Learning Real-world Autonomous Navigation by Self-Supervised Environment Synthesis

Zifan Xu*[1], Anirudh Nair*[1], Xuesu Xiao[1], and Peter Stone[1,2]

***Abstract*— Machine learning approaches have recently enabled autonomous navigation for mobile robots in a data-driven manner. Since most existing learning-based navigation systems are trained with data generated in artificially created training environments, during real-world deployment at scale, it is inevitable that robots will encounter unseen scenarios, which are out of the training distribution and therefore lead to poor real-world performance. On the other hand, directly training in the real world is generally unsafe and inefficient. To address this issue, we introduce Self-supervised Environment Synthesis (SES), in which, after real-world deployment with safety and efficiency requirements, autonomous mobile robots can utilize experience from the real-world deployment, reconstruct navigation scenarios, and synthesize representative training environments in simulation. Training in these synthesized environments leads to improved future performance in the real world. In our experiments, the effectiveness of SES in synthesizing representative simulation environments and improving real-world navigation performance has been verified by a large-scale deployment in a high-fidelity, realistic simulator[1].**

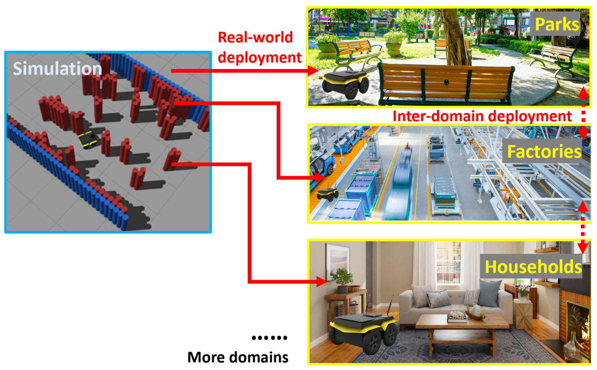

Fig. 1. A navigation policy trained in simulation is expected to be deployed in completely different domains of navigation environments in the real world (e.g., households, factories, and parks). The policy may also need to face different real-world inter-domain deployments, in which a navigation policy learned in one real-world domain will be deployed in another.

I. INTRODUCTION

While classical navigation systems have been able to move mobile robots from one point to another in a collision-free manner for decades [2], [3], learning-based approaches to navigation have recently gained traction [4] due to their ability to learn navigation behaviors purely from data without extensive engineering effort. For example, learned navigation systems can learn from human demonstrations [5] or self-supervised trial and error [6]; they can learn navigation cost functions that consider social norms and human preferences [7]. They can also be combined with classical navigation systems to assure navigation safety and enable adaptive behaviors in different scenarios [8]–[12].

Due to the expense of trial-and-error training in the real world (e.g., safety concerns and sample efficiency), most navigation behaviors are learned in artificially created environments in simulation, which may not generalize well to the real world (shown in Fig. 1). Despite the effort in creating simulation environments similar to the real world or enabling efficient sim-to-real transfer, it is inevitable that robots will encounter unfamiliar scenarios, especially during large-scale real-world deployment.

The goal of this work is to improve real-world autonomous navigation with safety and efficiency requirements based on mobile robots' own navigation experiences during actual deployment. These conservative, potentially suboptimal, real-world experiences (without risky real-world exploration) may not be sufficient to directly train an RL agent, but are sufficient to reconstruct the real-world navigation scenarios in simulation which an RL agent can interact with and actively explore. On the other hand, given the large amount of real-world deployment experiences available to many robots deployed in the field (consider 7 million connected iRobot Roombas vacuuming homes day to day), it is infeasible to reconstruct all these deployment environments and train in simulation on a daily basis. Therefore, using our approach, Self-supervised Environment Synthesis (SES), mobile robots deployed in the field can first reconstruct navigation scenarios from experiences and then synthesize a representative set of simulation environments that is feasible for RL training. Training in these simulated environments, robots can learn to address real-world challenges in the future. Moreover, the distribution of real-world navigation scenarios is often unbalanced, including mostly trivial open scenarios. Therefore, we use an efficient strategy that filters out the trivial scenarios by a measure of navigation difficulty and focus learning on the difficult navigation scenarios. To synthesize the training environment set from the reconstructed difficult

