Learning Models and Cost Functions from Unlabeled Data for Off-Road Navigation

Samuel Triest

CMU-RI-TR-23-07

April 12, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Sebastian Scherer, *chair*
Aaron Johnson
Cherie Ho

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Off-road driving is an important instance of navigation in unstructured environments, which is a key robotics problem with many applications, such as exploration, agriculture, disaster response and defense. The primary challenge in off-road driving is to be able to take in high dimensional, multi-modal sensing data and use it to make intelligent decisions on where and how to traverse over unstructured terrain. While off-road driving has been the focus of much research, it remains challenging to design systems that are capable of off-road driving quickly, robustly, and in a variety of environments.

It is common practice for modern off-road driving systems to perform some form of semantic segmentation to bin terrain into one of several discrete classes (such as trails, vegetation, obstacle) and perform path planning and trajectory optimization to navigate through low-cost terrain classes. While such systems have generated strong empirical results, they are often engineering intensive, requiring thousands of densely labeled images. Such a process can take tens to hundreds of hours of an engineer's time. Furthermore, the mapping of terrain classes to cost ignores geometric information (e.g. some bushes may be more dangerous than others), and the relative cost of different terrain classes is not defined. As a result, these systems often need to be tuned extensively in the field, which generally requires the effort of many skilled engineers.

In order to avoid this engineering-intensive labeling and tuning process, recent work has proposed learning end-to-end navigation policies from autonomously collected data. While such an approach has the promise of reducing the engineering burden to deploy off-road driving systems, these approaches often lack interpretability and performance guarantees. Additionally, relying on random exploration to collect training data is unsafe for full-scale robots, which can seriously damage the environment and themselves. As such, these methods are generally infeasible to deploy on full-scale robots.

In this thesis, we propose a learning-based system for off-road driving for a full-scale autonomous all-terrain vehicle. Key to the work in this these are two assertions: 1) We can design a high-performance system without the need of any human-annotated data, and 2) learning should be used in concert with existing trajectory optimization methods for off-road driving. To address the first point, we collect a large dataset

containing several hours of human-driven trajectories through challenging off-road terrain at high speeds. This dataset contains many traversability-stressing scenarios, such as steep slopes, dense vegetation and high speeds. To address the second point, we leverage this dataset to learn two key functions in trajectory optimization, the dynamics model and the cost function.

Overall, we find that our learning-based methods outperform traditional common-practice navigation baselines in isolation. More importantly, we also demonstrate that these models lead to improved performance when incorporated in path planning and control algorithms through large-scale, multi-kilometer navigation trials.

Acknowledgments

First, I'd like to thank my collaborators and committee. The research presented in this work is the product of your advice and skills. Field testing in particular is a team effort, and none of the hardware results would have come together without your help. In particular, I'd like to thank Wenshan Wang and Sean Wang for helping me get started in off-road research, and Mateo Guaman Castro and Matthew Sivaprakasam for being the people who really helped me make things happen on hardware and field testing.

Second, I'd like to thank everyone who I worked with as an undergrad, who shaped my research philosophy and inspired me to pursue a graduate career. In particular, I want to thank Yuhao Zhu, Tom Howard, John Dolan and Adam Villaflor for helping me get my start in research, and robotics research. I remember Tom Howard once told me in one of our meetings for my undergrad thesis on RL for self-driving cars: "Would you get in a car running your algorithm?" I'm a few years late, but I can now answer yes to that question.

Lastly and most importantly, I'd like to thank my family for giving me the education, skills and opportunities that made all of this possible.

Funding

This work was supported by ARL awards #W911NF1820218 and #W911NF20S0005.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation, Challenges and Prior Work

Robot navigation in unstructured environments is a key challenge that has been the subject of much research [1, 3, 5, 8, 25, 44, 48, 66]. At a high level, this problem requires robots to interpret large streams of high-dimensional sensory data, and use said data to decide where and how to navigate to some goal. Humans are generally able to complete this task safely, at high speeds. State of the art in navigation for off-road robots remains comparatively slow.

Navigation in unstructured environments is a key component of many potential robotics applications, such as search and rescue, last-mile delivery and exploration. Off-road navigation serves as a challenging instance of navigation in unstructured environments, with major challenges including interaction with slopes, rocks and vegetation, and potentially at high speeds.

Designing robots and autonomy software that can reliably perform off-road navigation is challenging for a number of reasons. First, robots are required to compress large amounts of high-dimensional (and often multi-modal) sensory data into compact representations in order to perform planning and control on. Ideally, this representation should be expressive enough to encode a sufficient level of environmental nuance, while remaining compact enough to generalize to many environments and enable high-frequency re-planning. Second, robots can be exposed to a wide variety of unforseen scenarios during operation. A successful navigation system thus must have

Figure 1.1: Off-road navigation is a challenging problem that requires sophisticated state estimation, perception, planning and control to achieve near-human performance. Shown are examples of human teleoperation through challenging scenarios, such as deep puddles, tall vegetation and slopes.

a world representation that is robust and generalizable to novel scenarios. Finally, the robot must be able to avoid navigational mistakes that have the potential to be extremely harmful to the robot or its potential passengers, and be able to do so reliably over long distances and timeframes.

Classical approaches to off-road navigation typically rely on dividing the environment into traversable or non-traversable space based on geometry, and plan through kinematic [1] or simplified dynamic models [24] to find efficient routes to a given point through traversable space. Such an approach has been used many times in practice [1, 3, 44, 64] and at-scale. While these approaches are fairly robust and generalizable to many robot platforms, these benefits typically come at the cost of performance. A well-known failure case of geometric baselines is tall grass, which is traversable, but often labeled as an obstacle [28]. Furthermore, classical systems often require re-tuning in the field to account for novel scenarios [65, 70], which is often a slow, heuristic and laborious process.

Given the ability of humans to collect and label large amounts of high-quality driving data, a potentially appealing approach for improving the performance of navigation systems in off-road environments is to leverage that data in some sort of

learning framework. Such approaches have been considered by prior work [5, 9, 15, 28, 36, 45, 65]. These approaches generally fall into two categories: end-to-end learning, and semantic segmentation. While end-to-end learning approaches have theoretically high performance, they are typically data-hungry, non-interpretable and have limited to no safety guarantees. Furthermore, they often rely on long exploration phases that require large amounts of random actions [28]. This may not be feasible to deploy on large-scale systems due to the risk of damage to the robot and human operator.

Semantic segmentation methods are a potentially appealing way to improve the fidelity of a robot's planning representation by directly encoding navigation-relevant classes (e.g. trails, grass, etc.) into a robot's model of the environment [36, 45]. However, the mapping between semantic classes and robot affordances is often coarse (i.e. not all bushes are traversable). Furthermore, creating a semantic segmentation dataset is a highly time-consuming process. Consequently, most semantic segmentation datasets for off-road driving [27, 36, 58] are much smaller than datasets used for end-to-end learning, or traditional machine learning tasks [6, 13, 33].

1.2 Contributions

The work presented in this thesis will describe a navigation system for a Yamaha Viking All-Terrain Vehicle (henceforth referred to as the Yamaha ATV) in unstructured, off-road environments. There are two key philosophies behind the design decisions in this system. First, we require **interpretable** navigation representations. Interpretability is critical in large-scale navigation tests as the robot has the potential to harm its operators and itself. Thus a safety driver must be able to understand the robot's decision-making process and intervene before any unsafe behavior occurs. Furthermore, interpretability also allows for quicker adaptation and tuning in the field. The second key philosophy adopted by this work is to use learning methods that can leverage **self-supervision**. Large-scale, labeled datasets are time-consuming to create, and are often collected in a single environment. We assert that self-supervised methods are simpler to develop and deploy without sacrificing performance. Self-supervised datasets can also scale to be orders of magnitude larger than labeled datasets. As such, several core modules of the Yamaha navigation stack are learned using large corpora of unlabeled human driving data.

In order to make the techniques presented in this thesis as widely applicable as possible, we define our task to be to navigate to a series of waypoints (spatial positions which may come from a human operator or a global planning algorithm) using only on-board computation, without the aid of any prior observations of the terrain (e.g. no aerial imagery, or observations from additional robots). While these limitations may not exist for all applications of robot navigation (e.g. planning with satellite imagery [5, 41, 42], maintaining maps between autonomy runs [30], or sharing maps between multiple robots [3, 44]), this set of problem constraints represent a minimal set of assumptions and the methods presented in this thesis would likely be enhanced by relaxing these constraints.

The remainder of this thesis is split into six sections, which will discuss the following:

2. Conceptual preliminaries, namely viewing off-road navigation as a trajectory optimization.
3. An overview of the Yamaha ATV and testing site. This section primarily focuses on implementation details such as vehicle sensors and actuators, coordinate frames and setup of navigation courses.
4. Work on learning vehicle models from terrain interaction data (adapted from [51]).
5. Work on learning cost functions from driving trajectories collected from human experts, and evaluation on large-scale navigation tests (adapted from [52]).
6. Related work in model and cost function learning for off-road driving.
7. Conclusions and future work

Chapter 2

Preliminaries

2.1 Notation

Described in Table 2.1 are naming conventions for common variables in this thesis. In general, lower-case variables are reserved for vectors, and capital variables for matrices (or stacked vectors).

It is often useful in this domain to be able to extract positions from a vehicle state x, which may contain additional state variables such as velocities and actuator positions. To avoid overloading of notation, planar positions will be denoted as $\overline{p} = (p_x, p_y)$, and a general position extraction function P is defined such that $P(x) = (p_x, p_y)$. Lastly, let $C(P(\tau)) : \mathbb{R}^{k \times m} \to \mathbb{R}^k$ (and by abuse of notation, $C(\tau)$) represent the process of evaluating a trajectory over a costmap. This is accomplished by first computing map coordinates, and then stacking the appropriate cell values.

2.2 Trajectory Optimization

For the purposes of this thesis, off-road navigation can generally be viewed as a trajectory optimization problem. A trajectory optimization problem can be defined as the following:

Variable	Definition
x	robot state
n	size of robot state (i.e. $x \in \mathbb{R}^n$)
p	(three-dimensional) robot position
$\overline{p}$	planar robot position
q	robot orientation
u	robot control
m	size of robot control (i.e. $u \in \mathbb{R}^m$)
o	(high dimensional) robot observation
τ	trajectory or rollout
t	time
T	horizon
J	cost function
f	dynamics function
M	map ($M \in \mathbb{R}^{k_1 \times k_2 \times k_3}$)
C	costmap ($C \in \mathbb{R}^{k_1 \times k_2}$)
M_{res}	map resolution (m/cell)
M_x	map x-origin (location of lower-left cell)
M_y	map y-origin (location of lower-left cell)
$\mathcal{D}$	dataset

Table 2.1: Variable names for this thesis

$$\begin{aligned} \min_{u_{1:T-1}} \quad & J(x_{1:T}, u_{1:T-1}) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \qquad \forall t \\ & x_1 = \hat{x} \end{aligned} \tag{2.1}$$

Where the agent is responsible for selecting minimum-cost actions $u_{1:T}$ from an initial observed state $\hat{x}$. Key to this problem are two functions:

1. A dynamics function f that constrains the state transitions (x_t, u_t, x_{t+1}) of the robot. This dynamics function is applied recursively to state-action pairs (x_t, u_t) to generate rollouts τ.
2. A cost function J which measures the desirability of a given trajectory τ.

This representation is useful for this thesis as many cost functions and dynamics functions have interpretable, useful representations (e.g. trajectory rollouts, costmaps). In the off-road driving domain in particular, reasonable baseline approximations of

these two functions exist, and will be described in the following sections.

2.3 Baseline Models

A simple but effective approximation of the dynamics function of an Ackermann-steered vehicle is the kinematic bicycle model (KBM). While this model makes several strong assumptions such as lack of tire slip and flat terrain, which are generally not true for off-road navigation, these assumptions are often made in practice [1] to reasonable success. The basic form of the KBM is provided in Equation 2.2.

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}, u = \begin{bmatrix} v \\ \delta \end{bmatrix}, \dot{x} = \begin{bmatrix} vcos(\theta) \\ vsin(\theta) \\ v\frac{tan(\delta)}{L} \end{bmatrix} \tag{2.2}$$

A problematic assumption of the basic KBM is that it assumes that velocities v and steering angles δ can be changed instantaneously, which is rarely true in practice. Fortunately, it is relatively simple to modify the basic KBM to admit *desired* velocity and steering $[v_{des}, \delta_{des}]$ by adding additional steering and throttle dynamics (Eq 2.3).

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \\ v \\ \delta \end{bmatrix}, u = \begin{bmatrix} v_{des} \\ \delta_{des} \end{bmatrix}, \dot{x} = \begin{bmatrix} vcos(\theta) \\ vsin(\theta) \\ v\frac{tan(\delta)}{L} \\ K_v(v_{des} - v) \\ K_\delta(\delta_{des} - \delta) \end{bmatrix} \tag{2.3}$$

Note that both the steering and throttle dynamics are essentially modeled as a P controller with gains K_v and K_δ respectively. In practice, several quantities in these equations, such as steering rate and speed are clamped in accordance to actuation and safety constraints of the vehicle. Sample KBM outputs are shown in Figure 2.1.

2.4 Baseline Cost Functions

The cost function plays a critical role in the trajectory optimization problem as it is responsible for telling the agent whether a given trajectory is good or bad. Generally,

Parameter	Value
L	$3.0m$
K_v	1.0
K_δ	10.0
v limits	$[0.5, 3.5]m/s$
δ limits	$[-0.52, 0.52]rad$
$\dot{\delta}$ limits	$[-.0.2, 0.2]rad/s$

Table 2.2: Table of Parameter Values for the Yamaha ATV

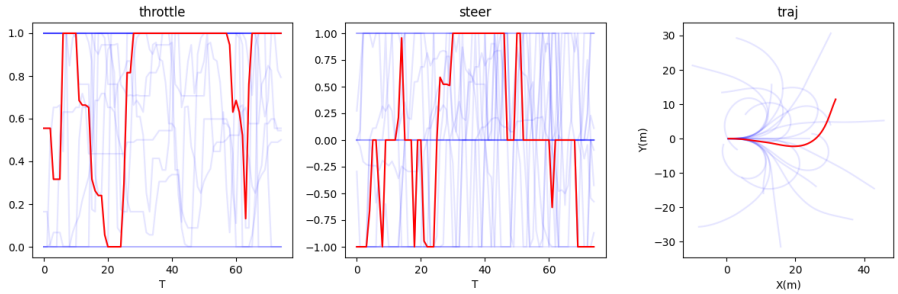


Figure 2.1: Twenty example trajectories from a KBM with unit-normalized throttle and steering commands from our data.

a cost function $J(\tau)$ is decomposed into a stage cost an terminal cost (Equation 2.4).

$$J(x_{1:T}, u_{1:T-1}) = \sum_{t=1}^{t=T-1} [J_S(x_t, u_t)] + J_T(x_T) \tag{2.4}$$

For navigation problems, it is typically sufficient to use Euclidean distance to a goal point as a terminal cost:

$$J_T(x_T) = ||P(x_T) - p_g||_2 \tag{2.5}$$

Similarly, stage cost is often represented using a costmap.

$$J_S(x_t, u_t) = C(p(x_t)) = C(x_t) \tag{2.6}$$

A simple baseline costmap instance for off road driving is an occupancy map 2.2. Given a set of obstacles, such a costmap can be constructed by assigning any cell that intersects an obstacle as high cost. All other cells are assigned a cost of zero. It

is also common practice to perform obstacle inflation, where cells near an obstacle are assigned some cost proportional to their distance to an obstacle. In practice, this allows robots to maintain a safe distance from obstacles.

