ISSN 0280–5316
ISRN LUTFD2/TFRT--5499--SE

State of Charge Observer for Hybrid Vehicles

Björn Fälth

Department of Automatic Control
Lund Institute of Technology
March 1994

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* MASTER THESIS
	Date of issue March 1994
	Document Number ISRN LUTFD2/TFRT--5499--SE
Author(s) Björn Fälth	*Supervisor* M. Andersson, LTH and A. Romare, AB Volvo
	Sponsoring organisation

Title and subtitle

State of Charge Observer for Hybrid Vehicles. (Batteristatusobserverare för Hybridfordon.)

Abstract

As a part of a research project, AB Volvo Technological Development is investigating the properties of traction batteries of different types. The aim is to find a suitable energy storage for electric/hybrid vehicles.

This master thesis examines the possibility of using a non-linear model of a traction battery to construct a device that accurately estimates the battery State of Charge (SoC) under typical hybrid operating conditions, i.e. the battery is rarely fully charged or discharged.

The work has consisted of three major parts. First, the battery model was fitted to measurement data from a specific NiCd battery cell. This model was then used to design a state derivative feedback observer which is implemented in a micro controller.

Certain precautions were taken when generating the source code, so that large parts of the code can be reused with other battery management applications.

The laboratory tests give good results compared to simpler methods, especially in the case of complicated discharge patterns or if the battery has aged and lost capacity. The observer method can, however, be further refined with the use of more complex models.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280–5316		*ISBN*
Language English	*Number of pages* 53	*Recipient's notes*
Security classification		

Acknowledgements

I would like to use this opportunity to express my gratitude to Mr. Anders Romare of AB Volvo Technological development for making this project possible, and for his guidelines on how it should be carried through.

Further, I would like to thank Mr. Patrik Larsson and Dr. Per Ekdunge, also of Volvo TU, for the invaluable help on both theory and common practice of NiCd cells. Furthermore, thanks to Lic. Anders Lasson and Mr. Johan Widell for many interesting discussions throughout this work.

I also appreciate the help from my tutor Mr. Mats Andersson at the Department for Automatic Control, Lund Institute of Technology.

The financial support from AB Volvo Technological Development is gratefully acknowledged.

Table of contents

Acknowledgements 1

Table of contents 2

1. Introduction 3

2. Background 4

2.1 Traction batteries 4
2.2 Battery State of Charge 5
2.3 Battery model 6
2.4 Micro controller 7
2.5 Signal conditioning unit 7

3. Development of a SoC observer 9

3.1 Finding a suitable model 9
3.2 Analysis of the battery model 12
3.3 SoC observer design 15

4. Simulation results 20

4.1 Battery model 20
4.2 SoC Observer 22

5. Measurement results 25

6. Program development 29

6.1 General outline. 29
6.2 EPC 166 Micro controller. 30

6.2.1 EPC interface 30

6.2.2 SoC observer 32

6.3 IBM PC 33

6.3.1 PC interface 33

6.3.2 Control program 35

7. Conclusions 37

8. Recommendations for future work 38

9. References 39

Appendix B: Standard discharge cycles. 40

Appendix C: Source Code listings 46

1. Introduction

This master thesis completes my studies in electrical engineering at the Lund Institute of Technology (LTH). I have carried out the work at AB Volvo, Technological Development in Göteborg, with supervisors at both Volvo and at the Department of Automatic Control (LTH).

As a part of their hybrid vehicle project, Volvo TU are performing tests on traction batteries from several different manufacturers. The aim is to thorough determine the properties and qualities of these batteries, in order to find a suitable energy storage for a hybrid vehicle.

The aim of my project was to develop an instrument that accurately estimates the amount of electrical charge in a Nickel-Cadmium (NiCd) cell during typical hybrid operating conditions: A State of Charge (*SoC*) observer. The cell is manufactured in France by SAFT-NiFe, and is delivered in 6V batteries.

The project plan consisted of a number of steps:

- To enhance and finish a simulation model of the SAFT battery.
- To develop the observer in the MATLAB/SIMULINK[1] environment.
- To code the observer in C with the new MATLAB/C-code Generator.
- To adopt the C code to Volvo's micro controller based evaluation board.
- To evaluate the performance of the observer in the laboratory.

This original plan had, however, to be slightly modified since Math Works Inc. were not able to deliver the C-code generator as planned. I instead put in some extra work into producing a useful interface towards the evaluation board, thus making it easier to handle for future programmers.

In the following report I will present the background of the project, as well as the hardware I had available (Chapter 2). Chapter 3 contains my experiences from the development of the observer in the SIMULINK environment. Chapter 4 contains simulation results and Chapter 5 measurement results from the laboratory. In Chapter 6, I explain the outline of the C-code interfaces, and how they are used. My conclusions from the project are found in Chapter 7. Chapter 8 contains my suggestions for further work with the evaluation board, and how to generate C-code for it.
I hope you will enjoy reading the rest of this report.

[1] MATLAB and SIMULINK are trademarks of Math Works Inc.

2. Background

Before this project could begin, some prerequisites had to be fulfilled. In this chapter I explain the background of the project, and the equipment I had available to carry it through.

2.1 Traction batteries

As of today, there are no existing traction batteries that are good enough to be used in an electric vehicle that is competitive with a conventional one. If you, e.g., consider the time it takes to fill a petrol tank, and compare it to the charging time of a battery, the drawbacks are obvious. The two dominating types of batteries in use today: Lead-Acid and Nickel-Cadmium, also contain large amounts of heavy metals, which make them potentially hazardous to our environment. Their fast ageing as well as their large sizes and weight, are also problems characteristic to the traction batteries of today.

However, as the problems with emissions from fossil fuel are increasing, the authorities in several countries are putting heavy restrictions on how much we can allow a vehicle to pollute the air in traffic intense areas. A good example is Los Angeles, California, where the aim is to completely remove the air pollution caused by vehicles. This, and of course other contributing factors, have put a pressure on the automotive industry to develop new technologies for batteries and to find new concepts for powering vehicles.

Since the big breakthrough on the battery side is not yet here, large efforts are being made to develop a so called hybrid vehicle. An electric motor propels the vehicle with the power taken from a conventional battery. The battery can be charged from an external source or from an Internal Combustion Engine (ICE), which is coupled to a generator. This configuration brings some large advantages:

- Pure electrical operation in city traffic, giving zero emissions.
- Continuous charging extends the driving range outside the cities.
- The batteries can be optimised for maximum power delivery since the ICE can provide more energy, when needed.
- ICE type can be chosen for minimum emissions without loosing performance. (gas-turbine, Otto, Sterling, e.t.c.)
- Reduced ICE size.
- Possibility to chose optimum working points for the ICE.

To fully take use of these advantages, a good control strategy [16] for the energy flow has to be developed to further economise the energy consumption and to extend the battery's life time. An important requirement for this controller is that it has an accurate estimate of the battery State of Charge at all times.

2.2 Battery State of Charge

State of Charge (*SoC*), is a relative quantity for expressing the amount of charge in a battery.
It is impossible to directly measure this quantity. There also exists a variety of definitions for *SoC*, which makes it difficult to compare the quality of different estimates of *SoC*. A comprehensive definition is that a fully charged battery should have *SoC* equal to one, and a completely discharged battery should have *SoC* equal to zero. In this thesis I have used a definition used by Volvo: *SoC* equals 1.00, right after the battery is charged according to the specifications from the manufacturer. *SoC* equals 0.00 when the battery is no longer capable of delivering any energy into the load. This leads to two definitions of charge: Q_b is function of the load current, and expresses amount of charge which is blocked within the battery (not available as current) and Q_s which is the charge that has been extracted from the battery:

$$\begin{aligned} Q_b &= f(i(t)) \geq 0 \\ Q_s &= \int_0^{T} i(t)dt \end{aligned} \tag{2.1}$$

If we normalise these quantities with Q_0 which is the actual capacity of the battery:

$$\begin{aligned} Q_0 &= \int_0^{\infty} i(t)dt \\ i(t) &= I = \frac{C[Ah]}{5h} \end{aligned} \tag{2.2}$$

where C is the nominal capacity in Ampere-hours, we can define the State of Charge as:

$$SoC = 1 - \frac{Q_s + Q_b}{Q_0} \leq 1 \tag{2.3}$$

Another quantity which is common in the literature, is the Depth of Discharge (*DoD*). In this report *DoD* is defined as:

$$DoD = \frac{Q_s + Q_b}{Q_0} = 1 - SoC \tag{2.4}$$

A common way of estimating *SoC* is to integrate the load current. This is called the Ampere-hour method which is defined as:

$$SoC = 1 - \frac{1}{C}\int_0^T i(t)dt \tag{2.5}$$

There are two obvious reasons why this method often give inaccurate results: It does not consider temporary capacity losses due to blocked charge, and more important, it does not account for diversions in actual capacity Q_0 from the nominal value *C*.
These are the reasons why model based methods (Chapter 3.3) are used to provide a more accurate estimate of *SoC*.

2.3 Battery model

Volvo has developed a general model of a battery cell, which has been implemented in the simulation software MATLAB/SIMULINK.
A system of ordinary first order differential equations and non-linear algebraic equations describe the model. With the current demand from, e.g. an electric motor as input, it models both the fast and slow dynamics of the NiCd battery during discharge and charge. The model uses five states which are: remaining and used capacity, electrolyte concentration, temperature and age. It then calculates the electric quantities from the states. The battery terminal voltage is the output from the model. Figure 2.1 shows an equivalent two pole network of the battery.

