Automatic tuning of an MPCC-based Motion Planner

Master Thesis

Weiming Chen

Delft University of Technology

TU Delft

Automatic tuning of an MPCC-based Motion Planner

by

Weiming Chen

to obtain the degree of Master of Science
in Mechanical Engineering
at the Delft University of Technology,

Student number:	5015006	
Supervisors:	Dr. L. Ferranti,	TU Delft-CoR
	OM (Oscar) de Groot,	TU Delft-CoR

TU Delft

Abstract

Research and development of motion control in the field of autonomous driving is significantly increasing nowadays. The model predictive control (MPC) is one of the most powerful and practical tools currently available. It is important to select the parameters of the MPC, such as weights, in a way so that different control objectives can be met within the desired performance constraints. The tuning procedure of the MPC variables can be achieved automatically by employing appropriate approaches. To make the tuning process more robust to different scenarios, one approach is to choose a decision-making architecture that provides guidance. This thesis therefore aims at developing a system integrating an automatic tuning method with a decision-making module. In order to achieve the objectives, Genetic Algorithm and Behavior Tree are employed on top of an existing motion planner. The motion planner is based on model predictive contouring control (MPCC) and the proposed method is tested in CARLA simulation environment. This report highlights the limitations of the proposed automatic tuning method and gives concrete recommendations on how to deal with the shortcomings.

Weiming Chen
Guangzhou, January 2022

Contents

List of Figures

List of Tables

1

Introduction

Research and development projects have been conducted extensively in the field of autonomous driving [1]. As autonomous vehicles become more common on public roads, fewer accidents, more cost-effective transportation, and reduced traffic congestion in urban areas are expected as a result. As promising as the benefits of autonomous driving are, there are also numerous challenges to confront, including safety, reliability, and adaptability of the autonomous system itself. There are critical tasks that autonomous vehicles must perform, including path planning and executing motion through an environment shared with traffic participants, and achieving robust performance using feedback controls. The stability, accuracy, and speed of control have been improved through implementation of different approaches [2]. One of the most powerful and practically useful approaches is model predictive control (MPC) [3]. This method offers the advantage of being able to make predictions about the immediate future under constraints over a given horizon.

A significant impact is exerted on control performance when there are weights for corresponding cost terms in a MPC problem. Generally, the fixed weights used in autonomous vehicle navigation are selected based on empirical knowledge. Such weight selection exploits the advantages of MPC and takes stability properties into account. The question remains, however, if this method fully utilizes the advantages of the MPC. Moreover, this strategy may not allow good generalization across a number of different scenarios.

1.1. Problem Description

In order to implement MPC, the dynamics of the system must be explicitly modelled. As an opposite example, proportional integral derivative (PID) control [4] does not require such explicit requirements. However, the tuning of MPC is still considered more burdensome. For example, for a single-input single-output (SISO) system or a decoupled multi-input multi-output (MIMO) system, each PID controller only has at most 3 gains to adjust, these gains have an intuitive connection to the characteristics of the system response in time domain. As a comparison, in MPC, the number of tuning parameters scales significantly with a quadratic-cost formulation. In addition, the response of the controlled system does not appear to have any obvious intuitive relationship between the parameters and resulting control performance. Consequently, significant calibration effort could be expended when tuning MPC.

Consequently, one of the most challenging aspects of designing MPC is to select the tuning parameters appropriately to achieve the control objectives within the desired performance constraints. Most commonly, these parameters are selected based on a particular criterion, for instance, in a manner that accounts for the stability of the system and exploits the controller's strengths. In autonomous driving, human preferences could also be heavily influential. Developing such a controller involves a large number of scenarios and uncertainties. It should be noted that static tuning parameters are not robust to state variations and other uncertainties. Additionally, the selection of these parameters on the basis of trial-and-error experiments may not be straightforward and may require considerable expertise. One of the challenges generated by this is to make the tuning procedures more robust to

various situations. A second challenge is determining how the preferences of the human calibrator can be learned and reproduced in the tuning process.

To make the tuning process of MPC be more robust to different scenarios, one approach is to treat it as a human-level driving behavior with regard to the surrounding environmental information, which can be greatly aided by choosing a decision-making system. In order to understand the role of a decision-making system, the general structure of an general autonomous driving system architecture can first be introduced, as shown in Figure 1.1. Accordingly, the autonomous vehicle is expected to operate independently in order to accomplish its mission as well as cooperate with other road users. Decision-making has an especially important function in this structure since it is responsible for translating intentions into actions. This is accomplished by evaluating the change of the current situation and comparing the possible actions in order to select the most appropriate one. As a general rule, the optimal action generated from decision-making should account the motion constraints to ensure efficient operation of an autonomous vehicle [5]. In this manner, the tuning process for MPC can be considered a part of the vehicle behavior, since it affects the specific maneuvers of the vehicle.

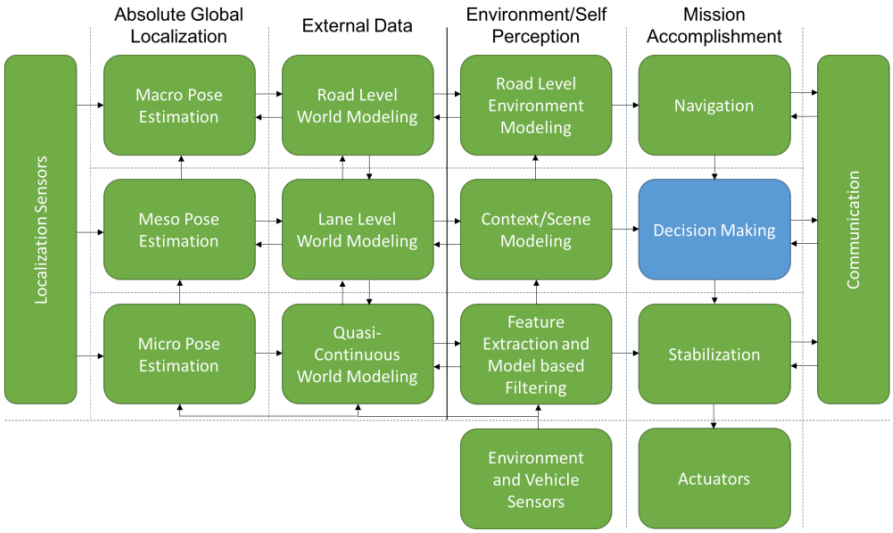


Figure 1.1: Autonomous Driving System Architecture [6]

This thesis is primarily motivated by addressing the challenges described above. As a result, the proposed work should include a decision-making architecture that changes the operation in the tuning process along with a tuning method that is able to minimize the tuning effort required for the implementation of MPC in practice.

1.2. Related Work

This thesis is based on an existing research platform SafeVRU [7]. SafeVRU is a complete system that integrates perception, vehicle localization, route planning and local motion controlling. The main goal of the platform is to ensure safety and performance when driving in complex urban environments involving vulnerable road users (VRUs). In particular, the motion control employed in SafeVRU relies on model predictive contouring control (MPCC) [8], [9], which formulates the planning problem as a multi-objective constrained non-convex optimization problem.

This section will discuss the existing MPC tuning methods and decision-making architectures as well as a literature review of the relevant study. The focus is on the work being done in the field of MPC tuning methods, followed by a discussion of different decision-making architectures used in the field of autonomous driving. Afterwards, a discussion of how they differ or are similar is given, along with what approach would be best suited for the design of the controller, eventually leading to a thesis outline well covered in this report.

1.2.1. Tuning of MPC

In the literature, there is an abundance of approaches that can be used to formulate the tuning problem. In particular, [10], [11], [12], published roughly one decade apart, provide an overview of MPC tuning during the period that they were written. The literature covers a variety of tuning approaches for the multitude of MPC formulations and their associated tuning parameters. The MPCC formulation [7] considered in this thesis is therefore not compatible with every approach found in the literature. The following sections categorize different aspects of the many tuning methods used by MPC.

Thematic Approaches

The MPC tuning problem is considered a practical procedure, and for this reason, different tuning methods may be categorized into a variety of thematic strategies based on the literature.

- **Heuristic** Typically, heuristic MPC tuning methods are based on rules-of-thumb tuning guidelines [3]. A list of instructions and simple algebraic expressions are presented in these guidelines to meet predefined criteria, which are often based on frequency domain analysis. [13] also presents some heuristic tuning procedures based on expert knowledge. However, heuristic methods often do not exploit the full potential of the closed loop system.
- **Analytical** There have been several studies of analytical methods based on pole placement [14], [15] and plant identification [16], [17]. A major limitation of these methods is that the analysis is conducted in the case when there are no constraints in place.
- **Algorithmic** A number of algorithmic approaches in tuning controller are applied, which usually involve another closed-loop optimization. The objective of the new optimization problem might be, for example, to determine a function of the closed-loop response with respect to the tuning parameters [18]. Figure 1.2 illustrates the schematic diagram of how the algorithm is executed. In addition, [19], [20] present several ways that deal with multi-objective optimization formulations. A reoccurring tool is the use of evolutionary algorithms, for instance, genetic algorithms [21]. A particular focus is placed on determining the MPC weights that minimize the overall energy usage with a reduced tracking error using genetic algorithm in [22]. In order to determine the specific performance metrics that are indicative of the control scenarios, a single objective function has been formulated using importance weighted performance metrics. Once all metrics have been attended to with the consent of the operator, the algorithm terminates.

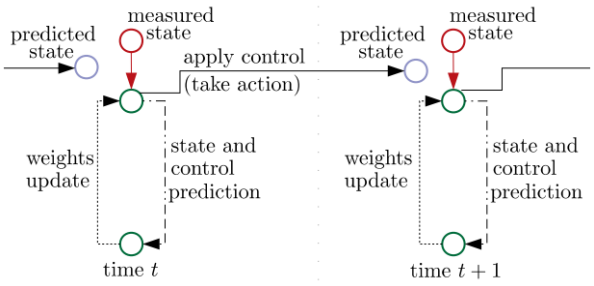


Figure 1.2: Schematic Diagram for the Execution Stages [18]

- **Learning-based** With learning-based approaches, tuning parameters are selected by first performing offline training experiments with different tuning parameters, in order to collect data. Models are then fitted to the data, from which tuning procedures may be derived. In [23], a conditional inverse reinforcement learning method is employed in conjunction with 1000+ hours of expert driving training data. A machine learning method is used in [24] to approximate a human-learned cost function with existing expert knowledge and consequently tuning the controller. It is possible to apply the framework to problems with a relatively large number of tuning parameters.

Tuning Parameters

Depending on the specific formulation of MPC being considered, tuning parameters will take on different roles. Since MPC offers a lot of flexibility in terms of applications, it is important to discuss the tuning

parameters individually. Consider the following example of a quadratic cost function for optimization without constraints:

$$J = \sum_{i=1}^{N} w_{x_i} \left(r_i - x_i\right)^2 + w_{u_i} \Delta u_i^2 \tag{1.1}$$

where N is the prediction horizon, x_i is the state variable, r_i is the reference, u_i is the input variable, and w_{x_i} and w_{u_i} are the weight parameters reflecting the relative importance of x_i and Δu_i, respectively.

- **Structural Parameters** Studies have been conducted to tune the structural parameters of MPC. For example, it is crucial to ensure that the prediction horizon N, that is, the number of future control intervals the MPC controller must evaluate when optimizing, is tuned correctly to ensure the closed-loop stability. To obtain the desired prediction horizon, [11] employs a first-order-plus-dead-time representation of the process and formulas derived from the time constants. In some MPC formulations, the prediction and control horizons are considered separately [3]. [25] also discusses the use of a genetic algorithm as a way of tuning the prediction and control horizons.
- **Weights** Most of the literature tend to focus primarily on tuning the weights within the cost function. The number of the weights can vary depending on the formulation. In general, weights are categorised as weights on the outputs, weights on the inputs and weights on the rate-of-change. Within the formulation above, for example, by increasing the value of w_{u_i}, more control efforts can be directed to the controller output for tighter control but at the expense of being more sluggish [11]. It is typical to leave the tuning of this parameter to the controller designer with experience and knowing the process requirements [11]. In addition, it may be appropriate to consider weights on inputs, as a means of relaxing a constraint on the optimization process and making it more computationally attractive. For example, the quadratic term with w_{x_i} is trying to constrain the deviation from the current state to the desired state.

It should be pointed out that the tuning of weights and structural parameters do not necessarily have to be mutually exclusive. one of the examples shows that weights and structural parameters have been tuned at the same time [26]. On the other hand, the inclusion of structural parameters as free tuning parameters may not always be desirable [27], since these parameters may be fixed as a result of the limitations of the hardware in which the controller is intended to operate.

Tuning Timescale

The implementation of different tuning methods varies depending on the timescale in which they operate and whether they are intended for online or offline tuning.

- **Offline** In general, offline methods require finding the desired weights prior to applying them to the actual plant. This usually requires the designer to perform multiple simulations in order to ensure good performance of the control system, either in the event of unforeseen changes (e.g., variations in operating conditions or unmeasured disturbances), or even in the event of performance criteria change. The authors in [22] employ genetic algorithm to obtain the desired set of weights in simulation and apply it to the actual plant. One drawback of offline tuning is that the simulation performance will not always reflect the performance on the actual plant. The cause of this may be due to modeling uncertainties, noise, disturbances, or error caused by discretisation during the simulation [27]. Thus, it is unclear whether a well-performing controller in offline tuning will also demonstrate good performance when applied to the actual plant. While this is true, in the case of a relatively representative plant, offline simulation is still recommended to evaluate whether the designed controller meets prescribed performance requirements [28].
- **Online** The implementation of online strategies involves correcting the tuning parameters in a real-time manner in order to fulfill specifications designed from a detected nonconformity. In [29], the authors propose a real-time weight tuning strategy. Optimal control inputs are used as the control command so as to meet vehicle safety, ride comfort, and fuel economy requirements simultaneously. In [30], the authors present an automatic tuning strategy for the online selection of weights. The weight factor will be tuned online during each sampling period and applied to the minimization procedure of the cost function at the next sampling period. As described in [31], the authors propose an automatic tuning method capable of receiving plant measurements, reference signals, and performance criteria to optimal the tuning parameters only when they are

required. This method is schematically illustrated in Figure 1.3. Due to the fact that in most cases two optimization procedures are calculated simultaneously per time step, online methods may be computationally more expensive or take longer to tune a controller.

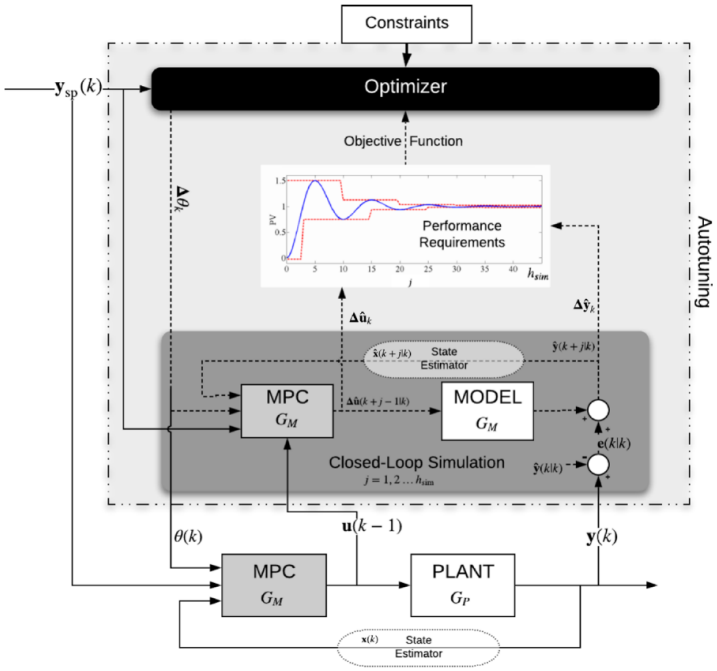


Figure 1.3: Automatic Tuning Layer [31]

Tuning techniques are ambiguous in the sense that they can either be used in an online or offline setting. A proposed algorithmic approach may be applicable if the performance of a given controller can be evaluated by some external indicators or the tuning parameters can be updated in between evaluations [27].

