International Journal of Intelligent Robotics and Applications (2020) 4:390–402
https://doi.org/10.1007/s41315-020-00152-1

REGULAR PAPER

Handling crowdsourced data using state space discretization for robot learning and synthesizing physical skills

Leidi Zhao[1] · Lu Lu[1] · Cong Wang[1]

Received: 14 July 2020 / Accepted: 19 October 2020 / Published online: 12 November 2020

Abstract
Intelligent physical skills are a fundamental element needed by robots to interact with the real world. Instead of learning from individual sources in single cases, continuous robot learning from crowdsourced mentors over long terms provides a practical path towards realizing ubiquitous robot physical intelligence. The mentors can be human drivers that teleoperate robots when their intelligence is not yet enough for acting autonomously. A large amount of sensorimotor data can be obtained constantly from a group of teleoperators, and processed by machine learning to continuously generate and improve the autonomous physical skills of robots. This paper presents a learning method that utilizes state space discretization to sustainably manage constantly collected data and synthesize autonomous robot skills. Two types of state space discretization have been proposed. Their advantages and limits are examined and compared. Simulation and physical tests of two object manipulation challenges are conducted to examine the proposed learning method. The capability of handling system uncertainty, sustainably managing high-dimensional state spaces, as well as synthesizing new skills or ones that have only been partly demonstrated are validated. The work is expected to provide a long-term and big-scale measure to produce advanced robot physical intelligence.

Keywords Robot learning · Physical intelligence · Robotic manipulation · Crowdsourcing

1 Introduction

Physical intelligence such as dexterous manipulation and dynamic mobility is crucial for robots to interact with the real world. Methods for realizing robot physical intelligence generally sit in two types. One relies on motion planning based on analytical models derived from laws of physics. Representative work includes the dynamic re-grasping robot hand by the University of Tokyo Furukawa et al. (2006) and the bipedal robot Cassie by Agility Robotics Ackerman (2017). Despite the rigorous guarantee of stability and the full utilization of mechanical potentials, the extensive case-specific engineering and complex ad hoc analytical models required by such methods hamper their ubiquity.

Thanks to the advance of artificial intelligence, robot learning methods such as learning from demonstration (LfD) Argall et al. (2009) and reinforcement learning (RL) Sutton and Barto (2018) have successfully allowed robots to acquire physical skills with less reliance on analytical models. Most of such methods adopt a “policy search” framework Deisenroth et al. (2013) and infer meta-parameter-based control policies using data collected from mentor demonstrations or autonomous practice. The learned control policies are often in the form of deep neural networks or other parameter-heavy structures that require heavy training, which limits their usage to mostly semi-static operations. A representative work is a recent project by OpenAI using deep reinforcement learning to realize robot in-hand manipulation. Both the intense training process and task-specific engineering have been noted by experts Knight et al. (2018). Improved computing efficiency can be achieved by formulating control policies as combinations of a select few base elements (e.g., motion primitives Ijspeert et al. (2003)), which allows significantly less

✉ Cong Wang
wangcong@njit.edu

Leidi Zhao
lz328@njit.edu

Lu Lu
lulu@njit.edu

[1] New Jersey Institute of Technology, Electrical and Computer Engineering, 323 Martin Luther King Blvd, Newark, NJ 07102, USA

training. Nevertheless, designing the base elements and the reward function takes highly task-specific engineering and jeopardizes ubiquity.

In addition to ubiquity and computation, another consideration is over the data source of skill acquisition. For complex physical skills, especially those involving dynamic maneuvers, it is impractical to rely solely on reward-driven self-practice (i.e., basic reinforcement learning) without advisory from mentors Johnson and Hasher (1987). Over the past decade, large-scale datasets obtained from a group of mentors (crowdsourced) have greatly accelerated the advancement of artificial intelligence Howe (2009). Together with crowdsourcing, human computation von Ahn (2005) has become an increasingly popular way to provide mentorship to learning agents. The concept is particularly popular in the field of cognitive machine intelligence such as language processing and image recognition. A classic example is Google Images, whose AI system is trained using a massive amount of sample images reviewed and labeled by human participants from over the world. Other successes include translation, merchandise review, and medical diagnosis Quinn and Bederson (2011), Doan et al. (2011), Geiger et al. (2011), Little et al. (2010).

Despite its promising potentials, replicating the success of crowdsourced human computation in robot physical intelligence has not been explored much. Challenge first comes from building intuitive control interfaces. Pioneering work regarding this topic includes MIT's Homunculus project Lipton et al. (2018) that develops a virtual reality remote robot control environment and Stanford's Roboturk project Mandlekar et al. (2018) that uses smartphones as remote robot controllers. Another challenge is on sustainable management and efficient utilization of the collected large datasets. Published projects that apply crowdsourced human computation to generate robot physical skills include grasping novel objects Sorokin et al. (2010), Forbes et al. (2014) and assembly Chung et al. (2014). So far, these projects are limited to realizing semi-static operations by searching for similar or available solutions in the dataset.

Our research on crowdsourced robot learning of physical skills develops a unique learning scheme that brings contributions with the following capabilities:

1. Allowing robots to learn physical skills from a group of human mentors constantly through demonstrations;
2. Sustainably managing a large amount of data that is constantly collected from the crowdsourced mentors;
3. Synthesizing dynamic physical skills that have been never or only partly demonstrated, without the need of heavy training/re-training; and
4. Efficient handling of high-dimensional large datasets.

2 Basic framework: data management and skill synthesis based on state space discretization

The proposed learning scheme treats physical skills as controlled state transitions in the state space, and uses state space discretization to manage a large and growing amount of data collected from groups of mentors. The scheme features a two-step process (Fig. 1) to synthesize autonomous robot physical skills. Demonstration trajectories provided by mentors are segmented and registered to the "cells" of the discretized state space of the system. A graph representation of the registered dataset is built using the cells as nodes and the inter-cell trajectory segments as edges. In the skill synthesis process, a physical skill is first "roughly" composed as a series of state transitions through a number of adjacent cells in the state space, which correspond to a route in the graph representation. Then, statistical inference is used to generate the control signals from the control-state data pairs of the trajectory segments archived to the cells along the route.

An elementary example explaining the framework is shown in Fig. 2. Consider a simple motion system that has only two state variables - position x_1 and velocity x_2 (e.g., a force-controlled linear moving block). Assume a simple state space discretization of a 4×4 full factorial design is used, with three demonstration trajectories archived. Consider a skill defined by a start state in cell $\textcircled{13}$ and a final state in cell $\textcircled{43}$. Using the segments of the demonstrated trajectories, a rough skill can be composed by routing through the graph representation of the dataset: $\textcircled{13} \rightarrow \textcircled{23} \rightarrow \textcircled{33} \rightarrow \textcircled{43}$. Note that the start and final states do not sit exactly on any demonstration trajectories, and the trajectory segments do not exactly intersect in the switching cells. In order to generate the final control signal, the second step of skill synthesis uses statistical