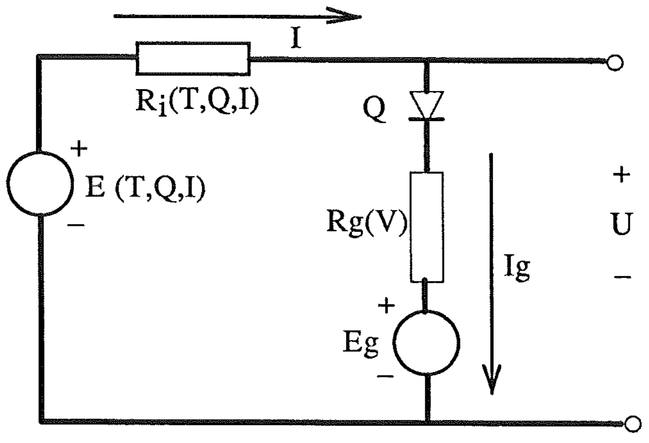


Figure 2.1: Equivalent two pole network of a battery cell. E is the electromotive force, Ri is the internal resistance. The current Ig through the limb Q-Rg-Eg represents the gas losses that occur when the battery is charged.

The overall internal resistance R_i and the electromotive force E depend non-linearly on temperature T, charge Q, and current I. When the terminal voltage U exceeds the gassing potential E_g, an equivalent gassing current I_g starts to flow through R_g which depends on the electrolyte concentration V. This current does not contribute to the increment of the available charge Q.

2.4 Micro controller

For implementation in a controller, the evaluation board EPC-166 [14] with the necessary software environment [13] is available.
It is based on the Siemens SAB-80C166 processor [15], and is operated via one full length ISA-card slot of an IBM PC. The board can communicate with the outer world through a 96-pin connector available at the back of the PC. When used in battery management applications, such as in this thesis work, the 96-pin connector is hooked up to an analogue signal-conditioning unit.
The main features of the EPC166 are:

- 256 kbyte on board RAM, which is directly mapped into extended memory on the PC.
- 10 bit A/D converter.
- Operating conditions are set by software through a hardware register, which appears in PC extended memory.
- The micro processor is high level programmable through a C-code compiler/debugger kit from KEIL Elektronik, Munich, Germany.

These features make the board easy to work with, and different applications can be developed rather quickly.

2.5 Signal conditioning unit

The A/D-converter which is integrated in the 80C166 processor accepts only voltages between 0V and +5V. The quantities temperature, voltage and current therefore have to be transformed into this voltage range. This is done in two steps: First, a set of sensors produce the three electric quantities Ulin, Ubat and Ilem as shown in Figure 2.3.

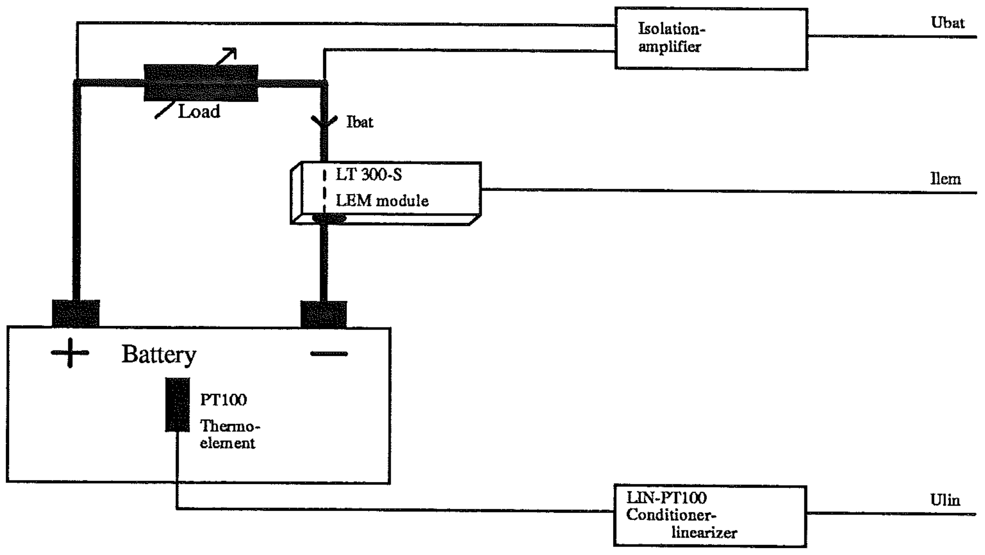


Figure 2.3: Sensor configuration for measuring the quantities needed in a battery management system.

Second, these quantities are transformed into the permissible voltage range with a set of Op-Amp circuits. A complete description of the signal conditioning unit is found in [2].

Figure 2.4 shows an outline of the equipment I use to collect an process data in the SoC observer.

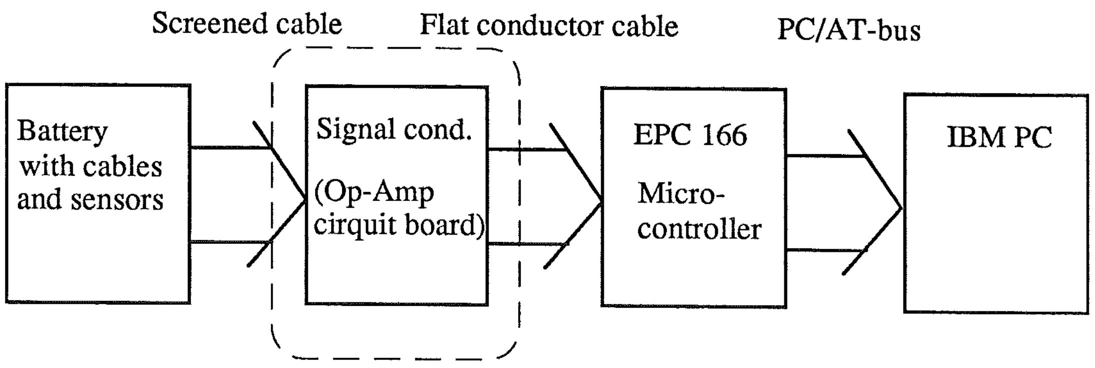


Figure 2.4: A battery management system based on the EPC 166 micro controller. The EPC board only accepts voltages between 0V and +5V, which makes the use of a signal-conditioner (dashed frame) necessary.

3. Development of a SoC observer

The State of Charge observer is designed as a continuous time state derivative feedback observer. It has been developed from a (non-linear) model of the battery cell, which was implemented in MATLAB/SIMULINK. Both battery model and *SoC* observer are analysed by means of a linear model.

3.1 Finding a suitable model

As I mentioned in chapter 2.3, the battery model developed by Volvo, consists of ordinary first order differential equations and algebraic equations. There are three state equations, which describe the extracted charge *Qs,* the blocked charge *Qb* and the electrolyte concentration *V*.

$$\begin{aligned} &\frac{dQ_s}{dt} = I \\ &\frac{dQ_b}{dt} + C_q Q_b = f_q(I) \\ &\frac{dV}{dt} + C_v V = C_v \frac{I}{J} \\ &t = 0 \Rightarrow Q_s = Q_b = V = 0 \end{aligned} \qquad (3.1)$$

The function f_q is chosen so that the empirical Peukert relation $Q = k \cdot I^n$ [8] is obtained for constant current discharging in the medium and high current range. In the case of the NiCd battery, $f_q(I) = kI^n$ where k and n remain to be found.
J is the exchange current flow between the electrolyte and the bulk.
Since Lead/Acid batteries normally are very sensitive to temperature changes, the model originally contained the bulk temperature as a state variable. Measurements on the SAFT NiCd battery showed, however, that when the temperature stayed within reasonable limits, it had very little effect on *EMK* and internal resistance. I have therefore excluded the temperature from the battery model.

With *DoD* defined as in (2.4) and *SoC* as in (2.3)the algebraic equations for the internal resistance *R,* and electromotive force *E*, may be written as:

$$\begin{aligned} &R = R_0 + R_1 \frac{Q_s}{Q_0} + \frac{R_2 + R_3 DoD * V}{1 - V} \\ &E = E_0 + E_1 \ln SoC - E_2 V \end{aligned} \qquad (3.2)$$

The first two terms in the expression for *R* represents the resistance in the electrolyte, and the last the resistance in the electrode. The expression for *E* is motivated by the Tafel equation [8], which indicates that the *EMK* decreases with the logarithm of the current density which is inversely proportional *SoC*. The last term models the effect of the increasing concentration gradients in the electrolyte as the battery discharges.

The equations above are general equations which can be applied to a number of different types of cells. In order to fit them to a specific battery, the time constants of the equations in (3.1) have to be determined. The constant values in the equations in (3.2) then have to be found.

I have used two dynamic discharge cycles and two constant current discharge cycles as references when fitting the model. All of them were recorded in Volvos battery laboratory. I have also used measurement results from other sources as an additional verification of the battery model.
A thorough description of the discharge cycles are found in appendix B.

Time constants:
The system's two time constants, describe the behaviour of blocked charge and electrolyte concentration. They are a bit tricky to determine because the two states influence the terminal voltage similarly.

Starting with C_V, it is best determined through an experiment, where the battery is discharged with a fairly small current, i.e. C/5. Such a small current is believed to block very little charge, but to have a significant influence on the concentration. After a certain time, (about 45 min to make sure that steady state is reached) the current is cut off, and we record the voltage rise. After subtracting the voltage drop over the internal resistance, the voltage rise is to the largest extent due to changes in the electrolyte concentration since *SoC* is still high (see (3.2)). The time constant C_V, is thus determined directly from the graph in Figure 3.1.

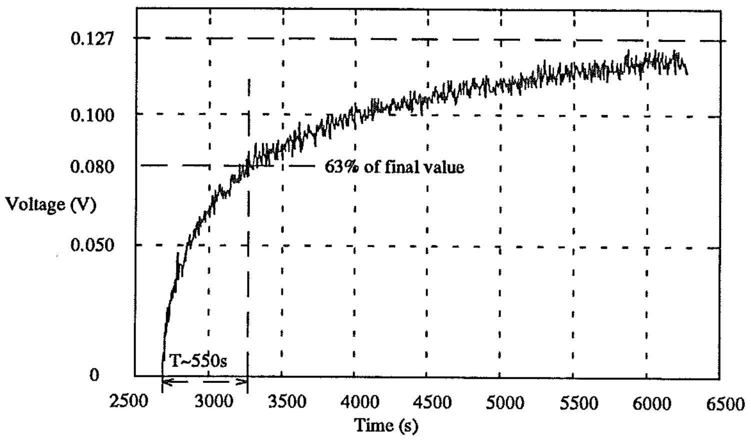


Figure 3.1: Graphical determination of the time constant in the expression for the electrolyte concentration.

