UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Jatan Shrestha

Sampling-based Bi-level Optimization aided by Behaviour Cloning for Autonomous Driving

Master's Thesis (30 ECTS)
Robotics and Computer Engineering

Supervisor(s): Arun Kumar Singh, PhD

Tartu 2023

Sampling-based Bi-level Optimization aided by Behaviour Cloning for Autonomous Driving

Abstract:
Autonomous driving has a natural bi-level structure. The upper behavioural layer aims to provide appropriate lane change, speeding up, and braking decisions to optimize a given driving task. The upper layer can only indirectly influence the driving efficiency through the lower-level trajectory planner, which takes in the behavioural inputs to produce motion commands for the controller. Existing sampling-based approaches do not fully exploit the strong coupling between the behavioural and planning layer. On the other hand, Reinforcement Learning (RL) can learn a behavioural layer while incorporating feedback from the lower-level planner. However, purely data-driven approaches often fail regarding safety metrics in dense and rash traffic environments. This thesis presents a novel alternative; a parameterized bi-level optimization that jointly computes the optimal behavioural decisions and the resulting downstream trajectory. The proposed approach runs in real-time using a custom Graphics Processing Unit (GPU)-accelerated batch optimizer and a Conditional Variational Autoencoder (CVAE) learnt warm-start strategy and extensive experiments on challenging traffic scenarios show that it outperforms state-of-the-art Model Predictive Control (MPC) and RL approaches regarding collision rate while being competitive in driving efficiency.

Keywords:
Autonomous Driving, Bi-level Optimization, Behavioural Cloning, Differentiable Optimization, Conditional Variational Autoencoder

CERCS:
T125 Automation, robotics, control engineering

Proovivõtupõhine bi-taseme optimeerimine, mida toetab autonoomse sõidu käitumise kloonimine

Lühikokkuvõte:
Autonoomsel sõidul on loomulik kahetasandiline struktuur. Ülemise käitumiskihi eesmärk on tagada sõiduraja asjakohane muutmine, kiirendamine ja pidurdamisotsused, et optimeerida konkreetset sõiduülesannet. Ülemine kiht saab sõiduefektiivsust mõjutada ainult kaudselt läbi madalama astme trajektoori planeerija, mis võtab sisse käitumuslikud sisendid, et toota kontrollerile liikumiskäske. Olemasolevad valimipõhised lähenemisviisid ei kasuta täielikult ära käitumis- ja planeerimiskihi vahelist tugevat haakumist. Teiselt poolt saab Reinformeerimine Learning (RL) õppida käitumuslikku kihti, lisades samal ajal tagasisidet madalama taseme planeerijalt. Puhtalt andmepõhised lähenemisviisid ei ole aga sageli seotud ohutusnäitajatega tihedates ja lööbelistes liikluskeskkondades. Käesolevas töös on esitatud uudne alternatiiv, parameetriline kahetasandiline optimeerimine, mis arvutab ühiselt välja optimaalsed käitumisotsused ja sellest tulenevad trajektoorid. Väljapakutud lähenemine töötab reaalajas, kasutades kohandatud graafika töötlemise üksust (GPU) -kiirendatud partii optimeerijat ja tingimuslikku variatsiooniautoencoderit (CVAE) õppinud sooja käivitamise strateegiat ja ulatuslikud katsed keerukate liiklusstsenaariumidega näitavad, et see ületab tipptasemel mudeli ennustuskontrolli (MPC) ja RL-i lähenemisviise kokkupõrke kiiruse osas, olles samal ajal sõidutõhususe konkurentsivõimeline.

Võtmesõnad:
Autonoomne sõitmine, bi-taseme optimeerimine, käitumuslik kloonimine, diferentseeruv optimeerimine, tingimuslik variatiivne autoenkoder

CERCS:
T125 Automatiseerimine, robootika, juhtimistehnika

Contents

List of Algorithms

List of Figures

List of Tables

1 Introduction

1.1 Motion Planning For Autonomous Driving

Motion planning is defined as using computational methods to generate a robot's motion to achieve a specified task. Motion planning is a critical task in autonomous driving. It involves finding a safe and efficient path for the vehicle to follow while remaining collision-free and within a lane. Motion planning for autonomous driving can be divided into two hierarchical components [3].

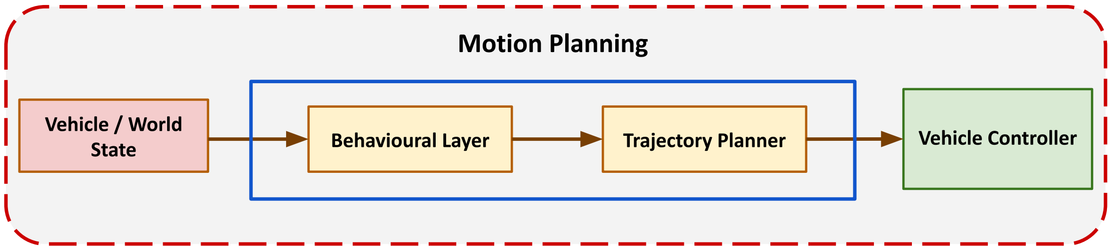

Figure 1. Motion planning for autonomous driving is divided into two hierarchical components- the higher-level behavioural layer and the corresponding lower-level trajectory planner. The behavioural layer makes lane-changing, speeding up, and braking decisions based on the traffic scenarios. The trajectory planner takes in the behavioural inputs to produce motion commands for the controller.

The higher-level behavioural layer computes decisions such as lane change, speeding up and braking based on the traffic scenario and the driving task. This layer indirectly influences the driving efficiency through the lower-level trajectory planner. In dense traffic scenarios, the behavioural layer can shift from the congested to the free lane. Similarly, the behavioural layer can request a slowing down of the vehicle when the traffic light switches to red. A slowing down can also be requested when the neighbouring vehicle cuts in sharply in front of the ego-vehicle.

The trajectory planner takes in the behavioural inputs, parameterized in terms of set points for longitudinal velocity and lateral offsets from the centre line, to produce motion commands. Parameterizing behavioural inputs in terms of set points for longitudinal velocity and lateral offsets allows us to naturally integrate the behavioural layer with the lower-level trajectory planner [4], [5] [6], [7]. The behavioural layer is critical for driving in dense traffic as it can guide the lower-level planner into favourable state-space regions; in the same way, a collision-free global plan can make the task of the local planner easier.

The two layers have a strong inter-dependency as the higher-level behavioural layer can indirectly affect the driving efficiency through the lower-level trajectory planner. However, existing trajectory sampling approaches [5] [6] do not fully exploit the strong coupling between the behavioural and planning layer (see Figure 2(a)). Prior works have no mechanism for modifying the behavioural inputs based on how the associated lower-level trajectory performs on the given driving task.

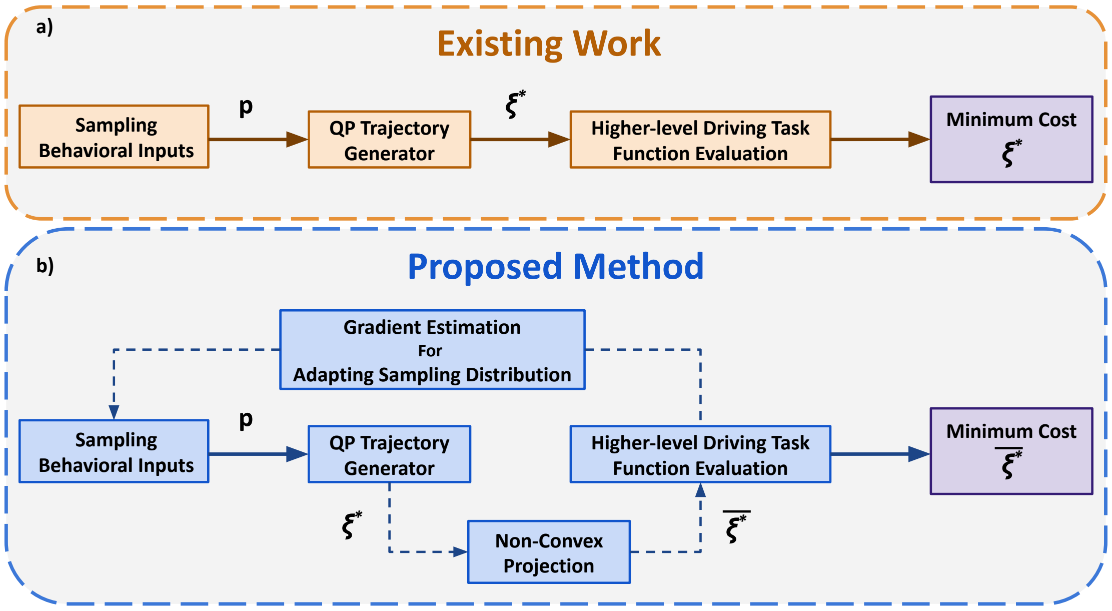
a)
Existing Work
Sampling Behavioral Inputs
p
QP Trajectory Generator
ξ^*
Higher-level Driving Task Function Evaluation
Minimum Cost ξ^*
b)
Proposed Method
Gradient Estimation For Adapting Sampling Distribution
Sampling Behavioral Inputs
p
QP Trajectory Generator
Higher-level Driving Task Function Evaluation
Minimum Cost $\overline{\xi}^*$
ξ^*
Non-Convex Projection
$\overline{\xi}^*$

Figure 2. (a). Existing works draw behavioural inputs from a distribution and solve a simple QP trajectory planner for all those inputs. However, there is no mechanism to modify the behavioural input sampling based on the performance of the lower-level planner on the driving task. (b). The proposed pipeline addresses this issue by adding a gradient estimation block and a projection operator to satisfy constraints.

Reinforcement learning (RL) techniques address this drawback by learning the behavioural layer. The rewards from the environment act as feedback to modify the behavioural inputs while considering the effect of the lower-level planner [8], [9]. Though effective, especially in sparse traffic scenarios, the purely data-driven approaches typically struggle with safety metrics in dense and rash traffic environments.

1.2 Contributions

The contributions of this thesis are the following:

- It proposes a bi-level optimization where the upper-level variables represent the behavioural inputs while that at the lower level represent the associated motion plans. The bi-level optimization estimates the direction in which the behavioural inputs need to be perturbed to improve the trajectory optimization with respect to the driving task.