* Equal contribution [1] Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 `{zfxu, ani.nair}@utexas.edu`, `{xiao, pstone}@cs.utexas.edu` [2]Sony AI

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARO (W911NF-19-2-0333), DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

[1]Due to the lack of access to large-scale real-world deployment data, we use simulated Matterport environments [1] as a surrogate of the real world.

navigation scenarios, three different environment synthesis approaches—Generative Adversarial Networks (GAN), K-means clustering with Principle Component Analysis (PCA), and random sampling—are employed to represent the difficult scenarios with a concise training environment set that is feasible for an RL agent to learn from. We evaluate all three approaches in Matterport, a dataset of large number of simulated realistic household environments (which serves as a surrogate of real world), and show that SES improves the deployment in these environments compared to policies trained in artificially generated environments [13], and the best synthesis approach generates more representative training environments and enables better deployment in Matterport.

II. RELATED WORK

This section reviews related work in classical and learning-based navigation, Adaptive Planner Parameter Learning, and domain adaptation.

A. *Classical and Learning-Based Navigation*

Mobile robot navigation has been investigated by roboticists for decades [2], [3]. Classical approaches can move robots from one point to another with a reasonable degree of confidence that they won't collide with any obstacles. However, these approaches require extensive engineering effort to develop and to adapt the system to different environments. When encountering the same environment where a robot has already failed or achieved suboptimal behavior before, without re-engineering the system, the robot will likely repeat the same mistake again.

Inspired by the success of machine learning in other domains, roboticists have also applied machine learning to autonomous navigation [4]. Most learning approaches to navigation adopt an end-to-end approach, i.e., learning a mapping from perceptual input directly to motion commands. Such approaches do not require much engineering effort, and learn navigation behaviors purely from data [14], e.g., from expert demonstrations [5], [15] or from trial and error [6], [16]. However, these approaches often lack safety guarantees and explainability, as provided by their classic counterparts. Therefore, roboticists have also investigated more structured ways of integrating learning with classical navigation, e.g., learning local planners [17]–[19], terrain-based cost functions [20], planner parameters [12], driving styles [21], and social norms [7]. Despite their success, learning-based navigation approaches inherit one drawback from machine learning approaches in general: poor generalizability when facing out-of-distribution data. When deployed in the real world, especially at large scale, it is inevitable that mobile robots will encounter scenarios that are not included in their training distribution.

SES combats classical navigation's inability to improve from experience [22] and learning approaches' poor generalizability to real-world scenarios. It improves navigation by synthesizing training environments from self-supervised deployment experiences.

B. *Adaptive Planner Parameter Learning*

Adaptive Planner Parameter Learning (APPL) [12] is a recently proposed paradigm to combine the benefits of classical navigation (e.g., safety and explainability) and learning-based navigation (e.g., adaptivity and improvement from experience). An APPL agent learns a parameter policy (in contrast to an end-to-end motion policy [23]), which interacts with a classical navigation system by dynamically adjusting the planner parameters to adapt to different environments. APPL can leverage non-expert human demonstrations (APPLD) [8], corrective intervention (APPLI) [9], and evaluative feedback (APPLE) [10] in the deployment environment to improve navigation performance. APPL can also use Reinforcement Learning (APPLR) [11] to learn a parameter policy before deployment in a randomly generated set of simulation environments via Cellular Automata (BARN) [13].

Because APPL combines the best of both worlds and is therefore more practical to be deployed at large scale in the real world, SES builds upon an APPLR agent pre-trained in the randomly generated BARN dataset [13] and improves navigation performance when facing realistic (likely out-of-distribution) navigation environments in the real world. Note that in principle SES is also applicable to end-to-end motion policies, but we leave evaluating its effectiveness in this case to future work.

C. *Sim-to-real Transfer*

Limited by the safety and efficiency requirements in the real world, a learning-based navigation system is usually trained in simulation. However, policies trained in simulation can perform poorly in the real world due to the mismatch between the simulation and the real world. This phenomenon is commonly referred to as the *sim-to-real gap*.

