

Learning to navigate efficiently and precisely in real environments

Guillaume Bono, Hervé Poirier, Leonid Antsfeld, Gianluca Monaci, Boris Chidlovskii, Christian Wolf

Naver Labs Europe, Meylan, France

{firstname}. {lastname}@naverlabs.com

Abstract

In the context of autonomous navigation of terrestrial robots, the creation of realistic models for agent dynamics and sensing is a widespread habit in the robotics literature and in commercial applications, where they are used for model based control and/or for localization and mapping. The more recent Embodied AI literature, on the other hand, focuses on modular or end-to-end agents trained in simulators like Habitat or AI-Thor, where the emphasis is put on photorealistic rendering and scene diversity, but high-fidelity robot motion is assigned a less privileged role. The resulting sim2real gap significantly impacts transfer of the trained models to real robotic platforms. In this work we explore end-to-end training of agents in simulation in settings which minimize the sim2real gap both, in sensing and in actuation. Our agent directly predicts (discretized) velocity commands, which are maintained through closed-loop control in the real robot. The behavior of the real robot (including the underlying low-level controller) is identified and simulated in a modified Habitat simulator. Noise models for odometry and localization further contribute in lowering the sim2real gap. We evaluate on real navigation scenarios, explore different localization and point goal calculation methods and report significant gains in performance and robustness compared to prior work.

1. Introduction

Point goal navigation of terrestrial robots in indoor buildings has traditionally been addressed in the robotics community with mapping and planning [9, 38, 59], which led to solutions capable of operating on robots in real environments. The field of computer vision and embodied AI has addressed this problem through large-scale machine learning in simulated 3D environments from reward with RL [31, 43] or with imitation learning [20]. Learning from large-scale data allows the agent to pick up more complex regularities, to process more subtle cues and therefore (in principle) to be more robust, to exploit data regularities to infer hidden and

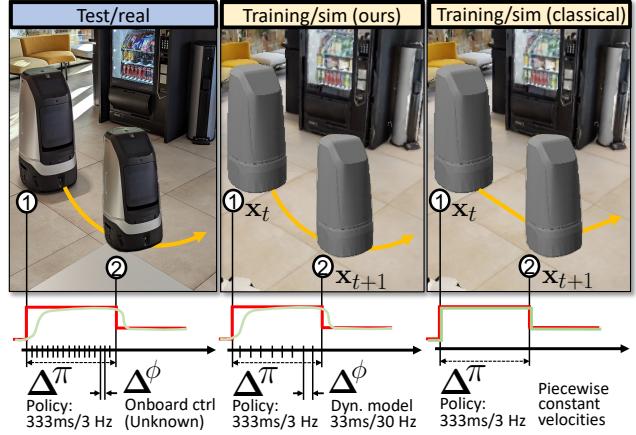


Figure 1. Efficient navigation with policies end-to-end trained in 3D photorealistic simulators requires closing the sim2real gap in sensing and actuation. Efficiency demands that the robot continues to move during decision taking (as opposed to stopping for each sensing operation), and this requires a realistic motion model in simulation allowing the agent to internally anticipate its future state. This requirement is exacerbated by the delay between sensing ① and actuation ② caused by the computational complexity of high-capacity deep networks (visual encoders, policy). To model realistic motion while training in simulation, we create a 2nd order dynamical model running with higher frequency, which models the robot and its low-level closed-loop controller. We identify the model from real data and add it to the Habitat [53] Simulator.

occluded information, and generally to learn more powerful decision rules. In this context, the specific task of point goal navigation (navigation to coordinates) is now sometimes considered “solved” in the literature [47], incorrectly, as we argue. While the machine learning and computer vision community turns its attention towards the exciting goal of integrating language models into navigation [21, 30], we think that further improvements are required to make trained agents perform reliably in real environments with sufficient speed.

Experiments and evaluations of trained models in real environments and the impact of the sim2real gap do exist in

the Embodied AI literature [15, 26, 52], but they are rare and were performed in restricted environments. Known models are trained in 3D simulators like Habitat [53] or AI-Thor [34], which are realistic in their perception aspects, but lack the accurate motion models of physical agents which the robotics community uses for articulated robots, for instance, where forward and inverse kinematics are often modeled and identified. This results in a large sim2real gap in motion characteristics, which is either ignored or addressed by discrete motion commands. The latter strategy teleports the agent in simulation for each discrete action and executes these actions on the physical robot by low-level control on positions instead of velocities, stopping the robot between agent decisions and leading to extremely inefficient navigation and overly long episodes [52].

In this work we go beyond current models for terrestrial navigation by realistically modeling both actuation and sensing of real agents in simulation, and we successfully transfer trained models to a real robotic platform. On the actuation side, we take agent decisions as discretized linear and angular velocities and predict them asynchronously while the agent is in motion. The resulting delay between sensing and actuation (see Figure 1) requires the agent to anticipate motion, which it learns in simulation from a realistic motion model. To this end, we create a second-order dynamical model, which approximates the physical properties of the real robot together with its closed-loop low-level controller. We identify this model from real robot rollouts and expert actions. The model is integrated into the Habitat simulator as an intermediate solution between the standard instantaneous motion model and a full-blown computationally-expensive physics simulation based on forces, masses, frictions etc.

On the sensing side, we combine visual and Lidar observations with different ways of communicating the navigation (point) goal to the agent, and different estimates of robot pose relative to a reference frame, combining two different types of localization: first, dead reckoning from wheel encoders, and secondly, external localization based on Monte Carlo methods from Lidar input. We perform extensive experiments and ablations on a real navigation platform and quantify the impact of the motion model, action spaces, sensing capabilities and the way we provide goal coordinates on navigation performance, showing that end-to-end trained agents can robustly navigate from visual input when their motion has been correctly modelled in simulation.

2. Related Work

Visual navigation — navigation has been classically solved in robotics using mapping and planning [10, 39, 41], which requires solutions for mapping and localization [9, 36, 59], for planning [35, 55] and for low-level control [25, 51]. These methods depend on accurate sensor models, filtering, dynamical models and optimization. End-to-end trained

models directly map input to actions and are typically trained with RL [31, 44] or imitation learning [20]. They learn representations such as flat recurrent states, occupancy maps [12], semantic maps [11, 26], latent metric maps [4, 28, 46], topological maps [3, 14, 56], self-attention [16, 22, 24, 50], implicit representations [42] or maximizing navigability directly with a blind agent optimizing a collected representation [8]. Our proposed method is end-to-end trained and features recurrent memory, but benefits from an additional motion model in simulation. The perception modules (visual encoders) required in visual navigation have traditionally increased in capacity over time, starting with small-capacity CNNs with a couple of layers, moving to ResNet variants (eg. in [63]), then to ViTs (eg. in [40, 64]) and to binocular ViTs for the case of ImageGoal (eg. in [7]).

Sim2real transfer — The sim2real gap can compromise strong performance achieved in simulation when agents are tested in the real-world [29, 37], as perception and control policies often do not generalize well to real robots due to inaccuracies in modelling, simplifications and biases. To close the gap, domain randomization methods [48, 58] treat the discrepancy between the domains as variability in the simulation parameters. Alternatively, domain adaptation methods learn an invariant mapping for matching distributions between the simulator and the robot environment. Examples include transfer of visuo-motor policies by adversarial learning [68], adapting dynamics in RL [23], lowering fidelity simulation [61] and adapting object recognition to new domains [69]. Bi-directional adaptation is proposed in [60]; Recently, Chattopadhyay et al. [15] benchmarked the robustness of embodied agents to visual and dynamics corruptions. Kadian et al. [32] investigated sim2real predictability of Habitat-Sim [53] for PointGoal navigation and proposed a new metric to quantify it, called Sim-vs-Real Correlation Coefficient (SRCC). The PointGoal task on real robots is also evaluated in [52]. To reach competitive performance without modelling any sim2real transfer, the agent is pre-trained on a wide variety of environments and then fine-tuned on a simulated version of the target environment.

