NavHD: Low-Power Learning for Micro-Robotic Controls in the Wild

Chae Young Lee[1], Sara Achour[2], and Zerina Kapetanovic[3]

***Abstract*— Micro-robots are emerging as powerful tools for search-and-rescue, precision agriculture, and cooperative manipulation, where their small size and low cost offer advantages over larger robots. However, enabling autonomous navigation on these robots remains challenging due to severe hardware constraints, such as limited memory, energy, and computational power. We explore a brain-inspired learning paradigm called Hyperdimensional Computing (HDC) to equip a cheap, lightweight navigation model that runs onboard micro-robots. We present NavHD, which features an adaptive HD encoder that learns spatial representations and incorporates loss-based training for both imitation learning and off-policy reinforcement learning. Our hardware implementation of NavHD uses eight ultrasound sensors and is optimized to run on an ARM Cortex-M4 core, using only 10.2 kB of memory, 900 clock cycles and 1.1 mJ of energy per inference. Through experiments in both simulation and the real world, we demonstrate that NavHD outperforms DNN-based and prior HDC-based RL methods in obstacle avoidance by more than 2x the performance, while achieving 2-26x more superior resource efficiency.**

I. Introduction

Micro-robots are emerging as promising solutions for exploring environments that are inaccessible to larger robots and humans. These robots have the potential to search for survivors in disaster zones [1]–[3], navigate under canopy for precision agriculture [4], [5], and cooperatively move heavy objects in tight spaces [6], [7]. They are typically cheaper to manufacture than full-scale robots and can be valuable in field missions with the risk of losing robots or can be deployed in large quantities [8], [9]. Despite their potential, a major challenge in micro-robotics is enabling these robots to autonomously navigate dynamic environments by perceiving their surroundings and making real-time decisions. Conventional navigation systems, which are designed for larger robots, are impractical for micro-robots due to their stringent hardware constraints, including limited processing power, memory, and energy availability. High-power sensors such as LiDAR and high-resolution RGB cameras are infeasible, and micro-robots rely on simpler low-power sensors such as proximity sensors for obstacle avoidance or photodiodes for light source seeking [8], [10], [11].

Moreover, in larger robots, deep neural networks (DNNs) are widely used to process sensory inputs and have shown remarkable success in tasks such as obstacle avoidance, path planning, and end-to-end navigation [12], [13]. However, these networks typically require substantial computational resources, and adapting DNNs for resource-limited hardware is an active area of research. Efforts include model compression and quantization [14], [15], neural architecture search [16], [17], and the development of low-power neural accelerators [18], [19]. Despite these advancements, DNNs inherently trade off performance with model size, and their reliance on frequent memory accesses and multiply-accumulate operations results in computational inefficiencies on embedded hardware. In this work, we propose an alternative approach using neuromorphic computing, which offers promising results in resource efficiency and performance.

Hyperdimensional Computing (HDC) is a brain-inspired computing paradigm that represents information using high-dimensional vectors, known as hypervectors. HDC uses lightweight, algebraic operations such as XOR, majority voting, and bit shifting to encode real-world data into hypervectors. Prior works on HDC-based machine learning have achieved accuracies comparable to DNNs in lightweight classification tasks while offering orders-of-magnitude improvements in speed, memory efficiency, and energy consumption [20]–[22]. Additionally, HDC algorithms are highly parallelizable and can be aggressively optimized, with implementations on hardware accelerators achieving significant performance gains [23], [24].

Recently, researchers have explored integrating reinforcement learning (RL) with HDC. RL is a machine learning framework that teaches an agent to make decisions through trial-and-error. Using both data from the real world and the simulation, an RL agent is able to learn common navigation tasks by rewarding favorable behaviors. One such effort, QHD [25], adapts off-policy RL to HDC by introducing an HD encoder that maps state vectors into hyperspace and an HD-based regression model to predict Q-values from encoded state hypervectors. ReactHD [26] builds on QHD to train sensorimotor controls of wheeled robots to learn meaningful navigation tasks in the wild using LiDAR data. However, existing HD-based RL approaches are not optimized for resource efficiency, and they rely on sensors and computational resources that are unavailable to micro-robots.

In this work, we extend these efforts by developing a low-power HDC-based navigation model for micro-robots called NavHD. We introduce an HD encoder that captures spatial information from arbitrary sensor positions and ranges, draw inspiration from state-of-the-art HD classifiers to enhance loss-based training, and optimize computations for deployment on resource-constrained hardware. Our system features eight low-power ultrasound sensors to perceive dynamic environments and runs the HD-based classifier on an ARM

[1]Department of Computer Science, Stanford University, Stanford, CA, USA chae@stanford.edu

[2]Department of Computer Science & Electrical Engineering, Stanford University, Stanford, CA, USA sachour@stanford.edu

[3]Department of Electrical Engineering, Stanford University, Stanford, CA, USA zerina@stanford.edu

TABLE I: Feature comparison of HD-based classifiers.

