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

Alexander Raistrick, Nilesh Kulkarni, David F. Fouhey
University of Michigan
https://araistrick.github.io/collision/

Abstract

What does bumping into things in a scene tell you about scene geometry? In this paper, we investigate the idea of learning from collisions. At the heart of our approach is the idea of collision replay, where we use examples of a collision to provide supervision for observations at a past time step. We use collision replay to train a model to predict a distribution over collision time from new observation by using supervision from bumps. We learn this distribution conditioned on visual data or echo location responses. This distribution conveys information about the navigational affordances (e.g., corridors vs open spaces) and, as we show, can be converted into the distance function for the scene geometry. We analyze this approach with an agent that has noisy actuation in a photorealistic simulator.

1. Introduction

Suppose you bump into something. What does the collision reveal? You know your current position is on the boundary of occupied space, but it is not clear what is known about other locations. If you consider where you were k steps earlier, and replay this observation, there must be a k-step path to *some* obstacle, but this does not rule out a shorter path to *another* obstacle or noise. The goal of this paper is to use this strategy of replaying a collision (*Collision Replay*) to convert the local collision signal to supervision for scene geometry for other modalities like vision or sound.

Of course, considerable effort is usually spent in vision and robotics trying to *avoid* collisions. Approaches range from building detailed geometric maps [39, 15, 36], to using these maps or other signals to learn depth estimation models [8, 46, 13], to learning full-fledged navigation systems [14, 45, 7] to everything in the middle. This is because a collision on its own is not particularly useful. Indeed, the most salient exception to the collision-avoidance trend [11] aims to collide only as a way to learn what *not to do*.

While we agree that a single collision has limited use on its own, we believe that collisions have tremendous value in larger numbers and when paired other time-locked modalities [38]. The key to unlocking their value, and the core insight of our paper, is that one should model the *distribution* of times to collision (often studied in stochastic processes [10]), *conditioned* on other signals. Each individual time to collision is merely a sample from some distribution and often an over-estimate of the shortest path to an obstacle. However, there is tremendous information in the full distribution (see, e.g., an estimated distribution in Fig. 1 conditioned on heading angle): the shortest plausible path is the distance to the nearest obstacle, and the layout of the scene geometry is visible in the distribution.

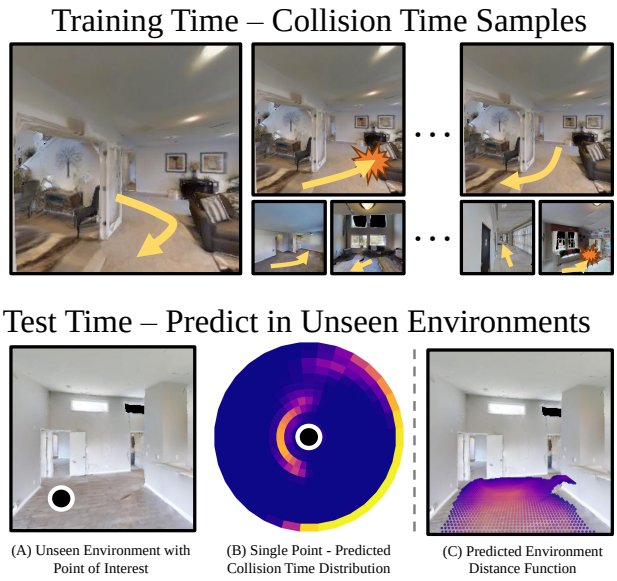


Figure 1. Our models learn from noisy samples of collisions (red crosses) in random walks. While a single collision is uninformative, multiple collisions provide signal about scene structure. At test time, given an image from an unseen environment and point on the floor (A), our model generates a distribution around that point conditioned on the heading angle (B) which can be converted to a distance function (shown for all points on the floor in C).

We show how to operationalize this idea in Section 3.2. Given a set of random walks in a simulator, our approach learns a heading-angle conditioned function from observation features to the distribution over collision times. The approach only requires recognizing "collisions" (or proximity detection) and local computations over a buffer of observations, and avoids building a full consistent map. To show the generality of the idea we explore two settings: predicting distributions at each location in the scene with PIFu [34]; predicting egocentric distributions using small images; Additionally, we also show how the idea of collision replay be used different modalities (sound or visual).

We test the value of our approach via experiments in Section 4. We demonstrate our approach in simulated indoor environments with realistic actuation noise, making use of the Habitat simulator [25] with Gibson [43] and Matterport [3] Dataset . We primarily evaluate how well we can estimate the distance function of the scene, either projected to the floor as shown in Fig 1 or seen from an egocentric view. Our experiments show that modeling the full distribution outperforms alternate approaches across multiple settings, and despite sparse supervision the performance is close to that of strongly supervised methods. We additionally show that image-conditioned estimates of the distribution capture human categories of spaces (e.g., corners or hallways).

2. Related Work

The goal of this paper is to take a single *previously unseen* image and infer the distribution of steps to collision. We show that this distribution carries information about scene shape and local topography via its distance function. We demonstrate how we can learn this distribution from collisions of an agent executing a random walk.

Estimating scene occupancy and distance to obstacles has long been a goal of vision and robotics. Collision replay provides a principled method to derive sparse noisy training signal from random walks in any number of training environments. Models resulting from collision replay are able to predict these quantities given only a single image observation of a previously unseen environment. This separates it from the large body of work that aims to build models of a specific environment over experience, e.g., via SLAM [39, 15, 9, 2] or bug algorithms [26] that use cheap sensors and loop closure. Instead, this puts our work closer to the literature in inferring the spatial layout [21, 47] of a new scene. In this area, the use of collisions separates it from work that predicts floor plans from strong supervision like ground-truth floor plans [19, 37, 32] or RGB-D cameras [14, 5]. This lightweight supervision puts us most closely to work on self-supervised depth or 3D estimation, such as work using visual consistency [46, 13, 40, 42, 41]. Bumps offer an alternate source of supervision that does not depend on 3D ray geometry and therefore can be used with a variety of different modalities (e.g., sound [12]). We show how our method can exploit echo-location responses in the scene to predict a distribution over to collision.

Our supervisory signal, time to collision, requires a few sensors and no human expertise. This is part of a larger trend in autonomous systems, where learning-based systems are trained on large-scale and noisy datasets of robots performing tasks often with a randomized polices. This has been successfully applied to learning to fly [11], grasping [30], poking [1], identifying when humans will collide with the robot [24], understanding surfaces through bounce trajectories [31] and understanding robot terrain [18]. Our work is inspired by this research and applies it by modeling the time to collision with the objects in the world (unlike [24], which modeled time to collision with a pedestrian). Among these works, the most similar is to us is LFC [11], which predicts whether a drone will crash in the next k steps from egocentric views. Our work has several key differences from [11]: we use a random, rather than targeted policy and output a richer angle- and location- conditioned distribution of steps to collision as opposed to a single probability of crashing within a fixed time horizon. Our output can be directly decoded to a distance function and is informative about scene topography. Finally, we estimate our output for remote points in the scene, as opposed to just egocentric views.

Throughout, we use implicit representation learning, which has become popular for 3D tasks [44, 27, 29, 35]. We build upon the architecture in PIFu [35], but rather than condition on image features to predict occupancy and color, we condition on point location, image features (or optionally features from spectrograms) and heading to predict a distribution over collision times

3. Method

The heart of our approach is modeling the distribution over time to collision conditioned on observations and heading angle. We start with an illustration of distributions of collision times in simplified scenes in (Section 3.1). These collision time distributions serve as learning targets for the estimators we introduce (Section 3.2). Having derived the general case, we conclude by describing the particular implementations for specific settings, namely estimating distributions in scenes and egocentric views (Section 3.3).

