

Remember Where You Walk: Learning Spatial Memory for End-to-End Legged Robot Exploration

Leon Magnus^{*,1}, Nico Bohlinger^{*,1}, Jan Peters^{1,2,3,4}, Oleg Arenz¹

Abstract—Humanoids and other legged robots operating in complex environments must remember information that leaves their field of view and only becomes relevant much later. Reinforcement learning policies address this using memory networks such as LSTMs that are trained end-to-end with the policy to memorize such task-relevant aspects. Yet, such architectures often fail to preserve sufficient information over long horizons. In this work, we propose a network architecture based on Hadamard memory that specifically targets robot navigation. Our Local Spatial Matrix Memory (LSMM) enables RL agents to store generic task-relevant information at specific, position-based memory cells, enabling flexible and spatially grounded memorization. We evaluate LSMM on the task of autonomous maze exploration, where it matches the maze coverage of strong recurrent and finite-history baselines while being the only architecture that remembers the maze layout, reducing the reconstruction error threefold. We provide videos and code for our experiments at: <https://leonmagns.github.io/RememberWhereYouWalkWebsite/>

I. INTRODUCTION

Thanks to breakthroughs in Deep Reinforcement Learning (DRL), legged robots can now traverse rough terrain, jump over obstacles, and navigate toward goal positions using policies trained in highly-parallelized simulations or directly in the real world [1], [2], [3], [4], [5]. These policies usually follow velocity commands or short-horizon waypoints [6], [7], leaving the higher level planning to a human operator and making the system not fully autonomous. Autonomous exploration of an unknown environment is a fundamentally different problem. Now, the robot must not only reason and control its body in a local frame, but also decide where to go and remember where it has already been, to navigate and explore efficiently [8]. This turns locomotion into a long-horizon partially observable navigation problem in which the current local observation is not enough [9].

Memory-augmented policies address this problem by leveraging an observation-action history with architectures such as Long Short-Term Memories (LSTMs) [10], Gated Recurrent Units (GRUs) [11], Transformers [12], or simply concatenating history for an Multilayer Perceptron (MLP) [13]. However, unstructured recurrent states, through LSTMs or GRUs, easily mix or overwrite spatially distinct information, as credit-assignment is tricky, and finite-context

This project was funded by National Science Centre Poland in the Weave programme UMO-2021/43/I/ST6/02711, by the German Science Foundation (DFG) under grant number PE 2315/17-1 and by the European Union’s Horizon Europe research and innovation programme (Grant agreement ID: 101298421). This project has been supported by a hardware donation by NVIDIA through the Academic Grant Program. ^{*} Equal contribution. ¹ Technical University of Darmstadt, Germany. ² Robotics Institute Germany (RIG). ³ German Research Center for AI (DFKI). ⁴ hessian.AI

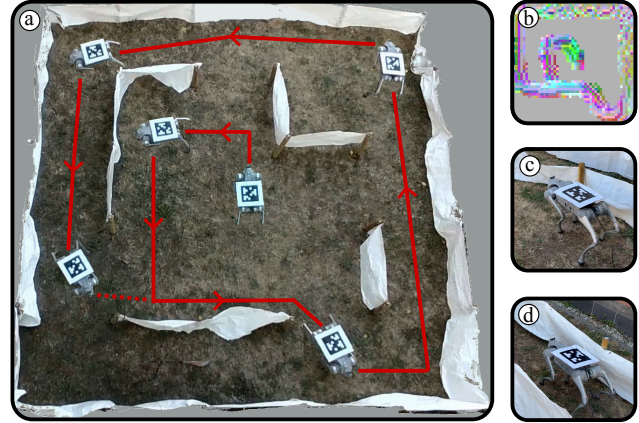


Fig. 1: (a) The Unitree Go2 autonomously explores a previously unseen 5×5 m outdoor maze. The red line shows the walked path. (b) The matrix memory at the end of the run. The policy has written into the cells along its path, shown with the three memory channels interpreted as RGB colors, while unwritten cells keep their initial value in gray. (c) and (d) Shows the robot navigating through the maze.

approaches, like Transformers or concatenation, discard old observations in less than a second with the typical history lengths [13], [14]. Classical navigation systems avoid this problem by explicitly registering and using observations in spatial maps, but typically separate mapping, planning, and locomotion into individual components, losing out on potential end-to-end efficiency [15], [16], [17].

In this work, we introduce LSMM and evaluate it for end-to-end maze exploration. The layout of our matrix memory is aligned with the environment’s frame and indexed using a position estimate. At every step, the policy modifies only a small window in the memory around the robot’s position through learned calibration and update matrices, while the rest of the memory is not updated. A Convolutional Neural Network (CNN) [18] encoder transforms the spatial memory into features used by both the policy and an auxiliary maze-reconstruction head. This combines the information persistence of index-based updates and the locality of map-based navigation with learned latent representations and end-to-end locomotion control, without any explicit path planning or waypoint commands.

We formulate long-horizon, goal-free maze exploration as an end-to-end locomotion problem. We train our method for the Unitree Go2 quadruped in procedurally generated mazes and compare LSMM against unstructured recurrent and

finite-history architectures. Our experiments provide comprehensive empirical evaluations in simulation and deployment in the real-world, demonstrating LSMM’s capabilities on maze reconstruction, exploration efficiency, robustness to localization noise, and generalization to unseen mazes.