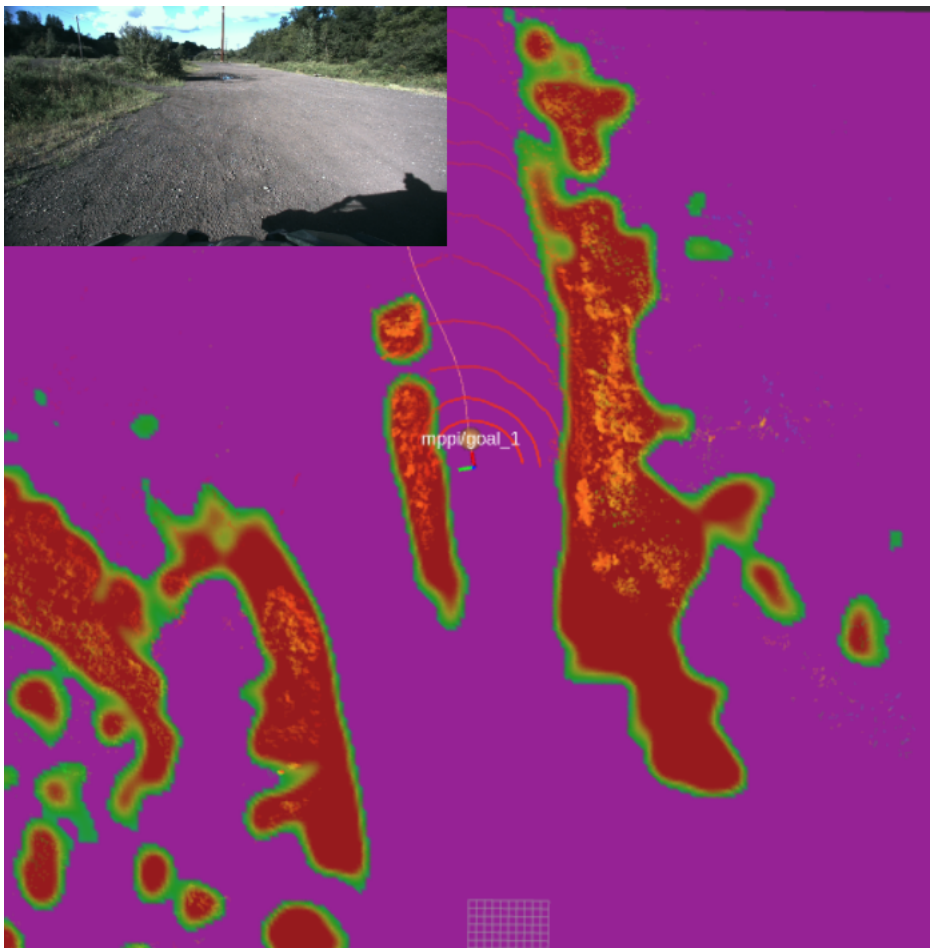


Figure 2.2: An example costmap based on occupancy, with the vehicle located at the center. Low cost is shown in purple, while high cost is shown in red. Areas containing tall bushes are considered high cost, and are inflated to account for the size of the ATV. The corresponding FPV image is provided for context.

Chapter 3

Experimental Setup

A major component of this thesis work is the testing of our methods on a full-scale ATV. As such, this section describes the means through which the ATV senses and interacts with its environment.

3.1 Testing Site

All large-scale experiments in this thesis were performed at Gascola, a testing site in Pittsburgh, PA. Gascola contains roughly 200 acres of rough terrain with several key off-road scenarios, including trails, tall grass, obstacles and slopes. Gascola contains many instances of each of scenario at varying degrees of difficulty (Figure 3.1).

3.1.1 Navigation Courses

The primary means of evaluating our algorithms is via navigation to a series of pre-defined GPS waypoints. The vehicle is required to drive within a small radius each of the waypoints in sequence. A qualitative example of this is shown in Figure 3.2. For this work, waypoints were determined by extracting poses from a human tele-operation of the vehicle.

Figure 3.1: Some representative terrains at Gascola, including slopes, trails and dense vegetation.

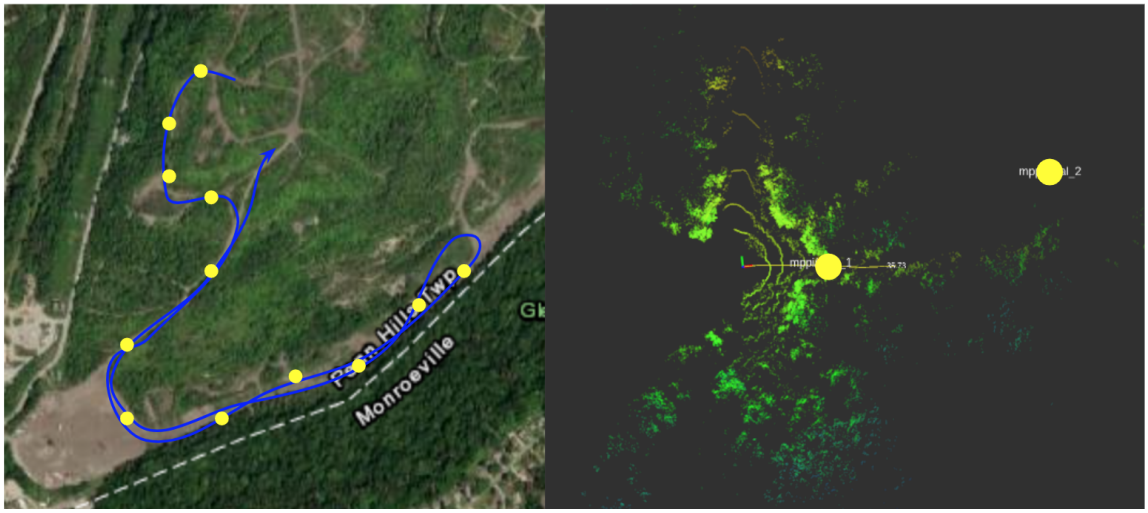


Figure 3.2: A qualitative illustration of our navigation courses. (Left) Given a human teleop demonstration (blue), we extract waypoints (yellow) at a fixed distance interval. (Right) Those waypoints are then transformed into the robot experiment frame and used as goals.

3.2 Yamaha ATV

Experiments for this work were performed using the Yamaha ATV, which was modified for autonomous driving by Mai et al. [34]. As hardware design is not a major component of this work, we will refer readers to Mai et al. for a more complete description of the ATV hardware.

3.2.1 Sensing

The primary sensors on the vehicle were the following:

1. A NovAtel PROPAK-V3-RT2i GNSS unit, which provided GPS poses (10hz) and IMU data (100hz).
2. A Velodyne UltraPuck, which provided pointclouds (10hz).
3. A Multisense S21, which provided RGB and stereo images (10hz).

3.2.2 Actuation

The steering and throttle of the ATV were actuated with a Kairos Autonomi steering ring and Dynamixel MX-28T servo, respectively. The steering ring was set up for position control via a low-level PID controller. Throttle could either be set directly via the Dynamixel servo position, or run through another PID controller to track desired velocities, given velocity estimates from the NovaTel.

3.2.3 State Estimation

For the majority of the work in this thesis, we use a version of Super Odometry [69] modified by VanOsten [54]. This provides the vehicle 3D state estimates $x = [p; q; v; \omega]$, given the IMU data from the NovaTel and pointclouds from the Velodyne.

3.2.4 Perception

A lidar-based mapping module was developed as part of this work. At a high level, the role of the lidar perception is to produce cell-wise feature vectors from lidar pointclouds and odometry which can be consumed by downstream costmapping modules. The specific details of the lidar feature extraction are left for a later section (5.3.2). In general, the algorithm works by maintaining a buffer of pointclouds, and the ego-poses $[p; q]$ that the pointcloud was measured at. Whenever a map should be produced, the pointclouds in the buffer are transformed to the current vehicle frame and aggregated (note that the pointclouds remain aligned with gravity and the map axes).

Mapping was done this way because it:

1. Maintaining a registered pointcloud allows for local maps to be generated at arbitrary sizes and resolutions
2. Maintaining a rolling buffer allows the perception to be reasonably robust to mis-registered points.

A qualitative example of a registered pointcloud is provided in Figure 3.3.

Algorithm 1: Lidar Perception Algorithm

Input:
Pointcloud buffer $\mathbb{X}$,
Vehicle state x_v,
$X_{agg} = \emptyset$ ◁ Initialize empty pointcloud aggregator
$T_m = [P^3(x_v); 1; 0; 0; 0]$ ◁ Compute location of local map frame
for $X, T_x \in \mathbb{X}$ **do**
  $T_x^m = T_m(T_x^{-1})$ ◁ Compute transform from pointcloud frame to map frame
  $X_{agg} = X_{agg} \cup T_x^m X$ ◁ Add transformed points to buffer
end
M = feature_extraction(X_{agg}) ◁ Perform feature extraction

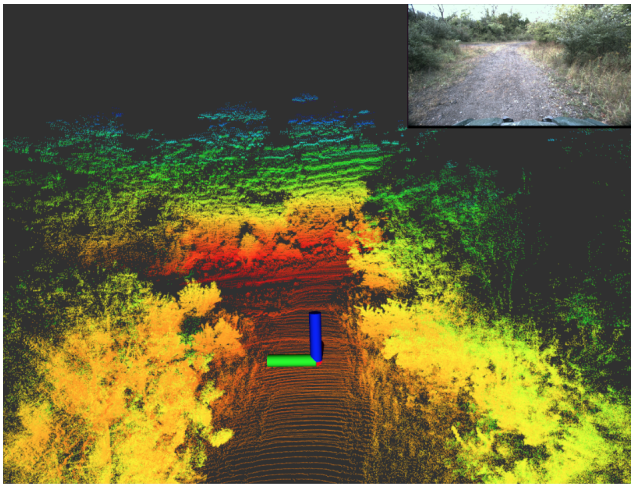


Figure 3.3: An example of a registered pointcloud, with the FPV image shown for context.

3.2.5 Planning and Control

For this thesis, we relied on sampling-based methods of planning and control. Such methods were desirable due to several key properties, such as:

1. Time and space budgets are simple to enforce via number of samples and lookahead length
2. Can naturally handle non-smooth, non-convex cost functions
3. High empirical performance on mobile robots [1, 59]

Additional details are provided later in the thesis.

3.2.6 Frames

Vehicle experiments were primarily conducted in an gravity-aligned inertial frame set at the vehicle position at the start of the experiment. Vehicle pose estimates and maps are computed with respect to this frame. Additionally, we maintain a GPS frame that remains consistent between runs, allowing use to merge data from multiple runs together and send globally consistent waypoints. More details are provided in Figure 3.4.

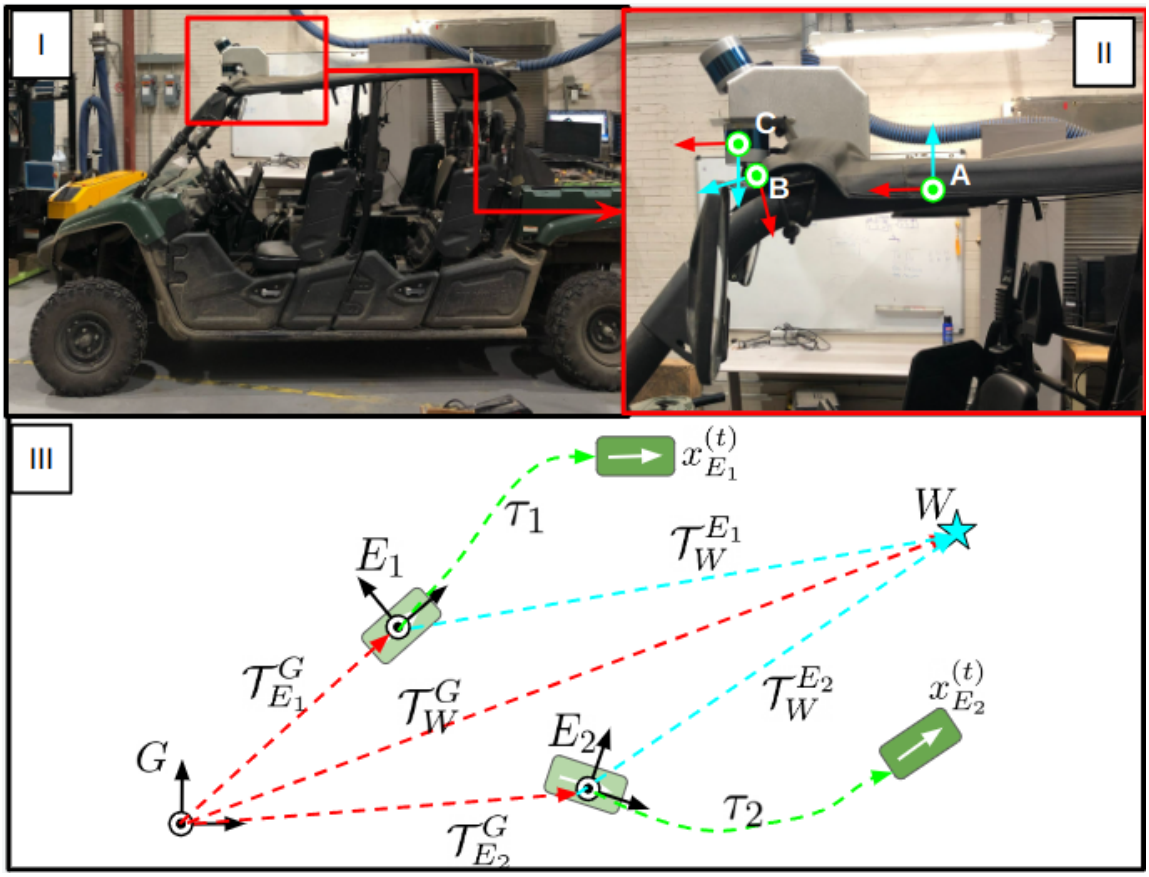


Figure 3.4: A diagram of the relevant frames for the ATV. (I) A picture of the Yamaha ATV. (II) The local frames of the relavant sensors including the NovaTel GPS and IMU (A), the Multisense (B) and the Velodyne (C). (III) A high-level diagram of the GPS and experiment frames. Waypoints (W) are provided in the GPS frame (G). At the start of a run, an experiment frame is set (E_1, E_2) at the vehicle start position. This frame is treated as the inertial frame that trajectories τ, states x and waypoints W are given in for preception, planning and control. Waypoint locations are calculated by inverting the transformation from GPS to experiment ($\mathcal{T}_E^G$)

.

Chapter 4

Learning Models

4.1 Motivation

Performing trajectory optimization generally requires access to a good vehicle model. While a number of analytical models, such as the kinematic and dynamic bicycle models, exist and are commonly used in off-road applications, they make a number of simplifying assumptions that are consistently violated in practice. Breaking these assumptions (such as differential flatness and tire contact points) is generally benign in applications with simpler terrain, such as in warehouses or in highway driving. Furthermore, it is generally possible in these simpler environments to treat areas where dynamics assumptions don't hold as obstacles. However, in outdoor unstructured environments, it is not sufficient to simply avoid these regions. It is rarely the case that the path from a robot to its goal point will be on flat, smooth ground. It is also likely that a robot will interact with objects such as logs or tall grass on its route to a goal.

While modeling methods exist that can simulate complex terrain interactions and interactions with deformable objects, such simulators are prohibitively slow to use in high-frequency optimization, and are often non-differentiable (e.g. mode switching for contacts). Additionally, the fidelity of said simulators are highly dependent on accurate model parameters such as friction coefficients, object poses, object/terrain geometry etc., which may not be able to be estimated reliably using on-board sensing.

As such, it seems to be a reasonable alternative to attempt to learn these models

of physics directly from interaction data, where these complex relationships between the vehicle and its environment can be approximated without needing to be explicitly modeled.

4.2 Problem Formulation

Practically speaking, learning a model from corpora of trajectory rollouts can be formulated as a standard maximum-likelihood estimation problem. That is, given a corpus of state transitions (s, a, s') in a dataset $\mathcal{D}$, we can fit a stochastic dynamics model $f_\theta(\cdot|s, a)$ as the following:

$$\max_\theta \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\big[f_\theta(s'|s,a)\big] \tag{4.1}$$

Where $f_\theta(s'|s, a)$ is typically parameterized as a diagonal Gaussian. For deterministic dynamics models, it is common practice to replace maximum-likelihood with minimizing mean squared error.

4.2.1 Practical Considerations

In practice, it is often more effective to predict state differences instead of new states [38]. That is, the dynamics model is trained to predict $\Delta = s' - s$, where new states can be computed as $s' = s + \Delta, \Delta \sim f_\theta(\Delta|s, a)$.

Additionally, we generally maximize the likelihood of entire trajectories instead of individual state transitions. That is, for a given horizon H, we sample an H-step sequence of state transitions $(s_1, a_1, s_2), (s_2, a_2, s_3), \ldots (s_{H-1}, a_{H-1}, s_H)$ and maximize the likelihood over the entire trajectory.

$$\max_\theta \mathbb{E}_{(s_{1:T-1},a_{1:T-1},s_{2:T})\sim\mathcal{D}}\big[\prod_{t=1}^{T-1} f_\theta(s_{t+1}|s_t,a_t)\big] \tag{4.2}$$

This has empirically been observed to increase the fidelity of long-horizon predictions [38, 55]. Note that s_{t+1} is dependent on the model prediction at time t. Unless otherwise noted, s_{t+1} will be generated using the mean prediction s_t.

