EDEN: Entorhinal Driven Egocentric Navigation Toward Robotic Deployment

Mikolaj Walczak*, Romina Aalishah*, Wyatt Mackey†, Brittany Story†, David L. Boothe Jr.,†, Nicholas Waytowich†, Xiaomin Lin*, Tinoosh Mohsenin*
*Johns Hopkins Whiting School of Engineering, Baltimore, Maryland, United States
Emails: {mwalcza1, raalish1, xlin52, tinoosh}@jhu.edu
†DEVCOM United States Army Research Laboratory, Aberdeen, Maryland, United States
Emails: {wyatt.t.mackey, brittany.m.story, david.l.boothe7, nicholas.r.waytowich}.civ@army.mil

***Abstract*—Deep reinforcement learning agents are often fragile while humans remain adaptive and flexible to varying scenarios. To bridge this gap, we present EDEN, a biologically inspired navigation framework that integrates learned entorhinal-like grid cell representations and reinforcement learning to enable autonomous navigation. Inspired by the mammalian entorhinal-hippocampal system, EDEN allows agents to perform path integration and vector-based navigation using visual and motion sensor data. At the core of EDEN is a grid cell encoder that transforms egocentric motion into periodic spatial codes, producing low-dimensional, interpretable embeddings of position. To generate these activations from raw sensory input, we combine fiducial marker detections in the lightweight MiniWorld simulator and DINO-based visual features in the high-fidelity Gazebo simulator. These spatial representations serve as input to a policy trained with Proximal Policy Optimization (PPO), enabling dynamic, goal-directed navigation. We evaluate EDEN in both MiniWorld, for rapid prototyping, and Gazebo, which offers realistic physics and perception noise. Compared to baseline agents using raw state inputs (e.g., position, velocity) or standard convolutional image encoders, EDEN achieves a 99% success rate, within the simple scenarios, and >94% within complex floorplans with occluded paths with more efficient and reliable step-wise navigation. In addition, as a replacement of ground truth activations, we present a trainable Grid Cell encoder enabling the development of periodic grid-like patterns from vision and motion sensor data, emulating the development of such patterns within biological mammals. This work represents a step toward biologically grounded spatial intelligence in robotics, bridging neural navigation principles with reinforcement learning for scalable deployment. A publicly available GitHub repository for EDEN is made available at `github.com/M-iki/EDEN`.**

I. Introduction

Accurate localization and efficient path planning are fundamental components of any autonomous navigation system. Localization determines the position and orientation of the agent relative to a global reference frame, while path planning generates collision-free trajectories toward target positions. Without reliable estimates of position and direction, a robot cannot avoid obstacles, reach its target, and adapt when the scenario changes. Small errors in estimating position or direction can accumulate over time and lead to task failure.

To address these challenges, most robotic systems rely on Simultaneous Localization and Mapping (SLAM) or visual odometry, which estimate the position of the agent by building a map of the scenario and updating it using sensor inputs such as LiDAR, depth, or camera frames. Once the map is available, motion planners compute collision-free paths within it. Reinforcement learning (RL) has emerged as an alternative approach, allowing agents to learn navigation behaviors through direct interaction with their surroundings [19, 35, 39]. RL agents learn action policies directly from observations without requiring an explicit map, which makes them more flexible in dynamic or partially-observable settings.

Fig. 1: Representation of EDEN inspiration. The proposed navigation system is inspired by grid cells found in the entorhinal cortex of the mammalian brain, which support spatial reasoning and goal-directed behavior. *The conceptual visualization was generated with the assistance of OpenAI's DALL·E.*

However, both of these approaches suffer from significant drawbacks. Several studies have shown that SLAM and visual odometry suffer from temporal drift, especially in scenarios with repetitive sensor data, eg. images [5, 25]. This error accumulates over time, leading to localization drift and degraded performance in planning tasks. Conversely, purely RL-based navigation policies often lack structure and poorly generalize, as they depend entirely on end-to-end learning and do not maintain a consistent representation of space. This results in inefficiencies and difficulties in long-horizon planning.

In contrast, animals navigate using flexible, extensible learned spatial representations. Hafting et al. [10] found that

the dorsocaudal medial entorhinal cortex (dMEC) in mammalian brains creates an internal cognitive map using special neurons called grid cells. These cells encode space through a multi-scale periodic representation that supports updating one's position using self-motion cues (path integration) and computing direct routes to goals (vector-based navigation) by comparing grid representations. Built on this, several studies have explored computational models inspired by grid-cell mechanisms [8, 37, 48, 4, 38]. For example, Banino et al. [2] trained a recurrent network to perform path integration, leading to the emergence of grid cell-like representations. This provided a foundation for efficient, vector-based navigation in complex scenarios. These also suggest that brain-inspired representations could enhance robotic navigation when integrated with learning frameworks. Recent work has further extended this approach to robotics, demonstrating that brain-inspired architectures using deep learning-emerged grid cell models can be deployed for navigation in both simulated and physical scenarios [42, 14, 49].

Unlike SLAM, which builds and stores a detailed map using LiDAR and point clouds, humans learn to navigate by tracking landmarks and memorizing object locations. London taxi drivers have even shown structural growth in their hippocampus as they develop a full mental map of the city [22]. Similarly, EDEN uses brain-inspired grid representations to keep track of its position internally and uses landmarks to reach the goal, storing important high level representations of space rather than estimated point clouds.

Motivated by these biological insights, we propose a brain-inspired navigation framework that combines the strengths of both neuroscience and machine learning as shown in Figure 1. Instead of relying on external maps or raw policies, EDEN embeds an internal grid-cell-like representation that is updated by velocity and visual landmarks. This representation serves as input to an RL policy for decision-making. As a result, the agent maintains an internal sense of position and learns efficient navigation strategies through rewards. Note this is different from traditional sinusoidal position encoding schemes, since the position encodings are inferred from self motion and visual cues.

Our contributions are summarized as follows:

- We design and train a an RL policy architecture that uses grid representations inspired by the mammalian entorhinal cortex. Our architecture enables efficient navigation under noisy or partial observations without relying on pre-defined external maps.
- We develop a grid cell encoder network distilled from ground truth activations, which enables the prediction of grid cell responses from high-level visual features and proprioceptive signals.
- We evaluate EDEN in both simple and complex scenarios using lightweight and high-fidelity simulators. We compare EDEN's performance against baselines that rely on explicit position data, raw visual inputs, and preprocessed object detection.

