CIENE9101: Civil Engineering Research Report Spring (2024) 1-26 Submitted on: May 12, 2024

Data Distillation & Control Signal Generation for Autonomous Vehicles using Waymo Open Dataset

Apurva Patel APURVA.PATEL@COLUMBIA.EDU
Department of Electrical Engineering
Columbia University
New York City, NY 10027, USA

Zhaobin Mo ZM2302@COLUMBIA.EDU
Department of Civil Engineering and Engineering Mechanics
Columbia University
New York City, NY 10027, USA

Editor: Apurva Patel (amp2365)

Abstract

Autonomous driving systems heavily rely on accurate perception and decision-making algorithms to navigate complex environments safely. In this research work, we propose a novel approach for generating control signals for autonomous vehicles by distilling information from the Waymo Open Dataset. Leveraging deep learning techniques, we extract valuable insights from the dataset to understand the dynamics of real-world driving scenarios. Specifically, we focus on generating control signals for turning right and left, as well as for braking and accelerating, based on the learned features. Our approach combines data-driven modeling with advanced control strategies to achieve robust and adaptive behavior in diverse driving conditions. test results demonstrate the effectiveness of our method in accurately predicting control signals, showcasing its potential for enhancing the autonomy and safety of future driving systems.

Keywords: Autonomous Driving, Adaptive Control, Perception, Deep Learning

1 Introduction

Autonomous driving technology has made significant advancements in recent years, promising safer and more efficient transportation systems. Central to the development of autonomous vehicles (AVs) are perception and decision-making algorithms that enable vehicles to understand their environment and navigate safely. One crucial aspect of AV control systems is the generation of control signals, which dictate the vehicle's behavior, including steering, braking, and acceleration.

In this research, we present a novel approach to control signal generation for AVs, leveraging insights distilled from the Waymo Open Dataset. The Waymo Open Dataset is a large-scale collection of real-world driving data captured by Waymo's autonomous vehicles. By analyzing this rich dataset, we aim to extract valuable information about driving behaviors, road conditions, and environmental factors that influence AV control.

Code can be accessed at https://github.com/Apurva3509/CEEM-Research.git.

Our approach combines deep learning techniques with data-driven modeling to learn complex relationships between sensor data and control signals.

Specifically, we focus on generating control signals for turning right and left, as well as for braking and accelerating. By training deep neural networks on the Waymo Open Dataset, we aim to develop models capable of accurately predicting control signals in diverse driving scenarios.

The contributions of this paper are twofold. First, we demonstrate the effectiveness of distillation-based methods for learning from large-scale driving datasets. By distilling knowledge from the Waymo Open Dataset, we aim to capture the essence of real-world driving behaviors and transfer this knowledge to our control signal generation models. Second, we showcase the potential of deep learning techniques for enhancing the autonomy and safety of AVs by enabling them to make informed decisions based on learned representations of the driving environment.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of related work in the field of AV control signal generation and distillation-based learning. In Section 3, we describe the Waymo Open Dataset. In Section 4 we discuss about our methodology for distilling knowledge from the dataset. In Section 5, we present our approach for control signal generation using deep learning techniques. In Section 6, we show the results of our preliminary research through extensive visualizations and control determination on real-world driving data from Waymo dataset's test section. Finally, in Section 7, we conclude our research work and discuss avenues for future research to increase the usage of this project.

2 Literature Survey

The advent of autonomous driving technology promises a paradigm shift in transportation, from agentaxis to self-driving trucks. This technology heavily relies on the availability of large-scale datasets and benchmarks, accelerating progress in various machine perception tasks such as image classification, object detection, tracking, and segmentation.

2.1 Previous work on data distillation

Data distillation techniques are essential in the field of machine learning, particularly for tasks like autonomous driving where large datasets are often used to train complex models. One prominent method is Meta Model Matching, which focuses on optimizing the transferability of models trained on data summaries to the original dataset. This involves an inner loop training process that employs a representative algorithm on the data summary, enhancing its transferability to the original dataset.

Another approach, Data Distillation by Gradient Matching, involves matching the distance between networks trained on the target dataset and distilled data. This method bypasses the need for inner loop optimization by directly optimizing the gradient matching between the two datasets. By doing so, it facilitates the synthesis of distilled data that closely approximates the original training data without the need for extensive computational resources.

Data Distillation by Trajectory Matching aims to align the training trajectories of models trained on the original dataset and the distilled data. This alignment process ensures that the distilled data captures the essential features and characteristics of the original dataset, leading to improved model performance and generalization.

In Data Distillation by Distribution Matching (DM), the focus is on aligning the distribution of the distilled data with the original dataset. This is achieved by utilizing parametric encoders to map high-dimensional data to low-dimensional latent spaces and approximating distribution mismatch using metrics such as Max Mean Discrepancy. By minimizing the distribution gap between the distilled and original datasets, DM enhances the effectiveness of data distillation for various machine learning tasks.

Furthermore, Data Distillation by Factorization parametrizes data summaries using independent base vectors and hallucinators. This approach optimizes both the data distillation process and the quality of the distilled data, leading to improved model performance and generalization across different tasks and datasets.

Mathematically, these techniques involve optimization algorithms that minimize the discrepancy between the distilled data and the original dataset. For instance, in Gradient Matching, the optimization objective may involve minimizing the loss function between the gradients of networks trained on the original and distilled datasets:

$$\min_{\theta} \sum_{i=1}^{N} \left\| \nabla f_\theta(x_i) - \nabla g_\theta(x_i') \right\|^2$$

where $f_\theta(x_i)$ represents the output of the original network, $g_\theta(x_i')$ represents the output of the network trained on distilled data, and θ denotes the network parameters.

Similarly, in Distribution Matching, the optimization objective involves minimizing the discrepancy between the distributions of the original and distilled datasets using metrics such as Max Mean Discrepancy:

$$\min_{\theta} \left\| \mathbb{E}_{x \sim p(x)}[\phi(x)] - \mathbb{E}_{x' \sim p'(x')}[\phi(x')] \right\|^2$$

where $p(x)$ and $p'(x')$ represent the distributions of the original and distilled datasets respectively, $\phi(\cdot)$ denotes a feature mapping function, and θ denotes the parameters of the feature mapping function.

2.2 Previous work on Control Signal generation in motion planning

2.3 Introduction

Generating control signals for autonomous navigation is a critical component that enables self-driving vehicles to safely and efficiently navigate through their environment. This literature survey delves into various approaches and algorithms for control signal generation, including the mathematical formulations involved. The discussion covers both classical control techniques as well as modern learning-based methods.

