Learning for Robot-centric Autonomy

Xiangyun Meng

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Commitee:

Dieter Fox, Chair

Byron Boots

Abhishek Gupta

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Learning for Robot-centric Autonomy

Xiangyun Meng

Chair of the Supervisory Committee:
Dieter Fox
Paul G. Allen School of Computer Science & Engineering

Autonomy is a foundational capability that frees robots from confined workspaces and lets them interact with the open world. The traditional approach to robot autonomy has relied heavily on a world-centric approach: building a global, geometrically accurate map and using it for localization and planning. However, this approach often proves inadequate or impractical in many real-world applications. This thesis adopts a robot-centric perspective to autonomy, addressing the challenges across three distinctive scales: (1) Globally, we learn to compress visual experiences into sparse, topological scene representations for long-horizon navigation; (2) At the semi-local level, we develop perception systems that reason about the traversability of the terrains around the robot to achieve robust off-road navigation; (3) Locally, we learn end-to-end perception-action models to navigate a robot to any object with high precision. We demonstrate the real-time performance of our approaches across diverse robotic platforms, highlighting the applicability and generalizability of these methods.

Contents

Acknowledgments

Looking back, I feel blessed to have been accepted into the PhD program at the University of Washington. My seven years here have fundamentally shaped my views on research, teamwork, and, most importantly, the kind of life I want to pursue. First, I would like to thank my advisor, Dieter Fox, for his guidance. His intuition, wisdom, and encouragement have motivated me to pursue my passion for robotics. His approach to thinking and his philosophy on work, life, and education resonate deeply with me and will have a lasting influence on my life. I would also like to thank Byron Boots for his mentorship on the off-road research projects. These projects opened my eyes to the intersection of research, engineering, and teamwork. This rare opportunity taught me many invaluable lessons that I truly cherish. Finally, I would like to thank my committee members, Abhishek Gupta and Sawyer Fuller, for their feedback on my thesis.

I am grateful to have met so many wonderful collaborators at UW, many of whom became not only colleagues but also good friends. I would like to thank Yu Xiang for helping me kickstart my PhD journey. I still remember the first few weeks of staying at his apartment while searching for housing. We collaborated on my first three papers and enjoyed playing sports together for several years. While working on the off-road projects, I was fortunate to collaborate with Amirreza Shaban, Brian Lee, and Sanghun Jung. Together, we overcame many obstacles (pun intended) and formed strong friendships. Lastly, I thoroughly enjoyed working with Yuxiang Yang on several quadrupedal locomotion projects.

My time interning at NVIDIA and Accenture also shaped my journey in meaningful ways. At NVIDIA, I learned a great deal from Nathan Ratliff's analytical thinking and worked closely with Xuning Yang and Fabio Ramos on my final research project. I am especially grateful to Xuning for her technical and emotional support during the toughest moments of the project. At Accenture, I was fortunate to work with Sanjoy Paul, Srid Sadhan Jujjavarapu, and Michael Chen, whose support in securing additional equipment and compute resources was critical to the success of the project.

I would also like to thank all the graduate and undergraduate students in the lab: Tanner, Conner, Arun, Daniel, Aaron, Junha, Chris, Adam, Mohit, Yi, Zoey, Wentao, Vinitha, Schmittle, Helen, Jiafei, Marius, Lirui, and Kyle. I cherish the time we spent together during reading groups, beers, retreats, and other adventures.

Finally, I extend my deepest gratitude to my parents for their unconditional love and support throughout my PhD journey.

Chapter 1

Introduction

Autonomy is one of the oldest and most extensively studied areas in robotics. Its rich history underscores its significance in the past and hints at its growing importance in the future. With increasing labor shortages, heightened focus on employee safety, and rising demands for efficiency in logistics, autonomous robots are expected to take on roles traditionally performed by humans, such as transportation, inspection, and manipulation. But are our robots truly ready to meet the challenges of this exciting new world?

To answer this question, let's first examine how current mobile robots operate. These robots typically rely on geometric sensors like LiDAR and depth cameras to perceive their surroundings. During the *mapping* phase, they collect sensor data to construct an accurate geometric map of the environment [73, 165]. In the subsequent *deployment* phase, the robots localize themselves within this map and navigate toward specific goals. But what's the limitation of this approach?

First, building and maintaining an accurate geometric map is challenging. A robot must move slowly to collect dense LiDAR and depth data to create an accurate map. However, consider an emergency response vehicle tasked with rescuing a soldier in a desert. Such a vehicle must drive at high speeds over hilly terrain with unknown hazards and no prior exposure to the environment.

The traditional mapping approach struggles to handle the vehicle's fast movements due to sparse and distorted sensor readings. Now, imagine another scenario where the robot is equipped with only a low-resolution monocular RGB camera, lacking LiDAR or depth sensors. Traditional visual mapping methods [118] would fail in this case, as they rely on detecting sufficient features for reconstruction, which is difficult with low-resolution images.

Second, the world is more complex than just geometry. While geometry provides information about the shape of the environment, it fails to convey the semantics. For example, a robot can drive through grass but must avoid rocks, even though grass and rocks may have similar geometric sizes. Similarly, a robot must slow down on a snow-covered road, even though it appears geometrically similar to a dry road. Furthermore, mobile manipulators, such as humanoid robots and autonomous forklifts, introduce new opportunities for automation. Consider a forklift loading a pallet: accurate positioning is essential for inserting the forks into the pallet slots. Thus, as robots are increasingly deployed in real-world scenarios, semantics and object-centric reasoning become crucial for effective navigation and interaction.

In this thesis, we explore robot autonomy beyond the traditional paradigm of geometric mapping and planning. We adopt a robot-centric approach, designing systems that enable robots to navigate robustly and accurately without relying on a global geometric map. Removing the assumption of a global geometric map poses challenges at multiple scales. We address these challenges in the following chapters:

In Chapter 2, we develop non-metric, global world representations by distilling a robot's visual experiences. Humans can navigate large-scale environments effortlessly without relying on metric maps. Inspired by human cognition, we design navigation systems that use topological scene representations for mapping and planning. These systems distill robot experiences by learning *reachability* and *behaviors*, saving orders of magnitude of memory while being robust. By training in diverse, visually realistic simulation environments, we demonstrate zero-shot transfer

to real-world robots and environments.

In Chapter 3, we focus on terrain modeling for off-road autonomy. Mobile robots operating in harsh environments, such as deserts and jungles, must adapt to unfamiliar terrain without prior exposure. The traversability of these environments depends on semantics (e.g., distinguishing grass from rock), geometry (e.g., identifying flat ground versus steep inclines), and robot capabilities (e.g., horsepower and suspension). We design a scalable data generation pipeline to build high-fidelity, robot-centric terrain models. We also develop perception systems capable of inferring these terrain models in real time using onboard sensors, demonstrating their performance on real-world, passenger-sized vehicles.

In Chapter 4, we tackle the "last-mile" task by enabling robots to locally navigate to objects accurately without relying on a metric map or object models. High-precision navigation is critical for tasks requiring exact robot positioning, such as docking, inspection, or manipulation. We adopt a robot-centric parametrization for goal specification and design a multi-modal transformer model for egocentric perception and control. By training this system in diverse, realistic simulation environments, it can be deployed directly in the real world without further fine-tuning. We envision these systems being used across various mobile robot platforms for high-precision mobile manipulation tasks in the future.

Chapter 2

Global Navigation via Robot-aware Topological Mapping and Planning

There has been an emergence of learning-based approaches [17, 30, 89, 140, 176] that enable embodied agents to navigate with vision. Compared to the traditional mapping, localization and planning approach [165] that relies on a metric map, these learning-driven systems leverage latent maps or topological maps as scene representations. This eliminates the necessity of meticulously reconstructing an environment which requires expensive or bulky hardware such as a laser scanner or a high-resolution camera. While these approaches show the promises of robots navigating beyond a conventional 2D/3D maps, the problem remains challenging because i) they are hard to scale, ii) they are fragile due to localization error, actuation noise and dynamic obstacles and iii) they usually assume a discrete action space. This makes it challenging to deploy such systems on real robots.

In this chapter, we tackle these challenges by building memory-efficient topological navigation systems that generalize to real-world robots and scenes. To improve the scalability, we present two key ideas: (i) learning controller-dependent reachability to discard redundant information

and (ii) learning behavior embeddings to compress long-horizon visual trajectories. To deploy such systems in the real world, we leverage two-stage hierarchical controllers and train models in realistic simulation environments. Our systems are memory-efficient, adhere to the robot's kinematics, and can handle dynamic obstacles.

2.1 Learning Controller Reachability for Scalable Topological Navigation

In this section, we present a simple and intuitive solution for efficient topological navigation. We show that by accurately measuring the capability of a local controller, robust visual topological navigation can be achieved with sparse experiences (Fig.2.1). In our approach, we do not assume the availability of a global coordinate system or robot poses, nor do we assume noise-free actuation or static environment. This minimalistic representation only has two components: a local controller and a reachability estimator. The controller is responsible for local reactive navigation, whereas the reachability estimator measures the *capability* of the controller for landmark selection and long-term probabilistic planning. To achieve this, we leverage the Riemannian Motion Policy (RMP) framework [127] for robust reactive control and deep learning for learning the capability of the controller from data. We show that with both components working in synergy, a robot can i) navigate robustly with the presence of nonholonomic constraints, actuation noise and obstacles; ii) build a compact spatial memory through adaptive experience sparsification and iii) plan in the topological space probabilistically, allowing robot to generalize to new navigation tasks.

We evaluate our approach in the Gibson simulation environment [178] and on a real RC car. Our test environments contain a diverse set of real-world indoor scenes with presence of strong symmetry and tight spaces. We show that our approach generalizes well to these unseen

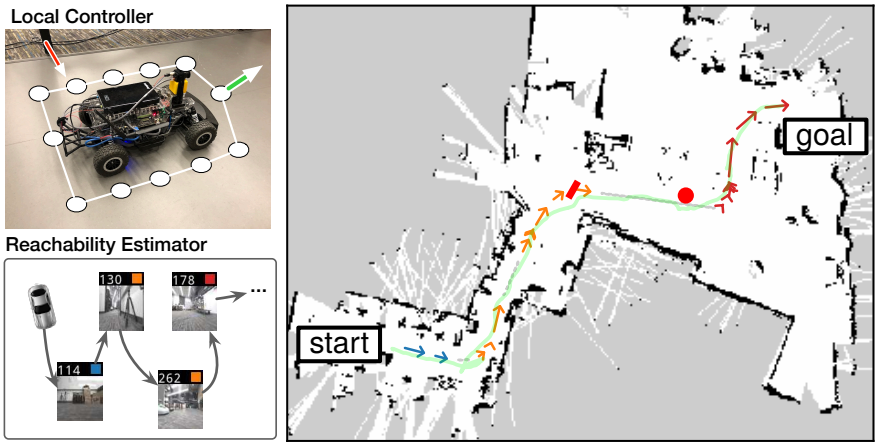


Figure 2.1: Overview of our method. The local controller drives the vehicle towards a given target image, and the reachability estimator plans a path by combining multiple experiences (colored arrows on the map) to provide the controller a sequence of target observations (bottom left) to follow. The vehicle is able to navigate robustly in the real environment (right) while avoiding unseen obstacles (red rectangle and circle). The model is trained entirely in simulation.

environments and surprisingly well to real robots without finetuning. Scalability-wise, our spatial memory grows only when new experiences are unseen, making the system space-efficient and compute-efficient.

2.1.1 Related Work

Cognitive spatial reasoning has been extensively studied both in neuroscience [43, 63, 121], and robotics [88, 114, 166]. The Spatial Semantic Hierarchy [88] divides the cognitive mapping process into four levels: control, causal, topological and metric. In our method, the local controller operates on the control level, whereas the reachability estimator reasons about causal and topological relationship between observations. We omit metric-level reasoning since we are not concerned about building a metric map.

Experience-driven navigation constructs a topological map for localization and mapping [17, 41, 54, 104, 114]. Unlike SLAM that assumes a static environment, the experience graph can

also be used for dealing with long-term appearance changes [99]. This line of works mostly focus on appearance-based localization and ignore the control aspect of navigation, and assume that a robot can always follow experiences robustly. This does not usually hold in unstructured indoor environments, where it is crucial to design a good controller while considering its capability.

Semi-Parametric Topological Memory (SPTM) [140, 141] is a recent work that adopts deep learning into topological navigation. Similar to SPTM, we build a topological map through past experiences. Unlike SPTM that uses image similarity as a proxy for reachability, we measure the reachability of a controller directly. This significantly improves robustness and opens opportunities for constructing sparse maps.

There have been recent works studying visual trajectory following that handles obstacles [13, 70], actuation noise [89], or with self-supervision [122]. Our approach differs from them in that our trajectory follower extends seamlessly to probabilistic planning. Our method also handles obstacles and actuation noise well, thanks to the RMP controller that models local geometry and vehicle dynamics.

Recent works on cognitive planning [61, 62] show that a neural planner can be learned from data. However, assumptions such as groundtruth camera poses are available with perfect self-localization are unrealistic. The use of grid map also limits its flexibility. Another line of research uses reinforcement learning to learn a latent map [115, 116], but it is data-inefficient and cannot be easily applied to real robots. In contrast, our planner is general and can adapt to new environments quickly. It bears a resemblance to feedback motion planning system such as LQR-Trees [163], where planning is performed on the topological map connecting reachable state spaces with visual feedback control.

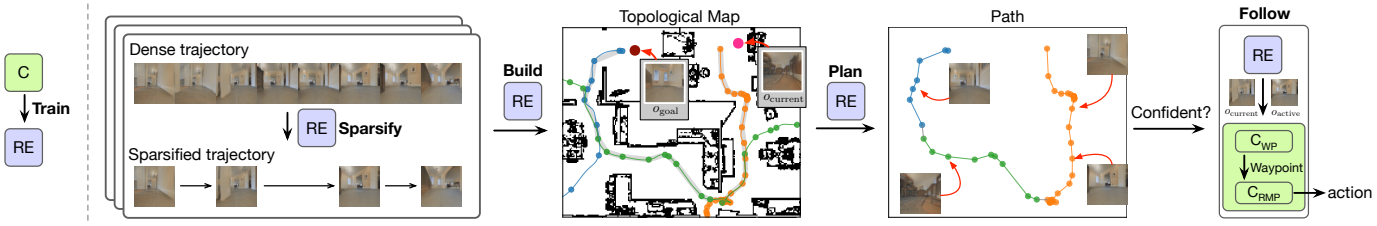


Figure 2.2: System overview. Given a controller $\mathbb{C}$, we train a reachability estimator $\mathbb{RE}$. $\mathbb{RE}$ is used for sparsifying incoming trajectories, building a compact topological map and planning a path. $\mathbb{C}$ and $\mathbb{RE}$ work in synergy to robustly follow the planned path.

2.1.2 Method

Overview

We consider the goal-directed navigation problem: a robot is asked to navigate to a goal G given an observation o_G taken at G. Robot does not have a map of the environment, but we assume it has collected a set of trajectories (e.g., via self-exploration or following language instructions) as its experiences. Each trajectory is a dense sequence of observations $o_1, o_2, ..., o_N$ recorded by its on-board camera. Using its experiences, robot decides the next action to take in order to reach G. The action space is continuous (e.g., velocity and steering angle) and robot could be affected by actuation noise and unseen obstacles.

We approach this problem from a cognitive perspective. Robot first builds a topological map from its experiences. The map is a directed graph, with vertices as observations and edges encoding traversability. Then, given its current observation o_{current} and goal o_G, robot searches for a path on the graph and follows that path to reach G. Our setup is similar to that of SPTM [140]. The difference is that we design our system to make it generalize to real robots and scale to real environments.

For such a navigation system to work, we first need a target-conditioned **Local Controller** $\mathbb{C}$. $\mathbb{C}$ takes current observation and a target observation, and outputs an action $a = \mathbb{C}(o_{\text{current}}, o_{\text{target}})$

to drive robot towards the target. The action is executed for a small time step to get an updated o_{current} and the process is repeated until o_{current} matches o_{target}. Given a path (a sequence of observations) computed by a planner, robot uses $\mathbb{C}$ to follow the path progressively to reach its final destination.

In practice, robot's experience pool can be large and grow indefinitely, thus the key issue is to build a sparse and scalable representation of an environment given dense, unstructured trajectories. Clearly, adjacent observations in a trajectory is highly correlated and it would be wasteful to keep every observation. One ad-hoc approach to sparsify a trajectory is to take every nth observation. However, this assumes that target n steps away is always reachable, which is not necessarily true. For example, without occlusion, an observation far away can be confidently reached (e.g., in a straight hallway), whereas an observation nearby may be hidden (thus not reachable) if it is blocked by obstacles. Moreover, motion constraints, sensor field of view, motor noise, etc. can all affect the reachability of a target.

Our intuition is that the sparsification of a trajectory should adapt to the capability of the controller. We propose learning a **Reachability Estimator** $\mathbb{RE}$ that predicts the probability of $\mathbb{C}$ successfully reaching a target: $\mathbb{RE}(o_{\text{current}}, o_{\text{target}}) = P(\text{reach}|o_{\text{current}}, o_{\text{target}}, \mathbb{C})$. We use $\mathbb{RE}$ as a probabilistic metric throughout the system, illustrated by Fig. 2.2. Given a controller $\mathbb{C}$, we train a corresponding $\mathbb{RE}$. The incoming trajectories are first sparsified by $\mathbb{RE}$ and then interlinked to form a compact topological map. Given o_{current} and o_G, we leverage $\mathbb{RE}$ to plan a probabilistic path and use $\mathbb{C}$ and $\mathbb{RE}$ in synergy to follow the planned path robustly.

Designing a Robust Local Controller

Real-world robots are subject to disturbances such as motor noise and moving obstacles, which can cause a robot to deviate from planned path and fail. Hence our first objective is to design a sufficiently robust local controller. Contrary to directly predicting low-level controls, we split our

controller into two stages: high-level waypoint prediction and low-level reactive control. The high-level controller $\mathbb{C}_{\text{WP}}$ predicts a waypoint x, y (in robot's local coordinate system) for the low-level controller. The waypoint needs not be precise, but only serves as a hint for the low-level controller to make progress. Hence $\mathbb{C}_{\text{WP}}$ is agnostic to robot dynamics (e.g., can be trained with A* waypoints as supervision) and absorbs the effects of actuation noise. For the low-level controller, we adopt the RMP representation [111] as a principled way for obstacle avoidance and vehicle control. Hence we have $\mathbb{C}(o_{\text{current}}, o_{\text{target}}) = \mathbb{C}_{\text{RMP}}(\mathbb{C}_{\text{WP}}(o_{\text{current}}, o_{\text{target}}))$. Note that this allows the same $\mathbb{C}_{\text{WP}}$ to be applied to different robots by replacing the low-level controller.

Fig. 2.3 illustrates the design of $\mathbb{C}_{\text{WP}}$. The robot state is represented by its current observation o_{current}. Denote ith observation in a trajectory as o_i. We represent the corresponding o_{target} at o_i as a sequence of neighbor observations centered at o_i:

$$o_{i-k\Delta T}, o_{i-(k-1)\Delta T}, ..., o_i, ..., o_{i+(k-1)\Delta T}, o_{i+k\Delta T},$$

where k controls context length and ΔT (set to 3) is the gap between two observations. The past frames expand the field of view of o_i which helps controller to do visual closed-loop control. The future frames encode intention at o_i, allowing a controller to adjust its waypoint in advance in order to follow subsequent targets smoothly and reliably.

Technically, we extract a feature vector by feeding stacked $[o_{\text{current}}, o_{i-k\Delta T}, o_{\text{current}} - o_{i-k\Delta T}]$ into a sequence of convolutions, followed by combining the $2k + 1$ feature vectors through one convolution and multiple fully-connected layers to predict a waypoint x, y. We find this design works much better than featurizing each image or stacking all images together. Additionally, the network predicts the heading difference between o_{current} and o_i to help the network anchor the target image in the sequence. Finally, in order to reason about proximity to a target (Sec.2.1.2), $\mathbb{C}_{\text{WP}}$ predicts mutual image overlap. Image overlap is a ratio that represents the percentage of

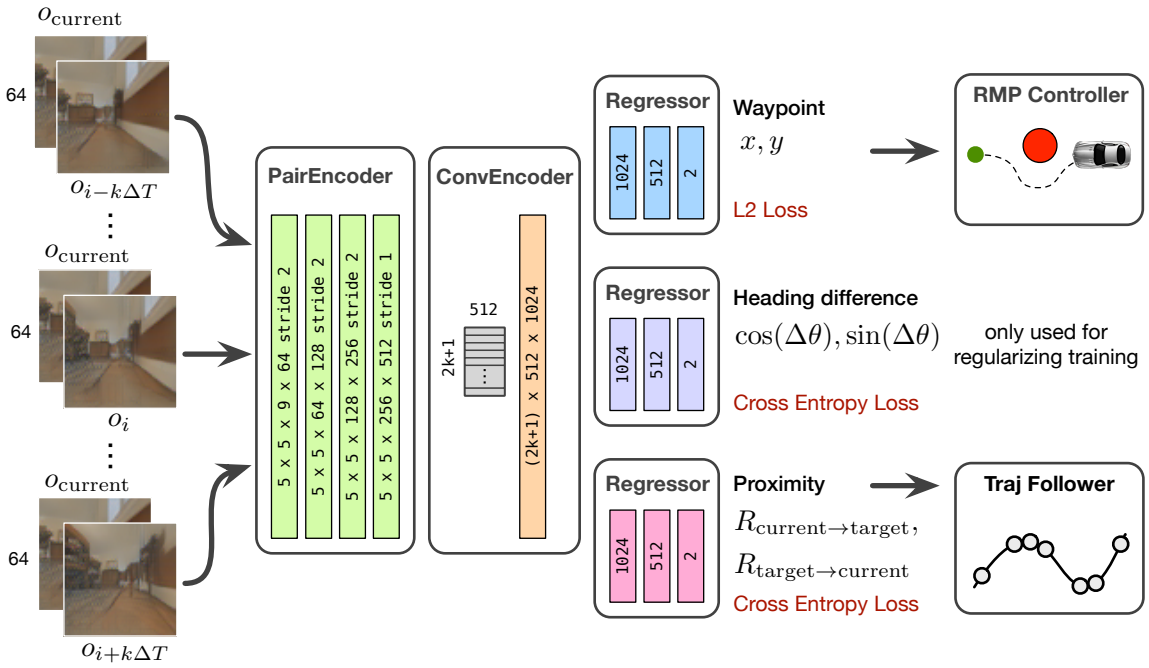


Figure 2.3: Architecture of $\mathbb{C}_{\text{WP}}$. The architecture of $\mathbb{RE}$ is similar, except that it regresses to a single probability and is supervised with cross-entropy loss.

content in one image that is visible in another image. Hence mutual image overlap is a pair of ratios $(R_{\text{current}\to\text{target}}, R_{\text{target}\to\text{current}})$.

We train $\mathbb{C}_{\text{WP}}$ in a supervised fashion (Sec.2.1.3). To evaluate our design, we randomly sample o_{current} from a trajectory and o_{target} being $-1.0m$ behind to $3.0m$ ahead of o_{current} in 4 unseen environments, and run each controller to see if robot reaches the target. Table 2.1 compares the success rate of each controller. Directly predicting low-level controls (forward acceleration and steering velocity) results in much lower success rate than our two-stage design. Compared with directly mapping images to low-level actions, we find it more robust to map images to higher-level abstractions such as waypoints, and then map waypoints to low-level controls using a representation (e.g., RMP) that explicitly models environmental geometry and robot dynamics.

	Control(k=0)	Ours(k=0)	Ours(k=2)	Ours(k=5)
Success%	46%	88%	91%	95%

Table 2.1: Success rate for each controller.

Learning the Reachability Estimator

Table 2.1 suggests that controller design and parametrization can heavily affect target reachability. Unlike [140] that uses image similarity as a proxy, we learn reachability by explicitly predicting the execution outcome of $\mathbb{C}$. During training, o_{current} and o_{target} are randomly sampled from demonstration trajectories (Sec. 2.1.3) and $\mathbb{C}$ is used to drive the robot from o_{current} to o_{target} to get a binary outcome. The criteria for success is that robot reaches the target within time limit defined as $t_{\max} = \text{A}^*(o_{\text{current}}, o_{\text{target}})/v_{\min}$, where $\text{A}^*(\cdot,\cdot)$ computes the A* path length and $v_{\min}$ is the minimum velocity. Hence $\mathbb{RE}$ measures the probability of $\mathbb{C}$ reaching the target *efficiently*, which is independent of the temporal and physical distance between o_{current} and o_{target}. This idea has an interesting connection to feedback motion planning systems [163], as $\mathbb{RE}$ can be seen as estimating visual funnels that are locally stable.

The design of $\mathbb{RE}$ is almost identical to $\mathbb{C}$, except that it predicts a single probability and is trained with a binary classification loss.

Sparsifying a Trajectory

For any observation o_i in a dense trajectory, if $\mathbb{RE}(o_i, o_{i+1}), ..., \mathbb{RE}(o_i, o_{i+k+1})$ are sufficiently high, we could confidently discard $o_{i+1}, ..., o_{i+k}$ because $\mathbb{C}$ does not need them to reach o_{i+k+1}. Hence a greedy approach to choose the next anchor is

$$\begin{aligned} &\max\ j \\ &\text{s.t. } \mathbb{RE}(o_i, o_k) > p_{\text{sparsify}}, \forall k, i < k \leq j \end{aligned}$$

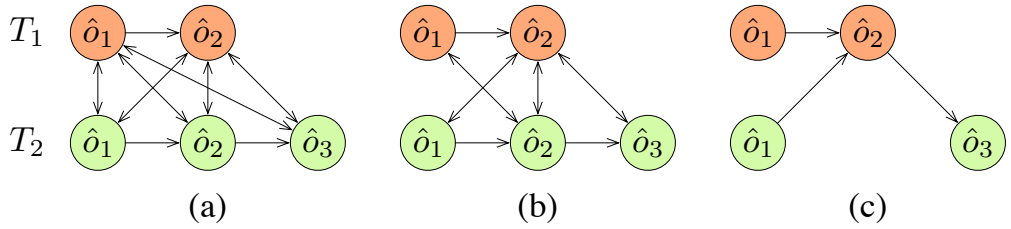


Figure 2.4: A topological map containing two trajectories. (a) densely connected graph. (b) after pruning low-probability edges. (c) after reusing nodes.

where i is previous anchor's position and p_{sparsify} is the probability threshold that controls sparsity. Hence a dense trajectory is converted to a sequence of contextified anchor observations $\hat{o}_1, ..., \hat{o}_m$. One may argue that contextification reduces the effective sparsification ratio. Since the time and space complexity is a function of the number of anchors, in practice it significantly saves computation during planning and following a trajectory, allowing our system to run on a robot in real time.

Building a Compact Probabilistic Topological Map

Our topological map is a weighted directed graph (Fig. 2.4a). Vertices are anchor observations and edge weight from $\hat{o}_i$ to $\hat{o}_j$ is $-\log \mathbb{RE}(\hat{o}_i, \hat{o}_j)$. Construction is incremental: for an incoming trajectory, we create pairwise edges between every vertex in the graph and every anchor in the new trajectory.

Compared to a graph constructed with dense observations, a graph built from sparsified observations has less than 1/10 of the vertices and 1/100 of the edges. To further improve scalability, we propose the following two optimizations to make the graph grow sublinearly to the number of raw observations, and eventually the size of the graph converges:

Edge pruning. Low-probability edges with $\mathbb{RE}(\hat{o}_i, \hat{o}_j) < p_{\text{edge}}$ are discarded since they contribute little to successful navigation (Fig. 2.4b).

Vertex reuse. It is common for two trajectories to be partially overlapped and storing this

overlapping part repeatedly is unnecessary. Hence when adding anchor $\hat{o}_i$ into a graph, we check if there exists a vertex $\hat{o}$ such that $\mathbb{RE}(\hat{o}_{i-1}, \hat{o}) > p_{\text{reuse}}$ and $\mathbb{RE}(\hat{o}, \hat{o}_{i+1}) > p_{\text{reuse}}$. If the condition holds, we discard $\hat{o}_i$ and add edges $\hat{o}_{i-1} \longrightarrow \hat{o}$ and $\hat{o} \longrightarrow \hat{o}_{i+1}$, as illustrated in Fig. 2.4c.

The graph will converge because for any static environment of finite size, there is a maximum density of anchors. Any additional anchor will pass the vertex reuse check and be discarded. Practically however, an environment may change over time. The solution is to timestamp every observation and discard outdated observations using $\mathbb{RE}$. We leave the handling of long-term appearance change as future work.

Planning

We add an edge (weighted by its negative log probability) from o_{current} to every vertex in the graph, and from every vertex in the graph to o_G. The weighted Dijkstra algorithm computes the path with the lowest negative log probability (i.e., the path that robot is most confident). Robot then decides whether the probability is high enough and may run the trajectory follower proposed in Section 2.1.2.

Mitigating Perceptual Aliasing

Practically, o_{current} may correspond to different locations of similar appearances. Traditional approaches usually formulate this as a POMDP problem [165] and try to resolve the ambiguity by maintaining beliefs over states. This requires having a unique state (e.g., global pose) associated with each observation which is difficult to implement since we do not have any metric information.

We use two techniques to resolve ambiguity. The first is to match a sequence of anchors during search and graph construction. In practice the probability of two segments having similar appearances is much lower than two single observations. Additionally we let robot re-plan a new path if it detects discrepancy (entering *Dead reckoning* state for too long) while following the

previous path. The intuition is that the location where robot detects the discrepancy is likely distinct. See Sec. 2.1.3 for an example. In the worst case where such distinctive anchor is absent, robot might follow a cycle of anchors without making progress. The solution is to count how many times the robot has visited an anchor (i.e., by collecting statistics from *last visited anchor*). Cyclic behavior can be detected so that the robot can break the loop by biasing its choice in future planning. We leave the handling of this extreme case as future work.

Following a Trajectory

Our trajectory follower constantly updates and tracks an active anchor to make progress, while performing dead reckoning to counter local disturbances. Specifically, given a sequence of anchor observations $\hat{o}_1, \hat{o}_2, ..., \hat{o}_m$, the trajectory follower acts as a state machine:

Search: robot searches for the best anchor: $\hat{o}^* = \text{argmax}_{o \in \{\hat{o}_1,...,\hat{o}_m\}} \mathbb{RE}(o_{\text{current}}, o)$. If $\mathbb{RE}(o_{\text{current}}, \hat{o}^*) > p_{\text{search}}$, it sets $\hat{o}^*$ as current active anchor $\hat{o}_{\text{active}}$ and enters *Follow* state, otherwise it gives up and stops.

Follow: robot computes the next waypoint $x, y = \mathbb{C}_{\text{WP}}(o_{\text{current}}, \hat{o}_{\text{active}})$ and uses it to drive $\mathbb{C}_{\text{RMP}}$. Meanwhile it tracks and updates the following two values:

- *last visited anchor*. Robot uses the predicted mutual image overlap to measure the proximity between o_{current} and anchors close to $\hat{o}_{\text{active}}$. The closest anchor is set as $o_{\text{lastvisited}}$. This is a form of approximate localization.
- *active anchor*. If $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active+1}}) > p_{\text{follow}}$ and is within proximity, it advances $\hat{o}_{\text{active}}$ to $\hat{o}_{\text{active + 1}}$, otherwise $\hat{o}_{\text{active}} = o_{\text{lastvisited + 1}}$. The intuition is to choose an $\hat{o}_{\text{active}}$ that is neither too close nor too far away.

Normally robot stays in *Follow* state. But if moving obstacles or actuation noise cause $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active}}) < p_{\text{follow}}$, it enters *Dead reckoning* state.

Dead reckoning: robot tracks the last waypoint computed in the *Follow* state and uses the waypoint to drive $\mathbb{C}_{\text{RMP}}$. The assumption is that disturbances are transient which the robot could escape by following the last successfully computed waypoint. Waypoint tracking can be done by an odometer and needs not be very accurate. While in this state, robot keeps checking if $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active}}) > p_{\text{follow}}$ and returns to *Follow* state if possible.

2.1.3 Experiments

We trained $\mathbb{C}_{\text{WP}}$, $\mathbb{RE}$ and all baselines in 12 Gibson environments. 100k training trajectories were generated by running an A* planner (used to provide waypoints) with a laser RMP controller similar to [111]. Simulation step size is 0.1. We use the laser RMP controller as $\mathbb{C}_{\text{RMP}}$ mostly for efficiency, but in practice an image-based RMP controller can also be used [111]. $\mathbb{C}_{\text{WP}}$ was trained by randomly sampling two images on the same trajectory with certain visual overlap, with the A* waypoint as supervision. After $\mathbb{C}_{\text{WP}}$ was trained, we trained $\mathbb{RE}$ by sampling two images that either belong to the same trajectory (prob 0.6) or different trajectories (prob 0.4), and ran a rollout with $\mathbb{C}$ to get a binary outcome. Image size is 64×64 with 120° horizontal field of view. We augmented the dataset by jittering robot's starting location and orientation to improve generalization. About 1.5M samples were used to train $\mathbb{C}_{\text{WP}}$ and $\mathbb{RE}$. Our training setup models a real vehicle similar to [5], so that the same model can be used for real experiments.

We present quantitative results in 5 unseen Gibson environments with diverse appearances. Our baseline is based on SPTM. Since SPTM is designed for small synthetic mazes with discrete action space, its original version would perform poorly in our setting. For a fair comparison, we let SPTM use the same controller and trajectory following logic as ours. The main differences between SPTM and ours are thus: i) how reachability is learned and ii) how graph is constructed and used. Our ablation study will thus be in the form of evaluating trajectory following and

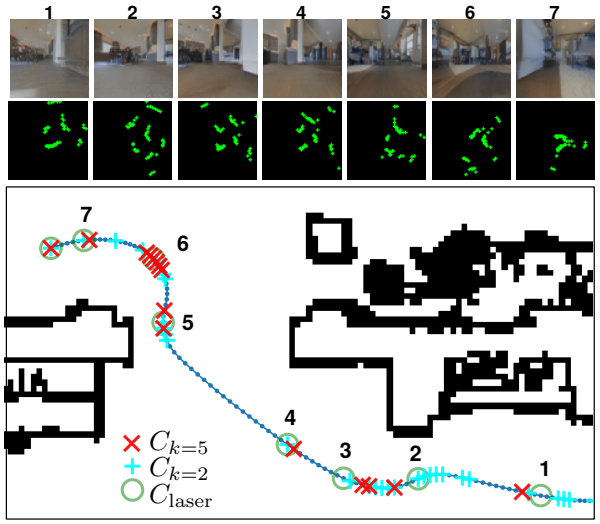


Figure 2.5: Trajectory sparsification. Blue dots: dense observations. Images correspond to numbered locations. 1D laser scans are visualized from the top view.

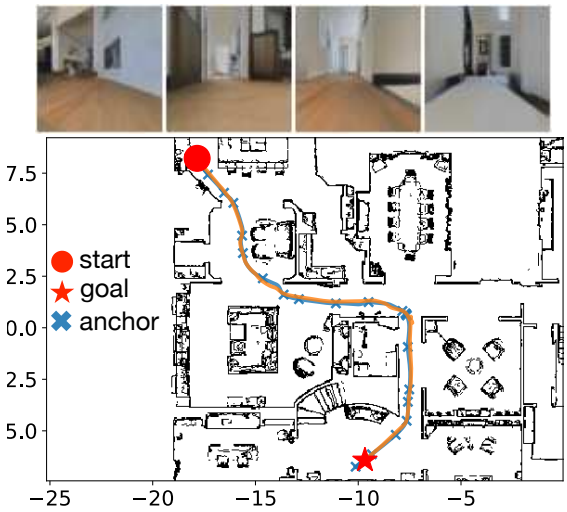


Figure 2.6: Following a 23m long trajectory. Blue trace: groundtruth trajectory. Orange trace is generated by following the anchor points.

planning performance in the following sections.

Trajectory Sparsification

Fig. 2.5 compares sparsification results of three controllers. The two visual controllers $\mathbb{C}_{k=2}$, $\mathbb{C}_{k=5}$ differ in their context length. To show that our model is general, we also trained a laser-based controller $\mathbb{C}_{\text{laser}}$ by modifying the input layer in Fig. 2.3 to take 64-point 240° 1D depth as input.

Fig. 2.5 shows placement of anchors with $p_{\text{sparsify}} = 0.99$. Comparing with $\mathbb{C}_{k=5}$, $\mathbb{C}_{k=2}$ requires denser anchors. Since $\mathbb{C}_{k=2}$ uses a shorter context, it is more "local" and has to keep more anchors to follow a path robustly. Nonetheless, anchors are more densely distributed in tight spaces and corners for both controllers, indicating that our sparsification strategy adapts well to environmental geometry. Interestingly, $\mathbb{C}_{\text{laser}}$ shows a more uniform distribution pattern. Since laser scans have a much wider field of view and measures geometry directly, it is not heavily affected by tight spaces and large viewpoint change.

Trajectory Following

We randomly generated 500 trajectories in the test environments (Fig. 2.6) with an average length of 15 m. When following a trajectory, we stop the robot when it diverges from the path or collides with obstacles. We report the cover rate, the percentage of total length of trajectories successfully followed by robot. For our trajectory followers, $p_{\text{search}} = p_{\text{follow}} = 0.92$.

Sparsity is the average ratio of number of images in a sparsified trajectory to the number of images in the corresponding dense trajectory. To change sparsity, we vary p_{sparsify} for our models. For SPTM we select every nth frame and vary n. Fig. 2.7 plots cover rates for varying sparsity conditions. Controllers with contexts (*-k2, *-k5) achieve higher than 95% cover rate, better than controllers without context (at most 90%). This indicates that having contextual frames can improve robustness. But since contextual frames are used, more observations have to be kept so storage-wise it is not as efficient as (*-k0).

SPTM performs comparably to ours when using a strong controller (*-k5), but for all controllers it starts to degrade before ours as sparsity lowers. Due to its fixed-interval subsampling, it does not adapt to controllers' capability well, as can be seen by the increasing gap between ours and the SPTM counterparts when less contextual frames are used (*-k2, *-k0).

We also evaluated performance under noisy actuation by multiplying a random scaling factor $s \sim \mathcal{N}(1.0, 0.33)$ to the control output. No noticeable difference was found. This is expected because the local controller runs at a high frequency (10 Hz) and uses visual feedback for closed-loop control.