1.2.2. Outline of Decision-making Architecture

Given the countless permutations of scenarios an autonomous vehicle can encounter on the road, making the right decision at the right time is one of the biggest challenges in autonomous driving. Making decisions feel not only safe but also human-like is key to the adoption of autonomous driving.

It is essential to understand the general framework of autonomous driving in order to design methods for efficient decision-making through specific research. Based on a summary of a number of related studies, this subsection provides an overview of decision-making in the field of autonomous driving. These contents summarize four aspects of a decision-making system for autonomous vehicles, namely inputs and outputs, design criteria, design constraints, and applications scenarios.

Inputs and Outputs

In autonomous driving, the decision-making system serves as an interface between perception and planning. A decision-making system typically receives inputs such as environmental information and the status of the ego vehicle then generates outputs such as driving behaviors or direct control commands, which are then sent to the motion planning module [32]. More specifically, the inputs of a decision-making system can be summarized as follows.

- **Environmental information** In general, perception data is collected from various sensors on vehicles, which are then processed to generate perception results that include information concerning obstacles, road condition, and traffic signals.
- **Status of ego vehicles** Localization information is obtained from a localization system, while motion information is derived from a motion estimation system.

Outputs of the decision-making system can be summarized as follows:

- **High-level driving behaviors** Different driving behaviors such as cruising, overtaking, and lane switching.
- **Low-level control commands** Variables selected to control the vehicle, including longitudinal velocity, acceleration and steering rate.

Design Criteria
Managing decisions is the goal of generating a safe and reliable driving strategy that is closer to a human. To achieve better decision making, several design criteria must be formulated, and four aspects are outlined below [33]:

- real-time performance
- balance safety and efficiency when making decisions
- ride comfort
- capacity to detect faults

Design Constraints
In order to develop a more complete system, research on decision-making methods requires the consideration of many factors. The following is a list of related works that provide several design constraints for decision-making systems.

- **Information of surrounding environment** There should be consideration of objects located within a certain distance around the ego vehicle. Among these are, for example, the status of other vehicles, static obstacles, the prediction of pedestrian behaviors, as well as traffic signals.
- **Local traffic regulations** Ego vehicles are expected to comply with traffic rules when making decisions, including speed limits, turning restrictions, etc.
- **Current status of ego vehicle** States of ego vehicle, including location, velocity and heading.
- **Results of path planning** Normally, path planning deals with two types of trajectory: global trajectory and local trajectory. Decision-making takes into account primarily the outcomes of current local trajectory.
- **Historical decision-making results** The sequence of historical decisions made by the ego vehicles at the previous moment or in the past few moments that need to be considered when making decisions at the current moment.
- **Driving ethics** During operation, vehicles must adhere to driving ethics [34], by giving pedestrians courtesy, giving way to special vehicles, etc.

Application Scenarios
It is necessary to make decisions almost in every scenario in which an autonomous vehicle is operating. Research has focused on some typical scenarios including highways, urban intersections, and merging traffic because of the increasing requirements for decision-making systems in a complex driving environment [5].

1.2.3. Methods of Decision-making

In general, decision-making are divided into classical methods and learning-based methods [5]. Specially, classical methods can be categorized into rule-based methods, optimization methods and probabilistic methods. Because of the fact that autonomous vehicles are operating in complex and dynamic environment with cooperation with other road users. Classical methods are not always effective in such condition due to poor adaptability to uncertainties, while learning-based methods are employed to achieve better adaptability but usually require plenty of training training data.

Rule-based Methods
In the rule-based decision-making system, strategies are determined by considering the status of various vehicles according to an extensive rule database that incorporates numerous traffic laws and driving experience.

As the most representative rule-based method, given in Figure 1.4, finite state machine (FSM) is understood as abstract machines that reside in one of a set of state transitions between which a given state can be changed. However, FSM can be unmanageable for large complex systems, which is referred to as the state and transition explosion [35]. As a result, maintaining and improving such systems becomes labor-intensive and problematic, especially when they grow.

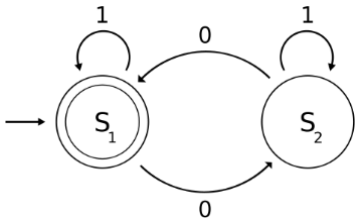


Figure 1.4: Finite State Machine

In an effort to reduce the heavy transition duplication required in FSMs, hierarchical finite state machines (HFSMs) were developed by D. Harel in 1987 in order to improve understanding of complex systems by introducing structure [36]. Figure 1.5 shows how an HFSM clusters states into a group, referred to as a superstate, where all the internal states are implicitly associated with the same superstate. One advantage of HFSMs is that it does not require each state to replicate transitions to a particular state, but enables some or all of the transitions to be inherited from a superstate. Although this is a more modular approach than FSM, it still inherits many of the disadvantages, such as limited reusability.

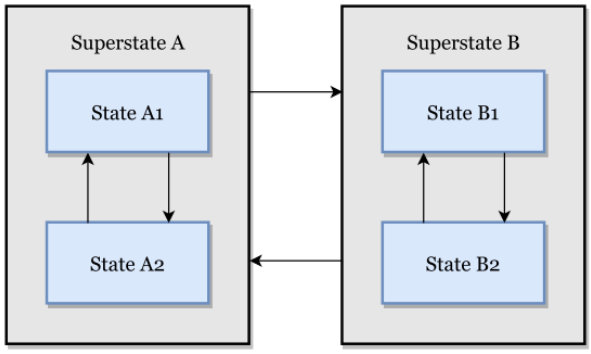


Figure 1.5: Hierarchical FSM

The most representative application of FSM and HFSM implemented in autonomous vehicles is the DARPA Grand Challenges, where Team TerraMax used FSM to decide which driving mode the high-level controller should select [37]. Another example can be found in the DARPA Robotics Challenge [38]. These hybrid architecture for complex autonomous systems use HFSM to define the overall system behavior and coordinate between sub-systems.

As an outperformed alternative approach, Behavior Tree (BT) is structured as a directed rooted tree and switches between different tasks in an autonomous agent. BT are modular and reactive to create complex systems efficiently. These properties have led to the spread of BT from computer game programming to many branches of robotics.

A BT consists of a variety of nodes. All nodes have the same external interface, but each has its own internal functionality. As a result, the structure appears modular. The return status of each node (success, failure and running) in a BT determines how the tree will be traversed upon evaluation. The three types of nodes in a BT are conditions, actions, and composites [39]. Condition node evaluates a particular property of an environment, while an action prompts the agent to act on the environment. Table 1.1 is a representation of all the nodes in a basic BT. Normally, leaf nodes are developed for specific application, while composite nodes are application independent and can be reused. Figure 1.6 illustrates the process of tree execution, showing the basic node types and execution flow. Ticks refer to

the execution of a BT. Not all nodes are evaluated in every tick, imposing a sort of priority on execution by ticking the leftmost nodes first.

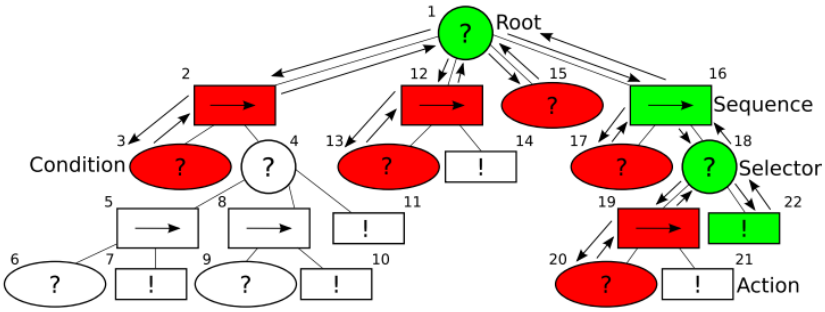


Figure 1.6: Typical Representation of the Behavior Tree [39]

Node type	Graph	Success	Failure	Running
Sequence		If all child succeed	If one child fails	If one child is running
Selector		If one child succeeds	If all child fail	If one child is running
Decorator	Repeat ∞	Varies	Varies	Varies
Parallel		If N child succeed	If M-N child succeed	If all child are running
Action	Action to perform	Upon completion	Impossible to complete	During completion
Condition	Predicate	If true	If false	Never

Table 1.1: Basic Node Types in Behavior Tree

A study of the how suitable it is to use BT as a decision-making architecture for autonomous driving is presented in [35]. Accordingly, the use of BT over FSM has the advantage that the transitions between states do not require labor-intensive manual transitions. Moreover, actions, conditions, and sub-trees can be added or removed without requiring modifications to other components in the BT. The authors of [40] do not claim that BT is superior to FSM from a purely theoretical perspective. In fact, all BT can most likely be formulated in terms of FSM. However, there are significant differences when it comes to modularity, readability, and reusability, which makes BT a reliable decision-making architecture. Thus, BT represents the most promising decision-making system that can be utilized in an autonomous vehicle system, out of all the explored options in rule-based methods.

Optimization Methods

Optimization methods typically employ a reward function to generate decision results. Methods such as Chance Controlled Optimization are employed [41] for lane change overtaking in urban areas. A Markov linear system is used in this case to model other road users' behavior, but it is modelled in a discrete time stochastic hybrid system. In [42], receding horizon control is integrated into game theory in order to provide the decision with information that is based on current and future predicted uncertain information. In order to evaluate actions during lane changes, a game is defined that includes a reachability analysis to determine an upper and lower bound for the position of the vehicle at each time step.

Probabilistic Methods

Based on probability theory in mathematics, probabilistic methods can generate behavior results. The Partially Observable Markov Decision Process (POMDP) is a representative example. By evaluating all possible action sequences and taking their effects into consideration, it is possible to maximize the expected total reward over a period of time. Probability distributions are applied to the states in

order to allow for uncertainty to be accounted for during the decision-making process. It is however computationally costly to solve, which makes it critical to use in practical applications. To avoid the complexity involved in computing a complex, long-term policy, approximate POMDP solutions [2] to simplified problem formulations are used to overcome the intractable problem of solving the most general POMDP.

Learning-based Methods
Generally, learning-based methods use artificial intelligence technologies to enable autonomous vehicle decision-making. In most cases, driving data require to be collected firstly, and different learning methods or networks frameworks are then adapted to achieve learning of vehicles to produce reasonable decisions based on various environmental circumstances. Learning-based strategies have the disadvantage of requiring a large amount of training data in order to train the learning model for all potential scenarios. The system resulting from this process is treated as a black box [43]. It appears to be a major issue as it impedes failure testing, and there can be no assurances regarding performance under untrained scenarios. The system must be retrained if undesired behavior occurs. Any subsequent data specific to that scenario must also be collected.

1.2.4. Conclusion from Related Work
The related work discussed above highlights the strengths and weaknesses of the methods deployed for MPC tuning and decision-making in autonomous driving. However, it also becomes evident that research on the integration on MPC tuning methods and decision-making is very disjointed. The reason behind is mainly that the outputs of decision-making system is typically the high-level behaviors instead of the direct control commands or parameters adjustment in the controller. Hence, one challenge in this thesis is to select the suitable methods for both systems and incorporate them. Base on this, along with the capabilities of the MPCC motion planner and the test scenarios, the most suitable method for this work can be chosen. This main criteria for choosing the approach in this thesis are:

- **Generality**: an approach that could be applied to the targeted type of driving scenario, namely, urban traffic where vulnerable road users are present.
- **Traceability**: an approach that allows for easy failure analysis and debugging.
- **Real Time Capability**: the method is cost-efficient to be used in later real world application.

Taking a closer look at automatic tuning methods based on this, algorithmic methods appear to be the most appropriate selection since they do not require extensive training data or expert knowledge to begin with the tuning procedure. Genetic algorithms appear to be the most suitable algorithmic method since they make use of importance-weighted performance metrics that can account for the driving performance. Additionally, it is possible to modify the structural parameters in order to change the complexity of the genetic algorithm. According to the majority of the reviewed literature, genetic algorithms are typically used offline before applying converged parameters to a plant. This thesis will propose an online implementation to explore both options and compare both with the default MPCC setting. In terms of the decision-making architecture, the behavior tree appears to satisfy most of the requirements in terms of its modularity and maintainability.

1.3. Proposed Approach
On the basis of the variable tuning challenges, capabilities and limitations of the decision-making architecture identified in the literature on the field of autonomous driving, the importance and requirements of the integrated methods needed to overcome these challenges has been highlighted. Therefore, this work proposes the use of a genetic algorithm that works along with behavior tree and tunes the controller parameters in an urban traffic scenario while avoiding collision with other vulnerable road users.

The fundamental component of the automatic tuning method that is responsible to achieve the aforementioned objectives is MPCC-based motion planner. A general description of the system architecture given in the following clarifies how the overall system looks like.

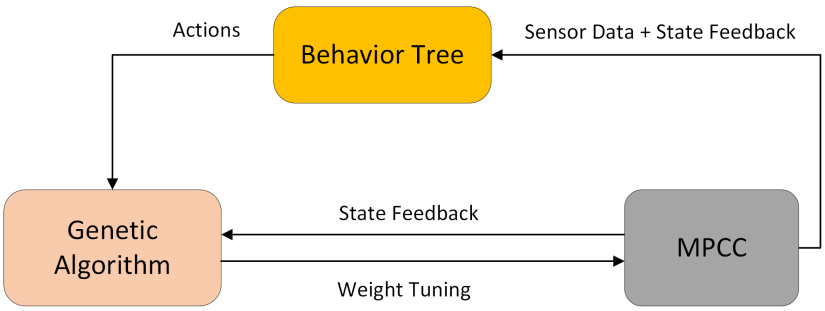


Figure 1.7: System Architecture Showing Major Components

Figure 1.7 illustrates the closed loop architecture of the automatic tuning method equipped with MPCC-based motion planner. The system is divided into *(i)* Automatic tuning approach where the genetic algorithm is implemented and generates tuned weight vectors, *(ii)* MPCC-based motion planner, which is equipped with MPCC controller and a solver that finally control the acceleration and steering motion of the vehicle to ensure trajectory following and collision avoidance, *(iii)* Behavior Tree, which generates corresponding actions based on the scenarios information.

1.4. Thesis Outline

The rest of the thesis presents the design of the automatic tuning method of the MPCC-based motion planner. Chapter 2 discusses background information on the MPCC-based motion planner, vehicle modelling and weight parameters. It gives an understanding of the system to be controlled, and presents the selected model that can be used as prediction model of MPC. Chapter 3 presents the design of automatic tuning method and implementation to achieve the objectives described above. Chapter 4 presents the design of the decision-making architecture, where behavior tree is employed to better understand vehicle behaviour due to the combined traffic scenarios. Chapter 5 discusses the simulator used in this work, construction of all scenarios the other components required for setting up the simulation setup. It also presents the performance comparison of different breakdown scenarios and also verifies the automatic tuning method performance compared with the default MPCC approach. Chapter 6 finally concludes the thesis and presents scope of future work.

2

Preliminaries

To design an MPCC-based automatic tuning method, several important components, such as the model of the system and the role played by the parameters, have to be determined first. This chapter begins with an introduction to the MPCC motion planner. The system model is then considered, which is a simple kinematic bicycle model. Finally, the impact of parameters on the objective function is discussed for further optimization.

2.1. MPCC-based Motion Planner

This thesis is based on the research platform SafeVRU [7]. Under this platform, the vehicle is able to perform localization, perception, motion planning, and control of its motion. In particular, the module for perception and planning is of particular importance. Figure 2.1 outlines the overall structure of SafeVRU.

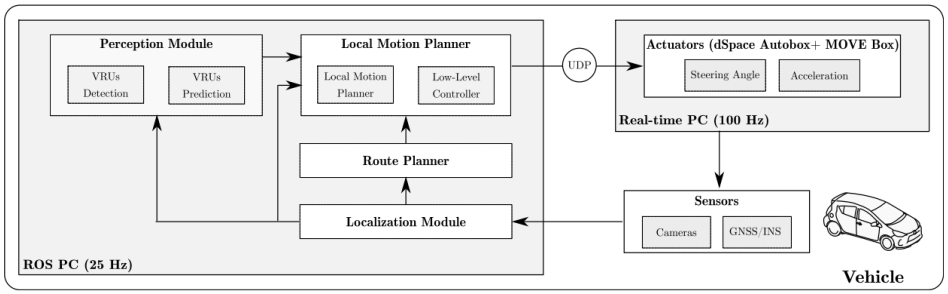


Figure 2.1: Overview of the architecture [7]

A particular objective of the motion planner is to provide an efficient and practical implementation of MPCC [9], which is suitable for real-time collision-free navigation of autonomous vehicles in complex environments with multiple agents. By integrating information gathered from the route planner, localization module, and perception module, the motion planner is able to compute a collision-free trajectory and control the vehicle directly, sending acceleration and steering commands. The MPCC formulation is designed to plan safe trajectories that take into account the predicted paths of the VRUs and to compute control commands for the vehicle. [7] provides further details on the formulation.

