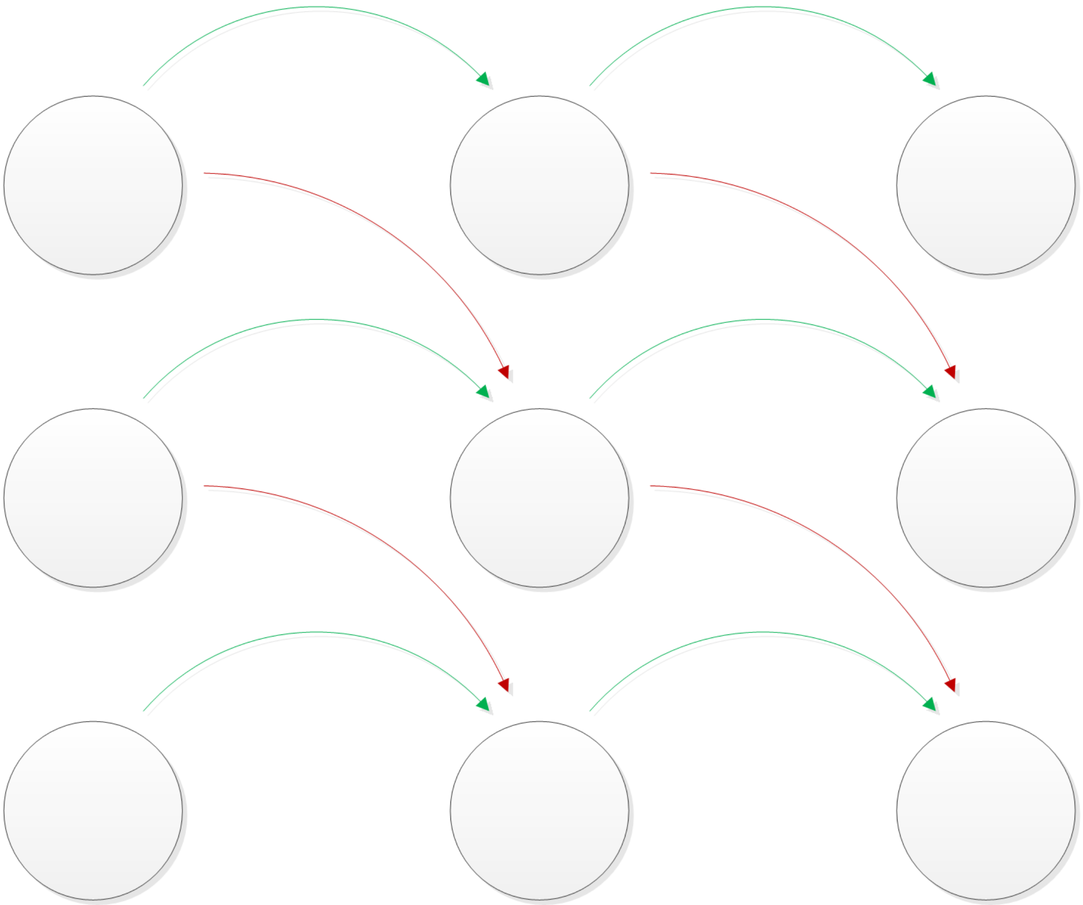
CHALMERS



Optimizing HEV Fuel Efficiency Using Cloud Stored Data

Exploring Use Of Reinforcement Learning In A Hybrid Vehicle Context

Master's thesis in Computer Science and Automotive Engineering

NIKLAS ÅKERBLOM
PATRIK NORDAHL

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2013
Master's thesis EX 025/2013

MASTER'S THESIS IN COMPUTER SCIENCE AND AUTOMOTIVE ENGINEERING

Optimizing HEV Fuel Efficiency Using Cloud Stored Data

Exploring Use Of Reinforcement Learning In A Hybrid Vehicle Context

NIKLAS ÅKERBLOM
PATRIK NORDAHL

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2013

Optimizing HEV Fuel Efficiency Using Cloud Stored Data
Exploring Use Of Reinforcement Learning In A Hybrid Vehicle Context
NIKLAS ÅKERBLOM
PATRIK NORDAHL

Master's thesis EX 025/2013
ISSN 1652-8557
Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Principle of the Markov Decision Process (MDP) as implemented in this thesis work

Chalmers Reproservice
Göteborg, Sweden 2013

Optimizing HEV Fuel Efficiency Using Cloud Stored Data
Exploring Use Of Reinforcement Learning In A Hybrid Vehicle Context
Master's thesis in Computer Science and Automotive Engineering
NIKLAS ÅKERBLOM
PATRIK NORDAHL
Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology

Abstract

A significant part of greenhouse gas emissions can be attributed to exhaust from vehicles using fossil fuels for propulsion. In addition, since the global supplies of fossil fuels are finite, a transition to more sustainable energy sources for transportation is desirable. In recent years, a step on the path to that goal has been to gradually reduce the dependency of such fuels by combining combustion engines with electric propulsion systems in hybrid vehicles.

Using a specific hybrid electric vehicle as starting point, the purpose of this thesis is to investigate a method to reduce the fuel consumption by using logged data. This is accomplished by building a system that applies reinforcement learning algorithms on driving statistics collected from that vehicle. To increase the usability and scope of the method, computation is done remotely using cloud services.

The developed system consists of several parts. An application in the vehicle records data over a route and sends it to a database on a cloud service. A server containing a vehicle simulation augmented with the SARSA(λ) reinforcement algorithm has access to that data. By iterating the algorithm on the vehicle simulation, a control strategy for the combustion engine is generated. Strategies can finally be sent back to the application in the vehicle for usage.

Results from simulations indicate that it is possible to decrease the fuel consumption compared to the static strategy used as reference in the tests. They also show that the strategy achieves better results on average over time than a strategy making uniformly random choices. This indicates that the algorithm can be used successfully in this domain.

Keywords: Reinforcement learning, Hybrid electric vehicles, Cloud programming, SARSA(λ)

Preface

The project behind this master's thesis was carried out at Volvo Car Corporation which provided us with all necessary equipment and support. We would like to direct a special thanks to our supervisor Rickard Arvidsson who has been very inspiring and supportive. We would also like to thank our examiner Professor Bo Egardt at Chalmers University of Technology for taking the time to support us and provide us with good advice through the project.

Nomenclature

α	Learning rate.
γ	Discount factor.
λ	Eligibility decay rate.
a	Action.
$e(s, a)$	Eligibility value in a state for a certain action.
$Q(s, a)$	Expected value in a state for a certain action.
r	Immediate reward.
s	State.
T	Boltzmann temperature.
$V(s)$	Value function for a certain state.
ACU	Accessory control unit, facilitates driver control of vehicle accessories.
CAN	Controller area network, a network standard commonly used for internal computer communication in a vehicle.
ECM	Engine control module, the control computer for the engine of a vehicle.
Ethernet	A technology used for local area computer networks.
GUI	Graphical user interface, a graphical way for a user to interact with a computer application.
HEV	Hybrid electric vehicle, vehicle with two or more on-board stored energy sources of which at least one is electric.
ICE	Internal combustion engine, engine where the combustion takes place in the working fluid circuit as in a conventional vehicle.
ISG	Integrated starter generator, an electric machine connected to an engine used both as starter and generator.
JSON	JavaScript object notation, a way to represent JavaScript data structures as human-readable strings
Markov property	When future states are only dependent on actions taken in the current state.
MDP	Markov decision process, describes a decision problem as a stochastic process fulfilling the Markov property.
PHEV	Plug-in hybrid electric vehicle, HEV that can be charged from the grid.
PID controller	Proportional-integral-derivative controller, used for control according to some desired value.
Q-learning	Reinforcement learning algorithm based on TD-learning.
Q-matrix	Matrix containing all the Q-values of an MDP.
Q-value	Same as Q(s,a).
SARSA	State-action-reward-state-action, a more explorative version of Q-learning.
SARSA(λ)	Like SARSA, but with eligibility traces to account for system with delayed reward.
SOC	State of charge in percent for the battery.
RMS	Root mean square, a measure of the size of variations in a signal.

SQL	Structured query language, a common query language for relational databases.
TCP/IP	The stack of protocols used for communication over the Internet.
TD-learning	Temporal difference learning, achieved through observing a system and adjusting previous estimates accordingly.
URL	Uniform resource locator, another word for Internet address.
VCC	Volvo Car Corporation.
XML	Extensible markup language, a format which is used extensively for document and data encoding.

Contents

1 Introduction

This thesis is the result of a master's degree project that was conducted at the research and development unit of Volvo Car Corporation (VCC). The following sections aim at introducing the topic of this Master's thesis as well as specifying the problem and the delimitations.

1.1 Background

Due to the prospect of impending global climate change and the finite availability of fossil fuels, reductions in the use of these energy sources for cars is of the utmost importance. As a consequence of this, it is of interest to develop more fuel efficient vehicles. The *hybrid electric vehicle* (HEV) is already a topic of intensive research for vehicle manufacturers and multiple approaches are possible. An HEV has both a combustion engine and an electric motor, and the idea is to use the combustion engine only when necessary, in order to minimize fuel consumption.

For hybrid vehicles in general it is important to take into account when and how to charge, discharge or sustain the stored energy in the battery. Therefore, strategies have to be developed for solving this control issue of the relationship between the combustion engine, electric motor and battery. Traditionally this has been done by static policies independent of drive cycle. Through recent advances in the computational power of computers and increases in general Internet connectivity, better solutions are possible.

1.2 Purpose

The main objective with this thesis is to describe a way to possibly decrease the fuel consumption of a series HEV. This is done by using a *reinforcement learning* algorithm for control of the on/off state for the *internal combustion engine* (ICE) on known routes. That is to say, an attempt is made to solve a part of the control issue mentioned in the previous section. The algorithm makes decisions based on logged data from previous runs along the same routes. A software system is created that accomplishes this task by offloading computation to a remote Internet server. The system also enables the vehicle to make use of the generated results.

1.3 Problem description

In a series HEV, the electric machinery is used for propulsion and a combustion engine connected to an electric machine is used for generating electricity. The electrical energy is either used directly or stored in batteries. When the *state of charge* (SOC) is low or when the demanded power is greater than the amount the batteries can provide, the ICE is started to either recharge the batteries or deliver the extra power needed. As mentioned previously, this is usually done in a static way, so an adaptive strategy could possibly be better from a fuel efficiency perspective. The approach of the thesis work can be embodied by the following tasks:

- Develop software to log driving statistics from a series HEV and upload it to a remote server on the fly.
- Use reinforcement learning algorithms together with a Matlab model of the HEV to improve the strategy for ICE on/off state control.
- Attempt to achieve higher fuel efficiency for an HEV by downloading and using aforementioned strategy while driving.

1.4 Delimitations

The availability of vehicles on which to test the solution is limited, so by necessity only a single type of series HEV is considered. Due to time constraints for the project, the algorithm is limited to just controlling ICE on/off state. For the same reason, route detection is determined to be outside the scope and only one specific route is handled.

2 Preliminaries

This chapter consists of a foundation on which the system presented in the report is designed. Subjects related to the vehicle as well as the optimization procedure are included.

2.1 Hybrid electric vehicles

A hybrid vehicle is characterized by two or more prime movers and power sources (Guzzella and Sciarretta 2007). For an HEV, at least one electric machine can be used for propulsion and electric energy can be stored on-board in an electric energy storage system, usually a battery or a super capacitor. A common hybrid powertrain configuration combines a conventional ICE propulsion system with an electric propulsion system. An HEV can often be charged directly from the grid and is then called a *plug-in hybrid electric vehicle* (PHEV).

Providing that the electricity has been generated without the use of fossil fuels, electric propulsion of vehicles means lower overall CO_2 emissions than using conventional propulsion systems. An additional effect is highly reduced contribution to local air pollution around the vehicle (Electric Power Research Institute 2007).

2.1.1 Series hybrid

In a series hybrid configuration (Figure 2.1.1), the vehicle is propelled only by the electric motor. The electrical energy can be supplied both from an engine-driven generator, which converts fuel energy to electric energy, as well as from a battery (Guzzella and Sciarretta 2007). By using the traction motor as a generator, it is possible to regenerate electrical energy to the battery during braking. Since the electric motor is capable of delivering torque at zero rotational speed and the ICE is not mechanically connected to the wheels, no clutch is needed and a simple one speed gearbox can be used between the electric motor and driven wheels. In total, three machines are needed for the series hybrid powertrain; an ICE, a generator and an electric motor.

