LEARNING GRID-LIKE UNITS WITH VECTOR REPRESENTATION OF SELF-POSITION AND MATRIX REPRESENTATION OF SELF-MOTION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposes a simple model for learning grid-like units for spatial awareness and navigation. In this model, the self-position of the agent is represented by a vector, and the self-motion of the agent is represented by a block-diagonal matrix. Each component of the vector is a unit (or a cell). The model consists of the following two sub-models. (1) Motion sub-model. The movement from the current position to the next position is modeled by matrix-vector multiplication, i.e., multiplying the matrix representation of the motion to the current vector representation of the position in order to obtain the vector representation of the next position. (2) Localization sub-model. The adjacency between any two positions is a monotone decreasing function of their Euclidean distance, and the adjacency is modeled by the inner product between the vector representations of the two positions. Both sub-models can be implemented by neural networks. The motion sub-model is a recurrent network with dynamic weight matrix, and the localization sub-model is a feedforward network. The model can be learned by minimizing a loss function that combines the loss functions of the two sub-models. The learned units exhibit grid-like patterns (as well as stripe patterns) in both 2D and 3D environments. The learned model can be used for path integral and path planning. Moreover, the learned representation is capable of error correction.

1 INTRODUCTION

Since the discovery of grid cells in the navigation system of the mammalian brain (Hafting et al. (2005); Fyhn et al. (2008); Yartsev et al. (2011); Killian et al. (2012); Jacobs et al. (2013); Doeller et al. (2010)), many mathematical and computational models (Burak & Fiete (2009); Sreenivasan & Fiete (2011); Blair et al. (2007); de Almeida et al. (2009)) have been proposed to explain the formation and function of grid cells. Recently, deep learning models (Banino et al. (2018); Cueva & Wei (2018)) have been proposed to learn the grid-like units for navigation. Such models consist of two sub-models. One is the motion sub-model, which is in the form of a recurrent neural network, to capture the change of the latent vector along the trajectory of the agent. The other is the localization sub-model, which is in the form of a feedforward neural network, to model the mapping from the latent vector to the position of the agent. The models of grid-like units can be used for two important tasks in navigation. One is path integral (Hafting et al. (2005); Fiete et al. (2008); McNaughton et al. (2006)) whose goal is to estimate the self-position of the agent given a sequence of it's self-motions. The other is path planning (Fiete et al. (2008); Erdem & Hasselmo (2012); Bush et al. (2015)) whose goal is to plan a sequence of self-motions in order to reach a given target destination. The model can be learned by supervised learning that minimizes the prediction error between the predicted position and the true position in path integral. It can also be learned by reinforcement learning for path planning.

Our work is inspired by the recent deep learning models on grid cells. However, instead of further generalizing and extending them, or making connections to the neuroscience of the grid cells, we seek to find a simple and explicit model for learning grid-like units for spatial awareness and navigation. Our motivation is that for such a basic system, we may only need a simple model that consists of one layer networks. The motion sub-model is a one layer recurrent network with dynamic weight matrix, while the localization model is a one layer linear feedforward network.

Specifically, in our model, the self-position of the agent is represented by a vector of a certain dimensionality, where each component of the vector is a unit (or a cell). The self-motion of the agent is represented by a matrix that depends on the self-motion or displacement. When the agent moves from the current position to the next position, the vector representation undergoes a linear transformation, where the vector representation of the next position is obtained by multiplying the matrix of the self-motion to the vector representation of the current position. For the sake of estimation accuracy and computational efficiency, we assume that the self-motion matrix is block diagonal. That is, we partition

the components of the vector into different blocks of sub-vectors, with each block consisting of a small number of units (e.g., 6 units), and each block undergoes its own linear transformation by a multiplication of a small matrix. In other words, the dynamics of the blocks are disentangled.

In addition to the above motion sub-model in the form of vector-matrix multiplication, we have the localization sub-model which also embraces simplicity. For any two positions, we define their adjacency as a kernel function of their Euclidean distance. This kernel function is given to the learning algorithm. It has the maximal value 1 at zero distance and decreases monotonically to 0 as the distance increases. In the localization sub-model, the adjacency between any two positions is given by the inner product between their vector representations. This localization model enables the agent to decode the position of a vector for the purpose of the path integral. Specifically, we compute the inner product between the given vector and the vectors of all the possible positions to obtain a heat map for all the possible positions, where the maximum is the inferred position. This heat map may inform the place cells, together with visual cues of the environment. The model also enables the agent to do path planning by a simple steepest ascent algorithm that sequentially maximizes the adjacency to the destination. Since the adjacency is 1 at zero distance, the vector is of unit norm, and the self-motion is represented by a rotation of this unit vector on a high-dimensional sphere, where the arc length on the sphere informs the Euclidean distance in the original 2D space.

Both the motion and the localization sub-models can be viewed as simple neural networks, with the motion model being a one layer recurrent network with disentangled blocks and with dynamic weight matrix, while the localization model being a feedforward one layer linear network. Thus, our model is a drastic simplification of existing deep learning models of grid cells, without inserting any artificial assumptions such as Fourier basis or clock arithmetics, etc. (Blair et al. (2007); de Almeida et al. (2009))

We show that the learned units of the vector representation exhibit grid-like patterns as well as stripe patterns. We also show that the learned model can do path integral and path planning. Moreover, the learned representation is capable of error correction.

The following are the contributions of our work. (1) We propose a novel model for learning grid-like units for navigation. The model has explicit representations of self-position and self-motion in terms of vector and matrix. (2) We show that the learned units exhibit periodic grid-like or strip patterns. (3) We show that the learned model can be used for path integrals. (4) We show that the learned model can perform path planning in the form of a simple steepest ascent algorithm. (5) We show that the learned representation is tolerant of additive and drop out errors. (6) We show that the model can be generalized to 3D environment, and learn meaningful grid-like patterns. The learned model can also be used for path integral and path planning.

2 Related work

Our work is inspired by recent deep learning models on grid cells (Banino et al. (2018); Cueva & Wei (2018)). However, our model is much simpler and more explicit than existing models. Our model only consists of one layer recurrent network and feedforward network, with highly interpretable sub-models of motion and localization. It is a simple and natural model without adding artificial components such as Fourier basis.

Compared to previous computational models on grid cells, our model only assumes one layer networks, without making assumptions such as Fourier plane waves or clock arithmetics (Blair et al. (2007); de Almeida et al. (2009)). As such, our goal is to find a simple loss function to learn the system.

Similar to word2vec (Mikolov et al. (2013)), the vector representation in our method may be called a place2vec. In word2vec, the inner product of two vectors measures the similarity of the words. In place2vec, the inner product of two vectors measures the adjacency of two positions.