- It combines Quadratic Programming (QP) with gradient-free optimization for solving the bi-level problem. In particular, it proposes an approach where sampled behavioural inputs are used to solve lower-level trajectory optimization for all of them as it allows for estimating gradients of the driving cost with respect to the behavioural inputs (see Figure 2(b)).

- The thesis also proposes a behavioural cloning framework to learn a distribution over behavioural inputs to accelerate the convergence of the proposed bi-level optimizer. More specifically, the sampling portion of the proposed bi-level approach employs a Conditional Variational Autoencoder (CVAE) to generate a distribution over behavioural inputs.

- The thesis empirically shows improvements over the state-of-the-art Model Predictive Control (MPC) and RL-based approaches in dense traffic scenarios.

1.3 Outline

- **Preliminaries**: This chapter introduces the mathematical preliminaries and covers essential background on bi-level optimization and behavioural cloning.

- **Related Works**: This chapter reviews prior works and connections to the proposed method.

- **Methodology**: This chapter explains the bi-level optimizer for combined planning and the behavioural cloning framework to warm-start the proposed optimizer.

- **Results**: This chapter describes the implementation details, benchmarking scenarios and presents the results of the experiments.

- **Conclusion**: This chapter summarizes the thesis and outlines directions for future work.

2 Preliminaries

This chapter introduces essential concepts necessary to understand the rest of the thesis. It describes the mathematical preliminaries- notation, trajectory and behavioural input parameterizations. It briefly overviews central concepts such as bi-level optimization and behavioural cloning.

2.1 Symbols and Notation

Normal font lower-case letters represent scalars, and bold font variants represent vectors. The upper-case bold font letters represent matrices. The superscript T denotes the transpose of a matrix or a vector.

2.2 Trajectory Parameterization

Using the differential flatness of the bi-cycle car model, we aim to directly plan in the positional space $(x(t), y(t))$ of the ego-vehicle. Thus, the position-level trajectory of the ego-vehicle is parameterized in terms of polynomials in the following form:

$$\begin{bmatrix} x(t_1), \dots, x(t_m) \end{bmatrix} = \mathbf{W}\mathbf{c}_x, \begin{bmatrix} y(t_1), \dots, y(t_m) \end{bmatrix} = \mathbf{W}\mathbf{c}_y, \tag{1}$$

where, $\mathbf{W}$ is a matrix formed with time-dependent polynomial basis functions and $(\mathbf{c}_x, \mathbf{c}_y)$ are the coefficients of the polynomial. The derivatives can be expressed in terms of $\dot{\mathbf{W}}, \ddot{\mathbf{W}}$.

2.3 Behavioral Input Parameterization

The following list of behavioural inputs is considered in the formulation:

- The planning horizon is split into m parts, and a desired lateral offset set point $(y_{d,m})$ is assigned to each of these segments (see Figure 3(a)). Similarly, longitudinal velocity set point $(v_{d,m})$ is assigned to each of the m segments.
- The goal positions are included along longitudinal (x_f) and lateral directions (y_f) as behavioural inputs.

The behavioural inputs are stacked into one parameter vector:

$$\mathbf{p} = \begin{bmatrix} y_{d,1}, y_{d,2}, \dots, v_{d,1}, v_{d,2}, \dots v_{d,m}, x_f, y_f \end{bmatrix}. \tag{2}$$

Note that not all elements of $\mathbf{p}$ need to be used simultaneously in the lower-level trajectory planner, and (2) represents all the behavioural input parameterization that can be accommodated within the proposed bi-level optimizer presented in Chapter 4.1.

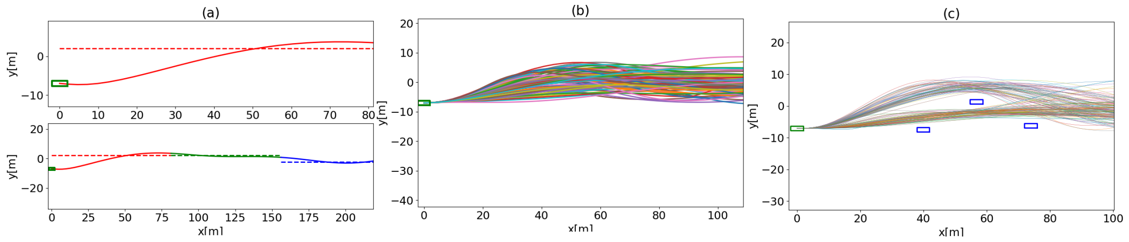

Figure 3. (a). A standard parameterization for behavioural inputs is set points for lateral offset used to induce lane-change manoeuvres in the ego vehicle. For long-horizon planning, the trajectory can be split into multiple parts and assign a lateral offset set point to each of them, as shown in the bottom figure. (b). Shows different trajectories generated by sampling lateral offsets $y_{d,m}$ and forward velocity $v_{d,m}$ set points from a Gaussian distribution and using them in (3a) - (3d). (c). The proposed bi-level optimizer can parallelly solve the lower-level trajectory planning for all the sampled behavioural inputs. The figure represents the distribution of trajectories obtained in a cluttered environment, with a blue rectangle representing stationary parked vehicles.

2.4 Lower-Level Trajectory Planning

The lower-level trajectory optimization is formulated in the Frenet-frame: longitudinal $x(t)$ and lateral $y(t)$ motions of the ego-vehicle occur along and orthogonal to the reference centre line, respectively [10].

$$\min \sum_{t} c_s(x(t), y(t)) + c_l(x(t), y(t)) + c_v(x(t), y(t)) \tag{3a}$$

$$(x(t_0), y(t_0), \dot{x}(t_0), \dot{y}(t_0), \ddot{x}(t_0), \ddot{y}(t_0)) = \mathbf{b}_0. \tag{3b}$$

$$(x(t_f), y(t_f), \dot{y}(t_f)) = (x_f, y_f, 0). \tag{3c}$$

$$g_j(x(t), y(t)) \leq 0, \forall j, t \tag{3d}$$

$$c_s(x(t), y(t)) = (\ddot{x}(t))^2 + \ddot{y}(t)^2 \tag{4a}$$

$$c_l(x(t), y(t)) = \ddot{y}(t) - k_p(y(t) - y_{d,m}) - k_v\dot{y}(t))^2 \tag{4b}$$

$$c_v(x(t), y(t)) = (\ddot{x}(t) - k_p(\dot{x}(t) - v_{d,m}) \tag{4c}$$

The first term ($c_s(.)$) in the cost function (3a) ensures smoothness in the planned trajectory by penalizing high accelerations. The last two terms ($c_o(.), c_v(.)$) model the tracking of lateral offset and forward velocity set-points, respectively and are inspired by works like [8]. Different $y_{d,m}, v_{d,m}$ are assigned to m trajectory segments. For the lateral-offset tacking, a Proportional Derivative (PD) like tracking with gain (k_p, k_v) is defined, and only a proportional term is used for the velocity part.

Equality constraints (3b) ensures that the planned trajectory satisfies the initial boundary conditions. Thus, vector $\mathbf{b}_0$ is simply a stacking of initial position, velocity, and acceleration. The final boundary conditions are represented through constraints (3c). Inequalities

(3d) enforces collision avoidance, velocity, acceleration, centripetal acceleration, curvature bounds, and lane boundary constraints. Table 1 summarizes the set of inequality constraints used in the lower-level optimization ($g_j(x(t), y(t)) \leq 0$).

Table 1. List of Inequality Constraints used in the lower-level optimization

Constraint Type	Expression	Parameters
Collision Avoidance	$-\frac{(x(t)-x_{o,i}(t))^2}{a^2} - \frac{(y(t)-y_{o,i}(t))^2}{b^2} + 1 \leq 0$	$\frac{a}{2}, \frac{b}{2}$: axis of the circumscribing ellipse of vehicle footprint. $x_{o,i}(t), y_{o,i}(t)$: trajectory of neighboring vehicles
Velocity bounds	$\sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \leq v_{max}$	v_{max}: maximum velocity of the ego-vehicle
Acceleration bounds	$\sqrt{\ddot{x}(t)^2 + \ddot{y}(t)^2} \leq a_{max}$	a_{max}: maximum acceleration of the ego-vehicle
Curvature bounds	$-\kappa_{max} \leq \frac{\ddot{y}(t)\dot{x}(t)-\ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2+\dot{y}(t)^2)^{1.5}} \leq \kappa_{max}$	$-\kappa_{max}$: maximum curvature bound for the ego-vehicle trajectory
Centripetal Acceleration bounds	$-c_{max} \leq \dot{x}(t)^2\kappa(x(t)) \leq c_{max}$	c_{max}: maximum centripetal acceleration bound for the ego-vehicle.
Lane boundary	$y_{lb}(x(t)) \leq y(t) \leq y_{ub}(x(t))$	y_{lb}, y_{ub}: Lane limits as a function of the ego-vehicle's position.

2.5 Bi-level Optimization

General optimization problems are single-level problems that can be formulated as:

$$\begin{aligned} &\text{minimize} && f_0(\boldsymbol{\xi}), && && (5)\\ &\text{subject to} && g_i(\boldsymbol{\xi}) \leq 0, && i = 1, ..., a && (6)\\ & && h_i(\boldsymbol{\xi}) = 0, && i = 1, ..., b && (7)\end{aligned}$$

where $\boldsymbol{\xi} \in \mathbb{R}^n$ is the optimization variable, $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the cost function, $g_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, ..., a$ is the inequality constraint function, and $h_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, ..., b$ is the equality constraint function [11].

