Exploring hyperdimensional computing recall of reactive behavior for 2D navigation

Alisha Menon, Youbin Kim, Braeden Benedict
{allymenon,youbin_kim,braeden}@berkeley.edu

I. Introduction

Hyperdimensional computing (HDC) is computing paradigm that models how the brain might perform distributed computation and representation [1]. HDC uses a limited set of operations over vectors of extremely large dimensions (in the order of thousands) to encode and compute information and has found many applications in learning and cognition tasks. Recently, Neubert et al. have proposed a HDC algorithm to perform one-shot supervised learning of reactive robot behavior in a navigation problem [2]. However, they do not provide details on their training methodology and environment, nor concrete results and analysis.

In this project, we apply the proposed HD recall of reactive behavior algorithm on a simple 2D navigation problem. We show how different sensor encoding strategies and algorithm hyperparameter tuning can affect our navigation performance. Ultimately, we achieve an 89% success rate in our navigation task and show that HD recall has the potential to become a powerful tool in learning goal-oriented navigation behavior through the integration of relevant sensor information.

II. Training and Testing Environment

The problem we attempt to solve is navigation and obstacle avoidance in a 2D world. The world is a 10×10 grid with randomly placed obstacles as shown in figure 1. The blue square indicates your current location while the green is the goal location. The robot wishes to traverse to the goal location while avoiding any obstacles in its path. At each time-step, the robot can move in four directions (up, down, left, right) according to its environment sensor data. To mimic a real-world scenario, the robot is only given sensor data for the presence of an obstacle in each of four directions around it, as well as the direction of the goal location and its previous movement. Since HD recall is a supervised algorithm, training data was provided by manually moving the robot to the goal location through various randomly generated maps with random start and goal locations. Testing was similarly done on 1000 trial runs on randomly generated maps.

Figure 1: Game world with robot (blue) and target location (green).

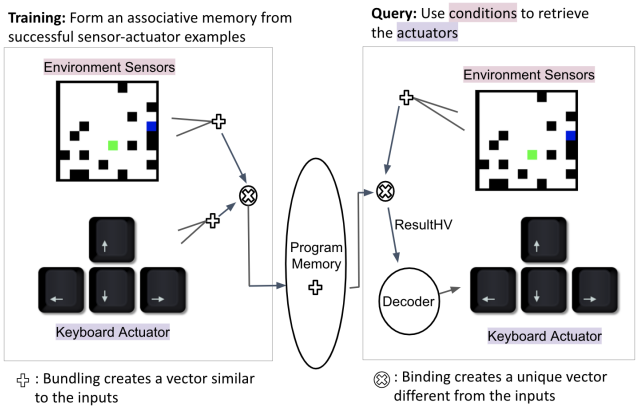


Figure 2: Overview of HDC recall of reactive behavior algorithm.

III. HD Recall Algorithm

The HD recall algorithm uses two HD operations to encode information. First, all sensor and actuator data is assigned to a random binary hypervector (HV). These assignments are fixed and stored in what is called an item memory (IM). The sensor data is encoded into a single HV called the condition vector using binding and bundling operations. Binding is an elementwise XOR and produces a HV that is dissimilar (according to Hamming distance) to its operands. Bundling is an elementwise majority function that produces a HV that is similar to its operands. The exact encoding methodology and exploration will be explained in the next section. For each training sample, the sensors are encoded into the condition vector and binded with the HV that represents the corresponding robot action. The resultant HV is then summed across all training examples into a single program vector.

During the testing phase, the same exact encoding is performed on the input sensor data. We then bind the resulting condition vector to the stored program vector. Now we compare the binded vector to the IM containing the representations for the possible actuator actions (in our case up, down, left right). The vector in the IM with the least Hamming distance to the query vector represents the desired movement for the given input sensor data. An overview of the entire process is shown in figure 2.