II. Related Work

A. Simultaneous Localization and Mapping (SLAM)

SLAM is a longstanding standard for flexible autonomous navigation [43, 6]. It uses data from multiple sensors, such as LiDAR, IMU, GPS, and cameras, which are merged to help mitigate the weaknesses of any single modality [44]. Shin et al. introduced a direct-vision SLAM method, showing that integrating LiDAR and vision camera data can enhance accuracy and real-time performance when sparse depth information is combined [34]. Herraez et al. achieved smooth and accurate onboard simultaneous localization and mapping by combining automotive radars with an IMU and exploiting the additional velocity and radar cross-section information provided by radar sensors [13]. However, despite the powerful aspects of SLAM, approaches in this area can suffer from temporal drift, especially in visually repetitive scenarios [5, 25]. Errors accumulate over time, leading to localization drift and degraded task planning performance. Furthermore, components used in SLAM can be noisy, and introduce inaccuracies to the calculations [45]. Even though there exist several denoising filters, such as the Kalman filter [17], they have have low flexibility and adaptability under complex conditions [3].

B. Path planning

Path planning is another key component of effective autonomous navigation [9, 47]. Classical algorithms such as A* [11] and D* [12] offer reliable solutions in well-mapped scenarios [21, 16] but often require full observability and accurate localization, which may not be possible in unknown scenarios. Song et al. developed a model that learns human driving behavior and adapts to different roads by a combination of CNN and LSTM-based networks [36]. More recent approaches based on deep reinforcement learning (DRL) have shown promise in learning end-to-end policies that combine perception, localization, and control [24, 1]. These methods enable agents to navigate complex scenarios without access to maps, instead using learned spatial representations and raw sensor inputs. However, without reliable cues, DRL agents often struggle with drift and complexities [23]. Benchmarks such as Habitat point out that many policies that excel in their training mazes collapse when the geometry or texture distribution shifts [31].

C. Bio-inspired navigation architectures

Recent research has begun to explore bio-inspired solutions, particularly mammalian grid cells, which support spatial and stable localization in biological systems. Notably, *in vivo* grid cells can maintain structured representations even in darkness for up to $\tilde{1}0$ minutes [10]. This highlights their role in consistent localization across time and sensory uncertainty. Grid cell-based models have been shown to enable agents to form internal maps and localize themselves effectively [2, 40]. Yang et al. showed a possible neural solution to overcome the serious drift of IMU-based inertial navigation of Unmanned Aerial Vehicles (UAVs) in the absence of external sensory cues [4]. Later, Sheng et al. extended the role of the grid

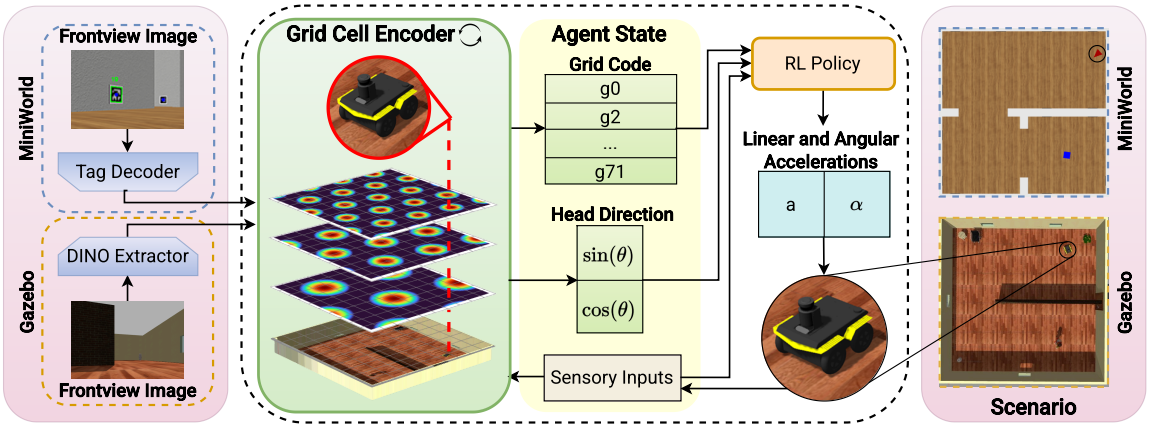

Fig. 2: Proposed architecture of EDEN deployed within the MiniWorld and Gazebo simulators. Grid cell encodings are developed from features extracted using front-facing views and velocity signals, features created in MiniWorld via detected fiducial tags, replaced by general-purpose features extracted using the DINO framework within Gazebo. The predicted grid cell activations and head direction are then combined with sensory inputs passed to the RL policy for controlling the agent via acceleration commands.

cell system from spatial navigation to visual concept space, meaning the information retrieved can be used for situational awareness [33].

III. Proposed Approach

This section outlines the approach used to implement EDEN within two different simulators: MiniWorld [7] and Gazebo [20], as illustrated in Figure 2.

A. Grid Cell and Head Direction Encoder

The grid and head direction modules maintain a continuous estimate of the position and orientation of the agent through a combination of egocentric motion cues, such as linear and angular velocities, and visual features to update a latent spatial code over time.

The core of the encoder is an LSTM operating on the MLP-transformed motion and sensory features. The output is a compact embedding that encodes the location of the agent in a distributed representation. This embedding is concatenated with other sensory inputs and passed to the policy network to produce the corresponding action. The encoder undergoes supervised pretraining using idealized grid representatives

$$\alpha \max\left(0,\ \cos(\hat{x}-\phi_x)+\cos\left(\frac{-\hat{x}+\phi_x}{2}+\frac{(\hat{y}+\phi_y)\sqrt{3}}{2}\right)+\cos\left(\frac{-\hat{x}+\phi_x}{2}-\frac{(\hat{y}-\phi_y)\sqrt{3}}{2}\right)\right),$$

where ϕ is a phase shift, α is a scaling factor, and $(\hat{x},\hat{y}) = R(x,y)$ is a rotation of the standard basis. This approach minimizes the gap between the predicted and actual positions over time and enables the encoder to learn a stable mapping from motion sequences to spatial embeddings. Multiple grid code embeddings are generated on various spatial scales α, each combined with several phase changes ϕ and repeated under different rotational frames $(\hat{x},\hat{y})$. This results in a high-dimensional representation that enables accurate localization through the overlapping of periodic patterns detailed in the Grid Cell Encoder block of Figure 2 with an overview of the implemented module outlined in VII-A.

Alongside the grid code embedding of position, a periodic encoding is applied to represent the agent's orientation, θ, relative to the global reference frame. To ensure smooth transitions across angular boundaries (e.g. near 0 and 2π), the raw angle θ is transformed as $[\sin(\theta), \cos(\theta)]$. This continuous embedding facilitates stable learning akin to the periodic representation found within head direction cells of biological mammals.

