END-TO-END EGOSPHERIC SPATIAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Spatial memory, or the ability to remember and recall specific locations and objects, is central to autonomous agents' ability to carry out tasks in real environments. However, most existing artificial memory modules have difficulty recalling information over long time periods and are not very adept at storing spatial information. We propose a parameter-free module, Egospheric Spatial Memory (ESM), which encodes the memory in an ego-sphere around the agent, enabling expressive 3D representations. ESM can be trained end-to-end via either imitation or reinforcement learning, and improves both training efficiency and final performance against other memory baselines on both drone and manipulator visuomotor control tasks. The explicit egocentric geometry also enables us to seamlessly combine the learned controller with other non-learned modalities, such as local obstacle avoidance. We further show applications to semantic segmentation on the ScanNet dataset, where ESM naturally combines image-level and map-level inference modalities. Through our broad set of experiments, we show that ESM provides a general computation graph for embodied spatial reasoning, and the module forms a bridge between real-time mapping systems and differentiable memory architectures.

1 INTRODUCTION

Egocentric spatial memory is central to our understanding of spatial reasoning in biology (Klatzky, 1998; Burgess, 2006), where an embodied agent constantly carries with it a local map of its surrounding geometry. Such representations have particular significance for action selection and motor control (Wang et al., 2018; Hinman et al., 2019). For robotics and embodied AI, the benefits of a persistent local spatial memory are also clear. Such a system has the potential to run indefinitely, and bypass both the memory and runtime complexities of large scale world-centric mapping. Peters et al. (2001) propose an EgoSphere as being a particularly suitable representation for robotics, and more recent works have utilized ego-centric formulations for planar robot mapping (Fankhauser et al., 2014), drone obstacle avoidance (Fragoso et al., 2018) and mono-to-depth from video (Liu et al., 2019).

In parallel with these explicit ego-centric mapping systems, a new paradigm of differentiable memory architectures has arisen, where an explicit memory bank is augmented to a neural network, which can then learn read and write operations (Weston et al., 2014; Graves et al., 2014; Sukhbaatar et al., 2015). When compared to Recurrent Neural Networks (RNNs), the persistent memory circumvents issues of vanishing or exploding gradients, enabling solutions to complex long-horizon tasks. These have also been applied to visuomotor control and navigation tasks (Wayne et al., 2018), surpassing baselines such as the ubiquitous Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997).

We focus on the intersection of these two branches of research, and propose Egospheric Spatial Memory (ESM), a parameter-free module which encodes the memory in an ego-sphere around the agent. To the best of our knowledge, ESM is the first end-to-end trainable egocentric memory with full panoramic representation, enabling direct encoding of the surrounding scene in a 2.5D image.

We emphasize the suitability of an ego-centric mapping computation graph for propagating gradients over long time horizons in an end-to-end manner, enabling an ego-centric memory of learned features. We demonstrate the superiority of learning features through the ESM module on both target shape reaching and object segmentation tasks. For other visuomotor control tasks, we show that even without learning features through the module, and instead directly projecting image color values into memory, ESM consistently outperforms other memory baselines.

Through these experiments, we show that the applications of our parameter-free ESM module are widespread, where it can either be dropped into existing pipelines as a non-learned module, or end-to-end trained in a larger computation graph, depending on the task requirements.

2 Related Work

2.1 Mapping

Geometric mapping is a mature field, with many solutions available for constructing high quality maps. Such systems typically maintain an allocentric map, either by projecting points into a global world co-ordinate system (Newcombe et al., 2011a; Whelan et al., 2015), or by maintaining a certain number of keyframes in the trajectory history (Newcombe et al., 2011b; Zhou et al., 2018; Bloesch et al., 2018; Clark et al., 2018). If these systems are to be applied to life-long embodied AI, then strategies must then be adopted in order to effectively select the parts of the map which are useful, and discard the rest from memory (Cadena et al., 2016). Taking CodeSLAM (Bloesch et al., 2018) as an example, keyframe removal strategies are needed to prevent the map size from growing indefinitely.

For robotics applications, prioritizing geometry in the immediate vicinity is a sensible prior. Rather than taking a world-view to map construction, such systems often formulate the mapping problem in a purely ego-centric manner, performing continual re-projection to the newest frame and pose with fixed-sized storage. Peters et al. (2001) outline an EgoSphere memory structure, with indexing via polar and azimuthal angles, with applications to humanoid robotics. Fankhauser et al. (2014) use ego-centric height maps, and demonstrate on a quadrupedal robot walking over obstacles. Cigla et al. (2017) use per-pixel depth Gaussian Mixture Models (GMMs) to maintain an ego-cylinder of belief around a drone, with applications to collision avoidance (Fragoso et al., 2018). In a different application, Liu et al. (2019) learn to predict depth images from a sequence of RGB images, again using ego reprojections. These systems are designed to represent only at the level of depth and RGB features. For mapping more expressive implicit features via end-to-end training, a fully differentiable long-horizon computation graph is required. Any computation graph which satisfies this requirement is generally referred to as memory in the neural network literature.

2.2 Memory

The concept of memory in neural networks is deeply coupled with recurrence. Naive recurrent networks have vanishing and exploding gradient problems (Hochreiter, 1998), which LSTMs (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Cho et al., 2014) mediate using additive gated structures. More recently, dedicated differentiable memory blocks have become a popular alternative. Weston et al. (2014) applied Memory Networks (MemNN) to question answering, using hard read-writes and separate training of components. Graves et al. (2014) and Sukhbaatar et al. (2015) instead made the read and writes 'soft' with the proposal of Neural Turing Machines (NTM) and End-to-End Memory Networks (MemN2N) respectively, enabling joint training with the controller. Other works have since conditioned dynamic memory on images, for tasks such as visual question answering (Xiong et al., 2016) and object segmentation (Oh et al., 2019). Another distinct but closely related approach is self attention (Vaswani et al., 2017). These approaches also use key-based content retrieval, but do so on a history of previous observations with adjacent connectivity. Despite the lack of geometric inductive bias, recent results demonstrate the amenability of general memory (Wayne et al., 2018) and attention (Parisotto et al., 2019) to visuomotor control and navigation tasks.

Other authors have explored the intersection of network memory and spatial mapping for navigation, but have generally been limited to 2D aerial-view maps, focusing on planar navigation tasks. Gupta et al. (2017) used an implicit ego-centric memory which was updated with warping and confidence maps for discrete action navigation problems. Parisotto & Salakhutdinov (2017) proposed a similar setup, but used dedicated learned read and write operations for updates, and tested on simulated Doom environments. Without consideration for action selection, Henriques & Vedaldi (2018) proposed a similar system, but instead used an allocentric formulation, and tested on free-form trajectories of real images. Zhang et al. (2018) propose a similar system, but with the inclusion of loop closure. Our memory instead focuses on local perception, with the ability to represent detailed 3D geometry in all directions around the agent. The benefits of our module are complementary to existing 2D methods, which instead focus on occlusion aware planar understanding suitable for navigation.