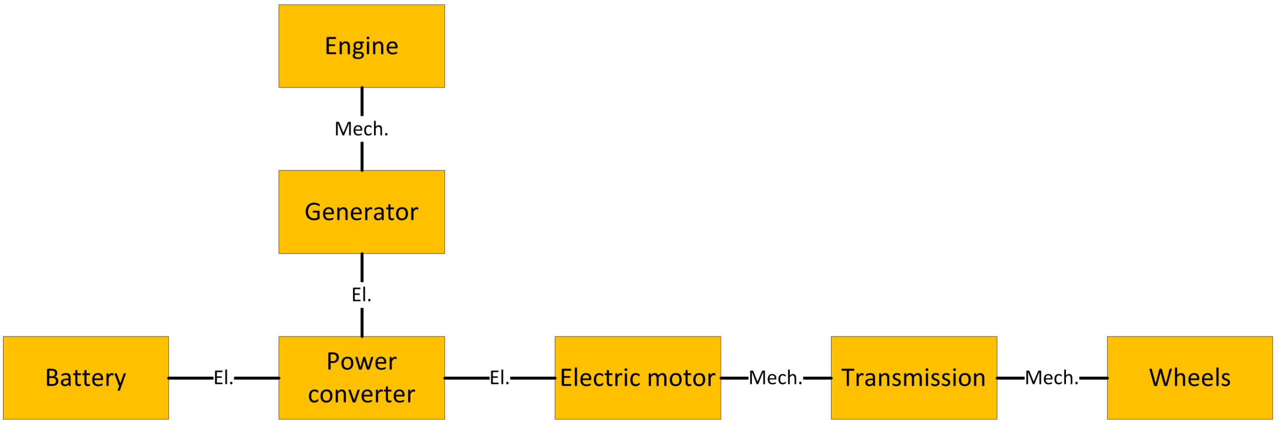


Figure 2.1.1: *Schematic view of a series hybrid powertrain.*

2.1.2 Parallel hybrid

In a parallel hybrid configuration (Figure 2.1.2), the vehicle can be propelled both by the electric motor and the engine (Guzzella and Sciarretta 2007). This can be done both separately and collaboratively, which also means that none of the machines has to be large enough to deliver maximum power on its own. Regeneration of brake energy is possible through the electric motor. It is also possible to charge the battery by propelling the vehicle with the ICE and letting the electric motor be driven as a generator. A clutch is needed since the ICE is mechanically connected to the transmission and wheels. In addition, some type of variable gearbox must be used for the ICE to be able to propel the vehicle through its entire velocity range. In contrast to the series hybrid, only two machines are needed in the parallel hybrid powertrain; an ICE and an electric motor.

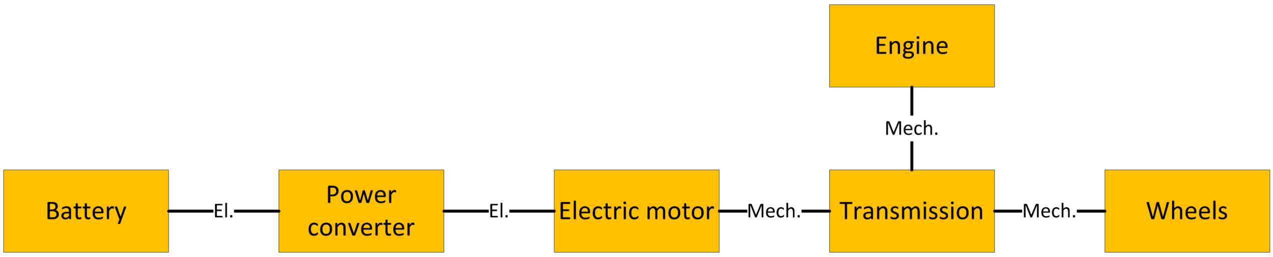


Figure 2.1.2: *Schematic view of a parallel hybrid powertrain.*

2.1.3 Range extender

Despite high energy conversion efficiency, electric vehicles of today are highly limited by the low amount of on-board stored energy. This leads to a very short driving range compared to a vehicle with a conventional powertrain. Furthermore, the time needed for battery charging is much longer than what it takes to fill up an ordinary gasoline vehicle. However, an electric vehicle can be modified to help reduce these disadvantages.

In order to extend the driving range and make an electric vehicle more autonomous, a *range extender* unit can be added (Wang et al. 2012). It generally consists of a relatively small ICE connected to a generator installed in a configuration similar to the series hybrid powertrain. The range extender can be used to charge the battery or supply electrical power directly to the traction motor. This makes the vehicle more independent from the availability of grid connections and improves its utility.

2.1.4 Test vehicle

In this project, an electric vehicle equipped with a range extender is used as test object, see Figure 2.1.3. The range extender unit consists of an ICE connected to an *integrated starter generator* (ISG), which is an electric machine that is used both as a generator and a starter motor. The ISG can be used to supply the electric motor with electric power directly or to charge the battery. Like most electric vehicles, the battery can also be charged from the grid.

The powertrain consists of a permanent magnet DC machine propelling the front wheels via a single speed gearbox. Electric energy is stored in a Li-Ion battery. The ICE is a spark ignited, naturally aspirated, three cylinder engine and the ISG is a permanent magnet DC machine. When operating the propulsion electric motor at high power levels, battery or ISG alone cannot supply sufficient current, but must be used together. For normal driving, each of the electric power supplies can be used separately.

There are a few different situations when the range extender is started by the *engine control module* (ECM). Firstly, if the demanded electric motor power exceeds the available battery power, the engine is started in order to satisfy the driver traction torque request. Secondly, the ICE is also started when the battery SOC level drops below a specified minimum level. When the ICE is running, the electric power is mainly delivered by the ISG. That means that the battery hardly delivers any electric power to the electric machine, unless the demand is higher than the maximum power the ISG is capable of delivering. In that case, the rest of the power exceeding the ISG supply will be delivered by the battery.

When the ICE is started by the ECM, it operates in a working point that is determined by the requested power. For each power request value, there is a specific corresponding engine speed and load found to give the best efficiency for the ICE and ISG together. In the normal case, the power request approximately equals the combined power demanded for propulsion and the auxiliary loads, which means that the SOC level will be sustained. When the battery SOC is approaching the specified minimum level, the power request to the ISG equals the combined propulsion and auxiliary power, as well as some additional power added for battery charging. As the SOC level recovers, the added charging power is reduced.

2.1.5 Energy efficiency

The cost of using on-board stored energy, i.e. fuel or battery, for propulsion of the HEV varies for different driving situations. Batteries, combustion engines and electric machines all have losses in their energy conversions. The resulting powertrain efficiency values can be maximized by choosing to run the vehicle on either battery energy or on fuel. In the case of a PHEV, the importance of this efficiency depends on the amount of energy that is needed in order to reach the next grid connection. If the battery stored energy is higher than the

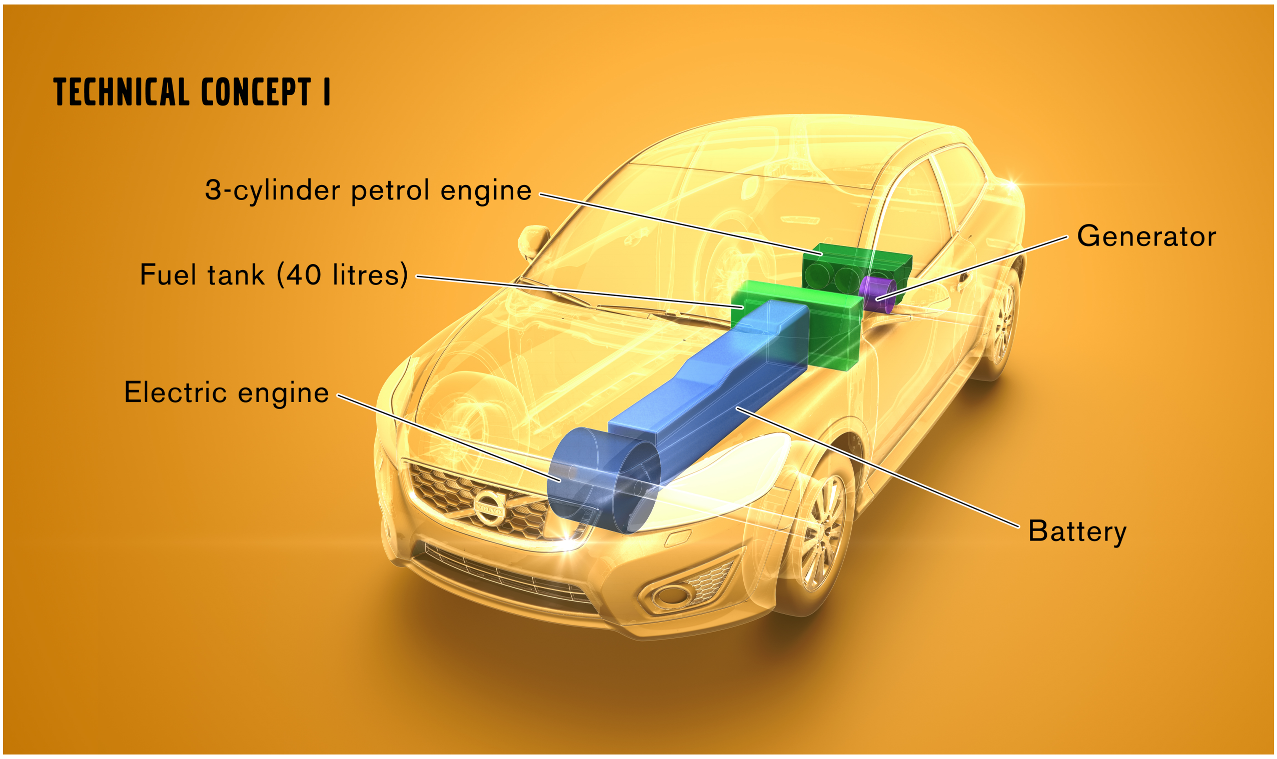


Figure 2.1.3: *Powertrain layout of the test vehicle (Volvo Car Corporation 2011).*

required amount, the lowest overall cost will be obtained by using the electricity in the battery, even though efficiency will not be at its maximum.

When delivering current, a battery is suffering from voltage drop and power loss due to its internal resistance (Guzzella and Sciarretta 2007). The internal resistance primarily varies with SOC and temperature, where low values for the two parameters result in high resistance. The power loss arising from the internal resistance is proportional to the squared current, resulting in drastically growing losses for an increasing current level. This leads to high efficiency at low current usage and much lower efficiency at high current usage. Consequently, it is preferable to use the battery as energy source when the demanded power is low.

An electric machine in combination with the power electronics needed for operation has high efficiency in a large part of the working area (Guzzella and Sciarretta 2007). In Figure 2.1.4, an example of an efficiency map for a brushless DC motor is presented. The x-axis represents rotational speed and the y-axis represents load. The graph for negative torques, i.e. when the electric machine is used as a generator, is very similar but mirrored in the x-axis. It can be seen that the highest efficiency is obtained around mid speed and mid to high load, which corresponds to relatively high power.

An ICE should preferably be run at low to mid speed and high load in order to operate within its most efficient working area (Heywood 1988). In particular, an ICE has poor efficiency at low load and is not capable of operating below its idling speed of around 900 rpm. Figure 2.1.5 shows an efficiency map for a typical gasoline ICE with rotational speed on the x-axis and load on the y-axis. It can be seen that the efficiency level in general is much lower for the ICE when compared to an electric machine. It may also be noticed that the difference between high and low efficiency in the ICE map is more obvious than the corresponding values for the electric machine. Therefore, it is of great importance to the overall efficiency that the ICE is run at favorable working points.

2.2 Reinforcement learning

This section contains an explanation of the basic concepts behind the reinforcement learning techniques that are used for control optimization of the state of the ICE.

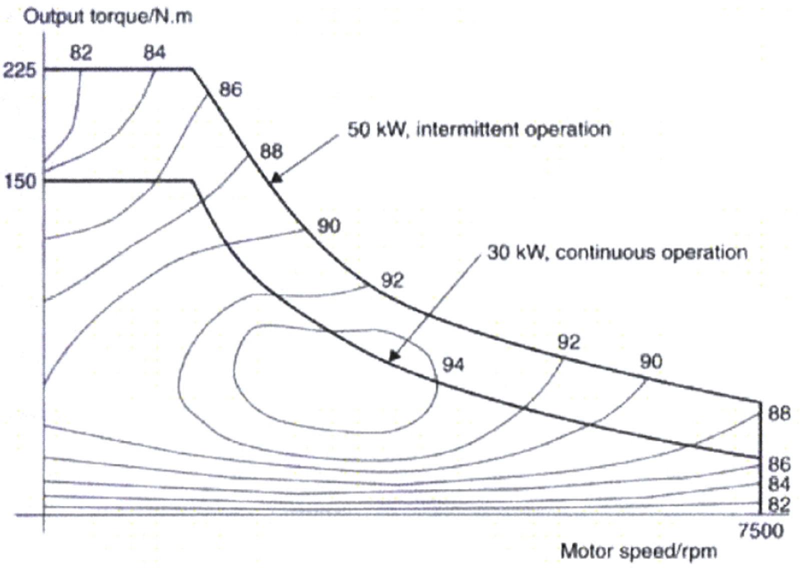


Figure 2.1.4: *Efficiency map for a typical brushless DC motor (Hashemnia and Asaei 2008).*

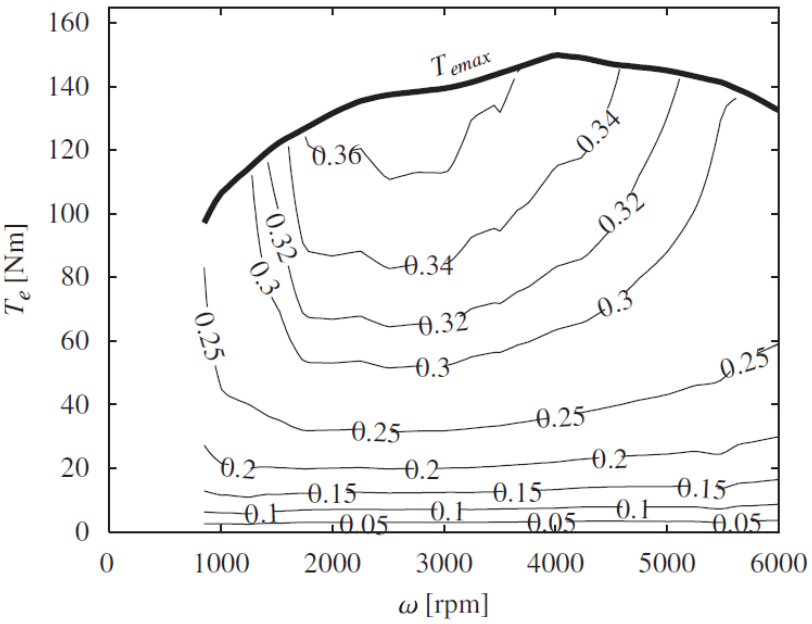


Figure 2.1.5: *Efficiency map for a typical gasoline ICE (Ambühl et al. 2010).*

2.2.1 Markov decision process

A *markov decision process* (MDP) is a description of a system as a stochastic process, see Figure 2.2.1. In each state of the MDP, it is possible to choose between a number of actions, which each has probabilities of leading to other states. When another state is reached, an immediate reward is collected for the transition from the old state to the new state. In other words, an MDP contains the following (Dayan and Watkins 2001):

- S, a finite set of states describing the current environment.
- A, a finite set of actions that can be taken in each state $s \in S$.
- $P(s'|s,a)$, probabilities of transitions between states $s, s' \in S$ given action $a \in A$.
- $R(s,s')$, immediate rewards for transitions between states $s, s' \in S$.