	RL Integration	Gradient Update		Value Encoding	Arbitrary Sensor Position
		Class HVs	Encoder		
OnlineHD [27]	✗	✗	✗	HD Compute	N/A
LeHDC [21]	✗	✓	✗	HD Compute	N/A
LDC [22]	✗	✓	✓	Neural Network	N/A
QHD [25]	✓	✗	✗	HD Compute	✗
ReactHD [26]	✓	✗	✗	Trig. Activation	✗
Ours	✓	✓	✓	Trig. Activation	✓

Cortex-M4 microcontroller. NavHD's navigation model can be trained from imitating human behaviors or learning in simulation from scratch. From experiments, we demonstrate that our approach outperforms both DNN and previous HD methods, achieving over twice the performance in Q-learning training within simulated environments and zero-shot real-world navigation tasks. Additionally, NavHD achieves the highest resource efficiency, requiring only 10.2 kB for model storage, 900 clock cycles per inference, and consuming just 1.1 mJ of energy per frame. Compared to the baseline DNN, NavHD is 26x smaller, 3x faster, and 3x more energy-efficient.

Contributions. We make the following contributions:

- **NavHD.** We introduce NavHD, a differentiable HDC-based sense-actuate model designed for training on common navigation tasks.
- **Q-learning setup for NavHD.** We integrate NavHD with Q-learning and demonstrate how Q-value updates drive model learning. We experiment with various hyperparameters, including different loss functions and reward structures, and report the best-performing configurations.
- **Hardware implementation of NavHD and baseline.** We optimize computational efficiency and memory usage across all models to ensure they can be evaluated on resource-constrained hardware with a single ARM Cortex M4 core and 512 kilobytes of flash memory.

II. Background and Related Work

A. Micro-Robotics & Navigation

There has been extensive research in developing low-power micro-robots. In [8], [11], [28], [29], robotics platforms operate at very low power, requiring only a small battery or operating solely on harvested energy. For example, MilliMobile is a 10 x 10 mm mobile robot that harvests solar energy and uses light-following to trigger locomotion. In all cases, low-power microcontrollers are used, limiting the device memory to <256kB RAM and <1MB Flash. Beyond hardware development, several efforts have focused on energy-efficient navigation techniques for resource-constrained robotics. For example, in [10], acoustic sensors are used to perform ultra-low-power spatial sensing, but their range is limited to 20 cm. In [30], a sensor network uses stationary sensor nodes as signposts for mobile robots to navigate. In [31], a deep RL policy was developed to run onboard nano quadcopters, enabling autonomous source-seeking of light. The system achieved a 94% success rate in simple and complex environments, while using only 20.5 kB of RAM.

B. Hyperdimensional Computing

Hyperdimensional Computing (HDC), or Vector Symbolic Architecture (VSA), is a brain-inspired computing paradigm that has been used for lightweight classification and control tasks. The basic datatype for HDC is a hypervector, where information is encoded in distances between hypervectors.

We focus on MAP-HDC [32], a version of HDC that works with n-dimensional real-valued hypervectors in $\mathbb{R}^n$ with elements within the range of $[-1, 1]$ and uses the dot-product similarity $dist(h, h') = h \cdot h'$ as the distance measure. In MAP-HDC, binding ($\otimes$) is multiplication and yields dissimilar hypervectors, and bundling ($\oplus$) is normalized addition and yields similar hypervectors.

In HD classification and prediction, the HD classifier first encodes classic data as a state hypervector using a process called *encoding*, and then compares the state hypervector to a set of class hypervectors using the distance function. The closest class hypervector is the winning class. HD controllers operate similarly, with the exception that class hypervectors map to actions a controller might take.

Prior Work. Table I summarizes the differences between HDC classifiers (rows 1-3) and controllers (rows 4-6) from prior research. OnlineHD [27] is an early work that uses an iterative training algorithm to refine the HD classifier's class hypervectors. LeHDC [21] improves on OnlineHD by using gradient-based methods to learn class hypervectors, but does not use a learned encoder. LDC [22] uses gradient-based methods to learn both the encoder and the class hypervectors, and achieves the highest classification accuracies of the three methods. ReactHD [26] and QHD [25] are both HDC-based robotic controller architectures that are trained with Q-learning and are therefore integrable with reinforcement learning frameworks. However, they use OnlineHD's training algorithm instead of more effective gradient-based methods. *We present an HD controller architecture and reinforcement learning algorithm that adapts the gradient-based training methods from LDC to HD controllers.*

