# Focus on Impact: Indoor Exploration with Intrinsic Motivation

Roberto Bigazzi, Federico Landi, Silvia Cascianelli, Lorenzo Baraldi, Marcella Cornia, Rita Cucchiara

***Abstract*—Exploration of indoor environments has recently experienced a significant interest, also thanks to the introduction of deep neural agents built in a hierarchical fashion and trained with Deep Reinforcement Learning (DRL) on simulated environments. Current state-of-the-art methods employ a dense extrinsic reward that requires the complete a priori knowledge of the layout of the training environment to learn an effective exploration policy. However, such information is expensive to gather in terms of time and resources. In this work, we propose to train the model with a purely intrinsic reward signal to guide exploration, which is based on the impact of the robot's actions on its internal representation of the environment. So far, impact-based rewards have been employed for simple tasks and in procedurally generated synthetic environments with countable states. Since the number of states observable by the agent in realistic indoor environments is non-countable, we include a neural-based density model and replace the traditional count-based regularization with an estimated pseudo-count of previously visited states. The proposed exploration approach outperforms DRL-based competitors relying on intrinsic rewards and surpasses the agents trained with a dense extrinsic reward computed with the environment layouts. We also show that a robot equipped with the proposed approach seamlessly adapts to point-goal navigation and real-world deployment.**

## I. Introduction

ROBOTIC exploration is the task of autonomously navigating an unknown environment with the goal of gathering sufficient information to represent it, often via a spatial map [1]. This ability is key to enable many downstream tasks such as planning [2] and goal-driven navigation [3], [4], [5]. Although a vast portion of existing literature tackles this problem [6], [7], [8], [9], it is not yet completely solved, especially in complex indoor environments. The recent introduction of large datasets of photorealistic indoor environments [10], [11] has eased the development of robust exploration strategies, which can be validated safely and quickly thanks to powerful simulating platforms [12], [3]. Moreover, exploration algorithms developed on simulated environments can be deployed in the real world with little hyperparameter tuning [13], [14], [15], if the simulation is sufficiently realistic.

Most of the recently devised exploration algorithms exploit deep reinforcement learning (DRL) [16], as learning-based exploration and navigation algorithms are more flexible and robust to noise than geometric methods [8], [17], [18]. Despite these advantages, one of the main challenges in training DRL-based exploration algorithms is designing appropriate rewards. In this work, we propose a new reward function that employs the impact of the agent actions on the environment, measured as the difference between two consecutive observations [19], discounted with a pseudo-count [20] for previously-visited states (see Fig 1). So far, impact-based rewards [19] have been used only as an additional intrinsic reward in procedurally-generated (*e.g.* grid-like mazes) or singleton (*i.e.* the test environment is the same employed for training) synthetic environments. Instead, our reward can deal with photorealistic non-singleton environments. To the best of our knowledge, this is the first work to apply impact-based rewards to this setting.

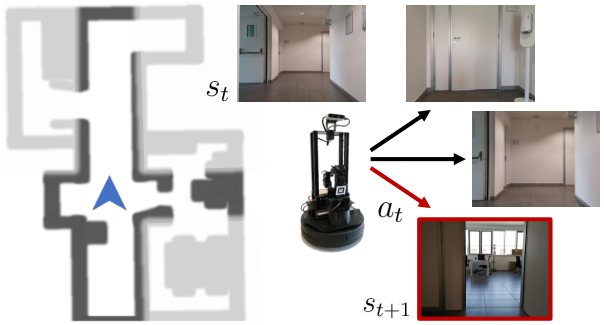


Fig. 1. We propose an impact-based reward for robot exploration of continuous indoor spaces. The robot is encouraged to take actions that maximize the difference between two consecutive observations.

Recent research on robot exploration proposes the use of an extrinsic reward based on occupancy anticipation [9]. This reward encourages the agent to navigate towards areas that can be easily mapped without errors. Unfortunately, this approach presents a major drawback, as this reward heavily depends on the mapping phase, rather than focusing on what has been already seen. In fact, moving towards new places that are difficult to map would produce a very low occupancy-based reward. Moreover, the precise layout of the training environments is not always available, especially in real-world applications. To overcome these issues, a different line of work focuses on the design of intrinsic reward functions, that can be computed by the agent by means of their current and past observations. Some examples of recently proposed intrinsic rewards for robot exploration are based on curiosity [21], novelty [18], and coverage [7]. All these rewards, however, tend to vanish with the length of the episode because the agent quickly learns to model the environment dynamics and appearance (for curiosity and novelty-based rewards) or tends

Manuscript received: September, 9, 2021; Revised December, 4, 2021; Accepted January, 3, 2022.

This paper was recommended for publication by Editor Eric Marchand upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the "European Training Network on PErsonalized Robotics as SErvice Oriented applications" (PERSEO) MSCA-ITN-2020 project (G.A. 955778).

The authors are with the Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Italy (e-mail: {name.surname}@unimore.it)

Digital Object Identifier (DOI): see top of this page.

to stay in previously-explored areas (for the coverage reward). Impact, instead, provides a stable reward signal throughout all the episode [19].

Since robot exploration takes place in complex and realistic environments that can present an infinite number of states, it is impossible to store a visitation count for every state. Furthermore, the vector of visitation counts would consist of a very sparse vector, and that would cause the agent to give the same impact score to nearly identical states. To overcome this issue, we introduce an additional module in our design to keep track of a pseudo-count for visited states. The pseudo-count is estimated by a density model trained end-to-end and together with the policy. We integrate our newly-proposed reward in a modular embodied exploration and navigation system inspired by that proposed by Chaplot *et al*. [7] and consider two commonly adopted collections of photorealistic simulated indoor environments, namely Gibson [11] and Matterport 3D (MP3D) [10]. Furthermore, we also deploy the devised algorithm in the real world. The results in both simulated and real environments are promising: we outperform state-of-the-art baselines in simulated experiments and demonstrate the effectiveness of our approach in real-world experiments. We make the source code of our approach and pre-trained models publicly available[1].

## II. Related Work

**Geometric Robot Exploration Methods.** Classical heuristic and geometric-based exploration methods rely on two main strategies: frontier-based exploration [22] and next-best-view planning [23]. These methods have been largely used and improved [24], [25], [17] or combined in a hierarchical exploration algorithm [16], [2]. However, when applied with noisy odometry and localization sensors or in highly complex environments, geometric approaches tend to fail [8], [17], [18]. In light of this, increasing research effort has been dedicated to the development of learning-based approaches, which usually exploit DLR to learn robust and efficient exploration policies.

