Supplementary Material: Collision Replay: What Does Bumping Into Things Tell You About Scene Geometry?

Alexander Raistrick
alexrais@umich.edu
Nilesh Kulkarni
nileshk@umich.edu
David F. Fouhey
fouhey@umich.edu

University of Michigan
Ann Arbor, MI

1 Summary

We provide extended and detailed versions of all salient model architectures, datasets, theoretical justifications and comparisions. First we provide model architecture, training and dataset details (Sections 2, 3, 4, 5 and 6). Next we provide additional analyses as referenced by our main paper, namely a theoretical analysis with reference to the Gambler's Ruin (Section 7), a comparison to L2 training loss (Section 8), and a comparison to Learning to Fly By Crashing [2] (Section 9). Finally, we show extended versions of Figures 4 and 5 from the main paper, with results from randomly selected test-set examples (Section 10).

Additionally, we provide a brief supplementary video, which is highlighted in Figure 1 and enclosed as an mp4 tested to be compatible with VLC media player. The video shows qualitative distance function results from our remote point prediction model. The results shown are from uncurated random trajectories in test-set environments unseen during training.

2 Remote Prediction Model Details

All remote point prediction models follow a PIFu [7] style architecture composed of two components:

1. an *Image Encoder* which predicts features used to condition implicit functions at each pixel.
2. an *Implicit Function Decoder* that maps from the feature map (plus auxiliary information) to an output that is target-dependent.

Image Encoder: All remote point prediction models use the same image encoder, a feature-pyramid network [5] applied to a ResNet-18 CNN [3] pretrained on ImageNet [6]. Given a

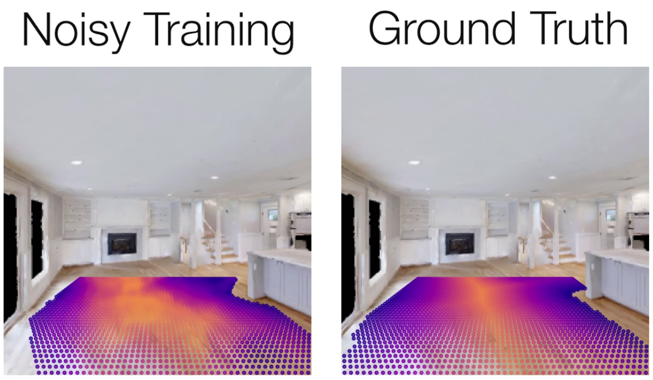


Figure 1: Screenshot of our supplementary video, showing a distance function predicted in a test set environment

$256 \times 256 \times 3$ input image, the encoder produces a pyramid of features with feature dimensions of 64, 64, 128, 256 and 512. Each of these features can be indexed with a (possibly fractional) coordinate corresponding the projected location of each query point. Fractional coordinates are accessed using bi-linear interpolation. The indexed features are then concatenated into a 1024-dimensional feature vector to condition the implicit function for each query point.

Implicit Function Decoder: The implicit function decoder maps a feature vector sampled at a location to an output.

Input: The decoder takes as input a feature vector that is bi-linearly sampled from the image encoder as described above. In all cases, we concatenate a scalar to this representing the projected depth $d_\pi(\mathbf{x})$, forming the feature denoted $\phi(\mathbf{x})$ in the main paper. Methods that use the relative heading angle as an input also have information about the angle concatenated to the feature; we encode angles using a sine/cosine encoding of $[\sin(\theta), \cos(\theta)]$. Thus angle-agnostic methods have an input dimension of 1025 and angle-conditioned ones have an input dimension of 1027.

Network: We decode the above concatenated feature vector with a multi-layer perceptron with hidden layer sizes 1024, 512, and 256; each layer is followed by a Leaky ReLU non-linearity (with negative slope 0.01).

Output and Loss Function: The output size and loss function used depends on the model:

- *Classification:* In the main multi-class case, the network produces a 11-dimensional output corresponding to the logits for steps $(0, 1, \ldots, 10+)$. The resulting loss function is the cross-entropy between the softmax of these logits and a target value derived from collision replay.