Planning

Navigation between Places We built one topological map for each environment (Fig. 2.8a). A map is constructed from 90 trajectories connecting 10 locations in a pairwise fashion. The

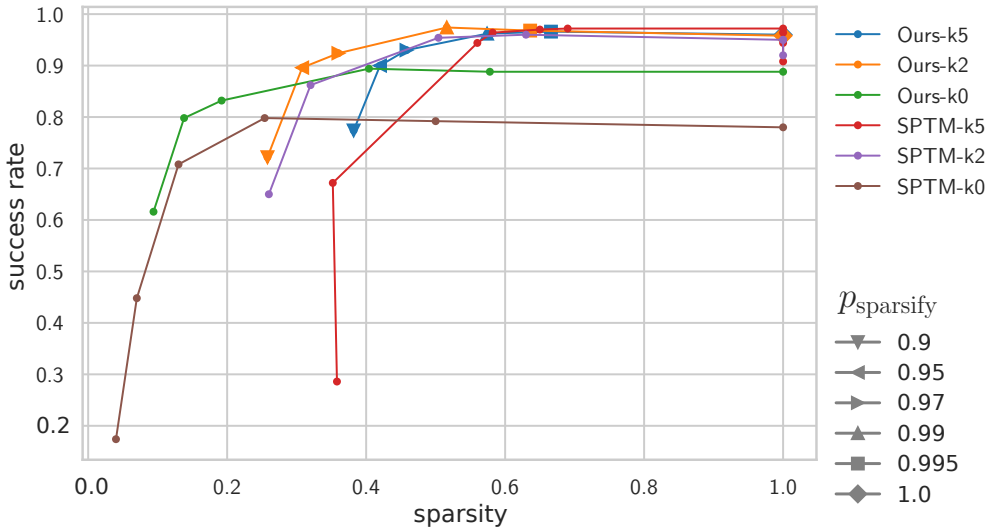
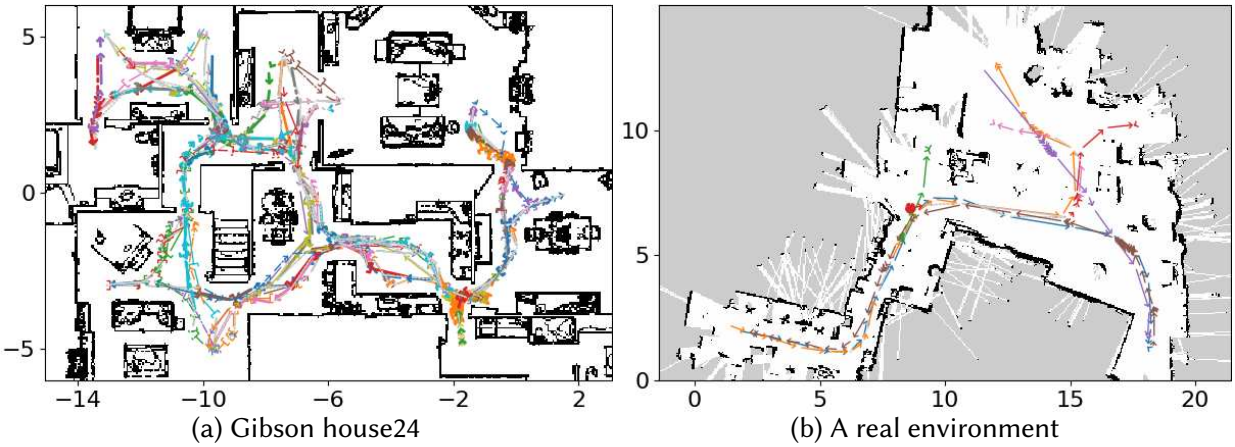


Figure 2.7: Trajectory following cover rate in 5 test environments. Number after k indicates the context length. Data points for Ours-k5 and Ours-k2 are marked with p_{sparsify}.



(a) Gibson house24 (b) A real environment

Figure 2.8: Example topological maps built from sparsified trajectories. Each trajectory is assigned a different color.

	space8	house24	house29	house75	house79
Area	$460m^2$	$207m^2$	$270m^2$	$403m^2$	$205m^2$
Images	30,342	31,167	28,679	39,788	33,617
SPTM	1,648/3,201	1,688/3,668	1,560/3,960	2,116/4,115	1,808/4,756
	48.1%	40.2%	45.6%	51.3%	47.2%
Ours	974/1,482	900/1,348	901/1,467	1,454/2,275	909/1,524
	86.9%	94.3%	91.2%	84.6%	95.7%

Table 2.2: Planning success rate, with #vertices/#edges shown above. Success rate is the outcome of 1,000 navigation trials.

locations are selected to make the trajectories cover most of the reachable area.

Robot starts at one of the locations (with jittered position and orientation) and is given an goal image taken at one of the other 9 destinations. Robot has no prior knowledge of its initial location. We re-implemented SPTM's planner and uses the best trajectory follower SPTM-k5 (Sec2.1.3) to make it a competitive baseline. We set the sub-sampling ratio to 20 and $\Delta T_l = 1$ to prevent the graph from getting too large. We performed a hyperparameter search to set $s_{\text{shortcut}} = 0.99$. For our method, we set $p_{\text{reuse}} = p_{\text{edge}} = 0.99$.

Table 2.2 presents the success rate for each environment compared to SPTM. Our method outperforms SPTM with much sparser maps. Graphs built by SPTM have unweighted edges and do not reuse vertices. SPTM also does explicit localization which sometimes causes planning failure. This results in worse scalability and reliability compared with our approach. Note that the slightly lower success rates in *space8* and *house75* are mostly caused by strong symmetry and rendering artefacts.

Comparing Trajectory Probabilities to Empirical Success Rate To show that path probability is a reasonable indicator of empirical outcome, we let a robot start at a random location (anywhere in a map), plan a path to one of the 10 destinations, and follow the path. 1,000 trajecto-

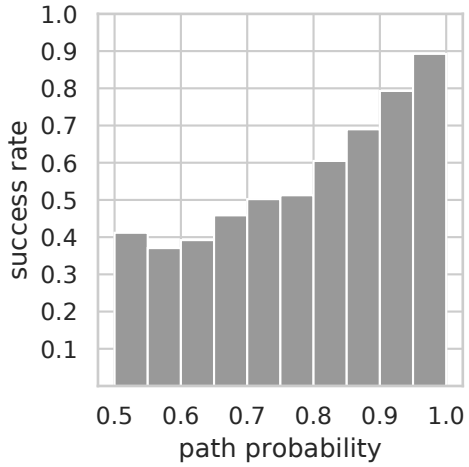


Figure 2.9: Comparing path probability to empirical success rate. Each bar is the average success rate of paths whose probabilities fall into that range.

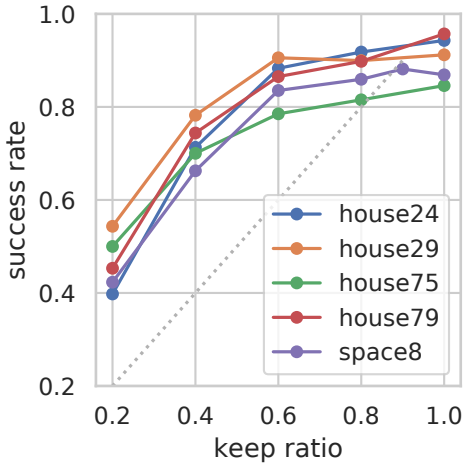


Figure 2.10: Success rate with pruned graphs. Dotted line shows no-generalization for reference.

ries were collected in each environment. Fig. 2.10 shows that path probability strongly corresponds to empirical success rate. This allows a robot to assess the risk before executing a plan, and ask for help if necessary. Note that SPTM does not provide any uncertainty measure.

Generalizing to New Navigation Tasks To test the generalizability of our planner, we randomly pruned the graphs to contain only a subset of the trajectories, and repeated the experiment in 2.1.3. Fig. 2.10 shows that with only 60% of the trajectories, robot already performs close to its peak success rate. In other words, robot is able to combine existing trajectories to solve novel navigation tasks. Fig. 2.11 shows an example.

Resolving Ambiguity Fig. 2.12 illustrates how perceptual aliasing is resolved in environments with strong symmetry. Robot initially starts at an ambiguous location (marked "1") and plans a wrong path (red path). While following this path, robot detects the discrepancy at "2" by realizing what is expected to be an office room is actually a hallway. As a result, it plans a new path (green) whereby it successfully reaches the goal.

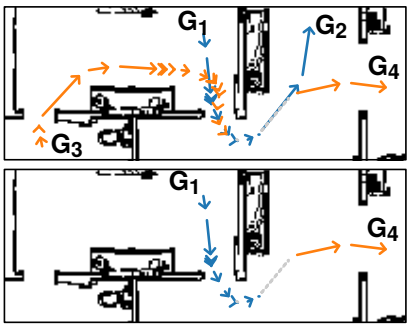


Figure 2.11: Plans a path from G_1 to G_4 by combining $G_1 \rightarrow G_2$ and $G_3 \rightarrow G_4$

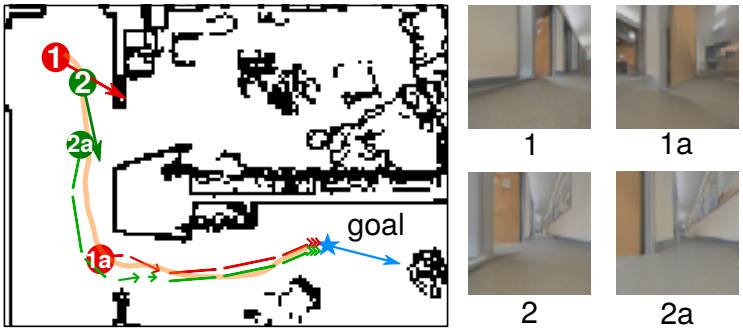


Figure 2.12: Online planning. Arrows indicate headings. 1a and 2a are the next anchor observations for the two paths respectively.

2.1.4 Testing in a Real Environment

Our model trained in simulation generalizes well to real images without finetuning. To map a real environment, we manually drove the RC car to collect 7 trajectories, totalling 3,200 images. The final map contains 206 vertices and 215 edges (Fig.2.8b). The car is able to plan novel paths between locations and follow the path while avoiding obstacles not seen during mapping (Fig.2.1). We refer the interested reader to the video supplementary material for more examples.

2.1.5 Discussion

In this work, we show that by learning the capability of a local controller, robust and scalable visual topological navigation can be achieved. Due to the simplicity and flexibility of our framework, it can be extended to support non-visual sensors and applied to other robotics problems. Future works include combining multiple sensors to improve the controller, developing better algorithms to resolve ambiguity, improving generalization, and extending to manipulation tasks.

The hyperparameters in this approach are mostly probability thresholds, which are easy to interpret and tune. One important scenario our approach does not handle is when robot deviates too much from all vertices in the navigation graph, where it would fail to find a plausible path.

An exploration model can help here, and it can also be used for autonomous map construction.

2.2 Learning Composable Behavior Embeddings for Visual Navigation

In the previous section, we show that by learning the reachability between two images, a robot can discard redundant information to build sparse topological maps. However, there is one undesirable artifact: the robot tends to keep dense observations while undergoing rotational motion, such as turning left or right. This is because while the robot is turning, the image content changes rapidly and thus it requires keeping denser images to maintain enough high reachability. But this is counter-intuitive according to our own experiences: we would not remember the intermediate details while we are turning left or right to enter hallway. In fact, we develop navigation routines and repeat them with little conscious effort: grabbing a cup of coffee from the kitchen, going to the printer room to collect papers, or retrieving mail from the mailbox. We form "muscle memories" to save cognitive load, allowing us to concentrate on more important tasks. From a robot learning perspective, enabling a robot to repeat such navigation routines robustly with minimal guidance is beneficial, because it saves memory, speeds up computation, and opens up opportunities to build sparse visual memory of environments.

Learning high-level behaviors or skills for robots has become an important area of research recently. Most existing works focus on synthetic control tasks or fixed workspace manipulation tasks [87, 103, 119, 149], where environments are fully observable and a robot can be position-controlled. This is hardly applicable to egocentric visual navigation, where environments are partially observable, ground truth states are unavailable, and robots may have non-holonomic kinematic constraints. Because of this, most recent works use predefined behaviors [30, 133, 168],

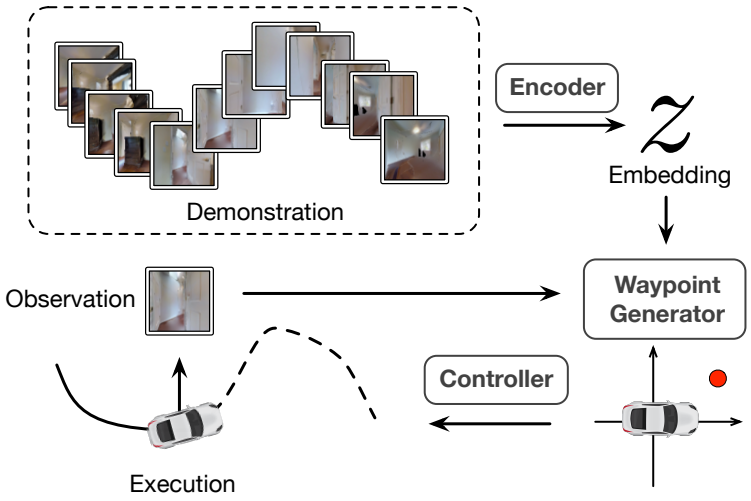


Figure 2.13: High-level overview of behavior learning and execution. CBE learns to embed image sequences to replicate a demonstrated trajectory using visual, closed-loop waypoint generation and control.

with a few attempts on unsupervised or self-supervised behavior learning [59, 90, 160]. However, these behaviors are usually short-horizon (e.g., "turn left"), discrete, and cannot follow precise specifications (e.g., distance to go or angle to turn). Due to these limitations, they are not able to encode *complex and long-horizon* behaviors in a general fashion, such as when following an instruction "go towards northeast by about 5 meters and then turn right to follow the hallway till the end". This limits their applicability in building sparse visual memory for downstream navigation tasks.

To address this problem, we propose *Composable Behavior Embedding*, a robot-agnostic behavior representation for visual navigation (Figure 2.13). At its core is a behavior encoder that compresses a high-dimensional visual demonstration sequence into a low-dimensional embedding. During execution, a waypoint generator is conditioned on the embedding and current observation to generate local waypoints for a low-level controller to replicate the demonstration. The embeddings are learned in an end-to-end fashion by minimizing the waypoint reconstruction loss. It effectively learns to extract path geometry from demonstrations, making it generalize extremely

well to novel environments.

CBE has two desired properties: i) it is compact. The embedding is only 32-dimensional, allowing a robot to build visual memory an order of magnitude smaller than existing approaches [89, 112], and ii) it is composable. A robot can robustly follow a long path via behavior segmentation, or combine behaviors from multiple demonstrations to perform goal-directed navigation tasks.

SLAM [56, 118] can be a strong alternative for building visual memory. CBE has several advantages over SLAM: i) CBE is more than $10x$ efficient at encoding demonstrations than SLAM; ii) CBE works with low-resolution images where SLAM breaks down, allowing it to be deployed on miniature robots without high-quality cameras; iii) CBE has a simpler design with few tuning parameters and is end-to-end trainable. Hence, CBE is a more attractive approach towards building a robust and efficient learning-based visual navigation system.

We show how the embeddings generated by CBE enable a non-holonomic robot to reach goals more than 150 time steps away with no intermediate guidance, even when unseen obstacles are present. We further illustrate how the learned embeddings can be applied to two downstream tasks: one-shot trajectory following and topological mapping. We show that with the learned embeddings we can build visual memory an order of magnitude smaller than existing approaches for these downstream tasks. We conduct detailed quantitative and qualitative analysis to verify our design decisions and how it is compared to a variety of baselines.

2.2.1 Related Work

Visual Navigation. Classical navigation systems rely on building a metric map from laser scans or visual images for robust state estimation, planning, and control [56, 165]. Recent advances in visual navigation move towards non-metric, learning-based methods, such as short-horizon goal-directed navigation [122], path following [70, 89, 168], or building a cognitive mapping system

for planning [29, 61, 112, 140]. Solving long-horizon navigation tasks requires some form of visual memory [49, 70, 89, 112, 140]. Due to visual occlusion, dense observations have to be stored, making it difficult to scale to large environments. Our main contribution is to learn a compact embedding so that a robot only stores a sparse set of visual features. These embeddings serve as sparse visual memory for diverse downstream tasks, such as path following and topological mapping.

Learning from Demonstrations. Perhaps the most direct approach to learn from demonstrations is imitation learning [38, 171]. Imitation learning learns fixed policies that are hard to generalize to novel tasks. Recent works learn latent distributions to encode a diverse skill set [103, 106, 107, 149]. These works focus on manipulation tasks in a fully observable workspace, and hence they cannot generalize to novel, partially observable environments as in indoor navigation. Contrary to existing works that hardcode environments into the skills, we learn a shared behavior encoder, allowing a robot to adapt to new environments quickly.

Unsupervised Skill Learning. Learning high-level skills helps to solve long-horizon tasks more effectively. However, most works on skill learning assume fully observable state spaces in known environments [37, 83, 87, 100, 150]. This is not applicable in egocentric visual navigation, where environments are partially observable and no ground truth robot state is available. So far only discrete, short-horizon navigation skills can be learned [26, 160], and these skills have only been used for exploration and point-goal navigation tasks. To the best of our knowledge, we are the first to show that diverse and long-horizon navigation skills can be effectively learned from visual data. Moreover, we show that these skills can serve as building blocks for constructing a sparse persistent spatial memory for navigating in novel environments.

Sequence-to-Sequence Models. Our method is inspired by seq2seq models, which have been widely used in language processing [8, 159] and trajectory prediction [6, 96]. An important distinction is that we save the latent states as part of the visual memory. Moreover, our decoding

process generates controls that are conditioned on the current rollout, which is essential for correcting drift and avoiding obstacles.

2.2.2 Composable Behavior Embedding (CBE)

Overview

We consider a goal-directed navigation task where a robot needs to navigate from its current location s to a goal g. We assume that a demonstration containing a sequence of RGB observations $o_1, o_2, ..., o_T$ connecting s and g is given to the robot. Since the trajectory can be long and complex, intermediate information needs to be memorized to help the robot follow the demonstration [70, 89, 112].

Similar to [111], our navigation system guides the robot by generating *relative waypoints* that are used by a low-level controller to compute motor commands (e.g., velocity and steering angle). To follow a demonstrated trajectory, the robot could use visual control to match its observations against the sequence of demonstrated observations as in [70, 89, 112]. However, such an approach is highly memory inefficient, since it requires rather densely stored images. To overcome this problem, CBE encodes the sequence of visual observations $o_1, o_2, ..., o_T$ into a low-dimensional behavior embedding z_D (Figure 2.13). During execution, at each time step, a waypoint generator uses z_D and the current observation to produce a waypoint for the low-level controller. Both state and action space are continuous, and the system operates in a closed-loop fashion to correct any drift due to noise and non-holonomic kinematic constraints. Different demonstrations and their executions can be of different lengths.

For very long and complex trajectories, a single z is insufficient due to compounding errors. To address this problem, we segment long trajectories into sequences of embeddings, interleaved by visual attractors for state calibration. The segmented behaviors are used for solving downstream

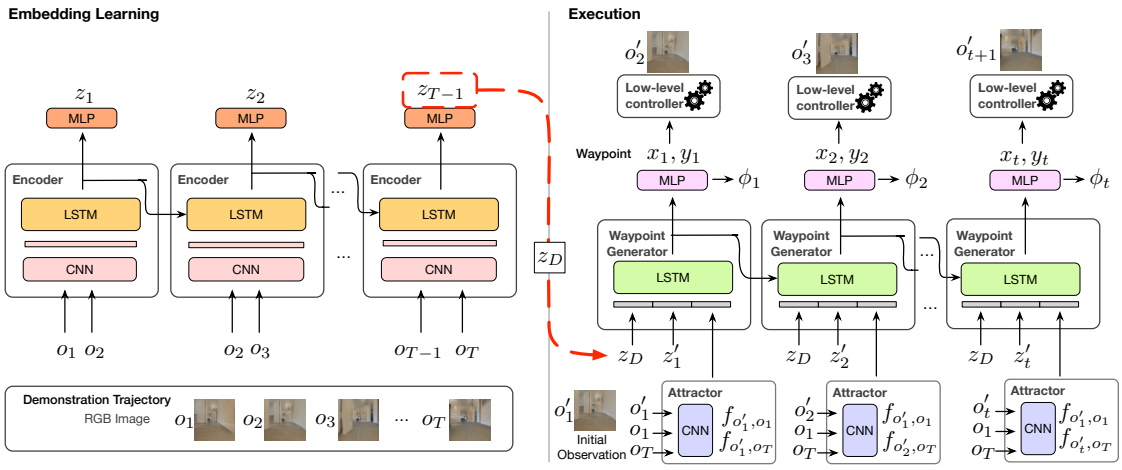


Figure 2.14: Overview of CBE. The encoder compresses the image sequence $o_1, ..., o_T$ observed during a demonstration into a low dimensional embedding z_D. This is done via a recurrent LSTM network that inputs pairs of consecutive images, (o_t, o_{t+1}), and generates a sequence of latent embeddings, z_t, with the final z_{T-1} providing the overall embedding of the demonstration. In the execution phase, the waypoint generator uses the demonstration embedding, z_D, and the embedding of the images observed so far, z'_t, to generate the next waypoint, (x_t, y_t), and a measure of the progress thus far ϕ_t. The embeddings of the executed trajectory, z'_t, are computed using the same network as the demonstration encoding. An additional "Attractor" network processes the current image and the first and last images of the demonstration to provide information that helps with the alignment at the beginning and end of the trajectory. At each time step t, the waypoint is sent to the local controller, which moves the robot and provides the image for the next iteration, o'_{t+1}. This process is repeated until the robot reaches the goal o_T, indicated by $\phi = 1$.

navigation tasks, which we detail in Sec 2.2.4.

Learning Continuous Navigation Behaviors

The behavior encoder $\mathbb{B}_{\text{enc}}$ (left half of Figure 2.14) maps observation streams $o_1, o_2, o_3, ..., o_T$ into a low-dimensional embedding $z_D = \mathbb{B}_{\text{enc}}(o_1, o_2, ..., o_T)$. To do so, each pair of adjacent images is input to a CNN that generates a feature vector fed into an LSTM to compute the embedding for each time step. Since the encoder is recurrent, it outputs a sequence of embeddings, where embedding z_i encodes the observed behavior from o_1 to o_{i+1}. The complete trajectory is encoded

into z_{T-1} (i.e., z_D).

Since encoding and execution are only coupled by the embedding, the whole CBE network can be trained end-to-end. Through end-to-end learning, the encoder learns a common behavior manifold (Sec.2.2.4). The embedding can be extremely low-dimensional (e.g., 32), which significantly saves memory compared to SLAM [118] or other learning-based approaches [89, 112].

Behavior-Conditioned Waypoint Generator

The waypoint generator (right side of Figure 2.14) executes a behavior while tracking the robot's progress. The robot starts with its initial observation, o'_1, which does not have to exactly match the beginning of the demonstration, o_1. At every time step t, an LSTM unit takes as input the embedding z_D of the demonstration and the embedding z'_t of the images observed so far (computed using the same encoder network used for demonstration embeddings), along with features provided by an "Attractor" network described below. Using these, the recurrent unit predicts the next local waypoint, x_t, y_t, and the current progress, ϕ_t. The waypoint is input into the robot's low-level controller to generate motor commands. The low-level controller can be a simple PID controller, or it may support local obstacle avoidance [111]. The *progress indicator* ϕ_t provides the fraction of the demonstration the robot has completed at time t. It is used as a condition for behavior switching. After receiving the next observation, o'_{t+1}, new attractor features and embedding z'_{t+1} are computed and input to the next LSTM step. This process is repeated until the robot reaches the goal, indicated by $\phi = 1$.

Attractor Network. During execution, the robot's initial location and orientation may not exactly match the beginning of a demonstration, requiring the robot to align its initial location sufficiently well to follow the demonstrated trajectory. Similarly, to determine when the robot has reached the goal, solely accumulating motion information from the observed images is not accurate enough. CBE solves these problems via the *attractor* network, which combines the robot's

current observation o'_t with o_1 and o_T (i.e, attractors) to provide features that can relate the current observation to the beginning and end of the demonstration. The attractor network is a CNN that generates $f_{o'_1,o_1}$ and $f_{o'_t,o_T}$ which are concatenated with the embedding (see Figure 2.14 and 2.16).

Long Range Navigation via Behavior Segmentation

Since behavior embeddings are learned from egocentric observations, compounding error is inevitable, implying that z may not encode a complex long-horizon behavior precisely. We solve this by segmenting a long trajectory into a sequence of behaviors, each of which is specified by its embedding z_D and initial and final observations, o_1 and o_T, respectively. Via the attractor features, o_1 provides robustness toward noisy locations when starting a behavior, and o_T helps the behavior reach the goal location accurately enough to transition to the next behavior (related to funnels in LQR-Trees [162]).

We find fixed-distance segmentation works well in practice (Sec. 2.2.4). Given an observation sequence $o_1, o_2, ...o_T$, we segment it into equally spaced segments, subject to the constraint that every segment contains no more than K observations, where K is determined by a validation set. Visual attractors are placed at the segmentation boundaries, and two adjacent segments share attractors.

Behavior Switching. When a robot executes a sequence of behaviors, it needs to know when it can safely switch from the current behavior to the next. It makes the switching decision by checking if the progress indicator of executing the current behavior ϕ_{current} is close to 1 (set to 0.95 in practice). If the condition holds, the robot resets its internal states and starts executing the next behavior z_{next}.

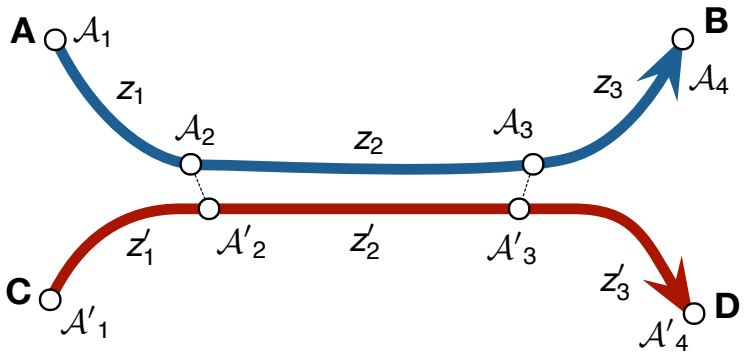


Figure 2.15: Linking attractors from two demonstrations. $\mathcal{A}_i$, $\mathcal{A}'_j$ are attractors. z_i, z'_j are embeddings between attractors. Dotted lines are connections between attractors that are visually close.

Composing Behaviors from Multiple Demonstrations

The segmentation method described in Sec. 2.2.2 can be extended to enable a robot to re-compose behavior segments from multiple demonstrations. In Figure 2.15, a robot is given demonstrations $A \rightarrow B$ and $C \rightarrow D$. If the attractor $\mathcal{A}_2$ and $\mathcal{A}'_2$ are close enough, then the robot can execute behaviors z_1, z'_2, z'_3 sequentially to go from A to D, even though no direct demonstration is available. This also allows us to further compress demonstrations by removing repeated behaviors (e.g., one of z_2 and z'_2).

Learning Choice Points. Fixed-distance segmentation does not guarantee that visual attractors from different demonstrations are placed at consistent locations, making it difficult to connect demonstrations. To mitigate this, we use a simple algorithm to find spatially consistent attractors. We train a classifier $d_t = \mathbb{C}(o_{t-k+1}, ..., o_t)$ that takes the most recent k observations and predicts the next waypoint direction (discretized into 128 bins) using the training dataset. We compute the variance σ_t of the directional distribution to measure the uncertainty. Intuitively, o_t with high variance suggests that future trajectories may diverge and thus o_t is usually associated with spatially consistent locations such as intersections and doorways. Given a trajectory $o_1, ..., o_T$, we use $\mathbb{C}$ to compute the directional variances $\sigma_1, ..., \sigma_T$. Then we use a peak finding algorithm to

find a set of choice points along a trajectory. While there are more sophisticated methods such as [102] that can potentially find better choice points, we find this simple approach to be effective (see Sec. 2.2.4).

2.2.3 Implementation Details

We collected 100k trajectories from 18 large Gibson [177] environments as the training set. The trajectories are generated by a laser-based RMP controller [111] driving a non-holonomic car to follow a sequence of local waypoints computed by an A* planner. This controller also serves as the low-level controller for behavior execution. The low-level controller uses laser scans for local obstacle avoidance and in practice it could be replaced with vision-based controllers [70, 111] at extra computational cost. Simulation runs at 10 Hz. Image resolution is 64×64 with 120° field of view. Camera height is set to 1.0 m above the floor. All evaluations are conducted in 5 large unseen Gibson environments. These large scenes are several times the size of an average Gibson scene, hence they are more suitable for evaluating long-horizon navigation performance.

We use a sequence length of 64 with a frame gap uniformly sampled between 0 to 2. Hence the average trajectory length is 128 time steps. We use the local waypoints in the same training set as supervision, and adopt DAgger [134] for data augmentation. In the DAgger phase, we jitter the robot's initial pose to simulate imperfect alignment and collect rollout trajectories generated by the current model. We then compute the correct waypoints and progress to train the next model. The correct local waypoints are computed by transforming (i.e., rotating and translating) the global ground truth waypoint associated with the closest trajectory sample to the robot. To compute the correct progress, we define the completed path as $o_1, ..., o_k$ where o_k is the closest observation to o'_t (in Euclidean distance). Hence ϕ_t is the fraction of completed path length to the total path length. By jittering the robot's pose in the DAgger phase, the robot learns a closed-loop

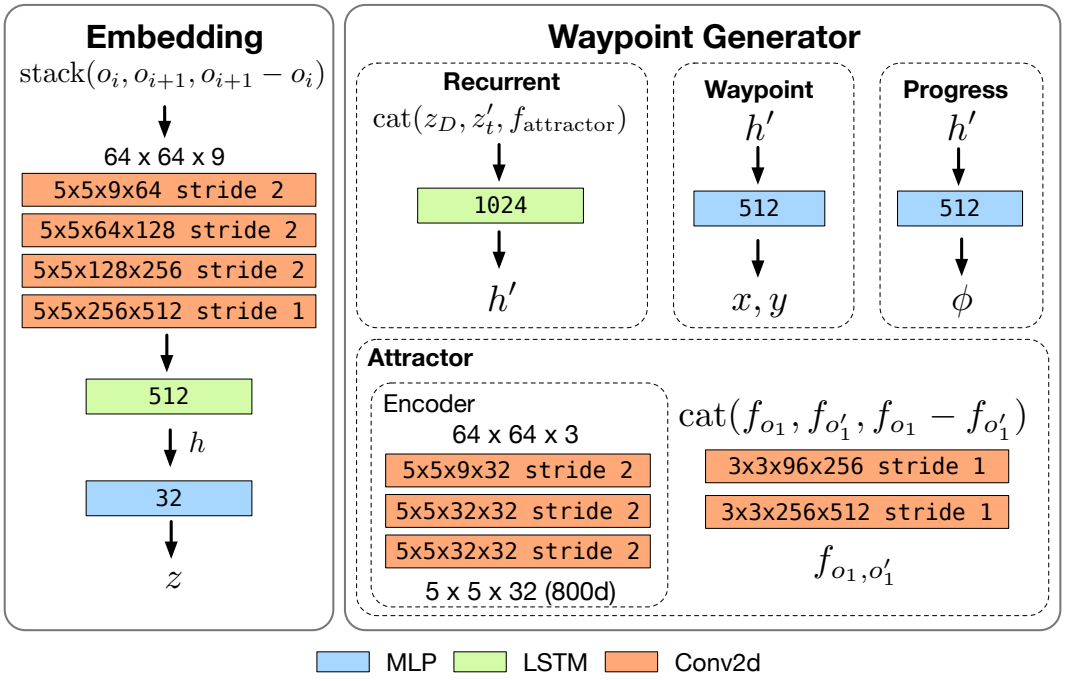


Figure 2.16: Neural networks used by each component in CBE. See Figure 2.14 for how these components work together.

policy that is robust to drift. This also enables the robot to robustly switch to the next behavior segment albeit the initial misalignment and errors in progress estimation by relating current observation to the attractors.

Network designs. Figure 2.16 details the network architectures of CBE modules. The networks are lightweight (70 MB) and can run in real time on an embedded system. We use the Adam optimizer with a learning rate of 0.0003 and a learning rate decay of 0.7. Every epoch contains 200k samples. We trained CBE for 5 epochs. All baselines were also trained using the same dataset for 5 to 7 epochs.

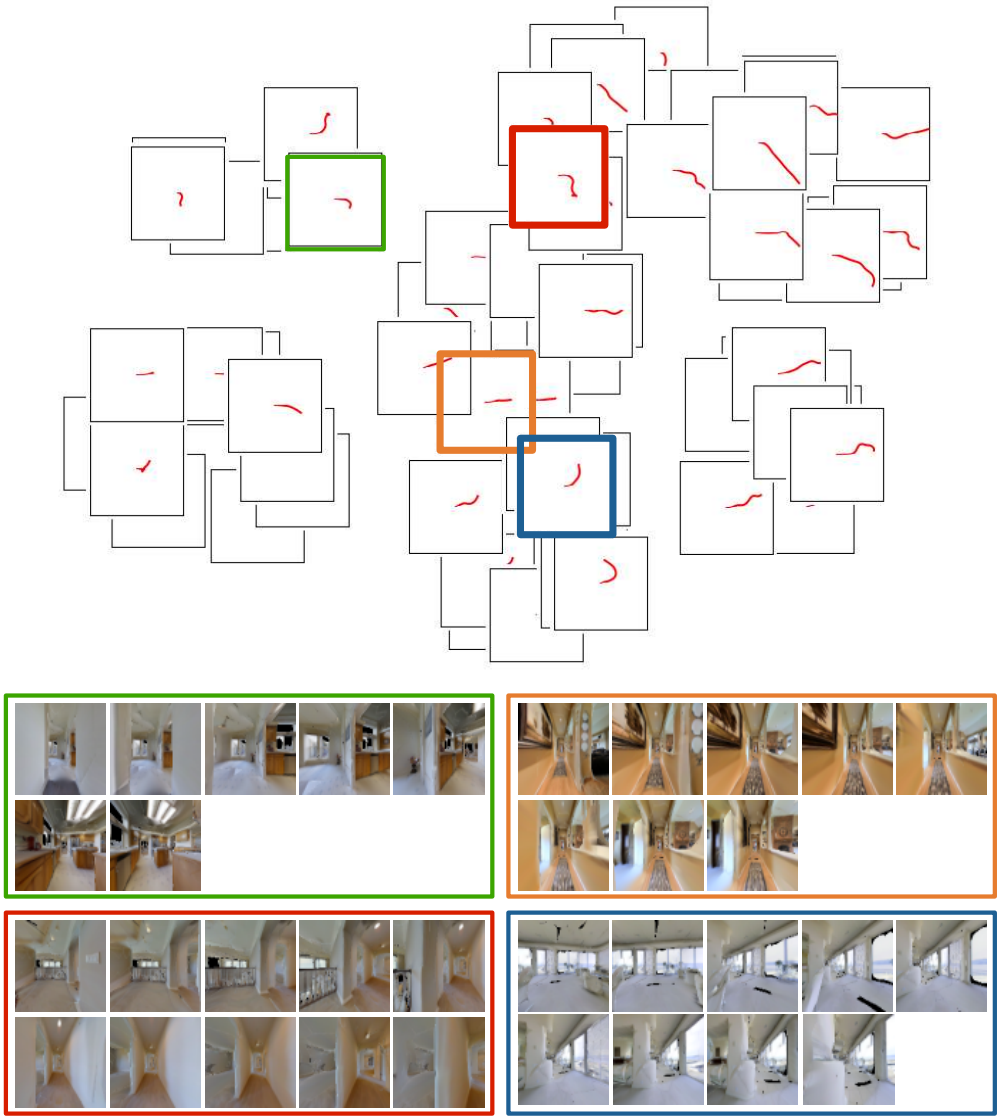


Figure 2.17: t-SNE visualization of the behavior manifold. Each red line visualizes an encoded trajectory. The initial pose of the robot is always at the center, pointing rightwards. 4 example trajectories are shown at the bottom.

2.2.4 Experimental Results

Behavior Embedding

Figure 2.17 shows the t-SNE plot of embeddings extracted from training trajectories. The plot shows that the embedding space encodes a meaningful behavior manifold. From left to right, trajectory lengths are increasing. From top to bottom, there is a smooth progression from "right turns" to "going straight" and to "left turns". The embedding space learns to encode visual odometry, even though it is not explicitly told to do so. We think this is why the learned embeddings generalize well to novel environments and can encode long-range behaviors, while being low-dimensional.

Single-behavior Navigation

We study how well a robot can navigate between two locations with a single CBE behavior in unseen environments. We collected a set of trajectories of lengths ranging from 16 time steps to more than 200 time steps, with 500 trajectories collected for each time step. We extract an embedding from each trajectory to condition the waypoint generator. We jitter the robot's initial pose to simulate imperfect alignment. We compare with the following baselines:

Visual SLAM We adopt ORB-SLAM2 [118] which is one of the state-of-the-art real-time SLAM methods. We first feed the image sequence to reconstruct the environment and the trajectory. During execution, we run SLAM in tracking mode which localizes and tracks the pose of the robot. We set the next waypoint to be the point on the trajectory that is 5 keyframes away from the robot's current location. If localization fails, the robot will use the previously computed waypoint until localization succeeds.

RPF RPF [89] extracts a feature vector from each observation and uses attention to track the progress of a robot. Original RPF assumes the availability of camera pose and action at each time

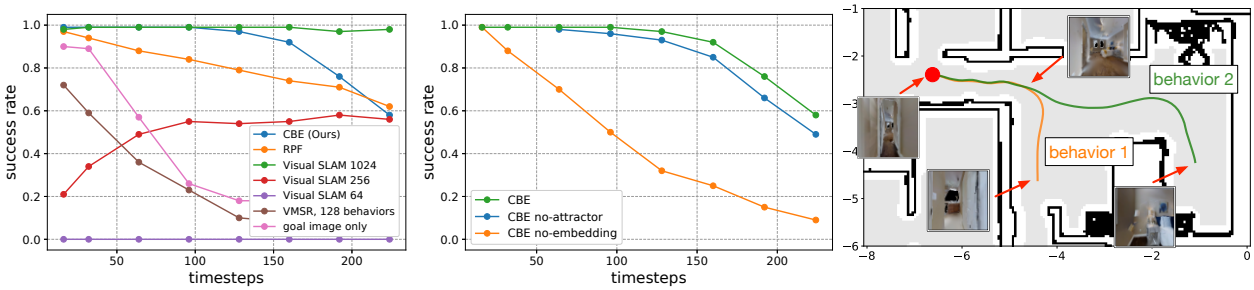


Figure 2.18: Evaluating CBE for single-behavior navigation. (a) Comparing CBE with baselines. The abnormal degradation of Visual SLAM 256 for short trajectories is due to initialization failures. (b) Model ablation. (c) Example rollouts of two behaviors with similar structures.

step. Here we only assume RPF has access to visual observations, same as ours.

VMSR VMSR [90] clusters fixed-length demonstrations into a discrete set of behaviors. To support variable-length trajectories, we use a recurrent encoder similar to ours instead of a convolutional encoder. Again, VMSR uses raw observations as input.

Goal image only we use the local controller in [112] because it shows strong performance when the goal image is visually reachable. We will compare [112] against CBE in Sec. 2.2.4 for its path following performance.