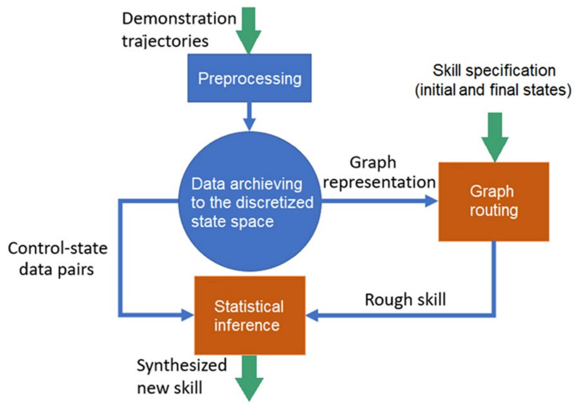

Fig. 1 Two-step skill synthesis

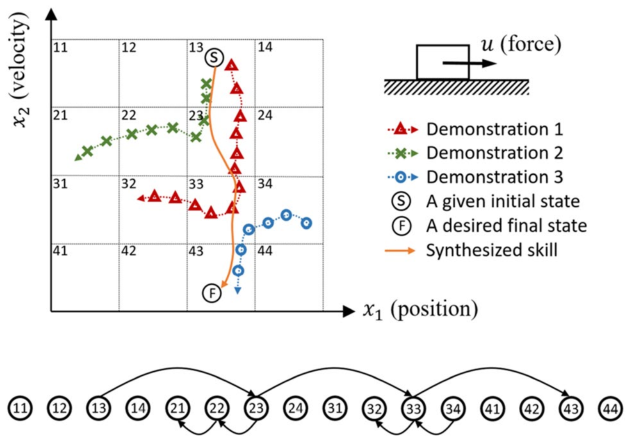

Fig. 2 An elementary example Zhao et al. (2020)

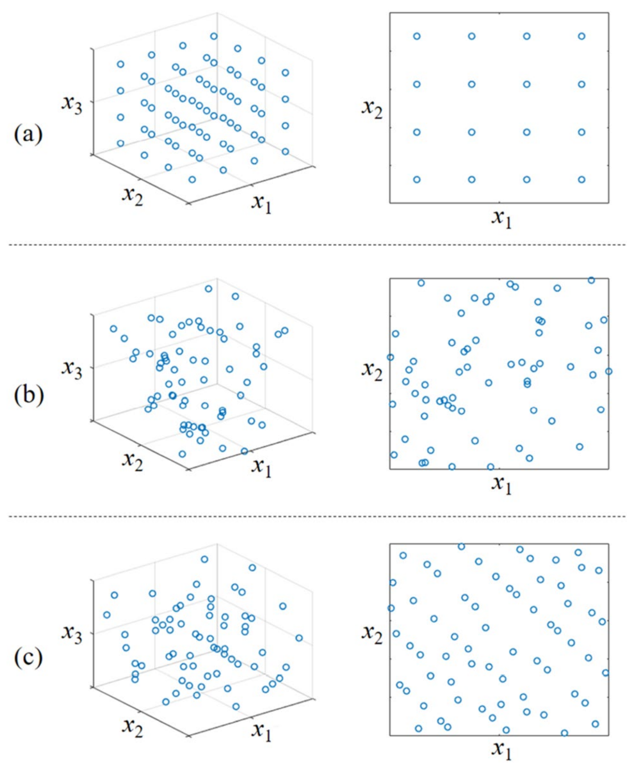

Fig. 3 3D state space discretization using 64 points of different distributions and their projections to a 2D subspace - (**a**) full-factorial design, (**b**) uniform random numbers, (**c**) low-discrepancy pseudo-random sequence

inference such as Gaussian Process Regression to handle the differences.

Based on the basic framework, advanced techniques are needed to make the learning scheme applicable to real systems. In particular, state space discretization is a key element. Robot physical skills often involve many state variables and have high-dimensional state spaces. The simple full-factorial design used in the elementary example requires enormous number of cells for high-dimensional state spaces and takes impractical amount of computing resource. Another major concern is over the sustainability of the discretization as the dataset grows. For crowdsourced learning over long terms, the amount of collected data grows rapidly and constantly. In this work, two types of state space discretization have been developed, and are presented in Sects. 3 and 6, with validation tests discussed in Sects. 5 and 7, respectively. Section 4 explains the graph routing and statistical inference techniques used in skill synthesis.

3 State space discretization using pseudo-random sequences

In the proposed robot learning scheme, physical skills are defined using state transitions in the state space. Some state space discretization techniques actually exist for reinforcement learning (RL) Lee and Lau (2004), Uther and Veloso (1998). RL applications usually tackle a single control task and these techniques discretize the state space according to the (one) control policy for the task. Meanwhile, our proposed learning scheme aims at treating various control tasks ubiquitously, which allows different control actions to correspond to the same state. In terms of synthesizing skills of a greater variety, it is desirable to have data populate the state space thoroughly, which in turn requires a uniform and thorough discretization of the state space. Meanwhile, for the sake of computational affordability, the state space should be discretized with as few cells as possible.

The total number of cells of a full factorial design (as used in the elementary example earlier) increases exponentially as the dimension of state space grows, which leads to excessive load of data handling. In addition, although a full factorial design provides a seemingly good uniformity, the coverage is actually quite inefficient. As illustrated in Fig. 3a, consider a three-dimensional state space discretized by a four-level full factorial design. The total number of cells is $4^3 = 64$. However, the cells' projection to a lower-dimensional subspace suffers severe overlapping. In this case, a two-dimensional subspace has only $4^2 = 16$ cells and a one-dimensional subspace has only 4 cells.

An intuitive better choice is a random discretization of a uniform distribution. The randomness minimizes the overlapping in the cells's projection to lower-dimensional subspaces. However, governed by the law of large number for random variables, a good uniformity can hardly appear if only a limited numbers of cells can be computationally afforded (Fig. 3b). Inspired by Wang et al. (2014), a low-discrepancy pseudo-random sequence is used to discretize the state space. Compared to the full factorial design, low-discrepancy pseudo-random sequences ensure the uniformity through all lower-dimensional subspaces. Low-discrepancy pseudo-random sequences provide the same excellent uniformity of random distributions, but are actually deterministic instead of random. Even when the available computing resource only allows a small amount of cells, a discretization created using a low-discrepancy pseudo-random sequence still provides high uniformity over subspaces of lower-dimensions (Fig. 3c). In other words, it gives the most uniform discretization that can be achieved with a certain computing resource.

In our work, the Sobol sequence is adopted to discretize the state space. It is a widely used low-discrepancy pseudo-random sequence Niederreiter (1988). The sequence is generated using the so-called direction numbers $v_{i,j} = m_{i,j}/2^i$, where $m_{i,j}$ is an odd integer from a specific recurrence sequence. The component of the k'th point in the Sobol sequence on the j'th dimension is given as $s_j^{(k)} = b_1 v_{1,j} \oplus b_2 v_{2,j} \oplus b_3 v_{3,j} \oplus \cdots$, where b_i is the i'th bit of k's binary code, and $\oplus$ is the bit-by-bit exclusive-or operator. More details can be found in Joe and Kuo (2003).