2.2. Vehicle Modelling

Simulations are an essential part of the development of the motion planner. Choosing a vehicle model that captures the kinematics and dynamics of the vehicle has a substantial impact on the reliability of simulation results. The vehicle model should be determined by practical application and the need to closely approximate the actual behavior of the vehicle. For example, the dynamics of a vehicle, including tire compliance, must be taken into account when driving at high velocity or performing aggressive

maneuvers. On the other hand, in urban traffic scenarios where vehicles have relatively low speeds, a kinematic model of the vehicle may be used for control purposes.

2.2.1. Kinematic Model

Kinematic bicycle models represent the general physical behavior of the vehicle through the use of nonlinear equations of motion and are based solely on the vehicle geometry. As a result, a four wheeled vehicle can be simplified into a two wheeled bicycle with constraints. As this model does not consider tire slip in lateral or longitudinal direction, it is a useful approximation of vehicle behavior in slow speed driving scenarios. The following nonlinear continuous time equations describes the kinematic bicycle model in the inertial frame, as also shown in Figure 2.2.

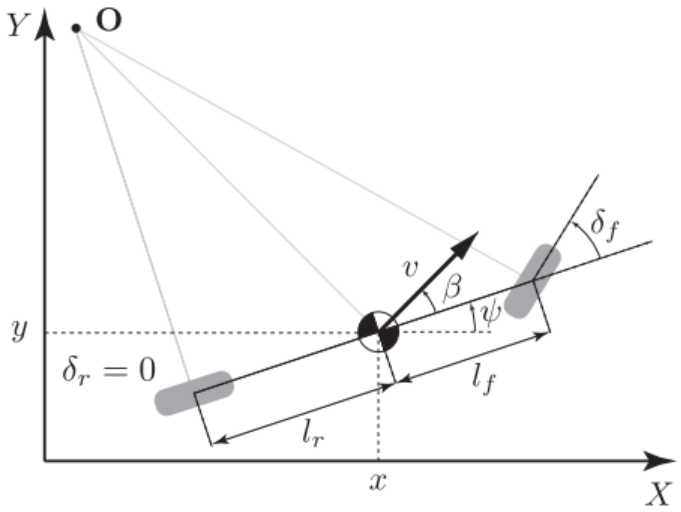


Figure 2.2: Kinematic bicycle model [44]

$$\dot{x} = v\cos(\psi + \beta) \tag{2.1a}$$
$$\dot{y} = v\sin(\psi + \beta) \tag{2.1b}$$
$$\dot{\psi} = \frac{v}{l_r}\sin(\beta) \tag{2.1c}$$
$$\dot{v} = a \tag{2.1d}$$
$$\beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r}\tan\left(\delta_f\right)\right) \tag{2.1e}$$

where x and y are the coordinates of the center of mass of the vehicle. ψ is the inertial heading angle and v is the vehicle's velocity. The distances between the center of mass of the vehicle and the front and rear axles are represented by l_f and l_r respectively. β is the angle of the current velocity with respect to the longitudinal axis of the car. a is the acceleration in the same direction as the velocity. As most vehicles do not have steering wheels at the rear, $\delta_r = 0$ is assumed in this calculation. The control inputs therefore include the front steering angles δ_f, and a.

The motion planner incorporates the kinematic bicycle model. The optimization problem in [7] solves the state vector $\mathbf{z} = [x, y, \psi, v, \delta]^{\mathrm{T}}$. At time t, the planner obtains information regarding the car and its approximated path parameter. Following this, the planner solves the optimization problem above to determine a sequence of commands. Finally, the planner sends the first control command $\mathbf{u} = [a, \dot{\delta}]^{\mathrm{T}}$ (acceleration and steering rate) to the plant while discarding the other elements of the sequence.

2.3. Role of the Weights

Each control objective is assigned a weight, which indicates its importance in the cost function. Two weighting vectors are used in the planner to determine the weights of the cost function: the output weight and the rate weight. Higher output weights will cause the controller to manipulate the tracking process in order to provide more accurate results. The rate weight affects the amount of incremental change in the control input. By increasing the rate weight, smoothness can be improved at the cost of increased tracking errors. Thus, optimal weight parameters must be determined for the purpose of minimizing the tracking error while reducing the consumption of control inputs. The cost function J defines the objectives of the planner:

$$J = J_{tracking} + J_v + J_a + J_{\dot{\delta}} \tag{2.2}$$

It is defined as follows: $J_{tracking}$ is the penalty for path tracking, J_v is the penalty for the error between the measured velocity and the reference, and J_a and $J_{\dot{\delta}}$ are the penalties for the inputs.

Tracking

Contouring control [45] consists of steering (x_k, y_k) along a global reference path consisting of sequences of path segments connecting waypoints $[x_m^p, y_m^p]$. The contouring error e_k^c is defined as the normal deviation from the desired path, which is expressed as:

$$e_k^c = \sin\phi(\theta)(x_k - x^r(\theta)) - \cos\phi(\theta)(y_k - y^r(\theta)) \tag{2.3}$$

$$\phi(\theta) = \arctan\left(\partial y^r(\theta)/\partial x^r(\theta)\right) \tag{2.4}$$

where $\theta(x, y)$ is the value of the path parameter where the distance between the point $(x^r(\theta), y^r(\theta))$ and (x_k, y_k) is minimal, as illustrated in Figure 2.3. θ_k is an approximation to $\theta(x_k, y_k)$. An estimation of the contouring error is then expressed as:

$$\tilde{e}^c(\mathbf{x}_k, \theta_k) = \sin\phi(\theta_k)(x_k - x^r(\theta_k)) - \cos\phi(\theta_k)(y_k - y^r(\theta_k)) \tag{2.5}$$

Let e^l denote the path distance that $(x^r(\theta_r), y^r(\theta_r))$ lags $(x^r(\theta_k), y^r(\theta_k))$ and approximate e^l as:

$$\tilde{e}^l(\mathbf{x}_k, \theta_k) = -\cos\phi(\theta_k)(x_k - x^r(\theta_k)) - \sin\phi(\theta_k)(y_k - y^r(\theta_k)) \tag{2.6}$$

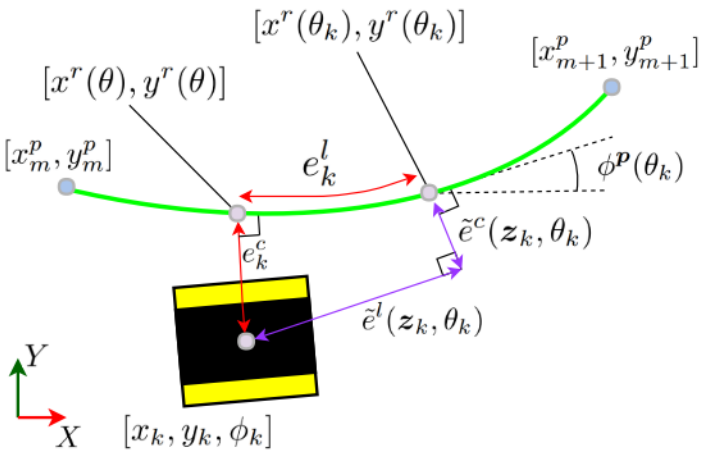


Figure 2.3: Contouring error, lag and their approximations [46]

Figure 2.3 illustrates the graphical representation of e_c, e_l and their approximations.The major objective of contouring control is to minimize the contouring error while maximising the path distance travelled at each time step. The error vector can then be expressed as:

$$\boldsymbol{e}_k = \left[\tilde{e}^c(\mathbf{x}_k, \theta_k), \tilde{e}^l(\mathbf{x}_k, \theta_k)\right]^{\mathrm{T}} \tag{2.7}$$

Consequently, the MPCC tracking cost is:

$$J_{tracking}\left(\mathbf{x}_k, \theta_k\right) = \boldsymbol{e}_k^T Q_e \boldsymbol{e}_k \tag{2.8}$$

There are two parameters in the penalty weight Q_e that are to be determined according to the relative importance of contouring accuracy, path speed and control deviations.

Velocity

In order to minimize the quadratic tracking cost as described above, the vehicle is directed towards the reference path. For the purpose of progress on the path, a cost term that tries to penalize the deviation of the vehicle velocity v_k from the reference velocity v_{ref} is introduced:

$$J_v\left(\mathbf{x}_k, \mathbf{u}_k\right) = Q_v\left(v_{ref} - v_k\right)^2 \tag{2.9}$$

where Q_v is the velocity penalty weight. There is a reference velocity variable v_{ref} provided by a higher-level planner and this variable may vary from segment to segment.

Input Penalty

Additionally, a quadratic penalty is imposed by J_a and $J_{\dot{\delta}}$ on acceleration and steering commands, respectively:

$$J_a\left(\mathbf{x}_k, \theta_k\right) = a_k^T Q_a a_k \tag{2.10}$$

$$J_{\dot{\delta}}\left(\mathbf{x}_k, \theta_k\right) = \dot{\delta}_k^T Q_{\dot{\delta}} \dot{\delta}_k \tag{2.11}$$

where Q_a and Q_δ are acceleration and steering rate penalty weights, respectively.

3

GA-based Automatic Tuning Strategy

The purpose of this chapter is to describe a method to automatically tune the MPCC weight parameters using genetic algorithms (GA). As compared to iterative weight tuning, GA-based weight tuning is reported to provide a faster convergence rate [22]. An implementation of both an online and an offline timescale is designed in the proposed work. Using the online approach, real-time tuning procedures can be provided, while the offline approach with defined termination criteria is used to capture the defined requirements, making the proposed design a reliable tuning procedure.

3.1. Introduction

GA is commonly used to find approximate solutions to optimization problems based on the application of evolutionary biology principles to computer science. Techniques that derived from biology, such as inheritance, mutation, natural selection, and recombination are employed in this process. It is usually implemented as a process in which a population of abstract representations, referred to as chromosomes, of candidate solutions to an optimization problem, referred to as individuals, evolves to a better solution over the search space. GA has the advantage of being a heuristic and iterative tool for determining MPC parameters since it offers both. GA involves the five steps, as shown in Figure 3.1, namely:

1. **Start** Initialize and generate random population of potential solutions (chromosomes)
2. **Loop** The following steps would be loop over until a convergence condition is satisfied
   (a) **Fitness** Compute and evaluate the value of fitness function of each chromosome
   (b) **Selection** Selection of optimal population according to the fitness values
   (c) **Crossover** Cross over the parents to form new offspring with a given probability .
   (d) **Mutation** Mutate the new offspring at each position with a given probability.
   (e) **Replacement** Replace the poorly-performed population with the new generated population for a new run
   (f) **Test** Check if the stop criterion is met
3. **Return** Return the best solution in the last generation

In this work, the implementation process is categorized into two methods, online and offline. Through the online method, the GA is optimized simultaneously with the MPCC, and the chosen parameters are adjusted in real-time. Offline method consists of two stages. Stage one consists of running the algorithm in several breakdown scenarios, determining the optimal set of weights for each scenario, and then applying them to controller during simulation. The operation of the GA is explained in more detail in the following section.

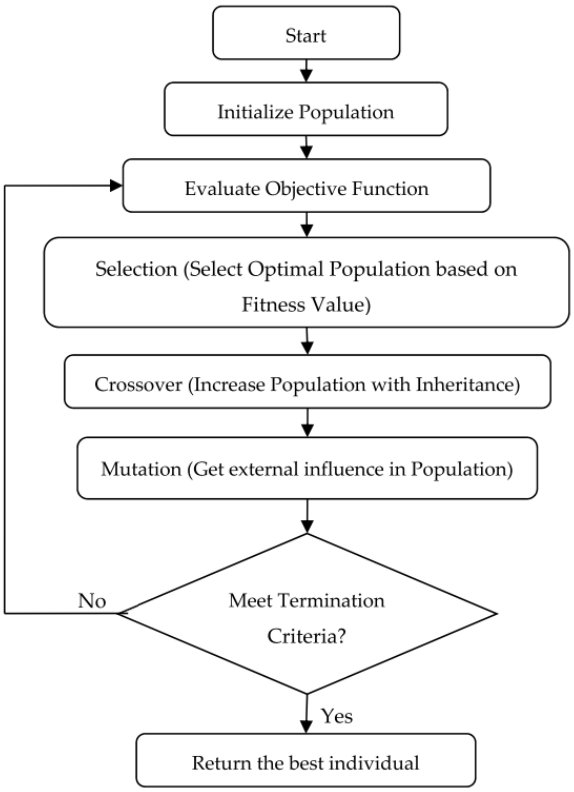


Figure 3.1: Operations in genetic algorithm

3.2. Operation

In this work, each combination of the parameters in the chromosome is represented as a different weight variable in the optimization problem search space. In GA terminology, the optimization function is termed the fitness function. The value representing the appropriateness of a chromosome is known as the fitness value, which is calculated by using the fitness function.

3.2.1. Encoding

The term "encoding" is used to describe the representation of potential solutions containing a set of weight parameters as strings of codes. There are various ways in which weight parameters can be encoded, such as binary coding, letter coding and real numbers [47]. However, binary coding is the most common encoding. In this work, the binary coding method has been selected for simplicity and convenience. Every element of the weight variable is encoded as a string of length, which consists of 0s and 1s to achieve the desired resolution. Uniform distribution is used to select the weight variable at random within a given boundary. The mapping from a the real number x_r into the binary string x_b is straightforward and is completed in two steps:

1. Compute the scale of the bit representation such that the converted value is within the boundary of the chromosome
2. Convert the real number to the scaled binary equivalent using the representation of bitset, which is a fixed-size sequence of bits and stores values either 0 or 1, the equation given by:

$$x_b = \frac{x_r - lb}{\frac{ub-lb}{2^n-1}} \tag{3.1}$$

where ub and lb are the upper bound and lower bound of the chromosome, respectively. In order to demonstrate this more clearly, Table 3.1 illustrates the conversion of weights into binary strings for two initial populations.

Population	Chromosome	Real Number	Binary String
1	Q_e	0.5	10110010100111001101
	Q_v	0.05	01010110110101000100
	Q_a	0.3	00101001111000110101
	$Q_{\dot{\delta}}$	0.02	11010101010100110100
2	Q_e	0.4	10110000100101101001
	Q_v	0.07	01001100110101010100
	Q_a	0.25	11101001010000100110
	$Q_{\dot{\delta}}$	0.03	11001010101001000101

Table 3.1: Example of Two Encoded Populations

3.2.2. Evaluation

It is important to evaluate the quality of a solution once it has been presented. This will require the formulation of the fitness function. It is likely that an individual with a lower fitness value will contribute one or more offspring in the next generation, which is discussed in the following subsection. The performance criteria are based on fitness function, and optimal weight parameters are determined by minimizing an objective, which incorporates a weighted combination of IAE. Further details regarding the formulation of the fitness function will be discussed in Section 3.3.

3.2.3. Selection

In selection operation, individuals are selected for further recombination. Proportional selection is the most widely used algorithms [48]. This method involves a selection process in which an individual has a probability of being selected depending on the relative fitness value. Consequently, the individuals in the generation have a biased chance of occurring, favoring the more suitable individuals. The selection probability can be calculated for any population i:

$$p_i = \frac{\frac{1}{J_i}}{\sum_{k=1}^{m} J_k} \tag{3.2}$$

where J_i is the fitness value for population i, and m is the population size. This selection algorithm could be seen as a pie chart, Figure 3.2 shows an example of four populations and their probability to be chosen.