Figure 2.18a compares the success rates of CBE against the baselines (we also experimented with the SPL [7] metric with almost identical results). Using goal image alone shows poor performance due to visual occlusion. CBE achieves $> 95\%$ success rate for trajectories of up to 128 time steps (approx. 6 m in metric length). While CBE degrades for longer trajectories, we perform segmentation to maintain strong performance (Sec. 2.2.4). RPF relies on accurate attention to track a path, but drift in attention may cause RPF to lose track and deviate from the path. VMSR clusters input trajectories into a discrete set of behaviors, hence it cannot capture the variations of behaviors well. While Visual SLAM outperforms single-embedding CBE for long trajectories,

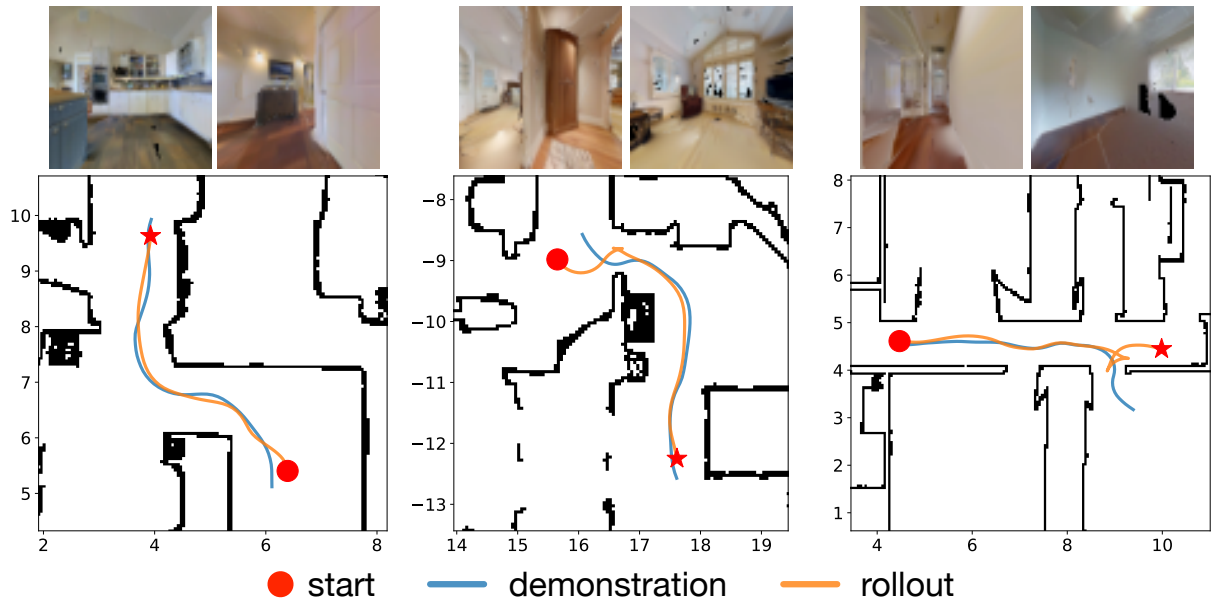


Figure 2.19: Example traces (including one failure case) in test environments using a single behavior embedding. Start and goal images are shown at the top. Note that starting locations of the robot are not always aligned with the beginnings of the demonstrations.

it degrades quickly as resolution decreases (degraded by 50% with 256×256 images and failed completely with 64×64 images). In contrast, CBE works well with low-resolution images, and can potentially be deployed on miniature robots with fast-moving cameras.

Figure 2.18c shows how learned embeddings can distinguish between two similar behaviors. These two behaviors share the same structure: go straight and turn right. However, behavior 2 needs to go straight for a longer distance before turning right. CBE captures the difference in distance so that a robot can reach both locations with no ambiguity. Figure 2.19 shows example traces.

Model ablation. Figure 2.18b compares CBE with two variants. Removing the attractor model is detrimental because it would not be able to capture the initial misalignment. This effect is more pronounced when following a sequence of behaviors (Sec. 2.2.4). Removing the embedding significantly degrades performance, as the robot has to rely on the goal attractor which can be

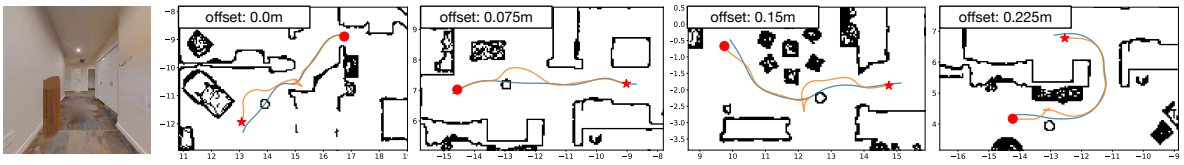


Figure 2.20: Handling unseen obstacles during behavior execution. Left image: robot's view of the obstacle. 4 example executions with different obstacle offsets are shown on the right. Blue trajectory: demonstration. Orange trajectory: rollout. Red dot: starting location.

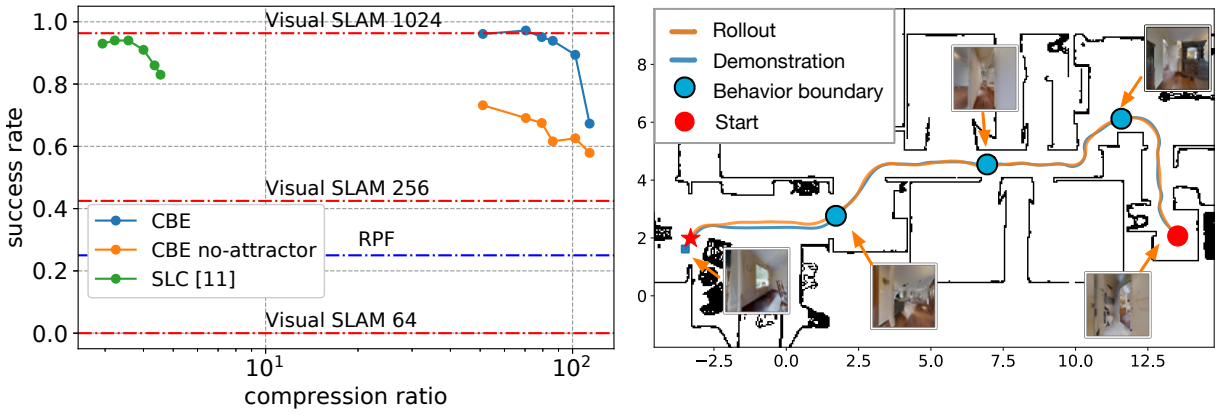
occluded in long trajectories.

Robustness to unseen obstacles. To understand the robustness of our model in a dynamic environment, we randomly place a trashcan of size $0.3 \times 0.3 \times 1.0$ m close to a trajectory and let the robot execute the corresponding behavior (128 time steps). Note that the behavior is encoded when the obstacle is not present. We evaluated our model on more than 300 trajectories and the following table shows the results by varying the offset of the obstacle to trajectories:

offset (m)	0.0	0.075	0.15	0.225
Success%	78.8	82.6	85.1	89.0

Figure. 2.20 shows example executions. The robot can successfully avoid most of the obstacles and reach the goal. The low-level controller *deliberately* makes the robot deviate from the demonstration to avoid the obstacle, but since CBE uses visual feedback to follow the encoded trajectory, it can generate corrective waypoints to get the robot back on track. Note that our model is trained without obstacles. Training the model with obstacles could further improve its robustness and we leave it as future work.

Robustness to actuation noise. We apply a random scale u to the controls of the robot at every time step. We compute $u = \text{clip}(x, -s, s) + 1.0$ where $x \sim \mathcal{N}(0.0, s/2)$. Intuitively, $s = 0.5$ means that we apply a +/- 50% random scale to the velocity and steering angle (independently). We observed 2% and 8% degradation at $s = 0.5$ and 1.0, respectively. This shows that our model is robust to actuation noise.



(a) (b)

Figure 2.21: (a) Comparing success rates at different compression ratio. For Visual SLAM and RPF [89] we only show single success rates because they do not subsample observations. (b) CBE robustly follows a long path by segmenting the path into a sequence of behaviors. See the supplementary video for more examples.

Long-horizon Visual Path Following

A common navigation task that robots perform is to navigate between two places [70, 89, 112]. We show that by incorporating behaviors, a robot can follow a long trajectory with very sparse guidance. We sparsify a trajectory by segmenting it into a sequence of behaviors (Sec. 2.2.2). We compare with [112] that sparsifies a trajectory by reasoning about target reachability. Compression ratio is defined as T/N, where N is the number of landmarks and T is the total number of observations. For CBE, N is approximately equal to the number of behaviors. We vary segment length K in Sec. 2.2.2 to adjust the number of behaviors. For Visual SLAM and RPF, we only report their success rates.

We selected semantically meaningful locations (e.g., rooms) in each test environment as starts and goals and generated 500 long trajectories with an average length of 20 m. A robot follows each trajectory with a jittered initial pose. Figure 2.21a compares the trajectory following success rates at different sparsity levels. Incorporating behaviors significantly increases sparsity compared

Method	Avg. Mem (KB)	SR	Breakdown of memory usage per trajectory (average)
CBE (Ours)	**19**	**97.2**	6 attractors (3.2 KB each) + embeddings (32 floats = 128 bytes each)
SLAM [118] 1024×1024	341	96.3	10933 descriptors (32 bytes each)
256×256	199	42.5	6382 descriptors.
64×64	-	0.0	Failed to initialize.
RPF [89]	424	21.7	212 feature vectors (512 floats = 2 KB each)
SLC [112]	1548	93.7	129 images (12 KB each)

Table 2.3: Comparing memory efficiency and success rate (SR) of different methods for long-horizon visual path following. All methods use 64×64 images except for SLAM which we evaluate on multiple resolutions. Note that SLAM requires extra memory to store the pose graph, which we did not include here due to the difficulty of estimating the value accurately.

to a behavior-less approach [112]. Without visual attractors, CBE performs considerably worse, as the robot is not able to calibrate its state well to switch to the next behavior reliably. Visual SLAM performs competitively with high-resolution images, but fails completely when using the same low-resolution images as other baselines. Figure 2.21b shows a qualitative example, where a 20 m long trajectory (450 time steps) is segmented into four behaviors and the robot executes the behaviors sequentially to reach the goal.

Memory efficiency. Table 2.3 shows that CBE is at least $10x$ more efficient at encoding visual demonstrations than the baselines. A trajectory sparsified by CBE usually contains fewer than 10 embeddings (32 floats each), interleaved by visual attractors (800 floats each). In comparison, Visual SLAM stores over ten thousand feature descriptors, and existing learning-based methods require storing either dense visual features or raw images. This opens up opportunities to build compact topological maps of novel environments, studied next.

Behavior-based Topological Mapping

We follow the same setup as in [112, 140], where a robot builds a topological map of an environment from a set of experience trajectories consisting of RGB observations. A topological map is a directed graph where vertices are anchor observations selected from the trajectories and edges encode

Envs	**Calavo**			**Frierson**			**Kendall**			**Ooltewah**			**Sultan**		
#images	29,449			48,835			51,059			80,394			31,685		
	#verts	storage	SR	#verts	storage	SR	#verts	storage	SR	#verts	storage	SR	#verts	storage	SR
SPTM [140]	3067	6.0	3.3	5097	10.0	0.0	5320	10.4	0.0	8287	16.2	2.2	3290	6.4	0.0
SLC [112]	617	36.1	**97.8**	935	54.8	90.4	805	47.2	98.1	1115	65.3	**96.7**	759	44.5	86.7
CBE (Ours)	**357**	**1.2**	**97.8**	**498**	**1.6**	**100**	**490**	**1.6**	**100**	**611**	**2.0**	92.3	**388**	**1.3**	**98.9**

Table 2.4: Comparing sizes of topological maps and planning success rates. Storage is in MegaBytes. SR indicates planning and trajectory following success rate. For SPTM we do a 10x subsampling of input observations. SLC does adaptive subsampling with a sparsification threshold of 0.98. For CBE we use $K = 100$ for creating behavior segments.

connectivity. This graph structure is often used in goal-conditioned navigation tasks, where a robot needs to plan a least-cost path to get to a specified goal.

We leverage behaviors to build sparse and well-connected topological maps. We first perform choice-point based segmentation as described in Sec. 2.2.2, followed by distance-based segmentation (Sec. 2.2.2) if needed. Each vertex stores an 800-dim attractor feature. Edges are either behavioral edges (via segmentation) or proximal edges (created by linking attractors). A robot first localizes itself and the goal using a network that predicts visual overlap [112]. Then the robot uses the Dijkstra algorithm to find the shortest path and executes the sequence of behaviors along the path.

We selected 10 to 14 semantically meaningful locations (e.g., rooms) in each test environment and collected pairwise trajectories to cover most of the traversable area. We built a topological map out of these trajectories. For planning, we let a robot start at one of the locations, plan a path to every other location, and follow the path. Table 2.4 compares the sizes of topological maps built by CBE and planning success rates against the baseline methods. CBE builds much more compact maps and is significantly more memory efficient. SPTM [140] subsamples observations and extracts a 512-dim feature vector from each observation. However, it is not robust enough for controlling a non-holonomic robot in a continuous state and action space. SLC [112] performs competitively, but at the expense of storing significantly more information per vertex (five 64×64

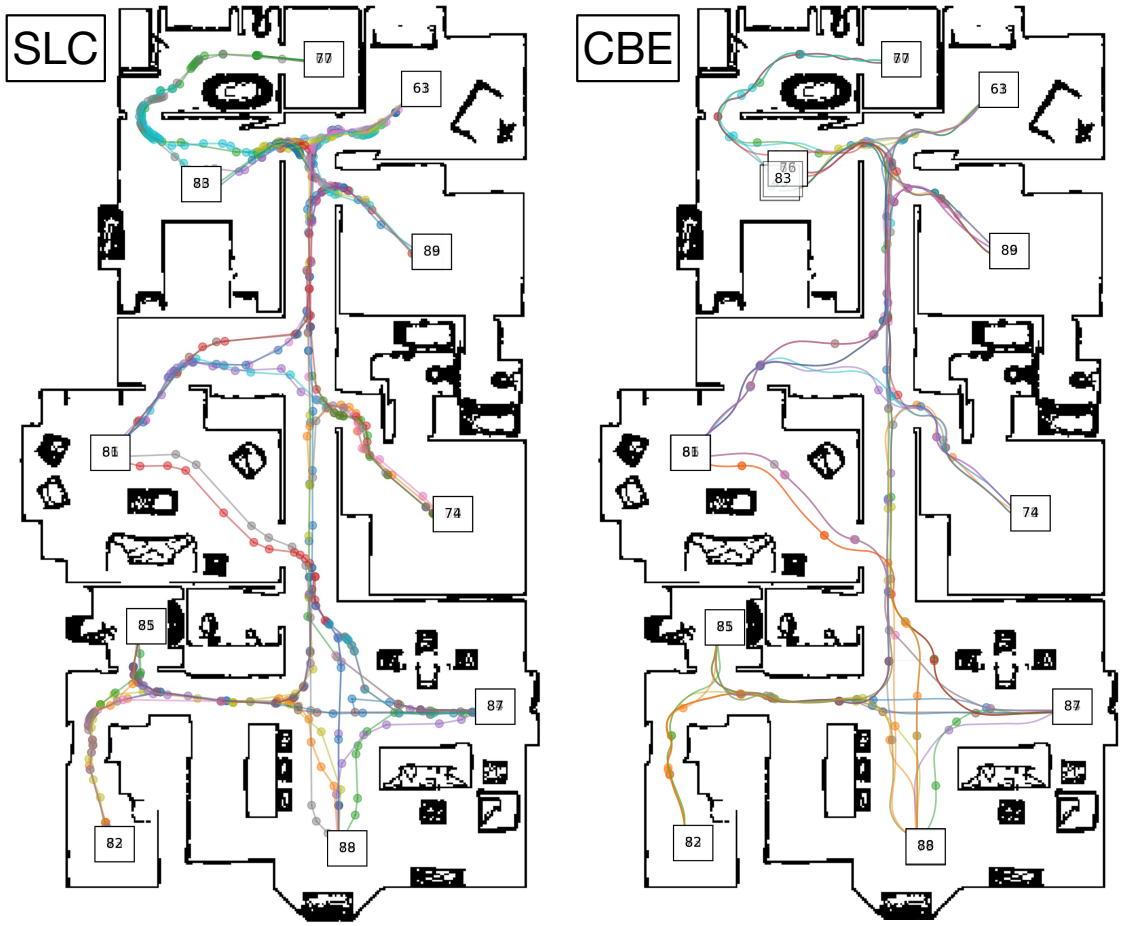


Figure 2.22: Visualization of the topological maps built by SLC [112] (no behavior) and CBE (with behaviors) in one of the test environments (Calavo). Each circle is a vertex. Each demonstration is assigned a different color. See the supplementary video for more examples.

RGB images). The main failure cases of SLC are caused by faulty edges in the map due to visual aliasing. In comparison, CBE maps have much fewer vertices, store only an 800-dim feature per-vertex, and achieve similar or higher planning success rates due to less visual aliasing. Figure 2.22 visualizes the distribution of vertices in the map.

Impact of choice points on generalization. To see the necessity of using choice points for mapping, we evaluate pairwise connectivity between locations when using a fraction of all pairwise demonstrations to build the map. In Figure 2.23, we can see that without choice points there is almost no generalization. This is because fixed-distance segmentation (Sec. 2.2.2) creates attractors that are inconsistently distributed in an environment, making it difficult to link attractors from different demonstrations. Detecting choice points significantly improves connectivity, but there is still a gap compared to a dense map. It can be a future work to improve choice point detection to close this gap.

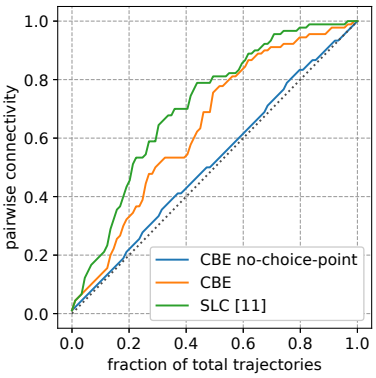


Figure 2.23: Comparing map connectivity when only a fraction of total trajectories are used to build the maps. Diagonal line indicates no generalization.

2.2.5 Discussion

We introduce Composable Behavior Embedding, a robot-agnostic behavior representation for visual navigation. With CBE, robots are able to robustly replicate visual navigation tasks using extremely compact representations; two attractor features and a low-dimensional vector per behavior. We show how CBE can be incorporated into larger scale navigation systems for path following and topological mapping. Here, CBE significantly improves memory-efficiency. Our model operates in continuous state and action spaces, and we conducted experiments in realistic simulation environments. We will test our system on a real robot once the hardware becomes

accessible, but based on other work using these environments we are confident that our results will transfer well to real environments and robots. The continuous trajectory embeddings learned by CBE are well suited to connect to similarly structured language embeddings and using our model to perform language-based visual navigation is an interesting direction for future research.

Chapter 3

Terrain Modeling and Perception for Robust Off-road Autonomy

While there has been great recent interest in the development of autonomous vehicles, the vast majority of the work has focused on *on-road* and *urban* driving. However, a wide range of application areas, including defense, agriculture, conservation, and search and rescue, could benefit from autonomous *off-road* vehicles that can operate in complex, natural terrain. In such environments, understanding the *traversability* of terrain surrounding the vehicle is crucial for successful planning and control. Perceiving whether terrain is actually traversable from on-board sensors in real time can be a challenging problem as off-road terrain is often characterized by rapid changes to the ground plane, heavy vegetation, overhanging branches, and negative obstacles. In other words, a successful off-road robot must reason about both the geometric and semantic content of its surroundings in order to determine what terrain is traversable and what is impassible.

An effective terrain traversability prediction system should efficiently 1) aggregate the observations over time [109, 179] with noisy odometry [27, 117], 2) reason about the partially seen or even yet to be seen parts of the environment [32, 64, 182], and 3) detect overhanging structures in

the environment, such as tree branches, tunnels, and power-lines [94, 130]. While previous work has addressed each of these issues individually, the aforementioned challenges are related, and solving each one should benefit the others.

In this chapter. We develop terrain perception systems that enable robots to navigate on unstructured off-road terrains using LiDARs and stereo cameras. We adopt a robot-centric approach by building robot-aware traversability maps. We detail the model design, the data curation process, and evaluate their performance on datasets and real robots.

3.1 LiDAR-based Semantic Terrain Classification

In this section, we design and implement *Bird's Eye View Network* (BEVNet), a recurrent neural network that directly predicts the traversability of the terrain in the form of a 2D grid around the robot from LiDAR scans. As shown in Figure 3.1, our model has three main parts: 1) a 3D sparse convolution sub-network to process the voxelized point cloud, 2) a Convolutional Gated Recurrent Unit (ConvGRU) which uses convolutional layers in gated recurrent unit [35] to aggregate the 3D information, 3) a 2D convolutional encoder-decoder with efficient backbone [25] that simultaneously inpaints the empty spaces and projects the 3D data into the 2D Bird's Eye View (BEV) map. To train the model, we use both past and future labeled LiDAR scans to build a complete 3D semantic point cloud and build the ground-truth 2D traversability map.

We make several contributions and empirical observations. We proposed a novel framework to build the BEV map by simultaneously 1) aggregating observations over time, 2) predicting the unseen areas of the map, and 3) filtering out irrelevant obstacles like overhanging tree branches that do not affect traversability. Experimental results on SemanticKITTI [15] and RELLIS-3D [77] demonstrate the novel framework works well in both on-road and off-road settings, and outperforms the strong baselines in all the aforementioned tasks.

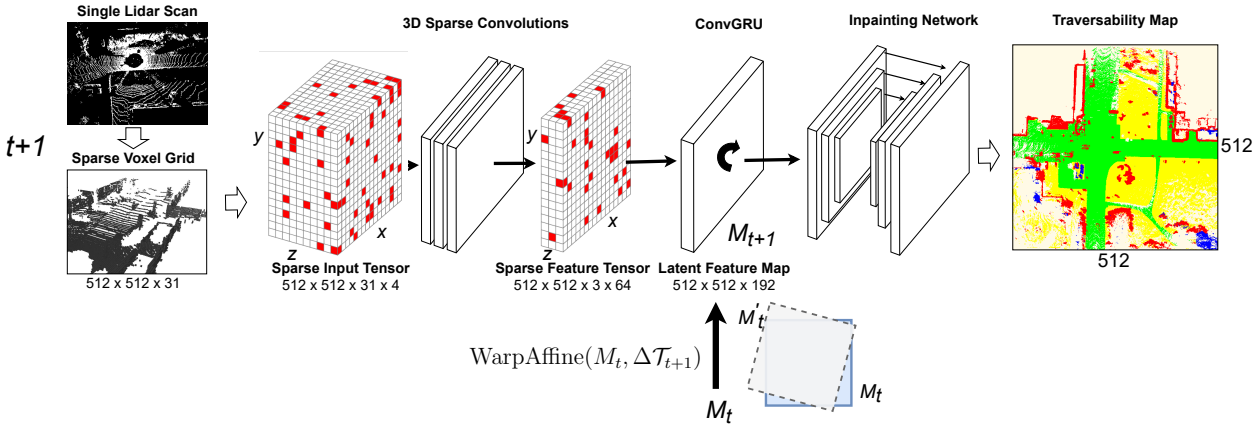


Figure 3.1: The network architecture of BEVNet. The incoming LiDAR scan is first discretized into a sparse voxel grid, which is then fed into a sequence of sparse convolution layers to compress the z dimension. The compressed sparse feature tensor is aggregated over time via the ConvGRU unit. We use differentiable affine warping to align the latent feature map with the current odometry frame. Finally, the inpainting network "inpaints" the latent map to output a dense traversability map.

3.1.1 Related Work

While there exists a vast literature on perception for autonomous driving [12, 18, 23, 72, 123, 136, 175, 185], most prior work focuses on *urban* environments and leverage large datasets and structure inherent in cities and road networks. Since there are not as many recent papers focused on off-road driving and rural environments, we compare our system to prior works that share the mutual components to ours, namely traversability analysis [14, 92, 153, 156], online semantic mapping [23, 109, 136, 175, 185], recurrency handling [23, 109, 136], bird's eye view semantic segmentation [40, 109, 123, 188], and semantic scene completion [32, 64, 182].

Traversability Analysis Traversability Analysis is less common in the task of urban autonomous driving but crucial for a successful off-road autonomy [92, 93, 131]. Traversability of surrounding terrain may be analyzed based on various criteria, including surface roughness [153], negative obstacles [92], and terrain classification [14, 156]. Our system performs end-to-end

semantic mapping by classifying each surrounding location based on traversabilty.

Online Semantic Mapping Semantic mapping provides structured information for an autonomous driving system, with the earliest works in this area dating back to well before deep-learning driven methods gained traction [164]. A core paradigm is the bird's eye view representation, which stores local information in a two-dimensional grid surrounding the vehicle [45]. While prior works such as [12, 136, 185] utilize a high definition map as additional prior knowledge, such a map may not be available for off-road terrain. Our system therefore does not assume a map *a priori*, and instead constructs one online. Similar to our system, [23, 175] learns to produce semantic maps for urban autonomous driving. In comparison, our system focuses on semantic mapping with a broader categorization of semantics based on terrain traversability, which is useful for off-road driving.

Recurrent Representations Temporally consistent accumulation of semantic information is crucial for mapping the environment and providing the information necessary for safe, efficient motion planning. Recent work [23, 136] directly concatenates the past 10 voxelized LiDAR scans as its input representation for memory efficiency. In comparison to our system, the author's recurrent architecture is specifically designed for semantic occupancy forecasting, whereas our recurrent architecture accumulates sensor data to better estimate the traversability of the current surrounding terrain. Maturana et al. [109] utilize a more conventional approach by accumulating information via Bayes filtering, which in our system is replaced with a recurrent neural network. There are various approaches to handling recurrency in neural networks including [22, 36, 48, 71, 151, 170, 179]. Our system utilizes ConvGRU to accumulate 2D representations for BEV mapping.

BEV Semantic Segmentation [123] learns to produce a BEV map from RGB camera for on-road driving, and [109] produces a BEV map for off-road driving by segmenting an image and projecting

it using depth information. In comparison, our system learns to directly project a LiDAR scan to produce a BEV map. We also compare our approach to LiDAR segmentation where the segmented point cloud from networks such as [40, 188] can be projected onto a BEV map. We evaluate this comparison in detail in Sec. 3.1.4.

Semantic Scene Completion (SSC) The goal of SSC is to generate a complete 3D scene given a single LiDAR scan as input. Existing works such as [32, 64, 182] utilize information from semantic segmentation to complete the scene, whereas our system learns to directly predict the completed scene and therefore does not require segmentation prediction from a secondary network or ground truth labels. In addition, our system performs point cloud projection and scene completion in 2D simultaneously, as the main task in our work is to produce a 2D BEV map. We evaluate our system's ability to complete scenes by comparing our model against [182], the details can be found in Sec. 3.1.4.

3.1.2 Method

Overview

We consider a mobile robot with a 360° LiDAR mounted at its top. In order for the robot to navigate efficiently and safely in a new environment (either on-road or off-road), the robot builds an *online traversability map* around itself. The traversability map resembles a conventional occupancy map as well as the semantic map from [109], where each cell stores a probabilitity distribution of traversability labels. In this work, we use four levels of traversability: *free*, *low-cost*, *medium-cost* and *lethal*. The number of traversability levels can be trivially extended if so desired. The traversability map is inside the robot's odometry frame, so that the robot is always at the center, with its heading pointing to the east. The traversability map is converted to a costmap by mapping

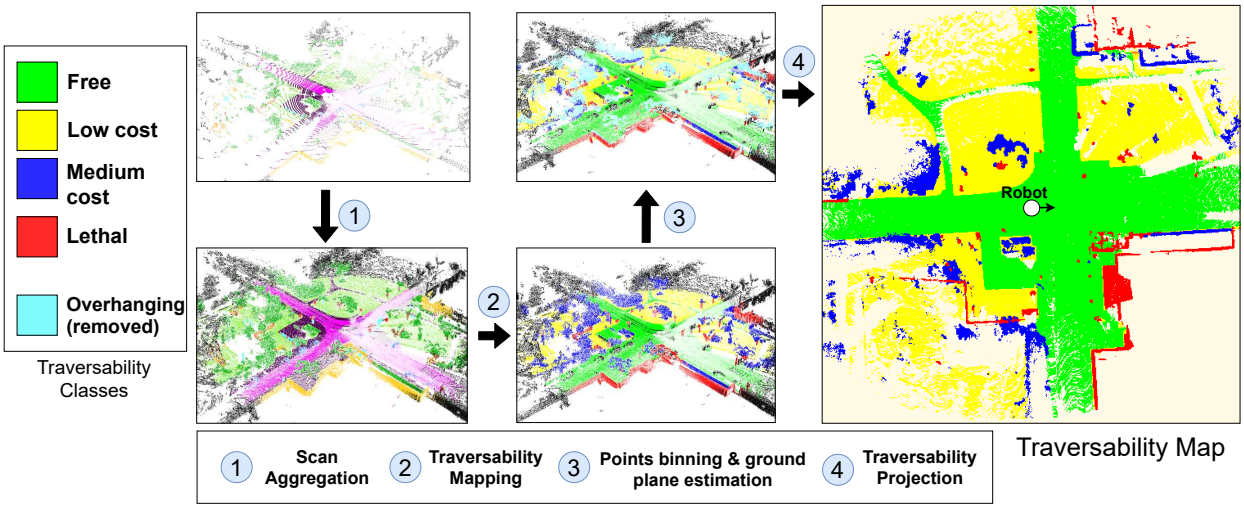


Figure 3.2: Process for generating the traversability dataset on SemanticKITTI. Single scan labels are aggregated to form complete scenes (scan aggregation), and their semantics are remapped to our 4 class ontology (traversability mapping), legend defined in figure. The remapped scans are then filtered with ground estimation to remove overhanging points (points binning and ground plane estimation). Finally, the filtered points are projected to a traversability map (traversability projection).

each traversability level to the corresponding cost value via a lookup table. The converted costmap can be easily interfaced with a local planner [174] or a global planner (e.g., A*) for finding the least-cost path to a goal.

We adopt a supervised-learning approach to predict this traversability map. We start by building a traversability dataset from LiDAR segmentation datasets [15, 77] via a traversability-aware projection procedure. Then, we introduce BEVNet, a recurrent neural network that takes the current LiDAR scan and utilizes its history to build a dense traversability map. In the following sections, we will describe each component in detail.

Building a Traversability Dataset

Recent work [109, 136] focuses on on-road driving where reasoning about a large number of fine-grained semantic classes is necessary. Here we consider a more general driving paradigm

where we simply care about the *traversability* of the surrounding terrain. This makes our model applicable to both on-road and off-road driving. Given a dataset with semantically labeled LiDAR scans, we convert it to a traversability dataset via the following procedure (illustrated in Figure 3.2).

Scan Aggregation. For each scan, we aggregate it with the past t and the future t scans with stride s to construct a larger point set. We set t to a large enough number (e.g., 71) to obtain dense traversability information for a large area around the robot. These parameters may be tuned depending on the vehicle speed and density of the LiDAR points.

Traversability Mapping. We map the semantic classes into our 4-level ontology. The general principle is to map semantic classes with similar costs to the same traversability label. For example, *car* and *building* are mapped to *lethal*, whereas *mud* and *grass* are mapped to *low-cost*. Detailed mapping can be found in Appendix A.

Points Binning and Ground Height Estimation. For each point in the aggregated scan, we do a down projection to find its location x, y on the traversability map. Hence each x, y location of the map contains a pillar of points. We estimate the ground height map by running a mean filter kernel over the lowest z coordinates of the points labeled as *free* and *low-cost* at each x, y location in the map. This height map is used as a reference for final traversability projection.

Traversability Projection. For each pillar of points, we filter out overhanging obstacles by removing points that are above the local ground level by a certain threshold because they will not collide with the robot. For the remaining points, we take the point with the lowest traversability at each x, y location as the final traversability label.

Feature Extraction via Sparse Convolution with Z Compression

The architecture of BEVNet is shown in Figure 3.1. An input LiDAR scan is first discretized into a $512 \times 512 \times 31$ grid with a resolution of $0.2m$. We perform sparse discretization so that only occupied voxels are preserved. Each voxel contains a 4-dimensional feature $f = \frac{1}{n}\sum_{i=1}^{n}[x_i, y_i, z_i, r_i]$, which is the average of the coordinates and remission values of the points inside the voxel. This sparse voxel grid is fed into a sequence of sparse convolution layers, which compress the z dimension via strided convolutions. We keep x and y dimensions unchanged. The output of the sparse convolution layers is a sparse feature tensor S of size $512 \times 512 \times C$, where C is the feature dimension.

Temporal Aggregation of Sparse Feature Maps

A single LiDAR scan becomes increasingly sparse as the distance increases, making it difficult to estimate traversability for areas far away from the robot. Contrary to classical SLAM that aggregates LiDAR measurements over time via a hand-engineered Bayesian update rule [165], we let the network learn to aggregate the sparse feature maps from past LiDAR scans via a Convolutional Gated Recurrent Unit (ConvGRU). The ConvGRU maintains a 2D latent feature map M that shares the same coordinate system and dimensions as the final traversability map. The latent feature map M is updated as

$$M_{t+1} = \text{ConvGRU}(\text{WarpAffine}(M_t, \Delta\mathcal{T}_{t+1}), S_{t+1}),$$

where $\Delta\mathcal{T}_{t+1}$ is the relative transform of the robot's odometry frame from t to $t+1$. The WarpAffine operation transforms the latent feature map M_t from the previous odometry frame to the current odometry frame so that the features from M_t and S_{t+1} are spatially aligned. Note that the WarpAffine operation is differentiable to allow the gradients to backpropagate through time.

Traversability Inpainting

Since the ConvGRU only aggregates sparse feature tensors, M contains little information for areas where there is no LiDAR point. Instead of treating no-hit area as unknown, we let the network fill in the empty space by leveraging the local and global contextual cues via the *Inpainting Network*. The inpainting network is a fully convolutional network inspired by FCHardNet [25] which is originally designed for fast image segmentation. It consists of a sequence of downsampling and upsampling layers with skip connections, making it effective for capturing local and global contextual information for predicting what is missing.

3.1.3 Implementation Details

We build the traversability datasets from SemanticKITTI [15] and RELLIS-3D [77] to evaluate BEVNet in both on-road and off-road scenarios. For SemanticKITTI we aggregate 71 frames with stride 2 to generate a single traversability map. For RELLIS-3D we aggregate 141 frames with stride 5. Both datasets provide per-frame odometry, which we use for the differential warping layer in the ConvGRU. The traversability maps have a size of $102.4m \times 102.4m$ with a resolution of $0.2m$. Note that the traversability maps contain an additional "unknown" class marking regions that have never been observed.

Network training. We train our network using the Adam optimizer [81] with an initial learning rate of $3e-4$ and a decay of 0.7 per epoch. We use a weighted Cross-Entropy loss. We start with training a single-frame model without ConvGRU until the model converges. Then we freeze the sparse convolution layers and insert the ConvGRU layer, and then train the ConvGRU and the inpainting network together. While technically we can train the whole network end-to-end , this two-stage training procedure is faster and is more memory-efficient. When training the ConvGRU,

we use a sequence length of 5 with a frame stride randomly chosen from [1, 10, 20]. Training takes about 12 hours on a single RTX 3090. The inference time of our network is 6 fps on a RTX 3090.

Data augmentation. During training, we randomly rotate every pair of LiDAR scans and the ground truth traversability map in $\mathcal{U}[-45°, 45°]$, and randomly drop 20% of the points. Furthermore, we perturb the groundtruth odometry with rotation drawn from $\mathcal{N}(0, 0.01^2)$ and translation drawn from $\mathcal{N}(0, 0.1^2)$. Note that the error in odometry will accumulate over time. We evaluate the effect of noisy odometry in Sec. 3.1.4.

3.1.4 Experiments

We conduct both quantitative and qualitative study on SemanticKITTI (on-road) and RELLIS-3D (off-road) datasets. We trained a separate model for each dataset. We compare with a variety of baselines, ranging from LiDAR segmentation to scene completion on the validation sequences. We also perform an ablation study to better understand the contribution of recurrence, and how our model behaves on the two datasets that have very different characteristics.

Evaluation Metrics

We use the mean Intersection of Union (mIoU) [25], a widely used metric for image segmentation, as the quantitative measure of the prediction accuracy. Note that our model predicts an additional "unknown" class to improve the visual consistency, and we exclude the "unknown" class in the evaluation. To better understand our model's capability of predicting the future, we report mIoUs in three modes: **seen**, **unseen**, and **all**. In the "seen" mode, we do not include ground truth labels obtained from future frames, effectively excluding any future predictions. For the "unseen" model we only include the future predictions. In the "all" scenario, we evaluate on both.

	SemanticKITTI			RELLIS-3D		
	All	Seen	Unseen	All	Seen	Unseen
BEVNet-S	0.416	0.465	0.308	0.559	0.518	0.545
Clean Odometry						
BEVNet-TA	0.468	0.534	0.335	0.615	0.605	0.586
BEVNet-R	**0.480**	0.547	**0.344**	**0.618**	0.577	**0.620**
Cylinder3D-TA	0.465	0.655	N/A	0.411	0.568	N/A
Cylinder3D-TA-3D w/o ray tracing	0.482	**0.660**	N/A	0.408	**0.649**	N/A
Cylinder3D-TA-3D w/ ray tracing	0.471	0.646	N/A	0.384	0.609	N/A
Noisy Odometry						
BEVNet-TA	0.379	0.415	0.310	0.452	0.372	0.560
BEVNet-R	**0.468**	**0.529**	**0.343**	**0.614**	**0.572**	**0.616**
Cylinder3D-TA	0.342	0.455	N/A	0.318	0.435	N/A
Cylinder3D-TA-3D w/o ray tracing	0.373	0.479	N/A	0.347	0.517	N/A
Cylinder3D-TA-3D w/ ray tracing	0.369	0.478	N/A	0.331	0.495	N/A

Table 3.1: Mean IoU of different methods on SemanticKITTI and RELLIS-3D.

Comparison with LiDAR Segmentation with Temporal Aggregation

A strong baseline for building a traversability map is to perform semantic segmentation of the incoming LiDAR scan, project it down to obtain a 2D sparse traversability map, and aggregate the traversability maps over time. To compare with this approach, we choose Cylinder3D [188] (finetuned on our 4-class ontology) as the LiDAR segmentation network for its strong performance, and use the same projection procedure in Sec 3.1.2 on the input LiDAR scan to obtain the single-frame traversability map. We perform the temporal aggregation by tracking the categorical distribution of traversability via a uniform Dirichlet prior. To do so, we keep a counter map M_C of size $H \times W \times 4$ (initialized to zeros). It is of the same size as the traversability map except that the last dimension counts the traversability labels observed so far. We update M_C incrementally. For each incoming single-frame traversability map, we warp M_C to the current odometry frame via bilinear interpolation, and increment the counts by adding the one-hot version of the incoming single-frame map. The actual traversability label can be obtained by taking the *argmax* of the last

dimension of M_C.

Results on SemanticKITTI. In the left half of Table 3.1 we compare the performance of BEVNet-Recurrent (BEVNet-R) with Cylinder3D+Temporal Aggregation (C3D-TA) on the SemanticKITTI validation set. When only considering what has been observed so far ("seen") and clean odometry, C3D-TA is better than BEVNet-R. This shows that LiDAR segmentation with accurate temporal aggregation can work very well in structured environments such as on-road driving. When evaluated on the full groundtruth ("full"), BEVNet-R outperforms C3D-TA because C3D-TA cannot predict the future traversability. When evaluating on noisy odometry, BEVNet-R surpasses C3D-TA for both "seen" and "full" test scenarios. BEVNet-R uses learned recurrency to "fix" the errors in odometry and to adaptively forget history in case the error in odometry is too large. In comparison, C3D-TA solely uses the provided odometry to aggregate information, which may result in large misalignment as errors accumulate over time.