Most operations in the proposed learning scheme require either registering or locating a state at a cell in the discretized state space, which is in essence a nearest neighbor search problem. Unlike the full factorial design, which has an aligned distribution and is naturally sorted, a pseudo-random sequence such as the Sobol sequence follows a sophisticated distribution. Rather than using a brutal force nearest neighbor search, an efficient algorithm (Algorithm 1) is developed by taking advantage of the property of a Sobol sequence - that its uniformity is fully preserved in its projection to subspaces of any lower dimensions. As an example, consider using a 100-point two-dimensional Sobol sequence to discretize a two-dimensional state space within the range of [0, 0] to [1, 1]. Suppose the cell nearest to the state $x = [0.2, 0.7]$ needs to be located. First, the sequence is sorted according to the values of the first state variable. Thanks to the uniformity property of Sobol sequences, the query state should sit close to the points around the $100 \times 0.2 = 20$th position in the sorted sequence. Then, the nearest point (the center of a cell) to the query state can be located quickly by searching over a small nearby range (e.g., the 17th–23rd points in the sorted Sobol sequence). Figure 4 shows an example of a trajectory registered to a discrete state space formed using a Sobol sequence.

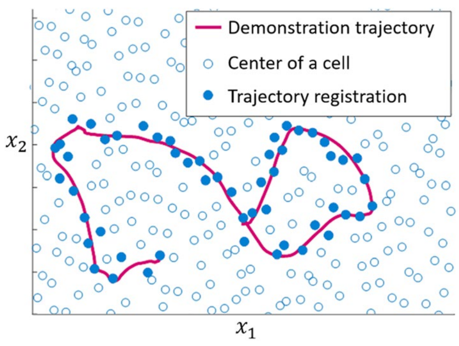

Fig. 4 A trajectory registered to a discrete state space

Algorithm 1: Nearest neighbor search in a state space discretized by a Sobol sequence

```
1  SOBOL_SEARCH(SOBOL, $q$);
2  Parameters: $N$-dimensional query point
     $q = [q_1, q_2, \cdots, q_N]$, a Sobol sequence of $n$ points
     $\{s^{(1)} \quad s^{(2)} \quad \cdots \quad s^{(n)}\}$ that discretizes the
     $N$-dimensional state space;
3  Results: the Sobol point closest to the query point;
4  Sort the Sobol sequence in ascending order according
     to the value of the first state variable $s_1^{(k)}$;
5  Calculate the search center as $q_1 \times n$. Specify an
     error boundary $e = [e_1, e_2, \cdots, e_N]$. The search
     interval is $[(q_1 - e_1) \times n, \quad (q_1 + e_1) \times n]$. for
     $i = (q_1 - e_1) \times n$ to $(q_1 + e_1) \times n$ do
6      Check if $s_j^{(i)} - q_j < e_j$ for $j = 2$ to $N$;
7      if only one $s^{(i)}$ satisfies the condition then
8          Return $s^{(i)}$.
9      else if more than one $s^{(i)}$ satisfy the condition
       then
10         Examine the Euclidean distance between the
             $q$ and the candidate $s^{(i)}$'s;
11         Return the $s^{(i)}$ of the shortest distance.
```

In terms of archiving new demonstration trajectories to the discretized state space, it is desirable to control the volume of the dataset so that it does not grow unlimitedly as new demonstrations being constantly collected. The trajectory segments (divided by the cells) of every new demonstration are examined by a merit criterion, such as time consumption, mechanical and electrical load of the robot. If a new segment is better than the archived one, it replaces the latter. Otherwise, it is discarded. Such a scheme avoids comparing a new demonstration to all previously collected demonstrations and keeps the total number of archived trajectory segments under nA, where A is the average number of adjacent neighbors of a cell, and is proportional to n the total number of cells of the discretized state space. n can be as large as the available computing resource allows.

4 Two-step skill synthesis

As explained in the simple example in Sect. 2 and Fig.2, the cells in the discretized state space and the demonstration trajectories archived to them can be represented by a directed graph for skill synthesis. Every node of the graph represents a cell in the discretized state space. A directed edge connecting two nodes represents a demonstration trajectory segment connecting the cells, indicating that a certain maneuver is available to drive the system from a state in the first cell to the state in the second. For each edge in the graph, a weight can be assigned to mark the merit of the corresponding maneuver (e.g., time, mechanical/electrical load, etc.). The first step of skill synthesis is to generate a "rough skill" in the form of a path of state transitions (edges) through the nodes in the graph representation. The procedure can be treated as a single-source shortest-paths problem and solved by the widely adopted Dijkstra's algorithm Dijkstra (1959).

The second step of skill synthesis is to generate the control signal based on the rough skill. Using the state-control data pairs $\{x, u\}$ of the archived demonstration data, statistical inference is used to infer the control action $u = u(x)$ that reacts to an actual state x in a synthesized skill. In our work, Gaussian Process Regression (GPR) is used. GPR has been widely used in robot learning control as a non-parametric Bayesian approach [e.g.,Nguyen-Tuong and Peters (2012) and Wang et al. (2015)]. A good introduction of GPR can be found in Rasmussen and Williams (2006). For each cell-to-cell state transition (an edge in the graph representation, a GPR model is trained to produce a segment of control sequence. The whole control sequence of an entire synthesized skill is in the form of a series of GPR models switched from one to another sequentially.

As explained at the end of Sect. 3, only the trajectory segments with the best performances are archived and used for training a GPR model. This strategy avoids training for every new demonstration. In addition, for each GPR model, only a (selected) segment of a whole trajectory is used for training, which reduces the computation load of training GPR models. While new demonstration trajectories can be collected unlimitedly, the total number of GPR models is bounded by nA, where n is the total number of cells of the discretized state space. A is the average number of adjacent neighbors of a cell and is proportional to n. Such an upper bound is important in terms of providing sustainability to crowdsourced learning, whose power comes from constantly collected demonstrations. Specifically, a major step in GPR training is the inversion of an $m \times m$ data covariance matrix Rasmussen and Williams (2006), where m is the number of training data points. A GPR model trained using a demonstration trajectory with $m = 1000$ data points would require the inversion of a matrix of the size 1000×1000. Assume that the trajectory has ten 100-point segments connecting through eleven adjacent cells in the state space, a GPR model trained from a single segment only requires operating a 100×100 matrix. Even if all segments of the trajectory end up being archived, all ten segments and their GPR models only need a fraction of the computation and memory that are required by a GPR model trained from the entire trajectory.

5 Validation I: the fidgeting test

The proposed learning scheme is first validated using a test of robot in-hand manipulation Zhao et al. (2019). A majority of human activities in various industries require dexterous in-hand manipulation skills such as adjusting the object posture when holding it in hand. Compared to grasping and picking, in-hand manipulation requires a lot more advanced skills, and is one of the bottlenecks that have been keeping many tasks from automation. The latest research on learning-based robot in-hand manipulation includes robotic in-hand rolling Van Hoof et al. (2015) and in-hand rotation of a cube Knight et al. (2018). These achievements are based on reinforcement learning and require a lengthy trial-and-error improvement process for every new skill. The proposed robot learning scheme is expected to provide an alternative solution of better ubiquity.