2.4 Classical Control Approaches

Classical control approaches for autonomous navigation typically involve modeling the system dynamics and designing controllers based on control theory principles. These methods often provide theoretical guarantees and interpretability but may struggle with complex, unstructured environments.

2.4.1 Potential Field Methods

Potential field methods treat the agent as a particle under the influence of an artificial potential field, where obstacles exert repulsive forces and the goal exerts an attractive force [6]. The control signal is computed as the negative gradient of the total potential field.

Let $U_{att}(q)$ be the attractive potential pulling the agent towards the goal configuration q_{goal}, and $U_{rep}(q)$ be the repulsive potential from obstacles. The total potential is:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{1}$$

The attractive potential is typically defined as:

$$U_{att}(q) = \frac{1}{2} \xi \rho(q, q_{goal})^2 \tag{2}$$

Where $\rho(q, q_{goal})$ is a metric like Euclidean distance between the agent's configuration q and the goal q_{goal}, and ξ is a scaling factor.

The repulsive potential has various formulations, like:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q, q_{obs}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where $\rho(q, q_{obs})$ is the distance to the nearest obstacle, ρ_0 is the region of influence, and η is a scaling factor.

The control signal $u(t)$ is computed as the negative gradient of the potential field:

$$u(t) = -\nabla U(q(t)) \tag{4}$$

While simple to implement, potential field methods suffer from local minima issues and oscillations in narrow passages [6].

2.4.2 Vector Field Histogram (VFH+)

The VFH+ algorithm constructs a polar histogram grid around the agent from sensor data, identifying directions with lower obstacle densities as candidates for steering [16]. Key steps include:

1. Construct a polar histogram grid around the agent from range sensor data.
2. Calculate a binary polar obstacle map by thresholding cell densities.
3. Find continuous free regions using a smoothing technique like fuzzy scoring.
4. Select the steering direction as the center of the largest free region.
5. Compute the steering command using the curvature velocity method.

The curvature velocity method computes the steering command ω as:

$$\omega = \omega_{max}\left(1 - \frac{2\Delta\theta_k}{\Delta\theta_{max}}\right) \tag{5}$$

Where $\Delta\theta_k$ is the angular deviation from the target direction, $\Delta\theta_{max}$ is the maximum allowed deviation, and ω_{max} is the maximum angular velocity [16].

2.4.3 Dynamic Window Approach (DWA)

The DWA is a velocity space-based reactive avoidance approach that directly generates velocity commands (v, ω) in the agent's control space [4]. It involves:

1. Sampling admissible velocity commands based on dynamic constraints. 2. Evaluating each sampled command based on objective functions like: - Heading towards the goal: $\text{heading}(v, \omega) = v\cos(\theta - \theta_{goal})$ - Clearance from obstacles: $\text{dist}(v, \omega) = \min_i \rho(v, \omega, p_i)$ - Velocity objective: $\text{velocity}(v, \omega) = v$ 3. Selecting the command that maximizes a weighted sum of the objectives.

The velocity command (v^*, ω^*) is selected as:

$$(v^*, \omega^*) = \underset{(v,\omega)}{\text{argmax}}\Big(\alpha\text{heading}(v, \omega) + \beta\text{dist}(v, \omega) + \gamma\text{velocity}(v, \omega)\Big)$$

Where α, β, γ are user-defined weights [4].

2.4.4 Model Predictive Control (MPC)

MPC is an optimal control approach that solves a constrained optimization problem over a finite horizon to generate control inputs [8]. The general formulation is:

$$\begin{aligned} \underset{u(t),\ldots,u(t+T-1)}{\text{minimize}} \quad & J(x(t), u(t), \ldots, u(t+T-1)) \\ \text{subject to} \quad & x(t+1) = f(x(t), u(t)) \\ & x(t) \in \mathcal{X}, u(t) \in \mathcal{U} \\ & x(t+T) \in \mathcal{X}_f \end{aligned} \tag{6}$$

Where J is the cost function, f is the system dynamics model, $\mathcal{X}$ and $\mathcal{U}$ are state and input constraints, and $\mathcal{X}_f$ is the terminal constraint set.

Common cost functions include:

- Quadratic costs: $J = \sum_{k=0}^{T-1} \|x(t+k) - x_{ref}(t+k)\|_Q^2 + \|u(t+k)\|_R^2$ - Obstacle costs: $J = \sum_{k=0}^{T-1} \sum_i \rho(x(t+k), o_i)$ - Terminal costs: $J = \|x(t+T) - x_{goal}\|_P^2$

Where Q, R, P are weight matrices, x_{ref} is a reference trajectory, and $\rho(x, o_i)$ is a distance function to obstacles o_i [8].

The optimization problem is solved at each time step, and only the first control input is applied (receding horizon control).

2.5 Learning-based Approaches

Learning-based approaches leverage machine learning techniques to learn control policies from data, either through expert demonstrations or through interaction with the environment. These methods can handle complex, unstructured environments but may lack interpretability and theoretical guarantees.

2.5.1 Imitation Learning

Imitation learning aims to learn a policy that mimics expert demonstrations. Common approaches include:

1. **Behavioral Cloning**: Supervised learning to map observations to actions from expert data (x_i, u_i).

2. **Inverse Reinforcement Learning**: Recover the reward function from demonstrations, then solve for the optimal policy.

3. **Generative Adversarial Imitation Learning**: Use a discriminator to distinguish expert and agent trajectories, and train the agent to fool the discriminator.

While simple, imitation learning can lead to compounding errors and fails to generalize beyond the training distribution [7].

2.5.2 Reinforcement Learning

Reinforcement learning (RL) learns an optimal policy π^* that maximizes the expected return $\mathbb{E}_\pi[\sum_t \gamma^t r_t]$ by interacting with the environment [11]. Popular RL algorithms include:

1. **Q-Learning**: Learn the optimal Q-function $Q^*(s,a) = \mathbb{E}_{\pi^*}[\sum_t \gamma^t r_t | s_0 = s, a_0 = a]$, then $\pi^*(s) = \text{argmax}_a Q^*(s,a)$.
2. **Policy Gradient Methods**: Directly optimize the policy π_θ by ascending the gradient $\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_t \gamma^t r_t]$.
3. **Actor-Critic Methods**: Learn a value function (critic) to reduce variance when optimizing the policy (actor).
4. **Model-based RL**: Learn a dynamics model $\hat{f}$ and solve for the optimal policy using planning algorithms.

RL has been successfully applied to autonomous driving tasks like end-to-end control from sensor inputs [5, 1]. However, sample inefficiency and safety issues remain key challenges.