- *Regression:* In the regression case, the network produces a 1-dimensional output corresponding to the log of the number of steps. The loss function is the Smoothed L1, or L2, loss between this and the log of the target number of steps.

- *Binary Freespace:* In comparisons with freespace classification, we produce a 1-dimensional output representing the log-odds that the space is free. The loss function is the binary cross entropy between this and the target value.

3 Distance Function Decoding Details

Once the networks have made predictions, we need to decode these into distance functions.

Classification Minima Distance Function Decoding: As described in Section 3.2 in the main paper, when computing a distance function in the multi-class classification case, we find the minimum step value where the cumulative probability of collision exceeds a threshold ε (i.e. $(\sum_{j\leq t} P(j)) \geq \varepsilon$). Applied naïvely, this leads to discrete jumps between bins and always over-estimates the time to collision. We therefore linearly interpolate between the bins. This reduces over-prediction and leads to smoothly varying distance function predictions.

Angle Minima Distance Function Decoding: In experiments considering the heading angle of each remote point, we must take the minimum over possible angles to produce a distance function for the scene. In the remote prediction case, we evaluate this by taking the minimum over 32 evenly spaced sample angles between 0° and 360°. In the egocentric case, we evaluate the minimum over the actions.

4 Egocentric Prediction Model Details

Backbone: In the egocentric setting, we apply a ResNet-18 backbone to each input image, followed by average pooling and then a multi-layer perceptron with hidden layer sizes of 512 and 256, using the same Leaky ReLU activation with a negative slope of 0.01. We then compute an output layer, with shape depending on whether we use classification or regression as an objective following

- *Classification:* In the classification case, we produce an output of shape 11×3, which we interpret as a logits for the conditional probability distribution $P(t|\alpha)$, where t is one of the discrete values $(0, 1, \ldots, 10+)$, and α is one of the three actions (*turn left*, *move forwards*, *turn right*). We do not consider the *turn around* action for egocentric training, as there is reduced information in an egocentric view to predict whether the space behind the agent is occupied. At train time, we take examples of the form (image I, action A, steps to collision T), and apply a Cross Entropy loss between $P(t|\alpha = A)$ and the label T.

- *Regression:* In the regression model, we mirror the classification case as closely as possible by creating a 3 dimensional output vector, where each element predicts a steps to collision value conditioned on the agent taking a particular action. At train time, we supervise the value from this vector corresponding to the action taken in a particular example. We use the same Smoothed L1, or L2, regression objective as in the remote prediction case.

5 Data Collection Details

We filter the points and images that are used for training. For start with a dataset of 500 steps for each of 10 episodes from 360 different environments (for our main Gibson dataset), for a total of 1.8M total images. We discard any training images in this dataset containing fewer than 1 collision point or fewer than 5 non-collision points within the camera frustum. This yields a dataset of 800K images.

6 Training Details

Optimization: At train time, we use a batch size of 128 images. In the remote point prediction setting, we randomly sample 150 trajectory points to query in each example. We train each remote prediction model for 20 epochs over the above 800K image dataset. We train egocentric models for 5 epochs, which maximizes performance on the validation set. All models are optimized with Adam [4], following a cosine schedule with a starting value of 2e-5, a maximum value of 2e-4 after 30% of training has elapsed, and a final value of 1e-5. Training a remote prediction model takes approximately 12 hours on two Nvidia 2080Ti GPUs.

Data Augmentation: We apply augmentation in the form of random horizontal flips, and random horizontal shifts with percentage shift distributed normally with $\sigma = 10\%$. All image augmentations are equivalently applied to the camera matrices, so that trajectory points still correctly line up with image contents. In the remote prediction setting, we also apply 2D Gaussian noise with $\sigma = 3cm$ within the floor plane to the query points' scene coordinates, to encourage smoothness and maximize the number of pixels being supervised across epochs.

7 Theoretical Modeling

Our setting can be connected with classic random walk problem settings such as the Gambler's ruin. These settings, albeit in simplified worlds, enable us to derive analytical results describing times to collisions. In turn, these analytical results explain empirical behavior of our system and of baselines. More specifically, if can characterize how likely particular path lengths are, we can ask questions like: is it likely that we will see short paths during training? what path length do we expect?