Another key difference across HDC models is in how numeric values and sensor positions are encoded. This is especially relevant for robotics applications that work with numeric sensor data; poor encodings impact model performance. OnlineHD, LeHDC, and QHD all use encoders composed of HD operators to translate numeric data to hypervectors. LDC uses a neural network to translate values to hyper-

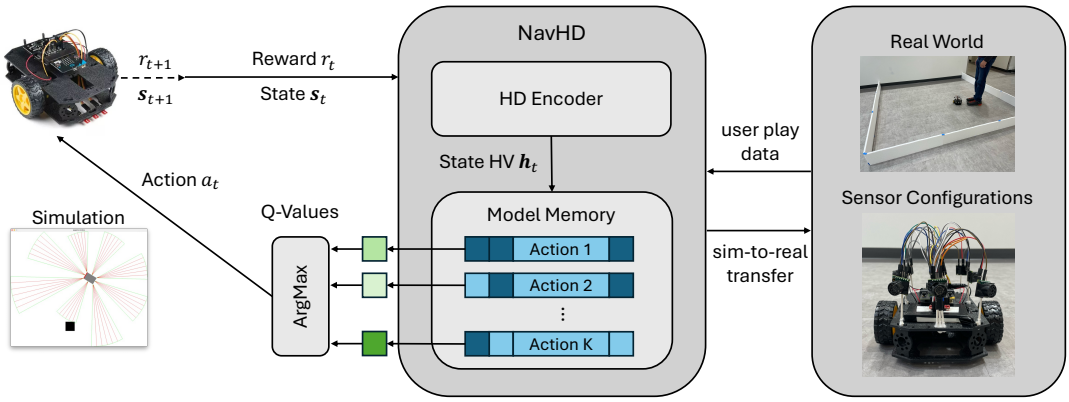


Fig. 1: NavHD Overview. The model is composed of an HD encoder and a model memory for predicting action classes or Q-Values. After training, the model knowledge can be transferred to the real world without finetuning.

vectors, which improves accuracy but significantly increases resource usage. ReactHD uses a trigonometric activation function ($\cos(\cdot)$) as its encoder. Our method uses a more sophisticated trigonometric activation function ($\sin(\cdot)\cos(\cdot)$) in its encoder. The same activation function is used in one benchmark implementation in the OnlineHD codebase, but is not discussed in their paper. Second, ReactHD and QHD require sensor positions to be explicitly encoded into the hypervector. In contrast, our method learns sensor positions.

III. Method

This paper presents NavHD, a lightweight HDC-based sensorimotor control framework for micro-robotic navigation. As shown in Figure 1, NavHD first encodes the sensor input $\mathbf{s}_t$ into a hypervector $\mathbf{h}_t$ using a HD encoder. The HD trainer uses imitation learning and off-policy reinforcement learning to train the model memory, which has k number of action classes. During inference, the encoded sensor input, called the state hypervector $\mathbf{h}_t$, is queried against the action hypervectors to find the winning action class. The robot acts on the decision to perform navigation tasks such as object avoidance. Section III-A describes NavHD's model design, including the HD encoder and the classifier. Section III-B describes NavHD's training algorithm, which uses both imitation learning and off-policy RL.

A. NavHD Model Design

HD Encoder. NavHD's encoder maps the real-world sensor data into hyperspace. At time t, the robot observes the world using d proximity sensors placed in arbitrary directions. This sensor input, normalized to the range $[0, 1]$, is denoted as $\mathbf{s}_t = \langle s_t^1, ..., s_t^d \rangle$. The encoding function $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ maps a sensor input into a n-dimensional hypervector $\mathbf{h}_t$. The encoding function works with a learned scaling matrix $\mathbf{B} \in \mathbb{R}^{n \times d}$ and a learned offset vector $\mathbf{b} \in \mathbb{R}^n$ that scale and shift the sensed values. The final encoded hypervector $\mathbf{h}_t$ is then computed using trigonometric activation functions:

$$\mathbf{h}_t = \phi(\mathbf{s}_t) = \cos(\mathbf{B} \cdot \mathbf{s}_t + \mathbf{b}) \odot \sin(\mathbf{B} \cdot \mathbf{s}_t), \qquad (1)$$

where $\odot$ is binding implemented as element-wise multiplication. In the above encoding, $\mathbf{B}$, $\mathbf{b}$ are used to linearly project the d sensor values to an n-element vector, which is then passed through a trigonometric activation function. At the start of training, the elements of the scaling matrix $\mathbf{B}$ are initialized from a normal distribution $N(0, \sigma^2)$, while each element of the offset vector $\mathbf{b}$ is drawn from a uniform distribution over $[0, 2\pi]$. Notice that the trigonometric activation function not only introduces nonlinearity but also bounds the vector elements within $[-1, 1]$. The encoded hypervectors are therefore valid hypervectors under MAP-HDC.