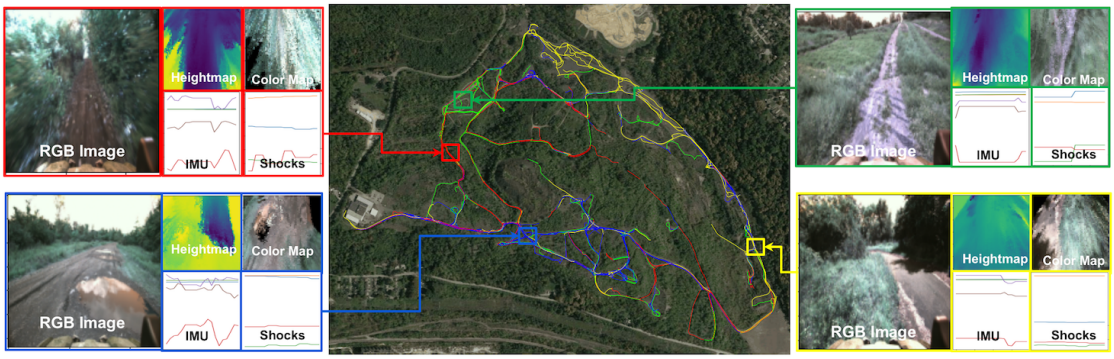


Figure 4.1: Our dataset contains several hours of driving data from a test site in Pittsburgh, PA. Our dataset contains diverse off-road driving scenarios and several sensing modalities, including front-facing camera and top-down local maps. Shown in the center is a satellite image of the testing site with trajectories superimposed. They are colored according to a clustering based on ResNet [22] features on the RGB images. Shown on the sides are four datapoints (one from each cluster) with five of the seven modalities available in the dataset shown. We can observe a diverse set of scenarios, including dense foliage (red), steep slopes (green) open road (blue) and puddles (yellow).

4.3 TartanDrive

Critical to the model learning process is a large, diverse dataset from which dynamics interactions can be learned. As such, we collected a dataset of roughly five hours of dynamics data using the Yamaha ATV. This dataset contained around 200,000 frames of interaction of the ATV in a number of interesting scenarios, including slopes, vegetation, and interaction with foliage. We then leverage this dataset, called TartanDrive [51], to train several state-of-the-art dynamics models [19, 50] and evaluate the effectiveness of our learned dynamics models over traditional baselines.

4.3.1 Dataset Overview

A key differentiator between out dataset and other datasets for off-road driving is the lack of hand-labeled semantics. Many off-road driving datasets such as RUGD [58] and RELLIS-3D [27] are designed primarily for semantic segmentation. As as result,

Dataset	Samples	State	Action	Image	Pointcloud	Heightmap	RGBmap	IMU	Wheel RPM	Shocks	Intervention
RUGD	2700	No	No	Yes	No	No	No	No	No	No	No
Rellis 3D	13800	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No
Montmorency	75000	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No
Ours	184000	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.1: Overview and comparison of various off-road driving datasets

Modality	Type	Dimension	Train Dimension
Robot Pose	Vector	7	7
RGB Image	Image	2 x 1024 x 512	128 x 128
Heightmap	Image	500 x 500	64 x 64
RGB Map	Image	500 x 500	64 x 64
IMU	Time-series	20 x 6	20 x 6
Shock Position	Time-series	5 x 4	20 x 4
Wheel RPM	Time-series	5 x 4	20 x 4
Intervention	Boolean	1	1

Table 4.2: The set of available dataset features and their sizes.

these datasets generally contain a small number of hand-labeled images, and are generally unable to scale in terms of number of datapoints because of this dependence on hand-labeling. In contrast to this approach, our dataset does not contain any hand-labeled features. While this prevents it from being useful for semantic segmentation, this allows the dataset to be an order of magnitude larger than other real-world datasets.

Additionally, our dataset contains a large amount of sensing modalities, including both proprioceptive and exteroceptive modalities. Table 4.1 compares our dataset to other common datasets for off-road driving. Table 4.2 provides a more detailed description of the different modalities in our dataset.

4.3.2 Collection Process

Data were collected using human tele-operation in the Yamaha ATV. Due to safety concerns, it was infeasible to perform random exploration. In total, we collected 630 trajectories across three days and five drivers at Gascola. Given the focus on dynamics modeling, trajectories were generally kept short (about 30 seconds per trajectory), and focused on interesting scenarios such as slopes and vegetation, and generally exhibited more changes in throttle and steering than normal driving behavior.

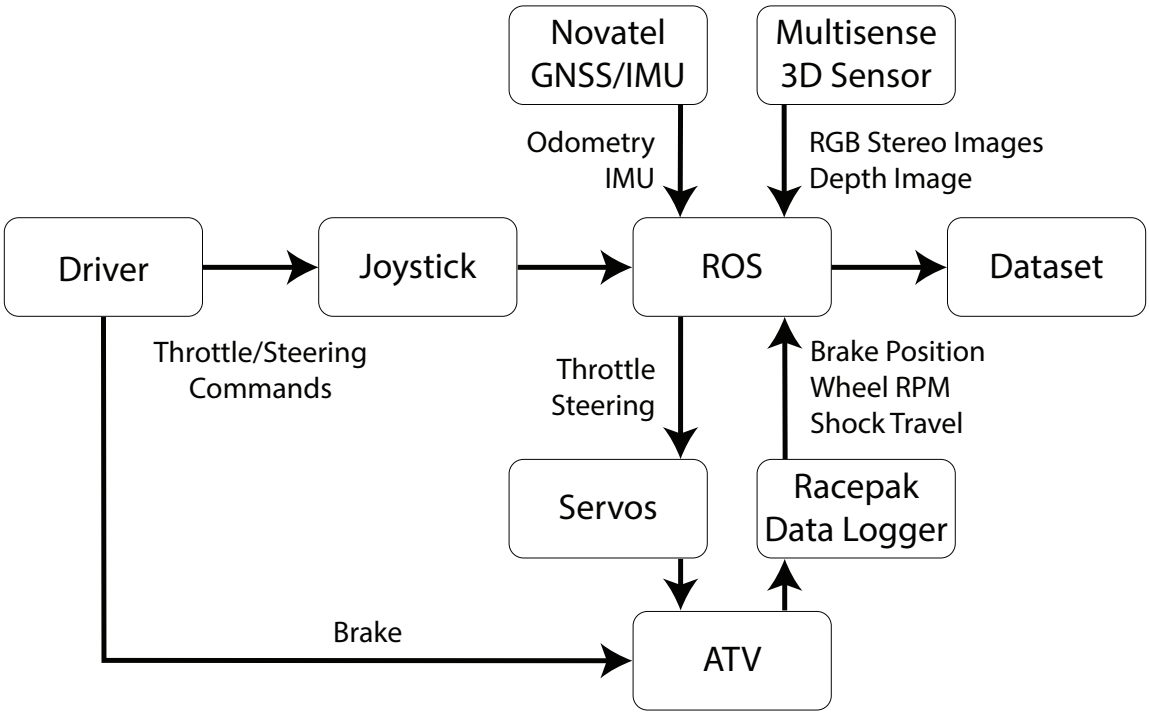


Figure 4.2: A system diagram of our data collection setup. We use multiple sensors to collect interaction data across many modalities, as well as throttle and steering commands.

Exteroceptive data (images, maps) and poses were collected at 10hz. Proprioceptive data (IMU, shocks, wheel RPM) were collected at 200Hz and stacked per-timestep to match the 10hz rate of poses and exteroception. For learning, exteroceptive data was significantly downsampled.

4.4 Motivational Experiment: Dynamical Variation in Practice

How the robot moves depends on the physical properties of the robot, the action command we send, and the physical properties of the environment. In simple environments such as urban roads, the robot properties and actions are sufficient to perform accurate trajectory prediction. We thus ask the question: does our dataset capture the dynamical variation induced by different types of terrains? To answer

this question, we first perform a motivational experiment to quantify the correlation between future states and action sequences. The rationale behind this experiment is that if significant dynamical variations exist due to different terrains, then we would expect that similar sequences of actions in the dataset may yield very different trajectories.

In order to perform this experiment, we first collected 10000 random subsequences of length 10 (one second) from our dataset and computed the displacement and rotation from the initial state to the final state (such that every trajectory began from the same initial state). We then performed time-series clustering (using [47]) on the corresponding sequences of actions. One important note is that we chose simple Euclidean distance instead of time-warping methods [37] as both the duration and temporal position of the actions in the sequence (and not just the shape of the sequence) affect state displacement. We then computed a t-Distributed Stochastic Neighbor Embedding (t-SNE) [53] of the state displacements and colored each embedded point according to its corresponding action cluster. To mitigate the effect that velocity has on the final state displacement, we binned our data based on initial speed and created a separate visualization for each bin. One of the resulting visualizations is shown in Figure 4.3. The remainder of the figures and the experiment hyperparameters are in Section 4.7

4.5 Multi-Modal Dynamics Modeling with TartanDrive

In addition to data collection, we benchmarked several recent neural network architectures for dynamics prediction from high-dimensional data. For this work, we consider the task of dynamics prediction to be the prediction of future states $x_{1:T}$, given an intital state x_0, a sequence of actions $u_{1:T}$, and a set of observations $O_0 = \{o_0^m\}$ from a set of modalities M. These models thus take the form $f_\theta(x_0, u_{1:T}, O_0)$.

We first describe the general architecture of our latent-space model for off-road dynamics prediction. Similar to prior work [19, 50], we use a latent-space model that is comprised of three parts:

1. An encoder $e_\theta(o_t) : \mathcal{O} \rightarrow \mathcal{Z}$ that maps a high-dimensional observation o_t into

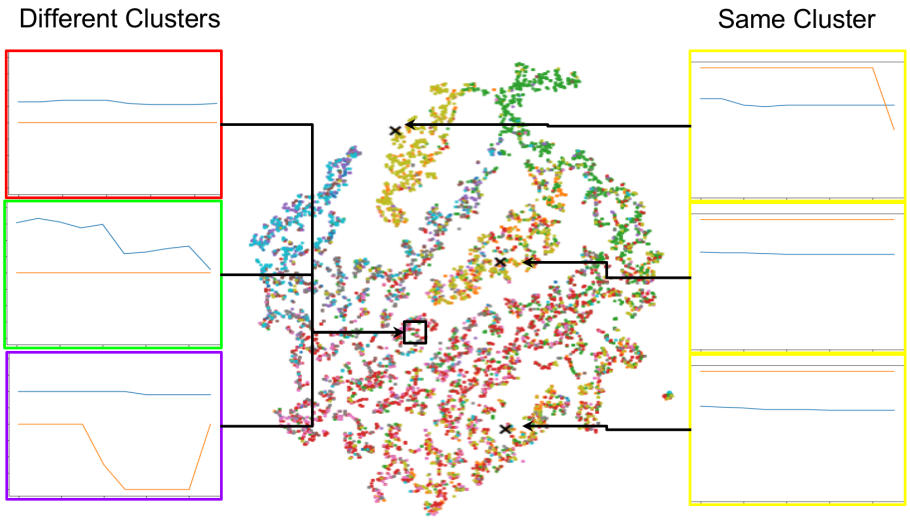


Figure 4.3: One t-SNE embedding of the state displacements in the dataset. Each point is colored according to its closest action sequence centroid. We can observe from this figure that there is some correlation between the clusters and their position in the t-SNE embedding, though the colors clearly mix. Shown on the left are three action sequences from different clusters that map to the same region in the embedding space. Conversely, shown on the right are three action sequences that map to very different regions of the embedding space, despite being very similar.

the latent space

2. A model $f_\theta(z_t, u_t) : (\mathcal{Z}, \mathcal{U}) \to \mathcal{Z}$ that forward-simulates the model in latent space, given actions.
3. A decoder $d_\theta(z_t) : \mathcal{Z} \to \mathcal{O}$ that maps the low-dimensional latent state back to the observation space.

Parameterizing the model in this fashion allows for efficient state prediction, as one only needs to encode the initial observation o_0 to get an initial latent state z_0. From this, state vectors $x_{1:T}$ can be recovered via forward-simulating the latent model to get $z_{1:T}$ and then decoding state without decoding the high-dimensional observations. A detailed description of the network architecture is provided in section 4.7. We now describe the specific implementation of our model for the ATV.

Multi-modal Encoders

The encoder for our model consists of a deep neural network for each modality m. The original size, and rescaled training size of each modality used is shown in Table 4.2. The neural network architectures used for the different modality types are as follows:

1. Vector inputs were passed through a dense network to produce a Gaussian distribution $p(z)$.
2. Images were passed through a convolutional neural network (CNN), flattened, and passed through a dense network to produce $p(z)$.
3. Time-series inputs were passed through a WaveNet encoder [39], flattened, and passed through a dense network.

To combine the multiple predictions on $p(z)$, we follow previous work [50, 61] which use a product of experts [23]. In this formulation, the aggregated probability of the latent state, $p(z)$, is determined by the product of probabilities of each expert $\prod_i q_i(z|o_i)$. This formulation is preferable over a mixture of experts for its ability to produce sharper distributions and allow experts to focus on smaller regions of the prediction space. Since our encoders output diagonal Gaussian distributions in $\mathcal{Z}$, we can compute the closed-form product of Gaussian experts using the result from Cao et al. [10],

$$p(z|\mathbf{O}) = \prod_i q_i(z|o_i) = \mathcal{N}\left(\frac{\sum_i(\frac{\mu_i}{\sigma_i})}{\sum_i \frac{1}{\sigma_i}}, \mathbb{I}\left(\sum_i \frac{1}{\sigma_i}\right)\right) \tag{4.3}$$

Latent Model

Our latent model is implemented as a GRU [11]. While prior work [19, 50] use the Recurrent State-Space model, we found that its performance was similar to a GRU for our particular task.

Decoders

We used several different decoder architectures to address the multi-modality of our observations.

1. Vector outputs were handled using a dense network.
2. Image outputs were handled by using deconvolutional layers to upsample z.
3. Time-series outputs were handled by using temporal deconvolutional layers to upsample z.

4.5.1 Training The Latent Space Model

We experimented with three different variations of training loss for our experiments.

State Reconstruction

This loss trained the model to maximize log-probability of ground-truth states (position and orientation) from the Novatel, given the initial states, initial observations and sequences of actions. Note that this loss does not train observation decoders.

$$\mathcal{L}_{state} = -logp_\theta(x_{1:T}|o_0, x_0, u_{1:T}) \tag{4.4}$$

Reconstruction Loss

This loss maximizes the log-probability of all observations in addition to the state.

$$\mathcal{L}_{rec} = \mathcal{L}_{state} - \sum_{m} [\beta_m log p_\theta(o^m_{1:T} | z_{1:T})] \tag{4.5}$$

Note that β_m allows us to re-weight the importance of each modality.

Contrastive Loss

Hafner et al. [19] observed that Bayes' rule can be applied to the reconstruction terms to derive a contrastive loss. Since the contrastive loss is expressed using the latent code z, a potential benefit is the ability to ignore distractors in the observation space that are irrelevant to dynamics prediction, e.g. image backgrounds. The contrastive loss is defined as:

$$\mathcal{L}_{con} = \mathcal{L}_{state} - \beta \left[log p_\theta(z_{1:T} | O_{1:T}) - \sum_{O'_{1:T}} log p_\theta(z_{1:T} | O'_{1:T}) \right] \tag{4.6}$$

where the added objective aims to maximize the log-probability of the latent code z given the corresponding observation O, while minimizing the log-probability of z given the other observations in the batch O'. Note that while the terms of the reconstruction loss can be decomposed into independent probabilities, the contrastive loss cannot. As such, there is a single weighting constant β.

4.6 Experiments and Analysis

Our experiments aim to answer the following questions:

1. Does using multi-modal sensory data lead to improved dynamics prediction in challenging environments?
2. Does varying the loss type improve model accuracy?
3. Does adding exteroception help model predictions?

	State	Reconstruction	Contrastive
KBM	1.1638	1.1638	1.1638
Image	0.5263	0.4740	0.4952
Image + Maps	0.3521	0.3386	0.3741
Time Series	0.2176	0.2285	0.1966
All	**0.1896**	**0.1674**	**0.1958**

Table 4.3: Model prediction results showing RMSE of mean state prediction for different models and loss functions.