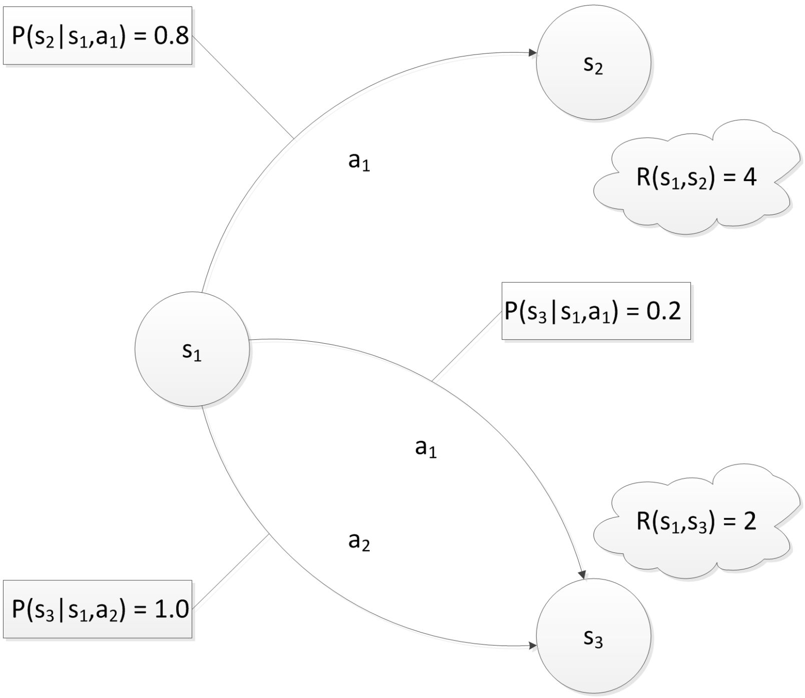


Figure 2.2.1: *Basic example of an MDP, where s_1,s_2 and s_3 are the names of the states.*

In addition, an MDP needs to satisfy the Markov property, which is defined as when given a state $s_t \in S$, where t is the point in time, all future states $s_{t+x} \in S$ are independent of any preceding states $s_{t-x} \in S$, where $x \geq 1$. This means that P and R depend only on the current state and action, which is crucial for the function of both dynamic programming and reinforcement learning methods.

One important problem that is often desirable to solve, is the value function of each state. This function expresses the total expected reward from starting in a specific state and traversing the MDP. A method commonly used to solve this problem is *dynamic programming*. This method requires R and P to be known, and works by starting from the end states and traversing the MDP backwards through all transitions while updating each state value. However, R and P are not always entirely known in advance, which makes alternative methods desirable.

2.2.2 Temporal difference learning

Temporal difference learning (TD-learning) is a method of making predictions about future rewards and adjusting those predictions when new values are observed. This can be used to learn the value function of an MDP without having access to complete knowledge about the system. This is in the simplest case achieved by observing the system in action and applying Equation (2.2.1). The value of a certain state $V(s_t)$ is updated with the reward of the state transition $r_{t+1} = R(s_t, s_{t+1})$, together with the value of the next state $V(s_{t+1})$. The factors α and γ are referred to as a learning rate and discount factor, respectively. The meaning of both is explained more thoroughly later in the thesis, but in short they control how much weight is given to newly discovered information.

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big] \tag{2.2.1}$$

2.2.3 Q-learning

One of the most common algorithms based on TD-learning is *Q-learning*, which is shown in Algorithm 2.1. It works by using the theory behind TD-learning, while extending it to not only make estimates about expected values of states $V(s)$, but also about the expected values of individual actions $Q(s,a)$, henceforth called *Q-values*. This provides a way to navigate through the MDP and enables its usage as a practical control algorithm. While a simulation is run or an episode is in progress, the algorithm uses the already existing Q-values in combination with some stochastic action selection method to choose the path through the MDP. At the same time, the Q-values are updated with new information collected through the run. This enables the algorithm to reinforce positive behavior.

When an action a has been taken in a state s, the reward r from the state transition is observed. $Q(s, a)$ is then updated with r and the maximum Q-value of the new state s'. The extent of which the different parts affect the updated value is determined by parameters such as the learning rate α and the discount factor γ. A smaller α makes the learning take longer time, but also takes more advantage of previously gathered information. A small γ means that the immediate reward plays a greater part in deciding the new Q-value (Sutton and Barto 1998).

Algorithm 2.1 Q-learning (Sutton and Barto 1998)

```
Initialize $Q(s, a)$ arbitrarily
for each episode do
    Initialize state $s$
    for each step of episode, until $s$ is terminal do
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., using an action selection method)
        Take action $a$, observe immediate reward $r$, new state $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    end for
end for
```

2.2.4 SARSA

While Q-learning performs well in many circumstances, the fact that it always uses the highest Q-value of the new state to update the previous state means that potentially beneficial paths through the MDP are discarded. Often, that is not a problem, but in some cases faster exploration is preferred. A modification of Q-learning called SARSA (*state-action-reward-state-action*), shown in Algorithm 2.2, possibly satisfies this need. The main difference between SARSA and Q-learning is how Q-values are updated. In contrast to Q-learning, SARSA waits one extra step to see which action is taken in the new state and updates using the Q-value of that action (instead of using the maximum Q-value). This means that SARSA not only propagates good outcomes, but also bad outcomes (Sutton and Barto 1998).

Algorithm 2.2 SARSA (Sutton and Barto 1998)

```
Initialize $Q(s, a)$ arbitrarily
for each episode do
    Initialize state $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., using an action selection method)
    for each step of episode, until $s$ is terminal do
        Take action $a$, observe immediate reward $r$, new state $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., using an action selection method)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    end for
end for
```

2.2.5 Q-matrix initialization

Depending on the size and nature of the rewards, the initialization of the matrix containing all the Q-values of the system, referred to as a *Q-matrix*, may significantly change the way in which the state space is explored. Since the rewards are negative in this case, setting $Q(s, a) = 0, \forall s \in S, a \in A$ will mean that it is always better to try any unexplored actions when they are available. The opposite would be true if the rewards were set to be positive, giving more weight to exploring known paths. The first way is called *optimistic* initialization, in that it is always assumed that the unexplored paths are better, with the second way similarly being called *pessimistic* initialization (Grześ and Kudenko 2008).

2.2.6 Action selection method

There are several methods for choosing the next action in a state depending on the Q-values of that state, among which *ϵ-greedy* and *softmax* are the most common. No comparative studies has been found to show if one or the other performs better, and therefore both had to be considered before making a choice.

The way ϵ-greedy works is that the action with the highest Q-value will be chosen with a probability of $(1 - \epsilon)$ and in the case that it is not chosen, one of the other actions is chosen uniformly at random. Low values of ϵ have been shown to provide a good balance between exploration of the state space and exploitation of the already known Q-values (Coggan 2004). There are two main drawbacks with this method. Firstly, the Q-value of the preferred action may not be much different from the Q-values of the other actions, while still giving it a great advantage. Secondly, one of the other actions will be chosen completely at random if the preferred action is not chosen, ignoring any differences between the Q-values, and thereby discarding potentially useful information.

Instead of using a constant probability of getting the action with the highest value, as is done in ϵ-greedy, Softmax uses probability distributions such as Boltzmann or Gibbs to determine which action to take. In doing so, it makes use of the actual differences between Q-values of actions. As shown in Equation (2.2.2), where a Boltzmann distribution is used to determine the probability that action a is chosen, the parameter T is the equivalent of the ϵ in ϵ-greedy. Here, greater values of T means that actions will be chosen more at random, while lesser T means increased probabilities for higher Q-values (Sutton and Barto 1998).

$$Pr_{softmax}(a) = \frac{e^{\frac{Q(s,a) - \max_b Q(s,b)}{T}}}{\sum\limits_{a} e^{\frac{Q(s,a) - \max_b Q(s,b)}{T}}} \tag{2.2.2}$$

2.2.7 SARSA(λ)

One disadvantage with both Q-learning and SARSA is that propagation of rewards backwards through the Q-matrix is slow, with information only traveling one step each episode. This will make it hard for the algorithm to notice if early decisions lead to late rewards or costs (Sutton and Barto 1998). Consequently, it will also be harder to make good decisions in time. To help correct this problem, there is a method called *eligibility traces* that can account for delayed reward.

Instead of updating only the previous state in a step, it updates the whole history of states up until that point in the episode, with the updates having less effect on the states by a factor λ for each state backwards in time. This means that when rewards or cost are great, earlier states bear some of the responsibility, albeit to a lesser degree than the most recent. The SARSA(λ) algorithm is shown in Algorithm 2.3.

Algorithm 2.3 SARSA(λ) (Sutton and Barto 1998)

```
Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$, for all $s$,$a$
for each episode do
    Initialize state $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., using an action selection method)
    for each step of episode, until $s$ is terminal do
        Take action $a$, observe immediate reward $r$, new state $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., using an action selection method)
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $e(s, a) \leftarrow e(s, a) + 1$
        for all $s, a$ do
            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
            $e(s, a) \leftarrow \gamma \lambda e(s, a)$
        end for
        $s \leftarrow s'$
        $a \leftarrow a'$
    end for
end for
```

3 System design

In this chapter the approach for the project behind this report is described, starting with a system overview and subsequently providing explanations to each of the component parts of the solution.

3.1 System overview

The system consists of four main parts showed in Figure 3.1.1:

- The *accessory control unit* (ACU) is an Android-based infotainment system found in future Volvo vehicles, which in addition to capabilities nearly identical to modern smartphones and tablets, also will have limited access to the multimedia and other systems of the vehicle. By taking advantage of the open nature of the Android framework, it is possible to create an application that has the following features:
  - Receive and interpret vehicle data from the CAN bus through a CAN/Ethernet gateway, such as vehicle speed, accelerator pedal position or SOC.
  - Upload and download data to and from a relational database on a Windows Azure cloud service.
  - Send an engine state control signal to the vehicle depending on a given strategy.
  - A *graphical user interface* (GUI) for calling the above functions.
- The *Matlab client* is responsible for computing and uploading strategies for engine control to the cloud service. To be able to do this, it has:
  - A GUI for parameter settings and starting/stopping the algorithm.
  - A function for iterating the simulation, to be used by the algorithm.
  - A modified vehicle model for simulations, incorporating a subsystem for use by the algorithm.
- The *cloud service* is used as an intermediary between the logging/control application on the ACU and the Matlab application doing the calculations, allowing for the possibility of multiple clients making use of the same data. It contains:
  - A database with tables containing driving logs, strategies for engine control and general information about the route, vehicle and options of a specific log or strategy.
  - A Windows Azure mobile service functioning as a front end for client access to the database, allowing secure access control, data querying and data insertion.
- The *engine control module* (ECM) is modified with the help of VCC to enable remote engine state change requests via the CAN bus and to provide more detailed driving statistics.

3.2 Vehicle simulation

For drive simulation of the vehicle a Matlab Simulink model is used. The model has been developed by VCC and consists of numerous subsystems for every part of the vehicle that needs to be taken into account in order to perform a drive simulation of the vehicle.

3.2.1 Original vehicle model

The original vehicle model, on which the simulation work in this project is based, is designed to only take a velocity profile as input. Figure 3.2.1 presents a principle view of the relation between the input data and the resulting energy demand. The velocity profile is used as the set point value to the virtual driver, which tries to make the vehicle follow the set point by changing the position of the accelerator pedal and the brake pedal. The virtual driver is implemented as a PID controller that compares the set point value to the calculated vehicle speed in all simulation steps. Vehicle acceleration, and thereby the speed, is a result of the traction force and the total resistive force acting on the vehicle, which in turn depends on the vehicle speed.

