EmbodiedSplat: Personalized Real-to-Sim-to-Real Navigation with Gaussian Splats from a Mobile Device

Supplementary Material

A. Real-world Deployment on Stretch Robot

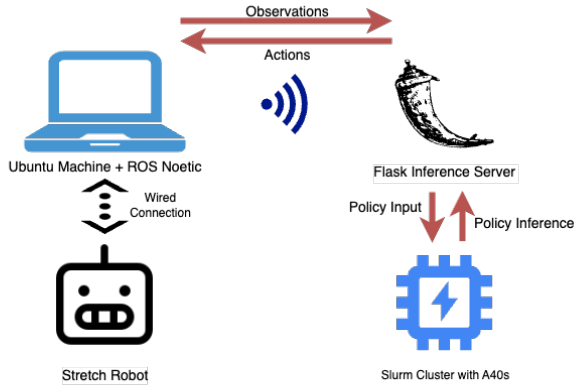

Figure 13. Framework for real-world evaluation on a Stretch robot.

Since the stretch robot is not equipped with GPUs, we develop a framework for deploying a policy on a remote cluster, and performing inference using POST requests from the robot. An overview of this arrangement is shown in Fig. 13. Specifically, we create a Flask server that runs on a remote compute node, port-forwarded to an Ubuntu 20.04 machine with ROS noetic installed. The same machine is connected to a Stretch robot via ethernet connection. We use the `home-robot` [43] repository as framework to convert discretized actions from the policy into continuous space using the “StretchNavigationClient”. A script running on the machine consumes the ROS topics for image observations from the robot, and passes this along with the goal image to the policy and receives the next action based on these observations.

For evaluation in real-world, we use random start-and-goal locations, place the Stretch robot at the goal location, and capture a goal image. Then, we mark these locations using colored masking tape on the floor. The policy is deployed with the goal image passed to the agent during the episode as above, keeping the start location and pose similar for each evaluation. The episode stops when the agent outputs a STOP action or the maximum number of steps (100 in our case) are reached. Finally, the distance of the base is measured from the the goal location using a measuring tape, and the episode is marked successful if the agent reaches within $1m$ of the goal location before stopping. The goal images used are shown in Fig. 15.

Figure 14. Hello Robot Stretch in `lounge` scene

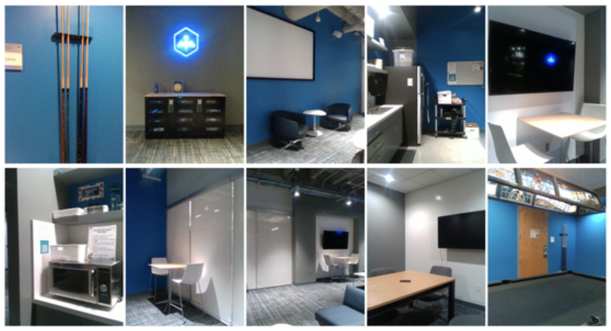

Figure 15. Goal images used in `lounge` scene

B. Generating ImageNav episodes

To generate ImageNav episodes, we build upon Habitat’s [39, 44] scripts for PointNav generation. During training, the PointNav goal locations are paired with an Imagegoal sensor to retrieve the image corresponding to the goal location prior to the start of the episode.

Habitat [39, 44] uses navmesh islands to define navigable areas in simulation. For episode generation, we leverage the stretch robot parameters [46] to compute the navmeshes.

Valid start and goal locations are sampled during episode generation, following the approach in [46]. The validity checks ensure that the goal is reachable, the distance from the start location to the goal is non-zero, and the navmesh island radius exceeds $2m$, thereby avoiding navigation on objects like beds or tables.

C. Examples for DN and POLYCAM rendering

Fig. 16 and Fig. 17 shows the rendering differences for the images in Habitat-Sim [39] for DN and POLYCAM meshes. DN meshes have diffused and darker colors, and some holes for regions where capture was not sufficient. In contrast, polycam produces a smoother and more realistic mesh.

