

# Reasoning in visual navigation of end-to-end trained agents: a dynamical systems approach

## Supplementary Material

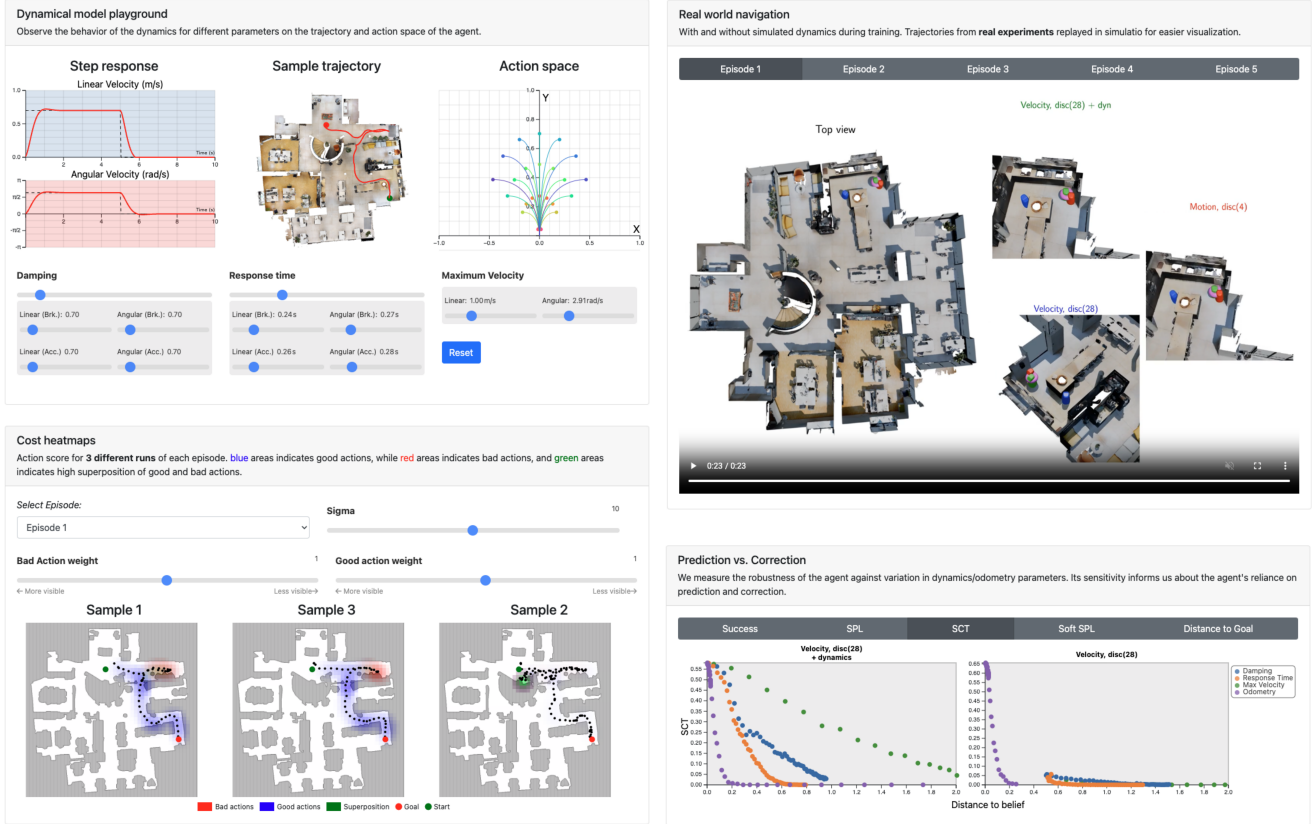


Figure 10. We created an interactive website featuring several data visualization tools to help illustrating our findings, such as a real-time dynamical model similar to the one used in the simulator, allowing to observe directly the impact of each parameter on the behavior of the robot. (<http://visual-navigation-reasoning.github.io>)