Setting: Suppose an agent is in a 1D grid world where cells $1, \ldots, a-1$ are free and there are walls at cell 0 and a that trigger the collision sensors. The agent starts at a position z and moves to the right with probability p and to the left with probability $q = 1 - p$. This corresponds precisely to the Gambler's ruin (our results, and thus notation, follow Feller [1], Chapter 14), where a gambler and casino repeatedly play a fixed game where the gambler wins dollar with probability p and the casino wins a dollar with probability q. The gambler starts with z dollars (an integer) and the casino with $a-z$ dollars and both play until either the gambler has 0 dollars (*ruin*) or wins a dollars (*gain*). Gambler's ruin problems focus on characterizing the number of rounds the gambler and casino play. While well studied in the stochastic processes area and readily simulated, closed form solutions to these problems resist casual solutions since the number of rounds is unknown in advance.

There are a number of results that are of interest to our setting. All but the shortest path one appear in Feller [1]. We will introduce them and then show how these are of use. Suppose T is the random variable representing the time to collision whose distribution depends on (p, a, and z). Given that our agents act with some direction, we are interested in games that are biased (or $p \neq q$). We will model this with $p < \frac{1}{2}$. This means the gambler is

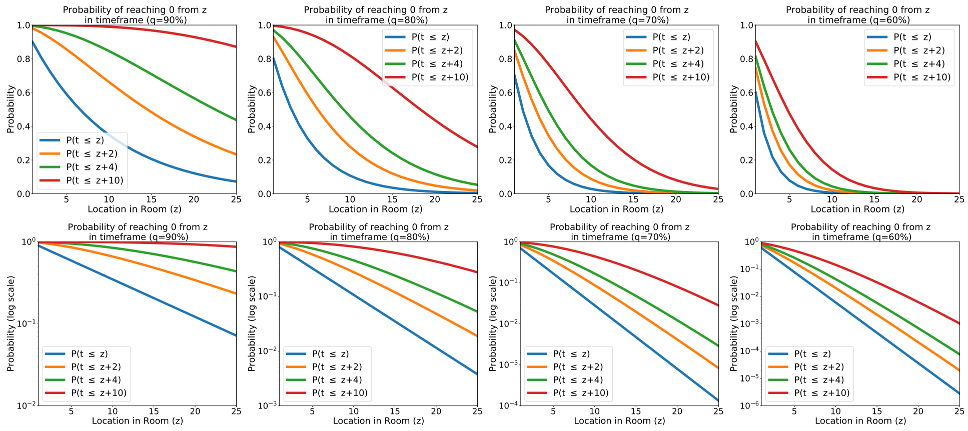


Figure 2: Plots of the probability of seeing a fairly short path as a function of location inside the room. Each figure plots probability of taking the shortest path **within** a set of tolerances as a function of the location in the room. In other words, the $P(t \leq z+4)$ plot is the probability of having a path that is no more than 4 steps farther than the optimal path and is calculated via $\sum_{i=1}^{z+4} P_T(t=i)$. The columns vary the probability of moving left q and the rows show linear (top) and logarithmic (bottom) scales. If the agent has a reasonably high chance of moving left ($q = 90\%$), then exactly short paths are surprisingly common. Even with a lower chance of moving left, nearly shortest paths are fairly common and well-represented in the training data.

likely to have ruin or, in our setting, the agent is likely to hit the left wall; one can reverse the roles of gambler and casino to obtain results for hitting the right wall. We focus on the case of ruin (i.e., hitting the left wall). The most most simple results are given for time to ruin, and ruin serves as a reasonable proxy for time to collision because ruin is virtually guaranteed for $p < \frac{1}{2}$ and reasonably-sized a).