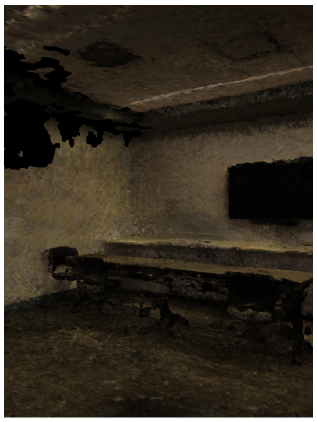
Figure 16. DN (`conf_b`)

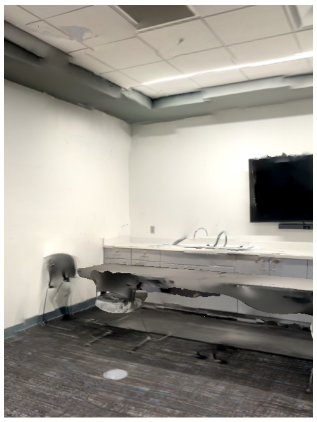
Figure 17. POLYCAM (`conf_b`)

D. Expanded Related Work

D.1. 3D Scene and Mesh Reconstruction

Recent advancements in 3D scene reconstruction have led to the development of several notable approaches. Neural Radiance Fields (NeRF) [35] and subsequent methods [4, 19, 36] have focused on training and improving neural scene representation techniques. The foundational 3D Gaussian Splatting (GS) method [24] has inspired a variety of extensions. DN-Splatter [49] improves reconstruction quality by incorporating depth and normal regularization with monocular networks. Gaussian Surfels [12] introduces a technique for flattening 3D Gaussians into 2D surfels, resulting in enhanced mesh reconstruction. GS2Mesh [51] incorporates real-world knowledge through stereo-matching to generate smoother meshes. SuGaR [15] proposes a fast mesh extraction method using surface-aligned Gaussian splatting.

D.2. 3D Scene Representation in Robotics

Several studies have explored the use of neural scene representations [33, 54] and 3D Gaussian Splatting (GS) for robotic tasks in simulation. SplatGym [40] presents a neural simulator for training robotic control policies using Gaussian Splatting. BEINGS [34] leverages 3D Gaussian Splatting as a scene prior to predict future observations and refine navigation strategies. A concurrent work, SOUS-VIDE [30], uses a simple drone dynamics model within a high visual fidelity 3DGS reconstruction for training an end-to-end policy and show sim-to-real transfer. We note that our work is a complementary effort focused on image-goal navigation, out-of-domain generalization, the effects of reconstruction quality and training strategies.

Recent research has also emphasized the application of neural scene representations for real-world robotic applications [7, 10, 22]. Object-Aware Gaussian Splatting [28] proposes semantic and dynamic 3D representations for robotic manipulation. RL-GSBridge [52] introduces a mesh-based 3D Gaussian Splatting method for zero-shot sim-to-real transfer in manipulation tasks. LEGS [58] and POGS [59] distil semantic features from foundation models into GS to enable downstream navigation and manipulation applications. While these studies investigate Gaussian Splatting for sim-to-real manipulation, our work focuses on using it for personalized sim-to-real Image-Goal Navigation. For additional methods utilizing Neural Fields and Gaussian Splatting in robotics, we refer to the comprehensive surveys [20, 60] which examines the applications of Neural fields in robotics.

E. Mesh Processing using Blender

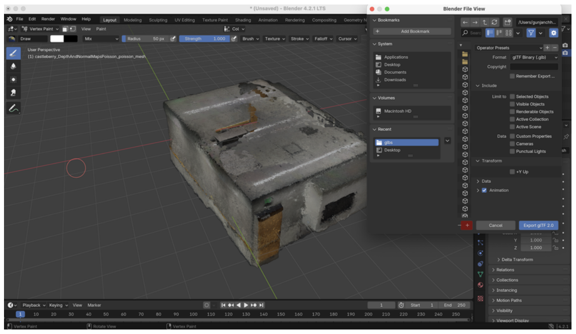
Figure 18. Converting the `.ply` files generated by DN-Splatter [49] to `.glb` files using Blender. Transform `+Y Up` is kept unchecked following conventions of Habitat-Sim [39].

DN-Splatter [49] gives us Poisson reconstructed meshes from the Gaussian Splats in `.ply` format. In order to load these meshes in Habitat-Sim [39], we need to convert these to `.glb` format. In order to do so, we import the `.ply` in Blender and then export them as `.glb` files while ensuring that `+Y Up` transform is unchecked, following the conventions used in Habitat-Sim.