Population	Fitness Value
1	1.25
2	1.5
3	2.5
4	5

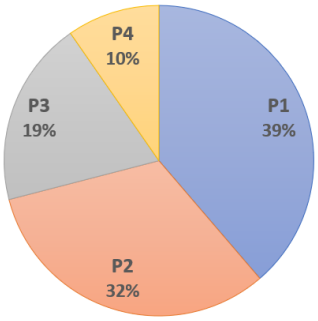


Figure 3.2: Proportional Selection

3.2.4. Crossover

Crossover is one of the operators that attempt to reproduce new individuals. To create a new offspring, the strings of the chromosomes are cut and mixed. There are three different crossover methods: one-point crossover, two-point crossover and uniform crossover [49]. The first two methods result in less diverse offspring. These involve dividing the chromosome into segments and swapping them. Uniform crossover evaluates every bit in the chromosome with the intent of exchanging it with a probability. A probability of 0.5 would suggest that around half of the chromosomes for the new offspring belong to

parent one and the other half to parent two, as shown in Figure 3.3. In this work, uniform crossover has been chosen because of its diversity between the parents and offspring.

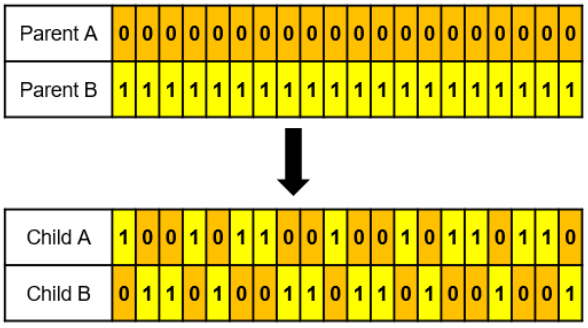


Figure 3.3: Example of Uniform Crossover (probability ≈ 0.5)

The crossover of all chromosomes in a population is not necessary or even recommended because this can neither tell us if the result will be better or worse than the parent chromosomes [50], consequently, the crossover is limited to the less desirable half of the chromosome.

3.2.5. Mutation

There is a tendency for the chromosome pool to become homogeneous after a certain number of reproductions, as one better chromosome begins to dominate after several generations, resulting in premature convergence of local minima. In order to overcome this undesirable convergence, mutation is introduced with appropriate probability. Ideally, this probability should be small enough to cause a small alteration in the chromosome. Otherwise, GA would resemble a random search. The binary coding employed in this work renders bit flip mutation [51] particularly suitable for use. In the bit flip method, bits are alternated from 0 to 1 or from 1 to 0 with the mutation points being determined at random. An example is shown in Figure 3.4.

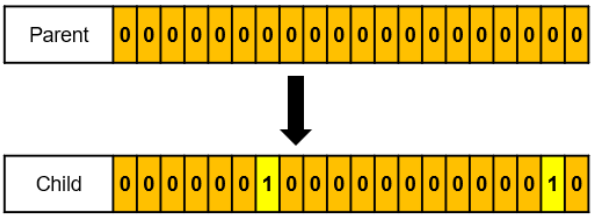


Figure 3.4: Example of Bit Flip Mutation (probability ≈ 0.1)

3.3. Formulation

In this work, the weight parameters of the MPC cost function are assigned to each individual of the population. Their characteristics are characterized by the set of weights constrained by a range of upper and lower bounds:

$$w_{GA} = \begin{bmatrix} Q_e & Q_v & Q_a & Q_{\dot{\delta}} & v_{ref} \end{bmatrix} \tag{3.3}$$

To determine the MPC weights w_{GA} which minimize the overall error, the fitness function for the GA is formulated. Since it is a multi-variable process, the objective function also incorporates four terms to describe the performance of each parameter. These two terms refer to tracking and velocity-following errors, respectively, defined in terms of integral absolute error (IAE):

$$IAE_{tracking} = \sum_{n=0}^{N} |\mathbf{p}^{\mathbf{r}}(n) - \mathbf{p}(n)| \tag{3.4}$$

$$IAE_{velocity} = \sum_{n=0}^{N} \left| v^{ref}(n) - v(n) \right| \tag{3.5}$$

where $\mathbf{p^r}$ and $\mathbf{p}$ are reference pose and current pose of the vehicle, respectively. The cumulative rate change of input variables (acceleration a and steering rate $\dot{\delta}$) are also considered. An input error is calculated on the basis of the accumulation of deviations of the manipulation variable from its value at the previous time step.

$$IAE_a = \sum_{n=0}^{N} |a(n) - a(n-1)| \tag{3.6}$$

$$IAE_{\dot{\delta}} = \sum_{n=0}^{N} \left| \dot{\delta}(n) - \dot{\delta}(n-1) \right| \tag{3.7}$$

In the proposed work, the weighted average method is used to formulate a single objective function. The importance weights are constrained to unity since the error terms are normalized.

$$J\left(w_{GA}\right) = w_1 \times IAE_{tracking} + w_2 \times IAE_{velocity} + w_3 \times IAE_a + w_4 \times IAE_{\dot{\delta}} \tag{3.8}$$

$$\sum_{i=1}^{4} w_i = 1 \tag{3.9}$$

To have the genetic algorithm running in real time, the importance weights are adjusted through the behavior tree architecture, which will be discussed in the next chapter.

3.4. Structure

While the online and offline method share the same flow of the GA operation, the structure and corresponding adjustment are different. The structure of the online method is shown in Figure 3.5, which illustrates how a simplified version of the solver with a lower prediction horizon has been developed under the GA interface to improve computational efficiency. Information other than weights is sent to the GA interface by the MPCC interface. In order to achieve optimal results, both solvers must be consistent in order to evaluate the optimal individual via the GA interface. Upon completion of the GA optimization, the updated weights are sent back to the MPCC interface and used for the MPCC optimization.

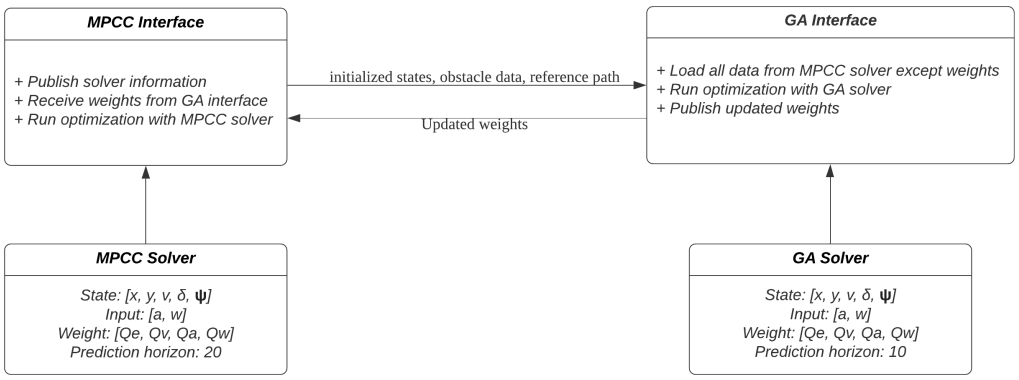


Figure 3.5: Structure of the Online Implementation

As a way of increasing the computational efficiency, the simplified solver uses the weight parameters from each of the individuals to run only one iteration. As the states and input are updated, the fitness values are then calculated. As previously mentioned, individuals with a lower fitness value are more likely to be selected for the new generation. As soon as the stopping criteria is met, the weight parameters from the best individual are sent to the MPCC interface and used to run the MPCC optimization.

3.5. Stopping Criteria

For evolutionary algorithms like GA, the following three kinds of stopping criteria have been traditionally employed [52]:

- An upper limit on the number of generations is reached
- An upper limit on the number of evaluations of the fitness function is reached
- the best solution during the evolution process doesn't change to a better value for a predefined value of generations

For the first two criteria, selecting sensible settings requires expert experience in order to estimate an appropriate maximum generation number, at which point it stops and provides the best results. In contrast, the third alternative does not require such knowledge. This work employs an interactive decision tree (IDT) approach [53] to decide the importance weights that are assigned to each of the performance metrics used in the fitness function. As a result, the improvement in performance metrics is determined based on the operator's response [22]. A uniform distribution of weights for each individual metric w_i is initially used in the fitness function.

In order to determine if the performance improvements in these metrics are significant, they are compared to the default MPC weight parameters ($m = 0$):

$$\%\,\mathrm{ImpIAE} = \frac{IAE\,|m - IAE|\,m - 1}{IAE \mid m} \times 100\% \tag{3.10}$$

A weakly improved metric j is determined by the algorithm in conjunction with the operator. Upon reaching the level of satisfaction for all metrics, the operator may select none ($j = 0$) then the algorithm is terminated. The weight associated with the identified weakly improved metric ($w_{i=j}$) is increased by a small factor (n_i):

$$w_{i=j} = n_i w_i \tag{3.11}$$

Then weights of the other performance metrics $w_{i\neq j}$ are decreased equally without loss of generality:

$$w_{i\neq j} = w_i \frac{(N - n_i)}{(N - 1)} \mid i \in \{1, 2, \ldots, N\} \tag{3.12}$$

This new set of fitness weights are applied to the GA procedures. Whenever the metrics need to be improved, the weights are increased by a factor n_i. Upon achieving an acceptable improvement, the weight for the specific metrics is set to be fixed, and the other weights are distributed in the same manner to the next weakly improved metric.

3.6. Conclusion

This chapter presented the design procedure of the GA. It explained the reasoning behind the choice of each method utilized in the operation. As part of the formulation of the fitness function used for evaluating individual suitability, the IAE of tracking, velocity, and inputs are penalized. The fitness function was formulated as a single objective function. Furthermore, the GA optimization process was terminated when defined conditions are met using a stopping criteria approach. In terms of implementation, the GA development was applied both online and offline with different structural parameters. Another difference was that in online tuning, the importance weight in the fitness function is automatically adjusted. During offline tuning, the weight parameters of the controller were applied by means of a lookup table to the controller.

4

Behavior Tree

This chapter discusses the implementation of the decision-making architecture behavior tree (BT). First some properties of the developed BT are discussed regarding priority and concurrency. Then a description of how the BT can be crafted for different implementations of the genetic algorithm is provided. Finally a conclusion is given.

4.1. Properties

It is important to note that a basic implementation of BTs involves blocking behavior, that is, once an action is launched it will continue until it is completed. In such case, the BT would not able to respond in this period. There is a possibility that the self-preservation behavior may not be utilized timely when faced with a potentially hazardous situation. It is imperative that this limitation be addressed in order for BTs to be suitable for autonomous driving. A distinction between the BT and FSM is that transitions are implicitly incorporated in the tree structure, reducing the designer's burden of explicitly implementing all transitions. The addition and deletion of a node do not affect other nodes, making development easier and less prone to errors.

4.1.1. Priority

From the root node, the tree is traversed in depth order, which essentially means that the leftmost node will be visited first, indicating that the behavior with the highest priority should be placed at leftmost. As a result, it seems appropriate to place the self-preservation behaviors such as collision avoidance as the leftmost action node of the tree. This also indicates that it is recommended to put the default behavior, such as cruising, on the far right side of the tree.

4.1.2. Concurrency

It is necessary for tasks to be run independently in order to overcome the blocking nature of the tree. In order to accomplish this, either asynchronous tasks can be run on the same thread as the main thread, or it can be implemented by executing a thread concurrently with the main thread while the tree is ticking over [35]. In some cases, the BT may be required to continue where it left off in a series of tasks rather than starting from the beginning. Concurrency in behavior trees is not standardized or established, so it is left to the developer to decide how to implement the concept. The three approaches are based on event-driven design, an open/closed node pattern, or recursive abort calls [35].

The event-driven approach enables the BT to be responsible for notifying nodes of activity within the tree, making them to be reset. In the simulation, this approach is applied. By using the open/closed pattern, one can keep track of which nodes have not yet begun executing and which have already begun. With this information, it is possible to reinitiate the nodes on the subsequent tick. The benefit of recursive abort calls is that no blackboard knowledge has to be communicated between the nodes. As soon as a task successfully executes and returns Running or Success, a recursive abort call is issued to all subsequent nodes.

4.2. Implementation of BT

Similar to the GA-based tuning method, the implementation of the behavior tree is also categorised into online and offline approaches, with the goal that both approaches exhibit identical driving behavior. The offline approach shares the same philosophy as the online approach, but the actions are carried out differently. When the scenario corresponds to the GA implementation, the actions will take the optimal weight parameters acquired through GA and pass them to the MPCC interface. Additionally, the behavior tree should only utilize the environmental information that could be obtained by the ego vehicle with sensors from the simulation.

4.2.1. AsyncAction

Due to the nature of the GA optimization being an long running process, simply running it and waiting for it to complete may prevent the system from being able to react to new risks as they arise. As a result, a basic BT would not be able to accomplish this task while maintaining full responsiveness. Rather, the BT will be implemented using an event-driven architecture as discussed previously.

A new type of action leaf node, AsyncAction, is introduced for this purpose [54]. As compared to the Action node, which serves only to modify parameters and settings that can be accomplished immediately, AsyncAction is for behavioral tasks, such as running the GA optimization with adjusted importance weights, that cannot be completed in a short period of time and therefore require an asynchronous thread to be spawned. As soon as the node starts the routine, it returns the status code for the BT running. In this manner, the parent composite node saves the index of the previous running child. In the next tick, the system will skip the preceding steps and directly update the running node. Following the completion of the execution of the coroutine, the AsyncAction node will be able to signal success or failure on the following tick.

4.2.2. Environmental Information

To develop a practical BT applied in an autonomous vehicle, it is important to first define the inputs. In this work, the inputs of the BT are environmental information and status of the ego vehicle. As already mentioned, the simulation focuses on the urban scenario where VRUs are present. In particular, pedestrian crossing is one of most typical situations that the vehicle may encounter. There are generally two types of pedestrian crossings: controlled and uncontrolled [55]. As a result of the former, traffic signals are employed to direct the interaction between pedestrians and vehicles. Figure 4.1 illustrates the latter case, in which there are no traffic signals, and a pedestrian must choose a gap in the traffic flow and cross.

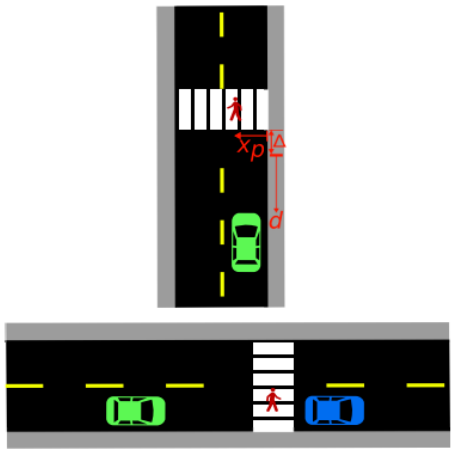


Figure 4.1: Uncontrolled Pedestrian Crosswalks [55]

The bottom of Figure 4.1 shows the schematic of an pedestrian approaching a stream of traffic at an intersection. d represents the distance from the vehicle to a safe point Δ ahead of the crosswalk. x_p

is the position of the pedestrian within the crosswalk, which can be negative for the case when the pedestrian is entering the traffic from the opposite side.

Gap acceptance is one of the primary factors influencing pedestrian behavior at uncontrolled traffic intersections. This refers to how much time is left before the vehicle would enter the crosswalk if it maintains its current speed. It is defined as:

$$\text{gap} = \frac{\text{distance to pedestrian}}{\text{vehicle speed}} = \frac{d}{v} \tag{4.1}$$

It has been reported that pedestrians typically do not cross if the vehicle would enter the crosswalk within 3 seconds or less, but they are very likely to cross if they have more than 7 seconds [55].

The behavior tree will remain in the cruising action unless a pedestrian ahead is detected to enter the driving lane or a cyclist rides in front of the vehicle. In the behavior tree, “beginning to enter” is mathematically defined as the moment the pedestrians begin moving towards the driving lane, regardless of whether they are currently on the sidewalk.

$$\text{inCrosswalk}\,(x_p, \dot{x}_p) = \begin{cases} 1 & : \dot{x}_p \neq 0 \vee 0 \leq x_p \leq x_F \\ 0 & : \text{ otherwise} \end{cases} \tag{4.2}$$

where x_F is the end of the driving lane area. In this work, the definition of x_F will be the same half of the driving lane as the vehicle. The ego vehicle should be able to determine whether to proceed through in an urban scenario that requires yielding and stopping for pedestrians, which can be aided by calculating the time advantage t_{adv} [56] as follows:

$$t_{\text{adv}} = \frac{x_v - x_p}{\dot{x}_p} - \frac{d}{v} \tag{4.3}$$

where x_v is the x position of the vehicle. The designed behavior tree explicitly allows the vehicle to continue driving with minimal deviation from speed of traffic as long as t_{adv} exceeds a predefined threshold.

