PERSONALIZED AUTONOMOUS VEHICLE MOTION CONTROL USING IRL AND MPC

DISSERTATION FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY IN ELECTRICAL AND COMPUTER ENGINEERING

ZHAODONG ZHOU

OAKLAND UNIVERSITY

2026

PERSONALIZED AUTONOMOUS VEHICLE MOTION CONTROL USING IRL AND MPC

by

ZHAODONG ZHOU

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL AND COMPUTER ENGINEERING

2026

Oakland University
Rochester, Michigan

Doctoral Advisory Committee:

Jun Chen, Ph.D., Chair
Micho Radovnikovich, Ph.D.
Jia Li, Ph.D.
Giselle Sosa Jones, Ph.D.

To my family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Jun Chen, for his continuous guidance, support, and encouragement throughout my Ph.D. study. His expertise, patience, and thoughtful advice have been invaluable to my research and academic development. I am sincerely grateful for the time and effort he has devoted to helping me grow as a researcher. Sincere thanks are also extended to my dissertation committee members, Micho Radovnikovich, Ph.D., Jia Li, Ph.D., and Giselle Sosa Jones, Ph.D., for their valuable time, constructive comments, and thoughtful suggestions. Their feedback and guidance have helped improve the quality of this dissertation.

I am deeply grateful to my family for their unconditional love, patience, and encouragement. Their understanding and support have been a constant source of strength throughout my years of study abroad. This dissertation would not have been possible without them.

Last but not least, I would also like to thank my peers and friends at Oakland University, including Ali Irshayyid, Yunge Li, Christopher Rother, Wanqun Yang, Luke Nuculaj, and Keer Chen for their support, encouragement, and companionship throughout my Ph.D. journey. I am grateful for the discussions, collaboration, and shared experiences that made this process more meaningful.

Zhaodong Zhou

ABSTRACT

PERSONALIZED AUTONOMOUS VEHICLE MOTION CONTROL USING IRL AND MPC

by

Zhaodong Zhou

Adviser: Jun Chen, Ph.D.

Autonomous vehicle (AV) motion control has been widely studied to improve path tracking accuracy, driving comfort, and safety. Model predictive control (MPC) is a promising method for AV control because it can explicitly handle vehicle dynamics, constraints, and multi-objective optimization. However, conventional time-triggered MPC requires solving an optimal control problem at every time step, which creates a high computational burden for real-time implementation. In addition, traditional MPC usually depends on manually tuned cost weights and predefined reference trajectories, making it difficult to adapt the controller to individual driving preferences. This dissertation addresses these challenges by developing learning-based personalized and computationally efficient motion control methods for AVs. First, event-triggered MPC is investigated for AV path tracking, where the optimal control problem is solved only when a triggering condition is satisfied. A switching prediction model is further used to support MPC operation across different speed ranges. To improve the controller behavior during non-triggered intervals, a linear based inter-event feedback method is developed to update the steering command using the current vehicle state without solving a new optimization problem at every sampling step. Second, inverse reinforcement learning is used to learn personalized lane change behavior from expert demonstrations. Interpretable trajectory features are designed to represent driving comfort, efficiency, lateral position, heading

behavior, yaw response, and steering effort. The learned cost weights are used for personalized lane change path generation and are further incorporated into MPC to generate personalized lane change maneuvers without manual cost tuning. The proposed methods are validated using CARLA simulation, real-vehicle path-tracking experiments, scale-vehicle experiments, and truck on-road lane change testing. Results show that the proposed framework can reduce computational demand while maintaining effective tracking performance, improve inter-event feedback behavior, and generate lane change trajectories that closely match individual driver behavior.

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

LIST OF TABLES

LIST OF TABLES—Continued

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Autonomous driving is not a single control problem, but an integrated system that connects perception, localization, planning, and control. Perception provides information about road boundaries, lane markings, surrounding vehicles, pedestrians, and other traffic participants, while localization estimates the vehicle position, velocity, and orientation in the driving environment [1, 2]. Based on the perceived environment and estimated vehicle state, the planning layer selects a feasible route, trajectory, or maneuver that satisfies road geometry, traffic rules, and interaction requirements with surrounding traffic [3, 4]. The control layer then converts this planned motion into actuator-level commands, such as steering, acceleration, and braking, while accounting for vehicle dynamics, actuator constraints, and physical limitations [5, 6]. In this pipeline, motion control is the final step before the command reaches the vehicle, and therefore errors in tracking, steering smoothness, or actuator timing can directly change the realized vehicle behavior. For lateral motion control in particular, the vehicle must follow the desired path while maintaining heading consistency, limiting steering effort, and respecting steering and steering-rate constraints [7–9]. Therefore, even if perception, localization, and planning provide accurate information, the planned maneuver cannot be safely and comfortably executed without a reliable path-tracking controller [10–12]. This makes autonomous vehicle motion control, especially lateral path tracking, a central problem for translating high-level driving decisions into real vehicle motion.

Among different vehicle motion control methods, model predictive control (MPC) has become one of the most widely studied approaches for autonomous vehicle path tracking. MPC predicts future system behavior over a finite horizon, solves an

optimization problem subject to system dynamics and constraints, and applies the first element of the resulting control sequence in a receding-horizon manner [13–15]. This structure is particularly useful for autonomous vehicles because the controller can include vehicle dynamics, actuator limits, steering-rate limits, and tracking objectives in the same constrained optimization problem [16–18]. In lateral vehicle control, MPC can explicitly penalize path-tracking error, heading error, steering effort, and actuator busyness, which allows the controller to balance tracking accuracy and passenger comfort rather than only minimize geometric path error [7, 8, 19]. Because of these advantages, MPC has been applied to lane keeping, trajectory tracking, lane change execution, collision avoidance, and cooperative driving scenarios [9, 11, 20–22]. These studies show that MPC provides a systematic framework for autonomous vehicle lateral motion control, especially when vehicle constraints and multiple performance objectives must be considered together.

This dissertation focuses on two issues that arise when MPC-based lateral motion control is applied to autonomous vehicles. The first issue is real-time implementation. Although MPC provides a flexible optimization-based control framework, repeatedly solving an optimal control problem at every sampling instant can be computationally demanding, and the resulting delay can influence closed-loop tracking performance in real vehicles. This motivates the study of event-triggered MPC and inter-event feedback control. The second issue is personalization. A fixed MPC cost function can generate safe and feasible lane changes, but it does not naturally reflect the preferred behavior of different drivers. This motivates the use of inverse reinforcement learning (IRL) to learn driver-specific feature weights from expert lane change demonstrations and incorporate them into path generation and MPC-based closed-loop control.

1.1 Real-Time MPC Implementation

The main drawback of MPC is that it requires repeated online optimization. In conventional time-triggered MPC, a new optimal control problem is solved at every

sampling instant, even when the vehicle state changes only slightly from one step to the next. This assumption is common in simulation and theoretical analysis, but it becomes a practical concern when the controller is implemented on real vehicles. The computation time grows with the complexity of the prediction model, the length of the prediction horizon, and the number of constraints [23–25]. Nonlinear vehicle dynamics and tire-force models can improve prediction accuracy, especially when the vehicle experiences significant lateral velocity and yaw-rate response, but they also increase the difficulty of real-time optimization [7, 26, 27]. In addition, the controller must share onboard computing resources with perception, localization, planning, communication, and safety-monitoring modules. Therefore, reducing unnecessary MPC computation is important not only for algorithmic efficiency, but also for practical deployment on autonomous vehicle platforms [12, 27, 28]. A controller that is accurate but too computationally demanding may fail to provide timely commands, while a controller that is computationally light but insufficiently predictive may produce poor tracking behavior. The design problem is therefore not simply choosing a more complex model or a faster solver, but balancing prediction quality, computation time, and closed-loop tracking performance.

Real-time implementation also introduces delay, which is often less visible in simulation than in physical experiments. The MPC solution is computed from the vehicle state measured or estimated at the beginning of the optimization. However, after the optimization is solved, the command still needs to pass through the software and hardware stack before it reaches the steering actuator. During this time, the vehicle continues to move, and the state used by the optimizer may no longer represent the actual state of the vehicle when the command is applied. This delay can be especially influential during curved-road tracking, lane changes, and other maneuvers with fast lateral motion. In simulation, the plant may effectively wait for the controller output, or the computation

delay may not be modeled explicitly. In real-world testing, however, optimization time, communication delay, actuator response, sensor noise, and model mismatch all become part of the closed-loop system [12, 27]. For this reason, MPC for autonomous vehicles should be evaluated not only by tracking accuracy in simulation but also by optimization frequency, command timing, and closed-loop behavior in real-world operation.

Event-triggered MPC provides a way to reduce unnecessary optimization while retaining the feedback and constraint-handling advantages of MPC. The general idea of event-triggered control is to update the control action only when a triggering condition indicates that a new update is needed [29]. This idea has been studied for networked control, nonlinear systems, discrete-time systems, and constrained MPC problems, where reducing communication or computation is an important design objective [30–33]. Related studies have also investigated robustness, sampling behavior, and stability properties of event-triggered and self-triggered control schemes [34–37]. In event-triggered MPC, the optimal control problem is not solved at every fixed sampling instant. Instead, the controller re-optimizes when the system state, tracking error, prediction error, or another performance-related metric violates a selected condition. This creates an aperiodic optimization pattern. The controller updates more frequently when the maneuver is demanding, such as during turns or lane changes, and updates less frequently when the previously computed solution remains sufficient, such as during straight-road tracking with small lateral error.

For autonomous vehicle path tracking, a natural triggering variable is the lateral offset between the current vehicle position and the reference path. During straight-road driving with small tracking error, repeatedly solving similar optimization problems may provide little additional benefit. During turns or lane changes, however, the vehicle state changes more quickly and the controller may need to re-optimize more often. Event-triggered MPC has therefore been studied in several vehicle-related applications,

including vehicle following, platooning, cooperative path following, collision avoidance, and autonomous vehicle lateral control [38–42]. Other recent studies further explore resource-efficient platooning, adaptive event-triggered MPC, and event-triggered learning-based control for autonomous systems [43–46]. Simulation studies in high-fidelity environments such as CARLA are useful for evaluating event-triggered MPC under repeatable traffic and vehicle conditions [28, 47]. However, simulation validation alone cannot fully represent real vehicle implementation, where computation delay, actuator response, and sensor uncertainty may change the closed-loop behavior [12, 27]. This creates a need to study event-triggered MPC not only as a theoretical or simulation-based computation-saving method, but also as a controller that must be validated under real-time implementation constraints.

Although event-triggered MPC reduces the frequency of optimization, the behavior between two triggered optimizations still requires careful design. A common implementation is to shift and apply the remaining elements of the previously optimized input sequence. This sequence-shifting strategy is simple and computationally efficient, but it is mostly open loop during non-triggered intervals. If the actual vehicle state deviates from the predicted state because of model mismatch, localization noise, disturbance, or actuator delay, the shifted input sequence may not provide enough correction until the next event is triggered. This issue becomes more important when the event-trigger threshold is large, because larger thresholds generally reduce optimization frequency but allow longer intervals between feedback updates. Therefore, event-triggered MPC introduces a trade-off. Reducing optimization frequency improves computational efficiency, but insufficient feedback between events can degrade tracking performance. Least-squares based approximation provides a simple numerical tool for extracting local relationships from trajectory data [48, 49]. Such a local approximation can be useful for designing lightweight feedback during non-triggered intervals, because the

latest MPC solution already contains state and input information around the current operating condition. This suggests that inter-event control should not be limited to shifting a previous input sequence; it can also use the current vehicle state to provide feedback correction without solving a new optimal control problem at every step.

1.2 Personalized Lane Change Control

Besides computational efficiency, autonomous vehicle motion control also needs to address personalization. A conventional path-tracking controller is usually tuned to follow a given reference path with fixed cost weights. This may produce safe and smooth motion, but it does not necessarily represent the preference of a specific driver. Human drivers perform lane changes with different styles. Some drivers complete the maneuver quickly and accept larger heading changes, yaw-rate responses, and steering inputs. Other drivers prefer a slower and smoother lateral transition with smaller steering effort. These differences are related to driving style, passenger comfort, and user acceptance [50–52]. Therefore, autonomous vehicles should not only satisfy safety and feasibility constraints; they should also be able to reproduce or adapt to driver-specific motion preferences when appropriate [53–55]. Lane change is a representative maneuver for studying this problem because reaching the adjacent lane is only one part of the task. The path length, lateral transition rate, lane-mark crossing location, curvature, heading profile, yaw-rate profile, and steering effort can all vary while still producing a feasible lane change. Traditional lane change planning and control methods often focus on feasibility, comfort, and tracking performance at a general level [9, 56–58]. However, these methods usually do not directly learn how an individual driver balances smoothness, efficiency, and control effort.

Learning from demonstration provides a more scalable way to capture driver-specific preferences. Imitation learning can learn driving behavior directly from examples, and it has been used for human-like autonomous driving and driver behavior

modeling [52, 59, 60]. However, a direct imitation policy may be difficult to interpret, and it may not explain which aspects of the demonstrated behavior are important. Reinforcement learning can optimize driving behavior through interaction, but it requires a reward function, and manually designing a reward function for human-like driving is difficult [61–63]. Inverse reinforcement learning (IRL) approaches the problem from the opposite direction. Instead of assuming that the reward is known, IRL estimates the reward or cost function that best explains the expert demonstrations [64, 65]. Because the learned cost can be expressed through meaningful features, IRL provides a useful connection between data-driven behavior learning and interpretable controller design [66, 67]. Maximum entropy inverse reinforcement learning is especially relevant for human driving demonstrations because the same driver may not repeat exactly the same lane change under different initial conditions. The maximum entropy formulation assigns probability to possible trajectories while matching expert feature expectations, avoiding the assumption that only one deterministic trajectory is correct [65, 68, 69]. In autonomous driving, IRL has been used to learn driving styles, infer human-like decision-making, and customize autonomous vehicle behavior from demonstrations [52, 55, 70–72].

For lane change personalization, the learned behavior must be represented in a form that is both expressive and compatible with vehicle control. One possible representation is a geometric path. A fifth-order Bezier curve can represent a smooth lane change trajectory with a small number of control points, and the curve shape can be adjusted through these control points while maintaining smoothness [73–75]. In a path-level learning framework, features such as curvature, lane change length, lane-mark crossing location, and final lateral position can describe different aspects of driver preference [76]. However, a personalized path alone is not sufficient if the vehicle controller cannot execute it reliably under dynamics and actuator constraints. This motivates a second representation in which the learned preference is embedded directly

into the MPC cost function. By doing so, the controller does not merely track a predesigned lane change path; it generates the lane change behavior according to learned driver-specific feature weights while respecting vehicle dynamics and steering constraints [77]. This connection between IRL and MPC provides a way to combine interpretable preference learning with model-based constrained control.

The above discussion reveals two gaps that motivate this dissertation. The first gap is between MPC performance in idealized simulation and MPC performance in real vehicle implementation. Existing MPC and event-triggered MPC studies demonstrate strong potential, but real-time computation, command delay, actuator response, and model mismatch must be considered when the controller is deployed on physical vehicle platforms. The second gap is between safe autonomous lane change execution and driver-specific lane change behavior. A fixed manually tuned controller can generate feasible maneuvers, but it cannot naturally capture how different drivers trade off smoothness, efficiency, and steering effort. These two gaps are connected by the role of the MPC cost function and the way it is solved in real time. Efficient MPC implementation asks when and how often the cost optimization should be solved. Personalized MPC asks what the cost should represent. This dissertation studies both questions in a unified autonomous vehicle lateral control context.

1.3 Dissertation Approach

To address the two gaps discussed above, this dissertation develops two connected groups of methods. The first group focuses on the real-time implementation of MPC for autonomous vehicle lateral motion control. A consistent vehicle modeling and MPC framework is first established, including kinematic and dynamic bicycle models, reference-path resampling, steering constraints, and steering-rate constraints. Based on this framework, an event-triggered MPC controller is developed so that the optimal

control problem is solved only when a new solution is needed. The triggering condition is designed according to the lateral offset from the reference path and the availability of the previously optimized control sequence. This design reduces unnecessary optimization during simple driving conditions while allowing the controller to re-optimize during more demanding maneuvers.

The event-triggered MPC framework is then evaluated beyond idealized simulation. CARLA simulation is used to provide a repeatable and high-fidelity environment for studying path tracking under different road sections and event-trigger thresholds. Real-vehicle experiments are further conducted to examine how computation delay, actuator response, sensor noise, and model mismatch affect closed-loop tracking performance. Since a basic event-triggered MPC controller may behave largely open loop during non-triggered intervals, this dissertation also introduces a linear based inter-event feedback strategy. Instead of only shifting the previously optimized input sequence, the linear approximates a local relationship between vehicle-state features and the optimal steering command from the latest MPC solution. This allows the controller to provide feedback correction between optimization events without solving a new optimal control problem at every sampling step.

The second group of methods focuses on personalized lane change behavior learning. Instead of manually tuning MPC cost weights for different drivers, this dissertation uses MaxEnt IRL to learn driver-specific preferences from expert lane change demonstrations. The personalization study is first formulated at the path level. A fifth-order Bezier curve is used to represent lane change paths, and features are designed to describe curvature, lane change length, lane-mark crossing location, and final lateral position. MaxEnt IRL is then used to infer the feature weights that explain the demonstrated lane change behavior. The learned weights can generate personalized geometric paths under different initial conditions.

The personalization study is further extended from geometric path modeling to closed-loop control. In this formulation, the IRL-learned feature weights are embedded directly into the MPC cost function. Therefore, the controller does not simply track a predesigned personalized path; instead, it generates lane change behavior according to the learned driver-specific cost while respecting vehicle dynamics, steering limits, steering-rate limits, and real-time state feedback. This connects the interpretability of IRL with the constraint-handling capability of MPC. Through this structure, the dissertation studies both what the controller should optimize to reproduce personalized lane change behavior and how the MPC optimization can be executed efficiently in real time.

1.4 Main Contributions

This dissertation contributes to autonomous vehicle lateral motion control from two connected perspectives: practical real-time MPC implementation and personalized lane change behavior learning. Rather than treating MPC only as an offline or simulation-based optimization tool, this dissertation studies how MPC can be implemented efficiently on autonomous vehicle platforms and how its cost function can be adapted to represent driver-specific lane change preferences.

The first contribution is the development and validation of an event-triggered MPC framework for autonomous vehicle path tracking. The proposed controller reduces unnecessary optimization by solving the MPC problem only when the lateral tracking condition or horizon-depletion condition indicates that a new solution is needed. This design allows the controller to update more frequently during demanding maneuvers and less frequently during simple tracking conditions. The framework is first evaluated in high-fidelity CARLA simulation under different road geometries and triggering thresholds, and is then validated through real-vehicle experiments. These studies show how event-triggered MPC affects optimization frequency, tracking accuracy, computation

delay, and closed-loop control performance in both simulation and physical vehicle implementation.

The second contribution is a linear based inter-event feedback method for event-triggered MPC. Basic event-triggered MPC often relies on shifting and applying the remaining elements of the previously optimized input sequence during non-triggered intervals. Although this strategy is computationally efficient, it provides limited feedback correction between two optimization events. To address this limitation, the proposed method extracts a local feedback relationship from the latest MPC optimal state and input sequences. The resulting K-matrix is used to compute steering commands from the current vehicle state during non-triggered intervals, providing lightweight feedback correction without solving a new optimal control problem at every sampling step.

The third contribution is an IRL-based personalized lane change modeling and control framework. At the path level, MaxEnt IRL is used with a fifth-order Bezier curve representation to learn driver-specific lane change preferences from expert demonstrations. The learned feature weights describe how the driver balances curvature, lane change length, lane-mark crossing location, and final lateral position. This path-level learning method is further extended to closed-loop control by embedding IRL-learned feature weights into the MPC cost function. Through this integration, the controller can generate personalized lane change behavior while still considering vehicle dynamics, steering limits, steering-rate limits, and real-time state feedback.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 reviews the background and related work on autonomous vehicle motion control, vehicle modeling, MPC, event-triggered MPC, inter-event control, lane change modeling, and IRL-based personalization. Chapter 3 presents the vehicle models and baseline MPC formulation

used throughout the dissertation. The chapter also discusses reference-path resampling and the main steering constraints used in the controller. Chapter 4 develops the event-triggered MPC framework for autonomous vehicle path tracking, including the triggering strategy, standard event-triggered MPC algorithm, and linear based inter-event feedback controller. Chapter 5 presents the experimental evaluation of event-triggered MPC through CARLA simulation, real-vehicle testing, and scale-vehicle experiments for the linear based inter-event control method. Chapter 6 presents the IRL-based personalized lane change path modeling method. Bezier curves are used to represent lane change paths, and MaxEnt IRL is used to learn driver-specific path feature weights. Chapter 7 extends the personalization idea from geometric path modeling to closed-loop control. The learned feature weights are embedded into MPC to generate personalized lane change behavior while accounting for vehicle dynamics and steering constraints. Chapter 8 summarizes the dissertation, discusses the main findings, and outlines future research directions.

CHAPTER TWO

BACKGROUND AND LITERATURE REVIEW

This chapter reviews the prior studies that are directly related to the methods developed in this dissertation. Since Chapter 1 has introduced the general motivation and overall research direction, this chapter focuses on the technical development of existing methods. The review is organized around five topics: model predictive control for autonomous vehicle path tracking, event-triggered MPC and inter-event control, lane change trajectory modeling and path generation, learning-based personalized driving and inverse reinforcement learning, and the research gaps addressed by this dissertation.

2.1 Model Predictive Control for Autonomous Vehicle Path Tracking

Model predictive control (MPC) has been widely used for autonomous vehicle path tracking because it provides a systematic framework for combining vehicle prediction, tracking objectives, and physical constraints. In a standard receding-horizon formulation, an optimal control problem is solved over a finite prediction horizon, and only the first control input is applied before the optimization is repeated at the next sampling instant [13–15, 78]. This structure is attractive for vehicle motion control because it can explicitly consider future path information, vehicle dynamics, steering constraints, and smoothness requirements within one optimization problem [16, 17, 79]. Robust and constrained MPC formulations further provide theoretical tools for handling uncertainty, feasibility, and input or state constraints [18, 80].

Existing MPC-based vehicle control studies can be grouped according to the prediction model used by the controller. Kinematic bicycle models are commonly used for low-speed or moderate-speed path tracking because they describe vehicle motion through geometric relationships among position, heading, velocity, wheelbase, and steering

angle [5, 6, 9, 81]. These models are computationally efficient and are relatively easy to discretize for online optimization. Dynamic bicycle models are used when lateral velocity, yaw rate, tire-force effects, and inertial properties need to be represented more explicitly [7, 8, 19]. Dynamic models are especially relevant for higher-speed driving, curved-road tracking, and maneuvers with more significant lateral dynamics. Other studies extend MPC to nonlinear vehicle models, obstacle avoidance, trajectory tracking, and cooperative vehicle control [4, 20–22, 82]. These studies show that the model used in MPC is closely related to vehicle speed, maneuver type, prediction accuracy, and real-time computational requirements.

The MPC objective function is another important component of vehicle path tracking. Most lateral MPC controllers penalize lateral tracking error, heading error, steering magnitude, and steering-rate variation [7, 9, 19, 21]. The tracking and heading terms keep the vehicle close to the reference path, while steering and steering-rate terms reduce unnecessary actuator activity and improve control smoothness. Steering angle constraints and steering-rate constraints are usually included to ensure that the generated command is physically feasible and compatible with the vehicle actuator. Depending on the driving task, additional terms may be introduced to represent stability, road-boundary constraints, obstacle avoidance, comfort, or cooperative behavior [4, 16, 20, 22]. Therefore, MPC provides a flexible framework in which the same controller structure can be adapted to different autonomous driving tasks through the choice of model, cost function, constraints, and reference trajectory.

MPC has been applied to lane keeping, lane changing, and general trajectory tracking. For lane keeping, the controller mainly reduces lateral and heading errors with respect to a lane centerline or a planned path [6, 9]. For lane change execution, MPC can optimize the steering input while considering the desired lateral transition, vehicle dynamics, and actuator limits [16, 21, 22]. For more complex trajectory-tracking

problems, MPC can combine path-tracking terms with obstacle avoidance, road-boundary constraints, or cooperative objectives [4, 82, 83]. These studies form the control foundation for this dissertation. Chapter 3 presents the baseline vehicle models and MPC formulation used throughout this dissertation, while later chapters build event-triggered and personalized control strategies on top of this MPC structure.

2.2 Event-Triggered MPC and Inter-Event Control

Although MPC is effective for constrained vehicle control, its repeated online optimization can be computationally demanding. Several approaches have been proposed to reduce the computational burden of MPC. Explicit MPC pre-computes the control law offline and evaluates it online as a piecewise function, which can reduce online computation for systems with moderate state dimensions and constraint complexity [17, 84]. Fast numerical optimization methods, warm-starting, and sensitivity-based methods reduce the cost of each online optimization by exploiting the structure of the optimal control problem [23, 24]. Parallel computation and hardware-aware implementations improve computation speed by using available computing resources more efficiently [85, 86]. Learning-assisted MPC has also been studied, where learned models, learned terminal components, or learned approximations are used together with MPC to improve prediction or reduce online computation while still retaining the constraint-handling capability of MPC [11, 25, 87].

For autonomous vehicles, real-time MPC performance is not only determined by the amount of computation, but also by command timing. The MPC optimization is solved using the measured or estimated state at the beginning of the computation. After the optimization is completed, the command still needs to pass through software, communication, and actuation layers before affecting the steering actuator. Therefore, computation delay, communication delay, actuator response, localization error, and sensor

noise can all influence the applied command in a real vehicle [12, 27, 88]. A controller that solves an optimal control problem at every sampling instant may still suffer from delayed control action if the optimization time is not negligible. This issue is especially important for autonomous vehicle lateral control because steering commands must be updated in a timely manner during turning, lane changing, and other dynamic maneuvers.

Event-triggered MPC reduces computation from a different perspective. Instead of solving the MPC problem periodically at every fixed sampling instant, event-triggered MPC solves a new optimization problem only when a triggering condition indicates that a new solution is needed. Event-triggered control has been studied for networked systems, nonlinear systems, discrete-time systems, and constrained predictive control problems [29, 36, 37, 89]. Eqtami et al. developed event-triggered MPC formulations for nonlinear and discrete-time systems, showing that MPC can be updated aperiodically based on system behavior [30, 31]. Li and Shi studied event-triggered MPC for constrained nonlinear systems, while Brunner and coauthors investigated robust event-triggered and self-triggered MPC under uncertainty [32–35]. These studies provide the general theoretical foundation for event-triggered predictive control.

Event-triggered control has also been applied to autonomous vehicle and intelligent transportation systems. Existing studies include vehicle following, vehicle platooning, cooperative driving, path following, collision avoidance, and lateral trajectory tracking [38–43, 90]. Recent works further study adaptive event-triggered MPC, event-triggered learning-based control, and resource-aware autonomous control [44–46, 91]. These studies show that event-triggered control is useful when computation or communication resources should be allocated according to vehicle behavior and tracking demand rather than according to a fixed update schedule.

For autonomous vehicle path tracking, the lateral deviation from the reference path is a natural event-triggering variable. When the vehicle remains close to the reference

path, the previously optimized MPC solution may still provide a useful control sequence. When the lateral offset exceeds a selected threshold, a new MPC problem can be solved using the current vehicle state. Previous work has evaluated this idea in CARLA simulation and demonstrated that event-triggered MPC can reduce the number of optimization calls while maintaining acceptable tracking performance [28, 47]. Real-vehicle experiments further showed that the relationship between optimization frequency and tracking performance can be different from simulation results because computation delay and actuation timing become important in practice [12, 88]. A switching-model event-triggered MPC formulation has also been developed to cover a wider speed range by using kinematic and dynamic prediction models under different driving conditions [27].

A key issue in event-triggered MPC is how the controller behaves between two triggered optimizations. In many event-triggered MPC implementations, once an optimization problem is solved, the resulting optimal control sequence is stored and then shifted forward during non-triggered intervals [12, 28, 31, 33]. This sequence-shifting strategy is simple and computationally efficient because no new nonlinear optimization is required between two triggering instants. However, the inter-event control action is largely open loop. If the actual vehicle state deviates from the predicted state because of model mismatch, localization noise, disturbance, or actuator delay, the shifted control sequence may not provide enough correction until the next event is triggered. This issue becomes more important when the triggering threshold is relaxed, because larger thresholds reduce the optimization frequency but also allow longer intervals between feedback-based MPC updates. Several related approaches have been studied to address the limitation of open-loop inter-event execution. Self-triggered MPC determines the next update time in advance based on predicted system behavior, which reduces unnecessary computations while preserving certain performance or stability properties [92]. Triggering and feedback

co-design methods jointly design the event-triggering rule and the feedback policy, so that the triggering mechanism is not separated from the closed-loop control law [93]. Robust event-triggered MPC considers bounded disturbances and robustness requirements, and can provide theoretical guarantees for constrained systems [32, 35]. Tube MPC is another closely related direction, where a nominal MPC trajectory is combined with an ancillary feedback controller to keep the actual state within a tube around the nominal trajectory [18, 94]. These methods show that feedback correction between planning or optimization updates is important for maintaining robustness and tracking accuracy.

However, these approaches may introduce additional design complexity or assumptions that are not always convenient for real-time autonomous vehicle implementation. Tube-based robust MPC often requires tube construction, constraint tightening, and an explicit bounded disturbance set. Triggering-control co-design and self-triggered MPC may require additional offline analysis or online prediction of future triggering instants. Fixed-gain inter-sample or inter-event feedback controllers can reduce the open-loop nature of sequence shifting, but a fixed gain may not adapt well to different operating conditions, vehicle speeds, or local path geometries [86]. Similarly, LQR-type feedback requires a local linear model and the solution of a Riccati equation, and its design can be less direct when the underlying vehicle model and MPC formulation are nonlinear [95]. These limitations motivate a lightweight inter-event feedback strategy that can use information already generated by the latest MPC optimization. Since each MPC solution contains a predicted state sequence and its corresponding optimal control sequence, the local relationship between state deviation and control input can be approximated from this predicted data. Least-squares regression provides a simple way to estimate such a local feedback gain without constructing invariant tubes, tightening constraints, or solving an additional nonlinear optimization problem [96]. The linear based inter-event controller developed later in this dissertation follows this idea. After

each triggered MPC optimization, a local k-matrix feedback relationship is fitted from the predicted state-input sequence. During non-triggered intervals, the fitted feedback gain generates updated control commands using the current vehicle state, instead of simply applying the shifted open-loop sequence. Therefore, the method is positioned between conventional event-triggered MPC and more complex robust or tube-based MPC methods: it preserves the computational benefit of event-triggered MPC while adding lightweight feedback correction between events.