The proposed encoder is fully differentiable and trained end-to-end within the RL pipeline, making it robust to input noise. Importantly, it serves as an alternative to raw coordinate inputs, enabling the agent to reason about space without direct access to positional information, thereby promoting more biologically plausible behavior.

B. Reinforcement Learning Policy and Algorithm

With the positional encoding of EDEN established, we adopt the Proximal Policy Optimization (PPO) algorithm [32], implemented via Stable-Baselines3 [29], as the RL backbone due to its robustness and reliable training performance. We apply a custom reward function that penalizes collisions with the scenario and the use of longer trajectories, while rewarding the agent for reaching the goal.

IV. Experimental Setup

In this section, we outline the experimental setup used to evaluate the proposed system for comparison with baseline input modalities. The section begins with defining task scenarios tailored to two simulators.

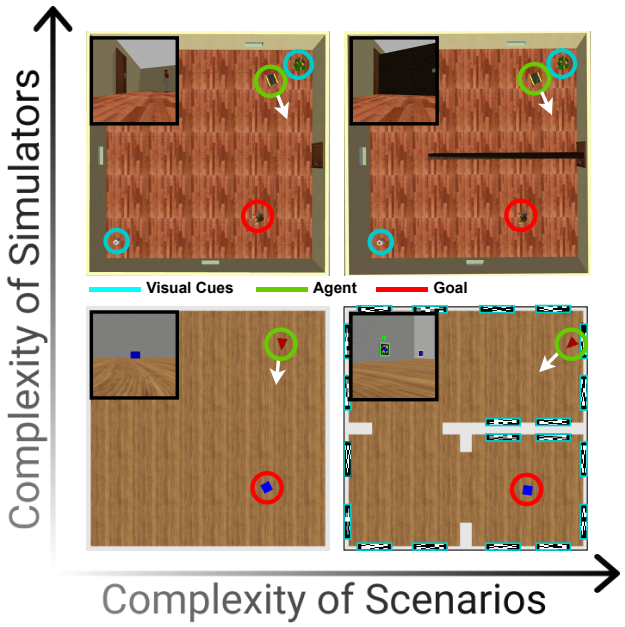

Fig. 3: MiniWorld and Gazebo scenarios used for training with both simple and complex floorplans. In MiniWorld, landmarks are defined using unique fiducial tags placed on walls (bottom right), while in Gazebo, these tags are replaced with realistic objects (top left).

A. Simulator and Agent Setup

Using a stepwise approach, we first deploy and develop EDEN in the MiniWorld [7] simulator—a lightweight platform designed for vision-based navigation tasks. Following this, we transition to the more realistic Gazebo [20] simulator, which offers high-fidelity robotic simulations making it better suited for evaluating the real-world applicability of EDEN.

Within these simulators, we construct two distinct scenarios to evaluate EDEN's performance. In the first, the agent must navigate within an open room. In the second, a wall separates the agent from the goal, requiring planing around occluded paths. In both cases, the agent is initialized at a random starting position and tasked with reaching a fixed goal located at the center of the lower right quadrant. These scenarios are visualized in Figure 3.

While the simulators vary in complexity, both pose challenges related to maintaining long-term spatial consistency and planning under partial observability. In biological systems, such challenges are met by integrating multiple sensory modalities. For instance, in the entorhinal cortex, visual cues are combined with vestibular input to stabilize internal spatial representations over time [18]. Inspired by this mechanism, we design agents and scenarios that incorporate visual cues aligned with the agent's motion, allowing EDEN to maintain a coherent internal estimate of its position and orientation.

1) Action Space: To emulate vestibular input observed in mammals, the agent interacts with the simulation through a continuous action space defined by linear acceleration (a) and angular acceleration (α) per step. This setup enables both translational and rotational motion, allowing for fine-grained control over the agent's trajectory and heading. The resulting velocities are then clipped to predefined maximums, normalized, and fed into EDEN. A description of how these actions are translated into motion commands is outlined in Section VII-B.

2) MiniWorld Agent and Landmarks: In MiniWorld, the agent is represented as a red triangle. During each step, the agent receives a forward-facing RGB image, which serves as the input for computing Grid Cell activations. To incorporate visual cues, AprilTag fiducial markers [26] are placed on the surrounding walls. The tag closest to the center of the agent's view is identified and its data is used as an additional input.

3) Gazebo Agent and Landmarks: The transition to Gazebo enables training and evaluation of EDEN in a more realistic and physically grounded simulator. Scenarios are developed using the CPR Gazebo framework [30], which models a Clearpath Jackal robotic agent equipped with a LiDAR sensor and a front-facing RGB camera. Leveraging this framework, along with assets from the Gazebo Models and Worlds collection[46], an office-like floorplan scenario is constructed.

To enhance realism and promote generalization beyond synthetic visual markers, the AprilTag fiducials used in MiniWorld are replaced with common real-world objects, such as an office chair, a potted plant, and a human figure. Visual features from these objects are extracted using DINOv2 [27], a self-supervised vision transformer model employed as a general-purpose feature extractor.

B. Grid Cell Encoder Training

We adopt a sequential approach to develop the trainable Grid Cell encoder. First, an initial RL policy is trained using ground-truth grid activations. Once this policy has sufficiently converged, the agent is deployed in repeated randomized episodes to collect vision and motion data. To enhance the encoders reliability, we create an additional dataset using a randomly acting agent within the scenario. Using the full dataset, we train an LSTM network to approximate the grid cell representations, with the training procedure detailed in Section VII-E. This network, which receives both proprioceptive and visual inputs, serves as a drop-in replacement for the ground-truth grid activations.

C. Baseline Input Modalities

To evaluate the effectiveness of grid cell encoding as the primary localization signal, we introduce a set of baseline input modalities. Each modality receives the normalized linear velocity (v) and angular velocity (ω) as proprioceptive input. In Gazebo, we additionally augment all networks with LiDAR point cloud data. The evaluated input modalities are as follows:

- **Position and Orientation.** The raw $[x, y]$ position of the agent, normalized by the scenario's maximum bounds, is provided as input alongside a sine and cosine encoding of the agent's orientation.