4.6.1 Does Adding Additional Modalities Help?

We divided our dataset into a set of training trajectories and evaluation trajectories. We then trained four latent-space models with the following varied input modalities:

1. RGB image only, as in [28] (Image)
2. RGB image, heightmap and RGB map (Image + Maps)
3. IMU, shock position and wheel RPM (Time-series)
4. All Modalities (All)

We trained each model using each loss function described in the previous section. We also implemented a baseline kinematic bicycle model (KBM) that leveraged the average wheel RPM to make predictions. Table 4.3 shows the accuracy of each model as the root mean squared error (RMSE) of the mean state prediction after 20 steps of forward-simulation. The model with the lowest score for a given loss (i.e. the best set of modalities) is bolded. The model with the lowest evaluation score for a given modality (i.e. the best loss function) is colored in red. Note that since the KBM is not a latent-space model, we copy its evaluation score across all loss columns.

Overall, we can observe that adding additional modalities to the latent-space model results in improved prediction accuracy. The most noticeable improvement comes from adding top-down maps to the image-only model, yielding a 33% decrease in prediction error. From our results, we can gather that the time-series data is very important to the overall dynamics prediction. This is evidenced by the large increase in model accuracy from adding the time-series data (roughly 45% improvement from Image + Maps to All across all training procedures), and the relatively high accuracy of the time-series model. This is to be expected, as wheel RPM in particular is highly

correlated with the velocity. However, we still observe that adding the image-based modalities to the time-series model yielded roughly a 15% increase in model accuracy across all training procedures. We note that the performance of the time-series and all-modality models are essentially the same under the contrastive loss.

4.6.2 Does the Loss Type Matter?

We find that all three loss functions led to similar model accuracy. We attribute this to the fact that in many cases, the high-dimensional sensory inputs are not necessarily correlated with robot motions as they are in the environments used by Hafner et al. [19]. In these simulated environments, predicting future observations is always possible since environments consist only of the agent and a static background. However, future observations are much more difficult to predict in our scenarios. For example, if the ATV drives around a corner, it will be unable to predict observations without some form of mapping and prior traversal. As such, we observe that the auxiliary task of predicting sensory input yields little performance increase.

4.6.3 When Does Adding Exteroception Help?

We re-evaluated the usefulness of additional input modalities in more difficult driving scenarios. We compared the prediction accuracy of time-series input only models to prediction accuracy of models incorporating exteroception from the maps and images in both the original dataset, and a new dataset separately collected exclusively in more uneven terrain. We define a trajectory difficulty metric as average change in height per second, which roughly corresponds to terrain steepness and unevenness. The original and new dataset had trajectory difficulties of $0.0866 m/s$ and $0.2253 m/s$, respectively. 87% of the trajectories in the new dataset were more difficult than the figsn difficulty in the original dataset. Table 4.4 shows the prediction accuracy of time-series and all-modality models on each evaluation set, as well as the percent improvement.

Overall, we observe that the additional vision-based modalities are more beneficial in the more challenging scenarios. This makes sense, as the difficulty of the terrain increases, it becomes increasingly difficult to accurately predict the future using proprioception alone.

Dataset	Prop. Error	Prop. + Ext. Error	Improvement
Original	0.2176	0.1896	13%
Difficult	0.7313	0.5394	26%

Table 4.4: Comparison of Proprioception-only (Prop.) and Proptioception + Exteroception (Prop. + Ext.) Models on the Original and More Difficult Evaluation Datasets

4.7 Implementation Details

4.7.1 Network Architecture and Training Procedure

In this section, we elaborate more on our network architectures and training procedures. The algorithm for generation state and observation predictions is presented in Algorithm 2. The general algorithm for encoding and decoding both image and time-series data is presented in Algorithms 3-6. The temporal downsample block follows the implementation of WaveNet [39] (i.e. gated, dilated, causal convolutions). However, as there is no temporal order to the latent code, temporal upsampling is handled simply by 1D convolution and upsampling along the time dimension. We present the full list of neural network architectures in Tables 4.5-4.10. We present our training hyperparameters in Table 4.12. Since we evaluate multiple different loss types, we add an additional column denoting which experiments used which hyperparameters (with 'R' standing for reconstruction and 'C' for contrastive).

4.7.2 Algorithm for T-SNE Clustering

In this section, we describe in more detail our algorithm for performing time-series clustering. This is presented in Algorithm 7.

4.7.3 T-SNE figures For Dynamical Variation Experiment

The full set of t-SNE figures and clusters from our motivational experiment are provided in Figures 4.4 and 4.5, respectively. The hyperparameters for the experiment are provided in Table 4.13.

Algorithm 2: Latent Model Forward Pass

Input: Modality set M, initial state x_0, initial observations $\{o_0^m, \forall m \in M\}$, action sequence $a_{1:T}$, modality prediction set $\tilde{M}$. Encoders $e_\psi^m, \forall m \in M$, Decoders $d_\psi^m, \forall m \in \tilde{M}$, latent model $f_\theta(z, a)$, action encoder $g_\psi(a)$, state decoder d_ψ^{state}

Output: State predictions $\tilde{x_{1:T}}$, observation predictions $\{o_{1:t}^m, \forall m \in \tilde{M}\}$

for $m \in M$ **do**
  $p^m(z) \leftarrow e_\psi^m(o_0^m)$ ◁ Encode each observation into $\mathcal{Z}$
end
$z_0 = \text{aggregate}(\{p^m(z), \forall m \in M\})$ ◁ Use Deepsets [67] or Product of Experts [23] to get single z
for $t \in 1 : T$ **do**
  $z_t = f_\theta(z, g_\psi(a_{t-1}))$ ◁ Embed action and predict next latent state
  $x_t = d_\psi^{state}(z_t)$ ◁ Decode state
  for $m \in \tilde{M}$ **do**
    $o_{t+1}^m = d_\psi(z_t)$ ◁ Decode observation
  end
end
return $x_{1:T}, \{o_{1:t}^m, \forall m \in \tilde{M}\}$

Algorithm 3: Upsample Block

Input: Image input x, upsample factor s, convolution kernel K, activation function f

Output: Upsampled image output $\tilde{x}$

$x \leftarrow \text{linear interpolate}(x, s)$
$x \leftarrow x * K$
$x \leftarrow f(x)$
return x

Algorithm 4: Downsample Block

Input: Image input x, downsample factor s, convolution kernel K, activation function f

Output: Downsampled image output $\tilde{x}$

$x \leftarrow x * K$
$x \leftarrow f(x)$
$x \leftarrow \text{linear interpolate}(x, s)$
return x

Algorithm 5: CNN Encoder

Input: Image input x, downsample blocks D_ψ, MLP f_θ
Output: Latent distribution $p(z)$
for d_ψ *in* D **do**
  $x \leftarrow d_\psi(x)$ ◁ using Algorithm 4 or [39]
end
$x \leftarrow \text{flatten}(x)$ ◁ Flatten x to 1D
$\mu, \sigma \leftarrow f_\theta(x)$
return $\mathcal{N}(\mu, \sigma)$

Algorithm 6: CNN Decoder

Input: Latent vector z, upsample blocks U_ψ, MLP f_θ
Output: Image reconstruction $\tilde{X}$
$x \leftarrow f_\theta(x)$
$x \leftarrow \text{pad\_front}\ (x, 2)$ ◁ $x \in \{1 \times 1 \times |x|\}$
for u_ψ *in* U **do**
  $x \leftarrow u_\psi(x)$ ◁ using Algorithm 3
end
return x

Layer	Input Dim	Output Dim	Kernel Size	Activation
Downsample 1	$3 \times 128 \times 128$	$4 \times 64 \times 64$	3×3	ReLU
Downsample 2	$4 \times 64 \times 64$	$8 \times 32 \times 32$	3×3	ReLU
Downsample 3	$8 \times 32 \times 32$	$16 \times 16 \times 16$	3×3	ReLU
Downsample 4	$16 \times 16 \times 16$	$32 \times 8 \times 8$	3×3	ReLU
Flatten	$32 \times 8 \times 8$	2048	-	-
MLP	2048	$2 \times \lvert\mathcal{Z}\rvert$	-	Tanh
Gaussian	$2 \times \lvert\mathcal{Z}\rvert$	$\mathcal{N} \in \mathcal{Z}$	-	-

Table 4.5: Visual CNN Encoder Architecture

Layer	Input Dim	Output Dim	Kernel Size	Activation
Downsample 1	$\{1,3\} \times 64 \times 64$	$4 \times 32 \times 32$	3×3	ReLU
Downsample 2	$4 \times 32 \times 32$	$8 \times 16 \times 16$	3×3	ReLU
Downsample 3	$8 \times 16 \times 16$	$16 \times 8 \times 8$	3×3	ReLU
Downsample 4	$16 \times 8 \times 8$	$32 \times 4 \times 4$	3×3	ReLU
Flatten	$32 \times 4 \times 4$	512	-	-
MLP	512	$2 \times \lvert\mathcal{Z}\rvert$	-	Tanh
Gaussian	$2 \times \lvert\mathcal{Z}\rvert$	$\mathcal{N} \in \mathcal{Z}$	-	-

Table 4.6: Local Map CNN Encoder Architecture

Layer	Input Dim	Output Dim	Size	Dilation	Activation
Downsample 1	$\{4,9\} \times 20$	$\{4,9\} \times 20$	2	2	[39]
Downsample 2	$\{4,9\} \times 20$	$\{4,9\} \times 20$	2	4	[39]
Downsample 3	$\{4,9\} \times 20$	$\{4,9\} \times 20$	2	8	[39]
Downsample 4	$\{4,9\} \times 20$	$\{4,9\} \times 20$	2	16	[39]
Flatten	$\{4,9\} \times 20$	$\{80,180\}$	-	-	-
MLP	$\{80,180\}$	$2 \times \lvert\mathcal{Z}\rvert$	-	-	Tanh
Gaussian	$2 \times \lvert\mathcal{Z}\rvert$	$\mathcal{N} \in \mathcal{Z}$	-	-	

Table 4.7: Temporal CNN Encoder Architecture

Layer	Input Dim	Output Dim	Kernel Size	Activation
MLP	$\lvert\mathcal{Z}\rvert$	128	-	Tanh
Unflatten	128	$128 \times 1 \times 1$	-	-
Upsample 1	$128 \times 1 \times 1$	$32 \times 4 \times 4$	3×3	ReLU
Upsample 2	$32 \times 4 \times 4$	$16 \times 8 \times 8$	3×3	ReLU
Upsample 3	$16 \times 8 \times 8$	$8 \times 16 \times 16$	3×3	ReLU
Upsample 4	$8 \times 16 \times 16$	$4 \times 32 \times 32$	3×3	ReLU
Upsample 5	$4 \times 32 \times 32$	$3 \times 128 \times 128$	3×3	ReLU

Table 4.8: Visual CNN Decoder Architecture

Layer	Input Dim	Output Dim	Kernel Size	Activation
MLP	$\|\mathcal{Z}\|$	128	-	Tanh
Unflatten	128	$128 \times 1 \times 1$	-	-
Upsample 1	$128 \times 1 \times 1$	$32 \times 4 \times 4$	3×3	ReLU
Upsample 2	$32 \times 4 \times 4$	$16 \times 8 \times 8$	3×3	ReLU
Upsample 3	$16 \times 8 \times 8$	$8 \times 16 \times 16$	3×3	ReLU
Upsample 4	$8 \times 16 \times 16$	$4 \times 32 \times 32$	3×3	ReLU
Upsample 5	$4 \times 32 \times 32$	$3 \times 64 \times 64$	3×3	ReLU

Table 4.9: Local Map CNN Decoder Architecture

Layer	Input Dim	Output Dim	Kernel Size	Activation
Unflatten	$\|\mathcal{Z}\|$	$1 \times \|\mathcal{Z}\|$	-	-
Upsample 1	$1 \times \|\mathcal{Z}\|$	2×64	2	Tanh
Upsample 1	2×64	4×32	2	Tanh
Upsample 1	4×32	8×16	2	Tanh
Upsample 1	8×16	16×8	2	Tanh
Upsample 1	16×8	$20 \times \{4, 9\}$	2	Tanh

Table 4.10: Temporal CNN Decoder Architecture

Layer	Input Dim	Output Dim	Activation
Action Encode 1	2	16	Tanh
Action Encode 2	16	16	Tanh
GRU	$(128, 23)$	$128, 128$	-
State Decode 1	128	128	Tanh
State Decode 2	128	$\mathcal{N} \in \mathbb{R}^7$	-

Table 4.11: Latent Model Architecture

Algorithm 7: T-SNE Clustering

Input: Dataset $\mathcal{D}$ (binned by velocity), consisting of states $s_{1:T}$ and actions $a_{1:T}$, time window k, numbers of clusters n,

Output: Cluster mappings $c_{1:T}$ and t-SNE embeddings $z_{1:T}$ for each timestep

$f_t = flatten(a_{t:t+k}), \forall t$ ◁ Get features for each state by flattening actions over the window

$c_{1:k} = kmeans(f_{1:T})$ ◁ Perform k-means to get cluster centers

$\mathcal{T}_t = (s_t)^{-1}, \forall t$ ◁ Compute the transform to start all state differences at 0,0

$\Delta s_{1:T} = \mathcal{T}_t(s_{t+k} - s_t), \forall t$ ◁ Compute state differences for all states

$z_{1:T} = tsne(\Delta s_{1:T})$ ◁ Perform t-SNE on the state differences

Hyperparameter	Value	Experiment
Optimizer	Adam [29]	All
Learning Rate	$1e-3$	All
Epochs	5000	All
Batch Size	64	All
Gradient Steps Per Epoch	10	All
Gradient Norm Clip	100.0	All
Train Timesteps	20	All
RGB Image Loss Scale	100	R
RGB Map Loss Scale	100	R
Heightmap Loss Scale	1	R
IMU Loss Scale	0.1	R
Wheel RPM Loss Scale	0.1	R
Contrastive Scale	10.0	C
EMA τ	0.05	C

Table 4.12: Training Hyperparameters

Hyperparameter	Value
# Subsequences	10000
Sequence length	10
# Clusters	10
# Velocity Bins	5
Clustering Distance Metric	Euclidean

Table 4.13: Motivational Experiment Hyperparameters

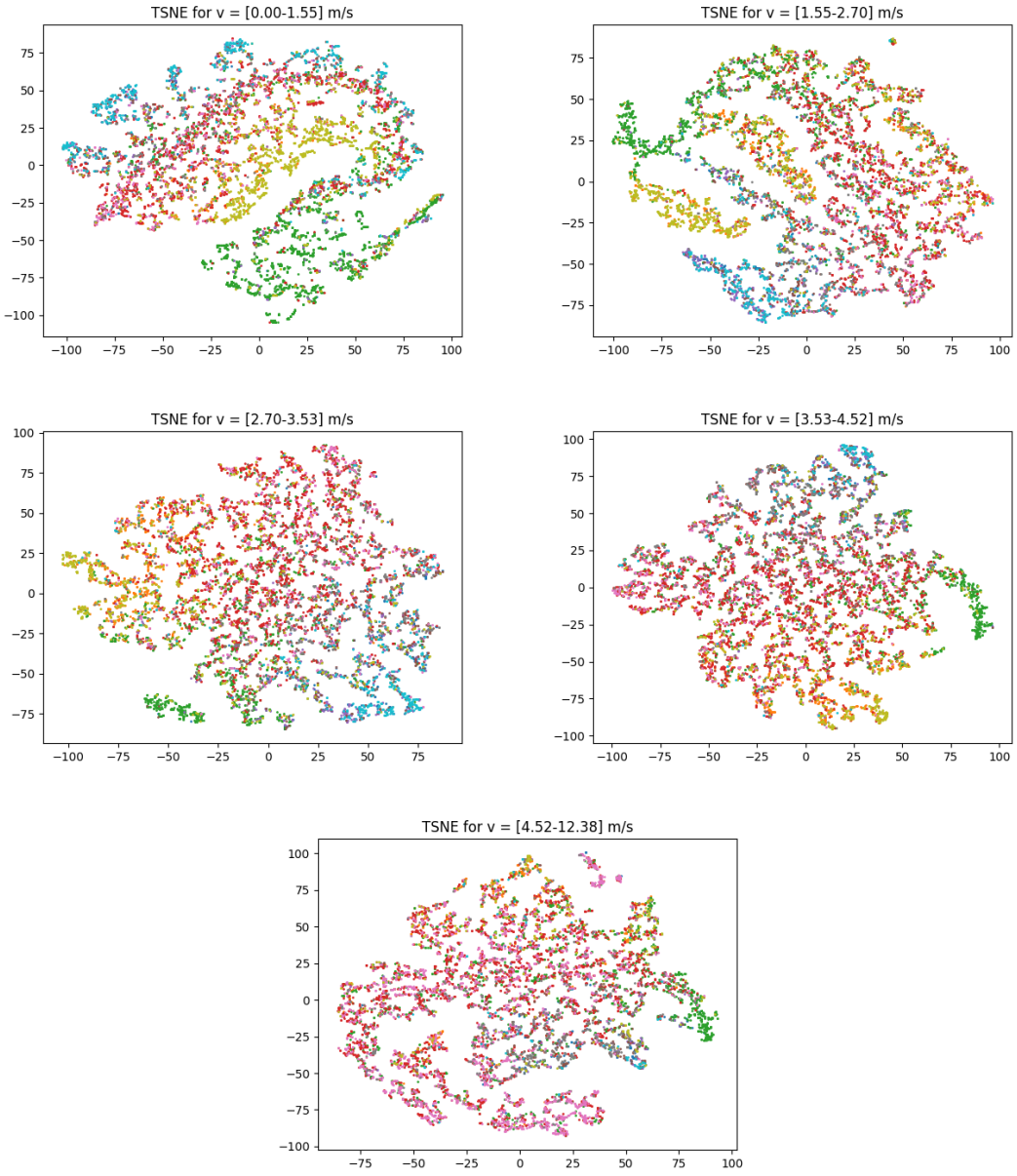


Figure 4.4: The t-SNE visualizations for all five velocity bins.

