Real-time Navigation with Virtual Magnetic Fields Supplementary Material

Majda Moussa and Giovanni Beltrame

January 7, 2021

1 Physics-Based Model

Consider an arena in which the magnetic field is given by a vector field $\vec{B}$. At each point $p(x, y)$ we have $\vec{B}(x, y)$ and its gradient $\vec{G}(x, y)$. $\vec{G}$ is a vector that points in the direction in which $\vec{B}$ rises most quickly and its magnitude determines how fast the field rises in that direction. The gradient information can be used to optimally connect any point in the environment to a goal.

In a cluttered environment with obstacles, the goal is assigned to the highest conductivity whereas the obstacles are assigned to zero conductivity. The environment is assigned to a degraded conductivity which fades away when moving away from the goal (see equation 1). Electrical currents are assumed to be floating in the environment, and the magnetic field induced by these currents is used to find a free and optimal path to the goal.

The magnetic field can be modeled with Maxwell's equations:

$$\frac{\partial \vec{B}}{\partial t} = \nabla \times \vec{E} \tag{1}$$

$$\nabla \times \vec{B} = \mu_0 \vec{J} + \mu_0 \frac{\partial \vec{D}}{\partial t} \tag{2}$$

$$\nabla \cdot \vec{B} = 0 \tag{3}$$

where $\vec{B}$ is the magnetic field, $\vec{E}$ is the electrical field, $\vec{J}$ is the density of electrical currents and μ_0 is the free space permeability. Equation 1 (Maxwell-Faraday) draws the relationship between the magnetic field $\vec{B}$ and the electrical field $\vec{E}$. This relationship states that a time-varying magnetic field will always coexist with a spatially varying, non-conservative electric field, and vice-versa. Maxwell-Ampere's equation (Equation 2) states that a magnetic field $\vec{B}$ can be generated in two ways: i) by electrical currents as expressed by Ampere's law (see Equation 4) and ii) by changing electric fields (a.k.a., displacement currents $\vec{D}$) generated by the magnetic field induced by the floating currents $\vec{J}$. Under a low frequency ω = 0.05, it is safe to neglect displacement currents. Hence, Equation 2 can be then reduced to Equation 4:

$$\nabla \times \vec{B} = \mu_0 \vec{J} \tag{4}$$

According to Ohm's law, the current density $\vec{J}$ is:

$$\vec{J} = \sigma \vec{E} \tag{5}$$

where σ is the electrical conductivity. The magnetic field $\vec{B}$ is usually an alternating field and can be expressed in a time-dependent form as:

$$\vec{B} = \vec{B_0} \cdot e^{i\omega t} \tag{6}$$

By applying Ohm's Law (Equation 5) and Faraday-Maxwell (Equation 1), Equation 4 can be written as:

$$\nabla \times \nabla \times \vec{B} = -k^2 \vec{B}. \tag{7}$$

where $k^2 = i\omega\mu_0\sigma$, defined for the goal, obstacles and the free environment as in Table S1. Gauss' law for magnetism (Equation 3) asserts that the net outflow of the magnetic field through any closed surface is zero. Using Gauss' law along with the vector calculus identity relationship $\nabla \times (\nabla \times \vec{B}) = \nabla(\nabla \cdot \vec{B}) - \nabla^2 \vec{B}$, Equation 7 reduces to:

$$\nabla^2 \vec{B} = -k^2 \vec{B} \tag{8}$$

2 Dataset

We implemented Equation 8 in COMSOL[1] and solved using its finite element solver. Table S1 summarizes all the physics model parameters. We used COMSOL LiveLink for MATLAB[2] to automate the dataset collection. Using MATLAB, we have randomly generated obstacles in a square arena (10x10m). The geometrical properties of the arena, obstacles and goal are depicted in Table S1. The generated scene is sent to COMSOL to compute the distribution of the magnetic field, which is then saved to disk. Fig. 1A shows an example of a magnetic field computed by COMSOL given a conductivity distribution. A dedicated script was implemented to process COMSOL's output and store the scene as an image-like array where each pixel corresponds to a magnetic field value.

We build a dataset with 20,000 samples. Each sample is an image-like array with two channels: i) the scene conductivity distribution and ii) the corresponding magnetic field distribution. The size of each sample is 101x101. This choice allows a fast training and is suitable for both indoor and outdoor use. For an indoor application, we can cover an area of 10x10m with a step resolution of 0.1m. For an outdoor application, we can use the same model to process 100x100m with a resolution of 1m. Furthermore, this choice makes us able to show the applicability of our

[1] https://www.comsol.com/

[2] https://www.comsol.com/release/5.4/livelink-matlab

approach with limited GPU resources. However, for industrial use, more advanced GPUs can be leveraged to train the model with finer resolution.