4.2.3. Analysis

In this work, the development of the BT is based on a thrid party library *BehaviorTree.CPP* [54]. To illustrate, the structure of the online implementation of the behavior tree is shown in Figure 4.2.

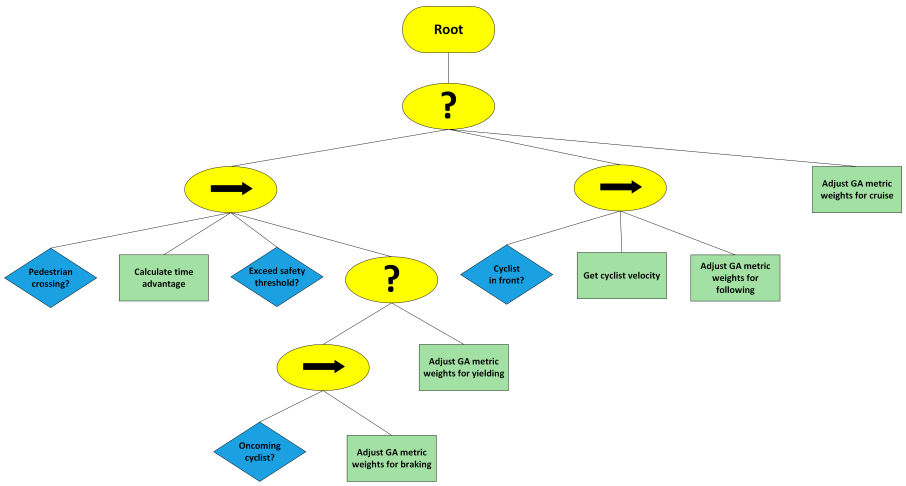


Figure 4.2: Behavior Tree for Online Implementation

It has already been mentioned that the leftmost action should have the highest priority. In the designed BT, the action "calculate time advantage" is defined as an instant action, which is evaluated every tick, which makes the the AsyncAction behavior "braking" the action with the highest priority. Due to

its safety-related consideration, this action is prioritized over the rest of the actions. Moreover, the condition nodes are placed to indicate the specific situation that the ego vehicle is facing. It is the default behavior of the vehicle to cruise, and if all conditions are not met, the vehicle will remain in that state.

The designed BT utilizes an event-driven approach, as described previously. It is based on the premise that nodes that require information about task interruptions are informed when an interruption occurs. Composite nodes, for example, maintain a current-child pointer in order to resume the execution of an already running child rather than starting from the very beginning. Should another action interrupt one of its children, this pointer should be reset. In addition, all AsyncActions receive notifications when an action is about to run, allowing them to abort their own execution in the event it is running. Because of the limited concurrency of workloads via coroutines, the execution of long-running actions has to take place at a higher shared context. The code structure of the behavior tree implementation is shown in Figure 4.3.

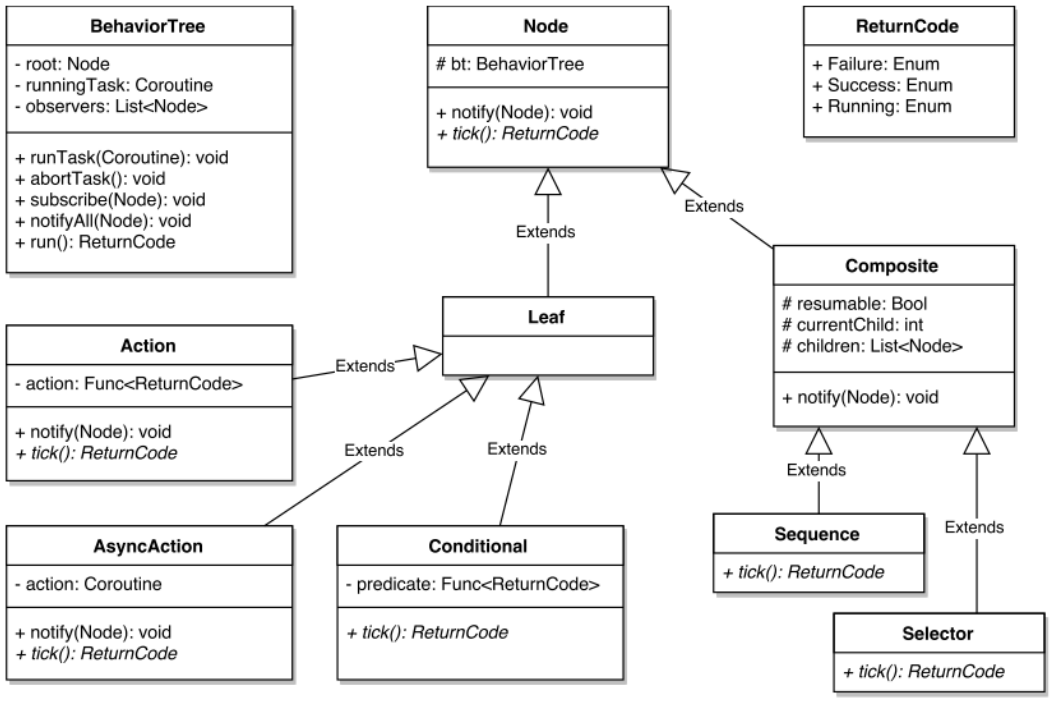


Figure 4.3: Behavior Tree Code Structure

4.3. Conclusion

This chapter described the design of the behavior tree architecture. Some shortcomings of the behavior tree in terms of task priority and concurrency were addressed by introducing asynchronous coroutine. During the execution, the environmental information was processed to tell the current situation that the ego vehicle is facing, where the decisive variable time advantage was introduced. The construction of the BT considered the possible scenarios in the simulation as well as the priority of the actions. Hence major contribution of this work, a decision-making architecture that was able to handle multiple tasks simultaneously in the simulation and output specific actions to the tuning procedure, was presented in this chapter.

5

Simulation Results

The GA-based automatic tuning method that works along with the BT is tested in an open source autonomous driving simulator CARLA [57]. The purpose of CARLA is to serve as a modular and flexible interface so that a wide range of tasks related to autonomous driving can be addressed, and hopefully, help democratize autonomous driving research by providing an easily accessible and customizable tool.

5.1. Vulnerable Road Users Behavior

Before performing the simulation, it is important to define the possible behaviors or actions of the VRUs, so that they act as authentic pedestrians or cyclists in urban traffic. In a purpose of evaluating the performance of the autonomous vehicle, the pedestrians needs to be able to:

- Spawn at the sidewalk near the initial position of the ego vehicle
- Walk or run with predefined and constant velocity
- Cross the road with random chance
- Disregard the ego vehicle

Additionally, the cyclists needs to be able to

- Spawn at the driving areas as the ego vehicle
- Cruise with predefined and constant velocity
- Detect and react properly to the activities of other road users

5.2. Scenarios for Simulations

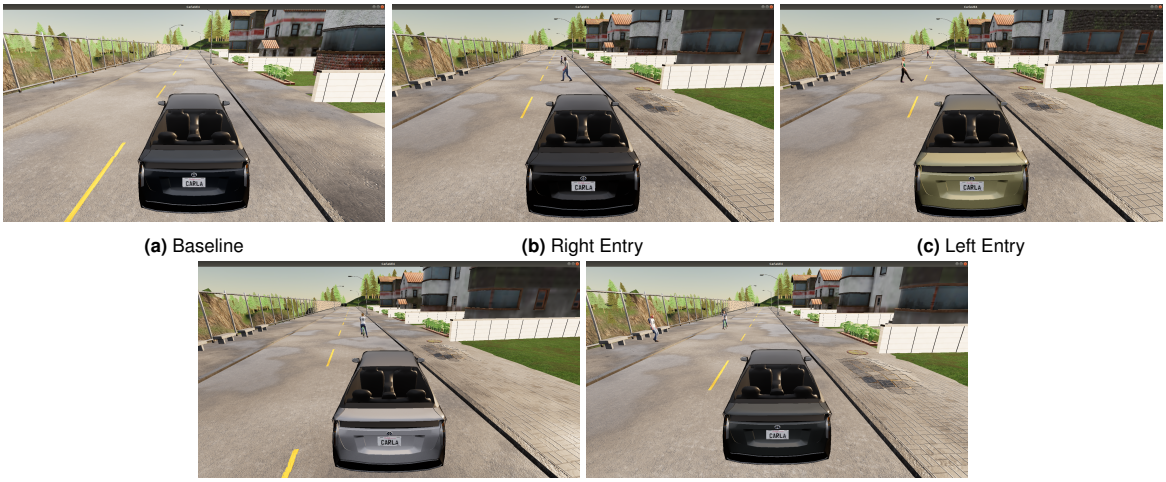
The tested scenario in this work is trying to duplicate an urban traffic conditions where VRUs are present. The ego vehicle is spawned at a given location and then drives towards to destination while avoiding collisions. The global reference path is therefore a straight line in the same driving lane. The designed scenario for simulation is called random crossing scenario, given in Figure 5.1, where a realistic urban traffic scene is built. This scenario is used to test both online and offline tuning method.

Figure 5.1: Random Crossing Scenario

Additionally, different breakdown scenarios are constructed to optimize the weight parameters for the

offline method. The scenarios are shown from Figure 5.2a to Figure 5.2e, which are classified as follows:

- **Baseline** : Scenario without any VRUs
- **Right Entry** : One pedestrian crosses the road from the right side of the ego vehicle
- **Left Entry** : One pedestrian crosses the road from the left side of the ego vehicle
- **Cyclist Following** : One cyclist riding in front of the vehicle with a given distance
- **Cyclist & Pedestrian** : One pedestrian crosses the road from the left while one cyclist riding in front of the vehicle with a given distance



(a) Baseline **(b)** Right Entry **(c)** Left Entry

(d) Cyclist Following **(e)** Cyclist & Pedestrian

Figure 5.2: Breakdown Scenarios

For purposes of comparison with the default MPCC setting, the GA-based automatic tuning method is referred to as GA-MPCC in the rest of the chapter. It is noted that each of the breakdown scenario is simulated for 10 times both for GA-MPCC and MPCC to reduce randomness that potentially occurs in the simulation.

5.3. Simulation Results for Offline Tuning

Figure 5.3 shows the perceptual decrease of the best fitness values from the optimization process of the GA-based offline tuning. The optimal weight parameters of five breakdown scenarios are obtained from the process. The figure shows that all five optimizations are with a consistent trend, the best fitness values decrease to a nearly constant value and close to convergence from around the tenth generation. The optimization is then terminated.

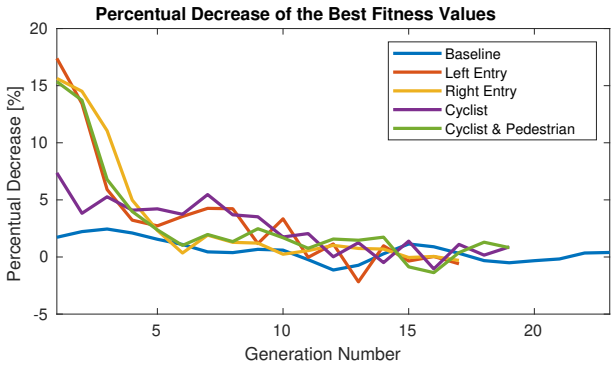


Figure 5.3: Perceptual Decrease of the Best Fitness Values

Table 5.1 shows the weight parameters from the default setting of the MPCC as well GA optimization. In

order to keep the parameters at the same order of magnitude, the constrain bounds of the weights are also determined.

Weights	MPCC	Offline GA					
		Bounds	Baseline	Left Entry	Right Entry	Cyclist	Cyclist & Ped
Q_e	0.02	[0.01, 0.1]	0.0246	0.0591	0.0872	0.0954	0.386
Q_v	0.03	[0.01, 0.1]]	0.0779	0.096	0.0985	0.0577	0.0819
Q_a	0.35	[0.1, 0.6]	0.1046	0.5318	0.4246	0.2082	0.3239
$Q_{\dot{\delta}}$	0.5	[0.1, 1.0]]	0.316	0.655	0.432	0.622	0.53
v_{ref}	4.0	[0.5, 4.0]	4.0	0.6653	0.7398	1.85	0.6469

Table 5.1: Optimized Weight Parameters

5.3.1. Baseline

The objective of the baseline scenario is to reach the goal with minimal deviation from the reference. It is noted that since the baseline scenario is without any VRUs, the variation of the reference velocity is unnecessary, hence this variable would remain a constant in this specific simulation. In order to achieve the objective, the vehicle tends to reach the reference velocity in a short time. Hence, at the expense of greater changes in acceleration, the priority of the velocity error is emphasized.

GA-MPCC vs MPCC

As shown in Figure 5.4, GA-MPCC enables the vehicle to reach the reference velocity in a shorter time with slight overshoot compared to the MPCC. However, GA-MPCC brings a greater change in acceleration and reaches the upper bound of the input.

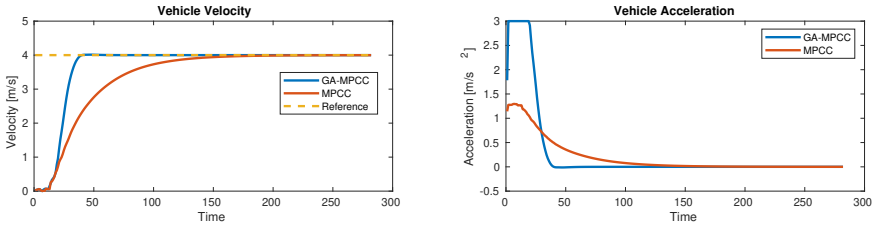


Figure 5.4: Longitudinal States in Baseline Scenario

Figure 5.5 presents the trajectories generated from all simulations, the blue and yellow dashed lines represent the road boundary and the driving lane. The heading of the vehicle is illustrated in Figure 5.6. Since the spawned location of the ego vehicle is not fully centered in the driving lane, initially the ego vehicle would try to steer a certain angle to track the reference path. The result of GA-MPCC shows that the vehicle tends to steer more in order to resume to desired path in a shorter time compared to the MPCC.

5.3.2. Right Entry

In this scenario, the objective of the vehicle is to yield to the pedestrian with minimal deviation from the reference path. Since there is a pedestrian in the scenario, the reference velocity would no longer remain a constant in order to put emphasis on the deviation from the reference velocity.

GA-MPCC vs MPCC

The generated trajectories from all simulations for both application are given in Figure 5.7. The blue dots with different transparency indicate the path of the pedestrian. It is easy to tell that the vehicle controlled by MPCC makes aggressive maneuvers in order to avoid the pedestrian from the right, while the GA-MPCC can remain in the reference path. As a result, GA-MPCC is able to leave a further distance towards the pedestrian, as shown in Figure 5.8.

