MapNet: An Allocentric Spatial Memory for Mapping Environments

João F. Henriques   Andrea Vedaldi
Visual Geometry Group, University of Oxford
{joao,vedaldi}@robots.ox.ac.uk

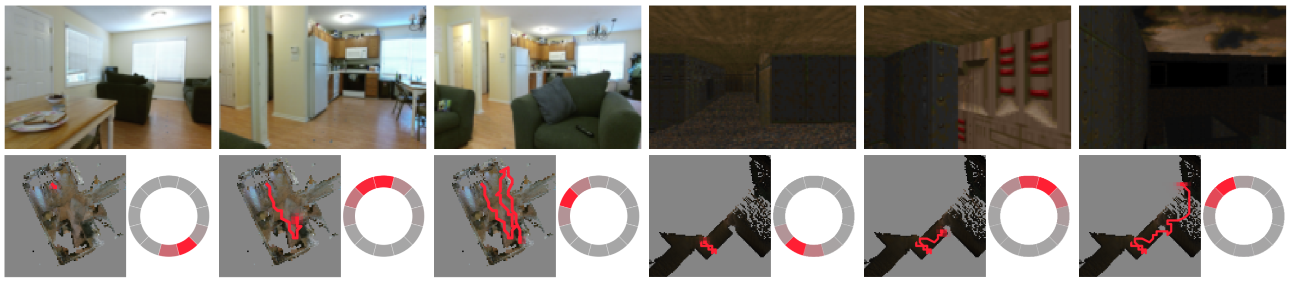

Figure 1: Results on real data (AVD [1], left) and game footage (ResearchDoom [14], right). Best seen in color. Our method predicts a joint heatmap of the camera position and orientation in absolute coordinates (shown separately). This is done online while creating a deep embedding of the surrounding environment, which was not seen during training.

Abstract

Autonomous agents need to reason about the world beyond their instantaneous sensory input. Integrating information over time, however, requires switching from an egocentric representation of a scene to an allocentric one, expressed in the world reference frame. It must also be possible to update the representation dynamically, which requires localizing and registering the sensor with respect to the world reference. In this paper, we develop a differentiable module that satisfies such requirements, while being robust, efficient, and suitable for integration in end-to-end deep networks. The module contains an allocentric spatial memory that can be accessed associatively by feeding to it the current sensory input, resulting in localization, and then updated using an LSTM or similar mechanism. We formulate efficient localization and registration of sensory information as a dual pair of convolution/deconvolution operators in memory space. The map itself is a 2.5D representation of an environment storing information that a deep neural network module learns to distill from RGBD input. The result is a map that contains multi-task information, different from classical approaches to mapping such as structure-from-motion. We present results using synthetic mazes, a dataset of hours of recorded gameplay of the classic game Doom, and the very recent Active Vision Dataset of real images captured from a robot.

1. Introduction

Machine learning nowadays plays a crucial role in most computer vision tasks. Compared to hand-crafted solutions, a key advantage of approaches such as deep learning is their ability to acquire automatically very efficient and robust representations of the data. Successes in image-centric tasks such as classification, segmentation, and object detection provide the clearest demonstration of the benefits of this approach [19, 6, 20]. Beyond static data, recurrent architectures such as LSTMs can be used to great effect in the analysis of temporal data streams as well [22].

Despite these successes, however, several aspects of image understanding remain difficult to approach directly using deep distributed representations. One of them is reasoning about 3D space and geometry, particularly in relation to large environments. Traditional SLAM techniques offer a reliable approach to the analysis of such data [15]. For example, these methods can build incrementally large maps of the world simply by observing video streams. Nevertheless, the *representations* built into these systems are still based on primitives such as 3D point clouds and image patches. While this may work well, it does not provide a natural fit for learnable representations such as deep neural networks. Given the success of the latter in so many areas of image analysis, it is then natural to ask whether deep, distributed, and effective representations of geometry are possible.

Such a representation may offer significant advantages compared to traditional approaches. For example, humans can effectively navigate small and large environments and yet are unlikely to build internally large-scale metric reconstructions of spaces akin to traditional SLAM systems [21]. Analogously, it may be possible to develop efficient and robust representations of geometry that are semi-quantitative, which may enable solving problems such as navigation just as well as traditional representations. Furthermore, such a representation may encode semantic information beyond the position of 3D points and the appearance of image patches. Such an approach may be particularly useful for autonomous agents, where it can reformulate the mapping component as a deep, learnable function, simplifying its integration with other components of the system that are likely to use the same formalism.

In this paper, we thus propose and study a new distributed representation of 3D environments that can be used in a deep learning context. The representation is used by a recurrent neural network to interpret an environment seen through a moving camera. The representation works as a mapping component, i.e. a dynamic spatial memory which is updated incrementally from the camera observations. This memory allows the recurrent network to remember places visited in the past, as well as to relocalize itself with respect to those.

