This is the peer reviewd version of the followng article:

Out of the Box: Embodied Navigation in the Real World / Bigazzi, R., Landi, F., Cornia, M., Cascianelli, S., Baraldi, L., Cucchiara, R.. - 13052:(2021), pp. 47-57. (19th International Conference on Computer Analysis of Images and Patterns, CAIP 2021 Virtual 27 September - 01 October 2021) [10.1007/978-3-030-89128-2_5].

Springer Science and Business Media Deutschland GmbH
Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

07/11/2026 14:45

(Article begins on next page)

Out of the Box: Embodied Navigation in the Real World

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

University of Modena and Reggio Emilia, Modena, Italy
{name.surname}@unimore.it

Abstract. The research field of Embodied AI has witnessed substantial progress in visual navigation and exploration thanks to powerful simulating platforms and the availability of 3D data of indoor and photorealistic environments. These two factors have opened the doors to a new generation of intelligent agents capable of achieving nearly perfect PointGoal Navigation. However, such architectures are commonly trained with millions, if not billions, of frames and tested in simulation. Together with great enthusiasm, these results yield a question: how many researchers will effectively benefit from these advances? In this work, we detail how to transfer the knowledge acquired in simulation into the real world. To that end, we describe the architectural discrepancies that damage the Sim2Real adaptation ability of models trained on the Habitat simulator and propose a novel solution tailored towards the deployment in real-world scenarios. We then deploy our models on a LoCoBot, a Low-Cost Robot equipped with a single Intel RealSense camera. Different from previous work, our testing scene is unavailable to the agent in simulation. The environment is also inaccessible to the agent beforehand, so it cannot count on scene-specific semantic priors. In this way, we reproduce a setting in which a research group (potentially from other fields) needs to employ the agent visual navigation capabilities as-a-Service. Our experiments indicate that it is possible to achieve satisfying results when deploying the obtained model in the real world. Our code and models are available at https://github.com/aimagelab/LoCoNav.

Keywords: Embodied AI · Sim2Real · Visual Navigation.

1 Introduction

Embodied AI has recently attracted a lot of attention from the vision and learning communities. This ambitious research field strives for the creation of intelligent agents that can interact with the surrounding environment. Smart interactions, however, require fine-grained perception and effective planning abilities. For this reason, current research focuses on the creation of rich and complex architectures that are trained in simulation with a large amount of data. Thanks to powerful simulating platforms [7,15,19], the Embodied AI community could achieve nearly perfect results on the PointGoal Navigation task (PointNav) [18].

However, current research is still in the first mile of the race for the creation of intelligent and autonomous agents. Naturally, the next milestones involve bridging the gap between simulated platforms (in which the training takes place) and the real world [8]. In this work, we aim to design a robot that can navigate in unknown, real-world environments [2].

We ask ourselves a simple research question: *can the agent transfer the skills acquired in simulation to a more realistic setting?* To answer this question, we devise a new experimental setup in which models learned in simulation are deployed on a LoCoBot [10]. Previous work on Sim2Real adaptability from the Habitat simulator [15] has focused on a setting where the real-world environment was matched with a corresponding simulated environment to test the Sim2Real metric gap. To that end, Kadian *et al.* [8] carry on a 3D acquisition of the environment specifically built for robotic experiments. Here, we assume a setting in which the final user cannot count on the technology/expertise required to make a 3D scan. This experimental setup is more challenging for the agent, as it cannot count on semantic priors on the environment acquired in simulation. Moreover, while [8] employs large boxes as obstacles, our testing scene contains real-life objects with complicated shapes such as desks, office chairs, and doors.