Hybrid methods — Modular and hybrid approaches decompose planning into different parts, frequently in a hierarchical way. Typically, waypoints are proposed by a high-level (HL) planner, and then followed by a low-level (LL) planner, as in [2, 6, 12, 13, 26, 52]. Hybrid methods combine classical planning with learned planning. Some of the modular approaches mentioned above can be considered to be hybrid, but there also exist hybrid approaches in the literature which combine different planners more tightly and in a less modular way [5, 18, 27, 62, 67]. In some of these cases, training back-propagates through the planning stage. Some methods combine the advantages of classical and learned planning by switching between them [19, 33]. These methods also aim to decrease the sim2real gap and are complementary to ours.

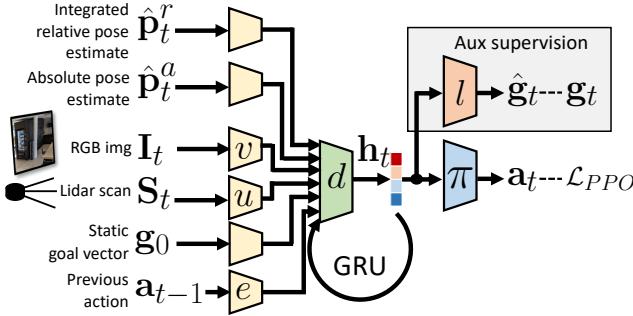


Figure 2. **The agent** uses a recurrent policy with a static point goal g_0 as input, i.e. the goal is constant and given wrt. to the initial reference frame. During training, the estimation of the dynamic point goal \hat{g}_t is supervised from privileged information.

Motion models — are usually kept very simple when training visual navigation policies. Several works use a discrete high-level action space in simulation [11, 12, 26] and rely on dedicated controllers to translate these discrete actions into robot controls when transferring to the real world. Alternative approaches predict continuous actions [17, 66] or waypoints [2, 57], that again are subsequently mapped to robot controls when executing the policies in real. Somewhat closer to our approach, Yokoyama et al. [65] adopt a discretized linear and angular velocity space, and introduce a new metric, SCT, that uses a simplified “unicycle-cart” dynamical model to estimate the “optimal” episode completion time computed by rapidly exploring random trees (RRT). However the unicycle model is only used for evaluation and it is not deployed for training nor testing: as in previous works, the commands are mapped to robot controls using handcrafted heuristics. In contrast to previous approaches, we include the dynamical model of the robot in the training loop to learn complex dependencies between the navigation policy and the robot physical properties.

3. End-to-end training with realistic sensing

We propose a method for training agents which (in principle) could address a variety of visual navigation problems, and without loss of generality, focusing on sim2real transfer and efficiency, we formalize the problem as a *PointGoal* task: an agent receives sensor observations at each time step t and must take actions \mathbf{a}_t to reach a goal position given as polar coordinates. The method is not restricted to any specific observations, in our experiments the agent receives RGB images $\mathbf{I}_t \in \mathbb{R}^{3 \times H \times W}$ and a Lidar-like vector of ranges $\mathbf{S}_t \in \mathbb{R}^K$, where K is the number of laser rays.

The proposed agent is fully trained and therefore does not require to localize itself on a map for analytical path planning. However, even in this very permissive setting the application itself requires to communicate the goal to the agent in some form of coordinate system, which requires localization at

least initially with respect to the goal (in *ImageGoal* settings this would be replaced by the goal image). We can classically differentiate between three different forms:

Absolute PointGoal — the goal vector $\mathbf{g}^a = [\mathbf{g}_x^a, \mathbf{g}_y^a]^T$ is given in a pre-defined agent-independent absolute reference frame. This requires access to a global localization system.

Dynamic PointGoal — the goal vector \mathbf{g}_t is provided with respect to the *current* agent pose and therefore updated at each time instant t . This requires an external localization system providing estimates which are then transformed into the egocentric reference frame.

Static PointGoal — the goal vector \mathbf{g}_t is provided with respect to the starting position at $t=0$ of the robot, and not updated with agent motion ($=\mathbf{g}_0$). This requires the agent to learn this update itself, and therefore some form of internal localization or integration of odometry information.

We target the *Static PointGoal* case, which does not require localization beyond the initial relative vector. However, we also argue that some form of localization and/or odometry information is useful, which will assist the dynamics information inferred from (visual) sensor readings. Furthermore, during training it will allow the agent to learn a useful latent representation in its hidden state and an associated dynamics in this latent space. We therefore provide the agent with two different localization components:

$\hat{\mathbf{p}}_t^a$ — an estimate of agent pose with respect to the initial agent position, asynchronously delivered and with low frequency, around 0.5-1 fps. At test, this signal is delivered by external localization, in our case from a Lidar-based AMCL module (see Section 5).

$\hat{\mathbf{p}}_t^r$ — an estimate coming from onboard odometry, which we integrate over time to provide an estimate situated in the same reference frame as $\hat{\mathbf{p}}_t^a$, i.e. with respect to the initial agent position.

The characteristics of the two localization signals are different: while $\hat{\mathbf{p}}_t^r$ corresponds to a *dead reckoning* process and is subject to drift, its relative noise is lower. The signal $\hat{\mathbf{p}}_t^a$ is not subject to drift but suffers from larger errors in certain regions, due to registration errors or absence of structure.

The agent policy is recurrent, based on a hidden memory vector \mathbf{h}_t updated over time from input of all sensor readings $\mathbf{I}_t, \mathbf{S}_t, \mathbf{p}_t^r, \hat{\mathbf{p}}_t^a$, the goal \mathbf{g}_0 and the previous action \mathbf{a}_{t-1} ,

$$\mathbf{h}_t = d(\mathbf{h}_{t-1}, v(\mathbf{I}_t), u(\mathbf{S}_t), \mathbf{g}_0, \hat{\mathbf{p}}_t^r, \hat{\mathbf{p}}_t^a, e(\mathbf{a}_{t-1})), \quad (1)$$

$$\mathbf{a}_t = \pi(\mathbf{h}_t), \quad (2)$$

$$\hat{\mathbf{g}}_t = l(\mathbf{h}_t), \quad (3)$$

where d is a GRU with hidden state \mathbf{h}_t , v is a visual encoder of the image \mathbf{I}_t (in our case, a ResNet-18); u is a 1D-CNN encoding the scan \mathbf{S}_t ; e is a set of action embedding vectors;