Our model and the associated loss function is connected to several well-known machine learning methods. The representation of the adjacency kernel between two positions by the inner product between their vector representations reminds us the kernel trick in support vector machine (Cortes & Vapnik (1995)), where the adjacency is the kernel function, and the vector is a high-dimensional representation of the 2D input. The difference is that in kernel machines, the high-dimensional vector is implicit, while in our work, we aim to learn the vector representation explicitly. The explicit representation of the kernel function in terms of inner product between vectors is related to symmetric matrix factorization (Lee & Seung (1999)), which is in turn related to spectral clustering (Ng et al. (2002)) and kernel k-means (Dhillon et al. (2004)). The motion sub-model is related to local linear embedding (Roweis & Saul (2000)), except that the vectors are related by matrix representations of self-motion.

In representation theory in mathematics (Fulton & Harris (2013)), the group elements are represented by matrices acting on the vectors. We learn a special system of representation for representing self-position and self-motion.

3 MODEL

Consider an agent navigating within a domain $D = [0, 1] \times [0, 1]$. We can discretize D into an $N \times N$ lattice. Let $x = (x_1, x_2) \in D$ be the self-position of the agent. Suppose the agent wants to represent its self-position by a d-dimensional hidden vector $v(x)$. We introduce two sub-models.

3.1 MOTION SUB-MODEL

Suppose at a position x, the self-motion is Δx, so that the agent moves to $x + \Delta x$ after one step. We assume that

$$v(x + \Delta x) = M(\Delta x)v(x), \tag{1}$$

where $M(\Delta x)$ is a $d \times d$ matrix that depends on Δx. While $v(x)$ is the vector representation of the self-position x, $M(\Delta x)$ is the matrix representation of the self-motion Δx. As we will show below, $|v(x)| = 1$ for all x, thus $M(\Delta x)$ is a rotation matrix, and the self-motion is represented by a rotation in the d-dimensional sphere. We can illustrate the motion model by the following diagram:

$$\text{Motion}: \quad \begin{array}{ccc} x & \xrightarrow{+\Delta x} & x_{t+1} \\ \downarrow & \downarrow & \downarrow \\ v(x) & \xrightarrow{M(\Delta x)\times} & v(x_{t+1}) \end{array} \tag{2}$$

Both $v(x)$ and $M(\Delta x)$ are to be learned.

For the sake of estimation accuracy and computational efficiency, we further assume that $M(\Delta x)$ is block diagonal, i.e., we can divide $v = (v^{(k)}, k = 1, ..., K)$ into K blocks of sub-vectors, and $v^{(k)}(x + \Delta x) = M^{(k)}(\Delta x)v^{(k)}(x)$. That is, the vector v consists of sub-vectors, each of which rotates in its own subspace.

We can learn a separate motion matrix M for each Δx within a limited range. Because of the block-diagonal form of M, we can afford to learn such motion matrices separately. However, we can also learn a parametric model for M. To this end, we can further parametrize $M = I + \tilde{M}(\Delta x)$ such that each element of $\tilde{M}(\Delta x)$ is a quadratic (or polynomial) function of $\Delta x = (\Delta x_1, \Delta x_2)$:

$$\tilde{M}_{ij}(\Delta x) = \beta_{ij}^{(1)}\Delta x_1 + \beta_{ij}^{(2)}\Delta x_2 + \beta_{ij}^{(11)}\Delta x_1^2 + \beta_{ij}^{(22)}\Delta x_2^2 + \beta_{ij}^{(12)}\Delta x_1\Delta x_2, \tag{3}$$

where the coefficients β are to be learned. The above may be considered a second-order Taylor expansion which is expected to be accurate for small Δx.

The motion model can be considered a linear recurrent neural network (RNN). However, if we are to interpret M as the weight matrix, then the weight matrix is dynamic because it depends on the motion Δx. One may implement it by discretizing Δx, so that we have a finite set of Δx, and thus a finite set of $M(\Delta x)$. Then at each time step, the RNN switches between the finite set of motion matrices. This is like the gearing operation of a multi-geared bicycle. Another implementation is to write $M(\Delta x)v = v + B^{(1)}(\Delta x_1 v) + B^{(2)}(\Delta x_2 v) + B^{(11)}(\Delta x_1^2 v) + B^{(22)}(\Delta x_2^2 v) + B^{(12)}(\Delta x_1 \Delta x_2 v)$, where the B matrices consist of the corresponding β coefficients. We first obtain new input vectors $\Delta x_1 v$, $\Delta x_2 v$, ..., $\Delta x_1 \Delta x_2 v$, and then treat $B^{(1)}, B^{(2)}, ..., B^{(12)}$ as the weight matrices operating on the new input vectors.

3.2 LOCALIZATION SUB-MODEL

Let $A(x, y)$ be the adjacency measure between two positions x and y such that $A(x, y) = f(|x - y|)$, where $f(r)$ decreases monotonically as the Euclidean distance $r = |x - y|$ increases. One example of f is the Gaussian kernel $f(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$. Another example is the exponential kernel $f(r) = \exp\left(-\frac{r}{\sigma}\right)$. As a matter of normalization, we assume $f(0) = 1$.

We assume the adjacency measure is given by the inner product between the hidden vectors

$$A(x, y) = \langle v(x), v(y)\rangle. \tag{4}$$

Since $f(0) = 1$, $|v(x)| = 1$ for any x, and $\langle v(x), v(y)\rangle = \cos\alpha$, where α is the angle between $v(x)$ and $v(y)$. Thus for a vector v, the distance between it's position and x is $f^{-1}(\langle v, v(x)\rangle)$. Since $f(r)$ is monotonically decreasing, $h(x) = \langle v, v(x)\rangle$ gives us the heat map to decode the position of v. Let the decoded position be $\hat{x}$, then

$\hat{x} = \arg\max_x \langle v, v(x) \rangle$. We can obtain the one-hot representation $\delta_{\hat{x}}$ of $\hat{x}$ by non-maximum suppression on the heat map $h(x)$. The heat map $h(x)$ can be combined with visual feature maps to infer the position if there are visual cues.

Let $V = (v(x), \forall x)$ be the $d \times N^2$ matrix, where each column is a $v(x)$. We can write the heat map of a d-dimensional vector v by a N^2-dimensional vector $h = V^\top v$, which serves to decode the position x encoded by v. Conversely, for a one-hot representation of a position x, i.e., δ_x, which is a one-hot N^2-dimensional vector, we can encode it by $v = V\delta_x$. Both the encoder and decoder can be implemented by a linear neural network with connection weights V and $V^\top$ respectively, as illustrated by the following diagram:

$$\begin{array}{llcll} \text{Localization}: & v & \xrightarrow{V^\top \times} & h & \text{(heat map and decoding to } \delta_x) \\ & \delta_x & \xrightarrow{V \times} & v(x) & \text{(encoding)} \end{array} \tag{5}$$

Note that in decoding $v \to h(x) \to \delta_x$ and encoding $\delta_x \to v$, we do not represent or operate on the 2D coordinate x explicitly, i.e., x itself is never explicitly represented, although we may use the notation x in the description of the experiments.

