# PILOT: Efficient Planning by Imitation Learning and Optimisation for Safe Autonomous Driving

**Citation for published version:**
Pulver, H, Eiras, F, Carozza, L, Hawasly, M, Albrecht, SV & Ramamoorthy, S 2021, PILOT: Efficient Planning by Imitation Learning and Optimisation for Safe Autonomous Driving. in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Institute of Electrical and Electronics Engineers (IEEE), pp. 1442-1449, 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems , Prague, Czech Republic, 27/09/21. https://doi.org/10.1109/IROS51168.2021.9636862

**Digital Object Identifier (DOI):**
10.1109/IROS51168.2021.9636862

**Link:**
Link to publication record in Edinburgh Research Explorer

**Document Version:**
Peer reviewed version

**Published In:**
2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

# PILOT: Efficient Planning by Imitation Learning and Optimisation for Safe Autonomous Driving

Henry Pulver*, Francisco Eiras*†, Ludovico Carozza*, Majd Hawasly*,
Stefano V. Albrecht*‡ and Subramanian Ramamoorthy*‡
*Five AI Ltd., United Kingdom
Email: first.last@five.ai
†University of Oxford, United Kingdom
‡University of Edinburgh, United Kingdom

***Abstract*—Achieving a proper balance between planning quality, safety and efficiency is a major challenge for autonomous driving. Optimisation-based motion planners are capable of producing safe, smooth and comfortable plans, but often at the cost of runtime efficiency. On the other hand, naïvely deploying trajectories produced by efficient-to-run deep imitation learning approaches might risk compromising safety. In this paper, we present PILOT– a planning framework that comprises an imitation neural network followed by an efficient optimiser that actively rectifies the network's plan, guaranteeing fulfilment of safety and comfort requirements. The objective of the efficient optimiser is the same as the objective of an expensive-to-run optimisation-based planning system that the neural network is trained offline to imitate. This efficient optimiser provides a key layer of online protection from learning failures or deficiency in out-of-distribution situations that might compromise safety or comfort. Using a state-of-the-art, runtime-intensive optimisation-based method as the expert, we demonstrate in simulated autonomous driving experiments in CARLA that PILOT achieves a seven-fold reduction in runtime when compared to the expert it imitates without sacrificing planning quality.**

## I. Introduction

Guaranteeing safety of decision-making is a fundamental challenge on the path towards the long-anticipated adoption of autonomous vehicle (AV) technology. Attempts to address this challenge show the diversity of possible approaches to the concept of safety: whether it is maintaining the autonomous system inside a safe subset of possible future states [1], [2], preventing the system from breaking domain-specific constraints [3], [4], or exhibiting a behaviour that matches the safe behaviour of an expert [5], amongst others.

Approaches to motion planning in AVs can be categorised in different ways, e.g., *data-driven* vs. *model-based*. The hands-off aspect of purely *data-driven* approaches is lucrative, which is evidenced by the growing interest in the research community in exploiting techniques such as reinforcement or imitation learning applied to autonomous driving [6], [7], [8], [9]. Moreover, inference in a data-driven model is usually efficient when compared to more elaborate search- or optimisation-based approaches, which is a key requirement in real-time applications. However, this does not come for free as these systems struggle to justify their decision making or to certify the safety of their output at deployment time without major investments in robust training [10], [11] or post-hoc analysis [12]. This constitutes a major setback to the deployment of such methods in safety-critical contexts. On the other hand, *model-based* approaches tend to be engineering-heavy and require deep knowledge of the application domain, while giving a better handle on setting and understanding system expectations through model specification. Moreover, they produce more interpretable plans [13], [14], [15], [16]. This, however, usually comes at the cost of robustness [3] or runtime efficiency [4]. We aim to bring the efficiency benefits of data-driven methods together with the guarantees of model-based systems in a *hybrid* approach for urban driving applications. Our goal is

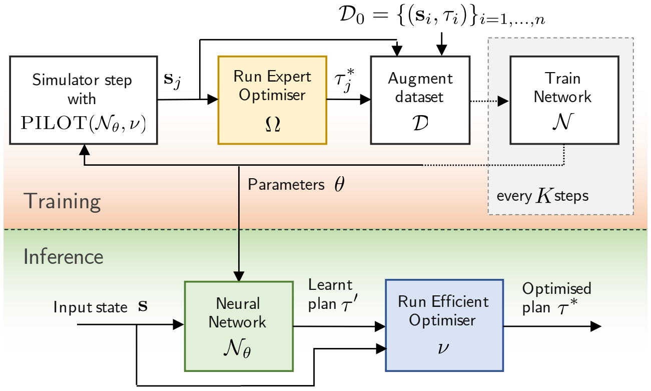
$\mathcal{D}_0 = \{(\mathbf{s}_i, \tau_i)\}_{i=1,\dots,n}$
Simulator step with PILOT($\mathcal{N}_\theta, \nu$)
$\mathbf{s}_j$
Run Expert Optimiser $\Omega$
$\tau_j^*$
Augment dataset $\mathcal{D}$
Train Network $\mathcal{N}$
Parameters $\theta$
every $K$ steps
Training
Inference
Input state $\mathbf{s}$
Neural Network $\mathcal{N}_\theta$
Learnt plan $\tau'$
Run Efficient Optimiser $\nu$
Optimised plan $\tau^*$

Fig. 1. PILOT *framework*: (top) PILOT uses an expert-in-the-loop imitation learning paradigm to train a deep neural network, $\mathcal{N}_\theta$, that imitates the output of an expensive-to-run optimisation-based planner $\Omega$. (bottom) At inference time, PILOT uses the output of $\mathcal{N}_\theta$ to initialise an efficient optimiser $\nu$ to compute a feasible and low cost trajectory.