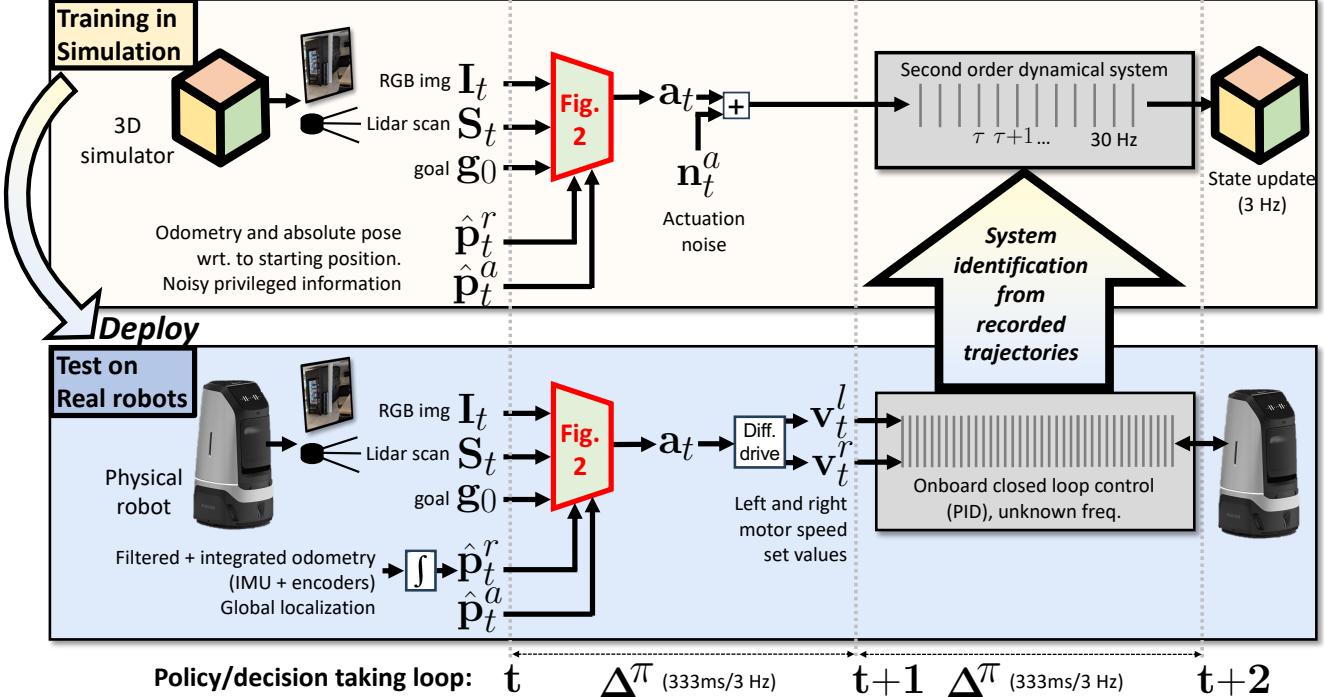


Figure 3. Training visual navigation with realistic motion models: we train an end-to-end agent in simulation (top) subject to two different simulation loops: a slower loop at 3Hz (indexed by t) renders visual observations and takes agent decisions, while a faster loop at 30 Hz (indexed by τ) simulates physics. Physics is approximated with a 2nd order model identified from real robot rollouts (bottom) and includes the robot physics as well as the behavior of the closed-loop control of the differential drive (neither onboard control algorithm nor control frequency need to be known). Operations in the intervals are pipelined, eg. sensing occurs at each time step, as does agent forward pass etc. The agent architecture is detailed in Figure 2.

π is a linear policy. Other inputs go through dedicated MLPs, not named to lighten notations. To help the agent deal with the noisy relative and absolute pose estimates, we put an inductive bias for localization on \mathbf{h}_t through an auxiliary head l predicting the dynamic goal compass $\hat{\mathbf{g}}_t$, supervised during training from the simulated agent’s ground truth pose, as shown in Figure 2. In the following paragraphs we describe the two localization systems and the noise models we use to simulate them during training in simulation. Section 4 then introduces the dynamical model used during training.

Integrated relative localization — the pose estimate $\hat{\mathbf{p}}_t^r$ can be obtained with any commercially available odometry system. In our experiments we integrate readings from wheel encoders. During simulation and training, $\hat{\mathbf{p}}_t^r$ is estimated by sampling planar noise $[\epsilon_x, \epsilon_\theta]^\top$ from a multivariate Gaussian distribution

$$\mathcal{N}\left(\begin{bmatrix} 0.01 \\ 0 \end{bmatrix}, \begin{bmatrix} 10^{-4} & 10^{-4} \\ 10^{-4} & 10^{-3} \end{bmatrix}\right),$$

which gets added to the step-wise motion of the agent, leading to the accumulation of drift over time.

Absolute localization — the pose estimate $\hat{\mathbf{p}}_t^a$ is obtained with the standard ROS 2 package for Adaptive Monte Carlo Localization (AMCL) [59], a KLD-sampling approach requiring a pre-computed, static map based on par-

ticle filtering. During simulation and training, $\hat{\mathbf{p}}_t^r$ is estimated by sampling planar noise from another Gaussian $\mathcal{N}(\mathbf{0}, \text{Diag}[0.03, 0.03, 0.05])$, but this time it gets directly added to the ground truth pose of the robot, thus modeling imprecise localization (eg. due to Lidar registration failures) without drift accumulation. In addition, we model the potential unreliability of such absolute pose estimates by only providing a new observation to the agent every few steps, at an irregular period governed by a uniform distribution $\mathcal{U}(8, 12)$. This potentially allows to plug-in low-frequency visual localization.

4. A dynamical model of realistic robot motion

One of our main motivations is to be able to use a trained model to navigate *efficiently*, i.e. fast and reliably. Stopping the robot between predicted actions is therefore not an option, and we train the model to predict linear and angular velocity commands in parallel to its physical motion, as illustrated in Figure 1. This requires the agent to anticipate motion to predict an action a_t relative to the next (yet unobserved) state at t , exacerbated by the delay between sensing at time t and actuation at $t+1$, during which computation through the high-capacity visual encoder and policy happens. This motion depends on the physical characteristics of the robot,

as shown in Figure 3 (bottom): the linear and angular velocity commands predicted by the neural policy π are translated into “set values” for left and right motor speeds of the differential drive system with a deterministic function and used as targets by the closed loop control algorithm, typically a PID (*Proportional Integral Derivative Controller*), attempting to reach and maintain the predicted speed between agent decisions. The behavior depends on the control algorithm but also on the physical properties of the robot, like its mass and resulting inertia, power and torque curves of the motors, friction etc. We decrease the actuation sim2real gap as much as possible by integrating a model of realistic robot behavior into the Habitat [53] platform, shown in the top part of Figure 3. We approximate the robot’s motion behavior by modeling the *combined* behavior of both the robot and the control algorithm implemented on the real platform as an asymmetric second-order dynamic system. The parameters of this low-level loop are estimated from recordings of the real robot platform with a system identification algorithm.

Classically, a sequential decision process can be modeled as a POMDP $\langle \mathcal{X}, \mathcal{U}, T, \mathcal{O}, O, R \rangle$. In pure 2D navigation settings with a fixed goal in a single static scene, the environment maintains a (hidden) state $\mathbf{x}_t \in \mathcal{X}$ consisting in the pose of the agent, i.e. position and orientation, $\mathbf{x}_t = [x_t, y_t, \theta_t]$. In most settings in the literature based on 3D photo-realistic simulators like Habitat [53], the state update in simulation ignores physical properties of the robot (mass/inertia, friction, accelerations, etc.), as it is implemented through “teleportation” of the robot to a new pose. In that case, the environment transition decomposes into a discrete state update $x_{t+1} = T(x_t, u_t)$ and an observation $o_{t+1} = O(x_{t+1})$. The discrete action space $\mathcal{U}_{\text{motion}} = \{\text{FORWARD } 25\text{cm}, \text{TURN_LEFT } 10^\circ, \text{TURN_RIGHT } 10^\circ \text{ and STOP}\}$ frequently chosen in the navigation literature results in a halting and jerky robot motion.

Adding realism — To be able to smooth robot trajectories, we need to model accelerations. We extend the agent state as:

$$\mathbf{x}_t = [x_t, y_t, \theta_t, v_t, w_t, \dot{v}_t, \dot{w}_t], \quad (4)$$

where x_t, y_t, θ_t are the absolute position and orientation of the robot in the plane of the scene (in m, m and rad); v_t, w_t are the linear (forward) and angular velocities of the robot (in m/s and rad/s); \dot{v}_t, \dot{w}_t are the linear and angular accelerations of the robot (in m/s² and rad/s²).

The proposed agent predicts a pair $\langle v_t^*, w_t^* \rangle$ of velocity commands, chosen from a dis-

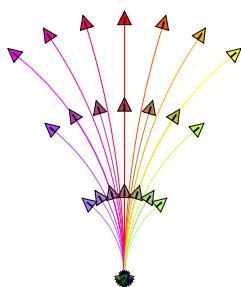


Figure 4. The action space: 28 actions, 4 choices of linear velocities $\in [0, 1]$ m/s, 7 choices for angular vel. $\in [-3, 3]$ rad/s. Arrows show the effect on pose of action held for $\frac{2}{3}$ sec.

crete action space \mathcal{U}_{vel} resulting from the Cartesian product of linear and angular spaces $\{0, 0.3, 0.6, 1\} \times \{-3, -2, -1, 0, \dots, 3\}$, as shown in Figure 4. To simplify notations, we will use $\mathbf{u}_t = \langle v_t^*, w_t^* \rangle \in \mathcal{U}$.