Our design addresses two challenges. The first one is to allow the model to factor *egocentric* and *allocentric* information. One of the most difficult challenges of perception is in fact to understand that images are the result of scenes and objects that move (largely) independently of the observer. For a deep network, doing so is difficult because information paths between pixels and its internal representation are hardwired, whereas a given object or scene element is generally associated to different pixels in different video frames due to motion. To address this issue, we allow the feature extraction network to dynamically rewire with respect to an *allocentric spatial memory*. Rewiring is based on an estimate of the absolute position of the observer in the world.

The second challenge we address is to decide which information should be stored in the map. Focusing on environments that mostly extend across a ground plane, such as a building floor or outdoor streets, we propose to consider a 2.5D representation of the world, in which any information related to the vertical dimension is implicitly encoded as feature vectors in a dense 2D field representing the ground. Writing information to this field is based on the current estimated location of the observer and of a depth map measured from the current view. Second, we allow the deep neural network to automatically learn which information should be extracted from the image, encoded, and stored in this feature field. This encoding process is learned with the goal of maximizing re-localization accuracy.

Technically, we propose a few implementation ideas to execute the aforementioned steps elegantly and efficiently. In particular, we reduce localizing an observation in the map as the application of a standard convolutional operator, and the registration of new observations to the map as its dual operator, deconvolution. Both operators act efficiently in the allocentric representation rather than the image space.

We apply our method to three datasets: a toy set of synthetic mazes, data from the Doom video-game, and real images captured from a robotic platform using a recently-released public benchmark [1]. We show in all cases excellent localization performance, outperforming baselines that lack spatial memory [24]. We also explore some emerging semantic properties of the latent map embeddings.

The rest of the paper is structured as follows. Sec. 2 reviews related work, sec. 3 discusses our technical approach, sec. 4 evaluates the method on the aforementioned data, and sec. 5 summarizes our findings.

2. Related Work

The goal of iteratively building a map from a video stream, while using it to perform localization, has deep roots in robotics, and is commonly referred to as Visual SLAM (Simultaneous Location And Mapping) [23]. Classic SLAM methods emphasize real-time operation from a continuous video stream, which allows the use of temporal continuity cues (unlike typical Structure-from-Motion, which operates offline on unordered images). Modern SLAM systems are usually composed of several carefully engineered modules, such as tracking, mapping, relocalization, loop closure, graph optimization, key-frame selection, or post-processing by bundle adjustment [15]. Drawing from the success of deep learning [19, 6, 27, 20], where end-to-end training is key, there have been recent efforts to replace some of these components with more flexible learnable architectures.

One such effort is the recent surge of deep Visual Odometry (VO) methods [25, 24, 4]. Though related to ours, they focus on frame-by-frame estimation of the immediate changes in camera pose. DeepVO [24], VINet [25] and VidLoc [3] all employ Long-Short Term Memory (LSTM) on top of Convolutional Neural Networks (CNN) to predict pose changes, and are trained in a supervised manner. VINet uses additional information from an Inertial Measurement Unit (IMU) inputs [25], while VidLoc [3] performs multiple passes over the sequence with an LSTM, which brings it closer to Structure-from-Motion methods. These methods fill in only part of the role of SLAM, since they do not attempt to build a map of the environment, and make necessarily incremental predictions.

Another recent line of research has focused on replacing the whole SLAM pipeline, with encouraging preliminary results [12, 28, 8]. Kanitscheider and Fiete [12] train LSTM networks with location supervision on toy 2D environments, and explore the connections to hippocampal

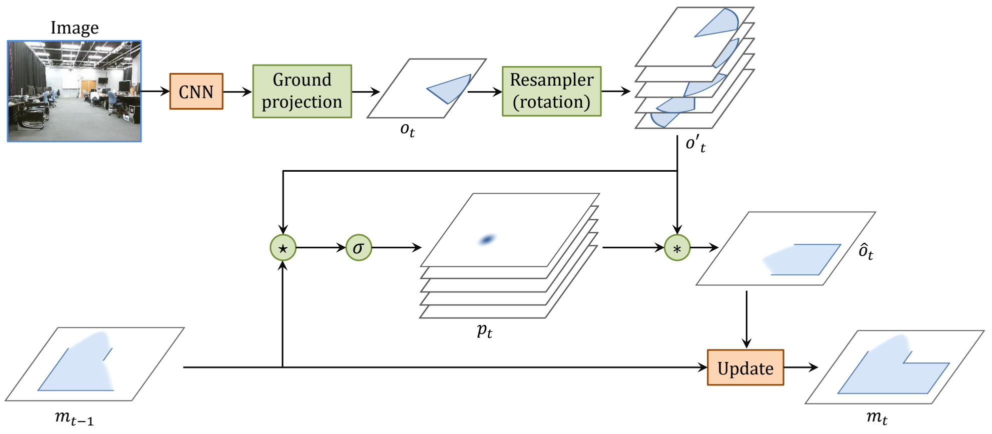

Figure 2: Proposed architecture. Our method performs localization and registration on a spatial memory via convolutional operators (sec. 3).

place cells [26]. Neural SLAM [28] and Neural Map [16], on the other hand, are trained with Reinforcement Learning (RL) to maximize a reward for successful navigation in a simulated environment. Both write to a differentiable memory to solve navigation tasks, and assume perfect information of either the agent's egomotion or position. Although promising, they are demonstrated only on small synthetic and toy tasks.