Probability of Ruin: We are generally concerned with settings where the agent moves forward with high probability. For a fair ($p = q$) game, the probability of hitting the left wall/ruin is $1 - z/a$. For an unfair game ($p \neq q$), the probability is given by [1] as

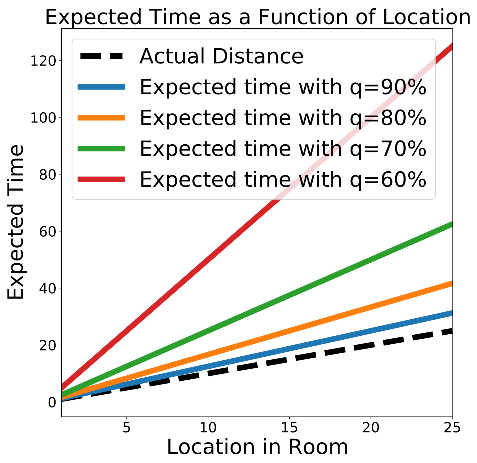
$$\frac{(q/p)^a - (q/p)^z}{(q-p)^a - 1}, \tag{1}$$

which becomes extraordinarily likely high as z and a get bigger so long as there is some advantage for the house (i.e., the agent is more likely to go left than right).

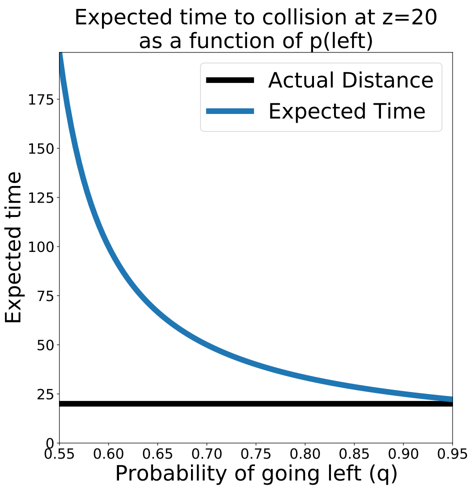
Expected duration: the expected duration of the game (i.e., $E[T]$) is important because the expected value of a distribution is the minimum for predicting samples for that distribution. Thus, a system that models time to collisions by minimizing the L2 error between samples from that distribution and the prediction has a minimum at the expected value of the distribution. The expected value has a clean, closed form expression of $z(a-z)$ for $p = \frac{1}{2}$ and

$$E[T] = \frac{z}{q-p} - \frac{a}{q-p} \cdot \frac{1-(q/p)^z}{1-(q/p)^a} \tag{2}$$

for $p \neq \frac{1}{2}$. In Fig. 3, we plot some plots of the expected time as (a) a function of location for a set of qs; and (b) a function of q for a fixed location.



(a)



(b)

Figure 3: Expected time: (a) as a function of location for a set of q; (b) as a function of q for a fixed location.

This $E[T]$ is important because a network trained to minimize the L_2 distance / MSE between its predictions and samples from T has its minimum on at $E[T]$. Across all locations and probabilities of moving towards the nearest wall q, $E[T] > z$ and is an overestimate by a factor that depends on q.

Probability of seeing the shortest path to the wall: One might wonder how frequently we would sample a *shortest* path to the wall given that we observe a time to collision. Without loss of generality, assume that $z \le a/2$: the shortest path to any wall goes to the left and takes z steps. The probability of seeing the shortest path given a sample is then given by q^z. This can be seen by noting that one only has to look at the tree of possible paths after z steps have played out. If all steps go to the left, then there is a collision; otherwise the result is not a shortest path. The probability of seeing the shortest path is relatively small for large rooms.

Distribution over probability of ruin in t steps. If the shortest path takes $t = z$ steps, we may be equally happy to reach the nearest surface in $t = z+2$ or $t = z+4$ steps since these distinctions may be washed out in actuation noise. There are known results for the particular case where we are only concerned with time to arrival at the leftmost wall or ruin. This helps us characterize how frequently we might see nearly-optimal paths to the wall. This is given by the involved but analytical formula [1] (replacing Feller's n with our t)

$$\begin{aligned} p_T(t|\text{ruin}) =& a^{-1} 2^t p^{(t-z)/2} q^{(t+z)/2} \\ & \sum_{v=1}^{a-1} \cos^{t-1}\left(\frac{\pi v}{a}\right) \sin\left(\frac{\pi v}{a}\right) \sin\left(\frac{\pi z v}{a}\right), \end{aligned} \tag{3}$$

which gives (assuming ruin occurs), the probability of it occurring on the tth step. This is a $p_T(t|ruin)$ rather than $p_T(t)$, the overwhelming likelihood of ruin means this gives close agreement to empirical simulation results for termination. This underestimates probabilities of getting a short path in the middle of rooms but otherwise gives good agreement.