3.1. Preliminaries

We begin with an illustration of collision-times in a simplified, noiseless, discrete 2D grid world shown in Fig 2. In this world, the agent is at a position $\mathbf{x}$ and has heading α and can rotate in place left/right or take a single step forward. Let us assume the agent follows a random walk policy that uniformly selects one of forward, rotate left, or rotate right with probability $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Then, let $T \in \{0, 1, \ldots\}$ be

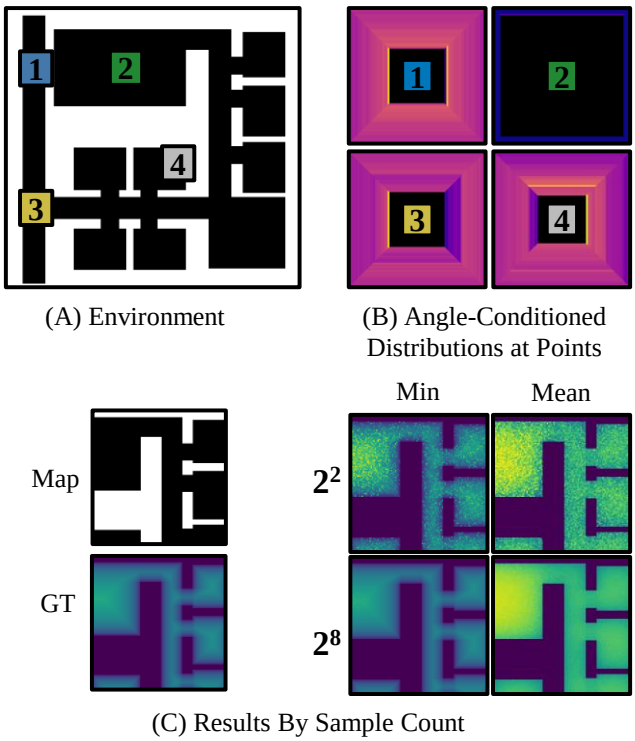


Figure 2. **A simplified overhead example**. In (A), we construct a grid-cell environment with free-space shown in black. We sample random walks at each grid cell for each start heading $\alpha \in \{N, E, S, W\}$. The distribution of hitting-times of these walks reflects the surrounding environment, as shown by the probability distributions (B) of example points 1-4, shown in log scale. Finally, we show a top-right corner of the map (C) and show how estimates of the minimum and mean converge as samples increase.

a random variable for the number of forward steps the agent takes until it next bumps into part of the scene. The distribution over the number of forward steps until the next bump is often known as the *hitting time* in the study of stochastic processes and Markov chains [10]. This work aims to learn this distribution conditioned on location $\mathbf{x}$ and heading α.

If we condition on location and angle, the probability mass function over T, or $P_T(t|\mathbf{x}, \alpha)$, tells us the likelihood about the number of steps to a collision if we start at a location and heading angle. For instance, in Fig 2, point 1, the agent is in a hallway and likely to bump into the walls if it heads out East or West, and is one step closer to the West wall (seen by the sharp colors on the right and left edges of the plot). If the agent sets out on point 3, going West will surely lead to a bump while going East will take longer.

One can convert this distribution into other quantities of interest, including the distance function to non-navigable parts of the scene as well as a floor-plan. The distance function at a location $\mathbf{x}$ is the first time t which has support

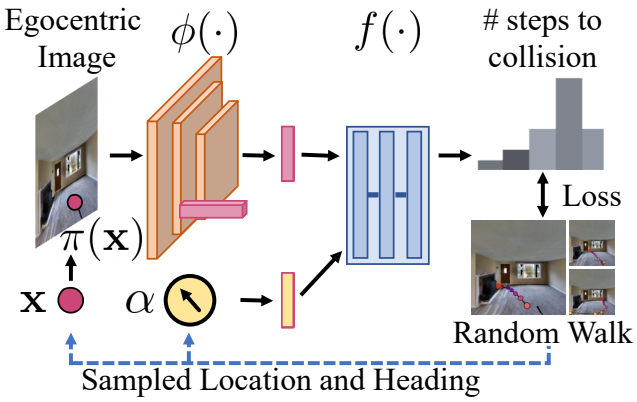


Figure 3. **Overview of training pipeline**: Our approach samples random point, $\mathbf{x}$, on a random walk and predicts the distribution of times to collision conditioned on features $\phi(\mathbf{x})$ and the corresponding heading angle α for that point on the trajectory. Our predicted distribution over steps to collision is supervised by the labels generated for the sampled point using collision replay depending upon how far the point is from collision

across any of the possible heading angles α, or

$$\text{DF}(\mathbf{x}) = \arg\min_t \text{ s.t. } \max_\alpha P_T(t|\mathbf{x}, \alpha) > 0, \qquad (1)$$

where the $\max_\alpha$ is not strictly necessary if the agent can freely rotate and rotation does not count towards steps. This distance can be converted to a floorplan by taking all points with a distance above a threshold.

Two modeling decisions require particular care: we model the agent as colliding and as following a random policy at train time. In practice, one might replace collision with being within a short tolerance (e.g., 10cm) of an obstacle, measured by the triggering of a proximity sensor. This make robots safe to operate in environments where real agent colliding with the scene is not safe, and results in a model that can learn a distance-to-near-collision. Additionally, useful robots do things rather than randomly walk. We use random policies to show that our system does not depend on any *specialized* policy, and ties with theory on random walks from the study of stochastic processes [10]. Different policies reshape P_T as opposed to precluding learning: the prevalence of fairly short paths is crucial for estimating scene geometry and these maybe more likely under goal-directed policies rather than ones that can spin in place. Additionally, the use of a random policy at train time does not require one use a random policy at test time.

3.2. Modeling Collision Times in Practice

In practice, we would like to be able to infer distributions over hitting times in new scenes, and would like to learn this from noisy samples from training scenes. We therefore learn a function $f(\phi(\mathbf{x}), \alpha)$ that characterizes $P_T(t|\mathbf{x}, \alpha)$, for instance by giving the distribution. The first input to f

is image features of the location $\mathbf{x}$ (such as the output of a deep network) concatenated with the $d_\pi(\mathbf{x})$, where d_π is the projected depth with respect to the camera π. We denote this concatenated feature as $\phi(\mathbf{x})$; the second is the heading angle. At training time, we assume each step gives a sample from the true hitting time distribution at the given location.

We frame the problem as predicting a multinomial distribution over $k+1$ categories, $0, \ldots, k-1$ and finally k or more steps. In other words $f(\phi(\mathbf{x}), \alpha)$ is a discrete distribution with with $k+1$ entries. If f is learnable, one can train it to match the predictions of f to $P_T(t|\mathbf{x}, \alpha)$ by minimizing the expected negative-log-likelihood over the distribution, or $\mathbb{E}_{s\sim P_T(t|\mathbf{x},\alpha)}\left[-\log f(\phi(\mathbf{x}), \alpha)_s\right]$. Typically, networks estimate a distribution with something like a softmax function that makes all values non-zero. Thus in practice, we modify Eqn. 1 to be first time t where the cumulative probability exceeds a hyperparameter ϵ, or $\arg\min_t \text{ s.t.} \max_a \sum_{i=0}^{t} f(\phi(\mathbf{x}), \alpha)_i \geq \epsilon$.

An illustrative alternate approach might be to predict a scalar. For instance, one could learn a mapping $g(\phi(\mathbf{x}), \alpha)$ by minimizing a regression loss like the Mean Squared Error. In other words, one would minimize the expected loss between each sample and the prediction, or $\mathbb{E}_{s\sim P_T(t|\mathbf{x},\alpha)}[(s - g(\phi(\mathbf{x}), \alpha))^2]$. The minimizer of this corresponds to the mean time to collision.

