Heidelberg University
Ziti – Institute of Computer Engineering
Automation Laboratory

TITLE

Master Thesis
submitted by

Elena Elenkova

Matrikelnummer 328 00 12

conducted at
Automation Laboratory,
Heidelberg University

Date: 21st June 2016

Supervisor: Prof. Dr. sc. techn. Essameddin Badreddin
Second marker: Prof. Dr. Peter Fischer

Declaration of Authorship

according to the § 16 (2) „Prüfungsordnung der Universität Heidelberg für den Master-Studiengang Angewandte Informatik “ from 20 July 2010.

I have written the following thesis completely on my own without using any other sources and help materials other than the ones explicitly specified.

Mannheim, 21st June 2016 _____

Elena Elenkova

Abstract

Acknowledgement

I would like to thank ....

Contents

List of Figures

List of Tables

1 Introduction

This chapter outlines the motivation for the choice of the topic and sets the scope and the goals of this thesis. Moreover, a brief history of the initial interest in hybrid vehicles and the the state of the art is presented, concentrating on the latest literature from the past 10 years written on the topic.

1.1 Motivation

A crucial aspect when it comes to understanding the interest of the author in a particular topic is the motivation. Due to the previous completion of two seminars in the area of Game Theory, the author developed a genuine interest in this topic. Another reason comes from the fact that a lot of real-life situations can be represented as games like investments in economic circumstances. They can be solved by optimal approaches which are mathematically defined and usually present maximization or minimization problems. The first seminar dealt with Pursuit-Evasion games with UAVs (Unmanned Aerial Vehicles) and was an essential introduction to the fundamentals of Game Theory for the author who had absolutely no experience in Game Theory beforehand. The second seminar was specifically pointed at this thesis and coped with the problem of power management in hybrid vehicles from a game-theoretical point of view. Thus, the necessary theoretical knowledge for completing the goals of this thesis was acquired prior to commencing the project.

1.2 Goals

Before starting the thesis, the various goals need to be set. First of all, even before defining the game, a hybrid vehicle model has to be developed. It must include all essential components of a hybrid car. For this reason a lot of parameters need to be taken from a real hybrid vehicle, for example a Toyota Prius Hybrid car. The hybrid model can be implemented in Simulink where it will be simulated later. Secondly, the game to be played must be formally defined. It needs to be decided how many players are going to take part in it, what kind of a game it is - cooperative or non-cooperative, according to whether the players are allowed to make agreements between themselves during the game. Thirdly, after the game definition, plausible solution approaches have to be chosen which can be applied to solve the game at every stage (time step). Thorough examination of at least 2 or 3 game-theoretical solution approaches is required so that their results can then be compared. In the end, not only 2 or 3, but 6 game-theoretical approaches were employed. Afterwards, the solutions have to be implemented as algorithms and each solution approach has to produce a single solution at every stage. Lastly, when the algorithms have been implemented, the defined game is dynamically simulated and solved at each time step. Regarding the simulation, it is best if a whole established drive cycle is simulated instead of a user-defined speed vs. time demand. To summarize, the following four primary goals of the thesis can be stated:

- Create hybrid vehicle model
- Specify game-theoretical definition
- Develop game-theoretical solution approaches
- Simulate game

1.3 State of the art

The first hybrid vehicle was built in 1900 by Ferdinand Porsche. The car was called Lohner-Porsche Mixte Hybrid. It was a serial hybrid car with two wheel motors, it had a 5.5 liter 18kW engine, a battery and an electric generator. The first mass produced hybrid car was the Toyota Prius from 1997 manufactured in Japan.

The first research papers about hybrid vehicles date back from the 1970s. LaFrance and Schult (1973) give an overview of one serial configuration and two parallel configurations - with a single and with a dual motor. They compare the suitability of different electric motor designs for various driving conditions.

Although there has been a lot of research in hybrid vehicles in the last two decades, there is a limited number of approaches which consider the power management problem from a game-theoretical point of view.

First of all, Gielniak and Shen (2004) describe a fuel cell hybrid electric vehicle and solve the power distribution problem as a two-player non-cooperative game. The vehicle has a fuel cell, battery, ultra capacitor and two 35 kW motors. The fuel cell tries to maintain a target 60% State of Charge (SOC) of the Energy Storage Subsystem (ESS) or the battery. Power is firstly taken from the fuel cell and then the rest is taken from the ESS. The voltage of the ESS depends on the SOC and it is also being cooled. The Ultra Capacitor (UC) is a lumped energy storage device, its SOC depends on the UC voltage and it is also air cooled. The motor has a maximum bus power limit. In addition, there is a transmission model and accessory loads. The accessories such as air conditioner and power steering are powered from the electric motor.

The game-theoretical approach involves two players - all power supplying components as the first player and all power consuming components as the second player. The Master Power Management Controller (MPMC) governs all components in the powertrain and calculates the solution. The objective of the game is to save fuel and to accelerate fast. The decisions of the players are how much power to supply at a given moment. The payoff of each player is represented by utility functions - efficiency, performance and composite utilities. The efficiency utility is a function of the strategy of the opponent - how much power they contribute. The performance utility is in this case acceleration. The composite utility is then computed for a time moment after all components have been mapped to an efficiency or performance function. The authors Gielniak and Shen (2004), however, do not provide any insights on how exactly the game-theoretical solution was computed.

Regarding simulation three drive-cycles were examined - Federal Test Procedure (FTP), US06 and Constant 60 Miles Per Hour. A simulation tool called ADVISOR (Burch, Cuddy and Markel, 1999) is used. A comparison is drawn between a Basic control strategy without game theory and a game-theoretical control strategy. In the US06 cycle, the Game Theory control achieves more miles per gallon than the Basic

control. Regarding acceleration the Basic control is able to accelerate from 0 to 85 miles in 17.48s, whereas the Game Theory control needs only 11.77s.

A further approach towards power control is presented by Chin and Jafari (2010). The hybrid is based on the Toyota Prius and has a gasoline engine and an electric motor which are the two players in a bimatrix game. The vehicle configuration consists of a Gasoline Engine Controller, Electric Motor Controller and a Power Controller as the main unit for the computation of the solution. On the one hand, the Gasoline Engine Controller manages the fuel and air injection. A three-way catalyst system is utilized for the gas emissions of HC, CO_2, CO and NO_x. On the other hand, the Electric Motor Controller includes also battery and capacitor units. Batteries have high energy capacity but cannot provide much power, whereas capacitors have less energy capacity, but can produce a lot of power. Capacitors help improving battery lifetime and are also useful for providing short bursts of power during acceleration. The Power Controller is responsible for computing the game theory solution. It takes the requested torque from the driver and determines the optimal strategies for the engine and the electric motor. It outputs the torque for each of the two power sources. In addition, it charges the battery when the SOC is low. The goal of the Power Controller is to minimize fuel usage and at the same time maximize torque and reduce gas emissions. This hybrid configuration also has different modes of operation - Engine only, Motor only, Mixed and Battery Charge.

The game-theoretical solution consists of a non-cooperative bimatrix game. The payoff matrices for both players - the engine and the electric motor, are of size $M \times N$. These denote the strategies of player 1 and 2. The game is solved by a Nash Equilibrium (Nash, 1951). When player 1 chooses strategy $i \in M$ and player 2 chooses strategy $j \in N$ the payoff is (a_{ij}, b_{ij}) and it constitutes the Nash Equilibrium if this pair contains the optimal strategies for both players. Pure strategies are extended to mixed by specifying the strategies as a vector of probabilities over all pure strategies. The strategies are how much torque to contribute, measured from 0 to 6000rpm. Each payoff entry in the matrix is a function of fuel consumption, gas emissions, engine temperature, SOC deviation, extra weight and driver's demands. The Nash Equilibrium of the game is computed by the Lemke-Howson algorithm (Lemke and Howson, 1964) in the Power Controller. However, a major weakness is that no simulation results are presented.

Dextreit and Kolmanovsky (2014) use the hybrid configuration of the Jaguar Land Rover Freelancer2. It has a diesel engine, two electric motors and a six-speed dual clutch transmission. The first electric motor is attached to the engine Crankshaft

Integrated Starter Generator (CISG), while the second is attached to the rear wheels and is called the Electric Rear Axle Drive (ERAD). There are five driving modes. The EV mode is when only the ERAD provides the torque, the Engine-only mode is when only the engine supplies the torque. The parallel mode means that both the engine and the motors provide the torque. There is a charging mode in which the engine produces the driving torque and the CISC torque. The last mode is a boosting mode, where the engine supplies the additionally needed driving torque.

The game-theoretical approach Dextreit and Kolmanovsky (2014) adopt is a finite-horizon non-cooperative game, where the two players are the driver and the powertrain. The cost function penalizes fuel consumption, NO_x emissions and the battery SOC deviation. In a typical dynamic programming approach the cost is a function of a state vector, a control vector and a vector of operating conditions. The driver chooses the operating conditions - requested wheel speed and wheel torque. The powertrain chooses the control variables and the state vector is the SOC of the battery. The game is solved using a feedback Stackelberg equilibrium (Von Stackelberg, 1952). Firstly, the game is solved statically where the first player is the leader, who maximizes a function $J(w,t)$ by selecting the powertrain operating demands $w(t) \in W$. The second player (the follower) is able to observe the first player's decision and depending on that it selects its control vector $u(t) \in U$. It is assumed that both players make rational decisions. The pair (w^*, u^*) is the Stackelberg equilibrium. Next, the dynamic game is described which consists of $(T-1)$ stages, also called the horizon of the game. Given the initial state vector $x(0)$ the follower chooses their move $w(0)$. Then the follower selects their move $u(0)$. Thus, after the first stage the state is $x(1) = f(x(0), u(0), w(0))$. This continues until the last $(T-1)$ stage of the game. The GT controller Dextreit and Kolmanovsky (2014) implemented computes the state, control and operating values offline in three modules - GT Maps, Mode Arbitration and Mapping to torque demand. In the GT Maps module the wheel torque and speed, gear and battery SOC are discretized. The wheel torque is in the range of 0 to 1000 Nm. The wheel speed is between 0 and 100 rad/s. The battery SOC is between 40% and 70%. As output the module produces two modes to choose from. The Mode Arbitrator chooses one of these, for example the EV or the parallel mode. The aim of the Mapping to torque demand module is to distribute the torque between the engine and the motors.

Dextreit and Kolmanovsky (2014) present a baseline controller solved using Dynamic Programming and compare it with the game theory controller. Measurements are conducted over three drive-cycles for CO_2 emissions, equivalent to fuel consumption,

NO_x emissions and deviation of SOC. These show that the GT controller outperforms the baseline controller.

There is another similar approach to the one of Dextreit and Kolmanovsky (2014). The idea is extended by Chen et al. (2014) who describe a single-leader multiple-follower game, where the follower is not only one (the battery) like in Dextreit and Kolmanovsky (2014), but also include other auxiliaries. The vehicle is a heavy-duty truck with an electric refrigerated semi-trailer and a battery. The aim is to manage the battery power and the refrigerated semi-trailer power. The vehicle model consists of the internal combustion engine. The total power which is requested from it is a sum from the driving power and the power for all auxiliaries. The semi-trailer model considers the air transfer from inside and outside and the cooling power. The problem is formulated as minimization of fuel consumption over a time interval. Energy balance is sought so that the battery energy and the air temperature in the semi-trailer at the end of the drive cycle are the same as in the beginning.

The game-theoretic approach assumes a single leader - the driver and multiple followers - all auxiliaries. The leader selects their action $w(t) \in W$ - the demanded wheel torque and speed. Then each follower f_i chooses their action $u_i(t)$. The game is split in two levels. The first level considers N two-player games between the leader and each of the N followers. It solves the games offline with a Stackelberg equilibrium where the driver maximizes and the leader minimizes the cost function. The strategies are stored in a lookup table for each follower. The second stage is computed online and it combines all followers who play their Stakelberg strategies as stored in the lookup table. They all have to reach a mutual decision and this can be solved by a Nash Equilibrium. The central computation happens in the Energy Management System Operator (EMSO). It gathers the actions from all followers and sends them back the computed overall strategy for this stage. This happens until an equilibrium is reached. For simulation Chen et al. (2014) compare two control strategies - optimizing only the battery and optimizing both the battery and the semi-trailer power with the game theory approach. The optimized battery and trailer achieve better fuel consumption by 0.08%.

After exploring the literature, it can be concluded that all game-theoretical approaches assume a non-cooperative game and solve it either by a Nash Equilibrium or by a Stackelberg Equilibrium if the players are regarded as a leader and a follower. Mostly, two players are involved in the game methods proposed. In the leader-follower concept the players are the driver and the powertrain, as described in two of the approaches above. In another approach the two players are divided into the power-consuming and

power-supplying devices in the powertrain. In another case Chin and Jafari (2010) described the two players as the electric motor and the engine. The most crucial point to bear in mind is that that no solution approach applies a cooperative game.

1.4 Content

2 Fundamentals

Firstly, this chapter outlines the concept of hybrid vehicles and how they work. Secondly, the cooperative game and specifically all required mathematical definitions are explained. Then six different game-theoretical solution approaches are presented.

2.1 Hybrid vehicles

This section explains the concept of hybrid cars and the distinct types which exist. It also describes how they operate under various driving conditions in order to achieve maximum efficiency.