One major source of the *sim-to-real gap* is the discrepancies between the sensor input rendered in simulation and the real robot's sensors. For example, to bridge the gap between real-world and synthetic camera images of a robotic system, prior work has employed techniques such as pixel-level domain adaptation, which translates synthetic images to realistic ones at the pixel level [24], [25]. These adapted pseudo-realistic images bridge the *sim-to-real gap* to some extent, so policies learned in simulation can be executed more successfully on real robots by adapting the images to be more realistic. Another source of the *sim-to-real gap* is caused by dynamics mismatch between simulation and the real world e.g., due to an imperfect physics engine. A common paradigm to reduce the dynamics mismatch is Grounded Simulation Learning (GSL), which either directly modifies (i.e., grounds) the simulator to better match the real world [26], or learns an action transformer that induces simulator transitions that more closely match the real world [27], [28].

In contrast to the two *sim-to-real gaps* introduced above, this work addresses a gap caused by the environmental mismatch (e.g., differences in the configurations and shapes of obstacles, and start-goal locations). Our method SES can be thought of as an environmental grounding method that minimizes the differences in navigation environments between simulation and the real world based on the navigation experiences collected during real-world deployment.

III. APPROACH

In this section, We first formulate large-scale real-world navigation as a multi-task RL problem in an unknown navigation domain, which is defined as a distribution of navigation tasks. Sec. III-A formally defines the navigation task and describes how a distribution of navigation tasks forms a navigation domain. Then, Sec. III-B and III-C discuss the two stages of SES: real-world navigation domain extraction from real-world deployment data and environment synthesis that generates a representative set of navigation tasks. The whole pipeline of SES is summarized in Alg. 1.

A. Navigation Task and Navigation Domain

We focus on a standard goal-oriented navigation task, in which a robot navigates in a navigation environment e from a provided starting pose α to a goal pose β. Each navigation task T is instantiated as a tuple $T = (e, \alpha, \beta)$. In real-world applications, robots are not deployed to navigate in one single environment or with the same start and goal all the time. Instead, actual deployment in the real world usually form a distribution over multiple environments with many start and goal poses. In this case, we represent the real-world deployment as a navigation domain p_{real} defined as a distribution of navigation tasks $T \sim p_{\text{real}}(T)$. SES generates a new navigation domain p_{SES} in simulation that resembles p_{real} as much as possible so that the navigation performance of policies trained in p_{SES} will be maximized in the real-world navigation domain p_{real}. SES uses APPLR [11] as the learning approach that solves the navigation tasks by training a parameter policy that dynamically adjusts the hyper-parameters of a classical navigation stack. Although our implementation of SES is based on a specific learning approach (APPLR), we leave the formulation sufficiently broad so that APPLR can be replaced with any RL approaches for autonomous navigation.

B. Real-world Navigation Domain Extraction

SES seeks to explore the distribution of the real-world navigation domain during real deployment of an existing navigation policy π_0 (e.g., a policy pretrained on artificially created environments or a classical navigation system). In each real-world deployment, a navigation task $T_n \sim p_{\text{real}}(T)$ is sampled from the real-world navigation domain. At each deployment time step, SES records the sensory input x, the robot's actual position s, and the number of suboptimal or failed behaviors c (in our case, the negative linear velocity command is employed as an indicator of suboptimal navigation performance). Each deployment will return a trajectory $\tau = (x_0, s_0, c_0, ..., x_k, s_k, c_k, ..., x_K, s_K, c_K)$. Then, the trajectories are segmented into different scenarios η as follows: (1) starting from an initial step (x_i, s_i, c_i), iterate the trajectory until a final step (x_f, s_f, c_f) where the robot is 5m away from the initial step; (2) record a scenario η as a sub-trajectory of all the steps between step i and step f; (3) after one scenario is recorded, set the final step as the new initial step for the next iteration. The procedure is repeated until the whole trajectory is processed. Assuming a total of N rounds of deployments with trajectories segmented into J scenarios, a trajectory set $\{\tau_n\}_{n=1}^N$ is collected and converted into a scenario set $\{\eta_j\}_{j=1}^J$.