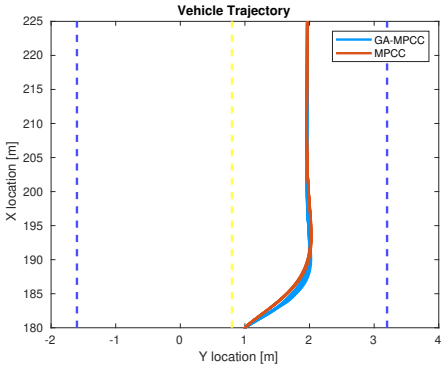


Figure 5.5: Vehicle Trajectories in Baseline Scenario

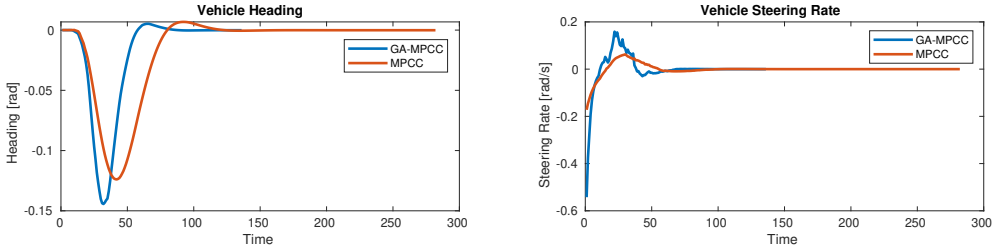


Figure 5.6: Lateral States in Baseline Scenario

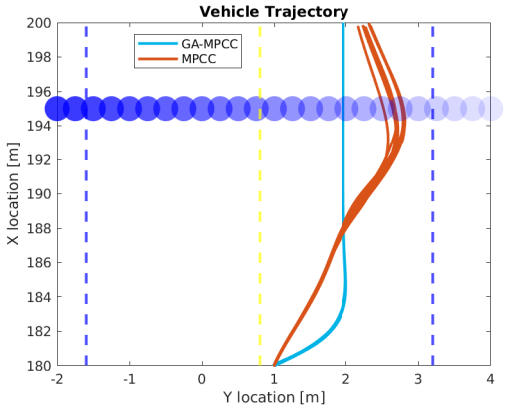


Figure 5.7: Vehicle Trajectories in Right Entry Scenario

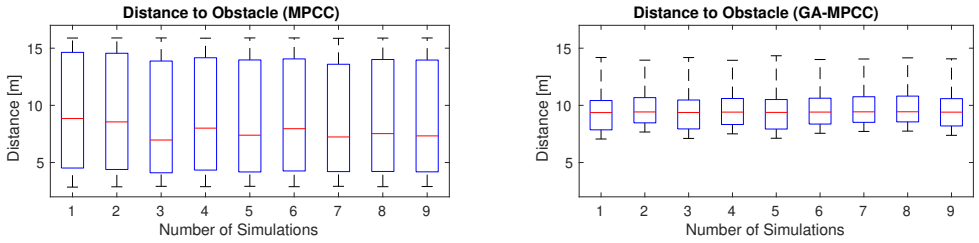


Figure 5.8: Distance to the Pedestrian in Right Entry Scenario

Figure 5.9 presents the longitudinal states along with the distance to the pedestrian. The yellow and black dashed line indicate the entry and exit of the pedestrian crossing, respectively. For GA-MPCC, the velocity remains at a stable level while the acceleration is gradually decreased when the pedestrian is detected and negative numbers are hardly present. On the other hand, the velocity in MPCC fluctuates

greatly because of the significant change in acceleration. It is noted that in MPCC, since the vehicle tends to resume to the reference velocity once it passes the pedestrian, the curve tries to turn back to previous states when near the end of the simulation.

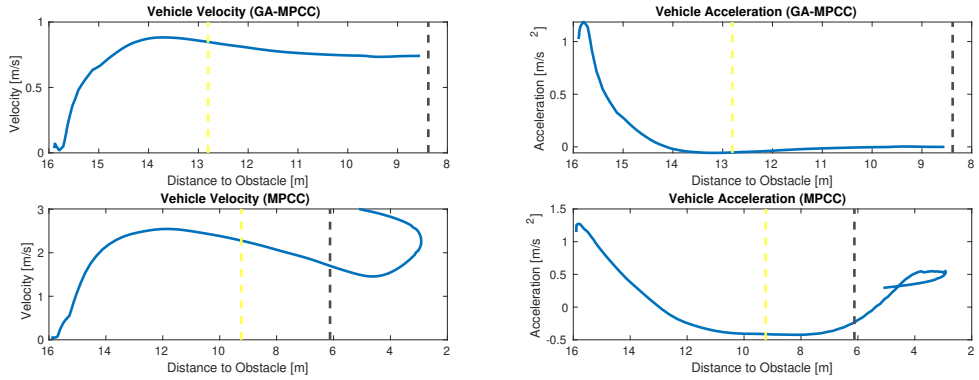


Figure 5.9: Longitudinal States in Right Entry Scenario

5.3.3. Left Entry

The purpose of the this scenario is to impose a special case. Because the vehicle is at a further distance from the pedestrian comes from the opposite side, if the time advantage remains sufficiently high, then the vehicle is able to comfortably continue driving without needing to slow down.

GA-MPCC vs MPCC

The simulated trajectories for both application are given in Figure 5.10. Similar to Right Entry scenario, the vehicle controlled by MPCC tries to make aggressive avoiding maneuvers in order not to lose much velocity, which introduces the risk of safety. While the GA-MPCC can follow the reference path well. Consequently, GA-MPCC brings a larger distance margin, as shown in Figure 5.11.

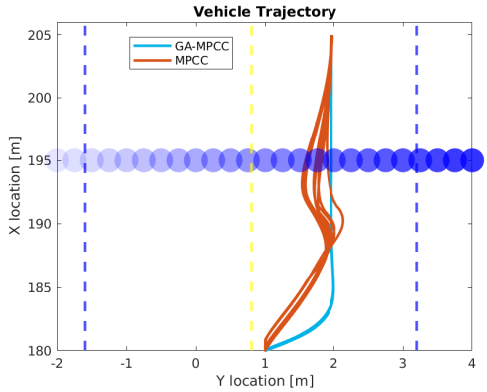


Figure 5.10: Vehicle Trajectories in Left Entry Scenario

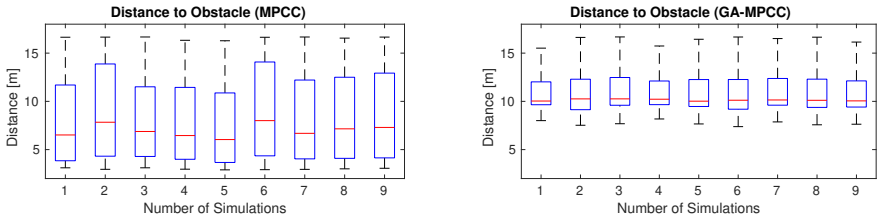


Figure 5.11: Distance to the Pedestrian in Left Entry Scenario

From Figure 5.12, GA-MPCC shows a similar change in term of longitudinal states when facing the pedestrian. However, the velocity in MPCC drops and rises much more significantly according to presence of the pedestrian. This happens because the computed safety margin is small so the vehicle has to slow down significantly to avoid collision, which can also be illustrated from the smaller heading of the vehicle.

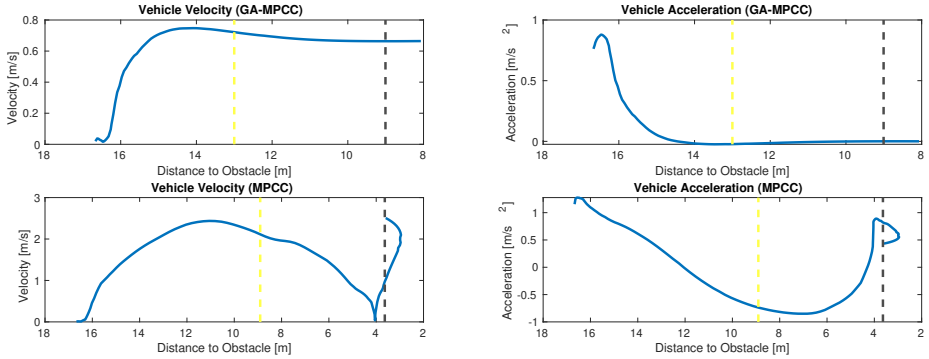


Figure 5.12: Longitudinal States in Left Entry Scenario

5.3.4. Cyclist Following

In this scenario, a cyclist with given constant velocity is spawned at a given distance in front of the pedestrian. The objective of the vehicle is to cruise with a suitable velocity and to remain a safety distance towards the cyclist.

GA-MPCC vs MPCC

The simulated trajectories and longitudinal states of the vehicle are given in Figure 5.13 and Figure 5.14. It shows that MPCC introduces a higher vehicle velocity at the beginning of the simulation, which drops significantly when it approaches the cyclist, and remain at around 1.5 m/s afterwards. On the other hand, the GA-MPCC is able to keep the vehicle in a steady state, where the velocity and acceleration are without significant changes.

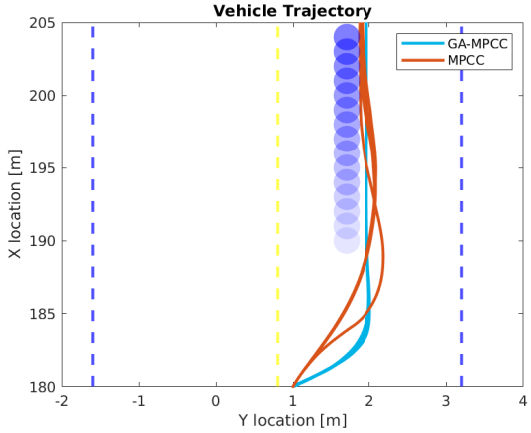


Figure 5.13: Vehicle Trajectories in Cyclist Following Scenario

As a result, GA-MPCC is able to leave a further distance compared to MPCC, as illustrated in Figure 5.15. The reason behind is the vehicle controlled by MPCC tries to pass the cyclist at the beginning of the simulation, as can be observed from the trajectories, and keeps accelerating and decelerating during the following behavior.

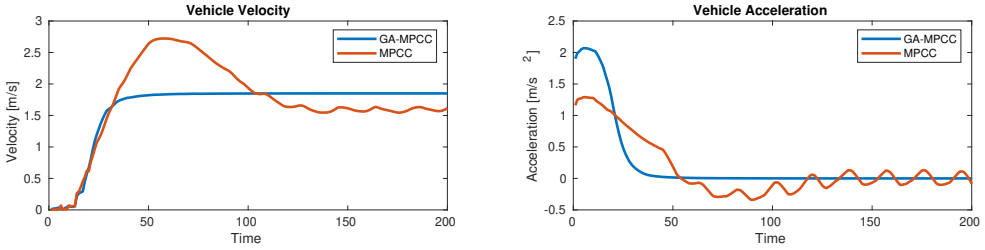


Figure 5.14: Longitudinal States in Cyclist Following Scenario

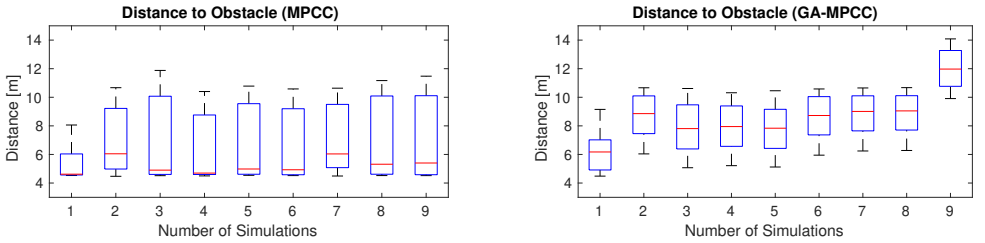


Figure 5.15: Distance to the Pedestrian in Cyclist Following Scenario

5.3.5. Cyclist & Pedestrian

The objective of this scenario is to test how the vehicle will react to the pedestrian ahead while the driving lane from the left is occupied by a cyclist. The pedestrian comes from the left entry for the purpose to impose constraint to the avoidance maneuvers of the vehicle.

GA-MPCC vs MPCC

The trajectories of the vehicle are given in Figure 5.16, where the green circles represent the trajectory of the cyclist. Compared to the Left Entry scenario, in this case, the vehicle from MPCC no longer tries to pass or aggressively avoid the pedestrian before the pedestrian reaches the sidewalk.

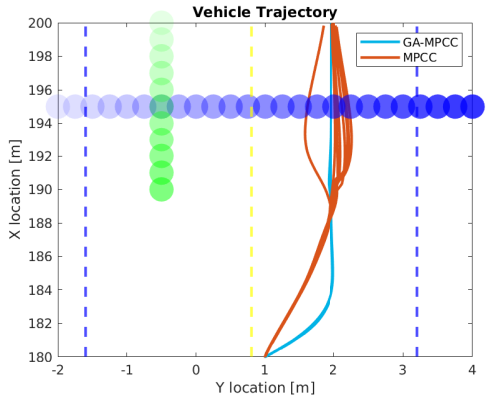


Figure 5.16: Vehicle Trajectories in Cyclist & Pedestrian Scenario

The velocity and acceleration of the vehicle, shown in Figure 5.18, is similar to the case in Left Entry scenario, where the velocity from MPCC drops to 0 from 2.5 m/s as the vehicle has to stop and wait. As a result, the distance to the pedestrian, as shown in Figure 5.15, is close. On the other hand, thanks to the relatively low tracked velocity, the GA-MPCC is able to drive the vehicle steadily, with a less significant change in acceleration.

5.3.6. Random Crossing

All the optimized weight parameters from above are utilized in the Random Crossing scenario with the help of decision-making from the designed BT to evaluate the performance of GA-MPCC against default MPCC. The simulation is run 5 times, each with different traffic condition, to test the adaptability of the

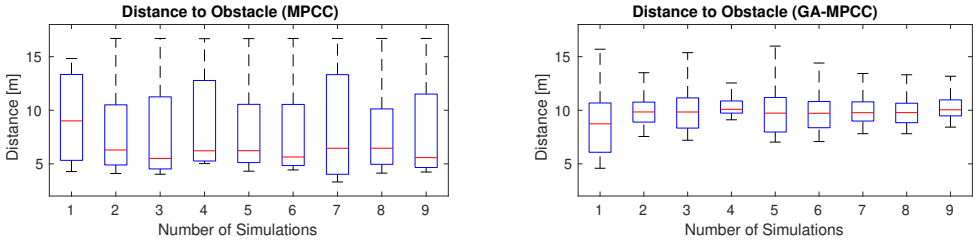


Figure 5.17: Distance to the Pedestrian in Cyclist & Pedestrian Scenario

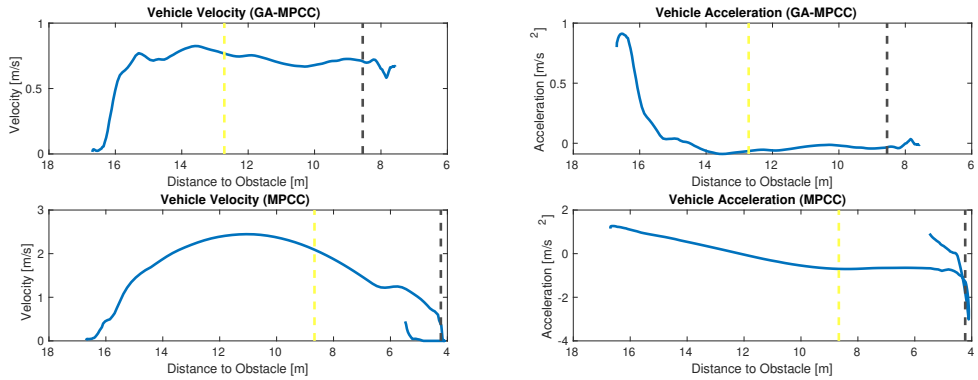


Figure 5.18: Longitudinal States in Cyclist & Pedestrian Scenario

methods. The following figures show the result from one specific simulation. As shown in Figure 5.19 and Figure 5.20, the vehicle with MPCC setting has more adjustment in terms of longitudinal and lateral directions. The velocity even exceeds the reference and nearly reaches 8 m/s for fast-passing at a certain moment. Similarly, the steering is much more aggressive. On the other hand, the GA-MPCC has a relative stable performance in lateral direction, although there are some oscillations from longitudinal and the acceleration turns negative for hard braking. The reason behind is mainly because of the sudden transition from one set of weight parameters to another, which brings a significant change on reference velocity.

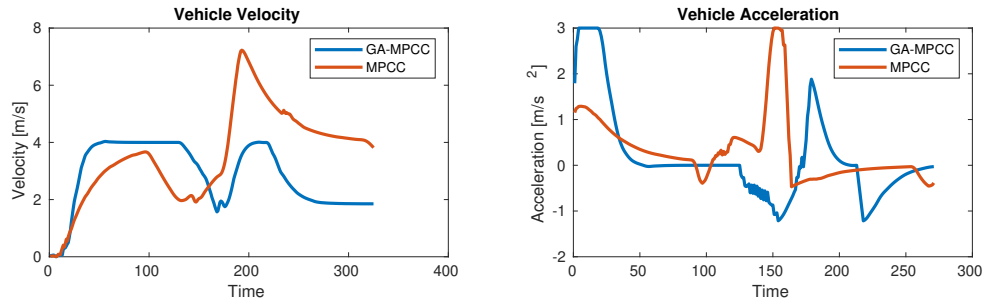


Figure 5.19: Longitudinal States in Random Crossing Scenario (Offline)

Figure 5.21 and 5.22 present the trajectories and safety distance from all 5 simulations. As illustrated above, due to the massive adjustment in throttle and steer, the vehicle under MPCC shows a more exaggerated maneuver, and even cross the driving lane for avoidance in one simulation. As for the GA-MPCC, there are also some emergency avoidance behavior but in a relatively smaller magnitude. Consequently, the GA-MPCC is able to maintain a safer distance towards the VRUs in the simulations.