However, autonomous driving has a natural bi-level structure divided into two hierarchical components leading to a bi-level optimization problem (see Figure 4) [12]. More formally,

$$\begin{aligned} &\underset{\mathbf{p} \in \mathbf{P}}{\text{minimize}} && F(\mathbf{p}, \boldsymbol{\xi}^*), && (8)\\ &\text{subject to} && \boldsymbol{\xi}^* \in S(\mathbf{p}) && (9)\end{aligned}$$

where $S(\mathbf{p})$ is the set of optimal solutions of $\mathbf{p}$-parameterized problem:

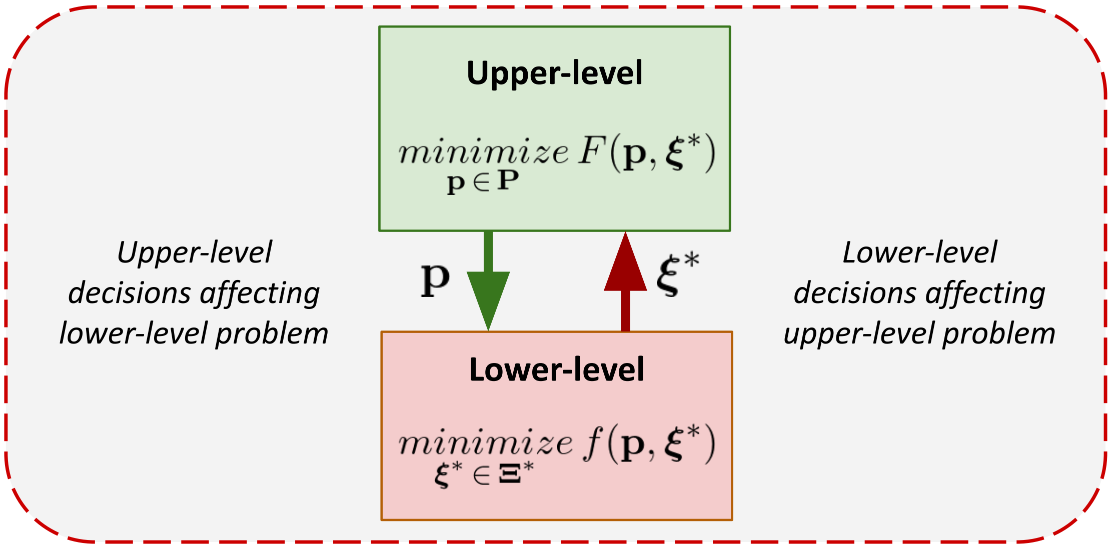
Upper-level
$\underset{\mathbf{p}\in\mathbf{P}}{minimize}\, F(\mathbf{p},\boldsymbol{\xi}^*)$
Upper-level decisions affecting lower-level problem
$\mathbf{p}$
$\boldsymbol{\xi}^*$
Lower-level decisions affecting upper-level problem
Lower-level
$\underset{\boldsymbol{\xi}^*\in\boldsymbol{\Xi}^*}{minimize}\, f(\mathbf{p},\boldsymbol{\xi}^*)$

Figure 4. Overview of Bi-level Optimization-a hierarchical optimization problem where one problem is embedded within another. The higher-level optimization task is referred to as the upper-level problem while the lower-level optimization task is called the lower-level problem.

$$S(\mathbf{p}) := \underset{\boldsymbol{\xi}^*\in\boldsymbol{\Xi}^*}{arg\,min}\ f(\mathbf{p},\boldsymbol{\xi}^*), \tag{10}$$

Problem (8) is the upper-level problem while Problem (10) is the lower-level problem parameterized by upper-level's decision $\mathbf{p}$. The variables $\mathbf{p} \in \mathbb{R}^{n_\mathbf{p}}$ are the upper-level variables and $\boldsymbol{\xi}^* \in \mathbb{R}^{n_{\boldsymbol{\xi}^*}}$ are the lower-level variables. The objective functions are given by $F, f : \mathbb{R}^{n_\mathbf{p}} \times \mathbb{R}^{n_{\boldsymbol{\xi}^*}} \rightarrow \mathbb{R}$.

2.6 Behavioural Cloning

Behavioural cloning is one of the most popular approaches for end-to-end autonomous driving. Behavioural cloning is a form of supervised learning that mimics expert demonstrations from offline collected expert data [13], [14]. Given an expert policy $\pi^*(\mathbf{s})$ with access to the environment state $\mathbf{s}$, we can execute this policy to produce a dataset $\mathbb{D} = \{\mathbf{o}_i, \boldsymbol{\tau}_e^i\}_{i=1}^N$, where $\mathbf{o}_i$ are sensory data observations and $\boldsymbol{\tau}_e^i = \pi^*(\mathbf{s}_i)$ is the resulting expert trajectory. The goal of behavioural cloning is to learn a policy π, parameterized by $\boldsymbol{\theta}$, to produce trajectories $\boldsymbol{\xi}^*$ similar to expert $\boldsymbol{\tau}_e$ based only on observations $\mathbf{o}$. The optimal parameters $\boldsymbol{\theta}^*$ are obtained by minimizing an imitation cost $\mathcal{L}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{arg\,min} \sum_i \mathcal{L}\left(\pi(\boldsymbol{o}_i;\boldsymbol{\theta}), \boldsymbol{\tau}_e^i\right) \tag{11}$$

2.6.1 Differentiable Optimization Layers

Existing approaches for behavioural cloning model relationship between the input observations **o** and output trajectory $\boldsymbol{\xi}^*$ explicitly in an imperative node for some differentiable function π (see Figure 5(a)):

$$\boldsymbol{\xi}^* = \pi(\mathbf{o}; \boldsymbol{\theta}) \tag{12}$$

Alternatively, the input-output relationship can be modelled implicitly in a declarative node as a solution to an optimization problem [2]. The imperative and declarative nodes can be parameterized and learned with backpropagation using standard gradient descent optimization algorithms [15]. Additionally, these nodes can co-exist in the same computation graph. In this thesis, the proposed behavioural cloning framework combines feedforward networks, an instance of an imperative node, with a declarative node that implicitly defines a layer as a solution to an optimization problem of the form (see Figure 5(b)):

$$\boldsymbol{\xi}^* \in \underset{\boldsymbol{\xi} \in C}{arg\,min}\, \pi(\mathbf{p}(\mathbf{o}; \boldsymbol{\theta}), \boldsymbol{\xi}) \tag{13}$$

Chapter 4.3 explains the proposed framework in more detail.

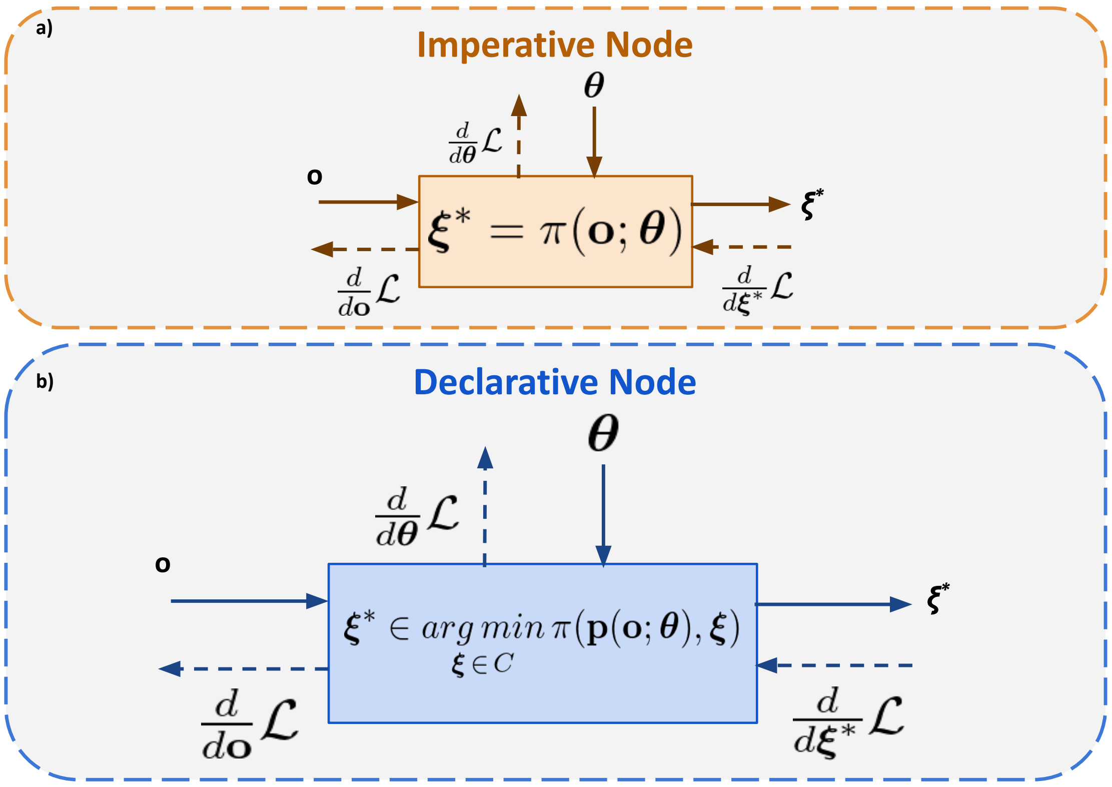
a)
Imperative Node
$\boldsymbol{\theta}$
$\frac{d}{d\boldsymbol{\theta}}\mathcal{L}$
o
$\boldsymbol{\xi}^* = \pi(\mathbf{o};\boldsymbol{\theta})$
$\boldsymbol{\xi}^*$
$\frac{d}{d\mathbf{o}}\mathcal{L}$
$\frac{d}{d\boldsymbol{\xi}^*}\mathcal{L}$
b)
Declarative Node
$\boldsymbol{\theta}$
$\frac{d}{d\boldsymbol{\theta}}\mathcal{L}$
o
$\boldsymbol{\xi}^* \in \underset{\boldsymbol{\xi} \in C}{arg\,min}\, \pi(\mathbf{p}(\mathbf{o};\boldsymbol{\theta}),\boldsymbol{\xi})$
$\boldsymbol{\xi}^*$
$\frac{d}{d\mathbf{o}}\mathcal{L}$
$\frac{d}{d\boldsymbol{\xi}^*}\mathcal{L}$

Figure 5. (a). During the forward pass of an imperative node, the input observation $\mathbf{o}$ is mapped to trajectory $\boldsymbol{\xi}^*$ by an explicit parameterized function π_θ. (b). During the forward pass of a declarative node, the trajectory $\boldsymbol{\xi}^*$ is computed implicitly as a solution of a parameterized objective function π_θ. Both imperative and declarative are parameterized data processing nodes in an end-to-end learnable model with loss function $\mathcal{L}$. During the backward pass for either node, the gradient of the loss function $\mathcal{L}$ with respect to the output trajectory $\boldsymbol{\xi}^*$, $\frac{d}{d\boldsymbol{\xi}^*}\mathcal{L}$ is backpropagated via the chain rule to produce gradients with respect to the parameters $\frac{d}{d\boldsymbol{\theta}}\mathcal{L}$. Backpropagation through the imperative node is easier because it follows the chain rule to produce gradients. In contrast, backpropagation through declarative nodes requires the use of an implicit function theorem to compute the gradients with respect to the parameters $\frac{d}{d\boldsymbol{\theta}}\mathcal{L}$ [1], [2].