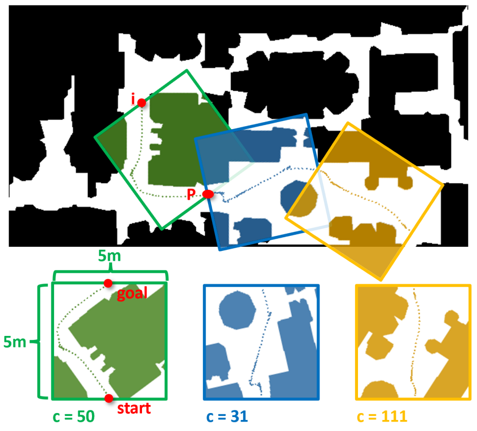

Fig. 2. An example of scenario segmentation and environment construction: dashed line denotes the actual trajectory of the robot. Three scenarios are segmented and constructed as navigation environments at the bottom. Red dots mark the positions of the start and goal in the environments. The failed motion command counts c are noted at the bottom.

For each of the scenarios, we reconstruct a 5m-by-5m navigation environment with start and goal set to be the positions of the initial and final steps, and we re-orient the environment so that the start and goal are located at the middle points of the bottom and top edges respectively. The obstacles and free spaces can be reconstructed from the recorded sensory inputs or from available maps. The constructed environment set is denoted as $\{e_j, \bar{c}_j\}$. Each environment is associated with a total number of suboptimal navigation behaviors $\bar{c}_j$ between the initial step and final step. Fig. 2 shows an example of such a scenario segmentation and environment construction process. The benefits of such scenario segmentation are: (1) encoding the start and goal into the orientation of the environment so that a single environment distribution can represent the real-world navigation domain distribution; (2) the segmented navigation environments have roughly the same length, which makes it easier for an RL agent to learn a universal value function. Alg. 2 summarizes the above process of generating an environment set that represents the real-world navigation domain distribution.

Since it is likely that the real-world navigation domain p_{real} includes many trivial navigation tasks (e.g., open spaces), efficient training should focus on the difficult navigation tasks. We use a threshold of 50 suboptimal behaviors within a scenario η as an indication of easy and difficult environments. Those difficult environments, $\{e_j, \bar{c}_j | \bar{c}_j > 50\}$, form a set for further environment synthesis. Our empirical results indicate that including the navigation tasks with only difficult scenarios instead of all the real-world scenarios is essential for improving the real-world deployment.

C. Environment Synthesis

While a large number of difficult environments can be constructed to precisely represent the real-world navigation domain distribution, it is impractical to train an RL agent in all these environments. In this case, a smaller set of training environments that represents the real-world navigation domain distribution is required. Therefore, we propose three methods of environment synthesis that perform this training data selection process and generate a concise and representative training environment set. The three environment synthesis methods are: (1) Generative Adversarial Nets (GAN) [29]; (2) K-means clustering with Principal Component Analysis (PCA) [30]; and (3) random sampling. We introduce the first two methods as follows (the third is as it sounds).

GAN learns a generator's distribution p_g over the environment e to match a uniform distribution p_{diff} over the difficult environment set. Given a prior on the input noise variables $p_z(z)$, a generator $G(z;\theta_g)$ is defined as a mapping from the input variable space to the environment space with θ_g representing the parameters of the network. A discriminator network $D(e;\theta_d)$ outputs the probability that e comes from p_{diff} rather than p_g. D is trained to maximize the probability of assigning the correct label to both difficult environments and generated environments from G. Simultaneously, G is trained to minimize $\log(1-D(G(z)))$. To summarize, D and G play the following two-player minimax game with value function $V(G,D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{e\sim p_{\text{diff}}(e)}[\log(D(e))] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]. \quad (1)$$

After a generator G is learned, we query the generator M times to draw M environments as the training set, where M is the size of the training environment set depending on the training budget of the RL algorithm.

K-means clustering first reduces the dimensionality of the original environment data by PCA so that each environment can be represented by their principle components (we empirically pick 100 principle components in our experiments because they can reasonably reconstruct the original environments). We then cluster the environments in the reduced space into m clusters and sample n environments from each cluster to select the $m \times n$ most representative environments in the difficult environment set. We use the principle components to reconstruct the representative environments and use them as our training set.