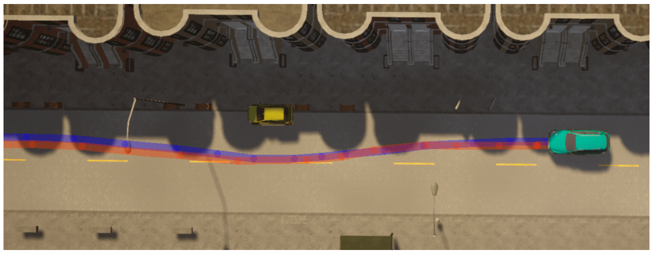

Fig. 2. An example CARLA scenario with trajectories generated with the planner-in-the-loop with a horizon of 8s per planning stage for 1) the expensive-to-run planner 2s-OPT (in red) that took 175 ms per planning stage on average, and 2) PILOT (in blue) that took 44 ms per planning stage on average. More examples in the accompanying video five.ai/pilot

to introduce a general planning architecture that is flexible enough to capture complex planning requirements, yet still guarantee the satisfaction of sophisticated specifications at deployment, without sacrificing runtime efficiency.

In this work, we propose an approach that combines model-based optimisation and deep imitation learning, using as the expert a performant optimisation-based planner that is expensive to run. We introduce **PILOT** – *Planning by Imitation Learning and Optimisation* – in Sec. III. At *training time* (Fig. 1, top), we *distil* [17] the expert planner's behaviour using imitation learning, with *online*, expert-in-the-loop dataset augmentation (e.g. DAgger – Dataset Aggregation [18]) to continually enrich the training dataset with relevant problems sampled from the state distribution induced by the learner's policy. At *inference time* (Fig. 1, bottom), to actively correct potential learning failures and improve safety, we employ an efficient optimisation component that optimises the same objective function as the expert but benefits from informed warm-starting provided by the network output.

In this paper, without loss of generality to our approach, we use the two-stage optimisation framework introduced by Eiras *et al.* [4] as the expensive-to-run planner to imitate in a simulated environment. As discussed by the authors, the framework in [4] suffers in terms of runtime, effectively trading off efficiency for better solution quality, which makes it a suitable choice for PILOT. We use the CARLA simulator [19] to validate our approach under a wide variety of conditions in realistic simulations. This is an important step in the direction of understanding the viability and safety of such methods before deploying on the roads. A qualitative example of PILOT's performance in CARLA is in Fig. 2.

The contributions of this work are:

- A robust and scalable framework that imitates an expensive-to-run optimiser, with expert-in-the-loop data augmentation at training time and active correction at inference time using an efficient optimiser.
- Applying this framework to the two-stage optimisation-based planner from [4] leading to a 7× runtime improvement in our benchmark CARLA datasets at no significant loss in solution quality (measured by the objective function cost of the output trajectory).

## II. Background and Related Work

In this section we review related work in motion planning for AVs regarding imitation learning with optimisation experts (Sec. II-A) and motion planning via optimisation (Sec. II-B). Then, we give an overview of a planning method we use in this work to demonstrate PILOT for (Sec. II-C).

### A. Imitation Learning with Optimisation Experts for AVs

With the complexity of specifying the objective function of safe, assertive driving, imitation learning offers a promising alternative. However, naïve attempts to leverage expert traces, e.g. with vanilla behavioural cloning [20], usually fails at deployment to exhibit safe behaviour in complex scenarios due to covariate shift between the training and deployment settings [21], [22]. To mitigate this issue, techniques for training data augmentation range from online methods that actively enrich the training dataset with actual experiences from the deployment environment [18], and offline synthesis of realistic scenarios for the expert to demonstrate recovery from perturbations [23] or near-misses [24].

Still, data augmentation by itself cannot guarantee the safety of decisions at inference time, which we believe is a fundamental requirement for any deployed system in the safety-critical autonomous driving task. Yet, most of the existing literature on imitation learning of optimisation [7], [25], [26], [27] propose pure end-to-end learning pipelines.

Pan *et al.* in [7] proposed an end-to-end system for off-road, fixed route, real-world planning that learns to map basic sensory input into controls with the guidance of a Model Predictive Control (MPC) expert that has access to better sensors and more compute. However, no safety guarantees are provided at deployment time beyond what a low-level controller employed to track the network output does. A related approach by Sun *et al.* in [26] employs a shallow neural network with selected state features as input to imitate an MPC expert that optimises progress and control effort in long-term, two-lane driving scenarios with two other vehicles. On top of that, an online, short-horizon MPC controller tracks the initial portion of the inferred trajectory, constrained by the same feasibility and collision constraints. For online augmentation, the optimisation problems in which the network output deviates away from the expert's are included in the augmented training dataset. Acerbo *et al.* in [27] pre-train a fully-connected neural network to map state features of a simple lane-keeping scenario involving no other vehicles into parameters of smooth, second-order polynomial curves, using a dataset generated by a short-horizon nonlinear MPC expert. In addition to the usual $L_2$ term, the training loss incorporates other terms related to collision avoidance, implemented with barrier functions. A related, supervised learning approach is Constrained Policy Nets [28] in which the loss of a policy network is derived directly from an optimisation objective. This, however, requires careful treatment of the constraint set to ensure differentiability.

Another approach to rectify an imitation network output employs control safe sets to validate acceleration and steering commands of an imitation network trajectory [23]. This, however, is limited to taming the predicted trajectory inside the safe set, but unable to suggest other viable corrections.

