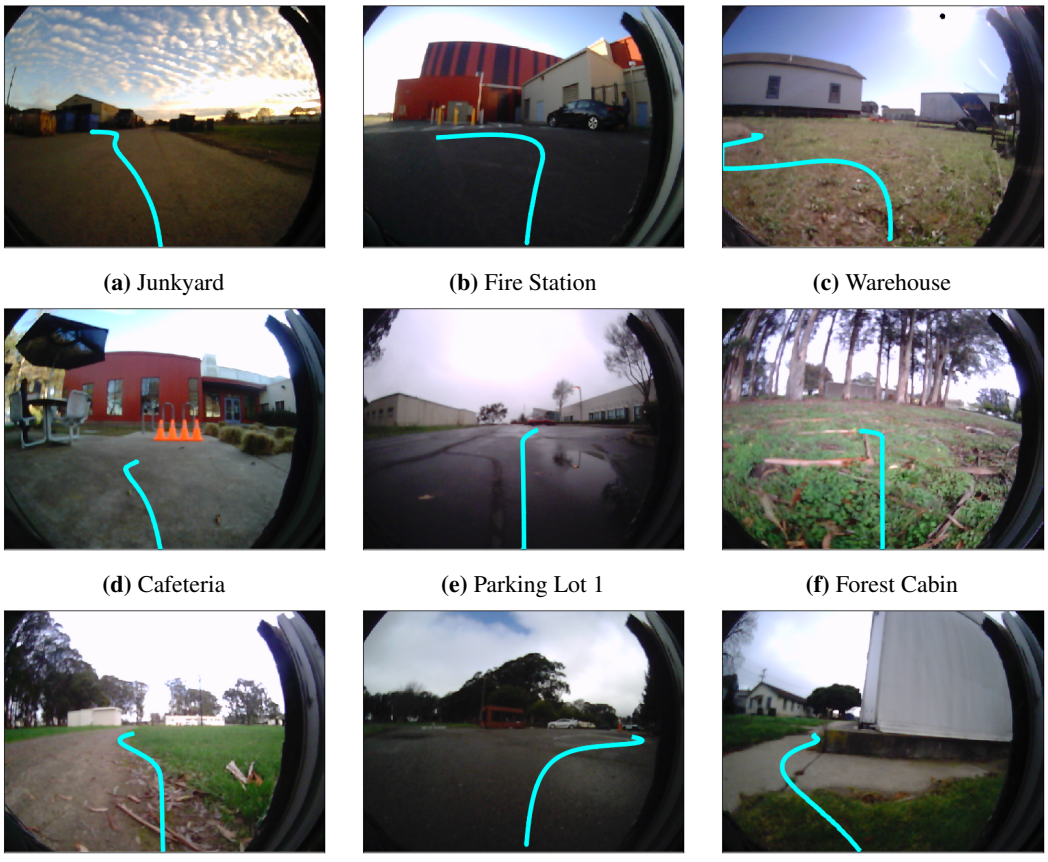
A Dataset

In this work, we emphasize that data collected from prior experience in unrelated environments can be a rich source of supervision, even if the interactions in the dataset are suboptimal. To demonstrate this, we curate a dataset of over 5000 self-supervised trajectories collected over 9 distinct real-world environments. These trajectories capture the interaction of the robot in diverse environments, including phenomena like collisions with obstacles and walls, getting stuck in the mud or pits, or flipping due to bumpy terrain. The dataset contains measurements from a wide range of sensors including a pair of stereo RGB cameras, thermal camera, 2D LiDAR, GPS and IMU to support offline evaluation using an alternative suite of sensors. While a lot of these sensor measurements can be noisy and unreliable, we believe that learning-based techniques coupled with multimodal sensor fusion can provide a lot of benefits in the real-world. This dataset was collected over a span of 18 months, including parts collected by Kahn et al. [56] and Shah et al. [11] for earlier research projects, and exhibits significant variation in appearance due to seasonal and lighting changes.