One possibility to model realistic robot behavior is to design a full dynamical model of the robot in the form of a set of differential equations, identify its parameters, discretize it, and simulate it in Habitat, together with running an exact copy of the control algorithm used on the physical robot with exactly the same parameters and control frequency, which is a stringent constraint. Instead, we opted for a more flexible solution and we approximate the combined behavior of the physical robot and its control algorithm by a second order dynamical model [45]. We decompose interactions with the simulator into two different loops running at different frequencies:

- Visual observations, simulator state updates and agent decisions (Equations (1) to (3)) are produced in a slower loop, indexed by letter t in the subsequent notation, and run at $f^* = 3\text{Hz}$ in our experiments.
- Between two subsequent steps of the slower loop, a faster loop models physical motion of the robot, without rendering observations. At the end of this loop, the simulator state is updated, and a visual observation is returned. Steps in this faster loop are indexed by a second index τ in the subsequent notation, $\mathbf{x}_{t,\tau}$.

In-between two environment steps $t-1$ and t , we simulate dynamics at frequency $f^\phi = 30 > f^*$. The number of physical sub-time steps per environment time step is therefore $K^\phi = \lceil \frac{f^\phi}{f^*} \rceil = 10$, their duration is $\Delta^\phi = \frac{1}{f^\phi} = 33\text{ms}$. The physical loop running between two environment steps can be formalized as the following set of state update equations,

$$\begin{aligned}
 \mathbf{o}_t &= O(\mathbf{x}_{t-1}). && \text{Observation} \\
 \mathbf{x}_{t-1,0} &= \mathbf{x}_{t-1} && \text{Init} \\
 \dot{v}_{t,\tau+1} &= \dot{v}_{t,\tau} + \Delta^\phi(f_{t,\tau}^v \delta_{t,\tau}^v - 2\zeta_{t,\tau}^v f_{t,\tau}^v \dot{v}_{t,\tau}) && \text{Upd. acc.} \\
 \dot{w}_{t,\tau+1} &= \dot{w}_{t,\tau} + \Delta^\phi(f_{t,\tau}^w \delta_{t,\tau}^w - 2\zeta_{t,\tau}^w f_{t,\tau}^w \dot{w}_{t,\tau}) && \text{Upd. vel.} \\
 v_{t,\tau+1} &= v_{t,\tau} + \Delta^\phi v_{t,\tau+1} \\
 w_{t,\tau+1} &= w_{t,\tau} + \Delta^\phi w_{t,\tau+1} \\
 \theta_{t,\tau+1} &= \theta_{t,\tau} + \Delta^\phi w_{t,\tau+1} && \text{Upd. pose} \\
 x_{t,\tau+1} &= x_{t,\tau} + \Delta^\phi v_{t,\tau+1} \cos \theta_{t,\tau+1} \\
 y_{t,\tau+1} &= y_{t,\tau} + \Delta^\phi v_{t,\tau+1} \sin \theta_{t,\tau+1} \\
 \mathbf{x}_t &= \mathbf{x}_{t-1,K} && \text{Final state}
 \end{aligned} \tag{5}$$

In these equations, f^\cdot are natural frequencies and damping coefficients of asymmetric 2nd order dynamic models for linear and angular motion. At each step τ , they are chosen from identified values given command errors as follows:

$$\begin{aligned}
 \delta_{t,\tau}^v &= v_t^* - v_{t,\tau} \\
 \langle f_{t,\tau}^v, \zeta_{t,\tau}^v \rangle &= \begin{cases} \langle f^{v\uparrow}, \zeta^{v\uparrow} \rangle & \text{if } \delta_{t,\tau}^v \cdot v_{t,\tau} > 0 \text{ (acceleration)} \\ \langle f^{v\downarrow}, \zeta^{v\downarrow} \rangle & \text{otherwise (deceleration)} \end{cases}
 \end{aligned} \tag{6}$$



Figure 5. **Robustness** — Left: the end-to-end trained agent is surprisingly robust and allows navigation very close to finely structured obstacles. Right: Navigation based on single ray Lidar and planning on occupancy maps, widely used in ROS based solutions, is difficult and error prone in situations where obstacles are thin at the height of the Lidar ray, are undetected and lead to collisions.

and similarly for angular model parameters $\langle f^w, \zeta^w \rangle$. Velocity and acceleration are also clipped to identified maximum values.

System identification — the model has 8 parameters, $f^{v\uparrow}, \zeta^{v\uparrow}, f^{v\downarrow}, \zeta^{v\downarrow}, f^{w\uparrow}, \zeta^{w\uparrow}, f^{w\downarrow}, \zeta^{w\downarrow}$, which we identify from trajectories of a real robot controlled with pre-computed trajectories exploring the command space, see the appendix for more details.

Training — the model is trained with RL, in particular PPO [54], and with a reward inspired by [15], $r_t = R \cdot \mathbb{I}_{\text{success}} - \Delta_t^{\text{Geo}} - \lambda - C \cdot \mathbb{I}_{\text{collision}}$, where $R=2.5$, Δ_t^{Geo} is the gain in geodesic distance to the goal, a slack cost $\lambda=0.01$ encourages efficiency, and a cost $C = 0.1$ penalizes each collision without ending the episode.

Recovery behavior — similar to what is done in classical analytical planning, we added a rudimentary recovery behavior on top of the trained agent: if the agent is notified of an obstacle and does not move for five seconds, it will move backward at 0.2 m/s for two seconds.

5. Experimental Results

Experimental setup — we trained the agent in the Habitat simulator [53] on the 800 scenes of the HM3D Dataset [49] for 200M env. steps. Real robot experiments have been performed on a *Naver Rookie* robot, which came equipped with various sensors. We added an additional front facing RGB camera and capture images with a resolution of 1280×720 resized to 256×144 . In our experiments, the Lidar scan S is not taken from an onboard Lidar, but simulated from the 4 *RealSense* depth cameras which are installed close to the floor and oriented in 4 different directions. We use multiple scan lines of the cameras and fuse them into a single ray, details are given in the appendix. We did, however, add a single plane Lidar to the robot (which did not come equipped with one) and used it for localization only (see section 3). All processing has been done onboard, thanks to a *Nvidia Jetson AGX Orin* we added, with an integrated *Nvidia Ampere* GPU. It handles pre-processing of visual observations and network forward pass in around 70ms. The exact network

architecture of the policy is given in the appendix.

Evaluation — evaluating sim2real transfer is inherently difficult, as it would optimally require to evaluate all agent variants and ablations on a real physical robot and on a high number of episodes. We opted for a three-way strategy, all tables are color-coded with numbers corresponding to one of the three following settings: **(i) “Real”** experiments evaluate the agent trained in simulation on the real Naver Rookie robot. It is the only form of evaluation which correctly estimates navigation performance in a real world scenario, but for practical reasons we limit it to a restricted number of 20 episodes in a large office environment shown in Fig. 5. **(ii) “Simulation (+dyn. model)”** is a setting in simulation (Habitat), which allows large-scale evaluation on a large number of unseen environments and episodes, the HM3D validation set. Most importantly, during evaluation the simulator uses the identified dynamical model and therefore realistic motion, even for baselines which do not have access to one during training. Similar to the “Real” setting, this allows to evaluate the impact of not modeling realistic motion during training. **(iii) “Simulation (train domain)”** evaluates in simulation with the same action space an agent variant uses during training, i.e. there is no sim2real gap at all. This setting evaluates the difficulty of the simulated task, which might be a severe approximation of the task in real conditions. High performance in this setting is not necessarily indicative of high performance in a real environment.

We evaluate on two different sets of episodes: **HM3D/2.5k** consists of 2500 episodes in the HM3D validation scenes, used in simulation only. **Office/20** consists of 20 episodes in the targeted office building, Figure 6. They are used for evaluation in both, real world and simulation.