We plot distributions of P_T in Fig. 2 for a large room $a = 51$: note that with a step size of 25cm, this room would be 12.5m across. The figure shows that if the agent has a reasonably high chance of moving towards the nearest wall ($q = 90\%$), then exactly short paths are surprisingly common. Even with a lower chance of moving towards the wall, nearly shortest paths are fairly common and well-represented in the training data. The only conditions under which one is unlikely to see fairly short paths is when the agent wanders (q small) or the room is enormous (a very large). Wandering can be fixed by picking a more forward-moving policy; the room can be fixed by increasing the step size, which effectively reduces a.

Concrete Results Now, we may generate concrete numbers under the following scenario: an agent starts at cell $z = 20$ in a 51 cell world enclosed by two walls, and moves left towards the wall with p=80% chance and right away from the wall with $1 - p$.

First, the agent has an effectively 100% chance of reaching 0, but the *expected* time, 33, is always an overestimate of the distance. This overestimate depends on p: (p=90% gives 25; p=45% gives $\approx$199). Thus, networks trained with the MSE should overestimate time to collision by a factor that depends on the specific policy being executed.

Second, approximately short paths are surprisingly likely. It is true that the exact shortest path's likelihood decays exponentially with distance: there is an $\approx 1/84$ chance of taking the shortest path at z=20. However, there is a $\approx 1/20$ chance of a path that is at-most 2 more steps and a $\approx 1/8$ chance of a path that is at-most 4 more steps than the shortest path. This suggests that networks trying to match $P_T(t|\mathbf{x}, \alpha)$ will frequently encounter samples that are nearly shortest. Moreover, while changing the policy changes the distribution, the distance function estimate is inaccurate only if train time frequency is very small for fairly short paths.

8 Comparison to L2 Training Loss

In Table 1 we provide an extended version of our Remote Prediction table from our main paper, now including a comparison to 'Regression-L2', a version of the model with L2 training loss. We provide Regression-L2 metrics for models trained with or without a heading angle input, and with or without ego-motion noise applied to remote point locations. We observe that it narrowly under-performs our Regression-L1 model in all settings, so we chose the L1 variant as our primary regression model for use in the main paper.

9 Comparison to Learning to Fly by Crashing

Learning to Fly By Crashing (LFC) [2] is a prior work which uses real world drone trajectories to learn a model to predict whether a drone's egocentric image is 'near' or 'far' from colliding with the environment, which approximately translates to predicting whether the agent is within K steps of colliding, for K chosen at training time.

To enable comparison, we created versions of our model which emulate LFC's original binary training task. LFC is most similar to egocentric setting, so we adopt the same prediction backbone and training methods as is described in Section 4. We replace the usual regression or multi-class classification output with a single scalar output. We then use a binary cross entropy loss, with labels specifying whether the steps-to-collision value is greater or less than a given K. We trained a version of this LFC-analogous model for K = 2, 4, 6, 8 in order to cover the 0 to 10 step range modelled by our main method.

Table 1: Quantitative results for distance functions and floorplans.

			Distance Function			Floor
	Angle	Noise	MAE	RMSE	$\% \leq \delta$	IoU
Classification	✗	✓	0.16	0.31	0.77	0.49
Regression-L1	✗	✓	0.25	0.38	0.66	0.47
Regression-L2	✗	✓	0.25	0.38	0.66	0.47
Classification	✓	✓	**0.11**	**0.21**	**0.83**	0.47
Regression-L1	✓	✓	0.13	0.24	0.79	0.45
Regression-L2	✓	✓	0.14	0.24	0.79	0.47
Free-Space	✗	✓	0.13	0.23	0.82	**0.52**
Classification	✓	✗	**0.10**	**0.20**	**0.86**	0.54
Regression-L1	✓	✗	0.11	0.22	0.83	0.49
Regression-L2	✓	✗	0.12	0.22	0.83	0.53
Supervised	-	✗	**0.08**	**0.19**	**0.90**	**0.66**
Depthmap	-	✗	0.09	0.20	0.87	0.57