Building on these works, Gupta et al. [8] propose a differentiable architecture that unifies mapping and navigation. The map is an egocentric ground projection of image embeddings, and at each step the egocentric map from the preceding frame is integrated via a differentiable warp. This mapper module is the one that comes closest to our own work. The navigation module uses a Value Iteration Network (VIN) [18] to train the whole architecture end-to-end from example trajectories.

Integrating navigation as proposed by Gupta et al. [8] is a natural extension, although here we restrict our attention to the mapping task. There are several other key differences from our own work. Our architecture includes a world-centric rather than an egocentric map, which avoids the spatial blurring associated with repeated warping operations. We exploit convolutions to efficiently perform operations over all map locations. Another key difference is in the input modality: they assume 360° input images, with constant orientation, as well as perfect knowledge of egomotion, none of which we assume. Their experiments are on a synthetic dataset generated from 3D scene scans, while we experiment on free-form trajectories and real images.

3. Method

In this section we develop a Recurrent Neural Network (RNN) that can dynamically build a representation of an environment using observations obtained from a moving camera. The key component of the system is an allocentric spatial memory module, which allows the network to understand the world independently of the observation point, as well as to relocalize itself with respect to the latter.

Figure 2 illustrates one time step of our recurrent architecture. At time t, the architecture takes as input a new image x_t, consisting of both RGB and depth values (the latter may be estimated from another module, such as a monocular deep regression network [7]). A CNN extracts features from the image and a *ground projection module* (sec. 3.4) maps them to a 2.5D representation $o_t \in \mathbb{R}^{s\times s\times n}$ of a $s \times s$ spatial neighborhood around the camera, with n features per spatial location. The latter is rotated r times, obtaining a stack o'_t, which is used for re-localization as explained next.

The network also has access to the state $m_{t-1} \in \mathbb{R}^{h\times w\times n}$ of the *allocentric spatial memory* from the previous time step $t-1$. The spatial memory state can be thought of as a $h \times w$ map, much larger than the neighborhood o_t. The main idea is then to *perform registration* of the new observation o_t to the map m_{t-1} via dense matching in the ground plane. This is carried out efficiently by convolutional operators applied to the rotational stack (sec. 3.2).

Registration can also be interpreted as addressing an associative spatial memory, where soft addresses correspond to strong activations in the response of the convolution. This information is used to inform an LSTM module that performs an *update* (sec. 3.3) of the memory, adding more details to the map.

The remainder of this section discusses in details all these steps.

3.1. Localization

The goal of the localization module is to find the location *and* orientation of the camera at time t with respect to the allocentric world map. In order to do so, the ground-projected camera features o_t are matched at all possible rotations and locations against the current map state.

In order to do this efficiently, the observations o_t are transformed into a stack o'_t by applying a rotational resampler,

$$\forall ijkr: \quad o'_{ijkl} = [R(o,\, 2\pi l/r)]_{ijk},$$

where $R(o, \theta)$ rotates all feature channels of o by θ radians using bilinear interpolation.

Note that the stack $o' \in \mathbb{R}^{s\times s\times n\times r}$ can be interpreted as a *bank of r filters* of size $s \times s$ with n feature channels each. This allows to quickly compare an observation o_t to all possible locations and rotations in the map by using the standard neural network convolution operator. In more detail, one obtains a probability field $p_t \in \mathbb{R}^{h\times w\times r}$ as

$$p_t = \sigma(m_{t-1} \star o'_t), \tag{1}$$

where $\star$ denotes cross-correlation[1], and σ denotes the softmax operation:

$$[\sigma(x)]_{ijk} = \frac{e^{x_{ijk}}}{\sum_{i'j'k'} e^{x_{i'j'k'}}}. \tag{2}$$

Before applying convolution, the map m_{t-1} is zero-padded so that the output of the operator has the same size as m_{t-1}.

The result of eq. (1) is a normalized tensor of scores p_t for several discretized positions (dimensions $h \times w$) and orientations (dimension r). The highest value of p_t corresponds to where the observation is maximally correlated with the map. After training, this can be read as the network's belief that the agent is at a given position and orientation.

3.2. Registration

The observation o_t needs to eventually be integrated into the map, for the next time step. In order facilitate this step, we propose to translate and rotate $o_t \in \mathbb{R}^{s\times s\times n}$ according to the position and orientation encoded in p_t, to obtain a *registered* observation $\hat{o}_t \in \mathbb{R}^{h\times w\times n}$ in map-space.