### A. Interactive website

We developed an interactive website to support our findings and help better visualize the results of our experiments. In particular, our project page features an interactive second order dynamical model similar to the one implemented in the simulator. Several sliders control the value of physical parameters from the model, and the animated figure displays the impact on the step response, the trajectory and the action space in real-time. We also replayed real episodes from the different methods in Table 1 synchronized on the same scene to better compare them — although these episodes are replayed in the simulator, these were recorded with the agent **running on real robots**, poses estimated and then shown in the simulator. Figure 2 is replicated with different metrics and visualization of the distance to belief for each point on the figure. The planning quality map (Figure

8) is also reproduced, with control over the parameters of the density estimation. Figure 10 shows some of the tools available on the website.

### B. Calculation of $D_{belief}$

The *distance to belief* measures the discrepancy between nominal trajectories within the in-domain environment and out-of-domain trajectories in the corrupted environment, hence modeling the impact of a change in configuration  $\Delta E$ . Formally, let us define a function  $F_\theta : \mathcal{A} \times \mathcal{P} \mapsto \mathcal{P}$  corresponding to the forward step of the environment parametrized by some physical parameters  $\theta \in \Omega$ . The function  $F_\theta$  maps an action  $a \in \mathcal{A}$  and a current state  $p_t \in \mathcal{P}$  (position + velocity) to the future state  $p_{t+1}$ . To simplify,  $F_\theta$  models solely the dynamics of the robot, collisions are ignored to compute  $D_{belief}$ .

The *distance to belief* is computed on a fixed set of  $k \in \llbracket 1, K \rrbracket$  action sequences and an initial state  $\{p_0, a_0, \dots, a_T\}_k$  where  $T$  is the length of the sequences. The metric is defined as follows:

$$D_{\text{belief}} = \frac{1}{TK} \sum_{t,k} \|p_t - \bar{p}_t\| \text{ s.t. } \begin{cases} p_{t+1} &= F_{\theta}(p_t, a_t) \\ \bar{p}_{t+1} &= F_{\theta'}(\bar{p}_t, a_t) \\ \bar{p}_0 &= p_0, \end{cases} \quad (4)$$

where  $\theta'$  is the set of corrupted environment parameters. In simple words, the distance to belief is proportional to the area between the in-domain and out-of-domain trajectories, and is measured in meters, so  $D_{\text{belief}} = 0.25$  can be interpreted as *the corrupted trajectory diverges from the in-domain by 0.25 m in average* (see Figure 11). We compute  $D_{\text{belief}}$  in Figure 2 using  $K = 1,000$  sequences on  $T = 15$  steps (corresponding to 5 s). The action sequences are collected by sampling navigation episodes from the train set of HM3D solved by the model being studied (**D28-dynamics** for Figure 2 (left), and **D28-instant** for Figure 2 (middle)). The distance to belief is actually independent from the policy, which is only used to collect meaningful action sequences corresponding to realistic movements. The calculation of  $D_{\text{belief}}$  only depends on the physical parameters of the environment.

As a rule of thumb,  $D_{\text{belief}}$  values above 1.0 m can be considered as highly corrupted environment, and reasonable values (arguably comparable with sim2real distance between the robot dynamics and the simulated model) lie in  $[0, 0.5]$ . Our interactive website shows the relation between the corrupted trajectory and the distance to belief.

### C. Details on Prediction vs. Correction

The dynamics of the real robot is modeled in the simulator using a second order dynamical model similar to [9]. Let  $v(t), \omega(t)$  be the linear and angular velocity of the agent, and  $c_v(t), c_\omega(t)$  be the linear and angular actions taken at time  $t$ . The dynamical model is

$$\begin{aligned} \ddot{v}(t) &= \frac{1}{\tau}(v(t) - c_v(t)) + \frac{2\gamma}{\tau}\dot{v}(t) \\ \ddot{\omega}(t) &= \frac{1}{\tau}(\omega(t) - c_\omega(t)) + \frac{2\gamma}{\tau}\dot{\omega}(t), \end{aligned} \quad (5)$$