2.5.3 Imitation and Reinforcement Learning

Combining imitation learning (IL) and reinforcement learning (RL) aims to leverage their complementary strengths. Common approaches include:

1. **IL to initialize RL**: Use imitation learning to warm-start the policy, then fine-tune with RL.
2. **RL with IL regularization**: Augment the RL objective with an imitation loss to stay close to expert data.
3. **Adversarial IL+RL**: Use a discriminator to classify expert vs. RL trajectories and train the RL agent to match expert data.

These techniques have shown improved sample efficiency and better generalization compared to pure IL or RL methods [7].

2.5.4 Learning for Control

While most learning-based methods directly output low-level controls like steering and acceleration, some approaches learn higher-level control representations:

1. **Learning Waypoints or Trajectory Parameters**: Instead of raw controls, learn to output waypoints or trajectory parameters that are tracked by a lower-level controller [1].
2. **Learning a Cost/Reward Function**: Learn a cost or reward function that captures the task, then use an optimal controller to generate actions minimizing/maximizing this learned function [3].

3. **Learning a Dynamics Model**: Learn an accurate dynamics model of the system, then use model-based planning and control techniques [2].

These hierarchical approaches can improve generalization, interpretability, and sample efficiency compared to end-to-end learning of raw controls [1].

Several research efforts have been conducted using the Waymo Open Dataset for control signal generation and related tasks in autonomous navigation. Here are some notable examples:

- **Imitation Learning for Autonomous Driving** [12]: This work by Waymo researchers explores the use of imitation learning for end-to-end control of self-driving vehicles. They propose a novel imitation learning approach called CILRS (Conditional Imitation Learning with Reinforcement Scoring) and demonstrate its effectiveness on the Waymo Open Dataset.

- **Model-based Reinforcement Learning for Autonomous Driving** [1]: In this paper, researchers from the University of California, Berkeley, and Waymo propose a model-based reinforcement learning approach for autonomous driving. They use the Waymo Open Dataset to learn a dynamics model and then leverage it for planning and control using reinforcement learning.

- **Occupancy and Flow Prediction for Autonomous Driving** [14]: This work by researchers from the University of Toronto and Waymo focuses on predicting occupancy and flow maps from lidar data for autonomous driving. They use the Waymo Open Dataset to train and evaluate their proposed approach, which can be useful for control signal generation by providing a better understanding of the surrounding environment.

- **3D Semantic Segmentation for Autonomous Driving** [15]: Researchers from the University of California, Berkeley, and Waymo propose a novel 3D semantic segmentation approach for lidar point clouds. They use the Waymo Open Dataset for training and evaluation, demonstrating the importance of accurate 3D scene understanding for control signal generation in autonomous navigation.

- **Motion Prediction for Autonomous Driving** [13]: This work by researchers from the University of Toronto and Waymo explores the problem of motion prediction for autonomous driving. They use the Waymo Open Dataset to train and evaluate their proposed approach, which can be valuable for generating control signals that anticipate the future movements of other traffic participants.

Control signal generation for autonomous navigation spans a wide range of techniques, from classical methods like potential fields and optimization-based controllers, to modern learning-based approaches leveraging imitation learning, reinforcement learning, and their combinations. The choice of method depends on factors like the complexity of the environment, availability of expert data, computational constraints, and the desired level of

interpretability and safety guarantees. As autonomous driving systems become more prevalent, research in this area will continue to advance, aiming to develop safe, efficient, and generalizable control algorithms.

3 Data Distillation

Data distillation involves various techniques aimed at compressing and synthesizing knowledge from a dataset. A detailed explanation of each technique is as follows.

3.1 Meta-Model Matching, Gradient Matching, and Trajectory Matching

Meta-Model Matching, Gradient Matching, and Trajectory Matching are techniques that focus on optimizing the parameters of a model to match the performance or behavior of another model trained on the original dataset. They achieve this by minimizing distances or divergences between the original and synthesized data distributions.

Mathematically, let's denote the original dataset as $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i represents the input data and y_i represents the corresponding labels. The goal is to synthesize a dataset D_{syn} such that a model trained on D_{syn} performs similarly to a model trained on D.

- **Meta-Model Matching**: This technique aims to match the performance of a target model Φ_θ trained on D by optimizing the parameters θ of another model $\Phi_{\theta'}$ on a synthesized dataset D_{syn}. For example, if Φ_θ is a neural network for image classification, θ' will be adjusted to minimize the classification loss on D_{syn}.

$$\theta' = \arg\min_{\theta'} \mathcal{L}(\Phi_{\theta'}(x), y), \quad (x, y) \sim D_{\text{syn}}$$

- **Gradient Matching**: Here, the goal is to match the gradients of the loss function of a target model Φ_θ trained on D with the gradients of a model $\Phi_{\theta'}$ trained on D_{syn}. This ensures that the models learn similar representations of the data.

$$\min_{\theta'} \|\nabla_\theta \mathcal{L}(\Phi_\theta(x), y) - \nabla_{\theta'} \mathcal{L}(\Phi_{\theta'}(x), y)\|^2, \quad (x, y) \sim D_{\text{syn}}$$

- **Trajectory Matching**: This technique involves matching the trajectories of model parameters obtained during training on the original dataset D with the trajectories of model parameters trained on D_{syn}. It ensures that the optimization paths followed by the models are similar.

$$\min_{\theta'} \sum_{t=1}^{T} D(\theta_t, \theta'_t), \quad \text{where } \theta_t, \theta'_t \text{ are model parameters at time } t$$

3.2 Distribution Matching

Distribution Matching techniques aim to directly match the distribution of the synthesized dataset D_{syn} with the distribution of the original dataset D.

Mathematically, let's denote the original data distribution as $p_D(x)$ and the synthesized data distribution as $p_{D_{\text{syn}}}(x)$. The goal is to minimize the discrepancy between these distributions using various metrics such as Maximum Mean Discrepancy (MMD) or Kullback-Leibler (KL) divergence.

$$\min_{D_{\text{syn}}} D(p_D, p_{D_{\text{syn}}})$$

For example, if D consists of images of cats and dogs, and D_{syn} is a synthesized dataset, distribution matching techniques aim to ensure that the distribution of cat and dog images in D_{syn} closely matches that of D.

3.3 Factorization-Based Data Distillation

Factorization-Based Data Distillation techniques parameterize the data summary using two separate components: bases and hallucinators. Bases represent independent vectors, while hallucinators map these vectors to data points. This approach allows for better knowledge sharing and potentially reduces data redundancy.