Conveniently, distributions like $P_T(t|\mathbf{x}, \alpha)$ are well-studied: in 1D, our setting corresponds to the classic Gambler's Ruin [10] problem for which many results are known. A full description appears in the supplement – the formulae are unwieldy and unintuitive. We summarize two salient results here, generating 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 ≈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 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 two more steps and a $\approx 1/8$ chance of a path that is at-most four more steps than the shortest paths. This suggests that networks trying to match $P_T(t|\mathbf{x}, \alpha)$ will frequently encounter samples that are close to the true shortest path. Moreover, while changing the policy changes the distribution, the estimate of the distance function is extremely off only if train time frequency is very small for fairly short paths.

3.3. Learning to Predict Collision Times

At training time, we have a random walk consisting of locations $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N)$. If there is a collision at time i, $\mathbf{x}_i$ is labeled as zero steps to collision, $\mathbf{x}_{i-1}$ as one step, etc. This yields tuples of locations and labels that are used as training samples in a setting-dependent fashion.

Predicting Remote Locations: Our primary setting is predicting the time to collision for a remote location on the ground-plane. This distribution can be converted into a distance function or floorplan for the scene. We predict the distribution using a PIFu [34]-like architecture shown in Fig. 3, in which a CNN predicts image features used to condition an implicit function for each camera ray. In our case, we condition on a heading angle α and projected depth represented in the egocentric frame (instead of only projected depth as in [34]). Thus, $\phi(\mathbf{x})$ is the feature map vector predicted at the location.

At train time, collision replay unwinds the agent's actuations to provide supervision at previous steps. In the case of predicting remote locations in the scene, we unwind twice to generate supervision. Suppose we consider three steps i, j, k with $i < j < k$ and a collision at step k. We know that j is $(k-j)$ steps from a collision. Then, we can project step j's location into the image at step i and give it the label $(k - j)$. The reasoning and projection is done assuming (incorrectly in our simulator) that the intended egomotion was executed correctly. This approach enables labeling from multiple views and through occlusion, while also not requiring explicit reconstruction. While the labels are noisy, this noise is simply propagated to the estimated distribution.

Predicting Current Locations: To show the generality of our method, we experiment with predicting the time-to-collision distribution for the agent's **current location**, conditioned on the agent's next action and current observation. This could be a small image in a low-cost robot, sound [12] or another localized measurement like WiFi signals. We demonstrate this approach using input from a low-resolution (32×32) RGB image, designed to simulate the variety of possible impoverished stimuli found on low cost robots. We input the agent's next action as it parallels the angle conditioning provided to the remote prediction model.

Implementation Details: *Architectures:* Both our remote location prediction and egocentric prediction network use a ResNet-18 [17] backbone followed by a series of fully connected layers. In the remote prediction case, we utilise the ResNet as a Feature Pyramid Network [23], before decoding each feature as an implicit function in a similar style to PIFu [34]. Please refer to the supplement for full architectural details. *Training:* We train all remote location networks for 20 epochs. The models for egocentric prediction are trained for 5 epochs due to their smaller image

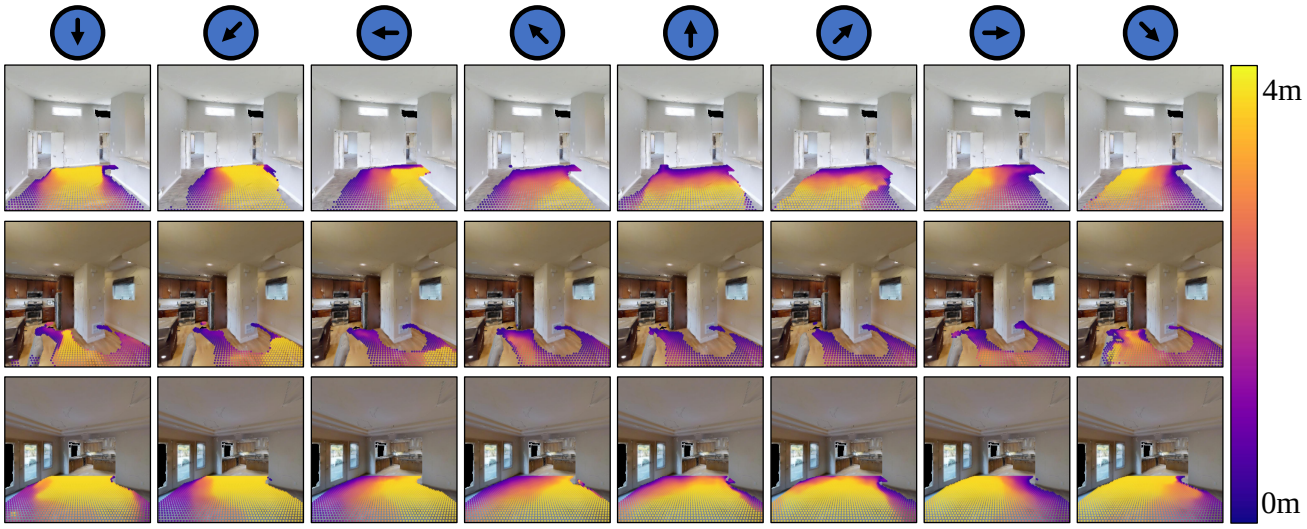


Figure 4. **Heading conditioned distance to collision:** Visualizations of the estimated first step with $> \epsilon$ probability of collision for grids of remote points with heading angles α shown as arrows. Shown in units of meters (where 1 step = 0.25m). Results are predicted by a classification model trained with noisy walks. Points are missing if the method predicted them to be occupied. In the second row we see approximately correct occupancy and plausible distances despite complex scene structure and some occlusion.

size. All models are optimized with Adam [20], 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. We apply random horizontal image flips, as well as $N(0, 10\%)$ horizontal image shifts. Points in the remote predictions setting augmented with Gaussian noise within the floor plane, with $\sigma = 3\text{cm}$. *Classification setting:* Throughout, we used a maximum label of $k = 10$ steps for our classification model. Labels above this value are clamped, so that a prediction of class 10 from our model effectively learns the probability of 10 *or more* steps before collision. Given our action space, we found this was sufficient to handle the interesting parts of most scenes. All visuals in the paper are from the classification model trained with noisy egomotion. A full description of implementation details appear in the supplemental material.

4. Experiments

We now describe a series of experiments that aim to investigate the effectiveness of the proposed approach. These experiments are conducted in a standard simulation environment, including results with actuation noise. After explaining the setup **(Section 4.1),** we introduce a series of experiments. We first evaluate predicting on remote points **(Section 4.2)**, comparing our approach to a variety of alternative architectures and upper bounds. We then analyze what the predicted time-to-collision distributions represent **(Section 4.3)**. Finally, we demonstrate the generality of the idea by applying it to egocentric views **(Section 4.4)**.

4.1. Environment and Dataset

Our experiments with predicting distance functions for remote points and egocentric time to collision we use the Habitat simulator [25] with the Gibson Dataset [43] . For each of the 360 training environments, we conduct 10 random walks of 500 time steps each, collecting collision state, egocentric image data, and egomotion at each timestep. We test on 30 held out environments from the provided test set, where we conduct 10 episodes of 500 step each.