2.3 Lane Change Trajectory Modeling and Path Generation

Lane change is one of the most important maneuvers in autonomous driving. A lane change trajectory must satisfy road geometry, vehicle feasibility, comfort, and safety requirements. Existing lane change trajectory generation methods include polynomial curves, spline curves, Bezier curves, optimization-based planners, and MPC-based methods. Polynomial and spline-based methods are widely used because they can generate smooth trajectories using boundary conditions on position, heading angle, and sometimes curvature [56, 57, 97]. Bezier curves are also commonly used because the path shape can be adjusted through a small number of control points, and higher-order Bezier curves can provide smooth geometric paths for lane change maneuvers [73–75]. Optimization-based lane change planners formulate trajectory generation as a cost-minimization problem that may include path length, curvature, comfort, safety, and tracking feasibility [4, 9, 82]. MPC-based planning and tracking methods further connect the generated trajectory with vehicle dynamics and actuator constraints [16, 21, 22]. Different path representations emphasize different aspects of the lane change maneuver. Polynomial curves are convenient when the initial and terminal boundary conditions are known. Splines can provide smooth piecewise trajectories and are useful when the path must pass through intermediate waypoints. Bezier curves use control points to shape the

path, which makes them suitable when the lane change geometry needs to be adjusted in a compact and interpretable way [73, 75]. For example, modifying the control points of a Bezier curve can change the lateral transition rate, lane-mark crossing location, final lateral position, and total lane change length. These geometric quantities are closely related to driving style. Therefore, Bezier curves are suitable for personalized lane change modeling, where the objective is not only to generate a smooth path, but also to represent how a particular driver prefers to perform the maneuver [76, 98].

Traditional lane change trajectory generation usually focuses on general objectives such as feasibility, smoothness, comfort, efficiency, and safety. Review studies show that lane change planning often considers road geometry, surrounding vehicles, collision risk, comfort limits, and execution feasibility [58, 99]. These objectives are necessary for autonomous driving, but they do not automatically describe individual driving preference. For example, two drivers may both complete a safe lane change, but one may prefer a short and direct transition, while another may prefer a longer and smoother transition. Similarly, one driver may cross the lane marking earlier, while another may remain in the original lane longer before moving laterally. Therefore, a lane change path model used for personalization should include interpretable features that can describe these differences.

Path-level lane change modeling and closed-loop lane change control are related but not identical. A path generator produces a desired geometric trajectory, while a controller must execute the maneuver under vehicle dynamics, steering constraints, actuator limits, and real-time state feedback. If a personalized path is generated offline or as a reference, a separate tracking controller is still needed to follow the path. In contrast, closed-loop lane change control can directly incorporate the maneuver objective into the controller and update the control action based on the current vehicle state. MPC is useful for this purpose because it can connect path or maneuver objectives with vehicle dynamics and constraints [9, 16, 22]. This distinction motivates the two personalized lane change

formulations developed later in this dissertation. Chapter 6 studies path-level personalized lane change modeling, while Chapter 7 embeds learned driver preference into a closed-loop MPC controller.

2.4 Learning-Based Personalized Driving and Inverse Reinforcement Learning

Learning-based driving behavior modeling has been studied using supervised learning, imitation learning, reinforcement learning, and inverse reinforcement learning. Supervised learning methods use driving data to learn mappings from input features to labels, predictions, or actions. These methods have been applied to driver behavior prediction, lane-change intention recognition, trajectory prediction, and end-to-end driving [100–104]. Driver-style recognition studies show that driving data contain measurable differences in speed choice, acceleration behavior, maneuver aggressiveness, volatility, and lane change behavior [50, 51, 60, 105]. Imitation learning directly learns a policy from expert demonstrations and has been used to reproduce human-like driving behavior [52, 59]. These methods are useful when the goal is to reproduce observed behavior from data, especially when sufficient demonstrations are available.

Reinforcement learning has also been studied for autonomous driving and decision-making because it can optimize long-term behavior through interaction with an environment [61–63]. In reinforcement learning, the reward function defines what behavior should be encouraged. Reward engineering and reward shaping are therefore important parts of the learning process [106]. In driving applications, the reward may include safety, efficiency, comfort, lane keeping, collision avoidance, and progress-related terms. For human-like or personalized driving, the reward function is closely related to driving preference. A reward emphasizing short maneuver time may produce aggressive behavior, while a reward emphasizing smoothness and small steering effort may produce conservative behavior. This relationship between reward design and driving style

motivates inverse reinforcement learning when the desired preference is better observed from demonstrations than manually specified.

Inverse reinforcement learning estimates the reward or cost function that explains expert demonstrations. Ng and Russell introduced the general IRL problem, and later studies developed Bayesian and maximum entropy formulations [64, 65, 107]. Maximum entropy IRL models a distribution over trajectories while matching expert feature expectations, which is useful when expert behavior is not perfectly deterministic [65–67, 108]. This property is relevant to driving because even the same driver may not repeat the exact same lane change under different initial positions, speeds, or traffic conditions. In autonomous driving, IRL has been used to learn human-like driving behavior, social interaction behavior, driving preference, trajectory planning, and personalized lane change behavior [52, 55, 68–72, 109].

Personalized driving research studies how autonomous vehicles can adapt their behavior to different drivers, passengers, or driving styles. Existing studies have considered driver-style classification, personalized adaptive cruise control, personalized trajectory prediction, personalized trajectory planning, and human-like autonomous driving [50, 52, 110–112]. Other studies focus specifically on personalized lane-changing decisions or driver-specific maneuver modeling [53–55]. These studies are relevant because lane change behavior is not uniquely determined by the final lane. The same lane change can be performed with different lateral transition rates, heading responses, yaw-rate profiles, steering efforts, and final lane positions. IRL is useful in this context because it can represent these preferences through learned feature weights rather than only through a directly copied trajectory.

IRL and MPC are complementary for personalized vehicle control. IRL focuses on learning the cost function that explains expert behavior, while MPC uses a cost function to generate control actions under vehicle dynamics and actuator constraints. Learning-based

MPC studies have shown that learned models or learned cost-related components can be incorporated into predictive control while still maintaining the basic MPC structure [11, 25, 87]. For personalized lane change control, this connection makes it possible to learn driver-specific feature weights from demonstrations and use those weights inside a constrained MPC problem [77]. Compared with a purely geometric path-generation approach, the IRL-MPC structure can produce personalized behavior in closed loop, where the controller reacts to the current vehicle state while enforcing the vehicle model and steering constraints.

2.5 Research Gaps and Dissertation Positioning

The reviewed literature shows that MPC provides a strong foundation for autonomous vehicle path tracking, but real-time implementation remains challenging. Existing work has studied explicit MPC, fast solvers, learning-assisted MPC, and event-triggered MPC to reduce computation [17, 23–25, 31–33, 84]. However, many event-triggered MPC studies are validated mainly in simulation or under simplified assumptions. Real vehicles introduce computation delay, actuation delay, localization noise, and model mismatch, which can change the observed relationship between triggering frequency and tracking performance. This dissertation addresses this gap by developing and validating event-triggered MPC for autonomous vehicle path tracking in CARLA simulation, closed-road experiments, and public-road experiments.

A second gap concerns the control behavior between two triggered optimizations. Many event-triggered MPC implementations use the remaining elements of the previous optimal sequence during non-triggered intervals [28, 31, 33]. This strategy reduces computation, but it provides limited feedback correction when the actual vehicle state deviates from the predicted state. Robust MPC and tube-based MPC provide feedback around nominal trajectories [18, 34, 80], but they may introduce additional design and

computation requirements. This dissertation studies a lightweight inter-event feedback strategy that locally approximates the MPC input sequence by using the latest predicted state-input data. The object is to improve feedback correction during non-triggered intervals while preserving the computational advantage of event-triggered MPC [113].

A third gap is related to personalized lane change behavior. Existing lane change trajectory generation methods can produce feasible and smooth paths using polynomial curves, splines, Bezier curves, optimization-based planners, or MPC-based methods [4, 9, 22, 56, 57, 73, 75]. However, a feasible lane change trajectory does not necessarily represent an individual driver's preference. Learning-based methods can model human driving behavior, and IRL can infer interpretable cost functions from demonstrations [52, 55, 64, 65]. Nevertheless, many existing personalized driving studies focus on style classification, decision making, longitudinal behavior, or offline trajectory generation. This dissertation addresses this gap by first learning personalized lane change geometry through a Bezier-curve-based IRL path model and then extending the learned preference into a closed-loop MPC lane change controller.

Therefore, this dissertation is positioned at the intersection of predictive control, event-triggered computation, and learning-based personalization. The event-triggered MPC and inter-event feedback studies address the real-time implementation side of autonomous vehicle lateral control, while the IRL-based path modeling and IRL-MPC studies address the personalization side. Together, these directions connect computationally efficient MPC implementation with driver-specific lane change behavior learning.

CHAPTER THREE

VEHICLE MODELING AND MPC FORMULATION

This chapter presents the vehicle modeling and baseline model predictive control (MPC) formulation used throughout this dissertation. The objective is to establish a unified modeling and control framework for autonomous vehicle lateral motion control, which serves as the foundation for the event-triggered MPC and personalized MPC strategies developed in later chapters. Specifically, this chapter introduces the kinematic and dynamic bicycle models, the tire-force model, the model discretization method, the reference path resampling strategy, and the constrained finite-horizon MPC formulation for path tracking [5–7, 15, 16].

The vehicle motion is described using two coordinate frames: a global frame and a vehicle frame. The global frame is used to describe the vehicle position with respect to the road, map, or reference trajectory. The vehicle frame is attached to the vehicle center of gravity (CG), with its longitudinal axis aligned with the vehicle heading direction. Depending on the operating speed and required prediction fidelity, two vehicle models are used.

3.1 Kinematic Bicycle Model

For low-speed path tracking, the kinematic bicycle model uses the reduced state vector

$$\mathbf{x}_{kin} = \begin{bmatrix} p_x & p_y & \psi \end{bmatrix}^T, \tag{3.1}$$

where p_x and p_y denote the vehicle position in the global frame, and ψ is the vehicle heading angle, which represents the orientation of the vehicle frame with respect to the global frame.

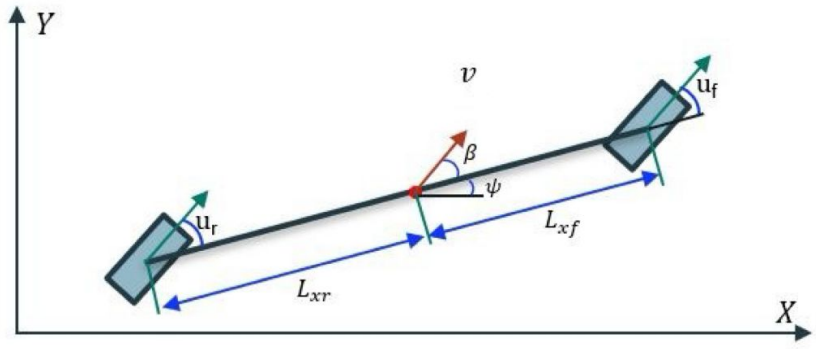

Figure 3.1: Kinematic bicycle model used for low-speed lateral motion prediction.

The kinematic bicycle model approximates the vehicle by combining the front and rear wheels into two virtual wheels located along the vehicle centerline [5, 6]. This model is commonly used for low-speed autonomous driving control because it is computationally simple and avoids the need to estimate lateral tire forces.

As illustrated in Fig. 3.1, the continuous-time kinematic bicycle model is given by

$$\dot{p}_x = V\cos(\psi+\beta), \tag{3.2a}$$

$$\dot{p}_y = V\sin(\psi+\beta), \tag{3.2b}$$

$$\dot{\psi} = \frac{V\cos\beta}{L_{xf}+L_{xr}}\left(\tan u_f - \tan u_r\right), \tag{3.2c}$$

$$\beta = \tan^{-1}\left(\frac{L_{xr}\tan u_f}{L_{xf}+L_{xr}}\right), \tag{3.2d}$$

where V is the vehicle speed at the CG, L_{xf} and L_{xr} are the distances from the CG to the front and rear axles, respectively, β is the body slip angle at the vehicle CG, representing the angle between the vehicle heading direction and the velocity direction at the CG, u_f is the front-wheel steering angle, and u_r is the rear-wheel steering angle. Since the vehicle platforms considered in this dissertation are front-wheel steering vehicles, the rear

steering angle is set to zero, i.e., $u_r = 0$. Therefore, the control input is defined as

$$u = u_f, \tag{3.3}$$

and the yaw-rate equation becomes

$$\dot{\psi} = \frac{V \cos\beta}{L_{xf} + L_{xr}} \tan u_f. \tag{3.4}$$

The kinematic model is especially useful for low-speed path tracking because it provides a simple and numerically robust prediction model without requiring explicit tire-force estimation. This makes it suitable for short-horizon MPC when lateral tire dynamics are not dominant.

3.2 Dynamic Bicycle Model

The dynamic bicycle model captures lateral velocity, yaw rate, and tire-force effects [5–7]. Compared with the kinematic model, the dynamic model is more suitable for higher-speed maneuvers where inertial effects and tire-road interactions become more significant. It is also useful when the control objective includes heading, yaw-rate, or steering-response characteristics.

The state vector of the dynamic bicycle model is defined as

$$\mathbf{x}_{dyn} = \begin{bmatrix} p_x & p_y & v_y & \psi & r \end{bmatrix}^T, \tag{3.5}$$

where p_x and p_y represent the vehicle position in the global frame, v_y is the lateral velocity in the vehicle frame, ψ is the vehicle heading angle, and r is the yaw rate.

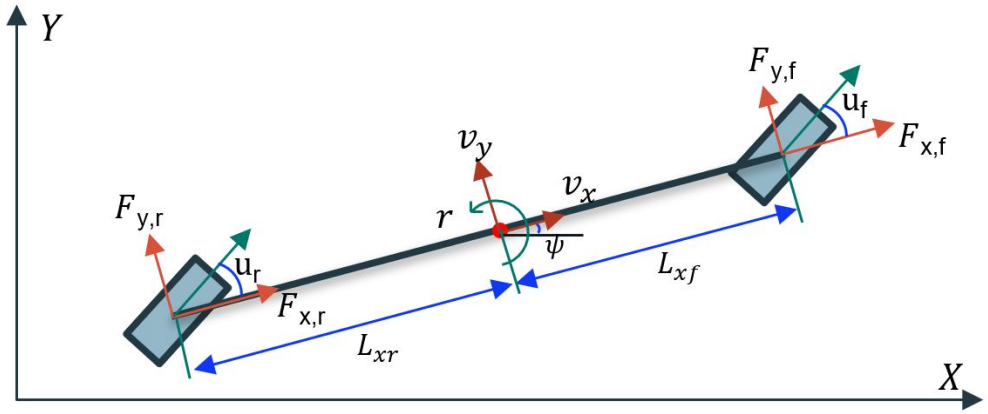

Figure 3.2: Dynamic bicycle model used for higher-speed lateral motion prediction.

As shown in Fig. 3.2, the dynamic bicycle model is formulated at the vehicle CG. The continuous-time equations are

$$\dot{p}_x = v_x \cos\psi - v_y \sin\psi, \tag{3.6a}$$

$$\dot{p}_y = v_x \sin\psi + v_y \cos\psi, \tag{3.6b}$$

$$\dot{v}_y = -v_x r + \frac{1}{m} \sum_{i\in\{f,r\}} F_{y,i}, \tag{3.6c}$$

$$\dot{\psi} = r, \tag{3.6d}$$

$$\dot{r} = \frac{1}{I} \left(L_{xf} F_{y,f} - L_{xr} F_{y,r} \right), \tag{3.6e}$$

where v_x is the longitudinal velocity, m is the vehicle mass, I is the yaw moment of inertia, and $F_{y,f}$ and $F_{y,r}$ are the lateral tire forces at the front and rear wheels, respectively. Since this dissertation focuses on lateral motion control, the longitudinal motion is treated as an external speed command or measured disturbance. Therefore, v_x is measured from the vehicle and treated as a time-varying disturbance during prediction instead of a control variable in the lateral controller.

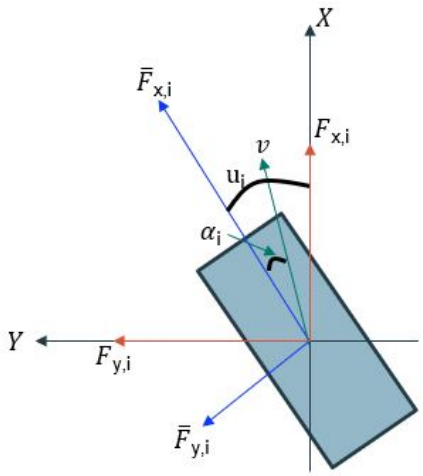

Figure 3.3: Tire-force relationship between the wheel frame and the vehicle frame.

Compared with the kinematic model, this dynamic model includes the lateral velocity and yaw-rate dynamics, making it more suitable for higher-speed path tracking where inertial effects cannot be ignored. However, the model depends on the lateral tire forces $F_{y,f}$ and $F_{y,r}$, which are further determined by the tire model introduced next.

The lateral tire forces in (3.6) are computed using a tire-force model [5]. The tire model relates the wheel-frame tire forces to the vehicle-frame tire forces and approximates the lateral force generated by the tire slip angle.

Let $\bar{F}_{x,i}$ and $\bar{F}_{y,i}$ denote the longitudinal and lateral tire forces in the wheel frame, where $i \in \{f, r\}$ represents the front or rear wheel. Let $F_{x,i}$ and $F_{y,i}$ denote the corresponding forces in the vehicle frame. The coordinate transformation between the wheel frame and vehicle frame is

$$F_{x,i} = \bar{F}_{x,i} \cos u_i - \bar{F}_{y,i} \sin u_i, \tag{3.7a}$$

$$F_{y,i} = \bar{F}_{x,i} \sin u_i + \bar{F}_{y,i} \cos u_i. \tag{3.7b}$$

For front-wheel steering vehicles, u_f is the steering input and $u_r = 0$.

A linear tire model is adopted to approximate the lateral force in the wheel frame:

$$\bar{F}_{y,i} = C_i \alpha_i, \tag{3.8}$$

where C_i is the cornering stiffness and α_i is the slip angle of the corresponding wheel. In implementations where the normal-load or friction distribution is explicitly considered, the lateral force can also be expressed as

$$\bar{F}_{y,i} = C f_i \alpha_i, \tag{3.9}$$

where f_i represents the friction force distribution on the front or rear wheel. The slip angle is computed from the wheel-frame corner velocities:

$$\alpha_i = \tan^{-1}\left(\frac{\bar{V}'_{y,i}}{\bar{V}'_{x,i}}\right). \tag{3.10}$$

The wheel-frame velocities are obtained from the vehicle-frame velocities as

$$\bar{V}'_{x,i} = \bar{V}_{x,i}\cos u_i + \bar{V}_{y,i}\sin u_i, \tag{3.11a}$$

$$\bar{V}'_{y,i} = -\bar{V}_{x,i}\sin u_i + \bar{V}_{y,i}\cos u_i. \tag{3.11b}$$

The corner velocities in the vehicle frame are

$$\bar{V}_{x,f} = v_x, \qquad \bar{V}_{y,f} = v_y + rL_{xf}, \tag{3.12a}$$

$$\bar{V}_{x,r} = v_x, \qquad \bar{V}_{y,r} = v_y - rL_{xr}. \tag{3.12b}$$

Substituting (3.12) into (3.11) and then into (3.10) gives the front and rear slip angles. The resulting lateral tire forces are transformed into the vehicle frame using (3.7) and used in the dynamic bicycle model (3.6).

Together, the dynamic bicycle model and tire-force model describe how the steering input affects the lateral tire forces, and how these forces further determine the vehicle lateral acceleration and yaw acceleration. Therefore, this model allows the MPC

controller to predict how steering commands influence future lateral position, heading angle, lateral velocity, and yaw rate.

3.3 Discretization for MPC Prediction

MPC solves an optimization problem over a discrete prediction horizon. Therefore, the continuous-time vehicle models introduced above must be discretized. In this dissertation, the forward Euler method is used because of its simplicity and computational efficiency for real-time implementation. For a continuous-time vehicle model

$$\dot{\mathbf{x}} = f(\mathbf{x}, u, w), \tag{3.13}$$

the discrete-time prediction model is

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t, u_t, w_t) T_s, \tag{3.14}$$

where T_s is the sampling time and w_t denotes the measured speed input used by the prediction model. For the kinematic model, w_t corresponds to the CG speed V_t, while for the dynamic model it corresponds to the longitudinal velocity $v_{x,t}$. The resulting discrete-time model is denoted compactly as

$$\mathbf{x}_{t+1} = f_d(\mathbf{x}_t, u_t, w_t). \tag{3.15}$$

For the kinematic model, the discrete-time equations are

$$p_{x,k+1} = p_{x,k} + V_k \cos(\psi_k + \beta_k) T_s, \tag{3.16a}$$

$$p_{y,k+1} = p_{y,k} + V_k \sin(\psi_k + \beta_k) T_s, \tag{3.16b}$$

$$\psi_{k+1} = \psi_k + \frac{V_k \cos\beta_k}{L_{xf} + L_{xr}} \left(\tan u_{f,k} - \tan u_{r,k}\right) T_s, \tag{3.16c}$$

$$\beta_k = \tan^{-1}\left(\frac{L_{xr} \tan u_{f,k}}{L_{xf} + L_{xr}}\right). \tag{3.16d}$$

For front-wheel steering vehicles, $u_{r,k} = 0$ and $u_k = u_{f,k}$. These equations are used inside the MPC prediction horizon when the kinematic model is selected.

For the dynamic model, the discrete-time equations are

$$p_{x,k+1} = p_{x,k} + \left(v_{x,k}\cos\psi_k - v_{y,k}\sin\psi_k\right) T_s, \tag{3.17a}$$

$$p_{y,k+1} = p_{y,k} + \left(v_{x,k}\sin\psi_k + v_{y,k}\cos\psi_k\right) T_s, \tag{3.17b}$$

$$v_{y,k+1} = v_{y,k} + \left(-v_{x,k}r_k + \frac{1}{m}\sum_{i\in\{f,r\}} F_{y,i,k}\right) T_s, \tag{3.17c}$$

$$\psi_{k+1} = \psi_k + r_k T_s, \tag{3.17d}$$

$$r_{k+1} = r_k + \frac{1}{I}\left(L_{xf}F_{y,f,k} - L_{xr}F_{y,r,k}\right) T_s. \tag{3.17e}$$

These equations are used inside the MPC prediction horizon when the dynamic model is selected.

3.4 Reference Path Construction and Resampling

The MPC controller requires a short-horizon reference trajectory at every control step. The reference path may come from a recorded GNSS path, a global planner in simulation, or a planned lane-change trajectory. Regardless of the source, raw waypoints are often nonuniformly spaced. Directly using nonuniform waypoints can lead to inconsistent tracking behavior because the prediction horizon would correspond to different physical distances depending on the local waypoint density. Therefore, this section focuses on the reference path resampling procedure used to construct the horizon reference points for MPC.

At each control instant, the current longitudinal velocity v_x is used to determine the desired distance between consecutive reference points in the prediction horizon:

$$d_{\text{ref}} = v_x T_s. \tag{3.18}$$

Since MPC uses a short prediction horizon, v_x is assumed approximately constant within the horizon. The reference sequence is then selected or interpolated such that the reference points are spaced by d_{ref} along the path. For a prediction horizon of length p, the reference position sequence is written as

$$\mathbf{P}_t^{\text{ref}} = \left\{ \left(p_{x,t+1}^{\text{ref}}, p_{y,t+1}^{\text{ref}} \right), \ldots, \left(p_{x,t+p}^{\text{ref}}, p_{y,t+p}^{\text{ref}} \right) \right\}. \tag{3.19}$$

This resampling strategy makes the MPC prediction horizon consistent with the physical motion of the vehicle and improves the numerical conditioning of the path-tracking problem.

When the original reference route contains sharp transitions or non-differentiable segments, an additional path-smoothing step may be applied before resampling. Since the purpose of this chapter is to establish the baseline MPC formulation, the detailed smoothing procedures used in specific simulation or lane-change studies are discussed in the corresponding experimental chapters.

3.5 Baseline Time-Triggered MPC Formulation

The baseline time-triggered MPC formulation used for autonomous vehicle path tracking. MPC has been widely used for constrained vehicle control because it can optimize future vehicle behavior while explicitly handling actuator limits and model constraints [7, 10, 15, 16, 81].

At each sampling instant, the current vehicle state is first measured or estimated. Based on this state and the reference trajectory over the prediction horizon, the optimal control problem (OCP) is solved to obtain an optimal control sequence. Only the first control input is applied to the vehicle, and the same procedure is repeated at the next sampling instant. This receding-horizon implementation provides feedback correction while allowing actuator constraints and model constraints to be explicitly considered.

The numerical solution of the OCP can be obtained using standard nonlinear programming or quadratic programming solvers, depending on the selected prediction model and cost function. Since the focus of this dissertation is on the control framework rather than solver development, the detailed numerical implementation of the solver is not discussed in this chapter.

Let the prediction horizon be p. The predicted state sequence and control sequence are defined as

$$\mathbf{X}_t = \left\{\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_{t+p}\right\}, \tag{3.20}$$

$$\mathbf{U}_t = \left\{u_t, u_{t+1}, \ldots, u_{t+p-1}\right\}. \tag{3.21}$$

The current measured or estimated state is denoted by $\hat{\mathbf{x}}_t$.

The baseline MPC cost function penalizes path-tracking error, steering magnitude, and steering rate. The optimal control problem is formulated as

$$\begin{aligned}
\min_{\mathbf{U}_t} \quad & J = \sum_{k=1}^{p} \left\| p_{x,t+k} - p_{x,t+k}^{\text{ref}} \right\|_{Q_p}^2 + \sum_{k=1}^{p} \left\| p_{y,t+k} - p_{y,t+k}^{\text{ref}} \right\|_{Q_p}^2 \\
& \quad + \sum_{k=0}^{p-1} \|u_{t+k}\|_{Q_u}^2 + \sum_{k=0}^{p-1} \|u_{t+k} - u_{t+k-1}\|_{Q_d}^2 && \text{(3.22a)} \\
\text{s.t.} \quad & \mathbf{x}_t = \hat{\mathbf{x}}_t, && \text{(3.22b)} \\
& \mathbf{x}_{t+k+1} = f_d(\mathbf{x}_{t+k}, u_{t+k}, v_{x,t+k}), \quad 0 \le k \le p-1, && \text{(3.22c)} \\
& u_{\min} \le u_{t+k} \le u_{\max}, \quad 0 \le k \le p-1, && \text{(3.22d)} \\
& \Delta u_{\min} \le u_{t+k} - u_{t+k-1} \le \Delta u_{\max}, \quad 0 \le k \le p-1. && \text{(3.22e)}
\end{aligned}$$

In (3.22a), the first two terms penalize the deviation between the predicted vehicle position and the reference path. The third term penalizes large steering commands, and the fourth term penalizes rapid steering changes to reduce actuator busyness and improve ride comfort. The parameters Q_p, Q_u, and Q_d are tuning weights for path-tracking

accuracy, steering effort, and steering smoothness, respectively. Constraint (3.22b) initializes the predicted trajectory using the current vehicle state. Constraint (3.22c) enforces the discrete-time vehicle prediction model. Constraints (3.22d) and (3.22e) impose steering angle and steering-rate limits. The MPC weights and constraints are selected according to vehicle platform, actuator limits, sampling time, and tracking requirements. The prediction horizon determines the look-ahead distance, while the input and rate constraints ensure that the steering command remains within the physical limits of the drive-by-wire system. Specific parameter values used in simulation and real-world experiments are reported in the corresponding experimental chapters.

After solving (3.22), the first element of the optimal control sequence is applied:

$$u_t = \mathbf{U}_t(1). \tag{3.23}$$

The remaining elements are discarded in the conventional time-triggered MPC implementation. This repeated optimization provides feedback correction at every sampling time but also creates a significant computational burden. This limitation motivates the event-triggered MPC framework developed in the next chapter.

3.6 Switching Prediction Model for Full-Speed Lateral Control

The kinematic and dynamic bicycle models have complementary advantages. The kinematic model is simple and numerically robust at low speeds, while the dynamic model captures lateral velocity and yaw-rate dynamics more accurately at higher speeds. To allow the MPC controller to operate over a wide speed range, a switching prediction model is used. The prediction model is selected according to the measured longitudinal velocity:

$$f_d(\mathbf{x}_t, u_t, v_{x,t}) = \begin{cases} f_{k,d}(\mathbf{x}_t, u_t, v_{x,t}), & v_{x,t} < v_{\text{sw}}, \\ f_{d,d}(\mathbf{x}_t, u_t, v_{x,t}), & v_{x,t} \geq v_{\text{sw}}, \end{cases} \tag{3.24}$$

where $f_{k,d}$ denotes the discretized kinematic bicycle model, $f_{d,d}$ denotes the discretized dynamic bicycle model, and v_{sw} is the switching speed threshold. In the real-world implementation discussed later in this dissertation, $v_{\text{sw}} = 10$ m/s is used to separate low-speed and high-speed operation [27].

The motivation for this switching strategy is twofold. First, at low speeds, the lateral tire-force calculation can become inaccurate or numerically stiff because the slip-angle calculation depends on velocity ratios. The kinematic model avoids this issue and provides stable prediction. Second, at higher speeds, vehicle lateral dynamics and tire forces become more important, so the dynamic model improves prediction accuracy and helps the MPC generate more appropriate steering commands. This switching model therefore provides a practical foundation for full-speed-range autonomous vehicle lateral control.