II. RELATED WORK

Partial observability in DRL is commonly addressed by conditioning the policy on a finite observation history or a recurrent latent state [19]. Recurrent architectures such as LSTMs and GRUs have been widely used for navigation tasks [10], [11], [9], [20], while attention-based policies can leverage longer observation sequences [12], [21]. Le et al. show that LSTMs, GRUs, and many other recurrent architectures update their memory in the same underlying way, calibrating the previous memory element-wise and then writing new information additively [22]. They formalize this update rule as the Hadamard Memory Framework (HMF) and derive Stable Hadamard Memory, whose calibration keeps the memory from decaying or exploding over long horizons. However, all of these memories aggregate observations over time and maintain no explicit link between the stored information and the spatial location where it was acquired.

Spatial neural memories address this limitation by aligning the latent state with the geometry of the environment. Neural Map stores features in a two-dimensional memory indexed by the agent’s position for maze navigation [23]. MapNet similarly constructs a self-centered latent map and learns to register new observations within it [24]. Active Neural SLAM combines a learned mapping module with global and local policies, but keeps an explicit analytical planner between the map and the executed actions [25]. Our LSMM keeps the spatial indexing of these methods but does not need the analytical planner of Active Neural SLAM, since its memory features enter the locomotion policy directly. Unlike Neural Map, which writes a single feature vector into the agent’s current cell, it retrieves a whole window of cells around the robot’s estimated position, applies the HMF update rule inside this window, and leaves the rest of the memory unchanged. The memory and the locomotion policy are trained jointly, and an auxiliary head reconstructs the maze layout from the memory features, which encourages it to retain task-relevant information about the maze.

Classical legged-navigation stacks put the locomotion policy as the last layer in their pipeline, where it executes velocity or waypoint commands produced by separate localization, mapping, and planning modules [15], [16], [17]. Learning-based approaches tried to merge parts of this pipeline. Rudin et al. train a single policy that reaches a commanded target position within a given time [7], and Zhang et al. learn a navigation policy that steers a separate locomotion policy to such a target [26]. In both cases the target position itself still comes from an external planner or operator. For exploration this planner typically builds an occupancy map and repeatedly picks the next target on the frontier between the mapped and unexplored space of the environment [8]. We

remove this remaining component, since the spatial memory takes over the role of the occupancy map and the decision where to move next is learned jointly with the locomotion.

III. SPATIAL MEMORY FOR END-TO-END LEGGED ROBOT EXPLORATION

We propose the LSMM, a spatially indexed latent memory that allows a single policy to remember its environment, decide where to go, and control the robot’s joints at the same time. First, we formalize the exploration task and the interface of the policy. Then, we introduce the local memory update of LSMM and describe how memory, encoders, and policy are trained jointly with Proximal Policy Optimization (PPO) [27] and an auxiliary maze reconstruction objective.

A. Problem Formulation

We consider the autonomous exploration of unseen mazes with the Unitree Go2 quadruped. The robot starts at the center of a 10×10 m maze built from a 5×5 grid of 2×2 m cells and must visit as many cells as possible within an episode of 70 s. At deployment, the policy receives no goal position, waypoint, or velocity command. Since the current observation does not reveal which parts of the maze the robot has already visited, the task is a Partially Observable Markov Decision Process (POMDP) in which the policy must remember information over time. We therefore learn a memory-augmented policy $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t, \mathbf{M}_t)$ that maximizes the expected discounted return $\mathbb{E}[\sum_t \gamma^t r_t]$, where $\mathbf{M}_t$ is the spatial memory introduced in subsection III-C and r_t is the reward described in subsection III-D.

The observation $\mathbf{o}_t$ consists of a three-dimensional waypoint command $\mathbf{c}_t$, which is only used in early training and set to zero otherwise, the proprioceptive state $\mathbf{o}_t^{\text{prop}}$ comprising joint positions, joint velocities, the previous action, the angular velocity of the trunk, and the projected gravity vector, and an exteroceptive observation $\mathbf{o}_t^{\text{h}}$ of 250 terrain height samples around the trunk. $\hat{\mathbf{p}}_t = (\hat{x}_t, \hat{y}_t, \hat{\psi}_t)$ of the robot’s position and heading in the environment frame. The critic has access to privileged information, namely the linear velocity of the trunk, its height above ground, the contact states of the feet, the ground-truth position, the per-cell visit counts, and a wider map of height samples. The actions $\mathbf{a}_t$ are target joint position offsets to a nominal standing configuration, tracked by a PD controller at 50 Hz.

B. Policy Architecture

Figure 2 shows the policy architecture, which consists of two parts, the memory module and the actor. Since the Go2’s IMU cannot measure linear velocity directly, an MLP first estimates it from the proprioceptive observation as $\hat{\mathbf{v}}_t = l(\mathbf{o}_t^{\text{p}})$. A second MLP encodes the height samples into a feature vector $\mathbf{z}_t^{\text{h}} = h(\mathbf{o}_t^{\text{h}})$. Then, the memory module takes $[\mathbf{o}_t^{\text{p}}, \mathbf{z}_t^{\text{h}}, \hat{\mathbf{v}}_t, \hat{\mathbf{p}}_t]$ and writes them into the LSMM, as described in subsection III-C. A CNN then encodes the updated memory into a feature vector $\mathbf{z}_t^{\text{m}} = m(\mathbf{M}_t)$. Finally, the action is sampled from a Gaussian distribution

$$\mathbf{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, (\boldsymbol{\sigma}_t^2)), \quad (\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = d([\mathbf{c}_t, \mathbf{o}_t^{\text{p}}, \mathbf{z}_t^{\text{h}}, \mathbf{z}_t^{\text{m}}]).$$