The test involves a simulated three-finger robot hand trying to arbitrarily fidget a small object (Fig. 5). The goal is to be able to manipulate the object from arbitrary initial position and orientation to arbitrary final ones. The robot learns from a group of human mentors using the proposed learning scheme. The mentors provide demonstrations by controlling the virtual robot hand using a virtual reality (VR) interface. Thanks to the popularity on the personal electronics market, VR technologies can now provide high quality motion sensing and real-time human-machine interaction. Compared to other demonstration interfaces such as lead-through teaching, VR-based interfaces are an immersed and intuitive way

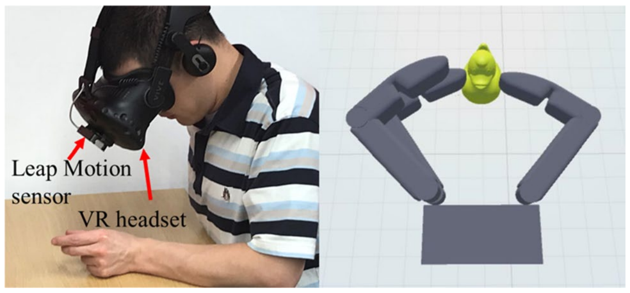

Fig. 5 VR interface and simulated robot hand Zhao et al. (2019)

to collect high-quality demonstrations Zhang et al. (2018). In our test, the HTC Vive headset is used to provide a visual interface. A Leap Motion sensor is attached to the headset to capture the finger motion. Simulation of the fidgeting physics is built using Unity.

The duck-shaped object in the test moves on a plane. The state variables include its two position coordinates and a single orientation coordinate. The state space is discretized by a three-dimensional Sobol sequence of 10^4 points. A total of 50 volunteers are invited to control the robot and manipulate the object using the VR interface. The mentors are encouraged to manipulate the object in as many ways as possible. Each mentor is asked to provide 10 demonstrations, which can be of different lengths and cover various areas and transitions in the state space. Raw demonstration trajectories are processed and archived as described in Sect. 3.

Along with the accumulation of new demonstrations, the robot hand is asked to synthesize and autonomously carry out manipulations of randomly picked initial and final states of the object. Depending on the distribution of the archived data in the state space, the synthesis may give more or less efficient skills and can sometimes result in a failure. The tests are studied in two ways. First, the success rate of skill synthesis is studied with respect to the amount of data archived. Without much surprise, as the archived data covers more cells of the working region in the discretized state space, the success rate of skill synthesis increases about proportionally (Fig. 6). The failed tests happen mainly because the picked initial state and/or final state have never been covered by any demonstration.

In addition, the performance of successfully synthesized new skills is examined. As the archived data reaches different levels of coverage in the state space, the robot is asked to repetitively synthesize a manipulation defined using the same initial and final state. Figure 7 shows the difference in the performance. The first manipulation is synthesized when only 100 demonstrations are collected, which covers 21% (of cells) of the working region in the state space. The second manipulation is synthesized when all 500 demonstrations are collected, which covers 67% (of cells) of the working region in the state space. Both manipulations can successfully manipulate the object from the specified initial position and orientation to the final ones, while the second one clearly shows a much cleaner transition with little unnecessary move.

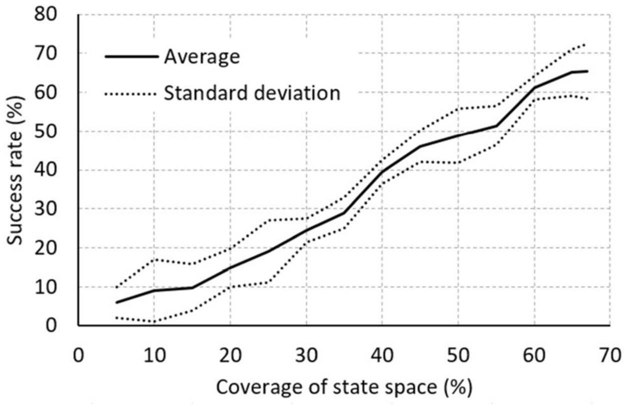

Fig. 6 Success rate of new skill synthesis Zhao et al. (2019)

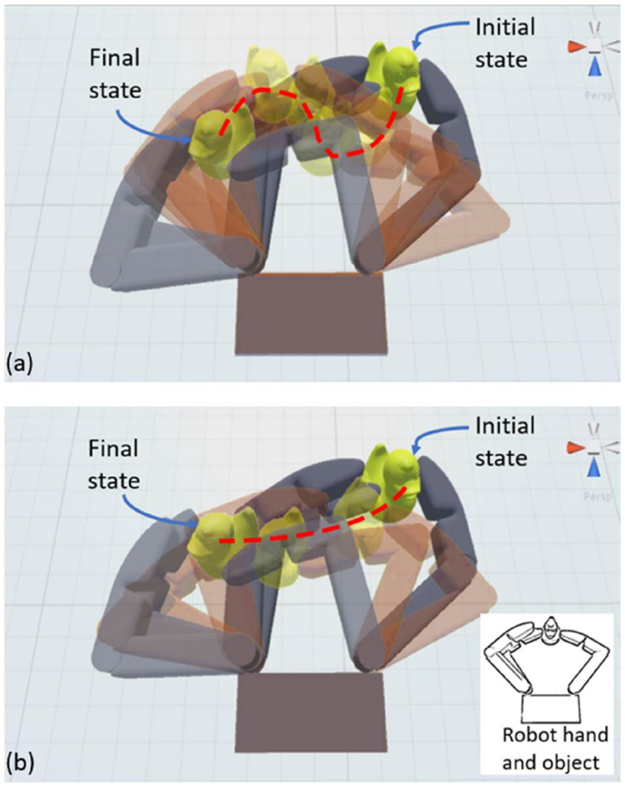

Fig. 7 A skill synthesized using a dataset that covers **a** 21% and **b** 67% of the working region in the state space Zhao et al. (2019)

6 Data-oriented state space discretization

6.1 Dynamic cell allocation

The state space discretization method based on pesudo-random sequences (Sect. 3) and the fidgeting test (Sect. 5) are motivated by the assumption that the robot system needs to visit all kinds of states over an "open" working region in the state space. Such an assumption is proper for some skills such as in-hand manipulation (e.g., fidgeting). Meanwhile, for many other skills, especially those involving the manipulation of multiple bodies, the working region of the robot in the state space usually spreads through highly irregular areas instead of a well-conditioned convex region (Fig. 8). This makes a large portion of cells being practically empty

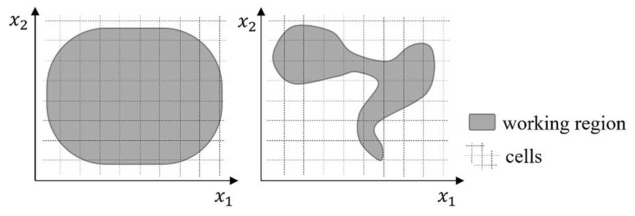

Fig. 8 Open v.s. irregular working regions in the state space

when a uniform discretization is used. The issue is especially problematic for systems of high-dimensional state spaces. Our latest work develops a data-oriented discretization strategy that dynamically adjusts the cell allocation according to the incoming data so as to achieve efficient data representation Zhao et al. (2020).

Rather than determining the state space discretization in advance, the data-oriented method allocates cells dynamically according to the incoming data. First, an initial discretization is generated by allocating cells along the trajectory of one of the first demonstrations every certain data points. As new data come in, the same way is used to generate candidate new cells. Cell allocation is updated based on the comparison between the candidate new cells and the existing cells. A candidate new cell is adopted to the state space discretization if the difference is greater than a certain threshold. Otherwise, it is merged to existing cells that cover (mostly) the same area in the state space. Compared to a uniform discretization, the data-oriented strategy makes sure that the cells can represent the collected data efficiently. A (minor) disadvantage of the new strategy is the increasing total number of cells, which is fixed when using a pre-allocated discretization. Nevertheless, the issue is bearable—after a certain amount of data is collected, the chance of needing to add new cells becomes trivial. This is because there will have been enough cells to cover the system's working region in the state space.