where  $\tau$  and  $\gamma$  are the response time and damping factor. We apply different values depending on the motion direction (acceleration or braking) and type (linear or angular), resulting in four different values for each constant parameter. We also apply saturation on acceleration (absent of this study) and on the velocity. Note that modification of the maximum velocity changes not only the saturation value, but also the distribution of discrete action which are sampled in  $v(t), \omega(t) \in [0, v_{\max}] \times [0, \omega_{\max}]$ . In other words, the 28 discrete actions are always scaled to fit the range of possible velocities. The dynamical model runs  $10\times$  faster than the policy to prevent aliasing effects.

Corrupted environments are generated by multiplying

one of the physical parameters by a constant *change factor*  $f$ , while leaving the other parameters untouched. Such a change results in a drop of performance on one hand, and an increase of the distance to belief on the other. As mentioned in the main paper, the factor  $f$  causes a change in environment parameters  $\Delta E$ , which has different interpretations depending on the physical quantity and its impact on the dynamics. We unwrap Figure 2 and exposed the change factor in Figure 12. In particular, we show the relationship between the change factor and the distance to belief, and its impact on SPL. We manually define suitable ranges for each corruption type (damping, max. velocity and response time) up to 0% SR, and evaluate the agents (b) and (c) from Table 1 on **(HM3D/250)** using linearly sampled change factors within the range.

Figure 12 shows that a fixed value of change factor  $f$  can correspond to very different values of distance to belief depending on the corrupted parameter, which motivates the use of a proxy metric such as the distance of belief. We also observe steeper curves for SPL when testing the **D28-instant** variant compared to **D28-dynamics**, which confirms that training dynamic-aware agent allows the adaptation to different dynamic unseen during training.

### D. Probing future pose: variants

Our goal is to probe the existence of a plan in the latent state of the navigation agent. To do so, we collected a dataset of 500,000 navigation episodes generated from a trained **D28-dynamics** agent and stored the latent state, action and path  $(a_t, p_t, h_t)$ . We split this dataset in proper train/validation/testing sets (80%, 10%, 10%). During training, a random time instant  $t$  is sampled in the episode, and the probing network is supervised to predict the future positions  $p_{t+1}, \dots, p_{t+H}$  from the first latent state  $h_t$ . Future latent states  $h_{t+i}$  are not provided to the probing network, which can not use new observations to improve the predicted path. The probing network is trained to minimize

$$\mathcal{L}_{\text{probing}} = \sum_{i=1}^H \underbrace{\left\| \begin{bmatrix} x_{t+i} \\ y_{t+i} \end{bmatrix} - \begin{bmatrix} \hat{x}_{t+i} \\ \hat{y}_{t+i} \end{bmatrix} \right\|_2^2}_{\text{pos. loss}} + \underbrace{\left\| \begin{bmatrix} \cos \theta_{t+i} \\ \sin \theta_{t+i} \end{bmatrix} - \begin{bmatrix} \cos \hat{\theta}_{t+i} \\ \sin \hat{\theta}_{t+i} \end{bmatrix} \right\|_2^2}_{\text{rot. loss}} \quad (6)$$

assuming  $p_t = [x_t \ y_t \ \theta_t]$  and  $\hat{p}_t$  being the predicted pose. We used the Adam optimizer (learning rate  $= 10^{-4}$ ) with a batch size of 64, a prediction horizon  $H = 20$  and performed 100,000 gradient updates. We tested different architectures of the probing network:

**Linear** uses a straightforward linear layer per time step to

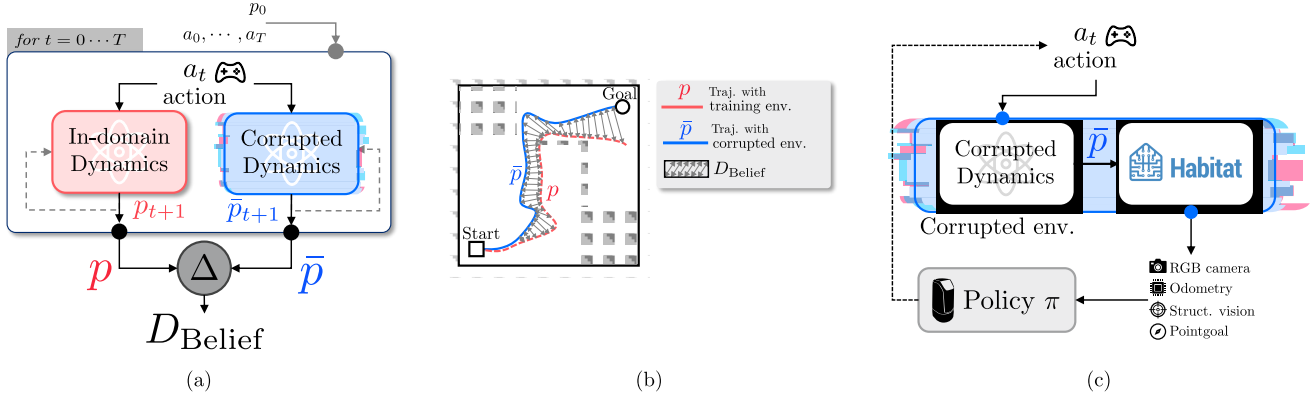


Figure 11. **Comparing Prediction vs. Correction** – steps in an end-to-end dynamic agent is achieved by testing the policy in an *corrupted environment* where one of the step is made less accurate. Since changes in environment parameters  $\Delta E$  are not comparable, we rely on the proposed *distance to belief* to measure the impact of a change on the agent trajectory. (a) To compute this metric, we simulate trajectories generated by two different dynamical systems albeit from the same sequence of actions. (b) The distance to belief corresponds to the distance between the resulting trajectories without taking collisions into account. (c) While  $D_{\text{belief}}$  is calculated by running an agent in the corrupted environment with the same actions as the agent had done in-domain, of course the actual success rate of the agent in the corrupted environment is calculated by letting the agent take its own decisions.