IV. Sensor encoding

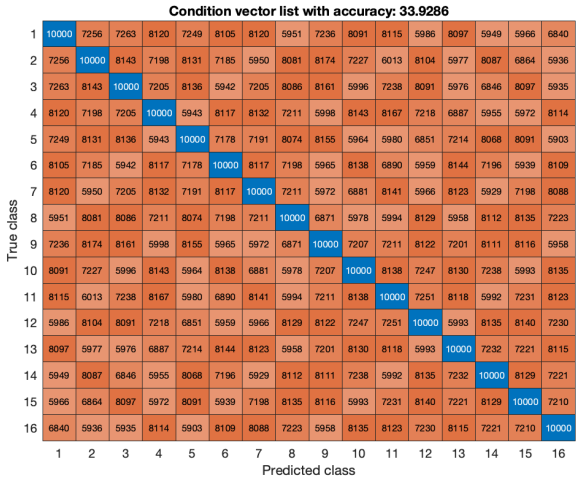
Determining an optimal strategy for sensor encoding was a focus of this work. While HDC has an advantage over other machine learning methods in that it is not a black box, the choice of the encoding methodology will play a large role in the algorithm's performance.

A. Preliminary obstacle avoidance findings

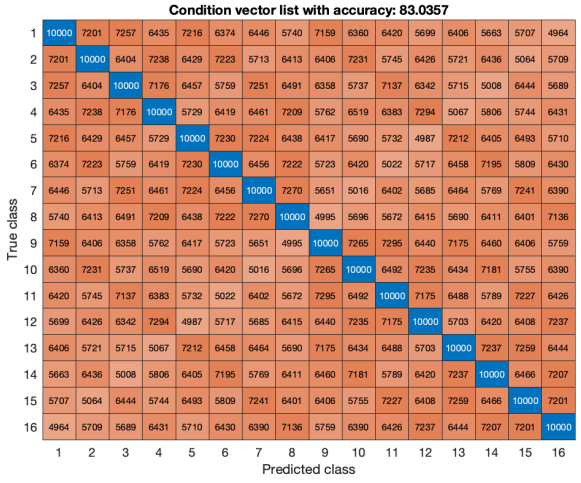
The first step towards implementation was performing recall based on just the obstacle detection sensors and a corresponding action that avoids the obstacle. Only unique conditions were included in this version and the encoding method matched the one detailed in [2]. An item memory (IM) was randomly generated for each the four different sensors; vectors representing the existence and absence of an obstacle were also generated. Initially, the same existence and absence vectors were used for all of the four sensors. For a given sample, each of the sensor IM vectors was binded with the corresponding existence or absence vector. Then, each of these sensors were bundled together and then binded with the corresponding actuator vectors. Since there are finite possibilities to the sensor combinations ($2^4 = 16$), each was tested during recall to verify that the output was the same actuator response that the sensor combination had been trained with.

From the start, this encoding method demonstrated low accuracy for such a simple task averaging around 60% accuracy though with a large range ($30\% - 80\%$) as a result of inherent randomness. A similarity matrix representing Hamming distances between the different sensor combinations for one of the lower accuracy examples is shown in Fig. 3a. It was clear that there were many condition vectors that were quite similar to each other which would easily result in mis-recalls due to the added noise of unbinding from the program vector. The first change that was implemented in order to increase the orthogonality between different condition vectors was to use a different existence/absence vectors per sensor. One method to achieve this is to initialize additional random vectors, another was to simply use permuted versions of each of the initial existence/absence vectors where the number of permutations is defined by the sensor number as the permutation operation also generates a pseudo-orthogonal vector. Both these methods achieved similar results which averaged around 90% accuracy. The updated similarity matrix is shown in Fig. 3b. The overall similarity has decreased across the chart even for the displayed low accuracy example, leading to the higher accuracy as compared to the original encoding scheme.

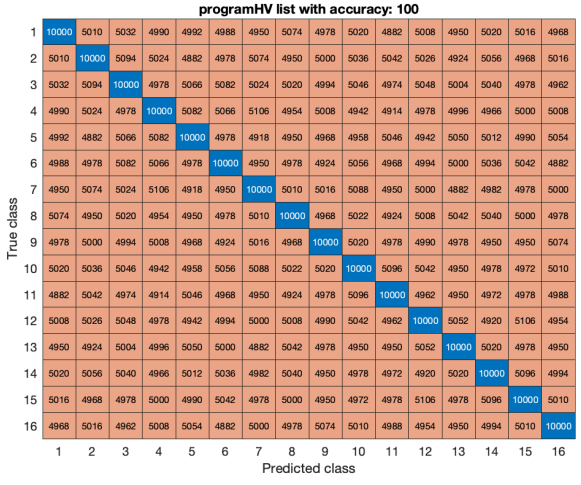
The last change that was implemented was to instead experiment with the encoding structure itself by binding each of the sensor vectors together instead of bundling in order to generate the condition vector. This created pseudo-orthogonal vectors for each of the condition vectors due to the property of the binding operation. This change resulted in an average of 100% accuracy. The corresponding similarity matrix is shown in Fig. 3c. In this chart, every condition vector is nearly orthogonal from the other sensor combinations leading to the perfect recall. This is acceptable for this application



