Inferring symbols from demonstrations to support vector-symbolic planning in a robotic assembly task

Xueyang Yao, Saeejith Nair, Peter Blouw, Bryan Tripp

Inferring symbols from demonstrations to support vector-symbolic planning in a robotic assembly task

Xueyang Yao
Department of Systems Design Engineering
University of Waterloo
Waterloo, Canada
x35yao@uwaterloo.ca

Saeejith Nair
Department of Mechanical and Mechatronics Engineering
University of Waterloo
Waterloo, Canada
smnair@uwaterloo.ca

Peter Blouw
Applied Brain Research
Waterloo, Canada
peter.blouw@appliedbrainresearch.com

Bryan Tripp
Department of Systems Design Engineering
University of Waterloo
Waterloo, Canada
bptripp@uwaterloo.ca

***Abstract*—While deep reinforcement learning is increasingly used to solve complex sensorimotor tasks, it requires vast amounts of experience to achieve adequate performance. Humans, by comparison, can correctly learn many tasks from just a small handful of demonstrations. A common explanation of this difference points to the human use of symbolic representations that constrain search spaces, enable instruction following, and promote the transfer and reuse of knowledge across tasks. In prior work, we developed neural network models that use vector-symbolic representations to plan complex actions, but the challenge of learning such representations and grounding them in sensorimotor data remains unsolved. As a step towards addressing this challenge, we use a programming-by-demonstration approach that automatically extracts symbols from sensorimotor data during a simulated robotic assembly task. The system first observes a small number of demonstrations of the task, and then constructs models of the associated states and actions. The system infers task-relevant states by clustering the positions and orientations of objects manipulated during the demonstrations. It then builds vector-symbolic representations of these states, and of actions defined in terms of transitions between states. This approach allows rapid and automatic sensory grounding of vector-symbolic representations, which in turn support flexible, robust, and biologically plausible control.**

***Index Terms*—vector-symbolic representation, clustering, planning, robotics**

I. Introduction

Many of the shared concepts that allow us to communicate with each other are grounded in our experiences. Deep reinforcement learning [1] allows artificial agents to convert experience into sophisticated behavior, via development of task-related internal representations. However, these representations are not explicit or discrete enough to support communication. In contrast, classical planning [2] achieves sophisticated behavior using explicit symbolic representations that are compatible with communication and instruction, but the symbols in these systems are grounded manually.

This work was supported by Applied Brain Research, and the Natural Sciences and Engineering Research Council (CRD Grant 519891).

Previously [3], we developed a model of vector-symbolic planning as a recurrent network of spiking neurons. Vector-symbolic architectures use high-dimensional vectors to represent symbols, and support binding of multiple concept vectors to create new, composite concepts. We used the Semantic Pointer Architecture [4], which combines vector-symbolic architectures with additional biological constraints. Our model used semantic-pointer representations of states and actions, and could reliably plan novel action sequences to achieve goals. We also showed that our approach scaled to environments of realistic complexity, involving tens of thousands of objects and possible actions. Because the model was implemented entirely in spiking neurons, we believe that it could be a plausible model of human vector-symbolic representations that are grounded in daily activities. However, in our past work, we did not address the grounding of these concepts in sensory data.

Recent work in robotics suggests an approach to filling that gap. Many studies have dealt with the problem of robotic learning from human demonstration (reviewed by [5]–[7]). Some of these studies have used symbol-based systems, along with symbolic planners [8]–[10] or motion grammars [11]. In such systems, heuristics are used to identify relevant states, and typically (e.g. [8], [9]) clustering is used to group such states so that they can be represented symbolically. In the present study, we develop such a system and integrate it with a vector-symbolic planning system (a simplified version of our previous model) to control a simulated assembly task. This is a small step toward grounding our biologically plausible model of action-based representations in a non-symbolic sensory stream. Given both the importance of action-based representations in contemporary theories of linguistic cognition [12], and the role that vector-symbolic representations often play in these theories [13], [14], our approach helps to set the stage for grounded yet structured models of the representations involved in higher level cognitive processes.