The full dataset (5GB) is available here: `https://drive.google.com/file/d/189jWIeEXEX0YKXwD3Bs9D7d6C-VZBRiK/view?usp=sharing`. Some examples of a maze-like map are sown in Fig. 1.

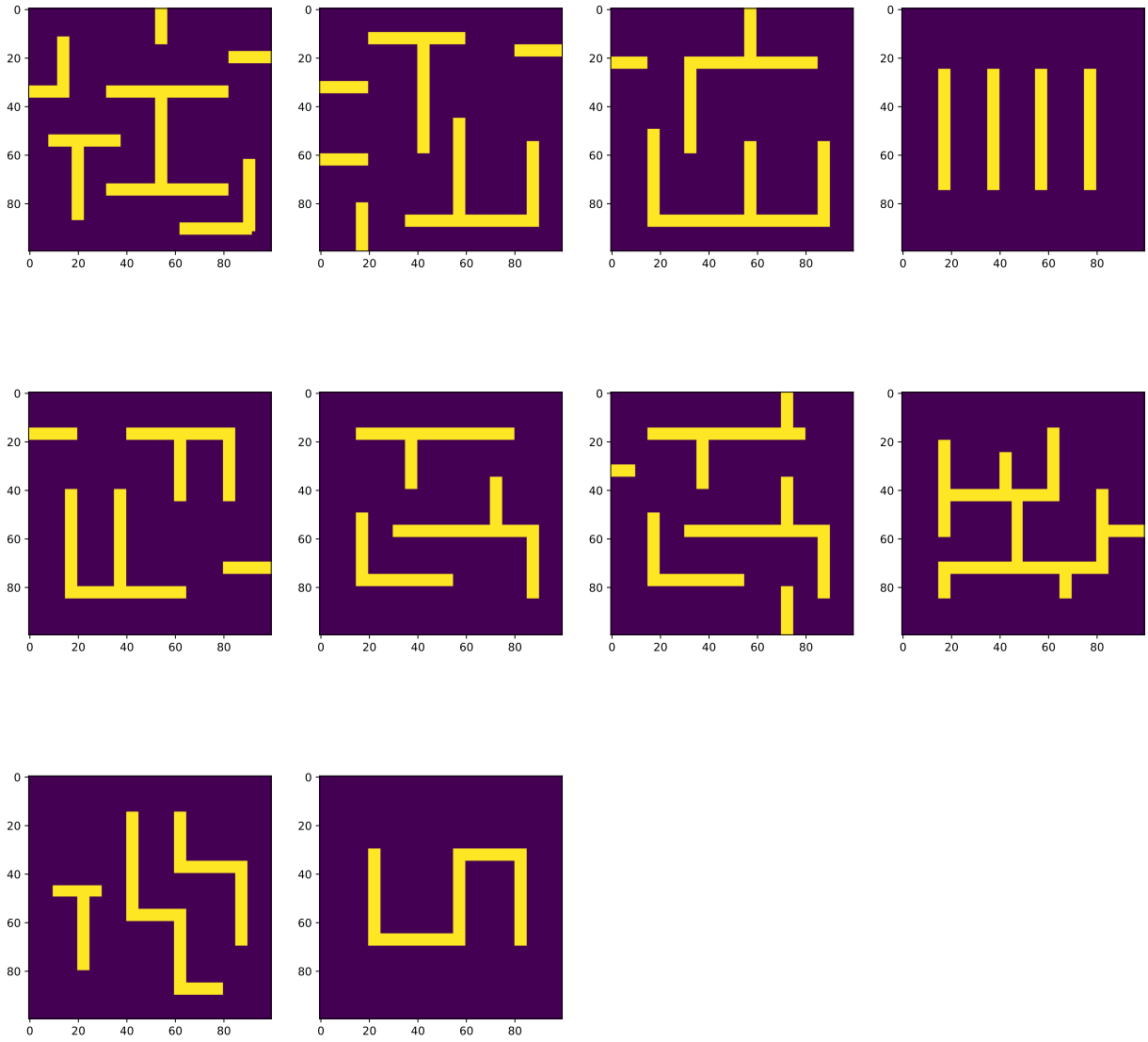

Figure 1: Some example maps used for the evaluation

3 Optimization and Training

The internal structure of MaxConvNet is shown in the Fig. 2 in the paper and Table S2. MaxConvNet is implemented as a convolutional auto-encoder trained in a supervised manner. In the encoder stage, the conductivity map of a single environment is processed by multiple layers of convolution, max pooling, non linearity and batch normalization. In particular, the encoder stage consists in five blocks of layers whereby each convolutional layer is followed by a pooling layer, a rectified linear unit (ReLU) and batch normalization.