The resulting hypervector is a distance-based information representation, where the similarity between sensor values is estimated with the dot product between their representative hypervectors.

Action Prediction. Once the input vector $\mathbf{s}_t$ is encoded to a hypervector $\mathbf{h}_t$, that hypervector is used as a query against the model memory for predicting the next action. The model memory $\mathbf{C} = \langle \mathbf{c}_1, ..., \mathbf{c}_k \rangle$ consists of k action hypervectors, where k denotes the number of actions that may be taken. Each action hypervector is initialized using Kaiming uniform initialization with a negative slope parameter $\sqrt{5}$. The similarity of the query hypervector and an action hypervector is computed using a dot product:

$$\text{sim}(\mathbf{h}_t, \mathbf{c}_k) = \mathbf{h}_t^T \cdot \mathbf{c}_k. \qquad (2)$$

The next action a_t is then determined by selecting the action k with the highest similarity to the encoded hypervector:

$$a_t = \text{argmax}_k \text{sim}(\mathbf{h}_t, \mathbf{c}_k). \qquad (3)$$

Differentiability. NavHD is designed to be fully differentiable, including the encoder and the similarity function, and thus can be trained using backpropagation to update model parameters based on a loss function. Unlike conventional HD classifiers, which typically update only class hypervectors [20], [21], [27], our approach jointly optimizes the scale matrix, offset vector, and class hypervectors. After training, the scale vector $\mathbf{b}_i$, the i-th column vector of $\mathbf{B}$, is expected to capture the spatial position and sensing range of sensor i, allowing flexibility for arbitrary sensor configurations. Similarly, a well-trained class hypervector effectively represents the data distribution of its respective class, ensuring that queries from the same class yield the highest similarity scores during inference.

Exchangeability. Encoder parameters $\mathbf{B}$, $\mathbf{b}$ are independently and identically distributed (i.i.d.) random variables at initialization. The loss function is defined based on hypervector distances, which are symmetric with respect to element indices, and gradient-based training preserves this symmetry throughout optimization. As established in [33], these properties ensure that NavHD satisfies exchangeability. This exchangeability preserves key HDC properties, such as fully-distributedness, which ensures that information is evenly encoded across all hypervector dimensions, and error-resiliency, which enhances robustness to minor perturbations in sensor readings [22]. We anticipate NavHD will be amenable to statistical early termination-based dynamic optimization to further reduce resource usage [33].

B. NavHD Training Algorithms

NavHD can be trained using two methods, supervised training using user play data or off-policy RL in simulation. These methods can be applied independently or in sequence to enhance performance. Supervised training, or imitation learning, leverages user play data to train the model and is effective when a large, high-quality dataset is available. In contrast, off-policy RL enables the model to generalize by exposing it to various situations, including edge cases that may not have been encountered in user-collected data.

Q-learning Algorithm. NavHD can be incorporated into an off-policy RL framework using Q-learning. We use ideas from Deep Q-Networks (DQN) [34], which use a DNN to approximate Q-values. Instead, we use NavHD to estimate Q-values by using the output of the similarity function from Equation 2. Given an input $\mathbf{s}_t$ and an action a at time t, the Q-value is computed as

$$Q(\mathbf{s}_t, a) = \text{sim}(\phi(\mathbf{s}_t), \mathbf{c}_a). \tag{4}$$

Using the Bellman equation [34], the target Q-value is computed as:

$$y_t = r_t + \gamma(1 - \text{done}_t) \max_{a'} Q'(\mathbf{s}_{t+1}, a'), \tag{5}$$

where r_t is the reward received at time t, γ is the discount factor that controls the importance of future rewards, and done_t is an indicator variable that is one if the episode terminates and zero otherwise. The target Q-value y_t is computed using a separate target model Q' to improve training stability. To optimize the model, we minimize the Huber loss.

$$l_t = \text{HuberLoss}(Q(\mathbf{s}_t, a_t; \mathbf{C}, \mathbf{B}, \mathbf{c}), y_t). \tag{6}$$

For efficient training, NavHD employs experience replay [34], where a buffer of past experiences is maintained and randomly sampled to reduce correlations between consecutive updates. This method ensures a more stable learning process. Additionally, NavHD integrates Double DQN [35] to mitigate overestimation bias. Instead of selecting the action that maximizes the next-state Q-value directly, Double DQN uses the main model to choose the best action and the target model to evaluate its value. This method helps reduce overestimation bias in Q-value updates.

