Bridging the Gap Between Perception and Navigation

A Self-Driving and LiDAR Perspective

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Mohd Omama
2020701006
mohd.omama@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
May 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " **Bridging the Gap Between Perception and Navigation - A Self-Driving and LiDAR Perspective** " by **Mohd Omama**, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. K. Madhava Krishna

.

To *Al-Qadr* (the destiny)

Acknowledgments

All praise is to **Allah**. *He taught man what he did not know (96:1).*

First and foremost, I would like to express my unparalleled gratitude toward my supervisor, Dr. Madhava Krishna, without whom a career in robotics would have remained a distant dream for me. He supported me at every step of my research journey with his exceptional guidance. I have gained unprecedented research maturity here at RRC, and a significant credit for that goes to Dr. Madhava. Further, I would like to thank my co-advisors, Dr. Sandeep Chinchali, Dr. Arun Singh, and Dr. Krishna Murthy, for their fundamental contributions throughout my research at RRC. I also thank my co-author, Sundar, for his central contributions to both of the publications associated with this thesis.

My time at RRC was toiling and difficult. I struggled a lot, learned a plethora of new skills, handled tough times of the pandemic while continuing research on campus, met many new people, and made unforgettable memories. None of this would have been possible if it wasn't for certain special people, including Shantanu, Shubodh, Udit, Karnik, Rahul, Ayappa, Kinal, Harshit, Unni, Kaustab, Jhanvi, Swati, Ananth, Sudarshan, Rishabh, Sasi, Gireesh, Parth and others at RRC.

Finally, I would like to thank my parents and my brother for being such solid emotional support throughout my entire life and particularly during the process of my master's degree. Because of their unconditional love and prayers, I stand where I am today.

Abstract

We embark on a hitherto unreported problem of an autonomous robot (self-driving car) navigating in dynamic scenes in a manner that reduces its localization error and eventual cumulative drift or Absolute Trajectory Error. Modern autonomous vehicles (AVs) often rely on vision, LiDAR, and even radar-based simultaneous localization and mapping (SLAM) frameworks for precise localization and navigation. However, modern SLAM frameworks often lead to unacceptably high levels of drift (i.e., localization error) when AVs observe few visually distinct features or encounter occlusions due to dynamic obstacles. This work argues that minimizing drift must be a key desiderata in AV motion planning, which requires an AV to take *active* control decisions to move towards feature-rich regions while also minimizing conventional control cost. To do so, we have two distinct formulations of this problem.

In the first approach, we learn actions that lead to drift-minimized navigation through a suitable set of reward and penalty functions. We use Proximal Policy Optimization, a class of Deep Reinforcement Learning methods, to learn the actions that result in drift-minimized trajectories. We show, by extensive comparisons on a variety of synthetic, yet photo-realistic scenes made available through the CARLA Simulator, the superior performance of the proposed framework vis-à-vis methods that do not adopt such policies.

In the second approach, we first introduce a novel data-driven perception module that observes LiDAR point clouds and estimates which features/regions an AV must navigate towards for drift minimization. Then, we introduce an interpretable model predictive controller (MPC) that moves an AV toward such feature-rich regions while avoiding visual occlusions and gracefully trading off drift and control cost. Our experiments on challenging, dynamic scenarios in the state-of-the-art CARLA simulator indicate our method reduces drift up to 76.76% compared to benchmark approaches.

We further proposed a novel approach that combines differentiablity with classical odometry algorithms (ICP) to identify drift minimizing features in a point cloud. We also developed a modular and scalable platform for testing and deploying self-driving systems on a real car. This works sets up the groundwork for scalable active navigation in real world.

Contents

List of Figures

Chapter 1

Introduction

Autonomous Robotics has seen unprecedented growth in the past few decades, which has been largely contingent on the advent of new data-driven approaches for the core robotics problems like perception and planning. A lot of literature has emerged that tries to improve the perception (often referred to as localization or SLAM) capabilities of these embodied agents using a combination of learning-based solutions, multi-sensor setups, etc. Similarly, the planning domain (often referred to as navigation) has seen the emergence of new and better approaches using a spectrum of techniques ranging from sampling to optimization. *However, the link between the domains of perception and planning still remains a relatively unexplored area.*

Usually, the perception and planning systems are developed independently, wherein each module has a different objective. Alternatively, these two modules can have a shared objective. For example, the navigation system can be designed to solve for the goal-reaching objective (standard navigation objective) and localization (perception) objective simultaneously, such that it comes up with trajectories that minimize the localization error while reaching the goal. This link between localization and navigation is often referred to as active-navigation or active-localization and has achieved comparatively less traction from the robotics community. Even less explored, to the extent that the literature primarily non-existent, is the problem of active navigation in self-driving and LiDAR settings. "Can a self-driving vehicle equipped with a LiDAR come up with trajectories that can reduce the localization error?" is the central theme that is addressed in this thesis.

The following section discusses the necessary background required for a worthwhile reading of this thesis:

1.1 Background

1.1.1 Taxonomy of a Standard Robotics System

A standard robotics stack requires a varied set of different modules to work in perfect symphony. Each module has a specific function that contributes to the overall performance of the system. At high level, these modules are categorized as:

1.1.1.1 Perception

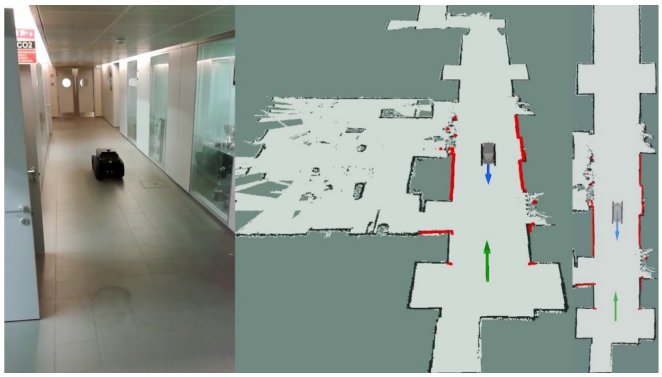

Figure 1.1 Output of a perception system in a mobile robot showing robot's current location estimate (blue) in the map as well as the free(white) and occupied(black) spaces.

The perception system endows the robot with the ability to perceive, comprehend, and reason about the surrounding environment. It enables to robot to have a sense of its motion (*odometry*), to pin point its location in a map (*localization*) and to build a map of its surrounding (*mapping*). Essentially, the *SLAM* (Simultaneous Localization and Mapping) problem falls under the umbrella of perception. This module also includes object-detection, tracking, free space estimation, etc. Throughout this thesis, we will use the terms *Odometry, Localization, SLAM, and Perception* to refer to robots ability to understand its current *SE3* state (position and orientation). Consequently, the better the perception system, the closer its *SE3* estimate is with respect to the ground truth and vice-versa.

1.1.1.2 Planning

Planning (or path planning) module is responsible for finding a collision-free path for a robot from its initial position to its goal position while avoiding obstacles. Path planners are designed to generate a set of states (positions or velocities) that the robot should follow to reach the goal from start while optimizing some criteria such as distance traveled, time taken, energy consumed, etc.

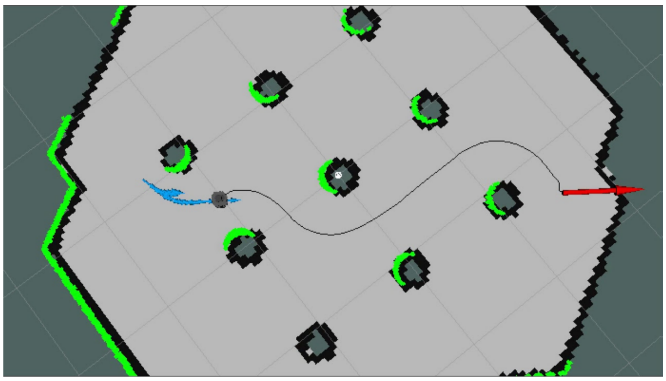

Figure 1.2 Output of a planning system in a mobile robot showing the navigation plan from the robot's current location (blue) to the goal location (red)

1.1.1.3 Control

Once a plan has been generated by the planning module, its manifestation is the handled by the control module. The control module is responsible for giving continuous commands in a feedback loop to the robot's actuation system so that the plan is executed as closely as possible. Various approaches can by incorporated for this purpose, including but not limited to classical feedback controllers(like *P, PD, PID* controllers), optimization based model-predictive controllers, etc. In this thesis, we use the terms *Planning, Control, and Navigation* to refer to the robot's ability to come up with a plan based on some cost. We assume that the robot has a low-level controller that can implement this plan.

1.1.2 LiDAR

LiDAR (Light Detection and Ranging) is a sensor technology that is used in self-driving cars to create a 3D map of the environment around the vehicle. A LiDAR system mounted on the roof or bumper of the car emits laser pulses that bounce off the objects in the vicinity. The system measures the time it takes for the pulses to return and calculates the distance and angle of the objects. The laser source emits light pulses that are reflected by the objects in the field of view. The scanner directs the laser beam across the scene and collects the reflected light. The detector converts the reflected light into electrical signals that are processed by the processor to generate a 3D point cloud of the scene. In this thesis, LiDAR is the primary sensory modality that we are looking into.

1.1.3 LiDAR Odomety and Mapping (LOAM)

LOAM or LiDAR Odometry and Mapping [38] is one of the seminal papers in LiDAR based SLAM systems. The LOAM paper proposes a real-time method for odometry and mapping using range measurements from 3D LiDAR moving in 6-DOF. The method divides the complex problem of simultaneous localization and mapping into two algorithms: one for high-frequency but low-fidelity odometry estima-

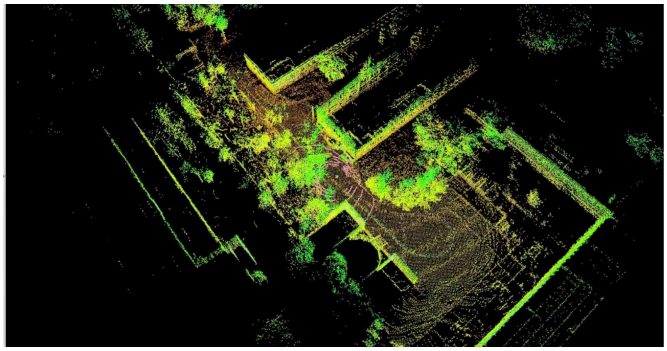

Figure 1.3 Output of LOAM algorithm showing the perceived point cloud and the robot's location estimate

tion, and another for low-frequency but fine matching and registration of the point cloud. The method achieves low localization errors and low computational complexity without the need for high accuracy ranging or inertial measurements. LOAM [38] is essentially an ICP [4] based algorithm wherein two consecutive scans are matched to get the relative motion (odometry) between the two. The method is evaluated by experiments and on the KITTI odometry benchmark, showing that it can achieve accuracy at the level of state of the art offline batch methods. LOAM (and its variants) have been the top performing algorihtms on the KITTI odometry benchmark and hence its the central perception algorithm that this theses focuses on.

1.1.4 Drift

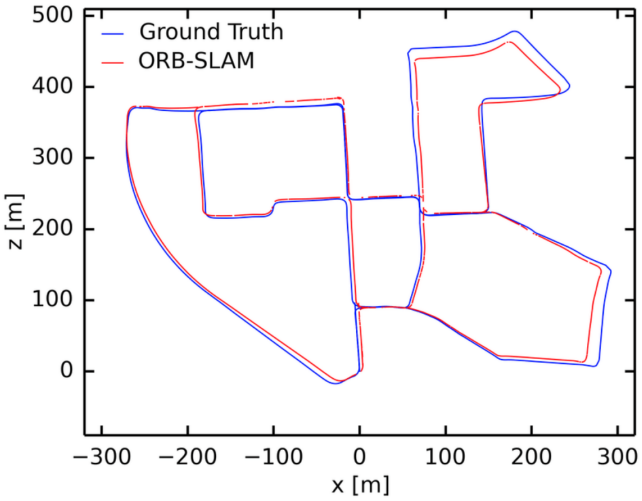

Figure 1.4 An example of drift accumulated by a SLAM algorithm on the KITTI dataset

The localization error or state estimation error in the robotics perception system is known as drift. As discussed in section 1.1.1 the job of the perception (localization) system is to accurately estimate the

robot's *SE3* state. The error accumulated in this process, i.e., the difference between the estimated and the ground-truth state is known as drift. In this thesis, whenever we mention an approach to "improve the perception system", etc., we implicitly mean that the approach leads to better localization estimates or lesser drift.

1.1.5 Active Navigation

As discussed earlier, perception and navigation are usually considered as decoupled problems. The approaches that try to improve the perception system usually incorporate information from multiple sensors, use more compute-heavy algorithms, etc. However, the perception system can also be improved by an intelligent navigation plan. A better navigation plan can lead to lesser drift without the use of multiple sensors or increase in computational cost. The idea that a robot's navigation system can have a perception improvement cost along with the standard navigation cost is referred to as Active Navigation (also referred to as active localization).

1.2 Contributions

In this thesis, we argue that the navigation system of an autonomous vehicles should not only optimize for control and goal-reaching costs but also a primary drift-minimization cost. Often, these are conflicting objectives because the best or shortest trajectory that an ego vehicle can take may not be the most drift-minimizing one. Thus, we require a multi-objective optimization framework to gracefully trade-off both criteria. The thesis is a culmination of two research articles that formulate and develop the active navigation problem in a LiDAR and self-driving setting, as well as subsequent works that lay the groundwork for scaleable active navigation in real-life. The major contributions can enumerated as follows:

1.2.1 Formulating Active Navigation Problem

We formalize the problem of active navigation in the LiDAR setting and show experimentally that its indeed possible to improve the perception system with a better navigation plan.

1.2.2 Active Navigation with Geometric Features and RL Based Planner

In the first research article, we proposed a novel reinforcement learning based approach that takes in the position of important features in a LiDAR map, defined using a geometric measure of curvature, and generates drift minimising trajectories. With a carefully designed reward function, the agent was able to learn behaviors like overtaking, slowing down, etc., to minimize drift. We show by extensive comparisons in a variety of dynamic scenes the efficacy and superiority of this architecture in achieving

drift-minimized navigation vis-à-vis methods that do not actively navigate to minimize state estimation errors. Henceforth, we call our system **LADFN** - **L**earning **A**ctions for **D**rift-**F**ree **N**avigation.

1.2.3 Active Navigation with Learned Features and Batch-Optimized Planner

Our first article LADFN [25], used an uninterpretable reinforcement learning (RL) which takes many iterations to train and generalizes poorly to new environments. In the second research article, we adopted a more systematic approach to significantly improve active navigation. A key insight was that a simple geometric measure is insufficient to robustly identify drift minimizing LiDAR points. Hence, a weekly supervised approach was used to learn drift minimized features using gradient based neural activation techniques, like GradCAM[32]. Further, instead of using an RL agent, we incorporated a GPU-driven batch optimized MPC (Model Predictive Control) for trajectory generation which can solved thousands of trajectories in real time over a long horizon. This gave us similar behaviors as the RL based planner of LADFN while being much more interpretable and secure.

1.2.4 Differentiable ICP and Autonomous Driving Platform (AutoDP)

To have a scalable active navigation system running on a real car, a lot of smaller milestones have to be achieved. We set the groundwork for a differentiable variant of LOAM that could learn to identify drift minimizing features using standard dataset like KITTI [16] without the need of custom dataset creation like in 1.2.3. Further, we developed a modular, general purpose autonomous driving platform (AutoDP) from ground-up on a real car that can be used to test active navigation. AutoDP is a complete software cum hardware platform for testing and deployment of self-driving systems and its scope is not limited to active navigation. It is a standardized platform that researchers and industrialists can use to develop, test and deploy their algorithms. It provides plug-and-play implementations of the state of the art algorithms for all the levels of the self-driving stack.
Though the implementation of active navigation algorithms on a real car are beyond the scope of this thesis, the work done in the areas of Differentiable ICP and AutoDP provide a solid starting point for future work in this direction.