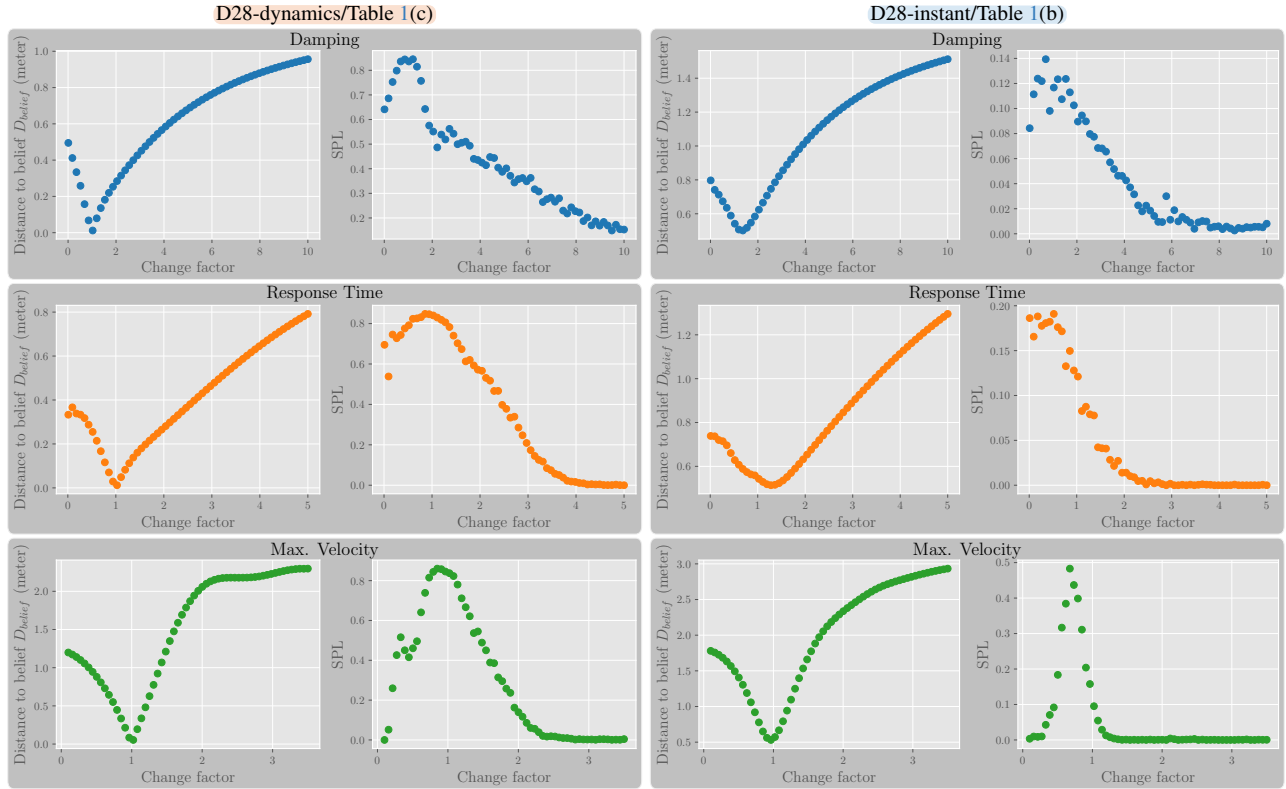


Figure 12. **Detailed results Prediction vs. Correction** – for D28-dynamics (left) and D28-instant (right). The figures shows, for each corruption type, the distance to belief and SPL score for different multiplicative factor  $f$ . We observe better robustness of the dynamic-aware agent against changes in the robot dynamics. Non-linear dependence between change factor and impact on the agent trajectory motivates the use of the distance to belief as a proxy to measure the effect of each corrupted environment.

predict the future position from the initial latent state.

$$\forall i, \hat{\mathbf{p}}_{t+i} = \text{Linear}_i(\mathbf{h}_t), \quad (7)$$

**Linear + non-linear(action,goal)** exploits a non-linear embedding of the previous action and the goal direction (given in polar coordinates with respect to the episode start).

$$\forall i, \hat{\mathbf{p}}_{t+i} = \text{Linear}_i(\mathbf{h}_t, \text{MLP}(\mathbf{a}_{t+i-1}, \mathbf{g})) \quad (8)$$

**GRU-agent** where we use the latent dynamics of the trained agent itself for prediction. Recall that the hidden state is updated by a GRU, cf. eq. (1), which we reproduce here in a simplified notation without gates and only a single layer,

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{o}_t), \quad (9)$$

where  $\mathbf{o}_t$  is a concatenation of all observation features,  $\mathbf{W}$  is the matrix modeling latent dynamics,  $\mathbf{V}$  projecting observations into the latent space, and  $\sigma$  an activation function. Yet, since future observations  $\mathbf{o}_{t+i}$  are not available during probing, we replace them by a transformation of the previous latent state performed by an MLP  $\psi_\theta$  (3 layers, 1024 units, TanH activated) which compensates the absence of observations.

$$\begin{aligned} \mathbf{h}'_{t+i} &= \sigma(\mathbf{W}\mathbf{h}_{t+i-1} + \psi_\theta(\mathbf{h}_{t+i-1})) \\ \hat{\mathbf{p}}_{t+\tau} &= \phi(\mathbf{h}'_{t+\tau}). \end{aligned} \quad (10)$$

All variants use a linear projection layer  $\phi$  to map the latent space to the predicted position  $\hat{\mathbf{p}}_t$ .