3 Related Works

This chapter discusses prior works related to the proposed method. It highlights key differences between the existing approaches and the proposed method [1].

3.1 Trajectory Sampling Approaches

Existing works like [6], [16] can be viewed as a particular case of the Algorithm 1, presented in Chapter 4.1, obtained by performing only one iteration of the bi-level optimizer. These cited works sample the behavioural inputs **p** (lateral offsets, forward velocity, etc.), albeit not from a Gaussian distribution but a pre-specified grid. This is followed by the execution of lines 6-8 and the ranking the upper-level cost (line 13) associated with the generated trajectories.

However, [6], [16] do not have any mechanism to adapt the sampling distribution (or grid) to reduce the upper-level cost. Authors in [17] address this drawback to some extent as they adapt the sampling strategy based on optimal trajectories obtained in the past planning cycles. Such an adaptation strategy would be akin to performing one iteration of Algorithm 1 and then warm-starting the sampling distribution of **p** at the next planning cycle with the updated mean and variance obtained from line 14.

The lower-level planners of [6], [16], [17] ignore inequality constraints and essentially solve the QP presented on line 7 of Algorithm 1. The constraint residuals (e.g. obstacle clearance) are augmented into the cost function, similar to line 11. The proposed approach includes an additional projection operator at line 8 of Algorithm 1 to aid in constraint satisfaction.

3.2 RL Based Approaches

Works like [8], [9], [4], [5] can be viewed as training a function approximator to learn the solution space of the bi-level optimizer presented in 1. In Section 4.3, a similar attempt is made using a supervised setting. The RL approaches of [8], [9], [4], [5] would achieve this in a self-supervised setting based on just feedback of reward (upper-level cost) from the environment.

3.3 Bi-level Optimization

The bi-level approaches are extensively used in motion planning. For example, see [18], [19]. The proposed approach is closely related to the latter. In [19], an offline bi-level

[1] Reading the methodology behind the proposed method is recommended before going through this chapter.

optimization is used to generate optimal higher-level behaviours (parameter **p**) for drones. Subsequently, a neural network is trained to learn this solution space. At run-time, the neural network's output is the true solution. In sharp contrast, the output from the CVAE, trained in Section 4.3, is used as a guess for the optimal inputs **p** and adapted in real-time in Algorithm 1.

3.4 Comparison with Gradient Descent

Bi-level optimizations are commonly solved through Gradient Descent [18]. It requires computing the Jacobian of the optimal solution $\boldsymbol{\xi}^*$ with respect to parameter **p** through implicit differentiation [20]. The main drawback of this approach is that implicit differentiation has technical difficulties in case there are multiple local minima and when the lower-level problem is infeasible. In contrast, the Algorithm 1 does not require the lower-level optimization to be feasible and allows for its early termination. In either case, the constraint residuals can measure the quality of the optimal trajectory; thus, it can be augmented into the upper-level cost on line 11.

4 Methodology

This chapter presents the main idea behind the proposed novel bi-level optimizer for joint behaviour and trajectory planning. It provides a detailed overview of the behavioural cloning framework to warm-start the bi-level optimizer.

4.1 Proposed Bi-Level Optimization

The combined behaviour and trajectory planning is formulated through the following bi-level optimization problem:

$$\min_{\mathbf{p}} c_u(\boldsymbol{\xi}^*(\mathbf{p})), \tag{14a}$$

$$\boldsymbol{\xi}^* \in \arg\min_{\boldsymbol{\xi}} \frac{1}{2}\boldsymbol{\xi}^T\mathbf{Q}\boldsymbol{\xi} + \mathbf{q}^T(\mathbf{p})\boldsymbol{\xi}, \tag{14b}$$

$$\mathbf{A}_{eq}\boldsymbol{\xi} = \mathbf{b}(\mathbf{p}), \qquad \mathbf{g}(\boldsymbol{\xi}) \leq \mathbf{0} \tag{14c}$$

where (14b) - (14c) is the matrix representation of (3a) - (3d) obtained using (1). The behavioural inputs $\mathbf{p}$ are defined in (2). Thus, a part comprising lateral offsets and forward velocity set points enters the cost while the goal positions enter the affine equality constraints. The variable of the lower-level problem is $\boldsymbol{\xi} = (\mathbf{c}_x, \mathbf{c}_y)$.

As shown, we have an upper-level cost $c_u(.)$ that models the driving task. It depends on the optimal solution $\boldsymbol{\xi}^*$ computed from the lower-level trajectory optimization. The lower-level optimization explicitly depends on the parameter $\mathbf{p}$ while the upper-level has an implicit dependency through $\boldsymbol{\xi}^*(\mathbf{p})$. The lower-level optimizer aims to compute an optimal solution for a given $\mathbf{p}$. The upper level, in turn, aims to modify the parameter itself to drive down the upper-level cost associated with the optimal solution.

4.2 Batch Optimization and Sampling-based Gradient for Bi-level Optimization

The main idea is to apply a gradient-free optimization technique on the upper-level cost. Algorithm 1 summarizes the main steps of the proposed bi-level optimizer, wherein the left superscript l is used to track variables across iterations. For example, ${}^l\boldsymbol{\mu}_p$ represents the mean of the sampling distribution at iteration l. On line 4, $\overline{n}$ samples of $\mathbf{p}_j$ are drawn from a Gaussian distribution with mean ${}^l\boldsymbol{\mu}_p$ and covariance ${}^l\Sigma_p$. On line 6, the lower-level trajectory optimization is solved for each sampled parameter. The two-step approach resulting in a sample output is shown in Figure 3(c).

The first step involves solving the trajectory optimization without the inequality constraints. The second step projects the obtained solution to the constrained set. On line 9, the constraint residuals, resulting from the optimal solutions, are computed. In line 10, top n samples with the least constraint residuals are selected to create the $ConstraintEliteSet$. Line 11 constructs an augmented cost obtained by evaluating the upper-level cost $c_u(\boldsymbol{\xi}_j^*)$ on the samples from the $ConstraintEliteSet$ and adding the corresponding constraint residuals to it. On line 13, the top q samples with the least augmented cost are selected to construct the $EliteSet$. On line 14, the mean and variance of the sampling distribution of $\mathbf{p}$ is updated based on the samples of the $EliteSet$.

4.2.1 Updating the Sampling Distribution

There are several ways to update the mean and variance on line 14 of Algorithm 1. The simplest among these is to just fit a Gaussian distribution to the samples of $\boldsymbol{\xi}_j^*$ belonging to the $EliteSet$. However, this approach ignores the exact cost associated with the samples. Thus, this work uses the following update rule from sampling-based optimization proposed in [21]:

$$^{l+1}\boldsymbol{\mu}_p = (1-\eta)^l\boldsymbol{\mu}_p + \eta\frac{\sum_{j=1}^{j=q} s_j\mathbf{p}_j}{\sum_{j=1}^{j=q} s_j}, \tag{15a}$$

$$^{l+1}\boldsymbol{\Sigma}_p = (1-\eta)^l\boldsymbol{\Sigma}_p + \eta\frac{\sum_{j=1}^{j=q} s_j(\mathbf{p}_j - {}^{l+1}\boldsymbol{\mu}_p)(\mathbf{p}_j - {}^{l+1}\boldsymbol{\mu}_p)^T}{\sum_{j=1}^{j=q} s_j} \tag{15b}$$

$$s_j = \exp\frac{-1}{\gamma}(c_u(\boldsymbol{\xi}_j^*) + r_j(\boldsymbol{\xi}_j^*)) \tag{15c}$$

where η is the learning-rate and γ is some scaling constant. As discussed in [21], the update rules (15a) and (15b) are obtained by exponentiating the cost and then performing a sample estimate of its gradient with respect to the sampled argument (in this case $\mathbf{p}$).

4.2.2 Computational Tractability of Algorithm 1

The main computational bottleneck of Algorithm 1 stems from the requirement of solving a large number ($\overline{n} \approx 1000$) of non-convex optimizations on line 6 - 8. However, each optimization is decoupled from the other; thus, they can be solved in parallel (a.k.a., the batch setting) to ensure computational tractability. To this end, first, consider the QP presented on line 7. Solving it for the j^{th} sample of $\mathbf{p}_j$ reduces to following linear equations:

$$\begin{bmatrix}\mathbf{Q} & \mathbf{A}_{eq}^T\\ \mathbf{A}_{eq} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{\xi}_j\\ \boldsymbol{\nu}_j\end{bmatrix} = \begin{bmatrix}\mathbf{q}(\mathbf{p}_j)\\ \mathbf{b}(\mathbf{p}_j)\end{bmatrix}, \tag{16}$$

where, $\boldsymbol{\nu}_j$ is the dual variable associated with the equality constraints. The left-hand side of (16) is constant and independent of the $\mathbf{p}_j$. Thus, the solution for the entire batch can be constructed in one shot in the following manner:

Algorithm 1: Bi-Level Optimization

```
1  $N$ = Maximum number of iterations
2  Initiate mean ${}^{l}\boldsymbol{\mu}_p, {}^{l}\boldsymbol{\Sigma}_p$, at $l = 0$
3  for $l = 1, l \leq N, l++$ do
4      Draw $\overline{n}$ Samples $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_j, ...., \mathbf{p}_{\overline{n}})$ from $\mathcal{N}({}^{l}\boldsymbol{\mu}_p, {}^{l}\boldsymbol{\Sigma}_p)$
5      Initialize $CostList$ = []
6      Solve the lower-level trajectory optimization $\forall \mathbf{p}_j$:
7        • Step 1: Solve the QP without inequalities
             $$\overline{\boldsymbol{\xi}}_j = \arg\min_{\boldsymbol{\xi}_j} \frac{1}{2}\boldsymbol{\xi}_j^T \mathbf{Q}\boldsymbol{\xi}_j + \mathbf{q}^T(\mathbf{p}_j)\boldsymbol{\xi}_j$$
             $$\mathbf{A}_{eq}\boldsymbol{\xi}_j = \mathbf{b}(\mathbf{p}_j)$$
8        • Step 2: Project to Constrained Set
             $$\boldsymbol{\xi}_j^* = \arg\min_{\boldsymbol{\xi}_j} \frac{1}{2}\|\boldsymbol{\xi}_j - \overline{\boldsymbol{\xi}}_j\|_2^2$$
             $$\mathbf{A}_{eq}\boldsymbol{\xi}_j = \mathbf{b}(\mathbf{p}_j), \qquad \mathbf{g}(\boldsymbol{\xi}_j) \leq \mathbf{0}$$
9      Define constraint residuals: $r_j(\boldsymbol{\xi}_j^*) = \|\max(\mathbf{0}, \mathbf{g}(\boldsymbol{\xi}_j^*))\|_1$.
10     $ConstraintEliteSet \leftarrow$ Select top $n$ samples of $\mathbf{p}_j, \boldsymbol{\xi}_j^*$ with lowest
         constraint residuals.
11     $cost \leftarrow c_u(\boldsymbol{\xi}_j^*) + r_j(\boldsymbol{\xi}_j^*)$, over $ConstraintEliteSet$
12     append $cost$ to $CostList$
13     $EliteSet \leftarrow$ Select top $q$ samples of $(\mathbf{p}_j, \boldsymbol{\xi}_j^*)$ with lowest cost from
         $CostList$.
14     $({}^{l+1}\boldsymbol{\mu}_p, {}^{l+1}\boldsymbol{\Sigma}_p) \leftarrow$ Update distribution based on $EliteSet$
15 end
16 return *Parameter $\mathbf{p}_j$ and $\boldsymbol{\xi}_j^*$ corresponding to lowest $c_u(\boldsymbol{\xi}_j^*) + r_j(\boldsymbol{\xi}_j^*)$ in the EliteSet*
```

$$\begin{bmatrix} \boldsymbol{\xi}_1 & | & ... & | & \boldsymbol{\xi}_{\overline{n}} \\ \boldsymbol{\mu}_1 & | & ... & | & \boldsymbol{\mu}_{\overline{n}} \end{bmatrix} = (\overbrace{\begin{bmatrix} \mathbf{Q} & \mathbf{A}_{eq}^T \\ \mathbf{A}_{eq} & \mathbf{0} \end{bmatrix}^{-1}}^{constant}) \begin{bmatrix} \mathbf{q}(\mathbf{p}_1) & | & ... & | & \mathbf{q}(\mathbf{p}_{\overline{n}}) \\ \mathbf{b}(\mathbf{p}_1) & | & ... & | & \mathbf{b}(\mathbf{p}_{\overline{n}}) \end{bmatrix}, \tag{17}$$

The inverse on the right-hand side of (17) needs to be computed only once, irrespective of the batch size. Thus the batch solution of QP reduces to just a large matrix-vector product that can be trivially accelerated over GPUs.

The Algorithm 1 is build on the recent work [22] to handle the projection operation on line 8. This recent work showed that the core numerical algebra associated with projecting sampled trajectories on the collision avoidance constraints and motion bounds has the same batch QP structure as (16) and (17). The proposed Algorithm 1 uses an extension of [22] that includes curvature, centripetal acceleration bounds, and lane boundary constraints while retaining its efficient batch projection update rule [23].

4.3 Learning a Good Initialization Distribution

This section presents a Behaviour Cloning (BC) framework to learn a neural-network policy that maps observations $\mathbf{o}$ to optimal behavioural inputs $\mathbf{p}$. We assume access to a dataset $\mathbf{o}, \boldsymbol{\tau}_e$ that demonstrates the expert (optimal) trajectory $\boldsymbol{\tau}_e$ for each observation vector $\mathbf{o}$. The core challenge is that demonstrations of optimal $\mathbf{p}$ are available indirectly through the expert trajectory $\boldsymbol{\tau}_e$.

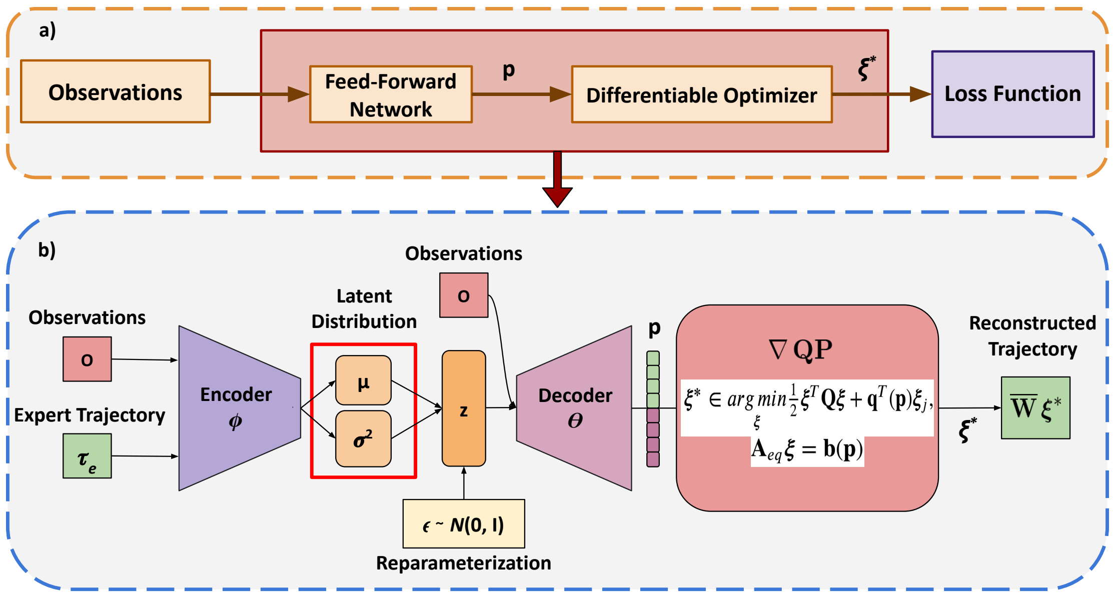

Figure 6. (a) The proposed BC framework combines conventional feedforward and differentiable optimization layers. (b) To learn a good initialization distribution, a CVAE is trained to approximate the optimal distribution of $\mathbf{p}$ that results in prediction $\boldsymbol{\xi}^*$ as close as possible to expert demonstrations $\boldsymbol{\tau}_e$.

The problem of indirect observation is addressed by using a network architecture combining conventional feedforward (imperative node) and differentiable optimization layers (declarative node) [20]. Figure 6(a) presents an overview of the main concept. The learnable weights are only present in the feedforward layer. It takes in observations $\mathbf{o}$ to output the behavioural inputs $\mathbf{p}$, which is then fed to an optimizer resulting in an optimal trajectory $\boldsymbol{\xi}^*$. The BC loss is computed over $\boldsymbol{\xi}^*$. The backpropagation required for updating the weights of the feedforward layer needs to trace the gradient of the loss function through the optimization layer.

4.3.1 Need for CVAE

The learned policy needs to induce a distribution over $\mathbf{p}$, such that for each observation $\mathbf{o}$, a sample is drawn from the learned distribution to initialize the bi-level optimizer presented in Algorithm 1 (line 4). With this motivation, a probabilistic generative model called Conditional Variational Autoencoder (CVAE) [24], [25], illustrated in Figure 6(b), is used as the learning framework. It consists of an encoder and decoder architecture constructed from a Multi-layer Perceptron (MLP) with weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ respectively. Additionally, the decoder network has an optimization layer that takes the output $\mathbf{p}$ of its MLP to produce an estimate of an optimal trajectory $\boldsymbol{\xi}^*$.

Let $\mathbf{z}$ be a latent variable such that the $p_\theta(\mathbf{z})$ represents the prior isotropic normal distribution $\mathcal{N}(0, \mathbf{I})$. The decoder network maps this distribution to $p_\theta(\boldsymbol{\xi}^*|\mathbf{z}, \mathbf{o})$. The encoder network on the other hand maps $(\mathbf{o}, \boldsymbol{\tau}_e)$ to a distribution $q_\phi(\mathbf{z}|\mathbf{o}, \boldsymbol{\tau}_e)$ over $\mathbf{z}$. In the offline phase, both the networks are trained end-to-end with loss function (18). The first term is the reconstruction loss responsible for bringing the output of the decoder network as close as possible to the expert trajectory. The second term in (18) acts as a regularizer that aims to make the learned latent distribution $q_\phi(\mathbf{z}|\mathbf{o}, \boldsymbol{\tau}_e)$ as close as possible to the prior normal distribution. The $\boldsymbol{\beta}$ hyperparameter acts as a trade-off between the two cost terms.

$$L_{CVAE} = \min_{\boldsymbol{\theta}, \boldsymbol{\phi}} \sum \|\overline{\mathbf{W}}\boldsymbol{\xi}^*(\boldsymbol{\theta}, \boldsymbol{\phi}) - \boldsymbol{\tau}_e\|_2^2 + \beta\, D_{\mathbf{KL}}[q_\phi(\mathbf{z}\,|\,\mathbf{o}, \boldsymbol{\tau}_e)\,|\,p_\theta(\mathbf{z})] \tag{18}$$

where $\overline{\mathbf{W}}$ is a diagonal matrix formed with $\mathbf{W}$. In the inferencing (online) phase, samples of $\mathbf{z}$ are drawn from the prior isotropic normal distribution and then passed through the decoder MLP, conditioned on observations $\mathbf{o}$, to get samples of optimal behavioural inputs $\mathbf{p}$. Finally, the input samples are passed through the optimizer to generate distribution for the optimal trajectory $\boldsymbol{\xi}^*$.

4.3.2 Choice of Differentiable Optimizer

Ideally, the entire lower-level trajectory optimizer (14b) - (14c) should be embedded into the CVAE decoder architecture. However, backpropagating through such non-convex optimization is fraught with technical difficulties. The recent successes of learning with optimization layers have come while embedding convex optimizers into neural networks [20]. Thus, by adopting a simplification as shown in Figure 6(b), the optimiztion layer is constructed with convex cost (14b) and affine equality constraints in (14c), both of which depend explicitly on $\mathbf{p}$.