Results on RELLIS-3D. The results on RELLIS-3D (right half of Table 3.1) share a similar trend as those in SemanticKITTI, except that BEVNet-R consistently outperforms C3D-TA with a larger gap. This suggests that off-road environment is more challenging, where accurate LiDAR segmentation is hard to obtain due to the lack of environmental structure. Indeed, Cylinder3D only achieves a 64.1 mIoU on RELLIS-3D for LiDAR segmentation, which is lower than the 87.9 mIoU on SemanticKITTI. Interestingly, noisy odometry has almost no impact on BEVNet-R. We hypothesize that it is because the RELLIS-3D dataset contains less clutter and occlusion so BEVNet-R does not rely heavily on the history for traversability prediction.

Qualitative results In left half of Figure 3.3, we highlight the fact that BEVNet-R can preserve small dynamic objects such as bicyclists better than C3D-TA. Hand-engineered temporal aggregation is prone to treating small dynamics objects as noise and ignoring them. In comparison,

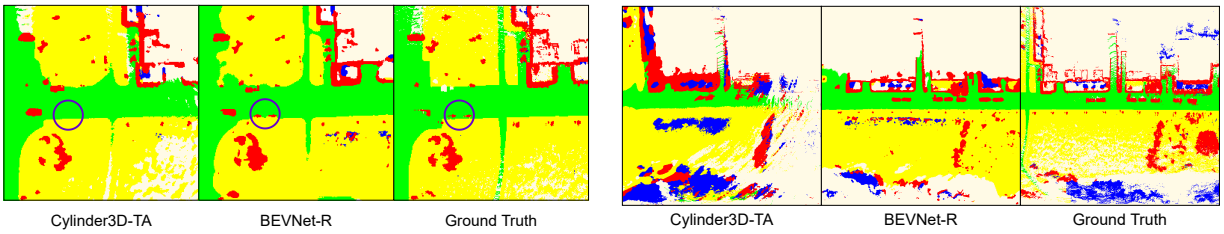
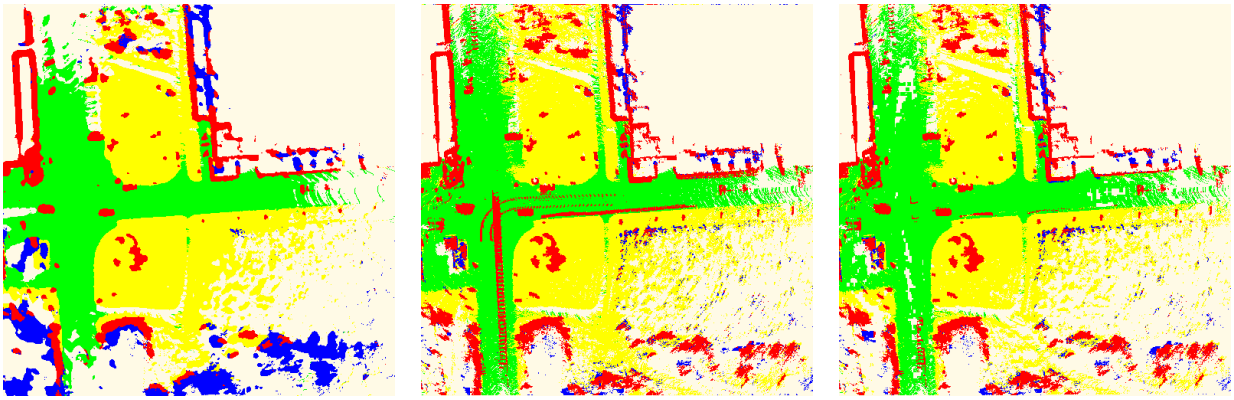


Figure 3.3: Qualitative comparison of our method and baseline. **Left:** BEVNet is better at preserving small fast-moving objects such as bicyclists (highlighted by the blue circles), which the hand-engineered update rule tends to ignore. (Maps are 50% zoomed in). **Right:** When noise is injected into the odometry, the learned recurrent network is able to fix errors in BEV map caused by the noise, while Cylinder3D+TA fails to do this, resulting in a blurry, inaccurate map.

BEVNet can learn to keep small dynamic objects, while preserving smoothness in static regions. The right half shows the impact of noisy odometry. We can see large misalignment and smear artefact for C3D-TA, whereas BEVNet-R produces significantly cleaner output. Finally, in Figure 3.5 we visualize examples on both SemanticKITTI and RELLIS-3D. In general BEVNet-R shows strong performance in predicting future traversability. It learns to predict whole cars, alley entrances, and trail paths with extremely sparse LiDAR points.

Comparison with Semantic Segmentation and 3D Temporal Aggregation.

We additionally provide a baseline that aggregates the points in 3D before projection (Cylinder3D-TA-3D) using Octomap [73]. The results are included in Table 3.1. 3D-TA works better than 2D-TA but runs significantly slower. Moreover, 3D-TA struggles more with handling dynamic obstacles (Figure 3.4). While turning on ray tracing [73] improves the handling of dynamic obstacles significantly, it also erroneously clear some ground points due to shallow LiDAR incident angles. More details can be found in Appendix A.7.



(a) (b) (c)

Figure 3.4: Comparing the seen areas of (a) Cylinder3D-TA, (b) Cylinder3D-TA-3D w/o raytracing and (c) Cylinder3D-TA-3D with raytracing. The seen areas are slightly different depending on how aggregation is performed.

Ablation Study

We conduct our ablation study on three variants of BEVNet: BEVNet-Single (BEVNet-S), BEVNet-Single+Temporal Aggregation (BEVNet-TA), and BEVNet-Recurrent (BEVNet-R). We aim to answer three questions: 1) is learned recurrence better than temporal aggregation? 2) does history help predict the future? and 3) where should information be aggregated in the network? We answer these questions through a set of experiments on both SemanticKITTI and RELLIS-3D datasets.

Is learned recurrence better than temporal aggregation? When evaluated on the full ground truth, we observe that BEVNet-R consistently outperforms BEVNet and BEVNet-TA on both on-road and off-road scenarios (Table 3.1). Notably, BEVNet-TA also outperforms BEVNet, which shows that any form of recurrence is beneficial. In particular, we observe that the learned recurrence makes best use of the temporal information in comparison to the hand-engineered TA. When noisy odometry is introduced we observe the same trend as discussed in Sec. 3.1.4, where BEVNet-R shows robustness to noise and outperforms BEVNet-TA.

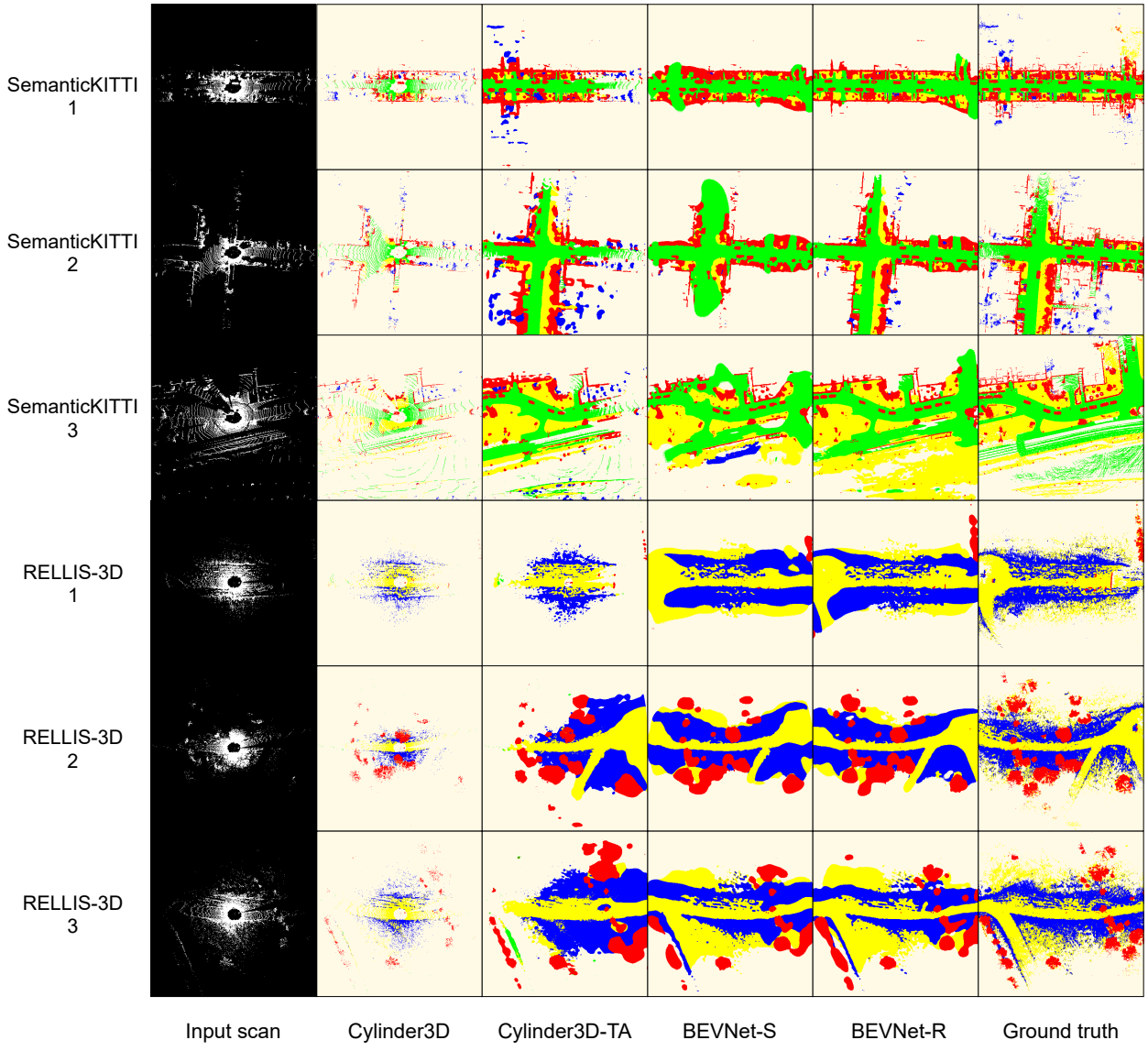


Figure 3.5: Qualitative comparison of our method on SemanticKITTI [15] and RELLIS-3D [77]. Learned recurrency in our end-to-end network can preserve previously observed information while predicting future observations for the occluded areas.

Does history help predict the future? In Table 3.1, we can see that any form of recurrence that accumulates history helps with predicting the unseen area. When evaluated on unseen ground truth, BEVNet-TA and BEVNet-R both consistently outperform BEVNet-S, even when noisy odometry is introduced. This yields an interesting conclusion that any form of memory acquired through recurrence provides more information for forecasting the future. We emphasize that learned recurrence especially proves strong improvement in this aspect, as it consistently outperforms all other approaches on the unseen ground truth.

Where to put ConvGRU? Recurrence may be applied right after the sparse convolutions (*early aggregation*) or may be applied after the 2D inpainting network (*late aggregation*). We compare the two approaches on the SemanticKITTI dataset with clean odometry and including the unseen area for evaluation. Note that here our model is trained with clean odometry. As we emphasize with Table 3.2, our experiments show that early aggregation yields better results than late aggregation. This is because when early aggregation is applied the inpainting network has access to temporally fused information, and therefore is given more information to complete the scene and maintain temporal consistency across scans. Furthermore, we may infer that if late aggregation is applied, it is more difficult for the recurrent network to learn to correct the odometry as it is given completed scenes with potentially noisy information instead of the sparse feature maps.

	mIoU
Early Aggregation	**0.535**
Late Aggregation	0.479

Table 3.2: Effect of GRU location in the network.

Comparison and Discussion with other Related Work

Several recent papers have focused on semantic understanding of scenes from sparse LiDAR scans, which we discuss below. These papers solve related, but slightly different problems. They produce smaller maps, and do not perform temporal aggregation.

Semantic Scene Completion. The scene completion task aims to predict a dense semantic voxel grid from a single LiDAR scan. To compare to our work, we build a traversability map by converting predicted dense voxel grids to 2D traversability maps using the same method described in Sec. 3.1.2.

We compare with JS3C-Net [182] because it is one of the top performing models on the SemanticKITTI scene completion task and the code is publicly available. We map the 19-class predictions of JS3C-Net to our 4-class ontology, and project the voxels down to generate 256×256 traversability maps. We adapt our approach to produce the same output format as JS3C-Net. JS3C-Net achieves an mIoU of 0.549, whereas BEVNet-S and BEVNet-R achieve an mIoU of 0.592 and 0.608, respectively. This shows that learning to project traversability and inpainting the map simultaneously can work better than first reconstructing the scene, followed by a rule-based projection.

Inpainting Network. [64] recently proposed an approach that uses GANs to inpaint a sparsely segmented BEV image. We trained our model using the same SemanticKITTI dataset with the groundtruth 19-class BEV images as supervision. Note that this task is different from traversability estimation because it does a simple topdown projection. Our single-frame model achieves 0.253 mIoU on this task, which is significantly higher than 0.131 reported in [64]. Note that [64] assumes the LiDAR scan has been already segmented, and it does inpainting in 2D. This makes it unsuitable for our traversability projection due to the lack of 3D reasoning.

Figure 3.6: High-speed driving in complex off-road environments requires joint reasoning of terrain semantics and geometry. Top row: a vehicle can drive at high-speed on a dirt road but has to be more cautious in snow due to wheel slipping. Bottom row: a vehicle needs to estimate terrain slopes and sizes of vegetation for safe planning and control.

3.2 Visual Terrain Modeling for High-speed, Offroad Navigation

There are several limitations in the LiDAR-based terrain classification system described in the previous section. First, it assumes the terrain is flat, but natural terrains have hills and valleys (Figure 3.6). Moreover, the point cloud from LiDAR can be rather sparse, making it tricky to build a *complete* map of the environment when the vehicle travels at high speeds [65]. Another downside of LiDAR is that it emits lasers into the environment: dust and snow can interfere with the measurement, and outside observers can detect the vehicle from the emitted lasers.

Cameras, on the other hand, provide a number of benefits over LiDAR. Cameras provide high-resolution semantic and geometric information, stealth due to their passive sensing nature, are less affected by dust and snow, and are considerably cheaper. Hence, a camera-only off-road

terrain perception system can potentially reduce the hardware cost, improve the system robustness at high speeds, and open up new possibilities for off-road navigation under extreme weather conditions and where stealth is desired.

Perhaps unsurprisingly, similar motivations have spurred recent major efforts of camera-only perception for *on-road* navigation [34, 68, 97, 123, 183]. This task mainly focuses on Bird's Eye View (BEV) semantic segmentation to assess traffic conditions. One notable work is Lift-Splat-Shoot (LSS) [123]. The core of LSS consists of a "lift" operation that predicts a categorical distribution over depth for each pixel and a "splat" operation to fuse the image features and project them to the BEV space. LSS and related work are entirely data-driven, so they can predict complete maps and are more robust to sensor noise and projection errors. But their applicability to off-road perception is challenged by several barriers. First and foremost, they only predict a ground semantic BEV map without any 3D terrain information that is critical for planning and control in off-road environments. Second, they are usually not optimized for real-time operation. For example, LSS predicts depth as a categorical distribution along the camera frustums to enable end-to-end learning, but this comes at a price: the size of the frustum features is large, creating a time and memory bottleneck, especially for field robots with limited hardware capabilities. Finally, to train such models, we need large-scale labeled terrain datasets. But, to the best of our knowledge, there are no such datasets for complex off-road terrains yet.

To this end, we design and implement *TerrainNet*, a real-time, camera-only terrain perception system that enables high-speed driving of a passenger-scale Polaris [2] vehicle on complex off-road terrains. We make several design choices and innovations to make TerrainNet suitable for off-road perception. First, TerrainNet supports multi-view RGB with *optional stereo depth* as inputs. Using stereo depth provides valuable geometric context that greatly improves prediction accuracy. Second, we enhance the predicted depth via an auxiliary loss on the output depth values. This extra supervision teaches the model to *correct* and *complete* the (potentially inaccurate) input

stereo depth. This turns out to be critical for accurately estimating terrain geometry. To create ground-truth depth images, we build a complete map of the environment offline by aggregating LiDAR scans and removing outliers from the entire point cloud. Note that LiDAR is only used to create the training dataset and is not needed in deployment. Third, we make TerrainNet more than 5× faster than LSS by lifting each image feature into a *single* 3D point and use a soft quantization technique in the "splat" step to keep the model end-to-end trainable. Lastly, TerrainNet predicts a *multi-layer* BEV map that captures both ground and overhanging terrain features.

TerrainNet is the first off-road, camera-only perception system for joint BEV semantic and geometric terrain mapping in a unified feed-forward model. To train and evaluate our model, we collect a new challenging large-scale, off-road dataset from different environments consisting of both *on-trail* and *off-trail* driving scenarios with *extreme elevation changes*. We believe our dataset better covers the diversity of the off-road driving challenges compared to RELLIS-3D [78], a publicly available dataset which is captured from a single environment and mainly consists of on-trail driving scenarios with limited elevation changes. We show that TerrainNet outperforms recent baselines in semantic and elevation estimation, while being much faster. Finally, we deploy TerrainNet inside a full navigation stack to have a Polaris vehicle traverse a 1.1 kilometer route over snow-covered hills.

3.2.1 Related Work

On-road BEV perception. LiDAR and cameras are commonly used sensors in on-road autonomous driving perception systems [91, 123, 188], providing crucial information about surrounding objects and their semantics. Convolutional Neural Networks (CNNs) have shown exceptional performance in image [31] and point cloud [188] segmentation, and they have become the core of perception systems in on-road scene understanding. Although many of these systems rely on

LiDAR [91, 137], there has been increasing interest in using cameras due to their lower cost. In camera-based methods, a critical aspect is learning to project pixel-wise features to the BEV space. Lift-Splat-Shoot [123] adopts a *backward* projection scheme that lifts the image features using predicted depth and then splats the features into BEV space. SimpleBEV [66] and BEVFormer [97] perform *forward* projection from a set of grid points in the BEV space to retrieve their corresponding image features. Another line of work [34, 132, 138, 187] learns the projection with an attention mechanism. TerrainNet adopts a backward projection scheme since it makes full use of image pixels and does not assume a flat ground. Moreover, TerrainNet leverages stereo depth completion and soft-quantization for projection. This improves both the speed and accuracy of TerrainNet.

Off-road terrain modeling. Off-road terrains often exhibit large variations in ground elevations which can significantly affect terrain traversability. Hence, there has been a plethora of work on geometric terrain mapping. A frequently used representation is a 2.5D elevation map by aggregating point measurements from LiDARs or stereo cameras [50, 51, 52, 113, 154, 167]. In more complex environments where overhanging objects need to be considered, a voxel-based representation [11] or a multi-level surface map [167] are more effective in capturing detailed geometric information. In practice, obstacles or terrain discontinuities leave areas with missing values in the elevation map, leading to suboptimal motion planning. Inspired by data-driven image in-painting methods, recent works [139, 154] propose self-supervised learning to reconstruct the occluded area from an incomplete elevation map.

Besides geometric terrain modeling, semantics also play a critical role in traversability assessment [10, 79, 108, 110]. For instance, tall grass appears as obstacles yet is traversable, whereas large puddles are perceived as a flat surface but can be dangerous. Semantic segmentation is typically done in the image space [10, 142, 157] or the BEV space [110, 144]. Since planning is more convenient in the BEV space, it is a common practice to project the pixel-wise segmentation

into a BEV cost map using LiDAR or stereo cameras [10, 110]. More recently, [144] present an end-to-end trainable, recurrent CNN that builds a temporally consistent BEV semantic map from LiDAR. For a comprehensive literature review in traversability estimation for mobile robots, we refer readers to surveys [19, 143].

Existing off-road terrain perception approaches are usually designed for low-speed operations. Thus, they are able to gather dense sensor measurements to filter out the sensor noise to estimate a good terrain model. Some systems support high-speed operations but they are limited to on-road, flat terrains. In contrast, TerrainNet is designed for *high-speed* operations on *any terrain*. It achieves low latency by using a compact, multi-layer terrain representation [167], while at the same time maintaining accuracy at high speed with end-to-end BEV perception [123, 144]. Unlike existing systems that require complex algorithms and careful tuning to maintain real-time operations and consistent mapping, TerrainNet is a simple feed-forward neural network. It is also fast and can be improved as more training data is available.

Planning. To navigate a vehicle in off-road environments, a motion planner can leverage the perceived terrain features to plan safe and efficient trajectories. The terrain features are usually converted into costs for a planner to rank and assess the risk of trajectories [20, 21, 42, 47]. In our experiments, we illustrate how to use the MPPI planner [174] for motion planning with terrain features and robot capability considered.

3.2.2 Off-Road Terrain Modeling

To enable a robot to drive safely and efficiently on off-road terrains, it is crucial to understand the traversability of its surroundings. Terrain traversability is the amount of cost or effort to traverse over a specific landscape. While many factors affect terrain traversability, we consider three primary factors: *semantics*, *geometry*, and *robot capability*.

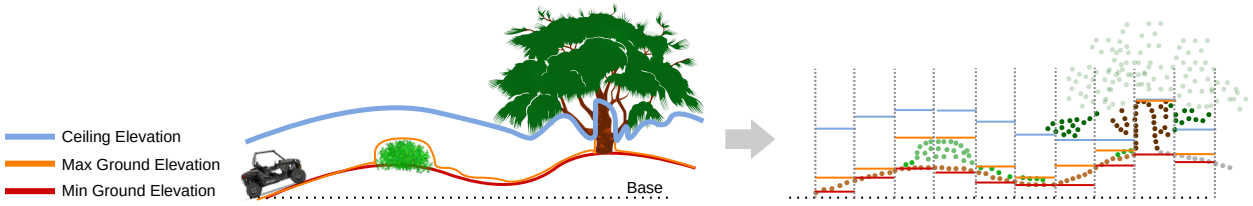


Figure 3.7: **Left** Multi-layer terrain representation. The ground layer consists of the minimum and maximum ground elevations to capture the sizes of porous objects like bushes and also preserve sharp elevation changes after discretization. The ceiling layer represents the geometry of the overhanging objects (e.g., canopy) that might potentially block the path. **Right** In our practical implementation, we obtain a dense point cloud of the environment, discretize the x-y plane into 2D grid cells (shown in 1D for simpler visualization), and compute the elevation values by analyzing the point distribution in each cell. We ignore anything higher than the desired vertical clearance. For each layer and each cell, we count the number of semantic points for each class and compute a normalized histogram as the semantics of that cell.

Semantics. The semantics of terrain refers to the classes of objects (e.g., bush, rock, tree) or materials (e.g., dirt, sand, snow) occupying the terrain. Different semantic classes typically have different physical properties, such as friction and hardness, which can affect the capabilities of the vehicle. For example, since dirt can supply more friction than snow, a vehicle can drive faster on a dirt road than on snowy ground. Moreover, off-road vehicles have higher chassis and better suspension, so they can traverse over bushes and small rocks, albeit at lower speeds due to the increased resistance and bumpiness. Hence, the semantics of terrain encodes a rich spectrum of traversability.

Geometry. Off-road terrains are typically non-flat. A vehicle may not have enough power to climb a steep slope, and driving along a slide slope at high speed poses a significant risk of rolling over. Additionally, the geometry of objects also affects terrain traversability. For instance, a large bush is harder to traverse than a small bush. Hence, understanding the geometry of terrain is another important aspect of traversability assessment.

Robot capability. A vehicle's physical and mechanical properties play another important role

in terrain traversability. A bigger and more powerful vehicle can traverse over larger bushes or rocks than a smaller vehicle with less power. Since robot capability is an intrinsic property of the robot and is independent of terrain properties, we consider robot capability when designing the cost function later in Section 3.2.5. To some extent, robot capability is also considered when generating the training dataset, as described in Section 3.2.4.

Multi-layer Terrain Representation

Since a ground vehicle traverses a 2D surface, it is convenient to use a gravity-aligned, 2D top-down grid map [46] to represent the terrain. Here we consider *local navigation*, where the map provides the vehicle with instantaneous information about its surroundings. Hence, we fix the size of the map and let the map "move" with the vehicle so that the vehicle stays at the center. This is commonly referred to as the *local map*. We do not consider building a global map in this work, though if required, we can leverage existing global SLAM algorithms to stitch the local maps together to obtain a global map.

The key question is what kind of terrain features to store in the top-down map. Previous work usually stores semantic classes [110, 144] or elevations [50, 154] for each grid cell. These representations have two key drawbacks. First, they do not model the hardness or porousness of the terrain, and hence they cannot capture the difference in traversability between a small and large bush. Second, they either do not consider overhanging objects or merge the semantic information of overhanging objects with ground objects. This would result in an inaccurate terrain model because the effect on traversability from overhanging objects is different from ground objects due to the geometry and the lack of gravity-induced force.

To address these issues, we extend the idea of MLS map [167] and propose a multi-layer terrain representation illustrated in Figure 3.7. It consists of two layers, a **ground** layer that captures terrain properties on the ground and a **ceiling** layer that models overhanging objects. For each

layer, we model their semantic and geometric properties separately as follows:

Ground layer. For each map cell on the ground, we store the semantic probability distribution C_{ground} and elevation statistics $\mathcal{H}_{\text{ground}}$ of the terrain. The categorical distribution $C_{\text{ground}} \in \mathbb{R}^K$ stores the relative proportions of the K semantic classes occupying each cell. We use the full distribution of semantic classes instead of a single class label to reduce the information loss caused by discretization (e.g., a map cell may contain both "dirt" and "bush" if it is at the boundary between dirt and a bush). The elevation statistics $\mathcal{H}_{\text{ground}}$ contains the minimum and maximum elevation values h_{min} and h_{max} on the ground. This allows the height of porous objects (such as grass and bushes) to be captured separately and resolves sharp elevation changes due to objects such as rocks and trees.

Ceiling layer. The ceiling layer models *overhanging* objects, such as canopies and tree branches. The semantic information C_{ceiling} is similar to C_{ground} but stores the semantic distribution of overhanging objects. The elevation information $\mathcal{H}_{\text{ceiling}}$ stores the height of the lowest overhanging point h_{ceiling}. If no overhanging points are present (e.g., on open terrains), we set $h_{\text{ceiling}} = h_{\text{min}} + h_{\text{clearance}}$, where $h_{\text{clearance}}$ is a predefined constant of the desired vertical clearance.

This two-layer terrain representation captures a number of properties that are crucial for off-road navigation: the separate modeling of ground and overhanging semantics allows a robot to reason about semantic traversability more accurately, and the ground elevation statistics captures the hardness and sizes of ground objects (refer to Figure 3.7). These features can then be input to a cost function to build an accurate traversability cost map for planning (see Sec. 3.2.5).

Computing the Terrain Representation

This section illustrates a practical approach that leverages LiDAR sensors to build the proposed multi-layer terrain representation. This process is summarized in Figure 3.7. While our goal is real-time visual terrain modeling, we can benefit from accurate geometric sensors and extensive offline

processing to produce high-quality ground-truth data for training that is otherwise challenging to obtain in real time.

Point cloud aggregation. Given recorded LiDAR scans, we use offline SLAM tools [69] to generate gravity-aligned poses for each scan and then aggregate them to obtain a globally aligned dense point cloud. We clean up the point cloud by removing points that are potentially outliers (see the Appendix B.1 for more details). Since this is done offline, we can thoroughly analyze the point cloud without worrying about the computational cost. Note that we could also use the stereo cameras for creating the dense point cloud, but we have found that stereo cameras have a shorter range and produce spurious depths on porous objects and the ground.

Divide and sort. We divide the x-y plane into 2D grid cells with a desired resolution and sort points that fall into a cell $C(i, j)$ by their z-coordinates in ascending order, giving us a sorted point set $\{(x_k, y_k, z_k) \in C(i, j)\}$ for each cell.

Compute elevations. We iterate over the points in each cell, starting from the lowest elevation. The minimum ground elevation of a cell is computed as

$$h_{\text{min}} = \frac{1}{m} \sum_{k=1}^{m} z_k, \tag{3.1}$$

where m is a tuning parameter to smooth out the sensor noise. Cells without enough points are marked as unlabeled. Then, we compute the z-gaps of consecutive points as $\Delta z_i = z_{i+1} - z_i$. Letting z_{gap} denote a predefined constant corresponding to the minimum gap between the ground and the ceiling, if $\Delta z_i > z_{\text{gap}}$, then z_{i+1} is considered the lowest overhanging point, and we set:

$$h_{\text{max}} = z_i \tag{3.2}$$

$$h_{\text{ceiling}} = z_{i+1}. \tag{3.3}$$

If no gap is found, then h_{max} is set to the highest point in the cell, and $h_{\text{ceiling}} = h_{\text{min}} + h_{\text{clearance}}$ where $h_{\text{clearance}}$ is the vehicle's vertical clearance.

Compute semantics. The ground and ceiling layers have separate semantic maps. Given a labeled point cloud (Section 3.2.4), we assign each point to either the ground layer or the ceiling layer. Specifically, for a point (x, y, z),

$$(x, y, z) \in \begin{cases} \text{ground}, & \text{if } z \leq h_{\text{max}}. \\ \text{ceiling}, & \text{if } h_{\text{max}} < z < h_{\text{ceiling}}. \\ \text{ignored}, & \text{otherwise.} \end{cases} \tag{3.4}$$

After the partitioning, we compute a normalized histogram for each cell

$$[p_1, p_2, \ldots, p_K] = \frac{1}{\sum_{k=1}^{K} n_k} [n_1, n_2, \ldots, n_K], \tag{3.5}$$

where n_k is the number of points of class k in that cell.

3.2.3 TerrainNet

In the previous section, we introduced our terrain representation. When a vehicle is deployed in a new off-road environment, it has no knowledge of the terrain and thus must build the terrain representation from its onboard sensors in real time. In this section, we introduce our terrain inference engine *TerrainNet*, which predicts the terrain representation from its onboard RGB or RGB-D cameras.

Overview

TerrainNet is a single neural network that takes RGB or RGB-D images and jointly predicts terrain semantics and geometry. Figure 3.8 illustrates the overall pipeline. First, multiple RGB or RGB-D sensors are passed into the RGB-D backbone. The backbone produces two outputs for each camera: a dense and corrected depth image, and a 2D feature map. Each valid (i.e., neither self-hit nor sky) pixel in the depth image is back-projected to a 3D point using the corresponding camera intrinsics and extrinsics. We compute the terrain embedding for each 3D point that encodes semantic and geometric features. The per-point terrain embeddings are then down-projected and aggregated in the gravity-aligned BEV space with a fast, differentiable splat operation. The resulting BEV feature map is decoded into multiple semantic and elevation maps via an inpainting network. These maps are converted into a costmap by mapping the semantic categories and geometric features via a cost function that accounts for robot capability. The costmap can then be used by a local planner to find the best trajectory to reach a waypoint. In the following sections, we describe each component in detail.

Depth Completion and Correction

To build an accurate terrain representation, the most crucial aspect is to have a good spatial understanding of the environment. Modern stereo cameras can provide dense and accurate depth at close range. However, the error in depth estimation increases quadratically over distance, and stereo matching works poorly for porous structures such as vegetation. Moreover, high-end stereo cameras [1] are equipped with high-sensitivity, low-distortion monochrome sensors for stereo matching, but they are not engineered for wide field-of-view semantic perception like RGB cameras. In Figure 3.8, we show the stereo depth output with the wide-angle RGB image on our hardware platform. The stereo depth only occupies a central area of the image, and there are

many missing depth pixels due to failures in stereo matching. On the other hand, while there have been substantial advances in monocular depth estimation, they have only been shown to work well in structured environments where prior knowledge of object sizes and depth can be learned. Off-road terrains are often much less structured, and we find that solely using RGB information is inferior compared to a stereo system. To combine the rich semantic information of wide-angle RGB cameras and the incomplete stereo depth, we perform *Depth Completion*, which fills in missing depth pixels and corrects the errors in stereo depth in a data-driven fashion.

Given an RGB image and a stereo depth image (note that these may come from two different cameras), we first transform the stereo depth points into the RGB image to obtain an aligned RGB-D image I. Then, we pass the RGB-D image into a U-Net similar to [76] to get a dense depth map. The depth map does not have to be full-resolution, and we find 8× downsampling provides a good trade-off between speed and accuracy. We adopt a classification approach for depth completion because it captures depth discontinuity better than regression [129].

Given the completed depth image and the feature map, we use the camera intrinsics $\mathbf{K}$ and extrinsics $[\mathbf{R}, \mathbf{t}]$ to compute the 3D location of each pixel in the gravity-aligned BEV space. Specifically, given a 2D pixel (u, v) with depth d, we transform it to the vehicle-centered, gravity-aligned BEV frame:

$$[x, y, z]^\top = \mathbf{R}\mathbf{K}^{-1}[u, v, d]^\top + \mathbf{t}. \tag{3.6}$$

Terrain embedding. For each 3D point, we use the corresponding image embedding as the semantic embedding f_{sem}. For elevation, we apply a multi-layer perceptron (MLP) on z to obtain the elevation embedding: $f_{\text{elev}} = \text{MLP}(z)$. We concatenate the semantic and elevation embeddings together and apply another MLP to obtain the per-point terrain embedding: $f = \text{MLP}(\text{concat}(f_{\text{sem}}, f_{\text{elev}}))$.

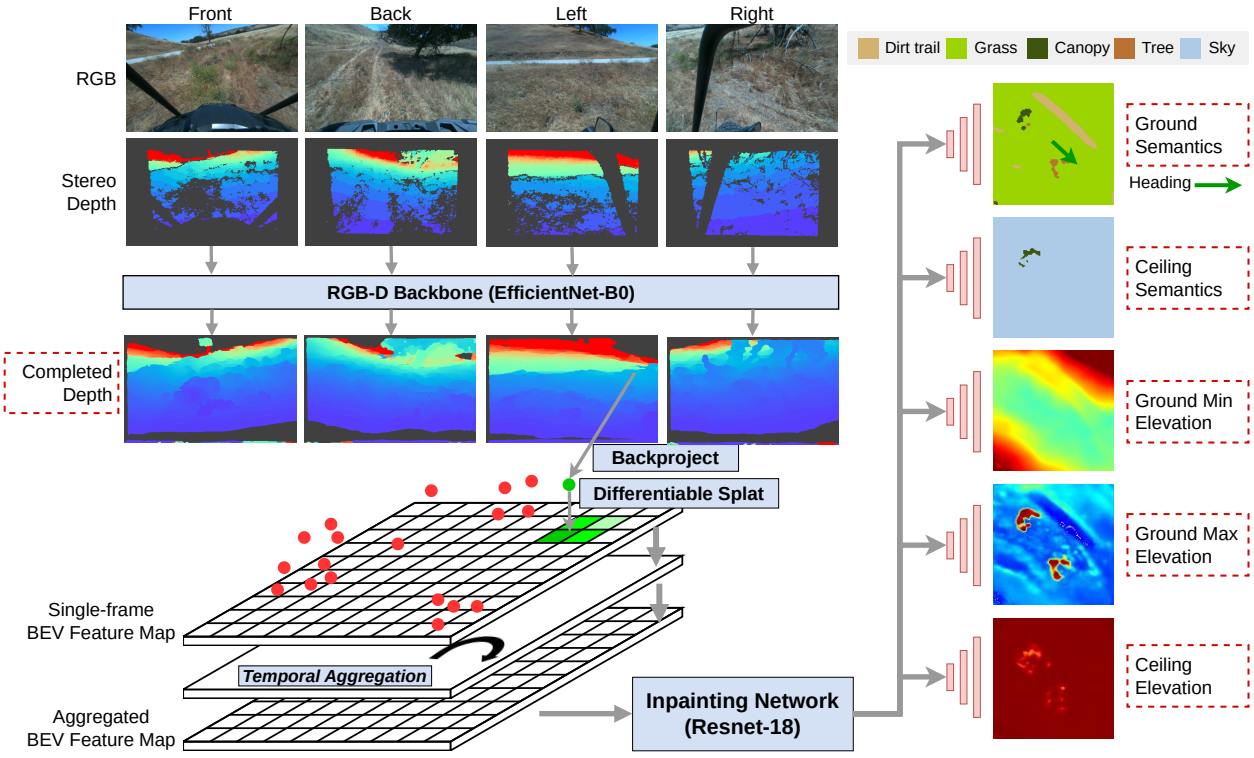


Figure 3.8: **Architecture of TerrainNet.** RGB images with stereo depths are input to the RGB-D backbone to obtain completed depths with semantic features. Note that completed depths exclude pixels (colored in dark gray) that are potentially self-hits or sky. The per-pixel image features are back-projected to 3D to compute their terrain embeddings. The terrain embeddings are then splatted onto the BEV feature map with soft quantization. The feature map is temporally aggregated and finally passed to the multi-head inpainting network to predict a multi-layer terrain map. Red dotted boxes indicate where losses are applied during training.

Fast and Differentiable BEV Projection

Given the 3D point set $\{(x, y, z, f)\}$, the next step is to project the points onto the BEV map by rounding each point into the nearest integer grid coordinates $(\text{round}(x/r), \text{round}(y/r))$, where r is the side length of each map cell. The main disadvantage of hard quantization is that we can no longer correct the grid coordinates via back propagation due to the loss of gradient w.r.t grid coordinates. Inspired by related work in differentiable geometric learning [124], we adopt a **soft quantization** approach, where we "splat" each point feature into the 4 neighboring map cells with weights computed by bilinear interpolation. This allows the depth completion module to learn to adjust its depth prediction by minimizing the loss in the projected map.

Local feature aggregation. Since multiple points may fall into the same map cell, we compute the weighted average of the embeddings in each cell, where the weights are given by the soft quantization to create a grid-based BEV feature map. Let $S(i, j) = \{(f_1, w_1), \ldots, (f_N, w_N)\}$ denote the set of point embeddings, and their corresponding weights within cell (i, j), the feature embedding for the cell is computed as

$$f_{\text{cell}}(i, j) = \frac{1}{W} \sum_{k=1}^{N} w_k f_k, \tag{3.7}$$

where $W = \sum_{k=1}^{N} w_k$.

Temporal feature aggregation. Due to occlusion and errors in estimated depth, the map built from a single frame may not be sufficiently stable and complete. Hence we introduce an optional *Temporal Aggregation* (TA) layer that aggregates BEV feature maps over time. The TA layer is a single ConvGRU similar to that of [144] but with one major difference: we use an orientation-stable odometry frame for aggregation instead of an ego-centric frame to better preserve fine-grained details. This is similar to the classical sliding window approach [110] where we shift the map and integrate the current sensor measurements, but we perform the shifting and aggregation on the feature map with a recurrent network.

Multi-head Terrain Inpainting Module

The terrain inpainting module decodes the BEV feature map into complete semantic and elevation maps. For semantic maps, it predicts a $H \times W \times K$ tensor to represent the probability distribution of the semantic classes, where H, W are the height and width of the map. For elevation maps, it predicts a $H \times W \times 1$ tensor to represent the corresponding elevation values. We adopt a U-Net as the inpainting module with a shared encoder and multiple convolutional decoding heads. Each head predicts a specific type of terrain map. This works better than a single decoding head with multiple channels because the semantic and elevation maps have different output spaces.