3.7 Summary

This chapter presented the vehicle modeling and baseline MPC formulation used throughout this dissertation. A kinematic bicycle model was introduced for low-speed prediction, while a dynamic bicycle model with tire-force modeling was introduced for higher-speed lateral dynamics. The continuous-time models were discretized using the forward Euler method for real-time MPC prediction. A reference path resampling strategy was presented to generate uniformly spaced horizon reference points based on the vehicle speed and sampling time. The baseline time-triggered MPC formulation was then defined with path-tracking, steering-effort, and steering-rate penalty terms subject to vehicle dynamics and actuator constraints. Finally, a switching prediction model was introduced to support full-speed-range lateral control. The next chapter builds on this formulation and develops event-triggered MPC to reduce the computational burden of solving the optimal control problem at every sampling time.

CHAPTER FOUR

EVENT-TRIGGERED MPC

MPC is an effective control framework for autonomous vehicle path tracking because it can explicitly consider vehicle dynamics, actuator constraints, and tracking objectives within a finite-horizon optimal control problem. As formulated in Chapter 3, the standard MPC controller computes an optimal steering sequence over a prediction horizon and applies only the first element of this sequence to the vehicle. At the next sampling instant, the optimization problem is solved again using the updated vehicle state.

Although this time-triggered implementation provides a systematic feedback control structure, it also introduces a practical limitation. The optimal control problem (OCP) must be solved at every sampling instant, even when the vehicle is already close to the reference path and the previously computed control sequence remains suitable. This repeated optimization can increase the real-time computational burden of the autonomous driving system. In practice, the onboard computer also simultaneously supports localization, perception, planning, and control. Therefore, reducing unnecessary MPC optimizations is important for real-time implementation.

The objective of this chapter is to develop an event-triggered MPC framework for autonomous vehicle path tracking. Instead of changing the MPC objective function or vehicle prediction model introduced in Chapter 3, the proposed method changes the timing at which the MPC optimization problem is solved. A new optimization is triggered only when the vehicle's lateral offset from the reference path exceeds a predefined threshold or when the previously optimized control sequence is no longer available. When no event is triggered, the controller applies the next element of the previously optimized control sequence.

The main idea is that the MPC optimization does not need to be solved at every control step if the vehicle remains sufficiently close to the reference path. By using an event-triggered mechanism, the controller can reduce the number of OCP solutions while still maintaining the predictive nature of MPC. This chapter presents the event-triggered MPC framework, the lateral-offset-based triggering condition, and the complete event-triggered MPC algorithm. The proposed framework serves as the control update strategy used in the simulation and real-world validation studies presented in the following chapter.

4.1 Event-Triggered MPC Framework

The event-triggered MPC framework is built on the baseline MPC formulation introduced in Chapter 3. Let the vehicle state at the current time step be denoted by $\mathbf{x}_t$, and let the steering control input be denoted by u_t. For a prediction horizon p, the predicted state sequence and control sequence are written as

$$\mathbf{X}_t = \mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_{t+p}, \tag{4.1}$$

$$\mathbf{U}_t = u_t, u_{t+1}, \ldots, u_{t+p-1}. \tag{4.2}$$

In conventional time-triggered MPC, the controller solves the OCP at every sampling instant and applies the first input of the optimal sequence $\mathbf{U}_t$. In event-triggered MPC, the same OCP is solved only when a triggering condition is satisfied.

Let e_t denote the event-triggering signal at time step t. The event-triggered MPC can be represented as

$$e_t = \begin{cases} 1, & \text{if a new MPC optimization is required,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

When $e_t = 1$, the OCP is solved using the current vehicle state and the updated reference path. When $e_t = 0$, the OCP is skipped, and the control input is selected from the optimal control sequence computed at the most recent triggering instant.

Assume that the most recent triggering instant is t_1, and the optimal control sequence computed at that instant is

$$\mathbf{U}_{t_1} = u_{t_1}, u_{t_1+1}, \ldots, u_{t_1+p-1}. \tag{4.4}$$

In the standard event-triggered MPC, if no new event is triggered after t_1, the controller uses the next available element of U_{t_1} rather than solving another OCP. Therefore, the steering command applied by the event-triggered MPC is

$$u_t = \begin{cases} \mathbf{U}_t(1), & \text{if } e_t = 1, \\ \mathbf{U}_{t_1}(k+1), & \text{if } e_t = 0, \end{cases} \tag{4.5}$$

where k is the number of control steps since the most recent triggering instant. In this way, the controller does not simply hold the previous steering input as a constant value. Instead, it applies the next element of the previously optimized sequence. The optimization schedule therefore becomes state-dependent and aperiodic. The controller solves the OCP more frequently when the vehicle deviates from the reference path and less frequently when the vehicle follows the reference path well.

The use of the previously optimized sequence also provides a practical safeguard for implementation. When the OCP is not solved, the controller does not generate an arbitrary steering command. Instead, it applies a command that belongs to the most recent MPC solution. The counter condition prevents this sequence from being used beyond the prediction horizon, so the event-triggered layer does not change the actuator constraints or the structure of the original MPC problem.

4.2 Lateral Offset-Based Triggering Condition

As discussed in Chapter 3, the lateral offset from the reference path is a key quantity for path tracking. In the event-triggered MPC, this offset is further used as the trigger variable to decide whether the MPC problem needs to be solved again.

Let (p_x, p_y) denote the current vehicle position in the global coordinate frame. To compute the lateral offset, two neighboring points on the reference path are selected. Denote these two points as (p_{x1}, p_{y1}) and (p_{x2}, p_{y2}), where they represent the local reference path segment closest to the current vehicle position. The lateral offset d_y is calculated as the perpendicular distance from the vehicle position to the line defined by these two reference points:

$$d_y = \frac{\left|(p_{x2} - p_{x1})(p_{y1} - p_y) - (p_{x1} - p_x)(p_{y2} - p_{y1})\right|}{\sqrt{(p_{x2} - p_{x1})^2 + (p_{y2} - p_{y1})^2}}. \tag{4.6}$$

The value of d_y measures how far the vehicle is from the reference path. A small d_y indicates that the vehicle is close to the desired path, while a large d_y indicates that the vehicle has deviated from the reference path and may require a new MPC optimization.

The event-triggering condition is defined as

$$e_t = \begin{cases} 1, & \text{if } d_y > \sigma \text{ or } k > k_{\max}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.7}$$

where σ is the lateral offset threshold, k is the number of consecutive control steps since the last MPC optimization, and $k_{\max}$ is the maximum number of consecutive steps allowed without solving a new OCP.

The first condition, $d_y > \sigma$, ensures that a new optimization is solved when the vehicle deviates too far from the reference path. The second condition, $k > k_{\max}$, ensures that the controller does not use an old control sequence beyond its valid prediction

horizon. Since the previously optimized sequence has length p, $k_{\max}$ should satisfy

$$k_{\max} \leq p. \tag{4.8}$$

This guarantees that an available control input always exists when the controller decides not to solve a new OCP.

The threshold σ is the main calibration parameter of the event-triggered MPC. A smaller σ makes the controller trigger more frequently and behave closer to time-triggered MPC. A larger σ reduces the number of optimization calls, but it may allow larger path-tracking error before a new OCP is solved. Therefore, σ determines the trade-off between computational efficiency and tracking accuracy.

The lateral-offset condition also serves as a feedback correction mechanism. If model mismatch, external disturbances, or accumulated tracking error cause the vehicle to deviate from the reference path, the condition $d_y > \sigma$ triggers a new OCP using the current vehicle state. Thus, the controller can skip unnecessary optimizations when the vehicle follows the path well, while still re-optimizing once the tracking error becomes significant.

4.3 Standard Event-Triggered MPC Algorithm

The standard event-triggered MPC procedure is summarized in Algorithm 4.1. This algorithm follows the event-triggered structure described above. At each control step, the controller first updates the counter, computes the lateral offset, and evaluates the triggering condition. If an event is triggered, a new MPC problem is solved. Otherwise, the controller applies the next element of the optimal control sequence computed at the previous triggering instant.

The inputs of Algorithm 4.1 include the predicted state sequence $\hat{\mathbf{X}}_t$, the counter k, and the state and control sequences obtained at the previous triggering instant. Line 3 updates the counter that records how many consecutive steps the MPC optimization has

Algorithm 4.1: Event-triggered MPC for autonomous vehicle path tracking

1. **Initialization:** $k \leftarrow 0$
2. **Procedure** eMPC($\hat{\mathbf{x}}_t$, k, $\mathbf{U}_{t_1}$, $\mathbf{X}_{t_1}$)
3. $k \leftarrow k+1$
4. Compute d_y using (4.6)
5. $e_t \leftarrow$ computing (4.7)
6. **if** $e_t = 1$ **then**
7. $\quad k \leftarrow 0$
8. $\quad (\mathbf{X}_t, \mathbf{U}_t) \leftarrow$ solving the MPC OCP in Chapter 3
9. $\quad u_t \leftarrow \mathbf{U}_t(1)$
10. $\quad U_{t_1} \leftarrow U_t$
11. $\quad \mathbf{X}_{t_1} \leftarrow \mathbf{X}_t$
12. **else**
13. $\quad u_t \leftarrow U_{t_1}(k+1)$
14. **end**
15. **return** u_t, k, $\mathbf{U}_{t_1}$, $\mathbf{X}_{t_1}$

not been triggered. Lines 4–5 compute the lateral offset and evaluate the event-triggering condition. If an event is triggered, Lines 7–11 reset the counter, solve a new MPC problem, apply the first element of the new optimal control sequence, and store the newly predicted state and control sequences. Otherwise, Line 13 directly applies the next element from the previous optimal control sequence. Finally, the steering command, the counter, and the stored sequences are returned for the next control step.

4.4 Event-Triggered MPC with Linear Inter-Event Control

The standard event-triggered MPC reduces computation by reusing the optimal control sequence computed at the most recent triggering instant. As shown in Algorithm 4.1, when no event is triggered, the steering command is selected as $\mathbf{U}_{t_1}(k+1)$. This strategy avoids solving a new OCP and still uses information from the latest MPC prediction. However, during the inter-event period, the control input is mainly determined

by the predicted trajectory generated at the previous triggering instant. Therefore, the current vehicle state is not explicitly used to correct the steering command until a new event is triggered.

This limitation becomes important when the actual vehicle response differs from the predicted motion. Such a mismatch may be caused by model uncertainty, localization error, actuator delay, road disturbance, or accumulated tracking error. Although the lateral-offset condition can eventually trigger a new OCP when the error becomes large, the shifted input may not provide enough feedback correction before the next triggering instant. To reduce this open-loop behavior during the inter-event period, this section introduces a linear-based inter-event control method.

The basic idea is to use the latest MPC solution to construct a local approximation of the MPC control policy. When an event is triggered, the MPC solver provides an optimal predicted state sequence and an optimal control sequence. These state-control pairs describe how the MPC would steer the vehicle along the predicted horizon. Instead of discarding this information after the first control input is applied, the proposed method uses it to identify a lightweight feedback law. During the following inter-event steps, this feedback law is evaluated using the current measured or estimated vehicle state, so the controller can update the steering command without solving a new OCP.

The inter-event control input is written as

$$u_t = KP(\hat{\mathbf{x}}_t), \tag{4.9}$$

where K is the gain vector identified from the most recent MPC solution, and $P(\hat{\mathbf{x}}_t)$ is a feature vector constructed from the current vehicle state. The feature vector is defined as

$$P(\hat{\mathbf{x}}_t) = \begin{bmatrix} 1 & \hat{p}_{x,t} & \hat{p}_{y,t} & \sin(\hat{\psi}_t) & \cos(\hat{\psi}_t) & \hat{p}_{x,t}^2 & \hat{p}_{y,t}^2 \end{bmatrix}^T. \tag{4.10}$$

The corresponding gain vector is

$$K = \begin{bmatrix} k_0 & k_1 & k_2 & k_3 & k_4 & k_5 & k_6 \end{bmatrix}. \tag{4.11}$$

The feature vector in (4.10) is selected to provide a compact local description of the vehicle state. The constant term allows the mapping to include a steering offset. The position terms $\hat{p}_{x,t}$ and $\hat{p}_{y,t}$ describe the local vehicle location. The heading angle is represented by $\sin(\hat{\psi}_t)$ and $\cos(\hat{\psi}_t)$ instead of directly using $\hat{\psi}_t$, which avoids discontinuity caused by angle wrapping. The quadratic position terms $\hat{p}_{x,t}^2$ and $\hat{p}_{y,t}^2$ provide a simple way to capture local nonlinear behavior while keeping the mapping low dimensional. Since the gain is updated online from a short MPC horizon, the feature dimension is intentionally kept small to reduce computation and avoid overfitting to a short predicted trajectory.

When an event is triggered at time step t, the MPC problem is solved and returns the optimal predicted state sequence

$$\mathbf{X}_t = \left\{ \mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+p-1} \right\}, \tag{4.12}$$

and the corresponding optimal control sequence

$$\mathbf{U}_t = \left\{ u_t, u_{t+1}, \ldots, u_{t+p-1} \right\}. \tag{4.13}$$

For each predicted state in $\mathbf{X}_t$, the feature vector is computed using (4.10). These feature vectors are stacked into the feature matrix

$$\mathbf{P}_{\text{opt}} = \begin{bmatrix} P(\mathbf{x}_t)^T \\ P(\mathbf{x}_{t+1})^T \\ \vdots \\ P(\mathbf{x}_{t+p-1})^T \end{bmatrix}. \tag{4.14}$$

The corresponding control sequence is written in vector form as

$$\mathbf{U}_{\text{opt}} = \begin{bmatrix} u_t & u_{t+1} & \cdots & u_{t+p-1} \end{bmatrix}^T. \tag{4.15}$$

The gain vector K is selected so that the linear mapping $KP(\mathbf{x})$ reproduces the MPC control inputs along the predicted horizon as closely as possible. This is formulated as a least-squares problem:

$$K = \arg\min_K \left\| \mathbf{P}_{\text{opt}} K^T - \mathbf{U}_{\text{opt}} \right\|_2^2. \tag{4.16}$$

The solution is computed using the Moore–Penrose pseudoinverse:

$$(K)^T = \mathbf{P}_{\text{opt}}^{\dagger} \mathbf{U}_{\text{opt}}, \tag{4.17}$$

where $(\cdot)^{\dagger}$ denotes the pseudoinverse. This avoids explicitly inverting $\mathbf{P}_{\text{opt}}^T \mathbf{P}_{\text{opt}}$, which is useful when the feature matrix is close to singular or when some features are locally correlated over the short prediction horizon.

After K is obtained, it is stored and used until the next event is triggered. During the inter-event period, the current state $\hat{\mathbf{x}}_t$ is measured or estimated at each control step, the feature vector $P(\hat{\mathbf{x}}_t)$ is recomputed, and the steering command is generated using (4.9). Therefore, although K is learned from the predicted MPC trajectory, the control input is still updated using the current feedback state during non-triggered steps. The OCP is not solved at these steps, but the control loop itself still runs at the original sampling rate.

The resulting event-triggered control law with linear inter-event control is

$$u_t = \begin{cases} \mathbf{U}_t(1), & \text{if } e_t = 1, \\ KP(\hat{\mathbf{x}}_t), & \text{if } e_t = 0. \end{cases} \tag{4.18}$$

Compared with the standard event-triggered MPC control law in (4.5), the difference appears only when $e_t = 0$. The standard method applies the shifted input $\mathbf{U}_{t_1}(k+1)$, while

the linear method computes the steering input from the current vehicle state. Therefore, the linear method can be viewed as a local feedback approximation of the MPC policy during the inter-event period.

The linear controller does not replace the original MPC optimization. It is only used as a short-horizon inter-event controller between two triggering instants. The event-triggering condition and the counter condition remain active. Once the vehicle deviates from the reference path or the inter-event period becomes too long, a new OCP is solved and the K matrix is updated using the latest MPC solution. In this way, the method preserves the constrained optimization capability of MPC while adding a low-cost feedback correction during non-triggered steps.

The steering command generated by the linear controller must still satisfy the actuator constraints used in the MPC formulation. Therefore, the steering angle and steering-rate constraints are enforced after computing (4.9). Specifically,

$$u_t \leftarrow \mathrm{clip}(u_t, u_{\min}, u_{\max}), \tag{4.19}$$

$$\Delta u_t = u_t - u_{t-1}, \tag{4.20}$$

$$\Delta u_t \leftarrow \mathrm{clip}(\Delta u_t, \Delta u_{\min}, \Delta u_{\max}), \tag{4.21}$$

and the final steering command is updated as

$$u_t \leftarrow u_{t-1} + \Delta u_t. \tag{4.22}$$

This implementation keeps the inter-event steering command consistent with the actuator limits used by the original MPC controller.

Example 1 *Fig. 4.1 illustrates an example of the proposed event-triggered MPC with linear inter-event control applied during the lane change scenario. At a certain time step, an MPC is triggered, and an optimal state sequence is obtained. The blue-cross indicate*

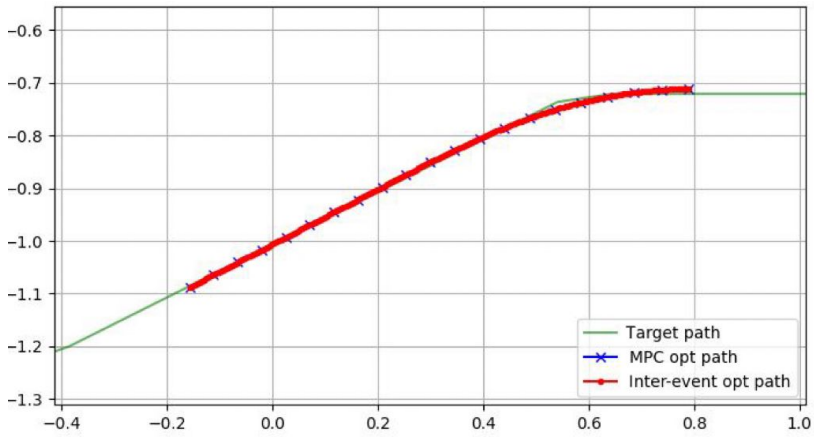

Figure 4.1: Demonstration of the proposed inter-event control during lane change maneuver.

the predicted optimal trajectory points generated by the MPC over the prediction horizon. The red-dot line denotes the predicted vehicle trajectory if the linear feedback gain K^* from the optimal input and state trajectories is used. It can be seen that this trajectory closely follows the original MPC prediction. Even in high curvature segment, the linear control can still keep the vehicle close to the optimal path without solving a new optimal control problem. This example demonstrates the feasibility and tracking accuracy of the proposed inter-event control, particularly in short prediction intervals and locally smooth regions of the path.

4.5 Event-Triggered MPC with Linear Inter-Event Algorithm

The event-triggered MPC with linear inter-event control is summarized in Algorithm 4.2. Compared with the standard event-triggered MPC in Algorithm 4.1, the main difference is the control input used when no event is triggered. Instead of applying the next input from the previous optimal sequence, the proposed method computes the steering command using the current vehicle state and the gain matrix K.

Algorithm 4.2: Event-Triggered MPC with Linear Inter-Event Control

Initialization: $k \leftarrow 0$; $K \leftarrow 0$
Procedure eMPC/K($\hat{\mathbf{x}}_t$, k, $\mathbf{U}_{t_1}$, u_{pre}, K)
$k \leftarrow k+1$
Compute d_y using (4.6)
$e_t \leftarrow$ computing (4.7)
if $e_t = 1$ **then**
    $k \leftarrow 0$
    $(\mathbf{X}_t, \mathbf{U}_t) \leftarrow$ solving the MPC OCP in Chapter 3
    $u_t \leftarrow \mathbf{U}_t(1)$
    $\mathbf{U}_{\text{opt}} \leftarrow \mathbf{U}_t$
    $\mathbf{P}_{\text{opt}} \leftarrow$ computing (4.14)
    $K \leftarrow$ computing (4.17)
    $\mathbf{U}_{t_1} \leftarrow \mathbf{U}_t$
else
    $P \leftarrow P(\hat{\mathbf{x}}_t)$
    $u_t \leftarrow$ computing (4.9)
    $u_t \leftarrow \text{clip}(u_t, u_{\min}, u_{\max})$
    $\Delta u_t \leftarrow u_t - u_{\text{pre}}$
    $\Delta u_t \leftarrow \text{clip}(\Delta u_t, \Delta u_{\min}, \Delta u_{\max})$
    $u_t \leftarrow u_{\text{pre}} + \Delta u_t$
end
$u_{\text{pre}} \leftarrow u_t$
return u_t, k, u_{pre}, K

The inputs of Algorithm 4.2 include the current vehicle state, the counter, the optimal sequence obtained at the previous triggering instant, the latest gain matrix K, and the previously applied steering command. Lines 3–5 update the counter, compute the lateral offset, and evaluate the event-triggering condition. If an event is triggered, Lines 7–13 solve a new MPC problem, apply the first element of the new optimal sequence, construct the feature matrix from the predicted states, and update K using least squares. If no event is triggered, Lines 15–20 compute the inter-event steering command using the current state and the stored gain matrix. The steering angle and steering-rate constraints are then enforced before the command is applied.

4.6 Summary

This chapter developed the event-triggered MPC framework used in this dissertation. Based on the vehicle model and MPC formulation introduced in Chapter 3, the proposed method keeps the same OCP structure but changes when the OCP is solved. Instead of updating the optimal control sequence at every sampling instant, a new optimization is performed only when it is needed according to the event-triggering condition.

The lateral offset from the reference path was used as the triggering variable. When the offset is larger than the threshold σ, the controller solves a new MPC problem using the current vehicle state. A counter k is also included to prevent the controller from using an old control sequence beyond the prediction horizon. Therefore, the triggering rule depends on both the current tracking error and the remaining valid portion of the previous optimal sequence.

Two inter-event control strategies were presented. In the standard event-triggered MPC, the controller applies the shifted control sequence when no event is triggered. In the enhanced framework, the latest MPC solution is used to identify a local linear feedback

gain K through least squares. During the inter-event period, the control input is then generated as $u_t = KP(\hat{\mathbf{x}}_t)$ using the current vehicle state. This allows the controller to retain feedback correction between two MPC optimizations while avoiding the cost of solving a new OCP.

The threshold σ, the maximum counter $k_{\max}$, and the inter-event gain K determine the main behavior of the controller. A smaller σ makes the controller closer to the conventional time-triggered MPC, while a larger σ can reduce the number of OCP solutions at the cost of potentially larger tracking error. The linear feedback further reduces the open-loop nature of conventional event-triggered MPC by updating the control command from the current state during non-triggered steps. The next chapter evaluates these methods through simulation and real-world path-tracking experiments.

CHAPTER FIVE

EXPERIMENTAL EVALUATION OF EVENT-TRIGGERED MPC

This chapter experimentally evaluates the event-triggered model predictive control (eMPC) framework developed in Chapter 4. The previous chapter presented the mathematical formulation of the controller, including the vehicle prediction model, the MPC optimal control problem, the event-triggering condition, and the algorithmic procedure for reusing the previously optimized input sequence when a new optimization is not required. Based on that formulation, this chapter focuses on experimental validation. The objective is to investigate whether the proposed event-triggered strategy can reduce the real-time computational demand of MPC while maintaining acceptable lateral path-tracking performance.

The experimental evaluation is organized into three parts. The first part uses the CARLA simulator to validate the proposed controller in a high-fidelity but repeatable simulation environment. This simulation study provides a controlled setting in which conventional time-triggered MPC (tMPC) and event-triggered MPC can be compared under identical vehicle, path, and controller parameters. The CARLA route includes straight driving, a lane-change maneuver, and turning maneuvers, which makes it possible to analyze how the event-triggered mechanism behaves under different levels of lateral motion demand.

The second part implements the controller on a full-size autonomous vehicle platform. The real-vehicle validation is performed in both a closed test track and a public road environment. The closed test track experiment provides a repeatable and controlled real-world setting, while the public-road experiment evaluates the controller under more practical driving conditions, including high-speed cruising, low-speed turns, and

stop-and-go maneuvers. These experiments are important because real-time vehicle implementation introduces practical effects that are often simplified or ignored in simulation, such as optimization latency, actuator response delay, localization noise, speed variation, and model mismatch.

The third part evaluates an enhanced event-triggered MPC strategy with linear based inter-event feedback control on a Quanser QCar2 scale autonomous vehicle platform. This experiment examines whether the open-loop input reuse in conventional eMPC can be improved by adding a lightweight feedback correction during non-triggered intervals. After presenting all simulation, full-size vehicle, and scale-vehicle results, this chapter provides an overall discussion that summarizes the main experimental findings and implementation insights.

5.1 Simulation Evaluation in CARLA

5.1.1 Simulation Environment and Reference Route

The simulation experiments are conducted in CARLA, a high-fidelity autonomous driving simulator. CARLA provides a virtual driving environment that includes road networks, vehicle dynamics, simulated sensors, and interfaces for autonomous driving algorithm development. Compared with a simple MATLAB or Simulink simulation, CARLA provides a more realistic evaluation environment because the vehicle motion is generated by the simulator's physics engine and the controller interacts with the simulated vehicle through a closed-loop control structure.

The implemented CARLA control framework is shown in Fig. 5.1. The overall control system consists of a global route generation module, a longitudinal controller, and a lateral controller. The global route provides the desired path for the vehicle to follow. The longitudinal controller is implemented using a PID controller to track the desired vehicle speed. Since this dissertation focuses on lateral motion control, the longitudinal controller is kept the same for all simulation cases. The lateral controller is implemented

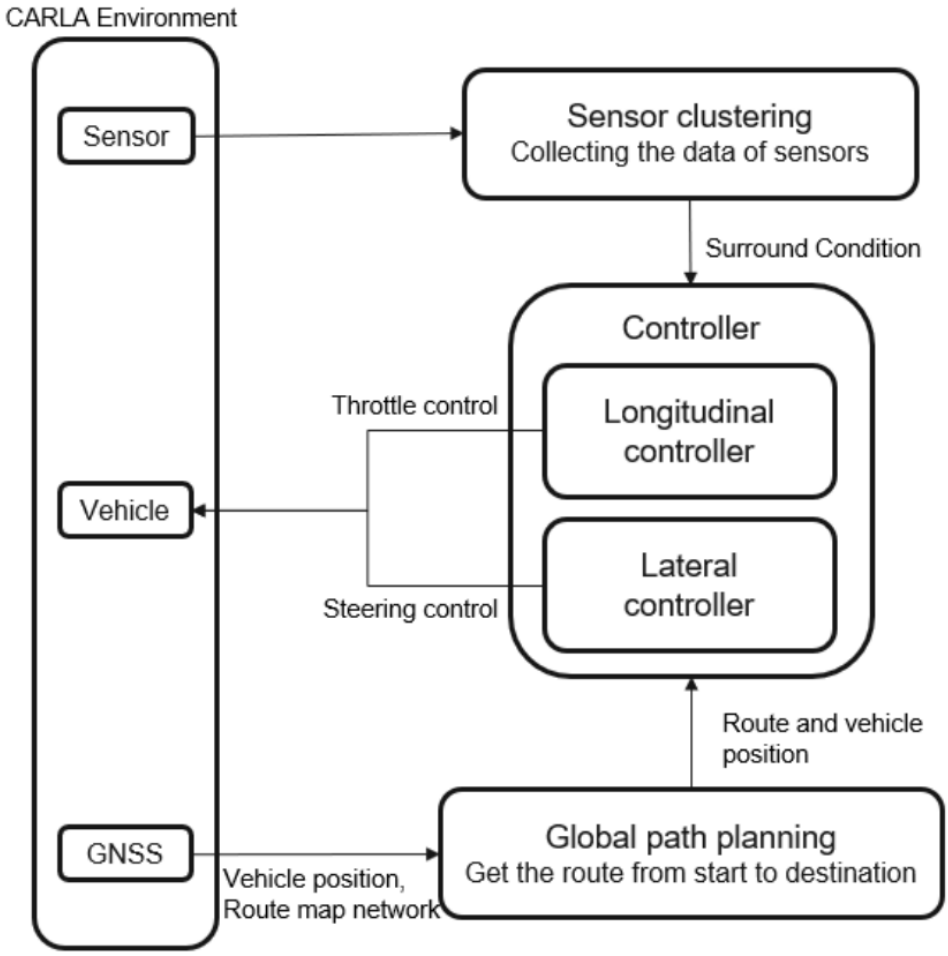

Figure 5.1: CARLA simulation framework for evaluating the MPC-based path tracking controller.

using either conventional tMPC or the proposed eMPC. Therefore, any performance difference between the tested controllers is mainly due to the lateral control strategy rather than differences in longitudinal speed control.