Our agent builds on a recent model proposed by Ramakrishnan *et al.* [12] for the PointNav task. As a first step, we research the optimal setup to train the agent in simulation. We find out that default options (tailored for simulated tasks) are not optimal for real-world deployment: for instance, the simulated agents often exploit imperfections in the simulator physics to slide along the walls. As a consequence, deployed agents tend to get stuck when trying to replicate the same sliding dynamic. By enforcing a more strict interaction with the environment, it is possible to avoid such shortcomings in the locomotor policy. Secondly, we employ the software library PyRobot [11] to create a transparent interface with the LoCoBot: thanks to PyRobot, the code used in simulation can be seamlessly deployed on the real-world agent by changing only a few lines of code. Finally, we test the navigation capabilities of the trained model on a real scene: we create a set of navigation episodes in which goals are defined using relative coordinates. While previous tests were mainly made in robot-friendly scenarios (often consisting of a single room), we test our model, which we call LoCoNav, in a more realistic environment with multiple rooms and typical office furniture (Fig.1). Thanks to our experiments, we show that models trained in simulation can adapt to real unseen environments. By making our code and models publicly available, we hope to motivate further research on Sim2Real adaptability and deployment in the real world of agents trained on the Habitat simulator.

2 Related Work

There is a broad area of recent research that focuses on designing autonomous agents with different abilities. Among these, a vast line of work concentrates on embodied exploration and navigation [3,5,9,12,13]. In this setting, the agent's

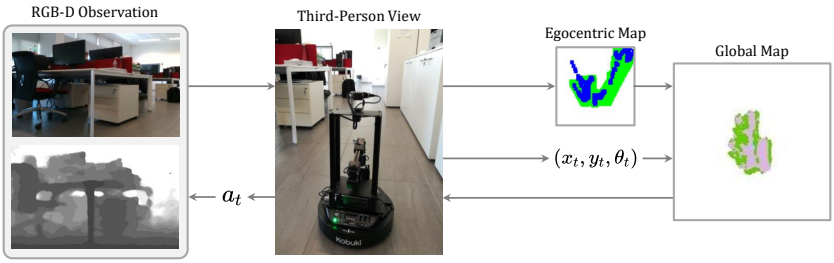

Fig. 1. We deploy a state-of-art navigation architecture on a LoCoBot and test it in a realistic, office-like environment. Our model exploits egocentric and global occupancy maps to plan a route towards the goal.

goal is to explore a new environment in the shortest amount of time. Architectures trained for this task usually employ reinforcement learning to maximize coverage (the area seen during a single episode) [3], surprisal [1], or a reward based on the novelty of explored areas [13]. Usually, this is done by creating internal map representations to keep track of the exploration progress and at the same time help the agent plan for future destinations [3,5,12]. The main advantage of these approaches is their ability to adapt to downstream tasks, such as PointGoal [12] or ObjectGoal [4] navigation. In PointGoal navigation, the target destination is specified using relative coordinates *w.r.t.* the agent's initial position and heading [15]. Using simulation and impressive computational power, Wijmans *et al.* [18] achieve nearly perfect results. However, their model is trained using 2.5 billion frames and requires experience acquired over more than half a year of GPU time. Unfortunately, models tend to learn simulator-specific tricks to circumvent navigation difficulties [8]. Since such shortcuts do not work in the real world, there is a significant Sim2Real performance gap.

Recent work has studied how to deploy models trained on simulation to the real world [7,8,14]. In their work, Kadian *et al.* [8] make a 3D acquisition of a real-world scene and study the Sim2Real gap for various setups and metrics. However, their environment is very simple as obstacles are large boxes, the floor has an even and regular surface in order to facilitate the actuation system, and there are no doors or other navigation bottlenecks. In this work, instead, we focus on a more realistic type of environment: obstacles are represented by common office furniture such as desks, chairs, cupboards; the floor is uneven as there are gaps between floor tiles that make actuation noisy and very position-dependent, and there are multiple rooms that must be accessed through doorways.

3 Real-World Navigation with Habitat

In this section, we describe our out-of-the-box navigation robot. First, we describe the baseline architecture and its training procedure that takes place in the Habitat simulator [15]. Then we present our LoCoNav agent, which builds upon the baseline and implements various modules to enable real-world navigation.

3.1 Baseline Architecture

We draw inspiration from the occupancy anticipation agent [12] to design our baseline architecture. The model consists of three main parts: a mapper, a pose estimator, and a hierarchical policy, that we describe in the following.