A hybrid vehicle is a vehicle which utilizes more than one source of power, an engine and an electric motor. Energy is stored in both of them and also in the battery. The engine can use either gasoline or diesel as fuel. The goal of hybrid vehicles is to save fuel by combining the power of an internal combustion engine and the power of the electric motor. This is achieved by reusing the surplus power from the engine, which is often lost in conventional vehicles. In hybrids it is instead stored in the battery. By combining the ability of the electric motor to provide high torque in the small revolution speed range and the ability of the engine to produce high torque in the middle to high revolution speed range, hybrid vehicles achieve the required driving behaviour at a lower power cost. Hybrids can benefit from these opposing characteristics of the engine and the motor.

2.1.1 Classification

Hybrid vehicles can be classified into three distinct types - series, parallel and series-parallel. In serial hybrid vehicles only a single resource of power can be used at a time. The electric motor transmits torque to the wheels when there is enough battery power. Only the electric motor is connected to the wheels and it receives power from either the generator and the engine or from the battery. When the battery needs to be recharged, power is taken from the engine, which acts as a generator. In contrast, in parallel hybrid vehicles the wheels are driven by both the engine and the electric motor; hence, the name, because both power sources are able to run in parallel. There is a mechanical connection both from the engine and from the motor to the wheels. The engine is thought to be the main power supplier and the motor is used for boosting. If the motor is recharging the battery, it is impossible for it to also drive the vehicle at the same time. The third mixed type series-parallel benefits from the advantages of both designs by combining them. In this hybrid type it is possible that the engine drives the wheels or that the wheels are completely driven by the electric motor. The Toyota Prius is designed as a series-parallel hybrid vehicle.

2.1.2 Driving behaviour

A typical driving cycle begins by starting up. In the beginning it is efficient to use the motor for the low speed range up to the middle speed range and the engine can remain shut down. While maintaining a constant speed during cruising, power is distributed between the motor and engine so that both of them provide torque. In acceleration mode power is taken also from the battery, while both the engine and motor drive the vehicle again. During deceleration, the power from braking is recovered instead of lost and used to recharge the battery. The motor acts as a generator to convert the energy. This process is called regenerative braking. If the state of the charge of the battery falls below a certain percentage, it needs to be recharged by the engine which acts as a generator.

2.2 Vehicle configuration of Toyota Prius

This section gives an overview of the main components which every conventional car nowadays, not only hybrid cars, possess. Furthermore, it describes the special hybrid

components while specifically adhering to one particular model of a hybrid car, namely the Toyota Prius, which was the first mass-produced hybrid car in 1997. The vehicle, whose parameters are used throughout this thesis for the Simulink vehicle model and for the simulation, is the Toyota Prius second generation, manufactured from 2003 to 2009.

2.2.1 Internal Combustion Engine

The internal combustion engine of the Toyota Prius is an Atkinson cycle engine. It is a 1.5 litres engine with maximum power of 57 kW or 76 hp at 5000 rpm. The maximum torque is 115 Nm at 4200 rpm. The Atkinson cycle is to be found mainly in hybrid-electric vehicles. Compared to a conventional Otto-cycle engine, the Atkinson cycle engine does not let as much air inside. The valve remains open longer and thus air can also escape. Consequently, the engine has a higher expansion ratio than compression ratio. A higher expansion ratio helps energy to be transformed from heat into mechanical energy, thus making the engine more effective. The goal is to acquire all available power of the engine by maintaining pressure balance so that the pressure after the power stroke is the same as the pressure of the atmosphere. This leads to reduced power in comparison with an Otto-cycle engine. However, additional power can be supplemented by the electric motor when necessary. Therefore, the Atkinson cycle was originally designed to provide more efficiency, but this happens at the cost of decreased power density.

2.2.2 Electric Motor

The electric motor, which Toyota uses in its hybrid vehicles is a synchronous A/C motor because these motors can provide relatively large torque also in the high revolution per minute range. In the specific case of this motor, the torque at the maximum revolution speed of 6000 rpm is 70 Nm. The maximum torque of the motor is 400 Nm at 0 rpm and maximum power is 50 kW or 67 hp at 1200 rpm. In total the engine and the motor can generate power of 82 kW or 110 hp together.

A synchronous electric AC motor runs with Alternating Current (AC) as opposed to DC motors running with Direct Current. A typical AC motor has an inside rotor generating a magnetic field and an outside stator which creates another magnetic field. The rotor magnetic field in AC motor can be produced in different ways, for example,

by permanent magnets such as in the Toyota Prius permanent magnet synchronous motor (PMSM). Permanent magnets produce a constant magnetic field. In synchronous motors the shaft is rotating with the same frequency as the supplying current. The permanent magnets on the rotor rotate in line with the stator magnetic field; hence, producing synchronized rotation. Such permanent magnet synchronous motors have to be started with power which is with variable frequency.

2.2.3 Battery

The battery is a 201.6V Nickel-metal hydride containing 28 modules, which can produce 1310 kWh in total. Each module consists of six 1.2V cells which are connected in series to generate in total 7.2V 6.5Ah. The weight of the whole battery pack is 53.3 kg. It consists of two battery packs, a high voltage (traction) and a low voltage. The low voltage battery pack is crucial when the vehicle starts because it provides initial power and the engine can remain shut down. The ultimate aim of the battery is to keep its SOC between 40-80%. The advantage of Nickel metal hydride batteries over other types of batteries is that they have long lives. However, a significant disadvantage is their poor performance in cold temperatures.

2.2.4 Power Split Device

Toyota patented the Hybrid Synergy Drive (HSD) which is a hybrid drive technology not only used in the Toyota Prius, but also in other models like Toyota Yaris and other car makers like Lexus and Nissan. The most crucial part is the gear set, called Power Split Device (PSD). It is a Continuously Variable Transmission (CVT) but it has a fixed gear ratio. Its main function is the combination of the power of the engine, the electric motor and the generator which all rotate with different speeds. The PSD is in itself a planetary (epicyclic) gear train such as the one shown in Figure 2.1. It consists of four parts. In the middle there is a sun gear (yellow) and planet gears (blue) are revolving around the sun gear. The carrier is green and the annular gear is pink. The carrier is the part which connects the centres of the planet and the sun gears so that the planet gear can rotate around the sun gear. These three components are inside the annular gear which is a fixed outer circle. In fact, the Power Split Device is similar to a differential device. Therefore, the next subsection explains the function of a differential, which is to be found in every car.

2.2.5 Differential

After the power split device has fulfilled its purpose to transmit the torque to the wheels, a differential is required to cope with the difference in rotation speeds of the left and right wheels during turns. The primary function of a differential in vehicles is to allow the outer wheel to rotate faster than the inner wheel while the vehicle is turning left or right. A differential consists of three shafts, where the first is the input drive shaft and the other two are the outputs to the two wheels. These shafts possess the characteristic that the angular velocity of one of them is the average of the angular velocities of the other two. A further attribute of the angular velocities is that the mean velocity of both wheels is the same as the velocity of the input drive shaft. Therefore, if the velocity of one wheel grows, this is balanced by reducing the velocity of the other wheel.

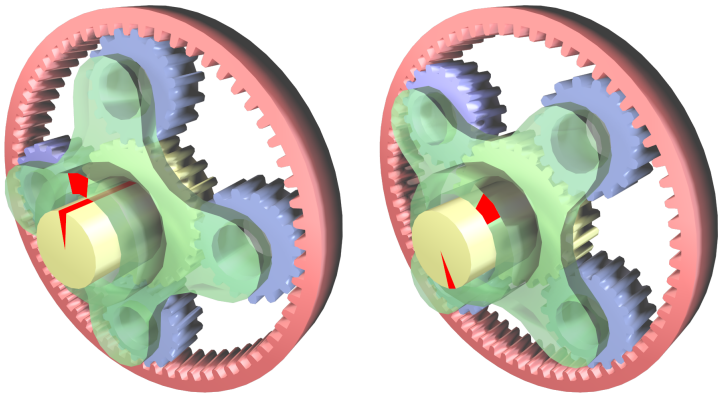

Figure 2.1.: *Planetary (epicyclic) gear train (Wikipedia, 2006)*

2.3 Game definition

This section handles the formal definition of the game and the applied game-theoretical solution approaches. Firstly, the generic game definition is given. Additional definitions for each of the solution approaches are defined in the corresponding subsections of each solution.

According to Holler and Illing (2006) a game $G = (N, S, u)$ is defined by:

- set of players $N = \{1, 2\}$, in this case two players, where player 1 is the Engine and player 2 is the Motor.

- strategy space S, which is the set of all possible strategy combinations $s = (s_1, s_2)$ for each player, where $s \in S$. The strategy space contains two sets with all strategy combinations of that player.
- utility function $u = (u_1, u_2)$, where $u_i(s)$ for $i = 1, 2$ gives the utility (or also called payoff) for that player when the strategy combination s is played.
- utility space (payoff space) which is the set of all possible utility combinations: $P = \{u(s)|s \in S\} = \{(u_1(s), u_2(s), \forall s \in S\}$

Let us denote the number of pure strategies of each player with m and n. Therefore, these constitute a bimatrix game, meaning that the payoffs of the game can be represented in two matrices of size $m \times n$. Let the strategy spaces constitute $s_1 = \{s_1^1, ..., s_1^m\}$ and $s_2 = \{s_2^1, ..., s_1^n\}$. Furthermore, let the payoff matrices A and B denote the payoffs for the first player, the engine, and the second player, the motor respectively, where $A = (a_{ij} : i \in \{1, ..., m\}, j \in \{1, ..., n\})$ and $B = (b_{ij} : i \in \{1, ..., m\}, j \in \{1, ..., n\})$. Their contents are shown in Table 2.1 and Table 2.2. It should be noted that $u_1 = a$, $u_2 = b$ and a and b notations have been introduced for simplicity. Sometimes player 1 is called the row player and player 2 is called the column player, since their strategies vary along the rows or along the columns of the matrices. The game is a non-zero sum game, since the sum of the two payoff matrices is not 0, $A + B \neq 0$.

	$\mathbf{s_2^1}$	$\mathbf{s_2^2}$	$\mathbf{s_2^3}$	$\mathbf{s_2^4}$	$\mathbf{s_2^5}$	$\mathbf{s_2^6}$	$\mathbf{s_2^7}$
$\mathbf{s_1^1}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$\mathbf{s_1^2}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$\mathbf{s_1^3}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$
$\mathbf{s_1^4}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$
$\mathbf{s_1^5}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$
$\mathbf{s_1^6}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$
$\mathbf{s_1^7}$	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$

Table 2.1.: *Engine payoff matrix*

As opposed to a non-cooperative game, in cooperative games the players are allowed to make binding agreements among themselves in order to achieve a better payoff, that is, they can form coalitions. We distinguish between the grand coalition of all players N or $\{1, 2\}$, where they all cooperate, and the individual coalitions $\{i\}, \forall i \in N$

	$\mathbf{s_2^1}$	$\mathbf{s_2^2}$	$\mathbf{s_2^3}$	$\mathbf{s_2^4}$	$\mathbf{s_2^5}$	$\mathbf{s_2^6}$	$\mathbf{s_2^7}$
$\mathbf{s_1^1}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$	$b_{1,6}$	$b_{1,7}$
$\mathbf{s_1^2}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	$b_{2,5}$	$b_{2,6}$	$b_{2,7}$
$\mathbf{s_1^3}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	$b_{3,6}$	$b_{3,7}$
$\mathbf{s_1^4}$	$b_{4,1}$	$b_{4,2}$	$b_{4,3}$	$b_{4,4}$	$b_{4,5}$	$b_{4,6}$	$b_{4,7}$
$\mathbf{s_1^5}$	$b_{5,1}$	$b_{5,2}$	$b_{5,3}$	$b_{5,4}$	$b_{5,5}$	$b_{5,6}$	$b_{5,7}$
$\mathbf{s_1^6}$	$b_{6,1}$	$b_{6,2}$	$b_{6,3}$	$b_{6,4}$	$b_{6,5}$	$b_{6,6}$	$b_{6,7}$
$\mathbf{s_1^7}$	$b_{7,1}$	$b_{7,2}$	$b_{7,3}$	$b_{7,4}$	$b_{7,5}$	$b_{7,6}$	$b_{7,7}$

Table 2.2.: *Motor payoff matrix*

which are {1} and {2}. There is also the empty coalition, where neither cooperates, but this coalition is irrelevant to our purposes. Each coalition has a value associated with it. The grand coalition forms its value as a sum of the payoffs of the engine and motor multiplied element-wise by a matrix with weights. The weights are distributed according to the torque deviation which the engine and motor produce. The torque deviation is defined as the difference between the required and the actual torque at the current time step. When the deviation is 0, the payoffs are weighted by 0.99, when it is between 0-10% of the required torque it is weighted by 0.991, when between 10-20% weight is 0.992 and so on up to 1.0 (the full sum of engine and motor torque).

The goal of the game is to save fuel and to maintain low gas emissions while achieving the required torque at any moment in time. Therefore, the payoffs are penalties as opposed to benefits and they have to be minimized. All of the solutions have been defined by taking into account that this is a minimization problem. Most of the solutions applied in the literature work with maximizing payoffs, but for the purpose of this thesis their definitions and implementations have been modified to minimize the two payoff functions instead.

There exist two types of cooperative games - with transferable and with non-transferable utility. In transferable utility (TU) games the payoff of one player can be transferred to another player without any loss. In contrast, in non-transferable utility (NTU) games the payoffs of each player cannot be redistributed among the other players. In this thesis the payoffs of the engine and the motor are not interchangeable, since decreasing the payoff of one player does not mean increasing the payoff of the

other player at the same time. Therefore, their payoff functions are thought to be independent from each other.