The reference route used in the simulation is shown in Fig. 5.2. The route includes three representative types of driving maneuvers: straight driving, a lane-change maneuver, and turning maneuvers. Straight driving is useful for evaluating the computational saving of eMPC because the vehicle lateral dynamics change slowly and the previously

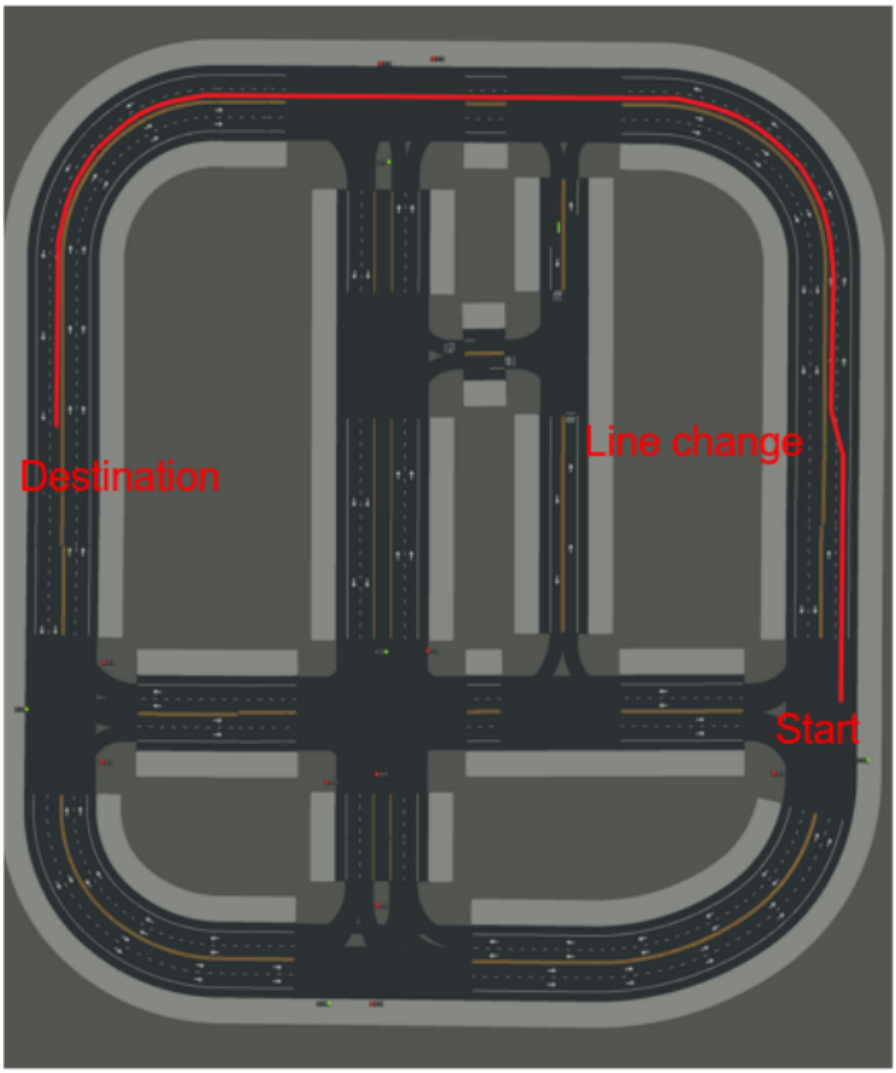

Figure 5.2: Reference route used in the CARLA simulation.

optimized input sequence can remain valid for a longer period. Lane changes and turns are more demanding because the vehicle must generate larger and faster steering corrections. Therefore, these maneuvers are useful for examining the tracking performance degradation caused by reducing the number of MPC optimizations.

The raw reference route generated by the route planner may contain non-smooth segments, especially during lane changes. A direct connection between the initial and final lane-change points can result in discontinuous curvature, which is not suitable for vehicle motion control. To avoid this problem, a Bezier curve is used to smooth the lane-change segment, as shown in Fig. 5.3. In this simulation, the lane-change segment is represented by a fifth-order Bezier curve determined by six control points, P_0 to P_5. The start and end points of the Bezier curve are constrained to match the start and end points of the lane-change maneuver. The first- and second-order derivatives at both ends are used to shape the curve and ensure a smooth transition:

$$\begin{aligned} B(0) &= P_0, & B(1) &= P_5, \\ \dot{B}(0) &= 5(P_1 - P_0), & \dot{B}(1) &= 5(P_5 - P_4), \\ \ddot{B}(0) &= 20(P_0 - 2P_1 + P_2), & \ddot{B}(1) &= 20(P_3 - 2P_4 + P_5). \end{aligned} \tag{5.1}$$

Here, $B(0)$ and $B(1)$ define the geometric endpoints of the smoothed lane-change path, while $\dot{B}(0)$, $\dot{B}(1)$, $\ddot{B}(0)$, and $\ddot{B}(1)$ determine the tangent and curvature-related properties at the two ends. By selecting the control points according to these boundary relationships, the originally linear lane-change segment is replaced with a smooth and differentiable curve. The smoothed reference path provides a more realistic lane-change command and reduces unnecessary sharp steering demands.

After smoothing, the reference path is resampled using the path-resampling procedure introduced in Section 3.4. This allows the smoothed CARLA route to be converted into the discrete reference waypoints required by the MPC prediction horizon.

In the CARLA simulation, the target vehicle speed is set to 10 $\mathrm{m/s}$. Four controllers are compared. The first controller is conventional time-triggered MPC, denoted as tMPC, which solves the optimal control problem at every sampling instant. The other three controllers are event-triggered MPC with different lateral error thresholds,

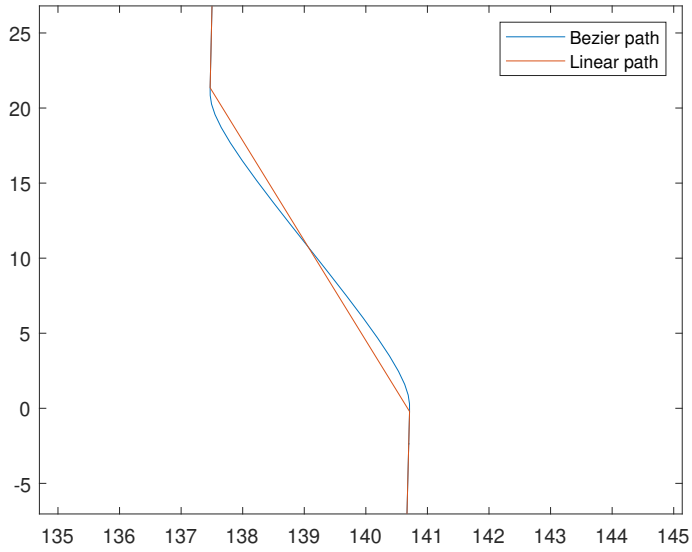

Figure 5.3: Bezier-curve-based smoothing for the lane-change segment in the CARLA reference route.

denoted as eMPC(0.01), eMPC(0.02), and eMPC(0.03). Here, the number in parentheses represents the event-trigger threshold σ used in the triggering condition. A smaller threshold triggers the MPC optimization more frequently and is expected to give better tracking accuracy. A larger threshold allows the controller to reuse the previous optimal control sequence for a longer period and is expected to reduce computation more aggressively.

The same MPC prediction model, cost function weights, sampling time, prediction horizon, steering constraints, and steering-rate constraints are used for all controllers. This ensures that the comparison is fair and that the observed differences are caused by the event-triggered mechanism rather than by controller retuning.

The vehicle dynamic model parameters used in the CARLA simulation are listed in Table 5.1. Some parameters, such as the vehicle mass and wheelbase, can be obtained

directly from CARLA, while the remaining parameters are identified offline so that the simplified bicycle model used by MPC can match the short-horizon behavior of the high-fidelity CARLA vehicle.

To identify the unknown vehicle parameters, a genetic algorithm (GA) is used as an offline parameter estimation method. First, input and state sequences are collected by running the vehicle in CARLA. For a candidate parameter vector $\hat{\theta}$, the bicycle model is simulated using the collected steering input sequence, and the simulated state response is compared with the state sequence obtained from CARLA. The parameter estimation problem is formulated by minimizing the mismatch between the CARLA state trajectory and the bicycle-model response:

$$\begin{aligned} \min_{\hat{\theta}} \quad & \sum_{k=1}^{K} \left\| X_k - \hat{X}_k \right\|_1 \\ \text{s.t.} \quad & \theta_{\min} \leq \hat{\theta} \leq \theta_{\max}, \\ & L_{xf} + L_{xr} = L, \end{aligned} \tag{5.2}$$

where X_k denotes the state collected from CARLA, $\hat{X}_k$ denotes the state predicted by the bicycle model, and $\hat{\theta} = [\mu, I, L_{xf}, L_{xr}, C]^T$ is the parameter vector to be identified. Since the parameter estimation is conducted offline, the GA computation does not affect the real-time performance of the MPC controller. The purpose of this step is to obtain a simplified prediction model that is sufficiently accurate over the short MPC prediction horizon.

The GA-based parameter estimation procedure is summarized in Algorithm 5.1. The algorithm starts by collecting the steering input sequence and the corresponding vehicle state sequence from CARLA. Then, a population of candidate parameter vectors is initialized. For each generation, the bicycle model is simulated using each candidate parameter vector, and the fitness value is computed using the mismatch between the

Algorithm 5.1: Genetic Algorithm for Vehicle Parameter Estimation

1. Collect input and state sequences (U, X) by running CARLA
2. Initialize the population with candidate parameter vectors $\hat{\theta}$
3. **for** *each generation* **do**
4. | **for** *each candidate parameter vector* $\hat{\theta}$ **do**
5. | | Compute $fit(\hat{\theta}) = \sum_{k=1}^{K} \left\| X_k - \hat{X}_k \right\|_1$
6. | **end**
7. | $a \leftarrow \min fit(\hat{\theta})$
8. | **if** *a converges* **then**
9. | | Break
10. | **else**
11. | | Select parent candidates with smaller fitness values
12. | | Generate the next population by crossover;
13. | **end**
14. **end**
15. **return** optimal parameter vector $\hat{\theta}$

Table 5.1: Vehicle dynamic model parameters used in CARLA simulation.

m (kg)	1265	I (kg·m^2)	6481	L (m)	2.6
L_{xf} (m)	2.3	L_{xr} (m)	0.3	μ (-)	0.289
C (deg^{-1})	3.07				

simulated response and the CARLA measurement. Candidate parameter vectors with smaller fitness values are selected to generate the next population through crossover. This process is repeated until the fitness value converges or the maximum number of generations is reached.

The MPC parameters used for the CARLA simulation are summarized in Table 5.2. These parameters are kept identical for tMPC and all eMPC cases. The only parameter that differs among the eMPC controllers is the event-trigger threshold σ.

Table 5.2: MPC parameters used in CARLA simulation.

p	10	Q_u	35	$u_{\min}$ (rad)	-0.97
T_s (ms)	200	Q_d	30	$\Delta u_{\max}$ (rad)	0.15
Q_p	2	$u_{\max}$ (rad)	0.97	$\Delta u_{\min}$ (rad)	-0.15

The evaluation metrics include maximum lateral tracking error, root mean square error (RMSE), and MPC trigger frequency. The maximum error reflects the worst-case tracking deviation, the RMSE evaluates the overall tracking accuracy, and the trigger frequency measures the computational demand relative to tMPC.

5.1.2 Simulation Results and Discussion

The overall path-tracking results in CARLA are summarized in Table 5.3. As expected, tMPC achieves the smallest maximum error and RMSE because it solves the optimal control problem at every sampling instant. This means that tMPC always computes a new optimal steering sequence based on the most recent vehicle state. However, this performance comes with the highest computational cost, since the optimization problem must be solved during every control cycle.

In contrast, eMPC reduces the number of MPC optimizations by solving the optimal control problem only when the event-triggering condition is satisfied. When no event is triggered, the controller applies the next element of the previously optimized input sequence. As shown in Table 5.3, increasing the event-trigger threshold reduces the trigger frequency. For eMPC(0.01), the trigger frequency is 74.12% of tMPC. For eMPC(0.02), the trigger frequency decreases to 62.22%. For eMPC(0.03), the trigger frequency further decreases to 55.21%. Therefore, with the largest tested threshold, eMPC avoids nearly half of the MPC optimizations compared with tMPC.

Table 5.3: Overall path-tracking performance in CARLA simulation.

Controller	tMPC	eMPC(0.01)	eMPC(0.02)	eMPC(0.03)
Max error (m)	0.095	0.156	0.180	0.200
RMSE (m)	0.042	0.050	0.059	0.069
Trigger frequency (%)	100	74.12	62.22	55.21

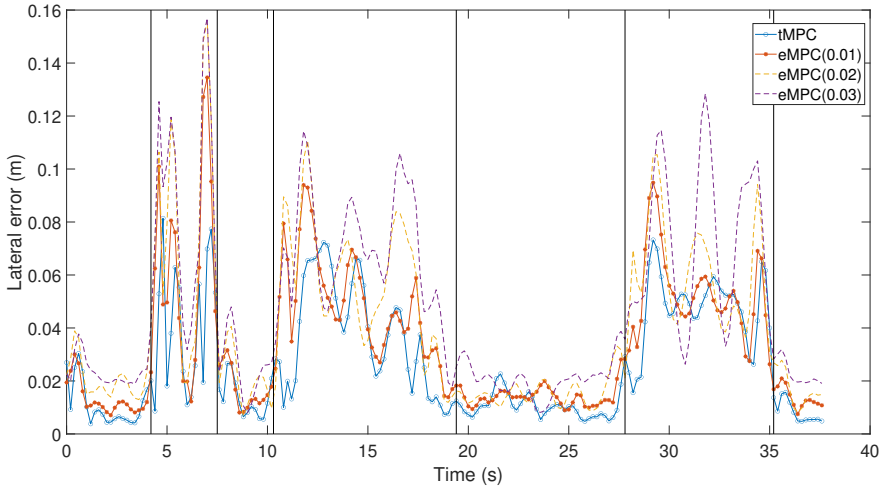

Figure 5.4: Lateral tracking errors of tMPC and eMPC controllers in CARLA simulation.

The reduction in trigger frequency leads to a moderate increase in tracking error. The maximum lateral error increases from 0.095 m for tMPC to 0.156 m for eMPC(0.01), 0.180 m for eMPC(0.02), and 0.200 m for eMPC(0.03). The RMSE follows the same trend, increasing from 0.042 m for tMPC to 0.069 m for eMPC(0.03). Nevertheless, the tracking degradation remains acceptable. Even for eMPC(0.03), the maximum lateral error remains no larger than 0.20 m. This result demonstrates that the proposed event-triggered mechanism can substantially reduce computational demand while maintaining reasonable path-tracking accuracy.

Table 5.4: Maximum lateral error under different driving maneuvers in CARLA.

Controller	Straight	Lane Change	Turn
tMPC	0.069	0.086	0.095
eMPC(0.01)	0.083	0.156	0.131
eMPC(0.02)	0.104	0.180	0.156
eMPC(0.03)	0.138	0.200	0.185

The lateral error curves are shown in Fig. 5.4. The lane-change maneuver occurs between approximately 4.2 s and 7.5 s, while the turning maneuvers occur between approximately 10.2 s and 19.5 s and between 28 s and 35.2 s. The remaining portions correspond to straight driving. The figure shows that the lateral tracking error is generally small for all controllers. The larger error peaks occur mainly during the lane-change and turning segments because these maneuvers require more aggressive steering actions and faster state changes. This observation suggests that the required MPC update frequency is maneuver dependent. During straight driving, a lower optimization frequency may be sufficient, while during lane changes and turns, more frequent optimization is beneficial for maintaining tracking accuracy.

To further analyze the effect of different driving maneuvers, the reference route is divided into straight driving, lane change, and turning. The maximum lateral errors are listed in Table 5.4, and the RMSE values are listed in Table 5.5. These tables provide a more detailed view of how eMPC performs under different lateral motion demands.

The straight-driving results show the strongest computational advantage of eMPC. Since the vehicle is mostly aligned with the reference path during straight driving, the lateral error changes slowly and the previously optimized steering sequence remains effective. Therefore, eMPC can skip many optimization calls without causing a large

Table 5.5: RMSE under different driving maneuvers in CARLA.

Controller	Straight	Lane Change	Turn
tMPC	0.018	0.045	0.052
eMPC(0.01)	0.025	0.072	0.062
eMPC(0.02)	0.031	0.089	0.074
eMPC(0.03)	0.040	0.099	0.089

increase in tracking error. For example, even with eMPC(0.03), the straight-driving RMSE is 0.040 m, which is still small in absolute value.

The lane-change results show larger performance degradation for eMPC. The lane-change maneuver involves a rapid lateral transition, which requires the steering command to change more actively. Reusing the previous optimal input sequence for too long can introduce a mismatch between the predicted trajectory and the actual vehicle motion. Therefore, the maximum error and RMSE increase as the threshold increases. The turn results follow a similar trend because turning maneuvers also require continuous steering corrections.

The CARLA results show that the triggering threshold directly determines the trade-off between computational efficiency and tracking accuracy. A larger threshold reduces the number of optimization calls, but it also increases the tracking error. A smaller threshold preserves tracking accuracy but reduces computation less aggressively. Therefore, the event-trigger threshold should be treated as a calibration parameter. In applications where computational resources are limited and moderate tracking error is acceptable, a larger threshold may be preferred. In applications where tracking accuracy is more critical, a smaller threshold should be used.

Figure 5.5: Full-size autonomous vehicle platform used for real-vehicle experimental validation.

5.2 Real-Vehicle Experimental Setup

5.2.1 Autonomous Vehicle Platform

The real-vehicle experiments are conducted using a full-size autonomous vehicle platform, shown in Fig. 5.5. The vehicle is equipped with a drive-by-wire system, a Polynav 2000P GNSS-inertial system, a Calmcar front-view camera with lane detection capability, and a dSPACE Autera computing unit. The drive-by-wire system allows the controller to command steering inputs electronically. The GNSS-inertial system provides the vehicle position, heading, and motion information required for feedback control. The computing unit runs the real-time control algorithm and communicates with the vehicle actuation system.

In these experiments, a previously recorded GNSS path is used as the reference trajectory. This design removes the influence of an online path planner and allows the comparison to focus on the lateral tracking controllers. The vehicle's longitudinal motion is not controlled by the proposed MPC controller. Instead, the proposed controller focuses on lateral motion control through steering. This separation is consistent with the

formulation in Chapter 3, where the longitudinal speed is treated as a measured input or disturbance for the lateral prediction model.

The distances from the vehicle center of gravity to the front and rear axles are 1.2 m and 1.65 m, respectively. These parameters are used in the vehicle prediction model. The real-vehicle platform introduces several practical challenges that are not fully present in simulation. First, the state feedback from the localization system contains measurement noise. Second, the vehicle model used in MPC cannot perfectly capture the full vehicle dynamics, tire behavior, actuator dynamics, and road conditions. Third, solving the MPC optimization problem requires non-negligible computation time, which introduces a delay between state measurement and command execution. These practical factors make real-vehicle validation necessary for evaluating the true performance of the proposed eMPC framework.

5.2.2 Testing Environments

Two real-vehicle testing environments are considered. The first environment is a closed test track located in Plymouth, Michigan, as shown in Fig. 5.6. A closed test track is used because it provides a controlled and repeatable testing condition. The controlled test environment removes the influence of regular traffic participants, allowing the same reference path to be repeated under comparable conditions for a fair comparison among different controllers. The reference path contains large curved segments and turning regions, which are useful for evaluating the tracking capability of the lateral controller.

The second environment is a public-road route in Plymouth, Michigan, as shown in Fig. 5.7. Compared with the closed test track, the public-road route is more challenging and more representative of daily driving. The route includes high-speed cruising, low-speed turns, and stop-and-go maneuvers. The vehicle speed varies from 0 to approximately 17 $\mathrm{m/s}$ depending on traffic conditions. Since the speed is influenced by

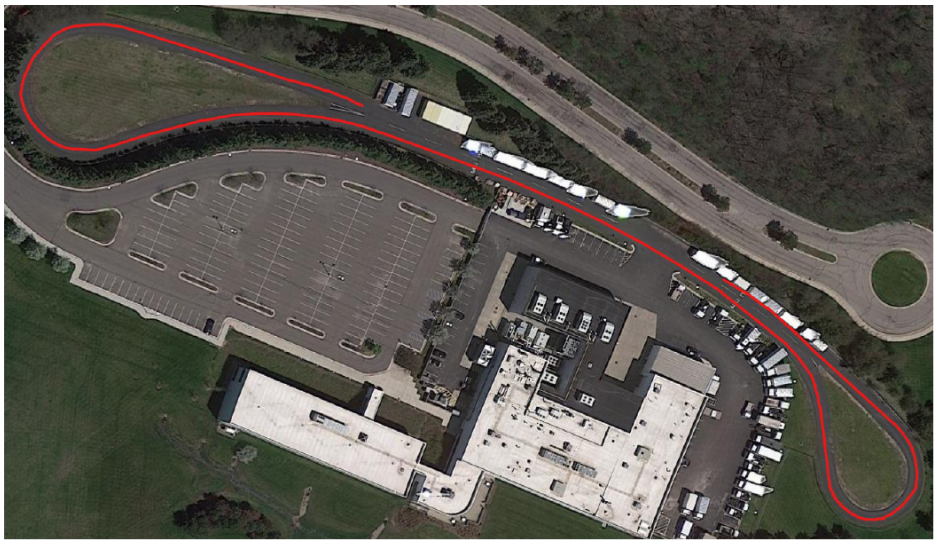

Figure 5.6: Closed-test-track environment used for real-vehicle validation.

the actual traffic flow, the test contains more uncertainty than the closed-track experiment. This makes the public-road test an important step for evaluating the practical feasibility and robustness of the proposed controller.

The use of both environments provides a comprehensive experimental evaluation. The closed test track experiment emphasizes repeatability and fair controller comparison, while the public-road experiment emphasizes practical implementation under realistic traffic conditions. Together, these two tests show whether the proposed eMPC framework can work not only in controlled experiments but also in more complex real-world scenarios.

5.2.3 Controller Implementation

To ensure a fair comparison, the same MPC parameters, cost weights, actuator constraints, and steering-rate constraints are used for both tMPC and eMPC. The parameters are summarized in Table 5.6. The prediction horizon is set to $p = 10$, and the

Figure 5.7: Public-road testing route used for real-vehicle validation.

Table 5.6: MPC parameters used in the real-vehicle experiments.

p	10	Q_u	35	$u_{min}\ (rad)$	-0.97
$T_s\ (ms)$	200	Q_d	30	$\Delta u_{max}\ (rad)$	0.15
Q_P	2	$u_{max}\ (rad)$	0.97	$\Delta u_{min}\ (rad)$	-0.15

sampling time is $T_s = 200$ ms. The path-tracking weight is $Q_p = 2$, the steering effort weight is $Q_u = 35$, and the steering smoothness weight is $Q_d = 30$. The steering angle is constrained between -0.97 rad and 0.97 rad, and the steering-rate constraint is limited between -0.15 rad and 0.15 rad per sampling step.

Both controllers use the same MPC formulation and calibration parameters. The only difference is whether the optimal control problem is solved periodically or according to the event-triggering condition. Therefore, the experimental comparison focuses on the effect of event-triggering on tracking accuracy and optimization frequency.

For the closed test track experiment, tMPC is compared with eMPC using different event-trigger thresholds. For the public-road experiment, tMPC is compared with eMPC(0.05). In addition, an eMPC case with a 100 Hz controller update-rate limit is tested. This additional test is motivated by a practical implementation issue: when no event is triggered, eMPC can generate commands very quickly because it does not need to solve a new optimization problem. However, sending commands too frequently may exceed the meaningful response capability of the steering actuator. Therefore, the 100 Hz update-rate limit is introduced to examine how limiting the command update rate affects tracking performance.

5.2.4 Real-Vehicle Experimental Results - Closed Test Track

The closed-test-track results are shown in Figs. 5.8 and 5.9. Fig. 5.8 compares the reference trajectory and the actual trajectories generated by tMPC and eMPC with different event-trigger thresholds. In general, all controllers are able to guide the vehicle through the entire reference path safely. The actual trajectories follow the reference trajectory closely over most of the path. The zoom-in view shows that the main differences among controllers occur near the turning regions, where larger steering corrections are required.

The lateral tracking errors are shown in Fig. 5.9. The error peaks mainly appear near the beginning and ending turning regions of the path. This is reasonable because the vehicle experiences larger curvature demand in these regions. Compared with straight or gently curved segments, turning regions are more sensitive to model mismatch, localization noise, actuator delay, and speed variation. Therefore, they are the most challenging parts of the closed-track test.

The numerical results are summarized in Table 5.7. The tMPC controller has a maximum lateral error of 0.5833 m and an RMSE of 0.1386 m. In comparison, the three

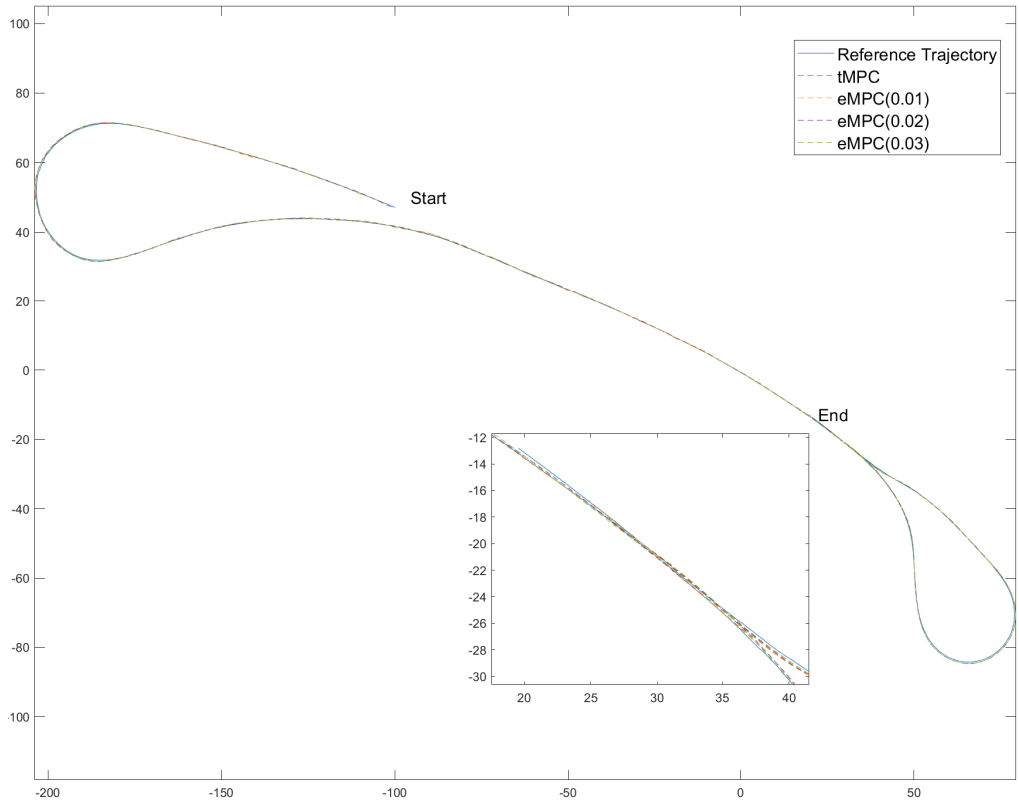

Figure 5.8: Tracking trajectories of tMPC and eMPC controllers in the closed-test-track experiment.

eMPC controllers achieve lower RMSE values. The RMSE values are 0.1056 m for eMPC(0.01), 0.1030 m for eMPC(0.02), and 0.1058 m for eMPC(0.03). Among these three event-triggered cases, eMPC(0.02) provides the lowest RMSE. The maximum errors of all three eMPC cases are also lower than that of tMPC.

These closed-test-track results are important because they differ from the trend observed in CARLA simulation. In the simulation study, tMPC gives the best tracking accuracy because it solves the optimal control problem at every sampling instant. However, in the real-vehicle experiment, eMPC achieves better tracking accuracy than tMPC. This difference is mainly caused by real-time computation delay. In the physical

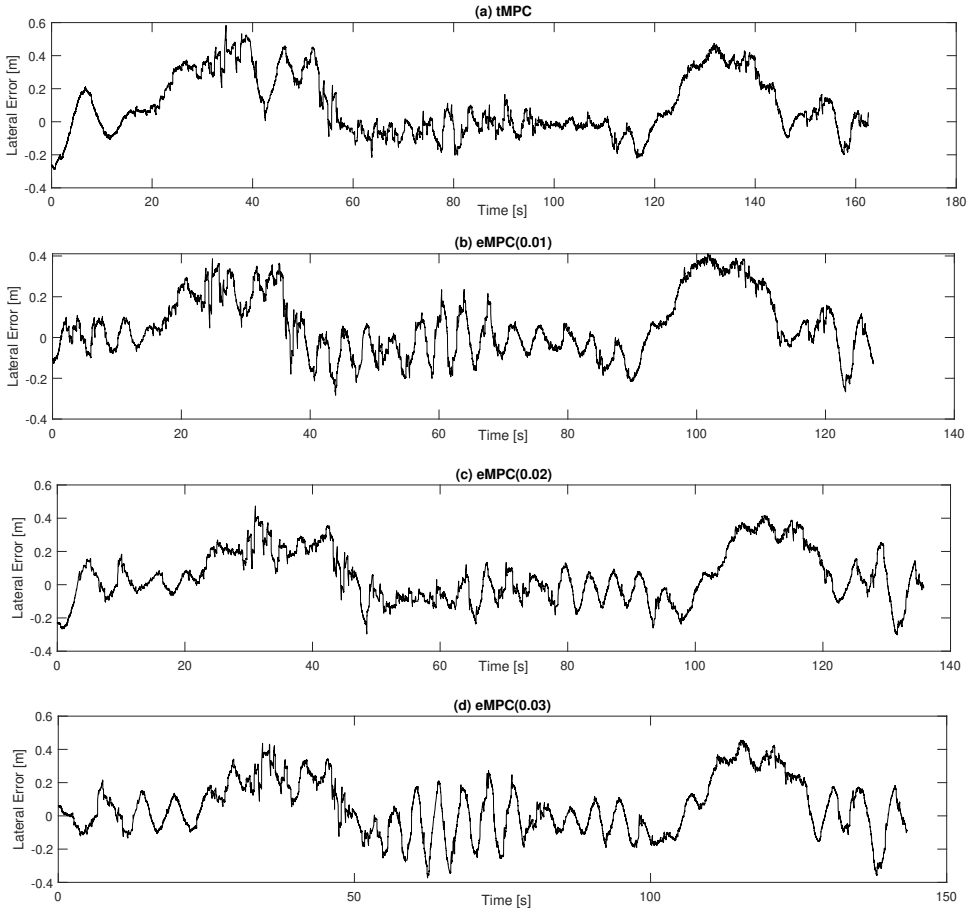

Figure 5.9: Lateral tracking errors of tMPC and eMPC controllers in the closed-test-track experiment.

vehicle, solving the MPC optimization problem takes a non-negligible amount of time. Based on the experimental data, the average MPC computation time is estimated to be approximately 75 ms. This creates a delay between the state used to compute the optimal steering command and the time when that steering command is actually applied to the vehicle.

Table 5.7: Closed-test-track experimental results.