II. METHODS

Our previous work [3] suggested the biological plausibility of STRIPS-style planning in daily activities. Our planner did not produce optimal plans, rather it produced viable short-term plans in about 200ms of simulated time, so that re-planning could occur frequently during a complex activity (e.g. preparing a meal). The goal of the present work is to experiment with a programming-by-demonstration (PbD) approach, to produce action representations that consist of preconditions and effects, compatible with our planning model. A variety of related approaches has been taken in robotics. Perhaps most closely related, [8] used PbD to infer relevant discrete states, and constraints on ordering of state transitions. Compared to that work, we experiment with a different task, and our system produces explicit action representations that are consistent with [3] (rather than ordering constraints). These representations could also perhaps, in future work, be re-used in multiple tasks, and/or used as pointers to lower-level action-specific skills developed through practice. Finally, we integrate and test these representations with a vector-symbolic planner.

A. Recording and Clustering States

In PbD systems that operate on discrete object states, the states are recorded only at certain times, and they are clustered to produce models of state categories. A key design decision is when to record object states for clustering. Here we consider robotic manipulation tasks, and we record object states at the moments when the gripper makes contact with an object (e.g. to pick something up) and relinquishes contact (e.g. to put it back down). This approach could be generalized to include inflection points in the gripper trajectory, if the trajectory does not take a direct path between start and end states.

The clustering approach is another key decision. A number of past studies have used k-means clustering with a Euclidean distance metric. A limitation of that approach is that it assumes hyper-spherical clusters, whereas practical states are often relatively invariant to certain dimensions. For example, the hooks of coat hangers have tight clustering around the bar, but widely varied positions along the bar. Sometimes, an object's precise orientation is an important part of a given state (e.g. when a bolt inserted into a nut), and sometimes the orientation is a highly variable property of a given state (e.g. when a bolt is a bin). In more naturalistic scenarios, even the states of simple objects may have many dimensions, such as positions and orientations relative to various other objects, as well as temperature, color, etc., and many of these dimensions may be irrelevant to a given goal. We therefore seek a distance metric in which the distances between states are small when states are similar in only a few dimensions. For this purpose, we use p-norms of the differences between positions,

$$d_{u,v} = \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{\frac{1}{p}}, \tag{1}$$

with $p < 0$, where n is the state dimension. Figure 1 further motivates this choice.

B. Vector-Symbolic Architecture

Vector Symbolic Architectures [15] (VSA) have been applied to a broad range of tasks related to manipulating structured information. In a VSA, concepts are represented by high dimensional vectors. Atomic concepts are typically represented as random vectors (which are nearly orthogonal in high dimensions), and more complex concepts are composed via addition and binding operations. For example, symbols *RED* and *BALL* could be represented as random 256-dimensional vectors, and the concept *RED_BALL* could then be composed from the *RED* and *BALL* vectors with a binding operation. Here we use holographic reduced representations (HRR) [16], a kind of VSA that uses circular convolution (written as $\circledast$) as a binding operator. Using HRRs, the concept *RED_BALL* can be represented as $RED\_BALL = RED \circledast BALL$, and the objects in a simple scene might be represented as $SCENE = RED \circledast BALL + BLUE \circledast BLOCK$. Importantly, an inverse operation can be used to query information from a compound HRR. For example, due to near-orthogonality of the high-dimensional vectors, the result of $SCENE \circledast BALL^{-1}$ will yield the vector for *RED* approximately, answering the question, "What color is the ball in the scene?"

C. Assembly Task

We tested our approach in a robotic assembly task, simulated in PyBullet (Figure 2). The task begins with nuts and bolts on a table, and the goal is to attach a nut and bolt together. A jig is available to ensure that the nuts and bolts are precisely vertical, so that they fit together properly. Figure 2 illustrates the PyBullet simulation. Because this work is preliminary and exploratory, we made a number of simplifications to better focus on the most relevant aspects of the system. In particular, we used ground-truth object type, position, and orientation data from the simulator. This is a reasonable simplification, insofar as state-of-the-art methods of estimating these from images are reasonably accurate. For grasping the nuts and bolts, we used known-good grasp points with opposing flat surfaces, because simple heuristics are often effective for choosing grasp points on isolated objects from vision, e.g. [17]. Finally, we simplified the nut and bolt to remove the threads, and we considered the task complete when the bolt shaft was inserted through the nut. This eliminated the need to simulate the physics of thread interactions, and to monitor forces to detect thread binding. Probably the main effect of these simplifications was to reduce the error rate. Our simulator did occasionally drop a nut or bolt, prompting the system to re-plan, but re-planning would have been needed more often if there had been additional sensory and motor errors. The process of formulating vector-symbolic representations for the task involves two main stages, demonstration and model building, which we describe next.