The payoff functions are constructed in the following way. The engine payoff is:

$$a_{ij} = w_1 \times fuelConsumptionRate + w_2 \times |requiredTorque - actualTorque| + w_3 \times HCemissions + w_4 \times COemissions + w_5 \times NOXemissions + w_6 \times fuelConsumed \quad (2.1)$$

Where the fuel consumption rate is in grams per second (gps), the difference between required and actual torque is in Newton meters (Nm), the gas emissions are all in gps and the consumed fuel from the beginning of the simulation up to this time step is in litres. The motor payoff is:

$$b_{ij} = w_2 \times |requiredTorque - actualTorque| + w_7 \times powerConsumed + w_8 \times SOCdeviation \quad (2.2)$$

Where the consumed power is in kW from the beginning of the simulation and SOC deviation is the difference between the target SOC of 70% and the current SOC of the battery.

To summarize, the game defined is a two-player cooperative bimatrix non-zero-sum game with non-transferable utility.

2.4 Game-theoretical solution approaches

This thesis examines a variety of solution approaches for cooperative games which are later applied and simulated as described. The following section describes the theoretical and mathematical definitions of the six solution concepts. These are Pareto Optimality, Nash Equilibrium, Nash bargaining solution, Kalai-Smorodinsky bargaining solution, the Core and the Shapley value.

2.4.1 Pareto Efficiency and Pareto Optimality

In the literature two terms are often used to refer to the same concept - Pareto Efficiency and Pareto Optimality. However, they are used with distinct meanings throughout this thesis. Pareto Efficiency denotes an allocation of resources such that no player can improve their outcome without impairing another player. In the strategy space of the game there can exist more than one Pareto efficient outcome. Therefore, we define a Pareto optimal outcome, or Pareto Optimality, as the single best outcome from the set of all Pareto efficient outcomes. The criteria for determining the best outcome is as follows. All Pareto efficient points are compared by their torque deviation and the point with the least torque deviation is taken as the Pareto optimal point. Torque deviation is defined as the absolute difference between the required and the actual torque at this stage of the game. If more than one outcomes have the same torque deviation the outcome with the smallest fuel consumption rate is taken as a second criterion. Similarly, if more than one outcome have the same fuel consumption rate the one with the smallest power consumed by the motor is taken. This is the third and last criterion.

2.4.2 Nash Bargaining solution

The Nash Bargaining solution or shortly the Nash solution was defined by Nash (1950b). It considers the game as a bargaining game where each player requests some portion of the existing good, in this case torque. A bargaining game G is defined by specifying a conflict point $c = (c_1, c_2)$ in addition to the payoff space P, comprising of the two payoff vectors as defined above $u = (u_1, u_2)$. The pair is denoted as (P, c) as in Holler and Illing (2006). The conflict point is the point where both players do not cooperate with each other. There exists a common misinterpretation because the word "conflict" point itself implies a point where both players face a disagreement and hence their payoffs are the worst. A Nash Equilibrium does not present such a worst case and therefore the term conflict point may seem misleading, but for the sake of consistency in the literature, the term will be kept as it is. Since the conflict point in the literature is often taken to be the Nash Equilibrium of the corresponding non-cooperative game, the next subsection deals with an algorithm to find a Nash Equilibrium.

Taking these into account, a Nash Bargaining solution is the vector u^* from the set P:

$$NP^* = max(c_1 - u_1^*)(c_2 - u_2^*) \tag{2.3}$$

so that $u^* = (u_1^*, u_2^*) \in P$ and $u_i^* < c_i$ for $i = 1, 2$, meaning that the Nash solution point u^* payoff is less than the conflict point payoff. The main idea of the Nash solution expressed graphically as in Figure 2.2 is to maximize the product of the differences between the conflict point and the Nash solution point coordinates in 2D.

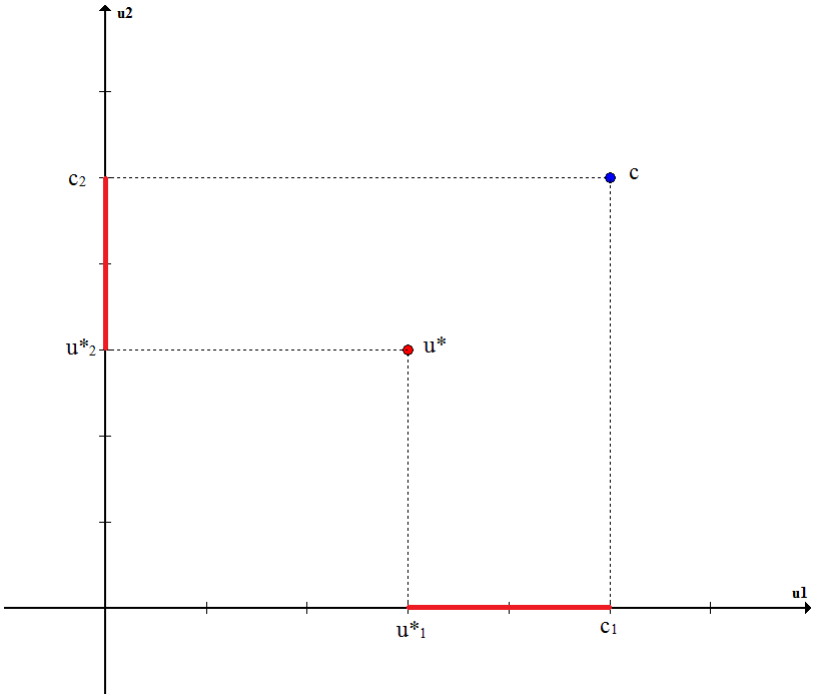

Figure 2.2.: *Maximized Nash Product*

The Nash solution is characterized by four axioms as defined in Holler and Illing (2006).

The first axiom is called Invariance with Respect to Affine Transformations of Utility. Given a bargaining game (P, c) and two random real numbers $a_i > 0$ and b_i, where $i = 1, 2$ for both players, it is true that $f_i(P', c') = a_i f_i(P, c) + b_i$. This holds if (P', c') is a game resulting from a linear transformation which preserves the order of all points u and c from P so that $y_i = a_i x_i + b_i$ and $c_i' = a_i c_i + b_i$, where $y \in P'$ and $c' \in P'$. In other words such a linear transformation of the game space does not affect the solution of the bargaining game.

The second axiom is Symmetry. If (P, c) is a symmetric bargaining game then $f_1(P, c) = f_2(P, c)$. A game is symmetric if $c_1 = c_2$ and if (u_1, u_2) and (u_2, u_1) are both in the payoff space P. This results in a conflict point lying on the 45°line from the origin of the coordinate system. P is symmetric with regard to this line. This axiom tells that if the payoffs of the two players can be interchanged and the game

does not change as a consequence of that, then its solution should also not distinguish between the two players.

The third axiom is Independence of Irrelevant Alternatives. It states that $f(P,c) = f(Q,c)$ if two games (P,c) and (Q,c) have the same conflict point c, $P \subseteq Q$ and $f(Q,c) \in P$. Therefore, only the conflict point is relevant, because if the payoff space of the game is extended to a superset or it is shrunk to a subset, then the solution itself does not change.

The last axiom is the Pareto Optimality. For a bargaining game (P,c) there is no $x \neq f(P,c)$ in P such that $x_1 \leq f_1(P,c)$ and $x_2 \leq f_2(P,c)$, which is an example of group rationality.

2.4.3 Nash Equilibrium

This subsection firstly describes formally the concept of Nash Equilibrium and then presents the Lemke-Howson algorithm for finding one Nash Equilibrium.

The notion of an equilibrium was firstly presented by Nash (1950a) in his one-page paper where he defined an equilibrium as a self-countering n-tuple of strategies. An n-tuple is said to counter another n-tuple if the strategy of each player in the first n-tuple gives him the largest possible payoff against the n-1 strategies of the other players in the second n-tuple (Nash, 1950a).

Let s^* be a strategy combination where each player chooses an optimal strategy s_i^* assuming that all other players have also chosen their optimal strategies. A strategy of a player i is called optimal when i cannot achieve a better payoff given the current decisions (strategies) of the other players denoted as $-i$. Then, a Nash Equilibrium is formally defined as a strategy combination such that (Holler and Illing, 2006):

$$u_i(s_i^*, s_{-i}^*) \leq u_i(s_i, s_{-i}^*),\ \forall i, \forall s_i \in S_i \tag{2.4}$$

Therefore, in an Nash Equilibrium state no player has the incentive to deviate from his strategy assuming that all other players also keep their strategies unaltered. Each player has to guess which strategy his opponent will play and based on that to choose his own best strategy. Nevertheless, there can be several best responses and this means several Nash Equilibria. Nash (1950a) classifies the equilibria into three types - solutions, strong solutions and sub-solutions. It is possible that a non-cooperative

game does not have a solution. However, if it has, then the solution must be unique which is what a strong solution is. A sub-solution is not necessarily unique. Holler and Illing (2006) present a theorem for the existence of a Nash Equilibrium in a game $G(N, S, u)$ with the following characteristics. If the strategy space is compact and convex $S_i \subset R^m, \forall i \in N$ and it is true that $u_i(s)$ is continuous and bounded in $s \in S$ and quasi-concave in s_i, then there exists a Nash Equilibrium. However, a lot of functions do not fulfil these prerequisites for continuity and quasi-concavity and hence have no Nash Equilibrium in pure strategies. In this case the game can be extended to mixed strategies by assigning probabilities to each pure strategy or also called randomization. By randomizing the strategies of matrix games the strategy space S_i can be transformed to a convex and compact space and the payoff $u_i(s)$ to quasi-concave.

2.4.4 Lemke-Howson algorithm

One of the most popular algorithms for finding a Nash Equilibrium for bimatrix non-zero-sum games is the Lemke-Howson algorithm (Lemke and Howson, 1964). The Matlab implementation developed by Katzwer (2014) was utilized to find one Nash Equilibrium in mixed strategies. This function contains a parameter, which affects the final result of the algorithm. Changing the parameter yields different Nash equilibria as output. This parameter k is the initial pivot, a number between 1 and $m+n$, where m and n are the number of strategies of player 1 and player 2 respectively. The general idea of the the algorithm is that it works on two graphs containing nodes and edges, one graph per player. It starts at the (0,0) point. It selects a k, the pivot, or the label of the graph, containing the strategy to be dropped first when traversing the graph. From there a path to the end is followed in order to find a Nash Equilibrium.

There a number of reasons for the different Nash Equilibrium solutions that the algorithm produces. According to Lemke and Howson (1964) there exist an odd number of Nash Equilibria in any non-degenerate game. A non-degenerate game is a game where no mixed strategy with a support of size k has more than k pure strategies (Nisan et al., 2007). Moreover, a support of a mixed strategy is defined as the set of pure strategies with positive probabilities. Since there is an odd number of Nash Equilibria, there must be at least one Equilibrium, which proves that the algorithm will always find one solution in mixed strategies. However, which of all Nash Equilibria is found depends on which strategy label is dropped first. The initial pivot label which

the algorithm drops can belong to either of the two players and be any of their m or n strategies.

As discussed in Shapley (1974) the Lemke-Howson algorithm possesses a significant weakness, namely that it is neither guaranteed that the algorithm will find all possible solutions, nor that it will tell if there are any unfound solutions.

The fundamentals of the Lemke-Howson algorithm are described next. Let us assume a scenario of a 2-player bimatrix game as the one used throughout this thesis, where the players 1 and 2 each have n and m number of pure strategies and their payoffs are in the matrices $A = (a_{ij} : i \in \{1, ..., m\}, j \in \{m+1, ..., m+n\})$ and $B = (b_{ij} : i \in \{1, ..., m\}, j \in \{m+1, ..., m+n\})$ respectively. The mixed strategies are the vectors $s = (s_1, s_2 ..., s_m)$ and $t = (t_{m+1}, t_{m+2}, ..., t_{m+n})$ where $S = \{s \geq 0; \sum_{i=1}^{m} s_i = 1\}$ and $T = \{t \geq 0; \sum_{j=m+1}^{m+n} t_i = 1\}$ are the sets for the mixed strategies spaces. Then, the payoff for player 1 is $\sum_{i=1}^{m} \sum_{j=m+1}^{m+n} a_{ij} s_i t_j$ and the payoff for player 2 is $\sum_{i=1}^{m} \sum_{j=m+1}^{m+n} b_{ij} s_i t_j$. An equilibrium is a pair of strategies (s^*, t^*) satisfying:

$$\sum_{i=1}^{m} \sum_{j=m+1}^{m+n} a_{ij} s_i^* t_j^* = \max_{s \in S} \sum_{i=1}^{m} \sum_{j=m+1}^{m+n} a_{ij} s_i t_j^* \tag{2.5}$$

$$\sum_{i=1}^{m} \sum_{j=m+1}^{m+n} b_{ij} s_i^* t_j^* = \max_{t \in T} \sum_{i=1}^{m} \sum_{j=m+1}^{m+n} b_{ij} s_i^* t_j \tag{2.6}$$