Finally, an APPLR policy π is trained in the synthesized training environments. In practice, the policy π is initialized to be the deployed policy π_0 that was trained on the artificially created environments to speed up the training.

IV. RESULTS

In this section, we describe an implementation of SES on a ground robot and show that SES can efficiently synthesize representative environments based on difficult navigation scenarios during real-world deployment and successfully improve navigation by learning from the synthesized environments. Due to the difficulty in accessing large-scale physical robot deployment data in the real world, we choose to use Matterport dataset [1], a set of high-fidelity simulation environments as a surrogate of the large-scale real-world deployment (see Fig. 3 (d)). We assume that this environment set is not available before actual deployment, and the robot needs to learn through its actual deployment experiences in those environment to improve navigation.

Algorithm 1 Self-supervised Environment Synthesis

```
Require: Original policy $\pi_0$ and real-world navigation domain distribution $p_{\text{real}}(T)$
1: Environment set $\{e_n\}_{\text{rep}} \leftarrow$ Real-world Navigation Domain Extraction based on $\pi_0$ and $p_{\text{real}}$ (Sec. III-B)
2: Training environment set $\{e_j\}_{\text{train}} \leftarrow$ Environment Synthesis based on $\{e_n\}_{\text{rep}}$ (Sec. III-C)
3: $\pi \leftarrow$ Initialize with $\pi_0$, then train on $\{e_j\}_{\text{train}}$
4: return $\pi$
```

Algorithm 2 Real-world Navigation Domain Extraction

```
Require: Original policy $\pi_0$, real-world navigation domain distribution $p_{\text{real}}(T)$ and total number of deployments $N$.
1: $E = \emptyset$
2: for $n \in \{1,...,N\}$ do
3:   $T_n \sim p_{\text{real}}(T)$
4:   Trajectory $\tau_n \leftarrow$ deploy $\pi_0$ on $T_n$
5:   Initial step $i = 0$ and the scenario set $\eta = \emptyset$
6:   for $x_k, s_k, c_k \in \tau_n$ do
7:     $\eta \leftarrow \eta \cup \{(x_k, s_k, c_k)\}$
8:     if $s_k$ is 5m away from $s_i$ then
9:       $e \leftarrow$ Construct Environment from $\eta$
10:      $\bar{c} = \sum_i^k c_i$
11:      $E \leftarrow E \cup \{e, \bar{c}\}$
12:      $i = k$, $\eta = \emptyset$
13:    end if
14:  end for
15: end for
16: return $E$
```

A. Deployment in Matterport Navigation Domain

We employ SES on a ClearPath Jackal differential-drive ground robot. The specifications of the robot are kept the same as in APPLR [11], including the ROS `move_base` navigation stack, the underlying classical local planner DWA, and the learned planner parameters.

The original APPLR policy was trained before any deployment in the BARN dataset [13]. The dataset contains 300 simulated navigation environments randomly generated by Cellular Automata. Even though those environments are sufficiently diverse to cover different difficulty levels from relatively open spaces to highly-constrained ones, when being deployed in Matterport at scale, the randomly generated environments in BARN are still not guaranteed to cover some specific navigation scenarios encountered by the robot. In

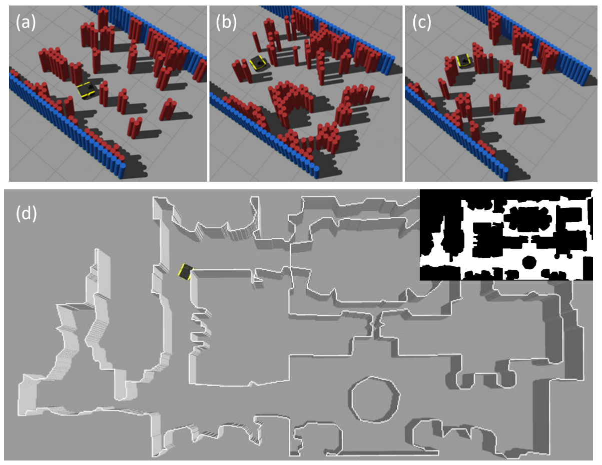

Fig. 3. Examples of training and deployment environments. (a), (b), (c): example environments from the BARN dataset; (d): a top-down view of an example Matterport environment created from the occupancy map shown at the top-right corner.