### B. Motion Planning as Optimisation for AVs

For any variable $v_j$ with $j \in \mathbb{Z}^*$, we will use the shorthand $v_{i:e} = \{v_i, ..., v_e\}$. In its most general form, motion planning via optimisation is defined as follows [16]: assume the input to the motion planning problem is given by a scene $\mathbf{s} \in \mathcal{S}$ (vehicle states/positional uncertainty, layout information, and predictions of other agents over a fixed horizon). The goal is to obtain a plan for the ego-vehicle (or *ego* for short) as a sequence of $N+1$ states, $\tau^* = \tau_{0:N} \in \mathcal{T}$, such that:

$$\begin{aligned} \tau^* = \underset{\tau}{\text{argmin}}\ & \mathcal{J}(\tau) \\ \text{s.t. } \tau_0 = \mathbf{s}_{\text{ego}}, \quad & \mathbf{g}_\mathbf{s}(\tau_{0:N}) \leq 0, \quad \mathbf{h}_\mathbf{s}(\tau_{0:N}) = 0 \end{aligned} \tag{1}$$

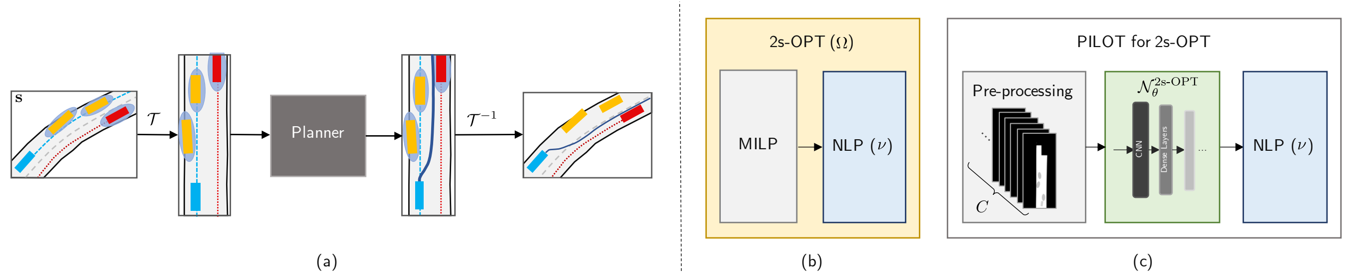

Fig. 3. PILOT *for* 2S-OPT: (a) The planning input state is first transformed from the global coordinate frame to the reference path-based coordinate frame. After the plan is obtained, the output is transformed back to the global coordinate frame by the inverse transform. See [4] for more details; (b) Architecture of 2S-OPT (expert, expensive-to-run planner Ω): a MILP solver initialises an NLP optimiser; (c) Architecture of PILOT for 2S-OPT: input pre-processing produces a sequence of images of the scene, encoding road surface information and the predicted future of dynamic road users, the network $\mathcal{N}_\theta^{2s\text{-OPT}}$ which was specifically designed for this problem, and the NLP problem as in 2S-OPT ($\nu$).

where $\mathcal{J}$ is a cost function of progress and comfort terms defined over the plan $\tau$, $\mathbf{s}_{\text{ego}}$ refers to the initial *ego* state in the scene $\mathbf{s}$, $\mathbf{g_s}$ and $\mathbf{h_s}$ are sets of general inequality and equality constraints, respectively, parameterised by the input scene on the ego-vehicle states. These constraints typically ensure the satisfaction of strict requirements of model dynamics and safety (up to the predicted horizon) of the output plans [16], [3], [4].

There is a wide-ranging literature on attempts to solve relaxations of the problem in Eq. 1, e.g. by turning it into unconstrained optimisation, taking a convex approximation, tackling simplified driving scenes, or limiting the planning horizon [16], [26], [27]. A recent work by Schwarting *et al.* solves Eq. 1 directly in a receding horizon fashion [3], yet the authors identify local convergence as a setback of their method[1]. In [4], Eiras *et al.* mitigate this issue by warm-starting the solver, which negatively affects runtime efficiency. We describe next the architecture of [4], which we then use in Sec. IV to practically demonstrate the effectiveness of PILOT.

### C. A Two-stage Optimisation-based Motion Planner

Fig 3(a, b) show the general architecture of the two-stage optimiser of [4], which we will refer to as **2S-OPT**. The input to the system are: 1) a birds-eye view of the planning scene, that includes the ego-vehicle, other road users and the relevant features of the static layout; 2) a reference route provided by an external route planner; and 3) predicted traces for all road users provided by a prediction module. Projecting the world state and predictions into a reference path-based coordinate frame produces the 2S-OPT input (Fig 3(a)).

The Nonlinear Programming (NLP) problem solved in [4] follows the structure of Eq. 1 with the following constraints: 1) *Kinematic feasibility* (equality): an *ego* state at time $k$ can be obtained by applying a discrete bicycle model to the state at time $k-1$; 2) *Velocity limits* (inequality): the speed is lower-bounded by the minimum speed (typically 0) and upper-bounded by the speed limit; 3) *Control input bounds* (inequality): the control inputs are lower- and upper-bounded; 4) *Jerk bounds* (inequality): the change in the control inputs is lower- and upper-bounded; 5) *Border limits* (inequality): the *ego* remains within the driveable surface (e.g., the road surface, or lane if overtaking is not allowed); 6) *Collision avoidance* (inequality): the *ego* does not collide with any other road user or object.

The cost function $\mathcal{J}_{\text{2s-OPT}}$ comprises a linear combination of quadratic terms of comfort (bounded acceleration and jerk) and progress (longitudinal and lateral tracking of the reference path, as well as target speed). More details on the precise formulation of the constraints, cost function and parameters are available in Appendix A.

To solve this optimisation problem, the two-stage architecture presented in Fig 3(b) is applied. The first stage solves a receding-horizon, linearised version of the planning problem using a Mixed-Integer Linear Programming (MILP) solver. The output of the MILP stage is fed in one go as a warm-start initialisation to the NLP optimiser. This second optimisation stage ensures that the output trajectory is smooth and feasible, while maintaining safety guarantees.

## III. PILOT: PLANNING BY IMITATION LEARNING AND OPTIMISATION

We now introduce PILOT, an efficient general solution to attain the benefits of expensive-to-run optimisation-based planners. As is well known in the community, while solving the general problem in Eq. 1 globally is NP-hard [29], [30], there are efficient solvers that can compute local solutions within acceptable times *if* a sensible initialisation is provided [31], [32]. We define $\nu : \mathcal{S} \times \mathcal{T} \to \mathcal{T}$ to be such an *efficient optimiser*. We denote by $\Omega : \mathcal{S} \to \mathcal{T}$ the 'expert', *expensive-to-run optimisation* procedure that has the potential to converge from an uninformed initialisation to lower cost solutions than $\nu$. Practical examples of $\Omega$ include recursive decompositions of the problem and taking the minimum cost [33], or informed warm-starting [4], [34].

