Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

GRID BASED CONTROL STRATEGY FOR HYBRID VEHICLES

Patricia Wessel

Examiners:
Prof. Dr. Erika Ábrahám
PD Dr. Walter Unger

Additional Advisor:
Dipl.-Inform. Johanna Nellen

Aachen, 29.09.2016

Abstract

In the past years, hybrid electric vehicles (HEVs) have become increasingly popular as they allow to combine the advantages of full electric vehicles and the advantages of vehicles with internal combustion engines. In HEVs a control strategy distributes the requested torque between the available engines. In this thesis, a control strategy is presented which relies on a discrete grid. Offline torque distributions for discretised inputs are computed using an optimisation algorithm and stored in a grid. Online our control strategy interpolates torque distributions for arbitrary inputs using the precomputed grid. As interpolation is fast and does not need much processing power, the running time of this grid based control strategy is much lower than the running time of control strategies based on optimal control.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Patricia Wessel
Aachen, den 29. September 2016

Acknowledgements

As I have ever been interested in cars and automotive engineering, I searched for a bachelor thesis which gives me the opportunity to combine these personal interests with computer science. I am very happy that this was made possible by the task of this thesis as I was able to use methods of theoretical computer science in order to develop solutions for problems in automotive engineering.

For this reason, I would like to thank Prof. Dr. Erika Ábrahám for offering me the chance to write this bachelor thesis and for being my primary reviewer. Moreover, I want to thank PD Dr. Walter Unger for being second reviewer of this thesis. Many thanks also go to Dipl.-Inform. Johanna Nellen, who spent countless hours supervising this thesis and the corresponding program code. Furthermore, I want to thank Lukas Netz and Rebecca Haehn, whose work at the project has made a huge contribution to the success of this thesis. Finally, I give thanks to my family and my friends for supporting me all the time, especially in the most difficult phases of this thesis. Last but not least, thank you Louis for your encouragement and patience during all the time in the last months, and for ensuring that we always had something delicious to eat in the evening!

Contents

Chapter 1

Introduction

In times where climate change and CO_2 emissions has turned to the public focus, the environmental acceptability of vehicles has become an important issue. To reduce the global fuel consumption and the pollutant emissions of vehicles, pure electric cars have been developed and sold. However, vehicles with only an electrical power unit are expensive, the battery charging time may take several hours and the driving range is limited.

Hybrid electric vehicles (HEVs) represent a good trade-off between pure electric vehicles and cars equipped with a combustion engine. They are characterised by containing an internal combustion engine (ICE) as well as at least one electrical motor (EM). Hence it is possible to benefit from the best aspects of each technology: The ICE is perfect for high torque values and guarantees a long driving range, whereas the EM does not produce emissions and is suitable for low torque values [NWN+15].

As the requested torque has to be distributed between the engines of a hybrid electric vehicle, control strategies are used to determine such a torque distribution. A control strategy takes as input the current state of the car as well as the drivers requested torque and computes a torque distribution with the goal to optimise output parameters as the fuel consumption or the emissions.

In this thesis a control strategy for hybrid electric vehicles is presented which is based on a discrete grid. The grid contains precomputed control values for a discretised state space. In our case, the state space consists of the current state of the car and the requested velocities for each time point in the prediction horizon. The strategy uses the grid to determine the control value for the current input. As normally the input is not contained in the discretised grid, the requested control must be interpolated using control values of available points in the neighbourhood in this grid.

The control values of the grid points are precomputed offline using a selected optimisation method. In this thesis, genetic algorithms are chosen to compute the control values in the grid. However, the strategy and the grid are not limited to

this approach. Different optimisation procedures can be used, e. g. it might also be possible to precompute the control values using dynamic programming. As a result, the grid based strategy is very flexible due to the fact that the algorithm which is used to precompute the control values in the grid is exchangeable.

Optimisation procedures are time expensive and the processing power within the car is limited. Thus a control strategy based on optimal control might not be suitable. The grid based control strategy however uses interpolation to compute the control values. This aspect is a huge advantage of the strategy due to the fact that interpolation is fast and does not need much processing power. Since the computation of the control values of the grid points is done offline as a preprocessing step, it is not time critical and can take an arbitrary period of time. However, the strategy takes advantages of the quality of the optimisation method as it is used to precompute the control values in the grid.

This thesis is structured as follows: In the next chapter, the vehicle model which is used to simulate the control strategy is introduced and a general overview of control strategies as well as linear interpolation is given. Furthermore, genetic algorithms are presented. In the third chapter, the grid based control strategy is explained. Thereby, we consider the construction of the grid as well as the interpolation of the control values. Afterwards, in the fourth chapter a procedure to optimise the grid is introduced: By inserting new entries, the aim is to refine the grid at specific points so that requested control values can be interpolated more precisely. We implemented the grid based control strategy as well as the grid and the optimisation procedure. Therefore, in the fifth chapter the results of our implementation are discussed. Finally, in the last chapter a conclusion is given.

Chapter 2

Preliminaries

In this chapter, the vehicle model which is used to simulate control strategies is introduced. Afterwards, a short overview of control strategies in general is given. The grid based control strategy uses linear interpolation to approximate requested control values. Therefore, in the third section an overview of linear interpolation is provided. We use a genetic algorithm to precompute the control values in the grid. For this reason, in the last section genetic algorithms are presented.

2.1 Vehicle Model

The grid based control strategy is simulated on a simplified model of a first generation Toyota Prius. A schematic diagram of the vehicle is presented in Figure 2.1. This vehicle consists of an internal combustion engine (ICE) as well as an electrical motor (EM) with a corresponding battery. Both power units are connected to the same axis which is related with a manual transmission gearbox. Thus the engines and the gearbox move with the same angular velocity: $\omega_{em} = \omega_{ice} = \omega_{gb}$. The gearbox is linked to the wheels and uses a gear ratio r_i for a gear i to transform the angular velocity at the gearbox to the angular velocity at the wheels: $\omega_{wh} = \omega_{gb}/r_i$ [NWN+15,

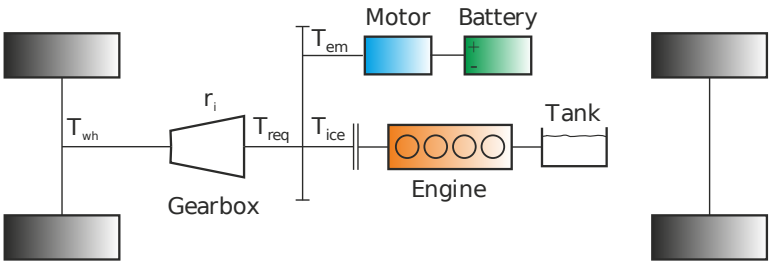

Figure 2.1: Simplified model of the hybrid electric vehicle [GJN+15b]

GJN+15a].

If the driver requests an acceleration a, the required torque at the wheels can be determined by

$$T_{wh} = r_{wh}\big(\frac{1}{2}\rho C_d A v^2 + (m + m_r)a + mgf_r cos(\theta) + mgsin(\theta)\big), \tag{2.1}$$

where r_{wh} denotes the wheel radius, ρ the density of air, C_d the air drag resistance, A the frontal area of the vehicle, $v = r_{wh}\omega_{wh}$ the velocity of the vehicle, m and m_r the mass of the vehicle and the equivalent mass of the rotating parts of the vehicle, a the acceleration, g the acceleration of gravity, f_r the rolling resistance and θ the road slope [NWN+15, GJN+15a].

The requested torque at the gear box which has to be generated by the engines to accelerate the car can be determined by

$$T_{req} = \frac{1}{\eta_{gb} r_i}(T_{wh} + T_{br}). \tag{2.2}$$

$1/(\eta_{gb} r_i)$ denotes a constant factor consisting of the mechanical transmission efficiency η_{gb}, which is assumed to be constant, and the corresponding gear ratio r_i. T_{br} describes the torque at the brakes [NWN+15, GJN+15a].

As the power units are connected to the same axis, they take effect on the drive train simultaneously. This has the advantage that the engines can be used either separately or in combination. In addition, the requested torque T_{req} which has to be generated can be split up to the two engines. Therefore, T_{req} can be described by

$$T_{req} = T_{ice} + T_{em}, \tag{2.3}$$

where T_{ice} is the torque which is produced by the ICE and T_{em} describes the torque to be provided by the EM. Moreover, the model is considered to be idealised, i. e. the dynamics of the engines are ignored and the torque responses are assumed to be immediately [NWN+15, GJN+15a].

While the combustion engine is powered with fuel, the electrical motor receives its energy from a battery. The fuel consumption $\dot{m}_f$ of the combustion engine and the input power P_{em} of the electrical motor can be described by using non-linear, discrete functions depending on speed and torque which will be interpolated during the simulation [GJN+15b]. The battery has a state of charge (SoC; symbol: s) whose change can be described by the following formula:

$$\dot{s} = -\frac{I}{Q_{max}} = -\frac{P_{em}}{U_{oc} Q_{max}}. \tag{2.4}$$

Q_{max} denotes the maximal battery capacity, I the battery current electricity and U_{oc} the battery open circuit voltage. To protect the battery and to guarantee its durability, the control strategy shall keep the SoC near a reference value $s_{ref} = 0.6$. Furthermore, it is possible to recharge the battery with an excess of torque generated by the ICE or by recuperating electric energy while braking [NWN+15].

In addition, the following physical limits of the system are given to ensure a long life time of the hybrid power train [NWN+15]:

$$\begin{array}{rcccl} \omega_{ice,min} & \leq & \omega_{ice} & \leq & \omega_{ice,max} \\ \omega_{em,min} & \leq & \omega_{em} & \leq & \omega_{em,max} \\ 0 & \leq & T_{ice} & \leq & T_{ice,max}(\omega_{ice}) \\ T_{em,min}(\omega_{em}) & \leq & T_{em} & \leq & T_{em,max}(\omega_{em}) \\ 0.5 & \leq & s & \leq & 0.7 \end{array} \tag{2.5}$$

While T_{ice} is always non-negative, T_{em} can also have negative values. This happens when the battery is recharged. $s_{min} = 0.5$ and $s_{max} = 0.7$ delimit the valid area for the battery state of charge to guarantee a long battery life time [NWN+15, GJN+15b].

The vehicle model presented in this section is used to simulate control strategies. In the next section, the general structure of a control strategy is considered.

2.2 Control Strategies

Generally, the purpose of a control strategy is to distribute the drivers requested torque between the engines of a hybrid electric vehicle so that output parameters like the fuel consumption, the battery state of charge level or the emissions are optimised. Thereby it has to take care of the physical limits of the corresponding technical system.

In this thesis, we consider control strategies for the vehicle model presented in Figure 2.1. Such a control strategy consists of two parts: In a first block (*split computation block*), a split value that indicates the torque distribution between the two engines is computed. In a second block (*control converter block*), the feasibility of this torque distribution within the physical limits of the car is checked and the concrete amount of torque for each engine as well as the brakes is computed. Moreover, in this block special cases like braking or standing are handled. A graphical illustration is given in Figure 2.2.

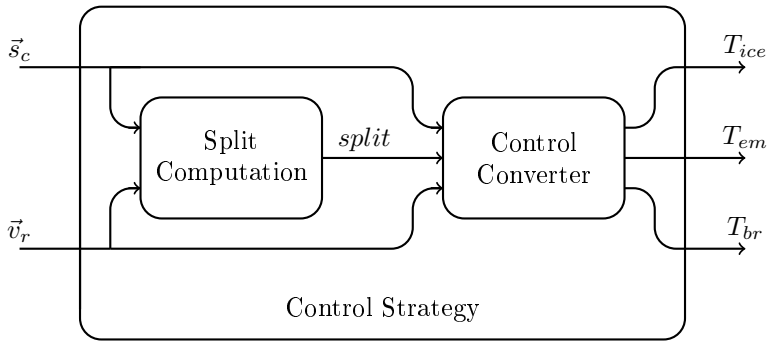

Figure 2.2: Control strategy consisting of the split computation block and the control converter block

The control strategy receives as input in each time step the requested velocities $\vec{v}_r$ as well as the current state $\vec{s}_c$ of the car. In the case that a predictive control strategy

is used, i. e. future driving conditions for the next time steps within the prediction horizon of size p are given, $\vec{v}_r$ contains the requested velocity for the current time step as well as the $p-1$ requested velocities for the following time steps. Otherwise, $\vec{v}_r$ describes only the requested velocity for the current time step. The current vehicle state contains information about for example the actual velocity of the car, the current SoC and the current gear. Moreover, the actual and the next requested velocity are used to determine the acceleration a. Then the requested torque for the current time step which shall be distributed between the engines can be obtained by using Formula (2.1) and Formula (2.2).

The split computation block checks if the car is braking at the moment, or standing at the current time step and is requested to stand still. In both cases, the split computation block returns an undefined split value of -1, and the control converter computes the torque distribution. If the requested torque value is smaller than zero, the car is deaccelerating at the moment. Then the control converter checks if it is possible to recharge the battery by recuperating brake energy. If so, braking is treated to be optimal, i. e. it is recuperated as much energy as possible to recharge the battery. The remaining torque is applied to the brakes. In the case that it is not possible to recharge the battery, the complete torque is distributed to the brakes. Moreover, if the current velocity of the car is zero as well as the next requested velocity, the car is standing at the moment and will continue to stand still. In this case, for both engines a torque value of zero is computed by the control converter.

Otherwise, if the car is neither braking nor standing at the current time step, the split computation block computes a split value $u(t) \in [0, 100]$. This is a percentage value that describes the amount of torque which is provided by the internal combustion engine. The remaining part is made available by the electrical motor [NWN+15]:

$$\begin{aligned} T_{ice}(t) &= u(t)/100 \cdot T_{req}(t) \\ T_{em}(t) &= (1 - u(t)/100) \cdot T_{req}(t) \end{aligned} \tag{2.6}$$

In the next step, the control converter checks the feasibility of this torque distribution with respect to the physical limits of the car which are given in (2.5). If they are violated, the control converter determines a new, feasible torque distribution for the car and computes the concrete amount of torque for each engine. Otherwise, the concrete amount of torque for each engine based on the split value which was computed by the split computation block is returned.