Mathematically, let $B = \{b_i\}_{i=1}^{|B|}$ denote the set of bases and $H = \{h_i\}_{i=1}^{|H|}$ denote the set of hallucinators. The synthesized dataset D_{syn} is then parameterized as $D_{\text{syn}} = \{h(b)\}_{b \in B, h \in H}$.

This parameterization allows for joint optimization of B and H using techniques such as Trajectory Matching or Gradient Matching.

For example, in image generation, bases could represent different styles or features, while hallucinators map these styles to specific images. By optimizing both bases and hallucinators, the synthesized images can capture various styles and features effectively.

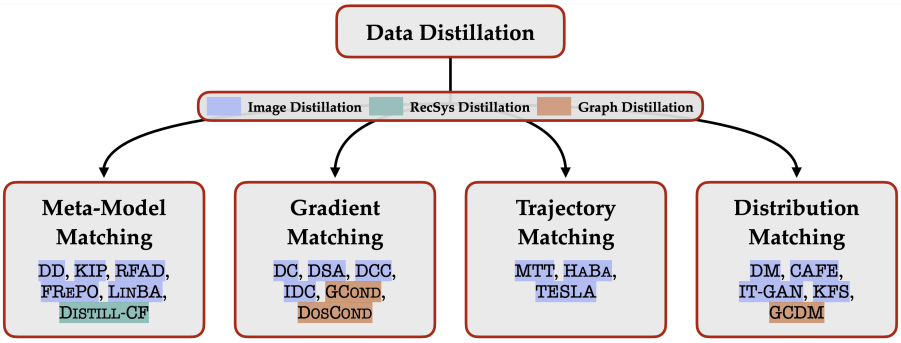

Figure 1: Various data distillation techniques (Figure adapted from [9])

In conclusion, data distillation involves various techniques aimed at compressing and synthesizing knowledge from a dataset. Each technique employs different mathematical formulations and optimization strategies to achieve its goal of generating a synthesized dataset that captures the essential information for training models effectively. By minimizing the discrepancy between the distilled and original datasets, these techniques enable more efficient and effective training of machine learning models, particularly in tasks like autonomous driving where large datasets are essential for achieving high performance and generalization.

4 About Waymo Open Dataset

The Waymo Open Dataset is a large-scale, multi-sensor dataset for autonomous driving research, released by Waymo, a leading company in the self-driving vehicle industry. This dataset provides a rich collection of real-world sensor data, including high-resolution lidar, camera, and radar data, along with corresponding ground truth labels for various perception tasks. The dataset's primary purpose is to advance research in autonomous driving by providing researchers and developers with a diverse and challenging real-world dataset to train, evaluate, and benchmark their algorithms.

4.1 Sensor Specifications

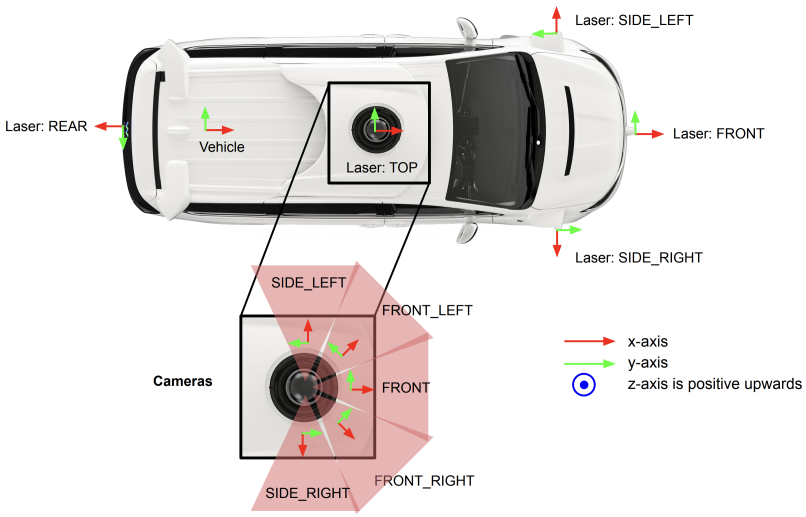

Figure 2: Sensor configuration in Waymo dataset (figure adapted from [10])

4.1.1 Lidar Sensors

The dataset includes data from 2 high-resolution Lidar sensor with the following specifications:

1. **Top Lidar**:
   - Range: 75 meters
   - Horizontal Field of View (HFOV): 360°
   - Vertical Field of View (VFOV): -17.6° to 2.4°
2. **Four Side Lidars** (front, rear, left, right):
   - Range: 20 meters
   - HFOV: 360°
   - VFOV: -90° to 30°

The lidar data is provided in the form of range images, where each pixel corresponds to a lidar return. The range images encode the range, azimuth, and inclination of each lidar point in spherical coordinates. The dataset includes data for the first two returns of each laser pulse.

4.1.2 Camera Sensors

The dataset includes data from five high-resolution pinhole cameras with the following specifications (see Fig. 2):

1. **Front Left (FL) Camera**:
   - Resolution: 1920 × 1280
   - HFOV: ±25.2°
2. **Front (F) Camera**:
   - Resolution: 1920 × 1280
   - HFOV: ±25.2°
3. **Front Right (FR) Camera**:
   - Resolution: 1920 × 1280
   - HFOV: ±25.2°
4. **Side Left (SL) Camera**:
   - Resolution: 1920 × 1040
   - HFOV: ±25.2°
5. **Side Right (SR) Camera**:
   - Resolution: 1920 × 1040
   - HFOV: ±25.2°

The camera images are captured with rolling shutter scanning, where each column is filled from left to right.

4.2 Data Format and Synchronization

The Waymo Open Dataset is stored in the TFRecord format, which is a simple format for storing a sequence of binary records. Each TFRecord file contains a sequence of `Frame` protocol buffer messages, where each `Frame` represents a single capture from all sensors at a specific timestamp.

A `Frame` message contains the following information:

- `context` metadata, including the timestamp, pose, and calibration information
- A collection of `laser` messages, each containing a range image for a single lidar return

- A collection of `image` messages, each containing a camera image and its camera name
- A collection of `projected_lidar` messages, containing the lidar points projected into each camera image
- A collection of `camera_object` messages, containing the 2D bounding box annotations for each camera image
- A collection of `laser_object` messages, containing the 3D bounding box annotations for the lidar data

The dataset provides synchronization between the camera and lidar data, enabling research in cross-domain learning and sensor fusion. Additionally, the dataset includes calibration parameters for projecting 3D lidar points to 2D camera image coordinates and vice versa.