The goal of PILOT is to safely achieve low costs on $\mathcal{J}$ comparable to the ones achievable by $\Omega$, in runtimes comparable to the efficient $\nu$. To do so, PILOT employs an imitation learning paradigm to train a deep neural network $\mathcal{N}_\theta$ to imitate the output of $\Omega$, which it then uses at inference time to initialise $\nu$. While in theory the network would naturally achieve a low cost while satisfying the constraints (perfect learning), in practice this is not the case. As such, $\nu$ works as an efficient online correction mechanism that

[1] While the cost function in that work is for semi-autonomous driving, this does not affect the general difficulty of the problem.

```
Algorithm 1: PILOT Inference Step
input  : state s, trained imitation network $\mathcal{N}_\theta$,
         efficient planner $\nu$
output: optimal plan $\tau^*$
Obtain initial trajectory $\tau_{\mathcal{N}_\theta} \leftarrow \mathcal{N}_\theta(\mathbf{s})$
Get $\tau^*$ by optimising $\mathcal{J}$ using $\nu(\mathbf{s}, \tau_{\mathcal{N}_\theta})$
return $\tau^*$
```

```
Algorithm 2: PILOT Training Procedure
input  : initial dataset $\mathcal{D}_0 = \{(\mathbf{s}_i, \tau_i^*)\}_{i=1:n}$, expert
         planner $\Omega$, efficient planner $\nu$, simulator $\mathcal{S}$,
         training problems count $J$, retrain count $K$
output: trained network parameters $\theta$
Initialise $\mathcal{D}$ to $\mathcal{D}_0$
Pre-train $\theta \leftarrow$ Train($\mathcal{N}, \mathcal{D}$)
for $j \in \{n+1, \ldots, J\}$ do
    if simulation finished then
        s' ← Initialise a new simulation
    else
        s' ← $\mathbf{s}_{j-1}$
    Obtain $\mathbf{s}_j$ from $\mathcal{S}$ by PILOT(s';$\mathcal{N}_\theta, \nu$) step
    Get $\tau_j^*$ by optimising $\mathcal{J}$ using $\Omega(\mathbf{s}_j)$
    Update $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_j, \tau_j^*)\}$
    // retrain network every $K$ steps
    if $(j-n)$ mod $K = 0$ then
        Update $\theta \leftarrow$ Train($\mathcal{N}, \mathcal{D}$)
return $\theta$
```

uses this informed initialisation to output low cost, safe and feasible trajectories. More details about the inference procedure is shown in Algorithm 1 and Fig. 1 (bottom).

In order to achieve that, we pre-train the network on problems solved by the expert planner $\Omega$, $\mathcal{D}_0 = \{(\mathbf{s}_i, \tau_i^*)\}_{i=1:n}$. Then, with the pre-trained network $\mathcal{N}_\theta$ and the efficient optimiser $\nu$ acting as a planner, we employ a DAgger-style training loop [18] in a simulator to adapt to the covariate shift in $\mathcal{D}_0$ to the learner's experience in the simulator. For more details about training, see Algorithm 2 and Fig.1 (top).

## IV. PILOT for the Two-stage Optimisation-based Motion Planner

To demonstrate the effectiveness of PILOT, we apply it to the use case of 2s-OPT. To do so, we take 2s-OPT as the expensive-to-run planner $\Omega$, and borrow its NLP constrained optimisation stage as the efficient optimisation planner $\nu$ – see Fig. 3(c). We design a deep neural network $\mathcal{N}_\theta^{\text{2s-OPT}}$ that outputs smooth trajectories given as input a graphical representation of the scene and other scalar parameters of the problem (e.g. ego-vehicle speed). We train the network using Algorithm 2 to imitate the output of 2s-OPT when presented with the same planning problem.

The planning scene, s, comprises the static road layout, road users with predicted trajectories, and a reference path to follow which acts as a behaviour conditioning input (Fig. 3(a)). As the problem is transformed to the reference path coordinate frame, the resulting scene is automatically aligned with the area of interest – the road along the reference path, simplifying the network representation.

To graphically encode the predicted trajectories of dynamic road users, $C$ greyscale, top-down images of the scene $I_t^{\mathbf{s}} \in \mathbb{R}^{W \times H}$ are produced by sampling the predicted positions of road users uniformly at times $t \in \{0, \frac{h}{C-1}, \ldots, h\}$, for a planning horizon $h = N\Delta t$. These images are stacked $\mathcal{I}^{\mathbf{s}} = I_{1:C}^{\mathbf{s}} \in \mathbb{R}^{C \times W \times H}$ and fed into convolutional layers to extract semantic features, as shown in Fig. 3(c). This is similar to the input representation in previous works, e.g. ChauffeurNet [24], with the exception that in our case the static layout information is repeated on all channels.

Additional information of the planning problem that is not visualised in the top-down images (such as the initial speed of the ego-vehicle) is appended as scalar inputs along with the flattened convolutional layers output to the first dense layer of the network. Refer to Appendix B for more details.

The desired output of the network is a trajectory in the reference path coordinate frame, encoded as a vector of time-stamped positions $\rho^\theta = \{(x_j, y_j)\}_{j=1,\ldots,N} \in \mathbb{R}^{2 \times N}$. With this representation, we define the training loss to be the $L_2$ norm between the expert trajectory and the network output:

$$\mathcal{L}_\theta(\mathcal{D}) = \frac{1}{nN} \sum_{i \in \mathcal{D}} ||\rho_i^\theta - \rho_i^*||^2 + \mu||\theta||^2, \qquad (2)$$

where $\theta$ is the neural network parameter vector, $\mathcal{D}$ is the training dataset, $\rho_i^*$ is the expert's time-stamped position at index $i$ from the dataset, and $\mu$ is a regularisation parameter.