3 METHOD

In this section, we describe the formulation of the memory module. The module operates as an Extended Kalman Filter (EKF), with an egosphere image $\mu_t \in \mathbb{R}^{h_s \times w_s \times (2+1+n)}$ and its diagonal covariance $\Sigma_t \in \mathbb{R}^{h_s \times w_s \times (1+n)}$ representing the state. The egosphere image consists of 2 channels for the polar and azimuthal angles, 1 for radial depth, and n for encoded features. The angles are not included in the covariance, as their values are implicit in the egosphere image pixel indices. The covariance only represents the uncertainty in depth and features at these fixed equidistant indices. Image measurements are assumed to be captured from projective depth cameras, which similarly store 1 channel for depth and n for encoded features. We also assume incremental agent pose measurements $u_t \in \mathbb{R}^6$ with covariance $\Sigma_{u_t} \in \mathbb{R}^{6 \times 6}$ are available, in the form of a translation and rotation vector. The algorithm overview is presented in Algorithm 1.

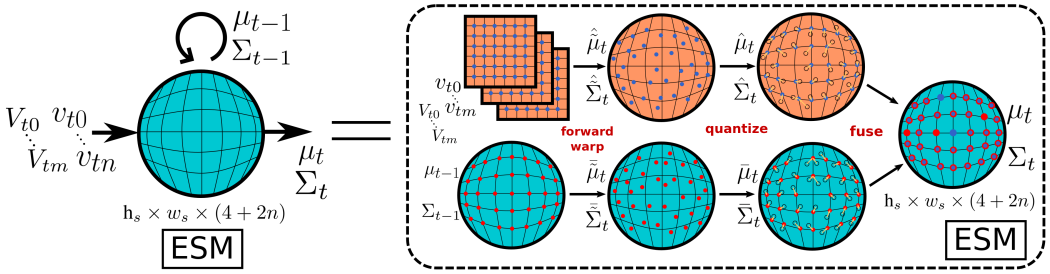

Figure 1: Overview of the ESM module. The module consists of projection and quantization steps, used to bring the belief from the previous agent frame to the current agent frame. The lefthand schematic, used throughout the rest of the paper, is shorthand for the entire righthand schematic.

First, the motion step takes the state from the previous frame μ_{t-1}, Σ_{t-1} and transforms this into a predicted state for the current frame $\bar{\mu}_t, \bar{\Sigma}_t$ via functions f_m, F_m and the incremental pose measurement u_t with covariance Σ_{u_t}. Then in the observation step, we use measured visual features $(v_{ti} \in \mathbb{R}^{h_{vi} \times w_{vi} \times (1+n)})_{i \in \{1,\dots,m\}}$ with diagonal covariances $(V_{ti} \in \mathbb{R}^{h_{vi} \times w_{vi} \times (1+n)})_{i \in \{1,\dots,m\}}$ originated from m arbitrary vision sensors, and associated pose measurements $(p_{ti} \in \mathbb{R}^6)_{i \in \{1,\dots,m\}}$ with covariances $(P_{ti} \in \mathbb{R}^{6 \times 6})_{i \in \{1,\dots,m\}}$, to produce a new observation of the state $\hat{\mu}_t \in \mathbb{R}^{h_s \times w_s \times (2+1+n)}$, again with diagonal covariance $\hat{\Sigma}_t \in \mathbb{R}^{h_s \times w_s \times (1+n)}$, via functions f_o and F_o. The measured poses also take the form of a translation and rotation vector.

Algorithm 1: ESM Step

Given: f_m, F_m, f_o, F_o
$\bar{\mu}_t = f_m(u_t, \mu_{t-1})$
$\bar{\Sigma}_t = F_m(u_t, \mu_{t-1}, \Sigma_{t-1}, \Sigma_{u_t})$
$\hat{\mu}_t = f_o((v_{ti}, p_{ti})_{i \in I})$
$\hat{\Sigma}_t = F_o((v_{ti}, p_{ti}, V_{ti}, P_{ti})_{i \in I})$
$K_t = \bar{\Sigma}_t \left[\bar{\Sigma}_t + \hat{\Sigma}_t\right]^{-1}$
$\mu_t = \bar{\mu}_t + K_t[\hat{\mu}_t - \bar{\mu}_t]$
$\Sigma_t = [I - K_t]\,\bar{\Sigma}_t$
return μ_t, Σ_t

Finally, the update step takes our state prediction $\bar{\mu}_t, \bar{\Sigma}_t$ and state observation $\hat{\mu}_t, \hat{\Sigma}_t$, and fuses them to produce our new state belief μ_t, Σ_t. We spend the remainder of this section explaining the form of the constituent functions. All functions in Algorithm 1 involve re-projections across different image frames, using forward warping. Functions f_m, F_m, f_o and F_o are therefore all built using the same core functions. While the re-projections could be solved using a typical rendering pipeline of mesh construction followed by rasterization, we instead choose a simpler approach and directly quantize the pixel projections with variance-based image smoothing to fill in quantization holes. An overview of the projection and quantization operations for a single ESM update step is shown in Fig. 1.

3.1 FORWARD WARPING

Forward warping projects ordered equidistant homogeneous pixel co-ordinates pc_{f1} from frame f_1 to non-ordered non-equidistant homogeneous pixel co-ordinates $\tilde{pc}_{f2}$ in frame f_2. We use $\tilde{\mu}_{f2} = \{\tilde{\phi}_{f2}, \tilde{\theta}_{f2}, \tilde{d}_{f2}, \tilde{e}_{f2}\}$ to denote the loss of ordering following projection from $\mu_{f1} = \{\phi_{f1}, \theta_{f2}, d_{f1}, e_{f2}\}$, where ϕ, θ, d and e represent polar angles, azimuthal angles, depth and encoded features respectively. We only consider warping from projective to omni cameras, which corresponds

to functions f_o, F_o, but the omni-to-omni case as in f_m, F_m is identical except with the inclusion of another polar co-ordinate transformation.