**Intrinsic Exploration Rewards.** The lack of ground-truth in the exploration task forces the adoption of reinforcement learning (RL) for training exploration methods. Even when applied to tasks different from robot exploration, RL methods have low sample efficiency. Thus, they require designing intrinsic reward functions that encourage visiting novel states or learning the environment dynamics. The use of intrinsic motivation is beneficial when external task-specific rewards are sparse or absent. Among the intrinsic rewards that motivate the exploration of novel states, Bellemare *et al*. [20] introduced the notion of pseudo visitation count by using a Context-Tree Switching (CTS) density model to extract a pseudo-count from raw pixels and applied count-based algorithms. Similarly, Ostrovski *et al*. [26] applied the autoregressive deep generative model PixelCNN [27] to estimate the pseudo-count of the visited state. Recently, Zhang *et al*. [28] proposed a criterion to mitigate common issues in count-based methods. Rewards that encourage the learning of the environment dynamics comprehend Curiosity [29], Random Network Distillation (RND) [30], and Disagreement [31]. Recently, Raileanu *et al*. [19] proposed to jointly encourage both the visitation of novel states and the learning of the environment dynamics. However, their approach is developed for grid-like environments with a finite number of states, where the visitation count can be easily employed as a discount factor. In this work, we improve Impact, a paradigm that rewards the agent proportionally to the change in the state representation caused by its actions, and design a reward function that can deal with photorealistic scenes with non-countable states.

**Learning-based Robot Exploration Methods.** In the context of robot exploration and navigation tasks, the introduction of photorealistic simulators has represented a breeding ground for the development of self-supervised DRL-based visual exploration methods. Ramakrishnan *et al*. [18] identified four paradigms for visual exploration: novelty-based, curiosity-based (as defined above), reconstruction-based, and coverage-based. Each paradigm is characterized by a different reward function used as a self-supervision signal for optimizing the exploration policy. A coverage-based reward, considering the area seen, is also used in the modular approach to Active SLAM presented in [7], which combines a neural mapper module with a hierarchical navigation policy. To enhance exploration efficiency in complex environments, Ramakrishnan *et al*. [9] resorted to an extrinsic reward by introducing the occupancy anticipation reward, which aims to maximize the agent accuracy in predicting occluded unseen areas.

**Deep Generative Models.** Deep generative models are trained to approximate high-dimensional probability distributions by means of a large set of training samples. In recent years, literature on deep generative models followed three main approaches: latent variable models like VAE [32], implicit generative models like GANs [33], and exact likelihood models. Exact likelihood models can be classified in non-autoregressive flow-based models, like RealNVP [34] and Flow++ [35], and autoregressive models, like PixelCNN [27] and Image Transformer [36]. Non-autoregressive flow-based models consist of a sequence of invertible transformation functions to compose a complex distribution modeling the training data. Autoregressive models decompose the joint distribution of images as a product of conditional probabilities of the single pixels. Usually, each pixel is computed using as input only the previous predicted ones, following a raster scan order. In this work, we employ PixelCNN [27] to learn a probability distribution over possible states and estimate a pseudo visitation count.

## III. Proposed Method

### A. Exploration Architecture

Following the current state-of-the-art architectures for navigation for embodied agents [7], [9], the proposed method comprises three main components: a CNN-based mapper, a pose estimator, and a hierarchical navigation policy. The navigation policy defines the actions of the agent, the mapper builds a top-down map of the environment to be used for navigation, and the pose estimator locates the position of the

[1]https://github.com/aimagelab/focus-on-impact

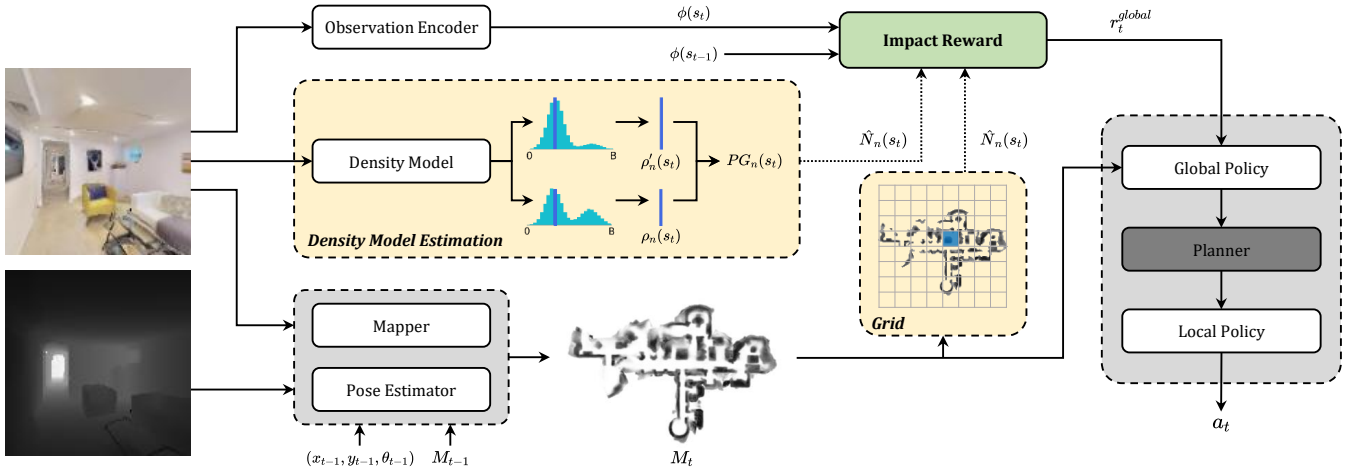


Fig. 2. Our modular exploration architecture consists of a Mapper that iteratively builds a top-down occupancy map of the environment, a Pose Estimator that predicts the pose of the robot at every step, and a hierarchical self-supervised Navigation Module in charge of sequentially setting exploration goals and predicting actions to navigate towards it. We exploit the impact-based reward to guide the exploration and adapt it for continuous environments, using an Observation Encoder to extract observation features and depending on the method, a Density Model or a Grid to compute the pseudo-count.

agent on the map. Our architecture is depicted in Fig. 2 and described below.