5.4. Simulation Result for Online Tuning

For online tuning, the importance weights in the fitness function is adjusted in real time with the assistance of the designed BT. Due to the limit computational power, the maximum iteration is limited

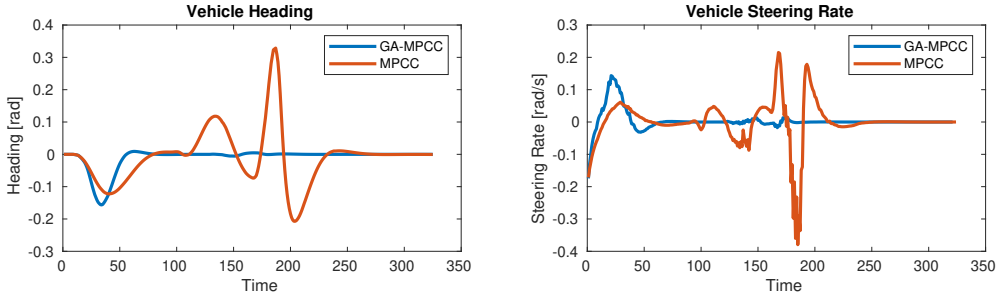


Figure 5.20: Lateral States in Random Crossing Scenario (Offline)

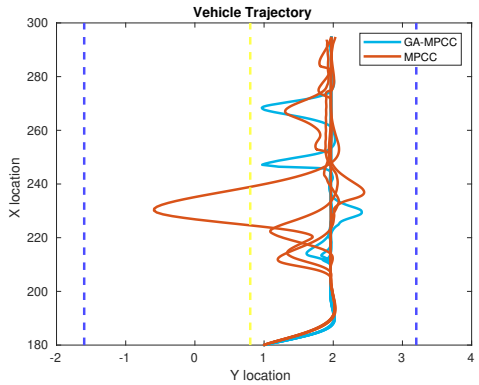


Figure 5.21: Vehicle Trajectories in Random Crossing Scenario (Offline)

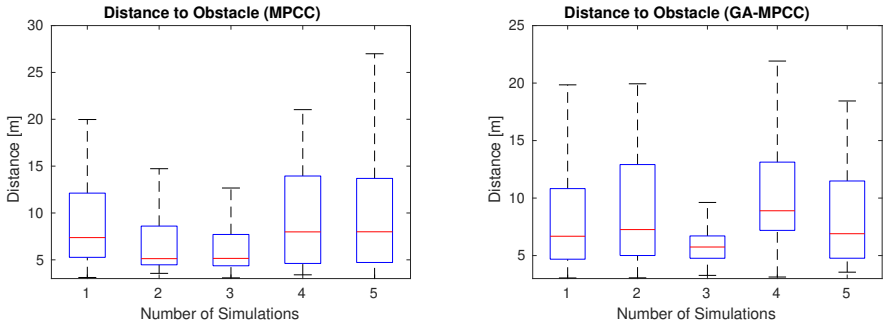


Figure 5.22: Distance to Obstacles in Random Crossing Scenario (Offline)

compare to the offline tuning. Similarly, simulations are run 5 times, each with different traffic condition. Table 5.2 shows the results of all simulations in terms of average computation time.

	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
MPCC (s)	0.0372	0.0344	0.0362	0.0361	0.0354
GA-MPCC (s)	0.0483	0.0455	0.0482	0.0461	0.0430

Table 5.2: Average Computation Time

The following figures show the result from one specific simulation. As shown in Figure 5.23 and Figure 5.24, the vehicle react differently when the VRUs are detected. Under MPCC, the vehicle tries to pass the pedestrian aside with a high velocity, resulting in a highly increased throttle and steer. Facing the same situation, the vehicle controlled by GA-MPCC yields and stops, which is preferable and brings less potential risks in terms of collision.

Figure 5.25 and 5.26 illustrate the trajectories and safety distance from all 5 simulations. Even though in

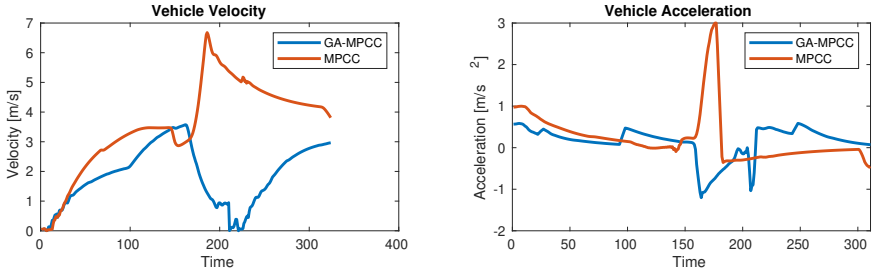


Figure 5.23: Longitudinal States in Random Crossing Scenario (Online)

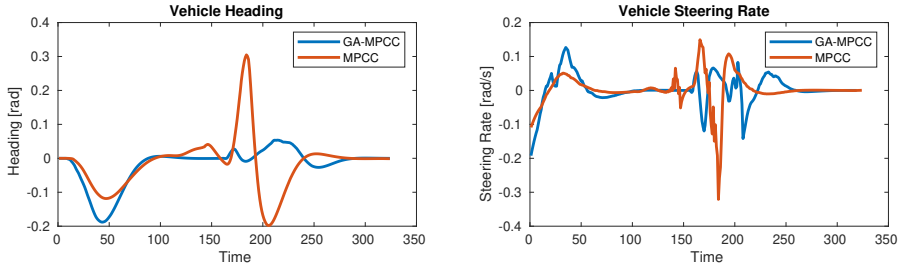


Figure 5.24: Lateral States in Random Crossing Scenario (Online)

one specific simulation, the vehicle under MPCC behaves more aggressively, in most of the simulations, similar behaviors also occur on the GA-MPCC controlled vehicle. The reason behind is due to the limit number of iteration, the weight parameters being optimized in GA haven't converged to a decent level, resulting in less improved performance. However, in terms of distance towards the VRUs, the GA-MPCC is still proved safer as the margins are higher in most of the cases.

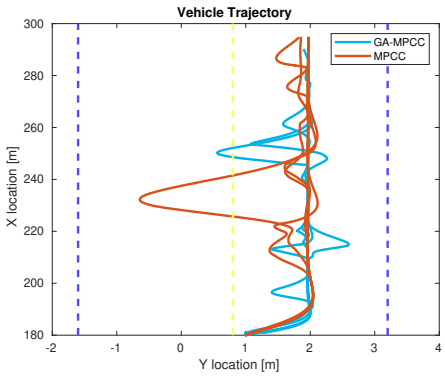


Figure 5.25: Vehicle Trajectories in Random Crossing Scenario (Online)

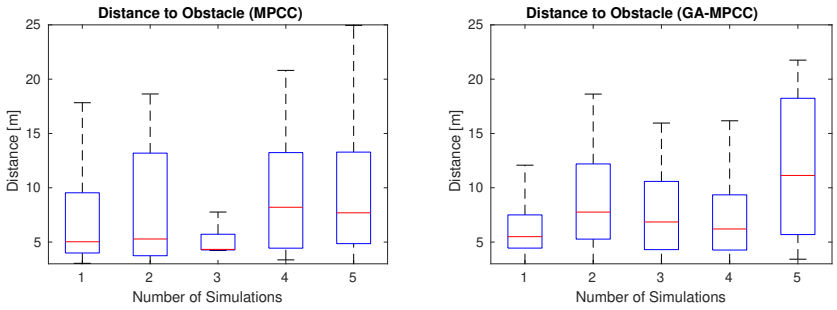


Figure 5.26: Distance to Obstacles in Random Crossing Scenario (Online)

5.5. Conclusion

This chapter discussed the simulation results that showcased the capabilities of the automatic tuning method. Design of the scenarios was important to conduct the simulations. CARLA simulator was used to replicate urban traffic conditions for simulations. Several breakdown scenarios were used to enumerate the potential situations that may occur in an urban scenario. Before discussing the actual simulation results, the results of the GA optimization used for later simulations were discussed.

The performance of the GA-MPCC was compared with the default MPCC setting. For offline tuning, 5 breakdown scenarios were simulated firstly. The results showed a decent improvement in terms of less adjustment in throttle and steer when facing the VRUs. All weight parameters from the breakdown scenarios were then tested in a more complex random crossing scenario with the aid of the designed BT. The vehicle equipped with GA-MPCC behaved the way it was expected to and brought less aggressive maneuvers with more consideration in terms of leaving more safety distance to the VRUs. As for the online tuning, a real time tuning procedure based on GA was applied to the controller. The results showed some improvement regrading different decisions when facing the VRUs. However, in some cases, the vehicle equipped with GA-MPCC still behaved some risky maneuvers, such as try to pass the pedestrian aside with high speed. Since these simulations were performed using CARLA, which provides an accurate physical vehicle plant and simulation environment, the results can be expected to be trustworthy.

6

Conclusion

This thesis explored the challenges in variable tuning in MPC, which formed the gap between two disjointed research fields together with decision-making system in autonomous driving. After going through the literature, it was found that there were extensive researches on both fields, but few combined them into a single approach. The literature about decision-making systems mainly focuses on high-level driving behaviors, while the researches on automatic tuning methods rarely rely on a decision-making system. However, tuning MPC variables depending on the environmental information would greatly improve the adaptability of controller. Genetic algorithm and behavior tree were then selected with reasoning as the primary approaches of the thesis.

The thesis work was based on the MPCC motion planner used in the platform SafeVRU, which focuses on the urban scenarios where VRUs are present. First, the tuning method based on GA was developed, different parameters and methods for operators were chosen. Then a BT that was able to acquire and process environmental information was developed. The BT was designed to be capable of handling multiple tasks simultaneously for long running behaviors, and prioritising specific actions for safety consideration. Then the algorithm was integrated with the designed BT and tested in the simulation. The proposed automatic tuning method was tested both online and offline. The method was compared with the default MPCC setting and performed well for different breakdown scenarios as well as a complex urban scenario.

6.1. Limitations and Future Work

The genetic algorithm and behavior tree employed in this thesis could be improved in the following aspects.

1. Only single objective GA is used in this work, however, more variants of genetic algorithm can be employed to test their performance. For example, multi-objective GA does not combine individual metrics but provides a Pareto front of the MPC weights, which may describe the dependencies of the metrics and also provide an insight on possible minimization of them.

2. Only weight variables are being automatically tuned in this thesis. However, to increase the adaptability of the controller, other structural parameters can also be considered.

3. As for the less improved performance from the online tuning method, with higher computational power, the maximum iteration of the online algorithm can be extended so that a better converged results may be acquired.

4. The results obtained from the simulation are mostly conservative, as illustrated from the safety distance towards the VRUs, one can adjust the fitness function to achieve different performance for other objectives.

5. As a rule-based approach, the behavior tree has a shortcoming of being not able to consider all possible situations or to handle the uncertainty in a traffic condition. Therefore, learning-based or

uncertainty-based approaches that could compensate this drawback can be employed to evaluate the possibility.

6. The simulation environment being tested in this thesis is limited to a two-lane scenario with simple operations. To test more functionality of the controller and the tuning method, a more complicated scenario can be employed.

A

Simulation Results

A.1. Genetic Algorithm

A.1.1. Parameters

Parameter	Online GA	Offline GA
Population size	10	10
Maximum generation	10	40
Crossover rate	0.5	0.3
Mutation rate	0.1	0.05