4.3 Ground Truth Annotations

4.3.1 Lidar Annotations

The dataset includes approximately 12 million 3D bounding box annotations with tracking IDs for four object classes: vehicles, pedestrians, cyclists, and traffic signs. These annotations are provided for 1,200 segments of the dataset.

4.3.2 Camera Annotations

The dataset includes approximately 11.8 million 2D bounding box annotations with tracking IDs for the same four object classes: vehicles, pedestrians, cyclists, and traffic signs. These annotations are provided for 1,000 segments of the dataset.

The annotations were created and reviewed by trained labelers using production-level labeling tools, ensuring high-quality ground truth data.

4.4 Evaluation Metrics

4.4.1 Object Detection Metrics

The object detection metrics used in the Waymo Open Dataset are based on the mean Average Precision (mAP) metric, which is a widely used metric in object detection tasks. The mAP is calculated by averaging the maximum precisions at different recall values.

The dataset defines two levels of difficulty for object detection, similar to the KITTI dataset:

1. **LEVEL 1**: This level includes objects with a minimum bounding box height and visibility.
2. **LEVEL 2**: This level is more challenging and includes all objects, regardless of their size or visibility.

The mAP is calculated separately for each object class (vehicles, pedestrians, cyclists, and traffic signs) and difficulty level.

4.4.2 Motion Prediction Metrics

The Waymo Open Dataset includes a Motion Prediction Challenge, which focuses on predicting the future trajectories of objects in the scene. The evaluation metrics for this challenge include:

1. **Average Displacement Error (ADE)**: This metric measures the average Euclidean distance between the predicted and ground truth trajectories.
2. **Final Displacement Error (FDE)**: This metric measures the Euclidean distance between the predicted and ground truth positions at the final time step.
3. **Soft mAP**: This metric is a variant of the mAP metric, which considers the overlap between the predicted and ground truth bounding boxes at each time step.

4.5 Mathematical Formulations and Algorithms

4.5.1 Lidar Point Cloud Processing

Coordinate Transformations: The lidar data is provided in spherical coordinates (range, azimuth, inclination), and researchers may need to convert these coordinates to Cartesian coordinates (x, y, z) for further processing. The conversion from spherical to Cartesian coordinates can be performed using the following equations:

$$x = range \cdot \cos(azimuth) \cdot \cos(inclination) \tag{7}$$

$$y = range \cdot \sin(azimuth) \cdot \cos(inclination) \tag{8}$$

$$z = range \cdot \sin(inclination) \tag{9}$$

Conversely, the transformation from Cartesian to spherical coordinates can be achieved using the following equations:

$$range = \sqrt{x^2 + y^2 + z^2} \tag{10}$$

$$azimuth = \tan^{-1}(y/x) \tag{11}$$

$$inclination = \tan^{-1}(z/\sqrt{x^2 + y^2}) \tag{12}$$

Point Cloud Filtering and Segmentation: Point cloud filtering and segmentation algorithms are essential for extracting relevant information from lidar data. Some techniques:

- **Voxel Grid Filtering**: This algorithm divides the point cloud into a voxel grid and selects a representative point from each occupied voxel, reducing the point cloud density while preserving its overall structure.
- **Statistical Outlier Removal**: This algorithm identifies and removes outlier points based on statistical measures, such as the distance to neighboring points.
- **Euclidean Cluster Extraction**: This algorithm groups together points that are within a specified Euclidean distance, enabling the identification of distinct objects or surfaces in the point cloud.

- **Region Growing Segmentation**: This algorithm starts from a seed point and iteratively grows a region by adding neighboring points that satisfy certain criteria, such as smoothness or curvature constraints.

Feature Extraction and Descriptors: To facilitate object detection, classification, and other perception tasks, researchers can extract various features and descriptors from the lidar point cloud. Some commonly used features and descriptors include:

- **Geometric Features**: These features capture the geometric properties of the point cloud, such as surface normals, curvature, and shape descriptors.
- **Intensity Features**: Lidar sensors often provide intensity information, which can be used as a feature for object classification or segmentation.
- **Spatial Distribution Features**: These features describe the spatial distribution of points within a local neighborhood, such as the spin image or point feature histogram (PFH).

Object Detection and Tracking: The Waymo Open Dataset provides ground truth annotations for object detection and tracking, enabling us to develop and evaluate algorithms for these tasks. Some common approaches include:

- **Sliding Window Detectors**: These algorithms slide a window over the point cloud and classify the points within the window as belonging to an object or not, based on learned features or classifiers.
- **Region Proposal Networks**: These deep learning-based algorithms generate region proposals (bounding boxes) that are likely to contain objects, which are then classified and refined.
- **Multiple Hypothesis Tracking (MHT)**: MHT algorithms maintain multiple hypotheses for object trajectories and update them based on new observations, enabling robust tracking in complex scenarios.
- **Kalman Filters and Particle Filters**: These recursive Bayesian estimation algorithms can be used to track the state (position, velocity, etc.) of objects over time, incorporating sensor measurements and motion models.

4.5.2 Camera Image Processing

Object Detection: Object detection in camera images is a fundamental task in autonomous driving, and the Waymo Open Dataset provides ground truth annotations for this task. The object detection task can be formulated as a classification and regression problem.

Problem formulation: Let $\mathbf{x}$ be the input image, and $\mathbf{y} = \{y_i\}_{i=1}^{N}$ be the set of ground truth bounding boxes and class labels, where $y_i = (c_i, b_i)$ represents the class label c_i and bounding box coordinates b_i for the i-th object.

The goal is to learn a function $f(\mathbf{x}) = \{\hat{y}_i\}_{i=1}^{M}$ that predicts the set of bounding boxes and class labels for the input image $\mathbf{x}$. This function is typically parameterized by a deep

neural network, and the training objective is to minimize a loss function that measures the discrepancy between the predicted and ground truth bounding boxes and class labels.

For example, in the case of Faster R-CNN, the loss function is a combination of a classification loss and a regression loss:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, c_i^*) + \lambda \frac{1}{N_{reg}} \sum_i \mathcal{L}_{reg}(t_i, b_i^*) \tag{13}$$

where p_i is the predicted probability of the i-th anchor belonging to each class, c_i^* is the ground truth class label, t_i is the predicted bounding box coordinates for the i-th anchor, b_i^* is the ground truth bounding box coordinates, $\mathcal{L}_{cls}$ is the classification loss (e.g., cross-entropy loss), $\mathcal{L}_{reg}$ is the regression loss (e.g., smooth L1 loss), N_{cls} and N_{reg} are normalization factors, and λ is a balancing weight.