6.2 Nearest neighbor search for data-oriented discretization

As explained in Sect. 3, most operations in the proposed learning scheme require either registering or locating a state at a cell in the discretized state space, which is in essence a nearest neighbor search problem. In addition to the non-aligned distribution, the data-oriented discretization strategy does not have the uniformity property of the Sobol sequence. The exact solution of nearest neighbor search would require enumerating through the distances between the query point to all other points in the set. For large datasets obtained from crowdsourcing, such computation is not affordable. Search methods that give approximating solutions generally sit in four types which use search trees [e.g., KD-tree Bentley (1975)], hashing (e.g., Locality Sensitive Hashing Gionis et al. (1999)), graphs [e.g., HNSW Malkov and Yashunin (2018)], and quantization [e.g., Product Quantization Jegou et al. (2011)] respectively. According to experimental reports Benchmarks of ANN (2020), graph-based methods tend to perform better than others.

Graph-based nearest neighbor search indexes the datasets to graphs. In our application, the update efficiency of the search graphs is of particular importance since the state space discretization can change frequently which causes new nodes being added to the search graph constantly. A high recall (i.e., precision), on the other hand is needed only when a candidate new cell is close to the existing cells. For ones that are far away from the existing cells, it is not really meaningful to find the exact nearest neighbor. This is because even the exact nearest neighbor is located, no effective control signal can be generated (via statistical inference) since the query point is too different from any data.

Based on the above considerations, a dual-graph search method is developed to provide nearest neighbor search for the data-oriented state space discretization method. Two directed graphs are used by the proposed search method. The nodes of both graphs correspond to the cells of the state space discretization (indexed by cell ID's) and the edges represent the neighboring relationships of the cells. Note that these graphs are constructed for the nearest neighbor search needed by the data-oriented state space discretization, and are different from the routing graph discussed in Sects. 2 and 4 for skill synthesis.

The first graph is called an In-Dataset Graph (IDG). This graph utilizes a particular search-and-build strategy (explained later) that guarantees to return the query candidate cell itself if a same cell is already indexed in the graph (i.e., in-dataset query). In case that no existing cell overlaps with the candidate new cell, the IDG returns a relatively nearby existing cell. The result will then be used by the second graph, called a Diversified Graph (DG) as an initial point to search for a potentially closer neighbor. If the nearest neighbor found in the DG is too close (under a certain threshold) to the candidate cell, the candidate will be merged to it. Otherwise, the candidate new cell will be adopted to the state space discretization as well as indexed to the two search graphs. Note that the nodes in the IDG and DG are the same (both representing all the existing cells), whereas the edges of them are different. Both graphs are structured using "neighbor lists". In terms of topology, including a node in the neighbor list of another means the former is connected to the latter with an edge directing to it.

The two graphs are built and updated as the state space discretization changes when new data come in and candidate new cells are generated. Algorithm 2 explaines the search strategy for an IDG. The first node in an IDG is used as the initial base node. The distance between a query point and a base node is calculated as a baseline. Then, the nodes in the base node's neighbor list are checked in the order of when they were added to the graph. If a node in the list is found closer to the query point, it becomes a new base node. The search continues until all nodes in a neighbor list are checked and the base node is still the nearest to the query point. The last base node will be the returned search result from the IDG. In case of a candidate cell eventually getting adopted, the update strategy for an IDG is simply registering it as the (new) last node in the last searched neighbor list. A node newly added to the IDG has an empty neighbor list. Such a search-and-build strategy ensures that the adoption of a new node does not alter the search paths of all previously adopted nodes - because the nodes are always checked following the same orders in the neighbor lists. The feature guarantees the detection of the situation when a candidate new cell overlaps with an existing cell, which leads to a merge. In other cases, if a query point is close to an existing node in an IDG, the search also has a great chance to locate that node.

```
Algorithm 2: Search using an IDG
1  IDG_SEARCH(IDG, $q$, $k$);
2  Parameters: the latest IDG (nodes $\{p_1, p_2, ...\}$ and
     their neighbor lists), candidate new cell $q$, initial
     search node $p_k$;
3  Result: ID of the nearest node in IDG;
4  while $i \le number\_of\_neighbors(p_k)$ do
       n = ID of $p_k$'s $i$'th neighbor;
       if $distance(q, p_n) < distance(q, p_k)$ then
           $k = n$, $i = 1$;
       else
           $i = i + 1$;
10 Return $k$.
```

The result from searching over the IDG is passed to the DG as an initial node to search for a potentially closer neighbor of the candidate new cell. A greedy algorithm is used to search over a DG (Algorithm 3). Unlike searching over an IDG, all nodes in a neighbor list in a DG are checked, and the scale of neighbor lists has a major impact on the search efficiency. In order to limit the scale of the neighbor lists, the DG adopts a pruning strategy used in many graph-based search methods such as FANNG Harwood and Drummond (2016) and HNSW Malkov and Yashunin (2018). For a given node P_i and a distance metric d, a new node q can be adopted to P_i's neighbor list $L = \{L_1, L_2, \dots\}$ if and only if $d(q, P_i) < min(d(q, L_1), \dots, d(q, L_k))$, i.e., q is closer to P_i than it is to all nodes in the neighbor list of P_i. Refer to Fig. 9 and consider a base node p whose neighbor list has two nodes a and b. A query point q is adopted

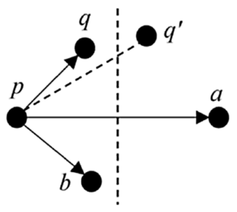

Fig. 9 Construction of neighbor lists for DG Zhao et al. (2020)

to the neighbor list of p as a new node if and only if $d(q, p) < min(d(q, a), d(q, b))$. The other query point q' cannot be adopted because $d(p, q') > d(a, q')$. Based on this rule, pruning is performed to truncate and bound the scale of the neighbor lists. Pruning has been known to promote the "diversification" of a graph Li et al. (2019), and gives the name of DG.

```
Algorithm 3: Search using a DG
1  DG_SEARCH(DG, $p$, $q$, $k$, $m$);
2  Parameters: the latest DG (nodes $\{p_1, p_2, ...\}$ and
     their neighbor lists), a candidate new cell $q$, initial
     search node $p_k$, search memory $m$, candidate pool
     $S = \emptyset$, flag $f = 0$, search path log $L = \emptyset$;
3  Result: ID of the nearest neighbor, search path;
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 $p_n \notin L$ then
10             Add $p_n$ to $S$;
11             Add $p_n$ to $L$;
12     Sort the elements of $S$ according to their
         distances to $q$ in ascending order ;
13     if $size(S) > m$ then
14         Truncate $S$ to keep its first $m$ elements;
15 Return $j$ = ID of $S(1)$, search path $L$.
```

Adding a new node to a complete DG requires enumeration - the new node should be first added to the DG with all other nodes in its neighbor list; then include the new node to all other nodes' neighbor lists; and finally perform pruning to all nodes' neighbor lists. The computational complexity for such an update procedure is O(nC), where n is the total number of nodes and C is the average out-degree (neighbor list size). For a large DG with a high update frequency required in our robot learning scheme, such a method is impractical. A reduced DG is constructed instead (Algorithm 4) to enable efficient DG updating. Although there is a sacrifice on the recall (precision), a strategy can be used to reduce the impact. Specifically, the search path of finding a candidate new cell's nearest neighbor using the DG is logged. When a new cell is added to the state space discretization and a corresponding node is added to the DG, its neighbor list will only include the nodes in the search path (instead of all other nodes). In the meantime, the new node will be adopted only to the neighbor lists of the nodes in the search path. If any neighbor list's size grows beyond the maximum allowed out-degree during the update, pruning will be performed using Algorithm 5.

