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

Zifan Xu[*], Anirudh Nair[*], Xuesu Xiao[†,‡], and Peter Stone[*,§]

***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. The effectiveness of SES at synthesizing representative simulation environments and improving real-world navigation performance is evaluated via a large-scale deployment in a high-fidelity, realistic simulator[1] and a small-scale deployment on a physical robot.**

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]–[11].

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 (see Fig. 1). Despite efforts to create simulation environments similar to the real world or enable efficient sim-to-real transfer, it is inevitable that robots will encounter unfamiliar scenarios, especially in large-scale real-world deployments.

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 may be sufficient to reconstruct the real-world navigation scenarios which an RL agent can interact with and actively explore in simulation. 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.

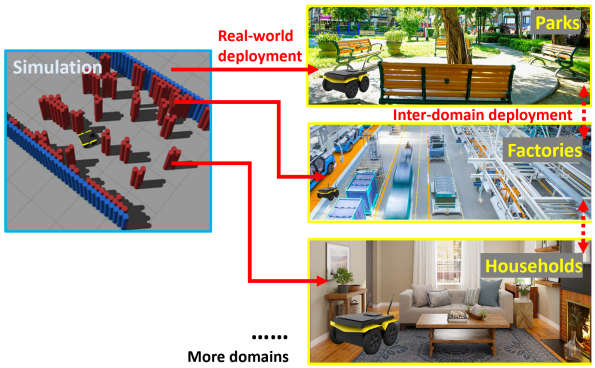

Fig. 1: A navigation policy trained in simulation is expected to be deployed in completely different domains of navigation environments in the real world (e.g., households, factories, and parks). The policy may also face different real-world inter-domain deployments, in which a navigation policy learned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

With this motivation in mind, this paper introduces Self-supervised Environment Synthesis (SES), which enables mobile robots deployed in the field to 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 enables robots to learn to address real-world challenges that they are likely to encounter.

Importantly, 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 challenging navigation scenarios. To synthesize the training environment set from the reconstructed challenging navigation scenarios, three different environment synthesis approaches—Generative Adversarial Networks (GAN), K-means clustering with Principle Compo-

[*]Department of Computer Science, The University of Texas at Austin [†]Department of Computer Science, George Mason University [‡]Everyday Robots [§]Sony AI {zfxu, ani.nair}@utexas.edu, xiao@gmu.edu, pstone@cs.utexas.edu

[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.

nent Analysis (PCA), and random sampling—are employed to represent the challenging scenarios with a concise training environment set that is feasible for an RL agent to learn from. We denote the pipelines with three environment synthesis approaches as SES-GAN, SES-PCA and SES-RS respectively. We evaluate the three SES pipelines in Matterport, a dataset of many 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 [12], and the best pipeline SES-GAN generates more representative training environments and enables better deployment in Matterport than the pipelines with other synthesis approaches.

II. RELATED WORK

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 to develop in the first place and to adapt to different environments. Moreover, when encountering an environment in which a robot has 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 require comparatively less engineering effort, and learn navigation behaviors purely from data [13], e.g., from expert demonstrations [5], [14] or from trial and error [6], [15]. However, these approaches often lack safety guarantees and explainability, as provided by their classical counterparts. Therefore, roboticists have also investigated more structured ways of integrating learning with classical navigation, such as learning local planners [16]–[18], terrain-based cost functions [19], planner parameters [10], [11], driving styles [20], or social norms [7]. Their success notwithstanding, 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 [21] and learning approaches' poor generalizability to real-world scenarios. It improves navigation by synthesizing training environments from real-world deployment experiences.

B. 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 [22], [23]. 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 [24], or learns an action transformer that induces simulator transitions that more closely match the real world [25], [26].

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). 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 deployments in the real world usually entail 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 $p_{\text{real}}(T)$. SES generates a new navigation domain p_{SES} in simulation that, with a limited number of tasks, models the distribution of tasks in p_{real} so that the navigation performance of policies trained in p_{SES}

is likely to be strong in the real-world navigation domain p_{real}. SES uses APPLR [10] 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-based approach to 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 k, SES records the sensory input x_k, the robot's position s_k in the world frame, and the number of suboptimal or failed navigation behaviors c_k (negative velocity command is used as the indication of failure in this paper).[2] Each deployment will return a trajectory $\tau = (x_0, s_0, c_0, ..., x_k, s_k, c_k, ..., x_K, s_K, c_K)$ with K being the total number of steps during a deployment. 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) at which 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 $\{\eta_j\}_{j=1}^{J}$.