Here, the voltage rise is plotted against the time. From the figure, I estimate the final value to be 0.127 V. The time constant T is found as the time it takes the voltage to reach 63% of the final value, or 0.080 V. Through a graphical estimation on the plot above, I found C_V to be approximately 1/550 s^{-1}. Based on the assumption that J is about 500 A, which is fairly accurate according to [1], E_2 in (3.1) can be calculated as

$$E_2 = \frac{U^0}{V^0} = U^0 \frac{J}{I} = 0.127 \frac{500}{28} V = 2.27V$$

The time constant C_Q can be found in a similar way, but with a much larger current (140 A). If performed in the low *SoC* range and with a good estimation of E_2 in (3.1), a curve for the voltage rise caused by unblocking of charge may be plotted after subtraction of the voltage rise from the concentration and the voltage drop over the internal resistance. My measurements and calculations gave, however, uncertain results, and I decided to use the value from the original model: $C_q = 8.33 \cdot 10^{-4} s^{-1}$

The factors k and n in $f_q(I) = kI^n$ of equations (3.1) can be determined by measuring the amount of charge that can be extracted with different load currents, and subtracting this from the maximum extractable capacity. From (3.1), we find that the blocked charge reaches balance at:

$$Q_b = \frac{k \cdot I_0^n}{C_q} \tag{3.3}$$

with discharge current I_0. MATLAB simulations in comparison with measurements on the battery with the discharge currents 1C, C/2, C/3 and C/5, have given k=1.0e-3. and n=1.6, which seems to well model the blocked charge in a wide range of currents.

Parameters:

Finding the parameters E_i and R_i in (3.2) is more or less pure curve fitting. Through a number of simulations and comparisons to measurements of voltage and internal resistance during different charge- and discharge cycles I have fine tuned the original parameters which were roughly estimated in [1]. I finally settled for the expressions which you find in Appendix A.

The more complicated dependencies on *DoD* in both E and R than in (3.2), were introduced in the model before I overtook it. They were necessary to better describe the sharp cut-off in voltage, as *SoC* approaches zero. A list of the final parameter values are also presented in appendix A. It is noticeable that some of them are different during charging. The internal resistance, for instance, is much larger during charging, than during discharging.

The model is now capable of calculating the battery terminal voltage during constant current charge and discharge, as well as during slower dynamic cycles within certain limits, by measurement of the current only. The model is valid for currents in the range [-150,30] Ampere, where the upper limit is valid only in the *SoC* range [0.0, 0.8]. This is because gas losses are not considered when the battery is charged. Since temperature effects are ignored, the model is not valid outside [0,+50] °C.
The model does not consider ageing or memory of previous discharges of the battery. It is thus necessary to introduce some sort of feedback in order to convert the model into a reliable SoC observer that can operate over a longer period of time.

3.2 Analysis of the battery model

After an acceptable simulation model is found, an analysis of the result is inevitable. In order to get a hint of what the system looks like, I have made a linear model around *SoC*=0.5 and zero current:

$$\dot{x} = Ax + B \cdot I$$
$$U = Cx + D \cdot I$$

where *U* and *I* are battery terminal voltage and load current respectively.
To simplify the calculations, and to get numbers of the same magnitude in the matrixes, I have made a transformation and normalisation of the state variables:

$$DoD_r = \frac{Q_s}{Q_0}$$
$$DoD_a = \frac{Q_s + Q_b}{Q_0} \qquad (3.4)$$
$$Q_0 = 5.76 \cdot 10^5\, As$$

Q_0 is the small current discharge capacity. The state space equations can then be expressed as:

$$\begin{pmatrix} \frac{dDoD_a}{dt} \\ \frac{dDoD_r}{dt} \\ \frac{dV}{dt} \end{pmatrix} = \begin{pmatrix} -C_q & C_q & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -C_v \end{pmatrix} \cdot \begin{pmatrix} DoD_a \\ DoD_r \\ V \end{pmatrix} + \begin{pmatrix} -\frac{k_q+1}{Q_0} \\ -\frac{1}{Q_0} \\ \frac{C_v}{J} \end{pmatrix} \cdot I \tag{3.5}$$

$$U = E(DoD_a, V) + R(DoD_a, DoD_r, V) \cdot I$$

where $f_q(I) = kI^n$ is substituted with $f_q(I) = k_q I$=3.8e-3I, which is a good approximation in the medium current area. The battery terminal voltage $U = E + R \cdot I$ is system output, but since the expressions for E and R, are highly non-linear, U first has to be linearised:

$$\Delta x = \begin{pmatrix} DoD_a - DoD_a^0 \\ DoD_r - DoD_r^0 \\ V - V^0 \end{pmatrix}$$
$$\Delta I = I - I^0 \tag{3.6}$$
$$\Delta U = \begin{pmatrix} \frac{\partial U}{\partial DoD_a} & \frac{\partial U}{\partial DoD_r} & \frac{\partial U}{\partial V} \end{pmatrix} \cdot \Delta x + \frac{\partial U}{\partial I} \Delta I$$

With equilibrium chosen at:

$$\begin{aligned} I^0 &= 0 \\ DoD_a^0 &= 0.5 \\ DoD_r^0 &= 0.5 \\ V^0 &= 0 \end{aligned} \tag{3.7}$$

and inserted in (3.5), the linear system becomes:

$$\begin{pmatrix} \frac{dDoD_a}{dt} \\ \frac{dDoD_r}{dt} \\ \frac{dV}{dt} \end{pmatrix} = \begin{pmatrix} -8.33 \cdot 10^{-4} & 8.33 \cdot 10^{-4} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1.8 \cdot 10^{-4} \end{pmatrix} \cdot \begin{pmatrix} DoD_a \\ DoD_r \\ V \end{pmatrix} + \begin{pmatrix} -1.742 \cdot 10^{-6} \\ -1.736 \cdot 10^{-6} \\ 4.0 \cdot 10^{-6} \end{pmatrix} \cdot I$$

$$U = \begin{pmatrix} -0.4654 & 0 & 1 \end{pmatrix} \begin{pmatrix} DoD_a \\ DoD_r \\ V \end{pmatrix} + 2.559 \cdot 10^{-3} I \tag{3.7}$$

The Bode diagram (Figure 3.1) for the system (3.7) indicates that the battery operates very much like a pure integrator. Equal amounts of zeros and poles, make the phase shift tend to zero from -90 deg, as the frequencies approach infinity.

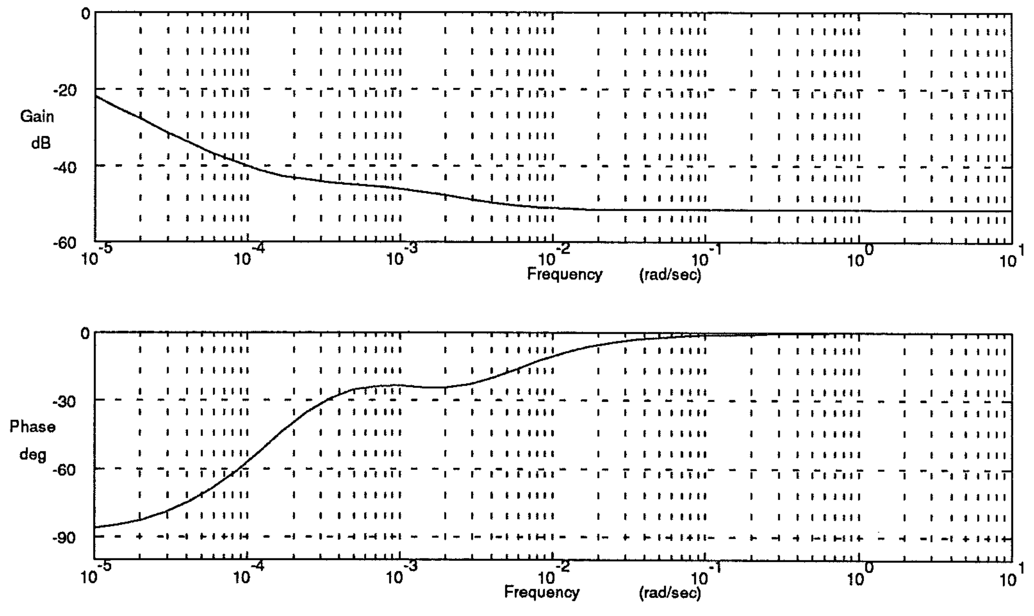


Figure 3.2: Bode diagram of the linearised battery model. For low frequencies, the battery operates as a pure integrator.

This is also confirmed as we look at the pole-zero plot in Figure 3.3. The two faster poles belong to the blocked charge, and electrolyte concentration.
The high damping of frequencies above 1.6mHz suggests that a sample rate of 1 second should be more than sufficient to catch all dynamics.

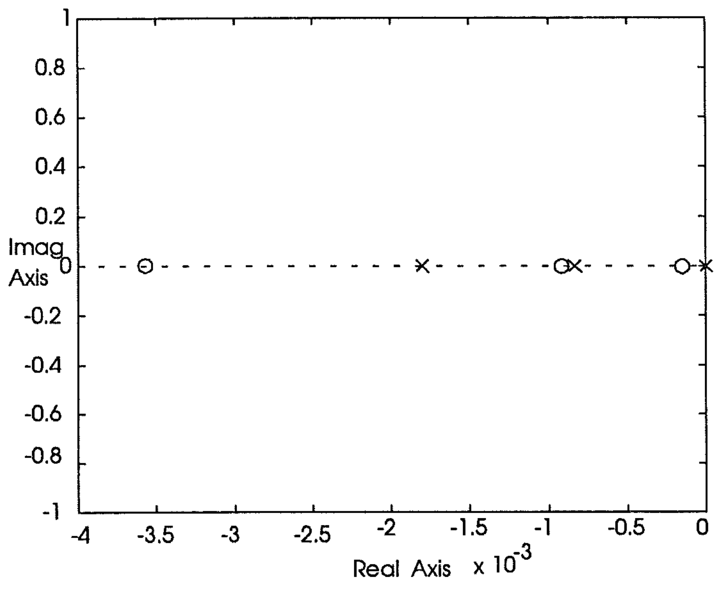


Figure 3.3: Pole-Zero plot of the linearised battery model.