Algorithm 4: Adopting a node to an IDG and a DG

```
1  REG_NEW_NODE(IDG, DG, $q$, $C$);
2  Parameters: the latest IDG and DG (nodes $\{p_1, p_2, ...\}$ and their neighbor lists), candidate new cell $q$, initial search node $p_k$, 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 virtually the same as $p_k$. The candidate cell is not adopted;
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 $p_n \in L$ 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         The candidate cell is not adopted.
```

In terms of computational efficiency, the monotonic search paths of an IDG give its search strategy a complexity of $O(log(n))$ Fu et al. (2019), where n is the total number of nodes. For the search strategy of a DG, our simulation shows an empirical average complexity of $O(Klog(n))$, where K is related to the search memory and the maximum out-degree. Such a complexity indicates that the reduced DG preserves the monotonic search character of a complete DG to some extent. The update strategy for an IDG takes trivial computation. The update strategy for a DG involves pruning, which is performed when the size of a neighbor list is greater than the maximum out-degree R. The total computational complexity for updating a DG is then bounded by $O(R^2log(R)Klog(n))$. Overall, the proposed dual-graph strategy gives a satisfactory balance between the recall and search/update efficiency.

Algorithm 5: Pruning for a DG

```
1  DG_DIV(DG, $q$, $R$);
2  Parameters: the latest DG (nodes $\{p_1, p_2, ...\}$ and their neighbor lists), base node $q$, its neighbor list $C$ before pruning, its neighbor list $L = \emptyset$ after pruning, maximum out-degree $R$;
3  Result: the selected neighbor list $L$;
4  Sort the neighbor list C in the ascending order of the distance to q;
5  for $i$ = 1:size(C) do
6      for $j$ = 1:size(L)+1 do
7          if $j$ == size(L)+1 then
8              Add $C(i)$ to $L$;
9          if distance(q, $C(i)$) > distance(q, $L(j)$) then
10             break.
11         if size(L)>$R$ then
12             break.
```

7 Validation II: the bottle puzzle

7.1 Simulation

The data-oriented state space discretization is motivated to handle systems of irregular working regions in high-dimensional state spaces. A "bottle puzzle" that has a 12-dimensional state space is conceived to challenge the method. Figure 10 shows the setup of the test. The goal of the bottle puzzle is to use an T-shaped tool to retrieve a ball from inside a bottle. The bottle is fixed and has a tight opening, which makes the only possible way to retrieve the ball being a special maneuver that involves first managing to place the

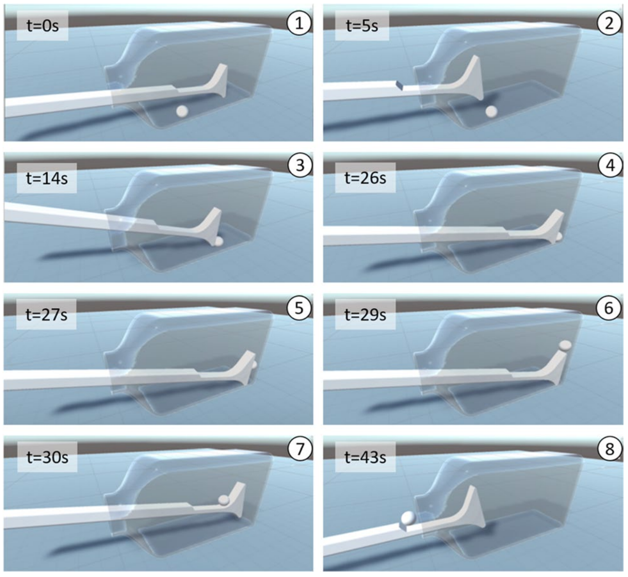

Fig. 10 Successful autonomous solving of the bottle puzzle (simulated) Zhao et al. (2020)

ball on top of the tool by bumping it against a corner. The 12 state variables of the system include the position and orientation of the T-shaped tool and the position and velocity of the ball. The control variables are the 6-axis force/torque applied to the tool. Other than having a high-dimensional state space, the bottle puzzle bears both "contemporary" and "sequential" characters. A skill is contemporary if no steps are involved and the control law is relatively time-invariant (e.g., balancing an inverted pendulum). Such skills often feature strong real-time dynamics. A skill is sequential if causal steps of specific sequences are needed whereas dynamical maneuvers are not necessarily involved (e.g., chess games). Skills of either of the two types have been well tackled separately by many learning methods. The unique skill synthesis function of the proposed learning scheme is expected to allow robots to handle skills of both contemporary and sequential characters.

The test is first conducted using a simulator Zhao et al. (2020). The physical dynamics is simulated using Unity Physics. Data management and robot learning are implemented using MATLAB. Similar to the simulation of the fidgeting test, the HTC Vive system is used to provide a virtual reality (VR) interface for human mentors to intuitively give demonstrations. The Unity Physics engine, MATLAB, and the VR interface are bridged using real-time UDP connections. Ten novice participants provided 50 demonstrations by controlling the (virtual) T-shaped tool using the HTC Vive hand-held controller. The control signals of the mentors and the response signals of the tool and the ball are collected as the demonstration data. Less than 8000 cells are created by the data-oriented discretization method. As a comparison, because of the high dimension of the state space, the method introduced in Sect. 3 using pseudo-random sequences can hardly provide satisfactory skill synthesis for this test even with more than 10^{20} cells.

Because of the challenging physical interactions involved, majority (43 out of 50) of the demonstrations are unsuccessful. Meanwhile, the few successful ones do not cover all possible situations. In particular, no successful demonstration has been collected with the ball initially under the tool, which is set to be the test case in the autonomous operation. Such a setting challenges the learning scheme on synthesizing new skills and reacting to unexpected situations that are never demonstrated or only partly demonstrated. Figure 13 shows a successful autonomous solving of the bottle puzzle. Due to the complicated physical contacts, the ball's response deviates from the expected trajectories easily if the control is applied in an open-loop manner (as opposed to real-time feedback control). Rather than blindly applying an entire synthesized control sequence from the beginning to the end, as the task goes on, it is necessary to conduct real-time skill re-synthesis to adjusts the control based on the actual system response. Such a strategy provides closed-loop stability to some extent. Its successful implementation also approves the real-time computing efficiency of the proposed algorithms.

7.2 Physical tests

Our latest development includes a physical test of the bottle puzzle. In the simulated test, the force and torque applied on the tool are directly controlled. The physical test uses an AUBO i5 6-axis robot arm and a bionic robot hand Zhao et al. (2018) to manipulate the tool (Fig. 11). Instead of rigidly mounting the tool to the robot, there is nontrivial looseness between the fingers of the robot hand and the tool. The slack grasping introduces additional degrees of freedom and causes significant passive movement of the tool with respect to the hand during the manipulation (Fig. 11). This factor makes the task much more difficult than in the simulation, and is kept as a new challenge to test the potential of the proposed method.