(a)



(b)



(c)

Figure 3: Similarity matrix of Hamming distances between condition vectors for all sensor combinations with obstacle and absence vectors that are (a) the same for each sensor (b) different for each sensor (c) different for each sensor with each sensor's vector binded together.

because the incoming sensor inputs will always be one of the 16 possible combinations, not a value similar or in between, and hence obviates the need for correlation between similar or different sensor inputs. By using this encoding scheme, obstacle avoidance is guaranteed using only these sensors on a 2D map. However, this work aimed to incorporate goal-oriented behavior into the system as well. This involved integrating additional sensor information into the encoding process that could influence the robot's behavior.

B. Target and obstacle avoidance encoding

In addition to the four previously described obstacle sensors, information about the direction of the target is required for the autonomous robot to navigate to the goal location. This was provided as a ternary value $(-1, 0, +1)$. For example, for the X direction this encoded whether the actor was on a column to the left of, in line with, or to the right of the target. The direction of the last move was also provided as a sensor input to help guide the actor on a "purposeful" course and prevent it from getting stuck just moving back and forth between two squares.

Combining the information from these target direction and last move sensors with the previously described obstacle sensors can be accomplished in multiple ways. Fig. 4 shows the overall process. Sensor IDs are bound to the sensor values to provide the data from each sensor. As mentioned previously, the sensor values for different sensors are represented by distinct vectors formed by permutation, e.g. $\text{leftSensorVal}(\text{sensor} = 1) \neq \text{rightSensorVal}(\text{sensor} = 1)$. This sensor data is then combined using some encoding strategy to form a condition vector, which is ultimately bound with the actuator vector.

Here we describe three methods shown in Fig. 5 to accomplish the encoding in detail; additional methods were also attempted. Method 0 involves a bundling of the three sensor modalities (obstacles, target, and last move), which creates a vector which is similar to the vector for each modality.

Method 1 creates representations for the X and Y directions through binding the horizontal obstacle data to the horizontal target data and the vertical obstacle data to the vertical target data. These directional representations are then bundled with the last move sensor. This method was inspired by how a person approaches this navigation task, which is likely not by combining each sensor modality together first, but by combining information on constraints/goals in the possible movement directions together first.

Method 2 bundles the target direction sensors and last move sensors and binds these to the combined obstacle representation. Conceptually, this represents a binding of the constraints and goals for the actor. Mathematically, this is equivalent to projecting the bundled direction and last move data into one of 16 subspaces depending on the constraint data. This ensures that obstacle avoidance is followed even if the bundled target direction and last move data is identical between two different obstacle scenarios.

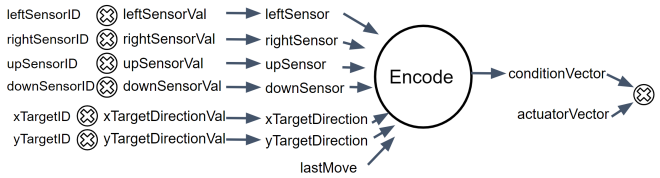


Figure 4: Sensor encoding overview.

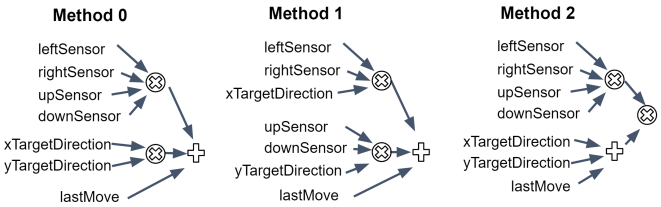


Figure 5: Sensor encoding methods.

V. Algorithm Optimization

Several parameters required optimization to yield maximum performance from the algorithm.

A. Repeated conditions

Recall of reactive behavior performance using HDC can be significantly degraded by inclusion of multiple vectors with a repeated condition vector but a different actuator vector. For example, assume a certain set of sensor inputs were trained corresponding to one actuation for one sample, while at a later time the same sensor inputs were trained while corresponding to a different actuation. Later during recall, when given the same condition vector, the output actuation vector will be somewhere in between both trained actuations, hence resulting in mis-recalls.