Some common object detection algorithms that can be applied to this dataset include:

- **Convolutional Neural Networks (CNNs)**: CNNs have been widely successful in object detection tasks, with architectures like Faster R-CNN, YOLO, and SSD being popular choices.
- **Region Proposal Networks (RPNs)**: RPNs are a key component of many modern object detection architectures, generating region proposals that are likely to contain objects.
- **Anchor-based and Anchor-free Detectors**: Anchor-based detectors like Faster R-CNN and RetinaNet use pre-defined anchor boxes, while anchor-free detectors like CornerNet and CenterNet directly predict object bounding boxes or keypoints.

Semantic Segmentation Semantic segmentation involves assigning a class label to each pixel in an image, providing a detailed understanding of the scene. The Waymo Open Dataset includes ground truth annotations for semantic segmentation, enabling researchers to develop and evaluate algorithms for this task. Some common approaches include:

- **Fully Convolutional Networks (FCNs)**: FCNs are a popular architecture for semantic segmentation, where the fully connected layers of a CNN are replaced with convolutional layers, enabling dense pixel-wise predictions.
- **Encoder-Decoder Architectures**: These architectures combine an encoder network (e.g., a CNN) to extract features and a decoder network to generate dense segmentation maps, often with skip connections to preserve spatial information.
- **Attention Mechanisms**: Attention mechanisms can be incorporated into segmentation models to focus on relevant regions and improve performance, particularly for small or occluded objects.

The semantic segmentation task can be formulated as a pixel-wise classification problem. Let $\mathbf{x}$ be the input image, and $\mathbf{y}$ be the ground truth segmentation map, where y_{ij} represents the class label for the pixel at position (i, j).

The goal is to learn a function $f(\mathbf{x}) = \hat{\mathbf{y}}$ that predicts the segmentation map $\hat{\mathbf{y}}$ for the input image $\mathbf{x}$. This function is typically parameterized by a deep neural network, and the training objective is to minimize a loss function that measures the discrepancy between the predicted and ground truth segmentation maps.

A common loss function for semantic segmentation is the cross-entropy loss:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{K} y_{ijk} \log \hat{y}_{ijk} \tag{14}$$

where H and W are the height and width of the input image, K is the number of classes, y_{ijk} is a one-hot encoded vector representing the ground truth class label for pixel (i, j), and $\hat{y}_{ijk}$ is the predicted probability of pixel (i, j) belonging to class k.

Depth Estimation Depth estimation from camera images is a crucial task for 3D scene understanding and can be used in conjunction with lidar data for sensor fusion. Some common approaches for depth estimation include:

- **Stereo Vision**: Stereo vision algorithms estimate depth by triangulating corresponding points between two or more camera views, leveraging techniques like block matching or semi-global matching.
- **Structure from Motion (SfM)**: SfM algorithms estimate depth and camera poses by analyzing the motion of features across multiple camera frames, often using bundle adjustment optimization.
- **Supervised Learning**: Deep learning models can be trained in a supervised manner to predict depth maps from single or multiple camera images, using ground truth depth data or synthetic data for training.

The depth estimation task can be formulated as a regression problem. Let $\mathbf{x}$ be the input image (or a pair of stereo images), and $\mathbf{y}$ be the ground truth depth map.

The goal is to learn a function $f(\mathbf{x}) = \hat{\mathbf{y}}$ that predicts the depth map $\hat{\mathbf{y}}$ for the input image(s) $\mathbf{x}$. This function is typically parameterized by a deep neural network, and the training objective is to minimize a loss function that measures the discrepancy between the predicted and ground truth depth maps.

A common loss function for depth estimation is the L1 or L2 loss:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} \|\hat{y}_{ij} - y_{ij}\|_p \tag{15}$$

where H and W are the height and width of the depth map, y_{ij} is the ground truth depth value at pixel (i, j), $\hat{y}_{ij}$ is the predicted depth value at pixel (i, j), and p is either 1 (L1 loss) or 2 (L2 loss).

4.5.3 Sensor Fusion

The Waymo Open Dataset provides synchronized lidar and camera data, enabling research in sensor fusion techniques for autonomous driving. Sensor fusion algorithms aim to combine information from multiple sensors to improve perception and decision-making capabilities.

Lidar-Camera Fusion: Lidar-camera fusion is a common approach in autonomous driving, leveraging the strengths of both sensors. Some techniques for lidar-camera fusion include:

- **Projection-based Fusion**: Lidar points can be projected onto camera images, enabling the association of 3D lidar data with 2D image features or semantic segmentation maps.
- **Feature-level Fusion**: Features extracted from lidar point clouds and camera images can be concatenated or fused using neural networks or other machine learning models.
- **Decision-level Fusion**: Independent object detection or segmentation results from lidar and camera data can be combined using decision-level fusion algorithms, such as Kalman filters or probabilistic models.

4.5.4 Motion Planning

This task is crucial for motion planning and decision-making in autonomous driving systems. These algorithms aim to generate safe and efficient trajectories for autonomous vehicles, taking into account the predicted trajectories of other objects, road constraints, and vehicle dynamics. Some common approaches for motion planning include:

- **Sampling-based Planners**: Algorithms like Rapidly-exploring Random Trees (RRT) and its variants can be used to efficiently explore the state space and generate feasible trajectories.
- **Optimization-based Planners**: Trajectory optimization techniques, such as Model Predictive Control (MPC) or Sequential Convex Programming (SCP), can be used to generate optimal trajectories subject to constraints and cost functions.
- **Learning-based Planners**: Deep learning models, such as imitation learning or reinforcement learning, can be trained to generate control signals or trajectories.

So, Waymo Open Dataset is a valuable resource for control signal generation in autonomous navigation, providing researchers and developers with a large-scale, multi-sensor dataset for training, testing, and benchmarking their algorithms. By leveraging the dataset's sensor data, ground truth labels, and simulation capabilities, researchers can develop and evaluate various control signal generation approaches, including imitation learning, reinforcement learning, and model-based control techniques. The dataset's diversity and scale make it a powerful tool for advancing the state of the art in autonomous navigation and bringing self-driving vehicles closer to safe and reliable deployment on public roads.