Metrics — Navigation performance is evaluated by success rate (SR), i.e., fraction of episodes terminated within a distance of $<0.2\text{m}$ to the goal by the agent calling the $\langle v=0, w=0 \rangle$ action enough time to cancel its velocity, and SPL [1], i.e., SR weighted by the optimality of the path, $SPL = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\text{success}} \frac{\ell_i^*}{\max(\ell_i, \ell_i^*)}$, where ℓ_i is the agent path length and ℓ_i^* the GT path length.

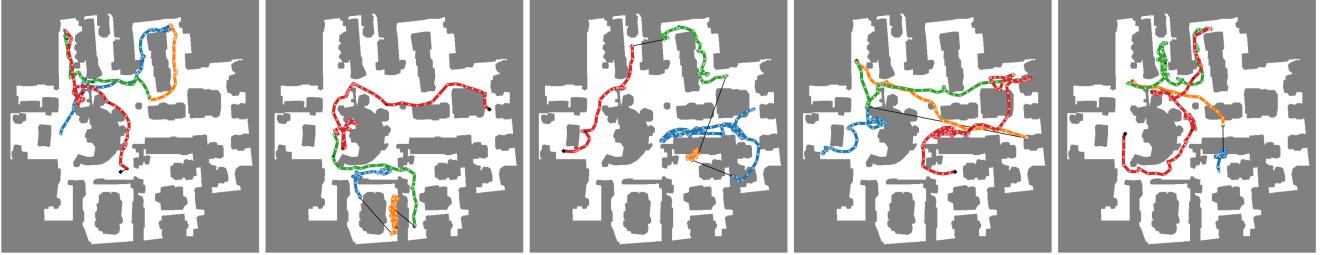


Figure 6. The 20 trajectories of `Office/20` are distributed over 5 plots and superimposed on the map, color coded. If an episode fails, the remaining distance is shown as a straight black line from the last position to the goal. The agent is variant β of Table 4.

Method & action space	+dyn (train)	Ref.	Sim (+dyn.) (HM3D/2.5k)			Sim (+dyn.) (Office/20)			Real (Office/20)		
			SR(%)	SPL(%)	SCT(%)	SR(%)	SPL(%)	SCT(%)	SR(%)	SPL(%)	SCT(%)
(a) Position, disc(4)	\times	[52]	58.2	48.2	16.8	35.0	30.3	13.2	15.0	11.8	2.4
(b) Velocity, disc(28)	\times	[65]	0.8	0.1	0.1	0.0	0.0	0.0	20.0	8.7	2.6
(c) Velocity, disc(28)	\checkmark	(Ours)	97.4	82.2	51.0	100.0	78.8	59.9	40.0	28.9	10.1

Table 1. Impact of training with a realistic dynamical model: we compare end-to-end trained models using position commands (a) as in [52] and, discretized velocity commands without a dynamical model (considering constant velocity) as in [65], and our proposed method (c). The references [52, 65] are cited for their action space and motion handling, but they have different agent architectures.

Method	Sim(train) (HM3D/2.5k)			Sim(+dyn) (HM3D/2.5k)		
	SR%	SPL%	SCT%	SR%	SPL%	SCT%
(a) Pos, disc(4)	92.7	81.7	30.3	58.2	48.2	16.8
(b) Vel, disc(28)	98.0	74.2	58.9	0.8	0.1	0.1
(c) Vel, disc(28)	97.4	82.2	51.0	97.4	82.2	51.0

Table 2. Difficulty of tasks given action spaces and motion models: we evaluate the baseline model variants in the same simulated environment in which they have been trained. High performance does not necessarily transfer to real. Letters are variants in Table 1.

SPL is limited in its ability to properly evaluate agents with complex dynamics. *Success Weighted by Completion Time* (SCT) [65] explicitly takes the agent’s dynamics model into consideration, and aims to accurately capture how well the agent has approximated the fastest navigation behavior. It is defined as $SCT = \frac{1}{N} \sum_{i=1}^N S_i \frac{t_i^*}{\max(c_i, t_i^*)}$, where c_i is the agent’s completion time in episode i , and t_i^* is the shortest possible amount of time it takes to reach the goal point from the start point while circumventing obstacles based on the agent’s dynamics. To simplify implementation, we use a lower bound on t^* taking into account linear dynamics along straight shortest path segments.

Checkpoints have been chosen as follows: performance in **Real** is given with checkpoints selected as the ones providing max SR in **simulation**. Performance in **Simulation** is given on the last checkpoint.

5.1. Results

Agent behavior — the agent is surprisingly robust and does not collide with the infrastructure even though it can quite closely approach it navigating around it, even if the obstacles

are thin and light. Examples are given in Figure 5 (left). This is in stark contrast to the ROS 2 based planner, which uses a 2D Map constructed with the onboard Lidar: thin and light structures do not show up on the map or a filtered, often because the larger part of the obstacle is not in the height of the single Lidar ray. Collisions are frequent, examples are given in Figure 5 (right).

Sim2real gap and memory — one interesting finding we would like to share is that the raw base agent sometimes tends to get discouraged from initially being blocked in a situation, for instance if the passage through a door is not optimal and requires correction. The initial variants of the agent seemed to abandon relatively quickly and started searching for a different trajectory, circumventing the passage way obstacle altogether. We conjecture that this stems from the fact that such blockings are rarely seen in simulation and the agent is trained to “write off” this area quickly, storing in its recurrent memory that this path is blocked. All our real experiments are therefore performed with a version, which resets its recurrent state h_i (eq. (1)) every 10 seconds, leading to less frequent abandoning. Future work will address this problem more directly, for instance by simulating blocking situations during training.

Influence of motion model — Table 1 compares results of different agents trained with different action spaces and with or without realistic motion during simulation. We can see that the impact of training the agent with the correct motion model in simulation is tremendous. The agent in line (b) uses the same action space, but no dynamical model is used in simulation, which means that changes in velocity are instantaneous and velocities are constant between decisions. The behavior is unexploitable, the agent is disoriented and

Point Goal	$\hat{\mathbf{p}}_t^r$	$\hat{\mathbf{p}}_t^a$	Sim (+dyn)			Real (Office/20)		
			SR%	SPL%	SCT%	SR%	SPL%	SCT%
g_t	\times	\times	40.0	34.5	27.4	35.0	23.6	5.2
g_0	\checkmark	\checkmark	70.0	51.9	36.9	50.0	37.5	11.4
$g_0 + \text{superv.}$	\checkmark	\checkmark	100.0	78.8	59.9	40.0	28.9	10.1
$g_0 + \text{superv.}$	\checkmark	\times	70.0	51.9	36.9	25.0	16.7	3.0

Table 3. **Localization and PointGoal calculation:** we compare the impact of the presence of external localization to the agent, and the point goal sources: dynamical point goal (through external localization) with static point goal w/ and w/o supervision. All agents are variant (c) from Table 1.

keeps crashing into its environment. Line (a) corresponds to a position controlled agent with action space {FORWARD 25cm, TURN_LEFT 10°, TURN_RIGHT 10°, STOP}. After training, it is adapted to motion commands by calculating the corresponding velocity commands given the decision frequency of 3 Hz. It has somewhat acceptable (but low) performance in simulation, although it does not dispose of a motion model during training. This fact that rotating and linear motion are separated and cannot occur in the same time simplifies the problem and leads to some success in simulation, but this behavior does not transfer well to the real robot. The agent with the identified motion model achieves near perfect SR in simulation, which shows that the model can learn to anticipate motion and (internal latent) future state prediction correctly. When transferred to the real robot, performance is the best among all agents, but still leaves room much room for improvement.

Difficulty of the tasks — Table 2 compares the same three agents also in simulation when validated with the same setting they have been trained on (action space, motion model or absence of). While this comparison is not at all indicative of the performance of these agents on a real robot, it provides evidence of the difficulty of the task in simulation. Surprisingly, the performances are very close: the additional burden the motion model puts on the learning problem itself can be handled very well by the agent.