The aim of a control strategy is to determine a torque distribution in such a way that different output parameters are optimised. The optimisation criterions include for example the following aspects [PIGV01, PR07]:

- Minimisation of the fuel consumption
- Reduction of the vehicle's emissions
- Preservation of a good driving behaviour
- Management of the SoC such that a long battery life time is guaranteed.

In order to compare different control strategies regarding these aspects, *driving cycles* can be used. They represent an exemplary track layout with the characteristics of a specific region or country and can be used to compare e. g. the fuel consumption and emissions of different vehicles. The purpose is to simulate real driving conditions as realistic as possible so that the results are representable for a real drive. There exist several driving cycles for different regions and road conditions, for example the *New European Driving Cycle* (NEDC), which is designed to represent a typical use of a passenger car in Europe and which is shown in Figure 2.3, or the FTP_75 and the HWFET cycle, that represent characteristical driving condition in a US city respectively on a US highway [NWN+15].

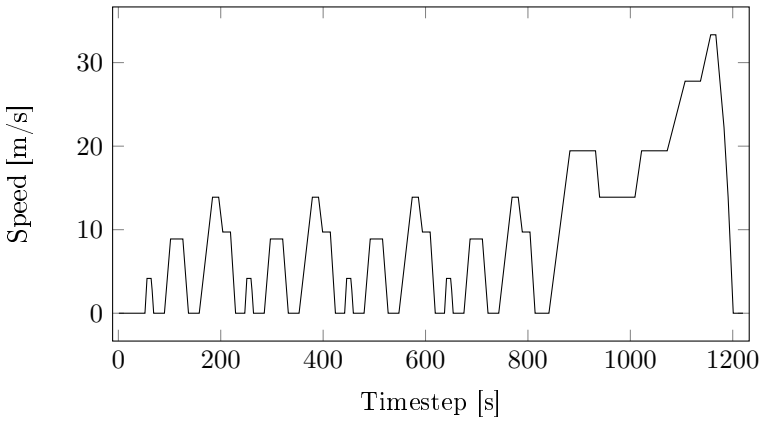

Figure 2.3: The New European Driving Cycle

There are different types of control strategies. Basically, they can be classified into two groups: Rule based and optimisation based control strategies. Rule based control strategies rely on heuristics, human intelligence, or mathematical models. The aim of an optimisation based control strategies is to compute a torque distribution that minimises a given cost function, for example the fuel consumption or the emissions. Further distinctions of control strategies refer to the aspect of prediction or real time capability: If the future driving condition is given, predictive control strategies can be used. Online control strategies must be able to compute a torque split within a limited period of time. Due to the fact that the processing power of a control device in a vehicle is limited, they have to be efficient concerning computation time and memory consumption [NWN+15, PB14]. Therefore, online control strategies have to be real time capable, which means that they have to be able to determine a new torque distribution every 0.02 seconds according to [NWN+15] .

Our grid based control strategy uses linear interpolation to compute a torque distribution. For this reason, the next section gives a short overview of linear interpolation.

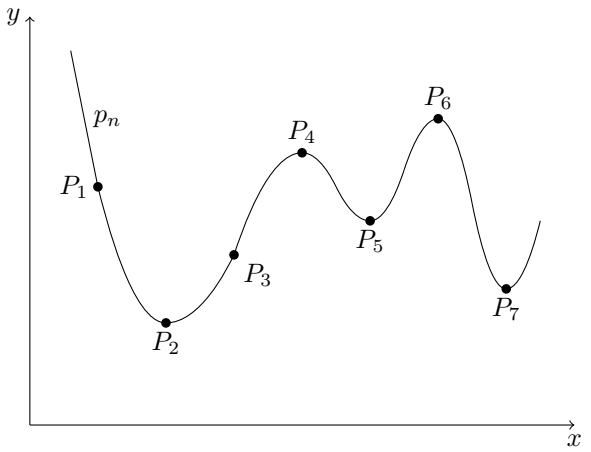

Figure 2.4: Interpolation problem for $n = 7$ points

2.3 Linear Interpolation

The grid based control strategy uses precomputed control values of existing points in a grid in order to approximate the requested control value for an input. Since we do not know the complete control function but only individual function values for each sample point in the grid, linear interpolation is used to approximate the requested control value for an input for which normally a precomputed control value is not contained in the grid. For this reason, in this section linear interpolation is introduced.

Generally, interpolation denotes the problem to find for $n + 1$ pairwise different points $P_i(x_i,y_i), i \in \{0, \ldots, n\}$ a polynomial function p_n with $\deg(p_n) \leq n$ so that the graph of this function passes all these points. This means that for all $i \in \{0, \ldots, n\}$ it is valid that $p_n(x_i) = y_i$. Figure 2.4 illustrates the interpolation problem for $n = 7$ points in a two-dimensional coordinate system. The graph p_n that passes all points $P_1, \ldots, P_7$ denotes one possible graphical solution of the interpolation problem [dtv].

Interpolation can be used to approximate requested function values of a function f of which only function values for selected sample points are given. Thereby, the function interpolating these points is evaluated for arbitrary points between the given sample points in order to approximate the function values of f at these points. *Linear interpolation* denotes the easiest way to approximate the values of a function between given points. Thereby, neighbouring sample points are connected by a straight line [Phi03]. The resulting interpolant is a piecewise linear function g. A requested function value $f(x)$ is approximated by the function value of $g(x)$. Figure 2.5 provides a graphical illustration for a two-dimensional state space: Given the function f, whose graph is represented by a gray line, and whose function values can only be accessed at the points P_{i_1}, P_{i_2} and P_{i_3}. By linear interpolating these points, we get an approximated piecewise linear function g. In order to estimate the requested function value at $x = x_{req}$, we can evaluate g at $x = x_{req}$.

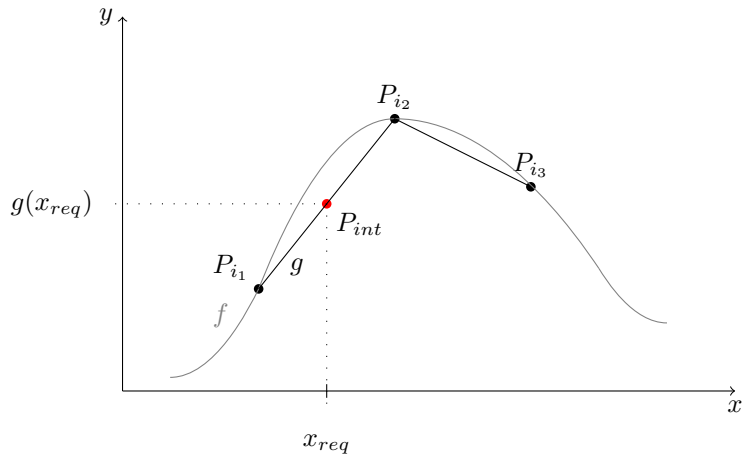

Figure 2.5: Using linear interpolation to approximate requested function values

In the case that the value of a function beyond the set of sample points is requested, extrapolation can be used to estimate this value. The easiest way to extrapolate is to assume constant values for points beyond the given observation range. *Linear extrapolation* denotes a better possibility to estimate function values outside the current observation area: The last line segment of the interpolant g is extended and we get a new function g'. Afterwards, the requested function values are approximated by the function values of this line. A graphical illustration is given in Figure 2.6: By linear extrapolating the points P_{i_2} and P_{i_3}, we get a piecewise linear function g' that is used to approximate the function value at $x = x_{req}$.

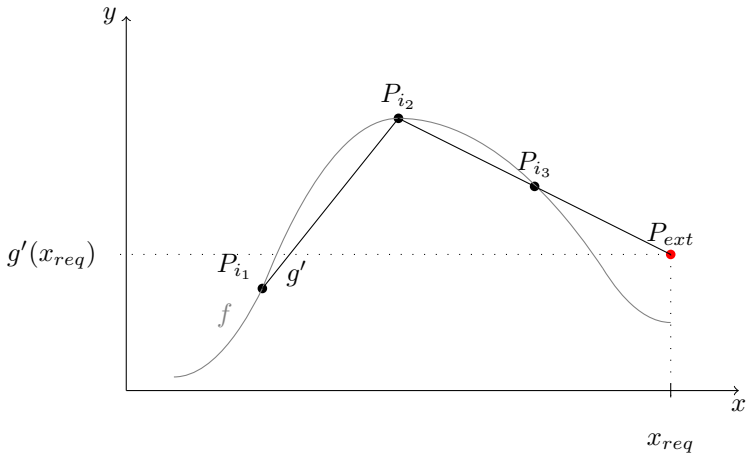

Figure 2.6: Using linear extrapolation to approximate requested function values outside the current observation range

It is obvious that there exists an error between the function to be interpolated

and the interpolating function. For linear interpolation and extrapolation, this error can be decreased by increasing the number of given sample points. In Figure 2.7, the function f is approximated by the piecewise linear functions g and h. g is constructed by linear interpolating/extrapolating the points P_{i_1}, P_{i_2} and P_{i_3}, whereas h is determined by linear interpolating/extrapolating the points $P_{i_1}, P_{i_2}, P_{i_3}, P_{j_1}, P_{j_2}, P_{j_3}$ and P_{j_4}. Since the number of given points is increased in contrast to the function g, h is able to approximate f with a smaller error than g. As a result, the estimated function values $h(x_{req,1}$ and $h(x_{req,2}$ are closer to the exact values of the function f than the function values of g at these points.

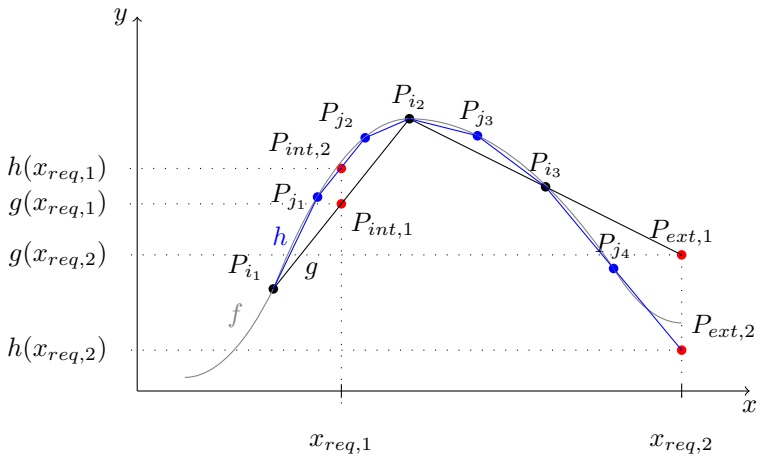

Figure 2.7: Decreasing the interpolation/extrapolation error by increasing the number of sample points

In our implementation of the grid based control strategy, the precomputation of the control values in the grid is done using a genetic algorithm. For this reason, in the next section genetic algorithms are presented.

2.4 Genetic Algorithms

We use genetic algorithms to compute the control values in the grid. Therefore, in this chapter the basic idea and the concept of these algorithms are introduced. Afterwards, we consider a concrete implementation of a genetic algorithm that is used in our implementation of the grid based control strategy to precompute the control values in the grid.

2.4.1 Theory of Genetic Algorithms

Genetic algorithms (GA) are iterative optimisation and search algorithms which are based on the biological principle of natural selection. They belong to the class of

evolutionary algorithms which are inspired by the evolution of natural creatures. The basic idea is to evolve in each iteration step a set of solution candidates by preserving and removing some of the old elements as well as coupling elements in order to get new solution candidates. The algorithm terminates as soon as a predefined stopping criterion has been reached. Afterwards, the current best solution candidate is returned.

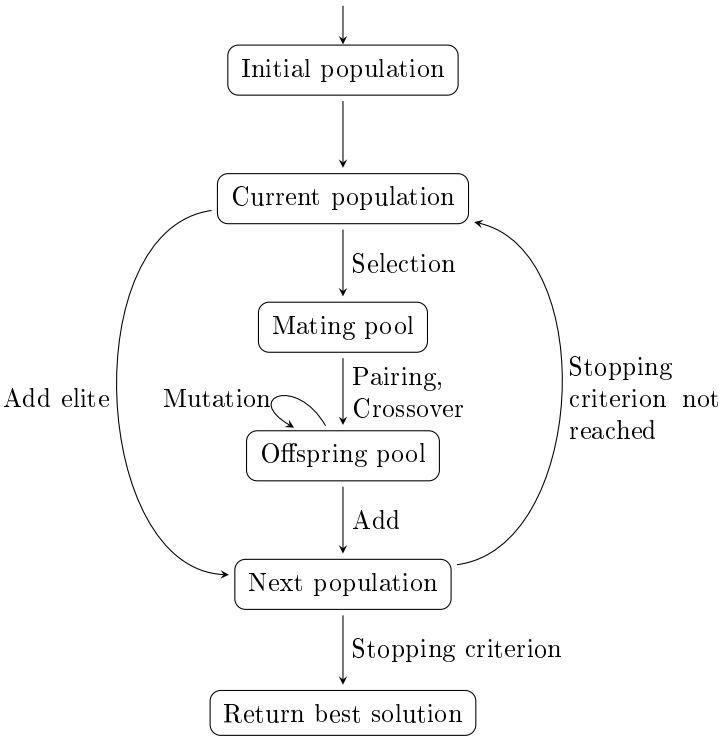

Figure 2.8: Graphical illustration of a genetic algorithm

Figure 2.8 gives an overview of the structure of a genetic algorithm. The algorithm starts with an initial set of solutions representing an encoding of the problem to be solved. A solution set is also called a *population*. According to that, the initial solution set is named the *initial population* and often, it is generated randomly. A solution candidate is also referred to as a *chromosome* and can be a single value (*gene*) or a sequence of values (*genes*). In each optimisation step, the current *generation* of a population is considered: The algorithm uses a *fitness function* to determine a value (*fitness*) for each chromosome. Afterwards, some chromosomes, usually the fittest ones with the highest fitness values are selected to create new solutions. Therefore, they are added to the *mating pool* [NWN+15, HH98].