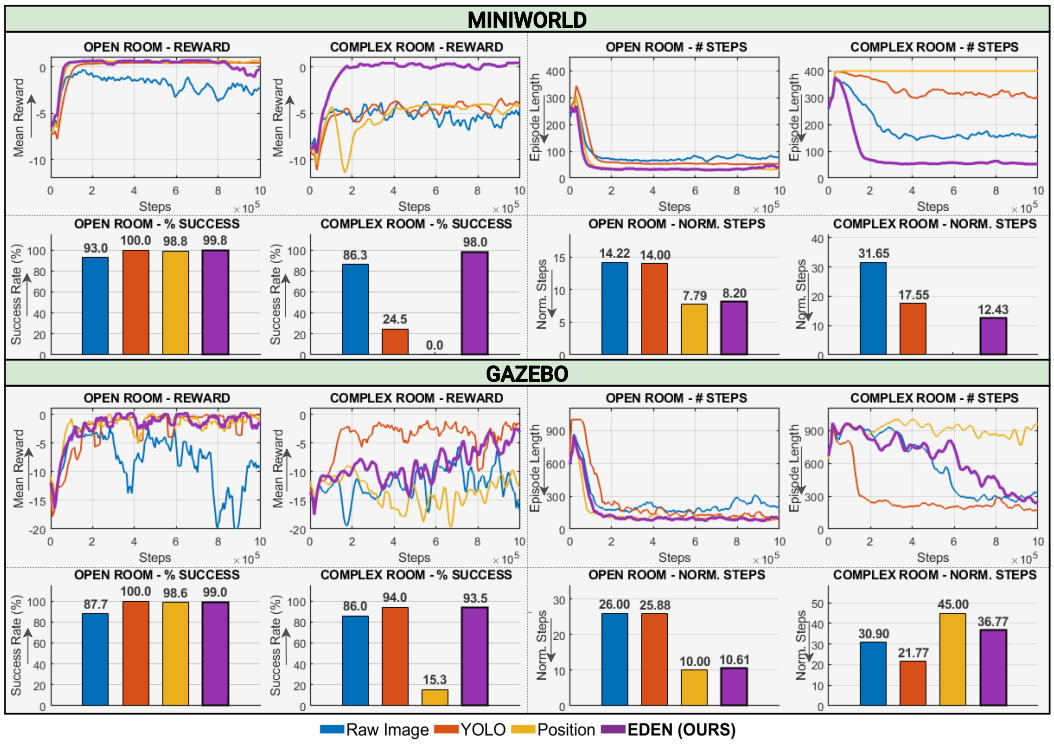

Fig. 4: Performance comparison across four training scenarios using three baseline observation modalities and EDEN. Top: Mean episode reward. Bottom: Success rate and steps normalized by shortest path distance, evaluated on successful episodes within 1000 testing episodes. All modalities receive the agent's current velocity with the addition of LiDAR within Gazebo. Due to there being no successful episodes within the MiniWorld complex scenario given raw position input, the normalized step metric is omitted.

- **Raw Image.** The front view image of the agent is extracted and fed directly into the policy.
- **YOLO Detection.** The raw image is passed through the YOLOv5 object detector [15], from which only the detection corresponding to the goal class is provided.

For fair comparison, identical policy architectures are used across all modalities, with the addition of convolutional layers in the raw image baseline for spatial feature extraction. An in detail overview of the policies is provided in Section VII-C.

V. Results

We evaluate EDEN in two key areas: (1) the effectiveness of grid cell activations in training RL policies using cumulative reward, success rates, number of steps, and distance-normalized steps, with detailed calculations outlined in Section VII-D, and (2) the feasibility of encoding visual and motion sensor data into grid cell representations.

A. Reinforcement Learning Agent Performance

We show success rates and episodic reward for open and complex rooms in both MiniWorld and Gazebo in Figure 4, including both the grid inspired and baseline models discussed in Section IV-C. We additionally show sample paths taken by the different agents in Figure 5. Our results show that across all models, the open room is easily and near-instantly resolved (bottom left bar charts of Figure 4). The exception to this is training with raw pixels, which results in the model learning artifacts that destabilize training. The complex scenario is substantially more difficult, however: when training on limited epochs, ground truth-normalized (x, y) coordinates failed entirely to adapt to the position of a wall. In the MiniWorld set-up, grid activations with learned grid cells are able to outperform both unprocessed and processed images in the complex scenario as well. One interesting point is that, while normalized position has the lowest success rate, it has a comparable reward to raw pixels and YOLO, suggesting that it is superior at avoiding obstacles, at the expense of failing to learn a viable path to the goal. In the Gazebo scenario, in which all models additionally had access to LiDAR data, YOLO-based image preprocessing resulted in the swiftest and most stable training, though grid cell encoding achieved a comparable success rate at one million training steps.

In the top of Figure 5, we show sample trajectories of the best version of each method in the complex scenarios. In MiniWorld, the YOLO based model never learns a strategy beyond swiveling, and therefore does not move from its position. In Gazebo, however, where YOLO is augmented with a recurrent policy and LiDAR data, it is able to smoothly converge. The raw normalized position fails in complex scenarios, while raw