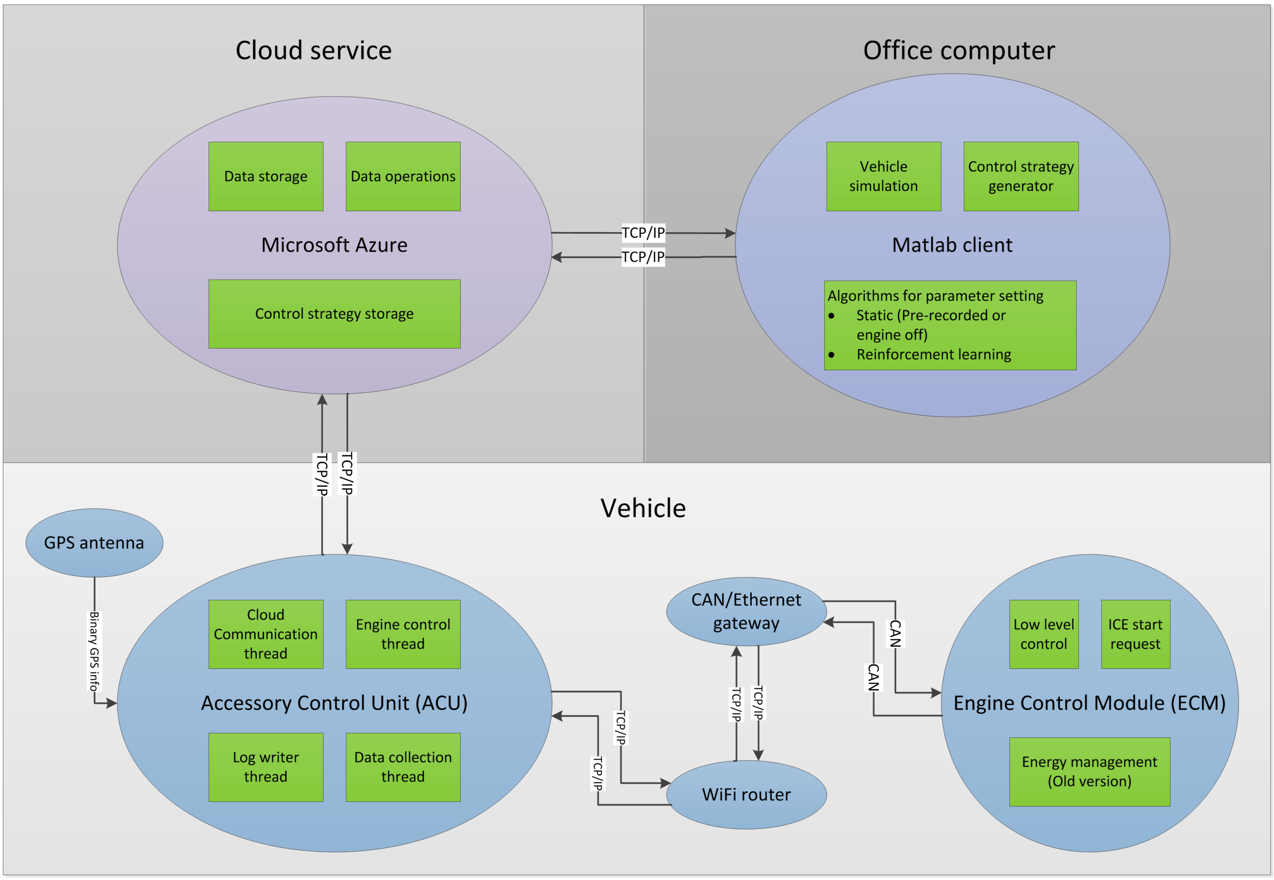


Figure 3.1.1: *Overview of the system in it's entirety.*

The method mentioned above is the natural way of using the vehicle simulation model to predict performance and fuel consumption for vehicles that are under development. In order for the model to be accurate, the total mass of vehicle, passengers and luggage together must be known, as well as the road inclination along the route and the effects from possible wind. Also the changed aerodynamic properties, for example from mounting a roof box, must be taken into account in this simulation approach.

3.2.2 Modification of model

The intention has been to use the original model to as large extent as possible and to create necessary new functionality by modifying the existing content in that model when possible. The result is a comprehensive model of the vehicle where only a few major changes have been necessary.

For this project the simulation model is used exclusively to repeat already known vehicle runs in order to find the energy consumption, and not for performance evaluation of the vehicle. The vehicle speed is solely used to determine the resulting electric motor speed. The model is modified in order to enable development of a control strategy for the ICE through simulations based on logged driving data. Instead of using a virtual driver as in the original model, the accelerator pedal position as well as the real vehicle speed is given directly from the logged data.

Figure 3.2.2 presents a schematic principle view of the electric propulsion system in the modified model. The major difference from the original model is that the modified model has limited connection between traction force and vehicle movement. Vehicle speed, through wheel speed and a specific ratio, determines the electric motor speed and thereby affects the available motor torque, efficiency and power. However, the motor torque does not affect the vehicle speed at all. The vehicle speed is instead an input vector to the simulation model, which makes the simulated vehicle run the same drive cycle as the real test vehicle. The energy consumption is determined from the motor speed and the accelerator pedal position together.

The modified simulation model has the advantage that the rolling resistance and aerodynamic drag, which earlier had to be taken into account, now can be ignored. This also means that the model becomes more robust

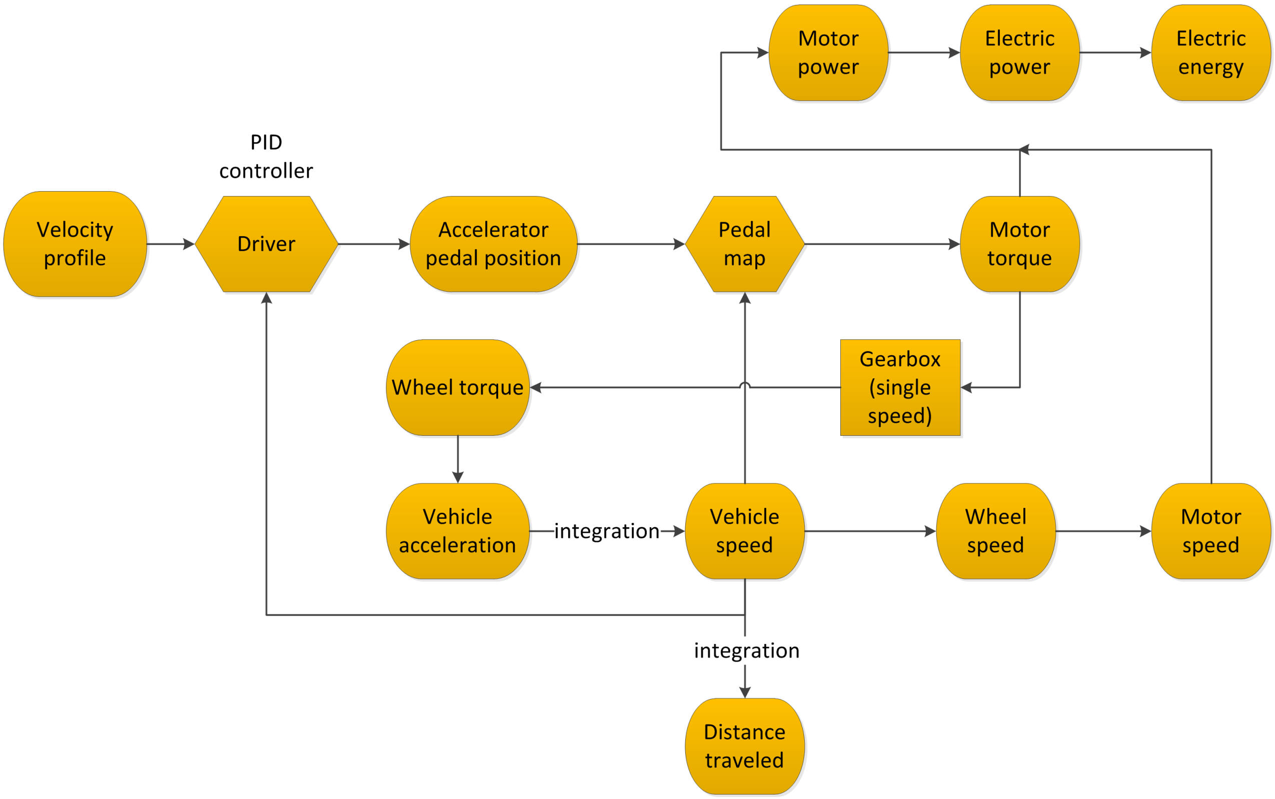


Figure 3.2.1: *Principle figure of the original vehicle model regarding input data usage in simulation.*

regarding varying resistive forces from wind, inclination or different vehicle mass as a result of different number of passengers or amount of luggage.

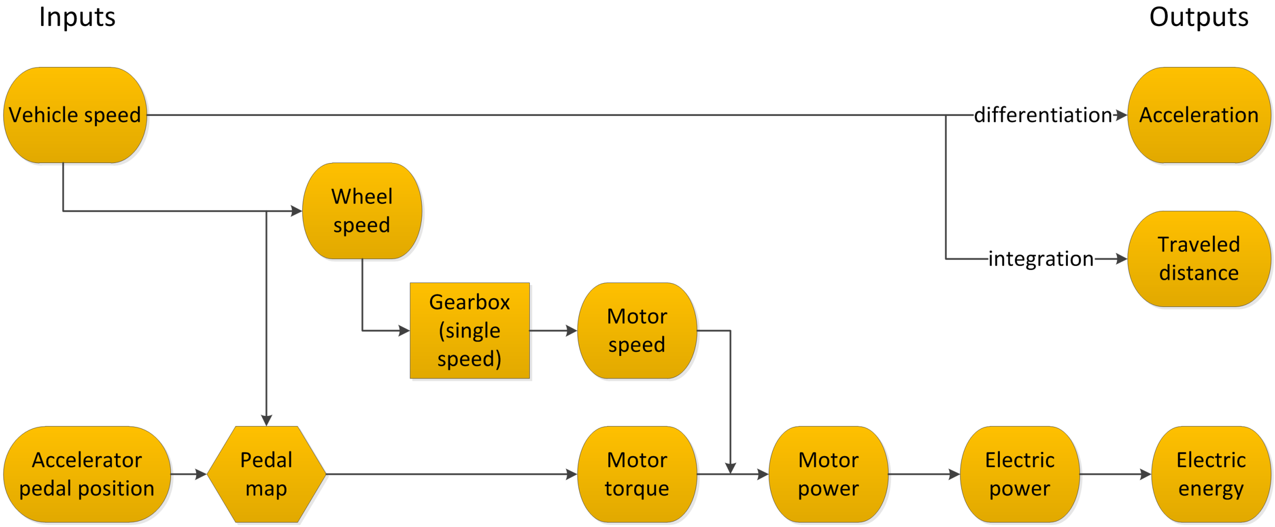


Figure 3.2.2: *Principle figure of the modified vehicle model regarding input data usage in simulation.*

3.2.3 Model validation and calibration

It is of great importance for all simulation activities that the model shows satisfying compliance to reality. Therefore, at an early stage in the model development, simulation results have been compared to logged data from the real vehicle driven on a test track. For the purpose of this project the most important correlation to be checked is the battery SOC level, since it is the measure of electric energy stored on-board. Even though small variations due to dynamic effects in the real battery are acceptable, the general shape and level must not differ more than a few single percent to be satisfactory. The result of a faulty assumed SOC level would be both that the battery electric driving range is overestimated or underestimated, and that the built in static strategy for low SOC level will affect the system at unexpected occasions.

In order to obtain both accurate prediction of SOC level and to get the dynamic battery power limit right, a current limitation function has been developed. The function calculates the *root mean square* (RMS) current and creates a running mean value over a specified time period. If the mean value exceeds a certain limit, the maximum and minimum battery current will be restricted for a while before the system restores the standard current limits. This modification affects the built in strategy for range extender start at high power request, as the battery power limits vary.

Validation of the range extender model is also important in order to achieve proper results for fuel consumption. Unfortunately, all test runs with the real vehicle had to be done with pure battery electric drive due to malfunction of the range extender unit at the time for model validation. Consequently, neither calibration nor validation of the simulation model regarding fuel consumption could be performed.

3.3 Control and logging application

This section describes the development of the Android application responsible for logging of driving statistics and control of engine state, intended to be placed in an ACU. Android is an open source operating system used mainly for mobile devices with touchscreens. A custom Java framework simplifies development for the platform (Google Inc. 2013).

3.3.1 Equipment

To be able to facilitate the communication between the application and the car, as well as to set up a useful and realistic test environment, several pieces of equipment are used. To develop the application in an environment as close in resemblance to the real ACU as possible, an official development kit is used. In order to connect the CAN bus of the vehicle to the ACU, a CAN to Ethernet gateway encapsulates the CAN frames into TCP/IP packets. Furthermore, a wireless router or an USB Ethernet adapter relays the packets to the ACU.

When testing in an office environment, the connection to the CAN bus of the car is exchanged with a CAN/LIN interface module connected to a PC. On the PC, there is software for logging and sending simulated CAN messages, either individually or according to a recorded drive.

3.3.2 Application structure and usage

The application is structured with a main persistent thread handling GUI operations and several worker threads for background operations, which are started when called upon by the GUI. The structure can be seen in full in Figure 3.3.1. The decision to start the worker threads only when necessary is motivated by the extra need for performance in a low-end system such as the ACU. As far as possible the threads are operated independently from each other, but when shared data cannot be avoided, it is protected with semaphores to avoid concurrency errors.

The various functions of the application are grouped in scrollable tabs in the GUI depending on if they are more closely related to logging or control. In addition, another tab has been created to house general status messages from the application. A user would begin by choosing the *logging options* tab and check the indicators to confirm that the application is connected to the CAN gateway and that it is receiving CAN messages. If so, the user can start logging data and eventually stop the logging when no more data is needed. Until uploaded, the logged data is stored locally in an XML file. When the user has decided to upload the file, it is transferred to the cloud database along with all other local XML files and subsequently deleted.

If the user, instead of logging, chooses to control the ICE according to some strategy, he/she scrolls over to the *control options* tab and starts downloading a strategy. Once it is indicated that a strategy is present, the control function can be initiated. It will continue until either the strategy is finished or the user decides to manually end it.