4 LEARNING

4.1 LOSS FUNCTION

We can learn v and M (or the β coefficients that parametrize M) by minimizing

$$\mathbb{E}_{x,y}\left[(A(x,y) - \langle v(x), v(y)\rangle)^2\right] + \lambda \mathbb{E}_{x,\Delta x}\left[\|v(x+\Delta x) - M(\Delta x)v(x)\|^2\right], \tag{6}$$

where $\lambda > 0$ is a tuning constant. The first term is the localization loss, and the second term is the motion loss. In the first expectation, we assume x and y are sampled uniformly from D. In the second expectation, we assume that x is sampled uniformly from D, while Δx is sampled uniformly within a limited range.

The above loss function can be conveniently minimized by gradient descent. We may re-parametrize $M(\Delta x)$ in a residual form $M(\Delta x) = I + \tilde{M}(\Delta x)$, and do gradient descent with respect to $\tilde{M}$. In the case where $\tilde{M}(\Delta x)$ is parameterized by a quadratic function of $\Delta x = (\Delta x_1, \Delta x_2)$, we can further back-propagate the gradient of $\tilde{M}$ to the β coeffciients.

We impose regularization terms $U(v) = \sum_{i=1}^d \left(\mathbb{E}_x[v_i(x)^2] - 1/d\right)^2$ to enforce uniform energy of the units for $v = (v_i(x), i = 1, ..., d)^\top$. It also helps to break the symmetry caused by the fact that the loss function is invariant under the transformation $v(x) \to Qv(x), \forall x$, where Q is an arbitrary $d \times d$ orthogonal matrix.

The motion loss $\mathbb{E}_{x,\Delta x}\left[\|v(x+\Delta x) - M(\Delta x)v(x)\|^2\right]$ is a single-step loss. It can be generalized to multi-step loss

$$\mathbb{E}_{x,\Delta x_1,...\Delta x_T}\left[\|v(x+\Delta x_1 + ... + \Delta x_T) - M(\Delta x_T)...M(\Delta x_1)v(x)\|^2\right], \tag{7}$$

where $(\Delta x_t, t = 1, ..., T)$ is a sequence of T steps of displacements.

4.2 IMPLEMENTATION DETAILS

We obtain inputs for learning the model by simulating agent trajectories with duration $\tilde{T}$ in the square domain D. D is discretized into a 40×40 lattice and the agent is only allowed to move on the lattice. The agent starts at a random location x_0. At each time step t, a small motion Δx_t (≤ 3 grids in each direction) is randomly sampled with the restriction of not leading the agent outside the boundary, resulting in a simulated trajectory $\{x_0 + \sum_{i=1}^t \Delta x_i\}_{t=1}^{\tilde{T}}$. $\tilde{T}$ is set to $1,000$ to obtain trajectories that are uniformly distributed over the whole area. It is worth mentioning that although the trajectories used for training are restricted to the lattice and with small motions, in Section 5.3 we show that the learned model can be easily generalized to handle continuous positions and large motions. In training, pairs of locations (x, y) are randomly sampled from each trajectory as the input to the localization loss, while consecutive position sequences $(x, \Delta x_t, t = 1, ..., T)$ are randomly sampled as the input to the motion loss term, with length specified by T (which is usually much smaller than the whole length of the trajectory $\tilde{T}$).

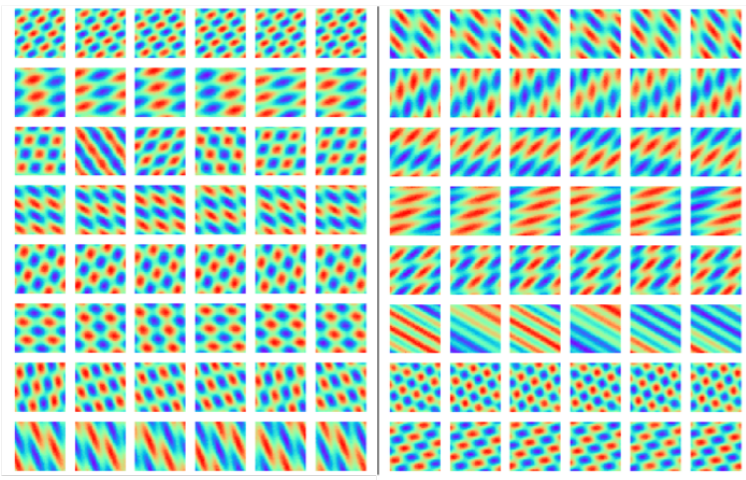
The model is trained with Adam optimizer (Kingma & Ba (2014)) (lr $= 0.03$) for $6,000$ iterations. A batch of $30,000$ examples (i.e., (x, y) and $(x, \Delta x_t, t = 1, ..., T)$) is sampled from the simulated trajectories at each learning iteration as the input to the loss function. λ is set to 0.1 to balance the magnitude of the two loss terms. The weight for the regularization term is set to $5,000$. In the later stage of learning ($\geq 4,000$ iterations), $|v(x)| = 1$ is enforced by projected gradient descent, i.e., normalizing each $v(x)$ after the gradient descent step. We use 16 blocks of units with block size 6, resulting in 96 units in total.

5 EXPERIMENTS

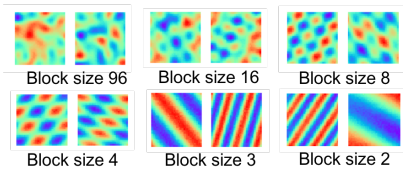
First, we conduct experiments on simulation data to learn the units $v(x)$ and motion matrices $M(\Delta x)$ parametrized by the β coefficients. Then, we show that the learned representation can perform path integral and path planning tasks. Next, we show that the learned representation is capable of error correction. Finally, we generalize the system to 3D environments.

5.1 LEARNED UNITS

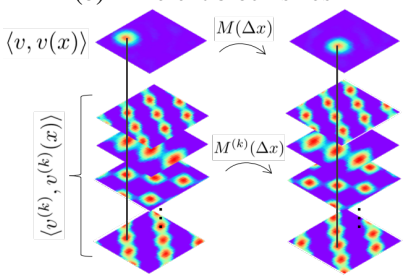
In Figure 1a, we visualize the learned units $v(x)$ over the 40×40 lattice of x. A Gaussian kernel with $\sigma = 0.08$ is used for the adjacency measure. Each block contains 6 units that belong to the same sub-vector in the motion model. Interestingly, some individual units exhibit obvious grid-like patterns, with multiple firing fields of roughly circle or ellipse shapes. Furthermore, the firing fields are arranged in a regular square or hexagon lattice. Within each block, the units exhibit very similar patterns, with different phases. In (Blair et al. (2007)), the grid cell response is modeled by three cosine gratings with different orientations and phases. In our model, we learn such patterns without inserting artificial assumptions. Besides grid-like responses, there are also some strip-like patterns at relatively large scales.