Agent and Action Space: We model the agent as a cylinder with a radius of 18cm, a forwards facing camera 1.5 meters above the ground plane, and a binary collision sensor. At each timestep, the agent selects one of four actions. We follow Active Neural Slam [4]: *move forward* (25cm), *turn left* ($-10°$), *turn right* ($10°$). We also give the agent a *turn around* ($180°$) action for use after a collision. When noise is active, we use the noise model from [4], which are a set of Gaussian Mixture Models extracted from a LoCoBot [28] agent; we assume the *turn around* action is distributed like *turn left* but with a mean rotation of 180. When collecting data for egocentric prediction experiments, we increase the mean rotation of the turn left and turn right actions to $-45°$ and $+45°$ respectively. This means the network is not forced to make fine-grained distinctions between angles given a small image while also ensuring the space of possible turn angles covers the $90°$ field of view of the camera.

Policy: Our agent conducts a random walk by selecting an action from the set (*move forward*, *turn left*, *turn right*) with probabilities (60%, 20%, 20%) so long as it did not col-

lide with an obstacle in the previous time step. Upon collision, the agent executes a *turn around* action to prevent getting stuck against obstacles and walls. As explained in 3.1, this random walk policy is chosen for its simplicity, and the method does not depend on this policy.

4.2. Predicting Remote Time-To-Collision

Qualitative Results: In Fig. 4 we show the maps of collision times for specific heading angles for all points within a 4 m x 4 m grid on the floor in front of the camera. We observe that the model correctly varies the time-to-collision such that lower values are predicted at points where the current heading value α faces a nearby obstacle, and higher values are predicted where the heading value faces into open areas. The prediction also correctly varies from high to low values in hallways and passageways as the heading angle varies between the orientation of the hallway, and perpendicular to it. The model is able to infer some regions where the distance function is nonzero despite that the floor plane itself is occluded by obstacles. In Fig. 5 we visualize the distance function obtained from our angle conditioned outputs with both noisy and (for comparison) noise-free egomotion training. The model produces a distance function considering the closest object in any direction to each point in the scene, rather than the distance in one direction of interest as in Fig. 4. The distance function attains high values in the middle of open spaces, and smaller values in enclosed hallways and near obstacles.

Our model is capable on generating distribution for steps to collision conditioned on the input heading angle α. In Fig. 6 we visualize this distribution for four points of interest by varying the heading angle. We show polar plots where the radius of the ring denotes the distance from the point, and the color denotes the probability. At point 1 collisions are likely for all directions except reverse, and imminent directly ahead. Similarly at point 3, the upcoming pillar appears as shown as high probability ahead.

Comparisons: We compare against various approaches and ablations. For fairness, unless otherwise specified, we use an identical ResNet18 [17] and PIFu [34] backbone as described in Section 3.3,changing only the size of the outputs to accommodate different targets. All outputs can be converted into a predicted floorplan and a scene distance function. Methods which produce floorplans can be converted to distance functions by applying a distance transform to their predicted floorplans. We train classification and regression models with and without angle inputs.
(*Regression-L1/Regression-L2*): We use smoothed L1 loss (or L2 loss) to regress the steps to collisions
(*Free Space Classification*): We discretize outputs into colliding and non-colliding points. We train a binary classification model which directly predicts this outcome at each trajectory time-step, thereby learning a free-space predictor.
(*Supervised Encoder-Decoder*): We predict floor plans using strong supervision in the form of ground truth floorplans collected from the environments. This baseline serves as an upper-bound for our method because it uses sparser and weaker supervision.
(*Analytical Depthmap Projection*): We use simulator depth to predict free-space for visible regions in the scene by analytically projecting the depth maps on the floor plane to compute the visible free space in the scene.

Table 1. Quantitative results for distance functions and floorplans.

	Angle	Noise	Distance Function MAE	RMSE	$\% \leq \delta$	Floor IoU
Classification	✗	✓	0.16	0.31	0.77	**0.49**
Regression-L1	✗	✓	0.25	0.38	0.66	0.47
Regression-L2	✗	✓	-	-	-	-
Classification	✓	✓	**0.11**	**0.21**	**0.83**	0.47
Regression-L1	✓	✓	0.13	0.24	0.79	0.45
Regression-L2	✓	✓	0.14	0.79	0.24	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

Metrics: We evaluate each timestep on a 64×64 grid of points covering a $4\text{m} \times 4\text{m}$ square in front of the agent. We compute both the distance to the navigation mesh and the whether the point is free space. We use these labels to compute multiple metrics. To evaluate distance functions, we compare the ground-truth y_i and prediction $\hat{y}_i$ and report: mean absolute error (MAE), $\frac{1}{N}\sum_i |y_i - \hat{y}_i|$; root-mean squared error (RMSE), $\sqrt{\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2}$; and percent within a threshold (δ) $\frac{1}{N}\sum_i 1(|y_i - \hat{y}_i| < \delta)$. Throughout, we use $\delta = 0.25$m, the average step distance. We evaluate floorplans with *Floorplan IoU*, the intersection-over-union between predicted and ground truth floorplans.

Quantitative Results: We report results in Table 1, with distance measured in meters. In each both noise-free and noisy settings, the approach of modeling with classification produces better distance functions compared to regression approaches. L2 regression, which is equivalent to estimating the average time to collision does particularly poorly, and we find it is usually an over-prediction. Floorplan IoU does not differentiate between distances higher than the threshold, and so most approaches do well on it. Unsurprisingly, training to predict free-space via collision replay outperforms systems that obtain the floorplan estimate

Figure 5. **Extracted 2D distance functions:** Examples of 2D scene distance functions extracted from (left) the simulator navigation-mesh; (middle) a model trained on noisy random walks; and (right) a model trained on noise-free random walks. The center the rooms are correctly predicted to have higher distances (marked by lighter coloring). As in Fig. 4, points are absent if they are predicted occupied.

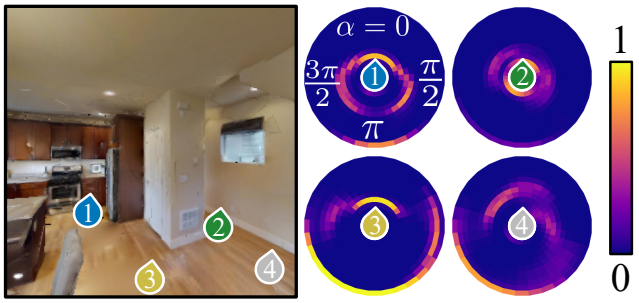


Figure 6. **Remote prediction for selected points:** Examples of $P(t|\alpha)$ for selected points in the test set. We plot the heading angle as the polar plot while radius is proportional to steps to collision. In the illustrative example for point 3, a collision is likely soon if one goes straight, whereas in case of point 2 its more likely if the agent goes left or right at this moment. This is evident from the polar plot and probability distribution

by indirect analysis of the distance function. Nonetheless, most methods trained via collision replay produces results that are close to those obtained by the strongly supervised approach (using RGB input) or the analytical depthmap projection (using RGBD input). Beyond particular metrics in estimating floorplan distance functions, we see advantages to rich representation of collision time distributions, and analyze them next.

4.3. Analysis of Learned Distributions

Qualitatively, we found that the angle-conditioned distance distributions have correlation with underlying scene geometry: the distribution in a hallway is different compared to one in the center of the room. We thus analyze whether the distributions do contain this information.

Given an image and a point we wish to be able to find other image and points that are similar to it. We express similarity between two distributions over collision times by measuring similarity between probability distribution of collision times for both given a heading angle. We start with distributions of $P_1(t|\mathbf{x}_1, \alpha_1)$ and $P_2(t|\mathbf{x}_2, \alpha_2)$ conditioned on two points $\mathbf{x}_1$ and $\mathbf{x}_2$ and compute dissimilarity using a function that also aligns the relative angle between them, followed by the Jensen-Shannon divergence (JSD) to measure how close the aligned distributions are, or

$$\min_{\theta} \text{JSD}(P_1(t|\mathbf{x}_1, \alpha), P_2(t|\mathbf{x}_2, (\alpha + \theta \bmod 2\pi))). \quad (2)$$

Qualitative Results: We find nearest neighbors for points using the dissimilarity function in Eqn. 2. For instance, the top row of Fig 7, shows that a query point in a doorway returns nearest neighbors that are also doorways or hallways with the similar distributions of $P(t|\mathbf{x}, \alpha)$. Similarly, the second row, a query point in the corner of a room yields the corners of other similarly large and open rooms.