In order to determine if the fitness value of a chromosome is good enough so that the chromosome is selected for the mating pool, a percentage value p can be used. Then, only the best p percent of the chromosomes of the current generation are selected for mating. For example, a limit of $p = 50\%$ indicates that only the best 50% of the chromosomes are selected for the mating pool. Another possibility is the usage

of a fixed threshold value. All chromosomes whose fitness value is better than a given threshold value are selected. But a fixed threshold value has the disadvantage that the number of selected chromosomes is not known in advance, and in the worst case there exist no chromosomes whose fitness values reach the threshold value [HH98].

In the next step, the chromosomes in the mating pool are paired in order to create new solutions. There exist several strategies to couple chromosomes: A very simple approach is to pair the chromosomes from top to bottom, i. e to combine every two successive chromosomes until each chromosome was selected for mating. Another possibility might be to couple the chromosomes randomly. The decision which chromosomes are paired has effect on the new generation of chromosomes. The characteristics of the future children depend on the characteristics of the corresponding parents [NWN+15, HH98].

Afterwards, the couples selected in the pairing process are used to create one or more offspring. This is called the *crossover phase*. Therefore, the characteristics of each parents are mixed or combined and a new chromosome is created. Hence the offspring contains elements of both parents. The number of offspring per parent depends on the concrete implementation, the number of overall offspring is limited by the population size. The children are put to the *offspring pool* [NWN+15, HH98].

In order to avoid that the genetic algorithm converges too fast, mutation is used to introduce new characteristics which are not yet contained in the current population. Therefore, the solutions in the offspring pool can be changed randomly with *mutation rate* μ. This phase is called the *mutation phase* [NWN+15, HH98].

Afterwards, a new generation of chromosomes is build composed of the fittest chromosomes of the last generation (the *elite*) and the chromosomes from the offspring pool. If necessary, a new set of randomly generated chromosomes is added. Afterwards, the algorithm continues by computing the fitness values for the new generation and the process repeats itself [NWN+15, HH98].

The algorithm terminates as soon as the *stopping criterion* has been reached. This could for example be a fixed number of iterations or a convergence check. Then, the algorithm returns the fittest chromosome of the last generation which is the best solution candidate computed so far [NWN+15, HH98].

Compared to other optimisation and search techniques, genetic algorithms are different within the following aspects [Gol89]:

- They do not work with the solution itself but use the encoding of a solution set instead.
- They do not search beginning from a single point, instead a set of points is used to start searching.
- They use fitness functions to evaluate the current set of solution candidates, not derivatives or other information.
- They use probabilistic transition rules instead of deterministic ones.

An advantage is the fact that the search for a solution is parallelised, i. e. they search beginning from multiple start points simultaneously in different directions. This increases the robustness of the algorithm and makes it more resistant to become entangled in suboptimal local maxima or minima: By searching parallel, the algorithm is able to turn away from a local maximum or minimum if a better solution in an other area of the search space is found [Gol89, SD08].

Furthermore, genetic algorithms provide in each iteration step a complete solution which is given by the current population. So the optimisation procedure can be stopped at any time and it is not necessary to wait for its termination in order to get a solution. The basic idea is simple and easy to understand and it is possible to use genetic algorithms for solving complex optimisation problems. Generally, genetic algorithms do not find optimal solutions, but for reasonable parameter settings they are able to provide acceptable solutions within a limited range of time. As a result, there exists a trade-off between the quality of the solution and the time needed by the algorithm.

2.4.2 GA Based Split Computation

In this section, a concrete realisation of a genetic algorithm is presented which allows to compute torque distributions for hybrid electric vehicles. The implementation was developed in [NWN+15] and is part of the extensible genetic algorithm library GeneiAL, which can be found in [gen]. We use this realisation of a genetic algorithm in order to precompute the control values in the grid in our implementation of the grid based control strategy. This genetic algorithm optimises a split sequence of length p, where p denotes the size of the prediction horizon, and returns the first value of this sequence which represents the split value for the current time step. The algorithm has the following structure [NWN+15]:

- A **chromosome** is a split sequence containing the split values $u(i), i \in \{t, t+1, ..., t+p-1\}$ for the time steps within the prediction horizon of size p. Thereby, we assume that the current time step is part of the prediction horizon, i. e. we consider the current time step t as well as the $p-1$ following time steps $t+1, \ldots, t+p-1$. Thus a chromosome in this implementation of the genetic algorithm has the structure:

$u(t)$	$u(t+1)$	$u(t+2)$	$\ldots$	$u(t+p-1)$

  The split values in this sequence are percentage values from the set $\{0, 1, \ldots, 100\}$ and denote the amount of torque to be provided by the ICE. The remaining part is produced by the EM.

- The **population** is a set of k chromosomes, the population size is fixed for all generations. Furthermore, the initial population is created randomly and thereby, smoothing is used to keep the differences between adjacent genes in a predefined range. This process preserves a good driving comfort. We use a population size of $k = 100$ in this thesis.

- The **fitness function** evaluates the quality of a chromosome and returns a fitness value in the interval $[0, 1]$. The higher the fitness value, the better the quality of the split sequence. The fitness function is a weighted sum of multiple evaluation functions including different aspects:
  - *Absolute fuel consumption*: The lower the fuel consumption, the higher the fitness value. The fitness value can be computed by the fitness function

$$e_f := 1 - \left(\frac{1}{p} \cdot \sum_{\tau=t}^{t+p-1} \dot{m}_f(\omega_{ice}(\tau), T^u_{ice}(\tau)) - \dot{m}_{f,min}\right)/(\dot{m}_{f,max} - \dot{m}_{f,min}).$$

    Thereby, $\dot{m}_{f,max}$ as well as $\dot{m}_{f,min}$ are estimated upper and lower bounds of the fuel consumption for the time step $t+p-1$. They are used to scale the average fuel consumption to the interval $[0, 1]$.
  - *Battery state of charge level* at the end of the prediction horizon: The lower the battery energy consumption, the higher the fitness value. The corresponding fitness value is calculated by using the fitness function

$$e_{sl} := (s_{t+p-1} - s_{min})/(s_{max} - s_{min}).$$

    s_{max} and s_{min} are estimated upper respectively lower bounds for the SoC at time step $t+p-1$ and are used to scale the battery state of charge level to the interval $[0, 1]$.
  - *Battery state of charge deviation* from s_{ref}: The SoC should be kept near a reference value $s_{ref} = 0.6$ to guarantee a long battery life time. The smaller the difference between the current SoC and s_{ref}, the higher the fitness value. This fitness value can be obtained by the fitness function

$$e_{sd} := \mathcal{N}_{\mu,\sigma}(s_{t+p-1})/\mathcal{N}_{\mu,\sigma}(\mu),$$

    where $\mathcal{N}_{\mu,\sigma}$ denotes the normal distributed function with expected value μ and standard deviation σ. μ and σ depend on the estimated lower respectively upper bounds for the SoC (s_{min}, s_{max}) as well as its reference value (s_{ref}). They are defined as follows:

$$(\mu,\sigma) = \begin{cases} (s_{max}, s_{max} - s_{min}/2), & s_{max} < s_{ref} \\ (s_{ref}, (s_{ref} + s_{diff})/2), & s_{min} \leq s_{ref} \leq s_{max} \\ (s_{min}, s_{max}/2), & s_{min} < s_{ref}. \end{cases}$$

    Thereby, we use $s_{diff} := max(|s_{min} - s_{ref}|, |s_{max} - s_{ref}|)$.
  - *Split difference minimisation*: For a better drivability and a higher driving comfort of the car, consecutive splits should not diverge too much. The more homogeneous a split sequence is, the higher the fitness value. The homogeneity of a chromosome can be evaluated by the fitness function

$$e_{ud} := 1 - \Big(\sum_{\tau=t}^{t+p-1} (u(\tau) - u(\tau-1))^2\Big)/(p \cdot (u_{max} - u_{min})^2).$$

    Thereby, $[u_{min}, u_{max}]$ denotes the allowed split range. Moreover, we assume that $u(0) = 0$.

– *Split difference transgression*: This evaluation function also is focused on the evaluation of the difference between consecutive splits in order to improve the drivability. If all differences between two consecutive splits within the split sequence are below a maximal allowed split delta Δu_{max}, the fitness value is maximal. The corresponding fitness function is given by

$$e_{ut} := 1 - \frac{\sum_{\tau=t}^{t+p-1} i_\tau \cdot (\Delta u_{max} - |u(\tau) - u(\tau-1)|)^2}{p \cdot (\Delta u_{max} - (u_{max} - u_{min}))^2}.$$

i_τ is defined as follows:

$$i_\tau = \begin{cases} 0, & |u(\tau) - u(\tau-1)| < \Delta u_{max} \\ 1, & else \end{cases}$$

The fitness value f can be computed by $f = w_f e_f + w_{sl} e_{sl} + w_{sd} e_{sd} + w_{ud} e_{ud} + w_{ut} e_{ut})$, where $w_f, w_{sl}, w_{sd}, w_{ud}, w_{ut} \in [0, 1]$ are weights that sum up to one. A *fitness configuration* is a tuple $(w_f, w_{sl}, w_{sd}, w_{ud}, w_{ut})$ containing the fitness weights. We use the fitness configuration $(0.45, 0.0, 0.45, 0.1, 0.0)$ in this thesis.

- **Selection** is realised by roulette wheel selection, a procedure based on the idea of spinning a roulette wheel. Thereby, the range of an imagined roulette wheel is divided into subranges so that each chromosome is related with one subrange that has a proportional size to the fitness value of the chromosome. The higher the fitness value, the larger the subrange in the roulette wheel for this chromosome, and therefore, the higher the probability that the chromosome is selected. Afterwards, a random number is determined which describes the location where the roulette wheel has stopped after spinning. The chromosome related to this location is selected [Gol89]. We choose in each iteration step 38 chromosomes for the mating pool.

- The **pairing** of the chromosomes is done randomly, i. e. chromosomes are picked at random to form pairs.

- In the **crossover** process two children per chromosome pair are created. A chromosome in the mating pool is used for crossover with a probability ρ. We use $\rho = 0.38$ in this thesis. N-point crossover is used, that means at n randomly selected points the parent's genes are swapped. We set $n = 2$. Moreover, we set the number of offspring that is generated to 38. Furthermore, smoothing is enabled, i. e. the split difference between adjacent genes of the offspring is kept within a predefined range to preserve a good drivability.

- The **mutation** in the offspring pool is done by using uniform mutation, this means m randomly selected genes of a chromosome are mutated by replacing the split value by a random value. m is defined using a percentage value of 10%. This value indicates the percent of genes to be mutated in a chromosome. Furthermore, in each iteration 10 chromosomes in the offspring pool are selected to be mutated with a mutation probability of μ. In this thesis, we use $\mu = 0.1$. Mutation is done by using smoothing, i. e. the genes in the neighbourhood of each mutated gene will be adapted so that the difference between adjacent split values is limited.

- The **new generation** is build by preserving the chromosomes with the highest fitness values from the last generation and by replacing the worst chromosomes by new ones from the offspring pool. If there is not enough offspring, randomly created chromosomes are added to the next generation. We use the $e = 5$ fittest chromosomes from the last generation and replace the 38 worst chromosomes from the last generation by the 38 chromosomes from the offspring pool. In case of double occurrences, additional chromosomes are randomly created.
- The **stopping criterion** is defined as a maximal iteration number. After a fixed number g_{max} of generations, the algorithm terminates in order to guarantee a fixed computation time. We use $g_{max} = 100$ in this thesis.

A configuration of this genetic algorithm is a tuple $C_{GA} = (k, \rho, \mu, g_{max})$ where k denotes the population size, ρ the crossover rate, μ the mutation rate and g_{max} the maximal number of generations [NWN+15]. In this thesis, we use the configuration $C_{GA} = (100, 0.38, 0.1, 100)$.

In this chapter, the vehicle model which is used to simulate control strategies as well as control strategies in general were introduced. Furthermore, linear interpolation as well as linear extrapolation were explained, and the theory of genetic algorithms and a concrete implementation of a genetic algorithm were presented. In the next chapter, this preliminary knowledge is used to introduce the grid based control strategy, a predictive control strategy which uses in each time step a discrete grid in order to compute control values.

Chapter 3

Grid Control Strategy

In this chapter, a grid based control strategy is presented. The control strategy gets as input the current state of the car as well as the requested velocities for the current time step and the prediction horizon. As output a distribution of the requested torque over the available engines is returned. This torque distribution is based on a split value which is computed by the split computation block of the strategy. The computation of this split is done by interpolating precomputed split values of sample points in a discrete grid. In this chapter, the structure of the strategy, especially the functionality of the split computation block, is introduced. Moreover, the grid construction as well as the detailed interpolation of split values is explained.

3.1 Structure of the Strategy

The grid based control strategy (*Grid Control Strategy*) is a rule based predictive control strategy which relies on a discrete grid. In each time step, the strategy gets as input the requested velocities $v_{r_1}, \ldots, v_{r_p}$ for the prediction horizon of length p and the current state of the car which is described by the the current gear g, the current battery state of charge s, the split value u_0 used in the previous time step, and the actual velocity v_a. The previous split value is considered in order to prevent high fluctuations within the torque distribution from one time step to another.

First, the strategy checks if the car is braking at the moment, or standing at the current time step and is requested to stand still. If so, an invalid split of -1 is returned and the control converter determines the torque distribution. Otherwise, the split computation block of the strategy is requested to determine a split value that proposes a torque distribution. The current state of the car as well as the requested velocities describe an input point $p_{in} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p})$ from the state space $S = D_g \times D_s \times D_u \times D_v^{p+1}$. The set S consists of $p + 4$ dimensions:

- $D_g = \{x \in \mathbb{N} | 1 \leq x \leq 5\}$ is the dimension which describes the gears of the car.