D. Collecting Demonstration Data

In the demonstration stage, the system is shown examples of a 7-DOF robotic arm performing the assembly task. In PbD systems, robots are often tele-operated by humans to provide

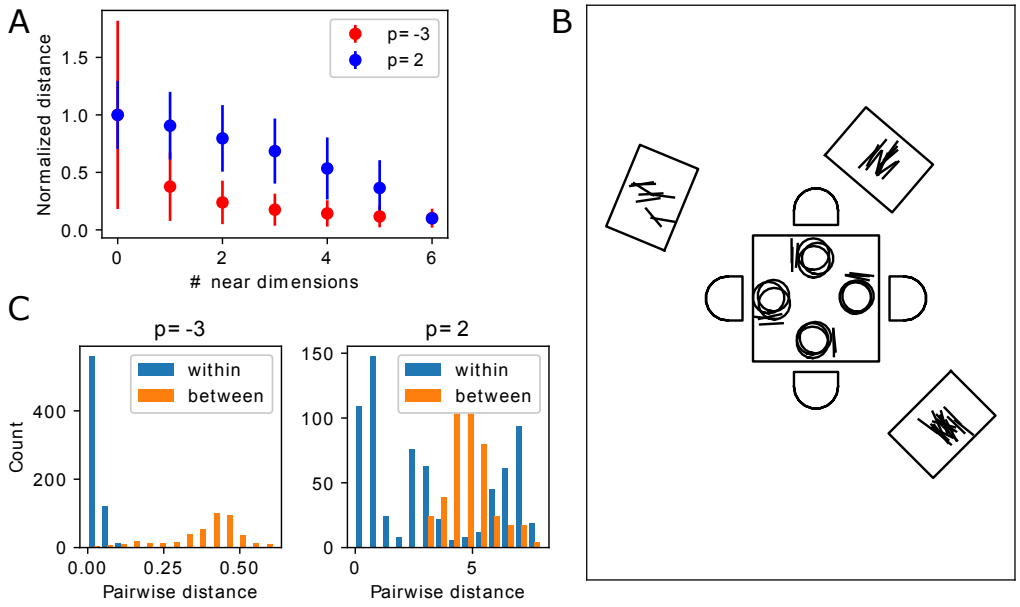

Fig. 1: Using p-norms with negative exponents to cluster states. A: With a negative exponent, the distance between states is fairly low even if the states are only similar in one or two dimensions. To illustrate, we show pairwise distances (mean +/- SD) between 6D states, with tight Gaussian distributions ($\sigma = .05$) in some dimensions, and broad distributions ($\sigma = .5$) in other dimensions. Using the standard Euclidean norm ($p = 2$; blue), pairwise distances are large even if all the states are close in three or four dimensions. However with $p = -3$, pairwise distances are lower when the states are close in only one or two dimensions. B: To further illustrate, we show a simplified simulation of a table-setting task. Positions of plates, knives, and a cart are overlaid from three example tables. Given such observations, we would like a robot to notice that knives are sometimes on the cart, and sometimes next to plates, orthogonal to the table edge. C: Pairwise distances between knife states in the table-setting example. The 8D state vector includes positions and orientations relative to the cart and the closest plate and table edge. Using a Euclidean norm with $p = 2$, pairwise inter-state distances within a group (i.e. on the cart or beside a plate) are similar to pairwise distances between groups (right), but these inter-state distances are well separated with $p = -3$ (left).