Quantitative Results: We quantify this by annotating a subset of points in the test set with one of 5 classes: Open Space, Wall, Hallway, Concave Corner, or Crowded Space. Many points fall in between or outside the categories, so we only evaluate on the examples in which 7 out of 9 annotators agreed. This results in a dataset of 1149 points with strong consensus. We then evaluate how well Eqn. 2 predicts that two locations share the same class. We evaluate performance with AUROC, where 0.5 indicates chance performance. Our method obtains an AUROC of 0.72 distinguishing all 5 labels. Some pairs of labels are easier to distinguish than others: Open-vs-Crowded Space has an AUROC of 0.87 and Wall-vs-Corner has an AUROC of 0.55.

4.4. Predicting Egocentric Time-To-Collision

Finally, we test the generality of our approach by using collision replay to provide supervision for egocentric obser-

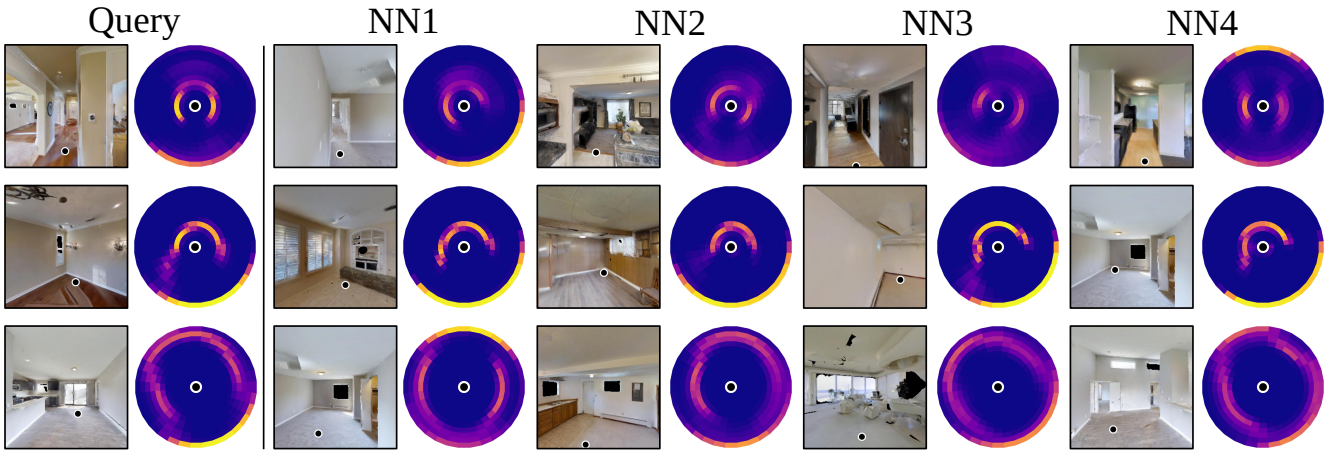


Figure 7. **Nearest neighbor lookup results:** Examples of nearest neighbors using Eqn. 2 on predicted hitting time distributions, and selecting only one per-scene. Top: passing through a doorway; Middle: corner of a room; Bottom: center of room.

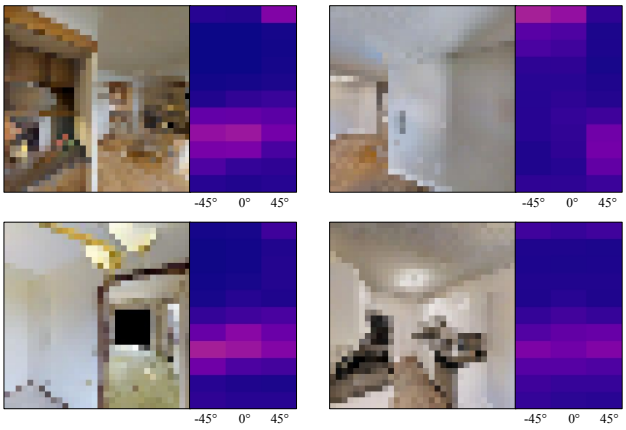


Figure 8. **Egocentric predictions:** Sample outputs for small images input to the egocentric prediction model. The center column denotes the hitting distance when there is no rotation, while the left/right column denotes the probability for distribution over collision times corresponding to turning left/right by 45°

vations. Moreover,w we demonstrate generalization to two distinct input modalities: small (32×32) images, and binaural echolocation spectrograms. Given one such input, our goal is to model the collision time conditioned on what one does next (*turn left*, *move forwards*, *turn right*).

Qualitative Results: We show qualitative results for the egocentric vision task in Fig. 8. Equivalent spectrogram results are available in the supplement. The resulting time-to-collision distribution functions as a rough depthmap: in Fig. 8(top-left), the distance function suggests that right side of the scene is substantially further away than the part that is ahead or to the left. Additionally, the model predicts a bi-modal distribution in Fig. 8(bottom-left), revealing uncertainty about whether the random walk policy will go through the doorway or collide with it.

Table 2. Quantitative results for visual egocentric prediction

	Distance Function		
	MAE	RMSE	$\% \leq t$
Regression-L1	0.96	1.14	0.13
Classification	**0.58**	**0.79**	**0.36**

Quantitative Results (Visual Input): There are no direct baselines in this setting and so we compare with regression from the previous section in Table 4. Modeling the distribution with classification again outperforms regression – regression produces systematic overestimates (overestimating in 90% of cases compared to 60% for classification).

The learning task used in Learning to Fly by Crashing (LFC) [11] is a special case of our method since it models the probability that $t < k$ for a fixed k as opposed to our full distribution over t. We note though that comparisons are difficult since LFC is aimed at producing a policy as opposed to scene structure, only applies in the egocentric setting, and also uses a specialized policy to collect data. If we compare our approach on the LFC task, predicting the binary outcome "would the agent crash in k steps", our approach outperforms LFC on F1 score for all discretizations ($k = 8$: 0.66-vs-0.62 and $k = 2$: 0.86-vs-0.85), likely due our multi-task learning effects. There is not a principled way to convert a binary probability to a distance function, and results are highly dependent on the binarization threshold k: large k can produce reasonable results (RMSE for $k = 8$ is 0.84) but small ones fare worse (RMSE for $k = 2$ is 1.46). Full experiments are in the supplement.

Quantitative Results (Echolocation): In the echolocation setting we compare against the architecture best of our egocentric vision-based models. We train both models on Matterport3D [3], which is necessary in order to collect echo spectrograms from Soundspaces [6], which does not sup-

Table 3. Quantitative results on the Matterport [3] Dataset. We compare the results of our method with different inputs (sound and visual) showcasing generalization.

Input	Distance Function		
	MAE	RMSE	$\% \leq t$
Visual	0.64	0.86	0.33
Sound	**0.58**	**0.79**	**0.36**

port Gibson at time of writing. The vision model is otherwise identical to the classification model in Table 4, and the sound model differs from this only in that it's ResNet-18 backbone is replaced with an Audio-CNN provided by the Soundspaces baseline code. We compare the relative performance of these models in Table 3.