To avoid training repeated conditions, we implemented a condition memory vector which sums all previous condition vectors that have been previously trained. During training, if the current condition vector was closer than a threshold distance from the stored condition memory (calculated using Hamming distance), the current sample would not be added to the program memory. If it was dissimilar, it was added to program memory and the condition was added to the condition memory. This ensured that the program memory would associate each set of conditions with a valid actuator output. The threshold parameter is selected based on observed performance at successfully reaching the target.

B. Actuator value decoding

During the query/testing phase, a noisy actuator vector is recovered from the program vector via unbinding with the current condition vector. A typical method to recover the actuator value from this noisy vector is to compute the Hamming distance between the vector and each actuator vector, choosing the closest as the actuator value. However, one of the issues which reduced the success rate of the actor reaching the target was getting "stuck," during which the actor oscillated between two positions, often near a corner or obstacle. To prevent this, the actuator selection is implemented probabilistically so that

eventually the actor will make a different move to escape this condition. The ”closeness” for each actuator vector to the noisy vector is calculated using Hamming distance, scaled by a value, and put into a softmax function. The output of this function is used to probabilistically select the actuator value. This scaling parameter is selected based on observed performance at successfully reaching the target.

VI. Results

A. Similarity threshold tuning

Method	Success Rate
0	52%
1	70%
2	81%

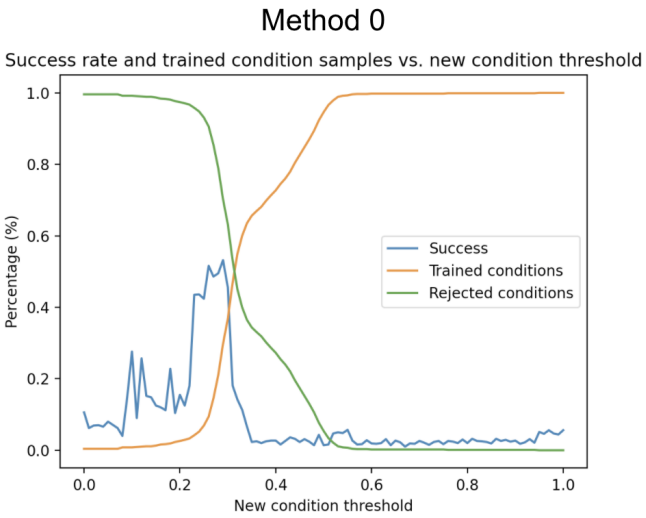
Table I: Success rate of each method prior to the additional of probabilistic output.

For each of the different encoding methods, a similarity threshold was selected by observing the performance of the algorithm across a range of thresholds. The threshold is a measure of similarity of vectors, thus too high would result in more and more of the 1001 samples of the training dataset being trained which would result in many repeat conditions, while too low would result in not enough vectors being trained leading to inability to recall correctly for many sensor combinations. Though there are elements of randomness in this algorithm, the optimal similarity threshold generally held across trials of the same implementation. The results are shown in Table I. One factor that was left unaccounted for in our success rate calculations was the inclusion of trials that were actually impossible to accomplish because either the robot or target was surrounded by obstacles. The number of cases where this was true was not measured, but these likely degraded the reported performance by a few percent based on observation and rough calculations. For each method, the effect of the thresholds on the number of crashes and hence the success rate - prior to the addition of the probabilistic output selection - are shown in Fig. 6.

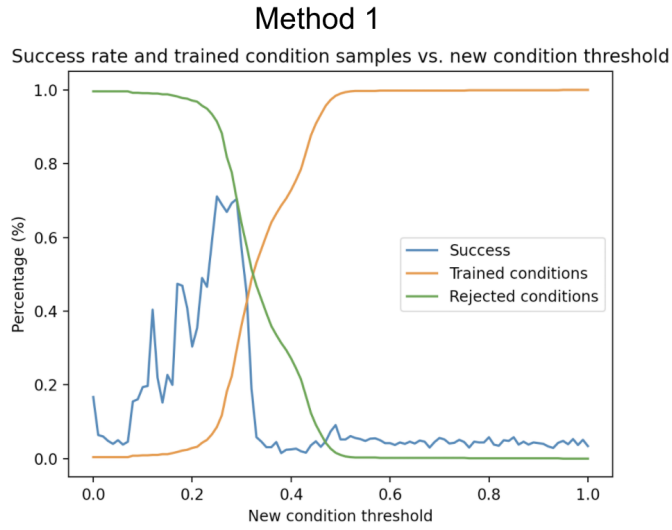
While the first two methods demonstrate peak performance when around $25\% - 30\%$ of the training dataset is trained (approx. $250 - 300$ samples), the final method only requires 0.08% of the training data (approx. 80 samples). The total number of possible conditions includes $2^4 = 16$ for just the obstacle sensors, $3^2 = 9$ for the directional sensors and 4 for the last output sensor which gives 576 possible conditions though some may be highly improbable if not impossible due to previous moves not leading to obstacles in certain places, etc. This demonstrates that the final method is accomplishing the high accuracy through learning beyond just memorization as 80 samples represent only around 14% of all the possible sensor combinations.