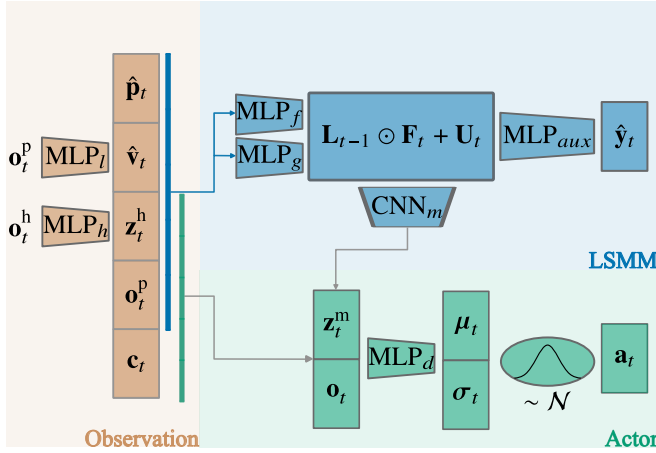


Fig. 2: Actor architecture with the proposed LSMM memory module. Both branches share z_t^h and o_t^p , while the memory branch additionally uses noisy position and velocity estimates $\hat{p}_t$ and $\hat{v}_t$ to update the matrix memory L_t . A CNN then extracts z_t^m from L_t and an auxiliary MLP predicts the maze wall layout $\hat{y}_t$. The actor maps z_t^m and its observation o_t to μ_t and σ_t , from which a_t is sampled.

Together, these networks form the policy π_θ , where θ collects the parameters of the velocity estimator l , the encoders h and m , the memory write functions f and g , and the actor d .

We train with an asymmetric actor-critic setup [28]. The critic is an MLP without memory. It receives the privileged observations from subsection III-A, where the per-cell visit counts summarize the exploration progress that the actor has to remember. An auxiliary head predicts the wall layout of the maze from the actor features at every step. The full layout can only be recovered from information gathered earlier in the episode, so predicting it requires the memory to store what the robot has seen. At deployment, the critic and the wall prediction are dropped, and only π_θ runs on the robot.

C. Local Spatial Matrix Memory

The LSMM is a tensor $M_t \in [0, 1]^{H \times W \times C}$ whose first two dimensions are aligned with the environment frame. We use $H \times W = 50 \times 50$ cells with $C = 3$ channels, so one cell covers 0.2×0.2 m of the maze. The channels carry learned features without a predefined meaning, and all entries are initialized to 0.5 at the beginning of an episode.

Updating the full memory at every step, as done by the recurrent architectures in the HMF [22], would constantly rewrite information about locations the robot currently cannot observe. Therefore, LSMM only updates a window $\mathcal{W}_t$ of 5×5 cells centered on the memory cell of the position estimate $\hat{p}_t$, corresponding to 1×1 m around the robot. Let $L_{t-1} = M_{t-1}[\mathcal{W}_t]$ denote the current content of this window. Two MLPs f and g compute a calibration tensor F_t and an update tensor U_t as

$$\begin{aligned} F_t &= f([o_t^p, z_t^h, \hat{v}_t, \hat{p}_t, L_{t-1}]), \\ U_t &= g([o_t^p, z_t^h, \hat{v}_t, \hat{p}_t, L_{t-1}]), \end{aligned} \quad (1)$$

where both tensors match the shape of the window and are bounded to $(0, 1)$ by a sigmoid. Through L_{t-1} the networks read the window before updating it, so F_t acts as a forget gate on the stored content and U_t writes new information into it. Following the update rule of the HMF, the window content is then overwritten with

$$\tilde{L}_t = L_{t-1} \odot F_t + U_t, \quad L_t = \frac{\tilde{L}_t}{\max(\tilde{L}_t) + \epsilon}, \quad (2)$$

where $\odot$ denotes the element-wise product and the normalization with $\epsilon = 0.02$ keeps all entries in $[0, 1]$. All entries outside $\mathcal{W}_t$ remain unchanged.

Observations made in different parts of the maze therefore modify different parts in the memory, and a written cell keeps its latent exactly until the robot returns to the corresponding location, independent of how much time passes in between. The calibration through F_t lets the policy decide how much of the previously stored information to keep when it revisits a location, and the spatial layout of M_t allows a direct visual comparison with the true maze, which we use in section IV.

D. RL Training Setup in Simulation

We train policy and critic with PPO in 5 000 parallel MuJoCo XLA (MJX) [29] environments with 300 000 pre-generated maze layouts. The full objective combines the clipped surrogate and value losses of PPO with the maze reconstruction and velocity estimation losses

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{PPO}} + \lambda_{\text{maze}} \mathcal{L}_{\text{maze}} + \lambda_{\text{vel}} \mathcal{L}_{\text{vel}}, \\ \mathcal{L}_{\text{maze}} &= -\frac{1}{n_w} \sum_{i=1}^{n_w} [y_{t,i} \log \hat{y}_{t,i} + (1 - y_{t,i}) \log(1 - \hat{y}_{t,i})], \\ \mathcal{L}_{\text{vel}} &= \|\mathbf{v}_t - \hat{\mathbf{v}}_t\|^2, \end{aligned} \quad (3)$$

where $\mathbf{y}_t \in \{0, 1\}^{n_w}$ indicates which of the n_w possible wall segments of the maze exist, $\hat{\mathbf{y}}_t$ contains the wall probabilities predicted by the auxiliary head, and $\mathbf{v}_t$ is the simulated linear velocity of the trunk. The gradients of $\mathcal{L}_{\text{maze}}$ reach the write functions f and g through the memory encoder m and train the memory to store the maze layout. The exploration task demands long-horizon credit assignment, as reaching the last unexplored cells at the end of an episode depends on decisions made many seconds earlier. Therefore, we use a high discount factor of $\gamma = 0.995$. Table I lists all learning hyperparameters.