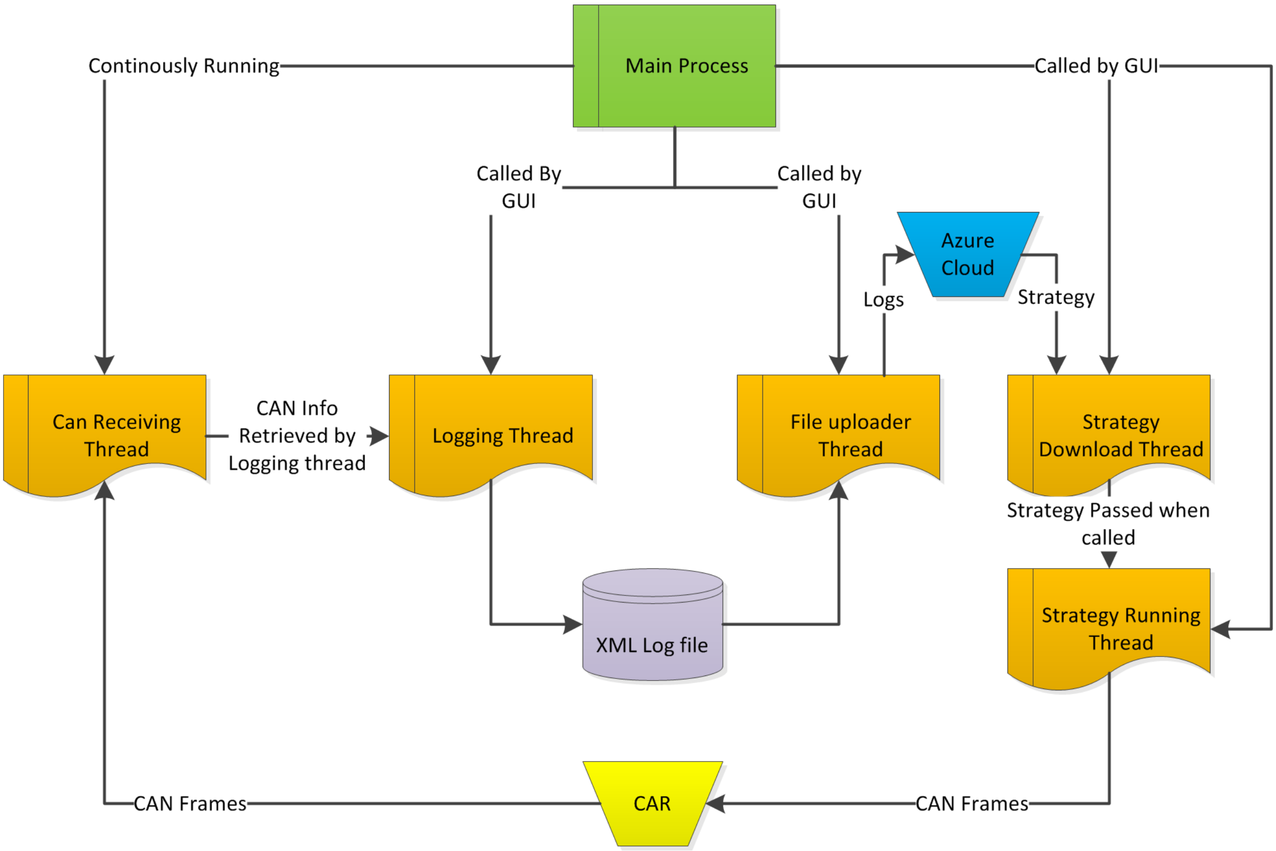


Figure 3.3.1: *An overview of the communication flow and hierarchical relationship of the application components.*

3.3.3 Logging of driving statistics

When the user initiates the logging, a new thread is started with access to the signals that are to be logged. Regardless of the frequency in which the signals are received, they are read with a constant frequency when logged. The main limitation of possible frequencies to choose is that the GPS dongle has a max frequency of 1Hz. At the beginning of the XML file, general information about the entire log is written, such as vehicle name, driver, etc. After that, for each time interval, the signals are read into the XML file along with the GPS position and a time stamp. When the logging is done, the XML file is saved with the time and date of the log as name.

The advantage of using XML as opposed to a less verbose format in this case is that due to the modular nature of XML, the number and nature of the signals are not hard coded, which makes it easier to add or remove signals. In addition, there are multiple commonly used packages for Java to simplify manipulation of XML.

3.3.4 Communication with vehicle

The class which facilitates communication between the application and the ECM of the vehicle was developed by VCC for an earlier project, but has been modified and will be described here. The internal network of the vehicle uses the CAN protocol for communication, and a single data frame is defined as in Figure 3.3.2. There are other types of frames, like error frames, but they are largely ignored by this application. The data field of the frame is subdivided into smaller parts with identifiers, options and values. These contain individual signals transmitted within the network, such as vehicle speed and SOC. The exact structure of the data frames for a particular network is defined in a definitions file, which the application has access to.

When the communication class is initiated, the application attempts to establish a connection with the CAN-Ethernet gateway and if successful, waits for and handles any CAN frames received. It starts by sorting out all frames that are not data frames, followed by all data frames that do not contain any of the signals in the list the application is instructed to listen for. When that is done, the same procedure is repeated but

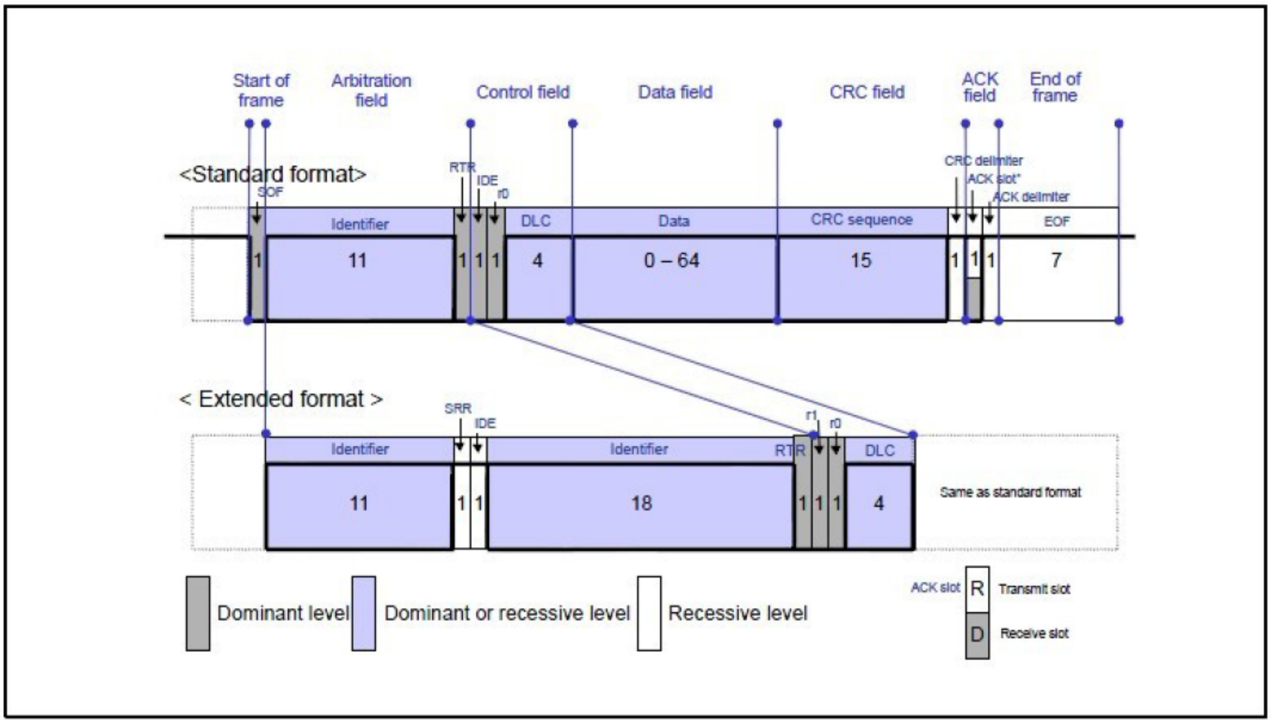


Figure 3.3.2: *The structure of CAN frames with extended and standard addressing format (Renesas Electronics Corporation 2006).*

instead sorting out the correct signals from each frame and interpreting their values. The signal list is then updated with the new values.

When a great number of frames are sent through in both directions, the CAN-Ethernet gateway has difficulty coping, so a modification has been made to lessen the load. It is possible to change the gateway's settings on the fly by sending commands to it, and a certain setting is to block or permit frames from coming through according to their identifiers. This is done by providing an acceptance mask and acceptance code to the gateway. The mask defines which bits in the identifiers of the incoming frames should be equal to the corresponding bits in the acceptance code. A function is created to produce the minimal acceptance mask and code that would let through all the necessary frames. Some unnecessary frames are still let through, but the load is significantly lessened. This function is called before the communication class is initiated.

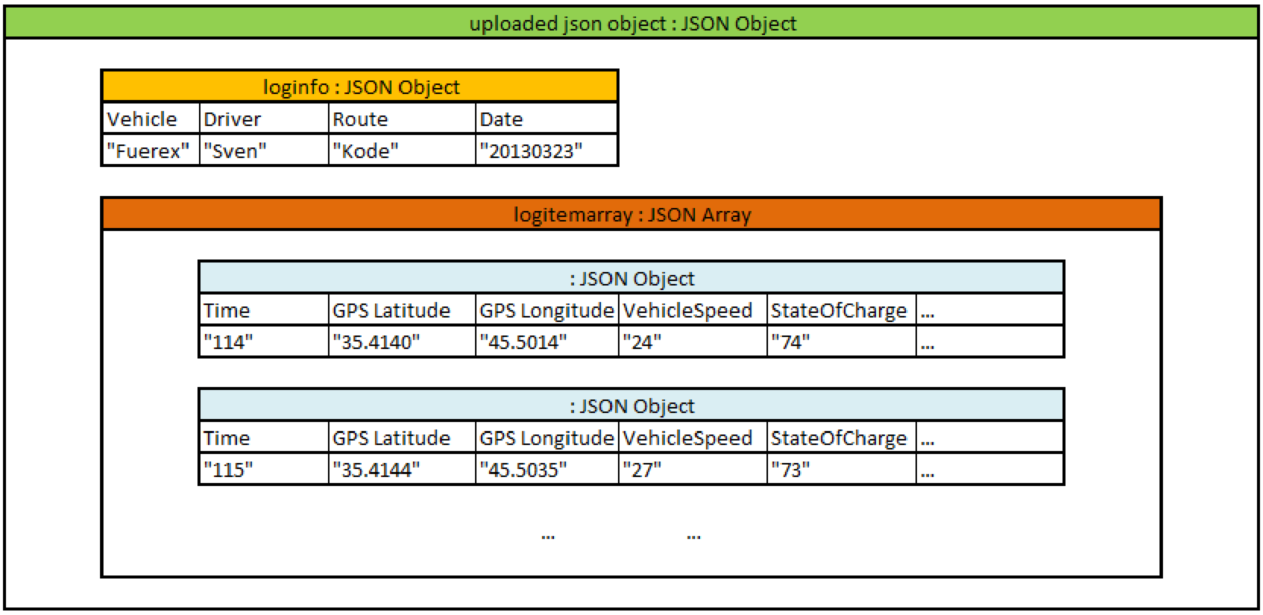
In addition, the frame interpreter has been modified to interpret signed signals, i.e. signals which allow for negative values.

3.3.5 Communication with remote server

The communication with the server is split up into two separate classes for upload and download. When the user requests that the local logs should be uploaded to the server, the first thing that is done is to search the folder where the logs are saved for all present XML files following the predetermined naming pattern. If and when appropriate files are found, they are processed sequentially starting with the earliest one to ensure that they end up in the right order in the online database. After that, the XML file is interpreted and the signal names and values are extracted.

The protocol used for uploading the data utilizes the JSON data format, which has support for representing basic data structures like arrays and key-value pair objects. This is used to create a structure in which the top-level object contains general information about the data being uploaded, in addition to an array containing objects representing individual items and their signal names and values, as shown in Figure 3.3.3.

On the cloud service, there is one table containing general information about strategies and a second table containing the actual strategies, which is further explained in Section 3.4.2. When the user wants to download the latest control strategy available for a specific route, the application initially queries the first table and asks for the ID of the strategy with the most recent time stamp. When a response is received, the application uses



Vehicle	Driver	Route	Date
"Fuerex"	"Sven"	"Kode"	"20130323"

Time	GPS Latitude	GPS Longitude	VehicleSpeed	StateOfCharge	...
"114"	"35.4140"	"45.5014"	"24"	"74"	...

Time	GPS Latitude	GPS Longitude	VehicleSpeed	StateOfCharge	...
"115"	"35.4144"	"45.5035"	"27"	"73"	...

Figure 3.3.3: *The hierarchy of the JSON object containing logs sent to the server.*

that ID to query the second table for all strategy items with that particular ID. The data is received as a JSON array which is separated and put in a Java data structure.

3.3.6 Engine control strategy deployment

Once a strategy for engine control has been downloaded to the application, the user can request for it to be started. As is explained in Section 3.5.1, the distance traveled is used to determine where along a route the application will request a change in the engine state, so the distance has to be calculated. This is done by listening for the vehicle speed signal on the CAN bus and integrating it to receive the distance. When the distance has been calculated, the corresponding engine state is read from the downloaded strategy. The desired engine state is placed in a CAN frame specifically designed for this purpose and then transmitted at a constant time interval to the CAN bus. When this interval was chosen, care had to be taken since it was noticed that sending a large amount of data in a short amount of time caused the CAN-Ethernet gateway to malfunction.

3.4 Cloud system

Cloud is a general term for vast scalable computer networks. Multiple companies rent out remote computational resources to those who need it. The cloud platform used in this project is Windows Azure, since it has functionality specialized for mobile devices (Microsoft Corporation 2013). This chapter describes the development of databases and scripts that handle the data of the system.