I will use this linear model to design, and investigate the properties of the *SoC*-observer which follows in chapter 3.3.

3.3 SoC observer design

When developing a SoC observer based on the battery models, as concluded in [5], three fundamental problems have to be solved,

- The construction of a good dynamic model of the battery.
- Accurate measurements of terminal voltage, current and possibly even electrolyte- and environmental temperature.
- To perform a suitable feedback of the error between measured and estimated terminal voltage.

P. Larsson and U. Andersson [2] have built the necessary equipment for accurate measurement of current, voltage and temperature. This set-up is available at Volvo and is used for testing the SoC observer in the laboratory.

I have considered three different strategies for estimating the State of Charge:

- Translating the Simulink model to a discrete linear state-space representation and using a Kalman filter to reconstruct the states of the battery from measured values of current and voltage.
- Using the Simulink representation of the battery to predict the voltage from measured current and perform a feedback of the voltage error to the charge derivative as proposed in [4] and [5]
- As above, but with feedback to all states, using traditional observer technique.

The first approach is proposed in [3]. Here, a lead acid battery is modelled as an equivalent RC-network, which is translated into a discrete, third order state space model of the form

$$\begin{aligned} x(k+1) &= \Phi(k) + \Gamma i(k) + v(k) \\ u(k) &= Cx(k) + Di(k) + e(k) \end{aligned} \tag{3.8}$$

where SoC is chosen as one of the states. *i(k)* is the load current and *u(k)* is the terminal voltage. The matrixes C and D are time varying.
Now, the Kalman filter algorithm

$$\begin{aligned} K(k) &= P(k)C^T(k)[C(k)P(k)C^T(k) + R(k)]^{-1} \\ x^+(k) &= \hat{x}(k) + K(k)[u(k) - \hat{u}(k)] \\ P^+(k) &= [I - K(k)C(k)]P(k) \\ \hat{x}(k+1) &= \Phi(k)x^+(k) + \Gamma(k)i(k) \\ P(k+1) &= \Phi(k)P^+(k) + \Phi^T(k) + Q \end{aligned} \tag{3.9}$$

is used to reconstruct the state vector $\hat{x}(k+1)$, from where SoC may be extracted. This method is fairly insensitive to differences between model and reality, but it needs large variations in the current that is pulled out from the battery to effectively reconstruct the state vector. When the battery is used for traction purposes, this is, however, often the case.

The third method of calculating SoC is somewhat of a simplified version of the method described above. It does not produce an optimal estimate in the same sense as a Kalman filter does.
We use the Simulink model of the battery, which better models the output voltage than a linear model. The voltage error is then used for feedback on the state derivatives. Figure 3.4 shows a block diagram of this observer. Because of the

superior modelling capacity of the non-linear model, this observer should produce better results than the Kalman-filter [3] or the single state feedback observer in [5].

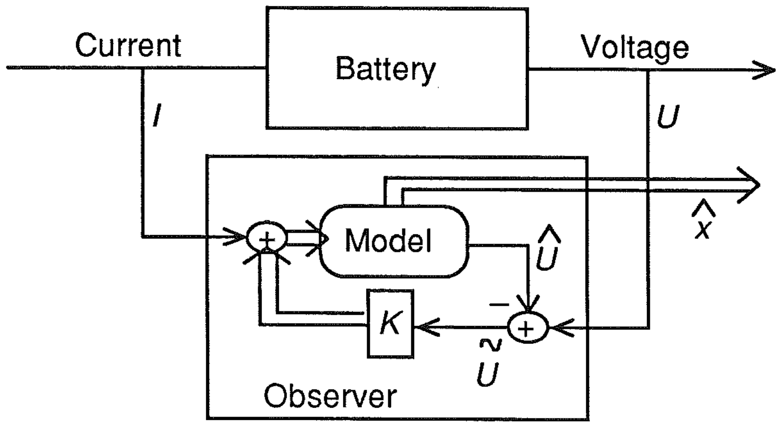


Figure 3.4: Block diagram of the SoC observer. State of Charge is extracted from the estimated state vector $\hat{x}$.

It is however difficult to find a suitable feedback vector *K* which gives a fast but stable and smooth response to the output error $\tilde{U}$. I have thus used the linear model (3.7) to investigate the location of the observer poles for different values of *K*.

If we denote the estimated state vector as $\hat{x}$, the state equation for the closed loop observer becomes:

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + BI + K(U - C\hat{x} - DI) \\ \dot{\hat{x}} &= (A - KC)\hat{x} + (B - KD)I + K \cdot U \end{aligned} \tag{3.10}$$

In order to get a stable reconstruction, the matrix (*A-KC*) must have its eigenvalues left of the imaginary axis. Through simulation, I have found promising values on the elements of the gain vector as:

$$K = -\begin{pmatrix} 2.0 \\ 4.0 \\ 0.2 \end{pmatrix} \cdot 10^{-3} \tag{3.11}$$

which places the observer poles as in Figure 3.2. The observer is thus considerably faster than the battery. The question is how fast it recovers from errors and how measurement- and model errors influence the estimated state vector.

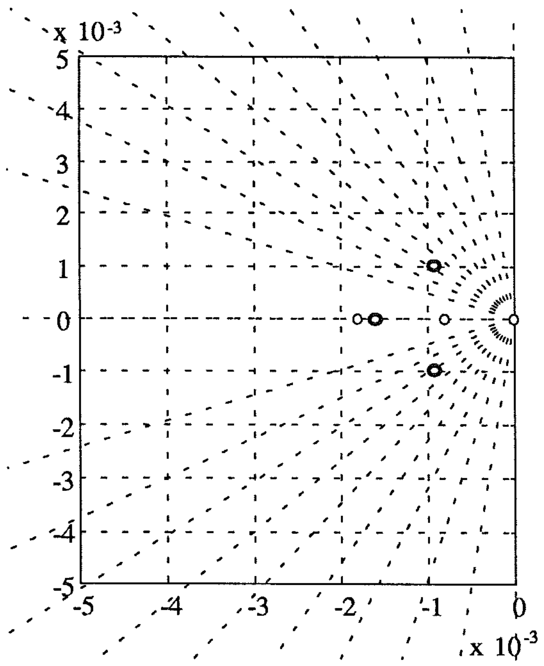


Figure 3.2: Location of the observer poles (thick) and original poles of the battery model (thin).

An investigation of the reconstruction error $\tilde{x}$ gives:

$$\tilde{x} = x - \hat{x}$$

$$\dot{\tilde{x}} = \dot{x} - \dot{\hat{x}} = Ax + BI - (A-KC)\hat{x} - (B-KD)I - K(Cx+DI) \tag{3.12}$$

$$\dot{\tilde{x}} = (A-KC)x - (A-KC)\hat{x} = (A-KC)\tilde{x}$$

The reconstruction error is governed by the eigenvalues of (*A*-*KC*). These are in fact the same as the observer poles, which are plotted in Figure 3.2. The complex conjugated pole pair is located at a distance $\omega_0 = 1.3 \cdot 10^{-3} s^{-1}$ from orig., and the pole on the real axis is located at $\omega_0 = 1.6 \cdot 10^{-3} s^{-1}$ from orig. This implies that the reconstruction error decreases with a time constant in the neighbourhood of 12 minutes. The damping of the complex pole pair is approximately $\zeta = 0.7$, which should produce an estimate without excessive overshot.

The effect of measurement errors can be investigated from (3.10). When in steady state, $\dot{\hat{x}} = 0$, we get:

$$\hat{x} = (A-KC)^{-1}(B-KD) \cdot I + (A-KC)^{-1} K \cdot U \tag{3.13}$$

With the values from (3.7) and (3.11) inserted in (3.13), we get:

$$\hat{x} = \begin{pmatrix} 9.76 \\ 11.93 \\ 2.32 \end{pmatrix} \cdot 10^{-3} \cdot I + \begin{pmatrix} 2.15 \\ 2.15 \\ 0 \end{pmatrix} \cdot U \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{3.14}$$

Equation (3.14) shows that the voltage measurement is crucial to the quality of the estimate: A 1% offset error in the measured voltage gives roughly 2% error in the estimated states DoD_a and DoD_r.

The effect of model error also has to be investigated. If we introduce the error δ to the time constant in the equation for the electrolyte concentration as:

$$\Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \delta \end{pmatrix}$$

The expression for the estimation error (3.12) follows as

$$\begin{aligned} \dot{\tilde{x}} &= Ax + BI - (A + \Lambda - KC)\hat{x} - (B - KD)I - K(Cx + DI) \\ \dot{\tilde{x}} &= (A - KC)x - (A + \Lambda - KC)\hat{x} = (A + \Lambda - KC)\tilde{x} - \Lambda x \end{aligned} \tag{3.15}$$

When the system has reached steady state, we get

$$(A + \Lambda - KC)\tilde{x} = \Lambda x \tag{3.16}$$

With the values from equations (3.7) and (3.11) inserted in (3.16)

$$\begin{pmatrix} -1.764 & 0.833 & 2.000 \\ -1.862 & 0 & 4.000 \\ -0.093 & 0 & \delta - 1.600 \end{pmatrix} \cdot 10^{-3} \tilde{x} = \begin{pmatrix} 0 \\ 0 \\ \delta x_3 \end{pmatrix}$$

Solving these equations leads to the expression below:

$$\tilde{x} = \frac{\delta}{\delta - 1.8 \cdot 10^{-3}} \cdot \begin{pmatrix} 2.15 \\ 2.15 \\ 1 \end{pmatrix} \cdot x_3$$

The relative error in x_1 is thus proportional to x_3, but as long as δ is small, the proportionality factor is very small. With an extremely large error of 100%, we get $\tilde{x}_1 \approx 0.2x_3$ which still is fairly small. This should be compared to the results of the same calculations performed on a system where feedback is applied only to the charge derivative as in [5]. By doing so on this system, we would increase the relative error in the estimate of *SoC* with a factor 10.