E. Rewards

Each training episode is either a goal episode, in which the robot is commanded toward a neighboring maze cell, or an exploration episode, in which it receives no command and must cover as much of the maze as possible. The reward r_t consists of task rewards, which depend on this episode type, and shaping penalties for the gait and the navigation. In exploration episodes, the robot collects a reward for being in rarely visited cells, which decays with the time already spent in the current cell, and a reward for the overall maze coverage. In goal episodes, it is instead rewarded for moving toward the commanded cell and reaching it. The shaping

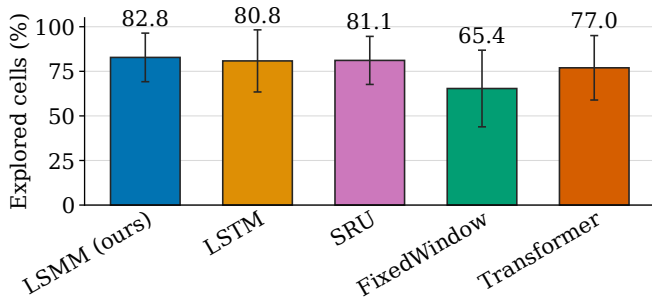


Fig. 3: Percentage of explored cells at the end of an episode. Bars show the mean over 100 unseen evaluation mazes and three seeds per architecture, whiskers the standard deviation. All architectures reach a comparable coverage.

penalties punish stalling, walking close to walls, joint limits, torques, action rates, and asymmetric gaits. Table II lists all terms and coefficients.

F. Curricula

Learning to walk and to explore at the same time is difficult, as the exploration rewards provide no useful signal while the robot cannot walk yet. Therefore, we bootstrap training with goal episodes, in which the command c_t contains the position of a reachable neighboring maze cell in the robot’s base frame. The least visited neighboring cell is selected to already bias the policy toward exploration, and once it is reached, the next goal is sampled immediately. The last entry of c_t flags the episode type, and in exploration episodes the command is the constant vector $[0, 0, 1]$, so the policy must decide on its own where to move. The fraction of exploration episodes grows with a performance-based curriculum coefficient [30], [31], which increases when the robot covers at least 80 % of the maze in a goal episode or 60 % in an exploration episode, and decreases on low coverage or early termination. The same coefficient ramps up the shaping penalties, so that they do not suppress the learning of exploration early in training. A second coefficient grows the same way and controls the fraction of the maze’s walls that are present, so the policy first learns to walk and follow goals in open spaces before it faces more and more complicated mazes. At the end of training, effectively all episodes are exploration episodes, and the deployed policy never depends on an external goal.

G. Domain Randomization

To enable sim-to-real transfer to the real robot, we apply extensive domain randomization during training. We randomize the robot model (trunk mass, center of mass, feet sizes, collision geometry, joint damping, armature, and friction), the environment (gravity, ground friction, and contact compliance), the actuation (motor strength and action delays), and the initial state of the robot. We further add noise to all proprioceptive observations, distort the height samples with noise, tilts, and delays of up to 60 ms, perturb the position estimate by up to ± 0.20 m, corresponding to one memory cell, and push the robot with velocities of up to 0.3 m/s.

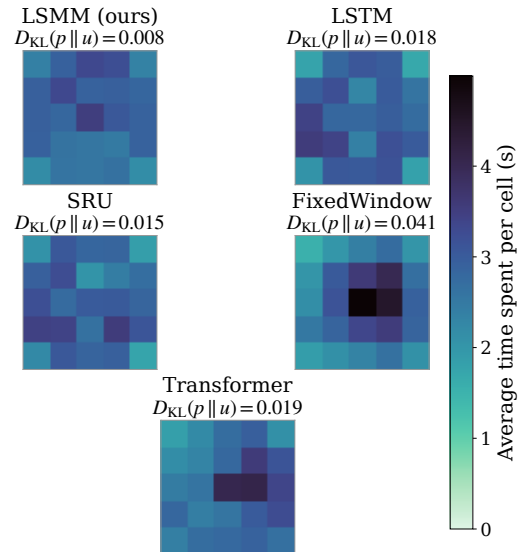


Fig. 4: Average number of visits per maze cell across all evaluation episodes. $D_{KL}(p \parallel u)$ measures the divergence between the normalized visit distribution p and a uniform distribution u , where lower values mean more even exploration. LSMM spreads its visits most evenly over mazes.

IV. EXPERIMENTAL RESULTS

Our experiments evaluate how LSMM explores compared to unstructured memory architectures, what the different memories actually retain about the environment, and whether the learned behavior transfers to the real robot.

We compare LSMM against an LSTM, a Spatially-Enhanced Recurrent Unit (SRU) [20], a Transformer, and an MLP that receives a fixed window of stacked past observations. All baselines receive the same observations, including the position estimate, are trained with the same auxiliary losses and the hyperparameters of Table I, and differ from LSMM only in their memory mechanism. To keep the comparison fair, we give every approach the same hardware budget: for each architecture, we choose the largest memory capacity for which training still fits on a single NVIDIA RTX 3090 with 24 GB of VRAM. This results in a hidden state of size 1000 for the LSTM, SRU and a context of the 10 most recent observations for the Transformer and the fixed-window MLP. Larger configurations exceed the available GPU memory. We train three seeds per architecture and evaluate each deterministically in 100 environments on previously unseen mazes.