3.4.1 Communication interface

When the database has been queried for information, the JSON format is used in the response similarly to when data is inserted to the table, but with the structure and contents depending on which parameters are supplied with the query. Windows Azure uses a REST API for database queries, which means that the URL used to address the querying packet is augmented with keys and values defining the query. While this API is not as expressive as SQL, which is the most common way in which relational databases are queried, it still serves the needs of this project by providing the ability to specify table columns, row order and filter results with predicate logic.

There is a limit on the amount of data that can be downloaded in a single packet from the database. Since this limit can be exceeded by the data in the tables, a function for splitting it up into smaller parts before being downloaded had to be implemented. This function works by using special parameters for the REST

API when making the original query. When the first result arrives, it will contain information about the max amount of items allowed, the total number of items in the queried data and the current item index intervals. Using the last index as a starting point for the next query and stopping once the total number of items is reached, all the desired data items are guaranteed to be received.

3.4.2 Databases

The database contains four tables for logs and strategies. Two of these tables, called *data tables*, contain individual data items. In the log data table, an item is defined as recorded driving data from a particular instance in time. In the strategy data table, an item is a distance triggered engine state request. Each item also contains, in addition to information uploaded from clients, an incremental item ID and a common *log/strategy ID* for all the items of that particular log/strategy. The other two tables, called *info tables*, contain a single entry for each log/strategy ID, with general information about, among other things, the route, vehicle and algorithm used.

Windows Azure databases normally only accept a single item in each packet uploaded for insertion into a table, which would make uploads of large amounts of data unnecessarily complicated. To solve this issue, an insertion script is created in each of the data tables. When a whole log/strategy has been received (or just the first part of the log/strategy if the number of items exceed a threshold value), it is split up into single items that are inserted individually. In addition, the first part of the packet contains general information about the log/strategy, which is inserted into the corresponding info table, with an incremental log/strategy ID being retrieved for the data table at the same time.

Using the Windows Azure mobile service front-end, instead of directly accessing the database through SQL, has at least two advantages. The first one is that when the *dynamic schema* setting is enabled, the table is automatically expanded with new columns when signals are added to the packets, which means that it is enough to define which signals to use in the client application. The other advantage is that by adding another layer between the database and the client, things like external access control is handled by using either user authentication or an application key, instead of having to supply direct login information as when using SQL.

3.5 Learning application

For the learning application to function, it is not only necessary to augment the vehicle model with a reinforcement learning algorithm, but also to create supporting functions for information handling and algorithm iteration. This system and the process of algorithm improvement are outlined in the following section.

3.5.1 MDP creation

To be able to use reinforcement learning for optimization, an MDP that accurately describes the system has to be formulated. The only possible actions in this case are to turn the engine on or off. To reason on how the state space should look like, the problem and environment have to be examined. In order to actually be able to use a strategy in a coherent way, it cannot be dependent on drive time, since the events that occur along the route will probably not happen at the same relative time each trial. Therefore, space indexation seems like a more appropriate approach. Since the project is limited in scope to only consider one specific route, it has been decided that distance traveled will be used to tell the vehicle where to turn the engine on and off.

If the distance is known, the corresponding drive time can be found and used. As already mentioned, the model has been modified to use logged information dependent on time to determine most of the environment. The exceptions are SOC and fuel consumption. SOC is dependent on the initial value in addition to which actions are taken during the run, while fuel consumption is what ultimately should be optimized. Since the goal is to maximize fuel efficiency, it is natural to use fuel consumption as a negative reward for the MDP, i.e. a cost.

The MDP needs to fulfill the Markov property, which means that each state has to fully formulate the current state of the environment. With that in mind, SOC and distance traveled are chosen as state variables. When deciding on the granularity, it is appropriate not to change the engine state too often, since it can lead to both unnecessary uncertainty about anomalies like cold start behavior (and thus potentially compromising the Markov property) and discomfort to the passengers. The final structure of the MDP is shown in Figure 3.5.1.

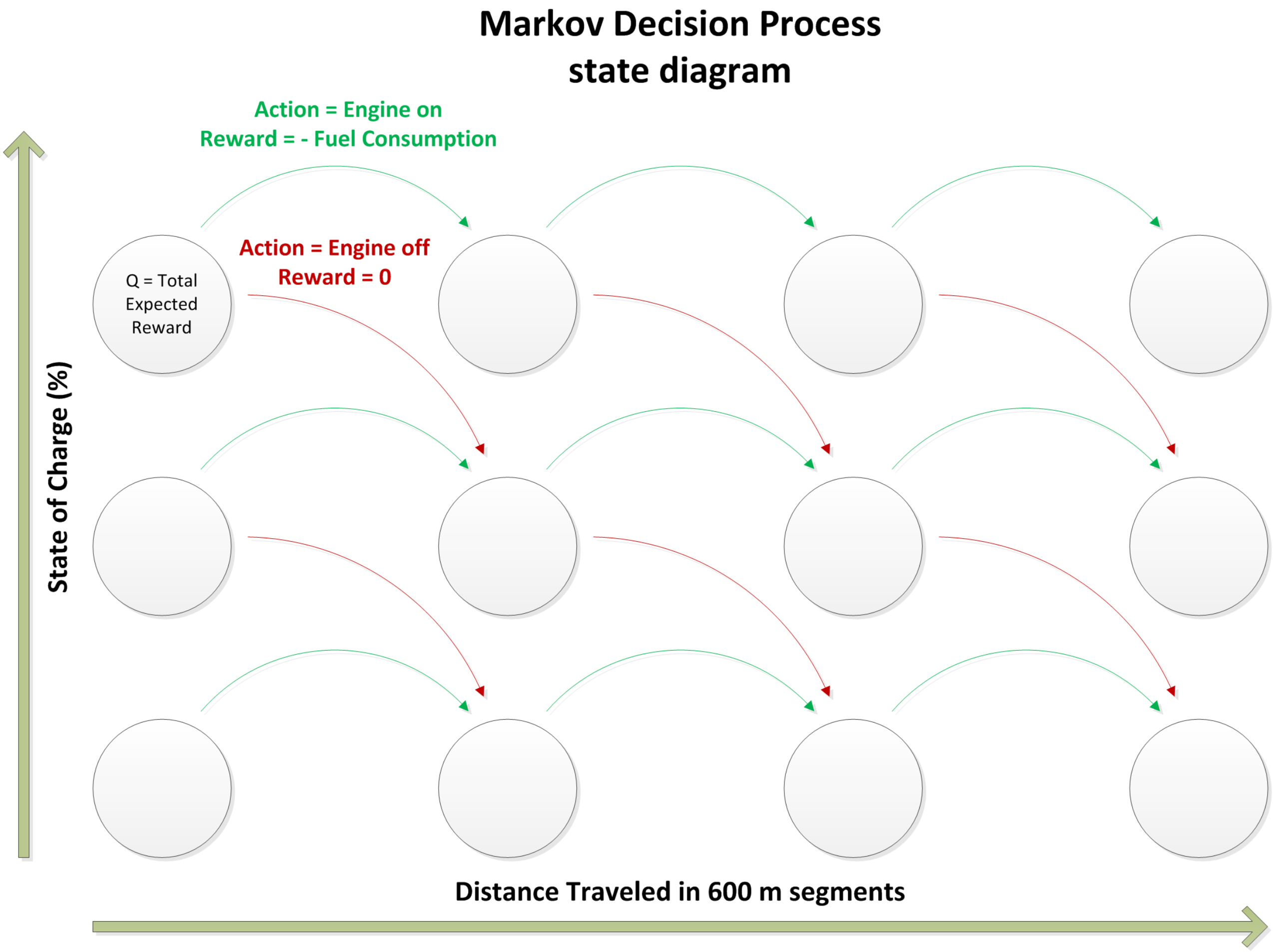


Figure 3.5.1: *The markov decision process describing the system. Note that the transitions can lead to higher SOC values as well, even though it is not shown in the figure.*

3.5.2 Algorithm design choices

By formulating the demands placed on the algorithm by the system, the process of making important design decisions is made easier. Since the algorithm is intended to find a strategy as close to the optimal one as possible, the algorithm needs proper incentive to explore the area of the MDP in which the optimal strategy exists. Additionally, there might be several areas of the MDP that contains locally optimal solutions, multiple of which may be easier to find than the global optimal solution. This can lead to the algorithm settling for a suboptimal solution if the degree of exploration is low. Consequently, a great degree of exploration is likely to be beneficial.

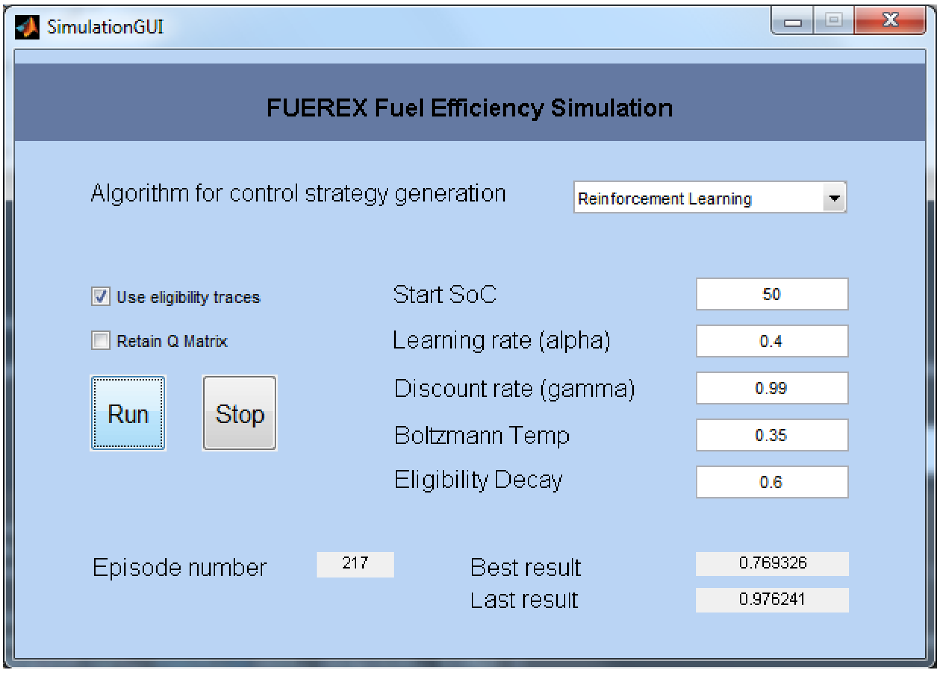
As has been mentioned earlier, there are two main ways in which the Q-matrix can be initialized. Depending on the situation or domain, both have their distinct advantages and disadvantages. It has already been established that the goal is to explore a greater part of the state space in order to be more likely to find the optimal path, and optimistic initialization seems to provide more benefits in this regard, since it leads to faster exploration of unexplored paths. Additionally, considering that the rewards of the MDP are negative, it is natural to initialize the Q-matrix with zeros.

With the decision between SARSA and Q-learning, it has been shown that SARSA frequently lead to higher rewards than Q-learning in environments where the optimal path is hard to find among many suboptimal paths. Since Q-learning does not propagate bad outcomes through the Q-matrix, it is more likely to stick to a locally optimal solution when it is found, rather than to continue looking for a better one (Sutton and Barto 1998)(Coggan 2004). Consequently, SARSA is used in this system.

As with the choice of algorithm, the action selection method used should favor exploration. Similarly to Q-learning, ϵ-greedy is more suited for greedy search through the state space, which makes it less than suitable in this case. Softmax is likely to be a better candidate since it takes more of the discovered information into account when selecting an action.

3.5.3 Simulation iterator and auxiliary systems

Various functions has been created around the vehicle simulation model to support the learning algorithm and to make the handling of the whole system more convenient. The most important part of these auxiliaries is the script that runs the simulation iterator, handles parameter setting and presents the results obtained in the iterations. For convenience as well as prevention of mistakes, there is a GUI from which the simulation can be controlled, see Figure 3.5.2. In the GUI, parameters and choices of interest for the operator are conveniently accessible and visible. Furthermore, relevant information about the simulation progress is displayed.



Algorithm for control strategy generation: Reinforcement Learning

☑ Use eligibility traces

□ Retain Q Matrix

Start SoC	50
Learning rate (alpha)	0.4
Discount rate (gamma)	0.99
Boltzmann Temp	0.35
Eligibility Decay	0.6

Episode number: 217

Best result: 0.769326

Last result: 0.976241

Figure 3.5.2: *Screenshot of the GUI of the Matlab simulation.*

A separate function for server communication exists in order to enable logged vehicle data to be downloaded from the database, and range extender control strategies to be uploaded. Information from each simulation is also saved on the computer that runs the simulations and can be used as a base for initialization of new simulations.

3.5.4 Algorithm subsystem

A subsystem inside the vehicle model performs algorithm steps which needs to be done while the simulation is running, see Figure 3.5.3. Even though the iterating procedure is governed from outside of the vehicle model, the learning algorithm must be run synchronously during the vehicle simulation in order to experience the effects of ICE state changes. Furthermore, the learning algorithm is dependent upon receiving certain information from the vehicle simulation in real time to be able to choose which actions to take.

3.5.5 Adaption of parameters

When deciding upon parameter values, multiple studies were examined to find the best ones for this domain (van Hasselt 2011)(Grześ and Kudenko 2008)(Coggan 2004). In the case of discount factor γ, a high value was decided upon, since immediate rewards are of less importance than the total reward. Different suggested values of learning rate α, Boltzmann temperature T and eligibility decay rate λ were tried out in a smaller experiment setting in order to find out the optimal ones.