*1) Mapper:* The mapper generates a map of the free and occupied regions of the environment discovered during the exploration. At each time step, the RGB observation $o_t^{rgb}$ and the depth observation $o_t^d$ are processed to output a two-channel $V \times V$ local map $l_t$ depicting the area in front of the agent, where each cell describes the state of a $5 \times 5$ cm area of the environment, the channels measure the probability of a cell being occupied and being explored, as in [7]. Please note that this module performs anticipation-based mapping, as defined in [9], where the predicted local map $l_t$ includes also unseen/occluded portions of space. The local maps are aggregated and registered to the $W \times W \times 2$ global map $M_t$ of the environment using the estimated pose $(x_t, y_t, \theta_t)$ from the pose estimator. The resulting global map is used by the navigation policy for action planning.

*2) Pose Estimator:* The pose estimator is used to predict the displacement of the agent in consequence of an action. The considered atomic actions $a_t$ of the agent are: *go forward 0.25m, turn left 10°, turn right 10°*. However, the noise in the actuation system and the possible physical interactions between the agent and the environment could produce unexpected outcomes causing positioning errors. The pose estimator reduces the effect of such errors by predicting the real displacement $(\Delta x_t, \Delta y_t, \Delta\theta_t)$. According to [9], the input of this module consists of the RGB-D observations $(o_{t-1}^{rgb}, o_t^{rgb})$ and $(o_{t-1}^d, o_t^d)$ and the local maps $(l_{t-1}, l_t)$. Each modality $i = 0, 1, 2$ is encoded singularly to obtain three different estimates of the displacement:

$$g_i(e_{t-1}, e_t) = W_1 \text{max}(W_2(e_{t-1}, e_t) + b_2, 0) + b_1, \quad (1)$$

where $e_t \in \{o_t^{rgb}, o_t^d, l_t\}$ and $W_{1,2}$ and $b_2$ are weights matrices and bias. Eventually, the displacement estimates are aggregated with a weighted sum:

$$\alpha_i = \text{softmax}(\text{MLP}_i([\bar{o}_t^r, \bar{o}_t^d, \bar{l}_t])), \quad (2)$$

$$(\Delta x_t, \Delta y_t, \Delta\theta_t) = \sum_{i=0}^{2} \alpha_i \cdot g_i, \quad (3)$$

where MLP is a three-layered fully-connected network, ($\bar{o}_t^r$, $\bar{o}_t^d$, $\bar{l}_t$) are the inputs encoded by a CNN, and $[\cdot, \cdot, \cdot]$ denotes tensor concatenation. The estimated pose of the agent at time $t$ is given by:

$$(x_t, y_t, \theta_t) = (x_{t-1}, y_{t-1}, \theta_{t-1}) + (\Delta x_t, \Delta y_t, \Delta\theta_t). \quad (4)$$

Note that, at the beginning of each exploration episode, the agent sets its position to the center of its environment representation, *i.e.*

$$(x_0, y_0, \theta_0) = (0, 0, 0). \quad (5)$$

*3) Navigation Module:* The sampling of the atomic actions of the agent relies on the hierarchical navigation policy that is composed of the following modules: the global policy, the planner, and the local policy.

The global policy samples a point on an augmented global map of the environment, $M_t^+$, that represents the current global goal of the agent. The augmented global map $M_t^+$ is a $W \times W \times 4$ map obtained by stacking the two-channel global map $M_t$ from the Mapper with the one-hot representation of the agent position $(x_t, y_t)$ and the map of the visited positions, which collects the one-hot representations of all the positions assumed by the agent from the beginning of the exploration. Moreover, $M_t^+$ is in parallel cropped with respect to the position of the agent and max-pooled to a spatial dimension $H \times H$ where $H < W$. These two versions of the augmented global map are concatenated to form the $H \times H \times 8$ input of the global policy that is used to sample a goal in the global action space $H \times H$. The global policy is trained with reinforcement learning with our proposed impact-based reward $r_t^{global}$, defined below, that encourages exploration.

The planner consists of an A* algorithm. It uses the global map to plan a path towards the global goal and samples a local goal within 1.25m from the position of the agent.

The local policy outputs the atomic actions to reach the local goal and is trained to minimize the euclidean distance to the local goal, which is expressed via the following reward:

$$r_t^{local}(s_t, s_{t+1}) = d(s_{t+1}) - d(s_t), \tag{6}$$

where $d(s_t)$ is the euclidean distance to the local goal at time step $t$. Note that the output actions in our setup are discrete. These platform-agnostic actions can be translated into signals for specific robots actuators, as we do in this work. Alternatively, based on the high-level predicted commands, continuous actions can be predicted, *e.g.* in the form of linear and angular velocity commands to the robot, by using an additional, lower-level policy, as done in [37]. The implementation of such policy is beyond the scope of our work.

Following the hierarchical structure, the global goal is reset every $\eta$ steps, and the local goal is reset if at least one of the following conditions verifies: a new global goal is sampled, the agent reaches the local goal, the local goal location is discovered to be in a occupied area.

### B. Impact-Driven Exploration

The exploration ability of the agent relies on the design of an appropriate reward for the global policy. In this setting, the lack of external rewards from the environment requires the design of a dense intrinsic reward. To the best of our knowledge, our proposed method presents the first implementation of impact-driven exploration in photorealistic environments. The key idea of this concept is encouraging the agent to perform actions that have impact on the environment and the observations retrieved from it, where the impact at time step $t$ is measured as the $l_2$-norm of the encodings of two consecutive states $\phi(s_t)$ and $\phi(s_{t+1})$, considering the RGB observation $o_t^{rgb}$ as the state $s_t$. Following the formulation proposed in [19], the reward of the global policy for the proposed method is calculated as:

$$r_t^{global}(s_t, s_{t+1}) = \frac{\|\phi(s_{t+1}) - \phi(s_t)\|_2}{\sqrt{N(s_{t+1})}}, \tag{7}$$

where $N(s_t)$ is the visitation count of the state at time step $t$, *i.e.* how many times the agent has observed $s_t$. The visitation count is used to drive the agent out of regions already seen in order to avoid trajectory cycles. Note that the visitation count is episodic, *i.e.* $N_{ep}(s_t) \equiv N(s_t)$. For simplicity, in the following we denote the episodic visitation count as $N(s_t)$.