A. Simulation

Figure 3 shows the fraction of explored cells at the end of an episode. All architectures reach a comparable coverage of 65.4 to 82.8 %, so pure exploration performance in our mazes does not require a spatial memory. The visit distributions in Figure 4 reveal a first difference, as LSMM spreads its visits most evenly over the maze, with a KL divergence to the uniform distribution of 0.008 compared to 0.015 to 0.018 to 0.019 for the SRU, LSTM and Transformer, while the

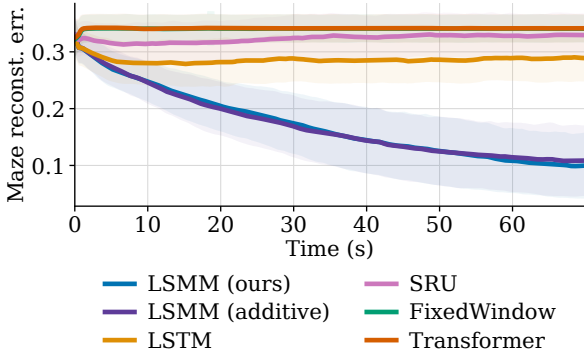


Fig. 5: Maze reconstruction error over the course of an episode. Lines show the mean over 100 unseen evaluation mazes and three seeds, shaded areas the standard deviation. LSMM is the only architecture that significantly reduces the error while exploring.

fixed window architecture concentrates on the cells around the start.

However, coverage alone does not show what a policy has remembered. Figure 5 tracks the maze reconstruction error over the episode, and LSMM is the only architecture that significantly reduces it while exploring, from 0.33 to 0.1, more than three times lower than any baseline. Figure 6 visualizes this difference, where the maze predicted by LSMM fills in wall by wall as the robot walks, until it closely matches the ground truth, while the baselines either never predict more than a few walls near their current position or hallucinate walls in free spaces. When visually inspecting the behaviour of the policies, the baselines explore more reactively and forget the areas they have seen before, which can even lead to circling around walls in loops. LSMM avoids returning to previously explored areas by building a persistent map of the maze. It rarely hallucinates walls and keeps true walls in memory as long as it has explored the cell enough. The policies act mostly restless and have a tendency to pace from cell to cell at $0.8 - 1.2 \text{ m/s}$ to maximize the intrinsic visitation reward.

To test whether the spatial memory can also support predictions beyond the maze layout, we train an auxiliary head, similar to the maze prediction head, to estimate locally varying environment properties, including gravity, ground friction, roughness, and ground damping. For each cell in a maze, we sample each property uniformly from a range scaled by the performance-based curriculum. Figure 8 shows consistently lower error than random guessing. Therefore, the policy can not only store where it has been but also aspects of the world it experienced there. We suspect that with a larger latent size for the memory cells, better estimations and even more predicted properties would be possible.

B. LSMM Memory Ablation

We ablate two design choices of the LSMM memory. First, HMF rescales the already written memory with a calibration tensor $\mathbf{F}_t$ at every write step. To evaluate its

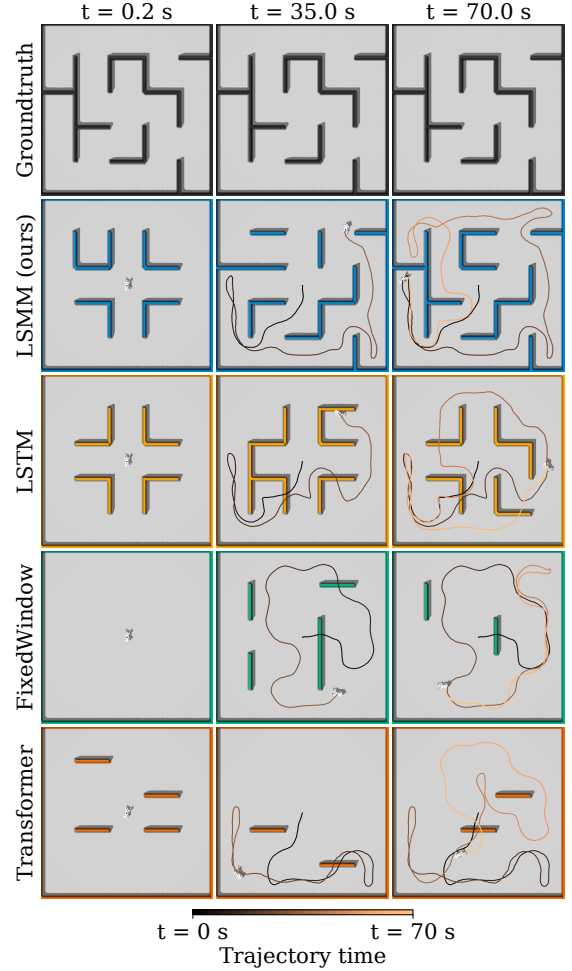


Fig. 6: Predicted maze layout at three times during an evaluation episode. The trajectory walked up to each time is overlaid, colored from black to orange by trajectory time. The prediction of LSMM fills in wall by wall as the robot explores and closely matches the ground truth at the end of the episode, while the baselines struggle to remember the walls in the areas they have been before.

relevance, Figure 5 compares LSMM with an additive variant that updates the memory only with the update tensor $\mathbf{G}_t$. Both variants reduce the reconstruction error similarly, which indicates that calibration is not required for remembering the maze layout in our setting. A likely reason is that the mazes are static during an episode, so once a wall is written to memory it never needs to be revised, and purely additive updates suffice.