The efficient NLP optimisation planner (Sec. II-C) expects as initialisation a time-stamped sequence of positions, speeds, orientations and control inputs (steering and acceleration) over the full-horizon, all in the reference path coordinate frame, as a single optimisation problem (cf. the traditional receding-horizon setting). We calculate speeds and orientations from the network's output sequence (after post-processing – Appendix C), and derive the control values from the inverse dynamics model.

## V. Experiments

In this section, we attempt to answer the following questions to demonstrate the effectiveness of PILOT:

1) *How does* PILOT *fare compared to the expert, expensive-to-run optimiser it is trained to imitate?*
2) *Is the imitation neural network alone sufficient to produce safe, feasible and low cost solutions, similar to those of the expert?*
3) *Is the imitation network necessary for the efficient optimiser $\nu$ to converge to feasible and low cost solutions, or are simple heuristics sufficient?*
4) *How does* PILOT *compare to a baseline that trains a network to directly optimise the objective of $\nu$?*

To answer question 1), we compare PILOT and 2s-OPT in terms of solving time and closed-loop trajectory cost using $\mathcal{J}_{\text{2s-OPT}}$ (Sec. V-B). We investigate question 2) by comparing constraint satisfaction in PILOT and in the imitation network

Fig. 4. Representative example scenarios from the CARLA LargeScale benchmarking dataset, showing a variety of conditions like handling moving vehicles, overtaking static vehicles, road stretches and junctions in `Town01`.

$\mathcal{N}_\theta^{\text{2s-OPT}}$ alone (Sec. V-C). To shed light on question 3) we perform an ablation on the initialisation of the efficient optimiser, in this case the NLP solver, by swapping the network with different heuristics and comparing the solution quality (Sec. V-D). Finally to answer question 4), we implement the state-of-the-art Constrained Policy Net (CPN) [28], in which a neural network is trained directly with a loss function that approximates the optimiser's objective, and compare it to PILOT with regard to constraint satisfaction (Sec. V-E).

### A. *Experimental Setup*

We trained and benchmarked PILOT using CARLA simulator (v 0.9.10) [19], where we can realise complex interactions with, and between, other vehicles that would be hard to generate by synthetically perturbing a scenario. To that end, we obtained 20,604 planning problems from randomly generated scenarios in `Town02` with up to 40 non-ego vehicles controlled by CARLA's `Autopilot`. These problems are then solved using 2s-OPT to get the base dataset $\mathcal{D}_0$. We trained PILOT using Algorithm 2, randomly spawning the *ego* and other vehicles in new simulations. For benchmarking, we generated a dataset of 1,000 problems in `Town01`, with representative example problems shown in Fig. 4. We refer to this dataset as LargeScale to differentiate it from the one used in Sec. V-E.

### B. PILOT *vs.* 2s-OPT

We compare the quality of the plans produced by PILOT and 2s-OPT with two metrics:

- *Solving times* (s) – the time required to initialise the efficient NLP stage (using the MILP stage in 2s-OPT, and using the neural network for PILOT), NLP solver runtime after initialisation, and the total time. Lower solving time is better.
- *Cost* – the $\mathcal{J}_{\text{2s-OPT}}$ cost of NLP output upon convergence as in Eq. 3 (Appendix A), reflecting the quality of the final solution, where lower cost values are better.

TABLE I

PILOT vs. 2s-OPT: solving time and cost ($\mathcal{J}_{\text{2s-OPT}}$) on 962/1,000 problems where both PILOT & 2s-OPT converge (mean ± standard deviation over a varied set of driving problems)

| Planner | Time (s) | | | Cost |
|---|---|---|---|---|
| | Initialisation | NLP | Total | |
| **PILOT** | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.10 \pm 0.15}$ | $\mathbf{0.12 \pm 0.15}$ | $0.58 \pm 0.69$ |
| 2s-OPT | $0.70 \pm 1.25$ | $0.17 \pm 0.23$ | $0.87 \pm 1.31$ | $\mathbf{0.57 \pm 0.68}$ |

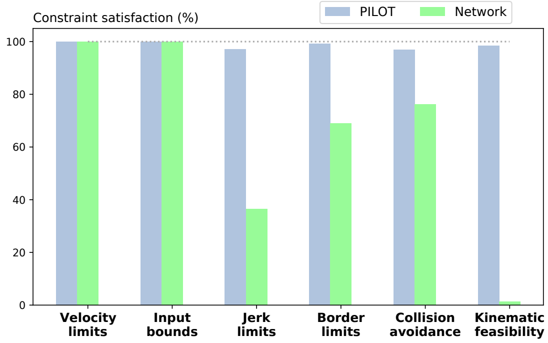

Fig. 5. PILOT vs. $\mathcal{N}_\theta^{\text{2s-OPT}}$: constraint satisfaction percentages in the LargeScale benchmarking dataset of 1,000 problems.

We report the value of these metrics in the LargeScale benchmarking dataset in Table I, vindicating our approach of combining an imitation learning network with an optimiser to produce an efficient, safe planner. PILOT shows a clear advantage in runtime efficiency when compared to 2s-OPT, leading to savings of ~**86%** in total runtime, with no significant deterioration in solution quality (~5% drop).

### C. PILOT *vs.* $\mathcal{N}_\theta^{\text{2s-OPT}}$

We showcase the advantages of having an efficient optimiser rectifying the network mistakes at inference time by comparing the trajectories obtained by PILOT to those obtained by $\mathcal{N}_\theta^{\text{2s-OPT}}$, PILOT's trained imitation network, when used by itself as a planner. We show in Fig. 5 the satisfaction rate of the constraints as defined in Sec. II-C in the LargeScale benchmark dataset.