4. Simulation results

Before coding the SoC-observer for practical use in the micro-processor, its behaviour in different load situations had to be tested through simulation. Comparisons to results obtained with the Ampere-hour method (2.5) are also made.

4.1 Battery model

The NiCd cell has been implemented in the MATLAB/SIMULINK simulation environment. It is thus easy to thoroughly test the properties of the battery cell model and to compare them to real measurements.
In Figure 4.1 below, the output voltage from the model is compared to the terminal voltage of the battery, when discharged with constant currents C/5, C/2 and 1C. The curves show that the model successfully estimates the output voltage in a wide range of currents during static operating conditions. The curves also indicate in which current range the model is valid: Larger currents than C produce less reliable results.

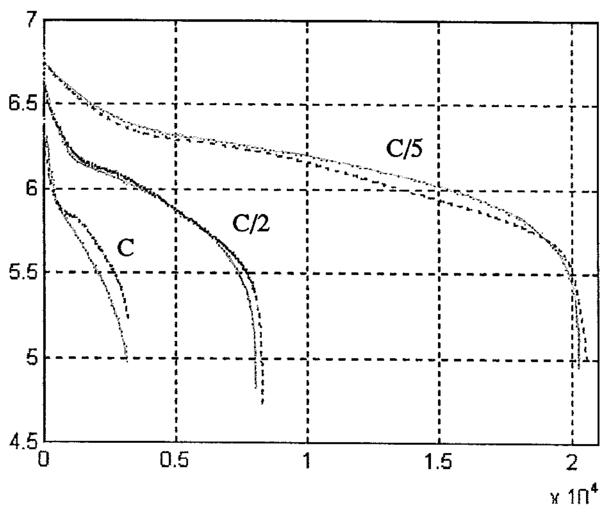


Figure 4.1: Measured (black) and estimated (grey) voltages versus time during discharge with constant currents.

Since the observer should work in a hybrid vehicle, where the batteries are continuously charged and discharged, the model also has to be accurate during charge. Figure 4.2 shows a plot of the measured and estimated voltages when the battery is discharged with C/5 from 100% *SoC* down to 20% *SoC*, up to 80% *SoC* and once again down to 20% *SoC*.

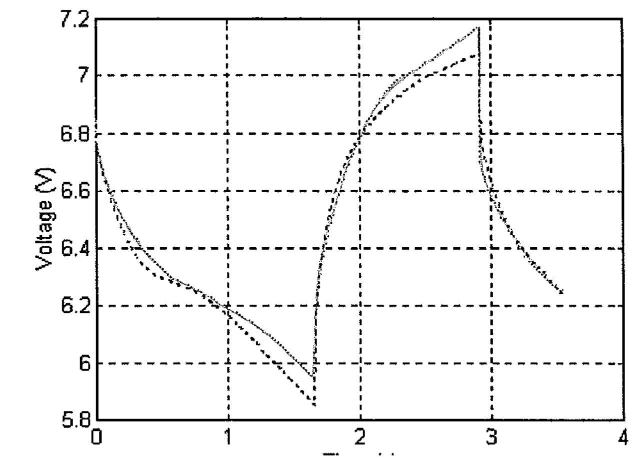


Figure 4.2: Measured (black dots) and estimated (grey) voltages versus time during discharge and charge with C/5.

From Figure 4.2, I conclude that the terminal voltage during charge is modelled quite well, but with one important constraint: Larger constant charge currents than C/5 can not be tested since this will effect the lifetime of the battery negatively. It is thus uncertain whether the model is correct for charge currents outside the interval [0,28] A.

One way to further validate the model is to apply dynamic discharge cycles. If the dynamics are considerably faster than the time constants of the battery, we may draw some conclusions about the internal resistance. The SFUDS (Appendix B) is such a cycle and the diagram in Figure 4.3 present the voltage behaviour during discharge.

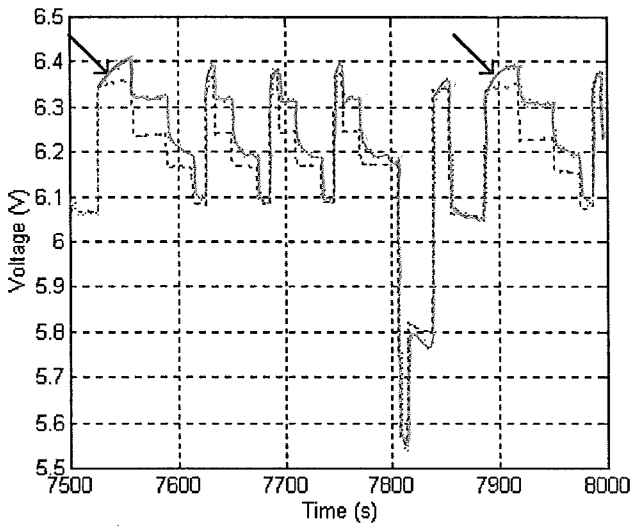


Figure 4.3: Simulated (dotted) and measured (grey) terminal voltage at SoC=0.6 when driven with SFUDS from SoC=1.0.

Figure 4.3 reveals an interesting fact about the model. There is an non modelled time constant (marked with arrow) in the battery, which is believed to belong to the draining of the electrode at sudden heavy discharges. This draining probably increases the internal resistance.
Another discharge cycle, which will be an international standard test for batteries in electric vehicles, is the ECE (Appendix B). This cycle exhibits both fast and slower dynamics, and the result when applied to this model is shown in Figure 4.4.

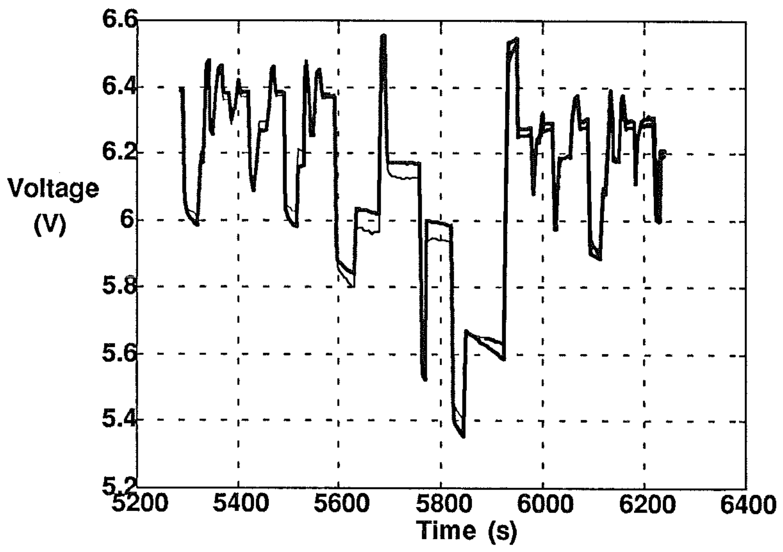


Figure 4.4: Simulated (dotted) and measured (filled) terminal voltage with the ECE motor current demand as load. On the plot above, SoC is between 0.6 and 0.5.

The results from the two dynamic cycles above, indicate that the internal resistance is correctly modelled, since the positive an negative peaks are of the right magnitude. The same tests as above, but performed at other levels of *SoC* further validates this.

4.2 SoC Observer

Like the battery model, the observer is also implemented in SIMULINK. I have thus been able to simulate the observer with both measured and simulated input data before it was coded.
Figure 4.5 shows the estimated *SoC* when simulated with measured data when the battery is discharged with C/2. As comparison, the *SoC* according to the Ampere-hour method (2.5) is shown as a straight line.
The observer is less accurate at the beginning of the discharge. These problems can be related to model errors, and could be avoided with a better model. The actual capacity Q_0 (2.3), as well as the nominal capacity C (2.5), is set to 160 Ah.

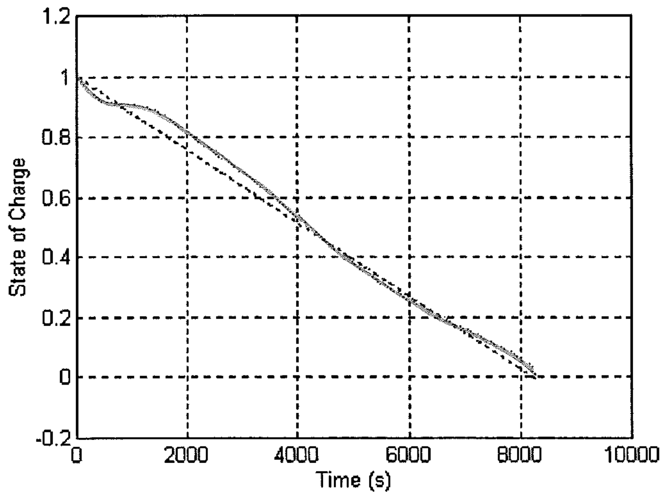


Figure 4.5: Two estimates of SoC with measured values of voltage, when discharged with C/2. Ampere-hour method(dotted) and observer (filled, grey).

One of the largest problems with batteries is however to determine the actual capacity Q_0, since variations of up to 15% from the nominal value are common. In Figure 4.6, I have simulated a 20% loss of capacity, to examine how fast the observer can recover from such an error.