(a) Learned units



(b) Different block sizes



(c) Disentangled blocks

Figure 1: (a) Learned units of the vector representation. Each block shows the units belonging to the same sub-vector in the motion model. (b) Examples of units with different block sizes. (c) Illustration of block-wise activities of hidden units (where the activities are rectified to be positive).

5.1.1 DISENTANGLED BLOCKS

We find that disentangling the hidden vector v into blocks and assuming motion matrix $M(\Delta x)$ to be block-diagonal is helpful for the emergence of grid-like patterns. Figure 1b displays examples of learned units with different block sizes except for the block size 6 used in Figure 1a. The overall dimension of the hidden units is fixed to 96 for a fair comparison. Grid patterns do not emerge when no disentanglement is involved (block size 96), but they gradually become clear when block size decreases (e.g., block size 8 or 4). If the block size is too small (3 and 2), stripe-like Fourier plane wave patterns dominate.

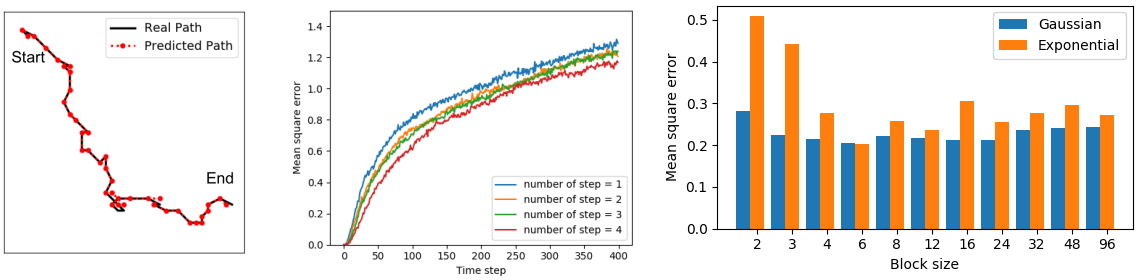
Both localization and motion models can be interpreted as a combination of sub-blocks. Figure 1c provides an illustration. For the localization model, given a vector v, the heat map of a single block $\langle v^{(k)}, v^{(k)}(x) \rangle$ has periodic firing fields and cannot determine a location uniquely. However, by combing heat maps of multiple blocks, which have firing fields of multiple scales and phases, the location is uniquely determined by $\langle v, v(x) \rangle$. For a motion Δx, every block rotates in its own subspace with motion matrix $M^{(k)}(\Delta x)$, resulting in phase shifting in the heat map of each single block. Combining motions in subspaces together, overall motion $M(\Delta x)$ is obtained.

5.2 PATH INTEGRAL

Path integral (also referred to as dead-reckoning) is the task of inferring the self-position based on self-motion (e.g., imagine walking in a dark room). Specifically, the input to path integral is a previously determined initial position x_0 and known or estimated motion sequences $\{\Delta x_1, ..., \Delta x_T\}$, and the output is the prediction of one's current position x_T. We test our learned system on this task. We first encode the initial position x_0 as $v(x_0)$. Then, by the motion model, the hidden vector $v(x_T)$ at time T can be predicted as:

$$v(x_T) = \prod_{t=T}^{1} M(\Delta x_t) v(x_0) = M\left(\sum_{t=1}^{T} \Delta x_t\right) v(x_0). \tag{8}$$

That is, the learned system of $(v(x), M(\Delta x), \forall x, \Delta x)$ should satisfy the above condition. Note that this does not imply $\prod_{t=T}^{1} M(\Delta x_t) = M\left(\sum_{t=1}^{T} \Delta x_t\right)$. The above equation is true only for a spatial system of $(v(x), \forall x)$. Thus, $(v(x), M(\Delta x), \forall x, \Delta x)$ forms a whole system for spatial awareness. By the localization model, the inferred position at time T is $\hat{x}_T = \arg\max_x \langle v(x_T), v(x) \rangle$.



(a) Predicted path (b) MSE over time step (c) MSE with different block sizes and kernels

Figure 2: (a) Path integral prediction. The black line depicts the real path while red dotted line is the predicted path by the learned model. (b) Mean square error over time step. The error is average over $1,000$ episodes. The curves correspond to different number of steps used in the multi-step motion loss. (c) Mean square error performed by models with different block sizes and different kernel types. Error is measured by number of grids.

Figure 2a shows an example of path integral result (time duration $\tilde{T} = 40$), where we use single step motion loss, unlike (Banino et al. (2018); Cueva & Wei (2018)), which train RNN or LSTM with long-term loss to fit the path integral task. Gaussian kernel with $\sigma = 0.08$ is used as the adjacency measure. We find single-step loss is sufficient for performing path integral. The mean square error remains small (~ 1.2 grid) even after 400 steps of motions (figure 2b). The error is average over $1,000$ episodes. The motion loss can be generalized to multi-step, as shown by equation 7. In Figure 2b, we show that multi-step loss can improve the performance slightly.

A question is whether the assumed block-diagonal motion matrix $M(\Delta x)$ affects the accuracy of path integral, since such assumption leads to much less parameters. In Figure 2c we compare learned models with fixed number of hidden units (96) but different block sizes. We also compare the performance of models using Gaussian kernel ($\sigma = 0.08$) and exponential kernel ($\sigma = 0.3$) as the adjacency measure in the localization model. The result shows that models with Gaussian kernel and block size ≥ 3, and with exponential kernel and block size ≥ 4 have comparative performances with the model learned without block-diagonal assumption (block size $= 96$). Models with Gaussian kernel perform slightly better than models with exponential kernel.