If we also define the sets:

$$\tilde{S} = S \cup \{s \geq 0 : \sum_{i=1}^{m} s_i \leq 1 \text{ and } \prod_{i=1}^{m} s_i = 0\} \tag{2.7}$$

$$\tilde{T} = T \cup \{t \geq 0 : \sum_{j=m+1}^{m+n} t_i \leq 1 \text{ and } \prod_{j=m+1}^{m+n} t_i = 0\} \tag{2.8}$$

this allows us to define closed convex polyhedral regions S^i and S^j which together form $S^k \in \tilde{S}$:

$$S^i = \{s \in \tilde{S} : s_i = 0\} \text{ for } i \in \{1, ..., m\} \tag{2.9}$$

$$S^j = \left\{ s \in S : \sum_{i=1}^{m} b_{ij} s_i = \max_{l \in \{m+1,...,m+n\}} \sum_{i=1}^{m} b_{il} s_i \right\} \text{ for } j \in \{m+1, ..., m+n\} \quad (2.10)$$

S^i contains all $\tilde{S} - S$ and S^j contains the mixed strategies for player 1 and the pure strategy j of player 2 which is his best outcome. S^i and S^j both make S^k and cover the whole set $\tilde{S}$. The same can be applied to define the regions $T^k \in \tilde{T}$:

$$T^i = \left\{ t \in T : \sum_{j=m+1}^{m+n} a_{ij} t_j = \max_{l \in \{1,...,m\}} \sum_{j=m+1}^{m+n} a_{lj} t_j \right\} \text{ for } i \in \{1, ..., m\} \quad (2.11)$$

$$T^j = \{ t \in \tilde{T} : t_j = 0 \} \text{ for } j \in \{m+1, ..., m+n\} \quad (2.12)$$

The Lemke-Howson algorithm represents the strategies of both players in two graphs with nodes and edges. The already mentioned definitions are required in order to define a labelling for the graphs. A labelling of a node consists of all of the labels of all surrounding regions of this node. Let the nonempty label of $s \in \tilde{S}$ be $L'(s) = \{k : s \in S^k\}$ and similarly the nonempty label of $t \in \tilde{T}$ be $L''(t) = \{k : t \in T^k\}$ and the label of the node pair with pure strategies $(s,t) \in \tilde{S} \times \tilde{T}$ be $L(s,t) = L'(s) \cup L''(t)$. A node pair (s,t) is completely labelled whenever $L(s,t) = K$, meaning that it contains the labels for all regions $k \in K$. A node pair is almost completely labelled if $L(s,t) = K - \{k\}$ for some $k \in K$.

Then, as in Shapley (1974) a node pair $(s,t) \in S \times T$ is an equilibrium point of (A,B) if and only if (s,t) is completely labelled. Let the two graphs be $G' \in \tilde{S}$ and $G'' \in \tilde{T}$. Two nodes are adjacent if they are on the two ends of the same edge which means that their labels differ in exactly one element. The set of k almost completely labelled nodes in the graphs and their edges are the disjoint paths of the graph and cycles. The equilibria of the game are always located at the end of these paths. Also, the starting node, by default (0,0) is called an artificial equilibria and it is also located at the end of a path.

The algorithm works by starting at $(s,t) = (0,0) \in G' \times G''$. Then a label k, which is to be dropped from (s,t), is chosen (initial pivot). This k can belong to either n or m. Let this node be (s,t) and its new label (after dropping k) be l. Then, if $l = k$ a Nash Equilibrium is reached. If $l \neq k$, then the algorithm continues by dropping another

label and continuing until a completely labelled node has been reached, which is an equilibrium point.

2.4.5 Kalai-Smorodinsky Bargaining Solution

The Nash Bargaining solution has the significant disadvantage that it is not monotonic. The third of the four axioms defined above Independence of Irrelevant Alternatives was criticized as in Kalai and Smorodinsky (1975). Therefore, Kalai and Smorodinsky (1975) propose another unique bargaining solution called the Kalai-Smorodinsky solution. They replace the third axiom with another axiom for Individual Monotonicity, which the Nash solution does not conform to.

The axiom states that if for two bargaining games (P, c) and (R, c) such that $P \subset R$ the equation $m_i(P) = m_i(R)$ holds for player i, then for player $j \neq i$ it is true that $f_j(R, c) \geq f_j(P, c)$.

$m_i(P, c)$ and $m_i(R, c)$ denote the minimum payoffs for players $i = 1, 2$. They are defined as $m_i(P) = min(u_i|(u_1, u_2) \in P$. The point $m(P) = (m_1(P), m_2(P))$ is called the ideal point of the game, because that is where both players achieve minimum payoffs; hence, the ideal outcome of the game. However, often this point is not feasible. As in the Nash solution, the conflict point is also crucial in the Kalai-Smorodinsky solution. A further term is required and this is the Utility Boundary of the utility space P of a game:

Consequently, the Kalai-Smorodinsky solution is defined as follows:

$$\frac{c_2 - u_2}{c_1 - u_1} = \frac{c_2 - m_2}{c_1 - m_1} \tag{2.13}$$

so that

$$u_i \leq v_i, \frac{c_2 - v_2}{c_1 - v_1} = \frac{c_2 - m_2}{c_1 - m_1} \tag{2.14}$$

where $(v_1, v_2) \in P$. The pair (u_1, u_2) is the Kalai-Smorodinsky solution. As Kalai and Smorodinsky (1975) point out, this solution is graphically represented as the intersection point of the line $L(c, m)$ and the utility (payoff) boundary $H(P)$ as shown in Figure 2.3. $L(c, m)$ is the line connecting the conflict point c and the ideal point m. The utility boundary $H(P)$ or also called the Pareto frontier is defined as the set of all Pareto efficient points. For all random payoffs of one player the boundary gives

the minimal possible payoff of the other player. If the Pareto frontier only contains one Pareto efficient point then the Kalai-Smorodinsky solution is the ideal point itself $u = m(P)$.

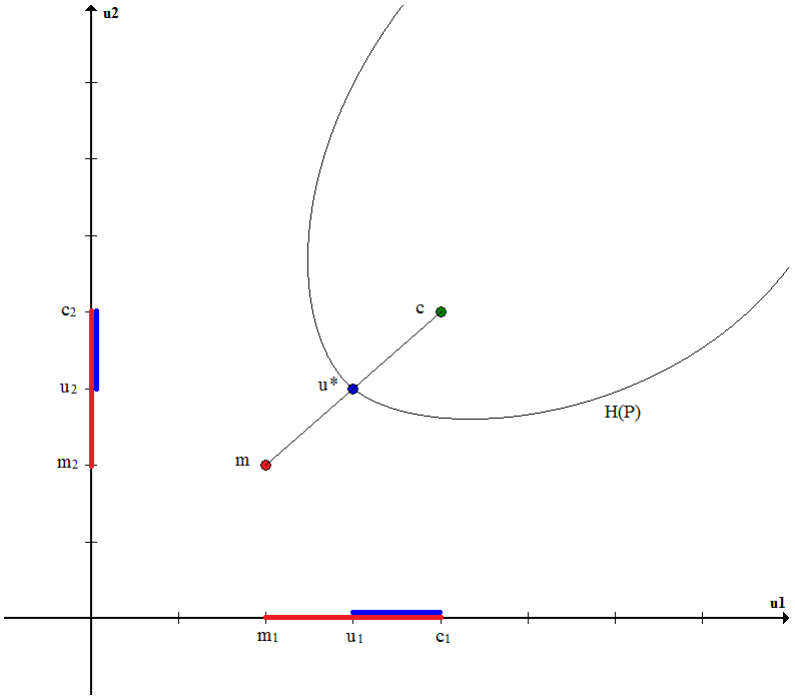

Figure 2.3.: *Kalai-Smorodinsky Solution*

2.4.6 Core

As opposed to the previous four solution approaches, which treated the game as individually cooperative, the next two approaches, the Core and the Shapley value regard the game as coalitional. The difference is that the previous approaches were applicable to 2-player individually cooperative games only, whereas the Core and the Shapley value can additionally be extended to coalitional n-player games. However, this thesis only deals with 2-player games and therefore does not need this extension. Moreover, there is a crucial distinction between the Core and the Shapley value because the Core produces a set of points as a solution, while the Shapley value is always only one single point. For this reason the Core is similar to the Pareto efficiency concept because both can contain more than one point. A single point from this set needs to be chosen and the criteria for choosing a single solution from the Core is the same as in the case of

Pareto efficiency - torque deviation, fuel consumption rate and power consumed by the motor are taken into account.

To define the core of a game, a new term called imputation is introduced. An imputation is such an allocation of resources that is both individually rational and group rational (or Pareto efficient). For transferable utility (TU) games individual rationality means that each player in the coalition receives at most the payoff that they would receive on their own expressed as $u_i \leq v(\{i\})$. Group rationality, which is equivalent to efficiency and in our case also to Pareto efficiency, means that the sum of the payoffs of all players in the grand coalition is the value of the game, $\sum_{i=1}^{N} u_i = v(N)$. For non-transferable utility (NTU) games such as ours a payoff vector u is an imputation when there is no vector u' which strictly dominates it. This is expressed by $u' \in V(N)$ such that $u'_i < u_i, \forall i \in N$. According to Holler and Illing (2006) a vector u' is said to dominate another vector u with regard to the coalition K if $u'_i \leq u_i, \forall i \in K$ and for at least one $i \in K$ the inequality $u'_i < u_i$ holds when $u' \in V(K), u \in V(K)$.

After describing the notion of imputation, the core C of a game G is the defined as the set of all non-dominated imputations:

$$C(G) = \{u \mid v(K) - \sum u_i \leq 0,\ \forall i \in K,\ \forall K \in N\} \tag{2.15}$$

If an imputation x is in the core $x \in C(G)$ then for all coalitions $K \in N$ there is no other coalition K for which another imputation y dominates x denoted as y *dom* x *via* K (Holler and Illing, 2006). Therefore, no coalition can make its participants better by substituting x with y; hence, x is thought to be coalitionally rational. There are two different possibilities when this can be true. There is either no coalition at all which can realize y, or y is worse than x meaning that it is not true that $y_i < x_i$ for at least one $i \in K$ and $y_i \leq x_i$ for all $i \in K$.

Since all imputations in the core are not dominated by any other imputation, the core is internally stable. The core of a game can be empty or can contain many imputations. This comes from the fact that the relation *dom* is not transitive; hence, if it is true that an imputation x dominates y and y dominates z, this does not imply that x dominates z.

2.4.7 Shapley Value

The Shapley value of a game with transferable utility according to Holler and Illing (2006) is defined as:

$$\Phi_i(v) = \sum_{i \in K, K \subset N} \frac{(k-1)!(n-k)!}{n!}[v(K) - v(K - \{i\})], \tag{2.16}$$

$$\sum \Phi_i(v) = v(N),\ \Phi(v) = (\Phi_i(v)) \tag{2.17}$$

k denotes the number of players in coalition K and n is the total number of players. $v(K)$ is the value of coalition K, whereas $v(K-\{i\})$ is the value of coalition K without the player i. The sum of the Shapley values of all players must equal the total value of the game, which is also the value of the grand coalition N.

The Shapley value was introduced by Shapley (1952) where it is described that the game must be represented by its characteristic function $v : 2^N \to \mathbb{R}$. Therefore, to apply the Shapley value in our game which is defined only in its strategic form, the game has to be transformed into its coalitional form in order to get the characteristic function. The procedure is as follows. For each of the three relevant coalitions $\{1\}$, $\{2\}$, $\{1, 2\}$ a value needs to be assigned. $v(N)$ or $v(1, 2)$ is easy to find, because it is the minimum sum of the two payoff matrices A and B. The values of each individual coalition $v(1)$ and $v(2)$ can be found by taking the Saddle point of the corresponding payoff matrix. A Saddle point of a two-player matrix game is both a column maximum and a row minimum. There can be more than one Saddle points in a matrix and they all have the same value. However, if no Saddle point exists in pure strategies, the game can be extended to mixed strategies in order to find a combination of pure strategies which give the value (the Saddle point) of the game. The exact implementation details are explained in 3.2.9.

In addition, the Shapley value is the single point which satisfies the following three axioms. The first axiom is Group Rationality or Efficiency, which is equivalent to Pareto efficiency. This axiom was already described in the subsection of Nash bargaining solution. The second axiom is Symmetry and it states that if the players change their order then the values of the players change accordingly. Therefore, the number, or the i, of the player does not have an influence on the value which that player receives.

The third axiom is based on the assumption $\Phi_i(v+w) = \Phi_i(v) + \Phi_i(w)$ where $v+w$ is the combination of two games v and w.

The general idea of the Shapley value is about permutations of the players and especially which player arrives first, meaning which player is able to make the decision first. The order of the players is irrelevant and all of the various orders have the same probability. Since there are $n!$ permutations of the players, the probability of each of them is $\frac{1}{n!}$ as shown in Equation 2.16. Additionally, the probability of player i being at the k-th place is $\frac{(k-1)!(n-k)!}{n!}$. On the one hand, a player who has an influence on the value of the coalition is called a crucial or pivot player. A player i is a pivot player whenever K is a winning coalition and without him the coalition $K - \{i\}$ is a losing coalition. On the other hand, a player is called a dummy player when a coalition K has the same value regardless whether i is participating in it or not, $[v(K) - v(K - \{i\})] = 0$.

3 Implementation

This chapter explains how the hybrid vehicle, which is similar to the Toyota Prius second generation, was modelled in Simulink. Then the implementation details in Matlab of each of the six game-theoretical approaches are described.

3.1 Hybrid Vehicle Model

The model of the hybrid vehicle is split into different controllers and each of them is responsible for controlling the corresponding part of the hybrid. The chapter describes the Drive Cycle Controller, the Power Controller, the Gasoline Engine Controller, the Electric Motor Controller, the Transmission and the Vehicle Dynamics. The game-theoretical logic is included in the Power Controller, which is the principal controller where the solution is computed. It is shown in the next section.