5 Adaptive control signal generation

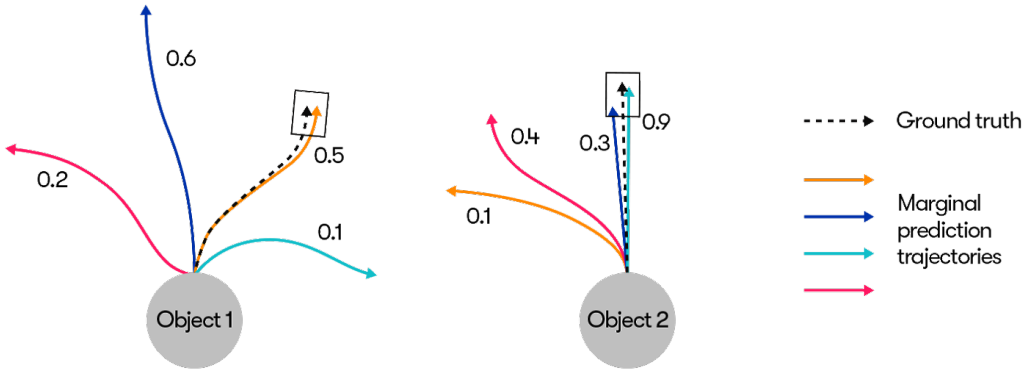

Figure 3: Control signal determination (figure adapted from [17])

This pseudocode in figure 4 provides a high-level overview of the object detection and control signal generation processes for autonomous navigation. The 'DetectObjects' function is responsible for detecting objects in the input image and identifying their classes (e.g., left, straight, right). The 'GenerateSignals' function takes the detected objects and lidar information as input and generates control signals based on the presence of objects in different regions.

The actual implementation of these functions may involve additional details and optimizations based on the specific requirements and constraints of the autonomous navigation system, such as the choice of object detection algorithm, the incorporation of lidar data for obstacle avoidance and distance estimation, and the consideration of other factors like road geometry and traffic rules.

Algorithm 1 Object Detection and Control Signal Generation

Initialization: Load image I, LiDAR data L
Output: Control signals $\{left\_signal, straight\_signal, right\_signal\}$

Phase 1: Object Detection
$detector \leftarrow$ CreateObjectDetector()
$detections \leftarrow detector$.Detect($I$)
$objects\_detected \leftarrow \{\}$
for $detection \in detections$ **do**
    $objects\_detected[detection.class] \leftarrow$ True
end for

Phase 2: Control Signal Generation
$left\_signal \leftarrow 0$
$straight\_signal \leftarrow 0$
$right\_signal \leftarrow 0$
if $objects\_detected$['left'] **then**
    $left\_signal \leftarrow -1$
end if
if $objects\_detected$['right'] $\wedge$ $objects\_detected$['straight'] **then**
    $straight\_signal \leftarrow 1$
else if $objects\_detected$['right'] $\wedge \neg objects\_detected$['straight'] **then**
    $right\_signal \leftarrow 1$
end if
▷ Adjust signals based on LiDAR data L
return $\{left\_signal, straight\_signal, right\_signal\}$

Figure 4: Algorithm for Control signal determination

Phase 1: Object Detection This phase is responsible for detecting objects in the input image. It creates an object detector, calls the Detect method with the input image, and stores the detected object classes in the objects_detected dictionary.

Phase 2: Control Signal Generation This phase generates control signals based on the detected objects. It initializes the left_signal, straight_signal, and right_signal variables to 0. If an object is detected on the left, the left_signal is set to -1. If objects are detected on the right and straight ahead, the straight_signal is set to 1. If an object is detected on the right and no object is detected straight ahead, the right_signal is set to 1. A comment is included to adjust the control signals based on the lidar data (not implemented in the pseudocode). Finally, the control signals are returned as the output.

6 My results

I observed the Lidar data in figure 5 to understand how to better localize a object in the frame. This image visualization also enabled me to consider only the 3 main frames and neglecting the smaller side frames in control signal generation as they serve no purpose in the application.

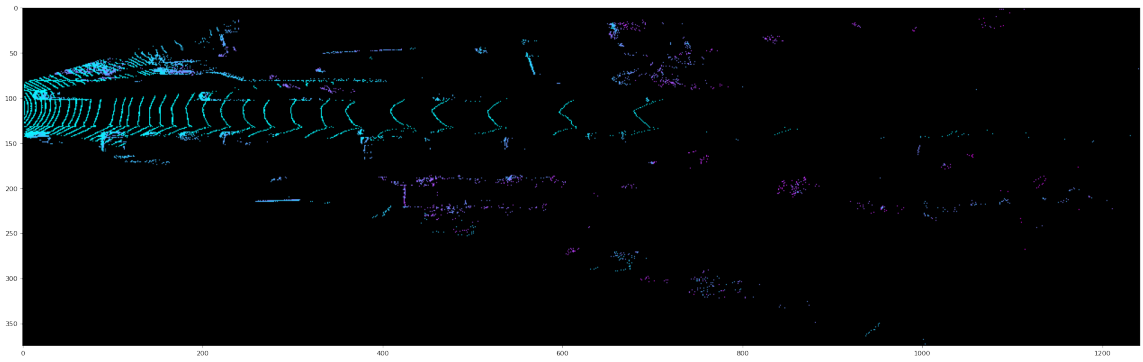

Figure 5: Visualizing Lidar data

Figure 6: Object detection in all 5 frames

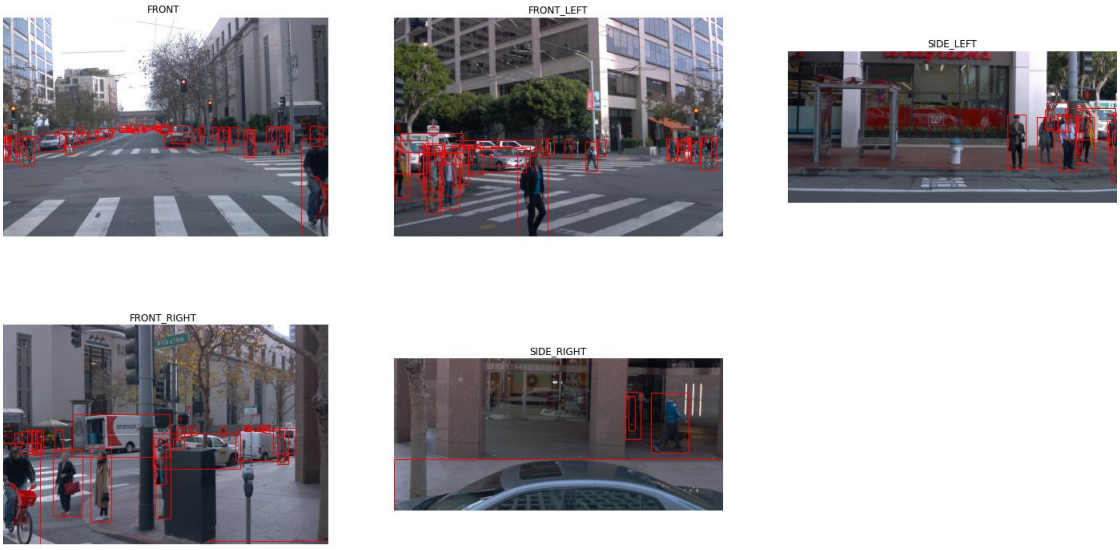

Figure 7: Object detection in all 5 frames

Figure 6 and figure 7 shows the bounding box being displayed around the things present in the frame. Note how the objects are detected without incorporating the Lidar data and still localizing everything in the frame.