3.2.4 Implementation Details

Hardware Platform

We collect our training and test data on a modified Polaris RZR vehicle [2] shown in Figure 3.6, which is capable of driving on off-road terrains at speeds up to 20 m/s (45 mph). The vehicle is equipped with 4 MultiSense stereo cameras [1] and 3 Velodyne 32-beam LiDAR sensors. LiDAR is only used for generating ground-truth with the method discussed in Section 3.2.2 and is not used during testing.

Dataset

We collected our training and validation data on the actual vehicle in several off-road environments with diverse appearances (Figure 3.9). We collected 5 sequences of manual driving data, totaling 20k frames. For each sequence, we ran Google Cartographer [69] to obtain gravity-aligned robot poses for building the ground-truth terrain maps and depth maps.

Create elevation maps. We follow the algorithm in Section 3.2.2 to generate the minimum ground, maximum ground, and ceiling elevation maps. For each time step t, we aggregate 300

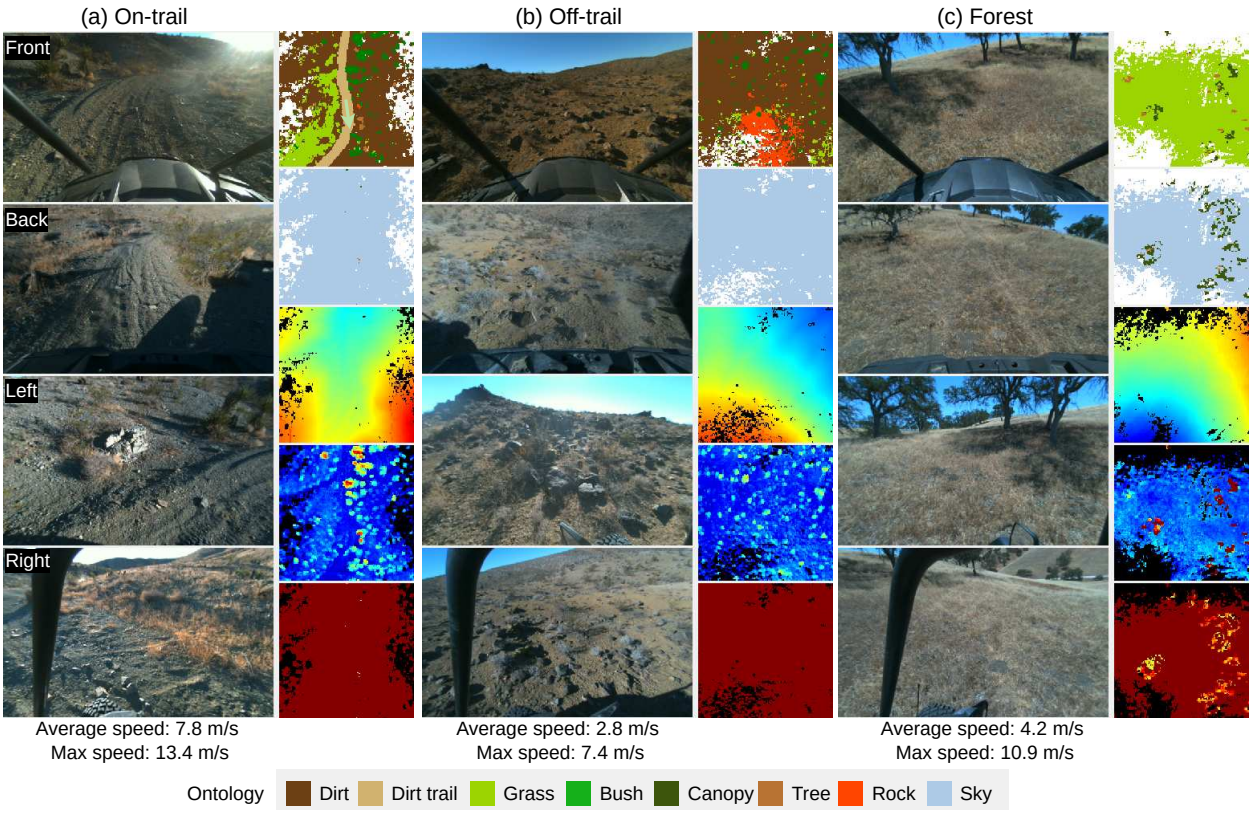


Figure 3.9: Our dataset consists of diverse off-road environments. **a)** vehicle driving at high-speed on a trail with bushes, grass, rocks, and Joshua trees on the two sides; **b)** vehicle driving off-trail on a hilly terrain with scattered grass, bushes, rocks and Joshua trees; **c)** vehicle driving on a hilly terrain with tall trees and overhanging canopy. For each environment, we also show (from top to bottom) the ground semantics, ceiling semantics, min ground elevation, max ground elevation, and ceiling elevation. For the ceiling semantic map, we use the *sky* class to mark areas where no overhanging objects are present. The max ground and ceiling elevation maps are *offsets* to the ground elevation.

LiDAR scans (running at 10 Hz) with a time range of $(t - 150, t + 149)$. Note that the vehicle is always at the origin. We set $m = 3$, $z_{\text{gap}} = 1$ m, and $h_{\text{clearance}} = 3$ m.

Create semantic maps. We ask human labelers to annotate 6k LiDAR scans. These scans are carefully selected such that they contained the relevant semantic objects. These annotated LiDAR scans are aggregated and projected to the BEV ground map or the BEV ceiling map using the algorithm in Section 3.2.2. We identify 7 semantic categories: *dirt*, *dirt-trail*, *grass*, *bush*, *canopy*, *tree*, and *rock*. It is straightforward to extend the ontology and finetune the model if additional data is available, as we show in Section 3.2.5.

Create dense depth maps. We aggregate 50 LiDAR scans for each time step and project the point cloud to each RGB camera. For each pixel, we keep the smallest depth value.

Pseudo Labeling

To leverage unlabeled data, we train a LiDAR segmentation network to predict *pseudo labels* [95] for the unlabeled points in the training set. Specifically, we train Cylinder3D [188] on the labeled LiDAR points and predict the labels for all the unlabeled LiDAR points. We use the pseudo-labeled LiDAR points to build our ground-truth semantic maps for training. We *do not* apply pseudo labeling on the validation set. In Section 3.2.5 we evaluate the effect of pseudo labeling.

Training

We split the dataset into 15k frames for training and 5k frames for validation by cutting each sequence into two non-overlapping train and validation segments. We manually labeled 4500 training frames and 1500 validation frames, and we pseudo-labeled the remaining training frames. All frames have elevation labels. The images are rectified and resized to 512×320. We crop and warp the images such that they all have the same intrinsic parameters. We train TerrainNet on 4 Nvidia A40 GPUs with a batch size of 8 and the Adam optimizer [82]. All models (including

applicable baselines) are trained for 100k iterations. The map size is 51.2 m × 51.2 m with a resolution of 0.4 m. There is no limit on the elevation range.

Loss. We first pre-train the RGB-D backbone on the depth completion task. We discretize the depth into 128 bins with a max range of 25.6 m and apply a cross-entropy loss. Then, we train the whole network end-to-end. For the semantic maps, we use an unweighted cross-entropy loss. For the elevation maps, we use a Smooth-L1 loss [3] with transition parameter $\beta = 0.2$. Instead of predicting h_{max} and h_{ceiling} directly, we predict their *offsets* to h_{min}. The total loss is $L = L_{\text{semantic}} + 0.1L_{\text{elevation}} + 0.1L_{\text{depth}}$.

3.2.5 Experiments

Comparison with Existing BEV Perception Methods

We quantitatively evaluate TerrainNet in terms of its accuracy and speed. We divide the experiments into a few sections, each focusing on a specific aspect.

There is a vast amount of literature on on-road BEV perception. Many of them use heavyweight neural nets, which are unsuitable for running on a compute-limited vehicle. To this end, we focus on three main BEV perception paradigms and pick the most representative baseline for comparison:

Baseline: Segmentation and Projection. We build a classical pipeline [60, 110] where we first perform image segmentation and then project the pixel labels using the stereo depths. Since we only label the LiDAR points, we project the labeled LiDAR points to each image as the ground-truth segmentation mask. We train a state-of-the-art image segmentation network SegFormer-B1 [180] on our dataset. Then, we use the same pipeline for generating the ground-truth maps to build a dense point cloud and project it down to the BEV map. We perform nearest neighbor inpainting to fill in the unknown space.

Baseline: Lift-Splat-Shoot. LSS [123] implicitly learns to predict a per-pixel depth map and splats the features along each depth ray, weighted by the depth distribution. Since vanilla LSS cannot predict elevation, we modified LSS to compute the terrain embedding in the same way as TerrainNet. The main difference between LSS and TerrainNet is that TerrainNet has explicit depth supervision and does a "one-hot" splat of terrain embeddings onto the map.

Baseline: SimpleBEV. SimpleBEV [66] performs forward projection with a uniform splat of image features without considering per-pixel depth. While it has been shown to outperform LSS and many other strong methods for on-road driving, it is not clear if the same holds for *off-road* driving where the terrain is not flat. We use a grid resolution of $128 \times 128 \times 32$ to cover a 3D volume of 51.2 m $\times$ 51.2 m $\times$ 51.2 m. (Using a larger z-resolution would make the memory consumption and training time prohibitive.)

We evaluate two versions of our system: the single-frame TerrainNet and TerrainNet-TA (i.e., with a temporal aggregation layer). For a fair comparison, LSS and SimpleBEV use the same image backbone and inpainting net as TerrainNet with the same hyperparameters for training. We also train a RGB-only version of TerrainNet, LSS, and SimpleBEV by removing the input depth.

Terrain Modeling Accuracy In Table 3.3, we compare TerrainNet with the baselines in terms of their accuracy in modeling the terrain. For semantics, we use the standard IoU metric. For elevation, we use the per-pixel mean absolute error.

TerrainNet surpasses all the baselines for RGB and RGB-D inputs. The fact that TerrainNet outperforms LSS shows that the explicit learning of per-pixel depth is beneficial. While it is possible to project a single depth for LSS during inference, the results are much worse, as shown in the *LSS one-hot depth* baseline. SimpleBEV and Seg&Proj perform worse than TerrainNet and LSS. Figure 3.10 presents a qualitative comparison and Figure 3.11 highlights a particular frame. The

	Method	Dirt	Dirt-trail	Grass	Bush	Canopy	Tree	Rock	mIoU ↑	Elevation (MAE) ↓			Time (ms) ↓
RGB	LSS	0.726	0.621	0.921	0.241	0.042	0.081	0.141	0.396	0.375	0.323	0.015	202
	SimpleBEV	0.727	0.614	0.922	0.245	0.040	0.041	0.106	0.385	0.417	0.322	0.018	110
	TerrainNet	0.730	0.617	0.919	0.257	0.071	0.081	0.177	0.407	0.361	0.315	0.018	25
RGB+Stereo	LSS	0.762	0.656	0.926	0.353	0.096	0.141	0.235	0.453	0.257	0.286	0.016	207
	LSS one-hot depth	0.751	0.642	0.919	0.308	0.094	0.098	0.202	0.431	1.028	0.379	0.019	28*
	SimpleBEV	0.754	0.647	0.923	0.320	0.076	0.070	0.140	0.419	0.286	0.299	0.015	113
	TerrainNet	0.765	0.666	0.926	0.380	0.125	0.145	0.274	0.469	0.244	0.277	0.015	28
	Seg & Proj	0.664	0.671	0.784	0.234	0.109	0.081	0.168	0.387	0.559	0.408	0.034	-
	TerrainNet-TA	0.796	0.679	0.928	0.513	0.161	0.192	0.276	0.506	0.243	0.240	0.024	29

Table 3.3: For semantics, we report the per-class IoU and mean IoU. For elevation, we report the Mean Absolute Error (MAE) for min. ground, max. ground, and ceiling elevation, respectively. We report the inference time on an RTX 3090. The inference time for LSS one-hot depth is an estimation. For Seg&Proj we do not report its inference time since it depends on the implementation of temporal aggregation.

RGB-D models preserve more semantic details, with TerrainNet-TA being closest to the ground truth. The artifacts in the Seg&Proj baseline are due to errors in stereo matching. For more qualitative examples, please refer to Appendix B.5.

In general, large and more frequent classes (*dirt, dirt-trail, grass* and *bush*) are easier to predict than small, less frequent objects (*tree* and *rock*). This is expected since small objects are harder to localize, especially when they are far. *Canopy* is also harder to localize due to its overhanging nature and occlusion. In terms of elevation, TerrainNet-TA does not show notable improvement in the ground elevation than TerrainNet. We hypothesize that it is due to the recurrent layer not being able to track the vertical movement of the vehicle well. We leave it as future work to improve this aspect.

Stereo depth provides significant gains. Models with stereo depths as additional input outperform their RGB-only counterparts by a large margin. For off-road environments, objects are randomly scattered, and they can have similar appearances but different sizes (e.g., small vs. large bushes). Thus, it is hard to estimate the depth from color information. This suggests that for off-road terrain perception, it is preferred to equip the robot with stereo cameras instead of monocular cameras. The performance improvement is usually worth the extra cost.

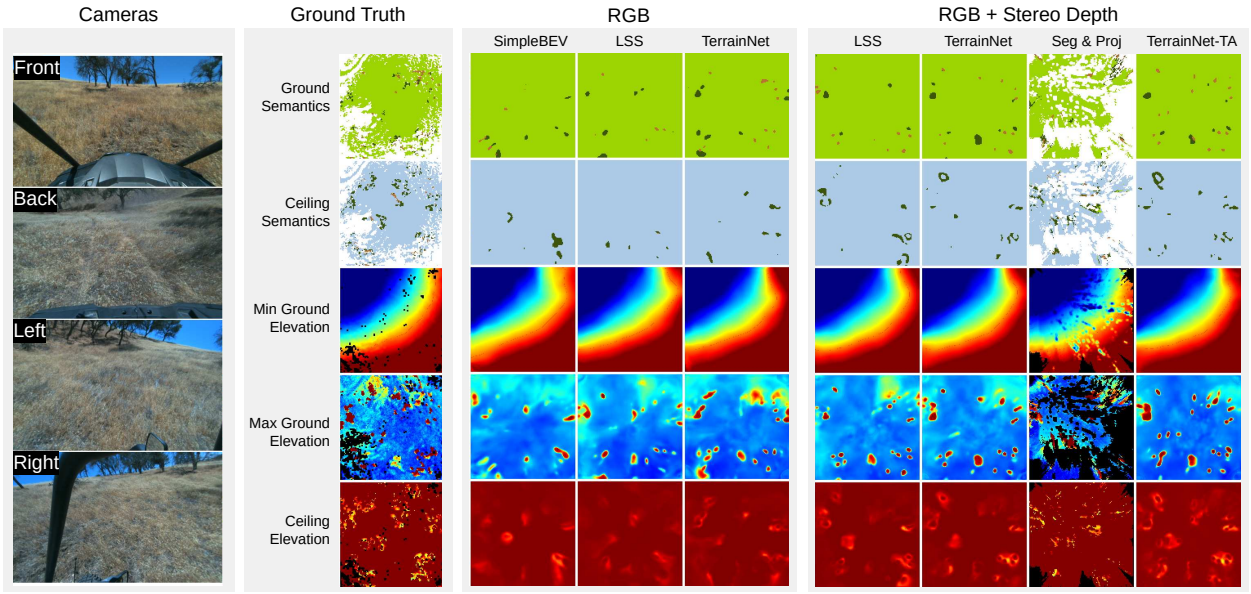


Figure 3.10: Qualitative comparison between TerrainNet and the baselines in one of the forest environments (best viewed when zoomed in). In general, RGB-D models see more trees. Seg&Proj shows many artifacts due to errors in stereo matching on the ground. TerrainNet-TA shows the highest fidelity. See Appendix B.5 for more examples.

Forward vs. backward projection. SimpleBEV performs on par with other methods on *dirt*, *dirt-trail*, and *grass*. These semantic classes cover a large area of the ground, so accurate depth estimation is not essential. But for objects such as *bush*, *canopy*, *tree*, and *rock*, SimpleBEV usually underperforms. SimpleBEV also performs poorly on elevation estimation, likely due to the low z-resolution of the grid. Increasing the grid resolution would linearly increase computation and memory consumption, which is not always feasible.

Learning-based inpainting is better than interpolation. In Figure 3.12, we compare the predicted ground elevation map from Seg&Proj with Navier-Stokes interpolation [16] and TerrainNet-TA. Only temporally aggregating the stereo depths creates an incomplete elevation map due to occlusion. A non-learning-based interpolation cannot leverage the context to predict the occluded areas, such as the hill crest and the flat ground behind dense vegetation. In contrast, TerrainNet learns shape priors to predict occluded areas more accurately. TerrainNet is also less

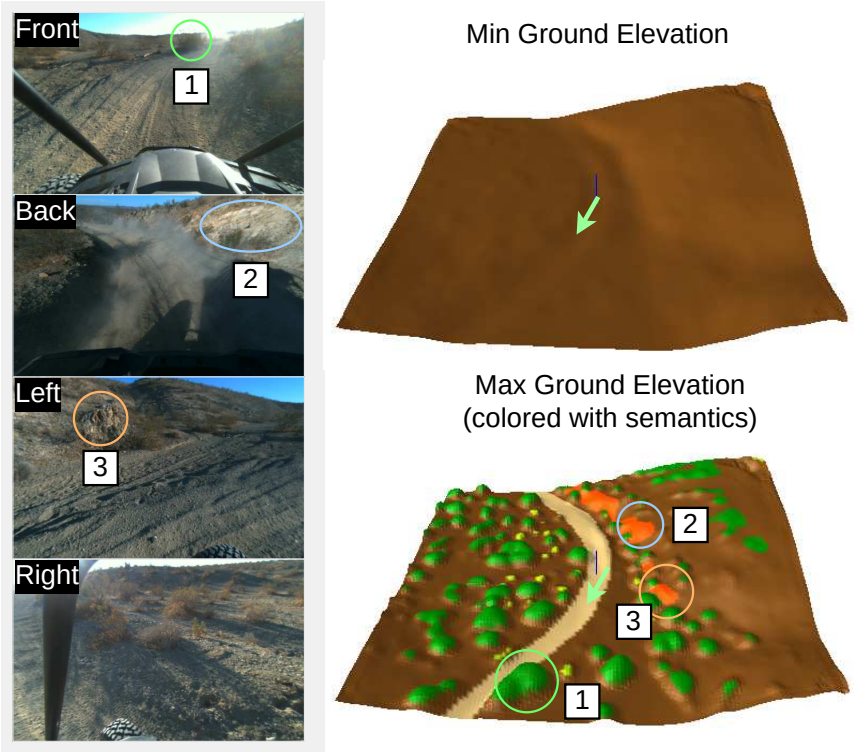


Figure 3.11: 3D visualization of TerrainNet-TA prediction on a high-speed on-trail sequence. The green arrow indicates the vehicle's heading. The Min Ground Elevation map shows a smooth ground support with porous objects such as vegetation removed. The Max Ground Elevation map contains the protruding vegetation on the ground. Note the bush far ahead (1) and rocks on the hills (2, 3) are well captured.

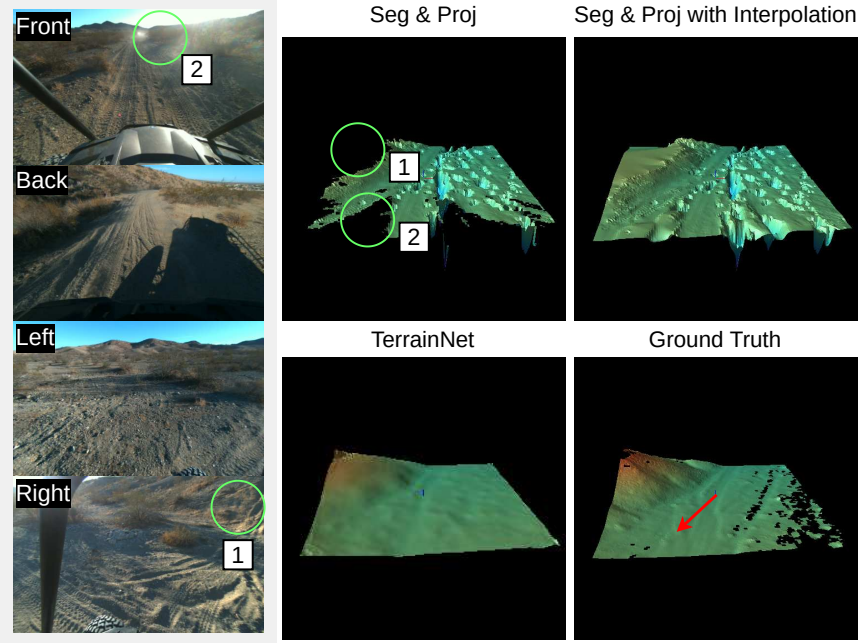


Figure 3.12: Comparing TerrainNet to Seg&Proj with interpolation. TerrainNet is able to predict the non-visible hill crest (area 1) and the occluded flat ground (area 2) behind dense vegetation, whereas interpolating aggregated stereo depths predicts a flat hill and a bumpy ground. TerrainNet is also robust to artefacts in stereo depths.

prone to the artefacts in the raw stereo depths.

Inference Speed Inference speed is vital for fast and safe off-road driving. In Table 3.3, TerrainNet is 7× faster than LSS. The significant speedup comes from the direct projection of point features, which avoids the cost of projecting the features to every location on a depth ray. Adding stereo depth to the input has a minor impact on the speed because it only introduces a few extra convolution layers. Similarly, the temporal aggregation module has a relatively low overhead.

Ablation Study

Architecture To understand the importance of each component in TerrainNet, we perform an ablation study by training a few alternative models with some of the components disabled. We summarize the results in Table 3.4 and have the following observations:

1. Without soft quantization (*"No soft quantization"*), the semantic mIoU degrades by 2 points, indicating the differentiable projection is crucial for correcting the end-to-end projection error from image space to BEV space. It has a small impact on elevation accuracy likely due to the terrain being naturally smooth.

2. The "*No z*" variant removes the elevation feature completely and only projects the image features. This results in a large error in ground elevation.

3. Removing the depth completion network and only using the raw stereo depth (*"No depth completion"*) degrades both semantic and elevation accuracy by a large margin.

4. We also trained a model using the ground-truth depth for the projection (*"Full model + GT depth"*). It has a sizeable improvement both in semantic mIoU (2 points) and elevation error.

The results show that every design decision made in TerrainNet is crucial for its performance. It also indicates that improving depth prediction accuracy is an effective way to improve the

terrain model.

Pseudo-labeling Since we train a pseudo-labeling model to generate the semantic BEV labels for the unlabeled training frames, we compare this scheme with two alternatives that do not leverage pseudo-labeling: 1) using only labeled data (about 4500 training frames), and 2) using the whole training set but not applying the semantic loss to unlabeled frames (note that elevation loss is applied to every frame).

Table 3.5 compares the pseudo-labeling scheme with the two alternatives on the validation set (we do not apply pseudo-labeling to the validation set). Pseudo-labeling provides the largest gain in semantic prediction (3.7 points) but at a small cost in elevation accuracy. Tuning the weight between the semantic and elevation loss can potentially mitigate this problem, which we leave as future work.

Map size We choose 50 m × 50 m maps for our main comparison because the cameras on the vehicle are tilted downwards such that the image content beyond 25 m gets heavily compressed and all models struggle at larger maps. Nevertheless, our model can scale to 100 m × 100 m maps better than Lift-Splat-Shoot. In Table 3.6 we compare Lift-Splat-Shoot, TerrainNet, and TerrainNet-TA on 100 m × 100 m maps with RGB-D inputs.

We see a similar trend in semantic accuracy: TerrainNet-TA performs the best in semantic prediction. In terms of scalability, TerrainNet is now 10× faster than LSS. TerrainNet projects each pixel to one map location, so its projection procedure is independent of map size.

Planning with TerrainNet

To show that the output of TerrainNet can be effectively used by a planner for off-road navigation, we also ran experiments with a planner in the loop.

	mIoU ↑	Ground Elevation (MAE) ↓
No soft quantization	0.449	0.254
No z	0.451	0.628
No depth completion	0.432	0.314
Full model	0.469	0.244
Full model + GT depth	0.496	0.232

Table 3.4: Ablation study with TerrainNet and RGB-D inputs.

Training data	mIoU ↑	Ground Elevation (MAE) ↓
Only manually labeled	0.432	0.281
Manually labeled + All elev.	0.427	**0.241**
Pseudo labeled + All elev.	**0.469**	0.244

Table 3.5: Effects of training data on the terrain modeling accuracy.

Method	mIoU ↑	Ground Elevation (MAE) ↓	Time (ms) ↓
LSS	0.399	0.628	402
TerrainNet	0.419	**0.586**	**39**
TerrainNet-TA	**0.464**	0.622	42

Table 3.6: Results on 100 m × 100 m maps with RGB-D inputs.

Planner. We use MPPI [174], a sampling-based model predictive control algorithm, as our planner as it is effective for high-speed off-road driving [173, 174]. At each time step, we perform 3000 rollouts with a kinematic bicycle model and evaluate the cost of each rollout using the terrain features along the trajectory, as described below. Then, we compute the optimized control trajectory as a weighted average of the rollouts, with the weights computed from the rollouts' aforementioned costs.

Cost function. The terrain cost C of a trajectory τ is a sum of costs evaluated at each state: $C(\tau) = \sum_{t=1}^{T} G(s_t) + M(s_t)$, where T is the planning horizon, s_t is the planar vehicle state (location on the map, velocity, and heading) at time step t, $G(s)$ is the distance from state s to some desired goal state, and $M(s)$ is the terrain cost at s.

The terrain cost M incorporates both semantic and geometric information based on the capabilities of the vehicle. There are several ways to do this. For the static comparative experiments in Figure 3.13, we compute it as follows:

$$
\begin{aligned}
M(s) &= c_{\text{semantic}}^{\text{ground}} + c_{\text{semantic}}^{\text{ceiling}} + c_{\text{elevation}} \\
c_{\text{semantic}}^{\text{groud}} &= (1 + \alpha_1(h_{\max} - h_{\min}))\gamma_{\text{ground}}^{\top} C_{\text{ground}} \\
c_{\text{semantic}}^{\text{ceiling}} &= (1 + \alpha_2(h_{\min} + h_{\text{clearance}} - h_{\text{ceiling}}))\gamma_{\text{ceiling}}^{\top} C_{\text{ceiling}} \\
c_{\text{elevation}} &= \beta_1 \theta_{\text{roll}}^2 + \beta_2 \theta_{\text{pitch}}^2,
\end{aligned}
$$

where α_1, α_2, β_1, and β_2 are scalar tuning parameters, γ_{ground} and γ_{ceiling} are tuning vectors of costs for each semantic class, and θ_{roll} and θ_{pitch} are the estimated roll and pitch angles of the vehicle based on the state s and the elevation map from TerrainNet. The height multiplier for semantic costs helps account for the fact that within any given semantic class, larger obstacles tend to be less traversable. The elevation cost penalizes large pitch and roll angles. Large pitch angles

Method	Avg. HD ↓	Avg. CD ↓	Time (ms) ↓
SimpleBEV	2.837	0.404	113
LSS	2.760	0.387	207
TerrainNet	**2.658**	**0.371**	**28**
TerrainNet-TA	2.758	0.384	29

Table 3.7: Planning evaluation with open-loop MPPI. Inputs are RGB-D. We compute the Average symmetric Hausdorff Distance and the Average Cost Difference between the optimized control trajectories on the predicted and the ground-truth costmaps.

may be too steep to ascend, whereas large roll angles may cause the vehicle to tip over. These angles depend on vehicle heading and can be computed using $h_{\min}$ (or $h_{\max}$) at the locations of the wheels.

For the real-world experiments (Figure 3.14), the terrain cost incorporates more physics:

$$M(s) = v^2 c^{\text{ground}}_{\text{semantic}} + C_{\text{lethal}} \mathbf{1}[c_{\text{rollover}} > \delta] \tag{3.8}$$

$$c_{\text{rollover}} = |\kappa v^2 + g \sin(\theta_{\text{roll}})| / \cos(\theta_{\text{roll}}), \tag{3.9}$$

where C_{lethal} is the lethal cost, $\mathbf{1}[\cdot]$ is the indicator function, v is vehicle velocity, κ is trajectory curvature, g is the acceleration due to gravity, and δ is a tuning parameter depending on vehicle geometry. Multiplying the semantic cost $c^{\text{ground}}_{\text{semantic}}$ by v^2 reflects the fact that collisions are more dangerous at higher speeds. c_{rollover} is the risk of rolling the vehicle based on speed, ground slope, and how sharply the vehicle is turning.

Static planning comparison. First, we set up an experiment on the validation dataset by running the MPPI planner on the costmap computed from the output of TerrainNet and other baselines (all with stereo inputs). The navigation task is as follows: we choose 12 waypoints in front of the vehicle at a distance of 25 m, spanning from -60° to 60°. Then, we run MPPI to plan a trajectory to each waypoint. Table 3.7 summarizes the results, and Figure 3.13 visualizes

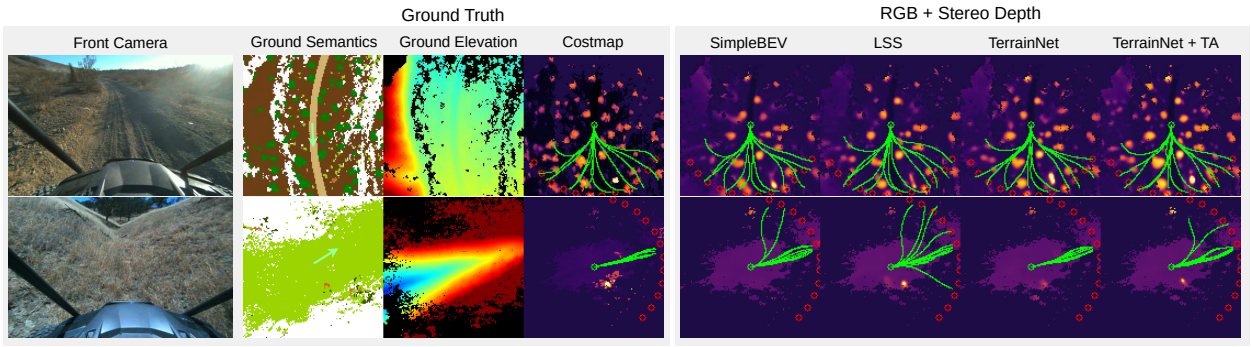


Figure 3.13: Visualization of MPPI control trajectories on two different terrains. Green arrow indicates vehicle's heading. Brighter areas correspond to higher costs. All models use RGB with stereo depth. **Top:** On-trail driving. The trajectories may go through non-lethal vegetation but do avoid the tree on the front left of the vehicle. **Bottom:** Driving in a steep valley. Due to the steep slopes, the trajectories should stay in the valley. We do not visualize the elevation cost here due to its dependence on vehicle heading.

example costmaps and trajectories. We observe the same trend where TerrainNet with RGB-D inputs performs the best. One interesting observation is that TerrainNet-TA performs worse than TerrainNet. This is likely caused by the planner being more sensitive to certain terrain features than others, and that the vehicle mostly driving forward (making history less useful). Currently, for generality, we do not discriminate between different terrain classes, but it would be beneficial to weigh their influence on planning during training. We leave this as future work.

Real-world navigation. TerrainNet has been integrated with an off-road autonomous driving stack and tested in real-world environments. Limited by the season, we tested TerrainNet in an off-road environment covered with deep snow and steep slopes. Since our dataset does not contain snow, we annotated 15 LiDAR scans with snow and finetuned our model with an updated ontology (including *snow* and *snow trail*). Figure 3.14 (best viewed zoomed in) shows sample costmaps, elevation maps, and planner output from an autonomous run. The system completed a 1.1 km run with a maximum speed of 7 m/s (average of 3.2 m/s) with two human interventions. The interventions were mainly due to errors in odometry from wheel slips. TerrainNet ran at

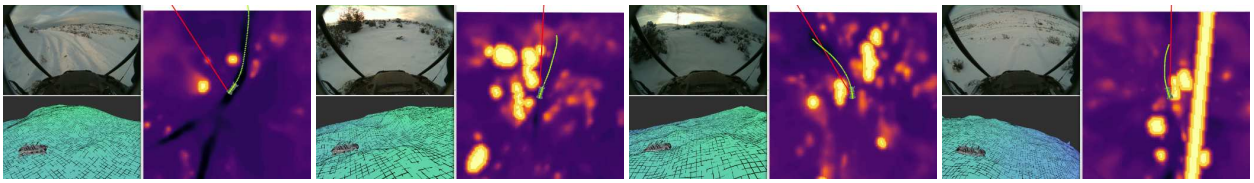


Figure 3.14: Qualitative results from a closed-loop real-world experiment. Each figure shows the front RGB camera (non-rectified), elevation map (showing h_{max}), and semantic costmap (showing c_{semantic}). Overlaid on the semantic costmap, the red line shows the direction of the goal point, and the green line shows the local plan. From the left: **(1) Driving on a trail with a hill to the left side.** Note that the trail is lower cost, so the local planner moves faster (as shown by the long green line). **(2) Driving off trail up a hill.** Note that the tall bushes on the left are very high cost. **(3) Driving uphill with many wheel slips on the snow.** The costmap and elevation map are "smeared" along the direction of travel, which highlights a limitation of temporal aggregation when odometry is poor. **(4) Approaching a steep downhill slope.** Although the slope itself is not yet directly visible, TerrainNet predicts a steep elevation map. The local planner slows down to reduce the risk of rollover. (The high-cost straight line through the costmap is a manually GPS-specified keep-out zone to mark the location of a nearby fence.)

20 Hz onboard the vehicle. Due to additional overhead from the system, TerrainNet provided map updates at 10 Hz. See the website for more details.

3.2.6 Discussion

LiDAR point density. One question a reader may ask is whether increasing the number of LiDARs is sufficient for building a dense map at high speed. While more LiDARs increase the point density, it also increases the cost of the system. Compared to typical deployment scenarios where robots may only have one LiDAR, cameras provide much higher pixel density at a lower cost. Cameras also provide additional advantages such as higher reliability, stealth, and being less affected by small particles such as dust. Nonetheless, it would be interesting to compare LiDAR-based mapping and camera-based mapping in challenging environments.

Limitations. While TerrainNet predicts high-fidelity terrain semantics and elevations, it has several limitations to be addressed:

- It may not generalize well to an environment that is substantially different from the training environments. This is common in learning-based systems. To improve the generalization, we can increase the diversity of the datasets and apply domain adaptation techniques.

- It may miss fine-grained features such as small, lethal rocks. Due to hardware limitations, the cameras are not precisely synchronized with the LiDARs (note that LiDARs are used to generate training data). Hence, there exists a small but noisy misalignment between the projected camera features and the ground-truth maps. This makes it harder for the model to learn fine-grained details of the terrain. To address this, we can improve the sensor alignment and make the model less sensitive to the localization error of small objects.

- TerrainNet does not predict uncertainty. This can lead to dangerous behaviors, such as the model predicting the other side of a cliff as a ramp. We can incorporate uncertainty estimation or risk assessment as done by [20] and [47] to make the model risk-aware.

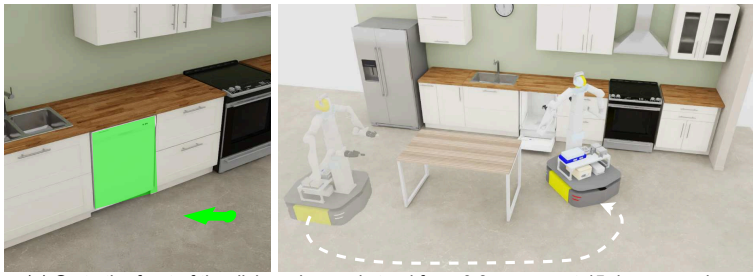
Chapter 4

Precise Local Navigation to Any Object

While there has been significant progress in learning-based systems that can navigate without a geometric map [44, 125, 126, 147, 148, 152], understand semantics [24, 125, 126, 128, 146], and follow human instructions [24, 126, 146, 186], these systems typically consider the goal reached when the robot is within 1m radius to the goal [125, 181]. This lax definition of success hinders their applicability to the growing need for mobile robots to navigate to objects with *high precision*[1]. For example, in a factory, a forklift must position itself correctly to a pallet so that the fork can be inserted into the pallet without collision (Fig. 4.1b); an inspection robot can more clearly read gauges when facing instruments perpendicularly at a proper distance. Similarly, a home robot needs to position itself properly to open a dishwasher (Fig. 4.1a), dock to a charging station, or place objects at accurate locations (e.g. *left* side of a table). Lack of precision in the final pose of the robot would incur failures due to collision, out-of-reach, or not adhering to task requirements.

Precision navigation requires a robot to understand the geometry of relevant objects (*where is the goal?*) and the local scene structure (*how to get there?*). One classical approach would be

[1]We use the word "precision" to indicate both low error and high consistency. This is different from the conventional meaning of "precise" that only refers to high consistency.



(a) Go to the front of the dishwasher and stand from 0.8 m away at 45 degree angle

(b) Go to the front of the pallet and stand from 2.0 m away at 0 degree angle

Figure 4.1: **Overview of AMR**. Given a masked image describing the target object and an object-centric pose (relative position and orientation), AMR tracks the object while moving, avoids obstacles, and aligns the robot to the target object with centimeter-level precision without maps or object 3D models.

to estimate the object's pose, from which the desired robot pose can be derived. But this usually requires specific object information such as 3D models [172], and the object being initially visible. This limits its applicability when the object 3D model is not available or the object is initially out of view. On the other hand, current learning-based navigation [44, 125, 126, 128, 146, 147, 148] is not quite applicable to real-world high-precision navigation tasks, as they lack precise goal conditioning, often assume discrete action spaces [44, 105], or trained on imprecise real-world data [147, 148].

In this work, we propose *Aim My Robot* (AMR), an end-to-end vision-based local navigation model that navigates to objects with *centimeter-level* precision. AMR does not require an object CAD model, and instead uses a reference image with object mask and relative pose for precise goal specification. AMR takes streams of RGB-D and LiDAR data as inputs and outputs trajectories directly, eliminating the need of a metric map. We achieve high precision and strong generalization

via two contributions: 1) a data pipeline for generating large-scale, photorealistic, and precise trajectories to diverse objects; 2) a transformer-based model that takes multi-modal sensor inputs (RGB-D + LiDAR) and plan precise and safe trajectories. Experiments show that AMR achieves a median error of 3 cm and 1° on unseen objects, with little degradation when deployed on a real robot. It supports robots of different sizes, and can be finetuned quickly to support more complex kinematics such as Ackermann steering vehicles. AMR is the first system for high-precision visual navigation with strong sim2real transfer. We hope it paves the way towards precision robot autonomy.

4.1 Related Work

Object-goal and instance-goal navigation: Object-goal navigation typically refers to a robot navigating to any object of the specified category (e.g., couch) [28, 58, 105]. Instance-goal navigation uses a close-up view of a specific object instance as the goal [24, 85, 86]. In both cases, success is triggered if the robot is within the neighborhood of the object (e.g., 2.0 m radius in [86]). Such tolerance is insufficient when robot-object interaction, such as docking, inspection, and manipulation, is required. Interestingly, recent works move from end-to-end learning back to classical approaches [24, 58, 85], citing the challenges of sim2real transfer with learning-based methods. In this work, we use precise goal conditioning, carefully designed models, and large-scale photorealistic training data to show that precision navigation trained in simulation achieves strong generalization to real-world scenarios.