The encoded features are assumed constant during projection $\tilde{e}_{f2} = e_{f1}$. For depth, we must transform the values to the new frame in polar co-ordinates, which is a composition of a linear transformation and non-linear polar conversion. Using the camera intrinsic matrix K_1, the full projection is composed of a scalar multiplication with homogeneous pixel co-ordinates pc_{f1}, transformation by camera inverse matrix K_1^{-1} and frame-to-frame T_{12} matrices, and polar conversion f_p:

$$\{\tilde{\phi}_{f2}, \tilde{\theta}_{f2}, \tilde{d}_{f2}\} = f_p(T_{12}K_1^{-1}[pc_{f1} \odot d_{f1}]) \tag{1}$$

Combined, this provides us with both the forward warped image $\tilde{\mu}_{f2} = \{\tilde{\phi}_{f2}, \tilde{\theta}_{f2}, \tilde{d}_{f2}, \tilde{e}_{f2}\}$, and the newly projected homogeneous pixel co-ordinates $\tilde{pc}_{f2} = \{k_{ppr}\tilde{\phi}_{f2}, k_{ppr}\tilde{\theta}_{f2}, \mathbf{1}\}$, where k_{ppr} denotes the pixels-per-radian resolution constant. The variances are also projected using the full analytic Jacobians, which are efficiently implemented as tensor operations, avoiding costly autograd usage.

$$\hat{\tilde{\Sigma}}_2 = J_V V_1 J_V^T + J_P P_{12} J_P^T \tag{2}$$

3.2 Quantization, Fusion and Smoothing

Following projection, we first quantize the floating point pixel coordinates $\tilde{pc}_{f2}$ into integer pixel co-ordinates pc_{f2}. This in general leads to quantization holes and duplicates. The duplicates are handled with a variance conditioned depth buffer, such that the closest projected depth is used, provided that it's variance is lower than a set threshold. This in general prevents highly uncertain close depth values from overwriting highly certain far values. We then perform per pixel fusion based on lines 6 and 7 in Algorithm 1 provided the depths fall within a set relative threshold, otherwise the minimum depth with sufficiently low variance is taken. This again acts as a depth buffer.

Finally, we perform variance based image smoothing, whereby we treat each $N \times N$ image patch $(\mu_{k,l})_{k\in\{1,..,N\},l\in\{1,..,N\}}$ as a collection of independent measurements of the central pixel, and combine their variance values based on central limit theory, resulting in smoothed values for each pixel in the image $\mu_{i,j}$. Although we use this to update the mean belief, we do not smooth the variance values, meaning projection holes remain at prior variance. This prevents the smoothing from distorting our belief during subsequent projections, and makes the smoothing inherently local to the current frame only. The smoothing formula is as follows, with variance here denoted as σ^2:

$$\mu_{i,j} = \frac{\sum_k \sum_l \mu_{k,l} \cdot \sigma_{k,l}^{-2}}{\sum_k \sum_l \sigma_{k,l}^{-2}} \tag{3}$$

Given that the quantization is a discrete operation, we cannot compute it's analytic jacobian for uncertainty propagation. We therefore approximate the added quantization uncertainty using the numerical pixel gradients of the newly smoothed image $G_{i,j}$, and assume additive noise proportional to the x and y quantization distances $\Delta pc_{i,j}$:

$$\Sigma_{i,j} = \tilde{\Sigma}_{i,j} + G_{i,j}\Delta pc_{i,j} \tag{4}$$

3.3 Neural Network Integration

The ESM module can be integrated anywhere into a wider CNN stack, forming an Egospheric Spatial Memory Network (ESMN). Throughout this paper we consider two variants, ESMN and ESMN-RGB, see Figure 2. ESMN-RGB is a special case of ESMN, where RGB features are directly projected into memory, while ESMN projects CNN encoded features into memory. The inclusion of polar angles, azimuthal angles and depth means the full relative polar coordinates are explicitly represented for each pixel in memory. Although the formulation described in Algorithm 1 and Fig 1 allows for m vision sensors, the experiments in this paper all involve only a single acquiring sensor, meaning $m = 1$. We also only consider cases with constant variance in the acquired images $V_t = k_{var}$, and so we omit the variance images from the ESM input in Fig 2 for simplicity. For baseline approaches, we compute an image of camera-relative coordinates via K^{-1}, and then concatenate this to the RGB image along with the tiled agent incremental pose before input to the networks. All values are normalized to $0 - 1$ before passing to convolutions, based on the permitted range for each channel.

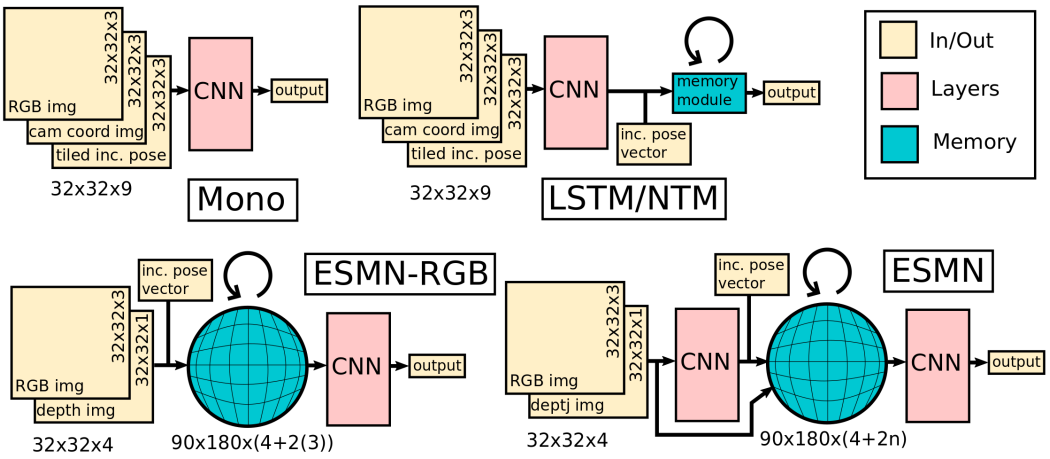

Figure 2: High level schematics of the ESM-integrated network architectures ESMN-RGB and ESMN, as well as other baseline architectures used in the experiments, Mono, LSTM and NTM.

4 Experiments

The goal of our experiments is to show the wide applicability of ESM to different embodied 3D learning tasks, where ESM outperforms existing memory baselines. We test two different applications:

1. Image-to-action learning for multi-DOF control (Sec 4.1). Here we consider drone and robot manipulator target reacher tasks using either ego-centric or scene-centric cameras. We then assess the ability for ESMN policies to generalize between these different camera modalities, and assess the utility of the ESM geometry for obstacle avoidance. We train policies both using imitation learning (IL) and reinforcement learning (RL).
2. Object segmentation (Sec 4.2). Here we explore the task of constructing a semantic map, and the effect of changing the ESM module location in the computation graph on performance.

4.1 Multi-DOF Visuomotor Control

While ego-centric cameras are typically used when learning to navigate planar scenes from images (Mnih et al., 2016; Jaderberg et al., 2016; Zhu et al., 2017; Gupta et al., 2017; Parisotto & Salakhutdinov, 2017), *static* scene-centric cameras are the de facto when learning multi-DOF controllers for robot manipulators (Levine et al., 2016; James et al., 2017; Matas et al., 2018; James et al., 2019b). We consider the more challenging and less explored setup of learning multi-DOF visuomotor controllers from ego-centric cameras, and also from *moving* scene-centric cameras. LSTMs are the de facto memory architecture in the RL literature (Mnih et al., 2016; Jaderberg et al., 2016; Espeholt et al., 2018; Kapturowski et al., 2018), making this a suitable baseline. NTMs represent another suitable baseline, which have outperformed LSTMs on visual navigation tasks (Wayne et al., 2018). Many other works exist which outperform LSTMs for planar navigation in 2D maze-like environments (Gupta et al., 2017; Parisotto & Salakhutdinov, 2017; Henriques & Vedaldi, 2018), but the top-down representation means these methods are not readily applicable to our multi-DOF control tasks. LSTM and NTM are therefore selected competitive baselines for comparison.