Controller	tMPC	eMPC(0.01)	eMPC(0.02)	eMPC(0.03)
Control counts	2146	2581	3160	3894
Event counts	2146	1789	1847	1870
Driving time (s)	162.38	128.35	135.64	143.15
Average speed (m/s)	3.35	4.59	4.35	3.94
Max error (m)	0.5833	0.4108	0.4736	0.4559
RMSE (m)	0.1386	0.1056	0.1030	0.1058

For tMPC, this optimization delay occurs at every control step because a new optimal control problem is solved at every sampling instant. As a result, the steering command sent to the vehicle may correspond to a slightly outdated vehicle state. For eMPC, however, when no event is triggered, the controller does not solve a new optimization problem. Instead, it directly shifts the previous optimal sequence and applies the next control input. This operation requires negligible computation time compared with solving the optimization problem. Therefore, during non-triggered steps, eMPC can provide more timely steering commands. This timing advantage can compensate for the tracking degradation that would normally be expected from skipping optimization calls.

Another factor is that the different test runs have different driving times and average speeds, as shown in Table 5.7. Because the vehicle longitudinal motion is not controlled by the MPC controller, the average speed is not exactly identical across tests. This can affect lateral tracking difficulty. Nevertheless, the results still show a clear trend that eMPC is practically feasible and can provide tracking performance comparable to or better than tMPC while avoiding unnecessary optimization calls.

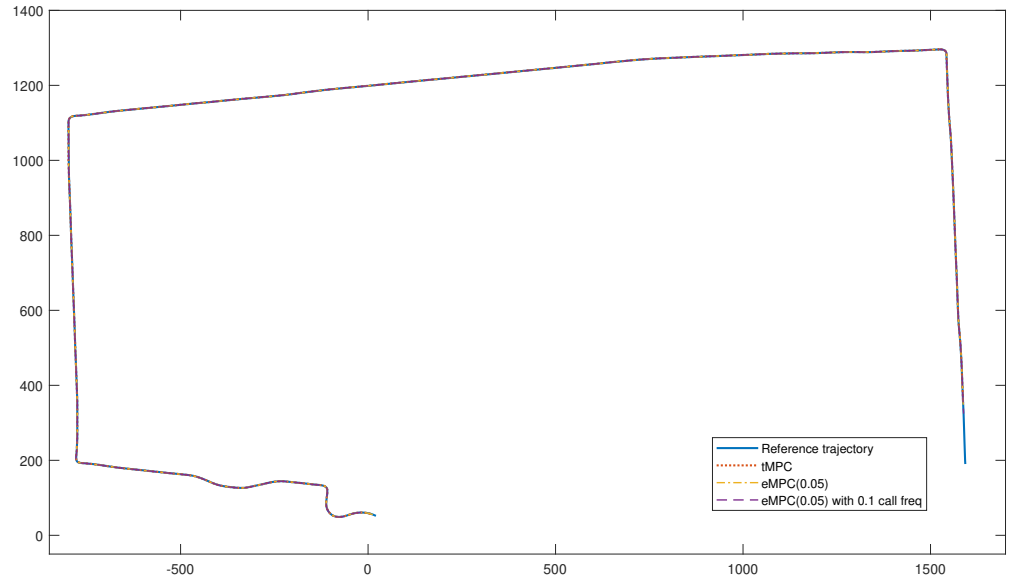

Figure 5.10: Tracking trajectories of tMPC and eMPC controllers in the public-road experiment.

5.2.5 Real-Vehicle Experimental Results - Public Road Results

The public-road experiment further evaluates the proposed controller under more realistic driving conditions. Unlike the closed test track experiment, the public-road test includes real traffic effects, speed changes, high-speed cruising, low-speed turns, and stop-and-go maneuvers. The route is therefore more representative of practical autonomous driving scenarios. The tracking trajectories are shown in Fig. 5.10. Overall, all controllers are able to follow the reference path without large offsets. The trajectories remain close to the reference path for most of the route, demonstrating that both tMPC and eMPC can be implemented successfully in the public-road environment.

The lateral tracking errors are shown in Fig. 5.11. The major error peaks occur around the turning regions of the route. This is consistent with the closed-track test and the CARLA simulation. Turns are more challenging because they require larger steering

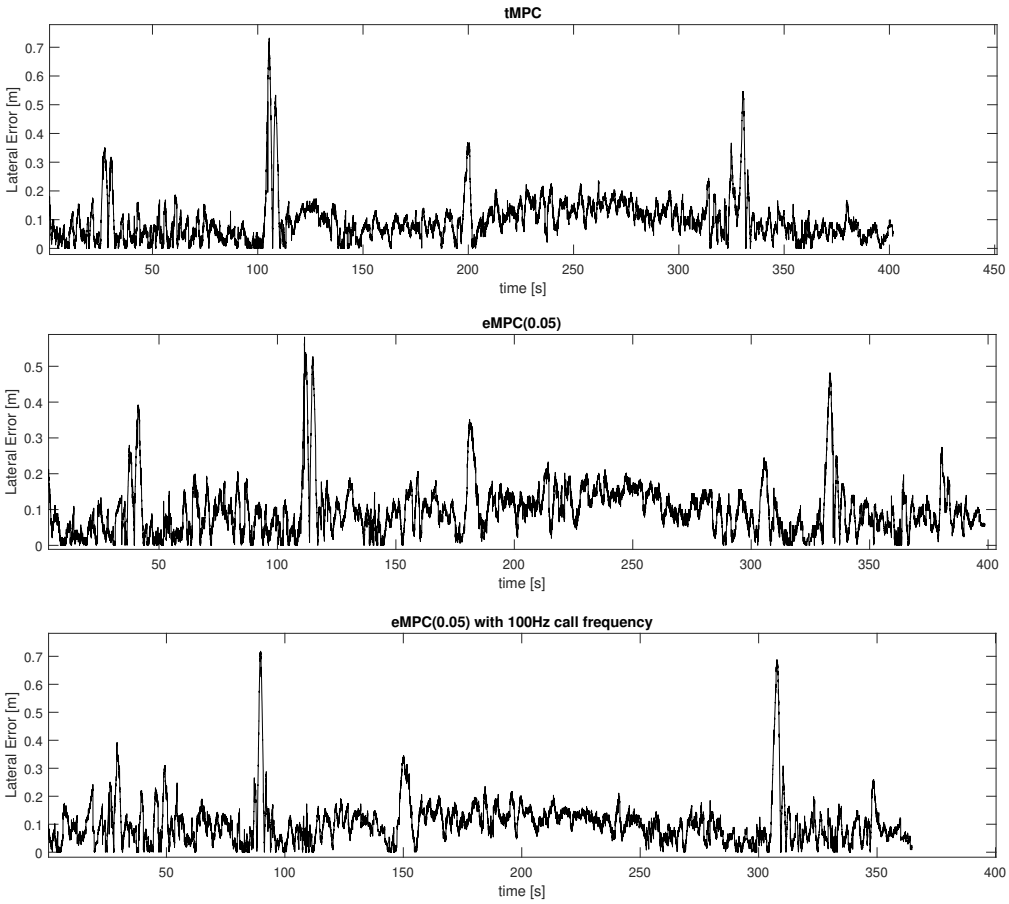

Figure 5.11: Lateral tracking errors of tMPC and eMPC controllers in the public-road experiment.

commands and are more sensitive to the mismatch between the prediction model and the actual vehicle response. In addition, the public-road test includes varying speed and traffic flow, which further increases the tracking difficulty.

The numerical results are summarized in Table 5.8. The tMPC controller has a maximum lateral error of 0.732 m and an RMSE of 0.1253 m. The standard eMPC(0.05) reduces the maximum lateral error to 0.582 m and slightly reduces the RMSE to

Table 5.8: Public-road experimental results.

Controller	tMPC	eMPC(0.05)	eMPC(0.05) with 100 Hz call freq.
Control counts	4120	20289	11710
Event counts	4120	4278	3328
Trigger frequency (Hz)	10.25	10.73	9.12
Driving time (s)	401.9	398.7	364.8
Average speed (m/s)	12.82	12.87	14.14
Max error (m)	0.732	0.582	0.718
RMSE (m)	0.1253	0.1232	0.1281

0.1232 m. These results show that eMPC remains effective in the public-road environment and can provide tracking performance comparable to or better than tMPC.

The public-road experiment also reveals an important implementation issue. When no event is triggered, eMPC can generate new steering commands very frequently because it only needs to shift the previously optimized control sequence. For eMPC(0.05), the experiment records 20,289 control commands within 398.7 seconds, while only 4,278 of these commands require solving a new MPC optimization problem. This means that most control commands are generated without solving a new optimization problem. From the perspective of computational efficiency, this is beneficial. However, from the perspective of actuator implementation, this can create a new challenge.

When the controller sends commands too frequently, the interval between two consecutive steering commands may become shorter than the response time of the steering actuator. In the eMPC(0.05) test, the interval between two non-triggered control commands can be as short as 5 ms or less. Such a high command update rate may not provide enough time for the steering actuator to physically respond. Therefore, simply increasing the command update frequency does not always improve real-vehicle performance. The controller update rate must be compatible with the actuator dynamics.

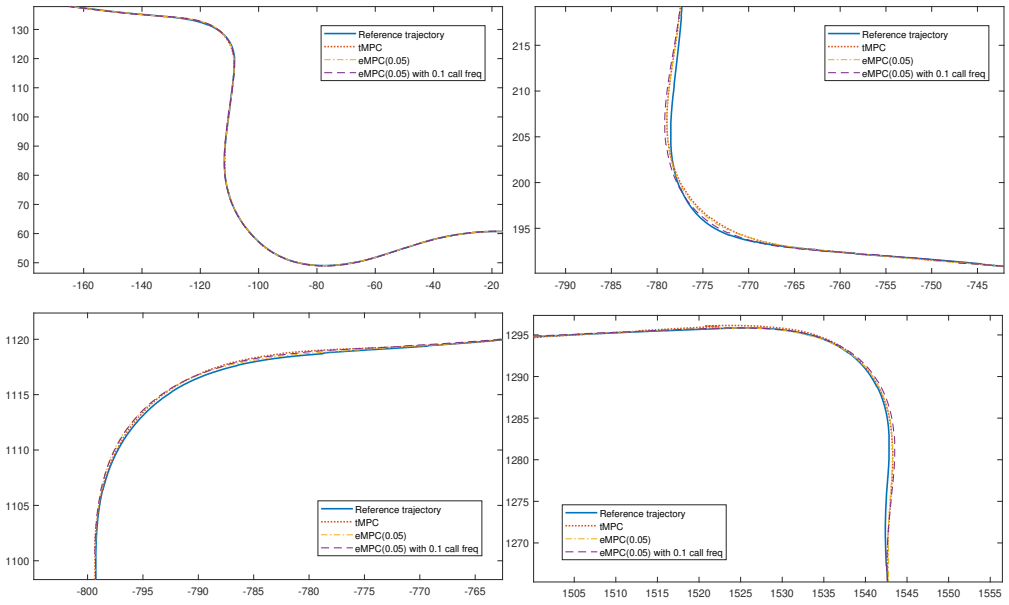

Figure 5.12: Zoom-in view of tracking trajectories during turning maneuvers in the public-road experiment.

To address this issue, an additional eMPC(0.05) test is conducted with a 100 Hz controller update-rate limit. In this setting, when no event is triggered, the controller waits until the time interval between two control commands reaches at least 10 ms. This prevents the controller from sending steering commands faster than 100 Hz. As shown in Table 5.8, the 100 Hz update-rate limit reduces the event-trigger frequency to 9.12 Hz. The maximum error becomes 0.718 m, and the RMSE becomes 0.1281 m. Although these values are slightly worse than the standard eMPC(0.05), they remain comparable to tMPC.

Fig. 5.12 provides a zoom-in view of the turning regions. The larger lateral errors observed in Fig. 5.11 correspond to these turns. This confirms that the most challenging portions of the public-road route are the curved segments rather than the straight cruising segments. The result also suggests that future event-triggered strategies could benefit from maneuver-dependent or speed-dependent thresholds. For example, a larger threshold may

be used during straight cruising to reduce computation, while a smaller threshold may be used during turning maneuvers to improve tracking accuracy.

5.3 Scale-Vehicle Validation with Linear Inter-Event Control

The CARLA and full-size vehicle experiments validate the effectiveness of event-triggered MPC for reducing unnecessary online optimizations. However, the conventional eMPC strategy still has one limitation: during non-triggered intervals, the controller executes the previously optimized input sequence in an open-loop manner. In other words, when the event-triggering condition is not satisfied, the newest state feedback is not explicitly used to update the steering command. This strategy is computationally efficient, but it can allow tracking error to accumulate during the inter-event period, especially when model mismatch, actuator delay, localization noise, or external disturbances are present.

To further examine this issue, this section evaluates an enhanced event-triggered MPC strategy with linear based inter-event feedback control on a Quanser QCar2 scale autonomous vehicle platform. This controller is denoted as eMPC/K. The main idea is to retain the event-triggered optimization structure while replacing the purely open-loop inter-event input replay with a lightweight feedback correction. After an MPC optimization is triggered, the optimized state and input sequences are used to construct a linear mapping between the predicted state and the corresponding control input. During the following non-triggered steps, this linear mapping computes the steering command using the current vehicle state. Therefore, eMPC/K preserves the computational advantage of eMPC while introducing feedback correction between two MPC optimizations.

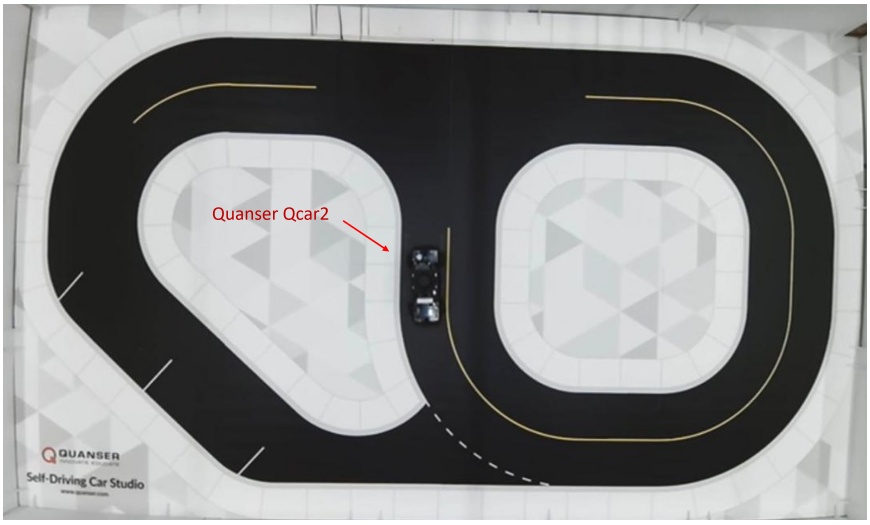

Figure 5.13: Quanser QCar2 scale autonomous vehicle platform used for validating eMPC with linear based inter-event control.

5.3.1 QCar2 Experimental Setup

The QCar2 platform used in this experiment is shown in Fig. 5.13. The vehicle is equipped with an onboard NVIDIA Jetson AGX Orin computer and a sensor suite for autonomous driving experiments. The vehicle pose is estimated by fusing wheel odometry, IMU, and 2D LiDAR measurements through an extended Kalman filter. All control commands are computed onboard and sent to the drive and steering actuators through the vehicle low-level interface. In these experiments, the longitudinal speed is kept constant, and the proposed controller is evaluated for lateral path tracking.

The proposed eMPC/K controller is compared with conventional eMPC under three event-trigger thresholds, $\sigma \in \{0.02, 0.04, 0.06\}$, and three longitudinal speeds, $v \in \{0.20, 0.26, 0.32\}$ m/s. Time-triggered MPC is also tested as a baseline. For each controller setting, the QCar2 completes 10 laps, and the reported values are computed as the mean $\pm$ standard deviation. This repeated-lap design reduces the effect of run-to-run

Table 5.9: MPC parameters used in the QCar2 eMPC/K experiments.

N	6	Q_u	1	$u_{\min}$ (rad)	-0.97
dt (ms)	500	Q_d	1	$\Delta u_{\max}$ (rad)	0.15
Q_p	20	$u_{\max}$ (rad)	0.97	$\Delta u_{\min}$ (rad)	-0.15

variation and provides a more reliable comparison among tMPC, conventional eMPC, and eMPC/K.

5.3.2 Linear Inter-Event Control Implementation

The MPC parameters used in the QCar2 experiments are listed in Table 5.9. The prediction horizon is set to $N = 6$, and the sampling time is $dt = 500$ ms. This gives a 3 s prediction window while keeping the optimization problem small enough for onboard real-time computation. The steering input and steering-rate constraints are selected to ensure feasible and smooth steering commands.

For the QCar2 implementation, the same MPC formulation and parameters are used for tMPC, conventional eMPC, and eMPC/K. The only difference is the control update rule after each sampling instant. In tMPC, the OCP is solved at every step. In conventional eMPC, the OCP is solved only when the triggering condition is satisfied; otherwise, the next element of the previously optimized input sequence is applied. In eMPC/K, the OCP is also solved only at triggered instants, but the resulting state-input sequence is additionally used to update the local feedback matrix K. During non-triggered steps, the steering command is computed from this linear feedback law using the current measured state.

This implementation is most relevant when the event-trigger threshold is large, because the controller then spends more time between two MPC optimizations. In that

Table 5.10: Time-triggered MPC tracking performance on QCar2 over 10 laps.

v (m/s)	RMSE (m)	Mean error (m)	Event freq. (Hz)	Events/lap
0.32	0.0144 ± 0.0019	0.0128 ± 0.0014	18.76 ± 0.72	1039.57 ± 23.6
0.26	0.0142 ± 0.0012	0.0112 ± 0.0009	19.36 ± 1.36	1351.18 ± 45.4
0.20	0.0151 ± 0.0011	0.0118 ± 0.0011	19.61 ± 0.96	1676.75 ± 62.4

case, the linear feedback law provides a lightweight correction mechanism without requiring an immediate OCP solve.

5.3.3 Tracking Performance

Figs. 5.14–5.16 show representative path-tracking results at $v = 0.32$ m/s under different event-trigger thresholds. The black curve denotes the reference path, while the blue and red curves represent the trajectories under conventional eMPC and eMPC/K, respectively. As the threshold increases, both controllers solve MPC less frequently, and the trajectory envelope becomes wider. However, the eMPC/K trajectories remain close to the reference path and are generally comparable to or better than those of conventional eMPC. This indicates that the linear based inter-event controller helps reduce error accumulation between MPC updates.

The time-triggered MPC baseline is summarized in Table 5.10. The RMSE remains within a narrow range from 0.0142 m to 0.0151 m across the tested speeds, showing stable tracking accuracy. However, the event frequency remains close to 19–20 Hz, resulting in a large number of MPC optimizations per lap. This confirms that tMPC provides consistent tracking performance but requires dense online optimization.

The tracking accuracy comparison between conventional eMPC and eMPC/K is summarized in Table 5.11. Across the tested thresholds and speeds, eMPC/K achieves tracking performance comparable to conventional eMPC and often improves both RMSE

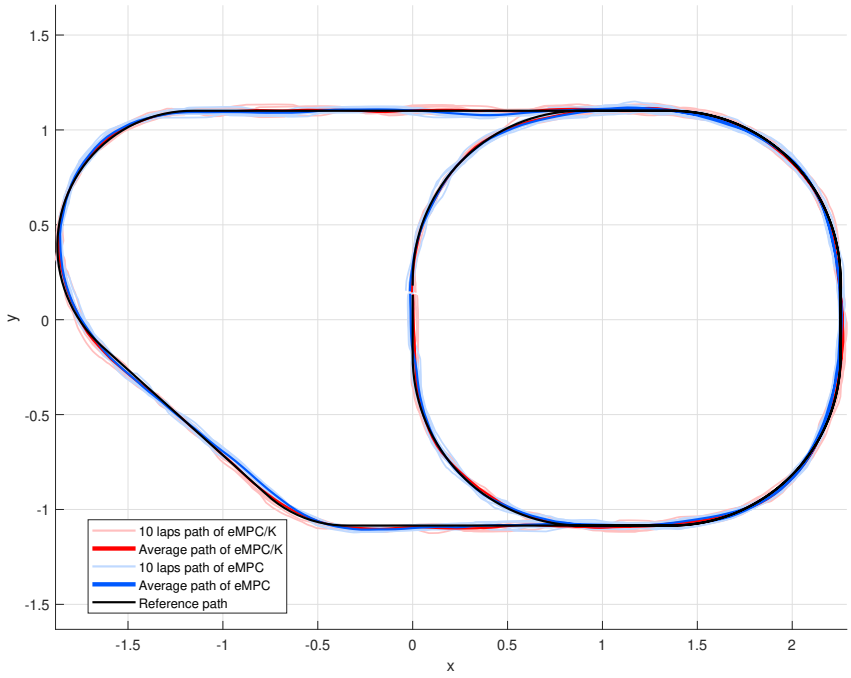

Figure 5.14: Path-tracking performance of eMPC and eMPC/K at $v = 0.32$ m/s with $\sigma = 0.02$ m.

and mean lateral error. The improvement becomes more significant as the event-trigger threshold increases. At $\sigma = 0.02$, eMPC/K provides modest improvement because both controllers still trigger MPC relatively frequently. At $\sigma = 0.06$, the improvement becomes more apparent because the controller spends more time in the inter-event period, where the linear feedback correction is most useful.

The results show two trends. First, increasing σ increases the tracking error for both eMPC and eMPC/K because fewer OCPs are solved. This is consistent with the basic trade-off of event-triggered MPC. Second, eMPC/K becomes more beneficial when σ is larger. For example, at $\sigma = 0.06$ and $v = 0.32$ m/s, eMPC/K reduces RMSE from 0.0290 m to 0.0239 m and reduces mean error from 0.0236 m to 0.0185 m. The corresponding improvements are 17.73% and 21.73%, respectively. This confirms that

Table 5.11: Tracking performance of eMPC and eMPC/K over 10 laps.

σ	v (m/s)	RMSE (m)		Mean error (m)		Improvement (%)	
		eMPC	eMPC/K	eMPC	eMPC/K	ΔRMSE	ΔMean
0.02	0.32	0.0154 ± 0.0009	0.0148 ± 0.0013	0.0126 ± 0.0008	0.0118 ± 0.0010	3.90	6.24
0.02	0.26	0.0135 ± 0.0008	0.0132 ± 0.0012	0.0110 ± 0.0008	0.0105 ± 0.0009	1.82	4.21
0.02	0.20	0.0154 ± 0.0034	0.0144 ± 0.0011	0.0114 ± 0.0022	0.0113 ± 0.0009	6.91	0.88
0.04	0.32	0.0203 ± 0.0019	0.0186 ± 0.0032	0.0155 ± 0.0016	0.0144 ± 0.0020	8.38	6.97
0.04	0.26	0.0172 ± 0.0013	0.0165 ± 0.0011	0.0145 ± 0.0012	0.0134 ± 0.0009	4.08	7.96
0.04	0.20	0.0189 ± 0.0024	0.0178 ± 0.0016	0.0151 ± 0.0023	0.0140 ± 0.0014	5.92	7.35
0.06	0.32	0.0290 ± 0.0039	0.0239 ± 0.0026	0.0236 ± 0.0034	0.0185 ± 0.0021	17.73	21.73
0.06	0.26	0.0250 ± 0.0023	0.0244 ± 0.0016	0.0219 ± 0.0017	0.0192 ± 0.0013	2.40	12.28
0.06	0.20	0.0268 ± 0.0011	0.0259 ± 0.0023	0.0237 ± 0.0011	0.0218 ± 0.0022	3.36	8.02

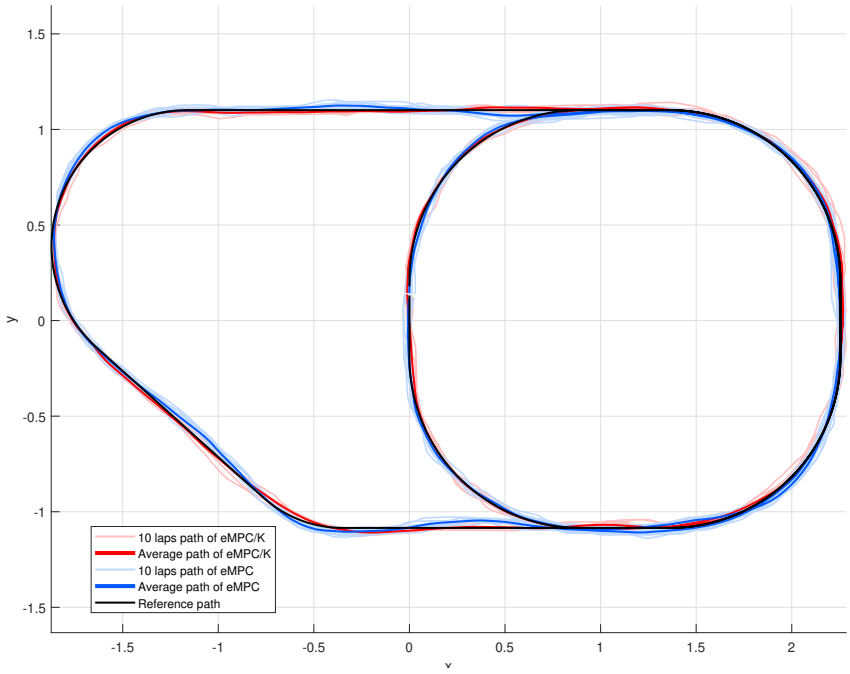

Figure 5.15: Path-tracking performance of eMPC and eMPC/K at $v = 0.32$ m/s with $\sigma = 0.04$ m.

inter-event feedback is most valuable when the controller spends longer periods without solving a new MPC optimization problem.

5.3.4 Triggering and Computational Efficiency

Fig. 5.17-Fig. 5.19 show one randomly selected run at $v = 0.32$ m/s. In each figure, the top plot is eMPC and the bottom plot is eMPC/K. The blue line is the lateral tracking error along the traveled distance, and the green vertical lines mark when the controller actually solves an MPC problem. At $\sigma = 0.02$, the green lines are very dense for both methods, meaning MPC is solved frequently. In this case, both controllers keep the error relatively small and the two error curves look similar, since both are constantly re-planning. When the threshold increases to $\sigma = 0.04$, the green lines become much

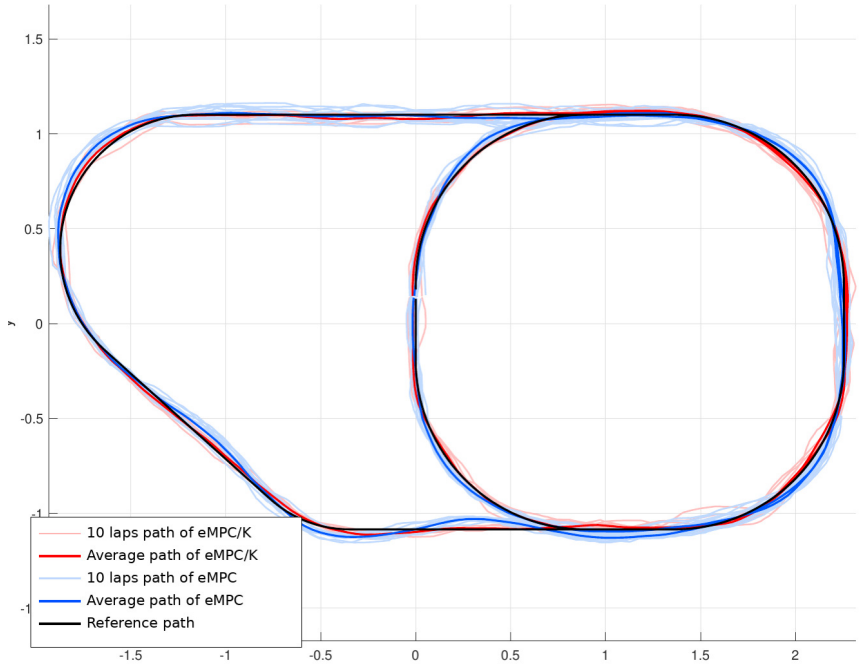

Figure 5.16: Path-tracking performance of eMPC and eMPC/K at $v = 0.32$ m/s with $\sigma = 0.06$ m.

sparser. The error still stays bounded, but occasional peaks become more visible because the controller allows larger deviation before re-solving MPC. Compared with eMPC, eMPC/K reaches a similar error level with fewer green lines, showing that the linear based inter-event control can increase the tracking performance between MPC events. At $\sigma = 0.06$, MPC is triggered only occasionally. Even then, eMPC/K still keeps the error in a comparable range while requiring fewer MPC solves, whereas eMPC shows more clustered MPC activations around segments where the error rises. Overall, these plots illustrate that eMPC/K can reduce MPC computations while keeping tracking error at a similar level, especially when the triggering threshold is relaxed.

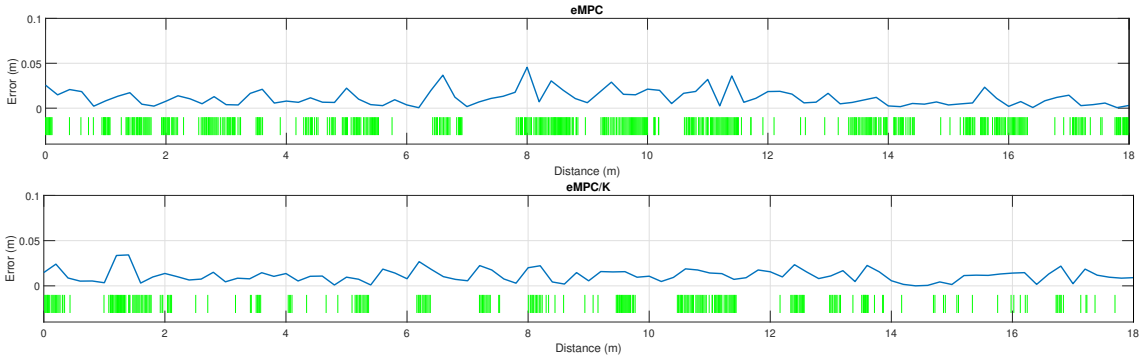

Figure 5.17: Tracking error and MPC activation when $\sigma = 0.02$ under different frameworks. Vertical lines indicate MPC computation.