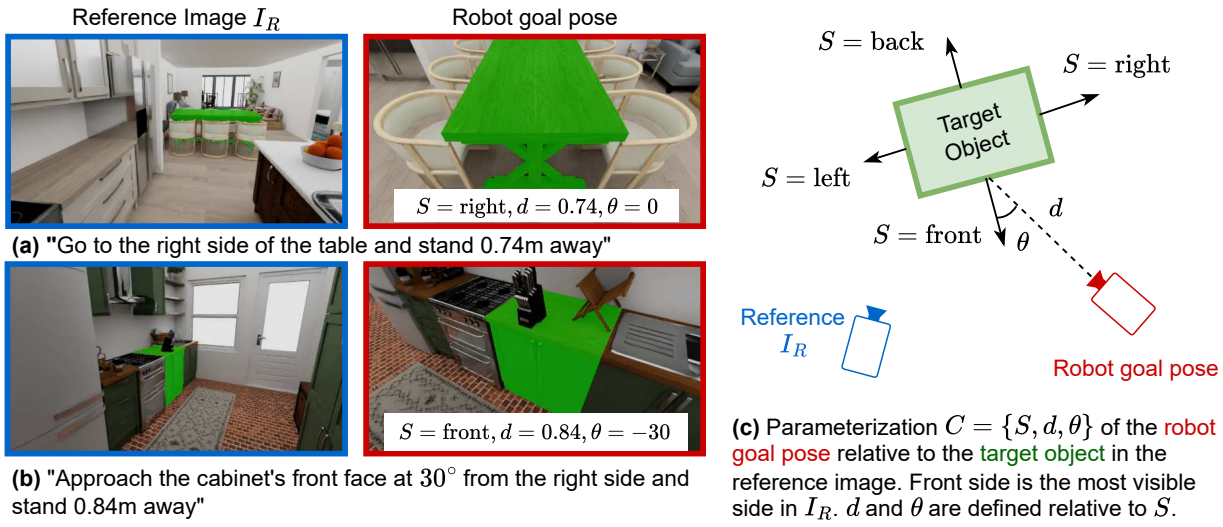
Mobile manipulation systems: There is a surge of interest in developing learning-based mobile manipulation systems [44, 55, 74, 75, 169, 184]. Some works assume additional information such as object pose or robot pose to be available ([74, 75, 169, 184]). Other systems show that

it is possible to directly map sensor observations to actions [44, 55], but they only consider a limited set of scenarios (e.g. no need to avoid unseen obstacles or handle unseen objects), so that high-precision base positioning is not a major concern. As of today, the most general and capable mobile manipulation systems are still modular [9, 101], and they require mapping the environment and objects beforehand. In particular, [101] shows that accurate base positioning is crucial for manipulation to succeed. Our work can be integrated into both learning and modular mobile manipulation systems so that a robot can reach an accurate base pose for manipulation without a prior map or object model.

4.2 Problem Definition

We consider the scenario where the robot has reached the vicinity (<10m, about a room size) of the object of interest and needs to perform a *local*, *object-centric* high-precision maneuver for docking, inspection and manipulation. We formulate this as a local navigation task where the goal G consists of the target object and the relative base pose (in $SE(2)$) to the object. For generality, we assume G is provided by another system. The robot has a tilt-enabled forward RGB-D camera at 1.5m high and a 2D LiDAR mounted on the base, providing 360° coverage. Concretely, the robot is given past sensor observations $\{o_t, o_{t-1}, ...\}$ along with the goal G, and needs to move its base to reach the specified pose while tilting its camera to gaze at the object. The robot does not have the object's 3D model or a 2D/3D map.

Goal specification. Due to the lack of the object's 3D model and a map, one challenge is how to unambiguously define the goal G. To address this, we assume the robot is given a reference image I_R with the target object mask M (Fig. 4.2a and 4.2b). I_R can be taken from the robot's long-term memory or from its recent observations. To make goal specification object-centric, we use the



(a) "Go to the right side of the table and stand 0.74m away"

(b) "Approach the cabinet's front face at 30° from the right side and stand 0.84m away"

(c) Parameterization $C = \{S, d, \theta\}$ of the robot goal pose relative to the target object in the reference image. Front side is the most visible side in I_R. d and θ are defined relative to S.

Figure 4.2: **Problem setup.** We specify the target object via a reference image I_R taken in the scene and an object mask M (in green). The goal condition C is defined as the relative side and pose of the object in I_R. A robot needs to navigate to the object conditioned on C and tilt its camera to gaze at the object.

fact that common objects have 4 dominant sides (i.e. described by its bounding box). Denoting the most visible side in I_R as the *front*, then the relative pose can be defined as $\mathbf{C} = \{S, d, \theta\}$, where $S \in \{\text{front}, \text{back}, \text{left}, \text{right}\}$ is the approach side, $d \in [0.0\text{m}, 1.0\text{m}]$ is the approach distance, and $\theta \in \{0^\circ, \pm 15^\circ, \pm 30^\circ\}$ is the approach angle. See Fig.4.2c for an illustration. Hence, $G = \{I_R, M, \mathbf{C}\}$.

Discussion. Existing object instance navigation systems require taking a close-up view for each object [86] or mapping object locations [9, 24, 101]. In comparison, our formulation uses *mask* to specify the object instance in an image. It does not require a close-up view or building a map. Moreover, it can be interfaced with a high-level task planner (e.g. SAM [84], TAMP [57] and LLM [98]) that outputs mask and pose parameters whereby AMR guarantees *precise* base and camera positioning.

Figure 4.3: Example rendered images of HSSD scenes in Isaac Sim.

4.3 Aim My Robot

In this section, we detail our technical approach. We first describe our data pipeline that generates large-scale, high-quality demonstrations entirely in simulation. Then, we introduce AMR, a multi-modal architecture for robust and precise local navigation.

4.3.1 Data Generation

Achieving strong generalization and high precision requires training data containing diverse scenes, objects, and precise trajectories. Since humans are poor at estimating distances and angles from images, we forgo teleoperation and leverage simulation and model-based planning for data collection.

Figure 4.4: **Sample objects in the scenes.** We consider all objects, including those that are not semantically labeled.

Assets and simulator. We import the Habitat Synthetic Scenes Dataset (HSSD) [80] which contains diverse room layouts (100+) and objects (10,000+) into Isaac Sim. Since HSSD contains PBR textures, Isaac Sim is able to render photorealistic images using ray tracing (Fig. 4.3). The diversity and realism of the perception data enable strong visual sim2real transfer.

Sampling goal condition. We model the robot as a cylindrical rigid body with a radius R sampled from $[0.1\text{m}, 0.5\text{m}]$. The robot is randomly placed in the traversable area of a room. A reference image I_R is rendered either from the robot's initial camera view or from a random camera view in the room. Then, the target object mask M (obtained from the simulator) is randomly chosen from I_R. Non-objects such as walls and floors are excluded. We sample a goal specification $\mathbf{C}$ by randomly choosing a side $S \in \{\text{front}, \text{back}, \text{left}, \text{right}\}$, distance $d \in [0.1\text{m}, 0.5\text{m}]$ and angle $\theta \in \{0°, \pm 15°, \pm 30°\}$. Finally, we place the robot at the goal and check for collisions.

Trajectory generation. The robot kinematic model is assumed to be a differential-drive robot. We run AIT* [155] using the ReedsShepp state space with a turning radius of 0 for planning the base motion. It is straightforward to change the state space to model robots with other kinematics.

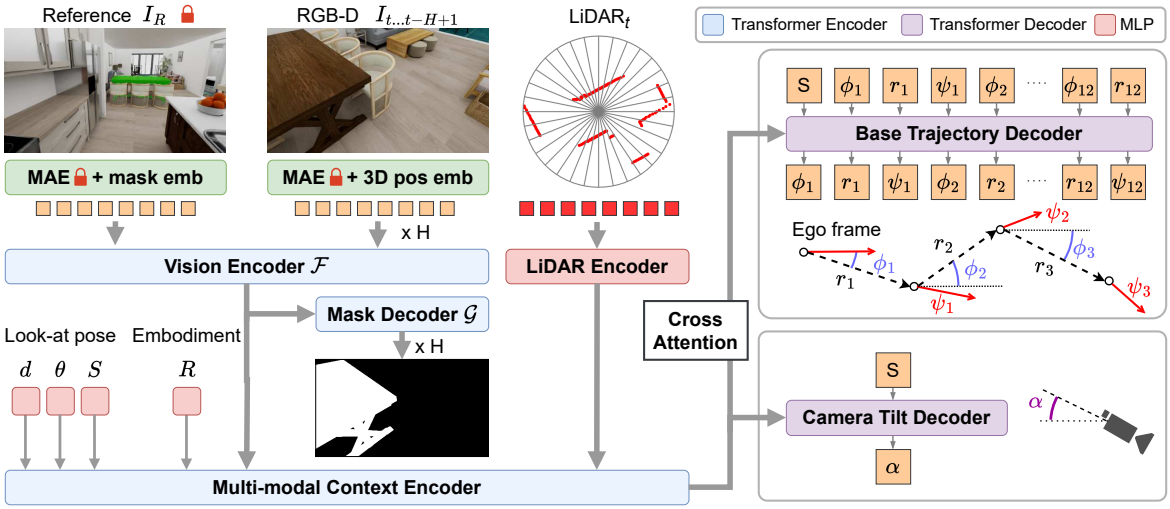


Figure 4.5: **Network architecture.** The reference image I_R and the robot's RGB-D observations I_t are tokenized with MAE [67]. The current LiDAR scan is tokenized by grouping points into directional bins. Image and LiDAR tokens are input into the multi-modal context encoder jointly with the look-at pose and embodiment tokens. Finally, the output tokens of the context encoder are cross-attended to the base trajectory decoder and the camera tilt decoder. S are the learned start tokens for the decoders.

The cost function encourages the camera to look toward the goal while penalizing excessive backward motion (small backward motion is allowed if the robot can reach the goal faster). For each feasible path, we render the camera observations along the path with a distance gap of 0.2m or an angular gap of 5°. The camera tilt angle is set such that the lowest vertex of the object mesh appears at ¼ above the bottom of the image (even when the object is out of view).

4.3.2 Model

AMR is a transformer-based model (Fig. 4.5) based on two design principles: 1) a unified approach to representing multi-modal sensory data, and 2) an action decoding scheme that generates precise and collision-free actions. The system is divided into three stages: multi-modal sensor encoding, goal-aware sensor fusion, and multi-modal motion generation. We detail each stage as follows.

Encoding Multi-modal Sensor Data

We use RGB, depth, and 2D LiDAR observations to achieve high precision and robustness. RGB is used for visual reasoning, depth provides precise 3D geometric information, and LiDAR ensures a 360° coverage for robust collision avoidance.

Encoding RGB-D: The current RGB image I_t ($224 \times 224 \times 3$) is passed into a frozen MAE-Base encoder [67] to obtain a $14 \times 14 \times 512$ feature map. Each depth image is resized to 14×14 and we compute the spatial location of each depth pixel in the robot's egocentric coordinate frame via $[x', y', z'] = \mathbf{R}\mathbf{K}^{-1}[x, y, d]^T + \mathbf{t}$, where $\mathbf{K}$ is the camera intrinsics and $[\mathbf{R}|\mathbf{t}]$ is the camera extrinsics. We apply a sinusoidal position embedding f for x', y', z', respectively, and concatenate them to obtain the position embedding for each depth patch as $[f(x'); f(y'); f(z')]$. We add depth position embedding to each corresponding RGB patch. To incorporate past observations I_{t-i}, we set the camera extrinsics $[\mathbf{R}|\mathbf{t}]$ to be the transform between the camera pose at $t - i$ and the robot base pose at time t, which can be obtained from the robot's odometry. Compared to concatenating RGB and depth tokens which require $2x$ number of tokens, using depth as positional encoding is more efficient since it keeps the number of visual tokens unchanged.

Encoding LiDAR: We resample the LiDAR points into $N = 256$ points. The points are grouped into 32 directional bins. The points in each bin (8 of (x, y) coordinates) are passed into a Multi-layer Perceptron (MLP) to obtain one LiDAR token. In total, there are 32 LiDAR tokens.

Goal-aware, Robot-aware Sensor Fusion

We tokenize the reference image with the same frozen MAE. The mask M is encoded by a shallow convolutional network, similar to [84]. All the visual tokens are flattened and passed to the Vision Context Encoder $\mathcal{F}$. The output tokens corresponding to the robot's observations are decoded into

target masks using the mask decoder $\mathcal{G}$. This helps $\mathcal{F}$ to learn to track targets. The embodiment token encodes the robot's radius R. Finally, the visual tokens, LiDAR tokens, goal tokens, and embodiment token are input to the Multi-modal Context Encoder. The output serves as the context for motion generation.

Motion Generation

We use a separate transformer decoder for the base movement and the camera tilt angle, respectively. AMR predicts base trajectory and camera tilt angle at different frequencies. For base trajectory, it operates in a receding-horizon fashion: the robot follows the predicted trajectory up to T steps and then predicts a new trajectory given the updated robot observations. For camera tilt angle, it predicts a new value at every time step. This separate decoding saves computation since the trajectory decoder only runs once for every T steps.

Base trajectory decoding: The base trajectory is parameterized by a sequence of waypoints in egocentric polar coordinates. To capture the multi-modal nature of robot trajectories, we use an autoregressive transformer decoder. Since autoregressive classification requires discretizing the actions, to preserve the precision, we adopt multi-token classification with residual predictions. Specifically, we represent each waypoint by a sub-action tuple (direction $\psi_i \in [0, 2\pi]$, distance $r_i \in [0, 0.2\text{m}]$, heading $\phi_i \in [0, 2\pi]$), $i = 0, ..., N-1$ and sample ψ, r, ϕ conditionally in sequence. This representation reduces the number of bins required for classification, as classifying ψ, r, ϕ at once would require a combinatorial number of bins. In practice, we discretize ψ, r, ϕ into 30, 32, 12 bins, respectively. For each output token z, we recover the continuous value $z' = C(z) + R(z, C(z))$ where $C(\cdot)$ is the output from the classifier, and $R(\cdot, \cdot)$ is an MLP that predicts the residual. Our scheme is faster than diffusion [33] and more temporally consistent than BeT [145].

Camera tilt angle decoding: Since camera tilt control is uni-modal, we continuously predict the camera tilt angle α via regression at each time step t.

4.4 Implementation Details

Dataset. We converted 54 HSSD [80] environments into Universal Scene Description (USD) format and generated 500k trajectories (7.5 M frames) in Isaac Sim [120]. We use 49 scenes for training and 5 scenes for evaluation. There are 3,119 target objects in the training scenes and 275 target objects in the test scenes. Among the 275 test objects, 160 are unseen during training. We randomly sampled robot starting locations and target objects, and created 2000 navigation tasks in unseen environments for evaluation.

Training. We train AMR for 150k steps with a batch size of 128 on 8 A100 GPUs. We use the following loss function:

$$L = L_{\text{mask}} + L_{\text{base}} + L_{\text{tilt}}$$

where L_{mask} is pixel-wise L2 loss, L_{base} is a sum of classification loss and regression loss akin to [145], and L_{tilt} is an L2 loss that regresses the camera tilt angle.

After the initial Behavior Cloning phase, we ran the model in simulation and identified failures (collision, tracking loss, out of tolerance), and used DAgger [135] to augment the dataset. For each failure, we run the planner on states *before* the failure point to generate the corrective plans. We create a new dataset by rendering the observations of the new plan. Due to the high cost of generating on-policy DAgger trajectories, we use the same DAgger dataset for all models.

Allowing sideways motion in the DAgger phase. A differential drive robot is incapable of sideways motions, which is inefficient for making small lateral adjustments. Since an omnidirectional base can traverse laterally, we allow a small amount of sideways motion in the DAgger

phase. Specifically for the omnidirectional robot, if the robot is within 0.5 m of the goal, we remove the kinematics constraint imposed by the differential drive mechanics and let the robot "teleport" to the goal.

Deployment. We deploy AMR on a real mobile manipulator with an omnidirectional mobile base [4] and a simulated forklift with Ackermann steering in Isaac Sim. Each trajectory contains $N = 12$ waypoints, separated by $dt = 0.2s$. To track the predicted trajectories, we use a Pure Pursuit controller [39] for the omnidirectional robot and a Model Predictive Control (MPC) controller for the forklift. The camera tilt angle is updated at every time step, whereas the trajectory is updated with a new prediction at every 8 steps. The models run at 12 Hz on a RTX 3090.

4.5 Experiments

We perform **quantitative experiments** both in simulation and in the real world. Each run performs closed-loop inference until the robot comes to a stop, collides with obstacles, or exceeds the time limit. We report the final distance and orientation error relative to the goal. Additionally, we report the collision rate in the ablation study. In the real-world experiments, we measure the ground truth goal pose against an occupancy map with 2 cm resolution. The robot's ground truth pose is obtained by classical Monte Carlo Localization [53].

For experiments with HSSD, we trained all models without history (*hist=0*), as we found that history did not show noticeable benefits in simulation. In our real robot experiment, we trained a model with 4 past frames and compare it against *hist=0* model.

We conduct additional **qualitative experiments** to highlight generalization to novel tasks using AMR: We show generalization to new tasks and scenes in the real-world, closing a fridge drawer using the robot body, and a forklift picking up a pallet.

Figure 4.6: **Test scenes used for quantitative evaluation in simulation.** These scenes have diverse layouts, objects, and textures.

4.5.1 Simulation Results with HSSD

Baseline: We compare our method against a classical pipeline based on object pose estimation. We use FoundationPose [172] to estimate the pose of the target object in the initial observation. The posed bounding box is used to compute the goal pose as described in Sec. 4.2. Then, we run the same planner to find a path using the groundtruth occupancy map, and navigate the robot to the goal. Note that such a system uses extra information not required by AMR: 1) a CAD model of the target object; 2) the object is visible in the robot's initial observation (required by pose estimation); and 3) perfect mapping.

Quantitative Results: Fig. 4.7 compares the error distribution between AMR and the classical approach. We consider two cases: FFR (First Frame is Reference) and non-FFR. FFR uses the initial observation as the reference image, which is required by object pose estimation. In the FFR scenarios, AMR performs well even when the object is far. It achieves significantly lower distance and angular errors than the classical approach. One interesting observation is that the angular error for the baseline increases when objects are too close ($0 - 2$m). This is usually caused by large objects (e.g. furniture) being partially out of view. In contrast, AMR actively moves the robot, making it more robust to state estimation errors. In Table 4.1 we compare the median pose errors between the seen objects and unseen objects in the test scenes. We do not see a clear gap, showing that AMR generalizes well to unseen objects.

Impact of object visibility. Non-FFR mode shows higher errors since the target object may not be visible in the initial observation. To study the impact of object visibility, we group the test cases based on whether the objects are initially visible, and present the results in Fig. 4.8. Object visibility has only minor impact on the accuracy when the object is within 4 m range. AMR is able to leverage visual context to navigate well. When objects are initially invisible *and*

Category	**Seen** (136 unique objects) MAE (°)	MDE (m)	**Unseen** (166 unique objects) MAE (°)	MDE (m)
Chair	1.45	0.05	1.08	0.04
Drawers	0.77	0.02	0.55	0.03
Couch	0.82	0.02	0.57	0.04
Picture	0.84	0.03	0.64	0.04
Shelves	0.65	0.07	0.68	0.02
Tables	7.14	0.04	1.20	0.04
Unlabeled	0.78	0.03	0.74	0.04

Table 4.1: Comparison of median angular errors (MAE) and median distance errors (MDE) for seen and unseen instances across shared categories in the test scenes.

too far, the median errors are stable, but the worst case performance degrades significantly. This highlights the local nature of AMR and its limited capability of searching for hidden, distant objects. Increasing the history window of AMR would help this aspect.

Qualitative results: Fig.4.9a and 4.9b present two successful runs (one FFR and one non-FFR) with different goal poses and robot sizes. AMR plans safe trajectories by considering the robot's size, and is able to track and reach target object precisely even when the target is far or out of view. Fig.4.9c shows typical failure cases for both the baseline and AMR. For the baseline, the failure is mostly caused by pose estimation in challenging scenarios, such as occlusion and object being too far. AMR sometimes fails to track the correct object when there are repetitive objects, and may go to the wrong side when the viewpoint is ambiguous.

4.5.2 Ablation Study

To verify our design choices, we ablate our models by disabling some of the components and analyze their impacts on the precision (Fig. 4.10) and robustness (Table 4.2). In particular, we find **Sensor Modality** has the biggest impact. Without LiDAR, the robot is more prone to colliding

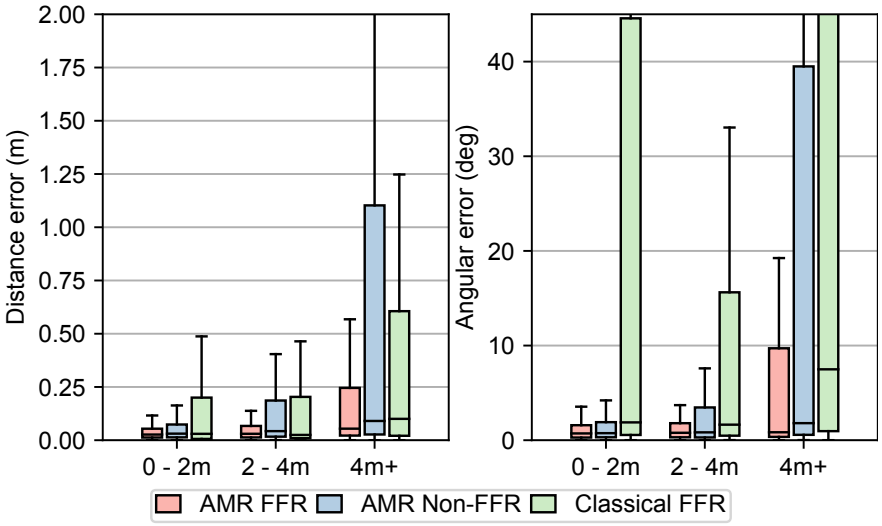


Figure 4.7: **Navigation accuracy in the test scenes for various object distances.** FFR refers to trajectories where the First Frame is the Reference image (FFR). Classical pose estimation only supports FFR.

Full	No mask dec.	ACT decoder	No DAgger	No depth	No embodiment	No LiDAR
3.8	7.1	5.8	5.5	17.7	26.2	25.6

Table 4.2: Collision rate (%) of ablated models.

with surrounding objects. Likewise, the 3D information from depth helps the robot avoid obstacles and approach targets accurately. Without the **Embodiment** token the model suffers from a high collision rate, because the robot cannot take its footprint into consideration when planning. Our **Autoregressive Trajectory Decoding** scheme outperforms Action Chunking Transformer (ACT) since it better captures the multi-modal nature of navigation trajectories. While **Mask Tracking** only moderately improves the precision, it improves the model interpretability, as we find strong correlation between incorrect mask tracking and robot navigating to the wrong object.

Large-scale evaluation matters: In Fig. 4.10, all ablated models have comparable median errors, indicating that they perform almost equally well for more than half of the test cases, but

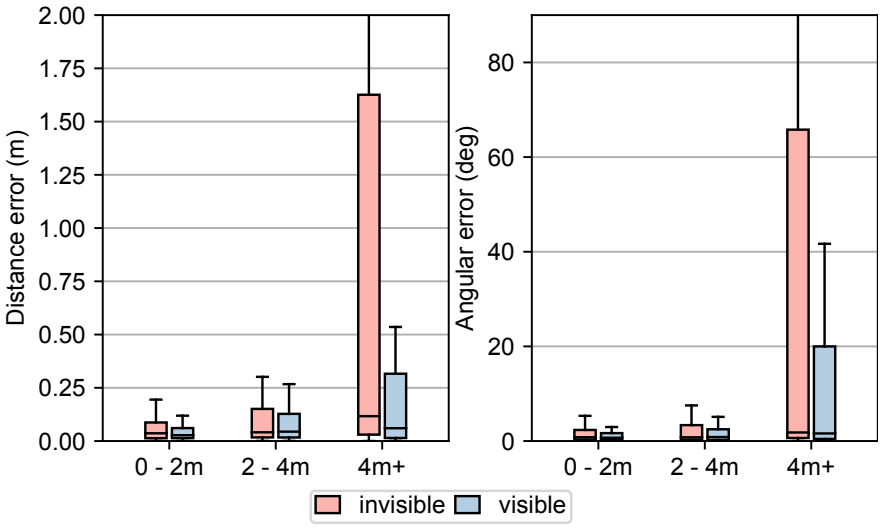


Figure 4.8: **Impact of initial object visibility on navigation accuracy for various object distances in Non-FFR mode.** AMR performs well regardless if the object is initially visible or not. When the object is not visible and too far, AMR experiences performance degradation for worst-case scenarios.

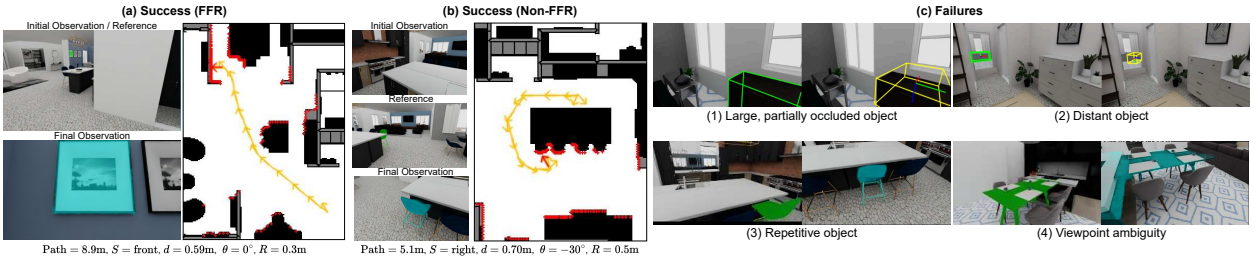


Figure 4.9: **Qualitative examples: (a) FFR** (object initially visible) **(b) Non-FFR** (object initially out of view). The initial masks are specified in the reference images (green). Cyan masks are predicted by the model. Trajectory accommodates the robot radius R for obstacle avoidance. **(c) Typical failures:** (1,2) Baseline: Object pose estimation may fail when the object is partially occluded or too far (green box: groundtruth, yellow box: prediction) (3) AMR may go to the wrong object when the object is repetitive. (4) AMR and Baseline: a robot may go to the wrong side when there is no dominantly visible side (i.e. looking at an object from 45° angle). This can be addressed by using a less ambiguous reference image.

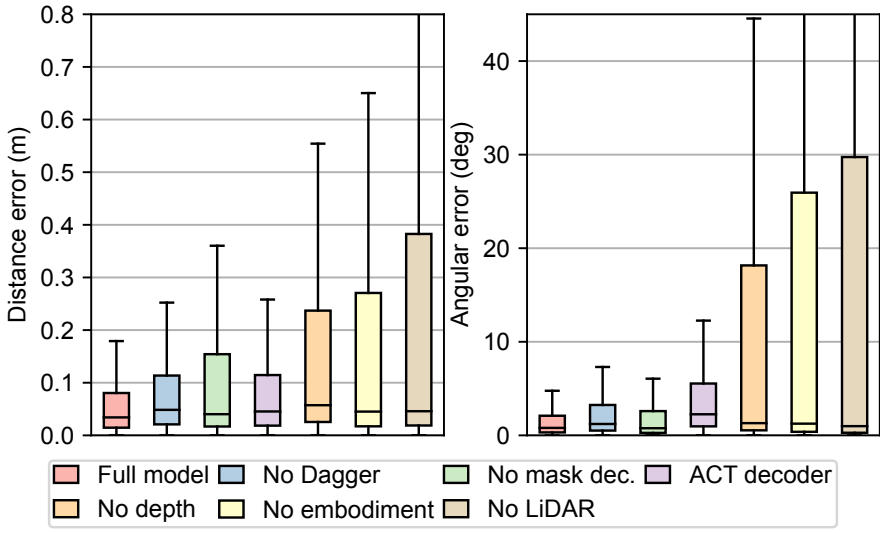


Figure 4.10: Ablation study demonstrating effects of various choices on navigation accuracy in our test scenes.

their long-tail failure cases vary significantly. This implies that our method is more *robust* across large datasets with environment variations.

4.5.3 Closed-loop Hardware Evaluation

We evaluate our system in a real kitchen on a mobile manipulator with an omnidirectional base and an RGB-D head camera with tilt support. We use one reference image covering the whole kitchen to specify 4 large objects (*fridge, sink, stove* and *cabinet*) and two zoomed-in images for small targets (*spam* and *cup*). The robot is initialized at three distinct starting locations. For each run, we continuously run AMR to navigate the robot to all 6 objects, with the relative goal condition describing the robot facing the object of interest, i.e., C=($d = 1$ m, $\theta = 0°$, S = front). For each object, we perform 3 runs.

In Table 4.3, we show the number of successful trials and compute the errors against hand-measured ground truths. *hist=4* succeeded in all tasks except for *spam* starting at location #3. Unlike in simulation, history is critical for the robot to track the target, as *hist=0* fails more often

Figure 4.11: **Real kitchen experiments. Left:** reference images and contours of target object masks. While the *fridge* is partially out of view, the model can still reach the fridge. **Middle:** Initial view of the scene through the tilt camera. **Right:** robot observations after reaching each object overlaid with predicted masks (in cyan).

	Fridge	Sink	Stove	Cabinet	Spam	Cup
Hist=0	1 / 2.4 cm / 2.3°	3 / 14.8 cm / 0.1°	3 / 2.0 cm / 0.8°	2 / 3.1 cm / 1.7°	1 / 1.8 cm / 0.6°	3 / 9.6 cm / 0.8°
Hist=4	3 / 3.1 cm / 0.4°	3 / 7.9 cm / 0.2°	3 / 2.4 cm / 0.8°	3 / 2.1 cm / 0.6°	2 / 2.0 cm / 0.1°	3 / 8.7 cm / 0.9°

Table 4.3: **Navigation accuracy in a real kitchen.** For each object we report (#successes / distance error / orientation error). 3 runs were performed per object. Cells are colored green according to the number of success runs.

on large (*fridge*) and small (*spam*) objects. Among the objects, *sink* has the largest distance error due to unstable tracking and the lack of a large surface area, as the sink is an object that is concave to the tabletop surface and can easily be obscured. With the exception of *sink* and *cup*, all other objects achieve distance errors on the order of 1.8 ~ 3.1 cm to the goal specification.

4.5.4 Qualitative Experiments

Generalization in new real-world scenes: In Fig. 4.12, we show additional results of AMR navigating to diverse objects in another real-world environment. AMR is able to handle objects being out of view and large viewpoint change (i.e. going to the back of a cabinet). We refer the reader to the website for the videos.

Closing the fridge drawer: In Fig.4.13, we show that by accurately aiming the robot to a target object a robot can perform downstream tasks using its body as the end effector. The goal is specified by $M = I_{\text{drawer}}$ and $\mathbf{C} = (d = 1 \text{ m}, \theta = 0°, S = \text{front})$. Once the robot reaches the goal, the robot moves forward to close the drawer in open loop.

Loading a pallet in a warehouse: We test AMR on a simulated forklift to load a randomly placed pallet in a warehouse. Existing work on controlling forklifts requires sophisticated modeling and motion planning [158, 161] with privileged state information. Here, we show that a learning system with onboard sensors can achieve the same precision. We empirically generate $\sim$ 500 demonstrations with forklift dynamics to fine-tune AMR. In Fig.4.13, we show AMR successfully navigates a forklift to face a pallet directly $\mathbf{C}$=(d=2 m, θ=0°, S=front). The precision is sufficient such that the fork can be completely inserted into a pallet by driving the forklift forward in an open loop fashion. More information is provided in the supplementary material and website.

4.6 Discussion

We present AMR, a vision-based navigation model that navigates to any object with centimeter precision. While trained completely in simulation, it transfers to real-world and unseen objects with little degradation in accuracy. AMR does not require object 3D models or a map to operate,

Figure 4.12: **Real-world qualitative experiments in new scenes. Left:** reference image with target object highlighted by the green mask. **Remaining columns:** robot's camera view at $T = \{0, 1/3, 2/3, 1\}$. The tracked object is highlighted by the cyan mask. The tasks are: **(a)** Go to the back of the cabinet and stand 0.8m away. **(b)** Go to the front of the landfill trashcan and stand 1m away. **(c)** Go to the front of the table and stand 0.5m away.

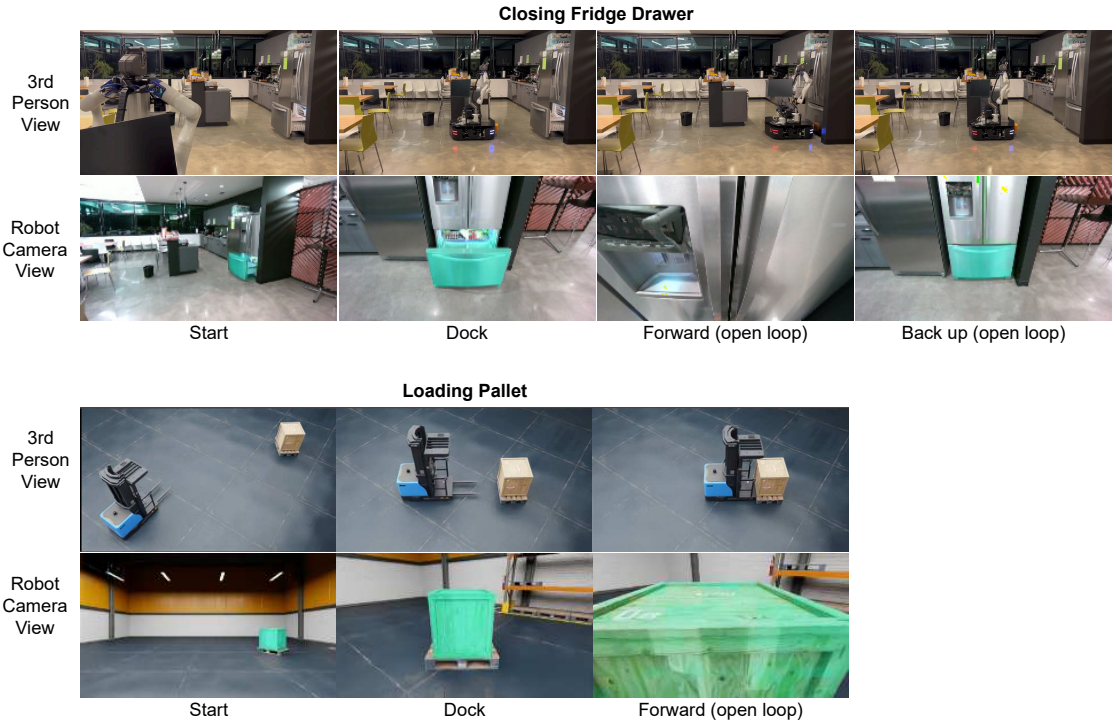


Figure 4.13: **Using AMR for mobile manipulation tasks. Top:** closing a fridge drawer by pushing the robot body towards the drawer. **Bottom:** forklift loading a pallet. AMR accurately tracks the target objects throughout.

runs in real-time at 10 Hz, and easily adapts to non-holonomic robots with fine-tuning.

Limitations. AMR is designed for local navigation as it is equipped with a short-term memory. Since it is only trained in household environments, its generalization in other scenarios such as factories and unstructured environments is potentially limited. Additionally, real robots have complex geometric shapes and sensor placements, and our assumption of a robot being cylindrical with a centered camera can hinder its applicability to diverse mobile robots such as legged robots. However, we anticipate that these limitations can be addressed by increasing the robot's memory window, improving the diversity of the training environments, and more comprehensive modeling of robot shape and sensor placements.

Chapter 5

Conclusion

In this thesis, we presented several learning-based approaches to robot-centric autonomy. First, we tackle the problem at the global scale. We show that robots can construct efficient topological environment representations by learning robot-dependent reachability and behaviors. Then, we go down to the semi-local level to study how to build a comprehensive robot-centric terrain model for safe navigation on off-road terrains. At last, we study the "last-mile" scenario. By adopting a robot-centric goal specification and action parametrization, we enable a robot to navigate to any object with centimeter-level accuracy without a global map.

But how far are we from building autonomous robots that surpass human capabilities? While robots already outperform humans in controlled environments for specific tasks, they remain far from operating robustly across *any task* in the wild. To bridge this gap, we outline a few promising directions:

Better world representations. A key limitation of current robot learning models is the choice of world representation. Using single RGB images are convenient but they provide limited knowledge about the world. Consequently, these models struggle with even minor changes in setup, such

as scene appearance, camera viewpoint, or robot embodiment. Metric representations like point clouds generalize better by preserving geometry, but they are only accurate at near range. Other modalities, such as tactile and auditory signals, offer rich world information but how to integrate them into a a unified world representation remains underexplored.

Forcing all sensing modalities into a rigid 3D representation may not be ideal. For example, long-range vision information is valuable but hard to localize in 3D, while tactile and auditory signals are transient and are not 3D. A promising approach could involve tokenizing sensor data in a unified manner, using positional embeddings to encode spatial and temporal locations. The AMR project illustrates an effective way to unify RGB-D and LiDAR data. Extending this to include additional sensing modalities and topological maps could yield a persistent, unified scene representation that better captures the complexity of real-world environments.

Better perception-action models. Most current perception-action models are trained on limited demonstrations, predominantly in tabletop manipulation settings. Consequently, their generalization capabilities are restricted to similar setups. Mobile manipulation requires designing perception-action models that allow robots to navigate, perceive, and interact with their environments seamlessly. This involves addressing several critical gaps:

1. Developing persistent, multi-modal world representations for sensing.
2. Building action models that precisely control all the degrees of freedom of a robot, with reactive, real-time feedback.
3. Incorporating active learning components that encourage robots to explore their environment, interact with objects, and improve autonomously through self-supervision.

Investing in these capabilities would enable robots to acquire knowledge dynamically and refine decisions continuously, enhancing their ability to operate robustly in diverse scenarios.

Scalable data generation. Robot data collection primarily relies on teleoperation. Unlike language data, which can be efficiently gathered from the Internet, teleoperation-based data collection has significant limitations:

1. It occurs in real time, meaning scaling requires additional hardware and human teleoperators. This is costly.

2. The data is tied to specific robot embodiments and environments, limiting its utility when deployed on a different robot and in a different environment.

3. Tasks are constrained by the capabilities of human teleoperators, but we want robots to perform tasks beyond human reach.

Simulation offers a promising solution for scalable data generation. Recent advances in simulators and large-scale assets have enabled models trained in simulation to transfer to real robots in a zero-shot manner. However, accurately modeling physics and setting up robot tasks in simulation remains labor-intensive. Improving the efficiency of these processes will be crucial to training robot policies for diverse embodiments with minimal effort.

Bibliography

[1] Multisense stereo cameras. URL: https://www.carnegierobotics.com/products/multisense-s27.

[2] Polaris RZR. URL: https://rzr.polaris.com.