4.1.1 Imitation Learning

For our imitation learning experiments, we test the utility the ESM module on two simulated visual reacher tasks, which we refer to as Drone Reacher (*DR*) and Manipulator Reacher (*MR*). Both are implemented using the CoppeliaSim robot simulator (Rohmer et al., 2013), and its Python extension PyRep (James et al., 2019a). We implement DR ourselves, while MR is a modification of the reacher task in RLBench (James et al., 2020). Both tasks consist of 3 targets placed randomly in a simulated arena, and colors are newly randomized for each episode. The targets consist of a cylinder, sphere, and "star", see Figure 3.

In both tasks, the target locations remain fixed for the duration of an episode, and the agent must continually navigate to newly specified targets, reaching as many as possible in a fixed time frame of 100 steps. The targets are specified to the agent either as RGB color values or shape class id, depending on the experiment. The agent does not know in advance which target will next be specified, meaning a memory of all targets and their location in the scene must be maintained for the full duration of an episode. Both environments have a single body-fixed camera, as shown in Figure 3, and also an external camera with freeform motion, which we use separately for different experiments.

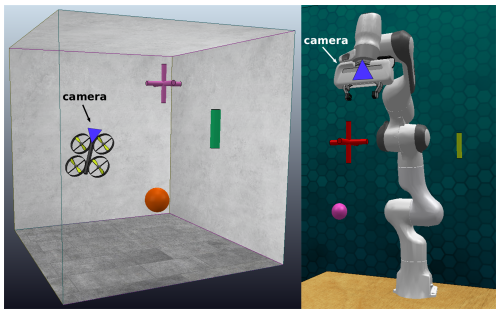

Figure 3: Visualization of (a) Drone Reacher and (b) Manipulator Reacher tasks.

For training, we generate an offline dataset of 100k 16-step sequences from random motions for both environments, and train the agents using imitation learning from known expert actions. Action spaces of joint velocities $\dot{q} \in \mathbb{R}^7$ and cartesian velocities $\dot{x} \in \mathbb{R}^6$ are used for *MR* and *DR* respectively. Expert translations move the end-effector or drone directly towards the target, and expert rotations rotate the egocentric camera towards the target via shortest rotation. Expert joint velocities are calculated for linear end-effector motion via the manipulator Jacobian. For all experiments, we compare to baselines of single-frame, dual-stacked LSTM, and NTM, and condition the policy networks on both the image features and per-pixel relative cartesian co-ordinates. We also compare against a network trained on partial oracle omni-directional images, masked at unobserved pixels, which we refer to as Partial-Oracle-Omni (PO2), as well as random and expert policies. PO2 cannot see regions where the monocular camera has not looked, but it maintains a pixel-perfect memory of anywhere it has looked. Full details of the training setups are provided in Appendix A.1. The results for all experiments are presented in Table 1.

	Drone Reacher				Manipulator Reacher			
	Ego Acq		Freeform Acq		Ego Acq		Freeform Acq	
	Color	Shape	Color	Shape	Color	Shape	Color	Shape
Mono	0.6(0.7)	0.9(1.7)	2.4(5.0)	0.5(1.6)	1.8(1.5)	1.6(1.1)	0.1(0.2)	0.1(0.2)
LSTM 256	11.8(3.1)	4.9(3.3)	0.7(0.7)	0.4(0.6)	1.1(0.5)	0.1(0.3)	0.1(0.4)	0.1(0.2)
LSTM 1024	12.7(3.4)	4.1(2.3)	1.0(1.0)	0.6(0.8)	1.0(0.5)	0.1(0.2)	0.1(0.2)	0.1(0.4)
NTM	4.4(2.0)	2.5(1.9)	0.7(0.7)	0.5(0.6)	1.0(0.6)	0.2(0.4)	0.1(0.3)	0.1(0.2)
ESMN-RGB	**20.6**(7.3)	4.1(3.8)	**16.1**(12.7)	1.1(2.6)	**11.4**(5.1)	3.1(3.2)	**10.5**(5.7)	**0.9**(1.6)
ESMN	**20.8**(7.8)	**18.3**(6.4)	**16.6**(12.9)	**8.5**(11.2)	**11.7**(5.3)	**4.7**(4.0)	**11.0**(5.8)	**1.0**(1.2)
Random	0.1(0.2)	0.1(0.2)	0.1(0.2)	0.1(0.2)	0.1(0.2)	0.1(0.2)	0.1(0.2)	0.1(0.2)
PO2	21.0(8.6)	14.4(6.1)	19.1(12.7)	3.9(8.1)	13.0(5.9)	4.1(3.5)	12.5(6.1)	2.6(2.5)
Expert	21.3(8.4)	21.3(8.4)	21.3(8.4)	21.3(8.4)	16.5(5.2)	16.5(5.2)	16.5(5.2)	16.5(5.2)

Table 1: Final policy performances on the various drone reacher (DR) and manipulator reacher (MR) tasks, from egocentric acquired (Ego Acq) or freeform acquired (Freeform Acq) cameras, with the network conditioned on either target color or shape. The values indicate the mean number of targets reached in the 100 time-step episode, and the standard deviation, when averaged over 256 runs. ESMN-RGB stores RGB features in memory, while ESMN stores learnt features.

Ego-Centric Observations: In this configuration we take observations from body-mounted cameras. We can see in Table 1 that for both DR and MR, our module significantly outperforms other memory baselines, which do not explicitly incorporate geometric inductive bias. Clearly, the baselines have difficulty in optimally interpreting the stream of incremental pose measurements and depth. In contrast, ESM by design stores the encoded features in memory with meaningful indexing. The ESM structure ensures that the encoded features for each pixel are aligned with the associated relative polar translation, represented as an additional feature in memory. When fed to the post-ESM convolutions, action selection can then in principle be simplified to target feature matching, reading the associated relative translations, and then transforming to the required action space. A collection of short sequences of the features in memory for the various tasks are presented in Figure 4, with (a),

(b) and (d) coming from egocentric observations. In all three cases we see the agent reach one target by the third frame, before re-orienting to reach the next.

We also observe that ESMN-RGB performs well when the network is conditioned on target color, but fails when conditioned on target shape id. This is to be expected, as the ability to retrieve shape information from the memory is strongly influenced by the ESM resolution, quantization holes, and strong angular distortion. However, with ESMN we train end-to-end with a pre-ESM convolutional encoder, and we can see that the issues are alleviated, with the network learning to store useful network encoded features in memory. We show the activations of the 3 most dominate feature channels projected to RGB space during a short sequence in Figure 4 (a).