We first define a spatial transformation function T, that registers the observation o w.r.t. translation (u, v) and rotation w parameters. This function is defined as (omitting the time subscript t for clarity)

$$T(o|u, v, w) = \tau_{uv}\left(R(o,\, 2\pi w/r)\right), \tag{3}$$

where $\tau_{uv}(x)$ shifts the tensor x by (u, v).

If $z \in \mathbb{R}^{h\times w\times r}$ is a one-hot encoding of the coordinates (u, v, w) (i.e. $x_{ijk} = \delta_{i=u,j=v,k=w}$), then the following identity holds:

$$T(o|u, v, w) = z * \bar{o}_t, \tag{4}$$

$$\text{with } \bar{o}_{ijlk} = o'_{ijkl}. \tag{5}$$

There are three useful interpretations of these equations. First, the tensor $\bar{o}$ represents the same filter bank as o', but with the input and output dimensions transposed. Thus the registration operator (3) can be implemented with a single application of a convolutional operator (4) on transposed filters. Second, this transposed filtering operation is exactly the same calculation performed when backpropagation is applied to convolution. Thus transposition can be skipped in code by simply calling the "convolution backward" routine implemented in all deep learning toolboxes. Third, the latter idea is also the same "trick" used to define the *convolution transpose* operator (also known as *deconvolution*). Hence, registration can be interpreted as deconvolution applied to the one-hot tensor z.

In practice, due to the softmax operator (2), the position/rotation estimate p_t is an approximation of a one-hot tensor: all values are in $[0, 1]$, and it sums to 1. We can thus register the observation by linearly combining T, weighted by p_t:

$$\hat{o}_t = \sum_{uvw} p_{uvw} T(o|u, v, w), \tag{6}$$

where the time subscript t in the tensor p_t is omitted for convenience. Plugging eq. (4) into eq. (6), and by the linearity of convolution, we obtain

$$\hat{o}_t = p_t * \bar{o}_t. \tag{7}$$

Equation (7) is very efficient: it relies on a single (de)convolution and the previously computed o' to perform registration, without any explicit spatial transformation T. Unlike a spatial transformer [11], it can deal with multi-modal distributions in p_t, and any uncertainty in the estimates will be reflected as uncertainty (blurring) in the output $\hat{o}_t$.

3.3. Update

After the new observations $\hat{o}_t$ have been localized and then registered to the current spatial memory m_{t-1}, the latter must be updated to incorporate the new evidence in a new state m_t. Since $\hat{o}_t$ and m_{t-1} are now aligned (they also have the same dimensions as tensors), this is relatively easy and can be achieved in a number of ways. We decided to use a LSTM [10], since it is an off-the-shelf RNN that already includes read and write gating mechanisms.

In order to maintain spatial invariance, we apply the same LSTM (shared weights) to all spatial locations independently:

$$m_{ij,t-1} = \mathrm{LSTM}\left(m_{ij,t}, \hat{o}_{ij,t}, W\right),$$

[1]Note that cross-correlation $\star$ is related to convolution $*$ by flipping the second argument: $x * y = x \star y'$, $y'_{ij} = y_{(-i,-j)}$. In most deep learning frameworks, the default convolutional operator is cross-correlation.

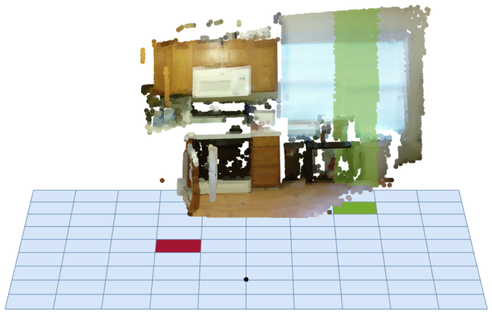

Figure 3: Example 3D point cloud, obtained from a RGBD image in the Active Vision Dataset (sec. 4.3). Part of the ground discretization grid is shown, with the center denoted by a black dot. Each 3D point is ground-projected to a cell (e.g. the points and cell shaded green). The CNN embeddings associated with these 3D points are accumulated into the corresponding cell of o using max pooling (see sec. 3.4 for details). Cells with no associated 3D points (e.g. the red cell) have their features set to 0.

where the function $\mathrm{LSTM}\,(h, x, W)$ returns the (single-step) updated state of an LSTM, given a hidden state h, input x and trainable parameters W.

3.4. Ground projection

So far, we have discussed observations o_t that are expressed in 3D space, whereas the input to the systems are RGBD images (x_t, d_t). In order to convert x_t into o_t, we assume that the ground plane is approximately known (e.g. downwards from the input image). For many applications of mobile robotics, in environment such as roads or indoors, this is a reasonable assumption.

Given an RGB image $x_t \in \mathbb{R}^{h' \times w' \times 3}$ at time t, we obtain a tensor of features x'_t with n channels using a standard CNN. Given the corresponding depth image $d_t \in \mathbb{R}^{h' \times w'}$ and known camera intrinsics, these features are projected onto the ground plane in camera-space (fig. 3). This results in a discretized, egocentric observation tensor $o_t \in \mathbb{R}^{s \times s \times n}$, which contains n features in an $s \times s$ spatial neighborhood around the agent.