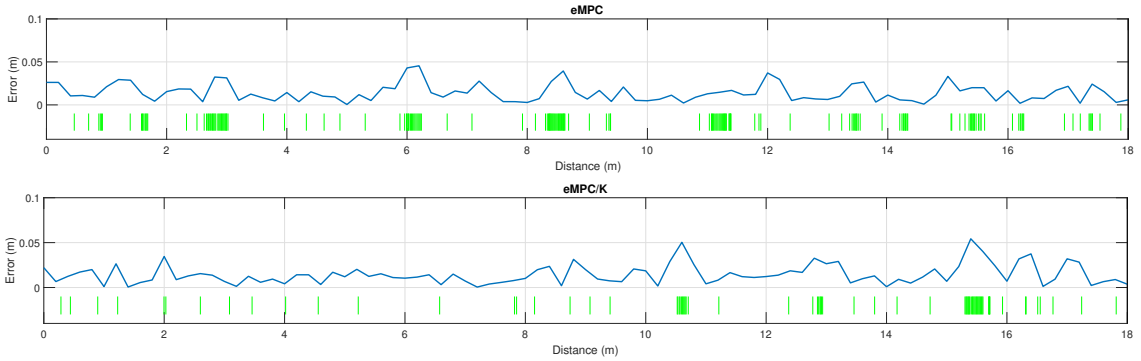

Figure 5.18: Tracking error and MPC activation when $\sigma = 0.04$ under different frameworks. Vertical lines indicate MPC computation.

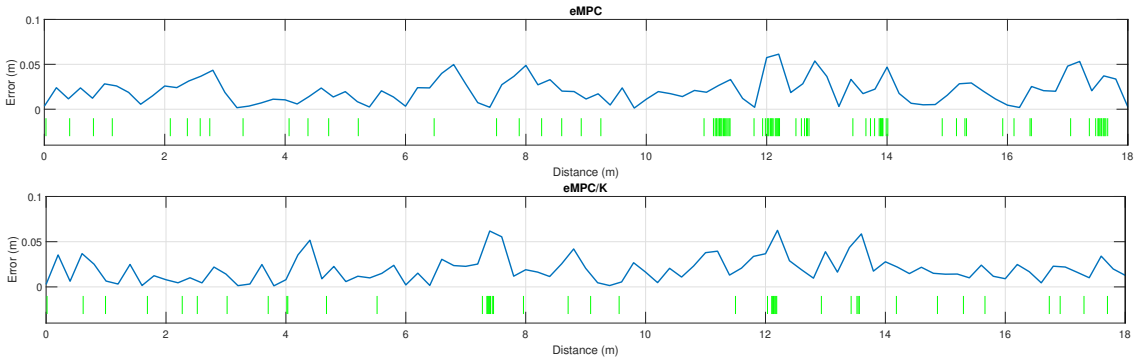

Figure 5.19: Tracking error and MPC activation when $\sigma = 0.06$ under different frameworks. Vertical lines indicate MPC computation.

The computation and triggering statistics are summarized in Table 5.12. For every tested speed and threshold, eMPC/K reduces the MPC event frequency compared with conventional eMPC. For example, at $\sigma = 0.04$ and $v = 0.32$ m/s, the event frequency decreases from 3.563 Hz for eMPC to 1.424 Hz for eMPC/K. The corresponding number of MPC events per lap decreases from 210.1 to 81.6. At $\sigma = 0.06$ and $v = 0.32$ m/s, the event frequency decreases from 2.656 Hz to 0.954 Hz, while the number of MPC events per lap decreases from 148.3 to 54.1. These results show that the linear based inter-event controller not only maintains tracking accuracy but also further reduces the frequency of computationally expensive MPC optimizations.

The reduced event frequency does not mean that the controller stops using feedback. Instead, it means that fewer full MPC optimizations are required. During the skipped optimization steps, eMPC/K still uses the current state through the linear inter-event mapping. This explains why eMPC/K can reduce event frequency while maintaining or improving tracking performance. Compared with conventional eMPC, which only reuses the old optimal input sequence, eMPC/K provides a better balance between computation reduction and closed-loop correction.

Table 5.12: Computation and triggering statistics of eMPC and eMPC/K over 10 laps.

σ	v (m/s)	Event freq. (Hz)		Events/lap	
		eMPC	eMPC/K	eMPC	eMPC/K
0.02	0.32	10.003 ± 0.591	6.245 ± 1.154	552.9 ± 49.9	349.3 ± 66.6
0.02	0.26	7.914 ± 0.469	4.389 ± 0.716	546.1 ± 34.2	287.0 ± 95.2
0.02	0.20	8.653 ± 1.180	5.534 ± 0.755	610.2 ± 53.3	476.6 ± 65.0
0.04	0.32	3.563 ± 0.870	1.424 ± 0.482	210.1 ± 44.9	81.6 ± 18.2
0.04	0.26	2.740 ± 0.598	0.819 ± 0.155	186.7 ± 40.6	56.7 ± 11.2
0.04	0.20	2.797 ± 0.931	1.268 ± 0.891	252.2 ± 69.4	121.0 ± 34.6
0.06	0.32	2.656 ± 1.126	0.954 ± 0.563	148.3 ± 63.6	54.1 ± 31.8
0.06	0.26	1.370 ± 0.419	0.885 ± 0.276	85.8 ± 25.6	60.6 ± 20.3
0.06	0.20	0.952 ± 0.161	0.707 ± 0.350	84.8 ± 14.7	66.2 ± 36.8

These QCar2 results complement the full-size vehicle experiments. The full-size vehicle tests demonstrate that event-triggered MPC is feasible in practical real-world path tracking and can reduce the effect of optimization latency. The QCar2 experiments further show that the inter-event execution strategy itself can be improved. Instead of simply replaying a previous input sequence, a lightweight linear feedback mapping can use the current vehicle state to correct the steering command between MPC events. As a result, eMPC/K provides a better balance between tracking accuracy and computation. The benefit is especially clear when the event-trigger threshold is relaxed, because the vehicle spends more time operating between MPC optimizations.

5.4 Discussion

The CARLA and real-vehicle results show that event-triggered MPC behaves differently in simulation and in physical implementation. In CARLA, increasing the triggering threshold reduces the number of MPC optimizations, but the tracking error also increases. This is expected because the simulation mainly reflects the effect of reusing the

previous optimal sequence. When the controller skips more optimizations, the applied input is based on an older prediction for a longer time, so the tracking accuracy decreases.

The real-vehicle tests show a more practical effect. In both the closed-test-track and public-road experiments, eMPC achieves tracking performance comparable to or better than tMPC. One reason is the computation delay of the MPC solver. In tMPC, every steering command is obtained after solving a new OCP, so each command contains the delay between state measurement and command application. In eMPC, no new OCP is solved during non-triggered steps, and the next command can be sent with much smaller delay. Therefore, although eMPC uses fewer optimizations, its commands can be more timely in real vehicle implementation. This explains why the real-vehicle trend is not the same as the CARLA trend.

The public-road test also shows that reducing MPC computation is not the only issue. When no event is triggered, eMPC can generate shifted steering commands very quickly. If these commands are sent faster than the steering actuator can meaningfully respond, the high update rate does not necessarily improve tracking. For this reason, the controller should be calibrated not only by the event-trigger threshold, but also by the command update rate sent to the actuator.

The QCar2 experiments further show why the inter-event control design matters. Standard eMPC uses the previous optimal input sequence during non-triggered intervals, so the newest state feedback is not directly used until the next event. This behavior becomes more limiting when the threshold is large. The eMPC/K method reduces this limitation by using the current measured state to compute the inter-event steering command. As a result, the controller keeps part of the feedback behavior of MPC while still avoiding unnecessary OCP solves.

5.5 Summary

This chapter evaluated the event-triggered MPC framework in three experimental settings: CARLA simulation, full-size vehicle testing, and QCar2 scale-vehicle testing. The CARLA simulation was used to compare tMPC and eMPC under the same reference route and controller parameters. As the event-trigger threshold increased, the number of MPC optimizations decreased, while the tracking error increased moderately. With $\sigma = 0.03$, eMPC reduced the trigger frequency to 55.21% of tMPC and kept the maximum lateral error within 0.20 m.

The full-size vehicle experiments showed that eMPC can be implemented on a physical autonomous vehicle platform. On the closed test track, the best eMPC case reduced the RMSE from 0.1386 m to 0.1030 m compared with tMPC. In the public-road test, eMPC(0.05) reduced the maximum lateral error from 0.732 m to 0.582 m and produced similar RMSE to tMPC. These tests show that the event-triggered structure remains effective outside simulation.

The QCar2 experiments evaluated the linear based inter-event feedback extension. Compared with standard eMPC, eMPC/K generally reduced the MPC event frequency and improved tracking accuracy, especially when the triggering threshold was large. This indicates that using current state feedback during non-triggered intervals can reduce the open-loop behavior caused by directly replaying the previous optimal input sequence.

The experiments in this chapter complete the evaluation of computationally efficient MPC-based path tracking. The next chapters focus on the second part of the dissertation: learning personalized lane change behavior from expert demonstrations and incorporating the learned preferences into path generation and MPC-based control.

CHAPTER SIX

IRL-BASED PERSONALIZED LANE CHANGE PATH MODELING

The previous chapters focus on model predictive control (MPC)-based path tracking and event-triggered control for improving the computational efficiency of autonomous vehicle motion control. In those chapters, the reference trajectory and the MPC cost weights are assumed to be available before the controller is implemented. This assumption is reasonable for conventional path-tracking problems, where the controller is designed to follow a predefined reference path as accurately as possible. However, it becomes restrictive when personalized autonomous driving is considered.

For a lane change maneuver, different drivers may exhibit different preferences. Some drivers may prefer a shorter and more direct lane change, while others may prefer a smoother and more gradual maneuver. Some drivers may cross the lane marking earlier, whereas others may stay longer in the original lane before completing the lateral transition. These differences indicate that a lane change trajectory is not only a geometric path but also a reflection of the driver's behavior and preference. Therefore, a personalized autonomous driving system should be able to learn such preferences from demonstrations instead of relying only on manually designed reference paths.

This chapter investigates inverse reinforcement learning (IRL) as a path-level behavior learning method for personalized lane change modeling. The main idea is to assume that the demonstrated lane change trajectory is generated by an implicit optimization process, where the driver balances several objectives such as smoothness, lane change length, lane-mark crossing location, and final lateral position. These objectives are described using interpretable trajectory features. Maximum entropy inverse

reinforcement learning (MaxEnt IRL) is then used to infer the feature weights that best explain the expert demonstrations.

The objective of this chapter is to verify whether IRL can learn a driver's lane change preference at the path-generation level. Therefore, the focus is not closed-loop vehicle control, but rather personalized path modeling. The output of this chapter is a learned cost function that can generate Bezier lane change paths similar to expert trajectories under different initial conditions. This provides the foundation for Chapter 7, where the learned preference is further incorporated into an MPC formulation for closed-loop personalized lane change control.

6.1 Bezier Curve Representation for Lane Change Paths

A Bezier curve is used in this chapter to represent the lane change path. The reason for selecting a Bezier curve is that it provides a compact and smooth parameterization of a trajectory through a finite number of control points. This property is useful for lane change modeling because the path shape can be adjusted by changing the control points, while the resulting curve remains continuous and smooth.

An nth-order Bezier curve is defined as

$$B(t) = \sum_{i=0}^{n} P_i b_{i,n}(t) = \sum_{i=0}^{n} P_i \binom{n}{i} (1-t)^{n-i} t^i, \tag{6.1}$$

where $t \in [0,1]$ is the curve parameter, P_i is the ith control point, and $b_{i,n}(t)$ is the Bernstein polynomial.

In this chapter, a fifth-order Bezier curve is used to model the lane change path. Therefore, the path is determined by six control points, denoted as $P_0, P_1, \ldots, P_5$. Substituting $n = 5$ into (6.1), the Bezier curve can be written in polynomial form as

$$B(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \tag{6.2}$$

where the coefficients are functions of the control points:

$$\begin{aligned} a_0 &= P_0, \\ a_1 &= -5P_0 + 5P_1, \\ a_2 &= 10P_0 - 20P_1 + 10P_2, \\ a_3 &= -10P_0 + 30P_1 - 30P_2 + 10P_3, \\ a_4 &= 5P_0 - 20P_1 + 30P_2 - 20P_3 + 5P_4, \\ a_5 &= -P_0 + 5P_1 - 10P_2 + 10P_3 - 5P_4 + P_5. \end{aligned} \tag{6.3}$$

For vehicle lane change maneuvers, the path should start and end smoothly. In particular, the curvature at the beginning and the end of the lane change should be zero to avoid abrupt lateral motion. The curvature of a Bezier curve is calculated as

$$\kappa(t) = \frac{\dot{B}(t) \times \ddot{B}(t)}{|\dot{B}(t)|^3}, \tag{6.4}$$

where $\dot{B}(t)$ and $\ddot{B}(t)$ are the first and second derivatives of $B(t)$ with respect to t.

For the fifth-order Bezier curve in (6.2), the first and second derivatives are

$$\dot{B}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4, \tag{6.5}$$

and

$$\ddot{B}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3. \tag{6.6}$$

At the beginning of the lane change, the zero-curvature condition requires

$$\dot{B}(0) \times \ddot{B}(0) = 0. \tag{6.7}$$

Substituting (6.5) and (6.6) into (6.7) gives

$$5(P_1 - P_0) \times 20\left((P_2 - P_1) - (P_1 - P_0)\right) = 0. \tag{6.8}$$

This condition is satisfied when P_0, P_1, and P_2 are colinear. Similarly, enforcing zero curvature at the end of the lane change requires P_3, P_4, and P_5 to be colinear. These constraints make the lane change path begin and end with a straight-line segment, which is desirable for smooth vehicle motion.

6.2 Trajectory Features for Driver Preference Learning

After defining the trajectory representation, the next step is to define how a lane change path is evaluated. In the proposed framework, a driver is assumed to prefer a trajectory according to several interpretable path-level features. These features describe the main characteristics of a lane change maneuver and provide the basis for the IRL formulation. In this chapter, four features are considered: comfort, lane change length, lane-mark crossing distance, and final lateral position.

6.2.1 Comfort Feature

Comfort is represented by the integral of the squared curvature along the Bezier path:

$$f_c = \int_0^1 \kappa(t)^2 dt = \int_0^1 \left(\frac{\dot{B}(t) \times \ddot{B}(t)}{|\dot{B}(t)|^3} \right)^2 dt. \tag{6.9}$$

A smaller f_c indicates a smoother path and usually corresponds to a more comfortable lane change.

6.2.2 Longitudinal Efficiency Features

The first longitudinal efficiency feature is the total longitudinal length of the lane change:

$$f_{T1} = P_{5x} - P_{0x}. \tag{6.10}$$

This feature measures how long the vehicle travels in the longitudinal direction before completing the lane change.

The second longitudinal efficiency feature describes the longitudinal distance from the lane change starting point to the lane-mark crossing point:

$$f_{T2} = \frac{(w - y_b)(x_a - x_b)}{y_a - y_b} + x_b - P_{0x}, \tag{6.11}$$

where (x_b, y_b) and (x_a, y_a) are two points on the trajectory immediately before and after the lane marking is crossed. This feature captures whether the driver tends to enter the target lane earlier or later during the maneuver.

6.2.3 Final Lateral Position Feature

The final lateral position feature is defined as

$$f_y = y_5. \tag{6.12}$$

This feature describes the final lateral location of the vehicle after the lane change. It is included because different trajectories may complete the lane change at slightly different positions inside the target lane.

The complete feature vector is

$$f(\zeta) = [f_c,\ f_{T1},\ f_{T2},\ f_y]^T, \tag{6.13}$$

where ζ denotes a lane change trajectory.

These four features are selected because they represent the main geometric characteristics of a lane change maneuver. The curvature feature describes path smoothness and ride comfort. The total longitudinal length reflects whether the driver prefers a short and direct lane change or a gradual lane change. The lane-mark crossing distance describes the timing of lateral transition from the original lane to the target lane. The final lateral position captures the driver's preferred endpoint inside the target lane.

Therefore, the feature vector provides an interpretable description of driver-specific lane change preference.

6.3 Lane Change Path Optimization Problem

After defining the trajectory representation and the feature functions, the lane change behavior can be formulated as an optimization problem. The main assumption is that a driver selects a trajectory by minimizing a weighted combination of the designed features. The role of IRL is to learn these unknown weights from expert demonstrations.

The starting point of the lane change, P_0, is assumed to be known. To satisfy the zero-curvature conditions discussed in Section 6.1, P_1 and P_2 are constrained to have the same lateral coordinate as P_0, while P_3 and P_4 are constrained to have the same lateral coordinate as P_5. Therefore, the decision variable can be written as

$$X = [x_1, x_2, x_3, x_4, x_5, y_5]^T, \tag{6.14}$$

where x_i is the longitudinal coordinate of control point P_i, and y_5 is the lateral coordinate of the final control point P_5.

The lane change path is obtained by solving the following optimization problem:

$$\begin{aligned}
\min_X \quad & J(X) = W_1 \left(\int_0^1 \kappa(t)^2 dt \right)^2 + W_2 (x_5 - x_0)^2 \\
& \quad + W_3 \left(\frac{(w - y_b)(x_a - x_b)}{y_a - y_b} + x_b - P_{0x} \right)^2 + W_4 y_5^2 && \text{(6.15a)} \\
\text{s.t.} \quad & B(t) = \sum_{i=0}^{5} P_i b_{i,5}(t), && \text{(6.15b)} \\
& x_{i-1} < x_i, \quad 1 \le i \le 5, && \text{(6.15c)} \\
& x_{lb} < x_5 - x_0 < x_{ub}, && \text{(6.15d)} \\
& y_{lb} < y_5 < y_{ub}. && \text{(6.15e)}
\end{aligned}$$

The first term in (6.15a) penalizes the total curvature of the lane change path. This term is associated with comfort because a path with large curvature variation usually produces a more aggressive lateral maneuver. The second term penalizes the total longitudinal distance of the lane change. The third term penalizes the longitudinal distance at which the vehicle crosses the lane marking. The fourth term penalizes the final lateral position. Together, these terms describe the main path-level characteristics of a lane change maneuver.

Constraint (6.15b) defines the Bezier curve. Constraint (6.15c) ensures that the control points are ordered in the longitudinal direction, which prevents the path from folding backward. Constraint (6.15d) bounds the longitudinal length of the lane change, and constraint (6.15e) ensures that the final point lies in the target lane.

The weight vector

$$W = [W_1, W_2, W_3, W_4]^T \tag{6.16}$$

represents the driver's relative preference among these objectives. In practice, these weights are unknown because a human driver does not explicitly provide numerical cost weights. Therefore, the next section introduces MaxEnt IRL to infer W from expert trajectories.

6.4 Maximum Entropy IRL for Lane Change Path Learning

Given a set of expert trajectories

$$D = \tilde{\zeta}_1, \tilde{\zeta}_2, \ldots, \tilde{\zeta}_N, \tag{6.17}$$

the empirical expert feature vector is calculated as

$$\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f(\tilde{\zeta}_i). \tag{6.18}$$

The goal of MaxEnt IRL is to find a weight vector W such that the expected feature vector under the learned model matches the empirical feature vector:

$$\mathbb{E}_{p(\zeta|W)}[f] = \bar{f}. \tag{6.19}$$

The cost of a trajectory is modeled as a linear combination of features:

$$J = W^T f(\zeta). \tag{6.20}$$

Following the maximum entropy principle, the probability of a trajectory is defined as

$$p(\zeta|W) = \frac{1}{Z(W)} e^{-W^T f(\zeta)}, \tag{6.21}$$

where $Z(W)$ is the partition function that normalizes the trajectory distribution.

The weight vector is learned by maximizing the log-likelihood of the expert trajectories:

$$W = \arg\max L(W) = \arg\max \sum_{i=1}^{N} \log p(\tilde{\zeta}_i|W). \tag{6.22}$$

The gradient of the log-likelihood can be written as

$$\nabla L(W) = \mathbb{E}_{p(\zeta|W)}[f] - \bar{f}. \tag{6.23}$$

Computing the expectation in (6.23) over all possible trajectories is difficult. Therefore, this chapter uses the feature vector of the most likely trajectory as an approximation:

$$\mathbb{E}_{p(\zeta|W)}[f] \approx f(\arg\max p(\zeta|W)). \tag{6.24}$$

In implementation, the most likely trajectory is obtained by solving the Bezier lane change optimization problem in (6.15) using the current weight vector.

The weight vector is updated iteratively as

$$W \leftarrow W + \eta \frac{\nabla L(W)}{|\nabla L(W)|}, \tag{6.25}$$

Algorithm 6.1: MaxEnt IRL for Personalized Lane Change Path Modeling

Input: Expert trajectories $D = \{\tilde{\zeta}_1, \tilde{\zeta}_2, \ldots, \tilde{\zeta}_N\}$, learning rate η

Output: Feature weight vector W

Compute expert feature vector $\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(\tilde{\zeta}_i)$
Initialize W as an all-ones vector and set $c \leftarrow 0$
while *W has not converged* **do**
    for *each expert trajectory $\tilde{\zeta}_i$ in D* **do**
        Solve (6.15) using the initial point of $\tilde{\zeta}_i$ and the current W
        Obtain the optimized trajectory $\zeta_{\text{opt},i}$
        Compute $f_{\text{opt},i} = f(\zeta_{\text{opt},i})$
    end
    Estimate $\mathbb{E}_{p(\zeta|W)}[f] \leftarrow \frac{1}{N}\sum_{i=1}^{N} f_{\text{opt},i}$
    Compute $\nabla L(W) = \mathbb{E}_{p(\zeta|W)}[f] - \bar{f}$
    Update $W \leftarrow W + \eta \frac{\nabla L(W)}{\|\nabla L(W)\|}$
    $c \leftarrow c + 1$
    if $c = 200$ **then**
        $\eta \leftarrow \eta/10$
    end
end
return W

where η is the learning rate. The learning rate is reduced during training to improve convergence stability.

Algorithm 6.1 summarizes the complete MaxEnt IRL procedure used in this chapter. The algorithm first computes the empirical feature vector from the expert lane change trajectories. Then, for each iteration, the Bezier lane change optimization problem in (6.15) is solved using the current weight vector and the same initial condition as each expert trajectory. The resulting optimized trajectories are used to approximate the expected feature vector in (6.13). Finally, the difference between the optimized feature vector and the empirical expert feature vector is used to update W according to (6.25). This process is repeated until the feature weights converge.

6.5 Experimental Evaluation

The purpose of the experimental evaluation is to verify whether the proposed MaxEnt IRL framework can learn a path-level lane change preference from expert demonstrations. Since this chapter focuses on behavior learning rather than closed-loop control, the evaluation is conducted at the geometric path level. Specifically, the experiment examines whether the learned weights can generate Bezier lane change paths that are consistent with expert trajectories under different initial conditions.

6.5.1 Expert Trajectory Dataset

The expert trajectories are generated using the Bezier lane change optimization problem in (6.15) with a selected weight vector. This selected weight vector is used only for generating the expert demonstrations and is not provided to the IRL algorithm. Therefore, the IRL model must infer the feature weights only from the observed trajectories.

The road width is set to 4 m. The original lane covers the lateral range from 0 m to 4 m, while the adjacent lane covers the lateral range from 4 m to 8 m. The total road length is 25 m, so each lane change maneuver is completed within this distance. The initial points are randomly selected from a region where the longitudinal coordinate ranges from -1 m to 1 m and the lateral coordinate ranges from 0 m to 4 m. This setup generates different lane change trajectories under different initial conditions while maintaining the same underlying driving preference.

A total of 30 expert trajectories are generated. The first 25 trajectories are used as the training dataset, and the remaining 5 trajectories are used for testing. Fig. 6.1 shows the expert trajectories used for training. Although these trajectories start from different initial points, they exhibit a consistent lane change style, which allows the IRL algorithm to learn the underlying path preference rather than memorize a single trajectory.

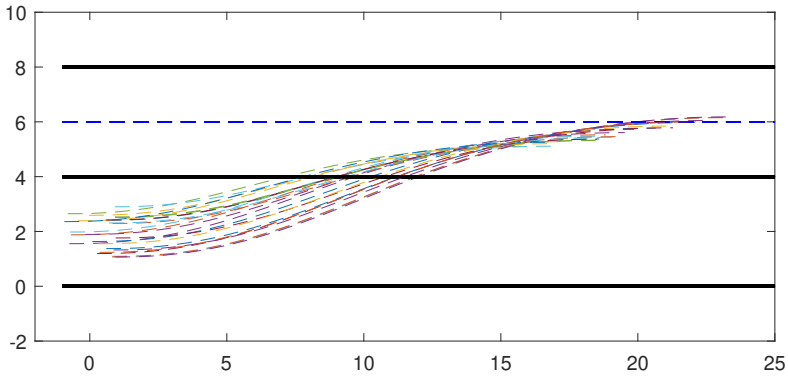

Figure 6.1: Expert trajectories used as the training dataset. The original lane is between $y = 0$ m and $y = 4$ m, and the target lane is between $y = 4$ m and $y = 8$ m. The dashed line indicates the target lane centerline.

Table 6.1: Expert feature ranges used for normalization.

	f_c	f_{T1}	f_{T2}	f_y
$f_{\min}$	0.0013	15.932	7.033	5.108
$f_{\max}$	0.0015	22.388	11.097	6.181

6.5.2 Feature Normalization and Learned Weights

To improve numerical stability during IRL training, min-max normalization is applied to the feature values. The feature ranges of the expert trajectories are listed in Table 6.1.

The learned feature weights are reported in Table 6.2. The final lateral position feature f_y has the largest learned weight, indicating that the endpoint lateral location is important for reproducing the expert behavior in this dataset. The curvature feature f_c and lane change length feature f_{T1} also receive significant weights, showing that smoothness and longitudinal maneuver length are both involved in the learned preference. The weight

Table 6.2: Learned feature weights for the IRL path modeling framework.

	f_c	f_{T1}	f_{T2}	f_y
W	1.434	1.3017	0.7947	4.4054

of f_{T2} is smaller than the other feature weights, suggesting that the lane-mark crossing location contributes to the path shape but is less dominant for this set of demonstrations.

6.5.3 Predicted Path and Expert Path Comparison

After training, the learned weights are used to generate predicted lane change paths for the five testing initial conditions. Fig. 6.2 compares the predicted paths with the corresponding expert paths. The red dashed curves represent the predicted paths generated using the learned weights, while the black curves represent the expert paths.

As shown in Fig. 6.2, the predicted paths closely follow the expert paths in all five testing cases. This result indicates that the learned cost function can reproduce the demonstrated lane change behavior under unseen initial conditions. In other words, the IRL model does not simply memorize the training trajectories. Instead, it learns a feature-weight representation of the driver's path preference and uses this learned cost function to generate new lane change paths. The second testing case shows a relatively larger discrepancy near the end of the lane change compared with the other cases. This difference is consistent with the feature-level error analysis in Table 6.3. One possible reason is that the feature expectation in the MaxEnt IRL training is approximated using the most likely trajectory rather than a complete distribution over trajectories. This approximation reduces computational complexity, but it can introduce small errors in some feature dimensions.

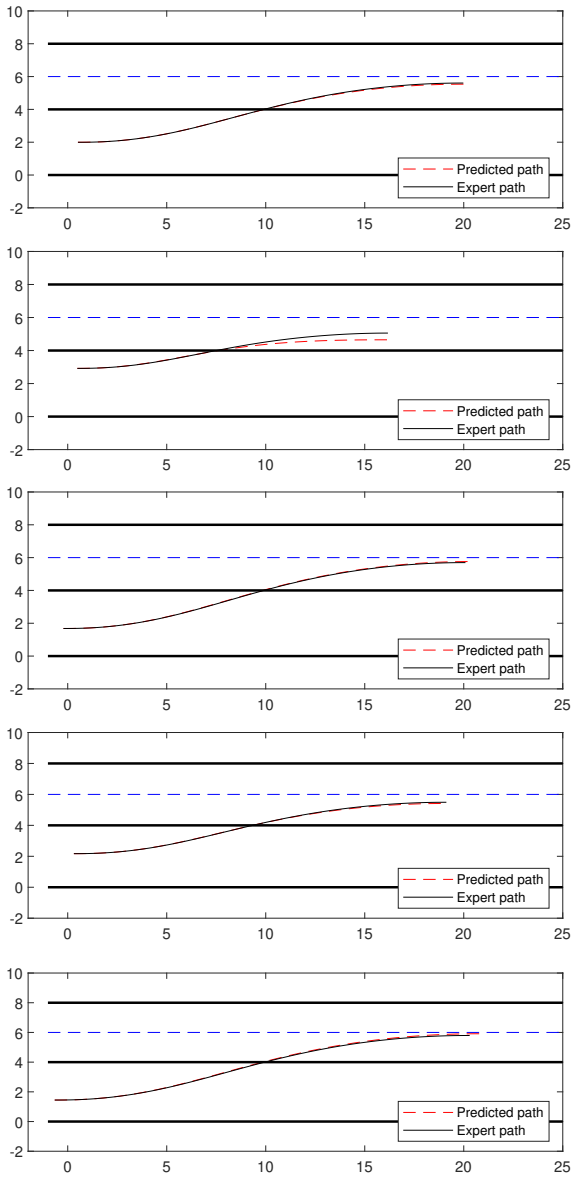

Figure 6.2: Comparison between predicted trajectories and expert trajectories under five testing initial conditions. The red dashed curves represent the predicted paths generated using the learned weights, and the black curves represent the expert paths.

Table 6.3: Feature differences between expert paths and predicted paths.

Initial point	Curvature	Length (m)	x-cross (m)	y-end (m)
$[0.515, 1.997]$	-3.02×10^{-6}	0.006633	0.022394	-0.06938
$[0.486, 2.919]$	9.43×10^{-5}	0.264324	0.124813	-0.40642
$[-0.216, 1.681]$	-5.30×10^{-7}	0.38732	-0.06683	0.065326
$[0.311, 2.171]$	1.22×10^{-5}	-0.05343	-0.0008	-0.07845
$[-0.658, 1.448]$	7.51×10^{-7}	0.534073	-0.10584	0.116749

6.5.4 Feature-Level Error Analysis

To further quantify the similarity between the predicted paths and the expert paths, Table 6.3 lists the feature differences for the five testing cases. Each row corresponds to one testing initial point. The compared quantities include curvature, lane change length, lane-mark crossing distance, and final lateral position.