Table 2: Quantitative results for distance function decoding

	Distance Function		
	MAE	RMSE	$\% \leq t$
LFC (K=2)	1.30	1.46	0.101
LFC (K=4)	0.95	1.17	0.22
LFC (K=6)	0.73	0.96	0.294
LFC (K=8)	0.61	0.841	0.335
Regression	0.96	1.14	0.13
Classification	**0.58**	**0.79**	**0.36**

Table 3: Quantitative results for threshold binary classification

	F1 Score			
	K=2	K=4	K=6	K=8
LFC	0.85	0.77	0.72	0.62
Ours	**0.86**	**0.79**	**0.74**	**0.66**

Distance function are difficult since LFC is aimed at producing a policy as opposed to scene structure or a distance function. Nonetheless, we compare on both our own task of producing a scene distance function, as well as LFC's original training task of distinguishing whether the given image is within K steps of colliding. To evaluate LFC on our distance function task we linearly re-scale it's output probability up to the same 0-4m range as our main method. To evaluate our multi-class classification model on LFC's task we produce a binary value by determining if the most likely step count (argmax) is greater or less than K.

In Table 2 we observe that our method outperforms LFC in distance function prediction across all K as expected. Additionally, as seen in Table 3 our method outperforms LFC on its binary classification task, likely due to multi-task learning effects.

10 Qualitative Results Tables

We show extended versions of select figures from our main paper in Figures 4 and 5 respectively. These examples are *randomly* selected from the same set of examples used for metric evaluation (examples with at least 10% of the 4m $\times$ 4m area infront of the agent being freespace).