1.3 Thesis Organisation

This thesis is divided into six chapters. First chapter highlights the problem of active navigation in a self-driving setting with LiDAR as the primary sensory modality. Second chapter provides the literature survey on active SLAM, active navigation, perception aware planning and localization in dynamic scenes. Third chapter elaborates our first contribution where we tackle active navigation with geometric features and reinforcement leaning based planner. Fourth chapter describes the second contribution where we tackle active navigation with learned features and a batch CEM-MPC planner. Fifth chapter covers the development of a custom autonomous driving platform on a real vehicle that can be used for

research and development in self-driving related domains. It also discusses the importance of developing differentiable variants of classical algorithms for a more scalable approach towards active navigation. Chapters 3-5 are the core technical chapters and each of them has a *prologue* and an *epilogue* that act as a summary of the major contributions, conclusions, drawbacks, and the scope of future work. Sixth chapter concludes the thesis and motivates further plausible work in the direction of active navigation in LiDAR and self-driving settings.

Chapter 2

Related Works

This thesis combines concepts from different domains such as state estimation and localization, LiDAR odomerty, active localization/navigation, etc. In this chapter, we summarize the available literature from each of these domains to provide the necessary context for the subsequent chapters.

2.1 State Estimation and SLAM

A typical state estimation or localization problem involves estimating the probabilistic state of the robot conditioned on the current observations, past states, and actions. State estimation comes in various flavors such those that recover only the last state (*filtering*) and those that recover all the past states (*smoothing*). Filtering based approached have been extremely popular in the robotics literature. A plethora of works have emerged that assumes the conditional likelihoods as multivariate Gaussians like the Kalman Filter and its variants [22]. Karman Filter essentially assumes a liner motion model of the robot. For non-linear motion models, improvements over the Karman Filter like the extended Karman filter (EKF) and the unscented Karman filter(URF) exist [36]. On the other hand, non-parametric approaches like particle-filtering [9] circumvent the need for a Gaussian assumption on the likelihoods. The advent of optimization based smoothing approaches for the localization/SLAM was a revolutionary change and led to the development of real-world large-scale localization systems. SLAM was modeled as a graph optimization problem in [3]. *Kaees and Dallart* introduced factor graphs to the SLAM optimization [10] followed by an incremental online variant *iSAM* [20]. Factor graphs have become the most popular choice to model the SLAM problem in the contemporary literature.

2.2 LiDAR Based Odometry and SLAM

LiDAR odometry is usually modelled as a scan matching problem wherein two consecutive LiDAR scans are matched to find the relative transform (often referred to as odometry) between them. Iterative scan matching methods such as ICP [4] and its variants [31] are the popular choice for LiDAR odometry. Most modern LiDAR odometry algorithms fall under the *smoothing* category of state-estimation. *Sanjiv*

and Zhang proposed a real-time variant of LiDAR based state estimation called LOAM or LiDAR Odomety and Mapping [38]. The method divides the complex problem of simultaneous localization and mapping into two algorithms: one for high-frequency but low-fidelity odometry estimation, and another for low-frequency but fine matching and registration of the point cloud. LOAM and its variants have been the state of the art in the LiDAR based SLAM benchmarks. Multiple improvments over loam were proposed, like LeGO-LOAM [33] which uses a plane segmentation based approach to remove unnecessary points in the scan matching process. LIO-SAM [34] proposed a variant of LOAM which incorporates infromation from multiple sensors and formulates the LiDAR odometry as a factor graph optimization problem.

2.3 Active Localization/Navigation

Traditional active localization frameworks typically move to regions that are low in state uncertainty. For example, [23] considers state and observation noise to be Gaussian and selects a trajectory from a candidate set by minimizing the trace of the Covariance Matrix. However, such methods cannot be extended to situations where noise priors are non-Gaussian; formulations in the presence of dynamic actors, such as other cars and pedestrians, become progressively difficult. Initial active localization frameworks in multi-hypothesis settings include [15], wherein the agent navigated actively to localize to a unimodal uncertainty from an initial multi-modal hypothesis that typically occurs due to perceptual aliasing. The complexity of active localization was analyzed in [12] and was shown to be NP-Hard, whereas a randomized algorithm for the same was presented in [28]. The first and possibly only such formulation for active localization in a multi-agent setting was presented in [5]. However, none of the above frameworks talk about actively navigating a self-driving car equipped with a LiDAR for better perception.

The work in the domain of perception-aware path planning by *Scaramuzza et al.* [8], aims to keep features with rich texture in a robot's field of view. Similarly [2] proposed a method to improve feature visibility for a robotic maneuver. Perhaps the closest work to this thesis is *Perception Aware MPC* [14] by *Falanga et al.,* which aims to unify the perception and planning objectives by proposing a controller that solves for goal reaching as well as tries to keep visual features in the field of view. The work aims to develop an optimization framework for a vision-based system mounted on a quad-copter such that the centroid of points of interest remains in the center of the camera while performing the required maneuver. However, the importance of features is defined in a vision domain and does not apply to LiDAR settings. Similarly, the controller developed is exclusive to quad-copters only. Further, the system works in constraint setups with only a few distinct visible features and won't scale to outdoor self-driving settings.

Chapter 3

Active Navigation with Geometric Features and Reinforcement Leaning Based Planner

3.1 Prologue

In this research article we argue that autonomous vehicles should not only optimize for control cost but also a primary drift minimization cost. Often, these are conflicting objectives because the best or shortest trajectory that an ego vehicle can take may not be the most drift-minimizing one (see Fig. 3.1). Thus, we require a multi-objective optimization framework to gracefully trade-off both criteria. We present a novel active navigation framework that seeks and learns actions that lead to and result in drift-minimized navigation in the presence of dynamic obstacles, as shown in Fig. 3.1. The framework is based on reinforcement learning paradigm and that takes in the position of important features in a LiDAR map, defined using a geometric measure of curvature, and generates drift minimising trajectories. With a carefully designed reward function, the agent was able to learn behaviors like overtaking, slowing down, etc., to minimize drift. We show by extensive comparisons in a variety of dynamic scenes the efficacy and superiority of this architecture in achieving drift-minimized navigation vis-à-vis methods that do not actively navigate to minimize state estimation errors. We call this reinforcement learning based system **LADFN** - **L**earning **A**ctions for **D**rift-**F**ree **N**avigation.

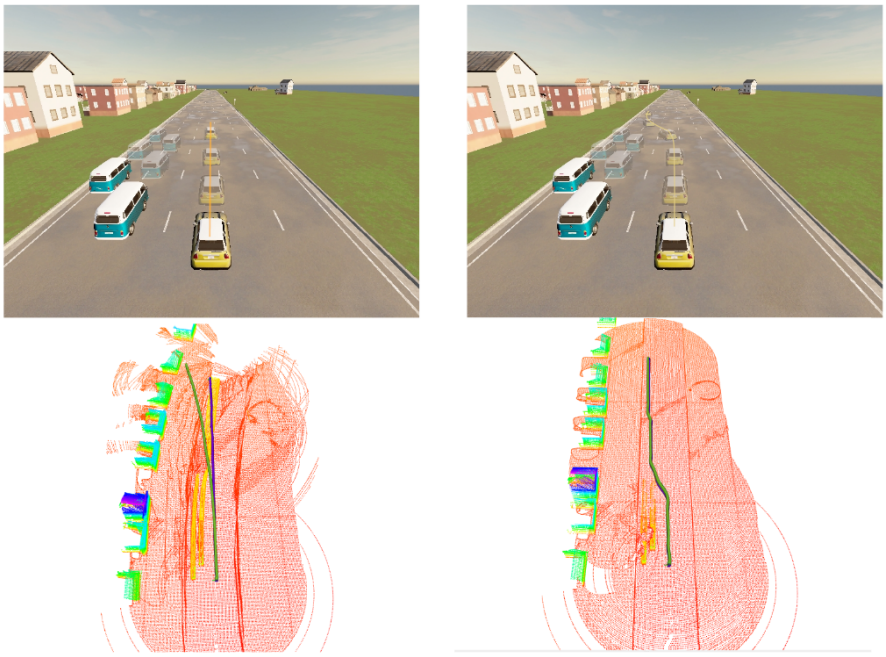

Figure 3.1 Active navigation significantly reduces autonomous vehicle drift. Left column represents baseline (Stanley Control) and **right column** represents our approach (LADFN). **Top Panel:** CARLA [11] environments showing ego vehicles planning their trajectories. **Bottom Panel:** LOAM [38] map visualizations of the same scenes. The green lines delineate ground-truth trajectories, and the purple lines delineate LOAM's generated odometry. While drift from the green ground-truth is pronounced on the left, with LADFN there is no such drift or deviation on the right. For a **100m** run, a Stanley-controlled ego drifts in the xy-plane by **14.05m**, eventually colliding with nearby traffic, whereas an ego controlled by our approach drifts by **0.62m**, due to traffic collision-avoidance and active navigation towards nearby features. This drift also affects LOAM's [38] mapping potential: the map on the left is distorted due to the ego's haphazard odometry, whereas the map on the right remains relatively unaffected.

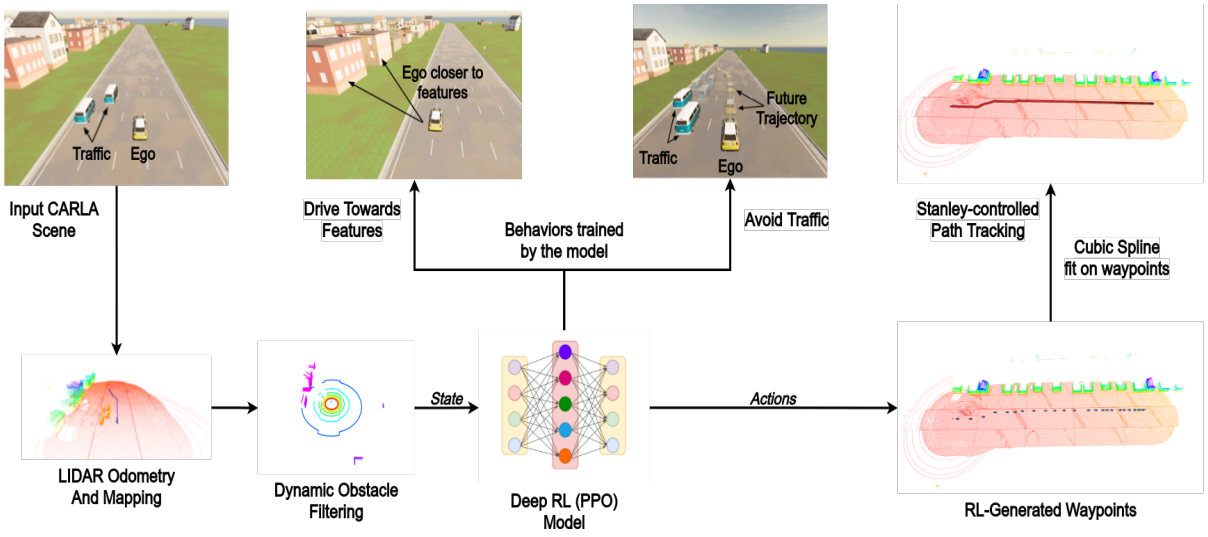

Figure 3.2 Pipeline of our approach. First, a map of the input CARLA [11] scene is built using LOAM [38]. Optionally, filtering is applied to remove dynamic traffic objects (To highlight the importance of active navigation even when an oracle is available to filter dynamic obstacles). The state obtained from this map is passed into a deep RL neural network that performs Proximal Policy Optimization [30], to help choose actions $a \in \mathcal{A}$, our action space. This model ensures that the ego drives towards feature-rich lanes and avoids traffic in its way. The action inferences from the model are then are used to generate a set of discrete waypoints for the ego to traverse. A cubic spline is fit on these waypoints to generate a smooth trajectory, which is then tracked using a Stanley Controller [18].

3.2 Pipeline Flow

The complete end-to-end architecture of LADFN is shown in Fig. 3.2. Our implementation has the following modules:

1. *Perception and Dynamic Obstacle Filtering*: We use an out-of-the-box implementation of LOAM [38]. Then, we filter LIDAR points originating from traffic vehicles.

2. *Waypoint Generation*: We use an RL model to choose actions that provide discrete waypoints for the ego to traverse. The model is unrolled for n steps to get a trajectory.

3. *Trajectory Interpolation*: We fit a cubic spline on the trajectory to smoothen it.

4. *Low-level Control*: We use a Stanley Controller [18] to perform path tracking on the trajectory.

3.2.1 Perception and Dynamic Obstacle Filtering

LIDAR Odometry and Mapping (LOAM) [38] is a SLAM technique [13] developed for 3D LIDAR sensors. LOAM splits the complex problem of optimizing several thousand variables simultaneously by splitting the mapping and localizing procedures into two: *laser odometry* and *laser mapping*. *Laser*

odometry runs at a high frequency and seeks to localize the LIDAR by calculating its velocity in every frame. *Laser mapping* runs at a lower frequency and finds edge-edge correspondences and plane-plane correspondences. These two processes are followed by iterative scan matching to get the translation T and rotation R. LOAM is a state-of-the-art odometry solution on several benchmark datasets, especially KITTI [17]. In our method, LOAM runs in the background interacting with the LIDAR points originating from CARLA. We use CARLA's [11] provided LIDAR segmentation to filter out points coming from dynamic vehicles and test our approach both in the presence and absence of this filtering. The odometry output from LOAM is piped into the waypoint generation module, discussed in detail in Section 3.2.2.

3.2.2 Waypoint Generation

The Waypoint Generation problem will be formulated as a Markov Decision Process (MDP) in Section 3.3. While there are a plethora of deep RL algorithms to approximately solve this MDP, we chose PPO [30] since it is well-established and well-suited for discrete action spaces that our setting requires. We also note that the primary novelty of our paper is the MDP formulation for active navigation to reduce drift, and other RL algorithms should perform well in practice. For both the actor and the critic, we have a fully connected network with 6 hidden layers. The number of neurons in each hidden layer are: 64, 64, 32, 32, 16 and 16 respectively. We use ReLU activation along with a batch-size of 128 and a discount factor γ of 0.9999. Our architecture conforms to the implementation of PPO in Stable-Baselines 3 [27].

We synthetically create a wide range of initial states. These states span across multiple combinations of ego positions, traffic positions, traffic speeds, positions of centroid of edge features, etc., During training, we assume that the position of centroid of edge features remains constant for that episode. This hypothesis is valid because our training episodes last for only 10 steps. We created an environment that takes as input at time t the state s_t, and the action a_t, and outputs at time $t+1$ the next state s_{t+1}, and reward r_{t+1}. This environment was used to train an RL agent without directly interacting with the CARLA [11] Simulator. The episode is terminated either at the completion of 10 steps, or when the ego breaches lane boundaries, or when the ego collides with the traffic.

3.2.3 Trajectory Interpolation

The waypoints obtained from PPO are then passed into a cubic spline planner for interpolation in order to obtain a smooth trajectory that our ego can traverse. During inference, we unroll the waypoint generation module n times to get an n-step trajectory. We then fit a cubic spline on it using an existing framework provided by Sakai *et al.* [29].

3.2.4 Low-level Control

The final step in our process is to traverse the smooth spline and ensure that the ego is tracking the path with minimal drift. For this purpose we use a standard Stanley Controller [18], but it should be noted that our approach is invariant to the choice of the path tracking algorithm. The implementation of Stanley used is also provided by [29].