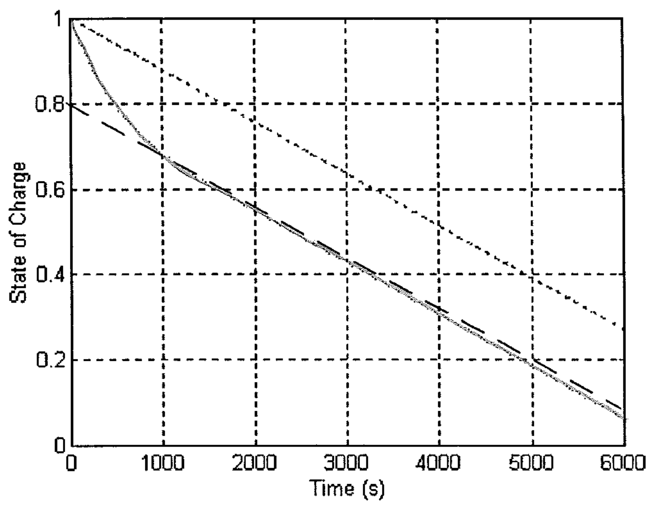


Figure 4.6: Simulation of a 20% capacity loss and discharge with C/2. The Ampere-hour method (dotted) integrates down from the nominal 160 Ah, while the observer finds the true value (dashed) within 15 minutes.

Figure 4.7 shows the result from the same experiment, but when the battery is discharged with the ECE cycle(simulated voltages are used). The graph indicates a stable reconstruction with the same time constant as in Figure 4.6. Figure 4.8 shows the reconstructed states during this discharge.

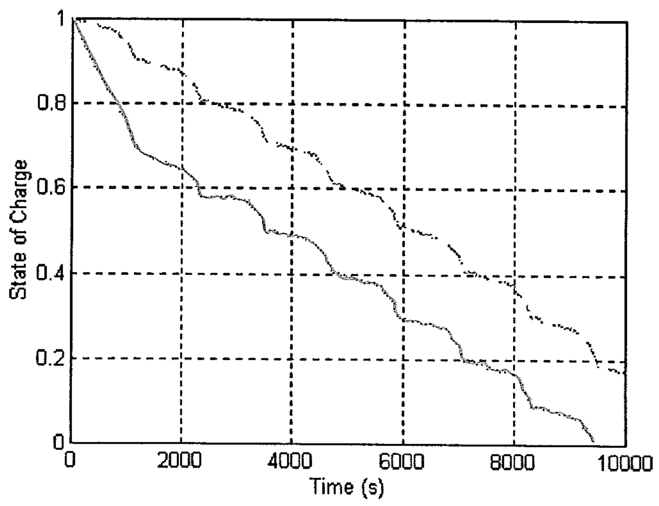


Figure 4.7: Simulation of a 20% capacity loss and discharge with ECE. Estimated SoC (filled) is compared to the Ampere-hour method result (dashed)

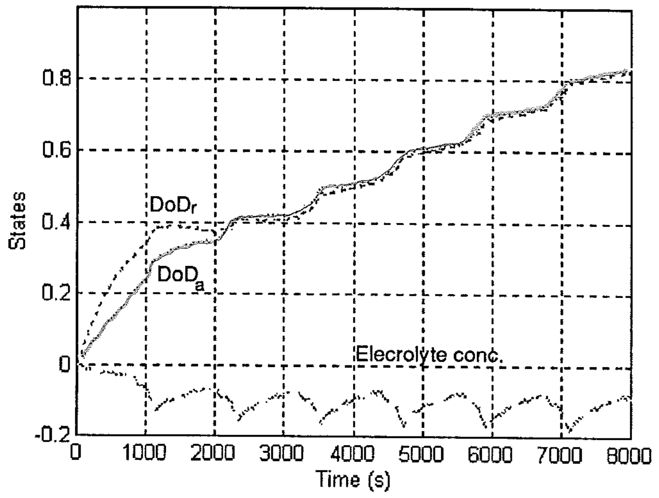


Figure 4.8: The reconstructed states during the ECE discharge with 20% capacity loss. Maximum electrolyte concentration is here defined as zero, which explains the behaviour of the third state (dash, black).

The tests show a good correspondence with the behaviour that was derived from the linear model in Chapter 3.3 in terms of settling time and overshot in the reconstruction error. DoD_a, which is the same as the estimation of (1-*SoC*), settles within 15 minutes, completely without overshot.

Results from real time measurements are presented in Chapter 5.

5. Measurement results

Although it is difficult to actually determine the true value of *SoC* during operation, some sort qualitative testing of the *SoC*-observer has to be performed. I have thus used the observer to predict the residual capacity after different kinds of discharges. The test is a sort of "worst-case" test, since it uses discharge currents which are on the validity limit for the original battery model.

Figure 5.1 below, shows the observers estimation of *SoC* when the battery is discharged with 1C (140 A). The Ampere-hour method is shown as a comparison. Both methods are supplied with the real actual capacity of the battery (150 Ah). After a two hour rest (I=0), the residual capacity is determined with a C/5 (28 A) discharge. The result is found in row 1 of Table 5.1. In this case the A-h method clearly provides a better estimate.

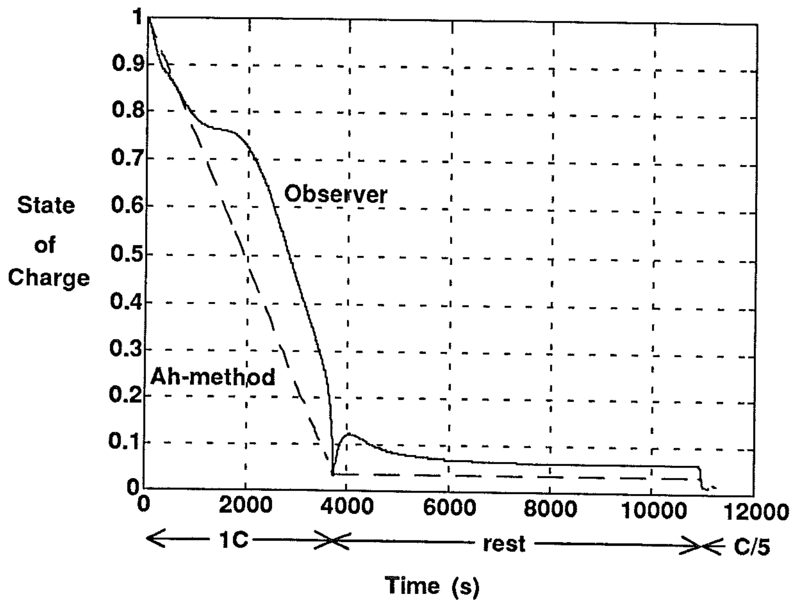


Figure 5.1: Two estimates of SoC during 1C discharge followed by a 2 h rest and determination of the residual capacity with C/5. Observer result shown as filled, Ampere-hour-method result shown as dashed.

Next, the same test is performed, but with the 1C exchanged for an ECE discharge. In this case, both methods provide fairly good estimates (row 4 of Table 5.1).

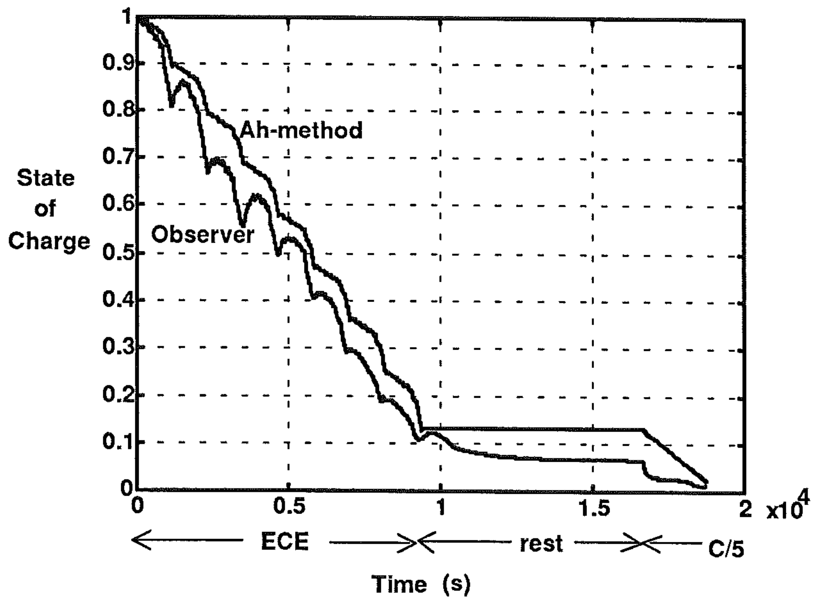


Figure 5.2: Two estimates of SoC during ECE discharge followed by a 2 h rest and determination of the residual capacity with C/5.

In order to determine the accuracy of the observer in the case of capacity loss- or gain, the same tests as above are performed with the observer set to different initial capacities. The test battery has an actual capacity of 150 Ah in all cases, while the observer is supplied with a 15 Ah excursion (up and down) from this value.

The result is presented in Table 5.1: The battery is discharged with the discharge pattern in column 1, until the cut-off voltage is reached. After a 2 hour rest, the estimates of the residual capacity are determined (column 3 and 4) and compared to what we get with a C/5 discharge (column 2).

	Measured res. cap. [Ah]	**Observer [Ah]**	**Amp.hour method [Ah]**	**Observer error [%]**	**Amp.hour error [%]**
1C Q_0=150Ah	3.8	9.0	5.1	3.5	0.9
1C Q_0=165Ah	3.8	10.1	20.3	3.8	10.0
1C Q_0=140Ah	3.8	8.4	4.9	3.3	6.2
ECE Q_0=150Ah	16.3	10.7	19.7	4.2	3.4
ECE Q_0=165Ah	16.3	10.9	34.7	3.2	11.1
ECE Q_0=140Ah	16.3	9.3	9.7	5.0	4.7

Table 5.1: Summary of the residual capacity test performed with the observer in comparison to what the Ampere-hour method would give.

There are two significant conclusions to be drawn from this test:

- The Ampere-hour method gives a very accurate estimate, provided that the actual capacity is known. If not, the accuracy decreases rapidly.
- The observer provides an estimate with a small error, which is below 5% and independent of the actual capacity of the battery.