As can be observed, $\mathcal{N}_\theta^{\text{2s-OPT}}$ struggles to reach the constraint satisfaction levels of PILOT, particularly with the equality kinematic feasibility constraints. While this particular kind of constraint could be addressed with additional kinematic output layers in the network [35], PILOT provides a simpler and a more general approach that improves the satisfiability of most constraints.

### D. *Efficient optimiser initialisation ablation*

We present an ablation study on the quality of the imitation network as an initialisation to the efficient NLP optimiser, compared to simple, heuristic alternatives. In particular we consider: **None** – an initialisation which sets *ego* position, yaw and speed to zero at all timesteps; **ConstVel** – a constant

TABLE II

$\nu$ INITIALISATION ABLATION: COMPARISON OF EACH METHOD W.R.T. 2S-OPT ON MEAN SOLVING TIME/NLP COST (IN PROBLEMS SOLVED BY BOTH 2S-OPT & INITIALISATION) AND CONVERGENCE PERCENTAGE.

| Initialisation | Δ NLP solve time (s) | Δ NLP cost (%) | Converged (%) |
|---|---|---|---|
| None | +0.66 | +9.3% | 89.5% |
| ConstVel | +0.18 | +2.9% | 95.3% |
| ConstAccel | +0.44 | **-0.1%** | 91.7% |
| ConstDecel | +0.35 | +8.9% | 95.3% |
| $\mathcal{N}_\theta^{\text{2s-OPT}}$ **(PILOT)** | **-0.07** | +2.3% | **96.8%** |
| MILP (2S-OPT) | - | - | 99.2% |

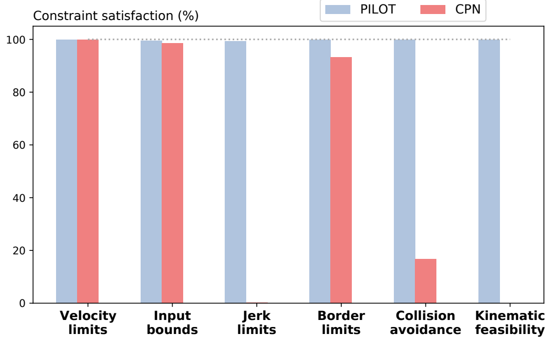

Fig. 6. PILOT vs. CPN: constraint satisfaction percentages in SMALLSCALE benchmarking dataset of 1,000 problems.

velocity initialisation that maintains the *ego*'s heading; and **ConstAccel/ConstDecel** – constant acceleration and deceleration initialisations for which the speed is changed with a constant rate until it reaches the speed limit or 0, respectively.

We compare the alternatives, relative to the original 2S-OPT MILP stage initialisation, in the LARGESCALE benchmarking dataset with three metrics:

- Δ *NLP solving time* and Δ *NLP cost*– we report the average difference in solving time (relative to MILP) and the percentage change in the cost of the output trajectory compared to MILP in the problems that both the initialisation method and 2S-OPT solved.
- *Percentage of solved problems* – constrained, non-linear optimisation in general is not guaranteed to converge to a feasible solution, hence the quality of an initialisation would be reflected in a higher percentage of solved problems. We report the percentage of solved problems out of the problems that 2S-OPT solved.

Results in Table II show that PILOT's neural network initialisation produces trajectories that are easier to optimise (reduced NLP solving time) with only a small averaged increase in the final cost compared to MILP. ConstAccel has a slight advantage in NLP cost on the problems it solves, but solves far fewer and takes significantly longer to converge.

### *E.* PILOT *vs. CPN*

We showcase the advantages of our framework by comparing it to an optimiser-free alternative: CPN [28], a state-of-the-art method that trains a neural network directly with a loss function that approximates the optimiser cost function.

Attempts to train CPN naïvely on LARGESCALE failed to result in an effective network, leading to the more elaborate training procedure discussed in Appendix D. Thus, to facilitate a fair comparison between PILOT and CPN, we created a simpler dataset in Carla's `Town02` (SMALLSCALE), with 20,000 problems that are limited to up to 3 static vehicles on a straight stretch of road. We use a dataset of 1,000 problems randomly generated in the same way for benchmarking.

Fig. 6 shows a bar plot of constraint satisfaction rates in SMALLSCALE benchmark dataset. CPN fails to guarantee kinematic feasibility, collision safety and comfort requirements in the output. On the other hand, PILOT is guaranteed to satisfy these requirements when the efficient optimiser $\nu$ converges – 99.4% of the problems in this dataset.

## VI. DISCUSSION AND CONCLUSION

We now review the questions posed in Sec. V in light of the experimental results. We demonstrated in Sec. V-B the effectiveness of PILOT in replacing an *expensive-to-run* optimiser (e.g., 2S-OPT) by showing a reduction of nearly 7× in runtime compared to the optimiser while only suffering a marginal increase in final cost. As mentioned in Sec. II, the expensive-to-run optimiser we use here builds on the fast framework of Schwarting *et al.* [3] that can re-plan at 10Hz but suffers from local convergence issues. 2S-OPT [4] offers an improvement in convergence and quality at the cost of runtime efficiency, with our implementation having a re-plan rate of around 1Hz. PILOT, when applied to 2S-OPT, allows for a re-plan rate of 8Hz, approaching the original speed of [3] but with improved convergence and lower cost solutions similar to 2S-OPT.

Our training procedure is an imitation learning paradigm with online dataset augmentation using the expensive-to-run optimiser as the expert. This could be interpreted as a technique of policy distillation [17], replacing the sophisticated expert with a much more efficient proxy. The generalisation power of the expert is maintained to some extent through the efficient optimiser stage that actively tries to satisfy the same constraints as the expert. The initialisation ablation of the efficient optimiser presented in Sec. V-D showcases the benefits and the quality of trained imitation network when compared to simple alternatives. The simplicity of our training paradigm is corroborated further by the comparison to CPN in Sec. V-E. The nature of the optimisation of CPN training for complex formulations with many constraints as we have in 2S-OPT results in a difficult training process, requiring careful fine-tuning (see Appendix D). PILOT, on the other hand, relies on the solutions of the expert with a simple $L_2$ loss, limiting the need for fine-tuning.