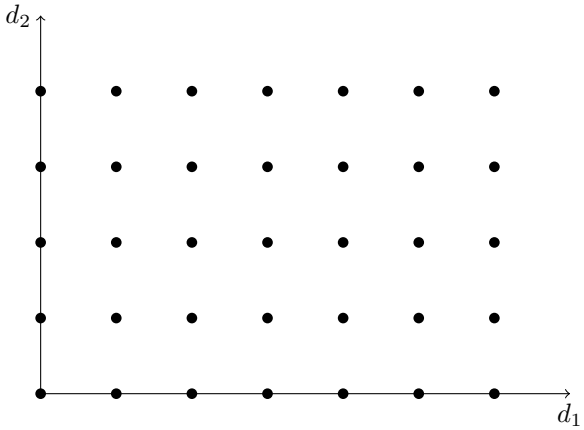

Figure 3.1: Grid for the input dimensions d_1 and d_2

- $D_s = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ denotes the dimension that describes the battery state of charge level of the car.
- $D_u = \{x \in \mathbb{R} | 0 \leq x \leq 100\}$ is the dimension which describes the set of split values.
- $D_v = \{x \in \mathbb{R} | 0 \leq x \leq 45\}$ characterises the set of velocities in m/s our vehicle can achieve.

The strategy uses a discrete grid in order to determine the requested control value for p_{in}. A grid is a set $S_{grid} = \mathcal{D}_g \times \mathcal{D}_s \times \mathcal{D}_u \times \mathcal{D}_v^{p+1} \subset S$ that contains selected discretised points $p = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$ for which a precomputed control value is stored. The points in S_{grid} are referred to as *grid points*.

For each grid point $p_{grid} \in S_{grid}$, a split value $u \in D_u$ is offline computed and can be accessed via a function $split : \mathcal{D}_g \times \mathcal{D}_s \times \mathcal{D}_u \times \mathcal{D}_v^{p+1} \rightarrow D_u$. Then, $split(p), p \in S_{grid}$ returns the split value for p_{grid}. All points $p_{grid} \in S_{grid}$ must have a split value in the range of $[0, 100]$, i. e. points $p \in S$ that describe a car which is braking at the moment, or standing at the current time step and is requested to stand still also in the next time step, and which have therefore an undefined split of -1, are not contained in S_{grid}. This is based on the fact that the split values of the grid points are used for interpolation, and using undefined split values of -1 for interpolation would lead to unreasonable and invalid approximated split values. Moreover, the cases braking and standing are handled separately by the strategy, we do not use the grid in these cases.

An optimisation method is used to precompute the split values in the grid. Since the grid is precomputed offline, the optimisation method is not required to be real time capable. We use a genetic algorithm to precompute the split values in the grid. However, the Grid Control Strategy is not restricted to this optimisation method. It is also possible to perform this precomputation process by using another optimisation algorithm.

In order to construct a set S_{grid} of grid points, the dimensions of S that have a continuous domain must be discretised, i. e. for each dimension a lower and an upper bound must be specified as well as the amount of discretisation levels. This discretisation approach is defined as follows:

Definition 3.1.1 (Discretisation). *Let D be a domain, x_{min} and x_{max} minimal and maximal values for the domain, and let $r \in \{x \in \mathbb{N} | x \geq 2\}$ be a resolution parameter. Then the interval $[x_{min}, x_{max}]$ can be homogeneously discretised to $\mathcal{S}_{dis} = \{x_i | x_i = x_{min} + i \cdot \frac{x_{max} - x_{min}}{r-1}, i \in [0, r-1]\}$.*

Figure 3.1 provides an example illustrating the structure of a grid for two input dimensions d_1 and d_2 with domains D_1 and D_2, respectively. The discretisation of the state space $D_1 \times D_2$ allows to specify grid points $p_{grid} \in S_{grid}$ which are represented by black dots in this space. For each of these grid points, a split value is precomputed and can be accessed by the split function.

The set S_{grid} of grid points $p_{grid} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p})$ is a subset of the state space S. The elements contained in this set have the following characteristics:

- Since we consider a car with five gears, the first dimension of the grid is $D_g = \{1, 2, \ldots, 5\}$. As the gear values are already discretised, we consider all five gears in our grid files, i. e. $\mathcal{S}^g_{dis} = D_g$. Gear shifts are not considered within the prediction horizon, i. e. we keep the requested gear for the current time step fixed. This has the advantage that each point $p \in S$ has only a single gear value instead of using a gear for each time step that is part of the prediction horizon. In doing so, the number of dimensions in the state space S and therefore, in S_{grid}, is reduced.
- The battery state of charge (SoC) theoretically can have values in the interval $[0, 1]$, where 0 denotes that the battery is discharged, and 1 describes a battery which is fully charged. However, in practice the Grid Control Strategy should keep this value in the range of $[0.5, 0.7]$ near a reference value $s_{ref} = 0.6$ to guarantee a long battery lifetime. For the grid, a minimal and a maximal SoC value $s_{min}, s_{max} \in D_s$ with $s_{min} < s_{max}$ are defined. Thereby, s_{min} is set to 0.5 and s_{max} to 0.7 since the control converter allows only small deviations from the interval $[0.5, 0.7]$. The discretised set of SoC values $\mathcal{S}^{s,g}_{dis}$ is constructed using s_{min} and s_{max} as well as for each gear g a resolution parameter r^g_s, whose concrete value can be specified.
- The velocity can vary between 0 m/s and 45 m/s, i. e. we have $D_v = \{x \in \mathbb{R} | 0 \leq x \leq 45\}$. The physical unit is m/s. The current gear restricts the velocity that can be obtained by the car. In Table 3.1 the minimal and the maximal velocity value for each gear g are defined. The actual velocity v_a as well as the requested velocities $v_{r_1}, \ldots, v_{r_p}$ can be discretised by using the minimal and maximal values in this table. So for each gear g exist minimal and maximal velocities $v^g_{a,min}, v^g_{a,max}, v^g_{r,min}, v^g_{r,max}$ which describe the corresponding minimal and maximal velocity. Furthermore, there exist resolution parameters $r^g_{v_a}, r^g_{v_r}$, whose concrete values can be defined. These values allow to discretise the intervals $[v^g_{a,min}, v^g_{a,max}]$, $[v^g_{r,min}, v^g_{r,max}]$ in order to construct the discretised sets of velocities $\mathcal{S}^{v_a,g}_{dis}$ and $\mathcal{S}^{v_r,g}_{dis}$.

$[v]$	v_{min}	v_{max}	g
	0.00	8.3056	1
	4.73849	11.083	2
m/s	7.486	18.022	3
	15.7969	20	4
	18.0102	45.00	5

Table 3.1: Minimal and maximal velocity depending on the current gear

- The split values $u \in D_u$ as well as the previous split values $u_0 \in D_u$ are specified as values in the range of $[0, 100]$ and denote the percentage of torque to be provided by the combustion engine. The remaining percentage $100 - u$ of torque is produced by the electrical motor. The previous split of the last time step is considered as input in order to prevent high fluctuations from one time step to the next one. Corresponding to this, the grid has a dimension which contains information about the previous split. The previous split value of a grid point is homogeneously discretised using $u_{min} = 0$ and $u_{max} = 100$. The discretisation of the previous split values contained in the grid points is done using the resolution parameter $r^g_{u_0}$, whose concrete value can be specified for each gear g. In doing so, we obtain the discretised sets $S^{u_0,\,g}_{dis}$ of previous split values. In addition, it should be noted that the computed split value u is not discretised.

The Grid Control Strategy uses the grid and the precomputed split values of the grid points in order to determine the requested split value of an input point p_{in}. The complete algorithm of the strategy is given in Algorithm 1. If p_{in} is a grid point, its split value can be accessed by the split function and is returned immediately. But in general, this is not the case and an input point is not contained in the grid. Then, an interpolation set $S^{p_{in}}_{intp}$ is computed which contains grid points in the neighbourhood of p_{in}. Afterwards, the requested split of p_{in} is interpolated by the split values of the points that are contained in $S^{p_{in}}_{intp}$.

The concrete computation of the interpolation set is described in the next section. Moreover, the interpolation of the requested split value of the input point based on this interpolation set is explained.

3.2 Interpolating Split Values

First, we consider the function $computeIntpolSet(S_{grid}, p_{in})$. This function has the purpose to compute an interpolation set $S^{p_{in}}_{intp}$ of neighbouring grid points of p_{in} which are suitable for interpolation. In a first approach, we defined this set by choosing a fixed number of the $k < |S_{grid}|$ grid points which are closest to the input point. However, this approach has the disadvantage that the distances between the input point and the selected grid points can become very large in the case that the density of grid points in the environment of the input point is small. Moreover, it is not

Input : $S_{grid} \subset S, p_{in} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$
Output: split $u \in D_u \cup \{-1\}$
function *computeSplit*(S_{grid}, p_{in})
  if *braking or standing* **then**
    return -1;
  else
    if $p_{in} \in S_{grid}$ **then**
      return *split*(p_{in});
    else
      $S_{intp}^{p_{in}} =$ *computeIntpolSet*(S_{grid}, p_{in});
      return *interpolation*$(S_{intp}^{p_{in}}, p_{in})$;
    end
  end
end

Algorithm 1: Basic algorithm of the Grid Control Strategy

ensured that within each dimension neighbouring grid points are contained in the interpolation set.

For these reasons, we decided to describe the interpolation set using a box around the input point. This box is defined as follows:

Definition 3.2.1 (Box of an input point)**.** *Given an input point* $p_{in} = (g^{in}, s^{in}, u_0^{in}, v_a^{in}, v_{r_1}^{in}, \ldots, v_{r_p}^{in}) \in S$*. The box of* p_{in} *describes the area around* p_{in} *whose expansion is limited in each dimension* $d \in \{g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}\}$ *by threshold values* $t_g, t_s, t_{u_0}, t_{v_a}, t_{v_r} \in \mathbb{R}^{\geq 0}$*. These threshold values are defined as follows:*

- $t_g = 0$,
- $t_s = \begin{cases} \frac{s_{max} - s_{min}}{r_s^g - 1}, & s \notin \mathcal{S}_{dis}^{s,g} \\ 0, & else \end{cases}$,
- $t_{u_0} = \begin{cases} \frac{u_{max} - u_{min}}{r_{u_0}^g - 1}, & u_0 \notin \mathcal{S}_{dis}^{u_0,g} \\ 0, & else \end{cases}$,
- $t_{v_a} = \begin{cases} \frac{v_{a,max} - v_{a,min}}{r_{v_a}^g - 1}, & v_a \notin \mathcal{S}_{dis}^{v_a,g} \\ 0, & else \end{cases}$,
- $t_{v_r} = \begin{cases} \frac{v_{r,max} - v_{r,min}}{r_{v_r}^g - 1}, & v_r \notin \mathcal{S}_{dis}^{v_r,g} \\ 0, & else \end{cases}$,

and they are at most as large as the difference between two discretised values of the input dimensions in the grid.

The interpolation set $S_{intp}^{p_{in}}$ contains all points within the box around p_{in} for which a precomputed control value is stored. Points located exactly on the borders of the

box are not contained in the interpolation set. If not equal to zero, the threshold values $t_s, t_{u_0}, t_{v_a}, t_{v_r}$ indicate for each dimension a supremum for the maximal deviation a point is allowed to have from the input point. In the special case that a dimension of p_{in} is contained in the grid, the corresponding threshold value is set to zero. As a result, the box is reduced by one dimension. Moreover, we assume that only grid points p_{grid} with the same gear g as the input point are used for interpolation. Therefore, t_g is set to zero and as a result, all grid points contained in $S_{intp}^{p_{in}}$ have the same gear value as p_{in}. In the case that an input point is located outside the grid, we determine additional extrapolated points which represent the missing neighbouring grid points of p_{in}. These extrapolated points are added to the interpolation set of the input point. For each extrapolated point, a fictive split value is computed and stored by linear extrapolating the split values of the two last known grid points. The complete definition of the interpolation set is given in the following:

Definition 3.2.2 (Interpolation Set)**.** *Given an input point* $p_{in} = (g^{in}, s^{in}, u_0^{in}, v_a^{in}, v_{r_1}^{in}, \ldots, v_{r_p}^{in}) \in S, p_{in} \notin S_{grid}$ *as well as the box around this input point, which is limited by the threshold values* $t_g, t_s, t_{u_0}, t_{v_a}, t_{v_r} \in \mathbb{R}^{\geq 0}$*. The interpolation set* $S_{intp}^{p_{in}}$ *contains all points* $p = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p})$ *for which a precomputed split value is stored and which are located in the box of* p_{in}*. Thus for all* $p \in S_{intp}^{p_{in}}$ *the following conditions are fulfilled:*

1. $g^{in} = g$, *i. e. the gear is fixed and is not interpolated,*
2. $t_s > 0 \implies (|s^{in} - s| < t_s)$,
3. $t_s = 0 \implies (|s^{in} - s| = 0)$,
4. $t_{u_0} > 0 \implies (|u_0^{in} - u_0| < t_{u_0})$,
5. $t_{u_0} = 0 \implies (|u_0^{in} - u_0| = 0)$,
6. $t_{v_a} > 0 \implies (|v_a^{in} - v_a| < t_{v_a})$,
7. $t_{v_a} = 0 \implies (|v_a^{in} - v_a| = 0)$,
8. $t_{v_r} > 0 \implies (\forall i \in \{1, \ldots, p\} : |v_{r_i}^{in} - v_{r_i}| < t_{v_r})$,
9. $t_{v_r} = 0 \implies (\forall i \in \{1, \ldots, p\} : |v_{r_i}^{in} - v_{r_i}| = 0)$.

This interpolation set consists of all points in the neighbourhood of p_{in} that are suitable for interpolation. The complete algorithm to compute the interpolation set is given in Algorithm 2.