*1) Visitation Counts:* The concept of normalizing the reward using visitation count, as in [19], fails when the environment is continuous, since during exploration is unlikely to visit exactly the same state more than once. In fact, even microscopic changes in terms of translation or orientation of the agent cause shifts in the values of the RGB observation, thus resulting in new states. Therefore, using a photorealistic continuous environment nullifies the scaling property of the denominator of the global reward in Eq. 7 because every state $s_t$ during the exploration episode is, most of the times, only encountered for the first time. To overcome this limitation, we implement two types of pseudo-visitation counts $\hat{N}(s_t)$ to be used in place of $N(s_t)$, which extend the properties of visitation counts to continuous environments: *Grid* and *Density Model Estimation*.

*Grid:* With this approach, we consider a virtual discretized grid of cells with fixed size in the environment. We then assign a visitation count to each cell of the grid. Note that, different from approaches working on procedurally-generated environments like [19], the state space of the environment we consider is continuous also in this formulation, and depends on the pose of the agent $(x, y, \theta)$. The grid approach operates a quantization of the agent's positions, and that allows to cluster observation made from similar positions. To this end, we take the global map of the environment and divide it into cells of size $G \times G$. The estimated pose of the agent, regardless of its orientation $\theta_t$, is used to select the cell that the agent occupies at time $t$. In the *Grid* formulation, the visitation count of the selected cell is used as $N(s_t)$ in Eq. 7 and is formalized as:

$$\hat{N}(s_t) = \hat{N}(g(x_t, y_t)), \tag{8}$$

where $g(\cdot)$ returns the block corresponding to the estimated position of the agent.

*Density Model Estimation (DME):* Let $\rho$ be an autoregressive density model defined over the states $s \in S$, where $S$ is the set of all possible states. We call $\rho_n(s)$ the probability assigned by $\rho$ to the state $s$ after being trained on a sequence of states $s_1, ..., s_n$, and $\rho'_n(s)$, or recoding probability [20], [26], the probability assigned by $\rho$ to $s$ after being trained on $s_1, ..., s_n, s$. The prediction gain $PG$ of $\rho$ describes how much the model has improved in the prediction of $s$ after being trained on $s$ itself, and is defined as

$$PG_n(s) = \log \rho'_n(s) - \log \rho_n(s). \tag{9}$$

In this work, we employ a lightweight version of Gated PixelCNN [27] as density model. This model is trained from scratch along with the exploration policy using the states visited during the exploration, which are fed to PixelCNN one at a time, as they are encountered. The weights of PixelCNN are optimized continually over all the environments. As a consequence, the knowledge of the density model is not specific for a particular environment or episode. To compute the input of the PixelCNN model, we transform the RGB observation $o_t^r$ to grayscale and we crop and resize it to a lower size $P \times P$. The transformed observation is quantized to $B$ bins to form the final input to the model, $s_t$. The model is trained to predict the conditional probabilities of the pixels in the transformed input image, with each pixel depending only on the previous ones following a raster scan order. The output has shape $P \times P \times B$ and each of its elements represents the probability of a pixel belonging to each of the $B$ bins. The joint distribution of the input modeled by PixelCNN is:

$$p(s_t) = \prod_{1}^{P^2} p(\chi_i | \chi_1, ..., \chi_{i-1}), \tag{10}$$

where $\chi_i$ is the $i^{th}$ pixel of the image $s_t$. $\rho$ is trained to fit $p(s_t)$ by using the negative log-likelihood loss.

Let $\hat{n}$ be the pseudo-count total, *i.e.* the sum of all the visitation counts of all states during the episode. The probability and the recoding probability of $s$ can be defined as:

$$\rho_n(s) = \frac{\hat{N}_n(s)}{\hat{n}}, \qquad \rho'_n(s) = \frac{\hat{N}_n(s)+1}{\hat{n}+1}. \tag{11}$$

Note that, if $\rho$ is learning-positive, *i.e.* if $PG_n(s) > 0$ for all possible sequences $s_1, ..., s_n$ and all $s \in S$, we can approximate $\hat{N}_n(s)$ as:

$$\hat{N}_n(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s)-\rho_n(s)} \approx (e^{PG_n(s)}-1)^{-1}. \tag{12}$$

To use this approximation in Eq. 7, we still need to address three problems: it does not scale with the length of the episode, the density model could be not learning-positive, and $\hat{N}_n(s)$ should be large enough to avoid the reward becoming too large regardless the goal selection. In this respect, to take into account the length of the episode, we introduce a normalizing factor $n^{-1/2}$, where $n$ is the number of steps done by the agent since the start of the episode. Moreover, to force $\rho$ to be learning-positive, we clip $PG_n(s)$ to 0 when it becomes negative. Finally, to avoid small values at the denominator of $r_t^{global}$ (Eq. 7), we introduce a lower bound of 1 to the pseudo visitation count. The resulting definition of $\hat{N}_n(s)$ in the *Density Model Estimation* formulation is:

$$\widetilde{PG}_n = c \cdot n^{-1/2} \cdot (PG_n(s))_+, \tag{13}$$

$$\hat{N}_n(s) = \max\left\{\left(e^{\widetilde{PG}_n(s)}-1\right)^{-1}, 1\right\}, \tag{14}$$

where $c$ is a term used to scale the prediction gain. It is worth noting that, unlike the Grid approach that can be applied only when $s_t$ is representable as the robot location, the Density Model Estimation can be adapted to a wider range of tasks, including settings where the agent alters the environment.

## IV. Experimental Setup

**Datasets.** For comparison with state-of-the-art DRL-based methods for embodied exploration, we employ the photorealistic simulated 3D environments contained in the Gibson dataset [11] and the MP3D dataset [10]. Both these datasets consist of indoor environments where different exploration episodes take place. In each episode, the robot starts exploring from a different point in the environment. Environments used during training do not appear in the validation/test split of these datasets. Gibson contains 106 scans of different indoor locations, for a total of around 5M exploration episodes (14 locations are used in 994 episodes for test in the so-called Gibson Val split). MP3D consists of 90 scans of large indoor environments (11 of those are used in 495 episodes for the validation split and 18 in 1008 episodes for the test split).