3.3 Model-Free RL for Drift Minimization

Reinforcement Learning (RL) has been extensively used to solve a vast variety of Markov Decision Processes (MDPs). A canonical MDP comprises of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transitional probability, $\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, that governs the transition from state s_t to state s_{t+1}, a reward function, $\mathcal{R}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, which describes a scalar reward that is associated with the transition from state s_t to s_{t+1} by taking an action a_t, and a discount factor γ which controls the importance we give to future rewards. When we solve the MDP, we find a policy π: $\mathcal{S} \rightarrow \mathcal{A}$, such that it maximises the value function V_π:

$$V_\pi(s_0) = \mathbb{E}\Big(\sum_{t=0}^{\infty} \gamma^t \mathrm{R}\Big(s_{t+1}, s_t, \pi\big(s_t\big)\Big)\Big), \tag{3.1}$$

Now, we will formulate LADFN's waypoint generation problem as an MDP and provide a detailed account of the state space $\mathcal{S}$, the action space $\mathcal{A}$ and the reward $\mathcal{R}$. The dynamics of our MDP depend on the motion of the ego-vehicle as well as the uncontrolled motion of other traffic. Specifically, we cannot analytically model where other traffic will move, nor what the ego will observe at future time-steps from its LIDAR sensors. Since we do not have an analytical model for the full dynamics of the MDP, we resort to model-free deep RL to solve for a control policy. We first define a frame whose X-axis points along the direction of the road, and whose Y-axis is perpendicular to the direction of the road. The ego's starting coordinate in this frame will be of the form (0, y_e). We refer to this frame as the *RL frame* (Fig. 3.3).

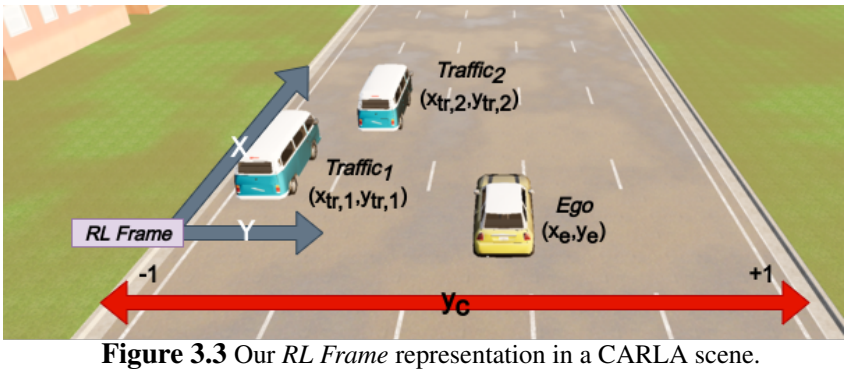

Figure 3.3 Our *RL Frame* representation in a CARLA scene.

3.3.1 State

We define the state of the RL agent as a vector such that it captures all necessary task-relevant information. The entries of the state space vector will be in *RL frame* unless stated otherwise. The state vector is as follows:

- x_e: x-coordinate of ego vehicle
- y_e: y-coordinate of ego vehicle
- y_c: y-coordinate of centroid of the edge features which is clipped and scaled to $[-1, 1]$; edge features are discussed in Section 3.3.3
- $edge_l$: left limit of the road
- $edge_r$: right limit of the road
- tr_{p_i}: Boolean representing the presence of i^{th} traffic vehicle
- x_{tr_i}: x-coordinate of the i^{th} traffic vehicle
- y_{tr_i}: y-coordinate of the i^{th} traffic vehicle
- v_{tr_i}: velocity of the i^{th} traffic vehicle.

3.3.2 Action Space

The RL agent has a set of discrete actions $\mathcal{A}$ that it can choose from. Each action $a \in \mathcal{A}$, represents the waypoint at the next time step for the vehicle in the vehicle's frame of reference. The action a is defined as a tuple (a_x, a_y) where $a_x \in \{1, 2, 3, 5\}$ represents x-coordinate of the next waypoint, and $a_y \in \{-2, , -1, 0, 1, 2\}$ represents y-coordinate of the next waypoint in the vehicle's frame. We ensured that our action space reflects a spectrum of ways in which the vehicle could move: from slow-straight motion to fast-turning motion.

3.3.3 Motivation For Reward Shaping

LOAM [38] geometrically categorizes the LIDAR points as edge or plane features which are then used in scan-matching. Edge features represent texture-rich regions which help in accurate scan-matching. We will refer to such regions as *feature-rich.* In order to sustain LOAM's mapping process, and more importantly, reduce the absolute pose error (APE) in ego-motion, the first consideration is the ego's proximity to feature-rich regions. This can be argued as follows.

Consider two vehicles ego_1 and ego_2 and a feature-rich region T_R as shown in Fig. 3.4, where ego_1 is closer to T_R than ego_2. We draw tangents from the two vehicles to T_R. Let the angles that these two sets of tangents subtend with T_R be θ_1 and θ_2 respectively. Also, consider a planar LIDAR which

has a 360° field view of a single plane and an angular resolution of n. The number of LIDAR rays per unit angle will then be $\frac{n}{360}$. Now, the number of LIDAR rays that interact with the feature-rich region in both the cases will be $\frac{n \times \theta_1}{360}$ and $\frac{n \times \theta_2}{360}$ respectively. Since, ego_1 is closer to T_R than ego_2, θ_1 will be larger than θ_2, indicating that more LIDAR rays will interact with T_R in the first case leading to its more dense reconstruction. The same argument can be extended for 3D LIDAR which is the primary hardware setting in our proposed solution. Therefore, when the ego is in the proximity of texture-rich regions, LOAM can more densely reconstruct them since the incoming pointcloud to LOAM will have more edge features contained in it.

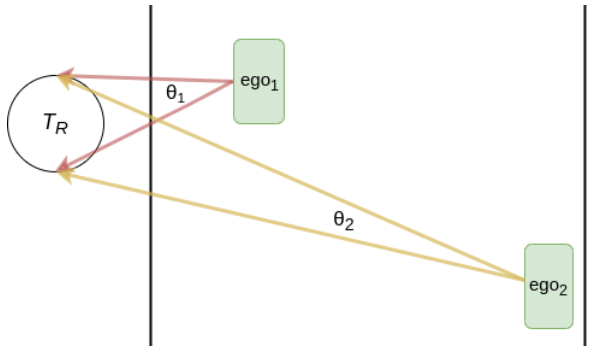

Figure 3.4 Benefits of active navigation towards feature-rich regions. More LIDAR rays from $\mathbf{ego_1}$ than $\mathbf{ego_2}$ interact with the texture-rich region T_R, since $\theta_1 > \theta_2$, resulting in a more dense reconstruction.

The second aspect that plays an impact on the APE is the distance from traffic vehicles. LIDAR points from dynamic objects lead to erroneous registrations in the scan-matching algorithm; in our case, the dynamic objects are said traffic vehicles. Further, proximity to dynamic vehicles can also occlude visibility of other texture-rich regions. Follow-up papers [35, 39] on LOAM have argued the same. Further, Thomas *et al.* [35], have specifically shown that the proximity to dynamic vehicles leads to substantial error in the rotational measurement, R. Our experimentation in dynamic scenes also resonates with the findings of these works but unlike us, none of them address drift-minimizing behaviors.

To handle the problems described above, the ego should be aware of the feature-rich regions and plan its trajectory accordingly. It should also avoid proximity with traffic wherever possible. Our proposed solution comprises of an RL agent that generates waypoints which are then traversed with the aid of a low-level controller. The input to the RL agent is a state that includes the centroid of edge features, the ego's pose, and the traffic vehicles' poses. We calculate the centroid using queried edge features obtained through LOAM. The pose of the ego is directly queried from LOAM. We also assume that we have the speed and pose information of the traffic readily available due to recent advancements in LIDAR-based semantic segmentation and tracking [24].

3.3.4 Reward

The reward was designed with two primary objectives: navigation and perception. The navigation objectives comprise of the follows.

1. Forward motion: We incentivize the ego to move forward towards the goal.

2. Road boundary constraint: $edge_l \leq y_e \leq edge_r$, i.e., the ego should move within the road boundaries.

3. Collision constraint: $\|(x_{tr_i}, y_{tr_i}) - (x_e, y_e)\| \geq 3$ meters, i.e., the ego must not collide with the traffic.

The perception objectives comprise of the follows.

1. Feature proximity: *minimize* $|y_e - y_c|$, i.e., the ego should try to maintain proximity with the feature-rich regions.

2. Traffic distance: $\|(x_{tr_i}, y_{tr_i}) - (x_e, y_e)\| \leq 10m$, where the ego should try to avoid proximity with traffic.

Our reward $\mathcal{R}$ models both the objectives described above and is defined as:

$$\mathcal{R} = \mathcal{P} \times \mathcal{T} + (1 - \mathcal{P}) \times \mathcal{F} + \mathcal{G}, \tag{3.2}$$

where $\mathcal{P} = \vee p_i$, which is the logical OR over all the *traffic-proximity* Boolean p_i. p_i will be described in detail in the *Traffic Reward* section. We now explain the other three terms in equation (3.2).

3.3.4.1 The Goal Reaching Reward, $\mathcal{G}$

$\mathcal{G}$ returns a positive reward for forward motion along the X-axis towards the goal. Although a simple term, this ensures that the ego prioritizes moving towards the finish line besides pursuing feature-rich regions and avoiding traffic. The other navigation constraints like traffic and road limits are also modeled here. The ego is penalized if it collides with the traffic or moves out of the road, and the episode terminates.

3.3.4.2 The Feature Reward, $\mathcal{F}$

The feature reward needs to be shaped in a way that the ego is rewarded if it maintains proximity with feature-rich regions. To achieve this, we need two behaviors that are captured in two terms in (3.3) as follows:

3.3.4.2.1 The first term The ego should move towards the left if the left side of the road is feature-rich and vice-versa. The lateral position of the feature-rich region is described by y_c, which is clipped and scaled to a normalized range of $[-1, 1]$. A value of -1 means the centroid of features is towards the left end of the road, and a value of 1 means the centroid is towards the right. To formulate the feature reward, we also need y_e in the range $[-1, 1]$. The term $(2((y_e - y_{e\min})/(y_{e\max} - y_{e\min})) - 1)$ handles that. We first raise y_c to an odd power, O, and then multiply it with normalized y_e. This means that the reward will be positive if both y_c and normalized y_e have the same sign, implying they are on the same side of the road; and the reward will be negative if they are on the opposite sides of the road. Further, y_c^O forces the term following it to quickly drop to 0 when y_c is close to 0, ensuring that this *first term* has no impact when there are feature-rich regions on both sides of the ego.

3.3.4.2.2 The second term If there are features on both sides of the road, the ego must prevent unnecessary lateral motion. To achieve this, we multiply the absolute of our lateral motion $|a_y|$ with $(1 - |y_c^O|)$. In this case, $|y_c^O|$ will be close to zero since features are on both sides and $(1 - |y_c^O|)$ will tend to 1. Further, $(1 - |y_c^O|)$ drops to zero when y_c is close to 1, ensuring that this second term has no impact when there are features on only one side of the road (left or right).

The complete mathematical formulation of the feature reward is given by:

$$\begin{aligned}\mathcal{F} = K_1 \times \left(y_c^O \times \left(2 \times \frac{y_e - y_{e\min}}{y_{e\max} - y_{e\min}} - 1 \right) \right) \\ - K_2 \times (1 - |y_c^O|) \times |a_y|,\end{aligned} \tag{3.3}$$

where K_1 and K_2 are constants.

3.3.4.3 The Traffic Reward, $\mathcal{T}$

We define a *traffic-proximity* region $\|(x_{tr_i}, y_{tr_i}) - (x_e, y_e)\| \leq 10m$ where the effect of traffic is large on localization. The traffic reward should be formulated such that the ego tries to move out of this region. We design a reward which incentivizes the ego if the relative difference in forward direction between the ego and the traffic increases. It also returns a positive reward if the ego increases its lateral distance with the traffic car, and a negative reward otherwise. The traffic reward can be mathematically written as:

$$\mathcal{T} = \sum_{i=1}^{n} p_i \Big[\big(K_3 \times |a_x - v_{tr_i}| \big) + K_4 \times \big(|y_{tr_i}^t - y_e^t| - |y_{tr_i}^{t-1} - y_e^{t-1}| \big) \Big], \tag{3.4}$$

where p_i is a Boolean which is 1 only when the ego is in a *traffic-proximity* region, otherwise this traffic-vehicle has no impact on the reward, and K_3 and K_4 are constants. The summation term outside accumulates the rewards/penalties from each traffic vehicle i.

The waypoint generation MDP is solved using deep RL using an architecture described in Section 3.2.2. We now evaluate our RL-based active navigation framework, LADFN, in challenging autonomous driving scenarios.

3.4 Experiments

3.4.1 Experimental Setup

The performance of LIDAR odometry algorithms can depend a lot on the physics of the environment as well as that of the ego. Hence, we use CARLA simulator [11] for our experimentation as it emulates real-world physics accurately. It is also a state-of-the-art high-fidelity simulation environment for autonomous driving and so it is ideal for our application. We simulate a 3D LIDAR in CARLA with 16 channels and a vertical field of view of 30° (-15° to 15°). The LIDAR emits 300,000 points per second which are evenly distributed over all channels. While running our experiments, we use three different LIDAR ranges: 45m, 50m, and 55m. These ranges correspond to the the maximum distance that can be measured by the LIDAR. The variety of LIDAR ranges during the experiments helps us analyze the performance of our method for various sensor ranges.

3.4.2 Scene Description

We have created a test bed of 5 different scenes in CARLA [11]. These scenes have a run length ranging from 100 to 500 meters, and include static as well as dynamic scenarios. We have created the scenes such that they have varying distributions of feature-rich regions and dynamic vehicles. Scenes 1 and 4 are static and don't have any dynamic traffic actors whereas scenes 2, 3 and 5 contain moving traffic cars or vans around the ego. The results shown have up to two dynamic vehicles, but the method is easily scalable to an arbitrary traffic count.

3.4.3 Metrics and Benchmarks

We benchmark our approach against what we call the *Vanilla Stanley Controller*. With *Vanilla Stanley*, the ego will not actively change its direction or speed to account for drift minimization. The ego will drive with a speed that is similar to the traffic around it. Hence, it represents *perception-unaware control*, a realistic scenario where the ego is tracking a trajectory without having any idea of the perception goals. Since there is no previous work in this area, Vanilla Stanley is an ideal benchmark to compare against.

In dynamic scenes, a general approach to decrease trajectory error is to filter out dynamic vehicles from the LIDAR scans. Hence we have another benchmark which we refer to as *Vanilla Stanley + F* which is just a *Vanilla Stanley Controller* running with filtering enabled. We test and compare our proposed approach both in the presence and absence of dynamic obstacle filtering. This helps us understand that even when filtering is available, active navigation is essential to further reduce drift in the trajectory.

We use three different metrics for evaluating our approach:

1. *Average Drift:* The average translation drift in xy-plane for the entire trajectory.

2. *Final Drift:* Final translation drift in xy-plane at the end of the trajectory.