5.3 PATH PLANNING

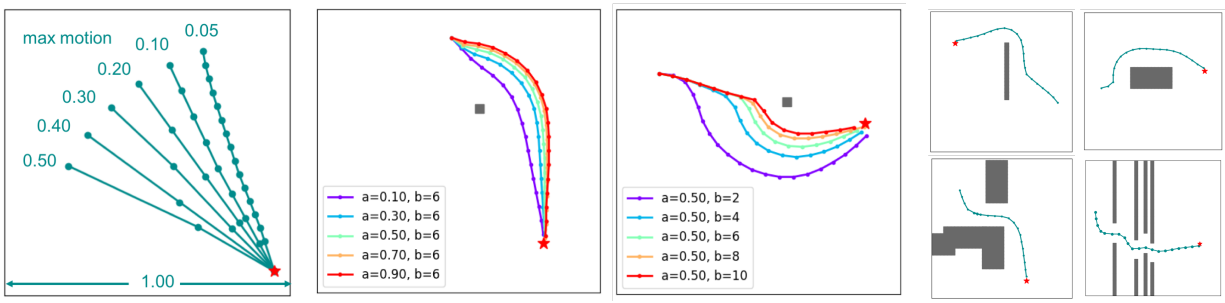
Path planning is the process in which the agent plans a sequence of self-motions given a start position x_0 and a target destination y. In the simplest scenario, where no obstacle is involved, the agent should be able to plan direct path between x_0 and y. The learned system enables the agent to do path planning by steepest ascent according to the adjacency measure: (1) encoding x_0 and y to hidden vectors $v(x_0)$ and $v(y)$, and (2) sequentially maximizing the adjacency of the current hidden vector to $v(y)$. Specifically, in the simplest scenario, suppose the hidden vector is v_t at time t, the agent chooses a motion Δx_t from a motion pool $\mathcal{M}$ by

$$\Delta x_t = \arg\max_{\Delta x \in \mathcal{M}} \langle v(y), M(\Delta x) v_t \rangle, \tag{9}$$

and then the hidden vector is updated by $v_{t+1} = M(\Delta x_t)v_t$. The planning process terminates if the adjacency $\langle v_t, v(y)\rangle$ is close to 1 or the number of planning steps reaches a predefined maximum value.

A critical question is how to choose the motion pool $\mathcal{M}$. To augment motion candidates, we no longer restrict the motion to be on the lattice. Instead, for a small length r, we evenly divide $[0, 2\pi]$ into n directions $\{\theta_i, i = 1, ..., n\}$, resulting in n candidate motions $\{\Delta x_i = (r*\cos(\theta_i), r*\sin(\theta_i)), i = 1, ..., n\}$. These n small motions serve as motion basis. Larger motion $k\Delta x_i$ can be further added to the pool by estimating the motion matrix $M(k\Delta x_i) = M^k(\Delta x_i)$. The starting position and destination can also be any continuous values, where the encoding to the latent vector is approximated by bilinear interpolation of nearest neighbors on the lattice.

In the experiments, we choose $r = 0.05$ and $N = 100$, and add another set of motion with length 0.025 to enable accurate planning. The system is learned with exponential kernel ($\sigma = 0.3$) for adjacency measure to encourage connection of long distance. Figure 3a shows planning examples with six settings of motion ranges. Including larger motions accelerates the planning process so that it finishes with less steps. We define one episode to be a success if the distance between the end position and the destination is less than 0.025. We achieve a success rate of $> 99\%$ over $1,000$ episodes for all the six settings.



(a) Simple planning (b) Planning with dot obstacle (c) Various obstacles

Figure 3: (a) Planning examples with different motion ranges. Red star represents the destination y and green dots represent the planned position $\{x_0 + \sum_{i=1}^{t} \Delta x_i\}$. (b) Planning examples with a dot obstacle. Left figure shows the effect of changing scaling parameter a, while right figure shows the effect of changing annealing parameter b. (c) Planning example with obstacles mimicking walls, large objects and simple mazes.

The learned system can also perform planning with obstacles. Specifically, suppose z is an obstacle to avoid, the agent can choose the motion Δx_t at time t by

$$\Delta x_t = \arg\max_{\Delta x \in \mathcal{M}} \left[\langle v(y), M(\Delta x)v_t\rangle - a\langle v(z), M(\Delta x)v_t\rangle^b\right], \tag{10}$$

where a and b are the scaling and annealing parameters. Figure 3 shows the planning result with a dot obstacle laid on the direct path between the starting position and destination, with tuning of a and b. We choose $a = 0.5$ and $b = 6$ in subsequent experiments.

More complicated obstacles can be included, by summing over the kernels $\sum_{i=1}^{m} a\langle v(z_i), M(\Delta x)v_t\rangle^b$ and choose Δx_t at time t by $\arg\max_{\Delta x \in \mathcal{M}} \left[\langle v(y), M(\Delta x)v_t\rangle - \sum_{i=1}^{m} a\langle v(z_i), M(\Delta x)v_t\rangle^b\right]$. Figure 3c shows some examples, where the obstacles mimicking walls, large objects and simple mazes. Please see more planning with obstacle results in the Appendix.

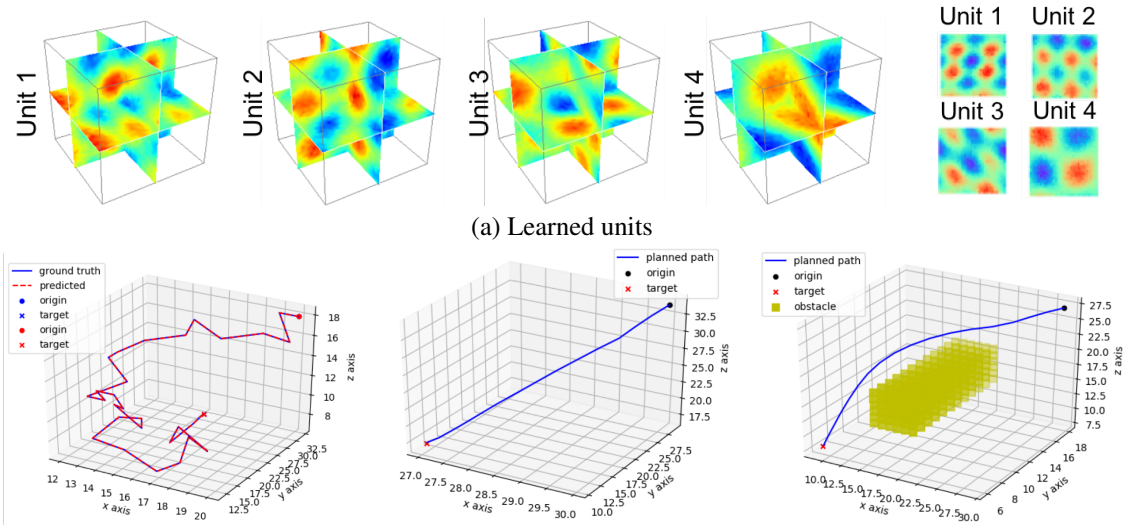
The advantage of the proposed path planning method is its simplicity. For simple environments, path planning can be accomplished by simple steepest ascent algorithm. The method can be easily extended to moving target and moving obstacles. There is no need for reinforcement learning or sophisticated optimal control.