The *SoC* observer is thus the most reliable alternative in case of a non-constant battery capacity, as often the case in traction batteries.

It is also of interest to determine the behaviour of the *SoC* observer when the battery is charged. The result from a residual capacity test when the battery has been charged and discharged with SFUDS is presented in Figure 5.8.

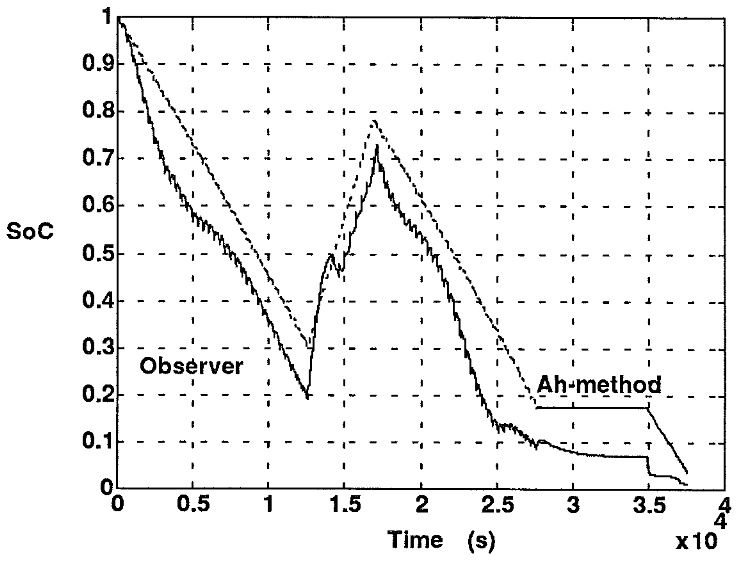


Figure 5.3: Two estimates of SoC when the battery alternately discharged and charged with SFUDS. Observer shown as filled and Ampere-hour method shown as dotted for comparison.

The observer estimates the residual capacity to 10.7 Ah, while the A-h method stops at 26.6 Ah. The actual residual capacity was in this case 20.7 Ah, resulting in an error of 6.7% for the observer and 3.6% for the A-h method.
It should be noted that the observer always seems to under estimate the residual capacity, while the A-h method generally provides an over estimate.

6. Program development

The translation of the SIMULINK structures into C code resulted in several different program modules. Each module handle a specific part of the equipment, and they are all meant to be reused in other applications.

6.1 General outline.

Since the EPC166 working platform will be used in several future projects, one of the goals with this thesis work was to create a general interface towards the EPC board. The interface should consist of a number of useful C-code functions available as a header-file. The functions should handle everything that is specific to the card, i.e. A/D-conversion, timing, reading and writing on ports etc. This header file could then be included in various types of measuring- or control applications.
In the near future, Math Works Inc. will make an automatic C-code generator for SIMULINK structures available. This will open the possibility of designing e.g. controllers directly in MATLAB/SIMULINK and then quickly transfer them to a micro processor for real time operation. This further increases the need for a generalisation of the C-code parts that are specific to the micro controller.

The result is a program structure, consisting of five separate modules. Figure 6.1 shows the modules and how they communicate with each other.

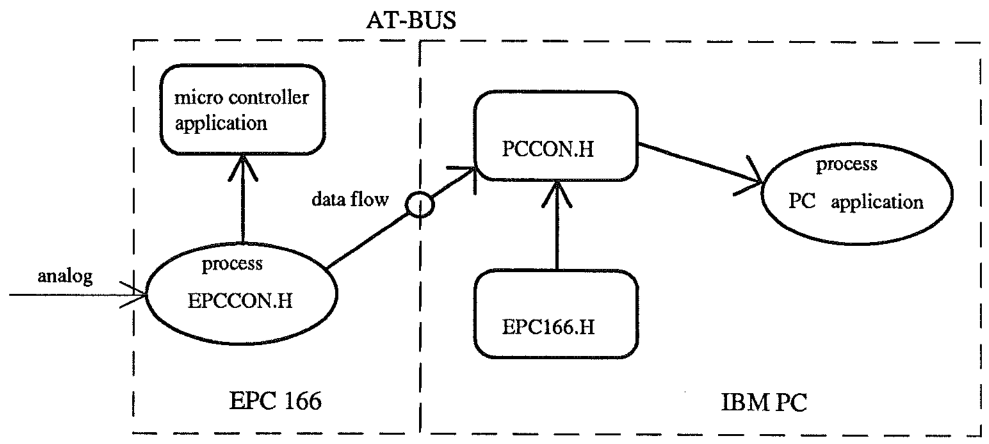


Figure 6.1: A modular program structure where three header file modules handle the data flow between EPC and PC and provide control functions for the EPC

The header file EPCCON.H is visible to an EPC application program, and exports functions and data structures. The data structures specify the format on which data is available for an application. The functions handle the operation of the EPC board.
The header file EPC166.H exports functions which handle access to that specific

part of the extended memory into which EPC RAM is mapped.
The header file PCCON.H is visible to a PC application program, and exports functions which handle upstart of the EPC board and data transfer from the board up to the PC.

6.2 EPC 166 Micro controller.

6.2.1 EPC interface

The interface consists of one separately compiled C-code source file, accompanied by a header file (EPCCON.H) that can be included in an EPC 166 application program. Source code listings are found in appendix C. I have used the C166 compiler [13], which should be used for all EPC 166 applications.

The interface provides three data structures, which define the formats for data.

input_data_struct	is the format for data received from input ports and A/D converter.
output_data_struct	is the format for output data which is sent to an output port. (i.e. control signal, display drive, etc.)
pc_data_struct	is the format for data delivery to PC extended memory.

The interface provides the application with pointers to these structures. These pointers are the only means by which an application can receive and send data. The interface also provides two functions that handle the processor.

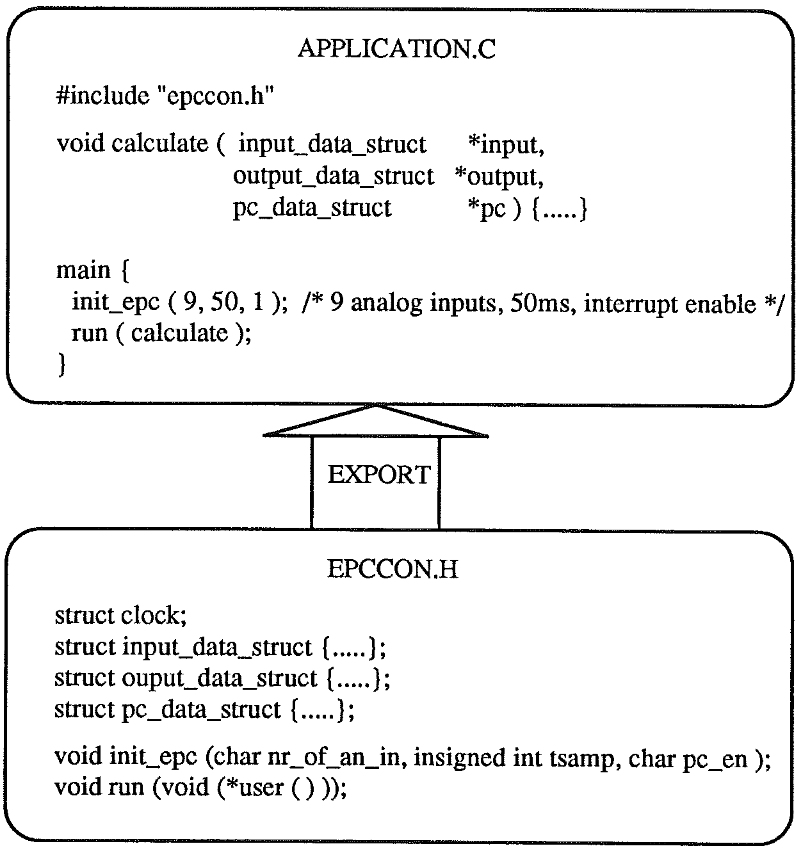
init_epc (char nr_of_an_in, unsigned int tsamp, char pc_en);

> initialises the processor to convert **nr_of_an_in** channels starting with channel 0, sample with an interval of **tsamp** milliseconds and to enable/disable (**pc_en=1/pc_en=0**) automatic transfer of the data in the **pc_data_struct** to PC.

run (void (*user ()));

> This function is actually a process, running in an infinite loop. It samples the inputs and calculates the outputs at every sample point, using the function which is pointed out by ***user.**

Figure 4.2 shows a sketch of how to use the interface and Figure 4.3 shows the flow chart for **run()**.



```
#include "epccon.h"

void calculate ( input_data_struct    *input,
                 output_data_struct   *output,
                 pc_data_struct       *pc ) {.....}

main {
  init_epc ( 9, 50, 1 );  /* 9 analog inputs, 50ms, interrupt enable */
  run ( calculate );
}
```

```
struct clock;
struct input_data_struct {.....};
struct ouput_data_struct {.....};
struct pc_data_struct {.....};

void init_epc (char nr_of_an_in, insigned int tsamp, char pc_en );
void run (void (*user ( ) ));
```