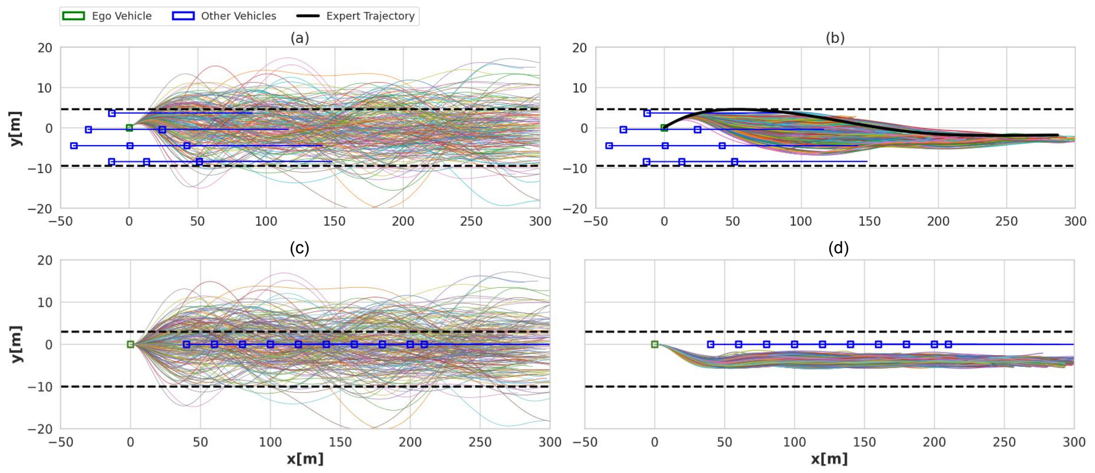

Figure 7. (a),(c) shows the trajectory distribution resulting from sampling behavioural inputs **p** from a Gaussian distribution. (b) shows the corresponding distribution when sampling **p** from a learned CVAE. As can be seen, CVAE results in a more structured and smoother trajectory distribution concentrated around the expert demonstration, shown in black. The trajectory samples also conform to lane boundaries shown in dotted black lines. The blue rectangle represents the neighbouring vehicles moving along straight-line (blue) trajectories. The green rectangle represents the ego vehicle. (d) shows CVAE generalizing to a new traffic scenario where the other vehicles drive in a straight line at slower speeds. It results in a trajectory distribution conforming to the lane boundaries and towards the free lane.

The inequality constraints are ignored since the behavioural inputs **p** are not depended on it. The intuition is that the expert trajectory will be collision-free and kinematically feasible, therefore automatically satisfying the inequality constraints. Thus, the main goal is to figure out the right set of **p** to mimic the expert behaviour as closely as possible.

4.3.3 Example of a CVAE Output

Figure 7 contrasts initialization of Algorithm 1 from a naive Gaussian and the learned CVAE distribution. For Figure 7(a),(c), the **p** are sampled from a Gaussian distribution to solve the QP presented in line 7 of Algorithm 1, resulting in a distribution over trajectories $\boldsymbol{\xi}^*$. The same process applies to drawing **p** from the CVAE to solve the QP, resulting in a distribution over trajectories $\boldsymbol{\xi}^*$ in Figure 7(b),(d). Evidently, the distribution resulting from **p** drawn from CVAE is smoother than Gaussian sampling, conforms to lane boundaries, is concentrated around the expert demonstration trajectory and generalizes to new traffic scenarios.

5 Results

This chapter describes the implementation details of the proposed bi-level optimizer and CVAE model. It introduces the baselines, benchmarking scenarios and presents the findings of the experiments.

5.1 Implementation Details

The Algorithm 1, including the lower-level optimizer, is implemented in Python using JAX [26] library as the GPU-accelerated linear algebra backend. The matrix $\mathbf{W}$ in (1) is constructed from a 10^{th} order polynomial. The simulation pipeline was built on top of the Highway Environment (highway-env) simulator [27]. The neighbouring vehicles used Intelligent Driver Model (IDM) [28] for longitudinal and MOBIL [29] for lateral control.

5.1.1 Hyper-parameter Selection

The sampling size $\overline{n}$ in Algorithm 1 was 1000. The $ConstraintEliteSet$ (line 10, Algorithm 1) and $EliteSet$ (line 13, Algorithm 1) had 150 and 50 samples respectively. The behavioural input $\mathbf{p}$ was modelled as four set points for lateral offsets and desired longitudinal velocities for the bi-level optimizer. That is, $\mathbf{p} = \begin{bmatrix} y_{d,1}, \dots, y_{d,4}, v_{d,1}, \dots, v_{d,4} \end{bmatrix}$. The planning horizon is divided into four segments, and one pair of lateral offset and desired velocity is associated with each segment. The Algorithm 1 computes the optimal $\mathbf{p}$ along with the associated trajectory. γ value in (15c) is 0.9.

5.1.2 CVAE Training

The details of the encoder and decoder network architecture of our CVAE are presented in Table 2. During training, the input to the CVAE is the expert trajectory and a 55-dimensional observation vector $\mathbf{o}$ containing the state of the ego-vehicle, the ten closest obstacles and the road boundary. For the ego-vehicle, the state consists of a heading, lateral and longitudinal speeds. The state consists of longitudinal and lateral positions and the corresponding velocities for the ten closest obstacles. The position-level information is expressed with respect to the centre of the ego vehicle.

During inferencing, the decoder network only needs $\mathbf{o}$ and samples of $\mathbf{z}$ drawn from the prior isotropic Gaussian. The expert demonstration of optimal trajectories for CVAE training is collected offline using the cross-entropy method with a batch size of 1000. We note that our demonstrations could be sub-optimal. However, even with such a simple data set, the CVAE can learn useful initialization for the proposed bi-level optimizer presented in Algorithm 1 (see Figure 7).

Encoder Network
ϕ

Block	*Layers*	*Output Size*	*Activation*
MLP 1-4	Linear, Batchnorm	1024	ReLU
MLP 5	Linear, Batchnorm	256	ReLU
Mean	Linear	2	None
Variance	Linear	2	Softplus

Decoder Network
θ

Block	*Layers*	*Output Size*	*Activation*
MLP 1-4	Linear, Batchnorm	1024	ReLU
MLP 5	Linear, Batchnorm	256	ReLU
$\mathbf{p}$	Linear	8	None
$\boldsymbol{\xi}^*$	Optimization Layer	22	None

Table 2. CVAE architecture- composed of Encoder and Decoder Networks as explained in Figure 6(b).

The optimizer used for training the CVAE was AdamW [30] with a learning rate of 1e-4 and weight decay of 6e-5 for a total of 80 epochs. Moreover, the learning rate was decayed by $\gamma = 0.1$ every ten epochs. The posterior collapse or the KL vanishing problem is tackled by applying a monotonic annealing schedule of $\boldsymbol{\beta}$ coefficient [31], starting with 0 and gradually annealing the $\boldsymbol{\beta}$ at each optimizer step in loss function 18.

5.1.3 MPC Baselines

Algorithm 1 is used in a receding horizon manner to create an MPC variant of the bi-level optimizer- referred to as MPC-Bi-Level. It takes the same observation vector $\mathbf{o}$ as the CVAE and outputs polynomial coefficients of the optimal trajectories. These are converted to steering and acceleration input vectors, and the ego-vehicle executes the first five elements of these in the open loop before initiating the next replanning. The MPC-Bi-Level is compared with the following baselines:

- MPC-Vanilla: This baseline runs without a behavioural layer. It only includes the lower-level optimization (14b) - (14c). The input $\mathbf{p}$ is a scalar representing the set point for the desired longitudinal velocity.
- MPC-Grid: This baseline uses the same behavioural inputs $\mathbf{p}$ as the MPC-Bi-Level but samples them from a pre-specified fixed grid.

- MPC-Random: This baseline is similar to MPC-Grid but samples **p** from a Gaussian distribution.

- Batch-MPC of [32]: This baseline is similar to MPC-Grid but uses a different set of behavioural inputs, namely goal positions for the longitudinal and lateral components of the trajectory. That is, **p** $= \begin{bmatrix} x_f, y_f \end{bmatrix}$.

5.1.4 RL Baselines

MPC-Bi-Level is also compared against Deep Q-Network (DQN) and Proximal Policy Optimization (PPO)-discrete, developed using the framework outlined in [33] and [34] respectively. The input observation is the same as the MPC-Bi-Level. The action space is discrete with five different behaviours, namely *faster*, *slower*, *left-lane change*, *idle*, *right-lane change*.

These behaviours are then mapped to a set point for lateral offset or longitudinal velocity and tracked through a PID controller using appropriate steering and acceleration commands. DQN and PPO have been trained on the highway-env simulator using Stable-Baselines 3 [35]. The policy estimators employed in RL baselines are Multi-Layer Perceptrons (MLPs) with two hidden layers of 256 neurons each. The action space is discrete with the following different behaviours:

- *faster*: Increase velocity by 5 m/s

- *slower*: Decrease velocity by 5 m/s

- *left-lane change*: change lateral position by 4 m to the left of the current position.

- *idle*: Keep moving with the current velocity and lateral offset set-points.

- *right-lane change*: change lateral position by 4 m to the right of the current position.

Table 3 summarises the complete set of RL training hyper-parameters.

5.1.5 Environments, Tasks, and Metrics

The highway driving scenarios are presented in Figure 8. Each scenario has two different traffic densities except for the two-way environment. We use the internal parameter of highway-env named "density" to control how closely each vehicle is placed at the start of the simulation. The RL baselines did not perform well in very dense environments and thus were tested in sparser environments than the MPC-based approaches. In each scenario, we evaluated 50 episodes with different randomly initialized traffic. The

Table 3. RL baselines training hyper-parameters

Agent	Parameter	Value
	Number of training steps	5M
	Policy Scheduling Time	1s
	Input neurons	55
	Hidden layers	2
	Hidden layers neurons	256
	Output neurons	5
	Discount factor	0.8
	Learning rate	5e-4
DQN	Replay Memory size	15k
	Initial exploration constant	1
	Final exploration constant	0.1
	Target Network update frequency	50
	Batch size	32
PPO	number of steps	10
	Batch size	64
	Generalized Advantage Estimation(GAE) λ	0.95
	clipping coefficient	0.2
	value-function coefficient	0.5

random seed of the simulator was fixed to ensure that all RL and MPC baselines were benchmarked across the same set of traffic configurations.