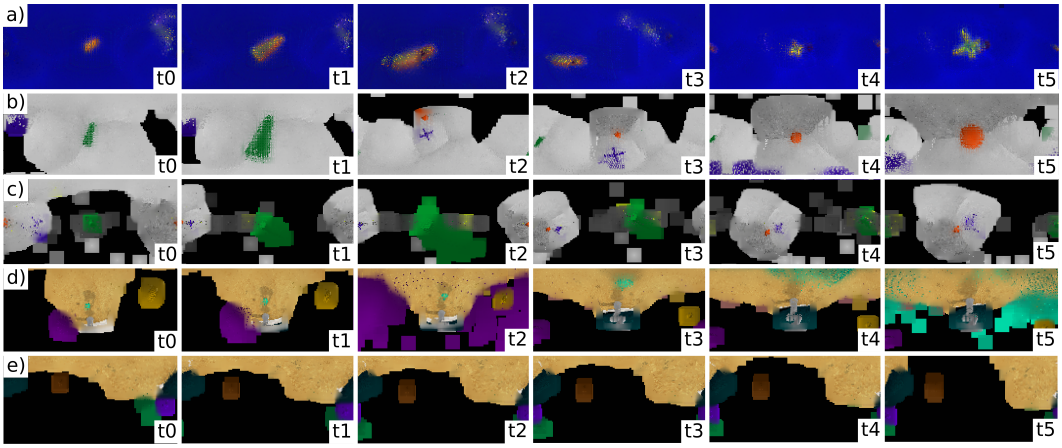

Figure 4: Sample trajectories through the memory for (a) ESMN on DR-Ego-Shape, and ESMN-RGB on (b) DR-Ego-Color, (c) DR-Freeform-Color, (d) MR-Ego-Color, (e) MR-Freeform-Color. The images each correspond to features in the full 180×90 memory at that particular timestep t.

Scene-Centric Observations: In this configuration we use images from cameras external to the agent. The poses of these cameras are randomized for each episode during training, and follow random freeform motion, with a rotational bias to face towards the centre of the scene. Here we explore the ability of ESM to generalize to unseen camera poses and motion. Again, we see that the baselines fail to learn successful policies, while ESM-augmented networks are able to solve the task. The memories in these tasks take on a different profile, as can be seen in Fig 4 (c) and (e). While the memories from egocentric observations always contain information in the memory image centre, where the most recent monocular frame projects with high density, this is not the case for projections from arbitrarily positioned cameras. External cameras can also move far from the agent, resulting in sparse projections into memory. The targets in Fig 4 (e) are all represented by only 1 or 2 pixels. The large apparent area in memory is a result of the variance-based smoothing, where the low-variance colored target pixels are surrounded by high-variance unobserved pixels in the ego-sphere.

Obstacle Avoidance: To further demonstrate the benefits of a local spatial geometric memory, we augment the standard drone reacher task with obstacles, see Figure 5. Rather than learning the avoidance in the policy, we exploit the interpretable geometric structure in ESM, and instead augment the policy output with a local avoidance component. We then compare targets reached and collisions for different avoidance baselines, and test these avoidance strategies on random, ESMN-RGB and expert target reacher policies. We see that the ESM geometry enables superior avoidance over using the most recent depth frame alone. The obstacle avoidance results are presented in Table 2, and further details of the experiment are presented in Appendix A.2.

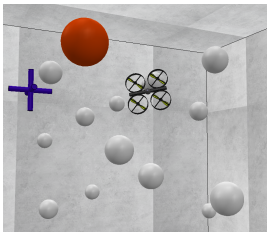

Figure 5: Avoidance task

Generalization performance: We now explore the extent to which policies trained from egocentric observations can generalize to cameras moving freely in the scene, and vice-versa. The results of these transfer learning experiments are presented in Table 3. Example image trajectories for both egocentric and free-form observations are presented in Figure 6.

		Policy					
		Random		ESMN-RGB		Expert	
		Reached	Collisions	Reached	Collisions	Reached	Collisions
Avoidance	No Avoidance	0.0(0.2)	29.4(27.9)	9.9(7.0)	64.8(67.4)	21.3(2.1)	121.7(15.8)
	Single Depth Frame	0.1(0.2)	9.6(11.8)	11.1(4.3)	10.8(13.0)	15.3(2.0)	7.7(6.9)
	ESM Depth Map	0.1(0.3)	**4.2**(7.1)	9.6(4.5)	**2.7**(5.5)	10.8(4.4)	**2.2**(3.8)
	Ground Truth	0.1(0.3)	0.1(0.5)	9.6(5.1)	0.0(0.0)	15.2(3.9)	0.0(0.0)

Table 2: Targets reached and number of collision for the obstacle avoidance drone task. Full details of this experiment are provided in Appendix A.2.

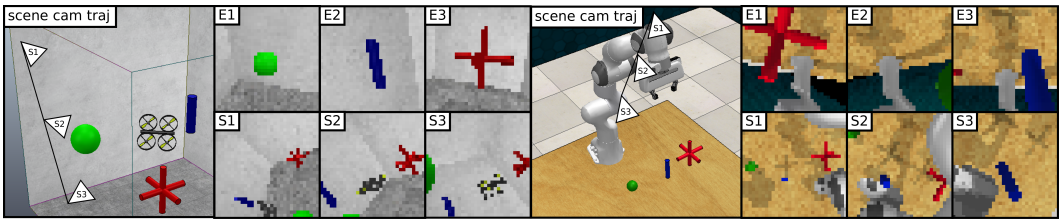

Figure 6: Example image sequences from both egocentric (E) and freeform (F) cameras, on both reacher tasks. The images are all time-aligned, and correspond to the same agent motion in the scene.

	Drone Reacher				Manipulator Reacher			
	Ego Acq		Scene Acq		Ego Acq		Scene Acq	
	Color	Shape	Color	Shape	Color	Shape	Color	Shape
LSTM 1024	12.7(3.4)	4.1(2.3)	1.0(1.0)	0.6(0.8)	1.0(0.5)	0.1(0.2)	0.1(0.2)	0.1(0.4)
Transfer	0.3(0.6)	0.5(0.7)	0.1(0.3)	0.3(0.7)	0.1(0.3)	0.0(0.2)	0.0(0.0)	0.0(0.2)
ESMN-RGB	20.6(7.3)	4.1(3.8)	16.1(12.7)	1.1(2.6)	11.4(5.1)	3.1(3.2)	10.5(5.7)	0.9(1.6)
Transfer	**20.3**(11.5)	**5.8**(4.7)	**15.3**(10.9)	0.8(1.6)	**11.2**(5.6)	**3.9**(4.3)	**9.8**(3.8)	1.1(1.3)
ESMN	20.8(7.8)	18.3(6.4)	16.6(12.9)	8.5(11.2)	11.7(5.3)	4.7(4.0)	11.0(5.8)	1.0(1.2)
Transfer	**10.4**(5.5)	0.7(0.9)	**8.9**(6.5)	1.0(1.4)	**7.5**(4.9)	**6.3**(6.2)	**6.8**(4.7)	0.6(1.0)

Table 3: Reacher performances both with and without camera transfer, using the same notation and setup as described in Table 1. Successful transfers are highlighted in bold.

