Gibson Env: Real-World Perception for Embodied Agents
Supplementary Material

Fei Xia*[1] Amir R. Zamir*[1,2] Zhiyang He*[1] Alexander Sax[1] Jitendra Malik[2] Silvio Savarese[1]
[1] Stanford University [2] University of California, Berkeley
http://gibson.vision/

Abstract

The following items are provided in the supplementary material:

1. *Details of point cloud interpolation and view selection in view synthesis, and details of network architecture and hyper parameters.*
2. *More information about semantic modality in our environment.*
3. *Additional view synthesis qualitative results.*
4. *RL training details and additional RL results in our environment.*

1. View Synthesis Details

1.1. Details of Point Cloud Rendering

Here we provide more details about the point cloud interpolation and view selection part of view synthesis. For a view v_j that we want to synthesis, we get the rendered point cloud from the nearest k views $v_{j1}, v_{j2}, \ldots, v_{jk}$ based on Euclidean distance. Denote the reprojected point clouds as $p_{j1}, p_{j2}, \ldots, p_{jk}$. Assuming the target resolution is $w \times h$, each point cloud is a set of projected points $\{(x, y)\}$, with $0 \leq x \leq w$ and $0 \leq y \leq h$. We do two operations on each project point cloud. First, we run a kernel density estimator on the point cloud with a Gaussian kernel. This provides information about the density of points from a certain view at a certain region. Second, for each grid point in x, y-plane we query the nearest 4 points and did a bilinear sampling in an irregular polygon. Denote the density of point cloud image of view k as $d_{jk} \in R^{w \times h}$. The weight for view k is $w_{jk} = d_{jk}/(\sum_i d_{ji})$. Denote the interpolated image from source view k as $I_{jk} \in R^{w \times h \times 3}$, then the final output of view selection and interpolation is $I_j = \sum_i w_{ji} \circ I_{ji}$. We implement these steps in CUDA and thus they have small effects on point cloud rendering speed. The benefit of the point cloud interpolation and view selection is that they create a smooth transition across point clouds from different source views, making the imperfection easier to fix by Neural Network Filler f.

1.2. Details of Training Neural Network Filler

Network Architecture. The details of the network architecture are shown in Table 1. We use a combination of convolutional layers, deconvolutional layers and dilated convolutional layers. Batch normalization is applied to each output feature map of all types of convolutional layers. We use leaky ReLU with slope 0.1 as activation function for all layers.

Identity initialization. For image processing or filtering, [11] advocates using adaptive normalization for identity initialization, where they also design kernels to force identity mapping. Empirically, we find that using a stochastic version that ensures an overall identity function but preserves the diversity of Gaussian random initialization works better and achieves faster convergence. We initialize half of the weight with Gaussian random and *freeze* them and optimize the rest with back propagation so the network outputs the same input image. After convergence, the weights are our stochastic identity initialization.

Hyperparameter setting. For the network architecture, the number of kernels of the network is configurable with a parameter n_f. In the experiments, we use two settings where n_f is either 24 or 128. For domain adaptation and view synthesis experiments we use $n_f = 128$. For RL training we use $n_f = 24$ to reduce RL training time. For the loss, there are two hyperparameters to tune. The first is λ_l, which is the coefficient for each layer of VGG feature map. We normalize it the number of elements in the feature map, i.e. $\lambda_l = 1/n_l$. The second parameter is the coefficient for color match loss γ. We set it to 0.05 across the experiments.

*Authors contributed equally.