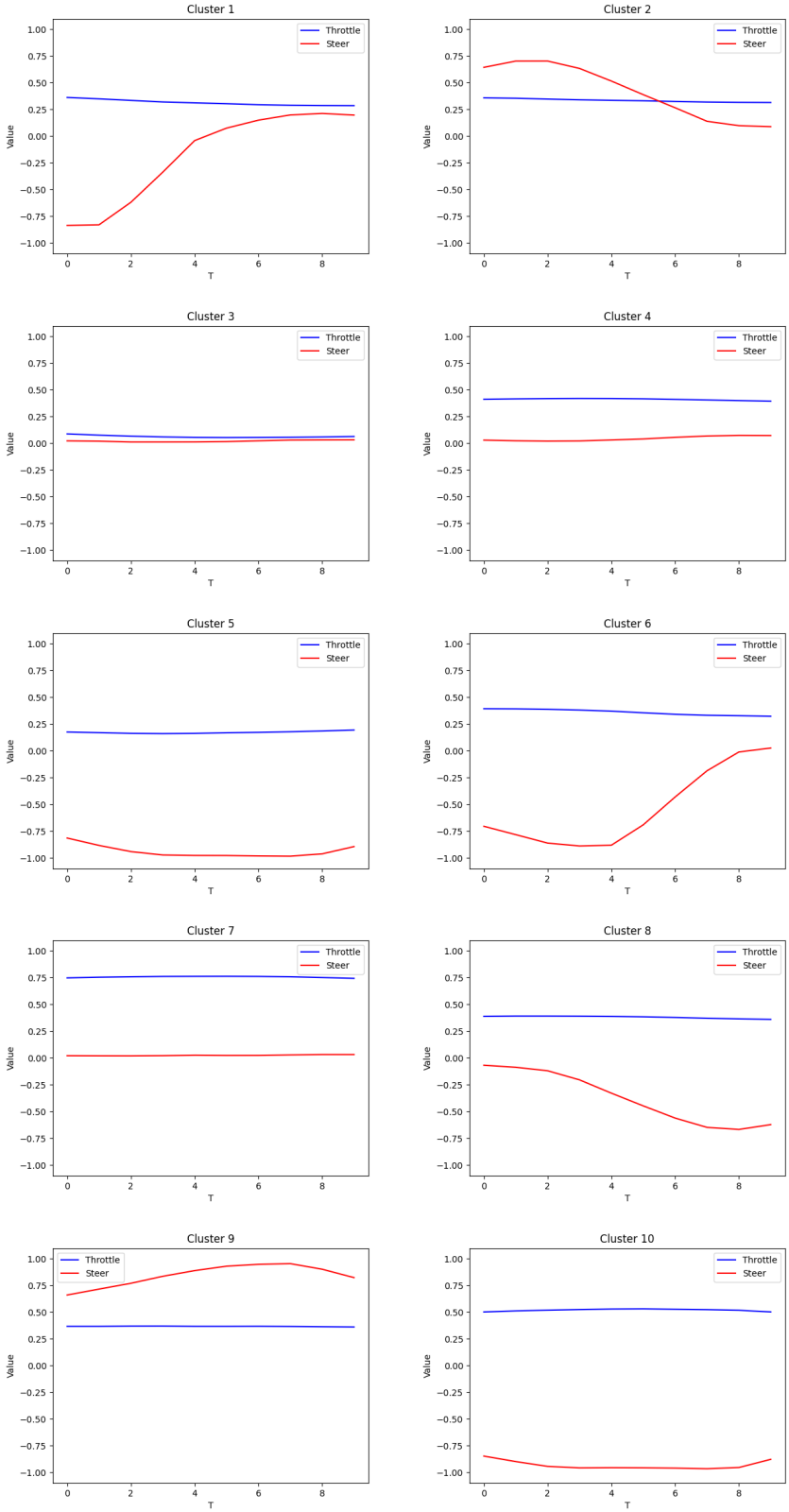


Figure 4.5: The cluster centroids for the motivational experiment

Chapter 5

Learning Cost Functions

5.1 Motivation

While learning more accurate models of vehicle dynamics may tell a vehicle *how* to drive, it is critical that the vehicle also know *where* to drive. We assert that simple obstacle avoidance is not sufficient for off-road driving for the following reasons:

1. Many common objects such as tall grass and low bushes may appear as obstacles despite being traversable.
2. Some objects may be more costly to traverse than others (i.e. tall grass and rocks should be costed differently).

5.2 Problem Formulation

In order to obtain more effective representations of cost for our problem, we choose to adopt an inverse reinforcement learning (IRL)-based method to produce costmaps from tensors of map features obtained from lidar pointclouds. The supervision for these costmaps is derived from a set of expert trajectories obtained from a human driving the ATV through certain scenarios. These trajectories can be thought of as samples from an expert policy. The objective of IRL is to find a cost function whose corresponding optimal policy closely resembles the expert.

5.3 Dataset Collection for Inverse RL

Unfortunately, the data collected in TartanDrive [51] is unsuitable for inverse RL for several reasons. (1) Since TartanDrive was focused on learning models, trajectories were often collected at a relatively slow speed with significant changes in steering, which cause trajectories to be highly sub-optimal. (2) The Velodyne UltraPucks were installed on the Yamaha ATV after TartanDrive data was collected. As such, the sensing horizon in TartanDrive is comparatively small (up to 10m in front of the vehicle). This severely limits the speed that is safe of the vehicle to traverse. In contrast, the lidar maps produced by the Velodyne have a horizon of 60m.

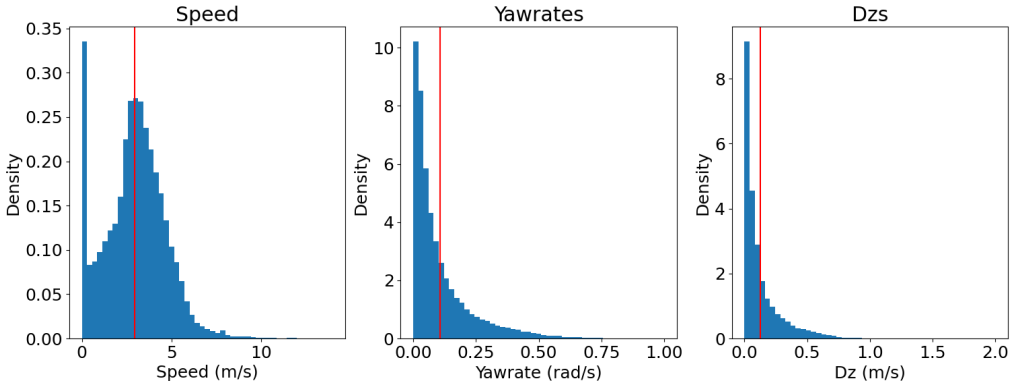
In order to have expert demonstrations for IRL, we collected a dataset of roughly an hour of aggressive off-road driving where human tele-operator was instructed to drive as aggressively as possible while still maintaining safety. We then quantify the difficulty of the resulting dataset using the following metrics:

1. Speed: The magnitude of the body velocity at each timestep in the dataset.
2. Yawrate: The magnitude of the z-component of the body twist at each timestep in the dataset.
3. Dz: The magnitude of the change in the vehicle's z position between two consecutive timesteps.

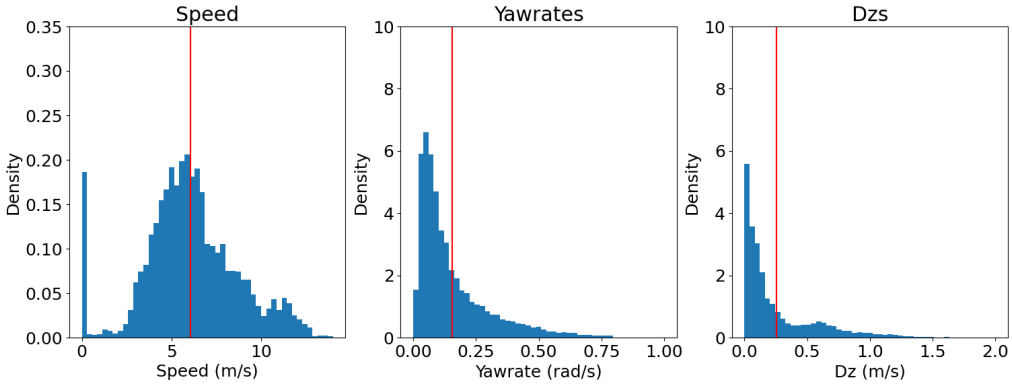
A comparison between TartanDrive [51] and the IRL dataset is provided in Figure 5.1. Overall, the IRL dataset has higher mean difficulty across all metrics, and also exhibits a wider distribution of difficulty across all metrics, as well.

5.3.1 Extracting Goals from Undirected Driving Data

A minor, but important caveat to the data collection process is that the expert demonstrations in the IRL dataset are *undirected*. That is, the human tele-operator was not instructed to navigate to any particular goal point. While this approach to data collection may render the dataset not applicable to techniques such as global planning over several hundreds of meters or more, given the focus on local planning, we found that the benefits of this data collection method outweighed the costs. The most significant benefit was that demonstrations could be collected much more quickly,



(a) Tartandrive difficulty



(b) IRL dataset difficulty

Figure 5.1: Comparison of dataset difficulties between our dataset and TartanDrive. Our IRL dataset has higher mean difficulty and a wider distribution of difficulty, as well.

Feature	Calculation
Height Low	$\min_p[p_z], \quad \forall p \vert p_z < t_z + k_{overhang}$
Height Mean	$\frac{1}{\vert P \vert}\sum_p[p_z], \quad \forall p \vert p_z < t_z + k_{overhang}$
Height High	$\max_p[p_z], \quad \forall p \vert p_z < t_z + k_{overhang}$
Height Max	$\max_p[p_z], \quad \forall p$
Terrain (T)	See Algorithm 8
Slope	$0.5\ (\vert\frac{\partial}{\partial x}T\vert + \vert\frac{\partial}{\partial y}T\vert)$
Diff	Height high - Terrain
SVD1	$\frac{\lambda_1-\lambda_2}{\lambda_1}$
SVD2	$\frac{\lambda_2-\lambda_3}{\lambda_1}$
SVD3	$\frac{\lambda_3}{\lambda_1}$
Roughness	$\frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$
Unknown	$\mathbf{1}\left[\vert P \vert = 0\right]$

Table 5.1: List of grid map features, and their calculations given the points in a cell

could be adapted rapidly if interesting scenarios presented themselves during the data collection process. Furthermore, the lack of pre-defined goal points can be easily ameliorated via a simple, offline goal-labeling process. Given a fixed time horizon H, a goal state x_g can be determined by simply using the robot state H steps in the future:

$$x_t^G = x_{t+H} \tag{5.1}$$

This goal-relabeling process has been shown to be effective in reinforcement learning problems [4]. Furthermore, goal-relabeling with a small enough H can avoid an issue in IRL noted by Ratliff et al. [42] where experts may choose paths in a sub-optimal homotopy class, especially with respect to obstacles.

5.3.2 Lidar Feature Maps

Key to the IRL problem is informative state features upon which learning can be performed. For the purpose of this work, we implemented a lidar perception module that takes in registered lidar pointclouds and produces a tensor of geometric map features. Given this map M, features can be determined for a given state x^N by projecting its position p^N onto the map features.

An overview of our set of map features is provided in Table 5.1, and our lidar

mapping algorithm is presented in detail in Algorithm 8.

Algorithm 8: Lidar Mapping Algorithm

Input: Buffer of registered pointclouds P_b, pointcloud skip k_p, Map origin (o_x, o_y), Map size (l_x, l_y), Map resolution r, overhang limit $k_{overhang}$
Output: Terrain feature tensor X

$n_x = \lfloor \frac{l_x}{r} \rfloor$
$n_y = \lfloor \frac{l_y}{r} \rfloor$
$X = \mathbf{0}^{n_x \times n_y \times 12}$ ◁ Initialize map tensor
$P = \sum_{i=0}^{|P_b|/k_p} P_{i*k_p}$ ◁ Aggregate pointclouds from buffer
for $i = 0 \dots n_x$ **do**
  for $j = 0 \dots n_y$ **do**
    $P_m = \{p, \forall p \in P | (\frac{P_x - o_x}{r} = i) \wedge (\frac{P_y - o_y}{r} = j)\}$ ◁ Get all points in a given column
    $X[i, j, 0] = min_{p \in P_m}[p_z]$ ◁ Get min height
    $X[i, j, 1] = max_{p \in P_m}[p_z]$ ◁ Get max height
  end
end
$X[:, :, 4] = G * inflate(X[:, :, 0])$ ◁ Generate terrain estimate by inflating and low-pass filtering min height
$X[:, :, 5] = S_x * |X[:, :, 2]| + S_y * |X[:, :, 2]|$ ◁ Get terrain slope via derivative filter
for $i = 0 \dots n_x$ **do**
  for $j = 0 \dots n_y$ **do**
    $P_m = \{p, \forall p \in P | (\frac{p_x - o_x}{r} = i) \wedge (\frac{p_y - o_y}{r} = j) \wedge (p_z < X[i, j, 2] + k_{overhang})\}$ ◁ Filter overhanging points
    $X[i, j, 2] = \max_z [p_z \forall p \in P_m]$ ◁ Get the max height of the cell, saturating at the overhang limit
    $X[i, j, 3] = \frac{1}{|P_m|} \sum_{p \in P_m} [p_z]$ ◁ Get the mean height of the cell
    $X[i, j, 6] = X[i, j, 2] - X[i, j, 4]$ ◁ Get the height of the cell relative to terrain $
    $\lambda_1, \lambda_2, \lambda_3 = SVD(P_m)$ ◁ Get the SVD decomposition of the cell points
    $X[i, j, 7] = \frac{\lambda_1 - \lambda_2}{\lambda_1}$ ◁ Get SVD1
    $X[i, j, 8] = \frac{\lambda_2 - \lambda_3}{\lambda_1}$ ◁ Get SVD2
    $X[i, j, 9] = \frac{\lambda_3}{\lambda_1}$ ◁ Get SVD3
    $X[i, j, 10] = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$ ◁ Get roughness
    $X[i, j, 11] = \mathbf{1}[|P_m| = 0]$ ◁ Get unknown
  end
end
return X

5.4 Methodology

5.4.1 MaxEnt IRL

Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) is a popular framework for extracting cost functions for a Markov Decision Process (MDP) from large corpora of human demonstrations [72]. MaxEnt IRL builds off of the results shown by Abbeel and Ng [2] that one can obtain a policy close to the expert policy by matching feature expectations. However, the baseline IRL problem is ill-posed, as there are an infinite number of cost functions that can explain a set of expert trajectories. Ziebart et al. [72] propose using the principle of maximum entropy [26] to address the ill-posed nature of unregularized IRL. They show that the gradient for maximizing the likelihood of the expert trajectories τ_E under the learned reward

function with entropy regularization can be computed as Equation 5.2.

$$\nabla_\theta \sum_{\tau_E \in \mathcal{D}_E} L(\tau_E|\theta) = \tilde{f} - \sum_{s_i} D^L_{s_i} f_{s_i} \tag{5.2}$$

$$\nabla_f \sum_{\tau_E \in \mathcal{D}_E} L(\tau_E|\theta) = \sum_{s_i} \left[D^E_{s_i} - D^L_{s_i}\right] \tag{5.3}$$

Here, f_{s_i} are the features associated with state s_i, $D^L_{s_i}$ is the learner's state visitation distribution, and $\tilde{f}$ is the expert's expected feature counts ($\tilde{f} = \sum_{s_i} D^E_{s_i} f_{s_i}$). Note that the computation of this gradient requires enumeration over the state-space of the MDP.