This dataset is available for download at `sites.google.com/view/recon-robot/dataset`, along with helper scripts to load and visualize the trajectories.



(a) Junkyard **(b)** Fire Station **(c)** Warehouse

(d) Cafeteria **(e)** Parking Lot 1 **(f)** Forest Cabin

(g) Farmlands **(h)** Parking Lot 2 **(i)** Residential

Figure 7: We collect data in 9 diverse environments. Example trajectories are shown in cyan.

A.1 Self-Supervised Data Collection and Labeling

We design the data collection methodology to enable gathering large amounts of diverse data with minimal human intervention. Due to the high cost of gathering data with real-world robotic systems, we choose to use an off-policy learning algorithm in order to be able to gather data using any control policy and train on all of the gathered data. To ensure that the control policy achieves sufficient coverage of the environment while also ensuring that the action sequences executed by the robot

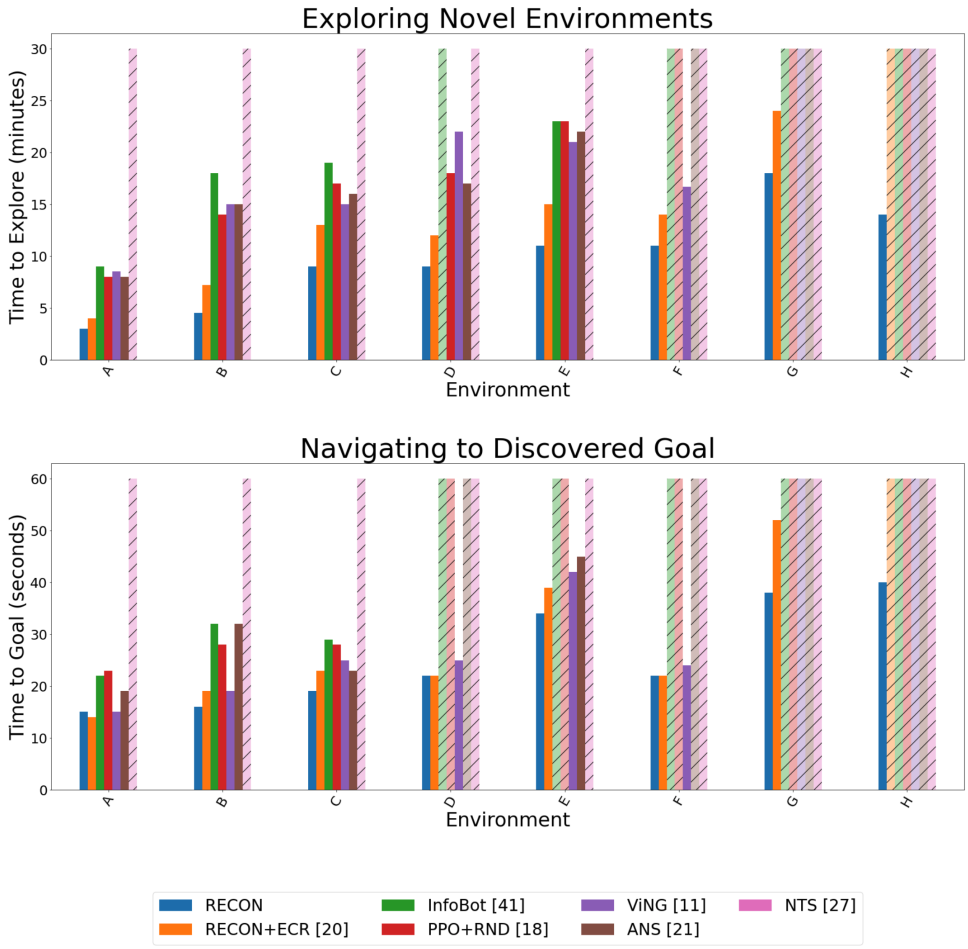


Figure 8: Exploring and learning to reach goals: *(left)* Amount of time needed for each method to search for the goals in a new environment (↓ is better; hashed out bars represent failure). *(right)* Amount of time needed to reach the goal a second time, after reaching the goal once and constructing the map, in seconds (↓ is better).

are realistic, we use a time-correlated random walk to gather data. A naïve uniform random control policy is inadequate because the robot will primarily drive straight due to the linear and angular velocity action interface of the robot, which will result in both insufficient exploration and unrealistic test time action sequences.