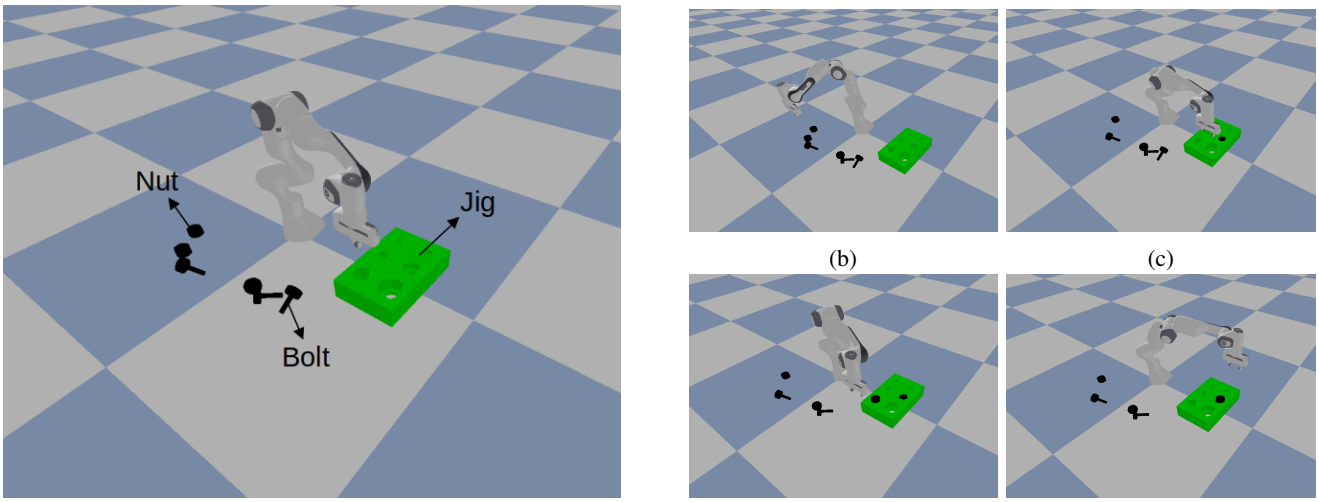

Fig. 2: Simulation environment and main stages of the robotic assembly task. (a) A view of the simulation environment with main objects labelled. (b) Bolts and nuts are on the table. (c) One of the nuts is placed in the jig. (d) One of the bolts is placed in the jig. (e) Task complete: bolt and nut are assembled.

demonstration data, but for efficiency we scripted the task. The task requires the following actions (note that the first and second actions can be performed in either order):

1) If there is a nut on the table and the nut slot in the jig is empty, the robot should place a nut in the jig.
2) If there is a bolt on the table and the bolt slot in the jig is empty, the robot should place a bolt in the jig (the bolt shaft protrudes down through a hole in the jig, so that gravity ensures the bolt is vertical).
3) If the jig contains both a nut and a bolt in their respective slots, the robot should assemble the bolt into the nut.

E. Building Action Representations

After a few demonstrations are complete, the next step is to cluster the states. We used 3D positions to cluster the states of each type of object, and we modelled the positions and orientations of each cluster with 6D Gaussians. Although some states are tightly clustered, the nuts and bolts have wide horizontal distributions when they are on the table. Following the motivation discussed earlier, we defined the distance between a pair of positions as the p-norm of the difference between positions, using $p = -2$. This resulted in the nuts and bolts on the table joining a single cluster each. Given n 3D positions, we compute a $n \times n$ distance matrix between pairs of states. We then run a bottom-up hierarchical clustering algorithm [18] on the distance matrix, recursively merging together pairs of points that minimally increase the average pre-computed linkage. No hints are provided regarding the expected number of clusters or size of each cluster. Instead, the method generates clusters of points satisfying a given distance threshold. Experimental results on randomly sampled data batches of sizes ranging from just 15 points to 300 points showed that the method could generate perfectly accurate clusters using a maximum threshold of (1.169 ± 0.001)mm. However, perfect accuracy is not necessary since we use robust estimates of the cluster statistics.

The discrete high-level states of the nuts and bolts, used in planning, correspond to the resulting clusters. For each cluster, a robust mean state was computed and saved after discarding the points in the top and bottom decile of each dimension. The position means were used to classify unseen data points during the planning stage. Both the position and orientation means were used to determine the target configurations of the robot when executing an action.