Note that the ground projection is not, in general, a one-to-one map: either 0, 1 or more elements of the image features x'_t can be ground-projected onto any given element of $o_t \in \mathbb{R}^{s \times s \times n}$. To resolve ambiguities, for each feature we simply take the maximum of any competing elements, or 0 if there are none. This form of aggregation of 3D points is advocated by Qi et al. [17] in their PointNet method, and we found it to be superior to aggregation by averaging or summing.

In more detail, let $K \in \mathbb{R}^{3 \times 3}$ be the camera calibration matrix, so that the 3D coordinate of an image point can be written as

$$\begin{bmatrix} \bar{p}_x(i,j) \\ \bar{p}_y(i,j) \\ \bar{p}_z(i,j) \end{bmatrix} = K \begin{bmatrix} j \\ i \\ f \end{bmatrix} d_{ij},\ i \in [1, \ldots, H], j \in [1, \ldots, W].$$

Here $\bar{p}_y$ is the 3D height of the point, $\bar{p}_z$ its depth, $\bar{p}_x$ its horizontal displacement, and f the focal length. These are mapped to a column o_{tk} of the $s \times s$ spatial neighborhood as follows:

$$\bar{t}(i,j) = \frac{s-1}{2}\bar{p}_x(i,j) - \frac{s+1}{2},$$
$$\bar{k}(i,j) = \frac{s-1}{2}\bar{p}_z(i,j) - \frac{s+1}{2},$$

where we assume that s is an odd integer. Note in particular that the camera center sits in the middle of the tensor o (fig. 2). Then we obtain the definition:

$$o_{tkl} = \max\{x'_{ijl} : t = [\bar{t}(i,j)] \ \wedge \ k = [\bar{k}(i,j)]\}$$

where $[\cdot]$ denotes integer rounding and the maximum of the empty set $\max\{\}$ is defined to be zero.

3.5. Training and loss function

Our RNN is trained end-to-end to solve the localization problem with sequences of RGB-D inputs, and associated ground-truth positions and orientations. For a given training sequence, the RNN is initialized with an environment-agnostic position p_0, which is always set to be a one-hot encoding of the center of the map, facing the right (corresponding to an angle of $0°$). The position for time t can be discretized into classes $H_t \in \{1, \ldots, h\}$ and $W_t \in \{1, \ldots, w\}$, and likewise for the orientation $R_t \in \{1, \ldots, r\}$. The objective is to minimize the negative log-probability of the estimate position and orientation at each time step, namely:

$$L(p) = -\log \sum_t p_{H_t W_t R_t, t}. \tag{8}$$

4. Experiments

We show results on a toy dataset of 2D mazes (sec. 4.1), synthetic data from the Doom video-game (sec. 4.2), and real data from a moving robotic platform (sec. 4.3).

4.1. Results on synthetic mazes

To validate our approach without the added complexities of a complex CNN for ingesting images and performing feature extraction and ground projection, we first experimented with simple 2D environments.

Data. We generated 100,000 random labyrinths (fig. 4-left) by randomized depth-first search, yielding 21×21 binary

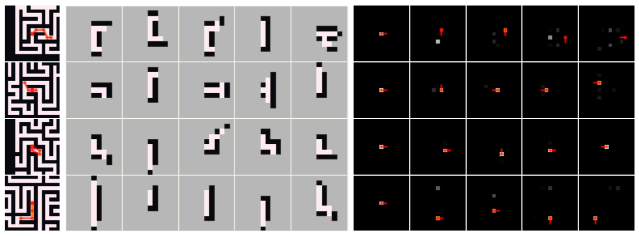

Figure 4: Visualization of 4 sequences of 5 frames, on a synthetic dataset of 100,000 labyrinths (sec. 4.1). Left: Ground-truth environment and trajectory (red). Middle: Observations in camera-space (i.e. rotated, local views of the environment centered on the trajectory). Right: Heatmap of position predictions (brighter means higher confidence), and corresponding ground-truth in red.

occupancy maps. 5,000 labyrinths were set aside for validation, and the rest used for training.

A simulated agent with limited, local information is considered. A camera viewpoint was simulated by raycasting from the agent's position and view direction, with a 180° field-of-view. The view directions are limited to $\{0°, 90°, 180°, 270°\}$ (thus $r = 4$). Example observations o generated this way can be visualized in the second column of fig. 4. The observations consist of two channels of one-hot encodings: one for walls and another for unoccupied cells. Note that, due to the local viewpoint, the camera is in the center and the view is always pointing to the right.

For training, we generated trajectories of 5 frames, randomly moving the agent to a visible unoccupied cell, with random view direction. At least 3 visible unoccupied cells are ensured at all times (to avoid getting stuck).