## E. Zeroing the hidden state of the agent

In Table 4 of the main paper we described results ablating the memory of the agent, ie. setting the hidden GRU state  $\mathbf{h}_t$  to zero. We argue that zeroing  $\mathbf{h}_t$  only makes sense if at the same time we reset the episode specific coordinate frame of the agent, which defines the static goal vector  $\mathbf{g}_0$  and the pose inputs  $\hat{\mathbf{p}}_t^r$  and  $\hat{\mathbf{p}}_t^a$ . This is motivated by the fact, that the *PointGoal* task requires the agent to understand where it is with respect to the goal, ie. it needs to have an understanding of the (unobserved!) dynamic goal vector  $\mathbf{g}_t$ , which is defined in its own egocentric frame. It can only do this by using the static goal vector  $\mathbf{g}_0$  defined w.r.t. the episode start, and localization information, provided by  $\hat{\mathbf{p}}_t^r$  and  $\hat{\mathbf{p}}_t^a$ , also in the same frame. The calculation can be done by a simple rigid transform, but the noise of these inputs will lead to a noisy signal. This noise can be filtered, for which the hidden memory of the agent is likely used. For this reason zeroing memory without resetting the episode centric coordinate can potentially lead to very undesirable effects.

## F. Details on the planning heatmap

In order to gain deeper insights into the specific areas where our robot encounters difficulties in navigation, we generated heatmaps that visually highlight challenging locations

within our environment. This allows us to focus our efforts on improving the robot's navigation in these identified areas. Below is the detailed description of the heatmap generation.

Let  $\mathbf{p}_t = [x_t \ y_t \ v_t \ \omega_t]^T$  represents the state of an agent, where  $x_t, y_t$  are its 2D coordinates,  $v_t$  is the linear velocity, and  $\omega_t$  is the angular velocity. Given a goal  $\mathbf{g}$ , we define  $\mathcal{T}(\mathbf{p}_t, \mathbf{g})$  as the *time to goal*, which is the travel time to reach the goal. This value is computed by solving the Eikonal equation with fast marching, assuming the agent navigates at full speed and slows down near the walls. For each navigable point on the grid, the velocity is computed as  $v(x, y) = V \times d(x, y)/K$  where  $V$  is the max velocity of the agent,  $d$  is the distance to nearest wall and  $K = 0.5$  a weighting coefficient.

We introduce a cost function  $\mathcal{C}(\mathbf{p}_t, a)$  at state  $\mathbf{p}_t$ , taking an action  $a$  as:

$$\begin{aligned} \mathcal{C}(\mathbf{p}_t, a) = & \overbrace{10 \times \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})}^{\text{pos}} \\ & + \overbrace{0.1 \times \tan\left(\frac{\nabla_x \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})}{-\nabla_y \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})}\right)}^{\text{angle}} \\ & + \overbrace{(v_{t+1} - \beta \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g}))}^{\text{slow down near goal}} \\ & + \overbrace{10^{-3} \times |\omega_t|}^{\text{rotation speed}} + \overbrace{10^3 \times \mathcal{P}(\mathbf{p}_{t+1})}^{\text{collision}} \quad (11) \end{aligned}$$

where  $\mathbf{p}_{t+1} = \mathcal{D}(\mathbf{p}_t, a)$  is the next state of the agent taking the action  $a$  given by the dynamical model  $\mathcal{D}$ ,  $\mathcal{P}(\mathbf{p}_t)$  the collision indicator, equal to 1 if the agent collides and 0 otherwise, and  $\beta$  represents the braking strength, or how rapidly the agent can decelerate.

Each term in the cost function has a specific purpose:

- **Position cost:**  $10 \times \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})$  based on the time estimated to reach the goal.
- **Angle alignment:**  $0.1 \times \tan\left(\frac{\nabla_x \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})}{-\nabla_y \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})}\right)$  encourages alignment with the goal direction.
- **Slowing near goal:**  $v_{t+1} - \beta \mathcal{T}(\mathbf{p}_{t+1}, \mathbf{g})$  slows the agent down as it approaches the goal.
- **Rotation speed cost:**  $10^{-3} \times |\omega_t|$  discourages high angular velocities.
- **Collision penalty:**  $10^3 \times \mathcal{P}(\mathbf{p}_{t+1})$  a large penalty applied if the agent collides.

