesis, similar to the quantitative analysis on the success rate of synthesis discussed in Ref. [23].

Figure 4 shows the screen shots of a successful autonomous solving of the bottle puzzle. Due to the complex contacts involved and the open-loop unstable nature of the maneuvers, the ball's response deviates from the expected trajectories easily without real-time feedback control. Thus, instead of blindly running one synthesized control sequence from the beginning to the end, real-time skill re-synthesis is conducted periodically as the task goes on. Such an online re-synthesis strategy adjusts the control based on the actual real-time response of the system. Its successful implementation also approves the acceptable efficiency of the proposed algorithms for real-time computing.

6 Conclusions and Future Work

This paper presents a new milestone of our work on crowd-sourced robot learning. In order to handle systems with high-dimensional state spaces, a data-oriented state space discretization method is developed. The method dynamically allocates cells to discretize the state space based on the changing distribution of the collected data. A dual-graph nearest neighbor search algorithm is developed to realize computationally efficient dynamic cell allocation. The method is validated using a simulated bottle puzzle that

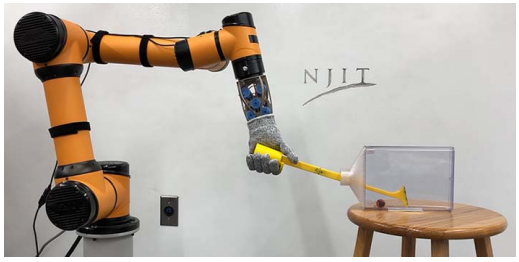

Fig. 5 Setup of the upcoming physical tests

features both contemporary and sequential characters. Test results have demonstrated the capability of the method to

(1) synthesize new skills,
(2) handle a system of a high-dimensional state space, and
(3) provide efficient online skill re-synthesis.

Advancing further from the simulation, a physical test (Fig. 5) is being setup and will be discussed in future publications.

References

[1] Seok, S., Wang, A., Chuah, M. Y. M., Hyun, D. J., Lee, J., Otten, D. M., Lang, J. H., and Kim, S., 2015, "Design Principles for Energy-Efficient Legged Locomotion and Implementation on the Mit Cheetah Robot," IEEE/ASME Trans. Mechatron., **20**(3), pp. 1117–1129.

[2] Senoo, T., Namiki, A., and Ishikawa, M., 2004, "High-Speed Batting Using a Multi-Jointed Manipulator," IEEE International Conference on Robotics and Automation, New Orleans, LA, Vol. 2, pp. 1191–1196.

[3] Argall, B. D., Chernova, S., Veloso, M., and Browning, B., 2009, "A Survey of Robot Learning From Demonstration," Robot. Auton. Syst., **57**(5), pp. 469–483.

[4] Sutton, R. S., and Barto, A. G., 2018, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

[5] Huang, S. H., Pan, J., Mulcaire, G., and Abbeel, P., 2015, "Leveraging Appearance Priors in Non-Rigid Registration, with Application to Manipulation of Deformable Objects," IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany, pp. 878–885.

[6] Levine, S., Wagener, N., and Abbeel, P., 2015, "Learning Contact-Rich Manipulation Skills with Guided Policy Search," IEEE International Conference on Robotics and Automation, Seattle, WA, pp. 156–163.

[7] Fu, J., Levine, S., and Abbeel, P., 2016, "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors," IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon, South Korea, pp. 4019–4026.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and Petersen, S., 2015, "Human-Level Control Through Deep Reinforcement Learning," Nature, **518**(7540), p. 529.

[9] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K., 2016, "Asynchronous Methods for Deep Reinforcement Learning," International Conference on Machine Learning, New York, NY, pp. 1928–1937.

[10] Knight, W., 2018, "An AI-Driven Robot Hand Spent a Hundred Years Teaching Itself to Rotate a Cube," https://www.technologyreview.com/s/611724

[11] Howe, J., 2009, *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*, Crown Business, New York.