3.1.1 Drive Cycle Controller

This subsection deals with the demand of speed which comes from the Drive Cycle and how it is transformed to acceleration demand, which corresponds to the acceleration pedal in a vehicle.

Two different drive cycles were simulated. The FTP-75 drive cycle is intended to test light-duty vehicles in the U.S. and measure their gas emissions and fuel economy. It lasts for 1874s, the total distance it covers is 17.77km and the average speed is 34.1 km/h. The drive cycle contains three phases - cold start from 0-504s, transient phase from 505-1369s and hot start from 1370-1874. The last phase is exactly the same as the first phase. Figure 3.1 shows the time and speed of this drive cycle.

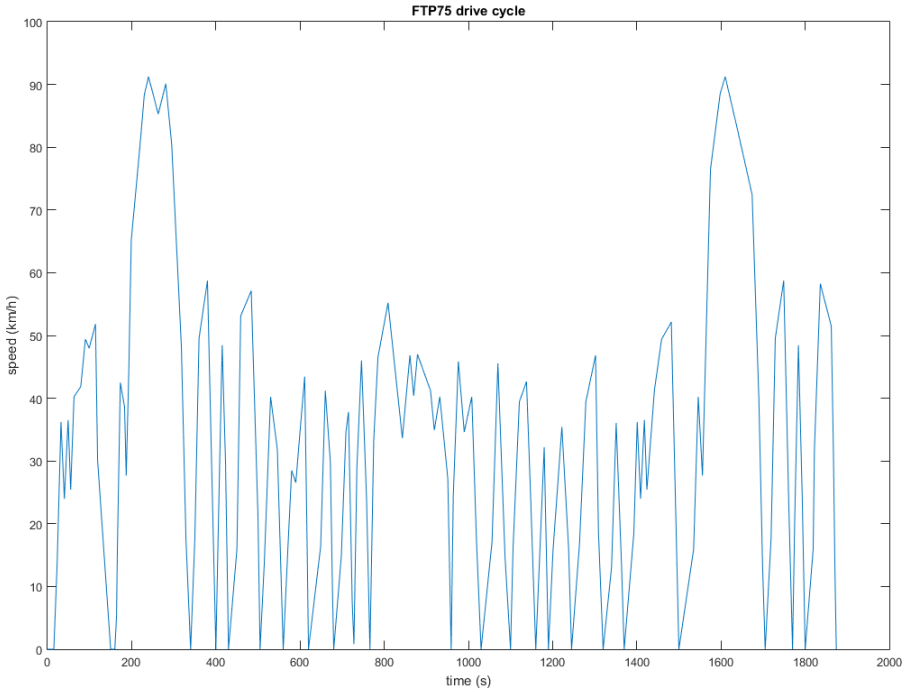

Figure 3.1.: *FTP drive cycle*

The NEDC drive cycle was invented to test light-duty cars in Europe. Its duration is 1180s, the distance is 11.02km and the average speed 33.6km/h. It contains four equivalent urban driving phases called ECE-15. They last from 0-780s and there is also a fourth highway driving phase called EUDC from 781-1180s. Figure 3.2 shows the time versus the speed in the NEDC drive cycle.

The Speed Control block takes as input the demanded speed from the drive cycle and the actual speed of the vehicle both in km/h and calculates as output the acceleration which lies between -1 and 1 where a negative value means braking and a positive value means acceleration. A value of 0 corresponds to cruising or maintaining a constant speed. In order to calculate the acceleration demand, the difference between the demanded and actual speed is taken.

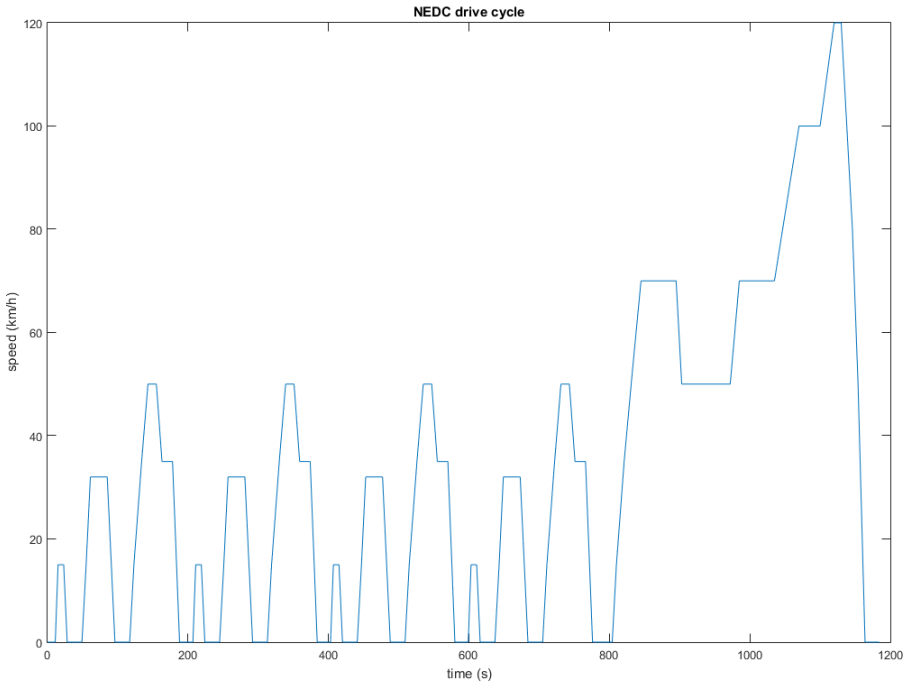

Figure 3.2.: *NEDC drive cycle*

3.1.2 Power Controller

The Power Controller takes as input the acceleration demand a and the state of charge SOC of the battery which comes from the Electric Motor Controller. In order to transform the acceleration demand into torque demand τ_{dem}, the acceleration is scaled by the maximum torque of 400 Nm which can be requested. This value was chosen since this is the maximum output torque of the Toyota Prius electric motor. The minimum demanded torque is -400 Nm which corresponds to deceleration. The current motor speed ω_{mot} is mapped to the torque of the motor which it is capable of generating at this moment. This value is taken as upper limit and its negated value is taken as a lower limit of the torque.

There are two subsystems in the Power Controller, the Battery Management and the Game Theory Controller. The first of them takes as input the battery SOC, its current in A, its voltage in V and computes the required recharge power P_{batReq} and the battery limit P_{batLim}. The recharge power is requested when the SOC of the

battery lies between 40 and 60%. If the SOC falls below 40% then the battery is forced to be recharged even if that means not meeting the drive cycle requirements and not achieving the demanded speed. After the battery has been charged up to 60%, the recharging stops and the requested battery power is set to 0. The battery limit calculation takes into account that the maximum voltage of the battery is 200V and that its maximum power is 21 kW.

The Game Theory Controller is the primary place where the computation of the solution to the power distribution problem happens. The outputted demanded torque τ_{dem} is fed into the Game Theory Controller block. In addition, the other inputs of this block are required power, which is:

$$P = \tau \times \omega \tag{3.1}$$

where power P is in W, torque τ is in Nm and the velocity ω is angular rad/s. The Game Theory Controller also receives the battery SOC in %, the generator ω_{gen}, engine ω_{eng}, motor ω_{mot} speeds all in rad/s, the Total Fuel consumed up to this moment from the beginning of the drive cycle TF in *litres*, the battery limit and the required battery recharge power P_{batReq} as described in the Battery Management subsystem above. The Game Theory Controller block contains the game-theoretical logic embedded as a Matlab function. Apart from it, it also regulates the speed of the engine by taking the difference between the required driving power P_{drive} and the recharge power for the battery P_{batReq}. The resulting power is given as an input to a lookup table to find the corresponding engine speed in *rpm*. To obtain the Generator torque τ_{gen} the engine torque and the generator speed ω_{gen} in rad/s are taken. If the motor needs to provide additional power for the battery, this power is taken to be the sum of the battery available power and the generator power. The battery available power P_{batAv} is computed in a separate subsystem using the engine torque τ_{eng}, speed ω_{eng} and power P_{eng}, battery recharge power P_{batReq} and battery limit P_{batLim}. Four values come as a result from the Game Theory Controller, the engine throttle θ, the motor torque τ_{mot}, the generator torque τ_{gen}, the reference engine speed in *rpm* ω_{eng}. These are fed to a final Speed Controller block for controlling the engine speed which outputs the final engine throttle θ as percentage from the maximum possible torque of the engine τ_{engMax}.

3.1.3 Engine Controller

The Engine Controller receives the throttle in % as input and sends the torque as output. The Gasoline Engine library block implements a gasoline fuel engine and its speed controller. Firstly, the masked library block and then the extensions added to it to will be explained.

In the Gasoline Engine block the input signal between 0 and 1 indirectly controls the engine speed. Whenever the input is outside of the allowed range it is limited to 0 or 1 respectively. The mapping of speed in *rpm* to torque in *Nm* is implemented by a lookup table. The throttle goes through a Switch block which ensures that the maximum speed of the engine is not exceeded. If this happens, the throttle is set to 0. Otherwise the torque and the angular velocity are measured and after that the torque flows to the output. The angular velocity measured in *rad/s* is fed back to the lookup table in *rpm* to find the corresponding torque and it is also given to the switch block for controlling the maximum speed.

An additional functionality which was implemented to extend the library block was to limit the torque from 0 to 136 and also speed from 0 to 6000 rpm. Figure **??** shows the torque speed curve of the engine. It provides maximum torque of 135 to 136 in the speed range 2800 - 3200 rpm. For this reason when the acceleration is constant and the engine provides maximum torque, there is a discrepancy in the simulation results, because the speed keeps jumping from 2800 to 3200 quickly, which is inefficient. Therefore, a hysteresis was implemented by a Relay block which has a switch on and a switch off point values. When the signal lies between 2800 and 3200 rpm it is either set to 2800 or to 3200 rpm to prevent such jittering.

A Driveline environment is connected to the output of the Gasoline Engine block which provides the simulation environment. In addition, attaching a Torsional Spring-Damper (MathWorks, 2016j) allows the torque to flows between two rotating axes, the base and the follower. The input (base) is connected to the output of the Gasoline Engine library block which is the torque, whereas the output (follower) is grounded using an Housing block to prevent it from rotating. The Torsional Spring-Damper takes as parameters the stiffness (spring rate) set to 0 *N*m/rad* and the damping (kinetic friction) set to 0.2079 *N*m*s/rad*. The torque depends on the relative displacement angle and relative angular velocity. Moreover, the outputted torque from the Gasoline Engine block has another connection to an Inertia block, which acts as a rigid rotating body. Torque and angular velocity are measured and given along with the throttle

to a Scope block for displaying the results of the engine. Power is also calculated by taking the product of torque and velocity.

A separate subsystem is responsible for the fuel and gas emissions calculations. It receives the engine speed in *rpm* and the torque and outputs Total Fuel TF in l, Fuel Consumption FC in $l/100km$, CO emissions, HC emissions, NOX emissions all in g/km. In order to calculate them lookup tables are applied, which hold the data for the gas emissions in grams per second g/s. To transform g/s into g/km the total distance travelled is required. That is why the average speed of the drive cycle is calculated and is multiplied with the total time.

3.1.4 Electric Motor Controller

The Electric Motor Controller is a large subsystem containing the battery, a DC/DC converter, the motor and the generator. It receives as input the demanded motor torque and generator torque. It outputs the battery characteristics and the actual motor torque and actual generator torque.

The battery is modelled by a Battery block of a Nickel-Metal-Hydride type. The nominal voltage is 200V and the rated capacity is 6.5Ah. At the beginning of the simulation the battery is fully charged, its state of charge is set to 100%. The discharge characteristics are shown in Figure **??**. The first figure shows the nominal discharge current and the second shows the discharge curves at the specific discharge currents. The discharge model f_1 and charge model f_2 of the battery are (MathWorks, 2016b):

$$f_1(it, i*, i, Exp) = E_0 - K \cdot \frac{Q}{Q - it} \cdot i* - K \cdot \frac{Q}{Q - it} \cdot it + Laplace^{-1}\left(\frac{Exp(s)}{Sel(s)} \cdot 0\right) \tag{3.2}$$

$$f_2(it, i*, i, Exp) = E_0 - K \cdot \frac{Q}{|it| + 0.1 \cdot Q} \cdot i* - K \cdot \frac{Q}{Q - it} \cdot it + Laplace^{-1}\left(\frac{Exp(s)}{Sel(s)} \cdot \frac{1}{s}\right) \tag{3.3}$$

where E_0 is the constant voltage, $Exp(s)$ is the exponential zone dynamics in V, $Sel(s)$ is the battery mode, 0 for discharging and 1 for charging, K is polarization constant

in Ah^{-1}, $i*$ is the low frequency current dynamics in A, i is the battery current in A, it is the extracted battery capacity in Ah, Q is the maximum battery capacity in Ah.

The DC/DC converter is directly attached to the battery. Its purpose is to convert direct current from one voltage to another, from the battery voltage to the motor and generator voltage. The DC bus voltage of the whole electric system is maintained at 500V.

The Electric Motor subsystem is modelled similar to the AC6 - 100 kW Interior Permanent Magnet Synchronous Motor (PMSM) Drive library model (MathWorks, 2016a). The motor is a PMSM as in the Toyota Prius. However, the implemented model in this thesis is a 50 kW instead of 100 kW and the bus voltage is 500VDC instead of 288VDC like in the AC6 library model. In total there are four main parts of the electric motor - the PMSM, the Three-phase Inverter, the VECT controller and the Speed Controller.