This cost function is designed to balance reaching the goal quickly, maintaining alignment, slowing down near the goal, and avoiding high rotation speeds and collisions. Then we can replay the recorded trajectory, knowing the taken action at every position, we calculate the following metric  $M(t) = \mathcal{C}(\mathbf{p}_{t+1}, a_{t+1}) - \mathcal{C}(\mathbf{p}_t, a_t)$ . To create a smooth, continuous heatmap, we apply a Gaussian kernel at each position  $(x_t, y_t)$ , with a mean  $\mu = M(t)$  and standard deviation



Method	Sim(train) (HM3D/2.5k)			Sim(+dyn) (HM3D/2.5k)		
	SR%	SPL%	SCT%	SR%	SPL%	SCT%
	SR%	SPL%	SCT%	SR%	SPL%	SCT%
(a) D4	91.6	76.4	20.4	29.1	18.1	2.0
(b) D28-instant	98.3	82.4	66.5	27.6	11.6	5.0
(c) D28-dynamics	97.6	82.3	52.2	97.6	82.3	52.2

Table 6. **Evaluation in the training domain.** (Left) the agent is evaluated in the same action space and dynamics used for training. This evaluates the difficulty of the task, and not the transfer to the real physical robot. (Right) the agent is evaluated in simulation with a dynamical model — reproduced from Table 1 of the main paper.

tion  $\sigma = 0.5$ . This results in a heatmap that provides a clear spatial representation of the robot’s performance across the environment.

## G. Evaluation in the training domain

We evaluate the three agents of Table 1 also in a third setting: “**Simulation (train domain)**” evaluates them in simulation w/o motion model, ie. with instantaneous velocity changes and constant velocities between time steps, and with their respective action spaces. This evaluates the difficulty of the training task and does not provide indications on performance in a real environment.

## H. Evidence of Tunnel vision

We found some evidence of “tunnel vision”, by which we mean that the agent attempts strategies, which a human could easily discard even without having access to a map. This is not necessarily a problem for successful completion of the episodes, as the agent detects blockings and searches for alternatives, eventually finding the goal. However, it is not efficient, and translates into lower than optimal SPL measures.

An example is seen in Figure 13. In this episode, starting at position ① and aiming for the goal position at ④, the agent tries to pass through the path indicated by the red trajectory, doing a turn into the area indicated by ③ although it is clearly visible (from position ① already), that there is no path possible between ② and ③. Although the dotted part is occluded, a human would be able to estimate that it is blocked.

## I. Details on visual localization

As an alternative to Adaptive Monte-Carlo Localization (AMCL)[76], we experimented with a custom visual localization system, cf. Table 5 in the main paper. Here we provide more details on the setup.

For the pre-mapping part, we follow a procedure similar to the one describe in [45]: a dedicated robot is driven



Figure 13. **Tunnel vision:** the agent attempts to navigate along the red trajectory, although it is clearly visible that it is blocked, although the dotted part is occluded.

through the environment capturing synchronized 3D LIDARs, RGB cameras and odometry data. Both standard *Simultaneous Localization And Mapping* (SLAM) using *Iterative Closest Point* (ICP) on the LIDAR point-clouds and *Structure-from-Motion* (SfM) matching local features in RGB frames are used to recover the poses of all RGB frames relative to an absolute, unified, coordinate system. An elastic-search database stores the 12k RGB frames, associated with a global descriptor, and local descriptors of keypoints computed by fast-R2D2 [63].

The navigating agent can then query the visual localization system by sending an image captured by its own RGB sensor and its last pose estimate, which locally combines the results of the last visual localization query and odometry. First, as described in [34], the global descriptor for the image is used to quickly retrieve a set of nearest neighbors from the database. Then local R2D2 key-points in the image are matched against the ones of the retrieved neighbors and from this relative poses estimates and the absolute camera poses of the neighbors, a consensus is established for the absolute pose of the camera of the agent, which is sent back to the agent, where it is fused into its own localization optimization graph (with previous loc and odometry).