F. Episode Statistics for Generated Datasets on `Captured` Scenes

Tab. 2 presents the shortest path lengths (Min, Max, and Avg.) for both training and validation episodes across the different types of GS-based scenes used in our work.

Notably, the scale of the `Captured` scenes is comparable to that of the `MuSHRoom` [42] scenes, suggesting a similar range of complexity in terms of environment size and

Scene Type	Scene	Train			Val		
		Min	Max	Avg.	Min	Max	Avg.
MuSHRoom	activity	7	127	59.623	17	120	59.82
	sauna	5	60	34.615	7	57	34.31
	honka	6	61	33.563	16	63	34.16
Captured	lounge	8	99	51.431	18	96	55.3
	classroom	7	78	35.161	11	70	32.39
	conf_b	5	42	20.766	5	34	19.77
	conf_a	5	46	19.996	6	43	19.92

Table 2. Shortest path lengths for episodes across MuSHRoom [42] and Captured scenes.

structure. In future, we will attempt to collect apartment-scale scenes which require considerably larger number of steps (∼200).

Within the Captured scenes, there is variation in scale, with certain scenes such as lounge having much larger path lengths compared to others like conf_b or conf_a, which are conference rooms.

Both GS training and agent training is likely to be influenced by this scale variability, as the agent must learn to navigate environments that may range from relatively simple, smaller spaces to more expansive, complex ones.

G. Depth and Normal Encoder Selection for DN-Splatter

Since we do not have ground-truth meshes for our Captured scenes, we use all 10 MuSHRoom [42] scenes as a proxy for evaluation of different depth and normal encoders towards 3D scene reconstruction.

We evaluate various depth encoders using monocular depth predictions and assess their reconstruction performance based on several metrics following MuSHRoom [42]: accuracy (Acc), completion (Comp), Chamfer distance (C-L1), normal consistency (NC), and F-score. The depth scale for each encoder is defined by the ground truth (GT) from an iPhone, with a transform (scale and bias) learned via gradient optimization to convert the monocular depths from the encoders into the appropriate scale.

As shown in Tab. 3, the GT depth from iPhone provides the best performance across most metrics, achieving the highest normal consistency (0.815) and F-score (0.748). Among the tested depth encoders, GT consistently outperforms others, confirming its robustness for reconstruction tasks. Since the GT depth encoder performed the best overall, we selected the same for our subsequent evaluations for normal encoders. As shown in Tab. 4, the Metric3D-v2 [16] encoder achieves the highest F-score (0.752) when used with GT depth, outperforming other normal encoders - DSINE [3] and Omnidata [14]. Therefore, for 3D reconstruction of our Captured scenes, we use the GT-depth and Metric3D-v2 normals.

H. Agent, Training, and Evaluation Details

Agent: We employ the Hello Robot Stretch robot embodiment as our agent, following the setup in [46]. In the Habitat Simulator, the agent is modeled as a cylinder with a height of 1.41m and a radius of 0.3m. The RGB camera sensor is positioned at a height of 1.31m from the ground and is vertically aligned. The sensor outputs images with a resolution of 640×480 (H×W) and a horizontal field of view of 42°. The goal sensor is configured with identical parameters to the RGB sensor. During training, the agent's rotation at the goal location is randomly sampled.

Training and Evaluation: We train our agents using DD-PPO [50] with ImageNav reward, with each environment generating 64 frames per rollout. The training process includes 2 PPO [45] epochs, each consisting of 2 mini-batches. Following [46], we use the AdamW optimizer [29] with a weight decay of 10^{-6} and a learning rate of 2.5×10^{-4}, unless stated otherwise. The visual encoder is kept unfrozen during training and data augmentation is applied. The visual encoder learning rate is set to 1.5×10^{-6} for HM3D and other settings, and 1.5×10^{-5} for HSSD. The goal and observation visual encoders share the same weights. The number of environments per GPU is set to 10 for HM3D/HSSD, and 8 for Captured scenes due to computational constraints. Policies are trained to convergence across all training setups and datasets. Checkpoints are saved approximately every $\sim 3M$ steps and evaluated on the validation sets of the corresponding datasets. The best checkpoint is selected based on the highest success rate (SR) achieved on the validation set. For training, we utilized 16 NVIDIA A40 GPUs per policy per dataset. For simulation-based evaluations, we utilize a single A40 GPU with one environment for Captured scenes and MuSHRoom scenes, and 20 environments for HM3D/HSSD. For real-world evaluations, we employ a single environment and a single A40 GPU. For details on real-world setup, please refer to Appendix A.