Figure 3.2 provides an illustration for a two-dimensional grid that summarises the different cases which could occur during the computation of the interpolation set. The threshold values for the input dimensions d_1 and d_2 are t_1 respectively t_2. In a first situation, an input point $p_{in,1}$ is located between given grid points. The box of $p_{in,1}$ is limited by threshold values t_1 and t_2. Therefore, the interpolation set $S_{intp}^{p_{in,1}}$ contains four grid points, marked by blue points in the figure. Afterwards, we consider the special case that at least one dimension of an input point is exactly contained in the grid. In this case, the box of the input point $p_{in,2}$ is reduced to a straight line and the

Input : $S_{grid} \subset S, p_{in} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$
Output: Interpolation set $S_{intp}^{p_{in}}$ of p_{in}
function *computeIntpolSet*(S_{grid}, p_{in})
  $S_{intp}^{p_{in}} := \emptyset$;
  for all $p \in S_{grid}$ **do**
    if $p \in box(p_{in})$ **then**
      $S_{intp}^{p_{in}} := S_{intp}^{p_{in}} \cup \{p\}$;
    end
  end
  if *outsideGrid*(p_{in}) **then**
    $S_{intp}^{p_{in}} := S_{intp}^{p_{in}} \cup getExtrapolPoints(S_{grid}, p_{in})$;
  end
  return $S_{intp}^{p_{in}}$;
end

Algorithm 2: Computation of the interpolation set

interpolation set of $p_{in,2}$ contains only two grid points. In a third case, the input point $p_{in,3}$ is located outside the grid. In order to interpolate its split value, we construct additional extrapolated points e_1 and e_2 that represent the missing neighbouring grid points. For each of these extrapolated points, a fictive split value is computed by linear extrapolating the split values of the two last known grid points. Thereby, it should be noted that these extrapolated split values are not necessarily located in the interval $[0, 100]$. As a result, the interpolation set of $p_{in,3}$ contains two grid points and two additional extrapolated points.

After determining the interpolation set of an input point p_{in}, we consider the function $interpolation(S_{intp}^{p_{in}}, p_{in})$. The purpose of this function is to interpolate the requested split value of p_{in} by using the split values of the points contained in the interpolation set.

The requested split value of p_{in} is computed by linear interpolating the split values of the $n = |S_{intp}^{p_{in}}|$ many neighbouring points of p_{in} which are contained in the interpolation set. In order to access not only the split values of the grid points, but also the split values of possible additional extrapolated points, we use an adapted split function $split'(p) : \mathcal{D}_g \times \mathbb{R}^{p+3} \to \mathbb{R}$. This function provides a precomputed split value for each point $p \in S_{grid}$ as well as for each extrapolated point. Afterwards, the requested split value of p_{in} is computed as follows:

$$u_{p_{in}} = \frac{\sum_{i=1}^{n} w_i \cdot split'(p_i)}{\sum_{i=1}^{n} w_i}. \tag{3.1}$$

The weight w_i for each point p_i includes the distance between the input point and the respective grid point/extrapolation point. This distance is based on the euclidean distance and is defined as follows:

Definition 3.2.3 (Distance function)**.** *Given two points* $p_1 = (g_1, s_1, u_{0,1}, v_{a,1}, v_{r_1,1}, \ldots, v_{r_p,1})$, $p_2 = (g_2, s_2, u_{0,2}, v_{a,2}, v_{r_1,2}, \ldots, v_{r_p,2}) \in S \cup \mathcal{D}_g \times \mathbb{R}^{p+3}$ *with* $g_1 = g_2 = g$.

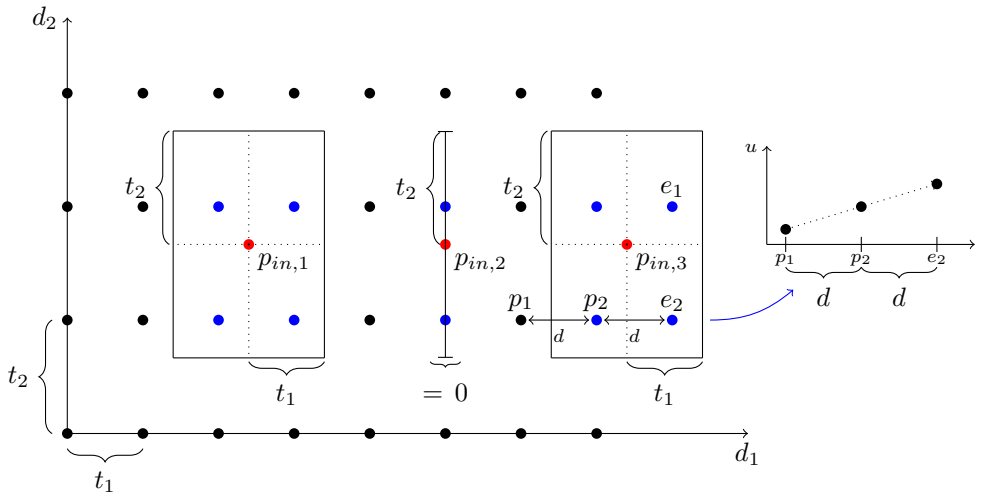

Figure 3.2: Determining the interpolation set

The distance function $dist(p_1, p_2)$ is defined as follows:

$$dist(p_1,p_2) = \Bigg(\Big(\frac{|s_1 - s_2|}{s^g_{max} - s^g_{min}}\Big)^2 + \Big(\frac{|u_1 - u_2|}{u^g_{max} - u^g_{min}}\Big)^2 + \Big(\frac{|v_{a,1} - v_{a,2}|}{v^g_{a,max} - v^g_{a,min}}\Big)^2 + \sum_{i=1}^{p} \Big(\frac{|v_{r_i,1} - v_{r_i,2}|}{v^g_{r,max} - v^g_{r,min}}\Big)^2 \Bigg)^{\frac{1}{2}}.$$

Due to the fact that different dimensions of a point have different domains, the differences are normalised by dividing by their maximal ranges. Thereby, we obtain values within $[0, 1]$ for each dimension. A small distance should result in a high value, for this reason w_i is defined as follows:

$$w_i = 1.0 - \frac{dist(p_{in}, p_i)}{\max_{j=1,\dots,n} dist(p_{in}, p_j)} \tag{3.2}$$

The weights can have values in the interval $[0, 1)$. Furthermore, the point in the interpolation set with the highest distance to the input point has a weight of zero and therefore, it is not influencing the split value of the input point. The complete interpolation algorithm is shown in Algorithm 3.

In this chapter, the structure of the Grid Control Strategy was introduced. The state space $S = \mathcal{D}_g \times \mathcal{D}_s \times \mathcal{D}_u \times \mathcal{D}_v^{p+1}$ describes the set of inputs of the strategy. The set $S_{grid} \subset S$ of grid points contains selected discretised points $p \in S$ for which a precomputed split value can be accessed by the split function $split(p), p \in S_{grid}$. In order to construct such a grid set, the dimensions of S which have a continuous domain must be discretised. The Grid Control Strategy uses the grid in order to interpolate split values for an input point $p_{in} \in S, p_{in} \notin S_{grid}$. Therefore, an interpolation set $S^{p_{in}}_{intp}$ is determined that contains points which are suitable for interpolation. This

Input : $S_{intp}^{p_{in}}, p_{in} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$
Output: split $u \in D_u$
function *interpolation*$(S_{intp}^{p_{in}}, p_{in})$
  $u := 0$;
  $sumWeights := 0$;
  for all $p \in S_{intp}^{p_{in}}$ **do**
    $w_i := 1.0 - (dist(p_{in}, p)/\max_{j=1}^{n} dist(p_{in}, p_j))$;
    $u+ = w_i \cdot split'(p)$;
    $sumWeights+ = w_i$;
  end
  ▷ Limit split to $[0, 100]$
  if $split/sumWeights < 0$ **then**
    return 0;
  else if $split/sumWeights > 100$ **then**
    return 100;
  else
    return $split/sumWeights$;
  end
end

Algorithm 3: Interpolation algorithm

interpolation set is defined by all points contained in a box around the input point. In the case that an input point is located outside the grid, linear extrapolation is used to compute split values for additional extrapolated points. Afterwards, the split values of the points contained in the interpolation set are used to interpolate the split value of p_{in}. Thereby, the split values of the points $p \in S_{intp}^{p_{in}}$ are weighted by their distance to the input point. The quality of the interpolated split value depends on the available grid points. In the next chapter, we consider a procedure to optimise a grid in order to improve the quality of the interpolated split values.

Chapter 4

Grid Optimisation

The Grid Control Strategy uses the precomputed split values of the points that are contained in the interpolation set $S_{intp}^{p_{in}}$ of an input point $p_{in} \in S$ in order to interpolate the split value for p_{in}. Thereby, the quality of the interpolated split value depends on the available grid points in the grid and their distance to p_{in}: The larger the distance, the worse a point represents the input point and therefore, the less influence the point has on the interpolation result. Moreover, the fewer the number of points in the interpolation set of an input point, the less significant the approximated split value is.

In this chapter, a procedure to optimise the grid is presented. By inserting additional grid points, the aim is to refine the grid and to increase its resolution, such that afterwards the grid is able to interpolate requested split values more precisely than before. Thereby, we assume that the smaller the distance between p_{in} and a point in the grid, the more influence the precomputed split value should have on the interpolation result. Furthermore, we assume that the higher the number of points in the interpolation set of an input point, the more precise the interpolated split value becomes. In the first section, the basic algorithm of the optimisation procedure is presented. Afterwards, the detailed construction of new grid points is explained.

4.1 Structure of the Algorithm

The optimisation procedure is an iterative approach which refines the grid in areas where the optimisation algorithm that is used to precompute the split values in the grid is poorly approximated. Thereby, new grid points are inserted into the grid so that split values of input points can be approximated more precisely. The aim is to increase the density of grid points in the environment of an input point p_{in} in order to improve the quality of the split interpolation of p_{in}. In contrast to increasing the resolution parameters during the grid creation, the optimisation procedure refines the grid only partially. As a result, the optimised grid is smaller than a grid with increased resolution parameters. Moreover, the optimised grid is no longer homogeneous. Figure

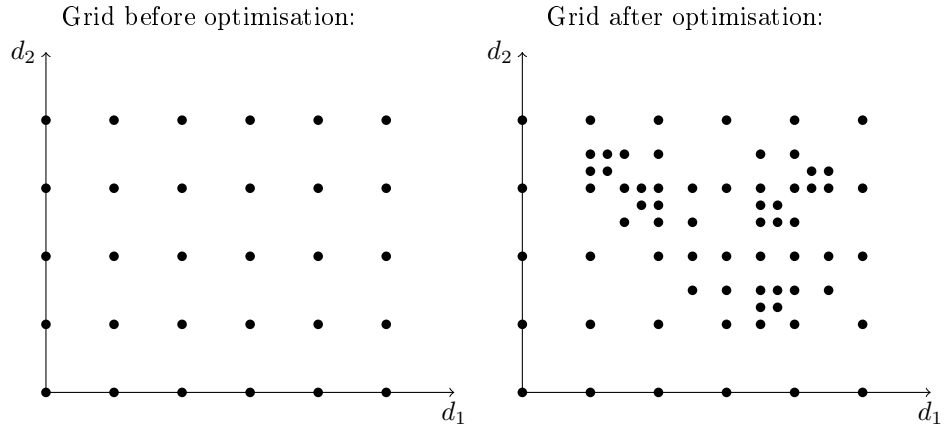

Figure 4.1: A two-dimensional homogeneous grid (left), and a two-dimensional inhomogeneous, optimised grid (right)

4.1 illustrates the difference between a homogeneous grid as considered so far, and a grid after optimisation.

The complete algorithm of the grid optimisation procedure is presented in Algorithm 4. The algorithm gets as input a grid S_{grid} which shall be optimised. Moreover, the Grid Control Strategy *gridStrat* as well as a reference strategy $refStrat$ are required. As output an optimised grid is returned. In each iteration, the split values of the Grid Control Strategy based on the current grid are compared to the split values of a reference strategy. In order to get comparable results, both strategies must be evaluated on the same car that is simulated on a driving cycle. Moreover, as reference strategy a control strategy which is directly based on the optimisation procedure that is used to precompute the split values in the grid is chosen.

Afterwards, the function *getMaxSplitDiff*$(refStrat, gridStrat, S_{ignore})$ determines the maximal split difference *maxSplitDiff* between the reference strategy $refStrat$ and the Grid Control Strategy *gridStrat*, whereby all time steps where the corresponding input point is contained in S_{ignore} are ignored. The set S_{ignore} is initially empty and contains all input points that are marked as 'ignored' during the optimisation. Further information about ignored input points are given below. In the case that the maximal split difference exceeds a predefined value δ and not all input points are ignored, the optimisation procedure determines the time step t_{max} corresponding to *maxSplitDiff* as well as the input point p_{max} of the Grid Control Strategy at this time step. Afterwards, the algorithm tries to insert new points into the grid. The function $insertNewPoints(S_{grid}, p_{max})$ is explained in detail in Section 4.2. Thus, additional grid points are used for interpolation which might decrease the split difference at this time step between the Grid Control Strategy and the reference strategy. After that, it is checked if the split difference at time step t_{max} is decreased. If so, the optimisation was successful. Otherwise, the corresponding input point is added to S_{ignore}. As a consequence, this input point will not be considered by the optimisation algorithm any more. This approach is necessary in order to guarantee the termination of the

Input : $S_{grid} \subset S, refStrat, gridStrat$
Output: refined Grid $S_{optGrid} \subset S$
function $optimiseGrid(S_{grid}, refStrat, gridStrat)$
  $S_{optGrid} := S_{grid}$;
  $S_{ignore} := \emptyset$;
  $maxSplitDiff = getMaxSplitDiff(refStrat, gridStrat, S_{ignore})$;
  while $((maxSplitDiff > \delta)$ && $(!containsAllInputPoints(S_{ignore}))$ **do**
    $t_{max} = getTimeStep(maxSplitDiff)$;
    $p_{max} = getInputPoint(t_{max})$;
    $S_{optGrid} := insertNewPoints(S_{optGrid}, p_{max})$;
    $d = getSplitDiff(p_{max})$;
    if $d \geq maxSplitDiff$ **then**
      $deleteInsertedPoints()$;
      $S_{ignore} := S_{ignore} \cup p_{max}$;
    end
    $gridStrat = runGridStrat(S_{optGrid})$;
    $maxSplitDiff = getMaxSplitDiff(refStrat, gridStrat, S_{ignore})$;
  end
  return $S_{optGrid}$;
end

Algorithm 4: Grid optimisation algorithm

procedure. Since inserting new grid points might have effects on the split values of other input points in the neighbourhood, it is necessary to rerun the Grid Control Strategy after each insertion process. In doing so, it is guaranteed that in each optimisation step the maximal split difference (without the time steps which are ignored) between the strategies is considered. Afterwards, the algorithm continues by recomputing the maximal split difference and the process is repeated. In the case that *maxSplitDiff* is below δ, or all input points of the Grid Control Strategy are marked as 'ignored', the algorithm terminates. However, as the optimisation procedure provides in each iteration step a complete grid, it is also possible to stop the optimisation before its termination.