We justify the design of our inference procedure in Sec. V-C, showing that PILOT's efficient optimiser effectively corrects the network output, leading to safer, constraint satisfying solutions. Moreover, the efficient optimiser operates on the full-length, long-horizon trajectory that is produced by the network, in contrast to existing approaches in which the optimisation at inference time is restricted to a limited horizon [26]. In the case of [26] in particular, the short-term MPC problem is highly conditioned on the network's output, which has the potential of creating sub-optimal, and even unsafe, solutions if the network yields a poor result.

The complexity of the expert optimiser and the cost of running it within our framework influences only the training phase of the imitation network and has no effect on the inference phase. Thus, in the future we are interested in exploring more advanced experts, e.g., returning the solution with the minimum cost using an ensemble of initialisations [33]. Furthermore, one could investigate applying conditional imitation learning [36] and other loss functions, e.g. $L_1$ [9], to improve further the quality of the initialisation provided by the network and bridge the existing gap between the expert and efficient optimisers.

## Appendix

### A. Nonlinear programming problem formulation

Following the definition from [4], we take $\Delta t$ to be the timestep between states, $N$ to be the desired plan length, and we assume the discretised kinematic bicycle model $\mathbf{x}_{k+1} = f_{\Delta t}(\mathbf{x}_k, \mathbf{u}_k)$ where $\mathbf{x}_k = (x_k, y_k, \theta_k, v_k)$ is *ego* state (pose and speed) and $\mathbf{u}_k = (a_k, \delta_k)$ is the control inputs (acceleration and steering) applied to the *ego* at step $k$. The goal of the 2s-OPT framework is to solve the following constrained optimisation problem:

$$\begin{aligned} \underset{\mathbf{x}_{1:N}, \mathbf{u}_{0:N-1}}{\operatorname{argmin}} \quad & \mathcal{J}_{\text{2s-OPT}}(\mathbf{x}_{1:N}, \mathbf{u}_{0:N-1}) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = f_{\Delta t}(\mathbf{x}_k, \mathbf{u}_k) \\ & 0 \le v_{\min} \le v_k \le v_{\max} \\ & |\delta_k| \le \delta_{\max} \\ & a_{\min} \le a_k \le a_{\max} \\ & |a_{k+1} - a_k| \le \dot{a}_{\max} \\ & |\delta_{k+1} - \delta_k| \le \dot{\delta}_{\max} \\ & \mathcal{E}(\mathbf{x}_k) \cap \left( \left[ \mathbb{R}^2 \setminus \mathcal{B} \right] \cup \mathcal{S}_k^{1:w} \right) = \emptyset, \, \forall k \end{aligned} \tag{3}$$

where $v_{\min}$ is the minimum desired speed, $v_{\max}$ is the road's speed limit, $\delta_{\max}$ is maximum allowed steering input, $[a_{\min}, a_{\max}]$ is the allowed range for acceleration/deceleration commands, $\dot{a}_{\max}$ is the maximum allowed jerk, $\dot{\delta}_{\max}$ is the maximum allowed angular jerk. Additionally, $\mathcal{B} \subset \mathbb{R}^2$ is the driveable surface that is safe to drive based on the layout, $\mathcal{S}_{1:N}^{1:w} \subset \mathbb{R}^{2 \times N}$ are unions of elliptical areas that encompass the $w$ road users, $\mathcal{S}_k^{1:w}$, for timesteps $k \in \{1, ..., N\}$, $\mathcal{E}(\mathbf{x}_k) \subset \mathbb{R}^2$ is the area the *ego* occupies at step $k$ with, and $\mathcal{J}_{\text{2s-OPT}}$ is a cost function comprising a linear combination of quadratic terms of comfort (reduced acceleration and jerk) and progress (longitudinal and lateral tracking of the reference path, as well as speed) [4]. In 2s-OPT, the *ego*'s area $\mathcal{E}(\mathbf{x}_k)$ is approximated by its corners, so that the intersection with the driveable surface – delimited by its borders which are defined as $C^2$ functions – and road user ellipses can be computed in closed form [4].

TABLE III
NLP weights and parameters

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $L$ | $4.8\ m$ | $\dot{\delta}_{\max}$ | $0.18\ rad/s^2$ | $\omega_x$ | 0.1 |
| $\delta_{\max}$ | $0.45\ rad/s$ | $v_{\max}$ | $10\ m/s$ | $\omega_v$ | 2.5 |
| $a_{\min}$ | $-3\ m/s^2$ | $v_{\min}$ | $0\ m/s$ | $\omega_y$ | 0.05 |
| $a_{\max}$ | $3\ m/s^2$ | $\omega_\delta$ | 2.0 | $\omega_a$ | 1.0 |
| $\dot{a}_{\max}$ | $0.5\ m/s^3$ | | | | |

The cost function to optimise is defined as

$$\mathcal{J}_{\text{2s-OPT}}(\mathbf{x}_{1:N}, \mathbf{u}_{0:N-1}) = \sum_{k=0}^{N} \sum_{\iota \in \mathcal{I}} \omega_\iota \theta_\iota(\mathbf{x}_k, \mathbf{u}_k) \tag{4}$$

where $\omega_\iota \in \mathbb{R}$ are scalar weights, and $\theta_\iota(\mathbf{z}_k, \mathbf{u}_k)$ are soft constraints that measure deviation from the desired speed ($\omega_v$), the reference path ($\omega_y$) and the end target location ($\omega_x$), and that control the norms of acceleration and steering control inputs ($\omega_a$ and $\omega_\delta$). We fine-tune the parameters of the optimisation using grid-search in the parameter space.

The parameters of the optimisation are in Table III.