The curvature differences are very small across all testing cases, indicating that the predicted paths preserve the smoothness characteristics of the expert paths. The lane change length errors are also small relative to the total maneuver length. The largest length difference is approximately 0.534 m, which is small compared with the full lane change distance. The lane-mark crossing distance errors are also limited, showing that the learned model captures the longitudinal transition behavior of the expert trajectories.

The largest discrepancy appears in the final lateral position of the second testing case, where the y-end difference is -0.40642 m. Although this error is larger than the other y-end errors, the predicted path still follows the overall expert trajectory shape. Therefore, the feature-level comparison confirms that the learned weights reproduce the main geometric properties of the demonstrated lane change behavior.

6.6 Discussion

The experimental results demonstrate that MaxEnt IRL can learn a path-level lane change preference from expert trajectories. The learned weights provide an interpretable representation of how different trajectory features contribute to the demonstrated behavior. Instead of directly copying a specific path, the proposed framework learns a cost function that can generate similar Bezier lane change paths from different initial conditions. This distinction is important for personalized autonomous driving. A fixed reference path can only describe one specific maneuver. In contrast, a learned cost function captures the preference behind the demonstrated behavior. Therefore, when the starting point changes, the learned model can still generate a lane change path that follows the same style. At the same time, this chapter has several limitations. First, the expert trajectories are generated in simulation using a selected cost function, so the experiment mainly verifies the learning capability of the IRL framework under controlled conditions. Second, the method focuses on geometric path modeling and does not explicitly consider vehicle dynamics, actuator constraints, or real-time feedback control. Third, the feature expectation in MaxEnt IRL is approximated by the most likely trajectory, which may lead to feature mismatch in some testing cases.

These limitations motivate the next chapter. To realize personalized autonomous driving on a vehicle platform, the learned preference must be integrated with a controller that accounts for vehicle dynamics and feedback. Chapter 7 extends this path-level learning framework by embedding IRL-learned feature weights into an MPC formulation.

6.7 Summary

This chapter presented an IRL-based personalized lane change path modeling framework. A fifth-order Bezier curve was used to represent the lane change path, and zero-curvature conditions were imposed at the beginning and end of the maneuver to

ensure smooth transitions. Four interpretable features were designed to describe the lane change behavior: curvature-based comfort, total lane change length, lane-mark crossing distance, and final lateral position.

MaxEnt IRL was used to infer the feature weights from expert trajectories. The experimental results showed that the learned weights can generate predicted lane change paths that closely match expert paths under unseen initial conditions. The feature-level comparison further confirmed that the predicted paths preserve the main characteristics of the expert behavior. Therefore, this chapter verifies that IRL can be used to learn path-level lane change preferences. The next chapter extends this idea from path generation to closed-loop control by embedding the learned preference into an MPC controller.

CHAPTER SEVEN

PERSONALIZED MPC VIA IRL

Chapter 6 demonstrated that MaxEnt IRL can learn driver-specific lane change preferences at the geometric path level, where the learned weights are used to generate personalized Bezier lane change paths from expert demonstrations. However, autonomous lane change execution requires more than geometric path generation. The vehicle controller must also account for vehicle dynamics, steering constraints, actuator limits, and real-time state feedback.

Building on the path-level personalization framework, this chapter extends IRL-based learning to closed-loop vehicle control. Instead of using IRL only to generate a geometric lane change path, the learned feature weights are embedded directly into an MPC cost function. In this formulation, IRL learns the relative importance of vehicle-state and control-input features, while MPC uses these learned weights to optimize the vehicle motion subject to dynamic and actuator constraints. Therefore, the learned driver preference is realized through feedback control rather than only through offline path generation.

7.1 MaxEnt IRL for MPC Cost Learning

The MaxEnt IRL formulation introduced in Chapter 6 is adopted in this chapter to learn the cost weights of an MPC controller. However, the learning target is different from that in Chapter 6. In Chapter 6, IRL learns the weights of geometric path-level features and the learned cost is used to generate a Bezier lane change path. In this chapter, IRL learns the weights of vehicle-state and control-input features, and the learned cost is directly embedded into the MPC optimization problem. Therefore, the learned preference affects the closed-loop vehicle motion rather than only the offline geometric path.

Consider a set of expert lane change demonstrations

$$L = \{l_1, l_2, \ldots, l_N\}, \tag{7.1}$$

where each demonstration l_i consists of a state sequence and a control sequence over a short prediction horizon. A feature mapping $f(l_i)$ is used to extract the driving characteristics of each segment, and the average expert feature vector is computed as

$$\bar{F} = \frac{1}{N} \sum_{i=1}^{N} f(l_i). \tag{7.2}$$

Following the MaxEnt IRL principle, the objective is to find a weight vector W such that the expected feature vector of the generated MPC trajectories matches the expert feature vector:

$$\mathbb{E}_{p(l|W)}[f] = \bar{F}. \tag{7.3}$$

The trajectory cost is defined as a weighted combination of the MPC-related features:

$$J(W) = W^T f(l_i). \tag{7.4}$$

The corresponding trajectory distribution is

$$p(l_i|W) = \frac{1}{Z(W)} e^{-J(W)}, \tag{7.5}$$

where $Z(W)$ is the partition function.

The weight vector is updated using the feature-matching gradient

$$\nabla L(W) = \mathbb{E}_{p(l|W)}[f] - \bar{F}. \tag{7.6}$$

As in Chapter 6, directly computing the expectation over all feasible trajectories is computationally difficult. Therefore, the expected feature vector is approximated by the feature vector of the most likely trajectory generated by the current MPC controller:

$$\mathbb{E}_{p(l|W)}[f] \approx f(\arg\max p(l|W)). \tag{7.7}$$

The weight update is then given by

$$W \leftarrow W + \eta \frac{\nabla L(W)}{\|\nabla L(W)\|}, \tag{7.8}$$

where η is the learning rate.

In this chapter, the most likely trajectory in (7.7) is obtained by solving the MPC optimal control problem with the current weight vector. This is the key difference from Chapter 6, where the most likely trajectory is obtained by solving a Bezier path optimization problem. Thus, although the MaxEnt IRL update rule remains the same, the learned weights in this chapter are associated with closed-loop control behavior.

Before defining the personalized MPC cost function, the driving behavior in each lane change segment is represented by a set of interpretable features. These features are selected to describe both the geometric progress of the lane change and the dynamic/control characteristics of the vehicle. Specifically, the lateral position deviation and heading angle deviation describe how the vehicle approaches and aligns with the target lane, while the yaw rate effort and steering effort describe the smoothness and control demand of the maneuver. Since these features can be computed from both expert demonstrations and MPC-generated trajectories, they provide a consistent basis for MaxEnt IRL feature matching.

7.1.1 Lateral Position Deviation

The lateral position deviation feature measures the difference between the vehicle lateral position and the centerline of the target lane during the lane change maneuver. This feature is included because reaching and staying close to the target lane is the primary objective of a lane change. A smaller lateral deviation indicates that the vehicle moves closer to the target lane centerline over the prediction horizon, while a larger value

indicates that the vehicle remains farther away from the desired lateral position.

$$f_l = \sum_{t=1}^{n} (y_t - y_{\text{ref}})^2, \tag{7.9}$$

where y_t is the vehicle lateral position at time step t, y_{ref} is the lateral coordinate of the target lane centerline, and n is the number of time steps in the trajectory segment.

This feature reflects the driver's preference for lateral convergence during a lane change. If the learned weight associated with f_l is large, the MPC controller places more emphasis on reducing the lateral distance to the target lane. As a result, the vehicle tends to move toward the target lane more quickly. If the weight is smaller, the controller allows a more gradual lateral transition. Therefore, f_l is useful for distinguishing different lane change styles in terms of how quickly the vehicle approaches the target lane.

7.1.2 Heading Angle Deviation

The heading angle deviation feature measures the difference between the vehicle heading angle and the desired heading direction of the target lane. During a lane change, the vehicle should not only move laterally but also maintain a reasonable orientation with respect to the lane direction. Excessive heading deviation may indicate a sharp maneuver, while very small heading deviation may correspond to a slower and more gradual lane change.

The heading angle deviation feature is defined as

$$f_h = \sum_{t=1}^{n} (\psi_t - \psi_{\text{ref}})^2, \tag{7.10}$$

where ψ_t is the vehicle heading angle at time step t, and ψ_{ref} is the heading angle of the target lane. For the straight-lane lane change considered in this chapter, after the coordinate transformation introduced in Section 7.3, the target lane direction is aligned with the transformed longitudinal axis, and therefore $\psi_{\text{ref}} = 0$.

This feature captures how strongly the vehicle rotates away from the target lane direction during the maneuver. A larger learned weight on f_h penalizes heading deviation more heavily and encourages the vehicle to keep its orientation closer to the lane direction. A smaller weight allows a larger heading change when needed. Together with the lateral position deviation feature, f_h describes the trade-off between lateral transition progress and vehicle orientation alignment.

7.1.3 Yaw Rate Effort

The yaw rate effort feature penalizes large yaw rate values during the lane change maneuver. Yaw rate represents the rotational speed of the vehicle around its vertical axis and describes how rapidly the vehicle heading is changing. Therefore, this feature is closely related to the smoothness of the lane change.

The yaw rate effort feature is defined as

$$f_y = \sum_{t=1}^{n} \dot{\psi}_t^2, \tag{7.11}$$

where $\dot{\psi}_t$ is the yaw rate at time step t. In the vehicle state vector used in this chapter, $\dot{\psi}_t$ corresponds to the yaw-rate state r_t.

This feature complements the heading angle deviation feature. The heading angle deviation describes the magnitude of the orientation error, while the yaw rate effort describes how fast the orientation changes. Two lane change trajectories may have similar heading profiles but different yaw-rate responses. A trajectory with a large yaw rate may feel more abrupt, whereas a trajectory with a smaller yaw rate is usually smoother. Therefore, f_y is included to capture the dynamic smoothness of the lane change maneuver.

7.1.4 Steering Effort

The steering effort feature penalizes large steering commands generated by the MPC controller. Since steering is the direct control input for lateral motion control, this feature describes the control effort required to complete the lane change. Large steering commands may produce sharper vehicle motion and increase actuator demand, while smaller steering commands generally correspond to a smoother maneuver.

The steering effort feature is defined as

$$f_s = \sum_{t=1}^{n} u_t^2, \tag{7.12}$$

where u_t is the steering command at time step t.

This feature is different from the yaw rate effort feature because it describes the input side of the control problem rather than the vehicle response. Due to vehicle dynamics, tire characteristics, and actuator limits, the same steering input may lead to different yaw-rate responses under different conditions. Therefore, including both f_y and f_s allows the learned cost function to account for both motion smoothness and control effort. A larger learned weight on f_s encourages the MPC controller to use smaller steering commands, while a smaller weight allows stronger steering actions when needed to reproduce the demonstrated lane change style.

The complete feature vector is

$$F = [f_l, f_h, f_y, f_s]^T, \tag{7.13}$$

and the personalized cost function is

$$J(W) = W_1 f_l + W_2 f_h + W_3 f_y + W_4 f_s. \tag{7.14}$$

Each feature represents one interpretable aspect of the lane change maneuver: lateral convergence, heading alignment, yaw smoothness, and steering effort. By learning

the corresponding weight vector $W = [W_1, W_2, W_3, W_4]^T$ from expert demonstrations, the proposed framework converts driver-specific lane change preferences into the MPC objective function. Unlike the path-level IRL formulation in Chapter 6, where the learned weights are used to generate a geometric Bezier path, the learned weights in this chapter directly affect the closed-loop steering command through MPC optimization. The following section formulates this personalized MPC problem by combining the IRL-learned cost function with vehicle dynamics and actuator constraints.

7.2 MPC Formulation with Learned Weights

The MPC controller uses the IRL-learned weights to generate personalized lane change behavior in closed-loop control. The vehicle model and the general MPC formulation have been introduced in Chapter 3; therefore, this section does not repeat the full derivation of the vehicle dynamics. Instead, the vehicle model is used as the prediction model inside MPC, and the focus of this section is on how the IRL-learned feature weights are incorporated into the MPC objective function.

At each time step, MPC predicts the future vehicle motion over a finite horizon and optimizes the steering input sequence according to the personalized cost function learned from expert demonstrations. In this way, the learned weights do not only evaluate a completed lane change trajectory. They directly influence the control action selected by MPC at each sampling instant. Therefore, different learned weight vectors can lead to different closed-loop lane change behaviors under the same vehicle model and actuator constraints.

The resulting finite-horizon optimal control problem is formulated as:

$$\min_{U} \quad J = W_1 f_l + W_2 f_h + W_3 f_y + W_4 f_s \tag{7.15}$$

$$\text{s.t.} \quad s_t = \hat{s}_t, \tag{7.16}$$

$$\text{vehicle dynamic model}, \tag{7.17}$$

$$u_{\min} \leq u_{t+n} \leq u_{\max}, \quad 0 \leq n \leq p-1, \tag{7.18}$$

$$du_{\min} \leq u_{t+n} - u_{t+n-1} \leq du_{\max}, \quad 0 \leq n \leq p-1. \tag{7.19}$$

Here, p is the prediction horizon, $\hat{s}_t$ is the measured or estimated vehicle state at time t, and

$$U = [u_t, u_{t+1}, \ldots, u_{t+p-1}]^T \tag{7.20}$$

is the control sequence over the prediction horizon. The learned weights W determine the trade-off among lateral deviation, heading deviation, yaw rate effort, and steering effort. Therefore, different learned weights lead to different lane change styles.

The vehicle prediction model used in this chapter follows the vehicle modeling framework introduced in Chapter 3. Therefore, the detailed derivation of the vehicle dynamics and tire-force model is not repeated here. For the personalized lane change controller, the vehicle state is defined as

$$s = [x, y, v_y, \psi, r]^T, \tag{7.21}$$

where x and y are the vehicle position in the global frame, v_y is the lateral velocity in the vehicle frame, ψ is the heading angle, and r is the yaw rate.

The discrete-time prediction model is written compactly as

$$s_{k+1} = g(s_k, u_k, v_{x,k}), \quad 0 \leq k \leq p-1, \tag{7.22}$$

where $g(\cdot)$ denotes the discretized vehicle model described in Chapter 3, u_k is the steering input, and $v_{x,k}$ is the longitudinal velocity. Since this chapter focuses on lateral lane change control, the longitudinal velocity is treated as a measured input during prediction.

The tire-force calculation and front-wheel steering assumption are the same as those introduced in Chapter 3. Thus, this chapter focuses on how the IRL-learned feature weights are embedded into the MPC objective function, rather than repeating the vehicle model derivation.

7.3 IRL-MPC Training Procedure

Before training, a coordinate transformation is applied so that the origin is located at the start of the lane change and the x-axis is aligned with the target lane direction. After this transformation, the target lane heading satisfies $\psi_{\text{ref}} = 0$ for the straight-lane lane change considered in this chapter.

The expert lane change trajectory is divided into 0.8 s segments and resampled using linear interpolation. Since the MPC sampling time is 0.2 s in the CARLA experiment, each segment contains four sampling intervals and therefore corresponds to a prediction horizon of $p = 4$. This segmentation is selected to make the expert trajectory segment and the MPC prediction horizon have the same temporal length. As a result, the feature vector extracted from each expert segment can be directly compared with the feature vector generated by MPC from the same initial state. This one-to-one correspondence between expert features and MPC-predicted features is essential for the MaxEnt IRL feature-matching update.

The training process is summarized in Algorithm 7.1. For each expert segment, the MPC problem is solved using the current weight vector. The predicted features are then compared with the expert features, and the weight vector is updated according to the MaxEnt IRL gradient. This process is repeated until the feature difference converges.

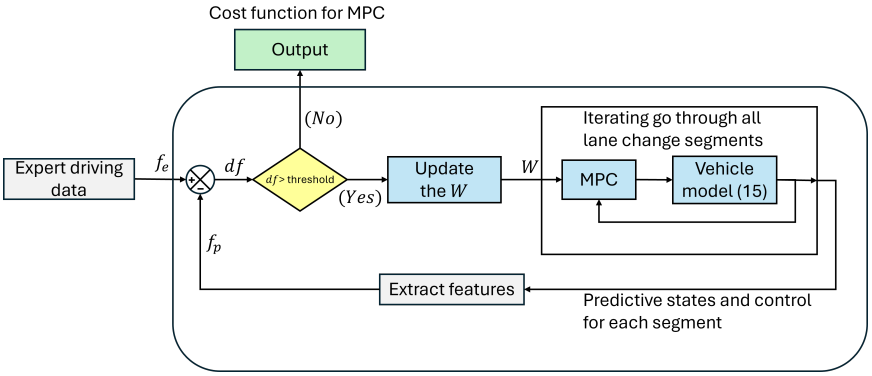

Figure 7.1: Flow chart of the IRL-MPC training process.

7.4 CARLA Simulation Results

7.4.1 Simulation Setup

The proposed IRL-MPC framework is first evaluated in the CARLA simulation environment. Two different driving styles are considered: aggressive and conservative. Expert trajectories are generated using CARLA's built-in autopilot under these two driving styles. The vehicle starts in the first lane, whose lateral range is from -3.6 m to 0 m. The target lane is from 0 m to 3.6 m. The red dashed line in the trajectory plots represents the centerline of the target lane.

A genetic algorithm is used to identify the vehicle parameters for the dynamic model used by MPC. The identified parameters are listed in Table 7.1.

The CARLA evaluation is conducted at a vehicle speed of 60 km/h. The lane change trajectories are segmented into 0.8 s intervals. With an MPC sampling time of 0.2 s, the prediction horizon is set to $p = 4$.

7.4.2 Learned Weights for Different Driving Styles

The learned weights for the aggressive and conservative driving styles are listed in Table 7.2.

Algorithm 7.1: MaxEnt IRL for Personalized MPC Cost Learning

Input: Expert lane change state sequences $S = \{S_1, S_2, \ldots, S_N\}$ and control sequences $U = \{U_1, U_2, \ldots, U_N\}$

Output: Feature weight vector W

Compute $F_i = f(S_i, U_i)$ for $i = 1, \ldots, N$
Compute average expert feature $\bar{F} = \frac{1}{N}\sum_{i=1}^{N} F_i$
Initialize $c \leftarrow 0$, $W \leftarrow$ all-ones vector, and $\eta \leftarrow 0.1$
while *W has not converged* **do**
    for *each initial state $S_i(1)$* **do**
        Solve the MPC OCP using the current W and initial state $S_i(1)$
        Obtain $S_{\text{opt},i}$ and $U_{\text{opt},i}$
        Compute $F_{\text{opt},i} = f(S_{\text{opt},i}, U_{\text{opt},i})$
    end
    Estimate $\mathbb{E}_{p(l|W)}[f] \leftarrow \frac{1}{N}\sum_{i=1}^{N} F_{\text{opt},i}$
    Compute $\nabla L(W) = \mathbb{E}_{p(l|W)}[f] - \bar{F}$
    Update $W \leftarrow W + \eta \frac{\nabla L(W)}{\|\nabla L(W)\|}$
    $c \leftarrow c + 1$
    if $c = 200$ **then**
        $\eta \leftarrow \eta / 2$
        $c \leftarrow 0$
    end
end
return W

The learned weights show clear style-dependent differences. For the aggressive driver, the weights on lateral position deviation and heading angle deviation are larger than those of the conservative driver. This indicates that the aggressive driver places more emphasis on reducing the lateral distance to the target lane and completing the heading adjustment within a shorter maneuver. Meanwhile, the steering effort weight is lower than that of the conservative driver, meaning that larger steering actions are more acceptable when reproducing the aggressive lane change behavior.

Table 7.1: Parameters for the vehicle dynamic model used in CARLA.

Parameter	m (kg)	I (kg·m^2)	L (m)	L_{xf} (m)	L_{xr} (m)	μ	C (deg^{-1})
Value	1265	6481	2.6	2.3	0.3	0.289	3.07

Table 7.2: IRL learned weights for CARLA simulation tests.

Driver	f_l	f_h	f_y	f_s
Aggressive	29.061	10.402	1	30.182
Conservative	6.252	8.971	1	40.618

In contrast, the conservative driver has a larger steering effort weight and smaller lateral and heading deviation weights. This weight combination encourages the MPC controller to avoid large steering commands and generate a flatter lane change maneuver. Therefore, the learned weights are consistent with the intended difference between aggressive and conservative driving styles, and they provide an interpretable explanation of how the learned cost function changes the closed-loop behavior.

7.4.3 Trajectory Comparison

The trajectory comparison first evaluates whether the learned cost function can shape the overall lane change path. Different from a conventional path-tracking MPC, the controller in this chapter does not track a separately generated personalized reference path. Instead, the lane change behavior is produced by minimizing the IRL-learned feature-based cost under the vehicle dynamics and actuator constraints. Therefore, if the IRL-MPC trajectory is close to the expert trajectory, it indicates that the learned weights are sufficient to reproduce the driver's path-level preference in closed-loop control.

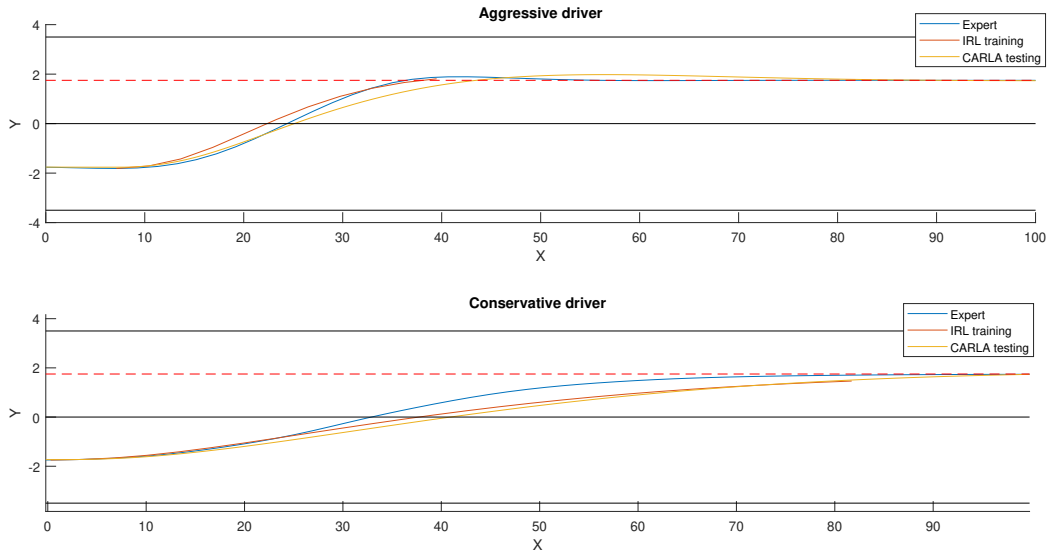

Figure 7.2: Lane change trajectories for different driving styles.

Fig. 7.2 compares the expert trajectories, IRL training results, and CARLA testing results for the aggressive and conservative drivers. The expert path, IRL training path, and CARLA testing path are highly consistent for both driving styles. This demonstrates that the IRL-MPC controller can reproduce distinct lane change styles in closed-loop simulation.

For the aggressive driver, the lane change is sharper and reaches the target lane more quickly. For the conservative driver, the lane change is flatter and more gradual. This distinction confirms that the learned weights encode style-dependent preferences and that these preferences are realized through the MPC controller.

7.4.4 Vehicle States and Control Inputs

Figs. 7.3 and 7.4 compare the expert trajectory, IRL training result, and CARLA testing result in terms of heading angle, yaw rate, and steering command. These plots are

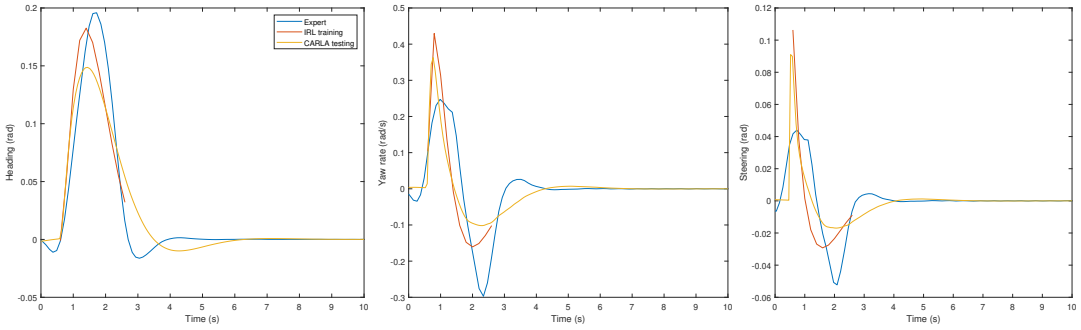

Figure 7.3: Comparison of expert, IRL training, and CARLA testing vehicle states for the aggressive driver.

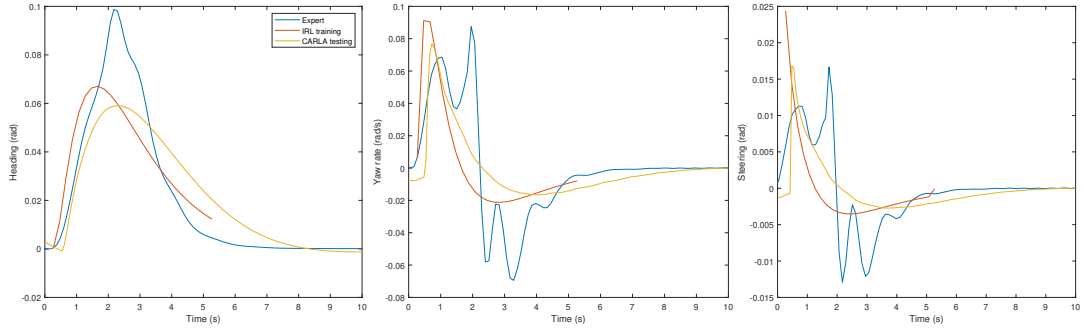

Figure 7.4: Comparison of expert, IRL training, and CARLA testing vehicle states for the conservative driver.

important because they show that the proposed framework does not only reproduce the geometric path. It also produces vehicle motion and control inputs that are consistent with expert behavior.

For the aggressive driver, the IRL training result closely follows the expert behavior, showing a sharper heading peak and larger yaw-rate and steering responses. For the conservative driver, the learned controller produces a smoother heading profile, lower peak yaw rate, and smaller steering variations. The CARLA testing results remain close to

Table 7.3: Features for expert, IRL training, and CARLA testing trajectories under different driving styles.

Driver	Trajectory	f_l	f_h	f_y	f_s
Aggressive	Expert	0.1675	0.0826	0.1549	0.0072
	IRL training	0.1797	0.0883	0.1443	0.0060
	CARLA testing	0.1704	0.0366	0.0747	0.0038
Conservative	Expert	0.1642	0.0114	0.0082	0.0004
	IRL training	0.1662	0.0081	0.0048	0.0002
	CARLA testing	0.1978	0.0076	0.0031	0.0001

the IRL training results, showing that the learned controller can generalize in closed-loop simulation.

7.4.5 Feature-Level Comparison

The averaged feature values for the expert, IRL training, and CARLA testing trajectories are listed in Table 7.3.

For the aggressive driver, the expert trajectory has relatively large heading and yaw-rate features, which is consistent with a faster and sharper lane change. The IRL training features remain close to the expert features, and the CARLA testing trajectory preserves the main style-dependent trend. For the conservative driver, the heading, yaw-rate, and steering effort features are significantly smaller, reflecting a smoother lane change. The learned controller preserves these lower feature values in both training and testing.

The remaining differences between training and CARLA testing are mainly caused by the mismatch between the simplified dynamic model used by MPC and the high-fidelity CARLA vehicle dynamics. Nevertheless, the trajectory, vehicle-state, steering-input, and feature-level comparisons consistently show that the learned cost function captures the main characteristics of the expert demonstrations and reproduces

different lane change styles in closed-loop simulation. These results motivate the on-road validation presented in the next section.

7.5 Truck On-Road Test

7.5.1 Experimental Setup

To further evaluate the real-world feasibility of the proposed IRL-MPC framework, on-road experiments are conducted using an ISUZU NPR-HD truck equipped with a drive-by-wire system and GPS for autonomous vehicle control. Compared with CARLA simulation, the truck experiment introduces additional practical factors such as sensor noise, actuator delay, road-surface variation, and mismatch between the prediction model and the actual vehicle dynamics. Therefore, this test is used to examine whether the IRL-learned cost weights can be transferred from demonstration learning to real vehicle control.

For safety, the real-world test is conducted on clear roads at a low speed of 20 km/h. During the experiment, a human driver first executes a right lane change to provide the expert demonstration. The recorded expert lane change path is then used for IRL training. To evaluate generalization to an unseen initial condition, the human demonstration starts slightly away from the lane centerline, while the MPC-controlled truck starts closer to the lane centerline during autonomous execution. This difference allows the test to evaluate whether the learned controller can preserve the demonstrated driving preference while adapting the actual maneuver to the current vehicle state.

7.5.2 Learned Weights and On-Road Trajectory

The learned weights for the truck driver are listed in Table 7.4.

The learned weight on lateral position deviation is the largest among the four features, indicating that the demonstrated driver behavior places strong emphasis on

Table 7.4: IRL learned weights for the truck driver.

	f_l	f_h	f_y	f_s
W	8.428	2.559	1	2.568

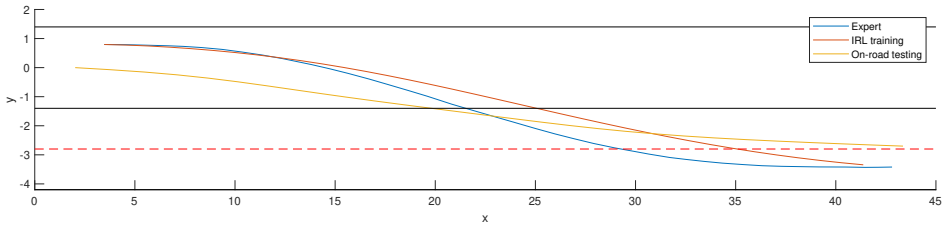

Figure 7.5: Lane change trajectories for on-road testing.

completing the lateral transition toward the target lane. The heading deviation and steering effort weights are smaller but still significant, meaning that the learned controller also considers vehicle orientation and control effort during the maneuver. The yaw-rate weight is normalized to one, and the other weights are interpreted relative to this value.