In the decoder stage, the received data is iteratively up-sampled using nearest neighbors [1] and processed using convolutions followed by non-linearity [2] and batch normalization layers [3]. To avoid overfitting, we use dropout [4] and $\mathcal{L}2$ regularization [5]. Dropout randomly selects some nodes and removes them along with all of their incoming and outgoing connections according to an hyperprameter which defines the rate of dropping. L2 regularization updates the general loss function by adding a regularization term. The cost function with the regularization term is:

$$Loss_{MSE} = \frac{(\phi - \hat{\phi})^2}{n} + \frac{\lambda}{2n} \sum_{n} w \tag{9}$$

where ϕ is the predicted potential, $\hat{\phi}$ is the potential computed by COMSOL, λ is a hyperparameter, n is the size of the training batch and w is weights of the network.

The learnable parameters of MaxConvNet are iteratively adjusted with the error backpropagation algorithm [6]. The neural network is trained by minimizing the sum of squared errors loss between COMSOL-computed and predicted magnetic field distributions. The use of a CNN allows us to considerably reduce the number of learnable parameters using localized and shared receptive field structures. This allows faster training compared to a conventional deep neural network. We use the Adam optimizer [7] to adjust the network parameters.

We initialize the weights using the method described in [8]. We implement the network in Tensorflow [9] and we train on an NVIDIA GTX Gforce 1080 TI GPU (11GB) with a batch size of 50 samples.

4 Path Following

The proposed planner generates a path as a set of points connected by straight lines. To navigate the path, a robot shall move from point to point until it reaches the end of the goal. We implement a modified version of Craig Reynold's path-following algorithm [10] to ensure smooth navigation. Algorithm 1 outlines the different steps of the steering behavior: given its current position and velocity, a robot estimates its future location and projects it on all the path's segments. The robot sets its next target to the closest projected point, also called the normal point. If all the normal points fall outside the path segments then the robot sets its next target to the l^{th} closest normal point where l is a fixed offset. Given the next target, the robot computes the velocity command if the distance between the next target and the predicted future location

Algorithm 1: Modified Version of Craig Reynold's path-following algorithm.

Input: A path $\mathcal{P} = \{\psi_1, \psi_2, ..., \psi_n\}$, where ψ_i is a line segment *s.t.* $\psi_i = [\alpha_i \beta_i]$, α_i is the segment's start point and β_i is the segment's end point, n is the number of segments in the path and r is the path radius.

Output: Velocity command $\vec{V}^{(t+1)}$

Get current position $p^{(t)}$;
Get current velocity $\vec{V}^{(t)}$;
Predict vehicule's future location $\hat{p}^{(t+1)} \leftarrow f(p^{(t)}, \vec{V}^{(t)})$;
for ψ_i *in* $\mathcal{P}$ **do**
    Get projection $p_i^{\perp}$ of $\hat{p}^{(t+1)}$ on ψ_i ;
    if $p_i^{\perp} \notin \psi_i$ **then**
        $p_i^{\perp} \leftarrow \beta_i$;
    end
    Calculate $d_i = \| \overrightarrow{\hat{p}^{(t+1)} p_i^{\perp}} \|$;
end
if $p_i^{\perp} \notin \psi_i \forall i \in [0..n]$ **then**
    Set target position $p^{(t+1)} = p_{k+l}^{\perp}$ *s.t.* $k = \text{argmin}_{i \in [1..n]} d_i$ where l is a fixed offset;
else
    Set target position $p^{(t+1)} = p_k^{\perp}$ *s.t.* $k = \text{argmin}_{i \in [1..n]} d_i$;
end
if $\| \overrightarrow{\hat{p}^{(t+1)} p_i^{\perp}} \| > r$ **then**
    calculate velocity command $\vec{V}^{(t+1)} = \overrightarrow{p^{(t)} p^{t+1}}$;
end
Strip the travelled distance from the current path $\mathcal{P}$

is greater than the path radius. With a smaller radius r, the robot has to follow the path more closely; a wider radius allows it to stray more. As it moves, the robot strips the traveled distance from the path to avoid oscillatory behavior. Table S4 summarizes Reynold's parameters.