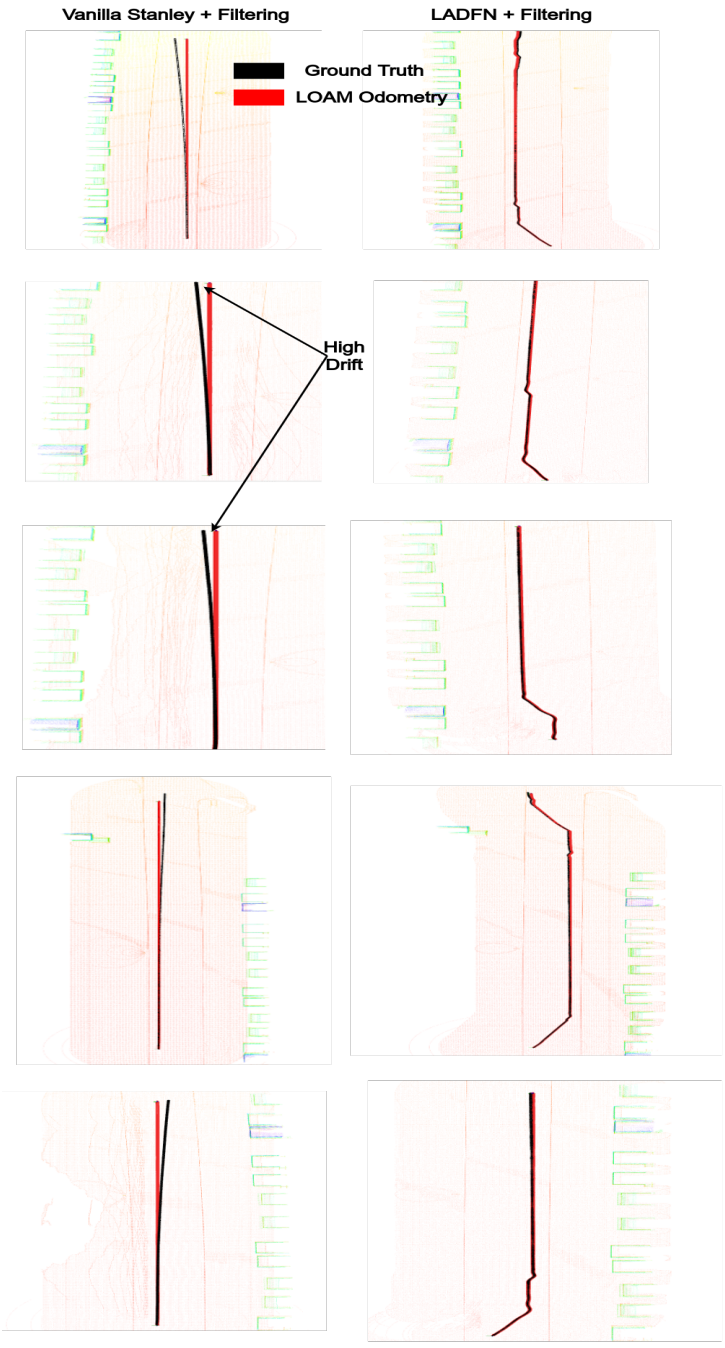

Figure 3.5 LADFN significantly reduces drift with a minor change in the path taken to reach the goal. The LOAM-generated trajectories are shown in red, the ground-truth trajectories are shown in black, and the green structures correspond to feature-rich regions. The left column shows *Vanilla Stanley + F* while the right column shows *LADFN + F*. Each row corresponds to a different scene. These scenes were run with a LIDAR range of 45 meters and dynamic obstacle filtering enabled.

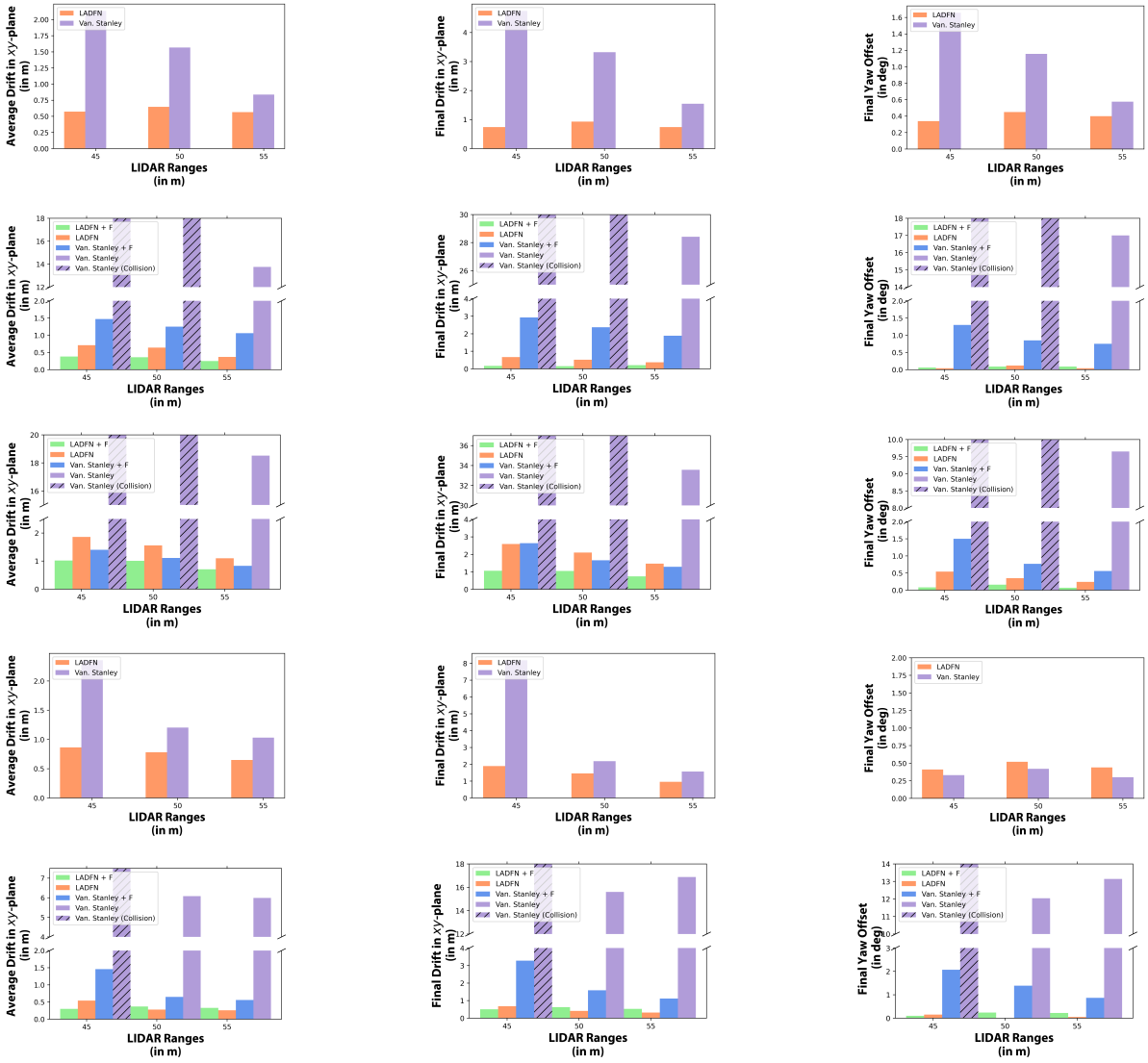

Figure 3.6 LADFN significantly reduces drift than competing benchmarks on a diversity of CARLA scenes. Firstly, we want to show that LADFN leads to lower average drift than the benchmark. This is clearly shown in the **first column** where LADFN reduces Stanley's drift by at least $1.48\times$. Secondly, we provide a framework for analyzing the performance of goal-reaching tasks, seen in the final translation offset plots in the **second column**, where once again, LADFN consistently outperforms the Vanilla Stanley controller by at least $1.63\times$. Thirdly, we show that rotational errors are significantly reduced by LADFN in the **third column**. In dynamic scenes (2,3,5), dynamic obstacle filtering was enabled. In **all** columns, *F* in the plot legends denotes that filtering of dynamic obstacles was carried out. Diagonal hatches '/' inside bars represent collision occurrence of the ego with its surroundings in the experiment.

3. *Rotational Offset:* Yaw offset with respect to the ground-truth yaw at the goal location.

Metric 1 gives a general idea of the quality of localization achieved for the entire trajectory. *Metric 2* gives an idea of how the methods will perform on goal-reaching tasks. *Metric 3* serves two purposes: like *Metric 2*, this is also a representation for goal-reaching tasks, and further, it gives an idea of the quality of the future trajectory, since rotational errors in odometry have a compounding effect on the future trajectory.

3.4.4 Qualitative Results

Fig. 3.5 shows a qualitative view of the trajectory and drift in all the five scenes. In the right column, as the estimated LOAM trajectory in red overlaps the ground-truth trajectory in black, unlike the baseline shown in the left column where the red deviates from the black, it can be clearly seen that with our approach, the drift is substantially reduced in every case. We can see that the ego tries to maintain proximity with the feature-rich regions shown in green. Rows 2, 3, and 5 correspond to dynamic scenes with filtering enabled and so traffic vehicles are not visible in the LIDAR scans shown in the Fig. 3.5. We observe that even in the presence of filtering, our approach outperforms the baseline, highlighting the need for active navigation in drift minimization.

We achieve this due to the complex behaviours learned by our model, the examples of which can be seen in the following link: `https://mohdomama.github.io/LADFN/`.

3.4.5 Quantitative Results

We depict the quantitative comparisons of our approach with the baselines in the five rows of Fig. 3.6 corresponding to the five scenes. Rows 1 and 4 do not have dynamic actors, and the comparison is between *LADFN* and *Vanilla Stanley*, shown with orange and violet bars. Rows 2, 3 and 5 correspond to the scenes populated with dynamic actors. Here, we include the filtering module in both the Stanley Controller and LADFN, denoted as *LADFN + F* and *Vanilla Stanley + F*, and shown with green and blue bars. In some scenes, the LOAM estimates have significantly deviated resulting in collision for the baseline *Vanilla Stanley*, these are shown with hatched violet bars.

Our first step is to show that LADFN leads to a lower average drift than benchmark competitors. This is clearly shown in column 1. In static scenes, we get up to $3.7\times$ improvement in average drift using LADFN. In dynamic scenes, with filtering enabled, we get up to $4.8\times$ improvement in average drift using LADFN, while with filtering disabled, the drift in *Vanilla Stanley* is often so high that it leads to collision. This is not the case with LADFN which remains relatively unaffected, even with filtering disabled. It should also be noted that the impact of dynamic vehicles and feature-proximity on benchmark competitors gets pronounced as the LIDAR range decreases, while LADFN is resilient to these changes in LIDAR range.

The second column and the third column show the final drift and the final yaw offset at the end of the run respectively. They give an idea of the goal-reaching capabilities of all the approaches along with

an intuition of how bad the future trajectory would be affected if the run was continued. We see that LADFN outperforms the benchmarks here with similar patterns as in column 1.

3.5 Epilogue

In this article, we presented a novel active navigation framework for autonomous vehicles that minimizes drift while avoiding dynamic obstacles. We have shown that our framework can learn to trade-off control cost and drift minimization cost using reinforcement learning and geometric features. We have also demonstrated the superiority of our approach over methods that do not actively navigate to minimize state estimation errors.

However, this work also has some limitations and directions for future research. First, the geometric measure for feature importance is not enough to capture the complexity and diversity of real-world environments. We need a more involved approach that can account for semantic information and contextual cues. Second, our reinforcement learning paradigm may not scale well to new scenes or scenarios that are very different from the ones used for training. Moreover, it is not explainable or transparent, which may pose security and ethical challenges. We need to develop methods that can generalize better, adapt faster, and provide justification for their actions.

Chapter 4

Active Navigation with Learned Features and Batch CEM-MPC Planner

4.1 Prologue

Our first article LADFN [25], used an uninterpretable reinforcement learning (RL) which takes many iterations to train and generalizes poorly to new environments. In the second research article, we adopted a more systematic approach to significantly improve active navigation. A key insight was that a simple geometric measure is insufficient to robustly identify drift minimizing LiDAR points. Hence, a weekly supervised approach was used to learn drift minimized features using gradient based neural activation techniques, like GradCAM[32]. GradCAM [32] is an explainable AI tool to determine which input features are most influential for predicting a differentiable output. We also develop a modification of the well-known triplet loss [7], which we call the *Directional Triplet Ranking Loss*, for training our neural network. These improvements to the perception module in our pipeline result in successful identification of features useful for the downstream planning module to minimize drift.

Our motion planning module relies on solving tens of hundreds of optimization problems in parallel, each initialized from trajectory samples drawn from a Gaussian distribution. This counters the inherent non-convexity of the problem, escapes poor local minima, and drives the vehicle closer to the features provided by the perception module (see Fig.4.5). We combine our batch trajectory optimizer with a Cross-Entropy Method (CEM) style adaptation of the initialization distribution. We run the combined approach in a receding-horizon fashion, which we henceforth call CEM-MPC.

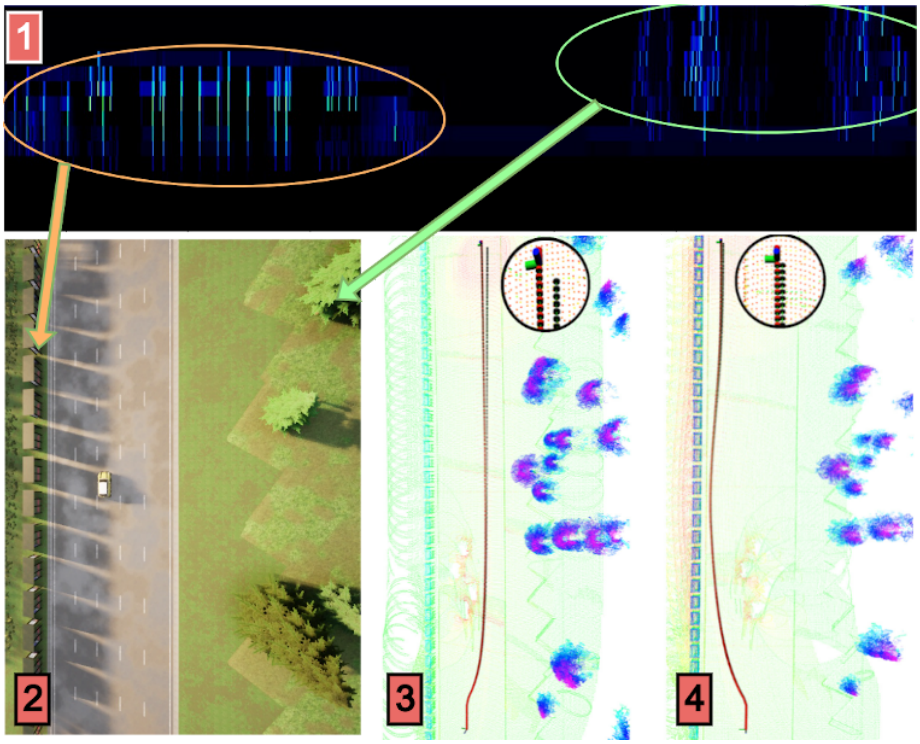

Figure 4.1 Actively moving towards salient semantic features reduces drift. An ego vehicle in the CARLA [11] scene (in **2**) observes trees on the right and better-defined poles and platforms on the left. **1** shows the corresponding range image (Section 4.2.1) of this scene. **3** and **4** show the trajectories executed by a prior approach LADFN [25] and our method respectively. By moving towards semantically-better features on the left in **4**, our approach outperforms LADFN by $\mathbf{54.3\%}$, increasing the control cost by only $\mathbf{0.4\%}$. Therefore, it is imperative that we identify semantically-better features to minimize drift in active navigation.

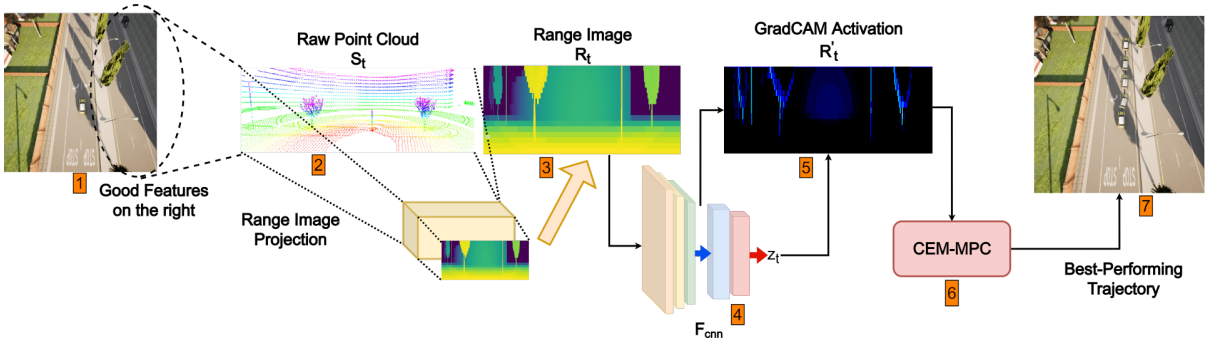

Figure 4.2 Pipeline of our system. **1** shows the CARLA [11] scene with trees and electric poles on the right side of the ego vehicle C. Then, we project a 3D point cloud $\mathbf{S}_t \in \mathbb{R}^{3\times N_{\text{pc}}}$ (**2**) to a 2D range image $\mathbf{R}_t \in \mathbb{R}^{w\times h}$ in **3**. Then, our neural network perception module F_{cnn} (**4**) learns to rank these range images based on the drift d_t they accumulate. **5** shows the GradCAM [32] activation $\mathbf{R}_t^{'} \in \mathbb{R}^{w\times h}$ of the range image shown in **3**, which highlights the features (seen in dark blue) that are conducive to minimizing drift d_t. These features are used by the CEM-MPC Controller **6** which generates and selects a best-performing trajectory for the ego vehicle C to execute (**7**).