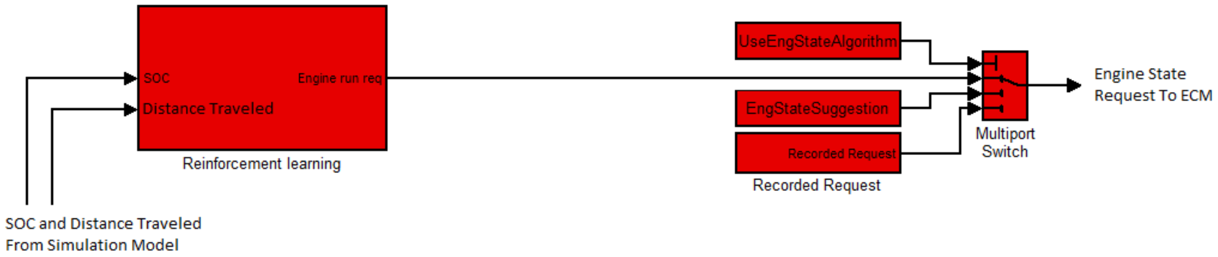


Figure 3.5.3: *The subsystem of the vehicle simulation implementing the algorithms.*

To determine a good value for the Boltzmann temperature T, the nature of the values in the Q-matrix had to be inspected. In order to make that task easier, a normalization of the compared Q-values was added to Equation (2.2.2). After a couple of runs of the algorithm (with eligibility traces enabled) along a test route, average values of the difference between Q-values of both actions were found and inserted into Equation (2.2.2). It was discovered that the difference in the resulting probability values decreased in longer runs and that as a consequence the randomness increased over time.

Since this means that the algorithm would increasingly discard good solutions, it was decided to implement a function to decrease T over the number of episodes. The same was done with *alpha* since smaller differences between Q-values otherwise would mean that new information would affect the values too much.

The starting SOC level was set low enough that the engine had to be turned on in order for the vehicle to be able to reach the destination. Otherwise, the optimal solution would trivially always be to leave the engine off.

4 Experimental results

In this chapter, the outcome of both model evaluation and simulations using the software and algorithms developed during the project are presented. Findings and possible error sources are discussed.

4.1 Test methodology

This section describes how the external conditions for the tests were determined and presents some brief reasoning about choices made. In addition, the method of test data collection is described.

4.1.1 Drive cycle

In order to test the algorithms, a drive cycle had to be chosen and used for collection of the necessary vehicle data. The idea was to imitate a typical car commuter who lives outside the city. In order to offer decisive conditions for the algorithm and thereby produce interesting results, the route was designed to have a varying road load along the distance. Through some road map research a suitable route was found, see Figure 4.1.1. From the start address, just outside the VCC PV gate at Volvo Jakobs väg, the drive went towards Kode via Hisingsleden, Kongahällavägen and E6. At Kungälv the road type changed from country road to motorway. The final destination at Kode railway station was reached after about half an hour and 35km of drive.

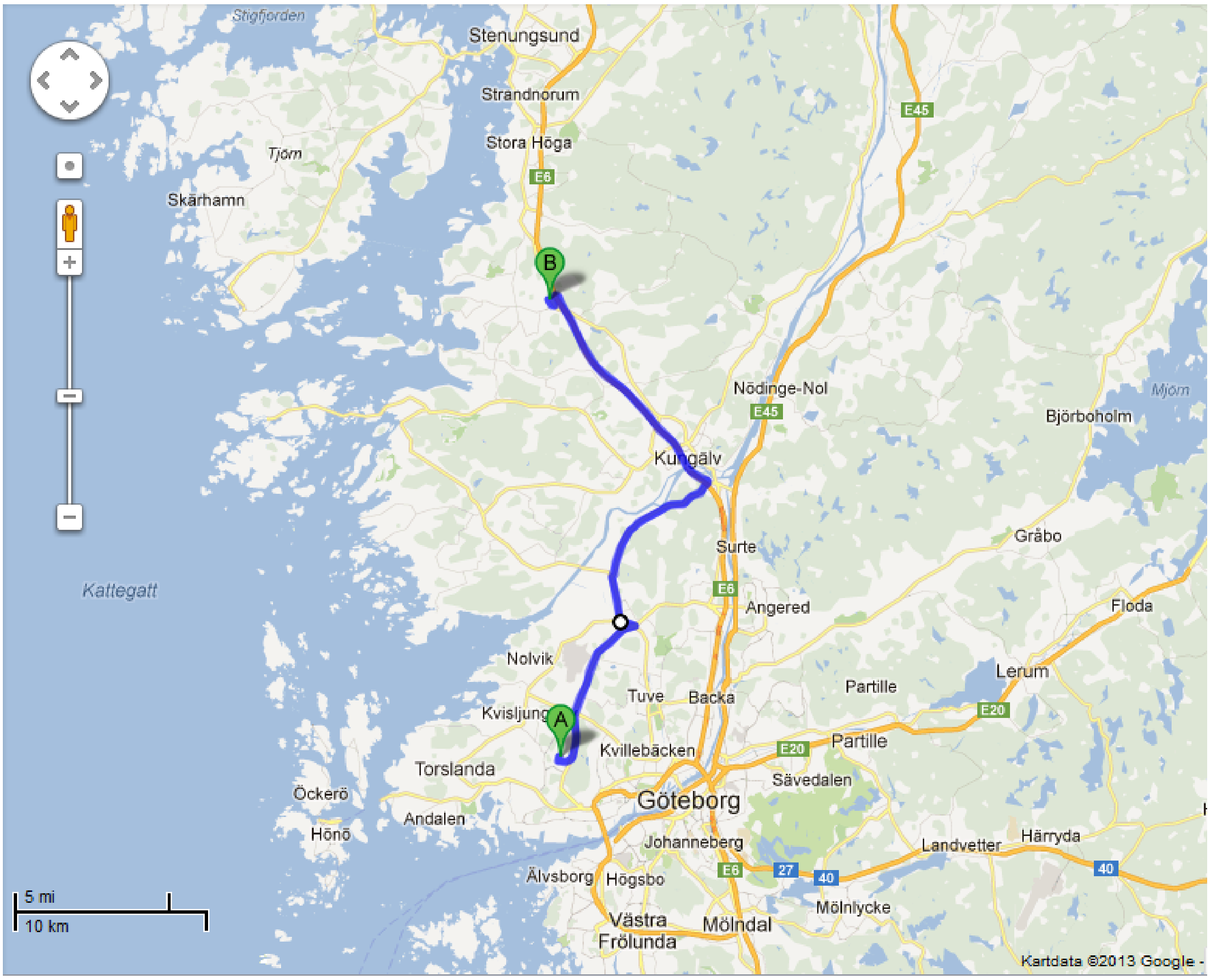


Figure 4.1.1: *Route that was used for evaluating the algorithm (Google Maps 2013).*

4.1.2 Data collection

Correct values for vehicle speed and accelerator pedal position are of great importance to get relevant input data for the simulations. The collection of vehicle data was however not performed with the intended test vehicle, but with an electric vehicle without any hybridization. This was necessary since the test vehicle was not available. The switch could be done since the two vehicles are equipped with the same accelerator pedal map, have equal external dimensions and have approximately the same mass. The pedal map is included in the vehicle software and calculates the requested traction torque based on vehicle speed and accelerator pedal position.

4.2 Model validation test results

As is mentioned in Section 3.2.3, model validation is necessary for the accuracy of results. The evaluation, based on a drive cycle performed at a test track of 5 km length, resulted in the curves for battery current and SOC presented in Figure 4.2.1. The black lines represent the measured values from the real vehicle and the red lines represent results from simulations.

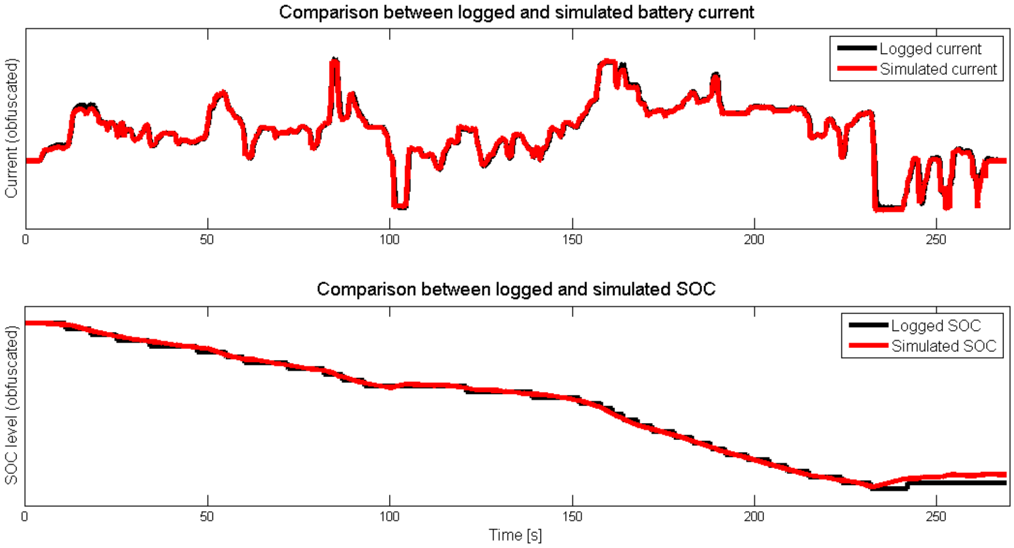


Figure 4.2.1: *Diagram presenting a comparison between measured and simulated values for battery current and SOC.*

The results from the simulation model and the real vehicle regarding the electric energy supply from the battery shows good compliance. Some variations exist, but for the purpose of this project they are considered to be acceptable. The largest deviation occurs at regeneration, i.e. when the battery is charged through conversion of kinetic energy to electric energy in the traction motor during braking. The difference is a result of overestimated regenerative braking at low speed in the simulation and this behavior can be corrected for future work.

The original vehicle simulation model is very detailed and extensive in order to produce accurate results. Some parts have not been used at all due to the modifications made to the model. For the parts that were used, some of the accuracy could probably have been sacrificed in order to get a faster simulation model, while mostly still retaining relevant information.

4.3 Algorithm test results

Initially, the intention was to perform tests on the whole system in real conditions. This would have included testing of computed strategies on the real vehicle along the test route. However, unexpected limited access to the test vehicle made such extensive testing impossible within the time frame of the project. Instead, simulation

Table 4.3.1: Parameter settings for the algorithm trials

Parameter name	Parameter value
Discount factor γ	0.99
Initial learning rate α_{init}	0.4
Final learning rate α_{final}	0.05
Initial Boltzmann temperature T_{init}	0.35
Final Boltzmann temperature T_{final}	0.005
Eligibility decay rate λ	0.6

tests were conducted.

The algorithm was tested in repeated trials with more than 1000 episodes each. Six trials using the same parameter settings were made in total. The parameter settings used are shown in Table 4.3.1. T and α reach their approximate final value after around 400 episodes.

In Figure 4.3.1, results from one of the trials is presented. As the plot with mean values shows, there seems to be a trend towards lower values of fuel consumption the longer the algorithm is run. Some anomalous low spikes exist in the beginning of the run, but low values increase in number over time. The increases in the mean value at certain points is probably due to the exploration behavior of the algorithm. However, since information about previous good paths is kept in the Q-matrix, returning to those paths is not difficult if the current path is unsatisfactory. The Q-matrix of the second algorithm trial is shown in Figure 4.3.2, with red areas indicating a higher expected reward for having the engine on and blue areas indicating the opposite.

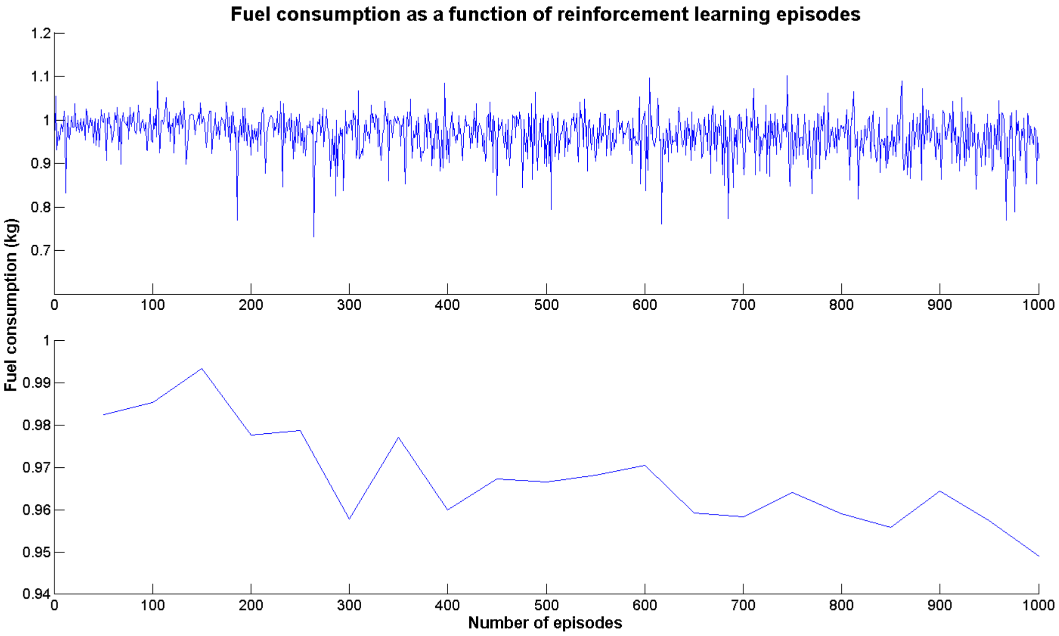


Figure 4.3.1: *Result from one of the algorithm trials, with the top plot showing raw data and the bottom plot showing the mean values over segments of 50 episodes.*

In order to evaluate the learning ability and usefulness of the algorithm, reference values were needed for comparison. Figure 4.3.3 shows a 1000 episode trial, but instead of using the reinforcement learning algorithm for action selection, actions were chosen uniformly at random. As would be expected from a random trial, none of the trending behavior from the algorithm trial is seen here. There seems to be a couple of low values, but with no learning present, the ability to obtain good results is dependent on luck. Since the number of possible paths rise exponentially with the number of action decisions, finding the optimal path becomes increasingly unlikely with bigger MDPs.