Mapper. The mapper is responsible for producing an occupancy map of the environment, which is then employed by the agent as an auxiliary representation during navigation. We use two different types of map at each time step t: the agent-centric map v_t that depicts the portion of the environment immediately in front of the agent, and the global map m_t that captures the area of the environment already visited by the agent. The global map of the environment m_t is blank at $t = 0$ and it is built in an incremental way. Each map has two channels, identifying the free/occupied and the explored/unexplored space, respectively; each pixel contains the state of a 5cm $\times$ 5cm area. The mapper module takes as input the RGB and depth observations (o_t^r, o_t^d) at time t and produces the agent-centric map $v_t \in [0,1]^{2\times V\times V}$. The RGB observation is encoded to a feature representation $\bar{o}_t^r$ with the first two layers of a pretrained ResNet-18 followed by a three-layered CNN. Instead, the depth observation is used to create a point-cloud and reprojected to form a preliminary map $\bar{o}_t^d$. The resulting agent-centric map v_t is computed by combining $\bar{o}_t^r$ and $\bar{o}_t^d$ with a U-Net. Then, v_t is registered to the global map $m_t \in [0,1]^{2\times W\times W}$, with $W > V$, using the agent's position and heading in the environment (x_t, y_t, θ_t).

Pose Estimator. While the agent navigates towards the goal, the interactions with the environment are subject to noise and errors, so that, for instance, the action *go forward* $25cm$ might not result in a real displacement of 25cm. That could happen for a variety of reasons: bumping into an obstacle, slipping on the terrain, or simple actuation noise. The pose estimator is responsible of avoiding such positioning mistakes and keeps track of the agent pose in the environment at each time step t. This module computes the relative displacement $(\Delta x_t, \Delta y_t, \Delta\theta_t)$ caused by the action selected by the agent at time t. It takes as input the RGB-D observations (o_t^r, o_t^d) and (o_{t-1}^r, o_{t-1}^d) retrieved at time t and $t-1$, and the egocentric maps v_t and v_{t-1}. Each modality is considered separately to obtain a first estimate of the displacement:

$$g_i = W_1 \max(W_2 \star + b_2, 0) + b_1, \tag{1}$$

The final output of the pose estimator is the weighted sum of the three displacement vectors g_i:

$$(\Delta x_t, \Delta y_t, \Delta\theta_t) = \sum_{i=0}^{2} \alpha_i \cdot g_i, \qquad \alpha_i = \mathrm{softmax}(\mathrm{MLP}_i([\bar{o}_t^r, \bar{o}_t^d, \bar{v}_t])), \tag{2}$$

where MLP is a three-layered fully-connected network, ($\bar{o}_t^r$, $\bar{o}_t^d$, $\bar{v}_t$) 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 $(x_t, y_t, \theta_t) = (x_{t-1}, y_{t-1}, \theta_{t-1}) + (\Delta x_t, \Delta y_t, \Delta\theta_t)$.

Hierarchical Policy. Following a current trend in Embodied AI [3,5,12], we employ a hierarchical policy in our baseline navigator. The highest-level component of our policy is the global policy. The global policy selects a long-term

goal on the global map, that we call global goal. The input of the global policy at time t is a 4-channel enriched global map $m_t^+ \in [0,1]^{4\times W\times W}$ obtained as the concatenation of the global map m_t with a spatial representation of visited states and a one-hot representation of the agent position at time t. Finally, we compute an 8-channel input of shape $G \times G$ for the global policy. To that end, we concatenate a cropped and a max-pooled version of m_t^+. The global policy outputs a probability distribution over the $G \times G$ action space. The global goal is sampled from this distribution and is then converted to (x, y) coordinates on the global map. A new global goal is sampled every N time steps during training and is set to the navigation goal during deployment and test. The middle-level component of our hierarchical policy is the planner. After the global goal is set, an A* planner decodes the next local goal within 0.25m from the agent and on the trajectory towards the global goal. A new local goal is sampled if at least one of the following three conditions verifies: a new global goal is sampled by the global policy, the previous local goal is reached, or the local goal is known to be in an occupied area. Finally, the local policy performs the low-level navigation and decodes the series of actions to perform. The actions available to the agents are *go forwards* $25cm$ and *turn* $15°$. The local policy samples an atomic action a_t at each time step t.