4.2 Feature Selection Module

The principal goal of our feature selection/perception module is to observe 3D LIDAR point clouds and identify drift-minimizing feature locations. However, the key challenge is that there is no oracle data set or mathematical model that provides ground-truth labels asserting which features are the best at reducing drift. Indeed, this often depends on the complex semantics of a scene and complex nonlinear operations in the LIDAR Odometry and Mapping (LOAM) [38] SLAM framework.

Instead, however, we can obtain a diverse training data set where we start the robot at a certain pose and observe the relative drift at a new pose after it executes a trajectory. Then, our key insight is that we can learn an *embedding* of visual scenes that helps rank whether the drift will increase or decrease if the robot moves to a target pose. Then, we can find which pixels in a visual scene are most responsible for this ranking and move the robot towards these features. We now describe how we (a) learn a representation to rank visual scenes based on drift (4.2.2) and (b) extract the most informative features for this ranking using GradCAM (4.2.4).

In our system, a VLP-16 LIDAR is attached to an ego vehicle C. This collects N_{pc} 3D point clouds of its surroundings at each discrete time step t, namely $\mathbf{S}_t \in \mathbb{R}^{3\times N_{\text{pc}}}$. The ego vehicle C incurs a drift d_t at time t. We project the 3D point cloud $\mathbf{S}_t$ onto a 2D image plane to obtain range image $\mathbf{R}_t \in \mathbb{R}^{w\times h}$, where the width and height of the range image $(w, h) = (1800, 16)$. From the N raw point clouds $\mathbf{S}_t$, we project N such range images $\mathbf{R}_t$ to build our input data set $\mathcal{D}$. This data set $\mathcal{D}$ is used to train a feature extractor to output a rank $z_t = F_{\text{cnn}}(R_t; \theta_{\text{cnn}})$, with the learned parameters θ_{cnn}. F_{cnn} is a Convolutional Neural Network that takes as input the range image $\mathbf{R}_t \in \mathbb{R}^{w\times h}$ and outputs a scalar embedding $z_t \in \mathbb{R}$. The scalar rank z_t represents an ordering of the input range image $\mathbf{R}_t$; higher rank z_t means lower drift d_t, and vice versa. As previously motivated, identifying drift-minimizing features is an indirect task

that we have no supervision over. This requires the need for a ranking-based approach. To obtain this rank z_t, we train our network F_{cnn} using a novel Directional Triplet Ranking Loss, inspired by [7], and detailed in 4.2.3.

Using the input range image $\mathbf{R}_t$ and the embedding z_t, we query our network F_{cnn} for obtaining the activation maps $\mathbf{R}_t^{'} \in \mathbb{R}^{w \times h}$ using GradCAM [32]. This step is one of the crucial parts of our pipeline as we exploit GradCAM to identify drift-minimizing features percolated through the network. Henceforth, we refer to the output of the Feature Selection module, namely the drift-minimizing features, as simply *features*, as they represent the "goodness" of the ego vehicle's perception, and play a key role in the control objective. We now detail the feature selection module.

4.2.1 Range Image Projection

Range images are a popular proxy representation for LIDAR data. We describe briefly the projection of a LIDAR point cloud input $\mathbf{S}_t$ into a range image $\mathbf{R}_t$. Given a point cloud $\mathbf{S}_t$ at time t containing the tuple $\langle p_x, p_y, p_z \rangle$ representing each point in Cartesian coordinates, we extract the range (depth) r, azimuth angle ϕ, and elevation angle θ as:

$$r = \sqrt{p_x^2 + p_y^2 + p_z^2}, \phi = \arctan(p_x/p_y), \theta = \arcsin(p_z/r) \tag{4.1}$$

We project this onto an image of width w and height h. Using VLP-16's number of channels (16), we set our range image height $h = 16$. Using VLP-16's angular resolution (0.2°), we set our range image width $w = 1800$ (i.e. $360^\circ / 0.2^\circ$). We also require beforehand the vertical field of view of our LIDAR scanner $\text{fov}_{\text{down}} < \text{fov} < \text{fov}_{\text{up}}$. Here, fov_{down} and fov_{up} are the boundaries of the vertical field of view; for VLP-16, these values are -15° and 15° respectively. The projected image coordinates are:

$$x_{\text{proj}} = \frac{1}{2} \times (\phi/\pi + 1), y_{\text{proj}} = 1 - \left(\frac{\theta + |\text{fov}_{\text{down}}|}{\text{fov}_{\text{total}}} \right), \tag{4.2}$$

where $\text{fov}_{\text{total}} = |\text{fov}_{\text{down}}| + |\text{fov}_{\text{up}}|$. We clamp x_{proj} and y_{proj} to the ranges $(0, w-1)$ and $(0, h-1)$ for array indexing.

4.2.2 Directional Triplet Ranking Loss

In order to train F_{cnn}, we design a new loss called the directional triplet loss that takes inspiration from the standard triplet loss [7]. Here we describe the motivation of designing a new loss function. Contrastive learning facilitates deep neural networks to differentiate between positive and negative samples. With careful selection of samples belonging to positive, negative, and anchor classes $\langle p, n, a \rangle$, we can learn to distinguish between visually similar input images. As previously stated, it is imperative that robots are able to navigate with drift minimization as a primary objective. This criterion informs us that our ego vehicle must change its lateral position intelligently in order to maintain proximity to good features that are conducive to minimizing drift, as established by prior work LADFN [25]. Here, lateral refers to the side perpendicular to the road length.

Following this motivation, in practice, we select range images from our data set $\mathcal{D}$ that belong to three different lateral positions on the same road to represent the positive p, negative n, and anchor a classes of our loss function. We note that these three range images are visually similar and therefore contribute to our motivation for using this loss function. Let us denote the drift incurred by the lateral positions corresponding to the positive, negative, and anchor samples by d_p, d_n, and d_a, respectively. These drifts satisfy the condition $d_p < d_a < d_n$, i.e., the drift incurred by the positive sample is less than that of the anchor sample, which is less than that of the negative sample. During training, we want our network F_{cnn} to be able to differentiate between the range images $\mathbf{R}_p$, $\mathbf{R}_n$, and $\mathbf{R}_a$ corresponding to the classes in our loss function $\langle p, n, a \rangle$. Hence, we train our network to predict a rank z_t, which is a scalar in latent space. The ranks corresponding to the positive, negative, and anchor classes are given by z_p, z_n and z_a respectively. We want these ranks to satisfy the condition $z_p > z_a > z_n$, but we also want these values to be close to each other in the latent space since the range images of these three samples are visually similar.

To ensure that the positive sample has a higher value than the negative sample, we introduce the term $(z_n - z_p)$. To ensure the ranks corresponding to positive and negative samples remain close to the anchor, we use a regularization term $((z_a - z_n)^2 + (z_a - z_p)^2)$. Finally, we maintain the margin β, present in the standard triplet loss [7], to preclude the rank degeneracy condition $z_p = z_a = z_n$. Therefore, our Directional Triplet Ranking Loss function becomes:

$$\mathcal{L} = max(\beta + z_n - z_p + (z_a - z_n)^2 + (z_a - z_p)^2, 0) \tag{4.3}$$

Significance: We now describe the benefits of our triplet loss compared to an alternative of predicting absolute drift directly from the ego vehicle's observations. During data collection, it was evident that these absolute drift values had high statistical variation and an extensive range dispersed irregularly throughout our samples. This made it difficult to collect an unbiased data set, skewing the network's results. Further, on any given road, the drift values are similar for different lateral positions, making it difficult to predict the precise value of drift at any road location.

Our new loss naturally handles these issues. Even with high statistical variance, it is easy to select anchor, positive and negative samples. By choosing these samples from the same road, we can enforce the network to learn the minute differences between range images that are instrumental in improving drift. Another advantage of our loss is its symphony with GradCAM. As detailed in Sec. 4.2.4, GradCAM can be used to query pixels that maximally activate a differentiable entity, such as the rank z_t. When we query GradCAM on our network F_{cnn} trained using our directional triplet ranking loss, we obtain the *features* that have the most significant impact in increasing the rank and reducing drift.

4.2.3 Training Procedure

We collect data set $\mathcal{D}$ using the popular CARLA Simulator [11]. An ego vehicle is spawned in various scenes containing features of various semantic classes such as buildings, trees, electric poles, etc. For time interval $t = [0, m]$, where m represents the overall run-time of inference in a single

scene $\mathcal{S}$, we run LOAM [38] to perform SLAM and obtain the car's odometry. During this simulation, we record the drift $d_{0:m}$ in the xy-plane using the ground truth odometry obtained from CARLA [11], and we also record the point cloud obtained from LOAM, $\mathbf{S}_{0:m}$, to use as the perception input in our pipeline. This is done by spawning the car C initially at time $t = 0$, then programatically querying the drifts $d_1, d_2, ..., d_m$ along one lateral position. We repeat this for many lateral positions on the same road to complete the data collection for scene $\mathcal{S}$. This allows us to distinguish the datapoints in one of the sample classes in the triplet loss $\langle p, n, a \rangle$. We collected around 6K total samples, split into 80% and 20% for training and validation respectively.

Our network architecture complements our contrastive loss function. The network is a simple CNN containing 5 blocks; each block contains a convolutional layer, followed by batch normalization and leaky ReLU activation. We perform circular padding on the input tensor's width dimension and zero padding on the height dimension, as this helps range image inputs propagate better through the network. After the 5 blocks, we perform max pooling, flatten the input, and send it through multiple fully connected layers to obtain the rank z_t. The CNN hyperparameters are as follows: a learning rate of 0.001, a batch size of 32, a triplet loss margin β of 1, and a circular pad length of 1 (in the convolutional layers).

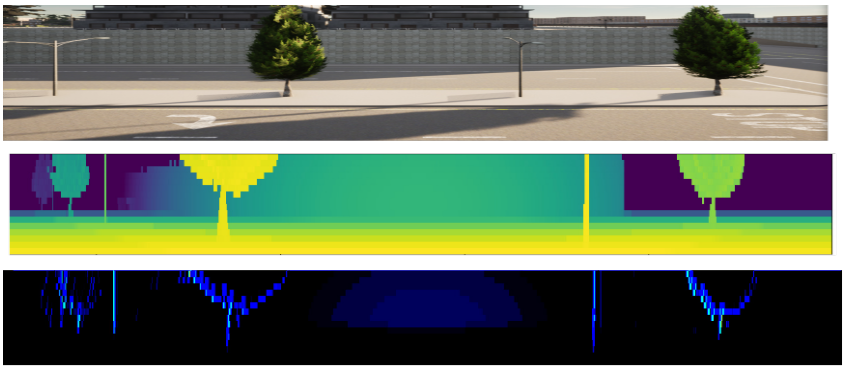

Figure 4.3 Drift minimizing feature detection using GradCAM. We show a CARLA [11] scene in row 1. Its corresponding range image $\mathbf{R}_t$ is in row 2 and its feature activation map $\mathbf{R}_t^{'}$ queried using GradCAM [32] in row 3. Pixels in bright blue highlight the drift-minimizing features. $\mathbf{R}_t^{'}$ is activated more for salient features like poles and tree trunks than planar features like walls.

4.2.4 Inferring feature locations using GradCAM

Our ultimate goal is not just to compute a rank, but also the *location* of good features, for which we utilize GradCAM [32]. While GradCAM's primary purpose is to visualize the hidden layers of large CNNs, we exploit it to pinpoint the location of the most influential features on the range image that lead to a high ranking by our network F_{cnn}. Specifically, we scope into one of the deeper layers of F_{cnn} using GradCAM to visualize the activations of our input range image $\mathbf{R}_t$, $\mathbf{R}_t^{'}$. The activation $\mathbf{R}_t^{'}$ contains the necessary locations y_{feat} (re-projected to 3D from $\mathbf{R}_t^{'}$) needed for our MPC controller to perform

trajectory generation and planning. Fig. 4.3 shows the input range image $\mathbf{R}_t$ and its corresponding activation $\mathbf{R}_t^{'}$ after it has percolated through F_{cnn}.

4.3 Motion Planning Module

Our planning module is responsible for computing safe, collision-free trajectories that drive the ego-vehicle safely towards the features generated by the perception pipeline described in the previous sections. We follow a trajectory optimization approach under the following assumptions.

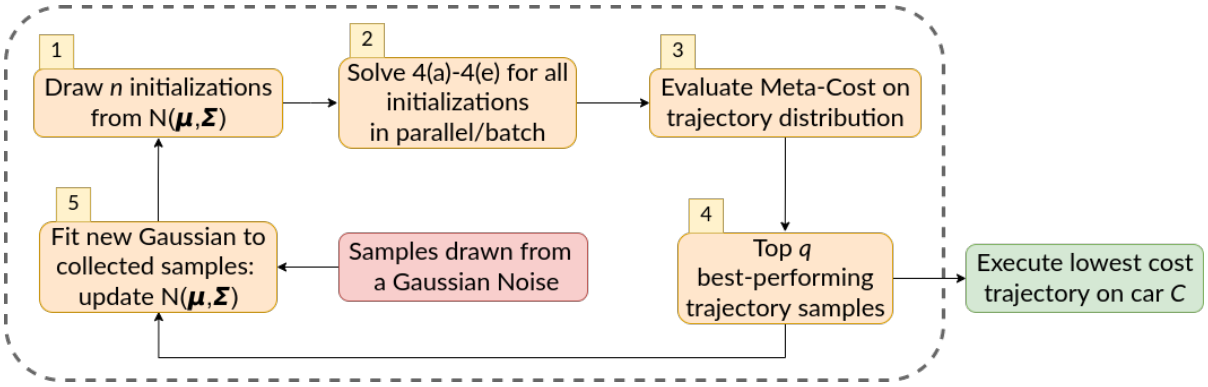

Figure 4.4 Our CEM-MPC Pipeline combines the exploration of CEM with a custom batch non-convex trajectory optimizer.

- The optimizer operates in the reference-frame attached to the center-line of the road, called the Frenet-Frame [37], which allows us to always plan assuming a straight road. We can map the computed optimal trajectory back to the global frame to align with the road's curvature.

- We assume a typical urban driving scenario where the heading of the ego-vehicle with respect to the center-line is small ($\approx \pm 13^{\circ}$) which allows us to bound the shape of the ego-vehicle and obstacles through axis-aligned ellipses without being overly conservative.