In the physical test, the state of the system, including the motion variables of the T-shaped tool and the ball, is sensed by a vision system consisted of a high frame rate camera as well as an embedded image acquisition and processing unit based on Nvidia's Jetson TX2 controller. Instead of shape recognition, color recognition is used to identify objects so as to shorten the sensing latency brought by image processing. In addition, instead of using simple differentiation to obtain velocity from position measurement,

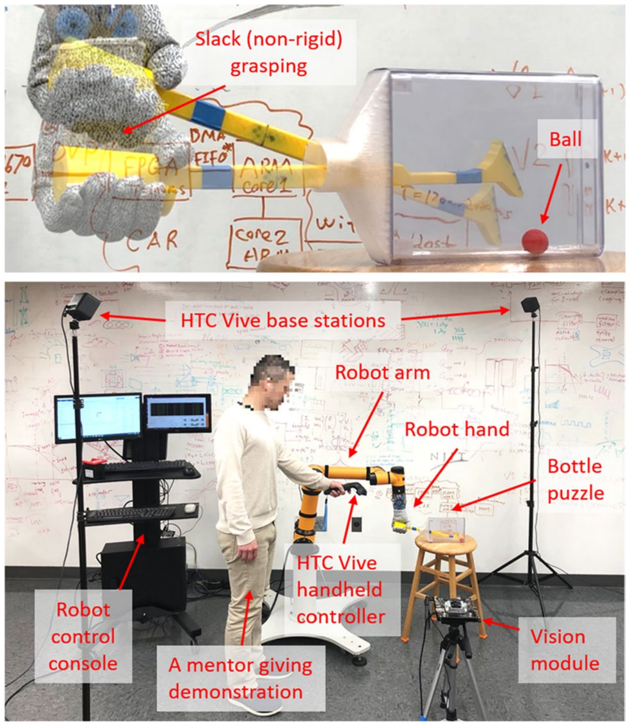

Fig. 11 Physical setup of the bottle puzzle test

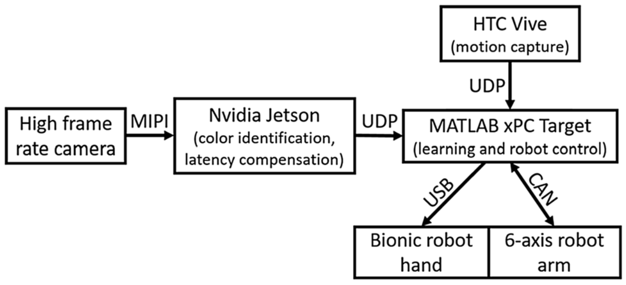

Fig. 12 Sensing and control deployment of the physical test

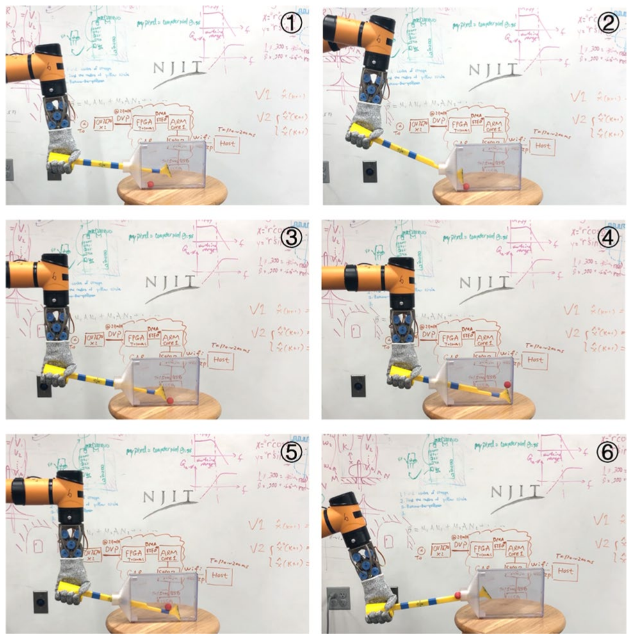

Fig. 13 Successful autonomous solving of the bottle puzzle (physical test)

predictive learning Wang et al. (2015) based on a dual-rate Kalman filter framework is used to enhance the accuracy of estimation and reduce negative effects brought by sensing latency. The vision acquisition and processing systems run on a standalone module which only sends the estimated state variables to the robot control console via UDP connections. The robot arm receives control signals via a Controller Area Network (CAN) bus. MATLAB is used to implement the learning scheme and generate control signals (Fig. 12).

Unlike in the simulation, much more uncertain factors exist in the physical test, especially those caused by the slack grasping of the robot hand (which is kept as a major challenge to test the proposed method). The friction variation and other surface irregularities also contribute to the system uncertainty. These factors cause the system state to deviate from the expected trajectories and significantly lower the repeatability of actions. As a countermeasure, online skill re-synthesis is used to provide a form of state feedback control. In terms of implementation, the measures described earlier to shorten the sensing latency as well as the real-time capability of communication protocols all contribute to ensure the effectiveness of online skill re-synthesis.

Demonstrations are provided by mentors through teleoperating the robot using the HTC Vive system. Again, a group of 10 novice human mentors provided 50 demonstrations. In order to make the operation as intuitive as possible, the motion capture interface is configured to map the motion of the handheld controller directly to the robot hand using velocity tracking. Similar to the simulated test, in order to validate the synthesis of new skills, the initial conditions (i.e., the tool-ball relative positions) used in the autonomous tests are designed to be at those where no successful demonstrations have been provided. Figure 13 shows a synthesized successful autonomous execution of the task. The test results also show that the proposed learning scheme can handle strong uncertainties (e.g., the uncertain caused by the slack grasping).

8 Conclusions

This paper summarizes our work on crowdsourced robot learning of physical skills. A unique learning scheme is developed to allow robots to learn from a group of mentors over long terms. State space discretization is used to sustainably manage constantly collected crowdsourced data and synthesize new skills. Two types of discretization methods are introduced. First, a discretization method based on pseudo random sequences is developed, featuring superior uniformity over the state space and a fixed upper bound of the archived data. Such a method is suitable for applications in which the robot system needs to visit all kinds of states over an open working region in the state space. For cases that the main working region of the robot in the state space spreads through highly irregular areas, a data-oriented state space discretization method is developed. The method aims particularly at handling systems with high-dimensional state spaces. Simulation and physical tests of a fidgeting challenge and a bottle puzzle are conducted. The test results validate the proposed learning scheme's capability on synthesizing new skills, sustainable management of crowdsourced data, efficient handling of high-order systems, and tolerating strong system uncertainties.

Funding This work is part of the project "CAREER: Enhancing Robot Physical Intelligence via Crowdsourced Surrogate Learning" funded by the National Science Foundation (Award number 1944069).