The inputs of the Motor block are torque in *Nm*, enable 0 or 1, speed in *rad/s* and the plus and minus from the battery. The outputs are the three motor characteristics - stator current in A, rotor speed in *rad/s*, electromagnetic torque in *Nm* and the speed control containing the reference torque in *Nm*. The Speed controller receives as input the speed and an Enable signal. The incoming torque is converted to speed using a lookup table and fed into the Speed controller. It outputs the torque and transmits it to the VECT controller, whose output are three reference line currents. They are given to the Three-phase converter which converts voltage. The Simulink library block Permanent Magnet Synchronous Machine (MathWorks, 2016f) is used for the implementation of the motor itself. On this motor there are 8 poles and the rotor is of type salient-pole, which means that the magnets are buried. It is a three-phrase motor and has a sinusoidal back electromotive force EMF. The mechanical input can either be the torque *Nm* or the speed in *rad/s*, but in this case it is the speed. Depending on the sign of the torque coming as input the block can either work as a motor or as a generator. When the sign is positive it operates as a motor and if it is negative, it acts as a generator.

The motor's characteristics are plotted in the Electric Motor scope. These include the current in A, the rotor speed *rpm*, the measured and the reference torque *Nm* and also the power in W. The outputted torque from the motor is limited between -400 and 400 *Nm* and is also used as output from the whole Electric Motor Controller.

The Generator block is modelled in the exact same manner as the Motor block. Again the Simulink block Permanent Magnet Synchronous Machine (MathWorks, 2016f) is utilized. Nevertheless, the generator is a 30 kW machine in contrast to the 50 kW motor. The minimum and maximum torque are the same as in the motor, -400 and 400 Nm and are limited within this range using a Saturation block. The generator's resulting stator current in A, rotor speed in rpm, measured and reference torque in Nm and power in W are displayed in the Generator scope.

3.1.5 Transmission

The main purpose of the transmission or the gearbox is to incorporate the torque from the engine, motor and generator and to transmit the combined torque to the vehicle. In the Toyota Prius this transmission system is called the Power Split Device (PSD).

The Transmission subsystem receives as input the torque from the engine, motor and generators. The output is the combined torque of these three which flows into the Vehicle dynamics subsystem where it is transmitted to the wheels. The transmission consists of a planetary gear with a carrier, ring and sun gears. Each of these three gears is connected to one source of torque. The engine is connected to the planetary carrier, the motor is attached to the ring gear and the generator to the sun gear. After merging the torques they are outputted from the ring, where the motor is attached. While the axle of the carrier and thus rotates the ring and sun gears using the inside planet gears, the ring gear axle transmits the torque to the wheels in the Vehicle model. The sun gear's axle, where the generator is attached, is responsible for converting the power of the engine to electrical energy.

The Planetary gear block itself is implemented by the model in MathWorks (2016g). It consists of one planet-planet gear and one ring-planet gear. The parameter ring/sun gear ratio is fixed and is set to 2.6. When the sun rotates in one direction, the ring co-rotates in the other direction with regard to the carrier gear and vice versa. This gear ratio g_{RS} also corresponds to the number of teeth in the ring and sun gear and to the torque ratio between them $g_{RS} = N_R/N_S = \tau_R/\tau_S$. There are two geometry constraints regarding the radii of the gears. The radius of the carrier is the sum of the radius of the sun and planet gears $r_C = r_S + r_P$, whereas the radius of the ring gear is the sum of the carrier radius and the planet radius $r_R = r_C + r_P$. In addition, there are two kinematic constraints which say that $r_C\omega_C = r_S\omega_S + r_P\omega_P$ and $r_R\omega_R = r_C\omega_C + r_P\omega_P$. This means that the angular velocity of the carrier is the

sum of the velocities of the sun and planet gear and the angular velocity of the ring is the sum of the carrier and planet velocities.

In the Transmission subsystem there are three scopes which show simulation results. The first scope is about torque and displays the three torques of the carrier, ring and the sun all in *Nm*. The second scope is the speed scope showing the angular velocity in *rad/s* of the carrier, sun and ring. Lastly, the power scope shows sun, ring and the carrier power in *W* which should be the sum of the ring and sun power.

3.1.6 Vehicle dynamics

The last subsystem models the vehicle dynamics. It receives as input the combined torque which is coming from the motor port, where all three torques of the motor, engine and generator are merged. The torque is measured with a Torque sensor, whose output is connected to a Torsional Spring-Damper. The parameters stiffness and damping are set to 0 *N*m/rad* and 0.1 *N*m*s/rad* respectively. A Simple Gear block (MathWorks, 2016h) transfers the torque. The block is a gearbox consisting of two axes, a base and a follower, which rotate with a fixed ratio of 4.113 in this case. Whenever the sign of the two axes is the same, they rotate in the same direction and when it is different, they rotate in opposite directions. An Inertia block with 0.5 $kg * m^2$ is attached to the torque output of the Simple Gear before it goes to the differential.

The Differential block as in MathWorks (2016c) represents a differential gear which receives a longitudinal axis as input and sends two lateral axes as output. Usually the two lateral axes rotate with distinct angular velocities. A single parameter is necessary for the Differential block and that is the drive gear ratio g_D, in this case 1. It predetermines the relationship $\omega_B = \frac{1}{2} \cdot g_D(\omega_{F1} + \omega_{F2})$ between the longitudinal base axis ω_B and the two lateral axes ω_{F1} and ω_{F2}. Either of the following two cases can be true. Firstly, the longitudinal shaft is the same as the two lateral shafts when they rotate with the same velocity $\omega_{F1} = \omega_{F2}$. Secondly, the longitudinal shaft can be locked, $\omega_B = 0$ when the lateral shafts rotate in the opposite direction $\omega_{F1} = -\omega_{F2}$. The last constraint is that the input and output power of the differential must be the same: $\omega_B \tau_B = \omega_{F1}\tau_{F1} + \omega_{F2}\tau_{F2}$.

Each of the two output shafts of the differential is connected to one inertia block with 0.5 $kg * m^2$. The left and right tires are modelled with the Tire block (MathWorks,

2016i). It requires the following parameters - effective rolling radius of 0.3 m, rated vertical load 3000 N, peak longitudinal force at rated load 3500 N, slip at peak force at rated load 10% and relaxation length at rated load 0.2 m. The inputs of the Tire block are V_x and F_z, where V_x is the wheel longitudinal velocity in m/s and F_z is the vertical load in N. The outputs are Ω, which is the wheel angular velocity expressed in rad/s and F_x is the longitudinal force in N. Figure 3.3 shows how these forces are applied on the wheel.

Figure 3.3.: *Forces affecting the tire (MathWorks, 2016i)*

Each of the two tires is connected to the primary block in the Vehicle Dynamics subsystem. This is the Longitudinal Vehicle Dynamics (MathWorks, 2016e), which implements the dynamics of a vehicle with two axles and four wheels. Its inputs are the incline angle β and the front and rear longitudinal forces F_{xf} and F_{xr}, which come from each of the front and rear tires' outputs F_x. The block produces as outputs the vehicle velocity V_x and also the vertical load forces for the front and rear tires F_{zf} and F_{zr}. These two are transmitted to the F_z input ports of the wheels. In addition, the block calculates the vehicle velocity which is also fed back to the wheels' ports V_x. Before sending the output of the whole Vehicle Dynamics subsystem, the speed is transformed from m/s to km/h.

3.2 Game-theoretical algorithms

This chapter deals with the implementation of all six game-theoretical approaches in Matlab. Firstly, a description of the initialization and the parent function, which calls

each of the solutions and that is embedded in the Simulink model, is given. Then, each function for the game theory solution is explained in detail.

3.2.1 Initialization functions

Before starting the simulation, the environment has to be prepared by initializing variables. The initialization function is called *prepare.m* and it calls the functions for preparing the fuel, exhaust emissions and drive cycle data.

Firstly, the fuel consumption rate lookup table is explained. The data comes from Argonne National Laboratory (1999) where a Toyota Prius first-generation model was tested. Nevertheless, due to lack of any newer publicly available reports which state the results of torque-speed relationship and the corresponding fuel consumption and gas emissions, this data is utilized. The fuel data is read from a file, which contains the engine speed in *rpm*, torque in *Nm*, fuel consumption rate in *gal/s* and engine power in *kW*. A conversion between *gal/s* to *g/s* is done by:

$$gps = \frac{galps * 719.7 * 1000}{264.172} \tag{3.4}$$

where 719.7 * 1000 g/m^3 is the density of gasoline and 264.172 gal/m^3 converts gal to m^3. After the conversions the minimum and maximum speed, torque, fuel consumption rate and power are given to the Matlab *linspace* function in order to generate linearly spaced vectors of size 10 for each of the inputs. Then the function *meshgrid* is applied to produce a grid from the input vectors of torque and speed. The torque and speed are also passed to two *fit* functions, once with the fuel consumption rate and once with the power in order to fit a polynomial surface to the data. Moreover, the fuel and power consumption surface is plotted. For this purpose a polynomial function has to be chosen. Experimentation was done with different polynomial curves like poly22, poly23 and finally poly22 was chosen. The first 2 corresponds to the degree in x and the second 2 or 3 corresponds to the degree in y. Finally, the fuel consumption rate and the power lookup tables are created by the function *feval* which takes the fuel and power fit and evaluates the function at the grid points, returned by the meshgrid function. The resulting 3D plots are shown in Figure **??** and Figure **??**.

The second initialization step is to generate the exhaust emissions lookup tables. The data also comes from Argonne National Laboratory (1999) and is read from a file. Exactly the same procedure with the *linspace*, *meshgrid*, *fit*, *feval* functions like in the

fuel consumption rate above is applied. The resulting fits of the CO, HC and NOX emissions are shown in Figures **??**, **??** and **??**.

Lastly, the final step is to prepare the drive cycle data of FTP75 and NEDC. The FTP75 data is read from a file, which has two columns, the first is the time in seconds and the second is the speed in miles per hour. The *mph* are converted to *km/h* by multiplication with 1,60934. Since the drive cycle's duration is 1874s, it is split into 5 different phases, so that simulation can be simplified in case any errors occur and the whole drive cycle has to be started again. The splitting points were selected so that every phase starts and ends with a demanded speed of 0 *km/h*. The following table shows the duration of each of the five phases:

3.2.2 Embedded Matlab function in Simulink

The Matlab function, incorporated in the Simulink model, is responsible for calculating the payoff functions and afterwards calling all other game-theoretical solutions. The inputs it receives are the required torque in *Nm*, the State of Charge (SOC) of the battery in % and the fuel consumed in *l* at this time step. In addition, it takes two lookup tables, containing the data of the fuel consumption and gas emissions. Any required torque between 0 and 83, which is the minimum torque that the engine can produce, is automatically passed only to the motor. No game has to be played, since the engine is incapable of providing such a small amount of torque. Negative required torque is also directly attributed to the motor only, because the engine is also unable to provide negative torque. During torque demands less than 0, the battery is recharged and the motor is transforming the mechanical energy to electric energy and sending it

Drive cycle phase	Start (s)	End (s)
FT75-1	0	340
FT75-2	341	680
FT75-3	681	1030
FT75-4	1031	1500
FT75-5	1501	1874

Table 3.1.: *Drive Cycle splitting in 5 phases*

to the battery. Therefore, only when the torque demand lies within the range 83-400, a game-theoretical solution is computed.

Strategies computation

A crucial parameter, which influences the performance of the whole embedded function and thus the whole computation time of the simulation, is the number of pure strategies for each player during the game. A value of 7 is chosen to be the number of strategies after experimenting with values between 5 and 15. For 15 strategies the computation times becomes too large and there is not a big difference to the solution with 7 strategies; hence, the smaller number 7 was chosen.

The strategies represent how much torque the engine and the motor contribute and this is expressed as % of the required torque or of the maximum torque they are capable of generating. Therefore, the maximum motor and engine strategies are taken to be the minimum of two values - required torque and maximum engine/motor torque. Then, the engine and the motor strategies are computed by the *linspace* Matlab function, which creates a linearly spaced vector between two values with a specific length, in this case 7. Consequently, 0%, 16.66%, 33.32%, 49.98%, 66.64%, 83.3%, 100% of the required torque are taken as the strategies. For the motor these two minimum and maximum values of the vector are 0 and the required torque respectively, since the motor can produce any torque between 0 and 400, which is also the maximum torque that can be requested. For the engine these two values are 83 (its minimum torque capability) and either the required torque or 136 (maximum torque capability), whichever is smaller. To illustrate this with an example, if the required torque is 120, then the strategies of the motor will be 0, 19.99, 39.98, 59.97, 79.96, 99.95 and 120 Nm. The strategies of the engine will be 83, 89.16, 95.32, 101.48, 107.64, 113.8, 120 Nm.

Payoff computation

The payoff function of the motor is computed as shown in Equation 2.2. Each of the components of the payoff has a different weight. After conducting experiments, the following weights were established: $w_1 = w_{FCR} = 3$ for fuel consumption rate, $w_2 = w_{TD} = 0.5$ for torque deviation, which corresponds to fulfilling the driver demands, $w_3 = w_{HC} = 45$ for HC emissions, $w_4 = w_{CO} = 25$ for CO emissions, $w_5 = w_{NOX} = 65$

for NOX emissions, $w_6 = w_{TF} = 0.5$ for accumulated total fuel consumed to this moment. The payoff function of the engine is also implemented as in Equation 2.1. The respective weights are $w_7 = w_{PC} = 0.5$ for the power consumed by the motor and the $w_8 = w_{SOC} = 1$ is for the battery SOC deviation from the target 70 %.