3.2 Training in Simulation

The baseline architecture described in the previous lines is trained in simulation using Habitat [15] and 3D scans from the Gibson dataset of spaces [19]. The mapper is trained with a binary cross-entropy loss using the ground-truth occupancy maps of the environment, obtained as described in [12]. The navigation policy is trained using reinforcement learning. We choose PPO [16] as training algorithm. The global policy receives a reward signal equal to the increase in terms of anticipated map accuracy [12]:

$$R_t^{glob} = \text{Accuracy}(m_t, \hat{m}) - \text{Accuracy}(m_{t-1}, \hat{m}), \tag{3}$$

where m_t and m_{t-1} represent the global occupancy maps computed at time t and $t-1$ respectively, and $\hat{m} \in [0,1]^{2\times W\times W}$ is the ground-truth global map. The map accuracy is defined as:

$$\text{Accuracy}(m, \hat{m}) = \sum_{i=1}^{W^2}\sum_{j=1}^{2} \mathbb{1}[m_{ij} = \hat{m}_{ij}], \tag{4}$$

where $\mathbb{1}[\cdot]$ is an indicator function that returns one if the condition $[\cdot]$ is true and zero otherwise. The local policy is trained using a reward that encourages the decrease in the euclidean distance between the agent and the local goal while penalizing collisions with obstacles:

$$r_t^{local} = d_t - d_{t-1} - \alpha * bump_t, \tag{5}$$

where d_t and d_{t-1} are the euclidean distances to the local goal at times t and $t-1$, $bump_t \in \{0, 1\}$ identifies a collision at time t and α regulates the contributions of the collision penalty. The training procedure described in this section exploits the experience collected throughout 6.5 million exploration frames.

3.3 LoCoNav: Adapting for Real World

The baseline architecture described above is trained in simulation and achieves state-of-art results on embodied exploration and navigation [12]. The reality, however, poses some major challenges that need to be addressed to achieve good real-world performances. For instance, uneven ground might give rise to errors and noise in the actuation phase. To overcome this and other discrepancies between simulated and real environments, we design LoCoNav: an agent that leverages the availability of powerful simulating platforms during training but is tailored for real-world use. In this section, we describe the main characteristics of the LoCoNav design. We deploy our architecture on a LoCoBot [10] and use PyRobot [11] for seamless code integration.

Prevent your Agent from Learning Tricks. All simulations are imperfect. One of the main objectives when training an agent for real-world use in simulation is to prevent it from learning simulator-specific tricks instead of the basic navigation skills. During training, we observed that the agent tends to hit the obstacles instead of avoiding them. This behavior is given by the fact that the simulator allows the agent to slide towards its direction even if it is in contact with an obstacle as if there were no friction at all. Unfortunately, this ideal situation does not fit the real world, as the agent needs to actively rotate and head towards a free direction every time it bumps into an obstacle. To replicate the realistic *sticky* behavior of surfaces, we check the $bump_t$ flag before every step. If a collision is detected, we prevent the agent from moving forward. As a result, our final agent is more cautious about any form of collision.

Sensor and Actuation Noise. Another important discrepancy between simulation and real-world is the difference in the sensor and actuation systems. Luckily, the Habitat simulator allows for great customization of input-output dynamics, thus being very convenient for our goal. In order to train a model that is more resilient to the camera noise, we apply a Gaussian Noise Model on the RGB observations and a Redwood Noise Model [6] on the depth observations. Unfortunately, the LoCoBot RealSense camera still presents various artifacts and regions with missing depth values. For that reason, we need to restore the observation retrieved from the depth camera before using it in our architecture. To that end, we apply the hole filling algorithm described in [17], followed by the application of a median filter.

Table 1. List of hyperparameters changes for Sim2Real transfer.