4.1.2 Reinforcement Learning

Assuming expert actions in partially observable (PO) environments is inherently limited. It is not necessarily true that the best action always rotates the camera directly to the next target for example. In general, for finding optimal policies in PO environments, methods such as reinforcement learning (RL) must be used. We therefore train both ESMN-RGB and an LSTM baseline on a simpler variant of the MR task via DQN (Mnih et al., 2015). We refer to this variant as MR-Seq, where the manipulator must reach red, blue and then yellow spherical targets from egocentric observations, after which the episode terminates. The only other difference to MR is that MR-Seq uses 128×128 images as opposed to 32×32. ESMN-RGB again outperforms the LSTM baseline, learning to reach all three targets, while the LSTM policy only succeeds in reaching one. Full details of the RL setup and learning curves are given in Appendix A.3.

4.2 Object Segmentation

We now explore the suitability of the ESM module for object segmentation. One approach is to perform image-level segmentation only in individual monocular frames, and then perform probabilistic fusion when projecting into the map (McCormac et al., 2017). We refer to this approach as Mono. Another approach is to first construct an RGB map, and then pass this map as input to a network. This has the benefit of wider context, but lower resolution is necessary to store a large map in memory, meaning details can be lost. ESMN-RGB adopts this approach. Zhi et al. (2019) take a different approach, and combine monocular predictions with multi-view optimization to gain the benefits of wider surrounding context. With ESMN, we construct a network with both image-level

and map-level convolutions, which similarly offers the ability to combine monocular inference with the wider map context. These variants adopt the same architectures as shown in Fig 2, with the full networks specified in Appendix A.4. We do not attempt to quote state-of-the-art results, but rather to further demonstrate the wide applications of the ESM module, and to further explore the effect of placing the ESM module at different locations in the convolutional stack. We evaluate segmentation accuracy based on the predictions projected to the ego-centric map. With fixed network capacity between methods, we see that ESMN outperforms both baselines, see Table 4 for the results, and Figure 7 for example predictions in a ScanNet reconstruction. Further details of this experiment are given in Appendix A.4.

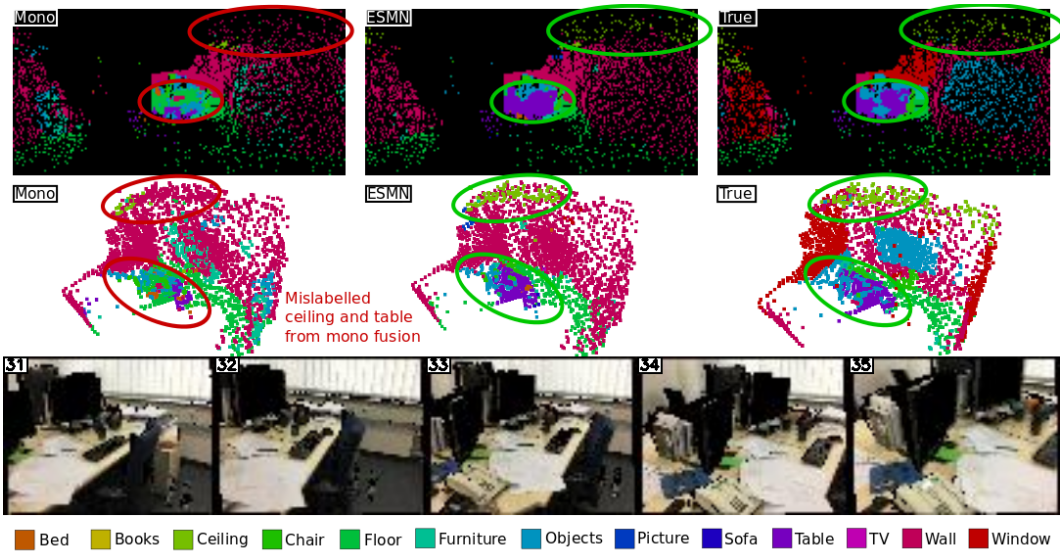

Figure 7: Some caption.

	Images 60×80		Images 120×160		Images 60×80	
	Memory 90×180		Memory 90×180		Memory 180×360	
	1-frame	16-frame	1-frame	16-frame	1-frame	16-frame
Mono	**54.5**(19.0)	55.1(13.6)	**54.9**(19.3)	55.6(13.9)	**54.6**(19.2)	55.3(13.7)
ESMN-RGB	51.4(19.6)	54.1(13.8)	51.7(19.3)	54.4(13.2)	**54.0**(19.2)	57.1(13.5)
ESMN	**55.0**(19.0)	**59.4**(12.7)	**55.3**(19.1)	**59.8**(13.0)	**55.2**(19.4)	**59.7**(12.9)

Table 4: Object segmentation results on the ScanNet test set for a monocular fusion baseline (Mono), as well as ESMN-RGB and ESMN.

5 Conclusion

Through a diverse set of demonstrations, we have shown that ESM represents a widely applicable computation graph, and trainable module for tasks requiring general spatial reasoning. When compared to other memory baselines for image-to-action learning, our module outperforms these dramatically when learning both from ego-centric and scene-centric images. One weakness of our method is that is assumes the availability of both incremental pose measurements of all scene cameras, and depth measurements, which is not a constraint of the other memory baselines. However, with the ever increasing ubiquity of commercial depth sensors, and with there being plentiful streams for incremental pose measurements including visual odometry, robot kinematics, and inertial sensors, we argue that such measurements are likely to be available in any real-world application of embodied AI. We leave it to future work to investigate the extent to which ESM performance deteriorates with highly uncertain real-world measurements, but with full uncertainty propagation and probabilistic per pixel fusion, ESM is well suited for accumulating noisy measurements in a principled manner.

REFERENCES

Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2560–2568, 2018.

Neil Burgess. Spatial memory: how egocentric and allocentric combine. *Trends in cognitive sciences*, 10(12):551–557, 2006.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Cevahir Cigla, Roland Brockers, and Larry Matthies. Gaussian mixture models for temporal depth fusion. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 889–897. IEEE, 2017.

Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J. Davison. Learning to solve nonlinear least squares for monocular stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. Robot-centric elevation mapping with uncertainty estimates. In *Mobile Service Robotics*, pp. 433–440. World Scientific, 2014.

Anthony T Fragoso, Cevahir Cigla, Roland Brockers, and Larry H Matthies. Dynamically feasible motion planning for micro air vehicles using an egocylinder. In *Field and Service Robotics*, pp. 433–447. Springer, 2018.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2017.

Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8476–8484, 2018.

James R Hinman, G William Chapman, and Michael E Hasselmo. Neuronal representation of environmental boundaries in egocentric coordinates. *Nature communications*, 10(1):1–8, 2019.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02): 107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

Stephen James, Marc Freese, and Andrew J Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019a.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019b.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Roberta L Klatzky. Allocentric and egocentric spatial representations: Definitions, distinctions, and interconnections. In *Spatial cognition*, pp. 1–17. Springer, 1998.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Chao Liu, Jinwei Gu, Kihwan Kim, Srinivasa G Narasimhan, and Jan Kautz. Neural rgb (r) d sensing: Depth and uncertainty from a video camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10986–10995, 2019.

Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *2017 IEEE International Conference on Robotics and automation (ICRA)*, pp. 4628–4635. IEEE, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136. IEEE, 2011a.

Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pp. 2320–2327. IEEE, 2011b.

Seoung Wug Oh, Joon-Young Lee, Ning Xu, and Seon Joo Kim. Video object segmentation using space-time memory networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9226–9235, 2019.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. *arXiv preprint arXiv:1910.06764*, 2019.

Richard Alan Peters, Kimberly E Hambuchen, Kazuhiko Kawamura, and D Mitchell Wilkes. The sensory ego-sphere as a short-term memory for humanoids. In *Proceedings of the IEEE-RAS international conference on humanoid robots*, number 1, pp. 451–459, 2001.

Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *International Conference on Intelligent Robots and Systems*. IEEE, 2013.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Cheng Wang, Xiaojing Chen, Heekyung Lee, Sachin S Deshmukh, D Yoganarasimha, Francesco Savelli, and James J Knierim. Egocentric coding of external items in the lateral entorhinal cortex. *Science*, 362(6417):945–949, 2018.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. Elastic-fusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *International conference on machine learning*, pp. 2397–2406, 2016.

Mengmi Zhang, Keng Teck Ma, Shih-Cheng Yen, Joo Hwee Lim, Qi Zhao, and Jiashi Feng. Egocentric spatial memory. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 137–144. IEEE, 2018.

Shuaifeng Zhi, Michael Bloesch, Stefan Leutenegger, and Andrew J Davison. Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11776–11785, 2019.

Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 822–838, 2018.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364. IEEE, 2017.

A Appendix

A.1 Multi-DOF Imitation Learning

A.1.1 Offline Datasets

The image sequences for the offline datasets are captured following random motion of the agent in both the DR and MR tasks, but known expert actions for each of the three possible targets in the scene are stored at every timestep. At training time, a new target is then randomly chosen from each of the three possibilities for each successive step in the unrolled time dimension in the training batch. This ensures maximum variation in sequences at train-time, despite a relatively small number of only 100k 16-frame sequences stored in the offline dataset. All images in the dataset are 32×32 resolution.

A.1.2 Training

All networks use a batch size of 16, an unroll size of 16, and are trained for 250k steps using an ADAM optimizer with $1e-4$ learning rate. None of the memory baselines cache the the internal state between training batches, and so the networks must learn to utilize the memory during the 16-frame unroll.

A.1.3 NETWORK ARCHITECTURES

The network architectures used in the imitation learning experiments are provided in Fig 8. Both LSTM baselines use dual stacked architectures. For NTM we use a similar variant to that used by Wayne et al. (2018), namely, we use sequential appending to the memory, and retroactive updates. Regarding the 16-frame unroll, we find that 16 steps is on average enough time to reach all targets once. In order to encourage generalization to longer sequences than only 16-steps, we limit the memory size to 10, and track the usage of these 10 memory cells such that subsequent writes can preferentially overwrite the least used of the 10 cells. This again is the same approach used by Wayne et al. (2018), which is one of very few works to successfully apply NTM-style architectures to image-to-action domains. Regarding image padding at the borders for input to the convolutions, the Mono and LSTM/NTM baselines use standard zero padding, whereas ESMN-RGB and ESMN pad the outer borders with the wrapped omni-directional image.

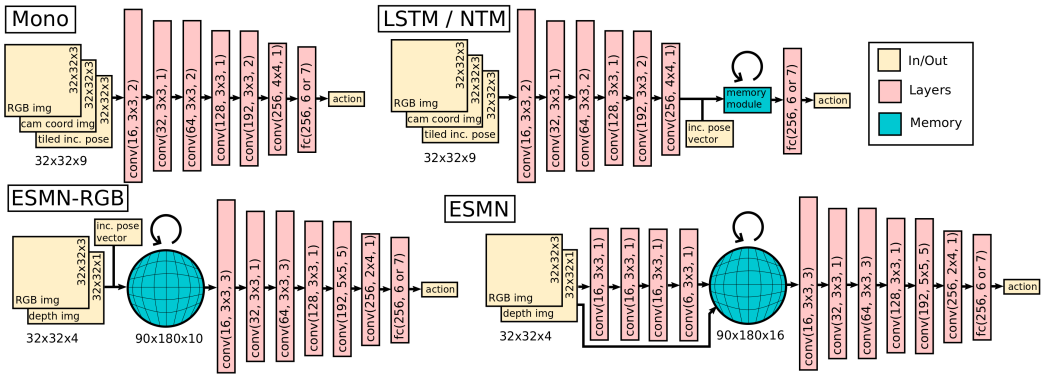

Figure 8: Image-to-action imitation learning network architectures for Mono, LSTM/NTM, ESMN-RGB and ESMN.

A.2 OBSTACLE AVOIDANCE

For this experiment, we increase the drone reacher environment by $2\times$ in all directions, resulting in an $8\times$ increase in volume. We then also add 20 spherical obstacles into the scene with radius $r = 0.1m$. For the avoidance, we consider a bubble around the agent with radius $R = 0.2$, and flag a collision whenever any part of an obstacle enters this bubble. Given the closest depth measurement available $d_{closest}$, the avoidance algorithm simply computes an avoidant velocity vector v_a whose magnitude is inversely proportional to the distance from collision, clipped to the maximum velocity $|v|_{max}$, Eq 5 shows the calculation for the avoidance vector magnitude. We run the avoidance controller at $10\times$ the rate of the ESM updates.

$$|v_a| = min\left[\frac{10^{-3}}{max\left[d_{closest} - R - r, 10^{-12}\right]^2}, |v|_{max}\right] \tag{5}$$

In order to prevent avoidant motion away from the targets to reach, we retrain the ESMN-RGB networks on the drone reacher task, but we train the network to also predict the full relative target location as an additional auxiliary loss. When evaluating on the obstacle avoidance task, we prevent depth values within a fixed distance of this predicted target location from influencing the obstacle avoidance. This has the negative effect of causing extra collisions when the agent erroneously predicts that the target is close, but it enables the agent to approach and reach the target without being pushed away by the avoidance algorithm.