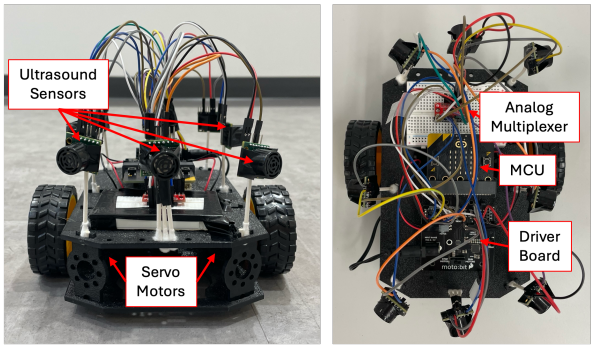
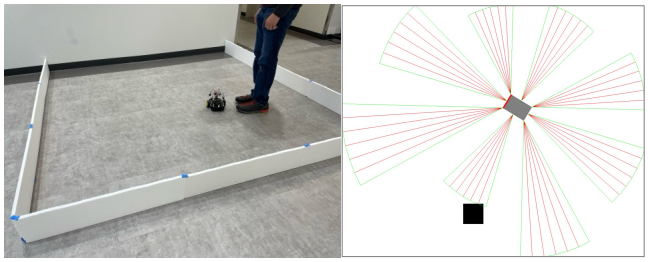


Fig. 2: SparkFun micro:bot kit used for real-world evaluation.



(a) Real World (b) Simulation

Fig. 3: Model training and evaluation environments.

IV. Implementation

A. Hardware

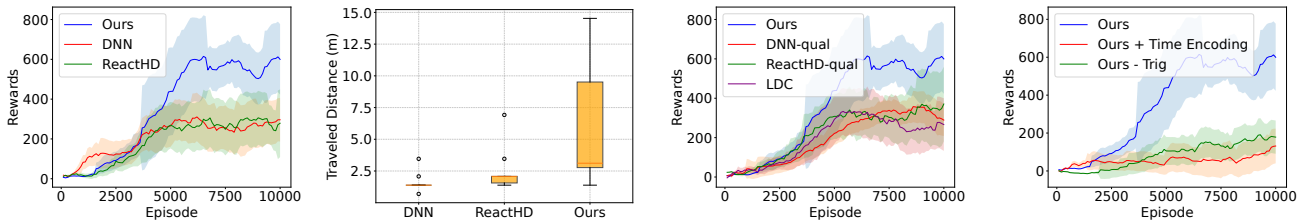
SparkFun micro:bot kit was configured to demonstrate NavHD. The robot uses the micro:bit v2 board, which runs the nRF52833 SoC. This microprocessor hosts a single ARM Cortex M4 core that runs up to 64 MHz clock and has 512 kB of flash memory and 128 kB of RAM. The board consumes about 39 mW at 3V running our firmware. We equip the robot with eight MaxBotix MB1010 LV-MaxSonar-EZ1 ultrasound sensors, placed on four corners and four sides of the robot. Each sensor consumes about 5 mW and has a range of over 6 meters. The ultrasound sensors have a cone-shaped beam pattern and have larger coverage than time-of-flight sensors or lasers used in LiDAR. However, the micro:bit v2 board does not support eight analog inputs, and hence a 74HC4051 multiplexer is used to poll sensor readings at 5 Hz. The system's power consumption is shown in Table II.

Component	Idle (mW)	Full Throttle (mW)
Micro:bit v2	39.0	39.0
Carrier Board	43.9	48.4
Motors	0.0	2307.0
Ultrasound Sensors	40.0	40.0
Multiplexer	0.2	0.2
Total	**123.1**	**2434.6**

TABLE II: System power consumption.

B. Firmware

Porting the perception-to-action classifiers onto the hardware is a non-trival task due to the severe resource constraints of the nRF52833 microcontroller, specifically on model



(a) Hardware-Optimized Models (b) Real World Experiments (c) CPU-Only Models (d) Ablation Study

Fig. 4: NavHD and baseline models are evaluated on the simulation and the real world.

storage, latency, and memory allocation. We use Zephyr, a real-time operating system designed for microcontrollers, to control hardware modules. The microcontroller reads sensor data via an analog-to-digital converter, interfaced with a multiplexer to cycle through multiple ultrasound sensors. The motors are controlled using the I2C interface. Inference is performed using TensorFlow Lite Micro and CMSIS-NN, both of which accelerate inference-time computations. To further enhance efficiency, expensive trigonometric functions are replaced with precomputed lookup tables, eliminating costly floating-point operations. Additionally, we minimize memory footprint by packing binary data into byte-aligned structures and eliminating dynamic memory allocation.