	Height	RGB FoV	Depth FoV	Depth Range	Obst. Height Thresh.
Default for Simulation	1.25	**H**: 90, **V**: 90	**H**: 90, **V**: 90	[0.0, 10.0]	[0.2, 1.5]
LoCoNav (ours)	0.60	**H**: 70, **V**: 90	**H**: 57, **V**: 86	[0.0, 5.00]	[0.3, 0.6]

Regarding the actuation noise, we find out that the use of the incremental pose estimator (employed in the occupancy anticipation model and described in our baseline architecture) is not optimal, especially when combined with the actuation noise typical of real-world scenarios. Luckily, we can count on more precise and reliable information coming from the LoCoBot actuation system. By checking the actual rotation of each wheel at every time step, the robot can update its position step by step. We adapt the odometry sensor of the LoCoBot platform to be compliant with our architecture. To that end, the pose returned by the sensor is converted by resetting it with respect to its state at the beginning of the episode. We name $\chi_0 = (x_0, y_0, \theta_0)$ the coordinate triplet given by the odometry sensor at $t = 0$. We then define:

$$\mathbf{A} = \begin{pmatrix} \mathbf{R}_0 & \mathbf{t}_0 \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_0 & -\sin\theta_0 & x_0 \\ \sin\theta_0 & \cos\theta_0 & y_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{6}$$

Let us define $\mathbf{x}_t$ as the augmented position vector $(x_t, y_t, 1)$ containing the agent position at each step t. We compute the relative position of the robot as:

$$\tilde{\mathbf{x}}_t = \mathbf{A}^{-1}\mathbf{x}_t, \qquad \tilde{\theta}_t = \theta_t - \theta_0 \tag{7}$$

where $\tilde{\mathbf{x}}_t = (\tilde{x}_t, \tilde{y}_t, 1)$ contains the agent position after the conversion to episode coordinates. The relative position and heading is given by $\tilde{\chi}_t = (\tilde{x}_t, \tilde{y}_t, \tilde{\theta}_t)$. Note that, for $t = 0$, $\tilde{\chi}_0 = (\tilde{x}_0, \tilde{y}_0, \tilde{\theta}_0) = (0, 0, 0)$.

Hyperparameters. Finally, we noticed that typical hyperparameters employed in simulation do not match the real robot characteristics. For instance, the camera height is set to 1.25m in previous works, but the RealSense camera on the LoCoBot is placed only 0.6m from the floor. During the adaptation to the real-world robot, we change some hyperparameters to align the observation characteristics of the simulated and the real world and to match real robot constraints. These parameters are listed in Table 1.

4 Experiments

Testing Setup. We run multiple episodes in the real environment, in which the agent needs to navigate from a starting point A to a destination B. The goal is specified by using relative coordinates (in meters) with respect to the agent's starting position and heading. Although the agent knows the position of its destination, it has no prior knowledge of the surrounding environment. Because of this, it cannot immediately plan a direct route to the goal and must check for obstacles and walls before stepping ahead. After each run, we reset the agent memory so that it cannot retain any information from previous episodes. We design five different navigation episodes that take place in three different office rooms and the corridor connecting them (Fig. 2a). For each episode, we run different trials with different configurations: obstacles are added/moved, or people are sitting/standing in the room. In total, we run 50 different experiments, resulting in more than 10 hours of real-world testing.

(a)

Path	Length(m)	Time(s)	# Step
A	3.80	124	23
B	6.75	239	45
C	5.95	223	43
D	6.55	217	42
E	4.20	227	33

(b)

Fig. 2. Layout of the navigation episodes (a). Path-specific information, as obtained with human supervision (b).