References

[1] D. Han, "Comparison of commonly used image interpolation methods," *Proc. of the 2nd International Conference on Computer Science and Electronics Engineering*, 2013.

[2] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Proc. of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proc. International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] B. Bilgic, I. Chatnuntawech, A. P. Fan, K. Setsompop, S. F. Cauley, L. L. Wald, and E. Adalsteinsson, "Fast image reconstruction with l2-regularization," *Journal of magnetic resonance imaging*, vol. 40, no. 1, pp. 181–191, 2014.

[6] A. Van Ooyen and B. Nienhuis, "Improving the convergence of the back-propagation algorithm," *Neural networks*, vol. 5, no. 3, pp. 465–471, 1992.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proc. International Conference on Learning Representations (ICLR)*, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proc. of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning," *OSDI*, vol. 16, pp. 265–283, 2016.

[10] C. W. Reynolds, "Steering behaviors for autonomous characters," *Game developers conference*, vol. 1999, pp. 763–782, 1999.

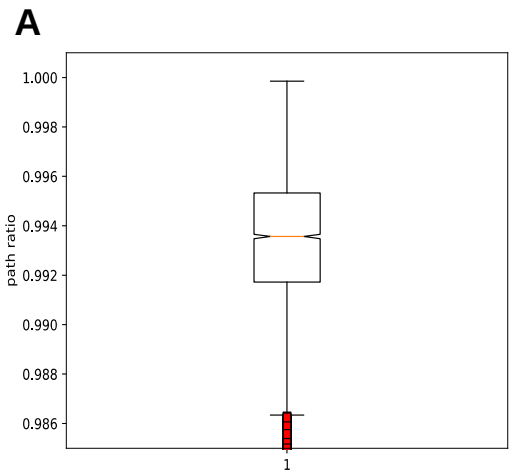

Fig. S1. The ratio between the path computed using the real magnetic distribution and the one computed using the machine predicted distribution: (A) is a different visualization to Fig. 6: The length ratio variable follows a Normal Distribution *N(μ=0.993, σ= 0.005),* The median value is 0.993.