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 $\{e_j, \bar{c}_j\}_{j=1}^{J}$. Each environment e_j 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 trajectory segmentation and environment construction process. Each environment is represented as a 30×30 binary matrix with each element denoting the occupancy state of a 1.5m×1.5m space. 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 under roughly the same magnitudes of return and episode lengths across different environments. Alg. 2 summarizes the above process of generating an environment set that represents the real-world navigation domain distribution.

[2]The navigation policy used in this paper is based on a classical navigation system DWA, in which, suboptimal recovery behavior assigns negative linear velocity to the robot

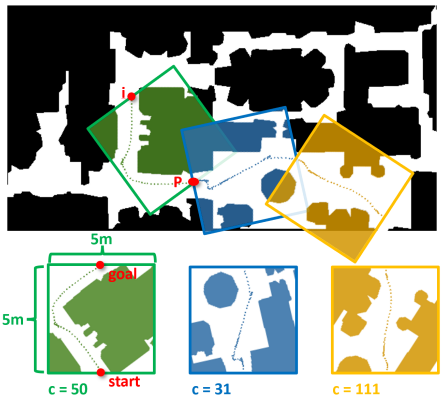

Fig. 2: An example of trajectory segmentation and environment construction: dashed line denotes the actual trajectory of the robot. Three scenarios are segmented and constructed as navigation environments at the bottom. Red dots mark the positions of the start and goal in the environments. The numbers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

A "raw" navigation domain $p_{\text{raw}} = U(\{e_i\}_{j=1}^{J})$ can be defined as a uniform distribution over all these constructed navigation environments. However, it is likely that p_{raw} includes many trivial navigation tasks (e.g., open spaces), whereas, efficient training should focus on the challenging navigation tasks. We use a threshold of 50 suboptimal behaviors within a scenario η as an indication of easy and challenging environments. The threshold of 50 is empirically picked such that the trivial scenarios with mostly empty spaces will be filtered out while not losing the challenging scenarios. Those challenging environments, $\{e_j, \bar{c}_j | \bar{c}_j > 50\}_{j=1}^{J}$, form a set for further environment synthesis. We denote the navigation domain with only challenging navigation environments as $p_{\text{chal}} = U(\{e_j | \bar{c}_j > 50\}_{j=1}^{J})$. Our empirical results indicate that training only on the navigation tasks with challenging scenarios instead of all the real-world scenarios is essential for improving real-world deployments.

C. Environment Synthesis

While simply using all the challenging environments $\{e_j | \bar{c}_j > 50\}_{j=1}^{J}$ can precisely represent p_{chal} in real world, 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) [27]; (2) K-means clustering with Principal Component Analysis (PCA) [28]; and (3) random sampling. We describe GAN, K-means clustering with PCA, and random sampling as follows.

GAN learns a generator's distribution p_g over all the possible environments $e \in \{1, 0\}^{30\times30}$ to match a uniform

distribution p_{chal} over the challenging 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_{chal} rather than p_g. D is trained to maximize the probability of assigning the correct label to both challenging 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 $V(G, D)$:

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

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 dependent on the training budget of the RL algorithm.

K-means clustering with PCA first reduces the dimensionality of the original environment representation (30×30 binary matrix) 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). Then, K-means is employed to cluster the environments in the reduced space into m clusters and sample n environments from each cluster to select $M = m \times n$ representative environments in the challenging environment set. We use the principle components to reconstruct the representative environments and use them as the training set.

Random sampling simply uses $p_{\text{SES}} = p_{\text{chal}}$ the uniform distribution over all the challenging environments. Then, M environments are drawn from p_{SES} to form the training set.

The synthesized navigation domain is denoted as p_{SES}. Finally, an APPLR policy π is trained in p_{SES}. 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 training.

Algorithm 1 Self-supervised Environment Synthesis

Require: Original policy π_0, real-world navigation domain distribution p_{real}, number of training environments M

Raw navigation domain $p_{\text{raw}} \leftarrow$ Real-world Navigation Domain Extraction based on π_0 and p_{real} (Sec. III-B)
Synthesized navigation domain $p_{\text{SES}} \leftarrow$ Environment Synthesis based on p_{raw} (Sec. III-C)
$\{e_i\}_{i=1}^M \leftarrow$ Sample M environments from p_{SES}
$\pi \leftarrow$ Initialize with π_0, then train on $\{e_i\}_{i=1}^M$
return π

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 challenging 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 use Matterport [1], a set of high-fidelity simulation environments as a surrogate for a 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 2 Real-world Navigation Domain Extraction

Require: Original policy π_0, real-world navigation domain distribution p_{real} and total number of deployments N.