Evaluation Protocol. An episode is considered successful if the agent sends a specific *stop* signal within $0.2m$ from the goal. This threshold corresponds to the radius of the robot base. For every navigation episode, we also track the number of steps and the time required to reach the goal. Since the absolute number of steps is not comparable among different episodes, we ask human users to control the LoCoBot and complete each navigation path via a remote interface (we report human performance in Fig 2b). We then normalize these measures using this information so that results close to 1.00 indicate human-like performances. We provide absolute and normalized length and time for each episode, as well as the popular SPL metric (Success rate weighted by inverse Path Length). We employ a slightly modified version of the SPL, in which the normalization is made basing on the number of steps and not on the effective path length to penalize purposeless rotations. Additionally, we set a boolean flag for each episode that signals whether the robot has bumped into an obstacle, and we report the average Bump Rate (BR). We also report the Hard Failure Rate (HFR) as the fraction of episodes terminated if the agent gets stuck and cannot proceed, or if the episode length exceeds the limit of 300 steps.

Real-world Navigation. In this experiment, we test our robot on five different realistic navigation paths (Fig. 2a). We report the numerical results for these experiments in Table 2, and we plot the main metrics in Fig. 3 to allow for a better visualization of navigation results across different episodes. When a path is contained in a single room (A), the agent achieves optimal results, as it always stops within the success threshold from the goal. The number of steps is slightly higher than the minimum required by the episode (33 instead of 23), but this overhead is necessary as the agent must rotate and "look around" to build a decent map of the surrounding before planning a route to the goal. Paths that involve going outside the room and navigating different spaces (B, C, D, E) are fairly complicated, but the agent can generally terminate the episode without hard failures. When the shortest path to the goal leads to a wall or a dead-end, the agent needs to find an alternative way to circumvent this obstacle (*e.g.* a door). This leads to a higher episode length because the robot must dedicate some time to general exploration of the surroundings. Finally, we find out that

Table 2. Navigation results. Numbers after ± denote the standard error of the mean.

Path	SR ↑	SPL ↑	HFR ↓	BR ↓	Abs. Steps	Norm. Steps ↑	Abs. Time	Norm. Time ↑
A	1.0	0.718	0.0	0.30	32.70±1.73	0.717±0.033	176.11±10.39	0.718±0.031
B	0.8	0.711	0.10	0.22	51.67±1.72	0.880±0.027	273.70±8.24	0.879±0.030
C	0.5	0.205	0.10	0.78	123.44±10.66	0.374±0.034	631.15±50.09	0.372±0.036
D	0.5	0.318	0.10	0.89	65.67±3.90	0.645±0.037	344.00±20.08	0.657±0.038
E	0.2	0.060	0.40	1.00	135.17±29.97	0.290±0.049	722.76±162.01	0.38±0.066
Overall	0.6	0.402	0.14	0.60	-	0.608±0.036	-	0.617±0.034

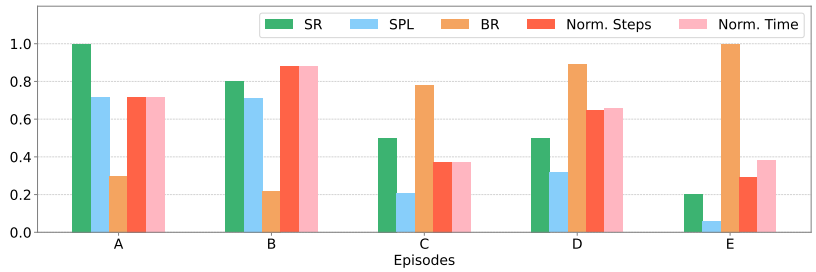

Fig. 3. Comparison of the main navigation metrics on different episodes.

the most challenging scenario for our LoCoNav is when reaching the goal requires to get out of a room and then enter a door immediately after, on the same side of the corridor (as in E). Since the robot sticks to the shortest path, the low parallax prevents it from identifying the second door correctly. Even in these cases, a bit of general exploration helps to solve the problem.

Discussion and Failure Cases. Overall, our experimental setup provides a challenging test-bed for real-world robots. We find out that failures are due to two main issues. First, when the agent must navigate to a different room, it has no access to a map representing the general layout of the environment. This prevents the robot from computing a general plan to reach the long-term goal and forces it to explore the environment before proceeding. If a map was given to the agent, this problem would have been greatly alleviated. A second problem arises when the goal is close in terms (x, y) coordinates but is physically placed in an adjacent room. To solve this problem, one could decompose the navigation between rooms in a multi-goal problem where neighboring nodes are closer. In this way, it is possible to reduce a complex navigation episode in simpler sub-episodes (like A or B), in which our agent has proved to be successful.