Performing gradient ascent using this objective has been shown to produce strong results for reward functions linear in state features (of the form $r = \theta^T f$). Wulfmeier et al. [62] extend this formulation by providing a gradient of the MLE objective with respect to state features (Equation 5.3). This gradient allows for the training of deep neural networks via IRL (MEDIRL).

Algorithm 9: Training Step for Fast MEDIRL

Input:
Dataset $\mathcal{D}$ of expert trajectories $\boldsymbol{\tau_E}$,
Map features $\mathbf{M}$,
Goal weight κ,
FCN Ensemble $F_\theta(M) : \mathcal{R}^{W\times H\times D} \rightarrow \mathcal{R}^{B\times W\times H}$,
MPPI $MPPI(x_s, x_g, C, \lambda) : \mathcal{X} \rightarrow (\eta_n, \tau_n) \times N$

while *not converged* **do**

- $\tau^E, M \sim \mathcal{D}$ ◁ Sample from dataset
- $x_0 = \tau^E_0, x_g = \tau^E_{t-1}$ ◁ set start/goal
- $f_\theta \sim \boldsymbol{F_\theta}$ ◁ sample FCN from ensemble
- $C = f_\theta(M_0)$ ◁ Compute costmap from FCN
- $\boldsymbol{\tau^L}, \boldsymbol{\eta^L} = MPPI(x_0, x_g, C, \kappa)$
- $D^E = SVF(\tau_E, 1)$ ◁ Compute state visitations
- $D^L = SVF(\tau^L, \eta^L)$ via Algorithm 10
- $\nabla_z J = D^E - D^L$ ◁ Gradient via [62]
- backprop$(\nabla_z J, f_\theta)$ ◁ Update FCN grads
- $\theta \leftarrow$ Adam$(\theta - \nabla_\theta J)$ ◁ Update via [29]

end

5.4.2 Optimizing Over Cost Functions with MPPI

Key to computing the MaxEnt IRL gradient is an algorithm that can quickly compute an optimal policy for a given cost function. In recent years, it has been popular to use sampling-based MPC methods, due to their simplicity, ease of parallelization and ability to make minimal assumptions as to the form of the dynamics model and cost function. In particular, MPPI [60] has been widely used, demonstrating strong empirical results in several domains including navigation [1, 59] and manipulation [7, 38].

MPPI can be thought of as a gradient-free optimizer that attempts perform importance sampling of an optimal distribution of controls where control sequences $u_{1:T}$ have probability inversely proportional to the exponential of their cost (Equation 5.4). It is assumed that cost can be calculated by rolling out an action sequence u through a (stochastic) dynamics function f with initial state x_0, and applying a cost function to the resulting trajectory τ.

$$q^*(u_{1:T}) \propto exp(-C(\tau))p(u_{1:T}), \tau \sim x_0, f \tag{5.4}$$

This distribution q^* is assumed to have the form of a control sequence $u^*_{1:T}$ with independent Gaussian noise. Given these assumptions, $u^*_{1:T}$ can be estimated by importance sampling with a proposal distribution $p(u)$ (Equation 5.5).

$$u^* = \sum_u q^*(u)u = \sum_u p(u)\frac{q^*(u)}{p(u)}u = \mathbb{E}_{u\sim p(u)}[\frac{q^*(u)}{p(u)}u] \tag{5.5}$$

Fortunately, the likelihood ratio $\frac{q^*(u)}{p(u)}$ is easily computable given the assumption as to the form of the optimal distribution (Equation 5.4). This results in the update rule:

$$u^*_{1:T} = \sum_u [exp(-C(\tau))u], u \sim \tilde{u}_{1:T} \tag{5.6}$$

where $\tilde{u}_{1:T}$ is a running estimate of the optimal distribution. In practice, the first action u_1 is then executed on the robot, and the running estimate is then shifted to warm-start the optimization process for the next timestep.

MPPI with Gaussian Walks

In practice, MPPI can be prone to getting stuck in local optima if the noise parameters are not tuned appropriately. Furthermore, the use of independent Gaussian noise can produce highly noisy action sequences, which can be damaging to robot actuators due to high jerk. In order to address these issues, we choose to sample action sequences via a Gaussian random walk (Equations 5.7, 5.8).

$$u_{1:T} = \tilde{u}_{1:T} + w_{1:T} \tag{5.7}$$

where

$$w_1 = 0, w_k = w_{k-1} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma) \tag{5.8}$$

Since all w_k are zero-mean, Equation 5.5 remains an unbiased estimator of $u^*_{1:T}$. Given the ability to maintain a significant variance over $u_{1:T}$ while maintaining a low variance in the random walk, action sequences produced from Gaussian random walks can be much smoother while maintaining sufficient coverage of the action space.

5.4.3 Computing State Visitations

Given the resulting continuous trajectory distribution from MPPI, we can project the resulting trajectories into the map through Algorithm 10. Note that since trajectories from MPPI are weighted by their probability under the optimal distribution, we can use multiple MPPI trajectories, weighted by their likelihood under q^*.

5.5 Uncertainty Estimation for Costmaps

While in theory the implicit regularization of MaxEnt IRL should result in consistent solutions, we observe variance in our trained costmaps in practice. In order to combat this, we train an ensemble of costmaps and combine their predictions using conditional value-at-risk (CVaR).

While originally used in econometrics applications, there has been interest from the robotics community in using CVaR as a risk metric for reasoning about distributions

Algorithm 10: Computation of State Visitation Frequencies (SVF) from Weighted Trajectories

Input: Initial State x_0, Weights $\boldsymbol{\eta}$, Trajectories $\boldsymbol{\tau}$, Map resolution M_{res}
Output: State Visitation Frequencies D
for $\tau_n, \eta_n \in \boldsymbol{\tau}, \boldsymbol{\eta}$ **do**
  for $x_t \in \tau_n$ **do**
    $i, j \leftarrow \lfloor p(x_t)/M_{res} \rfloor$
    $D[i,j] \leftarrow D[i,j] + \eta_n$
  end
end
$D \leftarrow \frac{D}{\sum_{i,j} D[i,j]}$
return D

of cost [9, 12, 15, 35] for its ability to capture more accurately the nuances of long-tailed and multimodal distributions. Intuitively, CVaR_ν can be thought of as the the mean value of a distribution, when only considering the portion of the distribution exceeding a given quantile (a.k.a. Value at Risk (VaR)) $\nu \in [0, 1]$ (described formally in Equation 5.9) [43].

$$\text{CVaR}_\nu = \int_{f(x)>\text{VaR}_\nu}^{\infty} f(x)p(x)dx \tag{5.9}$$

Note that CVaR can also be used to capture the behavior of the lower tail of the distribution by taking values *below* a given quantile [9]. Via a small abuse of notation, we will refer to this mode of CVaR as CVaR_ν for $\nu \in [-1, 0]$ (Equation 5.10). This gives us a range of $\nu \in [-1, 1]$ that we can use to smoothly vary the risk-tolerance of the vehicle. Note that CVaR_0 corresponds to taking the mean of the ensemble (and being neutral to risk), similar to Ratliff et al. [42].

$$\text{CVaR}_{-\nu} = \int_{-\infty}^{f(x)<\text{VaR}_{1-\nu}} f(x)p(x)dx \tag{5.10}$$

Algorithm 11: Inference Step for Fast MEDIRL

Input:
Map features M,
FCN Ensemble $F_\theta(M) : \mathcal{R}^{W \times H \times D} \rightarrow \mathcal{R}^{B \times W \times H}$,
Risk level ν

$C_\nu = \mathbf{0}^{m \times n}$ ◁ Initialize empty costmap
$\boldsymbol{C} = \boldsymbol{F_\theta}(M)$ ◁ Create costmap for each FCN
for $i, j \in ([0 \dots m] \times [0 \dots n])$ **do**
  $\boldsymbol{c} = \boldsymbol{C}[:, i, j]$ ◁ Get each FCN's cell output
  $C_\nu[i, j] = CVaR_\nu(\boldsymbol{c})$ ◁ Compute CVaR
end
return C_ν

5.6 Algorithm Overview

We thus arrive at a practical algorithm for IRL that is risk-aware. At a high level, our algorithm can be summarized as follows:

1. Extract a fixed time-window of expert trajectory $\tau^E_{t:t+H}$ (7.5s or 75 timesteps for our experiments), starting at the time the map features M_t were taken.
2. Set the first expert state τ^E_0 as the MDP start state.
3. Set the final expert state τ^E_T as the MDP goal state.
4. Obtain a costmap by randomly selecting a costmap predictor from the ensemble and running it on the lidar features at timestep 0.
5. Set state transition function to be KBM dynamics (2.3).
6. Set cost function as a weighted combination of the costmap C_t and final-state distance-to-goal (Eq. 5.11).
7. Compute a state visitation distribution for an optimal policy by optimizing the cost function via MPPI
8. Use the MPPI state visitation frequencies to update the selected costmap network via IRL.

$$J(\tau) = \sum_{s_i \in \tau} \Big[C[p(x_i)] \Big] + \kappa ||x_T - x_g||_2 \tag{5.11}$$

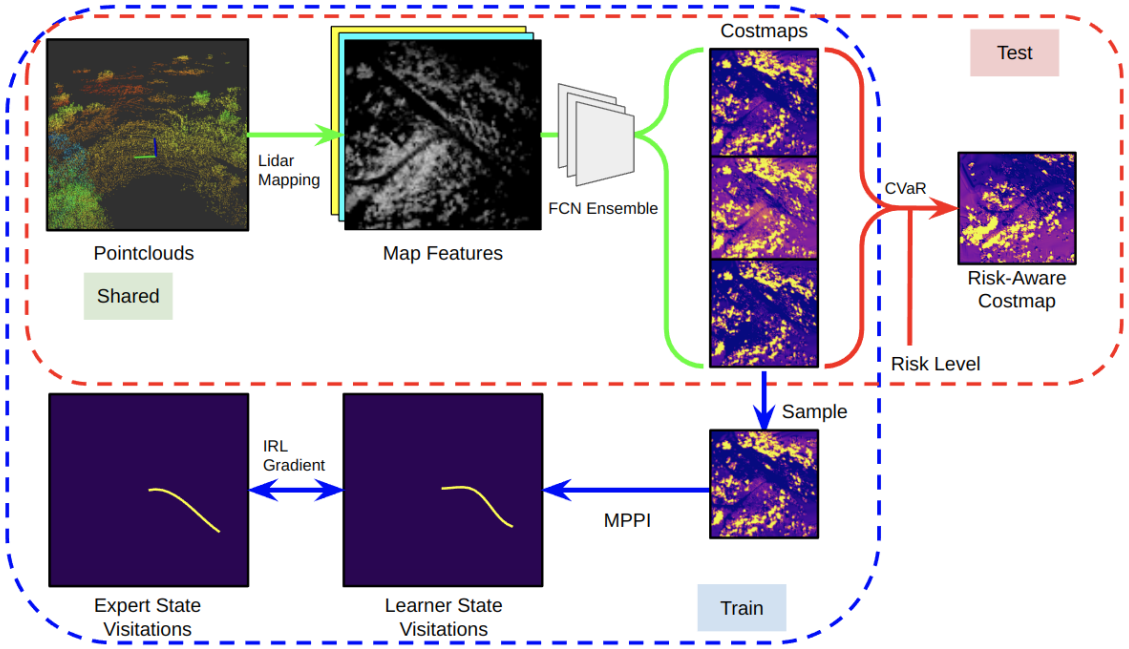


Figure 5.2: An overview of our algorithm. We first process lidar scans into a tensor of map features. We then use an ensemble of FCNs to create a set of costmaps. In order to train these networks, we can sample from our set of costmaps, solve the resulting MDP with MPPI, and use the resulting state distribution to supervise our networks with inverse RL. At test time, the costmaps can be aggregated together using CVaR to create a costmap associated with a given risk level.

Function Approximator	MHD
Occupancy	3.220 ± 0.027
Linear	1.847 ± 0.033
Linear Sigmoid	1.955 ± 0.046
Resnet	1.973 ± 0.022
Resnet Sigmoid	$\mathbf{1.794 \pm 0.033}$

Table 5.2: Performance of various function approximators on reproducing expert behavior

5.7 Offline Performance

We first evaluate the performance of our IRL-based method's ability to reconstruct expert paths. Similar to prior work, [57, 68, 71], we use modified Hausdorff distance [14] between the expert path and the optimal path from MPPI as our primary performance metric.

As a baseline, we compare to a simple occupancy-based baseline that first identifies obstacle cells as cells whose maximum height exceeds the terrain estimate by a certain threshold, and then inflates those obstacles to ensure that the vehicle maintains a reasonable distance from obstacles.

We evaluate a number of different costmap architectures for IRL. Namely, we compare a Resnet-5 architecture and a linear baseline. We also explore whether it helps to squash the cost outputs via a sigmoid activation. The resulting MHD over a held-out test set (consisting of around 2000 datapoints) are shown in Table 5.2. Each method was evaluated over three random seeds.

Overall, we observe a significant increase in performance over the occupancy-based baseline. This is to be expected, as the higher-capacity function approximator is trained directly to maximize likelihood of expert paths. Additionally, we observe a small, but statistically significant performance improvement when using a deep network.

5.8 Real-World Navigation Trials

We evaluated our method on several challenging navigation courses at Gascola. There were four courses used, totalling roughly $6km$ in length. Our courses were defined

Course	IRL Interventions	Baseline Interventions
Red	**0**	6
Blue	**3**	7
Green	**3**	11
Purple	**3**	7

Table 5.3: Interventions for each method.

by a series of GPS waypoints. Given the sensing horizon ($40m$ in all directions) and focus on local planning, waypoints were spaced $50m$ apart. In all experiments, all components of our navigation stack were identical, except for the costmap generation method. A more detailed description of the course used for the navigation experiments and representative terrains is presented in Figure 5.3.

As we are interested in evaluating the effect of the costmap on overall navigation performance, we report the number of interventions from a human safety driver as our primary metric. We also report the total distance traveled and average speed, excluding distance traveled while intervening. The safety driver was instructed to intervene in the following two cases:

1. If the ATV was going to drive into an obstacle
2. If the ATV drove past a waypoint without going within a $4m$ radius of it

On the full $6km$ course, we compare the performance of IRL to occupancy-based costmaps (Table 5.3). Overall, we observe a 70% reduction in interventions as a result of using the IRL costmaps.