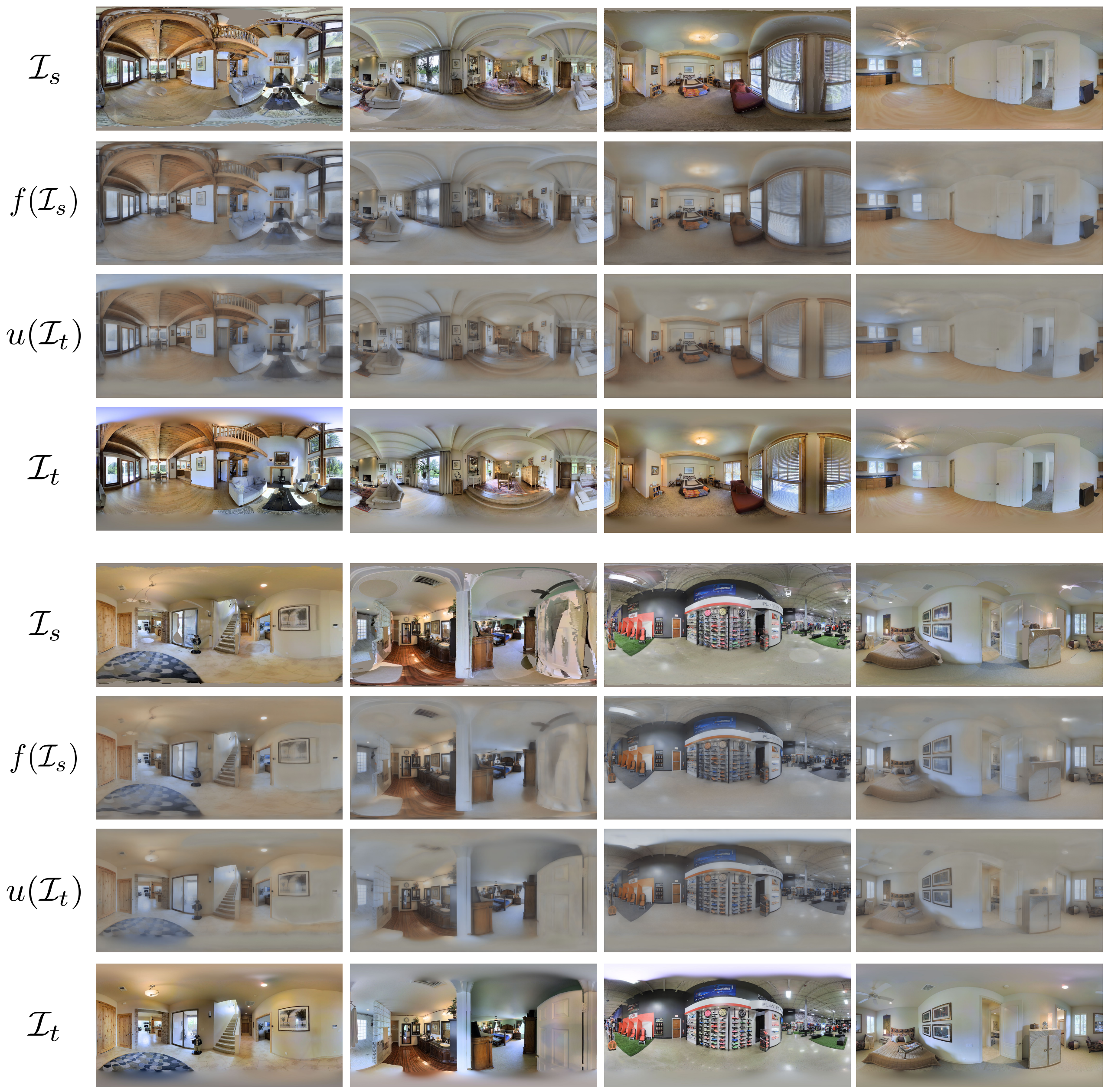

Figure 1: Additional view synthesis results. We provide additional view synthesis and domain adaptation results. The format is similar to Fig. 6 in the main paper. The results have high resolution and are very rich in details, so we encourage readers to zoom in to see details.

2. Details about Semantic Modality

Some models in our system are semantically annotated. For those models, we are able to output a semantic segmented frame. Examples are shown in Fig. 2. We provide semantic labels for 13 classes, including floor, ceiling, wall, beam, window, column, door, table, chair, bookcase, sofa, board, and clutter.

3. Additional Experiments

3.1. Additional View Synthesis Results

We provide additional view synthesis results to provide a more thorough understanding of the qualitative results. The results are shown in Fig. 1.

Type	Kernel	Dilation	Stride	Output
conv.	5×5	1	1×1	6
conv.	5×5	1	2×2	n_f
conv.	3×3	1	1×1	n_f
conv.	5×5	1	2×2	$4n_f$
dilated conv.	3×3	1	1×1	$4n_f$
dilated conv.	3×3	1	1×1	$4n_f$
dilated conv.	3×3	2	1×1	$4n_f$
dilated conv.	3×3	4	1×1	$4n_f$
dilated conv.	3×3	8	1×1	$4n_f$
dilated conv.	3×3	16	1×1	$4n_f$
dilated conv.	3×3	32	1×1	$4n_f$
dilated conv.	3×3	1	1×1	$4n_f$
dilated conv.	3×3	1	1×1	$4n_f$
deconv.	4×4	1	2×2	n_f
conv.	3×3	1	1×1	n_f
deconv.	4×4	1	2×2	6
conv.	3×3	1	1×1	6
conv.	3×3	1	1×1	3

Table 1: Network Architecture. The network architecture is detailed in this table. We use three types of layers: convolutional layers(conv.), dilated convolutional layers(dilated conv.) and deconvolutional layers(deconv.). The network size is configurable with parameter n_f.

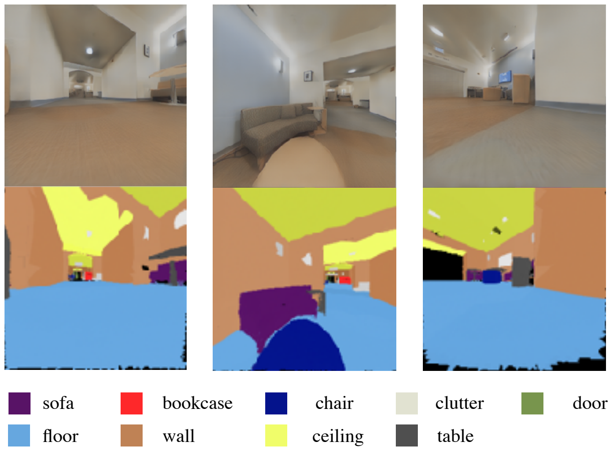

Figure 2: Sample frames with semantic annotation. Top: NN filled images; Bottom: Corresponding semantic labels of pixels.