The mathematical formulation of our trajectory optimizer can be described in the following manner. Here, $(x_{e,t}, y_{e,t})$ is the ego-vehicle's position at time t. $v_{\max}$ and $a_{\max}$ are the velocity and acceleration bounds, and y_{feat} are the feature locations supplied by our perception module. $\mathbf{b}_0$ and $\mathbf{b}_f$ are initial and final boundary conditions. a and b are the major and minor axes of the elliptical approximation of

obstacles. f_v and f_a are the speed and acceleration magnitudes.

$$\min \sum_t c(y_{e,t}, \dot{x}_{e,t}, \ddot{x}_{e,t}, \dot{y}_{e,t}, \ddot{y}_{e,t}, y_{\text{feat}}) \tag{4.4a}$$

$$(x_{e,t_0}, y_{e,t_0}, \dot{x}_{e,t_0}, \dot{y}_{e,t_0}, \ddot{x}_{e,t_0}, \ddot{y}_{e,t_0}) = \mathbf{b}_0. \tag{4.4b}$$

$$(\dot{x}_{e,t_f}, \dot{y}_{e,t_f}, \ddot{x}_{e,t_f}, \ddot{y}_{e,t_f}) = \mathbf{b}_f. \tag{4.4c}$$

$$f_v(\dot{x}_{e,t}, \dot{y}_{e,t}) \leq v_{max}, f_a(\ddot{x}_{e,t}, \ddot{y}_{e,t}) \leq a_{max} \tag{4.4d}$$

$$-\frac{(x_{e,t} - x^j_{o,t})^2}{a^2} - \frac{(y_{e,t} - y^j_{o,t})^2}{b^2} + 1 \leq 0, \tag{4.4e}$$

$$\text{c}(\text{y}_{e,t}, \dot{x}_{e,t}, \ddot{x}_{e,t}, \dot{y}_{e,t}, \ddot{y}_{e,t}, y_{\text{feat}}) = (\ddot{x}^2_{e,t} + \ddot{y}^2_{e,t}) + (y_{e,t} - y_{\text{feat}})^2 + (\sqrt{\dot{x}^2_{e,t} + \dot{y}^2_{e,t}} - v_{\text{des}})^2$$

$$f_v = \sqrt{\dot{x}^2_{e,t} + \dot{y}^2_{e,t}}, f_a = \sqrt{\ddot{x}^2_{e,t} + \ddot{y}^2_{e,t}} \tag{4.6}$$

The cost function (4.4a) minimizes (i) the magnitude of accelerations at each time instant, (ii) the lateral separation between the ego-vehicle and the features, and (iii) the ego-vehicles departure from the desired velocity (v_{des}) profile. The equality constraints (4.4b)-(4.4c) impose the initial and final boundary conditions. Inequality constraints (4.4d) enforce bounds on the norm of velocity and accelerations while (4.4e) ensures collision avoidance with the j^{th} obstacle with position $(x^j_{o,t}, y^j_{o,t})$ at time t.

The main complexity of optimization (4.4a)-(4.4e) stems from the non-convex collision avoidance constraints. In the next sub-section, we present our main algorithmic result on the planning side, wherein we combine notions from gradient-free CEM with a batch optimizer that can solve several perturbations of (4.4a)-(4.4e) in parallel.

4.3.1 Combining CEM Exploration With Batch Optimization

Solving non-convex optimizations like (4.4a)-(4.4e) require an initial guess of the optimal solution. The obtained solution depends on how close this guess is to the actual optimal solution. Our approach leverages this dependency on the initial guess to its advantage. We solve (4.4a)-(4.4e) from different random initializations drawn from a Gaussian distribution and then subsequently adapt the parameters of the distribution based on the resulting optimal trajectory samples.

Fig. 4.4 shows the pipeline of our approach which we refer to as CEM-MPC. It combines the adaptive exploratory properties of CEM with an analytical trajectory optimization to develop a computationally efficient and high performance MPC algorithm. Our CEM-MPC begins (step 1) by initializing a trajectory distribution characterized by a mean $\boldsymbol{\mu}$ and a co-variance $\boldsymbol{\Sigma}$. We draw $n(\approx 1000)$ trajectories from this distribution. We then solve n variants of optimization (4.4a)-(4.4e) (step 2) each initialized from a specific trajectory sample drawn in step 1. Since each variant is decoupled from each other, we can solve them in parallel (batch) fashion. The solution process at step 2 results in a distribution of optimal trajectories and we evaluate user-defined meta-costs (to be defined later) on them in step 3. We then rank (step 4) the optimal trajectories based on their associated meta-cost value and choose the top q best

performing samples. We then fit a Gaussian distribution to the best performing samples from step 4 to update the $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ for the next MPC iteration. A quintessential problem in CEM is that the variance of the best performing samples could shrink over MPC iterations and this in turn will reduce the diversity in the trajectory initializations. In other words, the exploration ability of the CEM-MPC could deteriorate. We counter this by drawing some trajectory samples from a Gaussian noise and appending them to the best performing samples before executing the mean and covariance update at step 5. The Gaussian noise uses the special co-variance matrix [21] which ensures smoothness in the sampled trajectories. The same noise is also used to initialize the MPC algorithm at the first iteration.

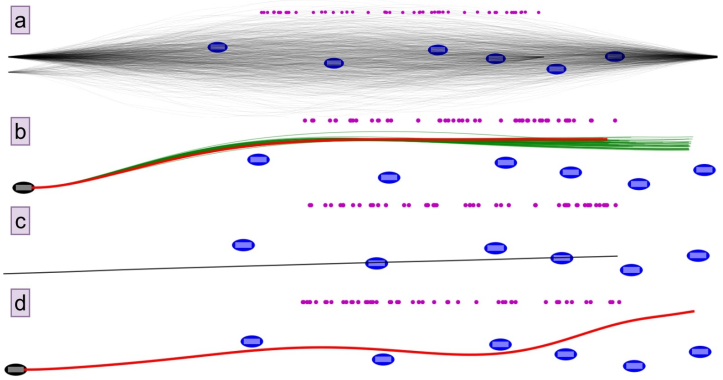

Figure 4.5 Benefit of our CEM-MPC that combines batch non-convex trajectory optimization and adaptive sampling of CEM a: CEM-MPC trajectory initialization. **b**: distribution of best performing samples after 10 MPC iterations. **c**: straight line initialization for a typical trajectory optimizer. **d**: corresponding optimal trajectory after 10 MPC iterations. Our CEM-MPC is able to converge to a homotopy, bringing the vehicle closer to features (pink points) while avoiding obstacles (blue). In stark contrast, the standard optimizer gets stuck in local minima far from the features.

Meta-Cost: Let $(\overline{x}^l_{e,t}, \overline{y}^l_{e,t}), \forall t$ be the l^{th} optimal trajectory obtained from the batch optimization at step 2 of Fig. 4.4. The meta-cost evaluates the utility of these trajectories:

$$c + \max(0, \|f_{\text{curv}}\| - \kappa_{\max}) + \max(0, \|\overline{y}^l_{e,t}(t)\| - y_d), \tag{4.7}$$

where f_{curv} represents the curvature function computed based on first and second derivatives of the position trajectories. The term $\kappa_{\max}$ is the maximum permissible curvature and y_d represents the road-width. Thus, our meta-cost (4.7) consists of the primary cost term from the trajectory optimizer (4.4a), and additional penalties that measure the violation on curvature and road boundary constraints.

Efficient Batch Optimization: The computational efficiency of our CEM-MPC pipeline rests on how fast we can solve hundreds of non-convex optimizations in parallel (step 2 in Fig. 4.4). To achieve real-time performance, we built a GPU-accelerated variant of our recent batch non-convex optimizer [1] and expanded the mathematical derivation to include the cost functions and constraints specific to the current work.

An Illustrative Example: Fig.4.5 shows a typical result from our CEM-MPC pipeline. **a** shows the trajectory initialization of our CEM-MPC. **b** shows the distribution of the top-performing trajectory samples in terms of meta-cost. The best cost trajectory, shown in red, converges to a homotopy that

moves the ego-vehicle closer to the pink features. The performance of a standard MPC initialized from a single straight-line trajectory is shown in **c**. Even though the resulting optimal trajectory is collision-free, it gets stuck in a poor local minimum with a large lateral distance from features.

4.4 Experimental Results

4.4.1 Experimental Setup, Metrics and Benchmarks

Throughout our experimentation, we use the CARLA Simulator [11] because its precise physics models of the world and the sensors are ideal for self-driving research. We implemented our CEM-MPC in Python using GPU accelerated linear algebra library JAX [6]. The typical run-time was around 0.04s on a i7-8500H laptop with 32GB RAM and RTX2080 Nvidia Graphics card.

We test our results against two benchmarks. They are the closest work to ours of LADFN (Learning Actions for Drift-Free Navigation) [25], and a single-objective Vanilla (Standard) Stanley Controller (VC). A striking contrast between our approach and LADFN is that the latter uses edge features, calculated geometrically based on a smoothness measure, to identify good features while we have a learned approach for it. We test the three approaches on various scenarios with different run lengths using the following metrics:

1. **Absolute Positional Error (APE):** It's a measure of how far off the LOAM pose prediction is with respect to the ground truth.

2. **Distance for Run Length (DRL):** For a given run-length along the road, the car can take trajectories of various curvatures, thereby making the distance travelled more than the run length. We call this distance DRL, and it's a measure of the control cost for the three approaches.

We show that we reduce the drift or APE with a slight trade-off in the control cost DRL via drift-aware navigation. Further our system can outperform LADFN in certain scenes where the semantics of features play an essential role.

We perform these tests on 10 scenes that we have created with varying distributions of semantically different features. Five of these scenes don't contain any traffic vehicles and the remaining five contain different configurations of traffic.

4.4.2 Qualitative Results

Here, we discuss three cases of varying degrees of complexity through which we qualitatively evaluate the effectiveness of our perception system compared to LADFN [25] and VC, as shown in Fig. 4.6. Our system performs at par with the previous state-of-the-art (LADFN [25]) in simple cases (rows 1 & 2). In row 3, which is a complex scenario, we outperform both LADFN [25] and VC. This is because LADFN [25] faces difficulty in assessing the quality of features using its smoothness-based geometric measure, as stated previously.

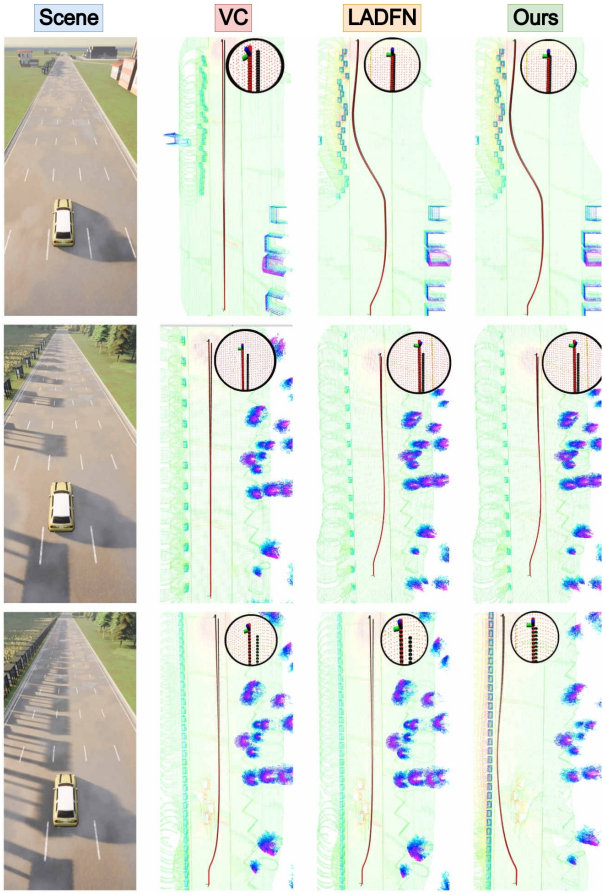

Figure 4.6 Our deep-learned model differentiates between drift-minimizing features of several semantic classes. Each row shows a different CARLA [11] scene. Columns 2,3, and 4 show the trajectories executed by VC, LADFN [25], and our approach respectively. Black and red lines show the ground truth trajectories and LOAM's [38] odometry respectively. The zoomed-in circles on top highlight drift. In scenes 1 and 2, we match LADFN [25], both giving an improvement of about $\mathbf{51\%}$ over VC in scene 1 and $\mathbf{2.9\%}$ in scene 2. In scene 3, our approach outperforms LADFN by $\mathbf{54.2\%}$ and VC by $\mathbf{42.6\%}$ respectively, because we identify semantically-better features like poles, platforms as opposed to LADFN's [25] focus on feature geometry.

4.4.2.1 Case 1 - High density of features with no semantic differences

The scene shown in the first row of Fig. 4.6 contains high density of features on one side of the road at a time, and no features on the other side of the road. It is evident that the Vanilla Controller (VC) executes a trajectory with least control cost, whereas LADFN [25] and our approach actively move towards the side which contains the higher density of features to minimize drift.

4.4.2.2 Case 2 - Low density of features with semantic differences

Fig. 4.6 contains the following: left side contains semantically better features like poles and platforms, but placed far apart, and the right side contains trees that are stacked close together providing LADFN [25] with more edge features. In this case, we observe the low density of features on the left

side are unable to activate the neurons in our network F_{cnn}, while the features on the right side do a better job of neuron activation (Fig. 4.7). Therefore, LADFN and our method move towards the right side. This is further confirmed by our ablation study in 4.4.4.

4.4.2.3 Case 3 - High density of features with semantic differences

This scene shown in the third row of Fig. 4.6 is similar to *Case 2*, however the poles and the platforms on the left side are stacked close together and the ego encounters traffic later in the scene. In this case, we observe that the features from the left side maximally activate the neurons of our network F_{cnn} (Fig. 4.7). As shown in Fig. 4.6, our method moves towards these semantically better features on the left, whereas LADFN still moves the ego car towards the right side that contains more edge features. This results in our approach outperforming both LADFN and VC by a substantial margin of $\mathbf{54.2\%}$ and $\mathbf{42.6\%}$ respectively.

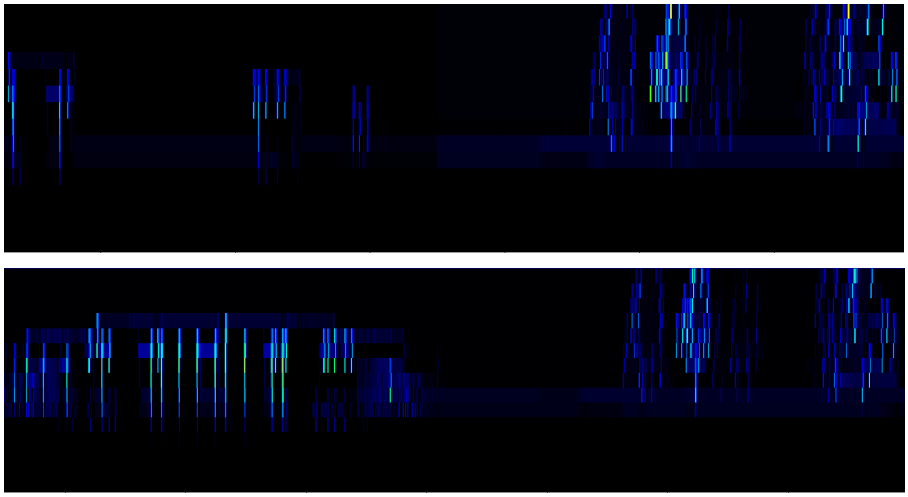

Figure 4.7 GradCAM ($\mathbf{R}_t^{'}$) Activation Comparison. From Cases 2 & 3 in 4.4.2, $\mathbf{R}_t^{'}$ is activated more based on which side of the road is populated with semantically-better features like poles, platforms.

Fig. 4.8 shows intelligent behaviours manifested as a result of batch optimization and evaluation of thousands of trajectories in real-time. As shown in green in the left column, the ego vehicle understands to move towards the left, but it also understands that it must first overtake traffic vehicles in front of it; this is reflected aptly in the speed profile on the right (in green). Moreover, the blue trajectory shown on the left assumes that the traffic vehicles are absent. So, the ego vehicle moves towards the feature-rich left side but doesn't expend control cost to perform any overtaking, thereby maintaining a stable speed profile (right, in blue). We note that [25] demonstrated similar drift-aware maneuvers using Reinforcement Learning. However, we argue that similar behaviours can be replicated in a more explainable and safer way using analytical generation and evaluation of a large variety of trajectories through a robust MPC method using batch optimization on a GPU.