4.2 Construction of New Grid Points

In this section, we consider the function $insertNewPoints(S_{grid}, p_{max})$, which has the purpose to add new points to the grid that are contained in the box around an input point. The function gets as input a grid $S_{grid} \in S$ as well as an input point $p_{max} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$. As output a refined grid is returned where the density of grid points in the environment of p_{max} is increased. For a two-dimensional grid, the approach of this function is illustrated in Figure 4.2. The input point is represented by a red point, grid points are marked as grey dots. The grey rectangular area defines the box around the input point which describes the interpolation set $S_{intp}^{p_{max}}$. The complete algorithm of the construction of new grid points is presented in Algorithm 5.

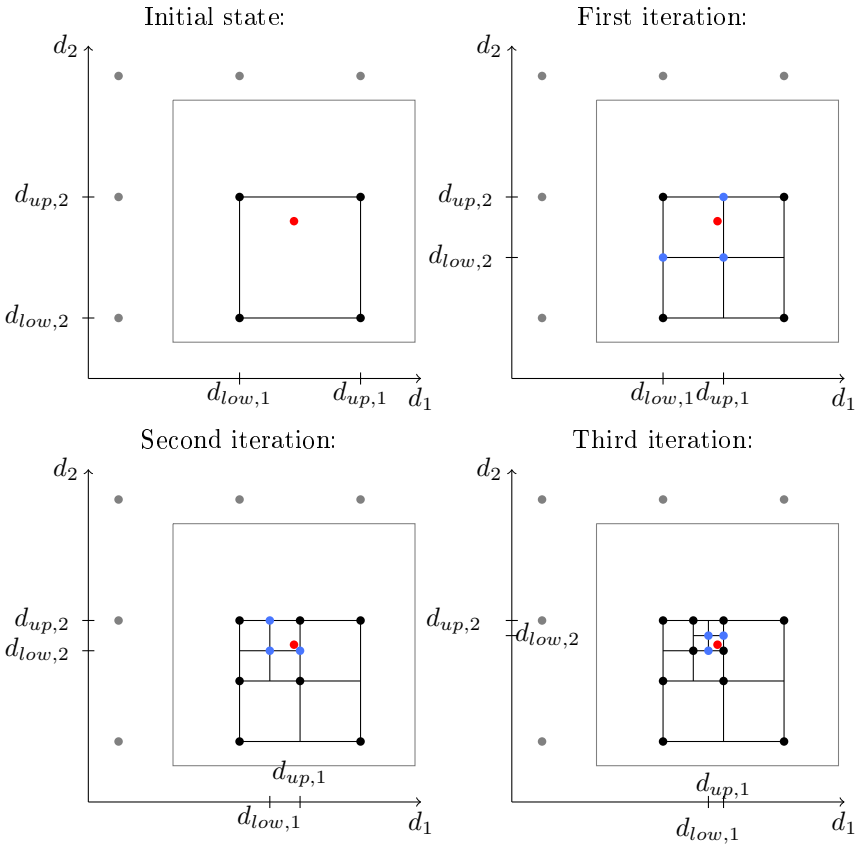

Figure 4.2: Optimisation procedure: First, for each dimension upper and lower bounds d_{low} and d_{up} are determined. Afterwards, the intervals $[d_{low}, d_{up}]$ are divided into two equidistant sub intervals $[d_{low}, m]$ and $[m, d_{up}]$, and in each dimension only the sub intervals that contain the input point are further considered. New points located on the edges of the intervals are added to the grid. This procedure can be repeated until a predefined iteration depth is reached.

The optimisation procedure allows to refine the dimensions s, v_a, v_r and u_0. Since all gears $g \in D_g$ are already contained in the grid, the gears are not refined. The algorithm starts by determining for each dimension d the next smaller as well as the next larger values:

Definition 4.2.1. *Let $S_{grid} \subset S$ be a grid and let $p_{max} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$ be an input point for which the grid should be refined. Moreover, the discretisation sets $\mathcal{S}_{dis}^{s,g}, \mathcal{S}_{dis}^{u_0,g}, \mathcal{S}_{dis}^{v_a,g}$ and $\mathcal{S}_{dis}^{v_r,g}$ are given. The next smaller as well as the next larger values d_{low} and d_{up} for each dimension $d \in \{s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}\}$ are defined as follows:*

- $d_{low} := e \in S_{dis}^{d,g}, e \leq d_{p_{max}} \wedge \neg\exists f \in S_{dis}^{d,g} : (e < f < d_{p_{max}})$
- $d_{up} := e \in S_{dis}^{d,g}, e \geq d_{p_{max}} \wedge \neg\exists f \in S_{dis}^{d,g} : (d_{p_{max}} < f < e)$

Input : $S_{grid} \subset S, p_{max} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}) \in S$
Output: refined Grid $S_{optGrid} \subset S$
function $insertNewPoints(S_{grid}, p_{max})$
  $S_{optGrid} := S_{grid}$;
  for all $d \in \{s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}\}$ **do**
    $d_{low} := nextLowerValue(d)$;
    $d_{up} := nextUpperValue(d)$;
  end
  for $i = 0; i < i_{max}; ++i$ **do**
    for all $d \in \{s, u_0, v_a, v_{r_1}, \ldots, v_{r_p}\}$ **do**
      $m := d_{low} + \frac{d_{up} - d_{low}}{2}$;
      if $m \leq d_{p_{max}}$ **then**
        $d_{low} := m$;
      end
      if $m \geq d_{p_{max}}$ **then**
        $d_{up} := m$;
      end
    end
    $S_{newPoints} := \{s_{low}, s_{up}\} \times \{v_{a,low}, v_{a,up}\} \times \{v_{r,low}, v_{r,up}\}^p \times \{u_{0,low}, u_{0,up}\}$;
    for all $p \in S_{newPoints}$ **do**
      $u = computeSplit(p)$;
      if $((!contained(p, S_{optGrid})) \;\&\&\; (u \neq -1))$ **then**
        $S_{optGrid} := S_{optGrid} \cup \{p\}$;
        $split(p) := u$;
      end
    end
  end
  return $S_{optGrid}$;
end

Algorithm 5: Construction of new grid points

The following example shows the computation of d_{low} and d_{up} exemplary for the SoC:

Example 4.2.1. *Let* $S_{dis}^{s,g} = \{0.5, 0.55, 0.6, 0.65, 0.7\}$ *and let* $p_{max} = (g, s, u_0, v_a, v_{r_1}, \ldots, v_{r_p})$. *The upper and lower SoC value for* $s = 0.59$ *are* $s_{low} = 0.55$ *and* $s_{up} = 0.6$.

The algorithm to determine additional grid points is based on the idea of a binary search: After defining the start interval $[d_{low}, d_{up}]$ for each dimension d, in each iteration step these intervals are divided into two equidistant sub intervals: $[d_{low}, m], [m, d_{up}]$. In the case that $d_{p_{max}} \in [d_{low}, m]$, we set $d_{up} := m$; Otherwise, if $d_{p_{max}} \in [m, d_{up}]$, we set $d_{low} := m$. Afterwards, the cartesian product of the sets $\{d_{low}, d_{up}\}$ is constructed, which gives the set of points $S_{newPoints}$ that contains candidates for new grid points. These new points are highlighted by blue dots in Figure 4.2. Furthermore, a split value $u \in D_u$ is computed for each new point. After that, all points $p \in S_{newPoints}$ whose split value is defined, i. e. a value in the interval $[0, 100]$, and which are not yet contained in the grid, are added to the grid. Thereby, we extend the *split* function that returns a precomputed split value for each grid

point such that this function is able to return also the split values for the new grid points. Due to the selection of the initial upper and lower values, all new grid points are located in the box around the current input point. As a result, only grid points that influence the split value of the current input point are added to the grid.

In the next step, the split difference between the Grid Control Strategy and the reference strategy at this time step is evaluated. In the case that the insertion of the new grid points has decreased the split difference, the optimisation of this time step is finished for the moment. However, the corresponding input point might be considered again in a further optimisation step if its split difference is detected to be maximal compared to all other available input points that are not yet marked as 'ignored'. In this case, the optimisation of this input point will be continued at its last used iteration depth. Otherwise, if the split difference is not decreased by the new grid points, the inserting process is repeated by further dividing the current intervals $[d_{low}, d_{up}]$. In order to guarantee the termination of the procedure, a maximal iteration depth i_{max} is specified. If this limit has been reached, the corresponding input point is marked as 'ignored', and its optimisation is definitely finished.

Since the inserted grid points converge towards the input point with increasing iteration depth, the insertion of new grid points results in a smoothed refinement of the grid. As a result, the density of grid points increases the closer we get to an input point to be optimised. Additionally, this optimisation method only adds grid points close to the input point, and not in the whole state space. In doing so, the performance of the procedure is improved.

In this chapter, a procedure to optimise the grid was presented. The aim is to increase the quality of the interpolated split values by adding new points to the grid. The optimisation algorithm is an iterative approach, which compares the split values of the Grid Control Strategy to those of a reference strategy. For this purpose, a driving cycle is used. In each optimisation step, the input point p_{max} of the Grid Control Strategy corresponding to the time step where the differences between the split values of the two strategies is maximal is determined. Thereby, time steps where the corresponding input point is marked as 'ignored' are skipped. Afterwards, the function $insertNewPoints(S_{grid}, p_{max})$ tries to add new points that are located in the environment of p_{max} to the grid. This inserting process is based on the idea of a binary search: It starts with lower and upper values d_{low}, d_{up} for each dimension d. Afterwards, the intervals $[d_{low}, d_{up}]$ are divided into two equidistant sub intervals, and for each dimension the upper and lower values are set to the limits of the sub interval that contains the dimension d of the input point. In order to construct a set $S_{newPoints}$ of new grid point candidates, the cartesian product of the sets $\{d_{low}, d_{up}\}$ is determined. After that, all points $p \in S_{newPoints}$ whose split values are defined and which are not yet contained in the grid, are added to the grid. If the split difference for the current input point is decreased by the new grid points, its optimisation is finished for the moment. Otherwise, the current intervals $[d_{low}, d_{up}]$ are further divided until a predefined maximal iteration depth is reached. We implemented the Grid Control Strategy as well as this optimisation procedure. In the next chapter, the experimental results of our implementation are evaluated.

Chapter 5

Experimental Results

After introducing the theoretical concepts of the Grid Control Strategy, the grid, and the grid optimisation procedure, in this chapter the experimental results of our implementation are discussed. We use a MATLAB/Simulink model in order to test the Grid Control Strategy and to compare it to other control strategies. For this, three driving cycles are used: The *NEDC*, the *FTP_75* and the *HWFET*. A detailed description of each driving cycle can be found in Appendix A. Moreover, we compare the Grid Control Strategy to the following reference strategies: The *GA Control Strategy* is a predictive control strategy which is based on optimal control and which uses in each time step a genetic algorithm to determine the split value. Further information on this control strategy can be found in [NWN+15]. The *ICE Control Strategy* uses only the combustion engine for driving.

g	s_{min}	s_{max}	r_s^g	$u_{0,\,min}$	$u_{0,\,max}$	$r_{u_0}^g$	v_{min}^g	v_{max}^g	$r_{v_a}^g$	$r_{v_r}^g$
1	0.5	0.7	5	0	100	11	0.00	8.3056	5	3
2	0.5	0.7	5	0	100	11	4.73849	11.083	4	3
3	0.5	0.7	5	0	100	11	7.486	18.022	5	3
4	0.5	0.7	5	0	100	11	15.7969	20.00	3	3
5	0.5	0.7	5	0	100	11	18.0102	45.00	5	3

Table 5.1: Grid settings

The Grid Control Strategy uses a grid that is based on the settings which are presented in Table 5.1. This grid contains about 59400 grid points, whereby 38610 points were left out due to the fact that their split value was undefined, i. e. -1. Furthermore, the grid creation time was about one day. The split values in this grid are precomputed using the genetic algorithm which was presented in Chapter 2.4.2. The same genetic algorithm is used by the GA Control Strategy. Thereby, we use the GA configuration $C_{GA} = (k, \rho, \mu, g_{max}) = (100, 0.38, 0.1, 100)$ and the fitness configuration $(w_f, w_{sl}, w_{sd}, w_{ud}, w_{ut}) = (0.45, 0.0, 0.45, 0.1, 0.0)$. Moreover, all predictive control strategies occurring in this chapter use a prediction horizon of size $p = 4$.

5.1 Split Values

In this section, we evaluate the quality of the interpolation algorithm used by the Grid Control Strategy. For this purpose, we compare the interpolated split values of the Grid Control Strategy to the exact split values of the GA Control Strategy. In order to get comparable results, we simulate a car driving with the torque distribution which is determined by the GA Control Strategy and determine additionally in each time step the split value which would be computed by the Grid Control Strategy for the same car state. The resulting split values are shown in Figure 5.1. The red graph displays the split values of the GA Control Strategy, whereas the blue graph shows the split values that would be computed by the Grid Control Strategy for the same car state.

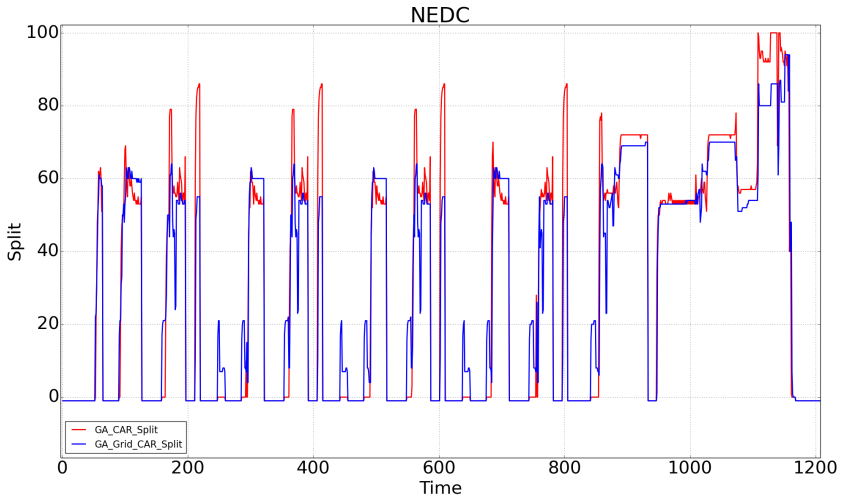

Figure 5.1: Split values of the GA Control Strategy and the Grid Control Strategy for the NEDC

Despite of some differences, the interpolation algorithm of the Grid Control Strategy provides split values which approximate the reference split values of the GA Control Strategy with an acceptable error. However, in the case that there are high peaks within the split values of the GA Control Strategy, it is difficult for the Grid Control Strategy to follow them. Moreover, if the GA Control Strategy provides very high ~ 100 or low ~ 0 split values, the interpolation algorithm has problems to cover these split values. This is based on the fact that the interpolation averages the split values of points in the neighbourhood of the input point. As a result, it is difficult to approximate a single very high or low split value precisely.