Second, Figure 7 shows the reconstruction error for different numbers of memory channels. All configurations lie within each other’s standard deviation, and the mean error of a single channel differs only slightly from that of 3 and 10 channels without being significant. The reconstruction performance is thus largely insensitive to the memory size, suggesting that a single channel per cell already provides enough capacity to encode the wall layout.

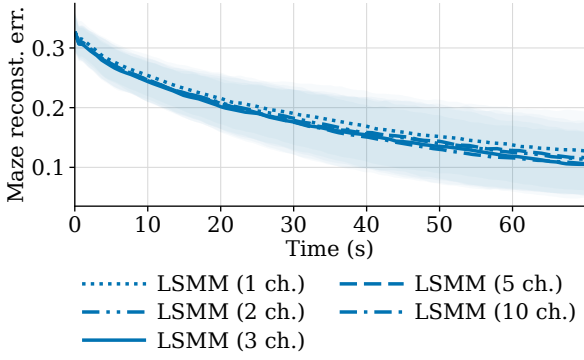


Fig. 7: Maze reconstruction error of LSMM for memory sizes of $50 \times 50 \times C$ with $C \in \{1, 2, 3, 5, 10\}$ channels. Lines show the mean over 100 unseen evaluation mazes and three seeds, shaded areas the standard deviation. All memory sizes perform comparably.

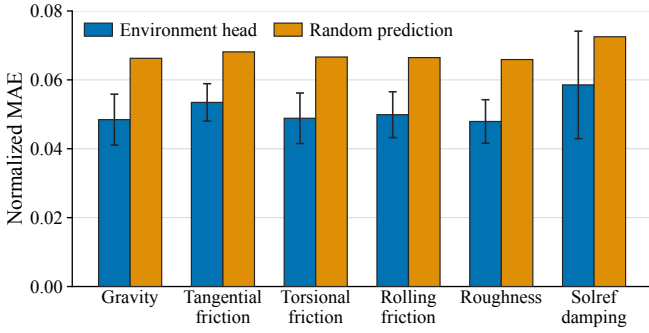


Fig. 8: Auxiliary prediction of spatially varying environment properties. Bars show the normalized mean absolute error over all visited cells at the final time step.

C. Real World

We deploy the LSMM policy on the Unitree Go2 in an unseen outdoor maze, as shown in Figure 1. We reduce the size of the maze from 10×10 m to 5×5 m due to constraints on the available space. We run the policy locally on the onboard CPU and estimate the robot’s pose from an AprilTag mounted on the robot, which is tracked by an external camera placed roughly eight meters above the ground (see Figure 1(a)). Using this pose estimate and the wall positions pre-measured in a fixed maze frame, we sample the height map around the robot at every time step, reproducing the height-scan observation from simulation without onboard terrain perception. The policy does not receive a goal or any an externally generated navigation command and explores the maze fully autonomously. It selects directions at intersections, traverses longer corridors, and continues exploring after returning to previously visited parts of the maze. The snapshot of the matrix memory in Figure 1 shows how the policy modifies the cells along its trajectory while cells outside the visited region retain their initial value. The same structure is visible in the time spent in each cell shown in Figure 9, where the regions that received no matrix memory updates are also the ones the robot never

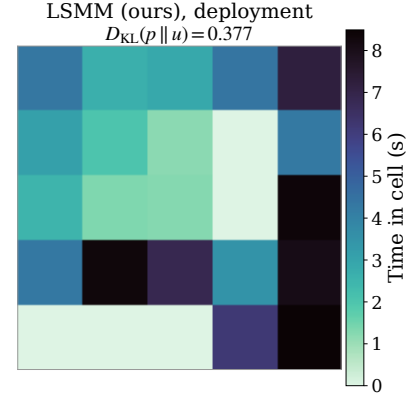


Fig. 9: Time spent in each cell by our deployed LSMM policy. Compared to the simulation results, this deployment run distributes its visits less uniformly and follows a more repetitive pattern, which is also visible in the matrix memory in Figure 1 (b).

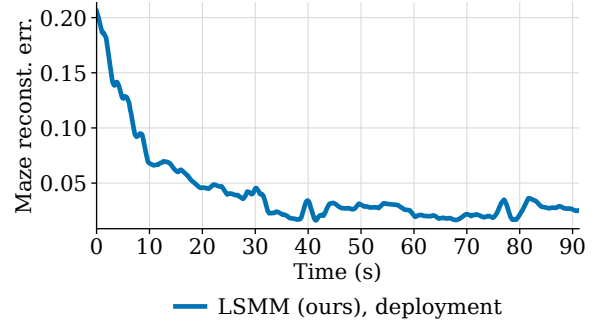


Fig. 10: Maze reconstruction error over deployment time of our LSMM policy. The line shows the reconstruction error of the deployment run shown in Figure 1. The policy reduces the error significantly within the first 30 s.

entered. However, the visit distribution covers enough of the maze to reduce the reconstruction error similarly to the simulation results, which is shown in Figure 10. As the maze is significantly smaller than in the simulation setup, the policy observes a higher fraction of it early on, which leads to a faster error reduction. During deployment runs, we noticed that the policy sometimes tries to cut corners or even touch the walls, which we could only reduce with strong penalties on cell centering and wall proximity. This is necessary in our more fragile real world maze setup, but with proper solid walls it might be beneficial to let the policy touch its surroundings as another form of re-orientation and grounding of its predictions, as long as the impact forces are mild.