Additional results for different settings of CVaR are provided in Table 5.4 and Figure 5.4. Overall, we are again able to observe significant improvement over the baseline when using IRL-based costmaps. All IRL-based methods had roughly the same number of interventions, but exhibited different navigation behaviors, especially with respect to tall grass. In general, more conservative settings of CVaR (> 0.8), caused the ATV to avoid patches of moderate-height grass when possible, while more aggressive settings of CVaR (< -0.8), caused the ATV to drive through taller patches of grass. As one would expect, average distance traveled increased with CVaR_ν, as there were monotonically more high-cost regions to avoid. The occupancy-based method traveled the shortest amount of autonomous distance as a result of having more human interventions. Interestingly, we observed that the difference between the speeds and distance traveled for $\text{CVaR}_{-0.9}$ and CVaR_0 were relatively small, while

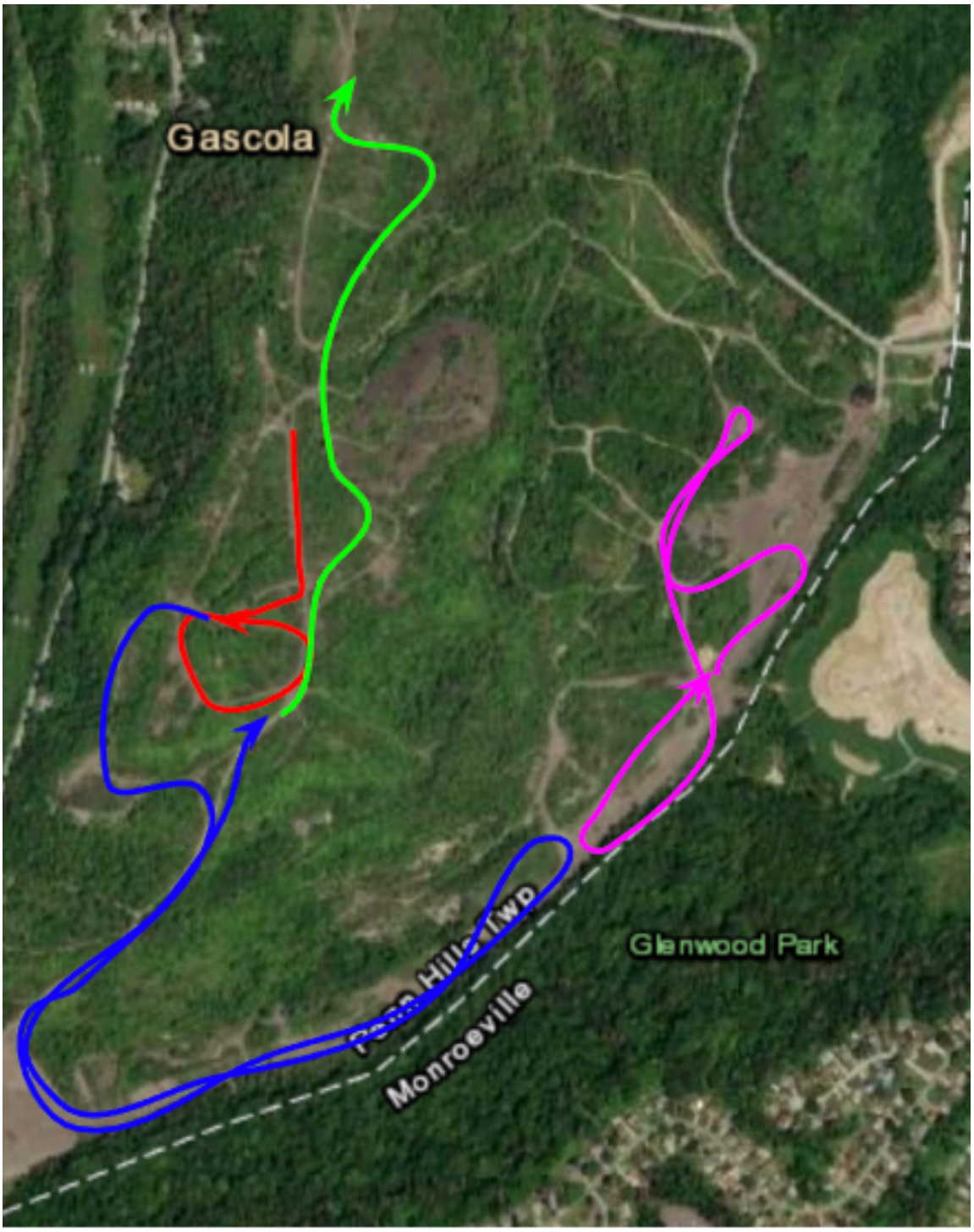


Figure 5.3: An overview of our navigation courses. In total, the courses were roughly 1.6km, and included several challenging scenarios such as going over slopes and through tall grass.

Method	Auto. Dist.	Auto. Speed	Interventions
Baseline	$1465m$	$3.21m/s$	7
IRL (CVaR -0.9)	$1482m$	$3.25m/s$	4
IRL (CVaR 0.0)	$1498m$	$3.29m/s$	**3**
IRL (CVaR 0.9)	$1592m$	$3.14m/s$	4

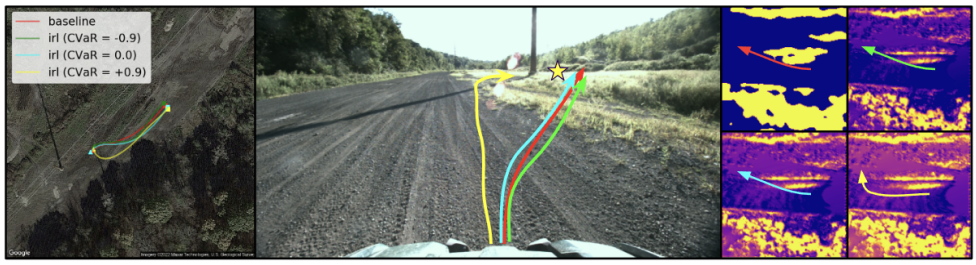
Table 5.4: Navigation metrics for each method on the purple course. Distance and speed columns are not bolded as higher/lower values don't necessarily mean better performance.

$\text{CVaR}_{0.9}$ was considerably slower and traveled farther.

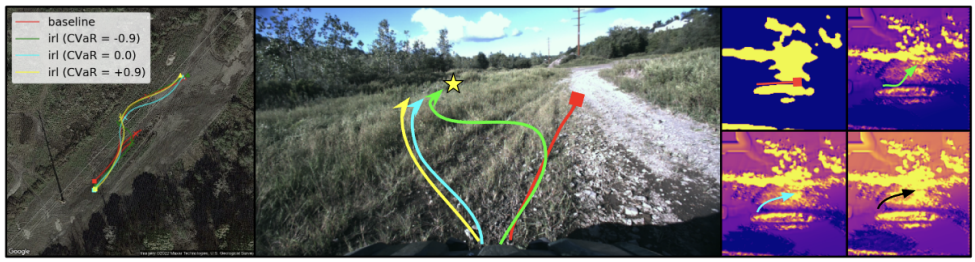
5.9 Qualitative Results

Presented below are additional qualitative visualizations of the costmaps learned from linear features and resnets. Visualizations are produced from representative examples of terrain from the test set in Figures 5.5 and 5.6. Note that *each* subplot has its own color normalization. That is, changes in color denote changes in cost *relative* to other costs in that particular subplot. This was done to mitigate higher CVaR potentially producing a constant bias and shift. Also note that the vehicle is located in the center of each BEV plot, and the orientation of the vehicle in the map is notated via the red arrow.

(a) Costmaps learned via IRL allow the robot to cut a corner over some low grass (about 15cm), while the baseline takes a longer route (note that obstacle inflation makes the grass patch appear as an obstacle to the baseline).

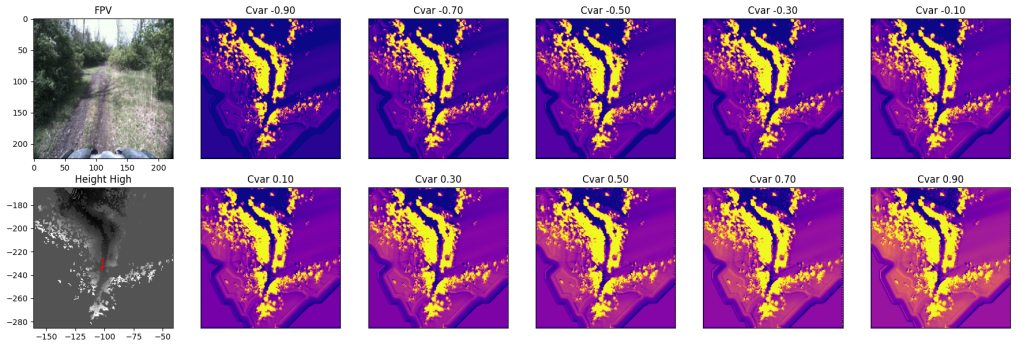


(b) Both the baseline and some IRL costmaps result in navigation over roughly 25cm grass, while IRL with a low risk tolerance (CVaR=0.9), takes a longer route around.

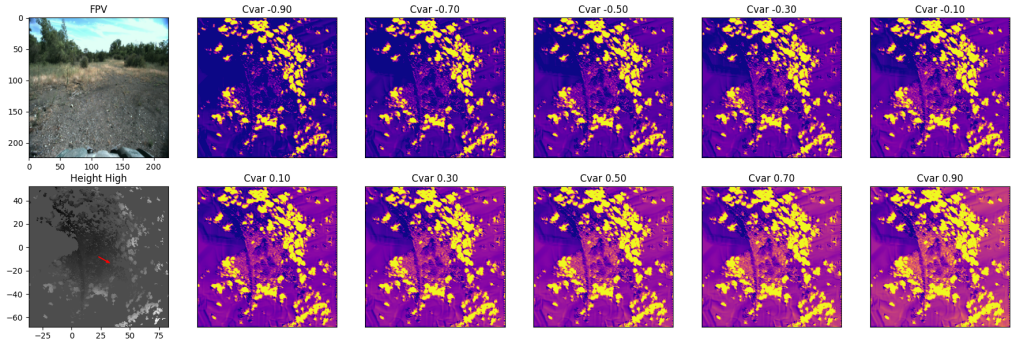


(c) The baseline method is unable to navigate to the waypoint in the tall grass, resulting in an intervention. IRL with a high risk tolerance (CVaR=-0.9) stays on the trail longer, then cuts through a taller grass patch. (black arrow used for clarity in costmap plot for CVaR=0.9)

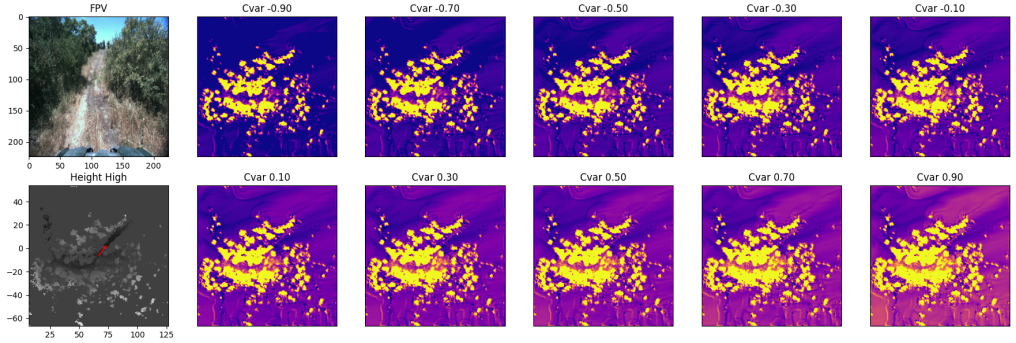
Figure 5.4: Several scenarios that during the navigation run that resulted in different behaviors. Left column: An enlarged BEV of the particular scenario, with trajectories from the individual runs superimposed (start=square, end=triangle, intervention=X, waypoint=yellow star). Middle column: FPV of the given scenario, with the paths from each trial annotated on. Right column: Visualization of the respective costmaps and trajectories for each. (top left: baseline, top right: IRL (CVaR -0.9), bottom left: IRL (CVaR 0.0), bottom right: IRL (CVaR 0.9).



(a) Corridor Scenario. The relative cost of obstacles and trail are consistent for all CVaR_ν.

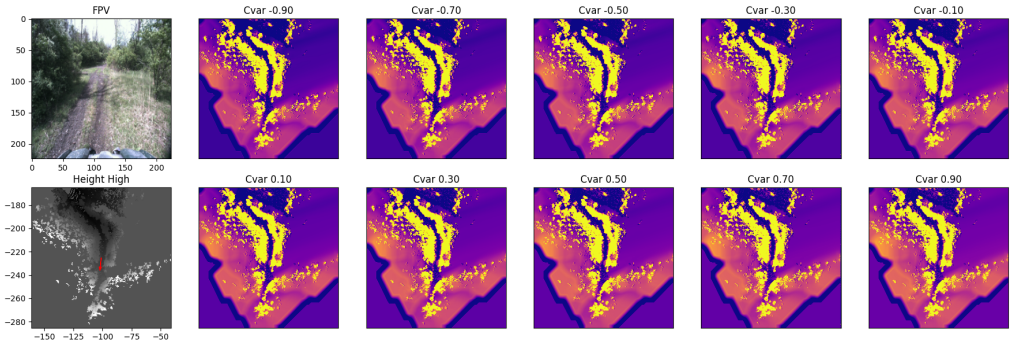


(b) Tall Grass Scenario. The relative cost of grass increases with CVaR_ν.

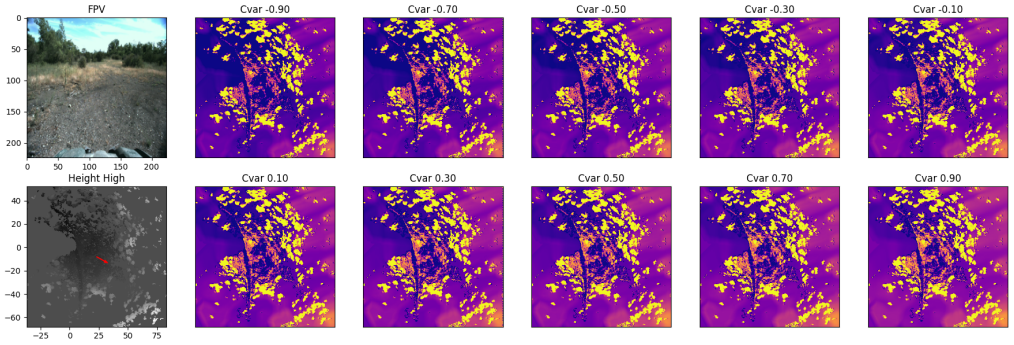


(c) Slope Scenario. The relative cost of the slope in the FPV increases with CVaR_ν.

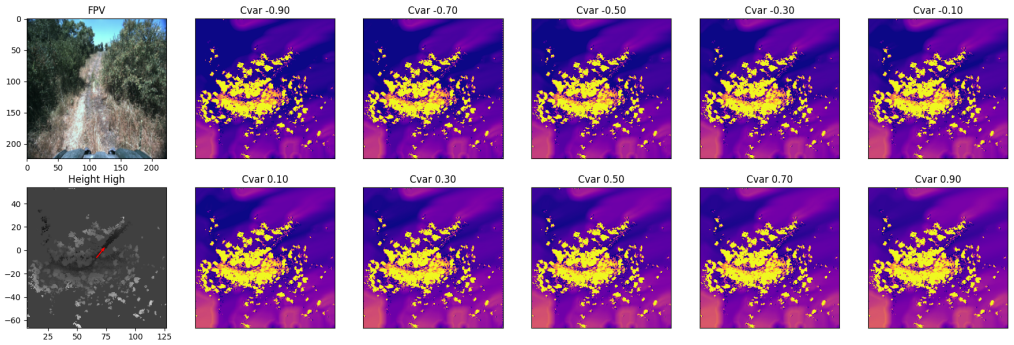
Figure 5.5: Resnet results on representative terrains in the test set. Note that in uncertain regions such as grass and slopes, the resnet-based costmap adjusts costs based on CVaR.



(a) Corridor Scenario



(b) Tall Grass



(c) Slope Scenario

Figure 5.6: Linear results on representative terrains in the test set. Note that the costmaps are more or less unchanged with respect to CVaR.

Chapter 6

Background

6.1 Learning Models

Most publicly-available off-road driving datasets focus on understanding environmental features instead of the interplay between the robot and the environment [40]. RUGD [58] consists of video sequences with segmentation label of 24 unique category annotations. Maturana et al. [36] collected a segmentation dataset, which tried to explore more fine-grained information such as traversable grass and non-traversable vegetation, but those discrete labels are still too abstract to understand how a robot is going to behave in the specific case. Gresenz et al. [18] collected a dataset containing over 10000 images of offroad bicycle driving and labels corresponding to the roughness of the terrain in the image, where the labels were generated from processing sensory data available on the platform. Similarly, this dataset does not provide enough interaction data or actions. Rellis 3D [27] consists of images, pointclouds, robot states, and actions, but the amount of trajectory data is rather small (about 20 minutes – 12,000 datapoints at 10hz – total). Generally, learning dynamics models requires on the order of hundreds of thousands of dynamics interactions. As our dataset is designed to focus on dynamics models with multi-modal sensory inputs without the need for hand-labeling, we are able to collect much more data than existing datasets.

More and more off-road driving research has taken account of the robot-environment interaction rather than environment features alone to improve the driving performance. Due to the lack of publicly-available real-world datasets, they usually train

their models in simulation environments or collect their own data in a small scale or for a specific research purpose. Tremblay et al. [50] trained a multi-modal dynamics model in a simulation environment based on Unreal Engine, and a small forest dataset (Montmorency) [49]. Sivaprakasam et al. [46] developed a simulation environment with random obstacles to learn a predictive model from physical interaction data. They also collected a small real-world dataset to test their algorithm on a small racer car. Similarly, Wang et al. [56] trained a probabilistic dynamics model for planning using a simulator and a small robot platform. Kahn et al. [28] employ a model-based RL technique in order to enable a wheeled robot to navigate off-road terrain using a monocular front-facing camera [28]. Their algorithm (BADGR) leverages many hours of trajectory data collected via random exploration in order to train a neural network to model the dynamics of the robot over various types of terrain. In addition to using the position of the robot, the authors also utilize hand-designed events calculated from the on-board IMU and LiDAR as additional supervision for the model. The authors make their data publicly-available, where the data consists of the hand-designed events (such as bumpiness and collision), RGB images, robot states, and actions.