**Evaluation Protocol.** We train our models on the Gibson train split. Then, we perform model selection basing on the results obtained on Gibson Val. We then employ the MP3D validation and test splits to benchmark the generalization abilities of the agents. To evaluate exploration agents, we employ the following metrics. **IoU** between the reconstructed map and the ground-truth map of the environment: here we consider two different classes for every pixel in the map (free or occupied). Similarly, the map accuracy (**Acc**, expressed in $m^2$) is the portion of the map that has been correctly mapped by the agent. The area seen (**AS**, in $m^2$) is the total area of the environment observed by the agent. For both the IoU and the area seen, we also present the results relative to the two different classes: free space and occupied space respectively (**FIoU**, **OIoU**, **FAS**, **OAS**). Finally, we report the mean positioning error achieved by the agent at the end of the episode. A larger translation error (**TE**, expressed in $m$) or angular error (**AE**, in degrees) indicates that the agent struggles to keep a correct estimate of its position throughout the episode. For all the metrics, we consider episodes of length $T = 500$ and $T = 1000$ steps.

TABLE I
Results for our model selection on Gibson Val for $T = 500$.

| Model | IoU ↑ | FIoU ↑ | OIoU ↑ | Acc ↑ | AS ↑ | FAS ↑ | OAS ↑ | TE ↓ | AE ↓ |
|---|---|---|---|---|---|---|---|---|---|
| **Grid** | | | | | | | | | |
| *G = 2* | 0.726 | 0.721 | 0.730 | 51.41 | 61.88 | 34.17 | 27.71 | 0.240 | 4.450 |
| *G = 4* | 0.796 | 0.792 | 0.801 | 54.34 | 61.17 | 33.74 | 27.42 | 0.079 | 1.055 |
| *G = 5* | **0.806** | **0.801** | **0.813** | **55.21** | **62.17** | **34.31** | **27.87** | **0.077** | **0.881** |
| *G = 10* | 0.789 | 0.784 | 0.794 | 54.26 | 61.67 | 34.06 | 27.61 | 0.111 | 1.434 |
| **DME** | | | | | | | | | |
| *B = 64* | 0.773 | 0.768 | 0.778 | 53.58 | 61.00 | 33.79 | 27.21 | 0.131 | 2.501 |
| *B = 128* | **0.796** | **0.794** | **0.799** | **54.73** | **62.07** | **34.27** | **27.79** | **0.095** | **1.184** |
| *B = 256* | 0.685 | 0.676 | 0.695 | 49.27 | 61.40 | 33.95 | 27.45 | 0.311 | 6.817 |

For our comparisons, we consider five baselines trained with different rewards. *Curiosity* employs a surprisal-based intrinsic reward as defined in [29]. *Coverage* and *Anticipation* are trained with the corresponding coverage-based and accuracy-based rewards defined in [9]. For completeness, we include two count-based baselines, obtained using the reward defined in Eq. 7, but ignoring the contribution of impact (*i.e.*setting the numerator to a constant value of 1). These are *Count (Grid)* and *Count (DME)*. All the baselines share the same overall architecture and training setup of our main models.

**Implementation Details.** The experiments are performed using the Habitat Simulator [3] with observations of the agent set to be $128 \times 128$ RGB-D images and episode length during training set to $T = 500$. Each model is trained with the training split of the Gibson dataset [11] with 40 environments in parallel for $\approx$ 5M frames.

*Navigation Module:* The reinforcement learning algorithm used to train the global and local policies is PPO [38] with Adam optimizer and a learning rate of $2.5 \times 10^{-4}$. The global goal is reset every $\eta = 25$ time steps and the global action space hyperparameter $H$ is 240. The local policy is updated every $\eta$ steps and the global policy is updated every $20\eta$ steps.

*Mapper and Pose Estimator:* These models are trained with a learning rate of $10^{-3}$ with Adam optimizer, the local map size is set with $V = 101$ while the global map size is $W = 961$ for episodes in the Gibson dataset and $W = 2001$ in the MP3D dataset. Both models are updated every $4\eta$ time steps, where $\eta$ is the reset interval of the global policy.

*Density Model:* The model used for density estimation is a lightweight version of Gated PixelCNN [27] consisting of a $7 \times 7$ masked convolution followed by two residual blocks with $1 \times 1$ masked convolutions with 16 output channels, a $1 \times 1$ masked convolutional layer with 16 output channels, and a

TABLE II
EXPLORATION RESULTS ON GIBSON VAL, MP3D VAL, AND MP3D TEST, AT DIFFERENT EPISODE MAXIMUM LENGTH.

| | Gibson Val (T = 500) | | | | | | | | | Gibson Val (T = 1000) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** |
| Curiosity | 0.678 | 0.669 | 0.688 | 49.35 | 61.67 | 34.16 | 27.51 | 0.330 | 7.430 | 0.560 | 0.539 | 0.581 | 45.71 | 67.64 | 37.19 | 30.45 | 0.682 | 14.862 |
| Coverage | 0.721 | 0.715 | 0.726 | 51.47 | 61.13 | 34.07 | 27.06 | 0.272 | 5.508 | 0.653 | 0.641 | 0.664 | 50.10 | 66.15 | 36.77 | 29.38 | 0.492 | 10.796 |
| Anticipation | 0.783 | 0.778 | 0.789 | 54.68 | 60.96 | 34.15 | 26.81 | 0.100 | 1.112 | 0.773 | 0.763 | 0.782 | 56.37 | 66.61 | 37.17 | 29.44 | 0.155 | 1.876 |
| Count (Grid) | 0.714 | 0.706 | 0.721 | 50.85 | 61.61 | 34.17 | 27.44 | 0.258 | 5.476 | 0.608 | 0.592 | 0.624 | 48.22 | 67.80 | 37.31 | 30.50 | 0.520 | 10.996 |
| Count (DME) | 0.764 | 0.757 | 0.772 | 52.81 | 60.69 | 33.68 | 27.01 | 0.148 | 2.888 | 0.708 | 0.694 | 0.722 | 52.67 | 66.91 | 36.81 | 30.12 | 0.282 | 5.802 |
| **Impact (Grid)** | **0.803** | **0.797** | **0.809** | 54.94 | 61.90 | 34.07 | 27.83 | **0.079** | **0.878** | **0.802** | **0.793** | **0.811** | **57.21** | 67.74 | 37.04 | 30.69 | **0.119** | **1.358** |
| **Impact (DME)** | 0.800 | 0.796 | 0.803 | **55.10** | **62.59** | **34.45** | **28.14** | 0.095 | 1.166 | 0.789 | 0.783 | 0.796 | 56.77 | **68.34** | **37.42** | **30.92** | 0.154 | 1.958 |