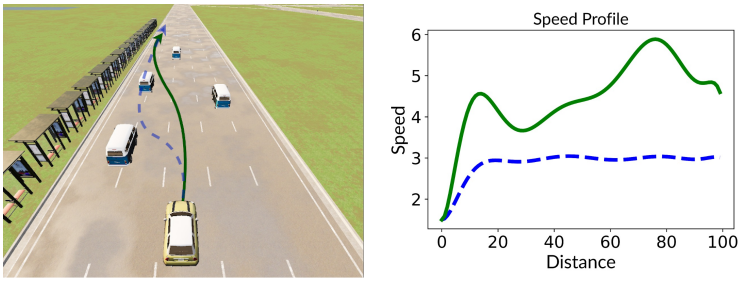

Figure 4.8 Intelligent drift-aware behaviours (left) and their corresponding velocity profiles (right). In green, we show that a yellow ego vehicle will speed up to overtake moving traffic while maintaining proximity to drift-minimizing features on the left. In blue, we show that *without* moving traffic, the ego will smoothly move to the left to achieve a desired velocity of $3ms^{-1}$.

4.4.3 Quantitative Results

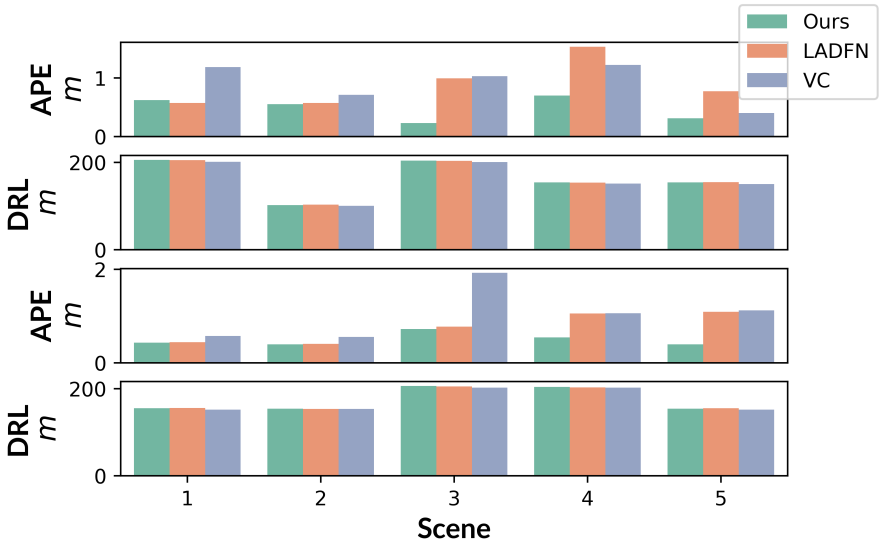

Figure 4.9 In static scenes (rows 1 & 2), we reduce drift by $\mathbf{76.76\%}$ and $\mathbf{47.54\%}$ compared to LADFN and VC respectively. Yet, we only increase control cost by $\mathbf{0.4\%}$ and $\mathbf{2.23\%}$ respectively. **In dynamic scenes** rows (3 & 4), we reduce drift by $\mathbf{64.22\%}$ and $\mathbf{65.17\%}$ compared to the same respective controllers. We *decrease* control cost by $\mathbf{0.6\%}$ and increase by $\mathbf{1.98\%}$ respectively.

Fig. 4.9 shows the trade-off between drift reduction (APE) and control cost (DRL) that our method provides vis-a-vis other benchmarks on static scenes. These experiments are conducted in five diverse scenes with no obstacles, in different towns of the CARLA [11] Simulator. In scenes 1 & 2, the trajectories executed by our method and LADFN [25] are similar, and hence, the drift improvement that both methods offer over VC are also similar. In scenes 3 to 5, our approach results in entirely different, and improved, trajectories compared to LADFN [25]. We observe that the features' semantics play a crucial role in drift reduction of up to $\mathbf{76.76\%}$ in scenes 3 to 5, and our method is able to exploit these semantic differences using GradCAM [32]. As previously explained in our qualitative results section

4.4.2, LADFN [25] is unable to make use of semantic differences in features, and makes the ego simply move towards regions having higher edge feature density.

Fig. 4.9 shows a similar trade-off between drift reduction (APE) and control cost (DRL) in dynamic scenes. Here, the experiments were conducted in scenes containing various configurations of traffic obstacles. In scenes 1 to 3 both LADFN [25] and our approach provide similar drift improvement. Scene 3 contains a low density of features, hence, we note drastic improvement in both LADFN [25] and our approach over VC. Once again, when we introduce semantically different features in scenes 4 and 5, we observe that our approach outperforms both LADFN [25] and VC by up to $\mathbf{65.17\%}$ for the aforementioned reasons.

Table 4.1 Ablation Experiment on Feature Density vs. Drift

Feature Density	VC (m)	LADFN (m)	Ours (m)
0.25	0.818	1.12	**0.66**
0.16	1.05	1.27	**0.84**
0.125	1.45	1.50	**1.05**
0.1	1.14	1.23	**1.03**
0.083	1.344	**1.309**	**1.309**

4.4.4 Ablation Study on GradCAM Inferences

We conduct an ablation study on feature semantics and their density, shown empirically in Cases 2 & 3 of 4.4.2. We created a scene containing many edge features on one side, and semantically better features on the other. We have a measure for the density of semantically better features which we vary from 0.25 (more dense) to 0.083 (less dense). For density values 0.25 to 0.125, we observe that our approach moves the ego vehicle closer to semantically-better features (poles), whereas LADFN [25] moves the ego vehicle closer to more edge features (trees). This results in better drift minimization in our approach. As we decrease the density of semantic features, the improvement of our approach over LADFN decreases. At a density value of .083 (last row in Table 4.1), the trajectories of both LADFN and our approach become similar, i.e., both methods moves the ego car closer to the side having more edge features. That is, when the semantically better features are too sparse, they don't provide any substantial improvement (see Fig. 4.10). Our approach inherently assimilates this understanding.

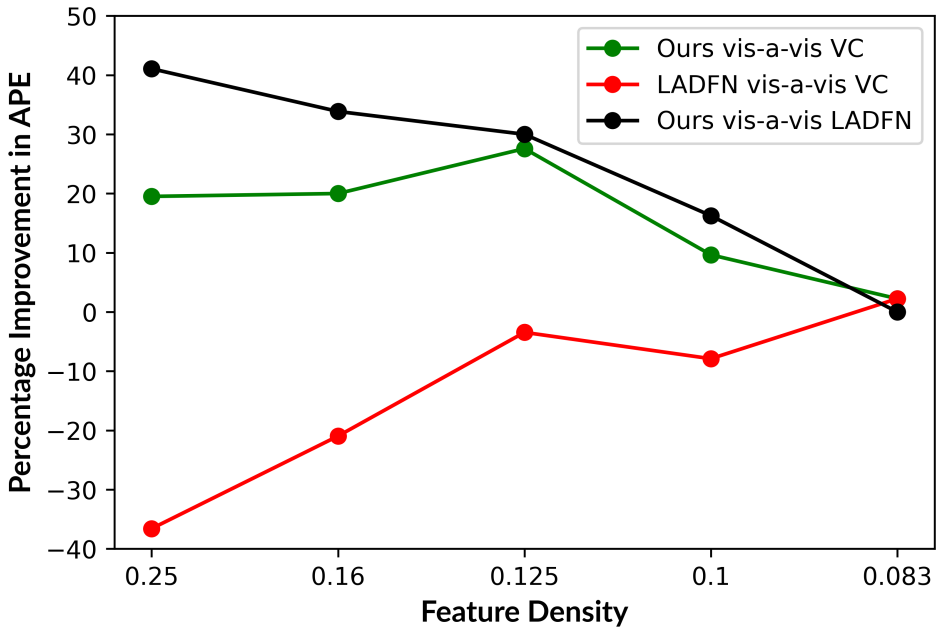

Figure 4.10 At higher semantic feature densities, we minimize drift by $\mathbf{40\%}$ and $\mathbf{20\%}$ over LADFN [25] and VC respectively.

4.5 Epilogue

In this article, we improved our active navigation framework for autonomous vehicles by using a weakly supervised approach to learn drift minimizing features and a cross-entropy method based model predictive control to generate optimal trajectories. We showed that our approach can identify and follow features that reduce drift while avoiding collisions with dynamic obstacles.

However, our work also has some limitations and directions for future research. First, our approach required the creation of a custom dataset that may not be representative of all possible scenarios and environments and is time consuming to create. We need a system that can take advantage of existing large scale self-driving datasets. Second, our scope was limited to simulation, which may not capture the complexity and uncertainty of real-world driving. We need to test our framework on real vehicles and environments, and address the challenges of sensor noise, communication delays, and computational constraints.

Chapter 5

Towards Scalable and Real World Active Navigation

5.1 Prologue

To run a scalable active navigation system on a real car, we need to accomplish many smaller goals. In this chapter, we lay the foundations for a differentiable version of LOAM [38] (a variant of ICP) that can learn to select features that reduce drift using standard dataset like KITTI [16]. If a classical algorithms like LOAM is made differentiable, we can use the flow of gradients to analyse which points contribute more for drift reduction, i.e, which are the most important points for the algorithm. This information can then be used by the low level planner. Unlike chapter 4, this process doesn't require creation of a separate dataset and can take advantage of large scale public datasets like KITTI. Differentiability in classical robotics algorithms has seen a lot of traction in recent years. We develop on the works like GradSLAM [19] to setup a differentiable ICP pipeline. We show, on KITTI dataset, that when the ground truth is available, we can back-propagate to find which point contribute most for drift reduction. Further, we show that we can train a network to recognise such points in the scene.
Moreover, we built a modular, general purpose autonomous driving platform (AutoDP) from scratch on a real car that can be used to test active navigation. AutoDP is a comprehensive software and hardware platform for testing and deployment of self-driving systems and its scope is not limited to active navigation. It is a standardized platform that researchers and industrialists can use to develop, test and deploy their algorithms. It offers ready-to-use implementations of the state of the art algorithms for all the levels of the self-driving stack.
Although making the complete LOAM pipeline differentiable and implementing active navigation algorithms on a real car is outside the scope of this thesis, the work done in the areas of Differentiable ICP and AutoDP provide a strong basis for future work in this direction.

5.2 DiffICP

We begin by formulating the standard ICP problem and identifying the sources of non-differentiability therein. The non-differentiability comes from hard distance metric and the LM solver. Then, we look

at ways of removing these non-differentiabilities. This is followed by exploring the advantages of a differentiable system and the ways in which a neural network can be coupled with it.

5.2.1 Standard ICP

The iterative closest point (ICP) problem is a problem of finding the best alignment between two sets of points. The points can represent shapes, images, or surfaces. The alignment is done by minimizing the distance between the points. The problem is solved by iteratively updating the alignment parameters until convergence.

Given two point clouds, Q and P, and the correspondence between the two, C:

$$\begin{aligned} Q &= \{q_1, \ldots, q_N\} \\ P &= \{p_1, \ldots, p_N\} \\ C &= \{(i, j)\} \end{aligned} \tag{5.1}$$

The ICP problem is to find the translation t and the rotation R such that squared errors in 5.2 is minimized.

$$E(R, t) = \sum_{(i,j) \in C} \left\| \boldsymbol{q}_i - R\boldsymbol{p}_j - t \right\|^2 \tag{5.2}$$

In a real scenario, the correspondence C in 5.1 is unknown. The ICP algorithms assumes the closest points in the two scans as correspondence and iterates until convergence.

The non-linear least squares in 5.2 can be solved using a variety on non-linear solvers. However, the most popular amongst them is the Levenberg–Marquardt (LM) solver. LM solver combines two approaches: the Gauss–Newton method, which is fast but may fail if the initial guess is far from the solution, and the steepest descent method, which is robust but slow. LM uses a damping parameter that controls how much the method behaves like Gauss–Newton or steepest descent. The parameter is adjusted dynamically depending on the progress of the optimization. The update step δx in an LM procedure is given in equation 5.3. J represents the jacobian while λ is the update step.

$$(J^\top J + \lambda I)\Delta x = -J^\top f(x) \tag{5.3}$$

5.2.2 Handling Non-Differentiability in Standard ICP

The non-differentiability in standard ICP comes from two sources. The first is the closest point selection. This selection is not differentiable because it involves finding the nearest neighbors between the point sets, which is a discrete operation. The second source of non-differentiability comes from the LM solver. The LM solver switches between Gauss Newton and Steepest Descent depending on the value of a damping parameter. When the parameter is large, the method behaves like Steepest Descent; when the parameter is small, the method behaves like Gauss Newton. The parameter is updated at each

iteration based on the improvement of the error function. However, this switching mechanism is non-differentiable, meaning that it does not have a well-defined derivative at the points where the switch occurs.

5.2.2.1 Stochastic relaxation of 1-NN

To make the nearest neighbour selection differentiable, we need to replace this discrete operation with a continuous one. One way of doing this is to use soft nearest neighbor assignment, which assigns a probability distribution over the possible neighbors for each point. In *Neural Nearest Neighbors* [26] multiple approaches are proposed for the relaxation on the general K-NN problem. In the case of ICP, we have a 1-NN selection. Hence, we use the stochastic relaxation of 1-NN proposed in [26] to make the selection procedure differentiable.

$$w[k|\alpha, t] = \frac{\exp(\alpha_k/t)}{\sum_{j \in K} \exp(\alpha_j/t)}, \tag{5.4}$$

Equation 5.4 gives probability of a point k being the nearest neighbor of the given query point. α is the inverse of euclidean distance metric and t is the temperature of the softmax. With this distribution of the nearest-neighbors, we can define a new point $p_i^{'}$ which is the soft nearest neighbor of the point p_i as shown in equation 5.5, where w_k^i is coming from 5.4.

$$p_i^{'} = \sum_k w_k^i p_k \tag{5.5}$$

This can then be used to generate a soft error function:

$$E(R, t) = \sum_{(i,j) \in C} \left\| \boldsymbol{q}_i - R\boldsymbol{p}_j^{'} - t \right\|^2 \tag{5.6}$$

5.2.2.2 Soft LM

To make the LM method differentiable, we can use a smoother way of adjusting the damping parameter and switching between the techniques. One way to do this, as done in [19], is to use the look ahead residual and the current residual to decide how much to increase or decrease the damping parameter. The look ahead residual is the error function value that we would get if we took a full Gauss Newton step. The current residual is the error function value that we have at the current iteration. If the look ahead residual is much smaller than the current residual, it means that we are close to the solution and we can reduce the damping parameter. If the look ahead residual is much larger than the current residual, it means that we are far from the solution and we need to increase the damping parameter.
The soft switch can happen using *generalized logistic function*. A generalized logistic function is a smooth function that transitions from one value to another as its input changes. We can use this function to blend the Gauss Newton and Steepest Descent directions based on the value of the damping parameter. When the damping parameter is large, the function gives more weight to the Steepest Descent

direction; when the damping parameter is small, the function gives more weight to the Gauss Newton direction. This way, we can avoid a sudden switch and make the LM method differentiable.
The soft switch of the damping parameter for LM is given in equation 5.7. r_{i+1} is the look ahead residual while D, σ are tunable parameters of the generalized logistic function.

$$\lambda = \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{1 + De^{-\sigma(r_{i+1} - r_i)}} \tag{5.7}$$

5.2.3 Advantages of Differentiability

Once we have made the ICP algorithm differentiable, we can use the gradient information to understand which point are more important for drift reduction. Consider the following weighted variant of ICP, where θ is the importance of a particular pair of points:

$$E(R,t) = \sum_{(i,j)\in C} \theta_{i,j} \left\| \boldsymbol{q}_i - R\boldsymbol{p}_j^{'} - t \right\|^2, \tag{5.8}$$

If ICP is differentiable, we can use the ground truth information of pose to back-propagate the error gradients and figure out the ideal values for the weights θ that minimise the error. The drift reducing point that we get using this procedure are similar to what we get with the approach discussed in chapter 4.