On the other hand, learning a dynamics model has been an active research area in various directions such as model-based reinforcement learning (RL). Most work in RL literature learns the model in simulation environments. For the off-road driving task, [28] learned from the interaction data a classification network to predict the hand-designed events (i.e. collision, smoothness, etc.). Tremblay et al. [50] modify the recurrent state-space model of Hafner et al. [20] to handle multiple modalities and provide a modified training objective. Wang et al. [56] improve trajectory prediction by incorporating uncertainty estimation and a closed-loop tracker. In this paper, we follow these papers and test several neural network architectures to demonstrate the potential value of the proposed TartanDrive dataset.

6.2 Learning Costmaps

Perhaps the most widely-used costmap generation method relies on computing cost from geometric properties of the terrain. The most straightforward form of this approach is to create an occupancy-based representation of the terrain and assign high cost to cells that are occupied with high-height terrain. Krüsi et al. [31],

Fankhauser et al. [17] and Fan et al. [16] extend this representation with additional costs derived from geometric terrain features such as curvature and roughness. They all demonstrate that these features allow for improved navigation over rough terrain.

Recent work in off-road driving has focused on leveraging semantic segmentation approaches to assign traversability costs to different terrain types. Traditional work performs this segmentation in a first-person view (FPV) [27, 36, 57, 58]. However, it remains challenging to convert this segmentation into a useful representation for navigation, as simple projection is not robust to occlusions or sensor mis-calibration. As such, recent work by Shaban et al. [45] instead performs semantic segmentation in a bird's-eye view (BEV), using pointclouds from lidar. By first defining a mapping between semantic classes and cost, they are able to train a network to directly output costmaps. While they are able to demonstrate impressive navigation and prediction results, the mapping between semantic classes and cost is both heuristically defined and coarse, limiting the network's ability to generalize between robots or easily adapt online.

Inverse RL for costmap generation for wheeled vehicles is also a well-explored problem [5, 41, 42]. Recent work in generating costmaps for off-road vehicles largely focus on introducing deep neural networks as cost function approximators. Wulfmeier et al. [63] trained a deep, fully convolutional network using inverse RL for urban driving scenarios and demonstrate improved performance in generating expert trajectories as compared to a hand-crafted baseline. They also observe an issue in that there are few gradient propagations in rarely-explored areas of the state space, causing their cost networks to perform more poorly on unseen state features. Lee et al. [32] focus on learning costmaps in a multi-agent highway driving scenario. In addition to replacing the value iteration step of IRL with MPPI [60], they also propose regularizing unvisited states to have zero cost to alleviate the costmap artifacts observed by Wulfmeier et al. They additionally add a time dimension to the state-space to account for dynamic agents in their environment. Zhu et al. [71] propose a modification to the standard IRL procedure that takes into account an approximation of the kinematics of Ackermann-steered vehicles. They demonstrate that their algorithm results in improved ability to re-create expert behavior on unimproved roads. Zhang et al. [68] extend the basic IRL framework to include kinematic information. They concatenate information such as velocity and curvature into their intermediate feature

representations in order to get costmaps that are able to reflect the current dynamics of their vehicle. Wigness et al. [57] combine semantic segmentation and IRL for a skid-steered robot in off-road scenarios. Unlike previous approaches which focus on extracting costmaps from lidar features, they instead produce costmaps from semantic classes from BEV-projected camera data. Using this feature space, they are then able to learn a costmap using a linear combination of these classes, as well as an obstacle layer. They also demonstrate the capability to quickly update navigation behaviors with a small number of additional demonstrations.

Chapter 7

Conclusions

We presented two methods for incorporating learning into the offroad driving problem; using learning to improve vehicle models with real-world data, and using learning to improve cost functions from human demonstrations. While the work presented in this thesis represents a significant improvement over standard baselines, there remain a number of shortcomings that will be addressed in future work.

A fairly significant limitation of this work is a lack of intelligent velocity planning. While costmaps are well-suited to making the vehicle change the shape of its trajectory, they are not able to effectively tell the robot to traverse through a region at a given speed. The use of distance-to-goal as a heuristic in the cost function exacerbates this issue in that faster trajectories are nearly always preferable. The end result of this is that we had to set a relatively low upper-bound on speed (around 4m/s) in our navigation trials.

A simple solution to this issue would be to add a velocity dimension to our inverse RL method. While this may improve performance, there are some potential problems with this method:

1. IRL methods scale poorly with the number of state dimensions
2. Intuitively, desired speeds depend both on map features and goal information.

An interesting line of work would be to extend the current IRL methods to include visual features. Methods such as that presented by Harley et al. [21] produce BEV-projected visual features that can be directly consumed by the IRL algorithm

presented in this work. Extendind this work to admit semantic probabilities, raw RGB, or deep features should enhance the decision-making capability of the IRL algorithm.

Another interesting line of work would be to couple learning both costmap parameters and MPC parameters. While much of the in-field costmap tuning issues are alleviated by IRL, there are still a large number of hyperparameters in the MPC that are tuned regularly.

Bibliography

[1] Arl autonomy stack, 2022. URL https://www.arl.army.mil/business/collaborative-alliances/current-cras/sara-cra/sara-overview/. Accessed 13-September-2022. 1.1, 2.3, 3, 5.4.2

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004. 5.4.1

[3] Ali Agha, Kyohei Otsu, Benjamin Morrell, David D Fan, Rohan Thakker, Angel Santamaria-Navarro, Sung-Kyun Kim, Amanda Bouman, Xianmei Lei, Jeffrey Edlund, et al. Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge. *arXiv preprint arXiv:2103.11470*, 2021. 1.1, 1.2

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017. 5.3.1

[5] James Andrew Bagnell, David Bradley, David Silver, Boris Sofman, and Anthony Stentz. Learning for autonomous navigation. *IEEE Robotics & Automation Magazine*, 17(2):74–84, 2010. 1.1, 1.2, 6.2

[6] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9297–9307, 2019. 1.1

[7] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022. 5.4.2

[8] Paulo Borges, Thierry Peynot, Sisi Liang, Bilal Arain, Matthew Wildie, Melih Minareci, Serge Lichman, Garima Samvedi, Inkyu Sa, Nicolas Hudson, et al. A survey on terrain traversability analysis for autonomous ground vehicles:

Methods, sensors, and challenges. *Field Robotics*, 2(1):1567–1627, 2022. 1.1

[9] Xiaoyi Cai, Michael Everett, Jonathan Fink, and Jonathan P How. Risk-aware off-road navigation via a learned speed distribution map. *arXiv preprint arXiv:2203.13429*, 2022. 1.1, 5.5, 5.5

[10] Yanshuai Cao and David J Fleet. Generalized product of experts for automatic and principled fusion of gaussian process predictions. *arXiv preprint arXiv:1410.7827*, 2014. 4.5

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing*, 2014. 4.5

[12] Arnav Choudhry, Brady Moon, Jay Patrikar, Constantine Samaras, and Sebastian Scherer. Cvar-based flight energy risk assessment for multirotor uavs using a deep energy model. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 262–268. IEEE, 2021. 5.5

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1.1

[14] M-P Dubuisson and Anil K Jain. A modified hausdorff distance for object matching. In *Proceedings of 12th international conference on pattern recognition*, volume 1, pages 566–568. IEEE, 1994. 5.7

[15] 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. 1.1, 5.5

[16] David D Fan, Kyohei Otsu, Yuki Kubo, Anushri Dixit, Joel Burdick, and Ali-Akbar Agha-Mohammadi. Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation. *arXiv preprint arXiv:2103.02828*, 2021. 6.2

[17] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. Robust rough-terrain locomotion with a quadrupedal robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5761–5768. IEEE, 2018. 6.2

[18] Gabriela Gresenz, Jules White, and Douglas C Schmidt. An off-road terrain dataset including images labeled with measures of terrain roughness. 6.1

[19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations*, 2019. 4.3, 4.5, 4.5, 4.5.1, 4.6.2

[20] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019. 6.1

[21] Adam W Harley, Zhaoyuan Fang, Jie Li, Rares Ambrus, and Katerina Fragkiadaki. A simple baseline for bev perception without lidar. *arXiv preprint arXiv:2206.07959*, 2022. 7

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. (document), 4.1

[23] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002. 4.5, 2

[24] Thomas M Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26 (2):141–166, 2007. 1.1

[25] Larry D Jackel, Eric Krotkov, Michael Perschbacher, Jim Pippine, and Chad Sullivan. The darpa lagr program: Goals, challenges, methodology, and phase i results. *Journal of Field robotics*, 23(11-12):945–973, 2006. 1.1

[26] Edwin T Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982. 5.4.1

[27] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. Rellis-3d dataset: Data, benchmarks and analysis, 2020. 1.1, 4.3.1, 6.1, 6.2

[28] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021. 1.1, 1, 6.1

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. **??**, 9

[30] Kenji Koide, Jun Miura, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Interactive 3d graph slam for map correction. *IEEE Robotics and Automation Letters*, 6(1):40–47, 2021. doi: 10.1109/LRA.2020.3028828. 1.2

[31] Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart. Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments. *Journal of Field Robotics*, 34(5):940–984, 2017. 6.2

[32] Keuntaek Lee, David Isele, Evangelos A Theodorou, and Sangjae Bae. Spatiotemporal costmap inference for mpc via deep inverse reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):3194–3201, 2022. 6.2

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 1.1

[34] John Mai. System design, modelling, and control for an off-road autonomous ground vehicle. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, July 2020. 3.2

[35] Anirudha Majumdar and Marco Pavone. How should a robot assess risk? towards an axiomatic theory of risk in robotics. In *Robotics Research*, pages 75–84. Springer, 2020. 5.5

[36] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Field and Service Robotics*, pages 335–350. Springer, 2018. 1.1, 6.1, 6.2

[37] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007. 4.4

[38] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020. 4.2.1, 4.2.1, 5.4.2

[39] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 3, 4.7.1, 5, ??, ??, ??, ??

[40] Zachary Pezzementi, Trenton Tabor, Peiyun Hu, Jonathan K Chang, Deva Ramanan, Carl Wellington, Benzun P Wisely Babu, and Herman Herman. Comparing apples and oranges: Off-road pedestrian detection on the national robotics engineering center agricultural person-detection dataset. *Journal of Field Robotics*, 35(4):545–563, 2018. 6.1

[41] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006. 1.2, 6.2

[42] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27 (1):25–53, 2009. 1.2, 5.3.1, 5.5, 6.2

[43] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000. 5.5

[44] Sebastian Scherer, Vasu Agrawal, Graeme Best, Chao Cao, Katarina Cujic,

Ryan Darnley, Robert DeBortoli, Eric Dexheimer, Bill Drozd, Rohit Garg, Ian Higgins, John Keller, David Kohanbash, Lucas Nogueira, Roshan Pradhan, Michael Tatum, Vaibhav K. Viswanathan, Steven Willits, Shibo Zhao, Hongbiao Zhu, Dan Abad, Tim Angert, Greg Armstrong, Ralph Boirum, Adwait Dongare, Matthew Dworman, Shengjie Hu, Joshua Jaekel, Ran Ji, Alice Lai, Yu Hsuan Lee, Anh Luong, Joshua Mangelson, Jay Maier, James Picard, Kevin Pluckter, Andrew Saba, Manish Saroya, Emily Scheide, Nathaniel Shoemaker-Trejo, Joshua Spisak, Jim Teza, Fan Yang, Andrew Wilson, Henry Zhang, Howie Choset, Michael Kaess, Anthony Rowe, Sanjiv Singh, Ji Zhang, Geoffrey A. Hollinger, and Matthew Travers. Resilient and modular subterranean exploration with a team of roving and flying robots. *Field Robotics Journal*, pages 678–734, May 2022. 1.1, 1.2

[45] 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. 1.1, 6.2

[46] Matthew Sivaprakasam, Samuel Triest, Wenshan Wang, Peng Yin, and Sebastian Scherer. Improving off-road planning techniques with learned costs from physical interactions. In *Proceedings - IEEE International Conference on Robotics and Automation*, Xi'an, China, May 2021. 6.1

[47] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020. URL `http://jmlr.org/papers/v21/20-091.html`. 4.4

[48] Stanford Racing Team. Stanford racing team's entry in the 2005 darpa grand challenge. *Published on DARPA Grand Challenge*, 2005. 1.1

[49] Jean-François Tremblay, Martin Béland, François Pomerleau, Richard Gagnon, and Philippe Giguere. Automatic 3d mapping for tree diameter measurements in inventory operations. *Journal of Field Robotics*, 2019. 6.1

[50] Jean-François Tremblay, Travis Manderson, Aurélio Noca, Gregory Dudek, and David Meger. Multimodal dynamics modeling for off-road autonomous vehicles. *IEEE International Conference on Robotics and Automation*, 2020. 4.3, 4.5, 4.5, 4.5, 6.1

[51] Samuel Triest, Matthew Sivaprakasam, Sean J Wang, Wenshan Wang, Aaron M Johnson, and Sebastian Scherer. Tartandrive: A large-scale dataset for learning off-road dynamics models. *arXiv preprint arXiv:2205.01791*, 2022. 4, 4.3, 5.3, 5.3

[52] Samuel Triest, Mateo Guaman Castro, Parv Maheshwari, Matthew Sivaprakasam, Wenshan Wang, and Sebastian Scherer. Learning risk-aware costmaps via inverse

reinforcement learning for off-road navigation. *arXiv preprint arXiv:2302.00134*, 2023. 5

[53] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 4.4

[54] Andrew VanOsten. Lidar-visual-inertial odometry via modifications and improvements to super odometry. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, July 2022. 3.2.3

[55] Sean J Wang, Samuel Triest, Wenshan Wang, Sebastian Scherer, and Aaron Johnson. Rough terrain navigation using divergence constrained model-based reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021. 4.2.1

[56] Sean J. Wang, Samuel Triest, Wenshan Wang, Sebastian Scherer, and Aaron M. Johnson. Rough terrain navigation using divergence constrained model based reinforcement learning. In *Conference on Robot Learning*. PMLR, 2021. 6.1

[57] Maggie Wigness, John G Rogers, and Luis E Navarro-Serment. Robot navigation from human demonstration: Learning control behaviors. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1150–1157. IEEE, 2018. 5.7, 6.2

[58] Maggie Wigness, Sungmin Eum, John G Rogers, David Han, and Heesung Kwon. A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems*, 2019. 1.1, 4.3.1, 6.1, 6.2

[59] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017. 3, 5.4.2

[60] 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*, pages 1714–1721. IEEE, 2017. 5.4.2, 6.2

[61] Mike Wu and Noah Goodman. Multimodal generative models for scalable weakly-supervised learning. *Advances in Neural Information Processing Systems*, 2018. 4.5

[62] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *CoRR, abs/1507.04888*, 2015. 5.4.1, 9

[63] Markus Wulfmeier, Dushyant Rao, Dominic Zeng Wang, Peter Ondruska, and Ingmar Posner. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, 36(10):1073–1087, 2017. 6.2

[64] Xuesu Xiao, Bo Liu, Garrett Warnell, Jonathan Fink, and Peter Stone. Appld: Adaptive planner parameter learning from demonstration. *IEEE Robotics and Automation Letters*, 5(3):4541–4547, 2020. 1.1

[65] Xuesu Xiao, Zizhao Wang, Zifan Xu, Bo Liu, Garrett Warnell, Gauraang Dhamankar, Anirudh Nair, and Peter Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022. 1.1

[66] Stuart H Young. Robot autonomy in complex environments. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, page 1174602. SPIE, 2021. 1.1

[67] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017. 2

[68] Yanfu Zhang, Wenshan Wang, Rogerio Bonatti, Daniel Maturana, and Sebastian Scherer. Integrating kinematics and environment context into deep inverse reinforcement learning for predicting off-road vehicle trajectories. *arXiv preprint arXiv:1810.07225*, 2018. 5.7, 6.2

[69] Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, and Sebastian Scherer. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8729–8736. IEEE, 2021. 3.2.3

[70] Kaiyu Zheng. Ros navigation tuning guide. *Robot Operating System (ROS) The Complete Reference (Volume 6)*, pages 197–226, 2021. 1.1

[71] Zeyu Zhu, Nan Li, Ruoyu Sun, Donghao Xu, and Huijing Zhao. Off-road autonomous vehicles traversability analysis and trajectory planning based on deep inverse reinforcement learning. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 971–977. IEEE, 2020. 5.7, 6.2

[72] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 5.4.1