A.3 MULTI-DOF REINFORCEMENT LEARNING

A.3.1 TRAINING

We train both ESMN-RGB and an LSTM baseline on a similar sequential target reacher task as defined in Section **??** via DQN (Mnih et al., 2015), where the manipulator must reach red, blue and then yellow targets from egocentric observations. We use (128×128) images in this experiment rather than (32×32) as used in the imitation learning experiments. We also use an unroll length of 8 rather than 16. We use discrete delta end-effector translations, with ± 0.05 meters for each axis, with no rotation (resulting in an action size of 6). We use a shaped reward of $r = -len(remaining_targets) - \|e - g\|_2$, where e and g are the gripper and current target translation respectively.

In order to make the RL more tractable, we use smaller network than used in the imitation learning experiments. Both methods use a Siamese network to process the RGB and camera coordinate inputs separately, and consist of 2 convolution (conv) layers with 16 and 32 channels (for each branch). We fuse these branches with a 32 channel conv and 1x1 kernel. The remainder of the architecture then follows similarly to the imitation learning experiments, but we instead use channel sizes of 64 throughout. The network output are 6 values corresponding to the Q-values for each action. Both methods use a learning rate of 0.001, target Q-learning $\tau = 0.001$, batch size of 128, and *leakyReLU* activations. We use an epsilon greedy strategy for exploration that is decayed over 100k training steps from 1 to 0.1. We show the average return during RL training on the sequential reacher task over 5 seeds in Fig 9.

A.4 OBJECT SEGMENTATION

A.4.1 DATASET

For the object segmentation experiment, we use both downsampled 60×80 and 120×160 images from the ScanNet dataset, which we first RGB-Depth align. We use a reduced dataset with frame skip of 30 to maximize diversity whilst minimizing dataset memory. Many sequences contain slow camera motion, resulting in adjacent frames which vary very little.

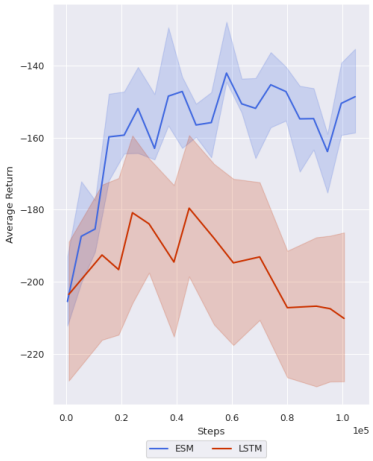

Figure 9: Average return during RL training on sequential reacher task over 5 seeds. Shaded region represent the min and max across trials.

A.4.2 NETWORK ARCHITECTURES

The Mono, ESMN-RGB and ESMN networks all exhibit a U-Net architecture, and output object segmentation predictions in an ego-sphere map. ESMN-RGB and ESMN do so with a U-Net connecting the ESM output to the final predictions, and Mono does so by projecting and probabilistically fusing the monocular segmentation predictions in a non-learnt manner. The network architectures are all presented in Fig 10. Regarding image padding at the borders for input to the convolutions, the Mono and LSTM/NTM baselines use standard zero padding, whereas ESMN pads the outer borders with the wrapped omnidirectional image.

A.4.3 TRAINING

For training, all losses are computed in the ego-sphere map frame of reference, either following convolutions for ESMN and ESMN-RGB, or following projection and probabilistic fusion for the Mono case. We compute ground-truth segmentation training target labels by projecting the ground truth monocular frames to form an ego-sphere target segmentation image. We chose this approach over computing the ground truth segmentations from the complete ScanNet meshes for implementational simplicity. All experiments use a batch size of 16, unroll size of 8 in the time dimension, and Adam optimizer with learning rate $1e-4$, trained for 250k steps.

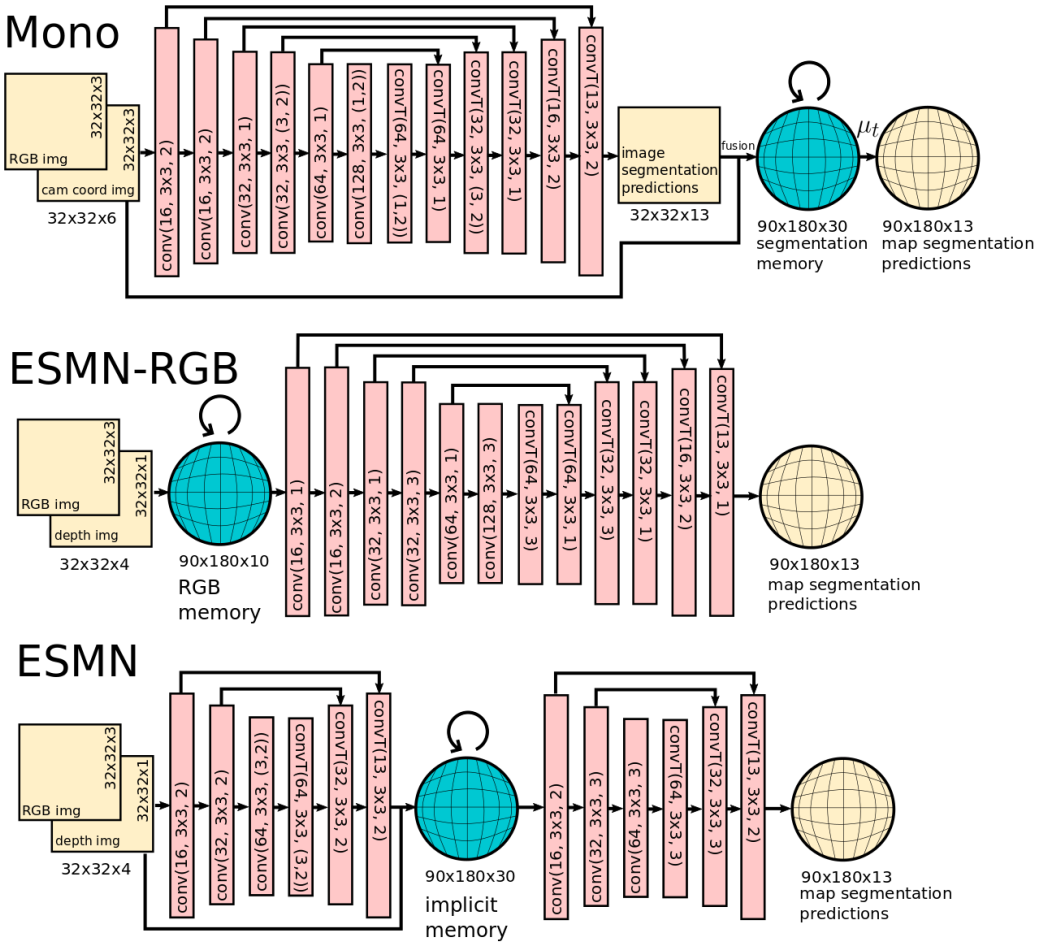

Figure 10: Object segmentation network architectures for Mono, ESMN-RGB and ESMN.