The payoff matrix of the engine is of size $m \times n$. The calculation of each component in the payoff function is explained in detail next. In order to calculate the fuel consumption rate in g/s a 2D lookup table of size 10×10 is utilized. For a given torque and speed of the engine, it outputs the fuel consumption rate. In order to find the closest value to the engine strategy from all 10 torque values in the lookup table, the minimum absolute difference between all 10 torque values and the engine strategy is taken. To find the engine speed, an interpolation between the torque and speeds values at the specific point of the engine strategy is required. Then, when the torque and the engine speed, corresponding to this torque, are known they are given to the fuel consumption table to find the fuel consumption rate. The torque deviation is computed as the absolute difference between the required torque and the total torque (sum of the motor and engine strategies). After that, the gas emissions are the next components in the payoff function. The same torque and corresponding engine speed are used as inputs to the CO, NOX and HC emissions lookup tables, which work in the same manner as the fuel consumption lookup table.

Similarly, the engine payoff is also a 10×10 matrix. The calculation of the first component in this payoff function, the torque deviation, was explained above. The power consumed by the motor in kW is computed according to:

$$P = \frac{\tau \times v}{9,5488 \cdot 1000} \tag{3.5}$$

where the velocity is in rpm and thus the factor 9,5488 is needed to convert it to rad/s and 1000 is for obtaining the power in kW instead of W. The SOC deviation is computed by taking the absolute difference between the 70 % target, which is the charged capacity that the battery tries to maintain, and the SOC at this time step. This deviation can at most be 30 % and this happens when the battery is fully charged at 100 % or when the battery is charged at its minimum allowed charge state of 40 %. The battery SOC must not fall below this percentage. If this happens, a recharging process immediately starts even if the driver demands cannot be met any more.

With this the description of the payoff function is finished. The next subsection explains the game-theoretical approaches implementation in Matlab.

3.2.3 Pareto Efficiency

Implementation of the Pareto Efficiency is done in the function *paretoset.m*. Firstly, this function receives the payoff of the engine and motor in two variables. All outcomes in both payoff matrices are examined in order. For each payoff it is checked whether there exists another payoff with a smaller engine payoff (x-coordinate) and a smaller motor payoff (y-coordinate). If that is the case, then the current payoff is not a Pareto outcome and a flag is set. If after checking all outcomes against the current outcome there is still no smaller payoff for both players, then this is regarded as a Pareto efficient outcome. Furthermore, it is verified that another outcome with a difference of at most 0.000001 from the current outcome is not already saved in the Pareto efficiency vector. The function outputs the set of Pareto efficient payoffs in a matrix where each row holds the x and y values of the Pareto outcome. In addition, the indices of the Pareto efficient points inside the payoff matrices are returned in the same way.

3.2.4 Pareto Optimality

Due to the fact that there can be more than one Pareto efficient points, a criteria has to be defined in order to choose a single point to be the solution. This single solution or Pareto Optimality is implemented in the function *bestpareto.m*. The function takes as input the outputs of the *paretoset.m* function and additionally the torque deviation, the fuel consumption rate and the motor power. Three different criteria are defined regarding the choice of the solution point. First of all, the point with the smallest absolute torque deviation is taken as the Pareto optimal point. In case that the size of the vector returned by this minimum absolute difference is bigger than 1, or when there are more than one such points, then a second criterion is applied. Thus, the point with the minimum fuel consumption rate is taken. Again, if the size of the returned vector is bigger than 1, a third criterion is exploited. The point with the minimal power, consumed from the motor, is taken as final solution. As output the function returns a pair of x and y coordinates of the Pareto optimal solution payoff.

3.2.5 Nash Equilibrium

The algorithm for finding Nash Equilibrium, the Lemke-Howson algorithm, was explained thoroughly in Subsection 2.4.4. There are various algorithms for computing Nash Equilibria, for example, the *NPG* Matlab function as described in Chatterjee

(2010). However, this function is too slow in comparison with the Lemke-Howson function (Katzwer, 2014). Therefore, it was decided not to use the *NPG* function at all. The Lemke-Howson function takes as input the two payoff matrices of the engine and motor and the initial pivot k. The pivot is the starting label to drop, as already explained in Subsection 2.4.4. Since there are $m + n$ labels (strategies) which can be dropped first and they can end up in a different Nash Equilibrium node in the graph, it is inefficient to experiment with all different starting strategies. Experiments were done with 1, $\frac{m+n}{4}$, $\frac{m+n}{2}$, $\frac{3(m+n)}{4}$ and $m + n$ as the first strategy to be dropped. In order to decide which number to use, the Matlab function NPG is run. The Lemke-Howson solution with the k-th strategy, which is closest to the NPG solution is chosen, $k = \frac{m+n}{2}$ and it is passed as input to the Lemke-Howson algorithm, whose detailed implementation will not be explained. In principle, it is implemented as a complementary pivoting algorithm for bimatrix games. Originally it returns the mixed strategy Nash Equilibrium in a pair of two cell arrays, one for each player's mixed strategy distribution. Nevertheless, cells are not supported in code generation for the Simulink model. Therefore, the algorithm was slightly altered so that it returns the pair of payoffs which represent one Nash Equilibrium in normal arrays instead of cells.

3.2.6 Nash Bargaining Solution

The next solution approach, the Nash bargaining solution is implemented in the function *nashsolution.m*. It takes as input one matrix of size $2 \times (m * n)$, where the first column is the engine payoff, whose matrix is transformed to a vector, and the second columns is the motor payoff. The function also requires as input the conflict point, which is taken to be the Nash Equilibrium point, produced by the Lemke-Howson algorithm. The algorithm works by going over the whole payoff pairs and computing the Nash Product by the formula in Equation 2.3. In the end, the payoff pair with the maximum Nash product is taken to be the Nash bargaining solution. The output of the function is the maximized Nash product and the index of the Nash Solution pair.

3.2.7 Kalai-Smorodinsky Solution

The Kalai-Smorodinsky bargaining solution is implemented in the function *kalais-morodinsky.m*. It takes as arguments the same matrix containing both player's payoffs as column vectors, the conflict point like in the Nash solution, the Pareto payoff

strategies and their indices. It returns the payoff pair of the Kalai-Smorodinsky solution.

Firstly, the function finds the ideal point, which is often an infeasible point because it is the minimum of both payoffs. Since the Kalai-Smorodinsky solution is defined as the intersection point of the ideal and conflict points with the Pareto Frontier line, an interpolation between the ideal and conflict points is required. To interpolate between the two values it must be checked whether there are repeated x or y values which can impede the function *interp1* by causing an error. If there are duplicate x values, then interpolation along the y values is performed and if there are duplicate y values vice versa.

Furthermore, interpolation between all Pareto efficient point has to be done in order to get all points on the line segments connecting the Pareto efficient points. Then, the closest point to the exact intersection point is taken. Since there can be more than two Pareto points, all of them need to be sorted either according to x or to y values. An interpolation from the smaller to the larger coordinates is done by using the function *linspace* and *interp1*. The linearly spaced vector created between any two Pareto points contains 100 points. This number is chosen since a large number is needed in order to accurately find the closest interpolated point to the real point. However, an even larger number than 100 will be too space-consuming and does not necessarily increase the accuracy when the two points' x or y coordinates differ only slightly, for instance, just with 1 or 2. The interpolation is applied to the first and second points, then to the second and third and so on until all Pareto efficient points are connected. Afterwards, the Kalai-Smorodinsky solution is computed by equalizing the ratios as in Equation 2.13. For all points on the line segments, connecting all Pareto efficient payoffs, it is tested whether the two ratios of the x and y coordinates - between the conflict point with the point on the Pareto Frontier and between the conflict point with the ideal point, are the same. Prevention of division by zero is also taken into account. Nevertheless, it can happen that the left and right sides of Equation 2.13 are never equal, because the interpolation has not produced exactly the same point as the real intersection point, but rather a point with a very small difference in the second or third decimal places. Therefore, the difference between the right and left hand-sides of each interpolated point on the Pareto Frontier is computed so that if this case occurs, the point with the minimum absolute difference is taken as the Kalai-Smorodinsky solution.

The function returns the pair of payoffs for the Kalai-Smorodinsky solution. Often the point does not coincide exactly with a payoff pair from the input payoff matrices; hence, the solution is not feasible in pure strategies. In this case the function additionally returns the two Pareto efficient points, between which the Kalai-Smorodinsky solution lies, and their indices. The reason is that these two Pareto points are later utilized to compute a mixed strategies solution.

Whenever randomization between between the two Pareto points by computing mixed strategies is necessary, probabilities are assigned to both Pareto strategies in the following way. It is assumed that the two Pareto points u_1, u_2 are sorted in descending order so that the u_1 is bigger in x and y than u_2. It is further assumed that x coordinates correspond to the engine or player 1 and y coordinates correspond to the motor or player 2. Let us denote the Kalai-Smorodinsky point with ks. There are three cases according to which the probabilities of the two strategies are computed. Firstly, when the x and y coordinates of both Pareto efficient points are different, then the ratio $P(s_1) = \frac{ks(x)-u_1(x)}{u_2(x)-u_1(x)}$ is taken to be the probability of u_1's strategy s_1. The probability of p_2's strategy s_2 is similarly $P(s_2) = \frac{ks(y)-u_2(y)}{u_1(y)-u_2(y)}$. Secondly, when the x coordinates of u_1 and u_2 are the same, then the ratios of the y coordinates are taken. For the second strategy the probability is again $P(s_2) = \frac{ks(y)-u_2(y)}{u_1(y)-u_2(y)}$ and for the first strategy it is $P(s_1) = 1 - P(s_2)$. Thirdly, when the y coordinates of u_1 and p_2 are the same, the ratios of the x coordinates are computed. Therefore, $P(s_1) = \frac{ks(x)-u_1(x)}{u_2(x)-u_1(x)}$ and $P(s_2) = 1 - P(s_1)$.

The payoffs of the two mixed strategies are computed as follows:

$$s_1 = u_1(x) + P(s_1)(u_2(x) - u_1(x)) \tag{3.6}$$

$$s_2 = min(u_1(y), u_2(y)) + P(s_2)|u_1(y) - u_2(y)| \tag{3.7}$$

In the end the torques of the engine τ_1 and the motor τ_2 corresponding to the strategy payoffs are calculated as follows:

$$\tau_1 = min(s_1(x), s_2(x)) + P(s_1)|s_1(x) - s_2(x)| \tag{3.8}$$

$$\tau_2 = min(s_1(y), s_2(y)) + P(s_1)|s_1(y) - s_2(y)| \tag{3.9}$$

3.2.8 The Core

Computation of the core requires finding all imputations. For this reason coalition payoff matrices have to be calculated and this happens in the function *coalitionpayoffs.m*. It receives the payoffs of the engine and motor, the torque deviation δ_{tau} and a linearly spaced vector τ from 0 to the required torque, which is of size 10. For instance, if the required torque is 108, then the vector $\tau = \{0, 12, 24, 36, 48, 60, 72, 84, 96, 108\}$. The idea of coalition payoffs is to modify the original payoff matrices so that the smaller the torque deviation is, the smaller the modified payoff becomes and therefore the more optimal. This is achieved by multiplication with the so called coalition coefficients, which vary from 0.99 to 1 within steps of 0.001. Therefore, torque deviation of 0 *Nm* corresponds to the smallest coalition coefficient of 0.99. Negative torque deviation from $-\tau(2) < \delta_\tau < 0$ has coefficient 0.991, from $0 < \delta_\tau < \tau(2)$ has 0.992 and so on until $\tau(9) < \delta_\tau < \tau(10)$ where it is 1.0. After the coalition payoffs are computed, the engine and motor payoffs are multiplied with the coefficients and the two resulting matrices are added to get the final grand coalition payoffs. The function *coalitionpayoffs.m* returns the modified engine, motor payoffs, the coalition coefficients and the grand coalition payoff.

The next step for computing the core is to extract the imputations from all allocations. The function *checkimputation.m* takes as arguments the original payoffs of the engine and motor, the modified engine and motor payoffs as column vectors, the grand coalition payoffs matrix and in addition the Pareto efficient payoffs. Since an imputation is always Pareto efficient when it fulfils the condition for Group Rationality as described in 2.4.6, it is sufficient to only extract the imputations from the Pareto efficient allocations. All Pareto efficient payoffs are examined and for each of them it is verified that no other Pareto payoffs dominates it and that it is individually rational. Domination means that there is no other payoff that is smaller and individual rationality is conformed to when the player receives a smaller payoff in the coalition than when they act alone. The function returns the imputation payoffs and their indices.

In the end, the function for computing the core *core.m* can be called with imputation payoff allocation. The function goes over all imputations and checks whether both players achieve a smaller payoff in the coalition than on their own like in Equation 2.15. If this is the case, then the imputation also belongs to the core of the game. As output the function returns the indices of the core allocations.