C. Simulation

We built a simulation environment using Gymnasium [36] and Pygame [37] that can run on standard laptops and GPU servers. As shown in Figure 3b, the simulation is designed to mimic the interaction of mobile robots. The agent follows the differential drive model using two independently controlled wheels on either side, and its direction of movement is determined by the velocity difference between these wheels. Given the left and right wheel velocities, v_L and v_R respectively, the agent operates using a discrete action space consisting of four motion primitives: stop ($v_L = v_R = 0$), forward ($v_L = v_R > 0$), forward left ($v_L < v_R$), and forward right ($v_L > v_R$). Furthermore, the agent observes the world using eight virtual ultrasound sensors. Each sensor emits a conical beam and returns the distance of the nearest obstacle or wall within its detection range.

V. Results

We evaluated NavHD's learning quality and performance in both simulation and real-world experiments.

A. Experimental Setup and Baselines

We evaluated NavHD against a deep neural network (DNN) and ReactHD [26], an existing HD-based RL model. All models were implemented using PyTorch 2.4.1, with architectures optimized to fit the constraints of the target embedded hardware. The DNN baseline consists of a four-layer network with hidden sizes of 256, 128, and 64, incorporating batch normalization and dropout between layers. Each layer uses the Leaky ReLU activation function to improve stability. The ReactHD baseline operates with hypervectors of length 5000, following the original implementation. Training details are summarized in Table III. We refer to the model's `qual` variant as the one achieving the highest test accuracy or reward, and the `opt` variant as the one optimized for minimal resource usage.

TABLE III: Hyperparameters used for training.

Hyperparameter	Value
Discount factor (γ)	0.99
Learning rate (α)	0.001
Exploration rate (ϵ)	1.0
Epsilon decay	0.999
Minimum epsilon	0.1
Batch size	64
Target model update interval	50
Hypervector length (n)	256

We evaluated all models on an object avoidance task, where the agent must navigate without colliding with static walls or dynamic objects that chase the agent. Higher rewards and longer travel distances indicate better learning performance. The reward at each timestep is computed as $r_t = \alpha - 100\beta$, where α is a binary variable that takes the value 1 if the agent remains at a safe distance from obstacles, and 0 otherwise. Similarly, β is a binary variable set to 1 if the agent collides with an object or goes out of bounds, and 0 otherwise. When a collision occurs, the agent receives a significant penalty of -100.

Imitation learning was independently evaluated by collecting user play data. The robot was manually controlled in an 8.76 ft × 8.76 ft enclosed environment with smooth flooring, as shown in Figure 3a. A human subject acted as a moving object, performing a random walk for 3 minutes and chasing the robot with sporadic pauses for another 3 minutes. This dataset was used to train NavHD and all baseline models. Training was performed on a MacBook Pro, using an 8:2 train-test split, and continued until convergence.

B. NavHD RL Model Evaluation

Simulation. All models were trained for 10000 episodes from scratch, ensuring full convergence before stopping

TABLE IV: Resource usage on the target hardware.

Model	Memory (kB)	Clock Cycle (k)	Energy (mJ)
DNN	263.8	2.9	3.4
ReactHD	19.5	8.4	9.9
NavHD	**10.2**	**0.9**	**1.1**

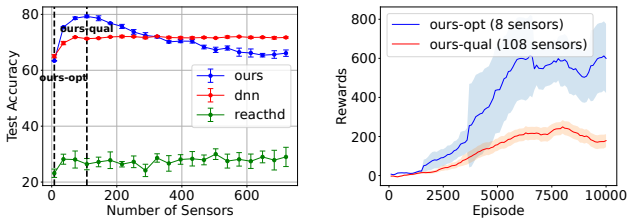
training. During training, models were evaluated every 100 episodes by running 10 test trials without learning updates, recording the mean and standard deviation of rewards per episode to track progress. Figure 4a shows that while all models initially improve, NavHD quickly surpasses the baselines after approximately 3000 episodes and continues to improve until about 6000 episodes. In contrast, the baseline models both converge around 5000 episodes. Ultimately, NavHD achieves nearly twice the reward values of DNN and ReactHD.

Real World. These trained models were then ported to the hardware and tested in the real world without finetuning. As shown in Figure 4b, the total traveled distance is measured before the agent collides against a wall or the chasing object in 10 trial runs. NavHD outperforms both DNN and ReactHD in object avoidance, as indicated by the significantly higher total traveled distance. Specifically, NavHD has multiple trials reaching more than 6 meters, while DNN and ReactHD struggle to exceed 3 meters in most cases. DNN performs poorly and inconsistently, often colliding within 0.5-1.5 meters of movement, indicating that its learned policies do not transfer well to real-world environments zero-shot and are sensitive to sensor configurations and noise.