During data collection using the random control policy, the robot requires a mechanism to detect if it is in collision or stuck, and an automated controller to reset itself in order to continue gathering data. We detect collisions in one of two ways, either using the LIDAR to detect when an obstacle is near or the IMU to detect when the robot is stuck due to an obstacle or uneven terrain. We program an automated backup maneuver that drives the robot out of collision (whenever possible) so it can initiate a new trajectory.

We also use these collision detectors as a weak source of supervision by generating *event labels* for the collected trajectories, giving us a self-supervised relabeling pipeline as proposed in BADGR [56]. We consider three different events: collision, bumpiness, and position. A collision event is detected the LIDAR measures an obstacle to be close or, in off-road environments, when the IMU detects a sudden drop in linear acceleration (jerk) and angular velocity magnitudes. A bumpiness event is calculated as occurring when the angular velocity magnitudes measured by the IMU are above a certain threshold. The position is determined by an onboard state estimator that fuses wheel odometry and the IMU to form a local position estimate. Note that all experiments reported in this paper only use the collision labels; these labels are used to dissect the random walks into smooth trajectories that end in collision.

A.2 Environments

To learn general navigational affordances across a wide range of environments, we curate over 40 hours of trajectories in 9 diverse open-world environments of varying complexity (see Figure 7).

Figure 8 shows the exploration and navigation performance of RECON and the baselines (see Sec. 5 for details) on the individual environments. As the environment complexity increases, most methods are not able to explore the environment efficiently to discover the goal. For videos of our system exploring these environments, please check out the supplemental video submission.

B Reproducibility

B.1 Algorithmic Components

The SubgoalNavigate function rolls out the learned policy for a fixed time horizon H to navigate to the desired subgoal latent z_t^w, by querying the decoder $q_\theta(a_t, d_t | z_t^w, o_\tau)$ in an open loop manner. The endpoint of such a rollout is used to update the visitation counts v in the graph $\mathcal{G}$ using the AssociateToVertex subroutine. To nudge the robot to the frontier, we use a heuristic LeastExploredNeighbor routine that uses the visitation counts of the neighbors to identify unexplored areas in the local neighborhood. At the end of each trajectory, the ExpandGraph subroutine is used to update the edge and node sets $\{\mathcal{E}, \mathcal{V}\}$ of the graph $\mathcal{G}$ to update the non-parametric representation of the environment. Pseudocode for these subroutines are given in Alg. 3.

Algorithm 3 Pseudocode for subroutines referenced in the exploration algorithm shown in Alg. 1

function SubgoalNavigate($z_t^w; H$)
  trajectory $\leftarrow ()$
  for $t \in [1, \ldots, H]$ **do**
    trajectory.append((o_t, a_t, t))
    $a_t, d_t^g \sim q_\theta(a_t, d_t \mid z_t^w, o_\tau)$ ▷ *Sample action*
    $o_t \leftarrow$ Env.step(a_t) ▷ *Execute action*
  $v_H \leftarrow$ AssociateToVertex($\mathcal{G}, o_H$)
  v_H.count $\leftarrow v_H$.count $+ 1$
  $\mathcal{D}_w \leftarrow ((o_t, o_H, a_t, H - t)$ for $(o_t, a_t, t) \in$ trajectory$)$
  return $\mathcal{D}_w, o_H$

function AssociateToVertex($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t$)
  $\mathbf{d} \leftarrow$ sort$((\bar{d}_t^v, v)$ for $v \in \mathcal{V})$ ▷ *Predict distances*
  $v, d \leftarrow \mathbf{d}[0]$ ▷ *Associate o_t with nearest vertex*
  return v