After learning the possible states of each object type, the system builds action representations, associating one action with each state that occurs at the end of gripper contact. For example, since a nut is sometimes in the jig when the gripper releases it, we form a corresponding action (one might call it, $PUT\_NUT\_IN\_JIG$). For each action in the demonstration data, we then form a global state vector G of length m before and after each action, where m is the total number of states across all object types (in this task, $m = 5$). Each element in G is a binary quantity indicating whether or not there is an object in the scene which satisfies the corresponding state. Table I shows an example of a global state vector for the action $PUT\_NUT\_IN\_JIG$, when there is a nut on the table.

Action: $PUT\_NUT\_IN\_JIG$		
States	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	0	0
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	0
BOLT_ASSEMBLED	0	0

TABLE I: Examples of global state vectors before and after the action $PUT\_NUT\_IN\_JIG$ for demonstration 1. The value assigned to each local state is 1 if an object exists in that state, and 0 otherwise. The state labels in the table have been added manually to help interpretation (they are not part of the system).

The global state is important when an action has preconditions related to multiple objects, as before the final assembly step when both a nut and bolt must be in the jig. Given the global state vectors, we build a set of preconditions as well as an associative memory. The preconditions indicate the object states necessary for an action to be executed and can be derived from the enabled bits of the global state vectors at the beginning of each instance of an action. Table I shows an example in which the state *NUT_ON_TABLE* preceeded the action *PUT_NUT_IN_JIG*. However, as seen in Table II, the states {*NUT_ON_TABLE, BOLT_ON_TABLE*} were present before a different instance of the same action. It is evident that *BOLT_ON_TABLE* is not a requirement for this action. To address such coincidences, the preconditions for an action are found by $argmax\,(\sum_{i=1}^{n} G_i)$, where n is the number of demonstrations, and G_i is the global state vector preceding the i^{th} instance of the action. This way, only the states that consistently occur before an action are interpreted as its preconditions. It is worth noting that $argmax\,(\sum_{i=1}^{n} G_i)$ might yield multiple solutions, which is reasonable because the one action could have multiple preconditions (e.g. the preconditions of action *ASSEMBLE* is *NUT_IN_JIG* and *BOLT_IN_JIG*). While we took an action's preconditions as the states that were consistently present before the action, one could also consider states that are consistently absent. This would allow the system to learn, for example, that the bolt part of the jig must be empty before putting a bolt in it. We instead used the additional heuristic that nothing can be put in a non-empty position.

After the true preconditions were obtained, we then defined each action as two HRRs, A_s and A_d. A_s was a shallow representation, a randomly assigned 256-dimensional vector that identified the action A. A_d was the corresponding deep representation, which had the same dimension as A_s but contained information about its preconditions. For example, if the precondition of action $PUT\_NUT\_IN\_JIG$ were found to be NUT_ON_TABLE, the deep representation of action $PUT\_NUT\_IN\_JIG$ would be defined as: $PUT\_NUT\_IN\_JIG_d = PRECONDITION_s \circledast NUT\_ON\_TABLE_s$.

The associative memory is a dictionary that maps an object state to the action which caused it, and can be derived from

Action: $PUT\_NUT\_IN\_JIG$		
States	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	1	1
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	0
BOLT_ASSEMBLED	0	0

TABLE II: Global state vectors before and after the action $PUT\_NUT\_IN\_JIG$ for demonstration 2. Note the unnecessary local state *Bolt on Table*.

the transition between two consecutive global state vectors. For example, in Table II, the value of the *NUT_IN_JIG* changes from 0 to 1, which indicates that the effect of action *PUT_NUT_IN_JIG* is that the local state *NUT_IN_JIG* becomes enabled. However, a single demonstration would not be robust to coincident events, e.g. actions by other people in the environment. Table III shows state transition for the action *PUT_NUT_IN_JIG* where a bolt was put in the jig at the same time by a person. It is obvious that *BOLT_IN_JIG* should not be the effect of the action *PUT_NUT_IN_JIG*. To address this, the index to the associative memory was defined as $argmax\left(\sum_{i=1}^{n}(H_i - G_i)\right)$, where n is the number of demonstrations, G is the global state vector preceding an action and H is the global state vector after an action. Given the 3 demonstrations shown in Table I-III, the system would derive that the effect of the action *PUT_NUT_IN_JIG* is *NUT_IN_JIG*. In summary, tasks demonstrations were used to build vector-symbolic representations of states and actions, and an associative memory that maps desired states to actions that lead to those states.