References

Ackerman, E.: Agility robotics introduces cassie, a dynamic and talented robot delivery ostrich. IEEE Spectrum **2017**, 28 (2017)

Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robot. Autonomous Syst. **57**(5), 469–483 (2009)

Benchmarks of ANN. https://github.com/erikbern/ann-benchmarks (2020). Accessed 16 Oct 2020

Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)

Chung, M.J.Y., Forbes, M., Cakmak, M., Rao, R.P.N.: Accelerating imitation learning through crowdsourcing. In: IEEE international conference on robotics and automation, pp. 4777–4784 (2014)

Deisenroth, M.P., Neumann, G., Peters, J., et al.: A survey on policy search for robotics. Found. Trends Robot. **2**(1–2), 1–142 (2013)

Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)

Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM **54**(4), 86–96 (2011)

Forbes, M., Chung, M., Cakmak, M., Rao, R.P.N.: Robot programming by demonstration with crowdsourced action fixes. In: The Second AAAI conference on human computation and crowdsourcing (2014)

Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endowment **12**(5), 461–474 (2019)

Furukawa, N., Namiki, A., Taku, S., Ishikawa, M.: Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In: IEEE international conference on robotics and automation (ICRA), pp. 181–187 (2006)

Geiger, D., Seedorf, S., Schulze, T., Nickerson, R.C., Schader, M.: Managing the crowd: towards a taxonomy of crowdsourcing processes. In: Americas conference on information systems (2011)

Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of The 25th international conference on very large data bases, pp. 518–529 (1999)

Harwood, B., Drummond, T.: Fanng: Fast approximate nearest neighbour graphs. In: IEEE conference on computer vision and pattern recognition (CVPR), pp. 5713–5722 (2016)

Howe, J.: Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business. Crown Business, New York (2009)

Ijspeert, A.J., Nakanishi, J., Schaal, S.: Learning attractor landscapes for learning motor primitives. Adv. Neural Inf. Process. Syst. **2003**, 1547–1554 (2003)

Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**(1), 117–128 (2011)

Joe, S., Kuo, F.Y.: Remark on algorithm 659: implementing sobol's quasirandom sequence generator. ACM Trans. Math. Softw. **29**(1), 49–57 (2003)

Johnson, M.K., Hasher, L.: Human learning and memory. Annu. Rev. Psychol. **38**(1), 631–668 (1987)

Knight, W.: An AI-driven robot hand spent a hundred years teaching itself to rotate a cube. https://www.technologyreview.com/s/611724 (2018) Accessed 16 Oct 2020

Lee, I.S., Lau, H.Y.: Adaptive state space partitioning for reinforcement learning. Eng. Appl. Artif. Intell. **17**(6), 577–588 (2004)

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. IEEE Trans. Knowl. Data Eng. **2019**, 1 (2019)

Lipton, J.I., Fay, A.J., Rus, D.: Baxter's homunculus: virtual reality spaces for teleoperation in manufacturing. IEEE Robot. Autom. Lett. **3**(1), 179–186 (2018)

Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Turkit: Human computation algorithms on mechanical turk. In: The 23rd Annual ACM symposium on user interface software and technology, pp. 57–66 (2010)

Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. **2018**, 1 (2018)

Mandlekar, A., Zhu, Y., Garg, A., Booher, J., Spero, M., Tung, A., Gao, J., Emmons, J., Gupta, A., Orbay, E., et al.: Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In: Proceedings of The 2nd Conference on Robot Learning, vol. 87, pp. 879–893 (2018)

Nguyen-Tuong, D., Peters, J.: Online kernel-based learning for task-space tracking robot control. IEEE Trans. Neural Netw. Learn. Syst. **23**(9), 1417–1425 (2012)

Niederreiter, H.: Low-discrepancy and low-dispersion sequences. J. Number Theory **30**(1), 51–70 (1988)

Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: The SIGCHI conference on human factors in computing systems, pp. 1403–1412 (2011)

Rasmussen, C.E., Williams, C.K.: Gaussian Processes for Machine Learning, vol. 1. MIT Press, Cambridge (2006)

Sorokin, A., Berenson, D., Srinivasa, S.S., Hebert, M.: People helping robots helping people: crowdsourcing for grasping novel objects. In: IEEE/RSJ international conference on intelligent robots and systems, pp. 2117–2122 (2010)

Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)

Uther, W.T., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: The annual conference on innovative applications of artificial intelligence (IAAI), pp. 769–774 (1998)

Van Hoof, H., Hermans, T., Neumann, G., Peters, J.: Learning robot in-hand manipulation with tactile features. In: The 15th IEEE-RAS international conference on humanoid robots, pp. 121–127 (2015)

von Ahn, L.: Human Computation. PhD Thesis, Carnegie Mellon University, Pittsburgh, PA (2005)

Wang, C., Zhao, Y., Lin, C.Y., Tomizuka, M.: Fast planning of well conditioned trajectories for model learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1460–1465 (2014)

Wang, C., Lin, C.Y., Tomizuka, M.: Statistical learning algorithms to compensate slow visual feedback for industrial robots. J. Dyn. Syst. Measure. Control **137**(3), 031011 (2015)

Wang, C., Zhao, Y., Chen, Y., Tomizuka, M.: Nonparametric statistical learning control of robot manipulators for trajectory or contour tracking. Robot. Comput.-Integr. Manuf. **35**, 96–103 (2015)

Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., Abbeel, P.: Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 1–8 (2018)

Zhao, L., Lawhorn, R., Wang, C., Lu, L., Ouyang, B.: Synthesis of robot hand skills powered by crowdsourced learning. In: IEEE International Conference on Mechatronics, pp. 211–216 (2019)

Zhao, L., Zhao, Y., Patil, S., Davies, D., Wang, C., Lu, L., Ouyang, B.: Robot composite learning and the nunchaku flipping challenge. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3160–3165 (2018)

Zhao, L., Lu, L., Wang, C.: Data-oriented state space discretization for crowdsourced robot learning of physical skills. ASME Lett. Dyn. Syst. Control **1**, 2 (2020)

Leidi Zhao is currently a PhD candidate in Electrical Engineering at New Jersey Institute of Technology. He received his MSc degree in Electrical Engineering from New Jersey Institute of Technology in 2016, and BSc degree in Electrical Engineering from Shanghai Maritime University in 2015. His current research focuses on data-driven learning schemes for robot physical intelligence such as dynamic and dexterous manipulation skills.

Dr. Lu Lu received the B.Eng. degree in Mechatronic Engineering from Zhejiang University, Hangzhou, China, in Aug 2008, the M.Sc. and the Ph.D. degree in Mechanical Engineering from Purdue University, West Lafayette, in Dec 2010 and Aug 2013, respectively. From Sep 2013 to Aug 2015, he worked as a postdoctoral Research Associate at the Center for Automation Technologies and Systems (CATS) at Rensselaer Polytechnic Institute. Since Aug 2015, he has been a faculty member in Mechanical and Industrial Engineering at New Jersey Institute of Technology. Dr. Lu's research interests include controls, robotics, UAV, artificial intelligence, and human-machine interaction.

Dr. Cong Wang is a faculty member in Electrical and Computer Engineering at New Jersey Institute of Technology. Before joining NJIT in 2015, Dr. Wang was a Lecturer and Research Engineer at University of California, Berkeley, where he earlier obtained a PhD degree in the area of Controls and Dynamics in 2014. He also attended Tsinghua University and obtained a master's degree in Automotive Engineering and a bachelor's degree in Manufacturing Engineering and Automation in 2010 and 2008 respectively. His research interests include nonlinear systems, learning control, and robot physical intelligence.