function LeastExploredNeighbor($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t, \delta_2$)
  $v \leftarrow$ AssociateToVertex($\mathcal{G}, o_t$)
  $\mathcal{V}_n \leftarrow \{v' : \mathcal{E}(v, v') < \delta_2, v' \in \mathcal{V}\}$ ▷ *Retrieve neighbors*
  $\mathbf{c} \leftarrow$ sort$((v'$.count$, v'$.o$)$ for $v' \in \mathcal{V}_n)$
  $v_c, o_c \leftarrow \mathbf{c}[0]$ ▷ *Retrieve neighbor with smallest count*
  return o_c

procedure ExpandGraph($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t$)
  $v_t \leftarrow$ Node(count $= 1, o = o_t$) ▷ *Create node for o_t*
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v_t, v_g) : \bar{d}_t^g, g \in \mathcal{V}\}$ ▷ *Add edges*
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_t\}$ ▷ *Add vertex*

B.2 Implementation Details

Inputs to the encoder p_ϕ are pairs of observations of the environment – current and goal – represented by a stack of two RGB images obtained from the onboard camera at a resolution of 160×120 pixels.

Hyperparam.	Value	Meaning
δ_1	4	Threshold of identification
δ_2	15	Threshold of neighbors
ϵ	10^{-2}	Exploration threshold on prior
β	1.0	Model complexity
γ	10	Epochs to finetune model
H	5 seconds	Horizon to navigate to subgoal

Table 3: Hyperparameters used in our experiments.

p_ϕ is implemented by a MobileNet encoder [61] followed by a fully-connected layer projecting the 1024-dimensional latents to a stochastic, context-conditioned representation z_t^g of the goal that uses 64-dimensions each to represent the mean and diagonal covariance of a Gaussian distribution. Inputs to the decoder q_θ are the context (current observation) – processed with another MobileNet – and z_t^g. We use the reparametrization trick [62] to sample from the latent and use the concatenated encodings to learn the optimal actions a_t^g and distances d_t^g. Details of our network architecture are provided in Table 4. During pretraining, we maximize Eq. 2 with a batch size of 128 and perform gradient updates using the Adam optimizer with learning rate $\lambda = 10^{-4}$ until convergence. We provide the hyperparameters associated with our algorithms in Table 3.

Layer	Input [Dimensions]	Output [Dimensions]	Layer Details
		Encoder $p_\phi(z \mid o_t, o_g) = \mathcal{N}(\cdot; \mu_p, \Sigma_p)$	
1	o_t, o_g [3, 160, 120]	I_t^g [6, 160, 120]	Concatenate along channel dimensio.
2	I_t^g [6, 160, 120]	E_t^g [1024]	MobileNet Encoder [61]
3	E_t^g [1024]	μ_p [64], σ_p [64]	Fully-Connected Layer, exp activation of σ_p
4	σ_p [64]	Σ_p [64, 64]	$\text{torch.diag}(\sigma_p)$
		Decoder $q_\theta(a, d \mid o_t, z_t^g) = \mathcal{N}(\cdot; \mu_q, \Sigma_q)$	
1	o_t [3, 160, 120]	E_t [1024]	MobileNet Encoder [61]
2	E_t [1024], z_t^g [64]	$F = E_t \oplus z_t^g$ [1088]	Concatenate image and goal representation
3	F [1088]	μ_q [3], σ_q [3]	Fully-Connected Layer, exp activation of σ_q
4	σ_q [3]	Σ_q [3, 3]	$\text{torch.diag}(\sigma_q)$
5	μ_q [3]	$\bar{a}_t^g[2], \bar{d}_t^g[1]$	Split into actions and distances.

Table 4: Architectural Details of RECON: The inputs to the model are RGB images $o_t \in [0, 1]^{3\times160\times120}$ and $o_g \in [0, 1]^{3\times160\times120}$, representing the current and goal image.

C Supplemental Video

For more results and videos of our system deployed in unstructured, outdoor environments in the real-world, please check out the supplemental video submission.