Fig. 3, (a)-(c) show three example BARN environments, and (d) shows a Matterport environment where Jackal will be deployed. This realistic Matterport environment is unlikely to be generated by random sampling.

After training the original APPLR policy with the BARN dataset, the robot will be physically deployed in the real world at scale. Due to the lack of access to large-scale real-world physical deployment data, we use 55 simulated house layouts from the Matterport dataset [1] as a surrogate for physical real-world deployment. Fig. 3 (d) shows an example of such a deployment environment. We randomly sample 50 environments from the Matterport dataset [1] as the environments where Jackal will be actually deployed. During actual deployment, these environments serve as the Matterport navigation domain and navigation experiences in those environments during deployment provide data for SES to synthesize representative environments of the Matterport navigation domain and to improve navigation. We leave five environments to test the learned policy on these held-out Matterport environments.

B. SES *Implementation*

Matterport navigation domain representation: in the large-scale deployment in Matterport environments, the robot with the original policy π_0 trained in BARN (artificially created navigation domain) is deployed 5000 times to navigate between different start-goal pairs in the Matterport environments. We construct 16528 navigation scenarios from the segmented trajectories using the method described in Sec. III-B as the distribution of the Matterport navigation domain p_{real}. Then 3769 difficult scenarios are selected based on a threshold 50 of the negative linear velocity count. Fig. 4 (left) demonstrates some examples out of the 3769 difficult scenarios. Each of the scenarios is represented as a binary matrix of size 30 × 30. Note that training in all 3769 environments is not computationally practical, and realistic large-scale robot deployment in the real world (in contrast to our surrogate) may even generate more difficult scenarios.

TABLE I
AVERAGE TIME COST AND SUCCESS RATE OF THE TESTED POLICIES

	Time cost (s)	Success rate (%)
GAN	27.23	85.0
K-means clustering	32.21	80.6
Random sampling	35.92	71.0
Learn from scratch	30.42	74.3
Original policy	36.33	71.9
DWA *slow*	43.76	75.5
DWA *fast*	31.72	67.2

Environment synthesis: given the extracted Matterport navigation domain distribution, the environment synthesis method described in Sec. III-C generates 100 training environments. More specifically, we use the difficult scenarios as an image dataset to train GAN. To specify the GAN method, We employ a normal distribution prior on the input noise variables $p_z(z) := \mathcal{N}(0,1)$, where $\mathcal{N}$ is the standard normal distribution with mean and standard deviation equal to 0 and 1 respectively. The generator $G(z;\theta_g)$ is represented by a multilayer perceptron (MLP) of 256, 512, and 1024-hidden-unit layers with batch normalization and leaky ReLU activation function. The generator outputs a 900-dimensional vector with a fully connected layer followed by a Tanh activation function. The discriminator $D(x;\theta_d)$ is represented by a second MLP of 512, 256, and 256-hidden-unit layers with leaky ReLU activation function. Each hidden layer is followed by a batch normalization layer and a dropout layer with a dropout rate equal to 0.5. The dropout serves as an ensemble strategy that prevents mode collapse in the generated images. Fig. 2 (right) shows samples of generated images drawn from the generator after training. We set $M = 100$ $(m = 20, n = 5, m \times n = 100)$ images of training environments, which are roughly in the same order of magnitude as the BARN dataset. We leave the investigation of how SES performs with different sizes of training environments as future work. To train the new policy π that solves the navigation in Matterport environments, we use the same set of hyper-parameters as the training of π_0 except for a smaller training step of one fourth of the original training step.