B. Probabilistic tuning

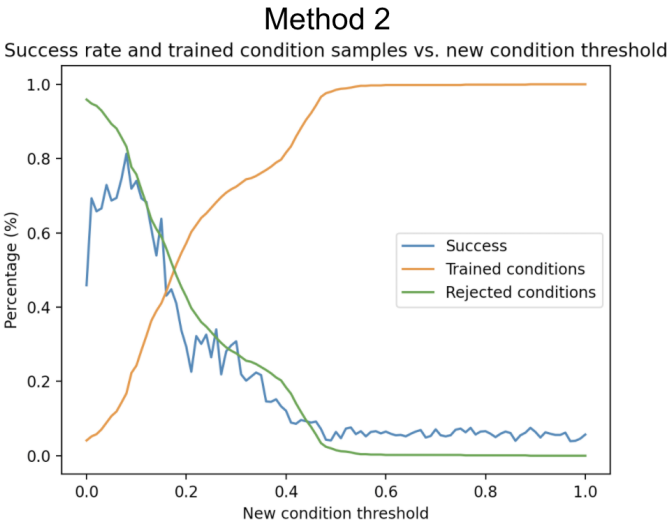
The probabilistic tuning was added to method 2 as a way to decrease the number of stuck occurrences. By adding a probabilistic outcome, the random element would break a repetitive sequence of actions allowing the actor to reach the goal. The



(a)



(b)



(c)

Figure 6: Success rate and trained/rejected training samples for a range of similarity thresholds with optimal values of (a) $thresh = 0.26$, for method 0 (b) $thresh = 0.29$, for method 1 (c) $thresh = 0.08$, for method 2.

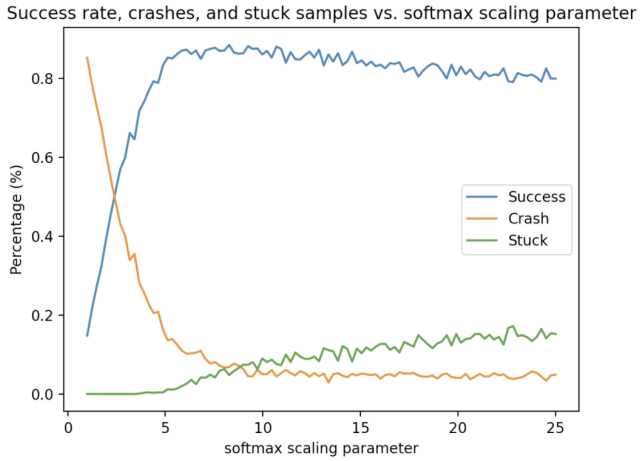
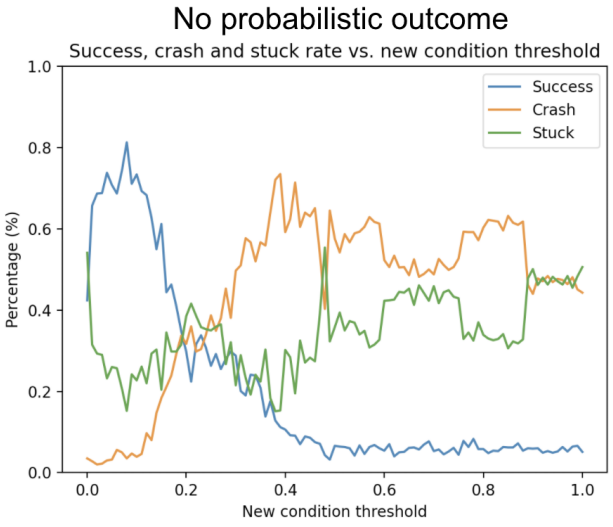
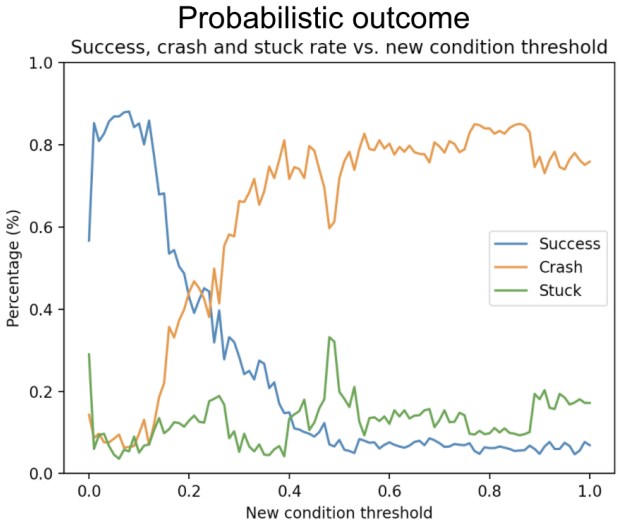