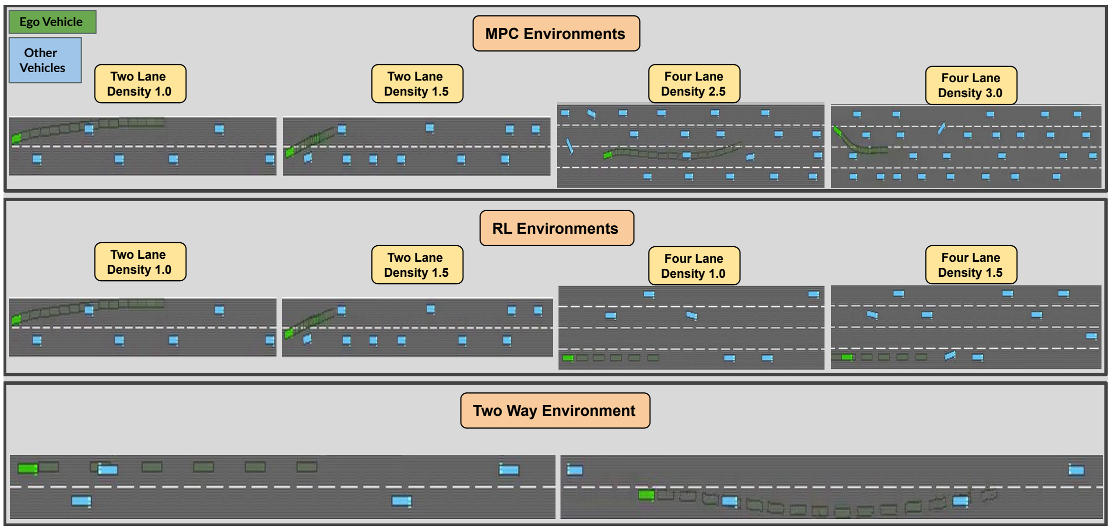

Figure 8. Two / four-lane and two-way driving scenarios with varying traffic density for benchmarking proposed approach with MPC and RL baselines.

The task in the experiment was for the ego vehicle to drive as fast as possible without colliding with the obstacles and going outside the lane boundary. Thus, the upper-level cost of the bi-level optimizer has the form $(\sqrt{(\dot{x}^*(t))^2 + (\dot{y}^*(t))^2} - v_{max})^2$, where $(\dot{x}^*(t), \dot{y}^*(t))$ are the optimal velocity profiles obtained from the lower-level optimization. Note that these are obtained from $\boldsymbol{\xi}^*$ through relationship (1). The constraints of the lower-level optimization handle safety. The evaluation metric has two components: (i) collision rate and (ii) average velocity achieved within an episode. Since the ego vehicle can achieve arbitrary high velocity while driving rashly, only velocities from collision-free episodes are considered.

5.2 Empirical Validation of Convergence

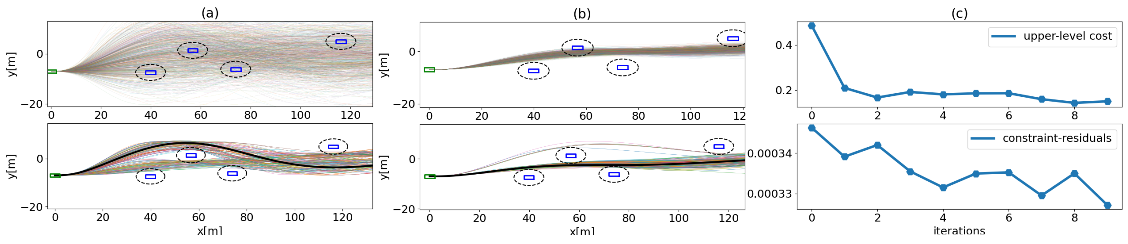

Figure 9. Empirical validation of Algorithm 1. (a), (b) The top figures shows the trajectory distribution resulting from sampling behavioural inputs **p** from a Gaussian distribution and solving the QP at line 7 of Algorithm 1. The bottom plots in both figures show the modified distribution after projection onto the constrained set. Both (a) and (b) correspond to the distribution at the first and the fifth iteration, respectively. (c). As the iterations progress, the variance of the trajectory distribution shrinks, and the upper-level cost in the top figure saturates, indicating convergence of the bi-level optimizer. Bottom figure shows the constraint residual of the lower-level optimization at each iteration.

Figure 9 shows the performance of the bi-level optimizer presented in Algorithm 1 on a typical environment with static obstacles. The top plot of Figure 9(a),(b) shows the trajectory distributions at the first and the fifth iteration, respectively. Here, the behavioural inputs **p** are sampled from a Gaussian distribution to solve the QP presented in line 7 of Algorithm 1. The bottom plots in these figures show how the distribution changes when we project them onto the feasible set of collision avoidance and kinematic constraints. The following key observations can be made from Figure 9:

- The projection operation results in trajectories residing in different homotopies, proving crucial for proper state space exploration.
- The variance of the trajectory distribution shrinks, and the upper-level cost, Figure 9(c) (top), saturates. This is a typical convergent behaviour observed in sampling-based optimizers such as Algorithm 1. Please note that the shrinking of trajectory

variance in Figure 9(a),(b) (top) also signifies that the sampling distribution for **p** has also converged to an optimal one.

- Finally, Figure 9(c) (bottom) validates the role of the projection operator.

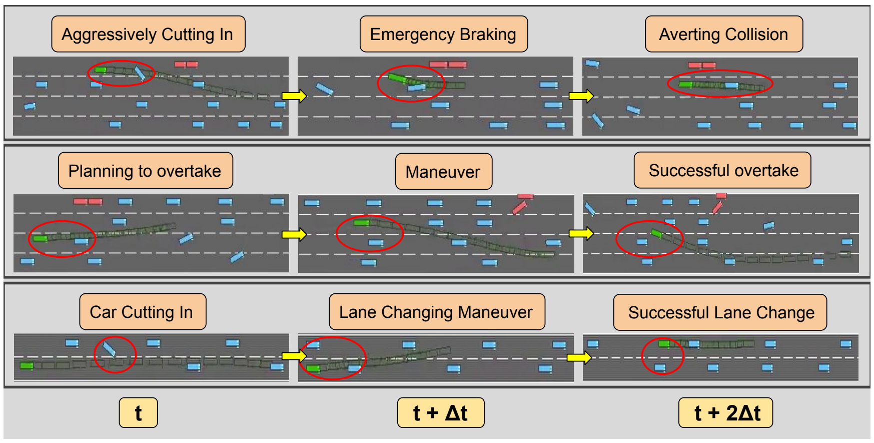

Figure 10. The proposed bi-level optimizer ensures safe driving in dense and potentially rash traffic scenarios. The top figure shows a scenario where a neighbouring vehicle (blue) suddenly cuts in front of the ego-vehicle (green), requiring emergency braking. The middle figure shows a successful overtaking in dense traffic. The bottom figure is similar to the one on the top but in a two-lane setting.

5.3 Importance of Bi-Level Adaptation

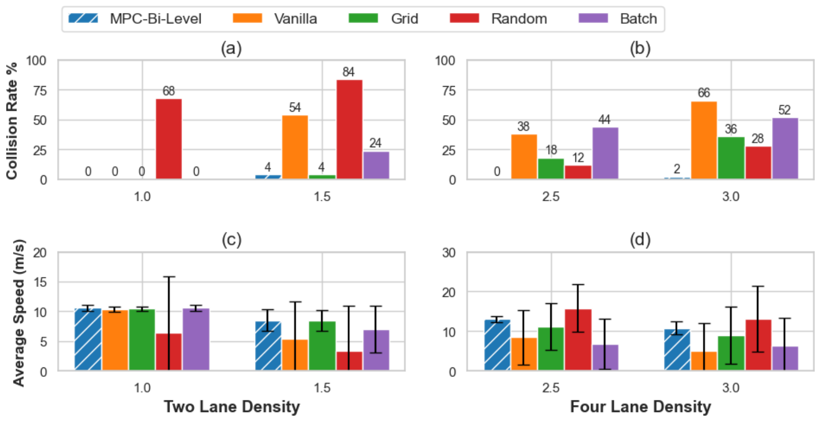

Figure 11. Comparison of MPC Bi-Level with other MPC-based baselines in two (a, c) and four-lane (b, d) driving scenarios.

In a two-lane driving scenario, there is only a limited set of manoeuvres that the ego-vehicle can do. Thus, on a low traffic density, all the baselines except MPC-Random achieve perfect collision rate (Figure 11(a)). This shows that having a dedicated behavioural layer on simple driving scenarios is not critical. The observation is unsurprising as existing results like [36], [37] have shown promising results without explicitly incorporating behavioural inputs, but in sparse environments.

Figure 11(a) also shows that in a sparse two-lane environment, a simple grid-search used by MPC-Grid and Batch-MPC [32] is enough to come up with the right set of behavioural inputs. However, as traffic density increases in the two-lane setting, the safety improvement provided by the MPC-Bi-Level becomes distinctly apparent (see Figure 10). The trend is particularly stark in highly dense four-lane driving scenarios, where the proposed approach provides a 4 - 10x reduction in collision rate over other baselines. Figure 11(b),(d) shows that the average speed achieved by the MPC-Bi-Level is generally either better or competitive with all the baselines.

5.4 Safety Improvements over RL

Despite training DQN and PPO-discrete for over 5 million steps, it could reasonably work in sparse two-lane / two-way traffic scenarios. Nevertheless, the collision rate and velocity (Figure 12(a, c)) achieved by DQN and PPO-discrete were still drastically worse than the MPC-Bi-Level. The performance gap increased further in a dense four-lane setting, as shown in Figure 12(b, d).

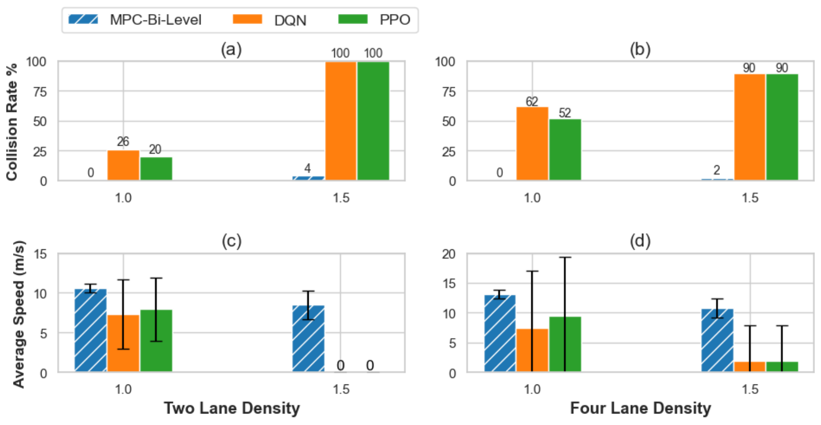

Figure 12. Comparison of MPC Bi-Level with other RL-based baselines in two (a, c) and four-lane (b, d) driving scenarios.