In order to decrease the differences of the split values between the strategies, we used the optimisation procedure to refine the grid in specific areas. For this purpose, the GA Control Strategy was used as reference strategy. We optimised the grid for the NEDC with a maximal allowed split difference $\delta = 10$. Moreover, we also

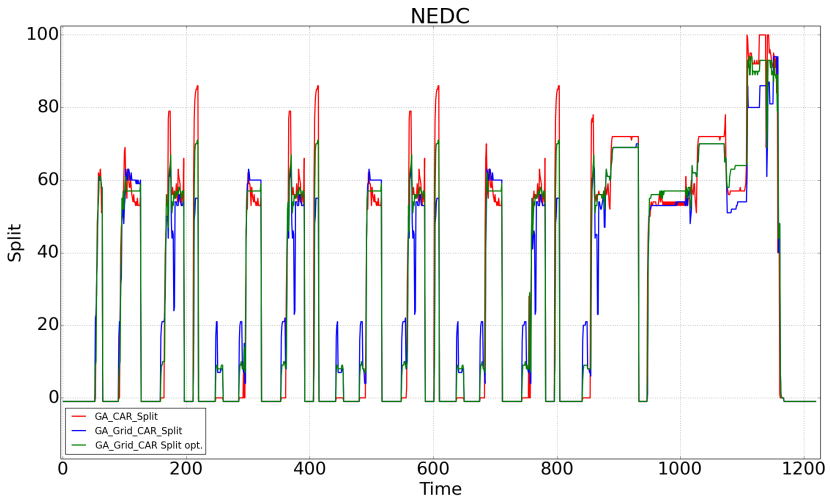

Figure 5.2: Comparison of the split values of the GA Control Strategy and the Grid Control Strategy for an optimised and an unoptimised grid and the NEDC

created a refined grid by increasing the resolution parameters. Thereby, we used $r_s = 7$ for all five gears as well as $r_{v_a} = (r^1_{v_a}, r^2_{v_a}, r^3_{v_a}, r^4_{v_a}, r^5_{v_a}) = (6, 5, 8, 4, 8)$ and $r_{v_r} = (r^1_{v_r}, r^2_{v_r}, r^3_{v_r}, r^4_{v_r}, r^5_{v_r}) = (3, 3, 5, 3, 5)$. The parameter $r^g_{u_0}$ as well as the minimal and maximal values for each dimension were similar to those given in Table 5.1.

The evaluation of the two grids leads to the result that the optimised grid provides better results than the refined grid. Since the creation time of the refined grid was about one week, whereas the optimisation procedure terminates after three days, it is clear that optimising the grid is the better possibility to get a refined grid. The resulting split values of this optimised grid are shown in Figure 5.2. In Figure 5.3 and Figure 5.4, different extracts of Figure 5.2 are displayed. As before, we consider the split values of the GA Control Strategy and the Grid Control Strategy based on the same car which is simulated on the NEDC and uses the torque distribution determined by the GA Control Strategy. The red and the blue graph still display the split values of the GA Control Strategy as well as the split values that would be computed by the Grid Control Strategy by using the unoptimised grid in this situation. The green graph shows the split values of the Grid Control Strategy for the same car using the optimised grid.

In the majority of cases, using the optimised grid instead of the initial homogeneous, unoptimised grid results in a reduced approximation error. The optimised grid contains additional grid points in areas that are often used by the driving cycle. As a result, requested split values can be approximated more precisely: The approximation described by the green graph is closer to the red graph than the approximation shown by the blue graph. Especially, the insertion of new grid points allows to decrease the interpolation error at time steps where the split values of the GA Control Strategy contain very small or very high split values, for example at time steps $210 - 220$ or

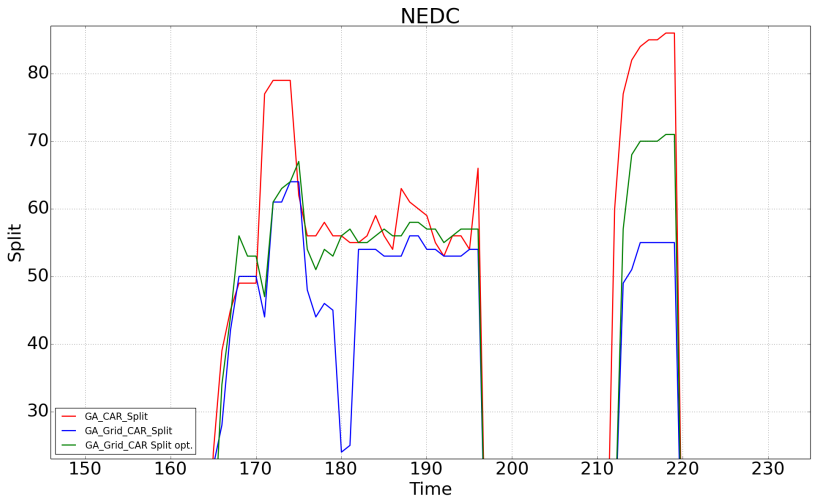

Figure 5.3: Extracts of the comparison of the split values of the GA Control Strategy and the Grid Control Strategy for an optimised and an unoptimised grid and the NEDC

in the area of time step 250. Only in the last part of the driving cycle, from time step 950 to 1110, the split difference is increased slightly. The unoptimised grid provides split values that are located below the reference split values of the GA Control Strategy, whereas the optimised grid results in overshooting split values. This effect might be caused by the fact that in general, the split values became higher in the last part of the driving cycle (time step $1100 - 1200$) by the optimisation. This increase might have influence on the split values at previous time steps since the interpolation algorithm considers all points in the neighbourhood of an input point. But for the optimised as well as for the unoptimised grid, the differences to the split values of the GA Control Strategy are below 10%, which is an acceptable result.

The total difference between the split values of the GA Control Strategy and the Grid Control Strategy for the NEDC and the unoptimised grid is 6074, which denotes an average difference of 8.83 for those of the 1220 time steps where the split values are defined, i. e. $\neq -1$. In contrast, by using the optimised grid the overall split difference is reduced to 4065, which describes an average difference of only 5.91 per time step. Thus using the optimised grid allows to reduce the split difference by 33%.

Moreover, we tested our optimised grid for the HWFET driving cycle. The resulting split values are shown in Figure 5.5. Also for this driving cycle, the Grid Control Strategy computes split values that approximate the exact split values of the GA Control Strategy with an acceptable error. Thereby, the optimised grid achieves slightly better results than the unoptimised grid. This effect is not necessarily to be expected since the grid was optimised for the NEDC. However, this observation shows that optimising the grid for one driving cycle might also have positive effects on other driving cycles.

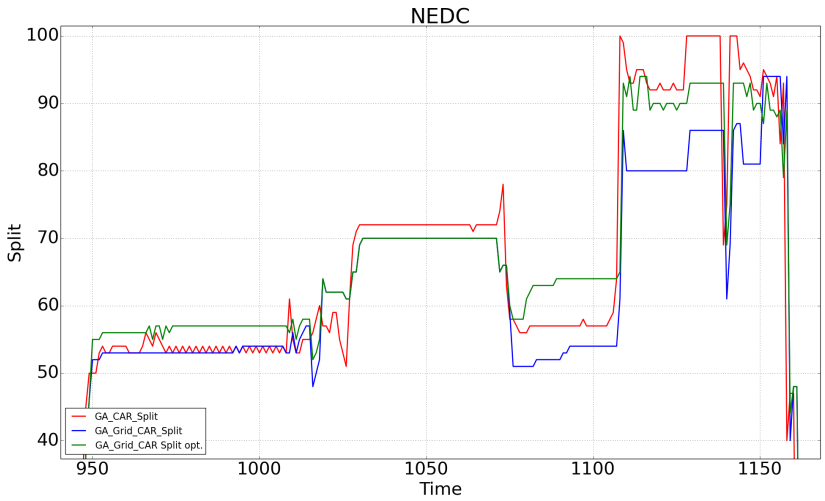

Figure 5.4: Extracts of the comparison of the split values of the GA Control Strategy and the Grid Control Strategy for an optimised and an unoptimised grid and the NEDC

5.2 Fuel Consumption, SoC, Running Time

In this section, we evaluate the fuel consumption, the battery state of charge, the drivability, and the running time of the Grid Control Strategy at the end of the driving cycle. For this, we compare our strategy with regard to these aspects to the GA Control Strategy as well as to the ICE Control Strategy. Therefore, we simulate four cars on the NEDC whereby the first one uses the GA Control Strategy for driving, the second and the third one drives with the torque distributions of the Grid Control Strategy (unoptimised and optimised grid), and the fourth one uses only the combustion engine.

The results are shown in Table 5.2. The car that drives only with the combustion engine (ICE) has the highest fuel consumption on all three driving cycles. The fuel consumption of the GA Control Strategy (GA) as well as the Grid Control Strategy ($GRID$, $GRID_{opt}$) is significantly lower. Both strategies provide good results which do not differ much: The GA Control Strategy achieves a lower fuel consumption for the NEDC, whereby the Grid Control Strategy provides a slightly lower fuel consumption for the FTP_75 as well as the HWFET. For the NEDC and the HWFET, we obtain a lower fuel consumption by using the optimised grid instead of the unoptimised grid.

Since the ICE Control Strategy does not use the electrical motor, the battery of the car that drives only with the combustion engine is fully charged at the end of the driving cycle due to the recuperation of energy while braking. We evaluate the deviation of the SoC at the end of the driving cycle from the reference value $s_{ref} = 0.6$. The GA Control Strategy and the Grid Control Strategy should keep the

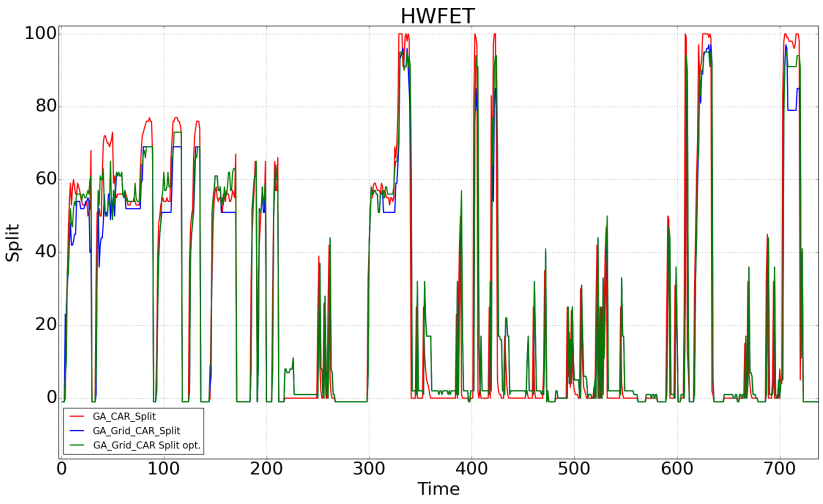

Figure 5.5: Comparison of the split values of the GA Control Strategy and the Grid Control Strategy for an optimised and an unoptimised grid and the HWFET driving cycle

battery state of charge level near this reference value. Both strategies achieve the best results for the FTP_75.

The quality parameter is a value in the interval $[0, 1]$ which allows to set the fuel consumption and the battery state of charge deviation at the end of the driving cycle in relation to each other. The higher this value, the better the strategy with regard to these aspects. Thereby, this value is computed by evaluating the fitness functions e_f as well as e_{sd} at the end of the driving cycle and by weighting them with the corresponding fitness weights: $Quality = \frac{w_f \cdot e_f + w_{sd} \cdot e_{sd}}{w_f + e_{sd}}$. We use the following minimal and maximal values ($fuelCons_{min}$, $fuelCons_{max}$, s_{min}, s_{max}) in order to scale each fitness value to the interval $[0, 1]$:

- NEDC: $(380.0, 430.0, 0.62, 0.71)$
- FTP_75: $(580.0, 728.0, 0.53, 0.71)$
- HWFET: $(345.0, 380.0, 0.58, 0.71)$

For the NEDC, the GA Control Strategy provides the best combination of fuel consumption minimisation and battery state of charge level management. However, the Grid Control Strategy is only slightly behind the GA Control Strategy: Thereby, the optimised grid achieves better results than the unoptimised grid. Concerning the other driving cycles, both strategies provide good results that do not differ much. The Grid Control Strategy which uses the optimised grid provides a slightly better combination of fuel consumption and battery state of charge management than the GA Control Strategy or the Grid Control Strategy that uses the unoptimised grid.

	NEDC				
Strategy	*Fuel cons.*	*SoC*	*Quality*	*Drivability*	*Running time*
GA	381.185	0.625893	0.925405	35624	05 : 41
GRID	383.260	0.625961	0.904346	34114	00 : 57
$GRID_{opt}$	383.030	0.626054	0.906225	31891	00 : 44
ICE	429.077	0.700031	0.076811	0	00 : 17

	FTP_75				
Strategy	*Fuel cons.*	*SoC*	*Quality*	*Drivability*	*Running time*
GA	596.294	0.613648	0.926669	42228	07 : 51
GRID	592.828	0.588564	0.943752	37251	01 : 39
$GRID_{opt}$	592.859	0.590910	0.948362	37378	01 : 35
ICE	725.771	0.700004	0.075186	0	00 : 25

	HWFET				
Strategy	*Fuel cons.*	*SoC*	*Quality*	*Drivability*	*Running time*
GA	354.782	0.578967	0.817921	17900	08 : 30
GRID	355.696	0.578886	0.804553	20114	01 : 31
$GRID_{opt}$	354.185	0.578910	0.826225	24202	00 : 48
ICE	380.800	0.700002	0.056229	0	00 : 17

Table 5.2: Comparison of fuel consumption in gram, battery state of charge level, drivability, and running time in mm:ss at the end of the driving cycle. The running time is evaluated on a 64-bit Ubuntu 14.04 machine with an $i5(4200U)$ processor as well as 6 GB RAM.