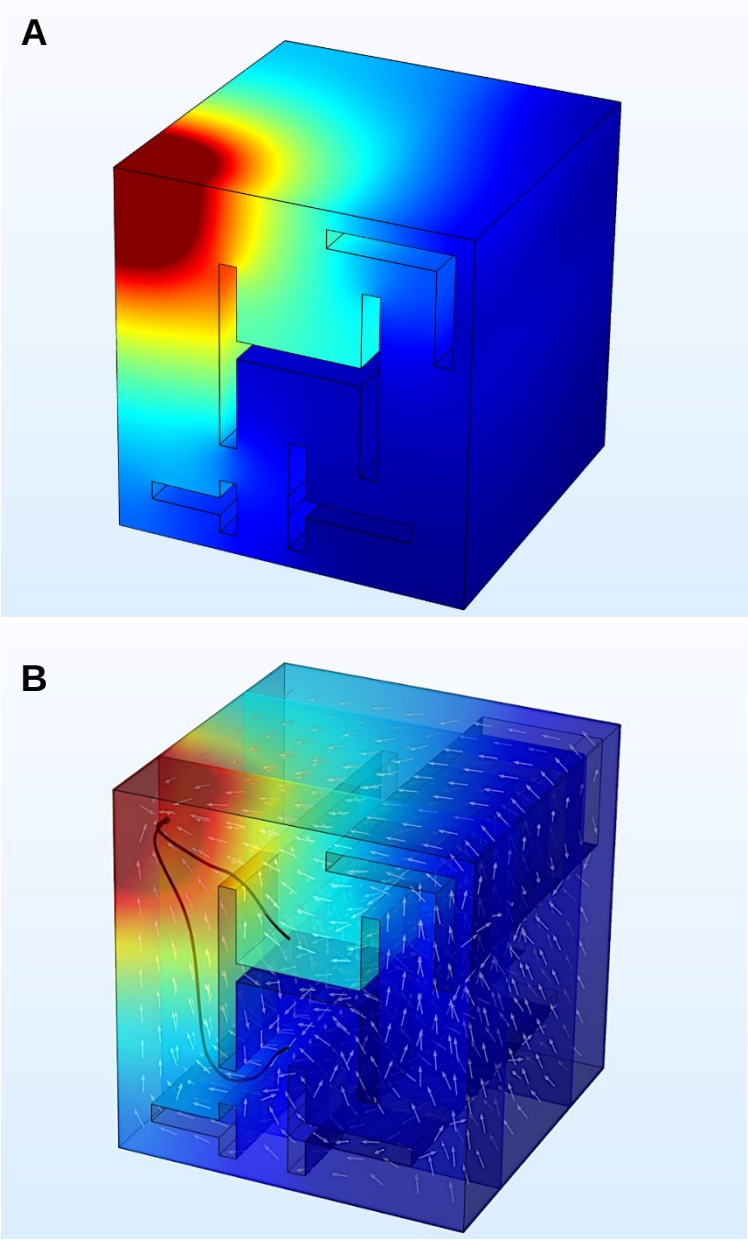

Fig. S2. (A) Magnetic field distribution computed in a 3D environment using our physics model. (B) We computed the gradient information based on the 3D distribution shown in (A). A path is computed for two different locations to the goal (in red). This shows our approach can be easily extended to handle 3D environments.

Table S1.

	Parameters		**Definition**	**Values**
Physical Properties	σ_g		Goal conductivity constant	1e6
	σ_o		Obstacles conductivity constant	0
	σ_e		Environment conductivity constant	100
	μ_0		Free space permeability	12.56e-7
	ω		Frequency	0.05
	k^2	K_g^2	Goal wave number	$\mu_0*\sigma_g*\omega*i$
		k_o^2	Obstacle wave number	$\mu_0*\sigma_o*\omega*i$
		k_e^2	Environment wave number	$\mu_0*\sigma_e*\omega*i$
Geometric Properties	$Shape_g/size_g$		Goal shape/size	Circle/radius=0.09m
	$Shape_o/size_o$		Obstacles shape/size	Random/Random
	$Shape_e/size_e$		Environment shape/size	Square/side=10m
	(x_g,y_g)		Goal local coordinates	([0,10], [0,10])
	Meshing	Type	Type of the mesh element	Triangular
		Hmin/ Hmax	Approximate size of the mesh element	0.3/0.5m

Table S1. Physics model: the physical properties of the different mediums in the model (goal, obstacles and free environment) and their geometrical properties including their shapes, sizes and the meshing.

Module Name	Filter size	#Filters/Channels	Stride/Up_ Factor	Dropout	Pooling Size/stride	Batch Norm	Non-Linearity
Conv1	11x11	32	1x1/-	0.1	1x1/2x2	Y	ReLU
Conv2	7x7	64	1x1/-	0.1	1x1/2x2	Y	ReLU
Conv3	5x5	128	1x1/-	0.1	1x1/2x2	Y	ReLU
Conv4	3x3	256	1x1/-	0.1	1x1/2x2	Y	ReLU
Conv5	3x3	512	1x1/-	0.1	1x1/2x2	Y	ReLU
NearestUpSample **Conv6**	- 3x3	512	-/7 1x1/-	0.1	- -	Y	- ReLU
NearestUpSample **Conv7**	- 3x3	256	-/13 1x1/-	0.1	- -	Y	- ReLU
NearestUpSample **Conv8**	- 5x5	128	-/26 1x1/-	0.1	- -	Y	- ReLU
NearestUpSample **Conv9**	- 7x7	64	-/51 1x1/-	0.1	- -	Y	- ReLU
NearestUpSample **Conv10**	- 11x11	32	-/101 1x1/-	0.1	- -	Y	- ReLU
Conv11	1x1	1	1x1/-	-	-	-	ReLU

Table S2. Conv is a convolutional layer with a specified filter size, stride and number of filters. Conv1-5 are in the encoder stage, whereas Conv6-11 are in decoder stage. The Batch Norm column indicates whether Conv is followed by a Batch Normalization layer. The Nonlinearity column shows whether and what nonlinearity layer is used (preceding the Batch Norm if Batch Norm is used). NearestUpSample module allows to resize the input according to the Up_Factor parameter. Pooling is used for subsampling and nearest neighbor filtering for up sampling.

Adam's parameters	definitions	values
lr	Learning rate	[0.01,1]
β_1	Exponential decay rate for the first moment estimates	0.9
β_2	Exponential decay rate for the second moment estimates	0.999
ε	Used to prevent division by zero	10e-8

Table S3. Adam's parameters used to compute a feasible path to the goal.