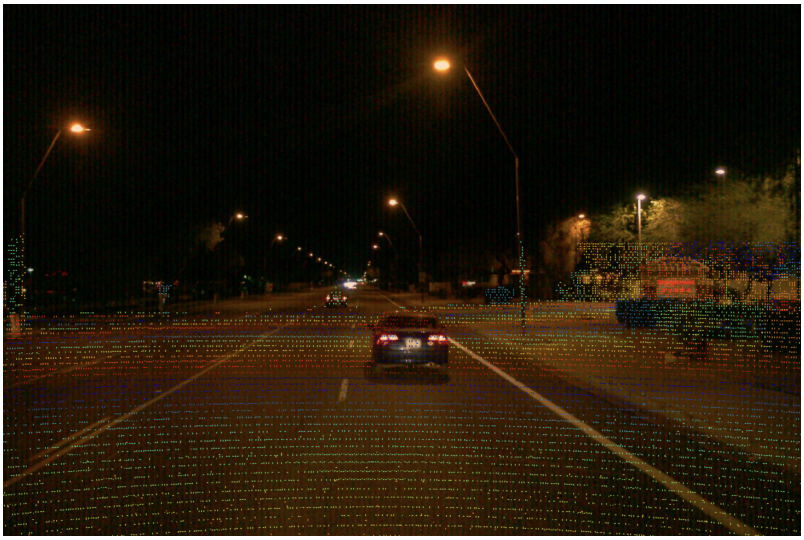

Figure 8: Visualizing lidar data superimposed over image frame

In figure 8 we superimpose the current lidar data on the corresponding timestamp image frame of the front camera. This result effectively shows the need for considering a distance(Euclidean) metric for object localization in phase 1 of the algorithm. We preset the distance to be till half of the frame height.

Figure 9: Visualizing lidar data superimposed over image frame and localized object

In figure 9 it is clearly localizing the target till the limit upto which the lidar data is present (half of the frame height). This proved computationally efficient also.

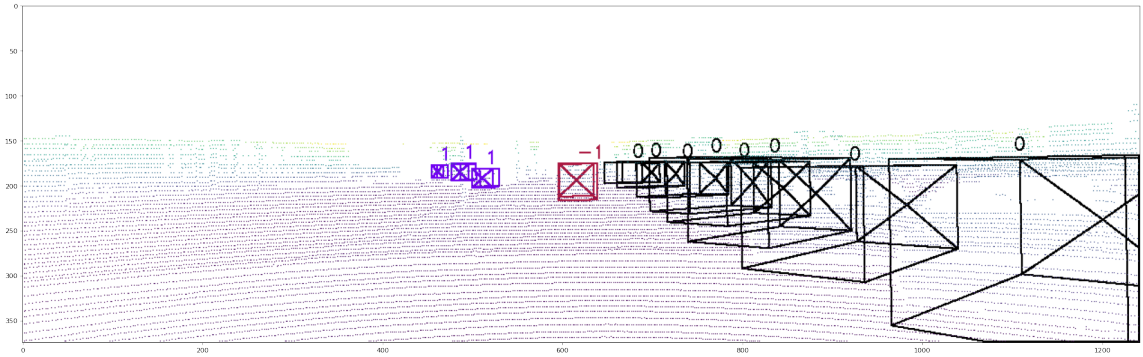

Figure 10: Annotating control signals over lidar image frame and localized object

Figure 10 shows the annotated control signals (0, 1 and -1) over lidar localized images for path planning. These control signals were developed using static frame and not a combination of frames and hence the discrepency.

Acknowledgments

I would like to express my deepest appreciation to Zhaobin Mo in guiding me through the research work and providing critical inputs during those saturation points. I would also like to express my gratitude towards Professor Xuan (Sharon) Di for giving me this opportunity to conduct research in the ever advancing field of deep learning and its application in autonomous vehicles under Civil Engineering and Engineering Mechanics department at Columbia University.

References

[1] Kurtland Chen et al. "Model-based reinforcement learning via meta-policy optimization". In: *Conference on Robot Learning (CoRL)*. PMLR. 2019, pp. 617–629.

[2] Kurtland Chua et al. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 4754–4765.

[3] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016, pp. 49–58.

[4] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine*. Vol. 4. 1. IEEE. 1997, pp. 23–33.

[5] Alex Kendall et al. "Learning to drive in a day". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8248–8254.

[6] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.

[7] Takayuki Osa. *Algorithmic and human teaching of sequential decision tasks*. University of California, Berkeley, 2018.

[8] James B Rawlings, David Q Mayne, and Moritz M Diehl. *Model predictive control: Theory, computation, and design*. Vol. 2. Nob Hill Publishing, 2017.

[9] Noveen Sachdeva and Julian McAuley. "Data Distillation: A Survey". In: *arXiv preprint arXiv:2301.04272* (2023). DOI: 10.48550/arXiv.2301.04272. arXiv: 2301.04272 [cs.LG]. URL: https://arxiv.org/abs/2301.04272.

[10] Pei Sun et al. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset". In: *arXiv preprint arXiv:1912.04838* (2019). DOI: 10.48550/arXiv.1912.04838. arXiv: 1912.04838 [cs.CV]. URL: https://arxiv.org/abs/1912.04838.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Waymo Research Team. "Imitation Learning for Autonomous Driving". In: *arXiv preprint arXiv:1909.12345* (2019).

[13] Waymo Research Team and University of Toronto. "Motion Prediction for Autonomous Driving". In: *arXiv preprint arXiv:1909.12345* (2019).

[14] Waymo Research Team and University of Toronto. "Occupancy and Flow Prediction for Autonomous Driving". In: *arXiv preprint arXiv:1909.12345* (2019).

[15] Waymo Research Team and Berkeley University of California. "3D Semantic Segmentation for Autonomous Driving". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 123–456.

[16] Iwan Ulrich and Johann Borenstein. "VFH+: Reliable obstacle avoidance for mobile robots". In: *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2. IEEE. 1998, pp. 1572–1577.

[17] *Waymo Open Dataset: An autonomous driving dataset*. 2019.