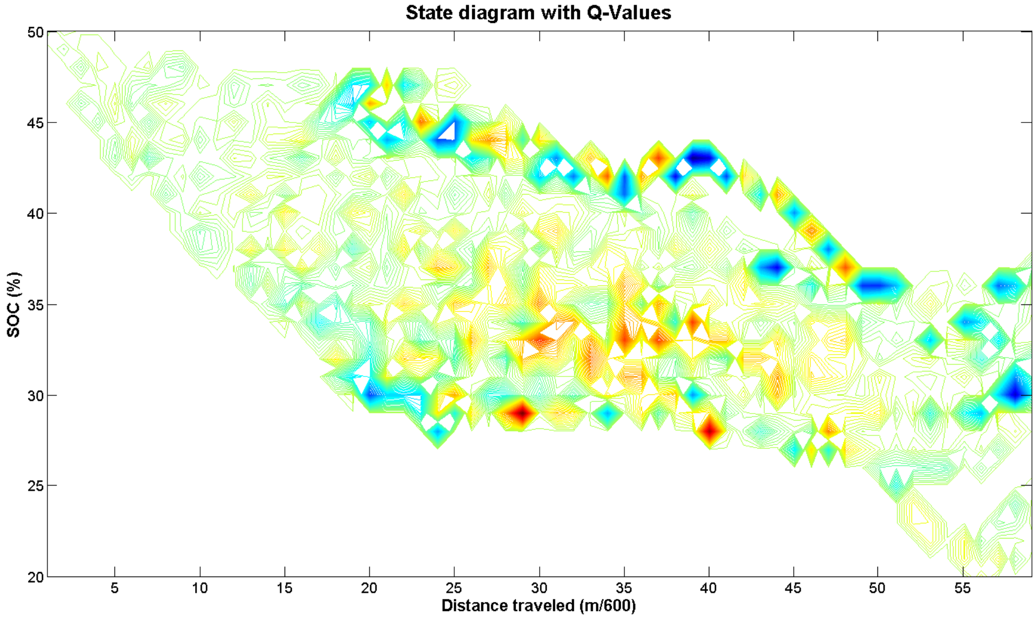


Figure 4.3.2: *The Q-matrix for the second trial after 1000 episodes.*

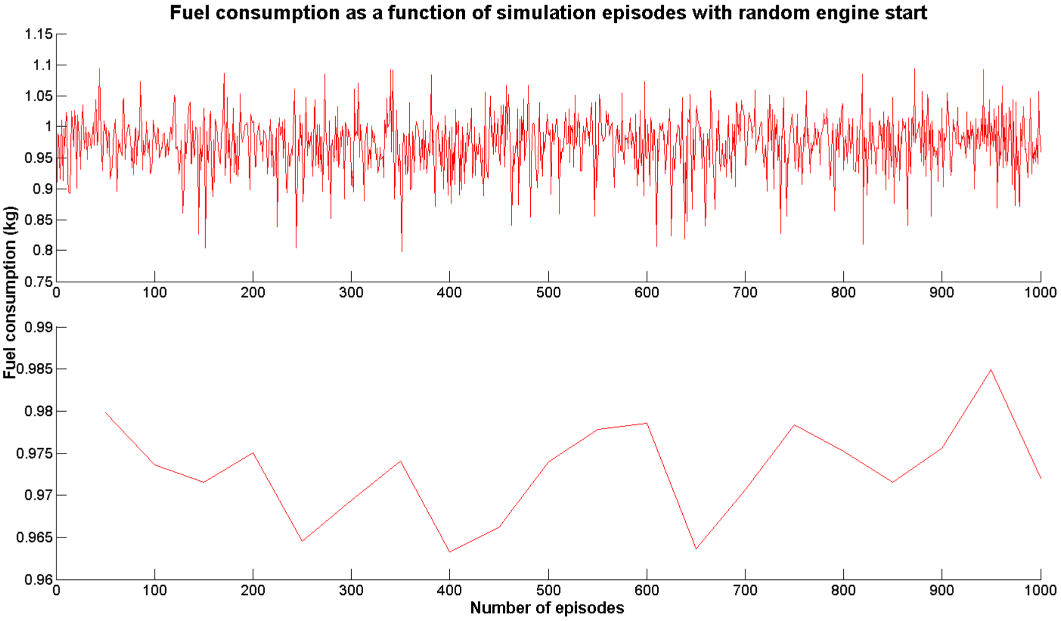


Figure 4.3.3: *A trial with random action selection, with the top plot showing raw data and the bottom plot showing the mean values over segments of 50 episodes.*

All six of the algorithm trials are shown together in Figure 4.3.4, confirming the trend observed in Figure 4.3.1. For reference, two other values are shown in the figure. The red line is the mean value of 2000 episodes with random action selection. The green line is the result of a static strategy starting the engine when the SOC level drops below 30% and then sustaining the level until the end of the drive. As can be seen in the figure, for this particular route, the random strategy is generally better than the static strategy. That would not be true for all routes, but in this case the initial SOC level combined with this particular route makes it beneficial to use the ICE for roughly half of the distance. Starting from a higher SOC level would make the static strategy better, since keeping the ICE off will naturally consume less fuel than randomly starting it. Still, the reinforcement learning algorithm ends up at a better mean value than both. Continued simulations with higher number of episodes are likely to result in even lower values.

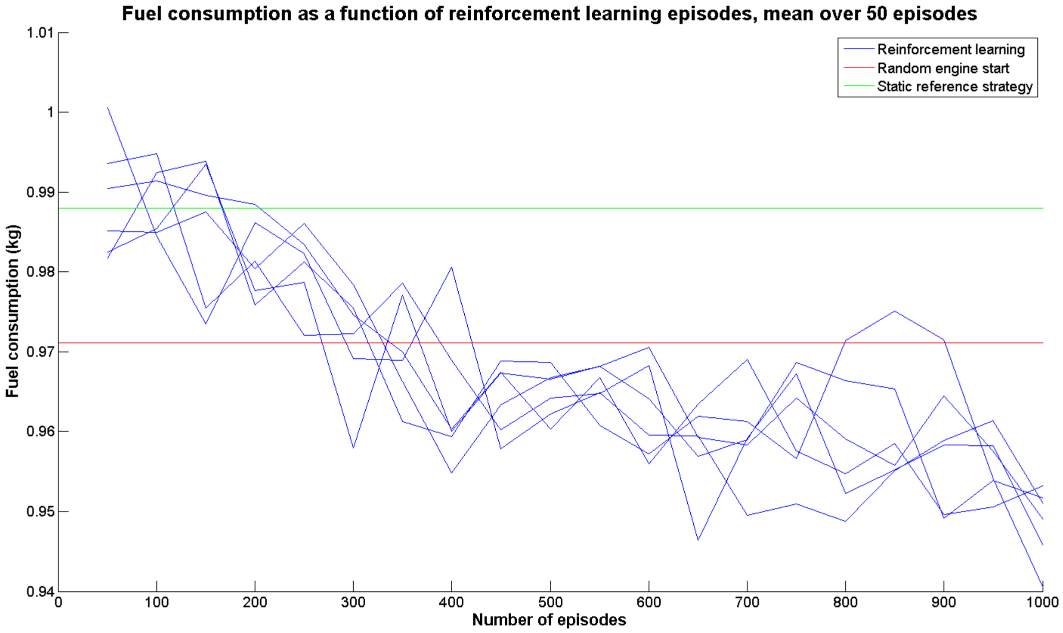


Figure 4.3.4: *The results of all the test trials.*

4.4 Error sources

Due to practical limitations during the project, the range extender simulation model could not be validated. This may have affected the results for fuel consumption that have been presented in this thesis. There is little reason to believe that the performance of the algorithm is affected, but using the results in a practical setting would make it necessary to have access to an accurate model.

5 Conclusion

By inspecting the results of this project, it can be concluded that most of the purpose of this thesis is fulfilled. A working system for control strategy deployment has been developed, served by a reinforcement learning algorithm computing strategies for engine state control. In addition, this is done remotely using a cloud service. All of these have been described thoroughly in the thesis.

Experimental results show the occurrence of beneficial learning when using reinforcement learning on the developed MDP. The performed trials consistently show a trend towards a lower final mean value of the fuel consumption than both reference strategies. More tests are however required to determine the extent of improvement.

Since the system has not been tested in a real setting, i.e. using the actual test vehicle on the test route, the accuracy and usefulness of the developed strategies is not confirmed. Neither are the used parameters confirmed to be the optimal ones for this domain.

For further development, an analytic solution combined with reinforcement learning is probably useful, utilizing known information about how vehicles behave during various circumstances. In addition, the computational resources of the cloud service would be better utilized if the simulation model could be run in the cloud. A faster and possibly distributed model would make the control strategy generation more efficient.

References

Ambühl, D. et al. (2010). “Explicit optimal control policy and its practical application for hybrid electric powertrains”. In: *Control Engineering Practice* 18.12, pp. 1429 –1439. ISSN: 0967-0661. DOI: 10.1016/j.conengprac.2010.08.003. URL: http://www.sciencedirect.com/science/article/pii/S096706611000184X (visited on 05/27/2013).

Coggan, M. (2004). *Exploration and Exploitation in Reinforcement Learning.* CRA-W DMP Project. Montreal, Quebec, Canada: McGill University. URL: http://www.cra.org/Activities/craw_archive/dmp/awards/2004/Coggan/FinalReport.pdf (visited on 05/13/2013).

Dayan, P and Watkins, C. (2001). “Reinforcement learning”. In: *Encyclopedia of Cognitive Science.* London, UK: MacMillan Press. URL: http://www.gatsby.ucl.ac.uk/~dayan/papers/dw01.pdf (visited on 05/13/2013).

Electric Power Research Institute (2007). *Environmental Assessment of Plug-In Hybrid Electric Vehicles. Volume 2: United States Air Quality Analysis Based on AEO-2006 Assumptions for 2030.* Publication No. 1015326. Palo Alto, CA: EPRI. URL: http://www.transportation.anl.gov/pdfs/TA/559.pdf (visited on 06/06/2013).

Google Inc. (2013). *Android, the world’s most popular mobile platform.* URL: http://developer.android.com/about/index.html (visited on 05/17/2013).

Google Maps (2013). *Southern Bohuslän.* URL: http://maps.google.com (visited on 06/17/2013).

Grześ, M. and Kudenko, D. (2008). “Robustness Analysis of SARSA(λ): Different Models of Reward and Initialisation”. In: *Artificial Intelligence: Methodology, Systems, and Applications.* Ed. by D. Dochev, M. Pistore, and P. Traverso. Vol. 5253. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 144–156. ISBN: 978-3-540-85775-4. DOI: 10.1007/978-3-540-85776-1_13. URL: http://dx.doi.org/10.1007/978-3-540-85776-1_13 (visited on 05/13/2013).

Guzzella, L. and Sciarretta, A. (2007). *Vehicle Propulsion Systems: Introduction to Modeling and Optimization.* 2nd. Springer Berlin Heidelberg. ISBN: 978-3-540-74691-1.

Hashemnia, N. and Asaei, B. (2008). “Comparative study of using different electric motors in the electric vehicles”. In: *Electrical Machines, 2008. ICEM 2008. 18th International Conference on*, pp. 1–5. DOI: 10.1109/ICELMACH.2008.4800157.

Heywood, J. B. (1988). *Internal Combustion Engine Fundamentals.* International Edition. Singapore: McGraw-Hill Book Co. ISBN: 978-0-07-100499-2.

Microsoft Corporation (2013). *Introducing Windows Azure.* URL: http://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/ (visited on 05/17/2013).

Renesas Electronics Corporation (2006). *Introduction to CAN.* Publication No. REJ05B0804-0100/Rev.1.00. URL: http://documentation.renesas.com/doc/products/mpumcu/apn/rej05b0804_m16cap.pdf (visited on 06/09/2013).

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* 1st. Cambridge, MA, USA: MIT Press. ISBN: 978-0-262-19398-6.

van Hasselt, H. P. (2011). “Insight in Reinforcement Learning. Formal analysis and empirical evaluation of temporal-difference algorithms”. PhD thesis. Utrecht, Netherlands: Utrecht University. ISBN: 978-90-39354964. URL: http://igitur-archive.library.uu.nl/dissertations/2011-0120-200243/hasselt.pdf (visited on 05/13/2013).

Volvo Car Corporation (2011). *Volvo Car Corporation develops Range Extenders for electric cars - adding 1,000 km extra range.* URL: https://www.media.volvocars.com/global/enhanced/en-gb/media/preview.aspx?mediaid=39105 (visited on 05/13/2013).

Wang, C.-F. et al. (2012). “A Permanent Magnet Integrated Starter Generator for Electric Vehicle Onboard Range Extender Application”. In: *Magnetics, IEEE Transactions on* 48.4, pp. 1625–1628. ISSN: 0018-9464. DOI: 10.1109/TMAG.2011.2173469.