5.4 Error correction

Unlike commonly used embedding in machine learning, here we embed a 2D position into a high-dimensional space, and the embedding is a highly distributed representation or population code. The advantage of such a redundant code is in it's tolerance to errors. We show that the learned system is tolerant to various errors of units. Specifically, in both path integral and path planning tasks, at every time step t, we randomly add (1) Gaussian noises or (2) drop out masks to the hidden units and see if the system can still perform the tasks well. We find that the decoding-encoding process (DE) is important for error correction. That is, at each time step t, given the noisy hidden vector v_t, we decode it to $x_t = \arg\max_x \langle v_t, v(x)\rangle$ and then re-encode it to the hidden vector $v(x_t)$.

	Path integral: MSE					Path planning: success rate				
Noise type	$1s$	$0.75s$	$0.5s$	$0.25s$	$0.1s$	$1s$	$0.75s$	$0.5s$	$0.25s$	$0.1s$
Gaussian	1.687	1.135	0.384	0.017	0	0.928	0.959	0.961	0.977	0.985
Gaussian (no DE)	6.578	2.999	1.603	0.549	0.250	0.503	0.791	0.934	0.966	0.982
	70%	50%	30%	10%	5%	70%	50%	30%	10%	5%
Dropout	2.837	1.920	1.102	0.109	0.013	0.810	0.916	0.961	0.970	0.978
Dropout (no DE)	19.611	16.883	14.137	3.416	0.602	0.067	0.186	0.603	0.952	0.964

Table 1: Error correction results on path integral and path planning.



(a) Learned units

(b) Path integral (c) Simple path planning (d) Path planning with obstacle

Figure 4: Learned results in 3D environment. (a) Four examples of learned units from different blocks. Left four figures show the 3D visualization while the rightmost figure shows the patterns of a slice of every unit. (b) One example of path integral with duration $\tilde{T} = 30$. (c) One example of simple path planning, where the agent is capable of planning a direct trajectory. (d) One example of path planning with a cuboid obstacle.

Table 1 shows the error correction results tested on path integral and path planning tasks. Each number is averaged over $1,000$ episodes. We compute the overall standard deviation of $\{v(x)\}$ for all x and treat it as the reference standard deviation for the Gaussian noise (s). For drop out noise, we set a percentage to drop at each time step. With the decoding-encoding process, the system is quite robust to the Gaussian noise and drop out error, and the system still works even 70% units are silenced at each step.

5.5 Modeling in 3D environment

The system can be generalized to 3D environments. Specifically, we assume the agent navigates within a domain $D = [0, 1] \times [0, 1] \times [0, 1]$, which is discretized into a $40 \times 40 \times 40$ lattice. We learn a parametric model for motion matrix M by a residual form $M = I + \tilde{M}(\Delta x)$. To estimate $\tilde{M}(\Delta x)$ more precisely, we parametrize each element of $\tilde{M}(\Delta x)$ as a cubic function of $\Delta x = (\Delta x_1, \Delta x_2, \Delta x_3)$.

A batch of 200,000 examples of (x, y) and $(x, \Delta x_t, t = 1, ..., T)$ is sampled online at every iteration for training. We use 16 blocks of units with block size 6 and choose Gaussian kernel ($\sigma = 0.08$) as the adjacency measure. Figure 4a shows four learned 3D units, where the grid-like patterns emerge. Figure 4b shows that the learned system can perform path integral well. Figure 4c shows that the agent is capable of planning direct a 3D trajectory using the trained system, and figure 4d shows that the agent can perform planning with a cuboid obstacle. Please refer to the Appendix for more 3D results.

6 DISCUSSION

The goal of this work is to find a simple and natural model and the associated loss function to learn the grid-like units for navigation, in the hope of contributing to our understanding of the possible computational models of grid cells. Our model consists of a localization sub-model and a motion sub-model. We show that it is possible to learn grid-like units (as well as strip-like units), and the learned model is capable of path integral and path planning. The vector representation is also robust to additive and drop out noises.

The model in this paper can be considered an internal system for spatial awareness. This system may interact with the external sensory inputs such as visual cues. For instance, the heat map can be combined with visual feature maps to infer the self-position and its vector representation.

Two key features that distinguish our model from existing models are: (1) The motion is encoded by the matrix that operates on vector representation of position. This separates the representations of self-motion and self-position. (2) The inner product of two vectors, or equivalently, the arc-length between the two unit vectors on the high-dimensional sphere, informs the Euclidean distance between the two underlying positions they encode. This is crucial for path planning, so that for simple path planning problems, steepest ascent algorithm can work, and there is no need to resort to reinforcement learning or dynamic programming optimal control.

Despite being relatively simple and natural, our model is very speculative in terms of biological plausibility (which may also be the case with other existing models). Nonetheless, it is our hope that our model and results will contribute to the discussion about computational underpinnings of grid cells.

REFERENCES

Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429, 2018.

Hugh T Blair, Adam C Welday, and Kechen Zhang. Scale-invariant memory representations emerge from moire interference between grid fields that produce theta oscillations: a computational model. *Journal of Neuroscience*, 27(12):3211–3229, 2007.

Yoram Burak and Ila R Fiete. Accurate path integration in continuous attractor network models of grid cells. *PLoS computational biology*, 5(2):e1000291, 2009.

Daniel Bush, Caswell Barry, Daniel Manson, and Neil Burgess. Using grid cells for navigation. *Neuron*, 87(3): 507–520, 2015.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Christopher J Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *arXiv preprint arXiv:1803.07770*, 2018.

Licurgo de Almeida, Marco Idiart, and John E Lisman. The input–output transformation of the hippocampal granule cells: from grid cells to place fields. *Journal of Neuroscience*, 29(23):7504–7512, 2009.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 551–556. ACM, 2004.

Christian F Doeller, Caswell Barry, and Neil Burgess. Evidence for grid cells in a human memory network. *Nature*, 463(7281):657, 2010.

Uğur M Erdem and Michael Hasselmo. A goal-directed spatial navigation model using forward trajectory planning based on grid cells. *European Journal of Neuroscience*, 35(6):916–931, 2012.

Ila R Fiete, Yoram Burak, and Ted Brookings. What grid cells convey about rat location. *Journal of Neuroscience*, 28 (27):6858–6871, 2008.

William Fulton and Joe Harris. *Representation theory: a first course*, volume 129. Springer Science & Business Media, 2013.

Marianne Fyhn, Torkel Hafting, Menno P Witter, Edvard I Moser, and May-Britt Moser. Grid cells in mice. *Hippocampus*, 18(12):1230–1238, 2008.

Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801, 2005.

Joshua Jacobs, Christoph T Weidemann, Jonathan F Miller, Alec Solway, John F Burke, Xue-Xin Wei, Nanthia Suthana, Michael R Sperling, Ashwini D Sharan, Itzhak Fried, et al. Direct recordings of grid-like neuronal activity in human spatial navigation. *Nature neuroscience*, 16(9):1188, 2013.