Sound simulation relies on pre-computed Room Impulse Response (RIR) files, which are only provided for a grid of locations and orientation. We account for this by snapping to the nearest provided orientation, then applying gaussian RBF interpolation between computed spectrogram outputs based on the agent location. This yields an approximate echo response for any continuous agent pose, which allows us to maintain our same continuous agent policy, and outperformed other methods of interpolation we tried.

In Table 3 we see that echolocation outperforms the 32x32 image input version of our egocentric approach. We attribute this to the surprisingly rich information present in the audio spectrograms - the return time of emitted sound is closely related to the underlying depth map of the scene [12], so it provides more useful signal than just coarse color information. Additionally, while the sound responses are directional, sound bounces encode information about occupancy all around the agent, which is useful since the distance function may depend on objects not visible in the camera's field of view.

5. Conclusions and Discussion

This paper has introduced the idea of collision-replay which enables translating observations of the scene and bumps into the ability to estimate scene geometry in new scenes. While a simple idea, we show the careful consideration of the modeling of the problem is important. We showcase two different applications of our method using sound and visual data as input. However, we see a variety of other settings in which this can be applied, ranging from manipulation in occluded regions to converting travel time to scene maps.

Acknowledgments: This work was supported by the DARPA Machine Common Sense Program. NK was supported by TRI. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity

References

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016. 2

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016. 2

[3] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 2, 8, 9

[4] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *International Conference on Learning Representations (ICLR)*, 2020. 5

[5] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. *arXiv preprint arXiv:1801.08214*, 2018. 2

[6] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Soundspaces: Audio-visual navigaton in 3d environments. In *ECCV*, 2020. 8

[7] Harm de Vries, Kurt Shuster, Dhruv Batra, Devi Parikh, Jason Weston, and Douwe Kiela. Talk the walk: Navigating new york city through grounded dialogue. *arXiv preprint arXiv:1807.03367*, 2018. 1

[8] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 1

[9] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *IEEE transactions on robotics*, 30(1):177–187, 2013. 2

[10] William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley and Sons, New York, 1950. 1, 3, 4, 14, 15

[11] 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. 1, 2, 8, 13

[12] Ruohan Gao, Changan Chen, Ziad Al-Halab, Carl Schissler, and Kristen Grauman. Visualechoes: Spatial image representation learning through echolocation. In *ECCV*, 2020. 2, 4, 9

[13] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017. 1, 2

[14] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017. 1, 2

[15] R Hartley and A Zisserman. Multiple view geometry in computer vision, cambridge uni. *Pr., Cambridge, UK*, 2000. 1, 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 12

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4, 6

[18] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *arXiv preprint arXiv:2002.05700*, 2020. 2

[19] Kapil Katyal, Katie Popek, Chris Paxton, Phil Burlina, and Gregory D Hager. Uncertainty-aware occupancy map prediction using generative networks for robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5453–5459. IEEE, 2019. 2

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015. 5, 13

[21] Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4865–4874, 2017. 2

[22] 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. 12

[23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 4

[24] Aashi Manglik, Xinshuo Weng, Eshed Ohn-Bar, and Kris M Kitani. Forecasting time-to-collision from monocular video: Feasibility, dataset, and challenges. *arXiv preprint arXiv:1903.09102*, 2019. 2

[25] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 2, 5

[26] Kimberly N McGuire, GCHE de Croon, and Karl Tuyls. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, 121:103261, 2019. 2

[27] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 2

[28] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019. 5

[29] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 2

[30] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016. 2

[31] Senthil Purushwalkam, Abhinav Gupta, Danny M. Kaufman, and Bryan C. Russell. Bounce and learn: Modeling scene dynamics with real-world bounces. *CoRR*, abs/1904.06827, 2019. 2

[32] Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. *ECCV 2020*, 2020. 2

[33] 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. 12

[34] Shunsuke Saito, , Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. *arXiv preprint arXiv:1905.05172*, 2019. 2, 4, 6

[35] 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. 2, 12

[36] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1):7–42, 2002. 1

[37] Rakesh Shrestha, Fei-Peng Tian, Wei Feng, Ping Tan, and Richard Vaughan. Learned map prediction for enhanced mobile robot exploration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1197–1204. IEEE, 2019. 2

[38] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artif. Life*, 11(1–2):13–30, Jan. 2005. 1

[39] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002. 1, 2

[40] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[41] Hsiao-Yu Fish Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. Self-supervised learning of motion capture. *arXiv preprint arXiv:1712.01337*, 2017. 2

[42] Jamie Watson, Michael Firman, Gabriel J Brostow, and Daniyar Turmukhambetov. Self-supervised monocular depth hints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2162–2171, 2019. 2

[43] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world per-

ception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018. 2, 5

[44] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *Advances in Neural Information Processing Systems*, pages 492–502, 2019. 2

[45] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018. 1

[46] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 1, 2

[47] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. Layoutnet: Reconstructing the 3d room layout from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2051–2059, 2018. 2

A. Remote Prediction Model Details

All remote point prediction models follow a PIFu [35] 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 [22] applied to a ResNet-18 CNN [16] pretrained on ImageNet [33]. Given a $256 \times 256 \times 3$ 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 Decoding: The implicit function decoding 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, denoted $\phi(\mathbf{x})$ in the main paper. In all cases, we concatenate a scalar to this representing the projected depth $d_\pi(\mathbf{x})$. 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 nonlinearity (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 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.

B. 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. $(\sum_{j \leq t} P(j)) \geq \epsilon$). 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.

C. 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 through 1, 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 condition value provided the agent takes 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 regression objective and log-transformed steps to collision label as in the remote prediction case.

D. Data Collection Details

We filter the points and images that are used for training. We start with a dataset of 500 steps for each of 10 episodes from 360 different environments, for a total of 1.8M total images. We discard any 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.

E. 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 [20], 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 scene geometry. 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.

F. Comparison to Learning to Fly by Crashing

Learning to Fly By Crashing (LFC) [11] 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.

Table 4. 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 5. 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**

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 C. 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.

Direct comparisons 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 4 we observe that our method outperforms LFC in distance function prediction across all K as expected. Additionally, as seen in Table 5 our method outperforms LFC on its binary classification task, likely due to multi-task learning effects.

G. 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

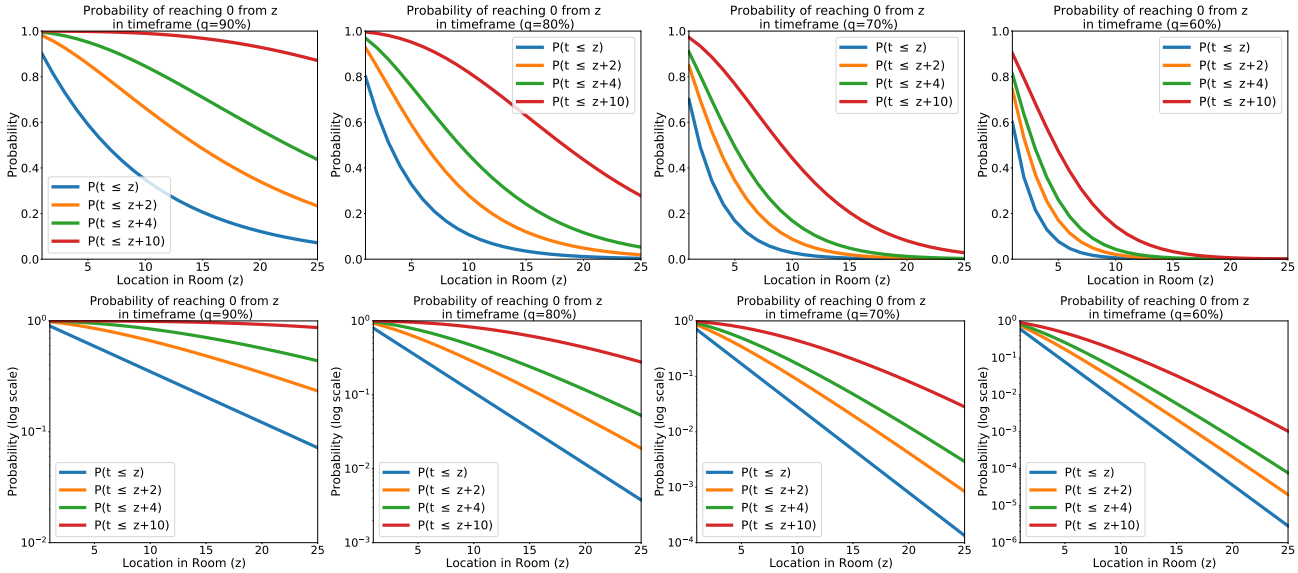
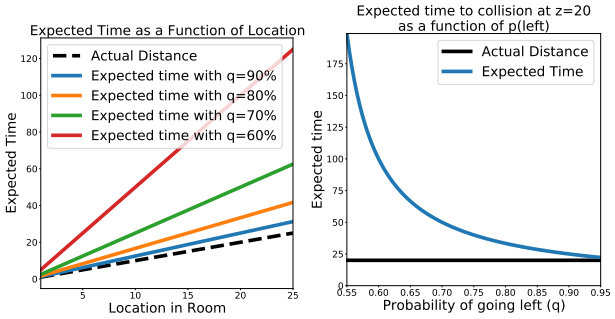