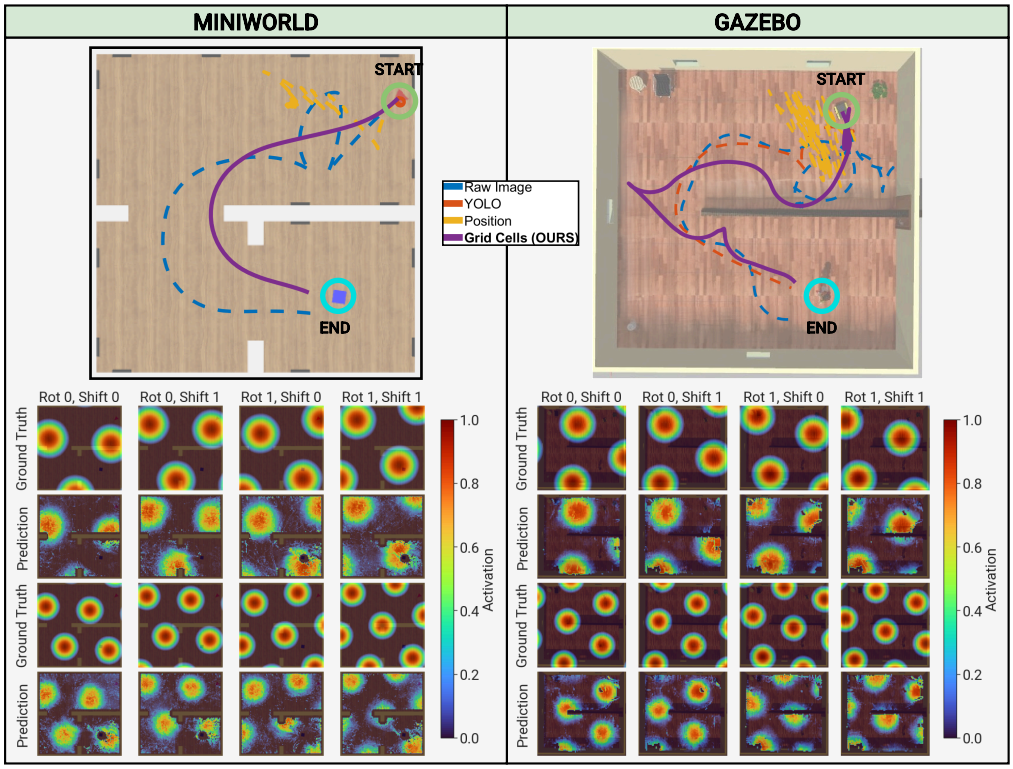

Fig. 5: Sample episodes from each of the four trained models performing goal-directed navigation in MiniWorld and Gazebo (top). For each episode, the ground-truth grid cell activations are shown, alongside the predictions obtained by averaging activations from both policy-driven and random trajectories. This comparison illustrates the decoder's ability to reconstruct spatial representations from visual inputs and effectively navigate within the complex scenario.

pixels and the grid cell method both ultimately succeed in both scenarios. Grid encodings of our model in Gazebo contain several notable cusps, which we believe are the result of the model not fully converging by 1 million steps. We anticipate that mild additional training would continue to optimize both the raw pixel and grid cell paths.

B. Developed Grid Cell Encodings

For final deployment, the ground truth activations used to train the EDEN RL policies are not directly accessible. To address this, using the procedure outlined in VII-A we trained an LSTM based grid cell encoder. The bottom half of Figure 5 presents a heatmap comparison between the predicted and real grid cell activations across eight representative cells sampled over multiple agent trajectories. The learned activations closely match the ground truth in terms of translation, scale, and rotation, demonstrating that the grid cell encoder effectively captures spatial representations.

VI. Conclusion

In this work, we present EDEN a biologically inspired framework that integrates a localization model and an RL agent inspired by grid cell structures found within the entorhinal cortex. By processing egocentric inputs (specifically velocity signals and visual cues) EDEN demonstrates effective navigation in both simple and complex scenarios in the MiniWorld and Gazebo simulators. The agent achieves an overall success rate exceeding 94% across all scenarios, exhibiting both efficient and reliable navigation behavior. Using the trained policies, we further develop a grid cell encoder module capable of accurately predicting ground truth activations solely from image and motion data.

EDEN presents a step toward the development of more autonomous and resilient robotic agents by leveraging principles from mammalian neural navigation systems. Through four test scenarios, we evaluate the feasibility of this approach in controlled environments. However, real-world deployment introduces significant complexity and variability, introducing the need for the development of policies that can generalize to dynamic, unfamiliar conditions. Building on this foundation, future work will explore the integration of local coordinate frames and related techniques to support zero-shot generalization in novel scenarios, eliminating the need for predefined maps or recurring spatial structures.

Acknowledgments

This work has been partially supported by the Army Research Laboratory Cooperative Agreement No W911NF2120211.

References

[1] Raed Alharthi et al. Novel deep reinforcement learning based collision avoidance approach for path planning of robots in unknown environment. *PLOS ONE*, 20(4): e0321810, 2025. doi: 10.1371/journal.pone.0312559.

[2] A. Banino et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705): 429–433, 2018. doi: 10.1038/s41586-018-0102-6.

[3] Yaakov Bar-Shalom et al. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. Wiley-Interscience, 2002.

[4] Yoram Burak and Ila R Fiete. Accurate path integration in continuous attractor network models of grid cells. *PLoS Computational Biology*, 5(2):e1000291, 2009. doi: 10.1371/journal.pcbi.1000291.

[5] Cesar Cadena et al. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6): 1309–1332, 2016. doi: 10.1109/TRO.2016.2624754.

[6] Peng Chen et al. A review of research on slam technology based on the fusion of lidar and vision. *Sensors*, 25 (5):1447, 2025. doi: 10.3390/s25051447.

[7] Maxime Chevalier-Boisvert et al. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

[8] Christopher J Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=B17JTOe0-.

[9] Alessandro Gasparetto et al. Path planning and trajectory planning algorithms: A general overview. In *Motion and Operation Planning of Robotic Systems*, volume 29 of *Mechanisms and Machine Science*, pages 3–27. Springer, 2015. doi: 10.1007/978-3-319-14705-5_1. URL https://doi.org/10.1007/978-3-319-14705-5_1.

[10] T. Hafting et al. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, 2005. doi: 10.1038/nature03721.

[11] Peter E. Hart et al. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, volume 4, pages 100–107. IEEE, 1968. doi: 10.1109/TSSC.1968.300136.

[12] Peter E. Hart et al. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*. IEEE, 1994. doi: 10.1109/ROBOT.1994.351061.

[13] Daniel Casado Herraez et al. RaI-SLAM: Radar-Inertial SLAM for Autonomous Vehicles. *IEEE Robotics and Automation Letters*, 10(6):5257–5264, June 2025. doi: 10.1109/LRA.2025.3557296.

[14] Maksims Ivanovs et al. Deep learning-emerged grid cells-based bio-inspired navigation in robotics. *Sensors*, 25(5):1576, 2025. doi: 10.3390/s25051576. URL https://www.mdpi.com/1424-8220/25/5/1576.

[15] Glenn Jocher et al. YOLOv5: Open-source object detection model. https://github.com/ultralytics/yolov5, 2020. Accessed: 2025-04-28.

[16] Chunyu Ju et al. Path planning using an improved a-star algorithm. In *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, pages 23–26. IEEE, 2020. doi: 10.1109/PHM-Jinan48558.2020.00012. URL https://ieeexplore.ieee.org/document/9296641.

[17] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82 (1):35–45, 1960.

[18] Alexandra T Keinath et al. Environmental deformations dynamically shift the grid cell spatial metric. *Elife*, 7: e38169, 2018.

[19] J. Kober et al. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32 (11):1238–1274, 2013. doi: 10.1177/0278364913495721. URL https://doi.org/10.1177/0278364913495721.

[20] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154. IEEE, 2004.

[21] Lisang Liu et al. Research on path-planning algorithm integrating optimization a-star algorithm and artificial potential field method. *Electronics*, 11(22):3660, 2022. doi: 10.3390/electronics11223660. URL https://doi.org/10.3390/electronics11223660.

[22] Eleanor A. Maguire et al. Navigation-related structural change in the hippocampi of taxi drivers. *Proceedings of the National Academy of Sciences*, 97(8):4398–4403, 2000. doi: 10.1073/pnas.070039597.