[3] PyTorch Smooth-L1 Loss. URL: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html.

[4] Ridgeback mobile base. URL: https://clearpathrobotics.com/ridgeback-indoor-robot-platform/.

[5] MIT racecar, 2018. URL: https://mit-racecar.github.io/.

[6] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human trajectory prediction in crowded spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[8] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[9] Max Bajracharya, James Borders, Richard Cheng, Dan Helmick, Lukas Kaul, Dan Kruse, John Leichty, Jeremy Ma, Carolyn Matl, Frank Michel, et al. Demonstrating mobile manipulation in the wild: A metrics-driven approach. *arXiv preprint arXiv:2401.01474*, 2024.

[10] Max Bajracharya, Andrew Howard, Larry H Matthies, Benyang Tang, and Michael Turmon. Autonomous off-road navigation with end-to-end learning for the LAGR program. *Journal of Field Robotics*, 26(1):3–25, 2009.

[11] Max Bajracharya, Jeremy Ma, Matt Malchano, Alex Perkins, Alfred A Rizzi, and Larry Matthies. High fidelity day/night stereo mapping with vegetation and negative obstacle detection for vision-in-the-loop walking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3663–3670. IEEE, 2013.

[12] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi:10.15607/RSS.2019.XV.031.

[13] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. *Conference on Robot Learning (CoRL)*, 2019.

[14] Dan Barnes, William P. Maddern, and I. Posner. Find your own way: Weakly-supervised segmentation of path proposals for urban autonomy. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 203–210, 2017.

[15] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[16] Marcelo Bertalmio, Andrea L Bertozzi, and Guillermo Sapiro. Navier-Stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2001.

[17] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Rol Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[18] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Larry Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. 04 2016.

[19] Paulo V. K. Borges, Thierry Peynot, Sisi Liang, Bilal Arain, Matthew Wildie, Melih G. Minareci, Serge Lichman, Garima Samvedi, Inkyu Sa, Nicolas Hudson, Michael Milford, Peyman Moghadam, and Peter Corke. A survey on terrain traversability analysis for autonomous ground vehicles: Methods, sensors, and challenges. *Field Robotics*, 2(1):1567–1627, 2022.

[20] Xiaoyi Cai, Michael Everett, Jonathan Fink, and Jonathan P How. Risk-aware off-road navigation via a learned speed distribution map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2931–2937. IEEE, 2022.

[21] Xiaoyi Cai, Michael Everett, Lakshay Sharma, Philip R Osteen, and Jonathan P How. Probabilistic traversability model for risk-aware motion planning in off-road environments. *arXiv preprint arXiv:2210.00153*, 2022.

[22] João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[23] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan, 2021. `arXiv:2101.06806`.

[24] Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, et al. Goat: Go to any thing. *arXiv preprint arXiv:2311.06430*, 2023.

[25] Ping Chao, Chao-Yang Kao, Yushan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. Hardnet: A low memory traffic network. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3551–3560, 2019. `doi:10.1109/ICCV.2019.00365`.

[26] 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)*, 2019.

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

[28] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33:4247–4258, 2020.

[29] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[30] Kevin Chen, Juan Pablo de Vicente, Gabriel Sepulveda, Fei Xia, Alvaro Soto, Marynel Vazquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. *Robotics Science and Systems (RSS)*, 2019.

[31] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. ViT-Adapter: Exploring plain vision transformer for accurate dense predictions. In *International Conference on Learning Representations*, 2023.

[32] Ran Cheng, Christopher Agia, Yuan Ren, Xinhai Li, and Bingbing Liu. S3cnet: A sparse semantic scene completion network for lidar point clouds. In *CoRL*, 2020.

[33] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[34] Kashyap Chitta, Aditya Prakash, and Andreas Geiger. NEAT: Neural attention fields for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15793–15803, 2021.

[35] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[36] C. Choy, JunYoung Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3070–3079, 2019.

[37] John D Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International Conference on Machine Learning (ICML)*, 2018.

[38] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[39] R Craig Conlter. Implementation of the pure pursuit path'hcking algorithm. *Camegie Mellon University*, 1992.

[40] Tiago Cortinhal, G. Tzelepis, and E. Aksoy. Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds. In *ISVC*, 2020.

[41] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 2011.

[42] Nitish Dashora, Daniel Shin, Dhruv Shah, Henry Leopold, David Fan, Ali Agha-Mohammadi, Nicholas Rhinehart, and Sergey Levine. Hybrid imitative planning with geometric and predictive costs in off-road environments. In *International Conference on Robotics and Automation (ICRA)*, pages 4452–4458. IEEE, 2022.

[43] Christian F Doeller, Caswell Barry, and Neil Burgess. Evidence for grid cells in a human memory network. *Nature*, 463(7281):657, 2010.

[44] Kiana Ehsani, Tanmay Gupta, Rose Hendrix, Jordi Salvador, Luca Weihs, Kuo-Hao Zeng, Kunal Pratap Singh, Yejin Kim, Winson Han, Alvaro Herrasti, et al. Imitating shortest paths

in simulation enables effective navigation and manipulation in the real world. *arXiv preprint arXiv:2312.02976*, 2023.

[45] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22:46–57, 1989.

[46] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[47] David D Fan, Ali-Akbar Agha-Mohammadi, and Evangelos A Theodorou. Learning risk-aware costmaps for traversability in challenging environments. *IEEE Robotics and Automation Letters*, 7(1):279–286, 2021.

[48] Hehe Fan and Yi Yang. Pointrnn: Point recurrent neural network for moving point cloud processing. *arXiv*, 1910.08287, 2019.

[49] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 538–547, 2019.

[50] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[51] Bianca Forkel, Jan Kallwies, and Hans-Joachim Wuensche. Probabilistic terrain estimation for autonomous off-road driving. In *International Conference on Robotics and Automation (ICRA)*, pages 13864–13870. IEEE, 2021.

[52] Bianca Forkel and Hans-Joachim Wuensche. Dynamic resolution terrain estimation for autonomous (dirt) road driving fusing lidar and vision. In *Intelligent Vehicles Symposium (IV)*, pages 1181–1187. IEEE, 2022.

[53] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *Aaai/iaai*, 1999(343-349):2–2, 1999.

[54] Friedrich Fraundorfer, Christopher Engels, and David Nistér. Topological mapping, localization and navigation using image collections. In *International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[55] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, 2024. `arXiv:2401.02117`.

[56] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[57] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 440–448, 2020.

[58] Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. *Science Robotics*, 8(79):eadf6991, 2023.

[59] Nakul Gopalan, Eric Rosen, George Konidaris, and Stefanie Tellex. Simultaneously learning transferable symbols and language groundings from perceptual data for instruction following. *Robotics Science and Systems (RSS)*, 2020.

[60] Tianrui Guan, Zhenpeng He, Ruitao Song, Dinesh Manocha, and Liangjun Zhang. TNS: Terrain traversability mapping and navigation system for autonomous excavators. In *Robotics: Science and Systems*, 2022.

[61] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[62] Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Unifying map and landmark based representations for visual navigation. *arXiv preprint arXiv:1712.08125*, 2017.

[63] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801, 2005.

[64] Yutao Han, Jacopo Banfi, and Mark Campbell. Planning paths through unknown space by imagining what lies therein. In *CoRL*, 2020.

[65] Yutao Han, Jacopo Banfi, and Mark Campbell. Planning paths through unknown space by imagining what lies therein. In *Conference on Robot Learning*, 2021.

[66] Adam W Harley, Zhaoyuan Fang, Jie Li, Rares Ambrus, and Katerina Fragkiadaki. Simple-BEV: What really matters for multi-sensor BEV perception? In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[67] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[68] Noureldin Hendy, Cooper Sloan, Feng Tian, Pengfei Duan, Nick Charchut, Yuesong Xie, Chuang Wang, and James Philbin. FISHING Net: Future inference of semantic heatmaps in grids. *arXiv preprint arXiv:2006.09917*, 2020.

[69] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D LIDAR SLAM. In *International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[70] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, 2019.

[71] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[72] Stefan Hoermann, Martin Bach, and K. Dietmayer. Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2056–2063, 2018.

[73] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[74] Jiaheng Hu, Peter Stone, and Roberto Martín-Martín. Causal policy gradient for whole-body mobile manipulation, 2023. `arXiv:2305.04866`.

[75] Xiaoyu Huang, Dhruv Batra, Akshara Rai, and Andrew Szot. Skill transformer: A monolithic policy for mobile manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10852–10862, October 2023.

[76] Maximilian Jaritz, Raoul De Charette, Emilie Wirbel, Xavier Perrotton, and Fawzi Nashashibi. Sparse and dense data with CNNs: Depth completion and semantic segmentation. In *International Conference on 3D Vision (3DV)*, pages 52–60. IEEE, 2018.

[77] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. Rellis-3d dataset: Data, benchmarks and analysis, 2020. `arXiv:2011.12954`.

[78] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. RELLIS-3D dataset: Data, benchmarks and analysis. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[79] Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, Pete Rander, Anthony Stenz, Nick Vallidis, and Randy Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *The International Journal of Robotics Research*, 25(5-6):449–483, 2006.

[80] Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Schacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X Chang, and Manolis Savva. Habitat

synthetic scenes dataset (hssd-200): An analysis of 3d scene scale and realism tradeoffs for objectgoal navigation. *arXiv preprint arXiv:2306.11290*, 2023.

[81] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[82] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[83] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning (ICML)*, 2019.

[84] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[85] Jacob Krantz, Theophile Gervet, Karmesh Yadav, Austin Wang, Chris Paxton, Roozbeh Mottaghi, Dhruv Batra, Jitendra Malik, Stefan Lee, and Devendra Singh Chaplot. Navigating to objects specified by images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10916–10925, 2023.

[86] Jacob Krantz, Stefan Lee, Jitendra Malik, Dhruv Batra, and Devendra Singh Chaplot. Instance-specific image goal navigation: Training embodied agents to find object instances. *arXiv preprint arXiv:2211.15876*, 2022.

[87] Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning (CoRL)*, 2017.

[88] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial intelligence*, 119(1-2):191–233, 2000.

[89] Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[90] Ashish Kumar, Saurabh Gupta, and Jitendra Malik. Learning navigation subroutines from egocentric videos. In *Conference on Robot Learning (CoRL)*, 2019.

[91] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[92] Jacoby Larson and M. Trivedi. Lidar based off-road negative obstacle detection and analysis. *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 192–197, 2011.

[93] Jacoby Larson, M. Trivedi, and M. Bruch. Off-road terrain traversability analysis and hazard avoidance for ugvs. 2011.

[94] Jacoby Larson, Mohan Trivedi, and Michael Bruch. Off-road terrain traversability analysis and hazard avoidance for ugvs. Technical report, CALIFORNIA UNIV SAN DIEGO DEPT OF ELECTRICAL ENGINEERING, 2011.

[95] Dong-Hyun Lee. Pseudo-Label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 896, 2013.

[96] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[97] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. BEVFormer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. In *European Conference on Computer Vision*, 2022.

[98] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[99] Chris Linegar, Winston Churchill, and Paul Newman. Work smart, not hard: Recalling relevant experiences for vast-scale but time-constrained localisation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 90–97. IEEE, 2015.

[100] Rudolf Lioutikov, Gerhard Neumann, Guilherme Maeda, and Jan Peters. Learning movement primitive libraries through probabilistic segmentation. *The International Journal of Robotics Research (IJRR)*, 36(8):879–894, 2017.

[101] Peiqi Liu, Yaswanth Orru, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.

[102] Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 2020.

[103] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning (CoRL)*, 2019.

[104] Will Maddern, Michael Milford, and Gordon Wyeth. Capping computation time and storage requirements for appearance-based localization with cat-slam. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 822–827, 2012.

[105] Arjun Majumdar, Gunjan Aggarwal, Bhavika Devnani, Judy Hoffman, and Dhruv Batra. Zson: Zero-shot object-goal navigation using multimodal goal embeddings. *Advances in Neural Information Processing Systems*, 35:32340–32352, 2022.

[106] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[107] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *Robotics Science and Systems (RSS)*, 2020.

[108] Roberto Manduchi, Andres Castano, Ashit Talukder, and Larry Matthies. Obstacle detection and terrain classification for autonomous off-road navigation. *Autonomous Robots*, 18(1):81–102, 2005.

[109] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Proceedings of 11th International Conference on Field and Service Robotics (FSR '17)*, pages 335 – 350, September 2017.

[110] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Field and Service Robotics: Results of the 11th International Conference*, pages 335–350. Springer, 2018.

[111] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with riemannian motion policy. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[112] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Scaling local control to large-scale topological navigation. In *IEEE International Conference on Robotics and Automaton (ICRA)*, 2020.

[113] Takahiro Miki, Lorenz Wellhausen, Ruben Grandia, Fabian Jenelten, Timon Homberger, and Marco Hutter. Elevation mapping for locomotion and navigation using GPU. In *IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 2273–2280. IEEE, 2022.

[114] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 403–408, 2004.

[115] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, pages 2419–2430, 2018.

[116] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *International Conference on Learning Representations (ICLR)*, 2018.

[117] Sherif AS Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019.

[118] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. `doi: 10.1109/TRO.2015.2463671`.

[119] Michael Noseworthy, Rohan Paul, Subhro Roy, Daehyung Park, and Nicholas Roy. Task-conditioned variational autoencoders for learning movement primitives. In *Conference on Robot Learning (CoRL)*, 2019.

[120] NVIDIA. Nvidia isaac sim. `https://developer.nvidia.com/isaac-sim`, 2021.

[121] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978.

[122] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[123] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *Proceedings of the European Conference on Computer Vision*, 2020.

[124] Rui Qian, Divyansh Garg, Yan Wang, Yurong You, Serge Belongie, Bharath Hariharan, Mark Campbell, Kilian Q Weinberger, and Wei-Lun Chao. End-to-end pseudo-LiDAR for image-based 3D object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5881–5890, 2020.

[125] Ri-Zhao Qiu, Yafei Hu, Ge Yang, Yuchen Song, Yang Fu, Jianglong Ye, Jiteng Mu, Ruihan Yang, Nikolay Atanasov, Sebastian Scherer, et al. Learning generalizable feature fields for mobile manipulation. *arXiv preprint arXiv:2403.07563*, 2024.

[126] Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments. *arXiv preprint arXiv:2309.04077*, 2023.

[127] Nathan D. Ratliff, Jan Issac, and Daniel Kappler. Riemannian motion policies. *CoRR*, abs/1801.02854, 2018.

[128] Sonia Raychaudhuri, Tommaso Campari, Unnat Jain, Manolis Savva, and Angel X Chang. Mopa: Modular object navigation with pointgoal agents. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5763–5773, 2024.

[129] Cody Reading, Ali Harakeh, Julia Chae, and Steven L Waslander. Categorical depth distribution network for monocular 3D object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8555–8564, 2021.

[130] Antonio J Reina, Jorge L Martínez, Anthony Mandow, Jesús Morales, and Alfonso García-Cerezo. Collapsible cubes: Removing overhangs from 3d point clouds to build local navigable elevation maps. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1012–1017. IEEE, 2014.

[131] G. Reina, A. Milella, and R. Rouveure. Traversability analysis for off-road vehicles using stereo and radar data. *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 540–546, 2015.

[132] Thomas Roddick and Roberto Cipolla. Predicting semantic map representations from images using pyramid occupancy networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11138–11147, 2020.

[133] Junha Roh, Chris Paxton, Andrzej Pronobis, Ali Farhadi, and Dieter Fox. Conditional driving from natural language instructions. In *Conference on Robot Learning (CoRL)*, 2019.

[134] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[135] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[136] A. Sadat, S. Casas, Mengye Ren, X. Wu, Pranaab Dhawan, and R. Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *ECCV*, 2020.

[137] Abbas Sadat, Sergio Casas, Mengye Ren, Xinyu Wu, Pranaab Dhawan, and Raquel Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *European Conference on Computer Vision*, 2020.

[138] Avishkar Saha, Oscar Mendez, Chris Russell, and Richard Bowden. Translating images into maps. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9200–9206. IEEE, 2022.

[139] Vojtěch Šalanskỳ, Karel Zimmermann, Tomáš Petříček, and Tomáš Svoboda. Pose consistency KKT-loss for weakly supervised learning of robot-terrain interaction model. *IEEE Robotics and Automation Letters*, 6(3):5477–5484, 2021.

[140] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations (ICLR)*, 2018.

[141] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *International Conference on Learning Representations (ICLR)*, 2019.

[142] Fabian Schilling, Xi Chen, John Folkesson, and Patric Jensfelt. Geometric and visual terrain classification for autonomous mobile navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[143] Christos Sevastopoulos and Stasinos Konstantopoulos. A survey of traversability estimation for mobile robots. *IEEE Access*, 10:96331–96347, 2022.

[144] Amirreza Shaban, Xiangyun Meng, JoonHo Lee, Byron Boots, and Dieter Fox. Semantic terrain classification for off-road autonomous driving. In *Conference on Robot Learning*, pages 619–629. PMLR, 2022.

[145] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[146] Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *Conference on Robot Learning*, pages 2683–2699. PMLR, 2023.

[147] Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. Gnm: A general navigation model to drive any robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7226–7233. IEEE, 2023.

[148] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A foundation model for visual navigation. In *7th Annual Conference on Robot Learning*, 2023. URL: `https://arxiv.org/abs/2306.14846`.

[149] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations (ICLR)*, 2019.

[150] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*, 2019.

[151] Xingjian Shi, Zhourong Chen, Hao Wang, D. Yeung, W. Wong, and W. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015.

[152] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. Nomad: Goal masked diffusion policies for navigation and exploration. *arXiv preprint arXiv:2310.07896*, 2023.

[153] David Stavens and Sebastian Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. UAI'06, page 469–476. AUAI Press, 2006.

[154] Maximilian Stölzle, Takahiro Miki, Levin Gerdes, Martin Azkarate, and Marco Hutter. Reconstructing occluded elevation information in terrain maps with self-supervised learning. *IEEE Robotics and Automation Letters*, 7(2):1697–1704, 2022.

[155] Marlin P Strub and Jonathan D Gammell. Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3191–3198. IEEE, 2020.

[156] Benjamin Suger, Bastian Steder, and W. Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3941–3946, 2015.

[157] Tarlan Suleymanov, Lina Maria Paz, Pedro Piniés, Geoff Hester, and Paul Newman. The path less taken: A fast variational approach for scene segmentation used for closed loop control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3620–3626. IEEE, 2016.

[158] Yizhen Sun, Junyou Yang, Zihan Zhang, and Yu Shu. An optimization-based high-precision flexible online trajectory planner for forklifts. In *Actuators*, volume 12, page 162. MDPI, 2023.

[159] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[160] Tristan Swedish and Ramesh Raskar. Deep visual teach and repeat on path networks. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[161] Tua Agustinus Tamba, Bonghee Hong, and Keum-Shik Hong. A path following control of an unmanned autonomous forklift. *International Journal of Control, Automation and Systems*, 7(1):113–122, 2009.

[162] R. Tedrake, I. Manchester, M. Tobenkin, and J. Roberts. Lqr-trees- feedback motion planning via sums of squares optimization. *Journal of Robotics Research (IJRR)*, 29:1038–1052, 2010.

[163] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[164] C. Thorpe, M.H. Hebert, T. Kanade, and S.A. Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988. doi:10.1109/34.3900.

[165] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, September 2005. ISBN 0-262-20162-3.

[166] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[167] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2276–2282. IEEE, 2006.

[168] N Trigoni, A Markham, and L Xie. SnapNav: learning mapless visual navigation with sparse directional guidance and visual reference. In *IEEE International Conference on Robotics and Automaton (ICRA)*, 2020.

[169] Shagun Uppal, Ananye Agarwal, Haoyu Xiong, Kenny Shaw, and Deepak Pathak. Spin: Simultaneous perception, interaction and navigation. *CVPR*, 2024.

[170] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon,

U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[171] A. Wahid, A. Toshev, M. Fiser, and T. E. Lee. Long range neural navigation policies for the real world. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[172] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. *arXiv preprint arXiv:2312.08344*, 2023.

[173] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[174] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[175] Pengxiang Wu, Siheng Chen, and Dimitris N. Metaxas. Motionnet: Joint perception and motion prediction for autonomous driving based on bird's eye view maps. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11382–11392, 2020.

[176] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation. In *International Conference on Computer Vision (ICCV)*, 2019.

[177] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 2020.

[178] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9068–9079, 2018.

[179] Yu Xiang and Dieter Fox. DA-RNN: semantic mapping with data associated recurrent neural networks. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017. doi:10.15607/RSS.2017.XIII.013.

[180] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. SegFormer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems 34*, pages 12077–12090, 2021.

[181] Karmesh Yadav, Jacob Krantz, Ram Ramrakhya, Santhosh Kumar Ramakrishnan, Jimmy Yang, Austin Wang, John Turner, Aaron Gokaslan, Vincent-Pierre Berges, Roozbeh Mootaghi, Oleksandr Maksymets, Angel X Chang, Manolis Savva, Alexander Clegg, Devendra Singh Chaplot, and Dhruv Batra. Habitat challenge 2023. `https://aihabitat.org/challenge/2023/`, 2023.

[182] Xu Yan, Jiantao Gao, Jie Li, Ruimao Zhang, Zhen Li, Rui Huang, and Shuguang Cui. Sparse single sweep lidar point cloud segmentation via learning contextual shape priors from scene completion. In *AAAI*, 2021.

[183] Weixiang Yang, Qi Li, Wenxi Liu, Yuanlong Yu, Yuexin Ma, Shengfeng He, and Jia Pan. Projecting your view attentively: Monocular road scene layout estimation via cross-view transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[184] Naoki Yokoyama, Alex Clegg, Joanne Truong, Eric Undersander, Tsung-Yen Yang, Sergio Arnaud, Sehoon Ha, Dhruv Batra, and Akshara Rai. Asc: Adaptive skill coordination for robotic mobile manipulation. *IEEE Robotics and Automation Letters*, 9(1):779–786, 2023.

[185] Wenyuan Zeng, W. Luo, Simon Suo, A. Sadat, Bin Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8652–8661, 2019.

[186] Zhen Zhang, Anran Lin, Chun Wai Wong, Xiangyu Chu, Qi Dou, and KW Au. Interactive navigation in environments with traversable obstacles using large language and vision-language models. *arXiv preprint arXiv:2310.08873*, 2023.

[187] Brady Zhou and Philipp Krähenbühl. Cross-view transformers for real-time map-view semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13760–13769, 2022.

[188] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. *arXiv preprint arXiv:2011.10033*, 2020.

Appendices

Appendix A

LiDAR-based Semantic Terrain Classification

A.1 Network Architecture

We provide a detailed description of our network here, illustrated in Figure A.1. The 3D projection is composed of a set of 3D convolution blocks, where the 3D SubM Conv block is a sequence of Submanifold 3D convolution, Bachnorm, and ReLU activation. The 3D Conv block maintains the XY resolution but compresses in the z dimension, and is made up of 3D convolution, Bathnorm, and ReLU activation. The Temporal Feature Aggregation module has a single ConvGRU layer for efficiency. The BEV Inpainting network is a modified variant of FC-HardNet with skip connections based on Harmonic DenseNet [25]. Each encoder block is a Harmonic Dense (HarD) block with Average Pooling, with no pooling for the middle layer. Each decoder block upsamples the features with Transposed convolution with bilinear upsampling, and features from the corresponding encoder block is concatenated. Each decoder block has a HarD block following the skip connection. The final BEV map is predicted with a Fully convolutional layer as the classification layer.

Network training. We train our network using the Adam optimizer [81] with an initial learning rate of $3e-4$ and a decay of 0.7 per epoch. We use a weighted Cross-Entropy loss. We start with training a single-frame model without ConvGRU until the model converges. Then we freeze the sparse convolution layers and insert the ConvGRU layer, and then train the ConvGRU and the inpainting network together. While technically we can train the whole network end-to-end , this two-stage training procedure is faster and is more memory-efficient. When training the ConvGRU, we use a sequence length of 5 with a frame stride randomly chosen from [1, 10, 20]. Training takes about 12 hours on a single RTX 3090. The inference time of our network is 6 fps on a RTX 3090.

Data augmentation. During training, we randomly rotate every pair of LiDAR scans and the ground truth traversability map in $\mathcal{U}[-45°, 45°]$, and randomly drop 20% of the points.

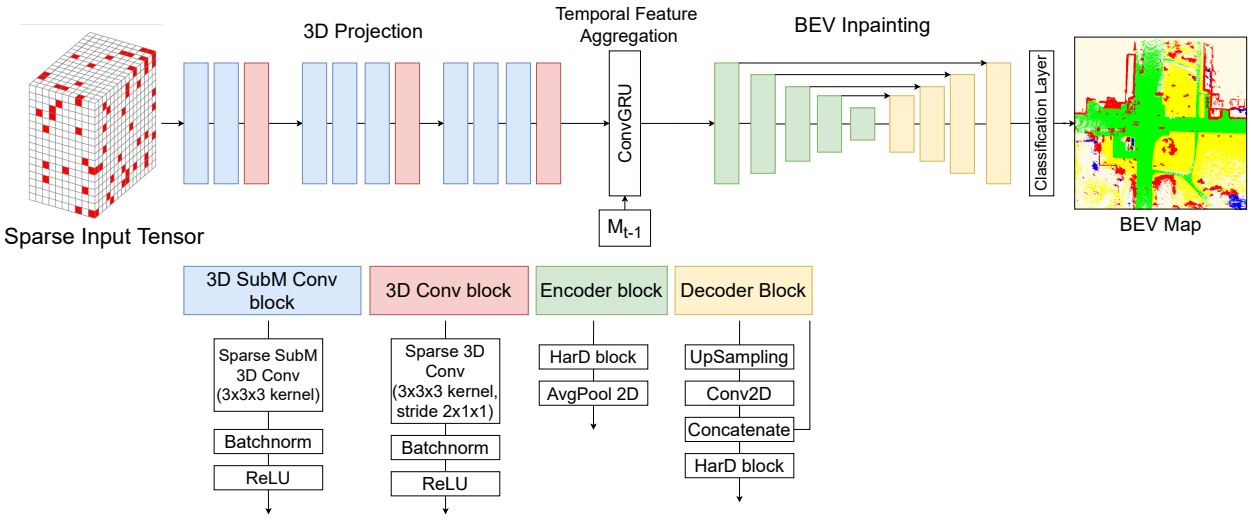


Figure A.1: Detailed illustration of our BEVNet-R architecture. BEVNet-S has the same architure with ConvGRU excluded.

Furthermore, we perturb the groundtruth odometry with rotation drawn from $\mathcal{N}(0, 0.01^2)$ and translation drawn from $\mathcal{N}(0, 0.1^2)$. Note that the error in odometry will accumulate over time.

A.2 Dataset

A.2.1 Traversability Mapping

Table A.1 shows how we map the raw class labels to the 4-level traversabilities.

A.2.2 Class distribution

Figure A.2 shows the class distribution of SemanticKITTI and RELLIS-3D using our BEV labels based on the 4-level traversability ontology.

A.2.3 Train/Validation Split

SemanticKITTI We choose sequence 08 as our validation sequence, which is typically done by other works including but not limited to which [188] we compare to.

RELLIS-3D We choose sequence 04 as our validation sequence. While the authors of the dataset [77] provide a split, it follows random selection of samples and therefore does not fit our training of recurrent model. Instead, we select a whole sequence as a holdout dataset.

Dataset	free	low-cost	medium-cost	lethal
SemanticKITTI	road parking sidewalk other-ground	terrain	vegetation	car bicycle bus motorcycle on-rails truck other-vehicle person bicyclist motorcyclist building fence trunk pole traffic-sign moving-car moving-bicyclist moving-person moving-motorcyclist moving-on-rails moving-bus moving-truck moving-other-vehicle
RELLIS-3D	dirt asphalt concrete	grass puddle mud rubble	bush	tree pole vehicle object building log person fence barrier

Table A.1: Mapping of the original class labels to the 4-level traversabilities for the SemanticKITTI and RELLIS-3D datasets.

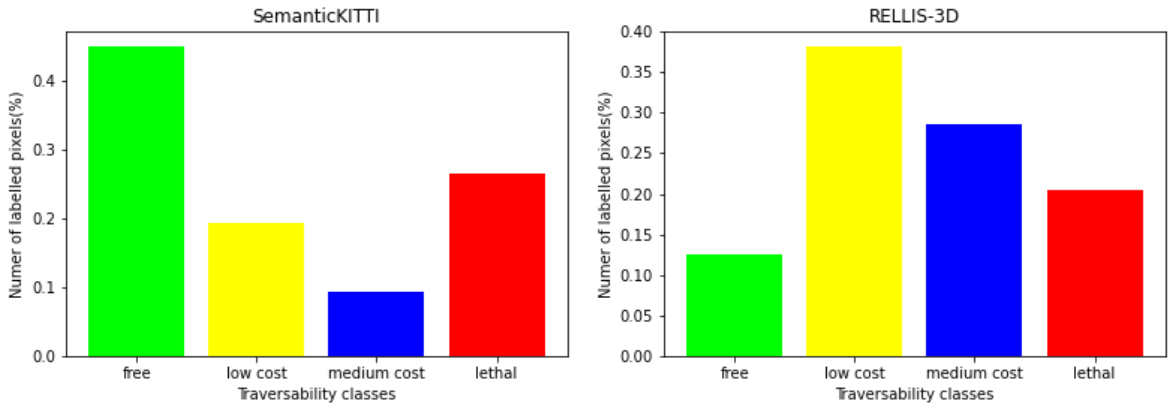


Figure A.2: Individual class distribution across our 4-class ontology on SemanticKITTI (left) and RELLIS-3D (right). Distribution is recorded from the total dataset as the proportion of total pixels each class has in the generated BEV labels.

A.3 More Quantitative Results

We present additional quantitative results here including classwise Intersection over Unions. Moreover, we add an additional model "BEVNet-R (C)" which is trained on clean odometry. The model in the main paper "BEVNet-R" is renamed to "BEVNet-R (N)" (trained on noisy odometry). Specifically:

- Table A.2,A.3,A.4 show the results on the SemanticKITTI dataset for the "all", "seen", and "unseen" modes, respectively.
- Table A.5,A.6,A.7 show the results on the RELLIS-3D dataset for the "all", "seen", and "unseen" modes, respectively.

A.4 Additional Comparison with Various Baselines

Several recent papers have focused on semantic understanding of scenes from sparse LiDAR scans, which we discuss below. These papers solve related, but slightly different problems. They produce smaller maps, and do not perform temporal aggregation.

- **Semantic Scene Completion.** The scene completion task aims to predict a dense semantic voxel grid from a single LiDAR scan. To compare to our work, we build a traversability map by converting predicted dense voxel grids to 2D traversability maps using the same method described in Sec 3.2.

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.666	0.484	0.113	0.456	0.430
Clean Odometry					
BEVNet-S + TA	0.672	0.516	**0.271**	0.500	0.490
BEVNet-R (N)	0.719	0.551	0.171	0.480	0.480
BEVNet-R (C)	**0.768**	**0.620**	0.215	0.538	**0.535**
Cylinder3D + TA	0.612	0.510	0.264	0.476	0.465
Noisy Odometry					
BEVNet-S + TA	0.578	0.449	**0.232**	0.328	0.397
BEVNet-R (N)	**0.705**	**0.532**	0.171	**0.464**	**0.468**
BEVNet-R (C)	0.669	0.523	0.192	0.451	0.459
Cylinder3D + TA	0.466	0.390	0.193	0.318	0.342

Table A.2: SemanticKITTI. Including both the seen and unseen area.

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.713	0.529	0.148	0.538	0.482
Clean Odometry					
BEVNet-S + TA	0.723	0.571	0.363	0.590	0.562
BEVNet-R (N)	0.776	0.613	0.227	0.572	0.547
BEVNet-R (C)	0.844	0.703	0.298	0.654	0.625
Cylinder3D + TA	**0.864**	**0.745**	**0.378**	0.635	0.655
Noisy Odometry					
BEVNet-S + TA	0.600	0.480	**0.308**	0.362	0.438
BEVNet-R (N)	**0.756**	**0.588**	0.224	**0.549**	**0.529**
BEVNet-R (C)	0.723	0.575	0.247	0.532	0.519
Cylinder3D + TA	0.617	0.532	0.272	0.399	0.455

Table A.3: SemanticKITTI. Only the seen area.

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.560	0.398	0.069	0.247	0.319
Clean Odometry					
BEVNet-S + TA	0.558	0.409	**0.147**	**0.278**	0.347
BEVNet-R (N)	0.597	0.430	0.094	0.255	0.344
BEVNet-R (C)	**0.608**	**0.461**	0.096	0.249	**0.354**
Cylinder3D + TA	-	-	-	-	-
Noisy Odometry					
BEVNet-S + TA	0.522	0.388	**0.129**	0.249	0.322
BEVNet-R (N)	**0.593**	**0.424**	0.098	**0.254**	**0.343**
BEVNet-R (C)	0.553	0.420	0.113	0.244	0.332
Cylinder3D + TA	-	-	-	-	-

Table A.4: SemanticKITTI. Only the unseen area.

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.493	0.550	0.597	0.596	0.559
Clean Odometry					
BEVNet-S + TA	**0.737**	**0.613**	0.544	0.565	0.615
BEVNet-R (N)	0.557	0.602	0.632	**0.682**	0.618
BEVNet-R (C)	0.675	0.593	**0.633**	0.676	**0.644**
Cylinder3D + TA	0.250	0.413	0.398	0.584	0.411
Noisy Odometry					
BEVNet-S + TA	**0.621**	0.522	0.293	0.370	0.452
BEVNet-R (N)	0.553	**0.598**	**0.628**	**0.675**	**0.614**
BEVNet-R (C)	0.266	0.361	0.473	0.450	0.387
Cylinder3D + TA	0.229	0.348	0.289	0.407	0.318

Table A.5: RELLIS-3D. Including both the seen and unseen area.

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.230	0.578	0.617	0.649	0.518
Clean Odometry					
BEVNet-S + TA	**0.611**	0.638	0.560	0.609	0.605
BEVNet-R (N)	0.323	0.622	0.650	0.711	0.577
BEVNet-R (C)	0.489	0.621	0.655	0.716	0.621
Cylinder3D + TA	0.478	0.660	0.509	**0.727**	0.568
Noisy Odometry					
BEVNet-S + TA	0.398	0.514	0.225	0.350	0.372
BEVNet-R (N)	0.322	**0.617**	**0.646**	**0.705**	**0.572**
BEVNet-R (C)	0.07	0.327	0.462	0.457	0.329
Cylinder3D + TA	**0.460**	0.438	0.357	0.485	0.435

Table A.6: RELLIS-3D. Only the seen area

	free	low-cost	medium-cost	lethal	mIoU
BEVNet-S	0.713	0.488	0.546	0.433	0.545
Clean Odometry					
BEVNet-S + TA	**0.843**	0.555	0.502	0.446	0.586
BEVNet-R (N)	0.754	**0.559**	**0.583**	**0.584**	0.620
BEVNet-R (C)	0.830	0.530	0.574	0.547	**0.621**
Cylinder3D + TA	-	-	-	-	-
Noisy Odometry					
BEVNet-S + TA	**0.811**	0.546	0.460	0.421	0.560
BEVNet-R (N)	0.748	**0.557**	**0.579**	**0.579**	**0.616**
BEVNet-R (C)	0.435	0.432	0.509	0.420	0.449
Cylinder3D + TA	-	-	-	-	-

Table A.7: RELLIS-3D. Only the unseen area.

- **Inpainting Network.** [64] recently proposed an approach that uses GANs to inpaint a sparsely segmented BEV image. We trained our model using the same SemanticKITTI dataset with the groundtruth 19-class BEV images as supervision. Note that this task is different from traversability estimation because it does a simple topdown projection.

Implementation Details. For a fair comparison, we re-train our models using the settings where the baselines were trained on: the map size is $51.2m \times 51.2m$, and the vehicle is located at the center left. To have a fair comparison, for each frame we aggregate Cylinder 3D predictions over the past 70 frames using the calibrated poses provided by SemanticKITTI and do a top-down projection by picking the highest z point at each location. JS3C-Net is a 3D scene completion algorithm. We use the model and the code provided by the authors. We take the 3D voxel prediction of JS3C-Net and project it to 2D using top-down projection. The result of Han et al. [7] is reported from the original paper since the output is already in 2D.

Results: The results are shown in Table A.8. When testing on full categories and using top-down projection, JS3C-Net is slightly better than our method, specifically for classes within the lethal obstacles group. Note that in our approach, the projection is learned from the data and not hand-engineered as in top-down projection. We hypothesize for simple top-down projection, it is more data-efficient to code the projection method instead of learning it from the data. However, in complex projection, our learning-based approach in the 4 category ontology significantly outperforms JS3C-Net.

Speedwise, JS3C-Net is significantly slower than ours since it predicts a dense voxel grid. On a 51.2m x 51.2m map, JS3C-Net runs less than 2 fps whereas BEVNet-S runs at 12 fps on a 1080 Ti. It will be hard to scale JS3C-Net to larger maps with recurrency (note that in our main results we use 102.4m x 102.4m maps). In comparison, BEVNet maintains a latent 2D feature map for cost reasoning, which is more memory-efficient.

Cylinder 3D's predictions are limited to the seen area of the map. It is surprising that our method performs as well as the Cylinder 3D on the seen area while being able to inpaint the entire map. Finally, we emphasize that our network is designed to perform multiple tasks simultaneously (semantic segmentation, inpainting, complex projection, time aggregation). It is not surprising that its performance does not exceed Cylinder-3D or JS3C-Net which perform a subset of these tasks using a network with a similar capacity.

A.5 Ablation on Odometry Noise

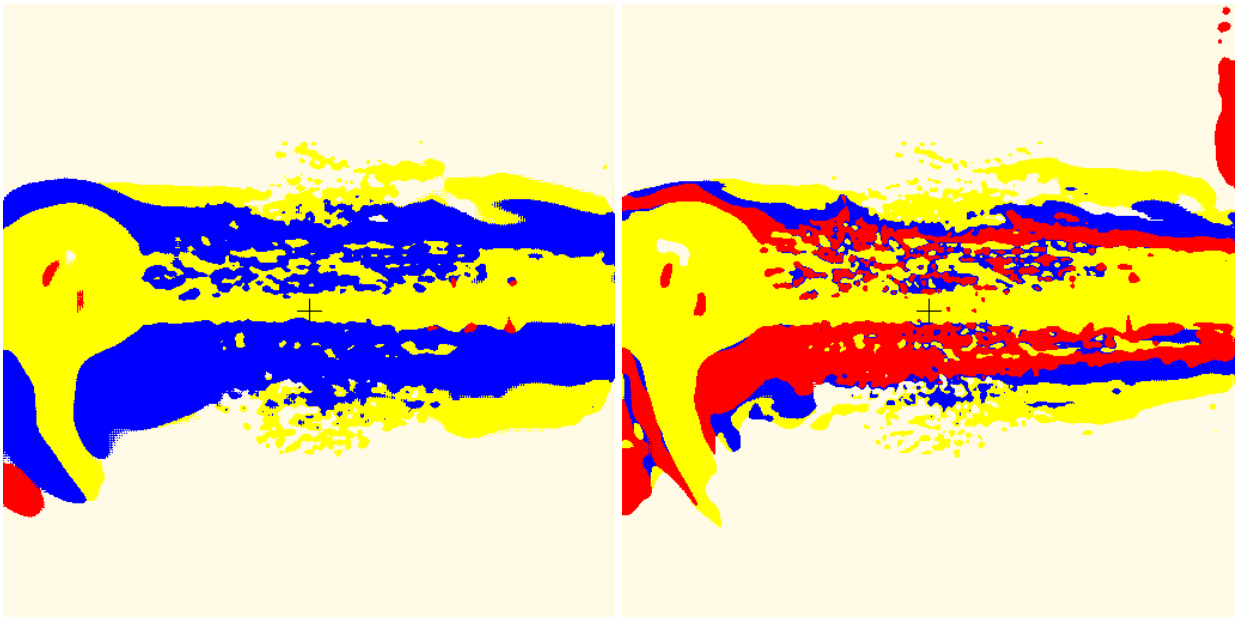
We vary the odometry noise level by introducing a scalar λ such that the rotation noise is drawn from $\mathcal{N}(0, (0.01\lambda)^2)$ and translation noise drawn from $\mathcal{N}(0, (0.1\lambda)^2)$. The results are presented in Table A.9. BEVNet-R degrades gracefully as noise level increases. Please check out the project website for qualitative videos on this.