Figure 9. Plots of the probability of seeing a fairly short path as a function of location inside the room. Each figure plots probability of taking the shortest path **within** a set of tolerances as a function of the location in the room. In other words, the $P(t \leq z + 4)$ plot is the probability 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.



(a) (b)

Figure 10. Expected time: (a) as a function of location for a set of q; (b) as a function of q for a fixed location.

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 [10], 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 [10]. 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 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 prob-

ability is given by [10] as

$$\frac{(q/p)^a - (q/p)^z}{(q-p)^a - 1}, \tag{3}$$

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

$$E[T] = \frac{z}{q-p} - \frac{a}{q-p} \cdot \frac{1-(q/p)^z}{1-(q/p)^a} \tag{4}$$

for $p \neq \frac{1}{2}$. In Fig. 10, 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.

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 $z \leq a/2$: 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 [10] (replacing Feller's n with our t)

$$p_T(t|\text{ruin}) = a^{-1} 2^t p^{(t-z)/2} q^{(t+z)/2} \sum_{v=1}^{a-1} \cos^{t-1}\left(\frac{\pi v}{a}\right) \sin\left(\frac{\pi v}{a}\right) \sin\left(\frac{\pi z v}{a}\right), \tag{5}$$

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. 9 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.

H. Qualitative Results Tables

We show extended versions of Figures 3 and 4 of the main paper in Figures 11 and 12 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).

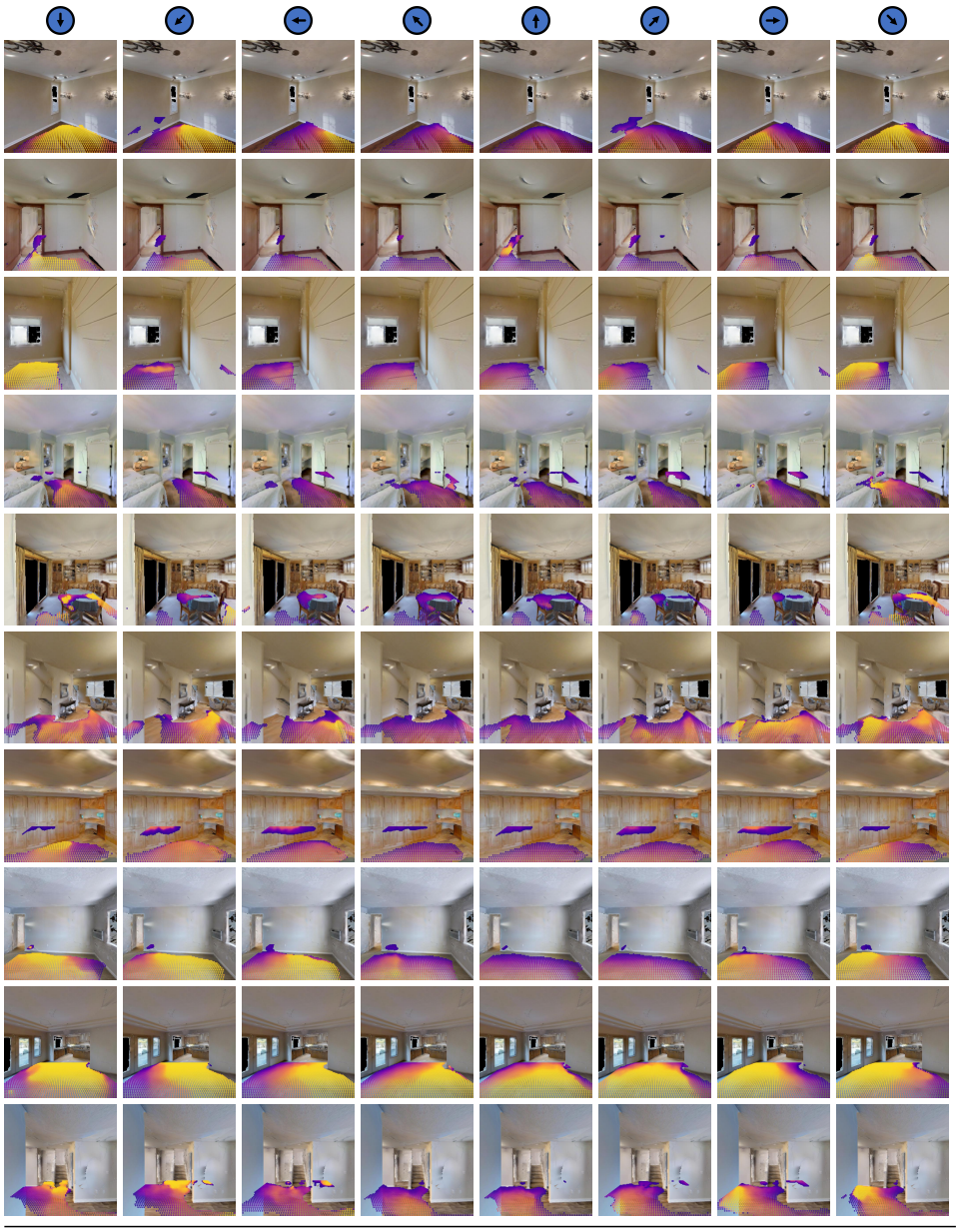


Figure 11. Randomly selected examples of the conditional probability $P(t|\alpha)$ for grids of scene points and a varying heading angle α as shown by markers in blue. Probabilities are obtained from a classification model trained with noisy random walks.

In Fig. 11, 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 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 points towards 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. Similarly, in the 5th column of row 2, we see the model infer that the visible points beyond the doorway will likely lead into empty space.