These learned weights are incorporated into the MPC cost function and used to control the truck for the lane change maneuver. Fig. 7.5 compares the expert path, the IRL training path, and the on-road testing path. The expert path is shown in blue, the IRL training path is shown in red, and the on-road testing path is shown in yellow.

The IRL training path closely follows the expert trajectory, indicating that the learned weights reproduce the demonstrated lane change behavior when the initial condition is aligned with the expert data. The on-road testing path does not exactly overlap with the expert trajectory because the autonomous truck starts from a different lateral position. Nevertheless, the IRL-MPC controller successfully completes the lane

Table 7.5: Features for expert, IRL training, and on-road testing trajectories.

Trajectory	f_l	f_h	f_y	f_s
Expert	0.2092	0.0782	0.0179	0.0088
IRL training	0.2475	0.0864	0.0221	0.0124
On-road testing	0.1087	0.0294	0.0025	0.0077

change maneuver and reaches the target lane along a comparable path. The total distance of the maneuver is approximately 42 m. This result demonstrates that the learned driving preference is not limited to memorizing a single demonstrated path. Instead, when embedded into MPC, the learned cost function can adapt the lane change maneuver to the current vehicle state during real vehicle control.

7.5.3 On-Road Feature Comparison

The averaged feature values for the expert trajectory, IRL training trajectory, and on-road testing trajectory are listed in Table 7.5.

The IRL training feature values are close to the expert feature values, confirming that the IRL process captures the driver's lane change style during training. The on-road testing feature values are lower than the expert and IRL training values, especially for heading deviation and yaw rate effort. This is expected because the autonomous test starts closer to the lane centerline than the expert demonstration. As a result, the truck requires less heading correction, smaller yaw-rate response, and reduced steering effort to complete the lane change.

Therefore, the lower on-road testing features do not indicate that the learned controller fails to reproduce the driver preference. Instead, they reflect the feedback nature of MPC, the learned cost function provides the preference structure, while MPC adapts the actual control sequence according to the current vehicle state and constraints. The

truck test therefore demonstrates both real-world feasibility and generalization to an unseen initial condition.

7.6 Discussion

The results in this chapter show the difference between learning a personalized path and learning a personalized controller. In Chapter 6, IRL was used to learn a path-level cost function, and the learned weights generated Bezier lane change paths similar to the demonstrations. In this chapter, the learned weights are placed inside the MPC cost function, so they affect the steering command through closed-loop optimization.

This change is important because the controller does not need to reproduce the expert trajectory point by point. Instead, it uses the learned feature weights to decide how to balance lateral convergence, heading alignment, yaw response, and steering effort under the current vehicle state and constraints. This explains why the CARLA testing trajectories and the truck on-road trajectory do not match the expert features exactly, while still preserving the intended driving style. The learned cost provides the preference structure, and MPC adapts the actual motion to the current initial condition, vehicle dynamics, and actuator limits.

The CARLA tests show that different learned weights lead to different lane change behaviors. The aggressive case produces a faster lateral transition with larger heading and yaw-rate responses, while the conservative case produces a smoother and flatter maneuver. The truck test further shows that the same idea can be implemented on a vehicle platform. Although the on-road feature values are lower than the expert values because the test starts closer to the target lane centerline, the controller still completes the lane change according to the learned cost structure. This supports the use of IRL-learned weights as a practical way to reduce manual MPC cost tuning for personalized lane change control.

7.7 Summary

This chapter developed a personalized MPC framework using MaxEnt IRL. Different from the path-level method in Chapter 6, the learned weights were embedded directly into the MPC cost function, allowing the controller to generate personalized lane change behavior through closed-loop optimization.

The framework was evaluated in CARLA simulation and truck on-road testing. In CARLA, the learned weights reproduced aggressive and conservative lane change styles in trajectory shape, heading response, yaw rate, and steering command. In the truck test, the learned MPC controller completed a personalized lane change from an unseen initial condition. These results show that IRL-learned preferences can be transferred into an MPC-based controller and used to reduce manual cost-weight tuning for personalized autonomous lane change control.

CHAPTER EIGHT

CONCLUSIONS AND FUTURE WORK

This dissertation investigated model predictive control (MPC)-based autonomous vehicle lateral motion control, with emphasis on computational efficiency, real-world implementation, and personalized lane change behavior learning. The dissertation first established the vehicle modeling and MPC formulation used for autonomous vehicle path tracking. Based on this foundation, event-triggered MPC was developed to reduce unnecessary online optimization while maintaining reliable tracking performance. The dissertation then extended the MPC framework toward personalized autonomous driving by using inverse reinforcement learning (IRL) to learn driver-specific lane change preferences and embed them into both path generation and closed-loop MPC control.

8.1 Summary of the Dissertation

Chapter 3 presented the vehicle modeling and baseline MPC formulation. Both kinematic and dynamic bicycle models were introduced to describe vehicle lateral motion under different driving conditions. The kinematic model provides a simple and numerically stable representation for low-speed operation, while the dynamic model captures lateral velocity, yaw-rate dynamics, and tire-force effects for higher-speed driving. The constrained MPC formulation was then developed based on the vehicle model, reference path, steering constraints, and steering-rate constraints. This chapter provided the modeling and control foundation for the rest of the dissertation.

Chapter 4 developed the event-triggered MPC framework for autonomous vehicle path tracking. Different from conventional time-triggered MPC, which solves the optimal control problem at every sampling instant, the event-triggered controller solves a new optimization problem only when necessary. The triggering condition is mainly determined

by the lateral offset from the reference path and the availability of the previously optimized control sequence. This design reduces unnecessary optimization while preserving feedback correction when tracking error becomes significant. An inter-event feedback method based on a locally identified K-matrix was also introduced to improve control performance during non-triggered intervals.

Chapter 5 evaluated the proposed event-triggered MPC framework in simulation and real-world experiments. The CARLA simulation results showed that event-triggered MPC can significantly reduce the number of optimization calls while maintaining acceptable path-tracking performance. Real-vehicle tests further demonstrated that the proposed controller can be implemented on a full-size autonomous vehicle platform. The experimental results also revealed an important practical insight: event-triggered MPC can sometimes improve tracking performance compared with time-triggered MPC because it reduces computation-induced control delay when no new optimization is triggered.

Chapter 6 focused on personalized lane change path modeling. A fifth-order Bezier curve was used to represent lane change trajectories, and MaxEnt IRL was used to learn driver-specific feature weights from expert demonstrations. The designed features describe curvature-based comfort, lane change length, lane-mark crossing distance, and final lateral position. The results showed that the learned weights can generate lane change paths that are close to expert trajectories under unseen initial conditions, demonstrating the ability of IRL to capture personalized path-level driving preferences.

Chapter 7 further integrated IRL-learned preferences into closed-loop MPC. Instead of using IRL only for offline path generation, the learned feature weights were directly embedded into the MPC cost function. The personalized MPC controller was evaluated in CARLA under aggressive and conservative driving styles and further tested on an ISUZU truck. The results showed that the learned controller can reproduce

style-dependent lane change behavior not only at the trajectory level, but also in vehicle states and steering-input responses.

8.2 Key Findings

Several findings can be drawn from the simulation, full-size vehicle, scale-vehicle, and truck experiments. First, event-triggered MPC can reduce the number of online optimizations while maintaining acceptable path-tracking performance. The CARLA simulation results show the expected trade-off between triggering frequency and tracking accuracy: larger triggering thresholds reduce computation but generally increase tracking error. However, the real-vehicle experiments show that this trade-off is affected by implementation delay. In physical vehicle testing, solving the MPC problem at every sampling instant does not always lead to better tracking, because the computed steering command may be delayed before it reaches the actuator. Under such conditions, event-triggered MPC can provide more timely inter-event commands and may achieve tracking performance comparable to, or better than, time-triggered MPC.

Second, the behavior of the controller between two triggered optimizations is important for practical event-triggered MPC. Simply shifting the previous optimal input sequence is computationally efficient, but it provides limited feedback correction during non-triggered intervals. The linear based inter-event controller addresses this issue by extracting a local state-input relationship from the latest MPC solution and using the current measured state to update the steering command. The QCar2 experiments show that this lightweight feedback mechanism can improve the inter-event behavior of event-triggered MPC without requiring a full MPC optimization at every sampling step.

Third, the IRL-based personalization studies show that driver-specific lane change preferences can be learned from demonstration data and represented through interpretable feature weights. At the path level, the Bezier-curve-based IRL framework generates lane

change paths that remain close to expert trajectories under unseen initial conditions. At the control level, embedding the learned feature weights directly into the MPC cost function allows the controller to generate personalized lane change behavior in closed loop while still considering vehicle dynamics and steering constraints. The CARLA and truck testing results demonstrate that the learned MPC controller can reproduce different lane change styles and adapt to new initial conditions.

8.3 Limitations and Future Work

Although the proposed methods were validated through simulation and real-world experiments, several limitations remain. The event-triggered MPC framework mainly focuses on lateral path tracking with a predefined reference path. Longitudinal motion is either controlled separately or treated as a measured input. Therefore, the interaction between lateral event-triggered MPC and longitudinal speed control has not been fully investigated.

The current event-triggering condition is also based on a fixed lateral-offset threshold and a prediction-horizon counter. This design is simple and practical, but a fixed threshold may not be optimal for all driving conditions. Straight driving, lane changes, and sharp turns have different control requirements. Future work can develop adaptive event-triggered MPC, where the triggering threshold changes according to vehicle speed, road curvature, tracking error, and maneuver type.

Another limitation is that the current event-triggered MPC framework is mainly evaluated experimentally. Although the results demonstrate practical effectiveness, further theoretical analysis is needed to study recursive feasibility, closed-loop stability, and robustness under model mismatch, actuator delay, sensor noise, and external disturbances. Future work can also further improve the linear based inter-event feedback method by considering more robust identification methods or nonlinear feedback structures.

For personalized lane change learning, the current IRL-based framework focuses on relatively simple lane change scenarios. Surrounding vehicles, collision avoidance constraints, and interactive decision-making are not fully considered. Future work can extend the feature set to include safety-related and traffic-interaction features, such as time-to-collision, relative distance, relative velocity, and available lane-change gap. This would allow the learned controller to represent not only individual comfort and efficiency preferences but also interaction behavior in traffic.

The IRL-MPC framework can also be extended from lateral lane change control to integrated longitudinal and lateral control. A more complete personalized controller could learn driver preferences in speed adjustment, acceleration comfort, lane change timing, and steering behavior at the same time. In addition, larger and more diverse real-world driving datasets are needed to improve the generalization of the learned models across different drivers, speeds, road geometries, vehicle platforms, and traffic conditions.

REFERENCES

[1] 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.

[2] M. Martínez-Díaz and F. Soriguera, "Autonomous vehicles: theoretical and practical challenges," *Transportation Research Procedia*, vol. 33, pp. 275–282, 2018.

[3] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[4] F. Ma, Y. Shen, J. Nie, X. Li, Y. Yang, J. Wang, and G. Wu, "Trajectory planning and tracking for four-wheel-steering autonomous vehicle with v2v communication," tech. rep., SAE Technical Paper, 2020.

[5] R. Rajamani, *Vehicle Dynamics and Control*. Mechanical Engineering Series, Springer US, 2012.

[6] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium*, (Seoul, Korea), pp. 1094–1099, June 28–July 1, 2015.

[7] P. Falcone, M. Tufo, F. Borrelli, J. Asgari, and H. E. Tseng, "A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems," in *2007 IEEE Conference on Decision and Control*, (New Orleans, LA), pp. 2980–2985, Deceber 12–14, 2007.

[8] S. Di Cairano, H. E. Tseng, D. Bernardini, and A. Bemporad, "Vehicle yaw stability control by coordinated active front steering and differential braking in the tire sideslip angles domain," *IEEE Trans. Control Syst. Tech.*, vol. 21, no. 4, pp. 1236–1248, 2012.

[9] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 174–179, IEEE, 2017.

[10] S. Yu, M. Hirche, Y. Huang, H. Chen, and F. Allgöwer, "Model predictive control for autonomous ground vehicles: a review," *Autonomous Intelligent Systems*, vol. 1, no. 1, p. 4, 2021.

[11] M. Rokonuzzaman, N. Mohajer, S. Nahavandi, and S. Mohamed, "Model predictive control with learned vehicle dynamics for autonomous vehicle path tracking," *IEEE Access*, vol. 9, pp. 128233–128249, 2021.

[12] Z. Zhou, J. Chen, M. Tao, P. Zhang, and M. Xu, "Experimental validation of event-triggered model predictive control for autonomous vehicle path tracking," in *2023 IEEE International Conference on Electro Information Technology*, (Romeoville, IL), May 18–20, 2023.

[13] J. B. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[14] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[15] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017.

[16] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat, "MPC-based approach to active steering for autonomous vehicle systems," *Int. Journal of Vehicle Autonomous Systems*, vol. 3, pp. 265–291, 2005.

[17] A. Bemporad, F. Borrelli, M. Morari, *et al.*, "Model predictive control based on linear programming˜ the explicit solution," *IEEE Transactions on Automatic Control*, vol. 47, no. 12, pp. 1974–1985, 2002.

[18] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[19] R. Yu, H. Guo, Z. Sun, and H. Chen, "MPC-based regional path tracking controller design for autonomous ground vehicles," in *IEEE International Conference on Systems, Man, and Cybernetics*, (Hong Kong, China), pp. 2510–2515, October 09-12, 2015.

[20] F. Mohseni, E. Frisk, and L. Nielsen, "Distributed cooperative mpc for autonomous driving in different traffic scenarios," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 299–309, 2021.

[21] M. Kim, D. Lee, J. Ahn, M. Kim, and J. Park, "Model predictive control method for autonomous vehicles using time-varying and non-uniformly spaced horizon," *IEEE Access*, vol. 9, pp. 86475–86487, 2021.

[22] D. Chu, H. Li, C. Zhao, and T. Zhou, "Trajectory tracking of autonomous vehicle based on model predictive control with pid feedback," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2239–2250, 2022.

[23] M. Jost, G. Pannocchia, and M. Mönnigmann, "Online constraint removal: accelerating MPC with a lyapunov function," *Automatica*, vol. 57, pp. 164–169, 2015.

[24] D. Liao-McPherson, M. M. Nicotra, A. L. Dontchev, I. V. Kolmanovsky, and V. Veliov, "Sensitivity-based warmstarting for nonlinear model predictive control with polyhedral state and control constraints," *IEEE Transactions on Automatic Control*, 2019.

[25] S. Gros and M. Zanon, "Learning for mpc with stability & safety guarantees," *Automatica*, vol. 146, p. 110598, 2022.

[26] Q. Gong, J. Xu, J. Ye, H. Feng, and A. Shen, "Nonlinear model predictive control for premixed turbocharged natural gas engine," *IEEE/ASME Trans. Mechatronics*, 2021.

[27] Z. Zhou, M. Tao, J. Qiu, P. Zhang, M. Xu, and J. Chen, "Autonomous vehicle path tracking using event-triggered mpc with switching model: Methodology and real-world validation," *IET Control Theory & Applications*, vol. 19, no. 1, p. e70046, 2025.

[28] Z. Zhou, C. Rother, and J. Chen, "Event-triggered model predictive control for autonomous vehicle path tracking: Validation using CARLA simulator," *IEEE Transactions on Intelligent Vehicles*, vol. 8, pp. 3547–3555, June 2023.

[29] W. P. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *IEEE Conference on Decision and Control*, pp. 3270–3285, IEEE, 2012.

[30] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Event-triggered control for discrete-time systems," in *Proceedings of the 2010 american control conference*, pp. 4719–4724, IEEE, 2010.

[31] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Novel event-triggered strategies for model predictive controllers," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, (Orlando, FL), pp. 3392–3397, December 12-15, 2011.

[32] H. Li and Y. Shi, "Event-triggered robust model predictive control of continuous-time nonlinear systems," *Automatica*, vol. 50, no. 5, pp. 1507–1513, 2014.

[33] F. D. Brunner, W. Heemels, and F. Allgöwer, "Robust event-triggered mpc for constrained linear discrete-time systems with guaranteed average sampling rate," *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 117–122, 2015.

[34] F. D. Brunner, M. Heemels, and F. Allgöwer, "Robust self-triggered MPC for constrained linear systems: A tube-based approach," *Automatica*, vol. 72, pp. 73–83, 2016.

[35] F. D. Brunner, W. Heemels, and F. Allgöwer, "Robust event-triggered MPC with guaranteed asymptotic bound and average sampling rate," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5694–5709, 2017.

[36] X. Chen and F. Hao, "Event-triggered average consensus control for discrete-time multi-agent systems," *IET Control Theory & Applications*, vol. 6, no. 16, pp. 2493–2498, 2012.

[37] X.-M. Zhang and Q.-L. Han, "Event-triggered dynamic output feedback control for networked control systems," *IET Control Theory & Applications*, vol. 8, no. 4, pp. 226–234, 2014.

[38] T. Bai, A. Johansson, K. H. Johansson, and J. Mårtensson, "Event-triggered distributed model predictive control for platoon coordination at hubs in a transport system," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 1198–1204, IEEE, 2021.

[39] J. Liu, Z. Wang, and L. Zhang, "Event-triggered vehicle-following control for connected and automated vehicles under nonideal vehicle-to-vehicle communications," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, pp. 342–347, IEEE, 2021.

[40] S. Huang and J. Chen, "Event-triggered model predictive control for autonomous vehicle with rear steering," *SAE Technical Paper*, no. 2022-01-0877, 2022.

[41] Y. Wang, Y. Yan, T. Shen, S. Bai, J. Hu, L. Xu, and G. Yin, "An event-triggered scheme for state estimation of preceding vehicles under connected vehicle environment," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 583–593, 2023.

[42] H. Yang, Q. Li, Z. Zuo, and H. Zhao, "Event-triggered model predictive control for multi-vehicle systems with collision avoidance and obstacle avoidance," *International Journal of Robust and Nonlinear Control*, vol. 31, no. 11, pp. 5476–5494, 2021.

[43] S. Xiao, X. Ge, Q.-L. Han, and Y. Zhang, "Resource-efficient platooning control of connected automated vehicles over vanets," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 579–589, 2022.

[44] J. Zhang, Y. Lu, Z. Wang, Y. Hu, and Y. Wu, "Event-triggered model predictive-preview control strategy for trajectory tracking of autonomous vehicle,"

Transactions of the Institute of Measurement and Control, p. 01423312241229386, 2024.

[45] F. Dang, D. Chen, J. Chen, and Z. Li, "Event-triggered model predictive control with deep reinforcement learning," *IEEE Transactions on Intelligent Vehicles*. Accepted for Publication, October 2023.

[46] C. Zhu, J. Chen, M. Iwasaki, and H. Zhang, "Event-triggered deep learning control of quadrotors for trajectory tracking," *IEEE Transactions on Industrial Electronics*, 2023.

[47] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *1st Conference on Robot Learning*, (Mountain View, CA), pp. 1–16, 2017.

[48] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[49] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[50] "The multidimensional driving style inventory—scale construct and validation," *Accident Analysis & Prevention*, vol. 36, no. 3, pp. 323–332, 2004.

[51] C. Chai, X. Shi, Z. Zhou, X. Zeng, W. Yin, and M. M. Islam, "Driving style recognition based on naturalistic driving: Volatilities, decision-making, and safety performances," *User Experience Design in the Era of Automated Driving*, pp. 359–394, 2022.

[52] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *IEEE Int. Conf. Robotics and Auto.*, pp. 2641–2646, IEEE, 2015.

[53] Z. Liu, Z. Wang, B. Yang, and K. Nakano, "Learning personalized discretionary lane-change initiation for fully autonomous driving based on reinforcement learning," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 457–463, IEEE, 2020.

[54] Y. Ma and J. Wang, "Personalized driving behaviors and fuel economy over realistic commute traffic: Modeling, correlation, and prediction," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7084–7094, April 2022.

[55] Y. Wang, G. Zhu, Y. Zhang, and G. Wen, "Personalized lane-changing behavior decision model considering driving habits," in *2024 39th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pp. 2251–2256, IEEE, 2024.

[56] J. Chen, P. Zhao, T. Mei, and H. Liang, "Lane change path planning based on piecewise bezier curve for autonomous vehicle," in *Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety, Dongguan, China, July 28-30, 2013*, pp. 17–22.

[57] Y.-G. Choi, K.-I. Lim, and J.-H. Kim, "Lane change and path planning of autonomous vehicles using gis," in *12th International Conference on Ubiquitous Robots and Ambient Intelligence*, pp. 163–166, IEEE, 2015.

[58] C. Ma and D. Li, "A review of vehicle lane change research," *Physica A: Statistical Mechanics and its Applications*, vol. 626, p. 129060, September 2023.

[59] Y. Zhou, R. Fu, and C. Wang, "Learning the car-following behavior of drivers using maximum entropy deep inverse reinforcement learning," *Journal of advanced transportation*, vol. 2020, no. 1, p. 4752651, 2020.

[60] G. Li, Z. Ji, S. Li, X. Luo, and X. Qu, "Driver behavioral cloning for route following in autonomous vehicles using task knowledge distillation," *IEEE Transactions on Intelligent Vehicles*, vol. 8, pp. 1025–1033, February 2023.

[61] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[62] J. Chen, X. Meng, and Z. Li, "Reinforcement learning-based event-triggered model predictive control for autonomous vehicle path following," in *American Control Conference*, (Atlanta, GA), June 8–10, 2022.

[63] A. Irshayyid, J. Chen, and G. Xiong, "A review on reinforcement learning-based highway autonomous vehicle control," *Green Energy and Intelligent Transportation*, vol. 3, no. 4, p. 100156, August 2024.

[64] A. Y. Ng, S. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[65] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[66] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with gaussian processes," *Advances in neural information processing systems*, vol. 24, 2011.

[67] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.

[68] Z. Huang, J. Wu, and C. Lv, "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 10239–10251, August 2022.

[69] T. Phan-Minh, F. Howington, T.-S. Chu, S. U. Lee, M. S. Tomov, N. Li, C. Dicle, S. Findler, F. Suarez-Ruiz, R. Beaudoin, *et al.*, "Driving in real life with inverse reinforcement learning," *arXiv preprint arXiv:2206.03004*, 2022.

[70] J. Liu, L. N. Boyle, and A. G. Banerjee, "An inverse reinforcement learning approach for customizing automated lane change systems," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 9261–9271, September 2022.

[71] L. Wang, L. Sun, M. Tomizuka, and W. Zhan, "Socially-compatible behavior design of autonomous vehicles with verification on real human data," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3421–3428, 2021.

[72] Z. Wu, F. Qu, L. Yang, and J. Gong, "Human-like decision making for autonomous vehicles at the intersection using inverse reinforcement learning," *Sensors*, vol. 22, no. 12, p. 4500, June 2022.

[73] H. Li, Y. Luo, and J. Wu, "Collision-free path planning for intelligent vehicles based on bézier curve," *IEEE Access*, vol. 7, pp. 123334–123340, August 2019.

[74] B. S. Oldaç, E. E. Özdemir, and F. Kosova, "A path tracking and collision prevention control system for an electric vehicle with trajectories generated by bezier curves," in *2024 11th International Conference on Electrical and Electronics Engineering (ICEEE), Marmaris, Turkiye, April 22-24, 2024*, pp. 146–151.

[75] J.-w. Choi, R. Curry, and G. Elkaim, "Path planning based on bézier curve for autonomous ground vehicles," in *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pp. 158–166, IEEE, 2008.

[76] Z. Zhou and J. Chen, "Modeling driver lane change behavior using inverse reinforcement learning," in *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)*, pp. 1–5, IEEE, 2024.

[77] Z. Zhou, M. Tao, J. Qiu, P. Zhang, M. Xu, Y. Sun, and J. Chen, "Personalized mpc for autonomous vehicle lateral motion control via inverse reinforcement learning," in *Proceedings of the American Control Conference*, 2026.

[78] J. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[79] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Trans. on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[80] A. Wischnewski, T. Herrmann, F. Werner, and B. Lohmann, "A tube-mpc approach to autonomous multi-vehicle racing on high-speed ovals," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 368–378, 2023.

[81] C. Shen, H. Guo, F. Liu, and H. Chen, "MPC-based path tracking controller design for autonomous ground vehicles," in *2017 36th Chinese Control Conference (CCC)*, pp. 9584–9589, IEEE, 2017.

[82] H. Yang, Z. Wang, Y. Xia, and Z. Zuo, "Empc with adaptive apf of obstacle avoidance and trajectory tracking for autonomous electric vehicles," *ISA transactions*, 2022.

[83] Y. Cui, J. Tang, Q. Luo, Z. Feng, and T. Huang, "A nash-stackelberg game theoretic planner for many-to-few multi-vehicle racing," *IEEE Transactions on Intelligent Vehicles*, 2024.

[84] T. Besselmann, J. Lofberg, and M. Morari, "Explicit MPC for lpv systems: Stability and optimality," *IEEE Transactions on Automatic Control*, vol. 57, no. 9, pp. 2322–2332, 2012.

[85] F. Tian, Z. Li, F.-Y. Wang, and L. Li, "Parallel learning-based steering control for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 379–389, 2022.

[86] W. Jallet, E. Dantec, E. Arlaud, N. Mansard, and J. Carpentier, "Parallel and proximal constrained linear-quadratic methods for real-time nonlinear mpc," *Robotics: Science and Systems XX*, 2024.

[87] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *Int. J. Adv. Manuf. Tech.*, vol. 117, no. 5, pp. 1327–1349, 2021.

[88] Z. Zhou, J. Chen, M. Tao, P. Zhang, and M. Xu, "Experimental validation of event-triggered model predictive control for autonomous vehicle path tracking," in *2023 IEEE International Conference on Electro Information Technology*, (Romeoville, IL), May 18–20, 2023.

[89] M. Donkers, W. Heemels, D. Bernardini, A. Bemporad, and V. Shneer, "Stability analysis of stochastic networked control systems," *Automatica*, vol. 48, no. 5, pp. 917–925, 2012.

[90] K. Zou, Y. Cai, L. Chen, and X. Sun, "Event-triggered nonlinear model predictive control for trajectory tracking of unmanned vehicles," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, p. 0954407021992165, 2021.

[91] F. Dang, D. Chen, J. Chen, and Z. Li, "Event-triggered model predictive control with deep reinforcement learning," *IEEE Transactions on Intelligent Vehicles*, vol. 9, pp. 459–468, January 2024.

[92] H. Li, W. Yan, and Y. Shi, "Triggering and control codesign in self-triggered model predictive control of constrained systems: With guaranteed performance," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 4008–4015, 2018.

[93] C. Liu, H. Li, Y. Shi, and D. Xu, "Codesign of event trigger and feedback policy in robust model predictive control," *IEEE Transactions on Automatic Control*, vol. 65, no. 1, pp. 302–309, 2019.

[94] B. T. Lopez, J.-J. E. Slotine, and J. P. How, "Dynamic tube mpc for nonlinear systems," in *2019 American Control Conference (ACC)*, pp. 1655–1662, 2019.

[95] E. V. Kumar and J. Jerome, "Robust lqr controller design for stabilizing and trajectory tracking of inverted pendulum," *Procedia Engineering*, vol. 64, pp. 169–178, 2013.

[96] G. S. Watson, "Linear least squares regression," *The Annals of Mathematical Statistics*, pp. 1679–1699, 1967.

[97] R. Lattarulo, L. González, E. Martí, J. Matute, M. Marcano, and J. Pérez, "Urban motion planning framework based on n-bézier curves considering comfort and safety," *Journal of Advanced Transportation*, vol. 2018, July 2018.

[98] Z. Zhou and J. Chen, "Privacy-preserving personalized autonomous vehicle lane change using inverse reinforcement learning," *IEEE Transactions on Vehicular Technology*, 2025.

[99] M. Samuel, M. Hussein, and M. B. Mohamad, "A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle," *International Journal of Computer Applications*, vol. 135, no. 1, pp. 35–38, 2016.

[100] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[101] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[102] Y. Xia, Z. Qu, Z. Sun, and Z. Li, "A human-like model to understand surrounding vehicles' lane changing intentions for autonomous driving," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4178–4189, May 2021.

[103] S. Teng, L. Chen, Y. Ai, Y. Zhou, Z. Xuanyuan, and X. Hu, "Hierarchical interpretable imitation learning for end-to-end autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 673–683, 2023.

[104] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen, "A survey on trajectory-prediction methods for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 652–674, 2022.

[105] E. Zerdali, M. Rivera, and P. Wheeler, "A review on weighting factor design of finite control set model predictive control strategies for ac electric drives," *IEEE Transactions on Power Electronics*, vol. 39, no. 8, pp. 9967–9981, 2024.

[106] S. Ibrahim, M. Mostafa, A. Jnadi, H. Salloum, and P. Osinenko, "Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications," *IEEE Access*, vol. 12, pp. 175473–175500, November 2024.

[107] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning.," in *IJCAI*, vol. 7, pp. 2586–2591, 2007.

[108] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, August 2021.

[109] P. Zhang, S. Xie, X. Lu, Z. Zhong, and Q. Li, "Maximum entropy inverse reinforcement learning-based trajectory planning for autonomous driving," in *Third International Conference on High Performance Computing and Communication Engineering*, pp. 146–151, 2024.

[110] M. F. Ozkan and Y. Ma, "Personalized adaptive cruise control and impacts on mixed traffic," in *2021 American Control Conference (ACC)*, pp. 412–417, IEEE, 2021.

[111] Y. Xing, C. Lv, and D. Cao, "Personalized vehicle trajectory prediction based on joint time-series modeling for connected vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1341–1352, 2019.

[112] S. Yang, H. Zheng, J. Wang, and A. E. Kamel, "A personalized human-like lane-changing trajectory planning method for automated driving system," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 7, pp. 6399–6414, July 2021.

[113] Z. Zhou and J. Chen, "Event-triggered mpc with linear inter-event control for av path tracking," *IEEE Robotics and Automation Letters*, 2026.