Nathaniel J Killian, Michael J Jutras, and Elizabeth A Buffalo. A map of visual space in the primate entorhinal cortex. *Nature*, 491(7426):761, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401 (6755):788, 1999.

Bruce L McNaughton, Francesco P Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser. Path integration and the neural basis of the'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663, 2006.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 3111–3119, 2013.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pp. 849–856, 2002.

Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

Sameet Sreenivasan and Ila Fiete. Grid cells generate an analog error-correcting code for singularly precise neural computation. *Nature neuroscience*, 14(10):1330, 2011.

Michael M Yartsev, Menno P Witter, and Nachum Ulanovsky. Grid cells without theta oscillations in the entorhinal cortex of bats. *Nature*, 479(7371):103, 2011.

A 2D PATH PLANNING WITH OBSTACLES

Please see figure 5.

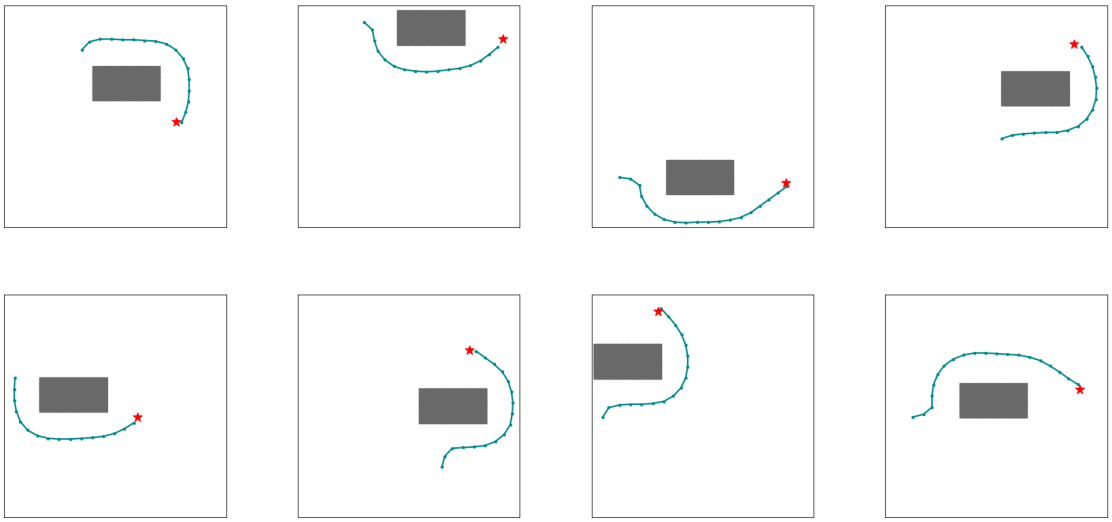


Figure 5: 2D path planning with obstacles

B LEARNED 3D GRID PATTERNS

Please see figure 6 and 7.

C 3D PATH INTEGRAL

Please see figure 8.

D 3D PATH PLANNING

Please see figure 9.

E 3D PATH PLANNING WITH OBSTACLES

Please see figure 10.