Drivability describes the characteristic of split values that consecutive splits do not differ too much. In the case that there are high fluctuations within the split values from one time step to another, the ICE noise emissions are increasing due to the very different torque values the ICE is requested to provide. In Table 5.2, the entry *Drivability* describes for each strategy the sum of the differences between consecutive split values. For the NEDC and the FTP_75 , the Grid Control Strategy has the better drivability, whereas for the HWFET the GA Control Strategy provides a better result. The drivability of the ICE Control Strategy is always zero since this strategy uses only the combustion engine for driving (split value of 100 for all time steps).

Finally, the running time is evaluated. The aim of this thesis was to develop a grid based control strategy for hybrid electric vehicles that provides a smaller running time than an optimisation based control strategy. For all three driving cycles, the running time of the Grid Control Strategy is significantly lower than the running time of the GA Control Strategy. An interesting aspect is the fact that the optimised grid performs better than the unoptimised grid. Since the optimised grid contains more grid points than the unoptimised grid, it would be expected that the unoptimised grid provides a smaller running time. An explanation could be that for the unoptimised grid linear extrapolation is used more often than for the optimised grid since the optimisation procedure insert new grid points for each dimension around an input point. Linear extrapolation has a negative effect on the running time of the Grid Control Strategy since the extrapolated points must be computed in addition.

In summary, the interpolation algorithm of the Grid Control Strategy is able to cover the exact split values of the GA Control Strategy with an acceptable error. By optimising the grid, this error can be reduced. Concerning the fuel consumption and the battery state of charge level management, the GA Control Strategy as well as the Grid Control Strategy provide good results that do not differ much. Finally, the evaluation of the running time has shown that our strategy is able to perform with a running time significantly lower than the optimisation based control strategy.

Chapter 6

Conclusion

After evaluating the experimental results of our implementation of the Grid Control Strategy, we conclude this thesis by providing a short summary and some ideas for further research.

6.1 Summary

In this thesis, a predictive control strategy for hybrid electric vehicles was presented which relies on a discrete grid. The grid contains precomputed control values that are used by the control strategy to interpolate requested control values. In each time step, our strategy receives as input the current state of the car as well as the requested velocities for the prediction horizon. Afterwards, the grid is used to determine the control value for the current input. As normally the input is not contained in the grid, its requested control value is interpolated by the control values of available points in the grid.

The control values contained in the grid are precomputed using an optimisation algorithm. We used a genetic algorithm for this purpose, which is inspired by the biological principle of natural selection. The precomputation of the split values in the grid is done as a preprocessing step for the Grid Control Strategy. For this reason, it is not time critical and there exist no requirements for the time needed by the grid creation.

Due to the fact that interpolation is fast, the Grid Control Strategy can be implemented with a comparably short execution time in contrast to control strategies which are based on optimal control. However, as an optimisation method is used to precompute the control values in the grid, our strategy is able to provide good results if the grid contains enough precomputed control values.

In order to decrease the approximation error of the interpolated split values, we presented a method to optimise the grid. Thereby, the grid is refined partially by

adding new grid points to the grid in areas that are often used by the strategy on a given driving cycle. In doing so, the aim is to increase the density of grid points in the environment of input points so that requested control values of input points can be approximated more precisely.

In the end, we considered the experimental results of our implementation of the Grid Control Strategy. Generally, the interpolation algorithm of the Grid Control Strategy is able to approximate the exact split values of the GA Control Strategy with an acceptable interpolation error. Thereby, the optimised grid achieves better results than the unoptimised grid. The evaluation of the fuel consumption and the battery state of charge has shown that the Grid Control Strategy as well as the GA Control Strategy provide good results with minor differences. Moreover, our strategy can be executed with a smaller running time compared to an optimisation based control strategy.

6.2 Future Work

Finally, this thesis is concluded by suggesting some ideas for future research. The Grid Control Strategy has potential for improvements and further development: In our implementation, we use a genetic algorithm to precompute the split values in the grid. Instead, it would be interesting to analyse the behaviour of the strategy if an other optimisation algorithm is used for this purpose.

We add new grid points to the grid in order to increase the quality of interpolated split values in areas that are often used. In contrast, another option would be to combine similar points in the grid which are rarely used. This would reduce the size of the grid.

Moreover, in order to define the interpolation set of an input point, we consider a fixed box around this input point. Another possibility might be to modify the size of this box, for example using a smaller or larger box or a box whose size varies depending on the density of grid points in the area of an input point. This aspect especially could be interesting for an optimised, inhomogeneous grid.

Furthermore, the following ideas could be tested and evaluated for our implementation of the Grid Control Strategy:

- Using different settings for the genetic algorithm, e. g. considering different fitness configurations, population sizes or numbers of maximal iterations.
- Testing the behaviour of the strategy for other driving cycles than those considered in this thesis.
- Further optimising the grids, not only for one driving cycle, but also training the grids with multiple driving cycles.

Increasing the size of the prediction horizon also increases the number of grid dimensions. The number of grid points grows exponentially for each additional di-

mension. Thus the strategy is only practicable for a small prediction horizon. In order to be able to consider also higher prediction horizons, a possibility would be to define different track types with different accelerations. Afterwards, each grid point is associated with one track type. In doing so, the requested velocities for the future time steps are no longer necessarily to be stored, instead they can be computed by assuming that the car is accelerating continuously with the acceleration given by the track type of the grid point.

Bibliography

[BLMB09] T.J. Barlow, S. Latham, I.S. McCrae, and P.G. Boulter. *A Reference Book of Driving Cycles for Use in the Measurement of Road Vehicle Emissions: Version 3.* Published project report ppr354. TRL Limited, 2009.

[dtv] *dtv-Atlas zur Mathematik - Analysis und angewandte Mathematik, Band 2.* Deutscher Taschenbuch Verlag GmbH & Co.KG, Muenchen.

[gen] Genetic Algorithm Library GeneiAL. `http://geneial.org`.

[GJN+15a] S. Geulen, M. Josevski, J. Nellen, J. Fuchs, L. Netz, B. Wolters, D Abel, E. Ábrahám, and W. Unger. Learning-based Control Strategies for Hybrid Electric Vehicles. In *Proc. of CCA*, pages 1722–1728. IEEE, 2015.

[GJN+15b] S. Geulen, M. Josevski, J. Nellen, J. Fuchs, L. Netz, B. Wolters, E. Ábrahám, W. Unger, and D. Abel. Online Lernen als Kontrollstrategie in Hybridfahrzeugen. In *Proc. of AUTOREG: Auf dem Weg zum automatisierten Fahren*, volume 2233 of *VDI-Berichte*, pages 101–112. VDI Verlag, 2015.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[HH98] R.L. Haupt and S.E. Haupt. *Practical Genetic Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[NWN+15] J. Nellen, B. Wolters, L. Netz, S. Geulen, and E. Ábrahám. A Genetic Algorithm based Control Strategy for the Energy Management Problem in PHEVs. In *Proc. of GCAI*, volume 36 of *EPiC Series in Computer Science*, pages 196–214. EasyChair, 2015.

[PB14] A. Panday and H.O. Bansal. A Review of Optimal Energy Management Strategies for Hybrid Electric Vehicle. *International Journal of Vehicular Technology*, 2014:19, 2014.

[Phi03] G.M. Phillips. *Interpolation and Approximation by Polynomials.* Springer-Verlag New York, Inc., 2003.

[PIGV01] A. Piccolo, L. Ippolito, V. Galdi, and A. Vaccaro. Optimisation of Energy Flow Management in Hybrid Electric Vehicles via Genetic Algorithms. In *Proc. of ASME*, pages 434–439. IEEE, 2001.

[PR07] P. Pisu and G. Rizzoni. A Comparative Study Of Supervisory Control Strategies for Hybrid Electric Vehicles. *IEEE Transactions on Control Systems Technology*, 15(3):506–518, 2007.

[SD08] S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Springer Verlag, 2008.

Appendix A

Driving Cycles

A.1 NEDC

The NEDC (New European Driving Cycle) is a stylised driving cycle which represents a typical use of a passenger car in Europe. Thereby, the first 800 time steps represent a city part, whereas the last 400 time steps display an extra-urban part. The city part consists of four equivalent sections which are repeated. The driving cycle has a duration of 1220 s. In this time, a route of 11017 m is covered. Moreover, the maximal velocity the car achieves is 120 km/h, which is about 33 m/s [BLMB09].

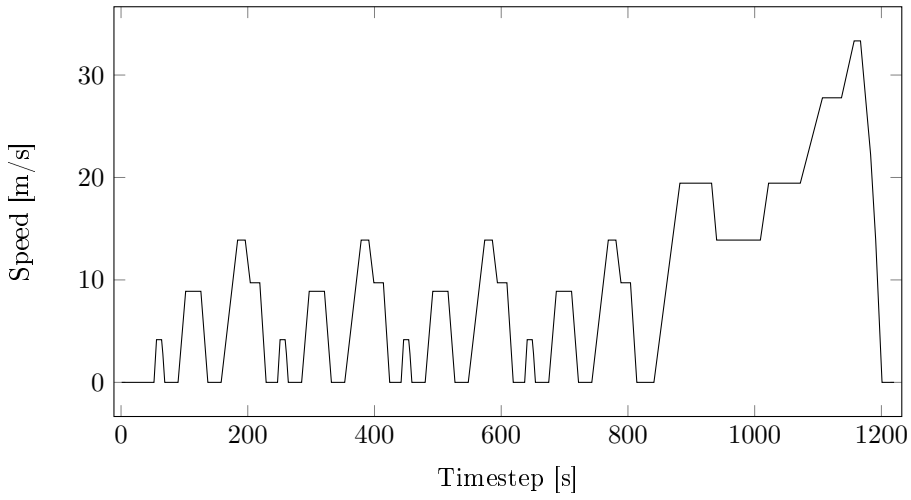

Figure A.1: The NEDC

A.2 FTP_75

The FTP_75 represents characteristic driving conditions in a US city. In contrast to the NEDC, the US driving cycles are not generated. Instead, they are constructed by averaging the data which was obtained by real cars driving the routes. For this reason, these driving cycle do not seem to be as generic as the NEDC. The total duration of the FTP_75 is 1874 s, whereby a route of 17786 m is covered. The maximum speed is about 91 km/h, which is ≈ 25 m/s [BLMB09].

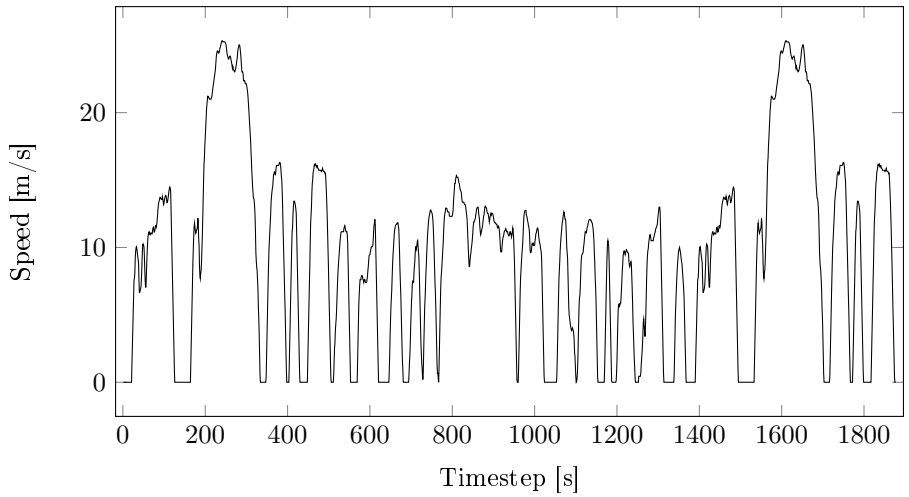

Figure A.2: The FTP_75 driving cycle

A.3 HWFET

The HWFET cycle represents characteristic driving conditions on a US highway. Its duration is 765 s, whereby the car received a maximum speed of about 96 km/h $\approx$ 27 m/s. In total, a distance of about 16503 m is covered [BLMB09].

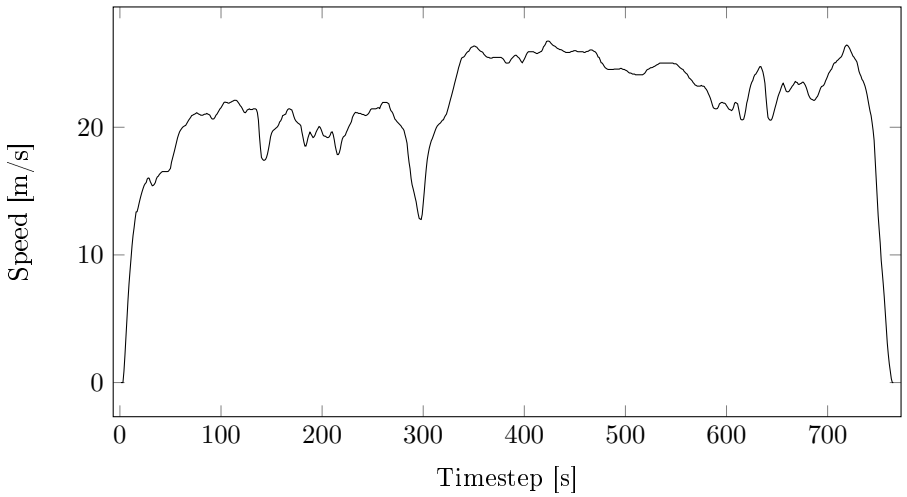

Figure A.3: The HWFET driving cycle