| | MP3D Val (T = 500) | | | | | | | | | MP3D Val (T = 1000) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** |
| Curiosity | 0.339 | 0.473 | 0.205 | 97.82 | 118.13 | 75.73 | 42.40 | 0.566 | 7.290 | 0.336 | 0.449 | 0.223 | 109.79 | 157.27 | 100.07 | 57.20 | 1.322 | 14.540 |
| Coverage | 0.352 | 0.494 | 0.210 | 102.05 | 120.00 | 76.78 | 43.21 | 0.504 | 5.822 | 0.362 | 0.492 | 0.232 | 116.58 | 158.83 | 100.76 | 58.07 | 1.072 | 11.624 |
| Anticipation | 0.381 | 0.530 | 0.231 | 106.02 | 114.06 | 72.94 | 41.13 | 0.151 | 1.280 | 0.420 | 0.568 | 0.272 | 126.86 | 147.33 | 93.56 | 53.78 | 0.267 | 2.436 |
| Count (Grid) | 0.347 | 0.488 | 0.206 | 99.00 | 116.77 | 75.00 | 41.76 | 0.466 | 5.828 | 0.350 | 0.474 | 0.226 | 112.75 | 157.13 | 100.03 | 57.10 | 1.074 | 11.686 |
| Count (DME) | 0.359 | 0.493 | 0.225 | 101.73 | 112.65 | 72.22 | 40.43 | 0.268 | 3.318 | 0.379 | 0.505 | 0.254 | 119.07 | 149.62 | 95.16 | 54.46 | 0.590 | 6.544 |
| **Impact (Grid)** | 0.383 | 0.531 | **0.234** | 107.41 | 116.60 | 74.44 | 42.17 | **0.120** | **0.860** | **0.440** | **0.595** | **0.285** | **133.97** | 157.19 | 99.61 | 57.58 | **0.202** | **1.294** |
| **Impact (DME)** | **0.396** | **0.560** | 0.233 | **111.61** | **124.06** | **79.47** | **44.59** | 0.232 | 1.988 | 0.427 | 0.587 | 0.268 | 133.27 | **166.20** | **105.69** | **60.50** | 0.461 | 3.654 |

| | MP3D Test (T = 500) | | | | | | | | | MP3D Test (T = 1000) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** | **IoU ↑** | **FIoU ↑** | **OIoU ↑** | **Acc ↑** | **AS ↑** | **FAS ↑** | **OAS ↑** | **TE ↓** | **AE ↓** |
| Curiosity | 0.362 | 0.372 | 0.352 | 109.66 | 130.48 | 85.98 | 44.50 | 0.620 | 7.482 | 0.361 | 0.365 | 0.357 | 130.10 | 185.36 | 121.65 | 63.71 | 1.520 | 14.992 |
| Coverage | 0.390 | 0.401 | 0.379 | 116.71 | 134.89 | 88.15 | 46.75 | 0.564 | 5.938 | 0.409 | 0.418 | 0.399 | 142.86 | 193.20 | 126.21 | 66.99 | 1.240 | 11.814 |
| Anticipation | 0.424 | 0.433 | **0.415** | 117.87 | 124.24 | 81.31 | 42.93 | 0.151 | 1.306 | 0.484 | 0.491 | 0.478 | 153.83 | 174.76 | 114.29 | 60.47 | 0.289 | 2.356 |
| Count (Grid) | 0.364 | 0.381 | 0.348 | 117.50 | 134.85 | 89.81 | 45.05 | 0.525 | 5.790 | 0.377 | 0.391 | 0.363 | 144.26 | 194.76 | 129.22 | 65.53 | 1.246 | 11.608 |
| Count (DME) | 0.391 | 0.397 | 0.385 | 114.02 | 123.86 | 81.86 | 42.00 | 0.287 | 3.322 | 0.418 | 0.419 | 0.418 | 140.21 | 172.44 | 113.25 | 59.19 | 0.657 | 6.572 |
| **Impact (Grid)** | 0.420 | 0.430 | 0.409 | 124.44 | 130.98 | 86.08 | 44.90 | **0.124** | **0.834** | **0.502** | **0.510** | **0.494** | 168.55 | 190.03 | 124.44 | 65.60 | **0.218** | **1.270** |
| **Impact (DME)** | **0.426** | **0.444** | 0.409 | **133.51** | **144.64** | **95.70** | **48.94** | 0.288 | 2.312 | 0.481 | 0.498 | 0.464 | **174.18** | **212.00** | **140.10** | **71.90** | 0.637 | 4.390 |

final $1 \times 1$ masked convolution that returns the output logits with shape $P \times P \times B$, where $B$ is the number of bins used to quantize the model input. We set $P = 42$ for the resolution of the input and the output of the density model, and $c = 0.1$ for the prediction gain scale factor.

## V. EXPERIMENTAL RESULTS

**Exploration Results.** As a first step, we perform model selection using the results on the Gibson Val split (Table I). Our agents have different hyperparameters that depend on the implementation for the pseudo-counts. When our model employs grid-based pseudo-counts, it is important to determine the dimension of a single cell in this grid-based structure. In our experiments, we test the effects of using $G \times G$ squared cells, with $G \in \{2, 4, 5, 10\}$. The best results are obtained with $G = 5$, with small differences among the various setups. When using pseudo-counts based on a density model, the most relevant hyperparameters depend on the particular model employed as density estimator. In our case, we need to determine the number of bins $B$ for PixelCNN, with $B \in \{64, 128, 256\}$. We find out that the best results are achieved with $B = 128$.

In Table II, we compare the Impact (Grid) and Impact (DME) agents with the baseline agents previously described on the considered datasets. For each model and each split, we test 5 different random seeds and report the mean result for each metric. For the sake of readability, we do not report the standard deviations for the different runs, which we quantify in around 1.2% of the mean value reported. As it can be seen, results achieved by the two proposed impact-based agents are constantly better than those obtained by the competitors, both for $T = 500$ and $T = 1000$. It is worth noting that our intrinsic impact-based reward outperforms strong extrinsic rewards that exploit information computed using the ground-truth layout of the environment. Moreover, the different implementations chosen for the pseudo-counts affect final performance, with Impact (DME) bringing the best results in terms of AS and Impact (Grid) in terms of IoU metrics. From the results it also emerges that, although the proposed implementations for the pseudo-count in Eq. 7 lead to comparable results in small environments as those contained in Gibson and MP3D Val, the advantage of using DME is more evident in large, complex environments as those in MP3D Test.