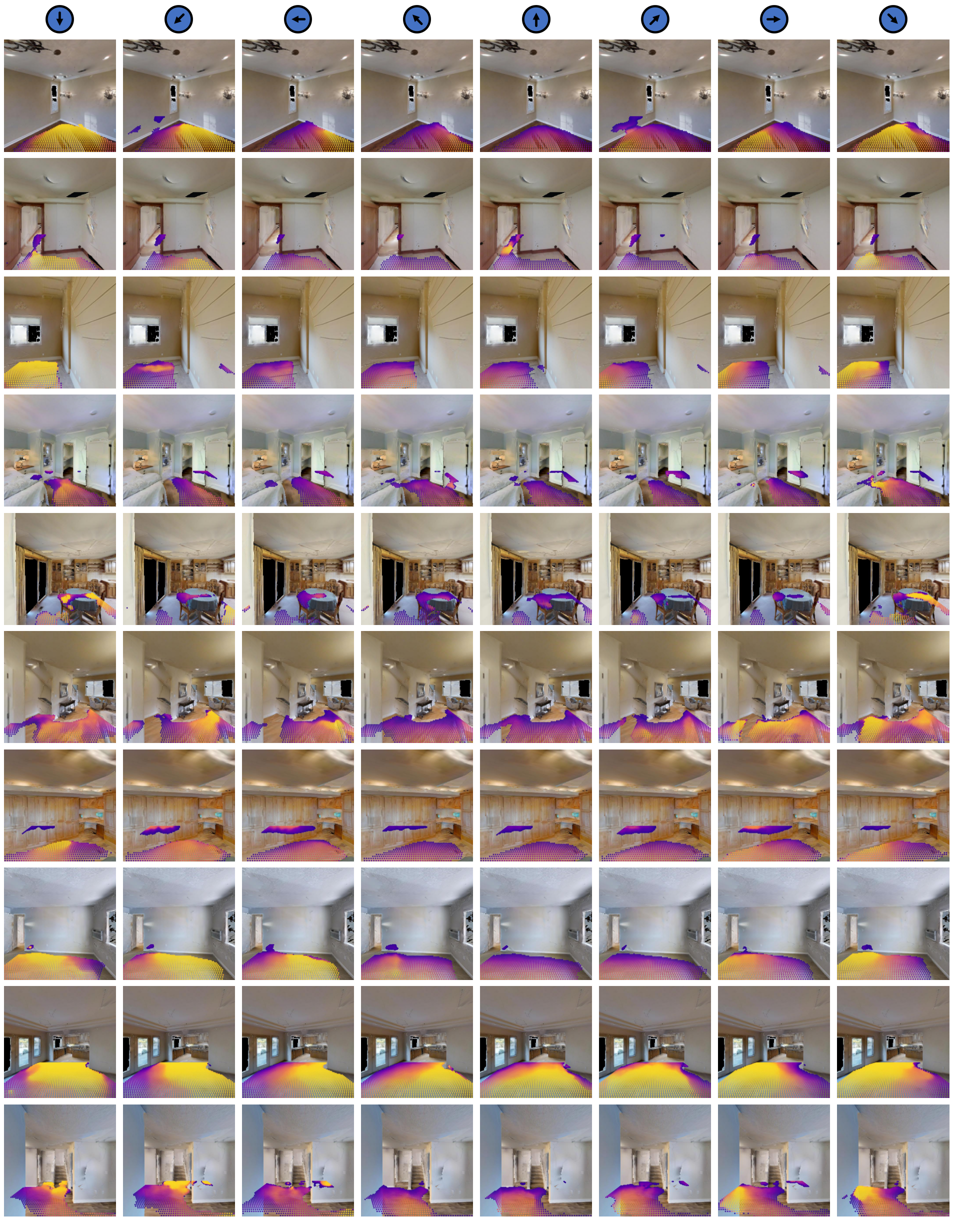


Figure 4: Randomly selected examples of the conditional probability $P(t|\alpha)$ for grids of scene points and a varying heading angle α as shown by markers in blue. Probabilities are obtained from a classification model trained with noisy random walks.

In Fig. 4, we show examples of the conditional probability $P(t|\alpha)$ for grids of scene points and a varying heading angle α. This visualizes the underlying representation used to produce distance functions. A distance function can be produced by taking the minimum prediction for each point across all visualized angles. Our method correctly predicts that points for which the heading angle faces a nearby obstacle or surface will have a low distance (shown in purple), whereas points where the heading angle leads to an open space have a higher distance (shown in yellow). In row 2, we see an example of the model's reasoning for various heading angles around a doorway. In the first column, the model recognizes that if the agent is below the doorway in the image it will likely continue into free space, but if the heading angle faces towards the doorway from above it in the image, it is unlikely that the random walk will successfully traverse through, so we see a low prediction.