[23] Victor R. F. Miranda et al. Generalization in deep reinforcement learning for robotic navigation by reward shaping. *IEEE Transactions on Industrial Electronics*, 2023. doi: 10.1109/TIE.2023.3290244. arXiv:2209.14271.

[24] Piotr Mirowski et al. Learning to navigate in complex environments. *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=SJMGPrcle.

[25] Raul Mur-Artal et al. Orb-slam: A versatile and accurate monocular slam system. In *IEEE Transactions on Robotics*, volume 31, pages 1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

[26] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, 2011. doi: 10.1109/ICRA.2011.5979561.

[27] Maxime Oquab et al. Dinov2: Learning robust visual features without supervision, 2023.

[28] Morgan Quigley et al. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source*

Software, volume 3, page 5, 2009.

[29] Antonin Raffin et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[30] Clearpath Robotics. cpr_gazebo: Additional indoor and outdoor simulation environments for clearpath robots. https://github.com/clearpathrobotics/cpr_gazebo, 2025. Accessed: 2025-05-20.

[31] Manolis Savva et al. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9339–9347, 2019. doi: 10.1109/ICCV.2019.00943.

[32] John Schulman et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] Huankun Sheng et al. A hippocampal–entorhinal system inspired model for visual concept representation. In *IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS*, volume 13, pages 429–441. IEEE, 2021. doi: 10.1109/TCDS.2020.2978918.

[34] Yeong-Sik Shin et al. Direct visual slam using sparse depth for camera-lidar system. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5144–5151, Brisbane, QLD, Australia, 2018. IEEE. doi: 10.1109/ICRA.2018.8461102.

[35] B. Singh et al. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022. doi: 10.1007/s10462-021-09997-9. URL https://doi.org/10.1007/s10462-021-09997-9.

[36] Sheng Song et al. Learning a deep motion planning model for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1137–1142, Changshu, China, June 2018. IEEE. doi: 10.1109/IVS.2018.8500703. URL https://doi.org/10.1109/IVS.2018.8500703.

[37] Ben Sorscher et al. A unified theory for the origin of grid cells through the lens of pattern formation. *Nature Communications*, 14(1):1614, 2023. doi: 10.1016/j.neuron.2022.10.003.

[38] Martin B Stemmler et al. Connecting multiple spatial scales to decode the population activity of grid cells. *Science Advances*, 1(11):e1500816, 2015. doi: 10.1126/sciadv.1500816.

[39] Chen Tang et al. Deep reinforcement learning for robotics: A survey of real-world successes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(27):28694–28698, 2025. doi: 10.1609/aaai.v39i27.35095. URL https://doi.org/10.1609/aaai.v39i27.35095.

[40] Huajin Tang et al. Cognitive navigation by neuro-inspired localization, mapping, and episodic memory. In *Proceedings of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS*, pages 751–761. IEEE, 2018. doi: 10.1109/TCDS.2017.2776965.

[41] Mark Towers et al. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

[42] Jing Wang et al. Bioinspired perception and navigation of service robots in indoor environments: A review. *Biomimetics*, 8(4):350, 2023. doi: 10.3390/biomimetics8040350. URL https://www.mdpi.com/2313-7673/8/4/350.

[43] Ke Wang et al. An in-depth examination of slam methods: Challenges, advancements, and applications in complex scenes for autonomous driving. In *IEEE Transactions on Intelligent Transportation Systems*, pages 1–22. IEEE, 2025. doi: 10.1109/TITS.2025.3545479.

[44] Zhangjing Wang et al. Multi-sensor fusion in automated driving: A survey. *IEEE Access*, 8:2847–2868, 2019. doi: 10.1109/ACCESS.2019.2962554.

[45] Penggao Yan et al. A fault detection algorithm for lidar/imu integrated localization systems with non-gaussian noises. In *Proceedings of the 2024 International Technical Meeting of The Institute of Navigation*, pages 561–574, 2024. doi: 10.33012/2024.19564.

[46] Chao Yao. Gazebo models and worlds collection. https://github.com/leonhartyao/gazebo_models_worlds_collection, 2025. Accessed: 2025-05-20.

[47] Han ye Zhang, Wei ming Lin, and Ai xia Chen. Path planning for the mobile robot: A review. *Symmetry*, 10 (10):450, 2018. doi: 10.3390/sym10100450. URL https://doi.org/10.3390/sym10100450.

[48] Liao Yishen and Naigong Yu. A real-time cognitive map construction method based on the entorhinal-hippocampal working mechanism of the rat's brain. *Cognitive Computation and Systems*, 09 2024. doi: 10.1049/ccs2.12101.

[49] Naigong Yu et al. A bionic robot navigation algorithm based on cognitive mechanism of hippocampus. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING,*, 16(4):1640–1652, 2019. doi: 10.1109/TASE.2019.2909638.

VII. Appendix

A. Grid Cell Module

Within this work we utilize the grid cell encoding equation using a total of 72 differing grid cell activations. These activations are limited by a maximum scale factor α of 10, within Gazebo and 5 within MiniWorld with 3 initial cells. This scale factor is multiplied over 12 shifts which are then further mmultiplied over 2 orientations. Effectively this configuration of 72 relates to the number of cells used within EDEN. In this work these values were selected as an initial base case which can in the future be replaced with optimal configurations determined based on the current application and complexity of the scenarios.

B. Movement Scheme

The actions produced by the reinforcement learning policy correspond to an acceleration normalized within $[-1, 1]$. To translate the raw acceleration outputs into executable motion commands by the agent, each value is combined with the previous velocity using a weighted sum. This approach ensures smooth transitions between steps. The resulting velocity updates are clipped to maintain valid bounds and are given by

$$\begin{aligned} v_t &= \text{clip}\left(v_{t-1} + \beta_v \cdot a,\ -1,\ 1\right) \\ w_t &= \text{clip}\left(\omega_{t-1} + \beta_\omega \cdot \alpha,\ -1,\ 1\right), \end{aligned}$$

where v_t represents the velocity at time t, and ω_t the angular velocity at time t, and β_v, β_ω are scalar coefficients used to smooth the acceleration actions over several steps.

In our experiments, both β values were set to 0.5. The resulting velocities are then scaled by their respective maximum values, with v bounded by $\pm 0.5 m/s$ in MiniWorld and $\pm 1.0 m/s$ in Gazebo with ω by $\pm\frac{\pi}{4}$ rad/s in both scenarios. This configuration allows the agent to accelerate by a maximum of $\pm 0.25 m/s$ in MiniWorld and $\pm 0.5 m/s$ in Gazebo and $\pm\frac{\pi}{8}$ rad/s where s relates to a step in MiniWorld and approximately one simulated second within Gazebo.