Action: $PUT\_NUT\_IN\_JIG$		
Labels	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	0	0
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	1
BOLT_ASSEMBLED	0	0

TABLE III: Example of a global state vectors before and after the action *PUT_NUT_IN_JIG* for demonstration 3. Note that a bolt was put in the jig by another subject at the same time.

F. Planning process

Using these state and action representations, and the associative memory, the planning process is executed as follows: For each step in the simulation, the planner assigns each object to the nearest state (the state with center at the shortest p-norm distance from the object's position). After all objects are clustered and the global state vector G is built, the planner considers the goal (e.g. *BOLT_ASSEMBLED*). First, the planner checks G to see if the goal is already met. If not, an action that leads to the goal is queried from the associative memory. Then, the planner finds the preconditions of the action by calculating $action_d \circledast PRECONDITION_s^{-1}$. The action is executed only if all of its preconditions are satisfied in G. Every unsatisfied precondition becomes a new goal, and the planning process repeats. In [3], a new goal was obtained at each step by subtracting the action's effect from the *GOAL* vector and adding its preconditions, whereas in our present simplified system, the preconditions replace the previous goal. Another difference is that in [3], actions were added to a stack (implemented as a recurrent network) during planning, whereas in the present system we only retain the earliest action in the plan, and re-plan at every step.

III. Results

A. Robustness of clustering

Figure 3 shows how the clustering algorithm identifies different states given the objects' position information. The state labels were added manually to help interpretation. The algorithm tended to cluster data points into one group as long as they were close in any of the dimensions. For instance, nuts on the table were clustered into one group because their z coordinates were similar, even though their x, y coordinates varied. After clustering, means and covariance matrices were calculated to build a Gaussian distribution model for each of the state clusters. These models were then tested on a test set that contained 602 unseen data points. As can be seen in Figure 4, the generated state models yielded 100% success rate on the test set.

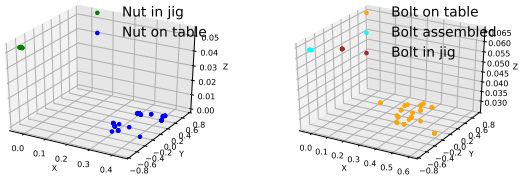

Fig. 3: Clusters of position information for bolts and nuts obtained from five demonstrations. Left: Positions of the nuts are clustered into two states. Right: Positions of the bolts are clustered into three states.

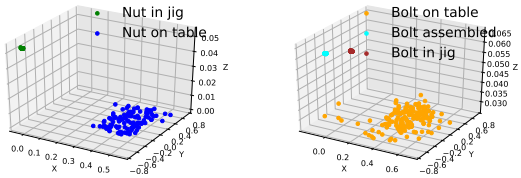

Fig. 4: Test results for the state models obtained from clustering. The test set contained 602 data points achieving 100% success rate.

B. Robustness of the HRR structures

As described in Section II, actions were represented with with HRRs: one shallow representation and one deep representation. States, on the other hand, were represented with only one HRR because different states were clustered based only on position information. However, such information is generally not sufficient to describe states in a task agnostic manner. To

set the stage for more generic state representations that reuse shared features, we explored an approach in which a shallow HRR was used to identify a state and a deep HRR was defined based on the object kind (e.g. *BOLT*, *NUT*), location of the object (e.g. *TABLE*, *JIG*), and the assembly state of the object (e.g. *NOT_READY*, *READY*, *ASSEMBLED*).

Three variants of this approach were tested. In the first, the deep HRR for a state took the form $object \circledast (location + assemble\_state)$, while in the second this deep HRR took the form $OBJECT_s \circledast object + LOCATION_s \circledast location + ASSEMBLE\_STATE_s \circledast assemble\_state$. In the third variant, the deep HRR took the form $object \circledast (location + assemble\_state) + EFFECT\_OF_s \circledast action$, where the action was the cause of the state being represented. The planning success rate of a system using each structural variant as a function of representational noise is shown in n Table IV. A plan is considered successful only if the correct action is executed at every step. The first structural variant, which contains fewer components reused across representation, yields higher success rate as expected. The concepts were represented as 256-dimensional vectors in the test.