Resource Usage. As shown in Table IV, NavHD achieves the best resource usage in terms of memory consumption, computational cost, and energy efficiency. NavHD uses 10.2 kB of flash memory, which is 2x smaller than ReactHD (19.5 kB) and 26x smaller than DNN (263.8 kB). In terms of computational efficiency, NavHD achieves the lowest clock cycle count (0.9k), making it 3x faster than DNN (2.9k) and 9x faster than ReactHD (8.4k). Although ReactHD is more lightweight than a DNN, it uses more clock cycles to compute 5000-length hypervectors. Furthermore, the bit-packing technique used to minimize memory footprint introduces additional unpacking overhead during encoding. This increased inference time leads to ReactHD having the highest energy consumption (9.9 mJ), whereas NavHD is 9× more energy-efficient, requiring only 1.1 mJ per inference.

CPU-Only Experiments. As shown in Figure 4c, models that did not fit on the embedded hardware were evaluated on the CPU. `DNN-qual` uses an additional hidden layer of size 512 and `ReactHD-qual` uses hypervectors of length 10000, which is double the version that fits on the hardware. LDC, the state-of-the-art HD classifier, was also evaluated in Q-learning. Since LDC is not designed to work with Q-learning, LDC uses the NavHD's Q-learning framework and training method. Experimental results in terms of rewards per episode show that NavHD outperforms all baseline models by a significant margin. Specifically, each `qual` version outperforms the hardware-optimized version by a small margin, but struggles to accumulate rewards over 400.

Encoder Ablation Study. Figure 4d presents the performance of NavHD without the trigonometric activation function (green), and with a permutation-based encoder that also captures the history of sensor values (red). We find that both models perform poorly in simulation, converging at lower rewards than those of other baselines. These results show that the trigonometric activation significantly helps the performance and that explicit encoding of the past state vectors can be detrimental.



(a) Imitation Learning (b) Q-Learning

Fig. 5: Sensor count affecting NavHD performance.

C. Sensor Count and NavHD Performance

We further evaluate NavHD across different sensor counts. Figure 5a shows test accuracy as the number of sensors increases. While DNN maintains stable performance, NavHD achieves its highest accuracy at 108 sensors (`ours-qual`), outperforming both baselines. Notably, at higher sensor counts, NavHD's accuracy declines. The encoding function averages out sensor readings and reduces NavHD's sensitivity to fine-grained variations. In contrast, at the minimum sensor count of 8 (`ours-opt`), NavHD maintains strong accuracy (63.39%).

Figure 5b compares the RL performance of `ours-qual` (108 sensors) and `ours-opt` (8 sensors). `ours-opt` consistently achieves higher rewards, showing that despite using fewer sensors, it learns more efficiently in simulation. Additionally, its larger variance suggests more exploration before stabilizing at higher rewards. Meanwhile, `ours-qual` improves steadily but converges to a lower final reward. These results reinforce that NavHD can learn effectively with minimal sensors, making it an ideal choice for resource-constrained robotic systems.

VI. Conclusion

We present NavHD, an HD-based RL framework optimized for resource-constrained micro-robotic navigation. By using adaptive HD encoding and loss-based training methods, NavHD outperforms both DNN and prior HD-based RL models, achieving over 2x the performance in Q-learning and zero-shot real-world object avoidance. Our results demonstrate that NavHD achieves optimal resource efficiency, using 26x less memory and 3x less energy than a DNN. While it performs well on object avoidance, future work includes extending evaluation to more complex tasks such as path planning. Additionally, while the trigonometric encoder is expressive and fully differentiable, it can be computationally expensive on microcontrollers without floating-point support. This motivates the need for more efficient encoding alternatives. Finally, since HD models use the same lightweight operations for both training and inference, NavHD can be explored for on-device learning.

REFERENCES

[1] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[2] P. T. Tran-Ngoc, D. L. Le, B. S. Chong, H. D. Nguyen, V. T. Dung, F. Cao, Y. Li, K. Kai, J. H. Gan, T. T. Vo-Doan *et al.*, "Intelligent insect–computer hybrid robot: Installing innate obstacle negotiation and onboard human detection onto cyborg insect," *Advanced Intelligent Systems*, vol. 5, no. 5, 2023.

[3] R. D. Arnold, H. Yamaguchi, and T. Tanaka, "Search and rescue with autonomous flying robots through behavior-based cooperative intelligence," *Journal of International Humanitarian Action*, vol. 3, no. 1, pp. 1–18, 2018.

[4] R. Maria-Hormigos, C. C. Mayorga-Martinez, and M. Pumera, "Magnetic hydrogel microrobots as insecticide carriers for in vivo insect pest control in plants," *Small*, vol. 19, no. 51, p. 2204887, 2023.

[5] A. N. Sivakumar, S. Modi, M. V. Gasparino, C. Ellis, A. E. Baquero Velasquez, G. Chowdhary, and S. Gupta, "Learned Visual Navigation for Under-Canopy Agricultural Robots," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[6] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[7] X. Dong and M. Sitti, "Controlling two-dimensional collective formation and cooperative behavior of magnetic microrobot swarms," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 617–638, 2020.