As the agent interacts with the scenario using a continuous action space, Gazebo accommodates this by providing Twist commands consisting of angular and linear velocities. However, in the case of MiniWorld the provided action space uses discrete actions consisting of: $turn\_left$, $turn\_right$, $move\_forward$ and $move\_backward$. To update the position of the agent using a continuous action, the agents movement is parameterized by v and ω.

Here, v represents the length of the arc across a circle with a central angle of ω (in radians) between the initial and final position. With this we can then compute the radius of the circular arc as

$$r = \frac{v}{\omega},$$

and the chord length d as

$$d = 2rsin(\omega/2).$$

The angle α between the chord and radius from the circle center to the agent's initial or final position as

$$\alpha = \frac{\pi}{2} - \frac{\omega}{2}.$$

Translating this to the 2D grid within MiniWorld, the next position can be found given the initial position x_0, y_0 and orientation to the global reference frame θ_o. This produces the next x coordinate as

$$\begin{aligned} x &= x_0 + d\cos\left(\theta_0 + \frac{\omega}{2}\right) \\ &= x_0 + r\left(\sin(\theta_0 + \omega) - \sin(\theta_0)\right), \end{aligned}$$

and similarly for the y coordinate

$$\begin{aligned} y &= y_0 + d\sin\left(\theta_0 + \frac{\omega}{2}\right) \\ &= y_0 - r\left(\cos(\theta_0 + \omega) - \cos(\theta_0)\right). \end{aligned}$$

The agents final orientation θ is then

$$\theta = \theta_0 + \omega.$$

This movement method for updating the coordinates x, y and angle θ are then integrated within MiniWorld to replace the original discrete actions producing an agent that can be controlled using continuous actions resembling movements similar to that of robotic agents.

C. Reinforcement Learning Policies and Training

Reinforcement Learning Policies To begin, the inputs for each modality are outlined below. Each modality additionally receives a 2-dimensional input vector representing the agent's current velocity, as well as a 36-dimensional LiDAR input vector. The LiDAR readings are averaged across discrete angular regions within *Gazebo*, clipped to a maximum range of 5 m, and normalized to the range $[0, 1]$.

1) **Raw Image:** A $96 \times 96 \times 3$ RGB image is provided directly to the network.
2) **YOLO:** To reduce complexity, only the x-coordinate of the detected object's centroid (with respect to the agent's egocentric view) is used. This value is encoded using sine and cosine functions to maintain continuity across image boundaries and to enable smooth transitions as the object moves across the field of view, resulting in a 2-dimensional input vector.
3) **Position:** The agent's current $[x, y]$ position is normalized with respect to environment bounds and encoded using sine and cosine, yielding a final 4-dimensional input vector.
4) **Grid Cell:** A 72-dimensional grid cell activation vector is concatenated with sine and cosine encodings of the agent's orientation, producing a 74-dimensional input vector.

Using these modalities, the policies interact with the scenarios using and the Gymnasium reinforcement learning interface [41] which is natively implemented within the MiniWorld simulator. In contrast, Robot control and perception are integrated through topics and services provided by the Robot Operating System (ROS)[28] within Gazebo and operate in pseudo real time, to account for this with our experiments we found that a step size of $0.05 sim\_seconds$ balances training performance and agent stability.

In MiniWorld, due to the simplified nature of the simulation environment, the policy architecture is designed with multiple fully connected layers shared between the actor and critic networks. For simplicity, the actor outputs a two-dimensional action vector, while the critic outputs a single scalar value. When using vector-based observations, the policy consists of a 128-dimensional input layer followed by two hidden layers of 128 neurons each, producing a 64-dimensional feature representation. Gaussian Error Linear Unit (GeLU) activations are applied between the hidden layers.

In contrast, for image-based observations, the policy incorporates a convolutional encoder followed by a fully connected feature projection. The encoder comprises three convolutional layers with ReLU activations and increasing channel dimensions of 32, 64, and 64. These layers use kernel sizes of 8×8, 4×4, and 3×3, and corresponding strides of 4×4, 2×2, and 1×1, respectively. Given an input image of size $96 \times 96 \times 3$, the final flattened output of the convolutional stack is a 4096-dimensional feature vector, which is then passed through a fully connected layer with 256 neurons to produce the final policy features.

Within Gazebo, a similar model architecture is employed, with the primary difference being the replacement of the PPO core with a recurrent-PPO LSTM-based structure. Additionally, the actor and critic networks are implemented as separate models. The convolutional feature extractor is retained for image-based inputs, while the vector-based policy undergoes a reduction of one hidden layer and a doubling of the hidden dimension, resulting in a feature extractor composed of layers with 256, 256, and 128 units.

The extracted features—whether from the image or vector-based policy—are passed through a single-layer LSTM with a hidden dimension of 128. The LSTM output is then processed by a two-layer fully connected network with 128 and 64 neurons, respectively, and TanH activation functions between layers. The final output layer also uses a TanH activation, encouraging the network to produce outputs within the $[-1, 1]$ range, which aligns with the bounds of the action space and reward function. With the policies constructed the resulting model size given the varying input modalities is displayed within column 2 of Table I.

TABLE I: Model sizes for the reinforcement learning policy, pre trained extractor and LSTM encoder used during training for the varying input modalities.

MINIWORLD				
Input Type	**RL Policy**	**Pre-Trained Extractor**	**LSTM Encoder**	**Total**
Image	4.29	-	-	**4.29**
YOLO	0.16	11.49	-	**11.65**
Position	0.16	-	-	-
EDEN (OURS)	0.2	-	0.7	**0.9**
GAZEBO				
Input Type	**RL**	**Extractor**	**Predictor**	Total
Image	10.51	-	-	**10.51**
YOLO	1.94	10.13	-	**12.07**
Position	1.94	-	-	-
EDEN (OURS)	2.08	83.21	1.77	**87.06**

To provide an overview of the resulting model sizes, Column 2 of Table I reports the additional parameters introduced by the pretrained extractors used for object detection or feature extraction. In the MiniWorld environment, a finetuned YOLO model is employed to detect the blue box goal. In the Gazebo setup, a YoloV5n model is used for object detection, while the DINOv2 model serves as a general-purpose feature extractor within the EDEN grid cell encoder. For the MiniWorld configuration, AprilTag detection is performed using a traditional algorithmic method; therefore, no additional parameters are reported for this component.