$E = \emptyset$
for $n \in \{1, ..., N\}$ **do**
  $T_n \sim p_{\text{real}}(T)$
  Trajectory $\tau_n \leftarrow$ deploy π_0 on T_n
  Initialize step $i = 0$ and the scenario set $\eta = \emptyset$
  for $x_k, s_k, c_k \in \tau_n$ **do**
    $\eta \leftarrow \eta \cup \{(x_k, s_k, c_k)\}$
    if Euclidean(s_k, s_i) > 5 **then**
      $e \leftarrow$ Construct Environment from η
      $\bar{c} = \sum_i^k c_i$
      $E \leftarrow E \cup \{e, \bar{c}\}$
      $i = k$, $\eta = \emptyset$
    end if
  end for
end for
return E

A. Deployment in Matterport Navigation Domain

We employ SES on a simulated ClearPath Jackal differential-drive ground robot. The specifications of the robot are kept the same as in APPLR [10], 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 [12]. The dataset contains 300 simulated navigation environments randomly generated by Cellular Automata. Even though those environments are designed to be diverse enough to cover different difficulty levels from relatively open spaces to highly-constrained ones, when a controller learned on these environments is deployed in Matterport, the randomly generated environments in BARN are found not to cover some specific obstacle shapes and arrangements encountered by the robot, and therefore lead to the failures at the deployment time. In Fig. 3, (a)-(c) show three example BARN environments, and (d) shows a Matterport environment where Jackal will be deployed. This realistic Matterport environment may contain challenging navigaiton scenarios that are 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

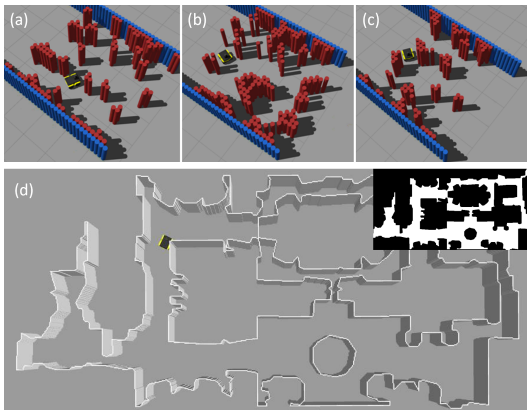

Fig. 3: Examples of training and deployment environments. (a), (b), (c): example environments from the BARN dataset; (d): a top-down view of a Matterport environment created from the occupancy map shown at the top-right corner.

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 100 times to navigate between the starting and goal locations randomly sampled from all the navigable start-goal pairs in each Matterport environment, which accounts for 5000 deployments in total. We construct 16528 navigation scenarios from the segmented trajectories using the method described in Sec. III-B as p_{raw} the distribution of the Matterport navigation domain. Then 3769 challenging environments are selected based on a threshold 50 of the negative linear velocity count, which forms p_{chal}. Fig. 4 (left) demonstrates some examples out of the 3769 challenging environments. Each of the environments is represented as a binary matrix of size 30×30. Note that training in p_{chal} with all 3769 environments is not computationally practical due the limit of CPUs that run the environments in parallel and the budget of total number of training steps. In the meantime, realistic large-scale robot deployments in the real world (in contrast to our surrogate) may generate orders of magnitude more challenging scenarios.

Environment synthesis: given the extracted Matterport navigation domain p_{chal}, the environment synthesis method

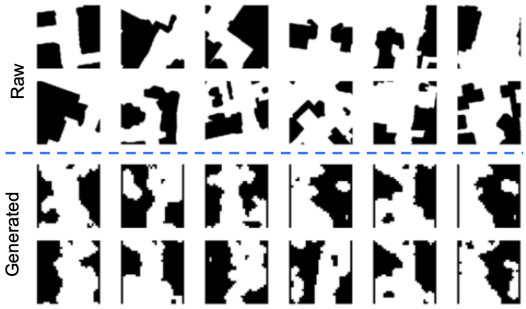

Fig. 4: Examples of raw challenging environments (top) and generated environments (bottom).

described in Sec. III-C generates 100 training environments (the maximum number of environments that can be trained in parallel in our training pipeline) by using the challenging environments as an image dataset to train a GAN. More specifically, 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 output vector is further rendered into a 30×30 binary matrix using a threshold of 0. 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 [29]. Fig. 2 (right) shows samples of generated images drawn from the generator after training. The generator is queried 100 times to generate synthesized navigation domain p_{SES} with 100 training environments, which is in the same order of magnitude as the BARN dataset. We leave the investigation of how SES performs with different numbers and sizes of training environments as future work. To train the new policy π that solves the navigation in challenging Matterport environments, we use the same set of hyper-parameters as when training π_0 except for a smaller training step of one fourth of the original 4M training step, because the pre-trained policy π needs fewer training step to converge.

TABLE I: Average time costs and success rates

	Time cost (s)	Success rate (%)
SES-GAN	27.23	85.0
SES-PCA	32.21	80.6
SES-RS	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

C. *Test Results*

We train the new APPLR policy π in p_{SES} as created by the three SES methods: SES-GAN, SES-PCA, and SES-RS.