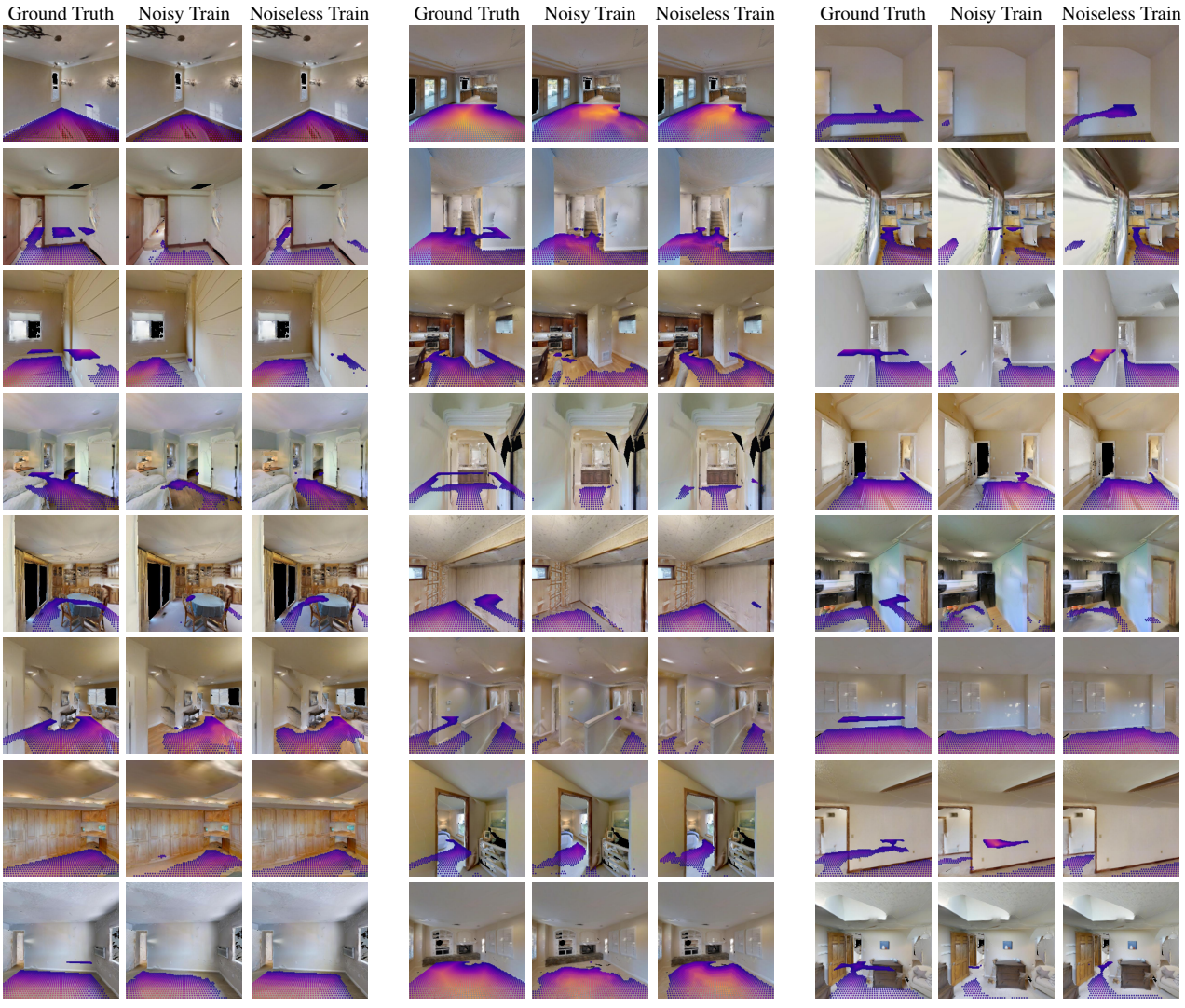


Figure 12. Randomly selected examples of 2D scene distance functions extracted from (left) the simulator navigation-mesh; (middle) a model trained on noisy random walks; or (right) a model trained on noise-free random walks

In Fig. 12, 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 above, the presence of points is used to indicate whether the model 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.

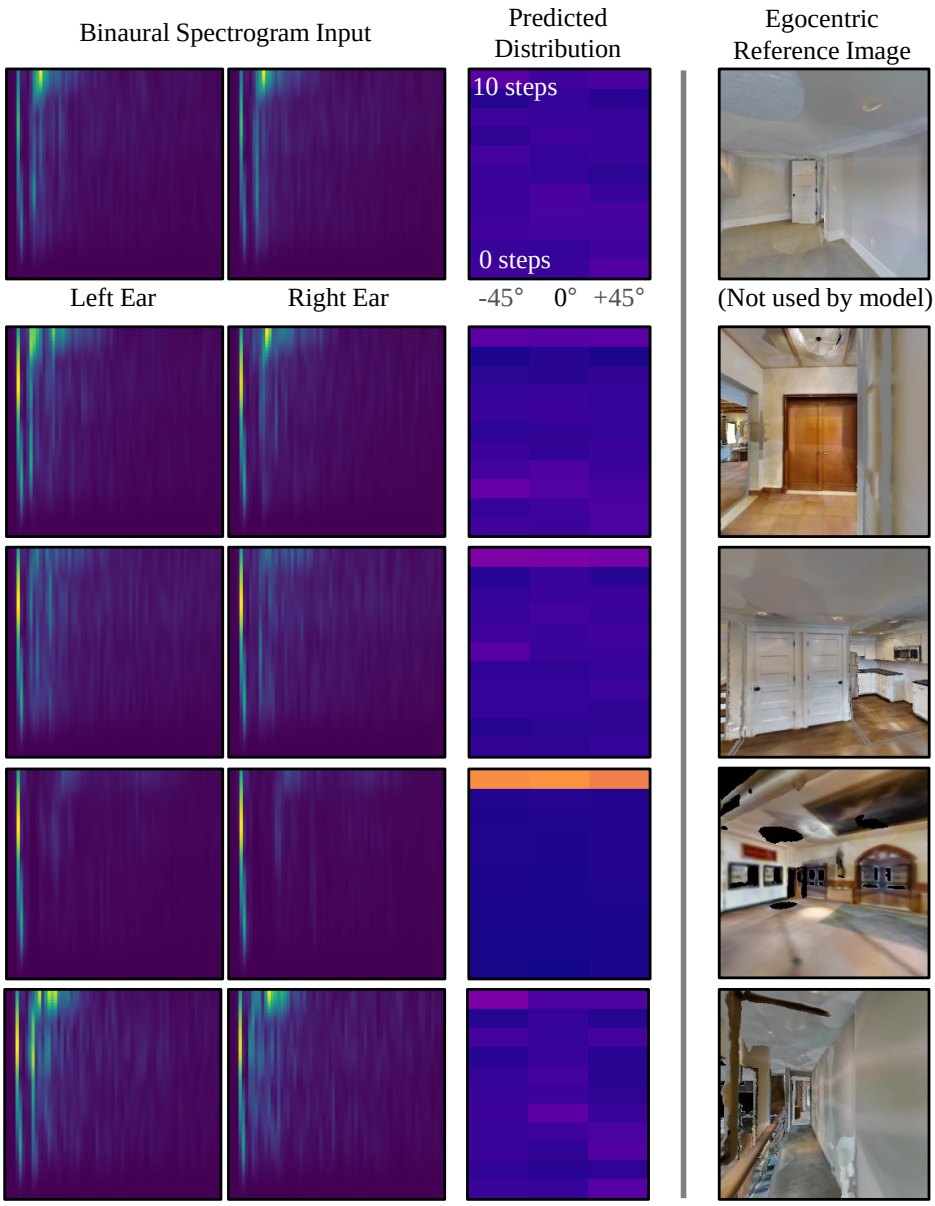


Figure 13. Selected examples of echolocation-based egocentric prediction distributions. Each row shows the model input (left), predicted $P(t|\alpha)$ distribution (middle) and a reference image (right) which was not provided as input to the model.

In Fig. 13 we show examples of per-timestep egocentric predictions by our egocentric sound-based model in 4.4. The two spectrogram images shown for each timestep are stacked to create a two channel 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 angles to be taken next by the agent.