Reward System and Training To complement the PPO algorithm, a reward system is defined consisting of three main components; per-step reward (r_s), upon collision (r_c), and reaching the goal (r_g). This reward structure incentivizes the agent to reach the goal efficiently while minimizing collisions. For efficient and reliable navigation, we employ a sparse reward setting with $r_g = (1 - 0.2 \times (\frac{steps}{max\_steps}))$ where max_steps denotes the maximum number of steps before episode truncation and $steps$ refers to the number of steps taken in the current episode, $r_c = -0.1$, and $r_s = -0.01$.

For stable learning and efficient use of available resources, the agent is trained using 10 parallel instantiations for MiniWorld and 6 parallel instantiations in Gazebo the generated plots use a statistic window size of 50 within Miniworld and 30 within Gazebo relating to $5\times$ the number of parallel instantiations. During training, the reinforcement learning policies were trained with a rollout size of 1024 steps and a batch size of 128 in MiniWorld, and a rollout size of 512 steps in Gazebo. To promote effective exploration and stable learning, the following hyperparameters were used: an entropy coefficient of 0.01, a discount factor γ of 0.99, a GAE (λ) value of 0.95, actor and critic loss coefficients of 0.5, gradient norm clipping with a threshold of 0.5, and a clipping range of 0.2.

D. Distance Normalized Steps

To outline the distance-normalized steps provided in Figure 4, let us take the example of the MiniWorld complex scenario where the agent's current position is denoted by (x, y), and the positions of the three doors are given by:

$$D_1 = (x_1, y_1), \quad D_2 = (x_2, y_2), \quad D_3 = (x_3, y_3)$$

We define the Manhattan distance to each door as:

$$d_i = |x - x_i| + |y - y_i|, \quad \text{for } i \in \{1, 2, 3\}$$

he total Manhattan distance to all three doors is then:

$$d_{\text{total}} = d_1 + d_2 + d_3$$

Given that the agent takes s steps to reach the goal, the distance-normalized steps are computed as:

$$s_{\text{normalized}} = \frac{s}{d_{\text{total}}}$$

A similar procedure is repeated within Gazebo and the open room structures producing a metric that provides a measure of

navigation efficiency relative to the overall structural layout of the scenario.

E. Grid Cell Encoder and Training

To train the grid cell encoder, a representative dataset was constructed for supervised sample generation. In *MiniWorld*, the agent was trained across 20,000 trajectories of 400 steps each, while in *Gazebo*, 1,280 trajectories of 1,000 steps were collected. Using these datasets, a decoder model was trained based on the following input modalities:

- **MiniWorld AprilTags:** At each timestep, the current tag ID and the coordinates of its four corners are recorded. The corner coordinates are encoded using sine and cosine representations for rotational stability, resulting in a 16-dimensional vector. A total of 22 AprilTags are placed throughout the environment and represented using one-hot encoding, producing a final input vector of size 38.
- **Gazebo DINO Features:** For Gazebo, visual features are extracted using a pretrained DINOv2 feature extractor, resulting in a 384-dimensional feature vector per observation.

In both cases, the agent's velocity is appended to the visual features, resulting in a final input dimensionality of 40 for MiniWorld and 386 for Gazebo.

Encoder Architecture: In *MiniWorld*, the encoder network begins with a two-layer feedforward network with 128 hidden units per layer, using the Exponential Linear Unit (ELU) activation function. This is followed by an LSTM with a hidden size of 72, corresponding to the grid cell encoding dimension. The final output stage consists of a three-layer feedforward network with dimensions 256, 256, and 74, again using ELU activations between layers. A final `Tanh` activation is applied to bound the outputs to the range $[-1, 1]$. The final output vector contains 72 grid cell activations and a 2-dimensional sine and cosine encoding of the agent's orientation.

In *Gazebo*, a similar architecture is adopted, with the following modifications: ELU activations are replaced with Gaussian Error Linear Units (GELU), and the input layers are expanded to dimensions 512 and 256. Layer normalization is applied after the initial layers for regularization. The output head also consists of 256, 256 layers with 20% dropout applied between them, followed by a final 74-dimensional output layer. Regularization techniques are applied to the gazebo case due to the reduced amount of available samples.

Encoder Training and Results: To ensure robustness and improve generalization, the encoder module is trained on sequences of varying lengths. Training is conducted for 250 epochs in *MiniWorld* and 500 epochs in *Gazebo*. To initialize the LSTM hidden state at the start of each batch, a weighted combination is used:

$$h_0 = 0.8 \times g_0 + 0.2 \times \theta_0,$$

where g_0 denotes the initial grid cell activation, and θ_0 represents the initial orientation relative to a global reference frame.

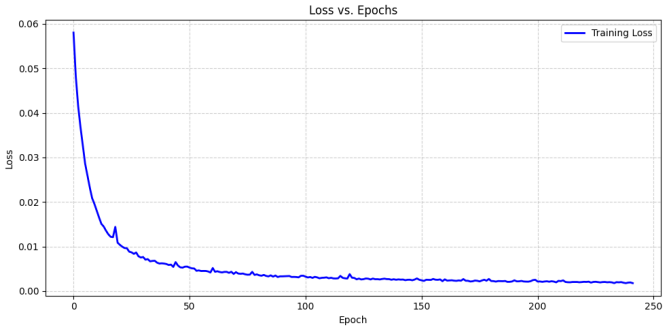

Fig. 6: Training loss for encoding AprilTags within Miniworld into grid code

The models are optimized using a Mean Squared Error (MSE) loss to predict the target grid cell encodings. Training results are shown in Figure 6 for the MiniWorld AprilTag-based decoder, and in Figure 7 for the Gazebo DINO-based decoder showing smooth convergence to predict grid cell activations.

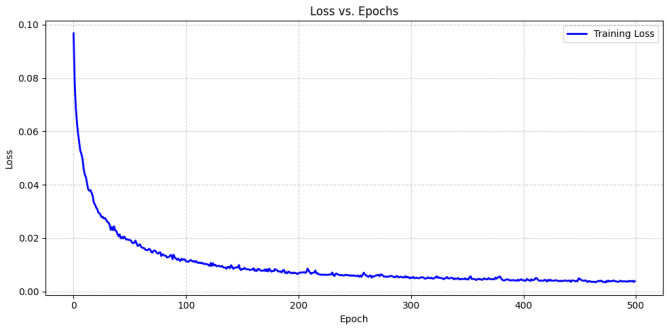

Fig. 7: Training loss for encoding DINO features within Gazebo into grid cells

As shown and discussed in Figure 5, the trained grid cell encoder is evaluated by deploying a policy-driven and random acting agent within the scenarios. Multiple trajectories are collected, and the movable area is divided into 100 equal regions. Histogram binning is then applied to generate the final heatmap, which serves as a qualitative tool to confirm the successful training of the encoder.