DQN and PPO fail to work reasonably in a relatively straightforward two-way highway driving scenario (see Figure 8). MPC-based approaches, including the MPC Bi-Level, achieved zero collision rate and higher velocity than RL baselines, as shown in Figure 13(a, b).

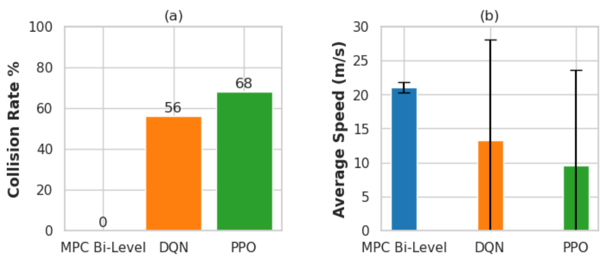

Figure 13. Comparison of MPC Bi-Level with other RL-based baselines in two-way (a, b) driving scenario.

5.5 Effect of CVAE Initialization

Figure 14(a) shows that for a relatively small batch size of 250, the learned CVAE initializer achieves a 4x reduction in collision rate over the naive Gaussian distribution. However, the performance gap between both initializations reduces a bit as the batch size is increased (Figure 14(b)). Thus, both Figure 14(a),(b) validate that a learned CVAE initializer can be particularly helpful when Algorithm 1 is run with a limited computation budget or on resource-constrained hardware, wherein we have to contend with a smaller batch size for Algorithm 1.

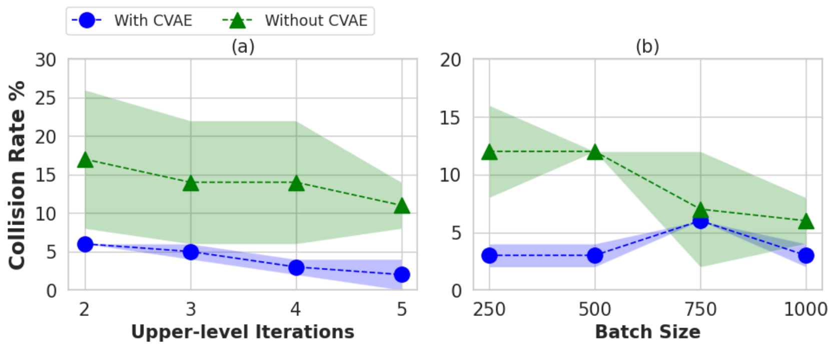

Figure 14. The computational advantage achieved by initializing Algorithm 1 through samples drawn from learned CVAE over baseline Gaussian distribution. CVAE allows for achieving a better collision rate with smaller batch sizes.

5.6 Computation Time

Figure 15 shows the computation time required for the Algorithm 1 on a laptop with RTX 3080 GPU. In sparse traffic scenarios, two iterations of Algorithm 1 proved enough to achieve a low collision rate when initialized with the learned CVAE. A batch size of 250 corresponds to a feedback rate of around 100 Hz. All the benchmarking presented in Section 5.3, 5.4 were obtained with the same batch size but used five iterations of Algorithm 1, totalling $0.03s$. Figure 15(b) demonstrates a moderate increase in the computation time with respect to batch size; even for a batch size of 1000, the computation time was less than $0.06s$. Finally, both Fig.15(a), (b) show that the additional overhead of inferencing the learned CVAE is very minimal.

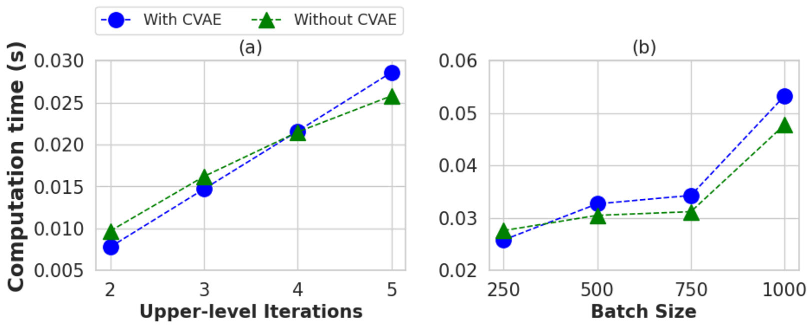

Figure 15. Samples drawn from learnt CVAE distribution leads to better collision rate in lesser iterations.

6 Conclusion and Future Work

This thesis proposes a novel bi-level optimization for combined behaviour and trajectory planning. It can simultaneously search for the optimal higher-level behavioural decisions and the lower-level trajectories necessary for executing them. The custom optimizer combines features from gradient-free and sampling-based optimization with QP. It runs in real-time due to an efficient GPU parallelization of the lower-level optimization.

This thesis also presents a CVAE architecture constructed from a feedforward neural network and differentiable optimization layers to learn good initialization for the proposed bi-level optimizer. Extensive experiments exhibit the importance of having a dedicated behavioural layer, and the proposed approach outperforms MPC and RL baselines in dense traffic scenarios. Finally, the learned CVAE initialization improved the computational tractability of the proposed bi-level optimizer by reducing the batch size and the number of upper-level iterations required to achieve a given collision rate.

One of the significant limitations of behavioural cloning of behavioural inputs is that expert demonstrations can be sub-optimal or challenging to obtain in dense traffic situations. The future work involves formulating a behavioural input learning pipeline with a purely self-supervision loss, reducing the need for collecting expert demonstrations in challenging dense traffic situations. Moreover, the relative success of learning with straightforward differentiable optimization layers paves the way for embedding more complex optimizers that help learn optimal behavioural inputs such that it can guide the bi-level optimizer to work with the reduced batch size and no upper-level iterations in dense traffic scenarios.

7 Appendix

The video demos are presented in the project page:
`https://sites.google.com/view/mpc-bi-level`

The source code to reproduce the experiments is available at the following GitHub repository:
`https://github.com/jatan12/MPC-Bi-Level`

Acknowledgements

I would like to thank

Prof. Arun Kumar Singh, for his guidance and support throughout this thesis,

Nicola Albarella for helping with setting up the benchmarking scenarios,

members from the Collaborative Robotics Group for their continued support and encouragement.

Jatanstha

Figure 16. Written Signature

References

[1] B. Amos, *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, May 2019.

[2] S. Gould, R. Hartley, and D. Campbell, "Deep declarative networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 3988–4004, 2022.

[3] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," 2019.

[4] M. Huegle, G. Kalweit, B. Mirchevska, M. Werling, and J. Boedecker, "Dynamic input for deep reinforcement learning in autonomous driving," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7566–7573, IEEE, 2019.

[5] J. Li, L. Sun, J. Chen, M. Tomizuka, and W. Zhan, "A safe hierarchical planning framework for complex driving scenarios based on reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2660–2666, IEEE, 2021.

[6] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 458–464, IEEE, 2014.

[7] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hybrid trajectory planning for autonomous driving in on-road dynamic scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 341–355, 2019.

[8] C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE transactions on intelligent vehicles*, vol. 5, no. 2, pp. 294–305, 2019.

[9] S. Han and F. Miao, "Behavior planning for connected autonomous vehicles using feedback deep reinforcement learning," *arXiv preprint arXiv:2003.04371*, 2020.

[10] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenét frame," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010), Anchorage, Alaska, USA, 3 - 8 May 2010*, pp. 987 – 993, Institute of Electrical and Electronics Engineers (IEEE), 2010.

[11] E. Hazan, "Introduction to online convex optimization," 2021.

[12] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," 2020.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[14] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," 2019.

[15] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[16] 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*, pp. 987–993, IEEE, 2010.

[17] S. Sun, Z. Liu, H. Yin, and M. H. Ang, "Fiss: A trajectory planning framework using fast iterative search and sampling strategy for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9985–9992, 2022.

[18] W. Sun, G. Tang, and K. Hauser, "Fast uav trajectory optimization using bilevel optimization with analytical gradients," in *2020 American Control Conference (ACC)*, pp. 82–87, IEEE, 2020.

[19] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, 2022.

[20] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, pp. 136–145, PMLR, 2017.

[21] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Conference on Robot Learning*, pp. 750–759, PMLR, 2022.

[22] H. Masnavi, J. Shrestha, M. Mishra, P. Sujit, K. Kruusamäe, and A. K. Singh, "Visibility-aware navigation with batch projection augmented cross-entropy method over a learned occlusion cost," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9366–9373, 2022.

[23] A. K. Singh, J. Shrestha, and N. Albarella, "Bi-level optimization augmented with conditional variational autoencoder for autonomous driving in dense traffic," 2022.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational {Bayes}," in *Int. Conf. on Learning Representations*.

[25] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[26] 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.

[27] E. Leurent, "An Environment for Autonomous Driving Decision-Making," 5 2018.

[28] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[29] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model mobil for car-following models," *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, 2007.

[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[31] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, (Berlin, Germany), pp. 10–21, Association for Computational Linguistics, Aug. 2016.

[32] V. K. Adajania, A. Sharma, A. Gupta, H. Masnavi, K. 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*, vol. 7, no. 2, pp. 4220–4227, 2022.

[33] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2148–2155, IEEE, 2018.

[34] T. Liu, H. Wang, B. Lu, J. Li, and D. Cao, "Decision-making for autonomous vehicles on highway: Deep reinforcement learning with continuous action horizon," *arXiv preprint arXiv:2008.11852*, 2020.

[35] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[36] B. Gutjahr, L. Gröll, and M. Werling, "Lateral vehicle trajectory optimization using constrained linear time-varying mpc," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1586–1595, 2016.

[37] P. Lin and M. Tsukada, "Model predictive path-planning controller with potential function for emergency collision avoidance on highway driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4662–4669, 2022.