5 Conclusion

We have presented LoCoNav, an out-of-the-box architecture for embodied navigation in the real world. Our model takes advantage of two main elements: state-of-art simulating platforms, together with a large number of 3D spaces, for efficient and fast training, and a series of techniques specifically designed for real-world deployment. Experiments are conducted in reality on challenging navigation paths and in a realistic office-like environment. Results demonstrate the validity of our approach and encourage further research in this direction.

Acknowledgment

This work has been supported by "Fondazione di Modena" under the project "AI for Digital Humanities" and by the national project "IDEHA" (PON ARS01_00421), cofunded by the Italian Ministry of University and Research.

References

1. Bigazzi, R., Landi, F., Cornia, M., Cascianelli, S., Baraldi, L., Cucchiara, R.: Explore and Explain: Self-supervised Navigation and Recounting. In: ICPR (2020)
2. Cascianelli, S., Costante, G., Bellocchio, E., Valigi, P., Fravolini, M.L., Ciarfuglia, T.A.: A robust semi-semantic approach for visual localization in urban environment. In: ISC2 (2016)
3. Chaplot, D.S., Gandhi, D., Gupta, S., Gupta, A., Salakhutdinov, R.: Learning To Explore Using Active Neural SLAM. In: ICLR (2019)
4. Chaplot, D.S., Gandhi, D.P., Gupta, A., Salakhutdinov, R.R.: Object Goal Navigation using Goal-Oriented Semantic Exploration. In: NeurIPS (2020)
5. Chen, T., Gupta, S., Gupta, A.: Learning Exploration Policies for Navigation. In: ICLR (2019)
6. Choi, S., Zhou, Q.Y., Koltun, V.: Robust reconstruction of indoor scenes. In: CVPR (2015)
7. Deitke, M., Han, W., Herrasti, A., Kembhavi, A., Kolve, E., Mottaghi, R., Salvador, J., Schwenk, D., VanderBilt, E., Wallingford, M., et al.: RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. In: CVPR (2020)
8. Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., Batra, D.: Sim2Real Predictivity: Does evaluation in simulation predict real-world performance? IEEE Robot. Autom. Letters **5**(4), 6670–6677 (2020)
9. Landi, F., Baraldi, L., Cornia, M., Corsini, M., Cucchiara, R.: Multimodal Attention Networks for Low-Level Vision-and-Language Navigation. CVIU (2021)
10. LoCoBot: An Open Source Low Cost Robot. https://locobot-website.netlify.com
11. Murali, A., Chen, T., Alwala, K.V., Gandhi, D., Pinto, L., Gupta, S., Gupta, A.: PyRobot: An Open-source Robotics Framework for Research and Benchmarking. arXiv preprint arXiv:1906.08236 (2019)
12. Ramakrishnan, S.K., Al-Halah, Z., Grauman, K.: Occupancy Anticipation for Efficient Exploration and Navigation. In: ECCV (2020)
13. Ramakrishnan, S.K., Jayaraman, D., Grauman, K.: An Exploration of Embodied Visual Exploration. IJCV pp. 1–34 (2021)
14. Rosano, M., Furnari, A., Gulino, L., Farinella, G.M.: On Embodied Visual Navigation in Real Environments Through Habitat. In: ICPR (2020)
15. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., et al.: Habitat: A Platform for Embodied AI Research. In: ICCV (2019)
16. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
17. Telea, A.: An image inpainting technique based on the fast marching method. J. of Graphics Tools **9**(1), 23–34 (2004)
18. Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., Batra, D.: DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In: ICLR (2019)
19. Xia, F., Zamir, A.R., He, Z., Sax, A., Malik, J., Savarese, S.: Gibson Env: Real-world perception for embodied agents. In: CVPR (2018)