C. *Test Results*

We train the new APPLR policy π with the 100 new training environments created by three methods: GAN, K-means clustering, and random sampling. We compare the methods with four baselines: the APPLR policy learned from scratch on the training environments generated by GAN, the original APPLR policy and the vanilla DWA with two sets of static hyper-parameters, one with default parameters and the other with a higher maximum linear velocity and sampling rates. We call the two classical motion planners DWA *slow* and DWA *fast* respectively. We use *learn from scratch* to denote the first baseline which is designed to verify the importance

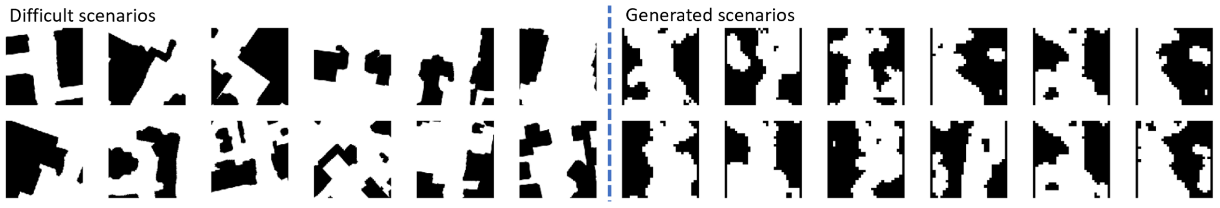

Fig. 4. Examples of difficult scenarios (left 6 columns, 12 out of 3769) and generated scenarios (right 6 columns, 12 out of 100).

of initializing the policy with the pre-trained policy π_0. Then we deployed the three adapted RL policies and four baselines in 5 held-out Matterport environments to test deployment performance. For each of the environments, we randomly sample five start-goal pairs, and deploy each policy 20 times for each pair. During deployment, we measure the success rate and the total time cost of the successful trials.

Table I presents the average time cost and success rate over all trials.[2] The policy trained by the GAN method achieves the best performances of a 27.23s average time cost and a 85% success rate. K-means clustering follows with a 32.21s average time cost and a 80.6% success rate. *Learn from scratch* shows worse performance on both metrics compared to GAN which indicates the importance of initializing the model with the pre-trained policy π_0. The policy trained on the randomly sampled training environments basically is comparable with the original policy, which manifests an inefficient transfer based on randomly sampling. While the DWA *slow* achieves an acceptable success rate of 75.5%, it takes much more time (43.76s) to finish the navigation. On the other hand, DWA *fast* can navigate relatively quickly, but more easily fails with the lowest success rate of 67.2%.

To analyze performance variance, we randomly select one start-goal pair from each environment and show the average time cost and success rate of 20 independent trials in Fig. 5. GAN and K-means clustering policies, in general, use less time to finish the navigation task and achieve higher success rate. Among five test environments, the GAN policy achieves the lowest time cost in three environments and the highest success rate in all the five environments. The proposed methods tend to show larger improvements in the difficult environments (environments 1 and 2) but also larger performance variance.

[2]Standard deviation is not included in the table due to the large variance between different environments and start-goal pairs, but we analyze the variance of the performance in every individual environment in Fig. 5.

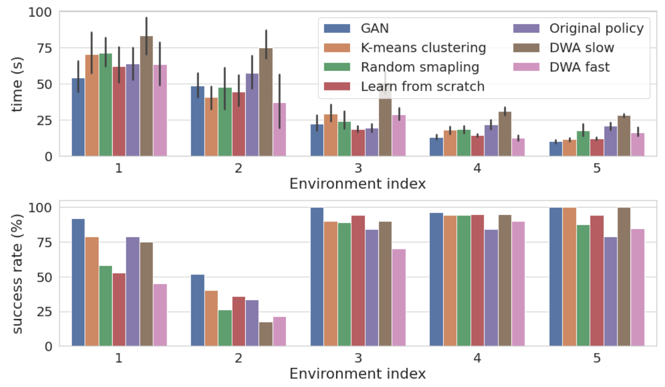

Fig. 5. The average time cost (top) and the success rate (bottom) of 5 policies tested in 5 held-out Matterport environments.

V. CONCLUSIONS

In this paper, we present a self-supervised environment synthesis approach that improves the real-world navigation performance of a learning-based navigation system by synthesizing realistic simulation environments based on navigation experiences during real-world deployment. To address the inevitable environmental mismatch between simulation and real world deployment environments, especially for large-scale real-world deployment, our SES method can successfully identify difficult navigation scenarios during real-world deployment and synthesize representative environments to be added to the training distribution. While this paper uses high-fidelity Matterport environments as a surrogate for the real world, future work should collect real-world large-scale deployment data (e.g., from robotics companies) to implement SES. Moreover, the environmental mismatch introduced in this paper may not only exist between simulation and the real world, but may also exist between different deployment scenarios in the real world (shown in Fig. 1). One interesting future direction is to develop new domain adaptation approaches so that the learned navigation policies can be adapted between different real-world scenarios.