Visual Encoder: The observation and goal images are resized to 160×120 before being input into the VC-1-Base visual encoder [32]. Patch embeddings are

Depth Type	Acc ↓	Comp ↓	C-L1 ↓	NC ↑	F-score ↑
DepthAnything-v2 (Indoor) [57]	0.054	0.090	0.072	0.800	0.688
GT	**0.047**	0.090	**0.068**	**0.815**	**0.748**
Metric3D-v2 [16]	0.052	0.089	0.071	0.804	0.700
UniDepth-v2 [37]	0.049	**0.087**	**0.068**	0.813	0.719
ZoeDepth [5]	0.061	0.091	0.076	0.780	0.621

Table 3. Average metrics for `MuSHRoom` Scenes computed against Faro-Scanner ground-truths for different depth types. The normal encoder used is Omnidata [14]. Ground truth from iPhone leads to the highest F-scores.

Normal Type	Depth Type	Acc ↓	Comp ↓	C-L1 ↓	NC ↑	F-score ↑
DSINE [3]	GT	0.046	0.090	0.068	0.815	0.750
Metric3D-v2 [16]	GT	0.046	0.090	0.068	**0.818**	**0.752**
Omnidata [14]	GT	0.047	0.090	0.068	0.815	0.748

Table 4. Average metrics for `MuSHRoom` scenes computed against Faro-Scanner ground truths for different normal encoders. The depth used is the ground truth from iPhone. Metric3D-v2 outperforms other normal encoders.

processed through a "compression layer" [56], which comprises of a 2D convolution followed by group normalization, to generate the final embeddings.

Policy: The policy is implemented as a 2-layer LSTM that takes as input the previous action embedding, visual observation embeddings, and goal image embeddings following [46]. It outputs one of the following discrete actions: `MOVE_FORWARD`, `TURN_LEFT`, `TURN_RIGHT`, or `STOP`.

Reward Function: The reward function is adapted from [46, 56], using the same hyperparameters as in [46]: success weight $c_s = 5.0$, angle success weight $c_a = 5.0$, goal radius $r_g = 1.0$, angle threshold $\theta_g = 25°$, and slack penalty $\lambda = 0.01$. The collision penalty is defined as $c_{coll} = 0.03$.

The full reward function is detailed in Eq. (1).

$$\begin{aligned} r_t = c_s &\times ([d_t < r_g] \wedge [a_t = \text{STOP}]) \\ &+ c_a \times ([\theta_t < \theta_g] \wedge [a_t = \text{STOP}]) \\ &+ \left(\hat{\theta}_{t-1} - \hat{\theta}_t\right) \\ &+ (d_{t-1} - d_t) - \gamma \\ &- c_{\text{coll}} \times [\text{collision = true}] \end{aligned} \tag{1}$$

I. Real-world metrics for successful episodes

Tab. 5 shows the distance-to-goal (in centimeters) and the number of steps taken on average for the successful episodes for each type of policy we deploy in the real world. We note that some of these average values may not include enough samples to be statistically significant. Therefore, we refrain from making conclusions about which policy is more efficient based on these, and provide the values for book-keeping purposes.

Policy	D2G (cm)	#Steps
HM3D	44.20	45.60
HM3D-FT-Poly	48.29	58.14
HM3D-FT-DN	29.64	52.86
HSSD	98.50	38.00
HSSD-FT-Poly	37.10	54.00
HSSD-FT-DN	40.87	36.25

Table 5. Average distance to goal and number of steps in the real-world `lounge` scene for successful episodes for each policy.

J. Simulation results on additional real-world scenes

In Tab. 6, we present additional results for HM3D zero-shot and fine-tuned policies across several scenes. Five of these are from the `MuSHRoom` dataset—`koivu`, `classrm2`, `kokko`, `coffeerm`, and `vr_rm`. We also include one real-world scene, `conf_c`, that we captured using our methodology. Fine-tuning consistently improves performance across all mesh types. The improvements are modest in smaller scenes (e.g., `koivu`, `coffeerm`), where the zero-shot policy already performs well. In contrast, larger scenes (e.g., `conf_c`, `vr_rm`) benefit significantly from fine-tuning.