Figure 7: Success, crash and stuck rate for a range of probabilistic tuning parameter values



(a)



(b)

Figure 8: Success, crash and stuck rate for method 2 for (a) without a probabilistic outcome (b) with the probabilistic outcome

impact of the probabilistic tuning parameter on the success rate is shown in Fig. 7. An optimal value of 7.79 resulted in the highest success rate of 89%; this was an improvement of 8% from method 2 without the probabilistic outcome. The effect of the probabilistic outcome on the success rate as a result of changes in the number of stuck trials and crashed trials is shown in Fig. 8. The probabilistic output increased the accuracy because of the decrease in the percentage of stuck trials with minimal increase in the crashes, especially at the optimal new condition threshold of 0.06.

The softmax function was applied for the other two methods as well, however because of the lower performance of the base algorithm in those methods, a decrease in the number of stuck trials directly translated into an increase in crashes as opposed to reaching the goal. As a result, the overall success rate stayed the same despite the added random element at the output.

Overall, based on the results, it was clear that the optimal method for encoding was the constraints vs. goal method (method 2) that resulted in precise obstacle avoidance with the addition of goal-oriented information to prioritize obstacle-avoiding actions that would lead to the desired outcome. The initial findings demonstrated the need for an encoding scheme different to the previous work in [2] through binding obstacle sensor data instead of bundling due to the similarity of the various binary sensor combinations that could result in one sensor's change not impacting the overall bundled vector. This provided perfect obstacle avoidance behavior, but the addition of a goal was a new challenge that required additional sensor information.

While the directional encoding scheme (method 1) was an improvement over the initial one, it still demonstrated a high number of crashes which suggested that the obstacle sensor information was not being weighted heavily enough. With method 2, a change in a single sensor would be in enough to change the entire condition vector; this was critical to reducing the number of crashes. To then further reduce the number of stuck trials, the probabilistic softmax function was applied to the actuator hamming distances to add an element of randomness that would allow a break from an endless cycle of actions so that the actor could reach the target. This encoding scheme resulted in a final success rate of 89%.

VII. Discussion

In this work, we demonstrated the use of HD computing for recall of reactive behavior to allow for autonomous obstacle avoidance and navigation to a target in a 2D world. Several sensor encoding methods were tested and optimized, and the final method achieved a 89% success rate at successfully navigating the actor to the game target. The sensor encoding strategy used was a major determinant of success rate. If HD computing is to be used for more complex behavioral recall tasks with more sensor inputs, attention must continue to be given to the sensor encoding/fusion.

Future work should include comparing the performance of this method to other methods such as training a neural network to complete this task. Here, all training was performed using a single non-exhaustive dataset of a human user playing the

game; greater success may be possible with a dataset which includes all possible scenarios.

APPENDIX

A. Code Repository

Python code for the game world and HD computing is found in our code repository [3].

REFERENCES

[1] Kanerva and Pentti, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, Jun. 2009. DOI: 10.1007/s12559-009-9009-8.

[2] P. Neubert, S. Schubert, and P. Protzel, "Learning vector symbolic architectures for reactive robot behaviours," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'16) and the Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics*, 2016.

[3] Y. Kim, B. Benedict, and A. Menon, *Hd_recall_2d_nav*, https://github.com/youbin-kim/hd_recall_2d_nav, 2020.