Architecture and training details. We implemented the method described in sec. 3, which we will refer to as MapNet. Instead of an image-space CNN and ground projection (sec. 3.4), in this section we used a small CNN, that transforms the two observed channels into an embedding of size $n = 16$. This CNN has two layers of 3×3 filters followed by batch-normalization, with 20 hidden channels and a ReLU.

The network was then trained by minimizing eq. (8) with the Adam optimizer [13] for 10 passes over the training data. The learning rate was set to 10^{-3}, with a batch size of 100. The map size was set to $h = w = 15$, and the observations size to $s = 11$.

Experiments and analysis. After training the network to convergence, we measured the Average Position Error (APE), defined as the average Euclidean error of the position only. The APE for the training and validation sets are nearly equal, 0.72 and 0.71 cells respectively. This means that the MapNet is wrong on average by less than one cell, after a 5-frames sequence where the camera may move several steps per frame. This is surprising given the difficulty of the task: the observations shown in fig. 4 have to be matched together to build a bigger picture, like puzzle pieces, with unknown positions and orientations, and very little overlap.

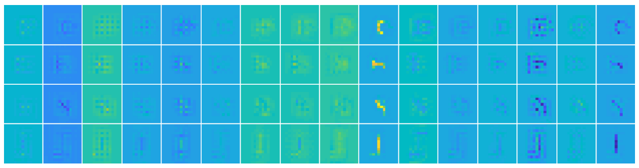

Figure 5: Map embeddings at the end of 4 sequences (same as in fig. 4), showing one channel per column.

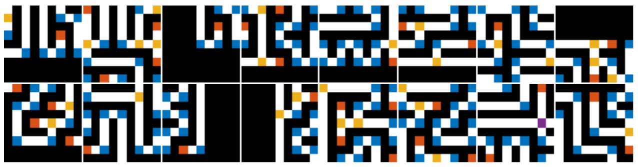

Figure 6: Visualization of the data used in the map decoding experiment (sec. 4.1). The classes are: corridor (white), turn (blue), dead end (red), fork (yellow), and crossroad (purple, visible only in the second-to-last tile, since it is a rare occurrence).

Corridor	Turn	Dead end	Fork	Crossroad	All
76.1%	73.3%	69.8%	68.8%	62.3%	71.3%

Table 1: Accuracies of binary classifiers for the semantic categories illustrated in Fig. 6, showing that map embeddings are correlated to semantics (random chance is 50%). The last column shows one-versus-all accuracy. See sec. 4.1 for details.

Some example predictions are shown on the right of fig. 4. Note the bi-modal distribution in the last row: the second observation may correspond to either end of the corridor in the first observation. This inherent ambiguity is resolved after more observations are collected.

The map embeddings obtained at the end of the example sequences of fig. 4 are visualized in fig. 5. The $n = 16$ channels are shown as tiles, with a brighter color meaning a higher value at a given map position. Some channels seem to follow the contours of visible portions of the ground truth map, while others are less interpretable. It is interesting that these contours cover a larger context than any single observation.

To perform a more quantitative analysis of the map embeddings, we decided to check whether they are correlated with intuitive semantic features of the mazes, such as long corridors, forks and dead ends. We divided unoccupied cells into categories, as a function of the number of entry points (one for dead ends, 2 for corridors/turns, 3 for forks and 4 for crossings), and their relative angle (180° for corridors,

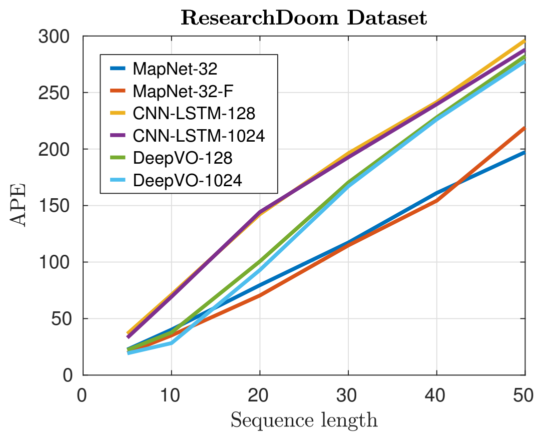

Figure 7: Average Position Error (APE) over different sequence lengths on the ResearchDoom dataset.

90° for turns), covering all possible states. We then took the MapNet embeddings obtained from the training set, and trained a one-versus-all Support Vector Machine to classify each map embedding into the corresponding semantic class.

The resulting accuracy is 71.3% (last column of table 1). Because this is a heavily unbalanced dataset (e.g. most cells are corridors and very few are crossings), we wanted to be sure that the classifier's accuracy reflected the discriminative power of the embeddings, and not just a-priori class probabilities. We then created a balanced dataset of positive and negative samples for each semantic category, by subsampling the majority class to match the number of samples of the minority class. If the map embeddings are predictive of a semantic category, then the accuracy of a binary SVM trained on the balanced dataset should be above chance (50%). Table 1 presents the results, showing that this is the case. This demonstrates that the map embeddings indeed encode recognizable aspects of the environment, as a side-effect of the primary goal of self-localization (eq. (8)).