We compare the methods with four baselines: the APPLR policy learned from scratch on the training environments generated by GAN, the original APPLR policy π_0, and vanilla DWA with two sets of static hyper-parameters, one having default parameters and the other having a two times higher maximum linear velocity and sampling rate. 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 of initializing the policy with the original 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 navigable start-goal pairs that have at least 5-meter start-goal distances , 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. [3] The policy trained by SES-GAN achieves the best performances of a 27.23s average time cost and a 85% success rate. SES-PCA 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 SES-GAN which indicates the importance of initializing the model with the pre-trained policy π_0. The policy trained by SES-RS performs comparably to the original policy that is not trained on the Matterport environments at all, which indicates the poor representation of the challenging Matterport environments with random sampling. While DWA *slow* achieves an acceptable success rate of 75.5%, it takes much longer (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 over 20 independent trials in Fig. 5. The policies trained by SES-GAN and SES-PCA, in general, complete the navigation task more quickly and achieve higher success rate. Among five test environments, the SES-GAN policy achieves the lowest time cost in three environments and the highest success rate in all five. SES methods tend to show larger improvements in the more difficult environments (1 and 2), but also larger performance variance.

D. Physical Experiment

To verify that SES is potentially applicable to real-world robots, we deploy the pre-trained APPLR policy on a physical Jackal robot. The pre-trained policy is deployed in a real-world environment and a map of the environment is built during deployment using `gmapping` [30] (shown in Fig. 6). Using the same criteria for suboptimal behavior as the Matterport experiments, the robot identifies two challenging scenarios shown by the orange and yellow areas in the map.

[3]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.

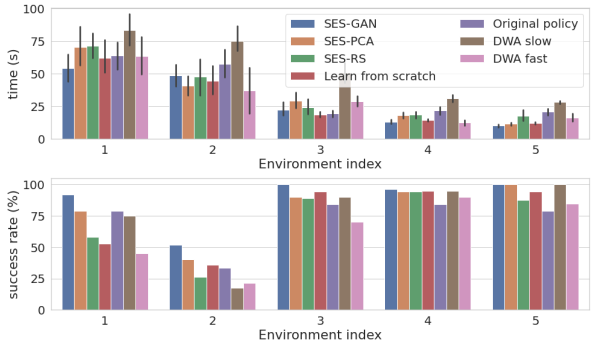

Fig. 5: The average time cost (top) and the success rate (bottom) of 5 policies tested in 5 held-out environments.

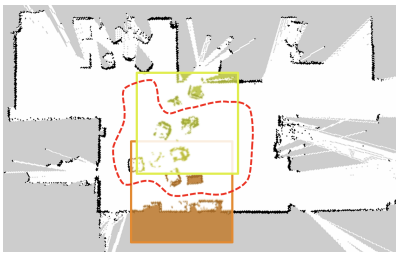

Fig. 6: Real-world navigation path (red dashed line) and map. The orange and yellow areas mark the two challenging scenarios identified during the real-world deployment.

Considering the limited scale of our real-world deployment, we directly construct two synthetic environments which are identical to the challenging real-world scenarios (instead of using a GAN) and fine-tune the pre-trained policy in them. We denote the policies before and after fine-tuning as *original policy* and SES respectively. Table II reports the time cost averaged over 5 trials of both policies to traverse the two difficult real-world scenarios, where SES achieves significant improvement compared to the *original policy*.

TABLE II: Average time cost of the tested policies in real-world challenging scenarios

	Scenario 1	Scenario 2
Original policy	18.5 ± 7.6 (s)	38.4 ± 9.8 (s)
SES	11.3 ± 0.9 (s)	15.6 ± 1.0 (s)

V. CONCLUSION

This paper introduces 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, SES can successfully identify challenging navigation scenarios during the deployment in high-fidelity Matterport environments and synthesize representative environments to be added to the training distribution. SES has also been demonstrated in a small-scale real-world deployment.

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] H. Ma, J. S. Smith, and P. A. Vela, "Navtuner: Learning a scene-sensitive family of navigation policies," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 492–499.

[9] S. Daftry, N. Abcouwer, T. Del Sesto, S. Venkatraman, J. Song, L. Igel, A. Byon, U. Rosolia, Y. Yue, and M. Ono, "Mlnav: Learning to safely navigate on martian terrains," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5461–5468, 2022.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] G. Mordido, H. Yang, and C. Meinel, "Dropout-gan: Learning from a dynamic ensemble of discriminators," *arXiv preprint arXiv:1807.11346*, 2018.

[30] B. Gerkey, "Ros wiki gmapping," http://wiki.ros.org/gmapping, 2018.