Goal vector calculation — As explained in Section 3, our base agent receives the static point goal g_0 , ie. a vector with respect to the initial reference frame at time $t=0$, which is not updated. Table 3 compares this agent with two other variants. A version where supervision is removed during training, which surprisingly shows good performance. We conjecture that the additional supervision learns integration of odometry which might not transfer well enough from simulation, indicating insufficient noise modeling in simulation. This will be addressed in future work. In another variant the dynamic point goal g_t is provided at each time step in the agent’s egocentric frame. It is calculated from an external localization source, noisy in simulation. Letting the agent itself take care of the point goal integration seems to be the better solution.

Method	Real (Office/20)			Real (Office/20-alt)		
	SR%	SPL%	SCT%	SR%	SPL%	SCT%
(α) ROS 2 (fused)	80.0	69.5	26.2	90.0	70.5	27.0
(β) Ours (finetuned)	55.0	40.4	11.2	60.0	42.0	9.7

Table 4. **Comparisons** — we compare with the ROS 2 NavStack, which has access to the 2D occupancy map beforehand, localizes itself with the single ray Lidar scan and AMCL (Monte Carlo localization), uses the fused laserscan for obstacle avoidance, followed by shortest path planning. **Metrics do no measure collisions**, which are much higher for the ROS 2 planner. To be comparable, our method is finetuned with RL on the Matterport scan of the same building. We also add an experiment on the same office building with a different furniture arrangement, (Office/20-alt).

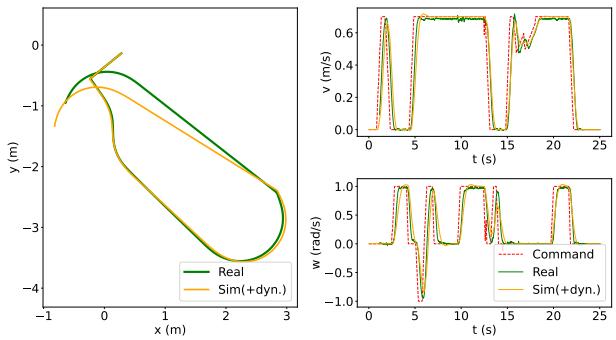


Figure 7. **Dynamical sim2real gap:** we compared recorded trajectories to simulated rollouts obtained with the same actions.

Comparison with a map based planner — Table 4 compares the method with a ROS 2 based planner, which uses a 2D occupancy map constructed beforehand (not on the fly). To be comparable, we finetuned our agent on a Matterport scan of the same building. While the ROS 2 based planner is still more efficient in terms of SR and time (SCT), it requires many experiments to finetune its parameters. For example, low values for the inflation radius will produce many collisions with tables. But when this parameter is too high, the planner cannot find any path through doors and corridors.

Rearrangement — we test the impact of rearranging furniture significantly in the Office environment (see the appendix for pictures) and show the effect in Table 4, block (Office/20-alt). The differences are neglectable.

Dynamical sim2real gap — Figure 7 compares real robot trajectories obtained by the agent on a small map of $4m \times 4m$ with rollouts of the motion model in simulation with the same actions, indicating very small drift. The dynamical models seem to approximate real robot motion very well.

Actions taken — Figure 8 shows histograms of the actions taken by the agent in simulation and in real on (Office/20). There is a clear preference for certain combinations of linear and angular velocity. Distributions in sim and real mostly match, except for a tendency to turn in-place in

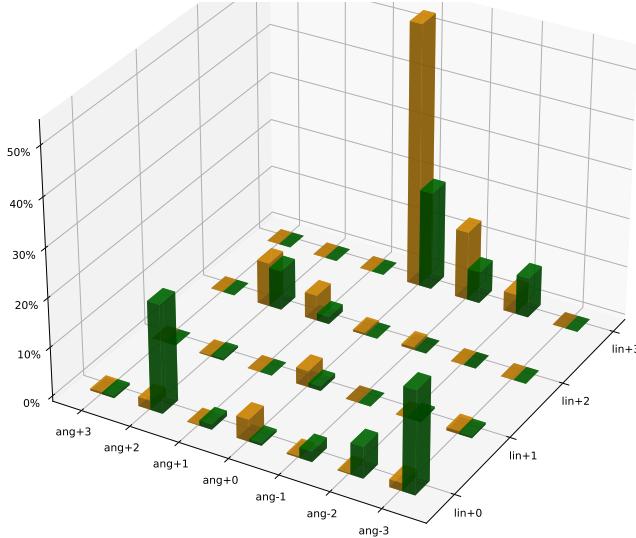


Figure 8. Distribution of actions, Office/20, — Sim, — Real.

real, which could be explained by obstacle mis-detections due to scan range noise.

6. Conclusion

We have presented an end-to-end trained method for swift and precise navigation which can robustly avoid even thin and finely structured obstacles. This is achieved with training in simulation only by adding a realistic motion model identified from recorded trajectories from a real robot. The method has been extensively evaluated in simulation as well as on a real robotic platform, where we assessed the impact of the motion model, the action space, and the way how a point goal is calculated and provided to the agent. The method is robust, future work will focus on closed-loop adaptation of dynamics and sensor noise estimation.

References

- [1] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *arXiv preprint*, 2018. [6](#)
- [2] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *CoRL*, 2020. [2, 3](#)
- [3] Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Learning to reason on uncertain topological maps. In *ECCV*, 2020. [2](#)
- [4] Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Egomap: Projective mapping and structured egocentric memory for deep RL. In *ECML-PKDD*, 2020. [2](#)
- [5] Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Learning to plan with uncertain topological maps. In *ECCV*, 2020. [2](#)
- [6] E. Beeching, M. Peter, P. Marcotte, J. Dibangoye, O. Simonin, J. Romoff, and C. Wolf. Graph augmented Deep Reinforcement Learning in the GameRLand3D environment. In *AAAI Workshop on Reinforcement Learning in Games*, 2022. [2](#)
- [7] Guillaume Bono, Leonid Antsfeld, Boris Chidlovskii, Philippe Weinzaepfel, and Christian Wolf. End-to-End (Instance)-Image Goal Navigation through Correspondence as an Emergent Phenomenon,. In *ICLR*, 2024. [2](#)
- [8] Guillaume Bono, Leonid Antsfeld, Assem Sadek, Gianluca Monaci, and Christian Wolf. Learning with a Mole: Transferable latent spatial representations for navigation without reconstruction. In *ICLR*, 2024. [2](#)
- [9] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2017. [1, 2](#)
- [10] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Aaaai/iaai*, pages 11–18, 1998. [2](#)
- [11] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, 2020. [2, 3](#)
- [12] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *ICLR*, 2020. [2, 3](#)
- [13] Devendra Singh Chaplot, Helen Jiang, Saurabh Gupta, and Abhinav Gupta. Semantic curiosity for active visual learning. In *ECCV*, 2020. [2](#)
- [14] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *CVPR*, 2020. [2](#)
- [15] Prithvijit Chattopadhyay, Judy Hoffman, Roozbeh Mottaghi, and Aniruddha Kembhavi. Robustnav: Towards benchmarking robustness in embodied navigation. *CoRR*, 2106.04531, 2021. [2, 6](#)
- [16] Shizhe Chen, Pierre-Louis Guhur, Makarand Tapaswi, Cordelia Schmid, and Ivan Laptev. Think Global, Act Local: Dual-scale Graph Transformer for Vision-and-Language Navigation. *arXiv:2202.11742*, 2022. [2](#)
- [17] Jinyoung Choi, Kyungsik Park, Minsu Kim, and Sangok Seok. Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view. In *ICRA*, 2019. [3](#)
- [18] Nitish Dashora, Daniel Shin, Dhruv Shah, Henry Leopold, David Fan, Ali Agha-Mohammadi, Nicholas Rhinehart, and Sergey Levine. Hybrid imitative planning with geometric and predictive costs in off-road environments. In *ICRA*, 2022. [2](#)
- [19] Sombit Dey, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf. Learning whom to trust in navigation: dynamically switching between classical and neural planning. In *IROS*, 2023. [2](#)
- [20] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *NeurIPS*, 2019. [1, 2](#)