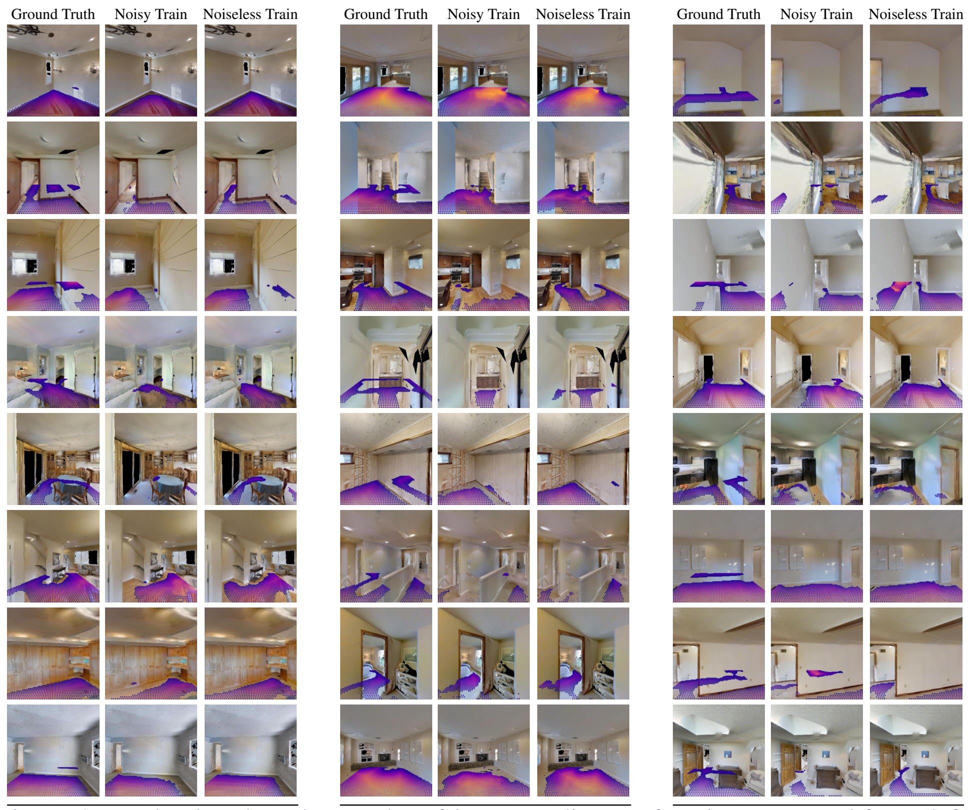


Figure 5: Randomly selected examples of 2D scene distance functions extracted from (left) the simulator navigation-mesh; (middle) a model trained on noisy random walks; or (right) a model trained on noise-free random walks

In Fig. 5, we show examples of the final distance function output of a classification model trained with either noisy or noiseless random walks, as compared to the ground truth. As in the main paper, the presence of points is used to indicate whether each method predicted the region to be freespace, so the results support that our model is generally accurate in predicting the visible regions of the scene that are traversable. The model produces lower values near obstacles, and high values in the middle of open spaces.

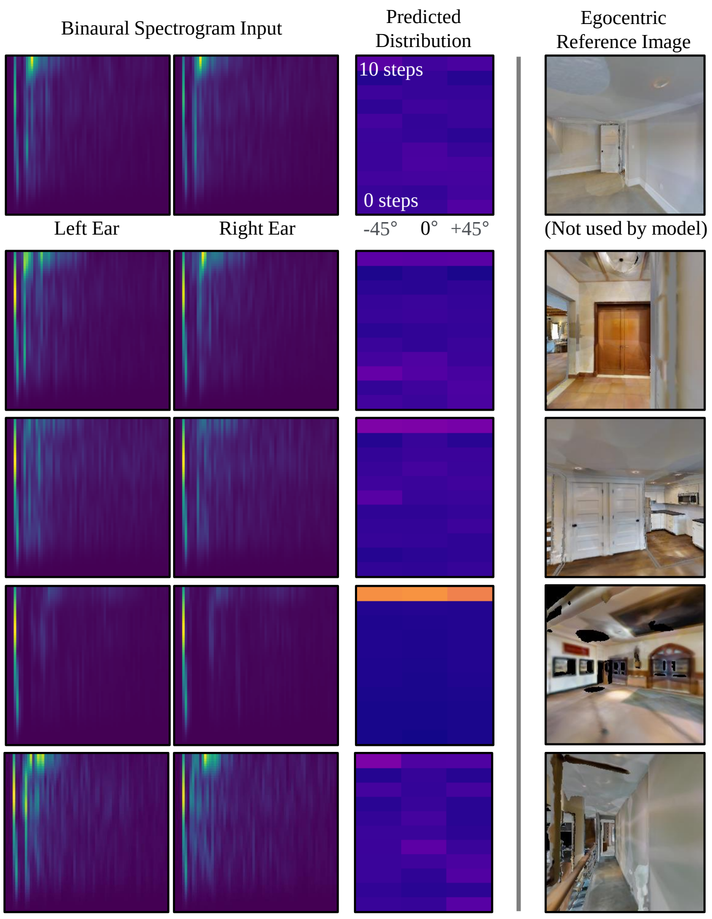


Figure 6: Selected examples of echolocation-based egocentric prediction distributions. Each row shows the model input (left), predicted $P(t|\alpha)$ distribution (middle) and a reference image (right) which was not provided as input to the model.

In Fig. 6 we show examples of per-timestep egocentric predictions made by our egocentric sound-based model. The two spectrogram images shown for each timestep are stacked to create a two channel image input to the model. As with the other egocentric images, the prediction takes the form of a distribution of $P(t|\alpha)$, with angles represented by the turn angle of the action next taken next by the agent.

References

[1] William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley and Sons, New York, 1950.

[2] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[5] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017. doi: 10.1109/CVPR.2017.106.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[7] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2304–2314, 2019.