3.2.9 Shapley Value

The last game-theoretical approach to be implemented, the Shapley Value, requires an additional transformation of the game space. Although all previous solution approaches work with the strategic form of the game, the Shapley Value cannot be computed solely from the strategic form, but rather calls for a coalitional form. The core also demanded the value of the coalitions, but these values are in matrix form and the coalition payoffs were computed by element-wise multiplication of the payoff matrix of the engine and motor with the coalition coefficients. In addition, for the grand coalition the sum of the two new engine and motor coalition payoff matrices was taken. However, these matrix forms are not sufficient for the Shapley Value. Rather, a single value is needed for each of the three coalitions - player one coalition, player two coalition and the grand coalition, denoted as $v(\{1\})$, $v(\{2\})$ and $v(\{1,2\})$ respectively. The value of the grand coalition is already available, because it is the minimum sum of both players' payoff matrices $v(\{1,2\}) = min(A+B)$. However, the first two values of the individual coalitions are more complex to find, because it is possible that there is no solution in pure strategies and hence no value. It is essential that the values of the two resulting games are computed. In the first game the coalition is individual and includes only the first player, the engine, while in the second game the individual coalition contains only the second player, the motor. That is why the strategic form of the game has to be transformed to coalitional form by finding the characteristic function. According to Ferguson (2014) the value of the resulting game is equivalent to the Saddle point as described in 2.4.7. When no Saddle point exists, thus no solution in pure strategies, mixed strategies have to be computed.

The function which computes the value of the two individual coalitions is *value.m* and takes as input the payoff of one player and the number of that player. The output is a vector with probabilities for each strategy of size m for player 1 or n for player 2. The function is called once for the engine and once for the motor. Firstly, it is tested whether there exists a Saddle point by checking if there is a payoff u_i which is simultaneously a row minimum and a column maximum: $min(max(u_i)) == max(min(u_i'))$. If there is such a payoff point, then in the vector this strategy is marked with probability 1 and all other probabilities are 0. Otherwise, mixed strategies have to be employed.

Extending the strategy space to mixed strategies involves assigning probabilities to every pure strategy. This problem can be solved by a system of linear equations

such that if the engine payoff matrix is the one shown in Table 2.1 and the motor payoff matrix is in Table 2.2, then the linear system of equations is constructed in the following way. The system can be solved by linear programming (LP) which is a method for minimizing or maximizing a linear objective function, depending on inequality and equality constraints. Furthermore, the function is constrained subject to lower and upper bounds. The Matlab function is called *linprog* and is invoked by $x = linprog(f,A,b,Aeq,beq,lb,ub)$. The purpose is to minimize the function f subject to the constraints:

$$\min_x f^T x; \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub \end{cases} \tag{3.10}$$

The first argument f is the function to be minimized. A is the matrix containing the left side of the system of linear inequalities, whereas b is the right hand-side of this system. Generally, Aeq is also a matrix with the left hand-side of the system of linear equations and beq is the corresponding left-hand side of the system. Since in our case we only have one equation, Aeq is a vector and beq is a scalar. Moreover, as optional arguments lower an upper bounds can be specified which are the vectors lb and ub. The last column of A is added to fulfil the constraint that the probabilities of all pure strategies have to sum up to 1. To summarize, only A is a matrix, as shown in Table 3.2 and all other arguments are vectors except for beq which is a scalar.

$f = [0, 0, 0, 0, 0, 0, 0, -1]$
$lb = [0, 0, 0, 0, 0, 0, 0, -Inf]$
$b = [0, 0, 0, 0, 0, 0, 0]$
$Aeq = [1, 1, 1, 1, 1, 1, 1, 0]$
$beq = 1$

The function produces a vector X containing the solution of the linear system of equations. The vector has length $m + 1$ or $n + 1$, the number of strategies, depending on whether the function is called for player 1 or player 2. The first m or n numbers of the vector contain the probabilities of each strategy and they must all add up to 1. The last number in the vector is the solution of the last equation where the column is with all of the 1's as in 3.2. That is also the value of the game, corresponding to the individual coalition formed from the player. After the function *value.m* is ran two

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$	1
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$	1
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$	1
$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$	1
$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$	1
$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$	1
$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$	1

Table 3.2.: *matrix A - left hand-side of system of inequalities*

times for both players, the values of the games are finally available and can be passed to the Shapley value function.

The function *shapleyvalue.m* takes three arguments - the value of the grand coalition, the value of the coalition of player 1 only and the value of the coalition of player 2 only, written as $v_{1,2}$, v_1, v_2. Then, according to Equation 2.16 the Shapley value of the engine is $\frac{1}{2}(v_{1,2} - v_2) + \frac{1}{2}v_1$ since there is one player in the coalition, $k = 1$ and the total number of players is two, $n = 2$. Similarly, the Shapley value of the motor is $\frac{1}{2}(v_{1,2} - v_1) + \frac{1}{2}v_2$.

Although the two Shapley values are computed, it is often the case that the paoyff they constitute is not feasible in pure strategies. When this occurs, two payoffs or two strategy combinations t_1, t_2 have to be found, whose combination in mixed strategies gives the Shapley value payoff s. In order to find the two nearest payoff points u_1 and u_2, the Matlab function *knnsearch* is utilized to find the K nearest neighbours of the Shapley value, where K is set to 2. Let a variable $i = \{1, 2\}$ take values of either 1 or 2. If the two x coordinates of the two nearest payoffs are different, then the x values can be compared or $i = 1$, otherwise the y values are compared and $i = 2$. There are three possibilities for the Shapley value, either it lies within the range $s_2(i) < s < s_1(i)$ or within $s_1(i) < s < s_2(i)$ or it is smaller than both of them, $s < p_1(i), s < p_2(i)$. In the first case the probability of the first closest payoff is $P(u_1) = \frac{s-u_2(i)}{u_1(i)-u_2(i)}$. In the second case it is $P(u_1) = \frac{s-u_1(i)}{u_2(i)-u_1(i)}$. In the third case the probability depends on whether u_1 or u_2 is bigger. When $u_1 < u_2$ the probabilities are $P(u_1) = \frac{s}{u_1(i)}$ $P(u_2) = 1 - P(u_1)$ and vice versa for $u_2 < u_1$ they are $P(u_1) = \frac{s}{u_2(i)}$ $P(u_2) = 1 - P(u_1)$. The final step

is to compute the engine and motor torque by combining the probabilities of the two closest strategies.

With this implementation description the last solution approach, the Shapley value, the Implementation chapter is finished. The next chapter deals with the Simulation of the whole game-theoretical behaviour.

4 Simulation

The following chapter extends the description of the Implementation chapter by providing the specific parameters for the simulation. Firstly, the behaviour during the FTP75 drive cycle is described.

4.1 FTP75 drive cycle

As already explained in subsection 3.2.1 the FTP75 drive cycle is divided into 5 parts with similar duration and with start and end speed demand of 0 km/h. A certain number of parameters need to explained, which are utilized in the Simulink model and need to be tuned so that no errors occur.

Table 4.1 shows the parameters about time which concern the simulation. The block is called Environment block for Simscape Power Systems Specialized Technology as in (MathWorks, 2016d) and offers a choice of Discrete or a Continuous solver. The first one solves the circuit with a variable-time solver and the second with a fixed-time solver. It is advisable that the powergui block is placed at the top of the Simulink model. For the solver type there are three options, Tustin, Backward Euler (TBE) or a combination of both. They are used for the discretization of the model.

Table 4.2 shows the configuration parameters for the Solver. The first one is Type, which can be either Variable-step or Fixed-step. On the one hand, a variable-sized solver slows down the simulation by decreasing the step size when a lot of changes happen in the model for a short period of time; thus, keeping the accuracy. On the other hand, it increase the step size when less alterations in the model occur; hence, preventing redundant steps. If the solver is of fixed-step size the step size is not changed during the simulation. However, the advantage of variable-size solvers is that

Name(s)	Value(s)
Simulation type	Discrete
Solver type	Tustin
Sample time (s)	6e-05

Table 4.1.: *powergui simulation parameters*

it is capable of achieving similar accuracy by reducing the total computation time because it adapts to the changes in the model over time.

Another essential topic is the choice of the Solver for the ordinary differential equation (ODE). A specific issue in ODEs is that they can have one or more derivatives of a variable which is dependent on time. Time is independent t and the dependent variable is y. In this case the solver is ode23tb and it is appropriate and quicker for big error tolerances.

Name(s)	Value(s)
Solver-type	Variable-step
Solver	ode23tb
Relative tolerance	0.01
Absolute tolerance	0.001
Min step size	auto
Initial step size	auto
Number of consecutive min steps	100000000
Zero-crossing control	Disable All

Table 4.2.

5 Results

6 Discussion

Bibliography

Argonne National Laboratory (1999). *Performance and Emissions of The Toyota Prius*. URL: http://www.selidori.com/tech/00000-04999/472-9tdlU.pdf (visited on 10/06/2016).

Burch, Steve, Matt Cuddy and T Markel (1999). 'ADVISOR 2.1 Documentation'. In: *National Renewable Energy Laboratory*.

Chatterjee, Bapi (2010). *n-person game*. URL: http://www.mathworks.com/matlabcentral/fileexchange/27837-n-person-game/content/npg/npg.m (visited on 12/05/2016).

Chen, H et al. (2014). 'Game-theoretic approach for complete vehicle energy management'. In: *Vehicle Power and Propulsion Conference (VPPC), 2014 IEEE*. IEEE, pp. 1–6.

Chin, Hubert H and Ayat A Jafari (2010). 'Design of power controller for hybrid vehicle'. In: *System Theory (SSST), 2010 42nd Southeastern Symposium on*. IEEE, pp. 165–170.

Dextreit, Clement and Ilya V Kolmanovsky (2014). 'Game theory controller for hybrid electric vehicles'. In: *Control Systems Technology, IEEE Transactions on* 22.2, pp. 652–663.

Ferguson, Thomas (2014). 'Game theory'. In: 2nd ed. Vol. 2014. Chap. 4. Games in Coalitional Form.

Gielniak, Michael J and Z John Shen (2004). 'Power management strategy based on game theory for fuel cell hybrid electric vehicles'. In: *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*. Vol. 6. IEEE, pp. 4422–4426.

Holler, Manfred J and Gerhard Illing (2006). *Einfuehrung in die Spieltheorie*. Springer-Verlag.

Kalai, Ehud and Meir Smorodinsky (1975). 'Other solutions to Nash's bargaining problem'. In: *Econometrica: Journal of the Econometric Society*, pp. 513–518.

Katzwer, Richard (2014). *Lemke-Howson Algorithm for 2-Player Games*. URL: http://www.mathworks.com/matlabcentral/fileexchange/44279-lemke-howson-algorithm-for-2-player-games (visited on 08/03/2016).

LaFrance, RC and RW Schult (1973). 'Electrical systems for hybrid vehicles'. In: *Vehicular Technology, IEEE Transactions on* 22.1, pp. 13–19.

Lemke, Carlton E and Joseph T Howson Jr (1964). 'Equilibrium points of bimatrix games'. In: *Journal of the Society for Industrial and Applied Mathematics* 12.2, pp. 413–423.

MathWorks (2016a). *AC6 - 100 kW Interior Permanent Magnet Synchronous Motor Drive*. URL: http://de.mathworks.com/help/physmod/sps/examples/ac6-100-kw-interior-permanent-magnet-synchronous-motor-drive.html (visited on 06/06/2016).

– (2016b). *Battery*. URL: http://de.mathworks.com/help/physmod/sps/powersys/ref/battery.html (visited on 03/06/2016).

– (2016c). *Differential*. URL: http://de.mathworks.com/help/physmod/sdl/drive/differential.html (visited on 08/06/2016).

– (2016d). *Environment block for Simscape Power Systems Specialized Technology models*. URL: http://de.mathworks.com/help/physmod/sps/powersys/ref/powergui.html (visited on 21/06/2016).

– (2016e). *Longitudinal Vehicle Dynamics*. URL: http://de.mathworks.com/help/physmod/sdl/drive/longitudinalvehicledynamics.html (visited on 08/06/2016).

– (2016f). *Permanent Magnet Synchronous Machine*. URL: http://de.mathworks.com/help/physmod/sps/powersys/ref/permanentmagnetsynchronousmachine.html (visited on 06/06/2016).

– (2016g). *Planetary Gear*. URL: http://de.mathworks.com/help/physmod/sdl/drive/planetarygear.html (visited on 07/06/2016).

– (2016h). *Simple Gear*. URL: http://de.mathworks.com/help/physmod/sdl/drive/simplegear.html (visited on 08/06/2016).

– (2016i). *Tire*. URL: http://de.mathworks.com/help/physmod/sdl/drive/tire.html (visited on 08/06/2016).

– (2016j). *Torsional Spring-Damper*. URL: http://de.mathworks.com/help/physmod/sdl/drive/torsionalspringdamper.html (visited on 08/06/2016).

Nash, John F (1950a). 'Equilibrium points in n-person games'. In: *Proceedings of the National Academy of Sciences of the United States of America* 36, pp. 48–49.

– (1950b). 'The bargaining problem'. In: *Econometrica: Journal of the Econometric Society*, pp. 155–162.

– (1951). 'Non-cooperative games'. In: *Annals of mathematics*, pp. 286–295.

Nisan, Noam et al. (2007). *Algorithmic game theory*. Vol. 1. Cambridge University Press Cambridge.

Shapley, Lloyd S (1952). *A value for n-person games*. Tech. rep. DTIC Document.

– (1974). *A note on the Lemke-Howson algorithm*. Springer.

Von Stackelberg, Heinrich (1952). *The theory of the market economy*. Oxford University Press.

Wikipedia (2006). *Epicyclic gearing*. URL: https://upload.wikimedia.org/wikipedia/commons/d/d5/Epicyclic_gear_ratios.png (visited on 03/06/2016).

Appendix A