We also visualize the relationship between HM3D zero-shot success rate, validation PSNR, and average shortest path distance in Fig. 19. We observe a consistent trend with the previous results: the zero-shot success rate decreases as the size of the scene increases, and improves as the validation PSNR of the trained Gaussian Splat (GS) increases.

An exception is `vr_rm`, which appears as an outlier in the `MuSHRoom` dataset and warrants further investigation. Aside from this case, the remaining scenes from `MuSHRoom` generally show improved zero-shot performance as PSNR increases.

Scene	Mesh	HM3D-ZS	HM3D-FT
koivu	DN	0.87	0.98
classrm2	DN	0.61	0.97
kokko	DN	0.66	0.99
coffeerm	DN	0.90	1.00
vr_rm	DN	0.39	0.99
conf_c	DN	0.16	0.84
	Poly	0.50	0.98

Table 6. Validation success rates on additional scenes from MuSHRoom and Captured datasets.

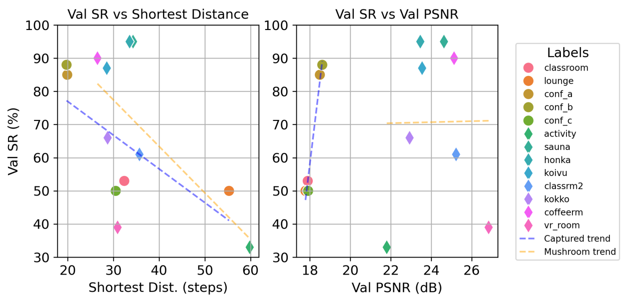

Figure 19. **HM3D zero-shot validation success rates on DN meshes vs. validation GS PSNRs vs. average shortest distances of validation episodes.** The zero-shot SR is inversely correlated with scene scale and directly correlated with GS validation PSNR. vr_rm is an outlier in the MuSHRoom dataset and requires further analysis.

K. Failure modes for HM3D zero-shot on lounge scene in simulation

To assess the role of visual fidelity, we analyze zero-shot HM3D policy trajectories and identified failure modes in the lounge scene using both DN and POLYCAM meshes (Tab. 7) in simulation. We found significant increase in "Maximum Steps Reached" failures with the DN mesh, indicating low visual fidelity prevents the policy from matching goal images— a key limitation of the pre-trained policy.

Termination Reason	**POLYCAM**	**DN**
Target Reached (success)	74	55
Early Stop (Goal Not Visible)	3	5
Early Stop (Goal Visible)	11	16
Early Stop (Similar Goal Visible)	4	5
Maximum Steps Reached	8	19

Table 7. Sim failure analysis for HM3D-ZS on lounge.

L. Overfitting vs. Fine-Tuning

In this section, we clarify why the overfitting results shown in Fig. 9 may appear stronger than the fine-tuning results in Fig. 6. Fine-tuning requires significantly fewer steps because the pre-trained policy already captures general navigation behaviors. We also deliberately limit the number of fine-tuning steps to preserve the policy's ability to generalize to real-world environments—leveraging knowledge acquired during large-scale pre-training across diverse scenes.

This pre-training not only enables better generalization in the real world (see Fig. 8) but also substantially reduces overall training time, which is critical for rapid deployment.

In contrast, our overfitting experiments involve training policies exclusively on simulated versions of specific real scenes. These policies typically achieve 80–90% success within 20–30M steps, but their performance is limited to the training environment. For completeness, we early stop around 100M steps, although most gains saturate earlier.

Tab. 8 reports the best validation success rates of overfitted policies at 20M steps. Despite strong performance in simulation, these policies fail to generalize to real-world settings due to factors such as lighting variations, sensor noise, and actuation inaccuracies.

Scene	DN val. SR.	Poly val. SR.
lounge	0.74	0.74
classrm	1.00	0.85
conf_a	0.99	1.00
conf_b	1.00	1.00
honka	1.00	—
sauna	0.99	—
activity	0.79	—

Table 8. Overfitted validation success rates after 20M steps.