3.2. Additional RL Experiment Setup

We used Gibson environment with default physical settings (gravity = $9.8m^2/s^2$, friction coefficient = 0.7) for our experiments. In Visual Obstacle Avoidance and Visual Navigation we used default discrete action space in Gibson for Husky robot, which translates four dimensional (left/right/forward/backward) control signal to four sets of predefined torques. In Visuomotor Control we used default continuous action space for Ant robot, which accepts eight real value torque signals.

At every time step, the nonviz sensor output received by the agent includes robot position, orientation, velocity, and distance and angle towards target. In Visuomotor Control experiment, this also includes the number of feed the Ant robot has in contact with the ground.

In Visual Obstacle Avoidance, Navigation and Visuomotor Control tasks, our primary reward for the agent is the change in potential (negative distance towards the target), specifically $-\frac{dL_{target}}{dt}$. In visual obstacle avoidance we added another distance term $c * L_{nearest}$, where $L_{nearest}$ is the length of the beam coming out from the front of the robot. This mimics the effect of front LiDAR and encourages the agent to move in unobstructed directions. We didn't use any additional term in our reward: alive score, collision, contact, etc. The agent learns to utilize it's DoF, minimize collisions and maximize its lifespan to reach the highest score.

3.3. Additional RL Experiment Results

Network Architecture and Training Details In order to account for the different input modalities of inputs, we use different Neural Network policies. For sensor-only RL agents, we used a policy with two identical networks for value and policy function: MLP with two hidden layers using $tanh$ activation. For perceptual agents, we used a sensor-vision policy that has the same MLP structure to process sensor input, and another CNN with two hidden layers using $relu$ activation to process camera input. We use a densely connected layer to combine MLP and CNN at the end. The implementations are based on OpenAI baselines.

We used Proximal Policy Optimization for training our RL policies. We used linearly decreasing learning rate, gamma of 0.99 and optimization batch size of 64. Our average episode length is 400 for Navigation and 60 for Visuomotor Control. We set time step per actor batch for these two tasks as 3000 and 2000. Since there is no ending condition for Obstacle Avoidance task, we set its time step per batch as 1024. Our reward curves for Obstacle Avoidance, Navigation, and Visuomotor Control stabilize after 100K, 600K and 700K time steps.

Visuomotor Control Results During training, we modified the ant robot's size and body shape to be reasonable based on the size of stairs in real life. We initialize the ant at random location at the midpoint of the stairs and train it to reach the bottom of the stairs using the sensor-only network, and sensor-vision fusion network with depth as perceptual input. Both agents know their relative location towards the target. To prevent the agent from learning suboptimal policies (e.g. rolling down the stairs), we terminate the agent when its leaning angle is more than 90 degrees.

Within the same amount of clock time, we trained the sensor-only agent for 4M frames, and perceptual agent for 900K frames, due to longer CNN processing time in fusion

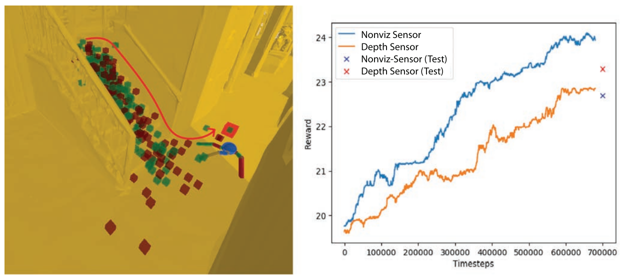

Figure 3: Ant climbing downstairs tasks reward curve. Left: ending position of sensor-only (red) and perceptual (green) agent. Right: reward curves and test score. Here we plot the reward curve over the first 700K frames when the agent starts to acquire stair climbing ability.

network. Sensor-only agent reached a final score of 34 and perceptual agent reached a final score of 25.6.

To evaluate how robust our learned visuomotor control policies are, we modified the target positions in test time. In Fig. 3 the target (light red cube) is moved from the bottom of stairs to behind the corner by $0.5m$. In order to reach it, the agent needs to first climb the stairs and then walk towards the target. Red and green small cubes show the death/finish locations of sensor-only and perceptual agents. Performance of sensor only agent drops significantly from 34.0 to 22.7 (-11.3), while the perceptual agent's performance slightly drops from 25.6 to 23.3. Failure cases for the sensor-only robot are: (1) being obstructed by the wall, (2) not approaching the target after going downstairs. This shows that policy learned by the sensor-only robot is limited in either only knowing to move in target direction, or only knowing to climb downstairs.