V. LIMITATIONS & FUTURE WORK

The LSMM memory covers a fixed area that is aligned with the environment frame and sized to the maze, so exploring environments of unknown sizes would require to grow or

shift the memory with the robot. The update further depends on an external position estimate, and although training with ± 0.20 m of noise makes the policy tolerate imperfect localization, we do not yet estimate the position from onboard sensing alone. Finally, our mazes are flat and small enough that reactive strategies reach similar coverage, so we expect the benefit of a spatial memory for pure exploration to grow with the size of the environment. In future work, we want to scale LSMM to larger and rougher environments, estimate the position internally from the robot’s own sensing, and use the reconstructed map for downstream tasks such as navigating back to previously seen locations.

VI. CONCLUSION

We introduced LSMM, a spatially indexed latent memory that lets a single policy explore unseen mazes end-to-end, from onboard observations to joint-level actions. By applying the HMF update only to a local window centered on the robot’s position estimate, information about a location persists in the memory until the robot returns there. In our experiments, unstructured recurrent and finite-history baselines match LSMM in pure maze coverage, but they explore reactively and forget the layout they have seen. LSMM instead builds a persistent map of the maze while exploring, reduces the reconstruction error by a factor of three, spreads its visits most evenly over the maze, and additionally stores physical properties of the environment. We deployed the policy on a Unitree Go2 and demonstrated goal-free exploration of a real outdoor maze without an explicit map, planner, or waypoint commands.

APPENDIX

To ensure a fair comparison, we train all memory architectures in our experiments with the same learning hyperparameters, which are listed in Table I.

TABLE I: Hyperparameters used in all experiments.

Hyperparameter	Value
Training steps	$2.5 \cdot 10^9$
Batch size (envs $\times$ steps per env)	$5\,000 \times 20 = 100\,000$
Minibatch size / updates per batch	$25\,000 / 5$
Discount factor γ / GAE λ	$0.995 / 0.9$
Clipping range	0.2
Entropy coef. (linearly annealed)	$4 \cdot 10^{-4} \rightarrow 10^{-4}$
Actor optimizer, learning rate	Adam, $3 \cdot 10^{-4}$
Critic optimizer, learning rate	AdamW, $5 \cdot 10^{-4}$
Vel. estimator & wall predictor opt., lr	AdamW, $3 \cdot 10^{-4}$
Gradient clip norm	1.0
Weight decay (AdamW)	10^{-4}
Loss weights $\lambda_{\text{maze}} / \lambda_{\text{vel}}$	$0.5 / 0.1$

Table II lists all reward terms and their coefficients. All terms are scaled with their coefficients and summed up, and the result is clipped to be above zero before the constant alive bonus is added outside the clipping.

TABLE II: Reward terms and coefficients, with $[x]_+ = \max(x, 0)$ and $\bar{\cdot}$ the mean over joints. E/G: exploration/goal episodes only; n_{cell} : steps in the current cell; δ_i : offset from the cell center toward walled side i ; $n_{\text{pairs}}^{\text{air}}$: leg pairs with both feet airborne; t^{air} : foot air time.

Reward	Term	Coeff.
<i>Task</i>		
Cell visit (E)	$\exp(-n_{\text{cell}}/200)$	0.16
Maze coverage (E)	fraction of visited cells	0.1
Goal direction (G)	$\frac{1}{2}(1 + \cos(\mathbf{v}_{xy}, \mathbf{g}_t)) \min(\ \mathbf{v}_{xy}\ , 1)$	0.04
Goal reached (G)	$\mathbb{1}[d_{\text{goal}} < 0.25]$	0.0625
<i>Navigation shaping</i>		
Alive	1	0.03
Stalling	$-[0.1 - \ \mathbf{v}_{xy}\]_+$	0.1
Wall proximity	$-\mathbb{1}[d_{\text{wall}} < 0.35] e^{-5(d_{\text{wall}} - 0.15)}$	0.015
Cell centering	$-\sum_i [\delta_i]_+^2$	0.04
<i>Gait shaping</i>		
Vertical velocity	$-v_z^2$	0.04
Roll-pitch velocity	$-\ \boldsymbol{\omega}_{xy}\ ^2$	10^{-3}
Roll-pitch position	$-\ \boldsymbol{\phi}_{xy}\ ^2$	0.2
Joint position limits	$-\frac{[\mathbf{q} - \mathbf{q}^{\text{max}}]_+ + [\mathbf{q}^{\text{min}} - \mathbf{q}]_+}{2}$	0.8
Joint velocity limits	$-\frac{[\dot{\mathbf{q}}] - 0.9 \dot{\mathbf{q}}^{\text{max}}]_+}{2}$	0.1
Joint velocity	$-\dot{\mathbf{q}}^2$	$8 \cdot 10^{-6}$
Joint acceleration	$-\ddot{\mathbf{q}}^2$	10^{-7}
Joint torque	$-\frac{(\boldsymbol{\tau} / \boldsymbol{\tau}^{\text{max}})^2}{2}$	$6 \cdot 10^{-4}$
Power draw	$-\frac{[\boldsymbol{\tau} \dot{\mathbf{q}}]_+ / (\boldsymbol{\tau}^{\text{max}} \dot{\mathbf{q}}^{\text{max}})}{2}$	$6 \cdot 10^{-4}$
Action rate	$-\frac{(\mathbf{a}_t - \mathbf{a}_{t-1})^2}{2}$	0.06
Nominal position	$-\frac{(\mathbf{q} - \mathbf{q}^{\text{nom}})^2}{2}$	0.2
Base height	$-\min((h - h^{\text{nom}})^2, 1)$	0.6
Air time	$-[0.3 - t^{\text{air}}]_+$	0.06
Feet pair air	$-n_{\text{pairs}}^{\text{air}}$	0.01
Ground collision	$-\mathbb{1}[\text{body below ground}]$	2.0