In Fig. 3, we report some qualitative results displaying the trajectories and the area seen by different agents in the same episode. Also from a qualitative point of view, the benefit given by the proposed reward in terms of exploration trajectories and explored areas is easy to identify.

**PointGoal Navigation.** One of the main advantages of training deep modular agents for embodied exploration is that they easily adapt to perform downstream tasks, such as PointGoal navigation [3]. Recent literature [7], [9] has discovered that hierarchical agents trained for exploration are competitive with state-of-the-art architecture tailored for PointGoal navigation and trained with strong supervision for 2.5 billion frames [4]. Additionally, the training time and data required to learn the

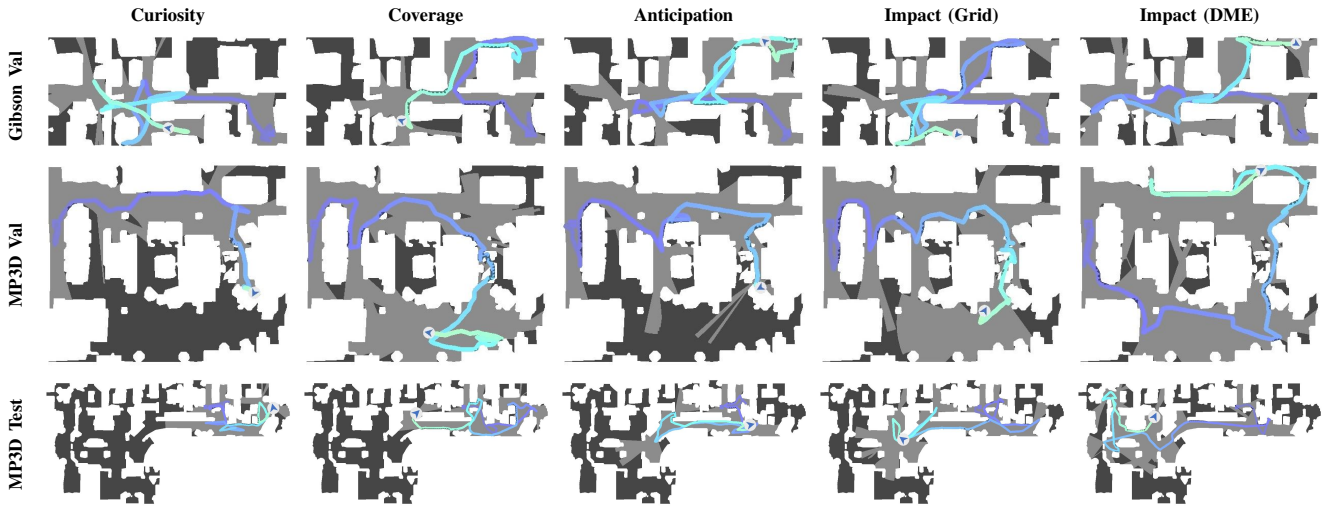


Fig. 3. Qualitative results. For each model, we report three exploration episodes on Gibson and MP3D datasets for $T = 500$.

TABLE III
POINTGOAL NAVIGATION RESULTS ON THE VALIDATION SUBSET OF THE GIBSON DATASET. UNDERLINED DENOTES SECOND BEST.

| | Noise-free Pose Sensor | | | | Noisy Pose Sensor | | | |
|---|---|---|---|---|---|---|---|---|
| **Model** | **D2G** ↓ | **SR** ↑ | **SPL** ↑ | **sSPL** ↑ | **D2G** ↓ | **SR** ↑ | **SPL** ↑ | **sSPL** ↑ |
| ANS [7] | - | 0.950 | 0.846 | - | - | - | - | - |
| OccAnt [9] | - | 0.930 | 0.800 | - | - | - | - | - |
| OccAnt [9][2] | - | - | 0.911 | - | - | - | - | - |
| Curiosity | **0.238** | **0.970** | **0.914** | **0.899** | 0.302 | 0.861 | 0.822 | 0.890 |
| Coverage | 0.240 | **0.970** | 0.909 | 0.895 | 0.288 | 0.827 | 0.788 | 0.886 |
| Anticipation | 0.285 | 0.965 | 0.906 | 0.892 | 0.309 | 0.885 | 0.835 | 0.884 |
| **Impact (Grid)** | 0.252 | 0.969 | 0.908 | 0.894 | **0.226** | **0.923** | **0.867** | **0.893** |
| **Impact (DME)** | 0.264 | 0.967 | 0.907 | 0.895 | 0.276 | 0.913 | 0.859 | **0.893** |
| *DD-PPO* [4] | - | *0.967* | *0.922* | - | - | - | - | - |

policy is much more limited (2 to 3 orders of magnitude smaller). In Table III, we report the results obtained using two different settings. The *noise-free pose sensor* setting is the standard benchmark for PointGoal navigation in Habitat [3]. In the *noisy pose sensor* setting, instead, the pose sensor readings are noisy, and thus the agent position must be estimated as the episode progresses. We consider four main metrics: the average distance to the goal achieved by the agent (**D2G**) and three success-related metrics. The success rate (**SR**) is the fraction of episodes terminated within $0.2$ meters from the goal, while the **SPL** and SoftSPL (**sSPL**) weigh the distance from the goal with the length of the path taken by the agent in order to penalize inefficient navigation. As it can be seen, the two proposed agents outperform the main competitors from the literature: Active Neural Slam (ANS) [7] and OccAnt [9] (for which we report both the results from the paper and the official code release).

When comparing with our baselines in the noise-free setting, the overall architecture design allows for high-performance results, as the reward influences map estimation only marginally. In fact, in this setting, the global policy and the pose estimation module are not used, as the global goal coincides with the episode goal coordinates, and the agent receives oracle position information. Thus, good results mainly depend on the effectiveness of the mapping module. Instead, in the noisy setting, the effectiveness of the reward used during training influences navigation performance more significantly. In this case, better numerical results originate from a better ability to estimate the precise pose of the agent during the episode. For completeness, we also compare with the results achieved by DD-PPO [4], a method trained with reinforcement learning for the PointGoal task on $2.5$ billion frames, $500$ times more than the frames used to train our agents.