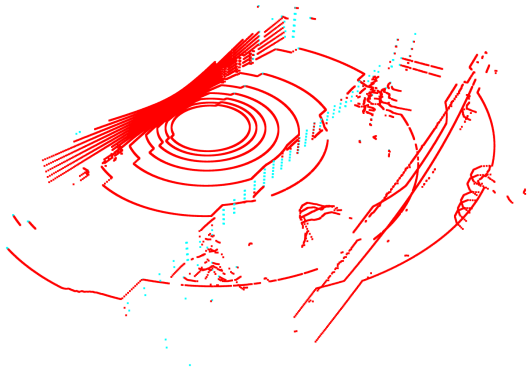

Figure 5.1 Points in blue are the most important points for drift reduction obtained using DiffICP

To get important, drift minimizing, points using differentiability, we just need a set of pointclouds and their relative transform. This information is readily available at large scale from standard self-driving datasets. We can then train a network to identify such points given a pointcloud (or their importance weights) and then use them in the weighted icp formulation 5.8. The formulation then becomes:

$$E(R,t) = \sum_{(i,j)\in C} \theta_{i,j}^{nn} \left\| \boldsymbol{q}_i - R\boldsymbol{p}_j^{'} - t \right\|^2, \tag{5.9}$$

where θ^{nn} is coming from a neural network.
Figure 5.2.3 shows the drift comparison of the standard ICP against weighted ICP where weights are learned from a neural network using the DiffICP procedure discussed above.

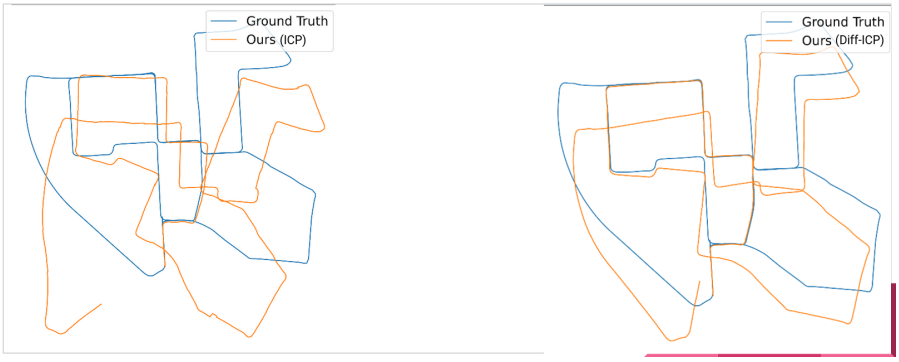

Figure 5.2 DiffICP outperforms standard ICP on the KITTI Odometry Benchmark (Sequence 00)

The DiffICP pipeline acts as a proof of concept that important drift minimizing features can be learned using differentiability. This means that the pipeline can optimize its parameters by computing gradients with respect to a loss function. However, making the complete LOAM pipeline differentiable requires development of additional differentiable modules which is beyond the scope of this thesis. For example, the feature extraction module in LOAM are based on heuristic rules that are not amenable to gradient-based optimization. Therefore, such modules would need to be replaced or modified with differentiable alternatives to enable end-to-end learning of the LOAM pipeline.

5.3 AutoDP — Autonomous Driving Platform

AutoDP is a complete software cum hardware platform for testing and deployment of self-driving systems. Recent years have seen a rampant rise of self-driving cars and the emergence of a plethora of algorithms for various levels of the self-driving tech stack. To facilitate extensive application-oriented research in this area, there is an imminent need for a standardized platform that researchers and industrialists can use to develop, test and deploy their algorithms. AutoDP is a one stop solution for this. It provides plug-and-play implementations of the state of the art algorithms for all the levels of the self-driving stack. Further, it provides APIs at multiple levels of complexity for developers to tap into the software stack and customize it as per their needs. It also provides templates for programming that developers can use to make their custom modules easily pluggable into the AutoDP system. AutoDP allows for extensive testing of algorithms on a high fidelity simulation environment like Carla before deployment on an actual vehicle. The symphony between the simulator and the car will enable developers to share the same codebase between the two use cases.

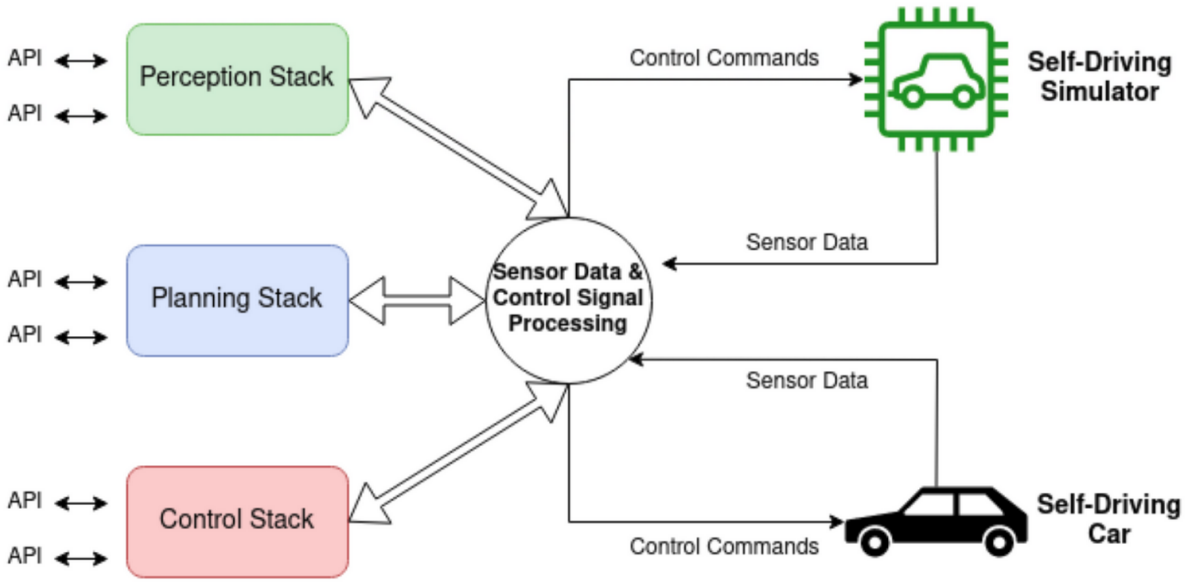

Figure 5.3 AutoDP: Autonomous Driving Platform

5.3.1 Software Stack and APIs

The software stack of AutoDP provides an implementation of the well known self-driving algorithms. These include algorithms for all three layers: perception, planning and control. The following is a non-exhaustive list of available algorithms that work out of the box:

1. Perception: Visual Inertial Odometry, LiDAR Odometry and Mapping (LOAM), ORB SLAM, Object Detection, RGB Segmentation, Lidar Segmentation, Traffic Trajectory Prediction, etc.
2. Planning: Sampling-based planners like Frenet, optimization-based planners like MPC, global planners like A*, etc.
3. Control: v(velocity) and w(angular velocity) control, thrust and steering angle control, feedback controllers with kinematic and dynamic models of the car, etc.

Each of these algorithms will be available as a plug and play module in the AutoDP stack. The user will have access to each level in this stack, with APIs exposed at varying complexity levels. The user will be able to use these APIs to modify the existing modules as required. Further, AutoDP will provide a framework to develop custom modules that can easily sync with the AutoDP stack and can be quickly tested and deployed. A typical use case of AutoDP might is shown in 5.4

5.3.2 Simulation and Real World Symphony

High fidelity sensors like Carla provide implementation for all common self driving sensors like LiDAR, camera, IMU, etc. Further, the physics model provided is sophisticated and editable enough to

```
import AutoDP

# Creating autoDP instance
autoDP = AutoDP(interface=`simulation`)

# Using an off the shelf perception system
odometry = autoDP.perception.LOAM()

# Using a custom planner, defined with autoDP template
planner = autoDP.planning.create_custom(custom_planner_definition)

# Using an off the shelf controller
controller =  autoDP.control.VW()

# creating the drive
autoDrive = audoDP.createDrive(odometry, planner, controller)

# Start the drive
# This will also expose interaction/visualization tools
# to allow giving goal locations, monitor progress, etc.
autoDrive.loop()
```

Figure 5.4 Proposed API Structure for AutoDP

manifest any realistic vehicle. AutoDP's interface can switch between the simulator and the real vehicle with just one parameter change. This will allow developers to stress test their algorithms on a simulator which can then be deployed to the real vehicle with minimal tweaking.

5.4 Implementation

The project, titled AutoDP (Autonomous Driving Platform), involves the implementation of perception, planning, and control stacks on a Mahindra E2O vehicle. Its progress can be tracked `https://robotics.iiit.ac.in/auto_dp/`. As of now, the car is capable of autonomous pickup and drop-off at any location on campus while avoiding dynamic obstacles. The following major modules have been implemented:

1. **Global Localization and Mapping:** We have used LOAM [38] for generating a global map of the campus and have marked roads on it. The same LOAM algorithm is run in *relocalization* mode for global localization in the campus.

2. **Global Planning:** A *Dijikstra* like path search algorithm is employed for generating a global plan.

3. **Basic Obstacle Detection:** LIDAR and Camera based segmentation approaches are used for obstacle detection and generating a local occupancy map.

4. **Basic Local Planner:** The local occupancy map is used by a Frenet [37] planner to generate obstacle free trajectories.

5. **Trajectory Tracking:** A Stanley controller [18] is used for trajectory tracking.

6. **Low Level Control:** We have developed an abstraction that takes in desired speed and steering angle and converts it into respective PID commands.

7. **Low Level Interface:** We have developed an abstraction that takes in the PID commands and convert in to the required CAN messages to be sent to the car internals.

5.5 Epilogue

In this chapter, we proposed a novel approach that combines differentiablity with classical odometry algorithms (ICP) to identify drift minimizing features in a point cloud. We have also developed a modular and scalable platform for testing and deploying self-driving systems on a real car. This works sets up the groundwork for scalable active navigation in real world.

However, there are still many challenges and limitations that need to be addressed in future work. For example, how to make the whole LOAM pipeline differentiable? How to test and compare different active navigation methods on a real car using our platform? How to incorporate high-level planning and decision making into active navigation? How to ensure safety and reliability of active navigation systems? These are interesting directions that can be pursued in subsequent works.

Chapter 6

Conclusions

This thesis has shown that active navigation in the LiDAR setting can significantly improve the perception system. Our first approach used reinforcement learning to generate drift-minimizing trajectories. By taking in the position of important features in a LiDAR map and using a carefully designed reward function, the agent was able to learn behaviors like overtaking and slowing down to minimize drift. We showed through extensive comparisons in a variety of dynamic scenes that our architecture was superior in achieving drift-minimized navigation compared to methods that do not actively navigate to minimize state estimation errors.

However, our first approach had its limitations. The reinforcement learning method used was uninterpretable and took many iterations to train. It also generalized poorly to new environments. In our second research article, we adopted a more systematic approach to significantly improve active navigation. A key insight was that a simple geometric measure was insufficient to robustly identify drift-minimizing LiDAR points. Hence, we used a weekly supervised approach to learn drift-minimized features using gradient-based neural activation techniques like GradCAM. Further, instead of using an RL agent, we incorporated a GPU-driven batch optimized MPC for trajectory generation which could solve thousands of trajectories in real time over a long horizon. This gave us similar behaviors as the RL-based planner while being much more interpretable and secure.

We further developed a novel approach that combines differentiability with classical odometry algorithms like ICP to identify drift-minimizing features in a point cloud. This allowed us to improve upon the limitations of our first approach and develop a more robust and effective method for active navigation. We also developed a modular and scalable software-cum hardware platform for testing and deploying self-driving systems on a real car.

Our research also paves way to future development in this area, including but not limited to more scalable ways of identifying drift minimizing points for LiDAR Odometry, testing and evaluation of active navigation systems on a real car, etc. Finally, we plan to extend the same core idea to navigate robots in a factory floor while always maintaining sufficient proximity and connectivity to WiFi access points, which is important in networked/cloud robotics applications. We also plan to develop a more general framework for active-navigation in LIDAR settings wherein we first hallucinate how a scene

would look like in the future and then have a prediction of drift on this hallucination. Then we make informed decisions on how to navigate while minimizing drift.

In summary, our research has made significant strides in the field of active navigation and self-driving technology. By combining reinforcement learning, weekly supervised learning, gradient-based neural activation techniques, and GPU-driven batch optimized MPC, we have developed a robust and effective method for active navigation in the LiDAR setting. We hope that our research will pave the way for further advancements in this field and contribute to the development of safer and more efficient self-driving systems.

Chapter 7

Publications

7.1 Relevant Publications

1. **M. Omama**, S. S. V. S., S. Chinchali and K. M. Krishna, *"LADFN: Learning Actions for Drift-Free Navigation in Highly Dynamic Scenes,"* **2022 American Control Conference (ACC),** 2022, pp. 1200-1207, doi: 10.23919/ACC53348.2022.9867473.

2. **M. Omama**, S. S. V. S., S. Chinchali, Singh A.K. and K. M. Krishna, *"Drift Reduced Navigation with Deep Explainable Features,"* **2022 International Conference on Intelligent Robots and Systems (IROS),** 2022, arXiv preprint arXiv:2203.06897.

Bibliography

[1] V. K. Adajania, A. Sharma, A. Gupta, H. Masnavi, M. Krishna, and A. K. Singh. Multi-modal model predictive control through batch non-holonomic trajectory optimization: Application to highway driving. *IEEE Robotics and Automation Letters*, 2022.

[2] H. Andersen, J. Alonso-Mora, Y. H. Eng, D. Rus, and M. H. Ang. Trajectory optimization and situational analysis framework for autonomous overtaking with visibility maximization. *IEEE Transactions on Intelligent Vehicles*, 5(1):7–20, 2019.

[3] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[4] P. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[5] S. Bhuvanagiri and K. M. Krishna. Motion in ambiguity: Coordinated active global localization for multiple robots. *Robotics and Autonomous Systems*, 58(4):399–424, 2010.

[6] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[7] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(3), 2010.

[8] G. Costante, C. Forster, J. Delmerico, P. Valigi, and D. Scaramuzza. Perception-aware path planning. *arXiv preprint arXiv:1605.04151*, 2016.

[9] P. Del Moral. Nonlinear filtering: Interacting particle resolution. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 325(6):653–658, 1997.

[10] F. Dellaert and M. Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[12] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. *SIAM Journal on Computing*, 27(2):583–604, 1998.

[13] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[14] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza. Pampc: Perception-aware model predictive control for quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[15] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3-4):195–207, 1998.

[16] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American Control Conference*, pages 2296–2301, 2007.

[19] K. M. Jatavallabhula, S. Saryazdi, G. Iyer, and L. Paull. gradslam: Automagically differentiable slam. *arXiv preprint arXiv:1910.10672*, 2019.

[20] M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[21] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[22] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[23] C. Leung, S. Huang, and G. Dissanayake. Active slam using model predictive control and attractor based exploration. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5031. IEEE, 2006.

[24] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220. IEEE, 2019.

[25] M. Omama, S. S. V. S., S. Chinchali, and K. M. Krishna. Ladfn: Learning actions for drift-free navigation in highly dynamic scenes. In *2022 American Control Conference (ACC)*, pages 1200–1207, 2022.

[26] T. Plötz and S. Roth. Neural nearest neighbors networks. *Advances in Neural information processing systems*, 31, 2018.

[27] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[28] M. Rao, G. Dudek, and S. Whitesides. Minimum distance localization for a robot with limited visibility. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2438–2445. IEEE, 2005.

[29] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques. Pythonrobotics: a python code collection of robotics algorithms. *arXiv preprint arXiv:1808.10703*, 2018.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[32] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[33] T. Shan and B. Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[34] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and R. Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020.

[35] Q. M. Thomas, O. Wasenmüller, and D. Stricker. Delio: Decoupled lidar odometry. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1549–1556. IEEE, 2019.

[36] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[37] M. Werling, J. Ziegler, S. Kammel, and S. Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.

[38] J. Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[39] S. Zhao, Z. Fang, H. Li, and S. Scherer. A robust laser-inertial odometry and mapping method for large-scale highway environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1285–1292. IEEE, 2019.