- [21] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PALM-E: An Embodied Multimodal Language Model. In *ICML*, 2023. 1
- [22] Heming Du, Xin Yu, and Liang Zheng. Vtnet: Visual transformer network for object goal navigation. *arXiv preprint arXiv:2105.09447*, 2021. 2
- [23] Benjamin Eysenbach, Shreyas Chaudhari, Swapnil Asawa, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. In *ICLR*, 2021. 2
- [24] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *CVPR*, 2019. 2
- [25] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997. 2
- [26] Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. *Science Robotics*, 8(79), 2023. 2, 3
- [27] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual nav. In *CVPR*, 2017. 2
- [28] João F. Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *CVPR*, 2018. 2
- [29] Sebastian Höfer, Kostas E. Bekris, Ankur Handa, Juan Camilo Gamboa Higuera, Florian Golemo, Melissa Mozifian, Christopher G. Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White. Perspectives on sim2real transfer for robotics: A summary of the R: SS 2020 workshop, 2020. 2
- [30] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *CoRL*, 2022. 1
- [31] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017. 1, 2
- [32] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020. 2
- [33] Linh Kastner, Johannes Cox, Teham Buiyan, and Jens Lambrecht. All-in-one: A DRL-based control switch combining state-of-the-art navigation planners. In *ICRA*, 2022. 2
- [34] Eric Kolve, Rozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *CoRR*, 1712.05474, 2017. 2
- [35] Kurt Konolige. A gradient method for realtime robot control. In *IROS*, 2000. 2
- [36] Mathieu Labb  and Fran ois Michaud. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019. 2
- [37] Chen Liu, Kiran Lekkala, and Laurent Itti. World model based sim2real transfer for visual navigation. In *NeurIPS Robot Learning Workshop*, 2023. 2
- [38] Iker Lluvia, Elena Lazcano, and Ander Ansustegi. Active Mapping and Robot Exploration: A Survey. *Sensors*, 21(7):2445, 2021. 1
- [39] Steve Macenski, Francisco Mart n, Ruffin White, and Jonatan Gin s Clavero. The marathon 2: A navigation system. In *IROS*, 2020. 2
- [40] Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Yecheng Jason Ma, Claire Chen, Sneha Silwal, Aryan Jain, Vincent-Pierre Berges, Pieter Abbeel, Jitendra Malik, Dhruv Batra, Yixin Lin, Oleksandr Maksymets, Aravind Rajeswaran, and Franziska Meier. Where are we in the search for an artificial visual cortex for embodied intelligence? In *arXiv:2303.18240*, 2023. 2
- [41] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *ICRA*, 2010. 2
- [42] Pierre Marza, Laetitia Matignon, Olivier Simonin, and Christian Wolf. Multi-Object Navigation with dynamically learned neural implicit representations. In *ICCV*, 2023. 2
- [43] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017. 1
- [44] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017. 2
- [45] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 2010. 5
- [46] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018. 2
- [47] Ruslan Partsey, Erik Wijmans, Naoki Yokoyama, Oles Dobrovych, Dhruv Batra, and Oleksandr Maksymets. Is mapping necessary for realistic pointgoal navigation? In *CVPR*, 2022. 1
- [48] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *ICRA*, 2018. 2
- [49] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3D dataset (HM3D): 1000 large-scale 3d environments for embodied AI. In *NeurIPS Datasets and Benchmarks Track*, 2021. 6

- [50] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A Generalist Agent. *arXiv:2205.06175*, 2022. arXiv: 2205.06175. [2](#)
- [51] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *European Control Conference (ECC)*, 2015. [2](#)
- [52] Assem Sadek, Guillaume Bono, Boris Chidlovskii, and Christian Wolf. An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments. In *ICRA*, 2022. [2, 7](#)
- [53] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *ICCV*, 2019. [1, 2, 5, 6](#)
- [54] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017. [6](#)
- [55] James A Sethian. A fast marching level set method for monotonically advancing fronts. *PNAS*, 93(4):1591–1595, 1996. [2](#)
- [56] Dhruv Shah and Sergey Levine. ViKiNG: Vision-based kilometer-scale navigation with geographic hints. In *RSS*, 2022. [2](#)
- [57] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A foundation model for visual navigation. In *CoRL*, 2023. [3](#)
- [58] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *RSS*, 2018. [2](#)
- [59] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics, 2005. [1, 2, 4](#)
- [60] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional Domain Adaptation for Sim2Real Transfer of Embodied Navigation Agents. *IEEE Robotics and Automation Letters*, 6(2), 2021. [2](#)
- [61] Joanne Truong, Max Rudolph, Naoki Yokoyama, Sonia Chernova, Dhruv Batra, and Akshara Rai. Rethinking sim2real: Lower fidelity simulation leads to higher sim2real transfer in navigation. In *CoRL*, pages 859–870, 2022. [2](#)
- [62] Kasun Weerakoon, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Sim-to-real strategy for spatially aware robot navigation in uneven outdoor environments. In *ICRA Workshop on Releasing Robots into the Wild*, 2022. [2](#)
- [63] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2019. [2](#)
- [64] Karmesh Yadav, Arjun Majumdar, Ram Ramrakhya, Naoki Yokoyama, Alexei Baevski, Zsolt Kira, Oleksandr Maksymets, and Dhruv Batra. OVRL-V2: A simple state-of-art baseline for ImageNav and ObjectNav. In *arXiv:2303.07798*, 2023. [2](#)
- [65] Naoki Yokoyama, Sehoon Ha, and Dhruv Batra. Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation. In *IROS*, 2021. [3, 7](#)
- [66] Naoki Yokoyama, Qian Luo, Dhruv Batra, and Sehoon Ha. Learning robust agents for visual navigation in dynamic environments: The winning entry of igibson challenge 2021. In *IROS*, pages 77–83, 2022. [3](#)
- [67] Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. Path planning using neural A* search. In *ICML*, 2021. [2](#)
- [68] Fangyi Zhang, Jürgen Leitner, Zongyuan Ge, Michael Milford, and Peter Corke. Adversarial discriminative sim-to-real transfer of visuo-motor policies. *Int. J. Robotics Res.*, 38(10-11), 2019. [2](#)
- [69] Xinge Zhu, Jiangmiao Pang, Ceyuan Yang, Jianping Shi, and Dahua Lin. Adapting object detectors via selective cross-domain alignment. In *CVPR*, 2019. [2](#)

Appendix

A. System identification

We identified the parameters of the dynamical model from $N=8$ recorded trajectories of various lengths K_n on the real robot using `ros2 bag` while executing different command steps patterns:

$$\left\{ t_{n,k}, v_{n,k}^*, w_{n,k}^*, v_{n,k}, w_{n,k} \forall k \in \{1..K_n\} \right\}_{n \in \{1..N\}},$$

where $t_{n,k}$ are time-stamps, $v_{n,k}^*, w_{n,k}^*$ are velocity commands and $v_{n,k}, w_{n,k}$ are odometry measurements sampled at a frequency of $\frac{1}{t_{n,k+1}-t_{n,k}} \approx 100\text{Hz}$ (commands are interpolated using zero-order hold to match odometry timestamps).