**Real-world Deployment.** As agents trained in realistic indoor environments using the Habitat simulator are adaptable to real-world deployment [13], [14], we also deploy the proposed approach on a LoCoBot robot[3]. We employ the PyRobot interface [39] to deploy code and trained models on the robot. To enable the adaptation to the real-world environment, there are some aspects that must be taken into account during training. As a first step, we adjust the simulation in order to reproduce realistic actuation and sensor noise. To that end, we adopt the noise model proposed in [7] based on Gaussian Mixture Models fitting real-world noise data acquired from a LoCoBot. Additionally, we modify the parameters of the RGB-D sensor used in simulation to match those of the RealSense camera mounted on the robot. Specifically, we change the camera resolution and field of view, the range of depth information, and the camera height. Finally, it is imperative to prevent the agent from learning simulation-specific shortcuts and tricks. For instance, the agent may learn to slide along the walls due to imperfect dynamics in simulation [13]. To prevent the learning of such dynamics, we employ the *bump* sensor provided by Habitat and block the agent whenever it is in contact with an obstacle. When deployed in the real world, our agent is able to explore the environment without getting stuck or bumping into obstacles. In the video accompanying the submission, we report exploration samples taken from our real-world deployment.

[2]https://github.com/facebookresearch/OccupancyAnticipation

[3]https://locobot-website.netlify.com

## VI. Conclusion

In this work, we presented an impact-driven approach for robotic exploration in indoor environments. Different from previous research that considered a setting with procedurally-generated environments with a finite number of possible states, we tackle a problem where the number of possible states is non-numerable. To deal with this scenario, we exploit a deep neural density model to compute a running pseudo-count of past states and use it to regularize the impact-based reward signal. The resulting intrinsic reward allows to efficiently train an agent for exploration even in absence of an extrinsic reward. Furthermore, extrinsic rewards and our proposed reward can be jointly used to improve training efficiency in reinforcement learning. The proposed agent stands out from the recent literature on embodied exploration in photorealistic environments. Additionally, we showed that the trained models can be deployed in the real world.

## References


[1] C. Stachniss, *Robotic Mapping and Exploration*. Springer, 2009, vol. 55.

[2] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments," *RA-L*, vol. 4, no. 2, pp. 1699–1706, 2019.

[3] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik *et al.*, "Habitat: A Platform for Embodied AI Research," in *ICCV*, 2019.

[4] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames," in *ICLR*, 2019.

[5] S. D. Morad, R. Mecca, R. P. Poudel, S. Liwicki, and R. Cipolla, "Embodied visual navigation with automatic curriculum learning in real environments," *RA-L*, vol. 6, no. 2, pp. 683–690, 2021.

[6] S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss, "Speeding-Up Robot Exploration by Exploiting Background Information," *RA-L*, vol. 1, no. 2, pp. 716–723, 2016.

[7] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning To Explore Using Active Neural SLAM," in *ICLR*, 2019.

[8] T. Chen, S. Gupta, and A. Gupta, "Learning Exploration Policies for Navigation," in *ICLR*, 2019.

[9] S. K. Ramakrishnan, Z. Al-Halah, and K. Grauman, "Occupancy Anticipation for Efficient Exploration and Navigation," in *ECCV*, 2020.

[10] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3D: Learning from RGB-D Data in Indoor Environments," in *3DV*, 2017.

[11] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *CVPR*, 2018.

[12] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford *et al.*, "RoboTHOR: An Open Simulation-to-Real Embodied AI Platform," in *CVPR*, 2020.

[13] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, "Sim2Real Predictivity: Does evaluation in simulation predict real-world performance?" *RA-L*, vol. 5, no. 4, pp. 6670–6677, 2020.

[14] R. Bigazzi, F. Landi, M. Cornia, S. Cascianelli, L. Baraldi, and R. Cucchiara, "Out of the Box: Embodied Navigation in the Real World," in *CAIP*, 2021.

[15] J. Truong, S. Chernova, and D. Batra, "Bi-directional domain adaptation for sim2real transfer of embodied navigation agents," *RA-L*, vol. 6, no. 2, pp. 2634–2641, 2021.

[16] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q.-H. Meng, "Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments," in *ICRA*, 2018.

[17] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," *RA-L*, vol. 4, no. 2, pp. 610–617, 2019.

[18] S. K. Ramakrishnan, D. Jayaraman, and K. Grauman, "An Exploration of Embodied Visual Exploration," *IJCV*, vol. 129, no. 5, pp. 1616–1649, 2021.

[19] R. Raileanu and T. Rocktäschel, "RIDE: Rewarding impact-driven exploration for procedurally-generated environments," in *ICLR*, 2021.

[20] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *NeurIPS*, 2016.

[21] R. Bigazzi, F. Landi, M. Cornia, S. Cascianelli, L. Baraldi, and R. Cucchiara, "Explore and Explain: Self-supervised Navigation and Recounting," in *ICPR*, 2020.

[22] B. Yamauchi, "A Frontier-Based Approach for Autonomous Exploration," in *CIRA*, 1997.

[23] H. H. González-Banos and J.-C. Latombe, "Navigation Strategies for Exploring Indoor Environments," *IJRR*, vol. 21, no. 10-11, pp. 829–848, 2002.

[24] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the Efficiency of Frontier-based Exploration Strategies," in *ISR and ROBOTIK*, 2010.

[25] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding Horizon "Next–Best–View" Planner for 3D Exploration," in *ICRA*, 2016.

[26] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos, "Count-based exploration with neural density models," in *ICML*, 2017.

[27] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," in *NeurIPS*, 2016.

[28] T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian, "Bebold: Exploration beyond the boundary of explored regions," *arXiv preprint arXiv:2012.08621*, 2020.

[29] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[30] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *ICLR*, 2018.

[31] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," in *ICML*, 2019.

[32] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2013.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *NeurIPS*, 2014.

[34] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *ICLR*, 2016.

[35] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *ICML*, 2019.

[36] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *ICML*, 2018.

[37] M. Z. Irshad, C.-Y. Ma, and Z. Kira, "Hierarchical Cross-Modal Agent for Robotics Vision-and-Language Navigation," in *ICRA*, 2021.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[39] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "PyRobot: An Open-source Robotics Framework for Research and Benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.