[12] von Ahn, L.,2005, "Human Computation," Doctoral Dissertation, Carnegie Mellon University, Pittsburgh, PA.

[13] Quinn, A. J., and Bederson, B. B., 2011, "Human Computation: A Survey and Taxonomy of a Growing Field," The SIGCHI Conference on Human Factors in Computing Systems, Vancouver, BC, Canada, ACM, pp. 1403–1412.

[14] Doan, A., Ramakrishnan, R., and Halevy, A. Y., 2011, "Crowdsourcing Systems on the World-Wide Web," Communi. ACM, **54**(4), pp. 86–96.

[15] Geiger, D., Seedorf, S., Schulze, T., Nickerson, R. C., and Schader, M., 2011, "Managing the Crowd: Towards a Taxonomy of Crowdsourcing Processes," Americas Conference on Information Systems, Detroit, MI.

[16] Little, G., Chilton, L. B., Goldman, M., and Miller, R. C., 2010, "Turkit: Human Computation Algorithms on Mechanical Turk," The 23rd Annual ACM Symposium on User Interface Software and Technology, New York, pp. 57–66.

[17] Lipton, J. I., Fay, A. J., and Rus, D., 2018, "Baxter's Homunculus: Virtual Reality Spaces for Teleoperation in Manufacturing," IEEE Robot. Autom. Lett., **3**(1), pp. 179–186.

[18] Wolff, B.,Oct., 2017, "Why Human-Controlled, Force-Multiplying Robots Are the Future of Work on Earth," IEEE Spectrum.

[19] Sorokin, A., Berenson, D., Srinivasa, S. S., and Hebert, M., 2010, "People Helping Robots Helping People: Crowdsourcing for Grasping Novel Objects," IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, pp. 2117–2122.

[20] Forbes, M., Chung, M., Cakmak, M., and Rao, R. P. N., 2014, "Robot Programming by Demonstration with Crowdsourced Action Fixes," The 2nd AAAI Conference on Human Computation and Crowdsourcing, Pittsburgh, PA.

[21] Sung, J., Jin, S. H., and Saxena, A., 2018, "Robobarista: Object Part Based Transfer of Manipulation Trajectories From Crowd-Sourcing in 3D Pointclouds," *Robotics Research. Springer Proceedings in Advanced Robotics*, Vol. 3, A. Bicchi, and W. Burgard, eds., Springer, Cham, Switzerland, pp. 701–720.

[22] Chung, M. J. Y., Forbes, M., Cakmak, M., and Rao, R. P. N., 2014, "Accelerating Imitation Learning Through Crowdsourcing," IEEE International Conference on Robotics and Automation, Hong Kong, China, pp. 4777–4784.

[23] Zhao, L., Lawhorn, R., Wang, C., Lu, L., and Ouyang, B., 2019, "Synthesis of Robot Hand Skills Powered by Crowdsourced Learning," IEEE International Conference on Mechatronics, Ilmenau, Germany, pp. 211–216.

[24] Bentley, J. L., 1975, "Multidimensional Binary Search Trees Used for Associative Searching," Commun. ACM, **18**(9), pp. 509–517.

[25] Malkov, Y. A., and Yashunin, D. A., 2020, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Trans. Pattern Anal. Mach. Intel., **42**(4), pp. 824–836.

[26] Gionis, A., Indyk, P., and Motwani, R., 1999, "Similarity Search in High Dimensions Via Hashing," The 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK, pp. 518–529.

[27] Jegou, H., Douze, M., and Schmid, C., 2011, "Product Quantization for Nearest Neighbor Search," IEEE Trans. Pattern Anal. Mach. Intel., **33**(1), pp. 117–128.

[28] Benchmarks of ANN: https://github.com/erikbern/ann-benchmarks

[29] Fu, C., Xiang, C., Wang, C., and Cai, D., 2019, "Fast Approximate Nearest Neighbor Search With the Navigating Spreading-Out Graph," Proc. VLDB Endowment, **12**(5), pp. 461–474.