REFERENCES

- [1] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, “Robot parkour learning,” in *Conference on Robot Learning (CoRL)*, 2023.
- [2] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, “Extreme parkour with legged robots,” in *RoboLetics: Workshop on Robot Learning in Athletics@ CoRL 2023*, 2023.
- [3] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, “Anymal parkour: Learning agile navigation for quadrupedal robots,” *Science Robotics*, vol. 9, no. 88, p. eadi7566, 2024.
- [4] M. Stasica, A. Bick, N. Bohlinger, O. Mohseni, M. J. A. Fritzsche, C. Hübner, J. Peters, and A. Seyfarth, “Bridge the gap: Enhancing quadruped locomotion with vertical ground perturbations,” in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025, pp. 13 370–13 370.
- [5] N. Bohlinger, J. Kinzel, D. Palenicek, L. Antczak, and J. Peters, “Gait in eight: Efficient on-robot learning for omnidirectional quadruped locomotion,” *arXiv preprint arXiv:2503.08375*, 2025.
- [6] N. Bohlinger, G. Czechmanowski, M. Krupka, P. Kicki, K. Walas, J. Peters, and D. Tateo, “One policy to run them all: an end-to-end learning approach to multi-embodiment locomotion,” *Conference on Robot Learning*, 2024.
- [7] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter, “Advanced skills by learning locomotion and local navigation end-to-end,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2497–2503.
- [8] B. Yamauchi, “A frontier-based approach for autonomous exploration,” in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97: Towards New Computational Principles for Robotics and Automation’*. IEEE, 1997, pp. 146–151.

- [9] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.
- [10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [11] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, eighth workshop on syntax, semantics and structure in statistical translation*, 2014, pp. 103–111.
- [12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [13] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4630–4637, 2022.
- [14] D. Li, B. Ai, N. Bohlinger, J. Peters, H. Su, and H. I. Christensen, "Rapid embodiment adaptation for quadrupedal locomotion," *arXiv preprint arXiv:2608.01506*, 2026.
- [15] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter, "Elevation mapping for locomotion and navigation using gpu," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2273–2280.
- [16] D. Aditya, J. Huang, N. Bohlinger, P. Kicki, K. Walas, J. Peters, M. Luperto, and D. Tateo, "Robust localization, mapping, and navigation for quadruped robots," in *2025 European Conference on Mobile Robots (ECMR)*. IEEE, 2025, pp. 1–8.
- [17] M. Mattamala, J. Frey, P. Libera, N. Chebrolu, G. Martius, C. Cadena, M. Hutter, and M. Fallon, "Wild visual navigation: Fast traversability learning via pre-trained models and online self-supervision," *Autonomous Robots*, vol. 49, no. 3, p. 19, 2025.
- [18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [19] F. Al-Hafez, G. Zhao, J. Peters, and D. Tateo, "Time-efficient reinforcement learning with stochastic stateful policies," in *International Conference on Learning Representations*, vol. 2024, 2024, pp. 49 532–49 568.
- [20] F. Yang, P. Frivik, D. Hoeller, C. Wang, C. Cadena, and M. Hutter, "Spatially-enhanced recurrent memory for long-range mapless navigation via end-to-end reinforcement learning," *International Journal of Robotics Research*, 2025.
- [21] M. Liu, D. Pathak, and A. Agarwal, "Locoformer: Generalist locomotion via long-context adaptation," in *9th Annual Conference on Robot Learning*, 2025.
- [22] H. Le, D. Nguyen, K. Do, S. Gupta, and S. Venkatesh, "Stable hadamard memory: Revitalizing memory-augmented agents for reinforcement learning," in *International Conference on Learning Representations*, vol. 2025, 2025, pp. 73 455–73 472.
- [23] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," in *International Conference on Learning Representations*, 2018.
- [24] J. F. Henriques and A. Vedaldi, "Mapnet: An allocentric spatial memory for mapping environments," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8476–8484.
- [25] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," *arXiv preprint arXiv:2004.05155*, 2020.
- [26] C. Zhang, J. Jin, J. Frey, N. Rudin, M. Mattamala, C. Cadena, and M. Hutter, "Resilient legged local navigation: Learning to traverse with compromised perception end-to-end," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 34–41.
- [27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [28] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *Proceedings of Robotics: Science and Systems (RSS)*, 2018.
- [29] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
- [30] B. Ai, L. Dai, N. Bohlinger, D. Li, T. Mu, Z. Wu, K. Fay, H. I. Christensen, J. Peters, and H. Su, "Towards embodiment scaling laws in robot locomotion," *Conference on Robot Learning (CoRL)*, 2025.
- [31] N. Bohlinger and J. Peters, "Multi-embodiment locomotion at scale with extreme embodiment randomization," *arXiv preprint arXiv:2509.02815*, 2025.