4.2. Results on 3D game recordings

We now turn to a more challenging setting, with 3D data and richer environments: the classic game Doom. We used ResearchDoom [14] to render pre-recorded playing sessions by human players. Even though the images are synthetic, they contain many of the complexities of natural images, such as occlusions, fast motions, and ambiguous textures. Importantly, the camera traverses large, hand-crafted environments that were designed to be complex and appealing to players, as opposed to small and simple toy scenes.

Data details. We used recordings from 4 speed-runs through the game, capturing RGB-D and camera pose data. This yielded 687,894 images, or over 6 hours of gameplay. The game sprites were turned off, so that the networks can focus on the environment and navigation. For training we sample sequences of 5 images, obtained every 2 frames of recorded video, and resized to 160×100 pixels.

Doom data [14]	Units	APE-5	APE-50	ATE-50
MapNet-32 (ours)	32	23	**197**	74
MapNet-32-F (ours)	32	20	219	**73**
CNN-LSTM	128	37	296	111
	1024	33	288	108
DeepVO [1]	128	22	282	109
	1024	**19**	278	107

Table 2: Absolute Trajectory Error (ATE) and Average Position Error (APE), for sequences of 5 and 50 frames, on the ResearchDoom dataset (sec. 4.2).

Architecture and training details. We fine-tune a ResNet-50 [9] (pre-trained on ImageNet and keeping only the first 21 convolutional layers) to extract features from the RGB image. This CNN's features are then ground-projected according to sec. 3.4, and input to a final 2-layers CNN with 64 hidden channels (same as in sec. 4.1), to obtain the MapNet's observations. The discretization cell size is 30 (in the internal units of the game world), and $r = 12$ rotations are considered. Finally, the map size was set to $h = w = 29$, with an embedding of $n = 32$ units, and an observation size $s = 21$. Training was again performed using Adam, with a learning rate of 10^{-3}, and a batch size of 20, for 20 epochs.

Baselines and error metrics. In this experiment, we focused on testing the proposed method's capabilities of self-localization, and compare it against more general approaches. As a simple baseline we tested a CNN-LSTM combination that predicts pose directly, with different numbers of LSTM units. It uses the same CNN as ours, followed by global average pooling, which improves performance. We also reimplemented the state-of-the-art DeepVO method [24], which is similar but uses a FlowNet CNN [5]. Note that this method computes optical flow, which is a strong indicator of short-term motion, by concatenating the preceding frame along the channels of each input image. For a fair comparison, we also test a variant of our method with the same concatenated input images (MapNet-F).

We measured two kinds of errors. The Average Position Error (APE) is the average Euclidean distance between the predicted position and the corresponding ground-truth. The Average Trajectory Error (ATE), commonly used when evaluating SLAM systems [15], consists of translating and rotating the predicted trajectory to minimize the squared error w.r.t. the ground-truth, and then computing the Root-Mean-Squared (RMS) error in position. This metric is thus more generous, and corrects for the inevitable long-term effects of drift, making it more useful for long sequences. We measured short-term APE over 5 frames (APE-5), and long-term APE (APE-50) and ATE (ATE-50) over 50 frames.

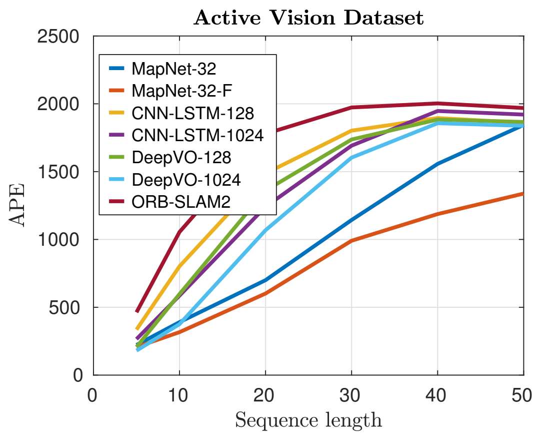

Figure 8: Average Position Error (APE) over different sequence lengths on the Active Vision Dataset.

Experiments and analysis. After training all networks, we measured the APE and ATE (table 2). Notice that long-term metrics (ATE/APE-50) test the networks' capabilities at extrapolating beyond the length of the training sequences (which is 5 frames). DeepVO [24] fares better than a generic CNN-LSTM, and shows better performance than ours in the short-term. This can be explained by its use of optical flow cues, which are very helpful over short time-scales. We verified that the proposed MapNet achieves better localization for most sequence lengths, converging for longer ones, despite using fewer LSTM units. The explicit spatial-memory matching seems to be most helpful in the long term, when simple odometry accumulates too many errors. Adding optical flow-like cues (MapNet-F) has a positive impact, as expected. Figure 1 shows qualitative results.