REFERENCES

[1] 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,” *International Conference on 3D Vision (3DV)*, 2017.

[2] S. Quinlan and O. Khatib, “Elastic bands: Connecting path planning and control,” in *[1993] Proceedings IEEE International Conference on Robotics and Automation.* IEEE, 1993, pp. 802–807.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[4] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion control for mobile robot navigation using machine learning: a survey," *arXiv preprint arXiv:2011.13112*, 2020.

[5] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 ieee international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.

[6] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2371–2378.

[7] N. Pérez-Higueras, F. Caballero, and L. Merino, "Teaching robot navigation behaviors to optimal rrt planners," *International Journal of Social Robotics*, vol. 10, no. 2, pp. 235–249, 2018.

[8] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "Appld: Adaptive planner parameter learning from demonstration," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4541–4547, 2020.

[9] Z. Wang, X. Xiao, B. Liu, G. Warnell, and P. Stone, "APPLI: Adaptive planner parameter learning from interventions," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[10] Z. Wang, X. Xiao, G. Warnell, and P. Stone, "APPLE: Adaptive planner parameter learning from evaluative feedback," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[11] Z. Xu, G. Dhamankar, A. Nair, X. Xiao, G. Warnell, B. Liu, Z. Wang, and P. Stone, "APPLR: Adaptive planner parameter learning from reinforcement," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[12] X. Xiao, Z. Wang, Z. Xu, B. Liu, G. Warnell, G. Dhamankar, A. Nair, and P. Stone, "Appl: Adaptive planner parameter learning," *arXiv preprint arXiv:2105.07620*, 2021.

[13] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2020, pp. 116–121.

[14] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie Mellon University, Tech. Rep., 1989.

[15] X. Xiao, J. Biswas, and P. Stone, "Learning inverse kinodynamics for accurate high-speed off-road navigation on unstructured terrain," *IEEE Robotics and Automation Letters*, 2021.

[16] H. Karnan, G. Warnell, X. Xiao, and P. Stone, "Voila: Visual-observation-only imitation learning for autonomous navigation," *arXiv preprint arXiv:2105.09371*, 2021.

[17] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Toward agile maneuvers in highly constrained spaces: Learning from hallucination," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1503–1510, 2021.

[18] X. Xiao, B. Liu, and P. Stone, "Agile robot navigation through hallucinated learning and sober deployment," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[19] Z. Wang, X. Xiao, A. J. Nettekoven, K. Umasankar, A. Singh, S. Bommakanti, U. Topcu, and P. Stone, "From agile ground to aerial navigation: Learning from learned hallucination," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[20] M. Wigness, J. G. Rogers, and L. E. Navarro-Serment, "Robot navigation from human demonstration: Learning control behaviors," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1150–1157.

[21] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2641–2646.

[22] B. Liu, X. Xiao, and P. Stone, "A lifelong learning approach to mobile robot navigation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1090–1096, 2021.

[23] Z. Xu, X. Xiao, G. Warnell, A. Nair, and P. Stone, "Machine learning methods for local motion planning: A study of end-to-end vs. parameter learning," in *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2021.

[24] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[25] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "Rl-cyclegan: Reinforcement learning aware simulation-to-real," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 157–11 166.

[26] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 39–46.

[27] H. Karnan, S. Desai, J. P. Hanna, G. Warnell, and P. Stone, "Reinforced grounded action transformation for sim-to-real transfer," in *IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 2020)*, October 2020.

[28] J. P. Hanna, S. Desai, H. Karnan, G. Warnell, and P. Stone, "Grounded action transformation for sim-to-real reinforcement learning," *Special Issue on Reinforcement Learning for Real Life, Machine Learning, 2021*, May 2021.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.