### B. PILOT *for* 2s-OPT*: deep neural network architecture*

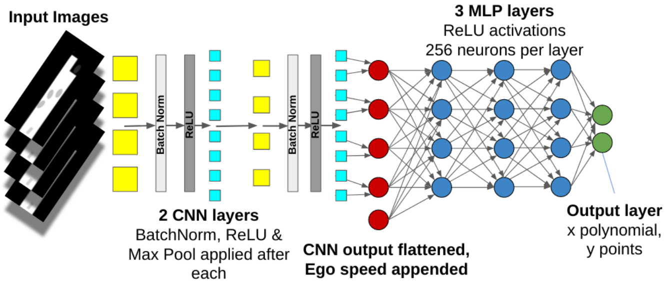

Fig. 7. PILOT for 2s-OPT network architecture.

### C. Output transformation checks

The network produces a sequence of spatial positions, then the rest of the required input of the efficient optimiser are computed from that sequence. A number of checks of upper and lower limits are applied to tame abnormalities in the network output and to improve the input to the optimiser:

- Velocity limits: $v_k \in [0, v_{\max}]$
- Acceleration/deceleration limits: $a_k \in [a_{\min}, a_{\max}]$
- Maximum jerk limit: $|a_{k+1} - a_k| \le \dot{a}_{\max}$
- Maximum steering angle limit: $|\delta_k| \le \delta_{\max}$
- Maximum angular jerk limit: $|\delta_{k+1} - \delta_k| \le \dot{\delta}_{\max}$

### D. CPN baseline training procedure

After many failed attempts at training with all constraints from a random initialisation, we applied a curriculum learning approach [37], sequentially introducing constraints and tuning their weights with each introduction. This approach does not scale well to complex constraint sets as it requires expert knowledge of the constraints.

In our case, all terms of $\mathcal{J}_{\text{2s-OPT}}$ satisfy differentiability. To make the hard constraint terms differentiable, we approximate them with ReLUs that penalise constraint violation as in [28]. The ReLUs have large gradients to ensure they are prioritised over soft constraints. As the hard constraints have different units, they require normalising to ensure the cost function and gradient used for training reflect this.

## References

[1] I. Batkovic, M. Zanon, M. Ali, and P. Falcone, "Real-time constrained trajectory planning and vehicle control for proactive autonomous driving with road users," in *Proceedings of the European Control Conference*. IEEE, 2019, pp. 256–262.

[2] J. Chen, W. Zhan, and M. Tomizuka, "Autonomous driving motion planning with constrained iterative LQR," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 244–254, 2019.

[3] W. Schwarting, J. Alonso-Mora, L. Paull, S. Karaman, and D. Rus, "Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 99, 2017.

[4] F. Eiras, M. Hawasly, S. V. Albrecht, and S. Ramamoorthy, "A two-stage optimization-based motion planner for safe urban driving," *IEEE Transactions on Robotics*, pp. 1–13, 2021, to appear.

[5] A. Sadat, S. Casas, M. Ren, X. Wu, P. Dhawan, and R. Urtasun, "Perceive, predict, and plan: Safe motion planning through interpretable semantic representations," in *European Conference on Computer Vision*. Springer, 2020, pp. 414–430.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv:1604.07316*, 2016.

[7] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[8] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah *et al.*, "Urban driving with conditional imitation learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 251–257.

[9] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Proceedings of the Conference on Robot Learning*. PMLR, 2020, pp. 66–75.

[10] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 3578–3586.

[11] E. W. Ayers, F. Eiras, M. Hawasly, and I. Whiteside, "PaRoT: A practical framework for robust deep neural network training," in *NASA Formal Methods*. Springer, 2020, pp. 63–84.

[12] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *arXiv preprint arXiv:1903.06758*, 2019.

[13] J. P. Hanna, A. Rahman, E. Fosong, F. Eiras, M. Dobre, J. Redford, S. Ramamoorthy, and S. V. Albrecht, "Interpretable goal recognition in the presence of occluded factors for autonomous vehicles," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[14] S. V. Albrecht, C. Brewitt, J. Wilhelm, B. Gyevnar, F. Eiras, M. Dobre, and S. Ramamoorthy, "Interpretable goal-based prediction and planning for autonomous driving," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[15] J. A. DeCastro, K. Leung, N. Aréchiga, and M. Pavone, "Interpretable policies from formally-specified temporal properties," in *Proceedings of the International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[16] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[17] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[18] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 78. PMLR, 2017, pp. 1–16.

[20] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[21] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9329–9338.

[22] A. Filos, P. Tigas, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal, "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?" in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[23] J. Chen, B. Yuan, and M. Tomizuka, "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2884–2890.

[24] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," in *Robotics: Science and Systems*, 2019.

[25] K. Lee, K. Saigol, and E. A. Theodorou, "Safe end-to-end imitation learning for model predictive control," *arXiv preprint arXiv:1803.10231*, 2018.

[26] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," in *Proceedings of the Dynamic Systems and Control Conference*, vol. 3. ASME, 2018.

[27] F. S. Acerbo, H. Van der Auweraer, and T. D. Son, "Safe and computational efficient imitation learning for autonomous vehicle driving," in *Proceedings of the American Control Conference (ACC)*. IEEE, 2020, pp. 647–652.

[28] W. Zhan, J. Li, Y. Hu, and M. Tomizuka, "Safe and feasible motion generation for autonomous driving via constrained policy net," in *Conference of the IEEE Industrial Electronics Society*, 2017, pp. 4588–4593.

[29] C. A. Floudas and P. M. Pardalos, *State of the art in global optimization: computational methods and applications*. Springer Science & Business Media, 2013, vol. 7.

[30] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[31] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[32] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *International Journal of Control*, pp. 1–17, 2017.

[33] A. L. Friesen and P. Domingos, "Recursive decomposition for nonconvex optimization," in *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI, 2015, pp. 253–259.

[34] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.

[35] H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, J. Schneider, D. Bradley, and N. Djuric, "Deep kinematic models for kinematically feasible vehicle trajectory predictions," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 563–10 569.

[36] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4693–4700.

[37] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the International Conference on Machine Learning*, 2009, pp. 41–48.