Method (#classes)	Segmentation	Inpainting	Projection	Temporal Aggregation	seen	unseen
Cylinder3D-TA (19)	Yes	No	Top Down	Yes	**0.316**	0.181
Han et al. (19)	No	Yes	Top Down	No	-	0.131
JS3C-Net (19)	Yes	Yes	Top Down	No	-	**0.258**
BEVNet-S (19)	Yes	Yes	Top Down	No	0.313	0.253
JS3C-Net (4)	Yes	Yes	Complex	No	-	0.549
BEVNet-R (4)	Yes	Yes	Complex	Yes	-	**0.608**

Table A.8: Comparing with various semantic segmentation and completion baselines on the SemanticKITTI dataset. Note that the map here is smaller than our main results to accommodate the baselines. We evaluate on two kinds of projection: top-down projection (only looking at the highest point at each location), and complex projection (taking both the traversability of each class and their heights into account).

	λ				
	0%	50%	100%	200%	500%
SemanticKITTI	0.480	0.474	0.468	0.458	0.431
RELLIS-3D	0.618	0.616	0.614	0.611	0.600

Table A.9: Effect of odometry noise level on the mIoU. Note that BEVNet-R is trained at 100% noise level.



(a) Predicted traversability map for a large robot. (b) Predicted traversability map for a small robot.

Figure A.3: BEVNet can reason both the semantic and geometric properties of the terrain.

A.6 Robot-dependent Traversability

We show that BEVNet can predict traversability by reasoning both semantic and geometric properties of terrains. We generate two groundtruth BEV costmap datasets from the RELLIS dataset for this experiment.

- The first dataset maps *bush* to the medium-cost class regardless of the height of the bush. This applies to a large offroad vehicle like the ClearPath Warthog.
- The second dataset maps *bush* to the lethal class if the bush is 0.5 m above the ground. This applies to a small offroad vehicle like the ClearPath Husky.

We train BEVNet on each dataset separately. Figure A.3 shows the prediction results on the RELLIS-3D validation set. BEVNet is able to predict robot-dependent traversability maps. In Figure A.3b, some bushes are labeled as lethal since they are too high and are thus dangerous for a small offroad vehicle.

A.7 Variations of Cylinder3D with temporal aggregation

For a more comprehensive comparison, we additionally compare our method against a more traditional 3D SLAM-fusion based fusion implementation (denoted as Cylinder3D-TA-3D). The 3D baseline performs the following:

Method	Full	Mask used for the seen area		
		Cy3d+TA	TA-3D w/o ray tracing	TA-3D w/ ray tracing
Cy3D + TA	0.465	**0.655**	0.613	0.618
Cy3D + TA-3D w/o ray tracing	0.482	0.636	**0.660**	0.660
Cy3D + TA-3D w/ ray tracing	0.471	0.629	0.646	**0.677**
BEVNet-R	**0.535**	0.625	0.613	0.615

Table A.10: Comparison of different baseline variants across different masks on the SemanticKITTI dataset. Full refers to the whole map (seen + unseen).

1. We first perform semantic segmentation on the LiDAR scans with Cylinder3D to generate segmented point clouds.
2. We aggregate the segmented point clouds in 3D using a voxel grid similar to how Octomap [73] works using the poses that are calibrated by a standard SLAM algorithm.
3. When aggregating the point clouds, we optionally apply ray tracing to remove the trace of moving objects as much as possible.
4. For the points falling into a voxel, we normalize the counts of their predicted labels into a categorical distribution. This is how Bayesian statistics estimates the parameter of a categorical distribution via a uniform Dirichlet prior. This produces a labeled voxel grid.
5. We project this labeled voxel grid down to 2D using the same rule as how we generate the groundtruth BEV costmaps.

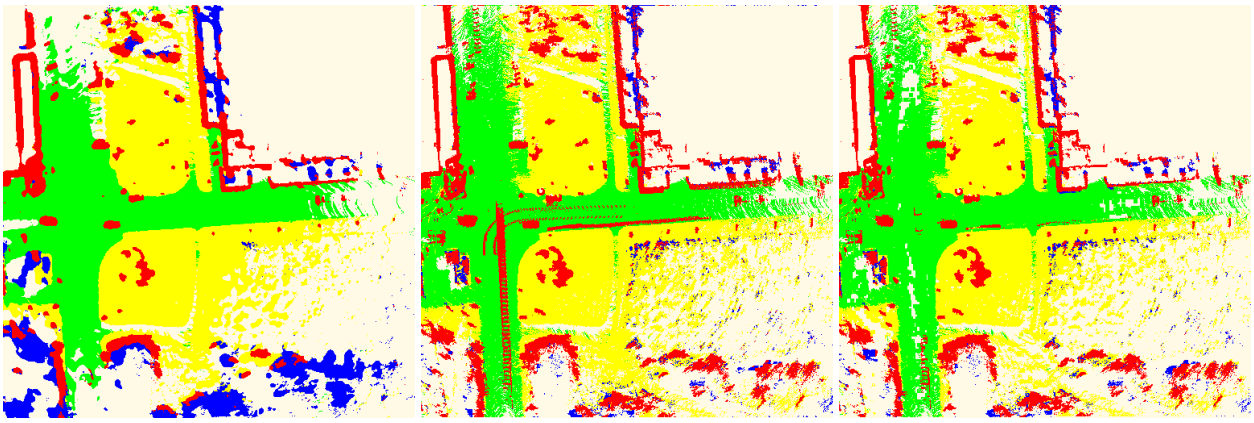
A.7.1 Nuances on comparing the methods on the seen areas.

Depending on how aggregation is performed, the seen areas vary slightly between Cylinder3D-TA, Cylinder3D-TA-3D (w/o ray tracing) and Cylinder3D-TA-3D (with ray tracing) (Figure A.4). In particular, when turning on ray tracing for Cylinder3D-TA-3D, some of the moving objects will get cleared, but some ground voxels may also get cleared due to discretization errors (this happens when a LiDAR ray hits the ground with a shallow incident angle). To make a fair comparison, in Table A.10 and A.11 we report mIoUs of the methods on the three seen areas generated by Cylinder3D-TA, Cylinder3D-TA-3D (w/o ray tracing) and Cylinder3D-TA-3D (with ray tracing).

A.7.2 Analysis

From the comparisons we observe the following:

- Compared to Cylinder3D+TA baseline that aggregates in the 2D BEV space, 3D aggregation with raytracing exhibits considerable improvement in the seen area: +8 points in RELLIS



(a) (b) (c)

Figure A.4: Comparing the seen areas of (a) Cylinder3D-TA, (b) Cylinder3D-TA-3D w/o raytracing and (c) Cylinder3D-TA-3D with raytracing. The seen areas are slightly different depending on how aggregation is performed. For a fair comparison, we evaluate the performance of the baselines on the three different kinds of seen areas.

		Mask used for the seen area		
Method	Full	Cy3d+TA	TA-3D w/o ray tracing	TA-3D w/ ray tracing
Cy3D + TA	0.411	0.568	0.573	0.567
Cy3D + TA-3D w/o ray tracing	0.408	0.576	**0.649**	0.643
Cy3D + TA-3D w/ ray tracing	0.384	0.539	0.609	**0.645**
BEVNet-R	**0.644**	**0.621**	0.618	0.615

Table A.11: Comparison of different baseline variants across different masks on the RELLIS-3D dataset. Full refers to the whole map (seen + unseen).

(+5.9 points in SemanticKITTI) (see the last column of Table A.10 and A.11), respectively. However, while 2D aggregation time in TA is negligible, 3D aggregation is the computational bottleneck in the pipeline as it takes approx. 1 second to aggregate a single scan into octomap and perform raytracing.

- When evaluating on the whole map, BEVNet outperforms Cylinder3D-TA-3D by more than 20 points in RELLIS (5 points in SemanticKITTI) because Cylinder3D-TA-3D does not predict the future(see the first column of Table A.10 and A.11).
- When limiting the comparison to the seen area, Cylinder3D-TA-3D outperforms BEVNet-R up to 3 points in RELLIS (6 points in SemanticKITTI) (see the last column of Table A.10 and A.11).

In conclusion, there is a trade-off between prediction area and accuracy. While combining state-of-the-art semantic segmentation with SLAM-based fusion produces better results on the seen area, it falls short of predicting a more complete map, is prone to odometry noise, and is computationally expensive. Our network is optimized to perform multiple tasks simultaneously (semantic segmentation, inpainting, complex projection, time aggregation) efficiently in a single forward pass.

Appendix B

Visual Terrain Modeling for High-speed, Offroad Navigation

B.1 Computing Terrain Representation

This section provides more details about how we build the terrain representation.

Point cloud aggregation. Given the raw LiDAR packets and the vehicle's ego motion, we convert the LiDAR packets into LiDAR points with the rolling shutter effects corrected. Then, we run Google Cartographer [69] on the LiDAR points (with IMU information) to compute the gravity-aligned pose for each LiDAR scan. We aggregate the LiDAR scans using their poses to get an aggregated point cloud.

Outlier removal. LiDAR may produce false returns due to the presence of dust, snowflakes, and water. These false returns manifest as noise in the aggregated point cloud. We identify these outliers in two ways:

1. Voxel filter. We voxelize the point cloud with a voxel size of 0.3m. We count the number of points in each voxel. If the number of points is smaller than a threshold (set to 5), we remove all the points in this voxel.

2. Semantic filter. When we label the LiDAR points, we ask the labelers to label the outliers as an additional *outlier* class. Our LiDAR segmentation network is trained with the additional outlier class. After we predict the pseudo labels for each point, we remove points classified as outliers.

B.2 Network Architecture

Backbone. Our image backbone follows that of Lift-Splat-Shoot[123]. There are two main differences:

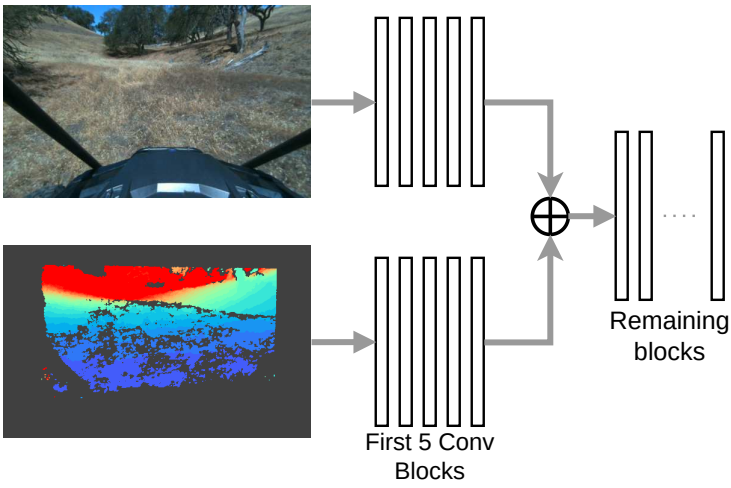


Figure B.1: The image backbone uses separate branches to process the RGB and depth inputs, respectively. Then, the outputs of the two branches are summed and passed to the remaining blocks in the backbone.

1. For RGB-D inputs, we duplicate the first five convolutional blocks to process the RGB and depth images separately (Figure B.1). This has a noticeable improvement over passing a 4-channel RGB-D image directly to the backbone.

2. We do 8× downsampling of the feature map and depth map (the original paper performs 16× downsampling). This provides a good trade-off between speed and accuracy. Note that all the baselines are trained with the same image backbone with the same 8× downsampling ratio.

Our input image size is 512×320. Hence, the image backbone produces a 472-channel feature map with a spatial dimension of 64×40.

Depth completion network. The output of the image backbone is passed to a single convolution layer with 472 input channels and D output channels with kernel size 3. D is the number of desirable depth bins. For 50 m × 50 m maps, $D = 128$. For 100 m × 100 m maps, $D = 256$.

Terrain embedding network. For each back-projected point (x, y, z), we first apply an MLP with layer sizes $(1, 64, 32)$ to z to get the 32-dimensional elevation embedding f_{elev}. Then, we concatenate f_{elev} with the 472-channel image feature f_{sem} and apply another MLP with layer sizes $(504, 96)$ to get the 96-dimensional terrain embedding f. ReLU is applied after each MLP layer.

Temporal aggregation layer. The temporal aggregation layer is a single ConvGRU layer with 96 input channels, 96 output channels, and a kernel size of 1. To train TerrainNet-TA, we first train the non-recurrent version, and then insert the recurrent layer. We freeze the weights of the model except the recurrent layer.

Inpainting network. The inpainting net follows the architecture of Lift-Splat-Shoot's BEV encoder network. The main difference is that we create a separate decoding head consisting of an

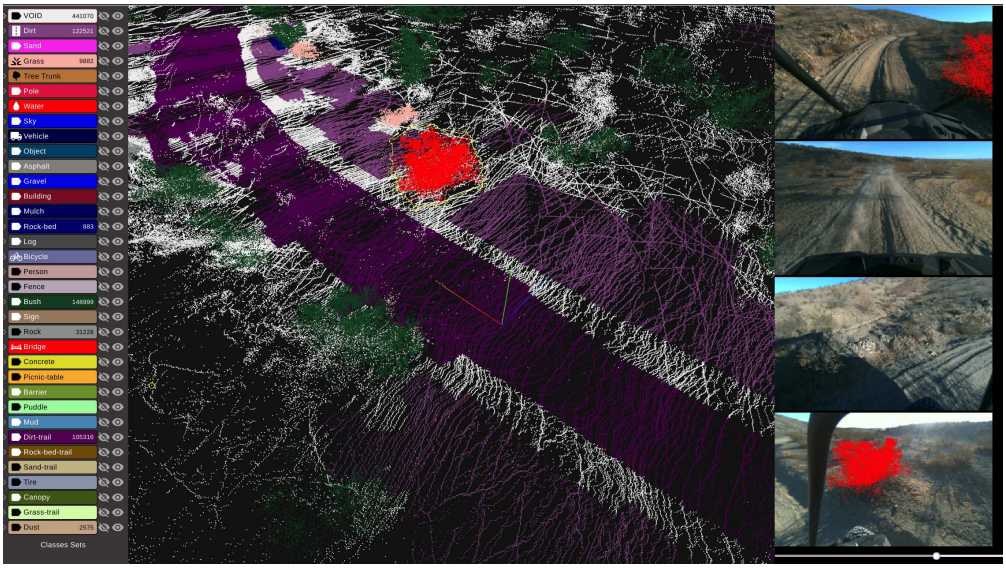


Figure B.2: **Annotation tool.** Labelers can select points in the point cloud and see their corresponding image projections. Here we show a bush is highlighted both in the point cloud and in the bottom image. This helps the labelers to verify the semantic class of those points. Moreover, labelers only annotate points that they are confident about. We leave points that are difficult to identify unlabeled.

upsampling layer and two convolution layers to predict a specific terrain layer.

B.3 Data Annotation

We create our data annotation tool (Figure B.2) to obtain ground truth semantically labeled point clouds. We aggregate 30 to 50 LiDAR scans for each selected frame to get a dense point cloud for labeling. We provide camera images for every LiDAR scan for reference so that a labeler can cross-reference to verify that the labels are correct.

B.4 Additional dataset examples

Figure B.3 shows additional dataset examples to highlight the diversity of the datasets. We collected our datasets from 3 distinctive geographic locations across the year.

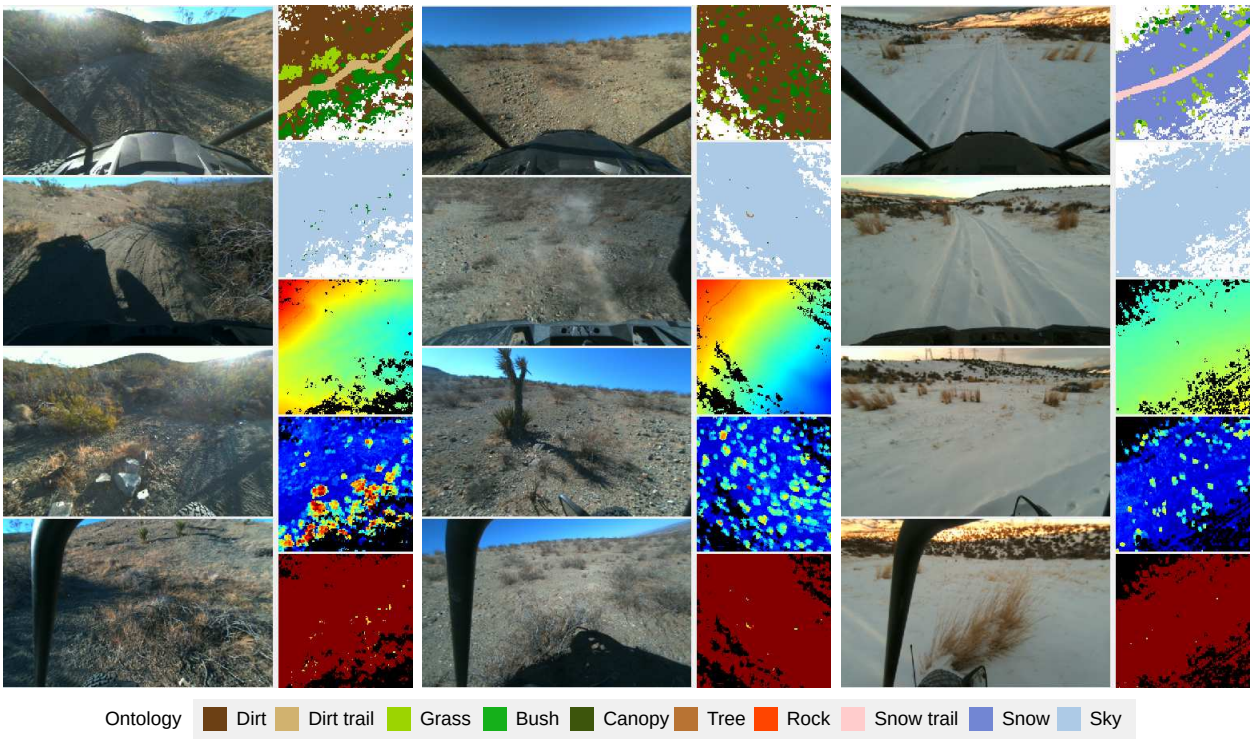


Figure B.3: Additional dataset examples. These examples were collected at different locations and in different seasons. For the real-world experiments, we finetuned the model with annotated snow data (rightmost column).

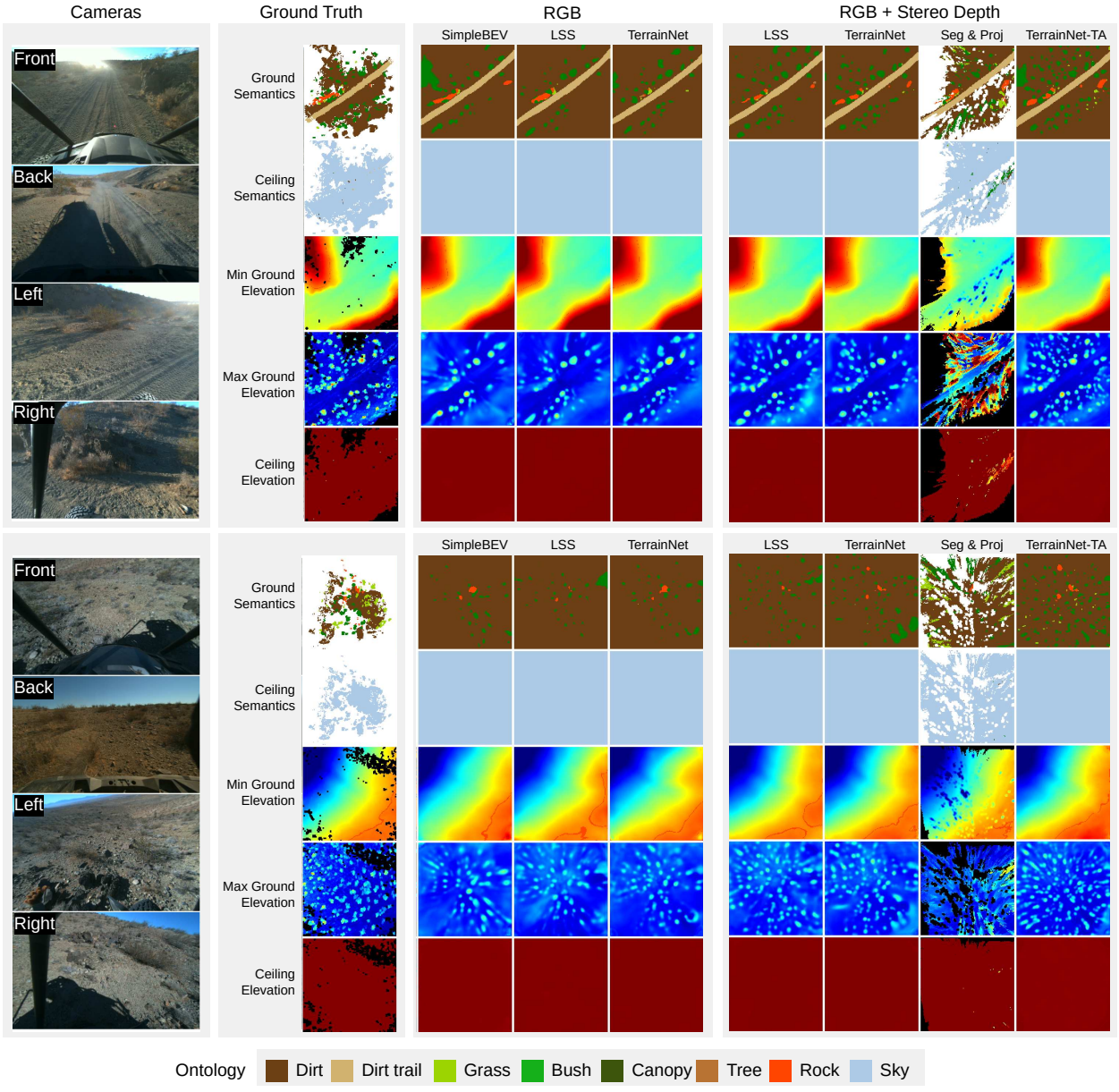


Figure B.4: Comparing the predictions of different models. **Top:** on-trail run with rocks and bushes on the two sides of the trail. **Bottom:** off-trail run with scattered rocks and bushes. These are open terrains, so the ceiling semantics consists almost entirely of *sky*.

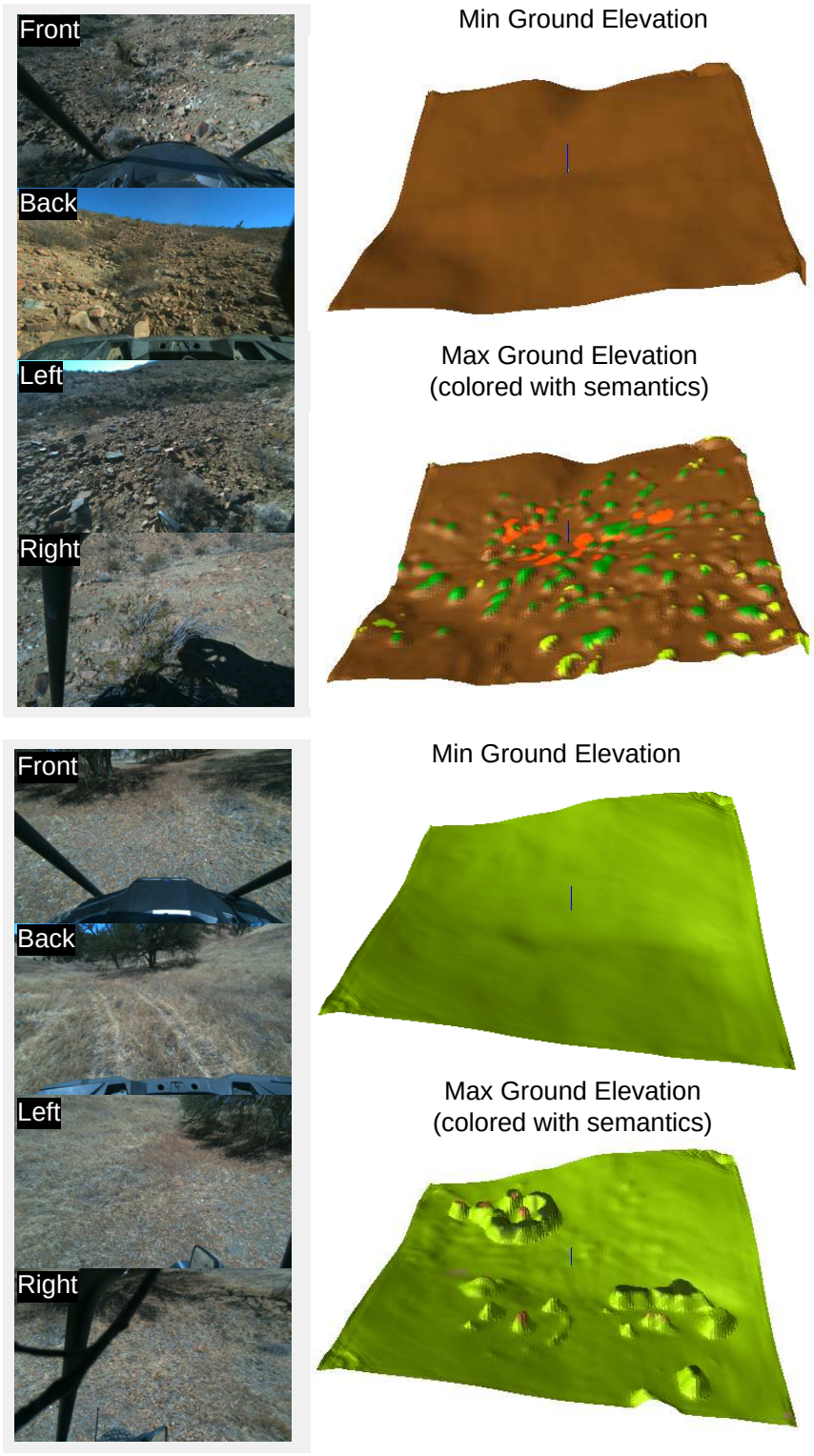


Figure B.5: **Visualizations of the predicted 3D terrain model from TerrainNet-TA**. We only show the min and max ground elevation here for clarity. Left: uneven terrain with scattered grass, bushes, and rocks. Right: a grass-covered valley with tall trees and low-hanging canopies.

B.5 More qualitative results

Figure B.4 shows other example predictions of TerrainNet and other baselines on the validation sets. Figure B.5 visualizes different predicted 3D terrains of TerrainNet-TA on the validation sets. For more details, including videos, please visit the website.

Appendix C

Aim My Robot

C.1 Assets Creation

We converted individual `glb` files into `usd` format. For each empty scene, we manually add appropriate lighting such as ceiling lights. At run time, we use the metadata file provided by HSSD to load individual objects into the scene and transform them to the correct location. We use the provided `objects.csv` to load the semantics for each object and set their semantic attributes accordingly in Isaac Sim. Fig.C.1 compares the rendering quality of the same scenes in Habitat and in Isaac Sim.

C.2 Data Generation

We simulate a RealSense D455 camera mounted at 1.5 m high above the floor, with tilt joint attached to the base. The data generation pipeline consists of three stages: (1) generate start and goal configurations; (2) generate plans; (3) render observations. We detail each stage as follows.

C.2.1 Generate start and goal configurations

Sample reference images We divide a scene into rooms, and put 4 cameras in each room to render the reference images. These cameras are placed at corners to cover different viewpoints. See Fig. C.2 as an example. Additionally, with a probability of 25%, we use the robot's initial observation as the reference image (*First Frame is Reference (FFR)* mode).

Sample robot start configuration We randomly sample robot radius $R \in [0.1\text{m} - 0.5\text{m}]$ as the robot's embodiment. The robot is then randomly placed in an area that is traversable and not in collision.

Figure C.1: Comparing the rendering quality of the same HSSD scene from Habitat Sim and Isaac Sim.

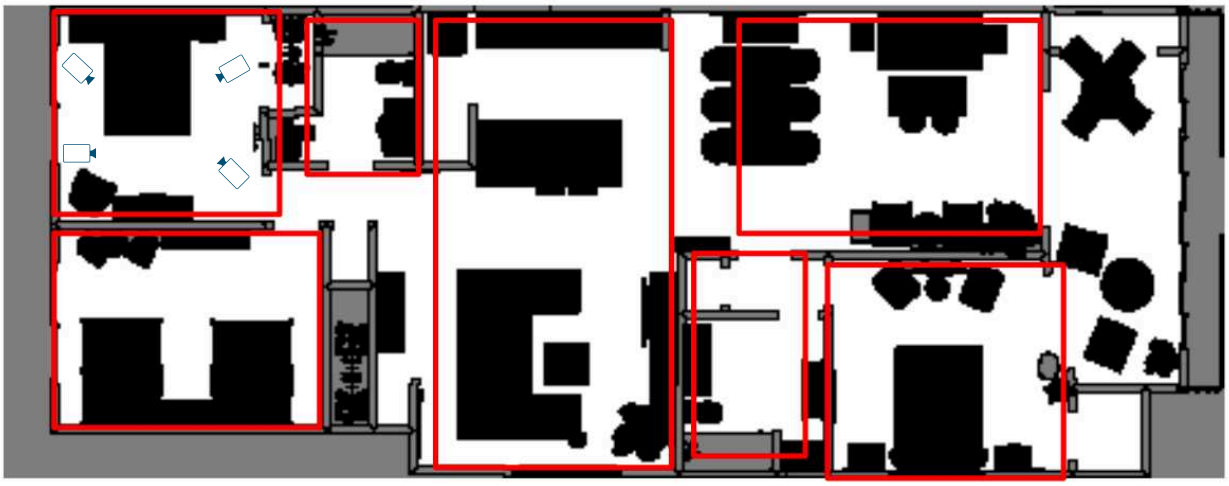


Figure C.2: Placement of reference images. Each scene is divided into rooms (red boxes). For each room, we render 4 observations from 4 camera viewpoints. An example camera placement is shown in the top left room.

Sample target object The target object is randomly selected from the reference image. We exclude non-object semantic classes such as walls and floors. Then, we sample a goal specification $\mathbf{C}$ by randomly choosing a side $S \in \{\text{front}, \text{back}, \text{left}, \text{right}\}$, distance $d \in [0.1\text{m}, 0.5\text{m}]$ and angle $\theta \in \{0°, \pm 15°, \pm 30°\}$. Finally, we place the robot at the goal pose and check for collisions.

C.2.2 Compute trajectory

Given the start and goal poses and the groundtruth floorplan, we run AIT* planner from OMPL to compute a trajectory. The planner uses ReedsShepp state space with a turning radius of 0 to model a differential drive robot. The cost function encourages the front of the robot to look toward the goal while penalizing excessive backward motion. Small backward motion is allowed if it is more efficient to reach the goal. If no plan is found, we discard the start-goal pair.

C.2.3 Render observations

For each feasible path, we render the camera observations along the path with a distance gap of 0.2 m or 5°. The camera tilt angle is set such that the bottom of the target object appears at the 1/4 from the bottom of the image.

C.3 Additional Details on the Network Architecture

Transformer encoders and decoders. All the transformer encoders and decoders have 4 layers, 4 heads, $d_{\text{model}} = 512$, and $dim_{\text{feedforward}} = 2048$ with GeLU activation. The exception is the vision context encoder $\mathcal{F}$ which has 8 layers and 8 heads.

MLP encoders: The laser encoder is a 2-layer MLP, each with 512 dimensions. The embodiment and look-at pose encoders are single-layer MLP with 512 dimensions.

MLP predictors: The base and camera tilt decoders are single-layer MLP with 512 dimensions.

C.4 Additional Experimental Results

C.4.1 Scaling properties

To study how the diversity of the training environments affects the model's generalization, we train AMR on a subset of the scenes (12 and 27, respectively) for the same number of training steps. Fig. C.3 compares the performance of the models in the unseen environments. All models have comparable median errors, but there is a significant boost in accuracy when increasing the number of scenes from 12 to 27. However, we observe only slightly improved performance in distance errors when increasing the number of environments from 27 to 49 scenes.

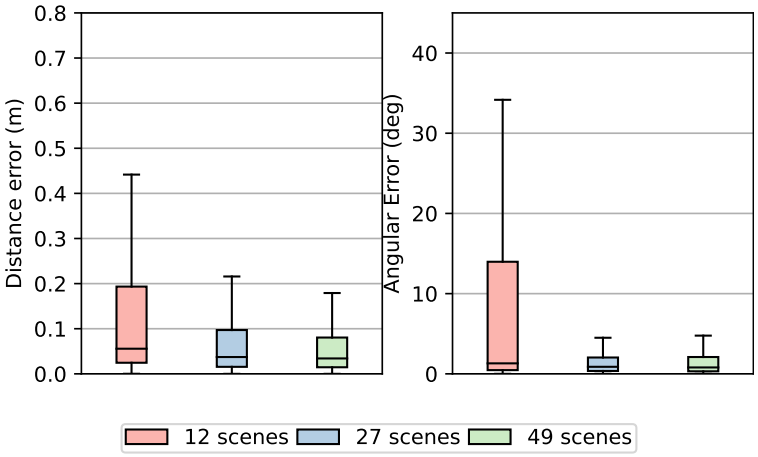


Figure C.3: Effects of number of training environments

C.5 Forklift Modeling

The forklift in our simulation experiment is based on the commercial forklift Crown SP3200. In this section, we provide additional information about the kinematic model and the tracking controller.

C.5.1 Kinematic model of a forklift

Fig. C.4 illustrates the kinematic model of a forklift. A forklift is an Ackermann steering vehicle. Its kinematics can be described by the following equations:

$$\dot{x} = v \cos\theta,$$
$$\dot{y} = v \sin\theta,$$
$$\dot{\theta} = \frac{v}{L} \tan\xi,$$

where x, y, θ denote the forklift's position and heading in the world, v is the linear velocity, ξ is the steering angle, and L is the distance between the steering wheel and the center of the rear axle. Note that the fork is located at the *back* of the vehicle.

C.5.2 Finetuning the model trained in HSSD + Isaac Sim

Since the forklift is big (more than 2 m long including the fork), it cannot fit into any of the scenes in HSSD. Therefore, we generate additional training data in a simulated warehouse, with random pallets placed on the floor.

The trajectories are generated with the vehicle facing *backwards*, as we are commanding the forks of the forklift to be in the forward direction of the vehicle. The turning radius of the planner

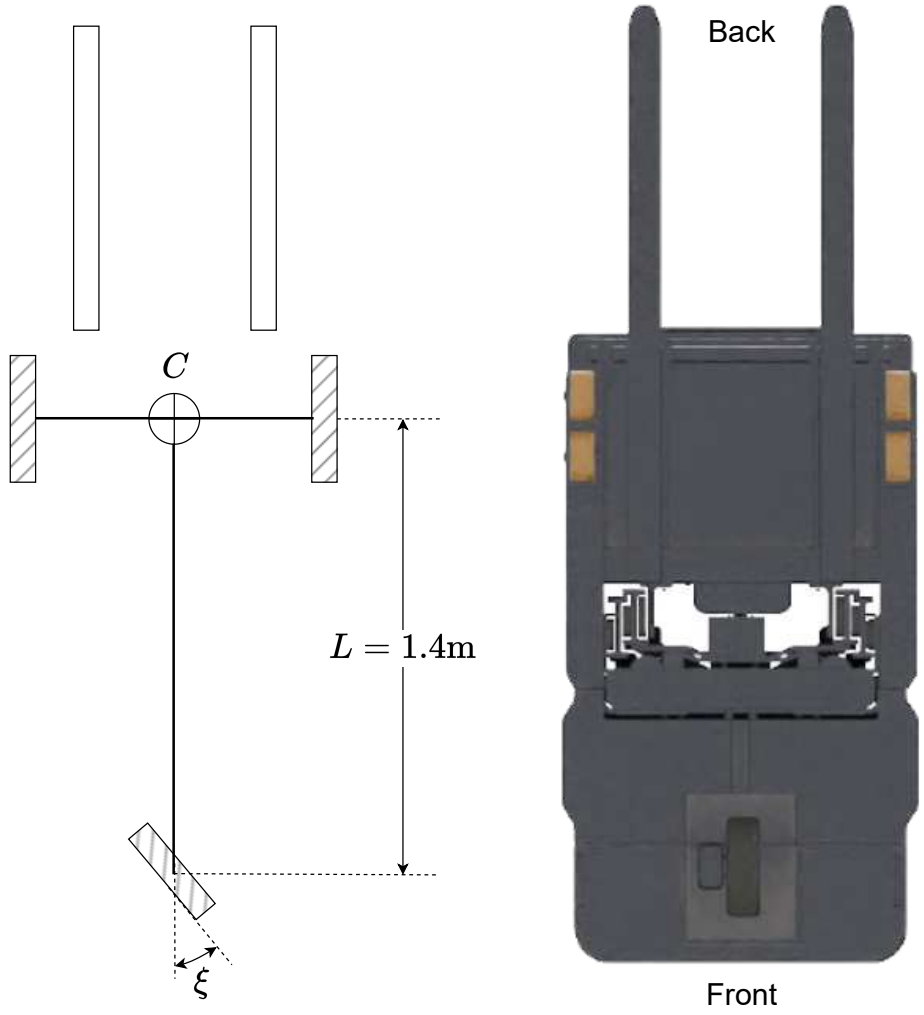


Figure C.4: **Left:** Kinematics of a forklift. C is the axle center. L is the wheel base, and ξ is the steering angle. Note that the fork is at the back of the vehicle. **Right:** bottom view of the simulated forklift.

is set to match the turning radius of the forklift. We perform DAgger training to improve the robot's positioning accuracy when near a pallet.

C.5.3 Controlling a forklift

We use a Model Predictive Control (MPC) controller to track the predicted trajectory. Given the current state s and a lookahead waypoint x, y, θ, we compute the optimal control v, ξ by

$$v^*, \xi^* = \underset{v,\xi}{\operatorname{argmin}} H(f(s, v, \xi), [x, y, \theta])$$

where $f(\cdot)$ is the dynamics model and $H(\cdot, \cdot)$ is a tuned cost function. New lookahead waypoints and controls are computed at every time step until the robot reaches the goal.