Table A.1: Parameters in GA

A.1.2. Baseline Scenario

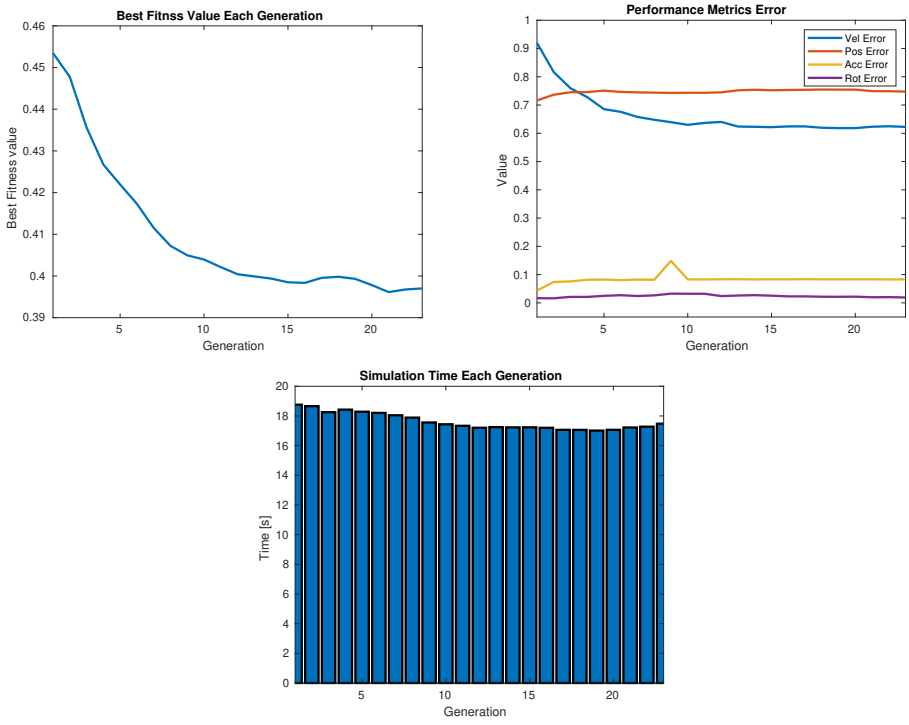


Figure A.1: Simulation Results in Baseline Scenario

A.1.3. Right Entry Scenario

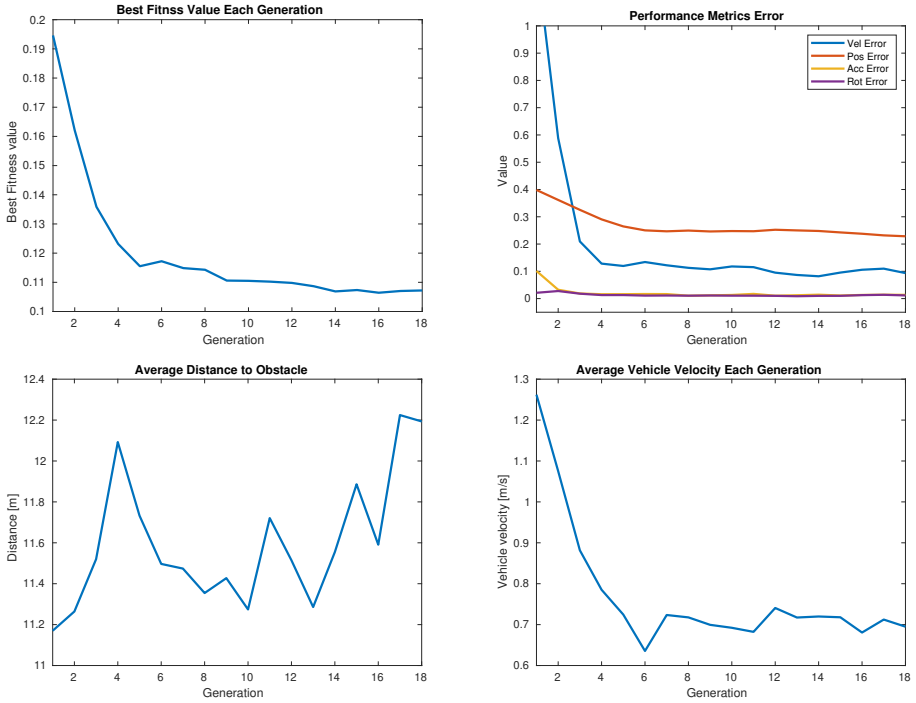


Figure A.2: Simulation Results for Right Entry Scenario

A.1.4. Left Entry Scenario

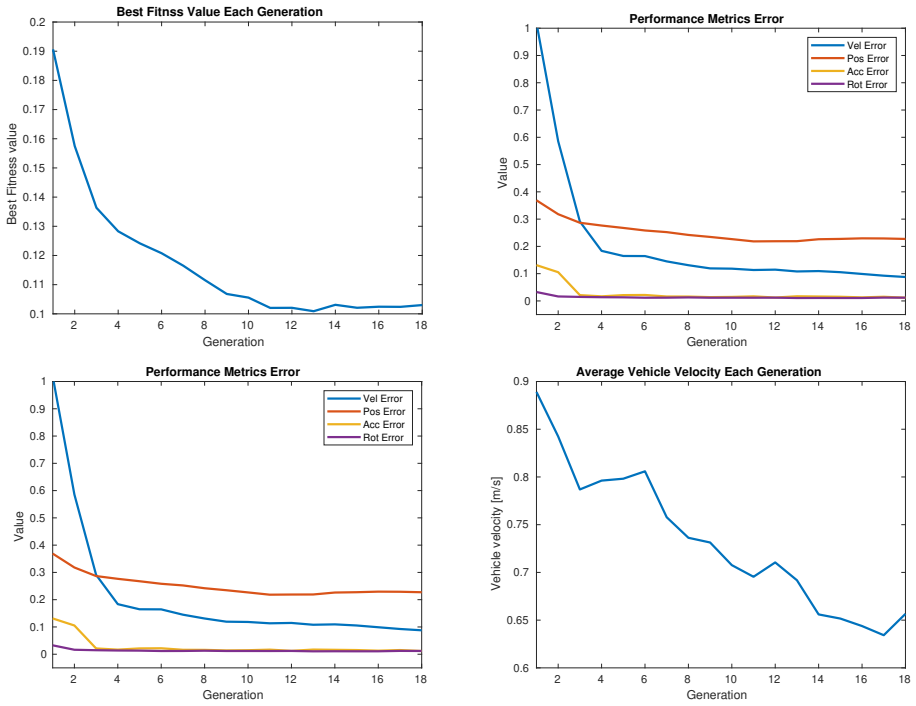


Figure A.3: Simulation Results for Left Entry Scenario

A.1.5. Cyclist Following Scenario

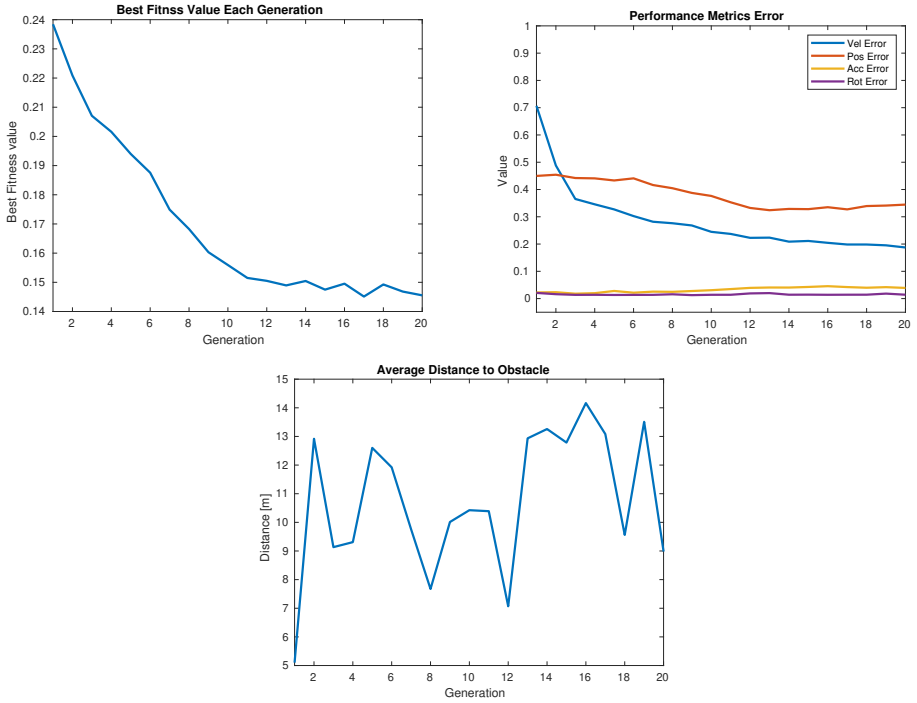


Figure A.4: Simulation Results for Cyclist Following Scenario

A.1.6. Cyclist & Pedestrian Scenario

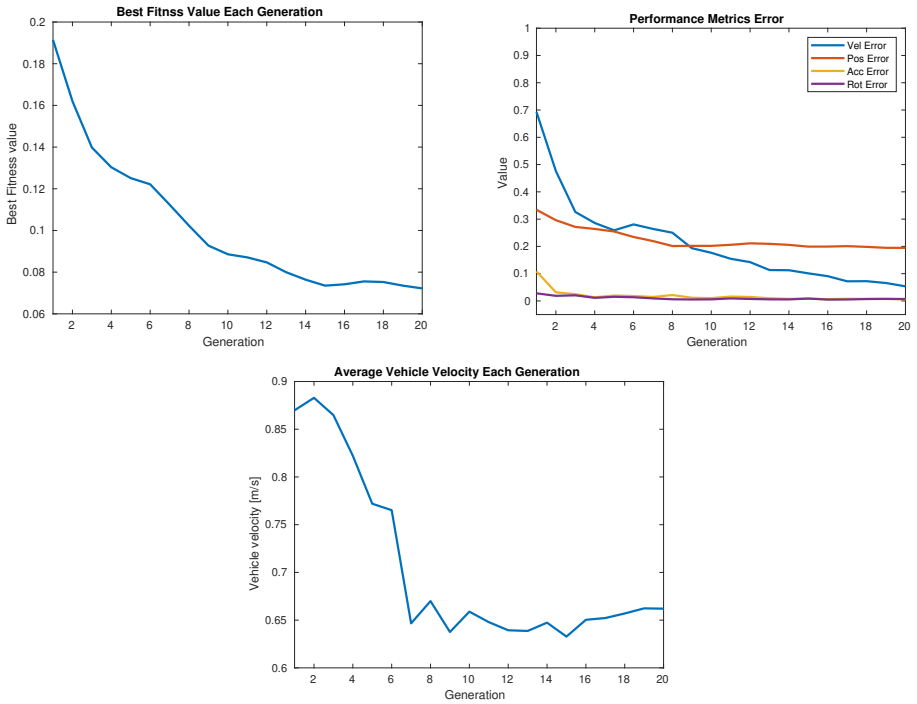


Figure A.5: Simulation Results for Cyclist & Pedestrian Scenario

Bibliography

[1] Jesse Levinson et al. “Towards fully autonomous driving: Systems and algorithms”. In: *2011 IEEE intelligent vehicles symposium (IV)*. IEEE. 2011, pp. 163–168.

[2] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. “Planning and decision-making for autonomous vehicles”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 187–210.

[3] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[4] Karl Johan Åström, Tore Hägglund, and Karl J Astrom. *Advanced PID control*. Vol. 461. ISA-The Instrumentation, Systems, and Automation Society Research Triangle . . ., 2006.

[5] Qi Liu et al. “Decision-Making Technology for Autonomous Vehicles: Learning-Based Methods, Applications and Future Outlook”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 30–37.

[6] Richard Matthaei and Markus Maurer. “Autonomous driving–a top-down-approach”. In: *at-Automatisierungstechnik* 63.3 (2015), pp. 155–167.

[7] Laura Ferranti et al. “SafeVRU: A research platform for the interaction of self-driving vehicles with vulnerable road users”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1660–1666.

[8] Timm Faulwasser, Benjamin Kern, and Rolf Findeisen. “Model predictive path-following for constrained nonlinear systems”. In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 8642–8647.

[9] Wilko Schwarting et al. “Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.9 (2017), pp. 2994–3008.

[10] K Yamuna Rani and Heinz Unbehauen. “Study of predictive controller tuning methods”. In: *Automatica* 33.12 (1997), pp. 2243–2248.

[11] Jorge L Garriga and Masoud Soroush. “Model predictive control tuning methods: A review”. In: *Industrial & Engineering Chemistry Research* 49.8 (2010), pp. 3505–3515.

[12] Mohammed Alhajeri and Masoud Soroush. “Tuning guidelines for model-predictive control”. In: *Industrial & Engineering Chemistry Research* 59.10 (2020), pp. 4177–4191.

[13] Willy Wojsznis et al. “Practical approach to tuning MPC”. In: *ISA transactions* 42.1 (2003), pp. 149–162.

[14] Peyman Bagheri and Ali Khaki Sedigh. “Analytical approach to tuning of model predictive control for first-order plus dead time models”. In: *IET Control Theory & Applications* 7.14 (2013), pp. 1806–1817.

[15] Jorge L Garriga and Masoud Soroush. “Model predictive controller tuning via eigenvalue placement”. In: *2008 American control conference*. IEEE. 2008, pp. 429–434.

[16] Quang N Tran, Leyla Özkan, and ACPM Backx. “Generalized predictive control tuning by controller matching”. In: *Journal of Process Control* 25 (2015), pp. 1–18.

[17] Tahereh Gholaminejad, Ali Khaki-Sedigh, and Peyman Bagheri. “Adaptive Tuning of Model Predictive Control Parameters Based on Analytical Results”. In: *AUT Journal of Modeling and Simulation* 50.2 (2018), pp. 109–116.

[18] Dimche Kostadinov and Davide Scaramuzza. “Online weight-adaptive nonlinear model predictive control”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 1180–1185.

[19] M Francisco and P Vega. “Automatic tuning of model predictive controllers based on multiobjective optimization”. In: *Latin American applied research* 40.3 (2010), pp. 255–265.

[20] André Shigueo Yamashita, Antonio Carlos Zanin, and Darci Odloak. “Tuning the model predictive control of a crude distillation unit”. In: *ISA transactions* 60 (2016), pp. 178–190.

[21] Arash Mohammadi et al. “Multiobjective and interactive genetic algorithms for weight tuning of a model predictive control-based motion cueing algorithm”. In: *IEEE transactions on cybernetics* 49.9 (2018), pp. 3471–3481.

[22] Valarmathi Ramasamy et al. “Optimal tuning of model predictive controller weights using genetic algorithm with interactive decision tree for industrial cement kiln process”. In: *Processes* 7.12 (2019), p. 938.

[23] Haoyang Fan et al. “An auto-tuning framework for autonomous vehicles”. In: *arXiv preprint arXiv:1808.04913* (2018).

[24] Alex S Ira et al. “A machine learning approach for tuning model predictive controllers”. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2018, pp. 2003–2008.

[25] Arash Mohammadi et al. “Optimizing model predictive control horizons using genetic algorithm for motion cueing algorithm”. In: *Expert Systems with Applications* 92 (2018), pp. 73–81.

[26] Gesner A Nery Júnior, Márcio AF Martins, and Ricardo Kalid. “A PSO-based optimal tuning strategy for constrained multivariable predictive controllers with model uncertainty”. In: *ISA transactions* 53.2 (2014), pp. 560–567.

[27] Robert Alfred Chin. “Model Predictive Controller Tuning by Machine Learning and Ordinal Optimisation”. PhD thesis. 2021.

[28] Sandip K Lahiri. *Multivariable predictive control: Applications in industry*. John Wiley & Sons, 2017.

[29] RC Zhao et al. “Real-time weighted multi-objective model predictive controller for adaptive cruise control systems”. In: *International journal of automotive technology* 18.2 (2017), pp. 279–292.

[30] Mohammad B Shadmand, Sarthak Jain, and Robert S Balog. “Autotuning technique for the cost function weight factors in model predictive control for power electronic interfaces”. In: *IEEE Journal of Emerging and Selected Topics in Power Electronics* 7.2 (2018), pp. 1408–1420.

[31] Raony M Fontes, Márcio AF Martins, and Darci Odloak. “An automatic tuning method for model predictive control strategies”. In: *Industrial & Engineering Chemistry Research* 58.47 (2019), pp. 21602–21613.

[32] Yang Yu et al. “Hierarchical Reinforcement Learning Combined with Motion Primitives for Automated Overtaking”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1–6.

[33] Ekim Yurtsever et al. “A survey of autonomous driving: Common practices and emerging technologies”. In: *IEEE access* 8 (2020), pp. 58443–58469.

[34] Nelson De Moura et al. “Ethical decision making for autonomous vehicles”. In: *2020 IEEE intelligent vehicles symposium (iv)*. IEEE. 2020, pp. 2006–2013.

[35] Magnus Olsson. *Behavior Trees for decision-making in Autonomous Driving*. 2016.

[36] David Harel. “Statecharts: A visual formalism for complex systems”. In: *Science of computer programming* 8.3 (1987), pp. 231–274.

[37] Qi Chen, Umit Ozguner, and Keith Redmill. “Ohio state university at the 2004 darpa grand challenge: Developing a completely autonomous vehicle”. In: *IEEE Intelligent Systems* 19.5 (2004), pp. 8–11.

[38] David C Conner and Justin Willis. “Flexible navigation: Finite state machine-based integrated navigation and control for ROS enabled robots”. In: *SoutheastCon 2017*. IEEE. 2017, pp. 1–8.

[39] Kirk YW Scheper et al. “Behavior trees for evolutionary robotics”. In: *Artificial life* 22.1 (2016), pp. 23–48.

[40] Alejandro Marzinotto et al. “Towards a unified behavior trees framework for robot control”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 5420–5427.

[41] Michael P Vitus and Claire J Tomlin. “A probabilistic approach to planning and control in autonomous urban driving”. In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 2459–2464.

[42] Fanlin Meng et al. “Dynamic decision making in lane change: Game theory with receding horizon”. In: *2016 UKACC 11th International Conference on Control (CONTROL)*. IEEE. 2016, pp. 1–6.

[43] Davide Castelvecchi. “Can we open the black box of AI?” In: *Nature News* 538.7623 (2016), p. 20.

[44] Jason Kong et al. “Kinematic and dynamic vehicle models for autonomous driving control design”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 1094–1099.

[45] Denise Lam, Chris Manzie, and Malcolm Good. “Model predictive contouring control”. In: *49th IEEE Conference on Decision and Control (CDC)*. IEEE. 2010, pp. 6137–6142.

[46] Bruno Brito et al. “Model predictive contouring control for collision avoidance in unstructured dynamic environments”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4459–4466.

[47] E Wijanarko and H Grandis. “Binary Coded Genetic Algorithm (BCGA) with Multi-Point Cross-Over for Magnetotelluric (MT) 1D Data Inversion”. In: *IOP Conference Series: Earth and Environmental Science*. Vol. 318. 1. IOP Publishing. 2019, p. 012029.

[48] Noraini Mohd Razali, John Geraghty, et al. “Genetic algorithm performance with different selection strategies in solving TSP”. In: *Proceedings of the world congress on engineering*. Vol. 2. 1. International Association of Engineers Hong Kong. 2011, pp. 1–6.

[49] Gilbert Syswerda et al. “Uniform crossover in genetic algorithms.” In: *ICGA*. Vol. 3. 1989, pp. 2–9.

[50] Mohamed sadek Ben Ameur, Anis Sakly, and Abdellatif Mtibaa. “Implementation of genetic algorithms using FPGA technology”. In: *Proceedings of the Annual FPGA Conference*. 2012, pp. 1–9.

[51] Oliver Kramer. “Genetic algorithms”. In: *Genetic algorithm essentials*. Springer, 2017, pp. 11–19.

[52] Martın Safe et al. “On stopping criteria for genetic algorithms”. In: *Brazilian Symposium on Artificial Intelligence*. Springer. 2004, pp. 405–413.

[53] Mihael Ankerst et al. “Visual classification: an interactive approach to decision tree construction”. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 1999, pp. 392–396.

[54] Davide Faconti. *Mood2be: Models and tools to design robotic behaviors*. 2019.

[55] Nitin R Kapania et al. “A hybrid control design for autonomous vehicles at uncontrolled crosswalks”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1604–1611.

[56] Baiming Chen, Ding Zhao, and Huei Peng. “Evaluation of automated vehicles encountering pedestrians at unsignalized crossings”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1679–1685.

[57] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.