We smooth odometry signals using a *Hanning* window of size 21 ($\sim 210\text{ms}$), then compute the first and second order derivatives $\dot{v}_{n,k}, \ddot{v}_{n,k}$ (resp. \dot{w}, \ddot{w}) by central finite difference:

$$\begin{aligned} \dot{v}_{n,k} &= \frac{v_{n,k+1} - v_{n,k-1}}{t_{n,k+1} - t_{n,k-1}}, & \forall n, \forall k \in [2, K_n - 1], \\ \ddot{v}_{n,k} &= \frac{\dot{v}_{n,k+1} - \dot{v}_{n,k-1}}{t_{n,k+1} - t_{n,k-1}}, & \forall n, \forall k \in [3, K_n - 2]. \end{aligned}$$

We compute the error $\delta_{n,k}^v = v_{n,k}^* - v_{n,k}$ and split the samples w.r.t. the sign of term $\delta_{n,k}^v v_{n,k}$ in order to identify the acceleration ($\delta^v v > 0$) and deceleration ($\delta^v v < 0$) phases of the trajectory, corresponding to parameters $f^{v\uparrow}, \zeta^{v\uparrow}$ (resp. $f^{v\downarrow}, \zeta^{v\downarrow}$) of our asymmetric second order models presented in Section 4 of the main paper.

We then solve 4 different least square problems

($\{\text{linear}, \text{angular}\} \times \{\text{accel}, \text{decel}\}$), such as:

$$\begin{bmatrix} \vdots & \vdots \\ \delta_{n,k}^v & \dot{v}_{n,k} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} f^{v\uparrow 2} \\ -2\zeta^{v\uparrow} f^{v\uparrow} \end{bmatrix} = \begin{bmatrix} \vdots \\ \ddot{v}_{n,k} \\ \vdots \end{bmatrix}_{\forall n, k : \delta_{n,k}^v v_{n,k} > 0}$$

from which we can easily deduce best-fit values of $f^{v\uparrow}$ and $\zeta^{v\uparrow}$ (resp. $f^{v\downarrow}$, $\zeta^{v\downarrow}$, $f^{w\uparrow}$, $\zeta^{w\uparrow}$, $f^{w\downarrow}$, $\zeta^{w\downarrow}$). We also use these samples to define our saturation values:

$$\begin{aligned} v_{\max} &= \max\{v_{n,k}, \forall n, \forall k\} \\ v_{\min} &= \min\{v_{n,k}, \forall n, \forall k\} \\ |\dot{v}|_{\max}^{\uparrow} &= \max\{|\dot{v}_{n,k}|, \forall n, \forall k : \delta_{n,k}^v v_{n,k} > 0\} \\ |\dot{v}|_{\max}^{\downarrow} &= \max\{|\dot{v}_{n,k}|, \forall n, \forall k : \delta_{n,k}^v v_{n,k} < 0\} \end{aligned}$$

(resp. w_{\max} , w_{\min} , $|\dot{w}|_{\max}^{\uparrow}$, $|\dot{w}|_{\max}^{\downarrow}$).

Manual adjustment — The PID included in the inner control loop of the robot which converts velocity commands to wheel speed to motor current inputs improves the response time of the entire system in a way a second order model cannot properly capture. Hence, the value obtained by the automatic identification can lead to some under-damped models, with oscillations, to keep up with the fast rise times. We manually tweaked these values to improve damping and reduce oscillations, setting all $\zeta = 0.7$, and compensating for slower rise times by increasing natural frequencies f accordingly.

B. Agent architecture

An Overview — of the agent’s network architecture is illustrated on Figure 9, providing all intermediate representation sizes. In total, the entire network has 25,589,541 trainable parameters.

The RGB encoder — is a standard ResNet18 with 64 base planes, and 4 layers of 2 basic blocks each.

The scan encoder — is a simple 1d-CNN composed of 3 (Conv, ReLU, MaxPool) blocks with respectively 64, 128, and 256 channels, kernels of sizes 7, 3 and 3, circular paddings of sizes 3, 1 and 1, and pooling windows of widths 3, 3 and 5, followed by a Linear layer projecting the final flattened channels to a vector of size 512.

Odom, ext loc, and goal encoders — are MLPs with a single hidden layer of size 1024, an output layer of size 64, and ReLU activations.

The action encoder — is a discrete set of 29 (28 actions plus one NO_ACTION token) embedding vectors of size 32.

The state encoder — is a GRU module with 2 layers, each having an internal recurrent state vector of size 1024.

C. Details on the fused simulated Lidar scan

As mentioned in the main paper, we use the Lidar of the robot only for AMCL localization. The agent itself receives a scan

which is similar to Lidar scan in its data representation, but is calculated from depth images. Four RealSense depth sensors are positioned around the robot as illustrated on Figure 10. In simulation they are rendered with a resolution of 80×60 pixels and a 90° horizontal field of view. However, three sensors on the front are rotated 90° along their optical axis (i.e. in “portrait-mode”) such that they have a larger vertical than horizontal field of view.

The (known) camera intrinsics and extrinsics are used to transform depth frames into pointclouds, where each pixel is mapped to local 3D world coordinates (relative to the robot’s base). The pointclouds are then filtered in such a way that only points with height coordinates y between 5cm and 1.2m are considered as obstacles to avoid (values outside this range are considered as navigable space where the robot can pass).

The x and z coordinates of cloudblocks are projected on the ground plane (ignoring height coordinate y), converted to polar coordinates and grouped by azimuth in 180 bins, each bin of size 2° , covering the full range from -180° to 180° . The scan ranges are obtained by taking the minimum radius in each of the 180 bins.

Optionally, a rolling buffer is used to maintain N last pointclouds and to create one fused scan. In that case, exact robot motion is used to register the old pointclouds into the new robot reference frame.

D. Furniture rearrangement

Figure 11 shows the differences between the arrangement of the office scene during the [Office/20](#) and the [Office/20-alt](#) experiments.

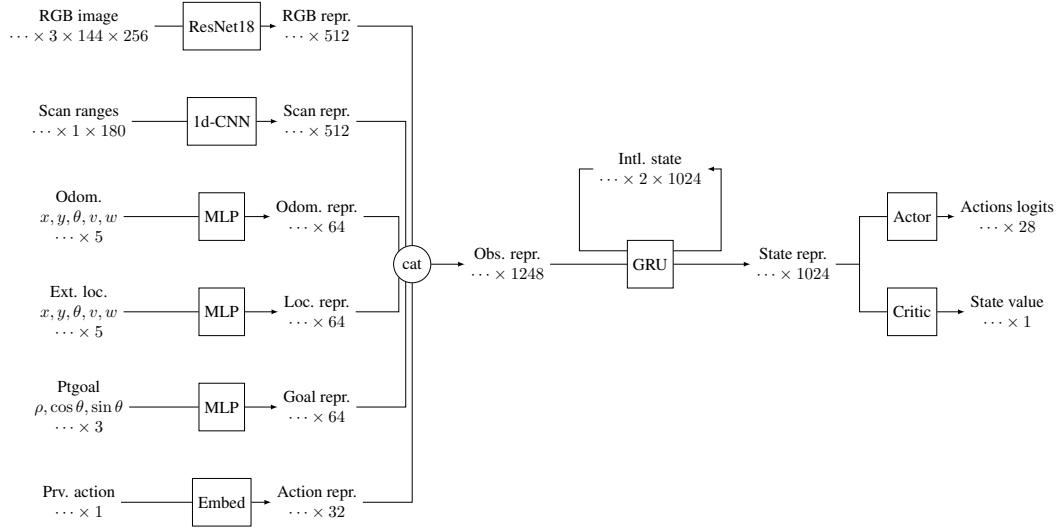


Figure 9. Agent architecture overview

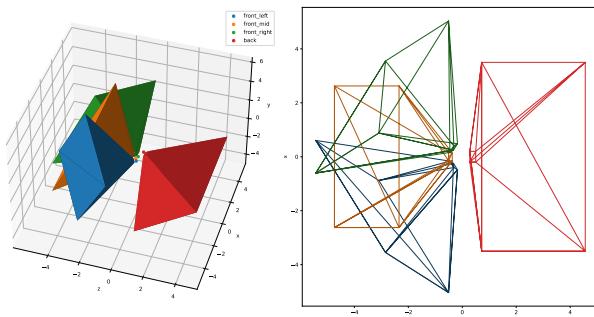


Figure 10. Configuration of depth sensors to create fused scan.



Figure 11. The scene configurations with two different tested furniture configurations. Top: Office/20. Bottom: Office/20-alt.