	$\sigma = 0.5$	$\sigma = 0.4$	$\sigma = 0.3$	$\sigma = 0.2$	$\sigma = 0.1$
Structure 1	80.5%	94.6%	96.9%	98.3%	98.8%
Structure 2	44.9%	68.3%	83.1%	90.6%	95.6%
Structure 3	13.5%	40.0%	68.7%	83.8%	91.5%

TABLE IV: Success rate of the planner with different HRR structures with respect to different noise levels. σ is the standard deviation of the Gaussian noise added. The standard deviation of the clean HRR is 0.0625.

C. Minimal Demonstrations and Failure Modes

The system can reliably model and plan the task after a single demonstration. However, the system fails if a test scenario differs too much from demonstration scenarios. In particular, if there are always multiple nuts or bolts at demonstration time, the system infers that at least one nut and bolt has to be on the table, as a precondition to any action. This can be avoided by demonstrating the task with only one nut and one bolt. However, this limitation should be addressed in future work.

IV. Discussion

In this work we experimented with programming-by-demonstration methods for grounding vector-symbolic representations. We found that this approach can be used to efficiently learn state and action representations that support robust vector-symbolic planning, with a small number of task demonstrations. While other work has often used k-means clustering of states, along with Euclidean distance, we found that p-norms with negative exponents can naturally account for dimensions that are semantically irrelevant to a state.

Limitations of our current approach largely fall into two categories. The first concerns realistic sensory estimation of object positions and orientations, which is a central problem that needs to be solved prior to transitioning out of simulation and into a real-world robotic device. The second limitation concerns generalizing to a wider range of tasks involving more complicated action sequences with variably structured preconditions.

The use of vector-symbolic representations in our planner may aid generalization in such cases, as these representations support both soft pattern-matching and structural comparisons that can aid in reasoning over action preconditions. Additionally, these representations could later be extended to include linguistic elements, such that in a scenario of the sort shown in Figure 1, the planning system could cue the movement of a utensil with a term like "beside a plate". Exploring such extensions will be a focus for future work.

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[2] D. E. Wilkins, *Practical planning: extending the classical AI planning paradigm*. Elsevier, 2014.

[3] P. Blouw, C. Eliasmith, and B. P. Tripp, "A scaleable spiking neural model of action planning," in *CogSci*, 2016, pp. 1583–1588.

[4] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain." *Science*, vol. 338, no. 6111, pp. 1202–5, nov 2012.

[5] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020.

[6] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7(2), 2018.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2008.10.024

[8] S. Ekvall and D. Kragic, "Learning task models from multiple human demonstrations," in *ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006, pp. 358–363.

[9] Y. Mollard, T. Munzer, A. Baisero, M. Toussaint, and M. Lopes, "Robot programming from demonstration, feedback, and transfer," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[10] Y. S. Liang, D. Pellier, H. Fiorino, and Sylvie Pesty, "A framework for robot programming in cobotic environments: First user experiments," in *Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering*, 2017, pp. 30–35.

[11] N. Dantam, I. Essa, and M. Stilman, "Linguistic transfer of human assembly tasks to robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 237–242.

[12] T. Harley, *Psychology of Language: From data to theory*, 4th ed. Psychology Press, 2014.

[13] P. Smolensky and G. Legendre, *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. MIT Press, 2006, vol. 1.

[14] M. Jones and D. Mewhort, "Representing word meaning and order information in a composite holographic lexicon." *Psychological Review*, vol. 114, no. 1, pp. 1–37, 2007.

[15] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *CoRR*, vol. abs/cs/0412059, 2004. [Online]. Available: http://arxiv.org/abs/cs/0412059

[16] T. A. Plate, "Holographic reduced representations." *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–41, jan 1995. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/18263348

[17] V. R. Osorio, R. Iyengar, X. Yao, P. Bhattachan, A. Ragobar, N. Dey, and B. Tripp, "37,000 human-planned robotic grasps with six degrees of freedom," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3346–3351, 2020.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.