[8] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 3293–3298.

[9] Z. Liu, W. Zhan, X. Liu, Y. Zhu, M. Qi, J. Leng, L. Wei, S. Han, X. Wu, and X. Yan, "A wireless controlled robotic insect with ultrafast untethered running speeds," *Nature Communications*, vol. 15, no. 1, p. 3815, 2024.

[10] Y. Bai, N. Garg, and N. Roy, "Spidr: Ultra-low-power acoustic spatial sensing for micro-robot navigation," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 99–113.

[11] K. Johnson, Z. Englhardt, V. Arroyos, D. Yin, S. Patel, and V. Iyer, "Millimobile: An autonomous battery-free wireless microrobot," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16.

[12] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, vol. 46, no. 5, pp. 569–597, 2022.

[13] W. Yu, J. Peng, Q. Qiu, H. Wang, L. Zhang, and J. Ji, "Pathrl: An end-to-end path generation method for collision avoidance via deep reinforcement learning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 9278–9284.

[14] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 5151–5159.

[15] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1712.05877*, 2017.

[16] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[17] C. R. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *CoRR*, vol. abs/2010.11267, 2020. [Online]. Available: https://arxiv.org/abs/2010.11267

[18] K. Choi and G. E. Sobelman, "An efficient cnn accelerator for low-cost edge systems," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 4, pp. 1–20, Aug. 2022. [Online]. Available: https://doi.org/10.1145/3539224

[19] A. Moss, H. Lee, L. Xun, C. Min, F. Kawsar, and A. Montanari, "Ultra-low power dnn accelerators for iot: Resource characterization of the max78000," in *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, 2022, pp. 934–940.

[20] C. Y. Lee, M. Fite, T. Rao, P. L. Yi, S. Achour, and Z. Kapetanovic, "Hypercam: Low-power onboard computer vision for iot cameras," 2024.

[21] S. Duan, Y. Liu, S. Ren, and X. Xu, "Lehdc: learning-based hyperdimensional computing classifier," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1111–1116. [Online]. Available: https://doi.org/10.1145/3489517.3530593

[22] S. Duan, X. Xu, and S. Ren, "A brain-inspired low-dimensional computing classifier for inference on tiny devices," 2022. [Online]. Available: https://arxiv.org/abs/2203.04894

[23] C. Liu, K. Wu, H. Liu, H. Jin, X. Liao, Z. Duan, J. Xu, H. Li, Y. Zhang, and J. Yang, "A reram-based processing-in-memory architecture for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[24] A. Menon, D. Sun, S. Sabouri, K. Lee, M. Aristio, H. Liew, and J. M. Rabaey, "A highly energy-efficient hyperdimensional computing processor for biosignal classification," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, no. 4, pp. 524–534, 2022.

[25] Y. Ni, D. Abraham, M. Issa, Y. Kim, P. Mercati, and M. Imani, "Efficient off-policy reinforcement learning via brain-inspired computing," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, 2023, pp. 449–453.

[26] H. Kwon, K. Kim, J. Lee, H. Lee, J. Kim, J. Kim, T. Kim, Y. Kim, Y. Ni, M. Imani *et al.*, "Brain-inspired hyperdimensional computing in the wild: Lightweight symbolic learning for sensorimotor controls of wheeled robots," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 5176–5182.

[27] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 56–61.

[28] K. Johnson, V. Arroyos, R. Villanueva, A. Schulz, S. Fuller, and V. Iyer, "Toward sub-gram helicopters: Designing a miniaturized flybar for passive stability," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 2701–2708.

[29] Y. Lai, C. Zang, G. Luo, S. Xu, R. Bo, J. Zhao, Y. Yang, T. Jin, Y. Lan, Y. Wang *et al.*, "An agile multimodal microrobot with architected passively morphing wheels," *Science Advances*, vol. 10, no. 51, p. eadp1176, 2024.

[30] M. Batalin, G. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, 2004, pp. 636–641 Vol.1.

[31] B. P. Duisterhof, S. Krishnan, J. J. Cruz, C. R. Banbury, W. Fu, A. Faust, G. C. de Croon, and V. J. Reddi, "Tiny robot learning (tinyrl) for source seeking on a nano quadcopter," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 7242–7248.

[32] R. W. Gayler, "Multiplicative binding, representation operators & analogy (workshop poster)," 1998.

[33] P. Yi, Y. Yang, C. Y. Lee, and S. Achour, "Early termination for hyperdimensional computing using inferential statistics," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 342–360.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[35] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[36] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[37] P. Community, "Pygame: Python game development library," https://www.pygame.org/, 2000.