4.3. Results on real data

Active Vision Dataset. For our final experiment, we used the very recent Active Vision Dataset (AVD) [1]. AVD uses a robotic platform to capture RGBD images densely every 30cm (on a 2D grid) and every 30° in rotation, over 19 indoor scenes. The dataset contains over 15,000 images, but they can be combined into different trajectories, simulating robot navigation using real data. We sampled 200,000 random trajectories of 5 frames for training, by moving along the shortest path between 2 random locations.

Architecture and training details. All hyperparameters in this experiment remain identical to those in sec. 4.2. The discretization cell size is 300 millimeters. The only other difference is that we fine-tune a faster VGG-F model [2].

Experiments and analysis. In addition to the previous methods, we tested the state-of-the-art system ORB-SLAM2 [15], which should be appropriate given that we are dealing with real images. Unfortunately it performed quite poorly, despite extensive parameter tuning and ensuring a correct camera model. This is explained by the fact that classic SLAM systems rely heavily on temporal continuity, and AVD has a very low frame-rate. By contrast, our algorithm performs a full matching over all (planar) poses every frame, and thus can cope with large displacements. The performance advantage seems to be larger than for ResearchDoom, both in the short and long-term, probably due to ResearchDoom's high speed (since it consists of player speed-runs). It is interesting to note that, in the short-term (APE-5), the MapNet error is on the same order of the discretization cell size (300mm). This suggests that finer discretization could possibly improve results, at the expense of more computation. Example results are shown in fig. 1.

AVD data [1]	Units	APE-5	APE-50	ATE-50
MapNet (ours)	32	222	1844	614
MapNet-F (ours)	32	207	**1338**	**464**
CNN-LSTM	128	335	1860	751
	1024	266	1921	717
DeepVO [1]	128	207	1866	735
	1024	**181**	1838	700
ORB-SLAM2 [15]		463	1969	786

Table 3: Absolute Trajectory Error (ATE) and Average Position Error, for sequences of 5 frames (APE-5) and 50 frames (APE-50), on the Active Vision Dataset (sec. 4.3). Quantities are in millimeters (movement per frame is in discrete steps of 300mm).

5. Conclusions

In this paper we considered the problem of developing an allocentric representation of 3D spaces that can be dynamically updated by a deep neural networks for solving problems such as mapping and navigation. The method is based on a 2.5D associative spatial memory that is addressed and updated dynamically using information extracted from a RGBD sensor. We have also reduced the problem of localizing and registering observations to the application of convolution/deconvolution operators in memory space.

We have shown very encouraging results on both synthetic and real data. Our main finding is that this system, which uses a representation noticeably simpler than traditional mapping algorithms, can still achieve good localization performance, robustly, efficiently, and in a manner that affords end-to-end learning as a component of a more complex system.

Future extensions include handling full 3D spaces, or at least spaces such as buildings that can be decomposed as a stack of 2.5D areas, as well as reorganizing the map globally to account for long-term loop closure constraints.

Acknowledgments. The authors acknowledge the generous support of ERC 677195-IDIU.

References

[1] P. Ammirato, P. Poirson, E. Park, J. Kosecka, and A. C. Berg. A dataset for developing and benchmarking active vision. *arXiv preprint arXiv:1702.08272*, 2017.

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[3] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen. VidLoc: 6-dof video-clip relocalization. *arXiv preprint arXiv:1702.06521*, 2017.

[4] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia. Exploring representation learning with cnns for frame-to-frame ego-motion estimation. *IEEE Robotics and Automation Letters*, 1(1):18–25, 2016.

[5] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[7] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *arXiv preprint arXiv:1609.03677*, 2016.

[8] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive Mapping and Planning for Visual Navigation. *arXiv:1702.03920 [cs]*, Feb. 2017. arXiv: 1702.03920.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[12] I. Kanitscheider and I. Fiete. Training recurrent networks to generate hypotheses about how the brain solves hard navigation problems. *arXiv preprint arXiv:1609.09059*, 2016.

[13] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[14] A. Mahendran, H. Bilen, J. F. Henriques, and A. Vedaldi. Researchdoom and cocodoom: Learning computer vision with games. *arXiv preprint arXiv:1610.02431*, 2016.

[15] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[16] E. Parisotto and R. Salakhutdinov. Neural Map: Structured Memory for Deep Reinforcement Learning. *arXiv:1702.08360 [cs]*, Feb. 2017. arXiv: 1702.08360.

[17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[18] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1312–1320, 2015.

[19] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):640–651, 2017.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 2015 International Conference on Learning Representations*, 2015.

[21] H. Stensola, T. Stensola, T. Solstad, K. Frøland, M.-B. Moser, and E. I. Moser. The entorhinal grid map is discretized. *Nature*, 492(7427):72–78, 2012.

[22] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[23] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[24] S. Wang, R. Clark, H. Wen, and N. Trigoni. DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2043–2050. IEEE, 2017.

[25] H. Wen. VINet: Visual-inertial odometry as a sequence-to-sequence learning problem. AAAI, 2016.

[26] M. A. Wilson and B. L. McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1059, 1993.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[28] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu. Neural slam. *arXiv preprint arXiv:1706.09520*, 2017.