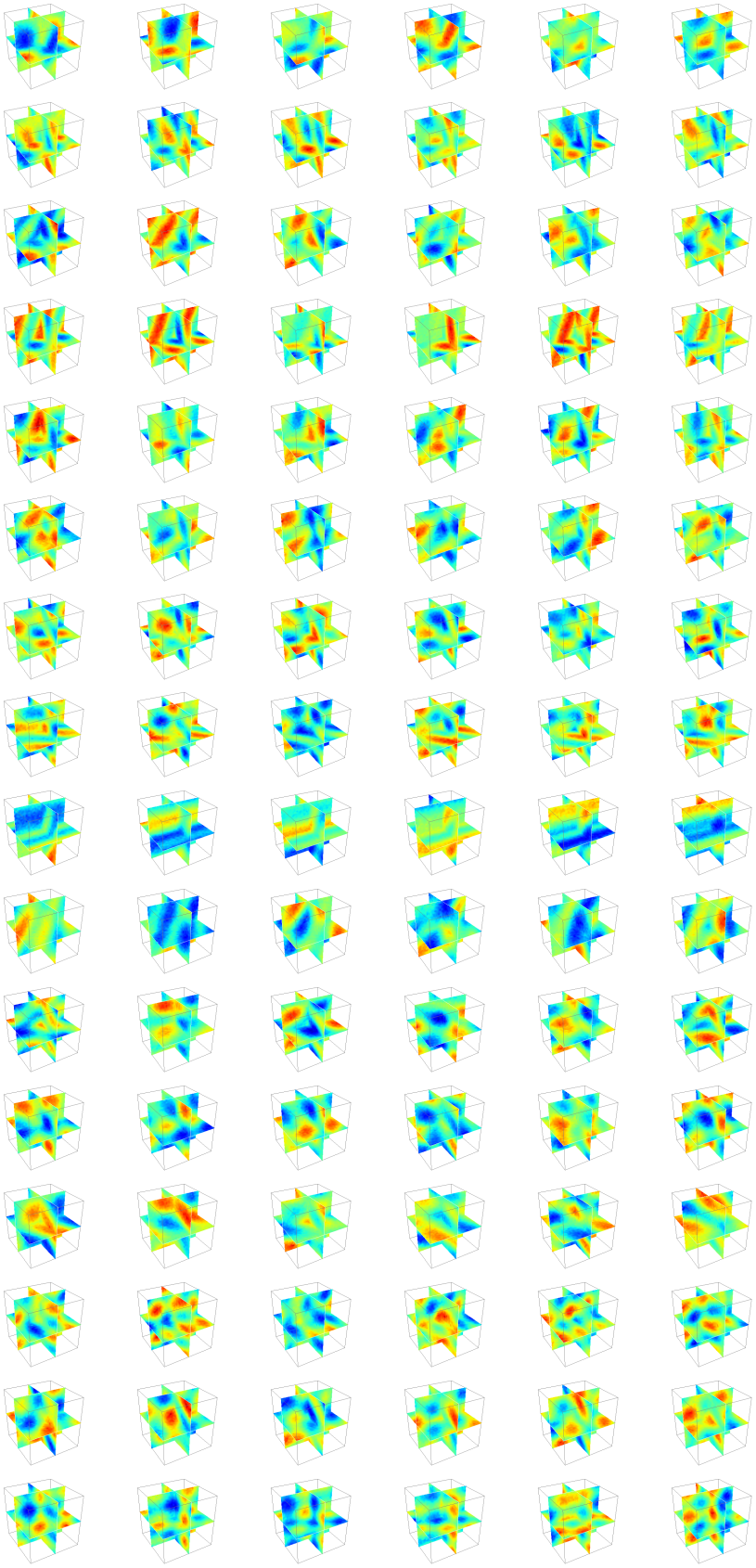


Figure 6: Learned 3D grid patterns

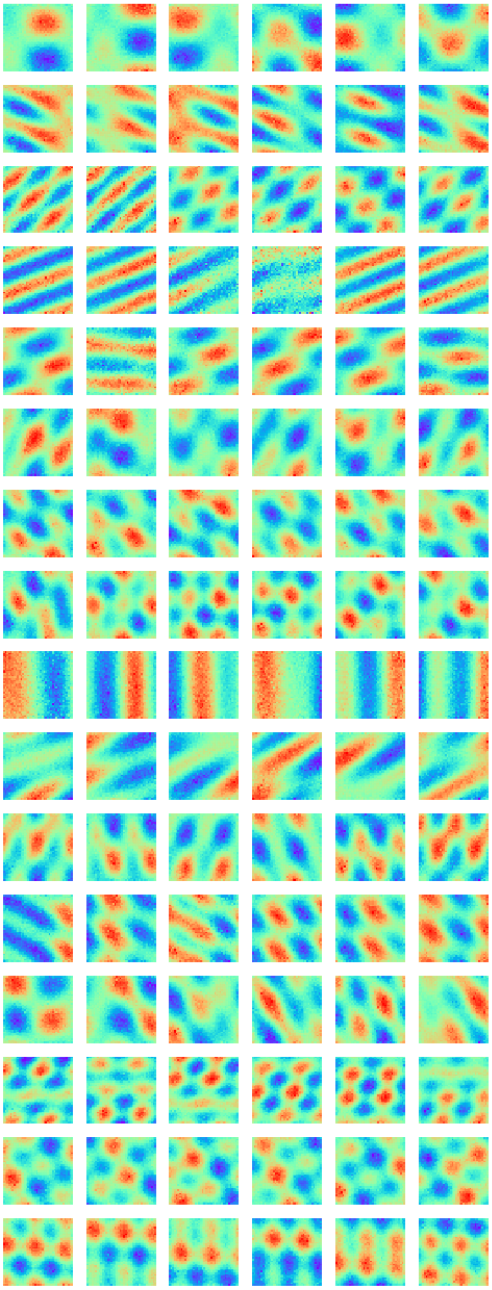


Figure 7: Slices of the learned 3D grid patterns

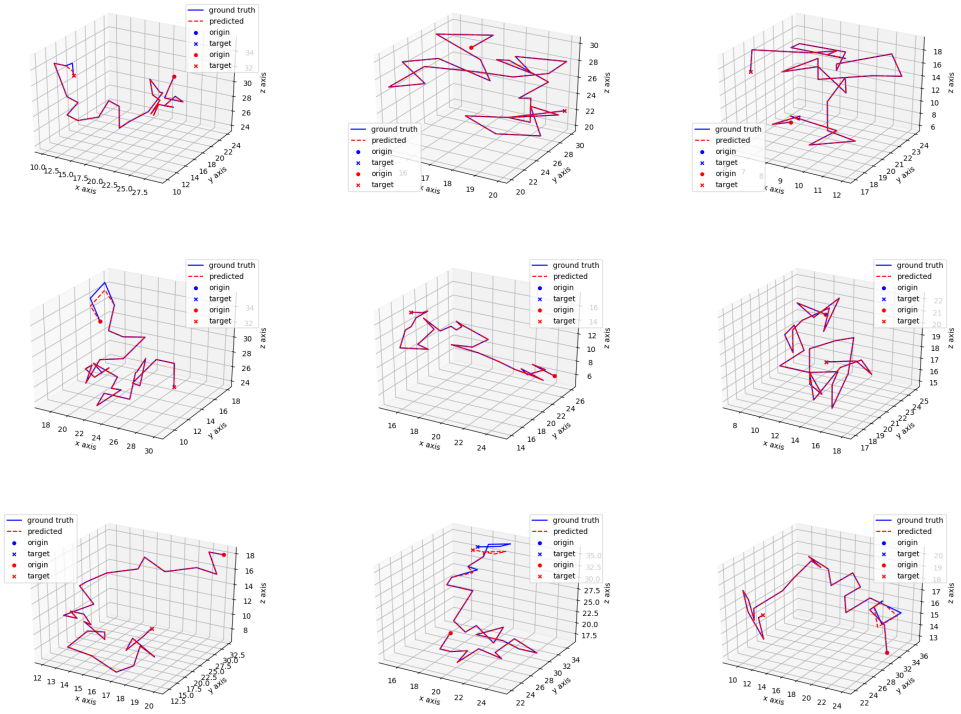


Figure 8: 3D path integral

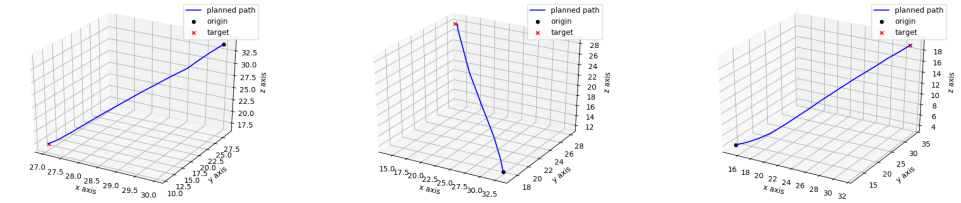


Figure 9: 3D path planning

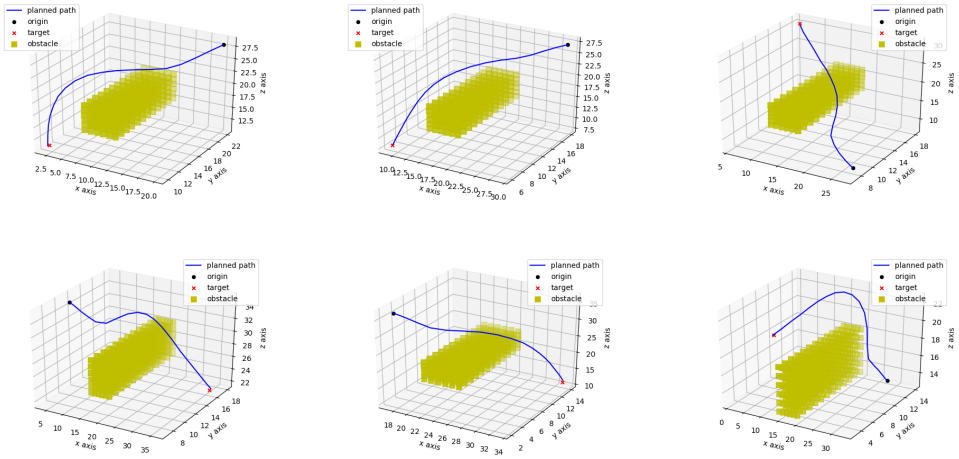


Figure 10: 3D path planning with obstacles