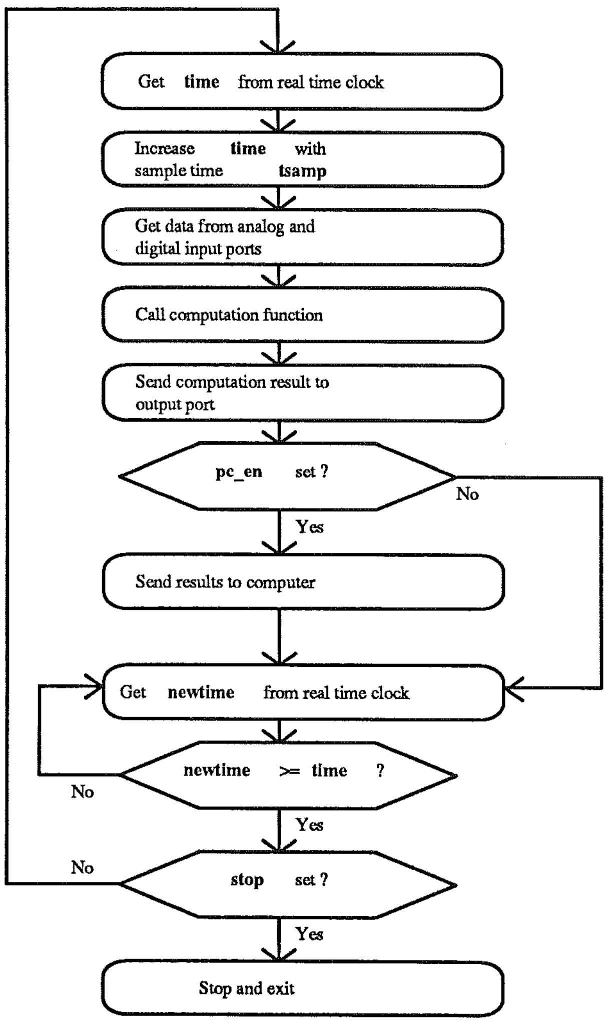
Figure 6.2: Usage of the EPC interface EPCCON.H

Note the rules on how do define the computation function (**calculate** in Figure 6.2). The name can be chosen arbitrarily, but the parameter list must only contain three pointers, which are pointers to one of the data structures **input_data_struct**, **output_data_struct** and **pc_data_struct**. respectively. The names of the pointers can of course be chosen arbitrarily.

When **run** () calls the computation function at which ***user** points, it makes the call with three pointers (as mentioned above) as input arguments. The reason for this construction is a matter of execution time and memory space saving. When the pointers are pointing to the absolute RAM addresses where data is stored, only the pointer values (and not entire structures) are copied between functions who want to alter the contents of the structures.

The real time clock is implemented as a timer which generates an interrupt upon overflow. The interrupt routine updates the clock every millisecond.

I have outlined the flow chart for **run**() in such a way that the sample period is constant, regardless of the execution time for the computation function. (A general description of this kind of controller routine may be found in [9].) If, however, the execution time should exceed the sample time, the program will produce an error message and then terminate.



Get **time** from real time clock
Increase **time** with sample time **tsamp**
Get data from analog and digital input ports
Call computation function
Send computation result to output port
pc_en set ?
No
Yes
Send results to computer
Get **newtime** from real time clock
newtime >= **time** ?
No
Yes
No
stop set ?
Yes
Stop and exit

Figure 6.3: Flow chart for the function run() provided by EPCCON.H.

6.2.2 SoC observer

The SoC observer program (SAFT140.H86) runs on the 80C166 processor, and is an example of an application that includes the EPC interface. It is based on the simulation model I have developed for the SAFT STM 140 battery (chapter 3) Once the SIMULINK structure was established, translation into C-code turned out to be quite easy.

I have used rectangular approximation to discretise the state equations (3.1). The expression for the electrolyte concentration becomes:

$$\frac{dV}{dt} = C_V\left(-V + \frac{I(t)}{J}\right)$$
$$\frac{V(k+1)-V(k)}{h} = -C_V V(k) + C_V \frac{I(k)}{J}$$
$$V(k+1)-V(k) = -hC_V V(k) + hC_V \frac{I(k)}{J}$$
$$V(k+1) = \left(1-hC_V\right)V(k) + hC_V \frac{I(k)}{J}$$

The expression for blocked charge is discretised as:

$$Q_s(t) = \int_0^T \tilde{I}(t)dt$$
$$Q_s(k+1) = Q_s(k) + h\tilde{I}(k)$$

And finally the extracted charge as:

$$Q_s(t) = \int_0^T \tilde{I}(t)dt$$
$$Q_s(k+1) = Q_s(k) + h\tilde{I}(k)$$

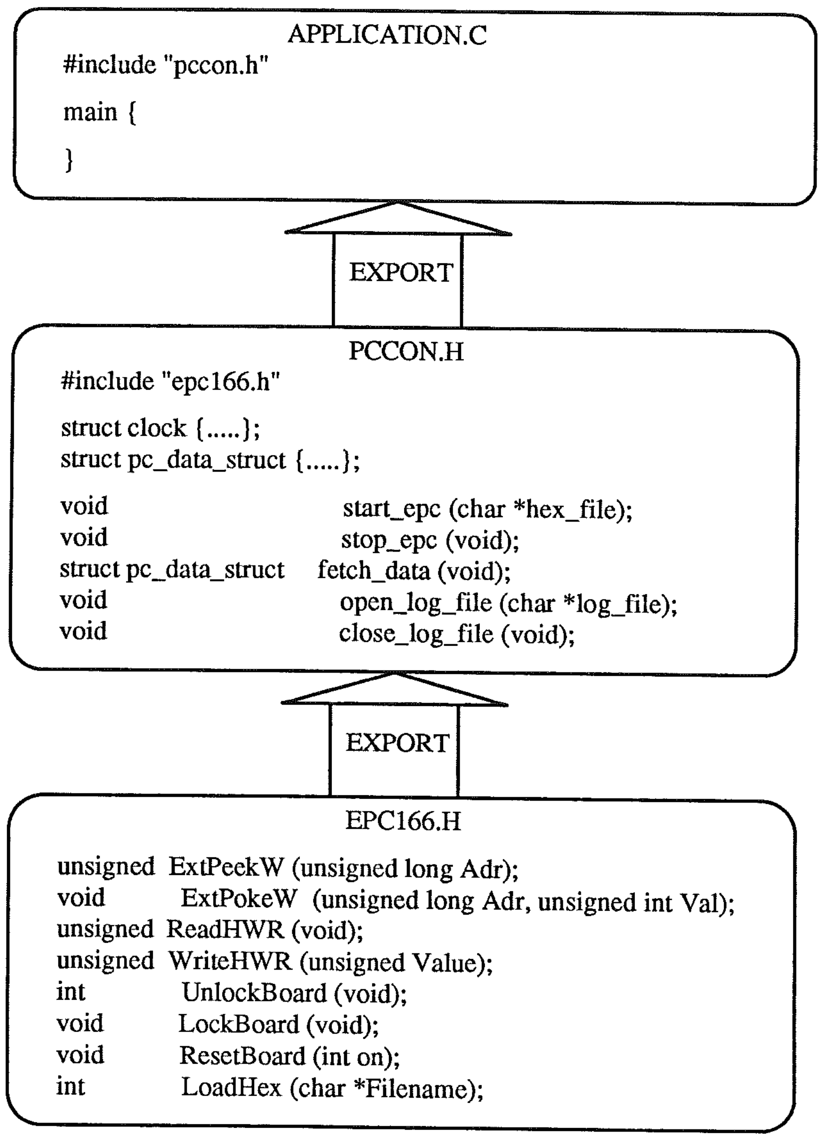
where $\tilde{I}(k)$ is feedback corrected current and h sample time.

The function **calculate** uses pointers as described in 5.2.1 to access the input structure. From the values of current and voltage, it calculates the State of Charge and updates the output structure with a new value of *SoC*. Via pointers **calculate** also updates the PC structure with current values of sample time, current, voltage, temperature and *SoC*.
A listing of the source code is found in Appendix C.

6.3 IBM PC

6.3.1 PC interface

The EPC 166 board has several features that are software programmable through the PC's extended memory. Most of these features are controlled through the *hardware register* (HWR). This register is a 16 bit word which is mapped into extended memory at the board's base address + 10000hex. To simplify the use of this register, the module EPC166.H provides some useful functions. The module also provides functions for direct memory access on the EPC board, as well as function for the download of hex-files (*.h86-files created with the C166P compiler [13]) onto the board.



```
#include "pccon.h"
main {
}
```

```
#include "epc166.h"
struct clock {.....};
struct pc_data_struct {.....};
void                    start_epc (char *hex_file);
void                    stop_epc (void);
struct pc_data_struct   fetch_data (void);
void                    open_log_file (char *log_file);
void                    close_log_file (void);
```

```
unsigned  ExtPeekW (unsigned long Adr);
void      ExtPokeW  (unsigned long Adr, unsigned int Val);
unsigned  ReadHWR (void);
unsigned  WriteHWR (unsigned Value);
int       UnlockBoard (void);
void      LockBoard (void);
void      ResetBoard (int on);
int       LoadHex (char *Filename);
```

Figure 6.4 Contents and export order of the modules that make up the interface between the PC and the EPC board.

In order to further simplify the use of the board, I have constructed an additional module: PCCON, where the settings for the HWR are pre-defined. The module exports two data structures and five functions. The structure **pc_data_struct** is the format for data which the function **fetch_data** returns as it reads from extended memory. The function **start_epc** uses a file name as input argument. It loads the requested file on the EPC board and starts it. The function also defines HWR settings such as memory segmentation, bus mode and IRQ channel. The function **open_log_file** also has a file name as input argument. It opens a file for automatic logging of the data which is transferred through extended memory with **fetch_data.** The other two functions in PCCON **close_log_file** and **stop_epc** do exactly what their names suggest. If the application program should call **stop_epc**

before it calls **close_log_file, stop_epc** closes it automatically. Note that only one log file is allowed to be open.
All functions are equipped with detection of misuse, to prevent the application from crashing. When one of the errors I assume to be most common occur, the program terminates normally and leaves a proper error message. The function **fetch_data** can also cause the program to terminate with an error message if the EPCCON interface is somehow misused. **Fetch_data** checks a designated error address in extended memory which EPCCON uses to inform the PC of unexpected errors.

6.3.2 Control program

The control program for the battery management system is actually very simple since it uses the powerful tools of the PCCON module. It runs in an infinite loop which mainly fetches data from the EPC board and displays it on the screen. There are three major ideas behind this tiny program:

- To control the SoC observer (of course).
- To serve as an example of how to use the PC/EPC interface.
- To be used as a control program for future SoC observers for other batteries.

The flow chart of the control program is presented in Figure 6.5.

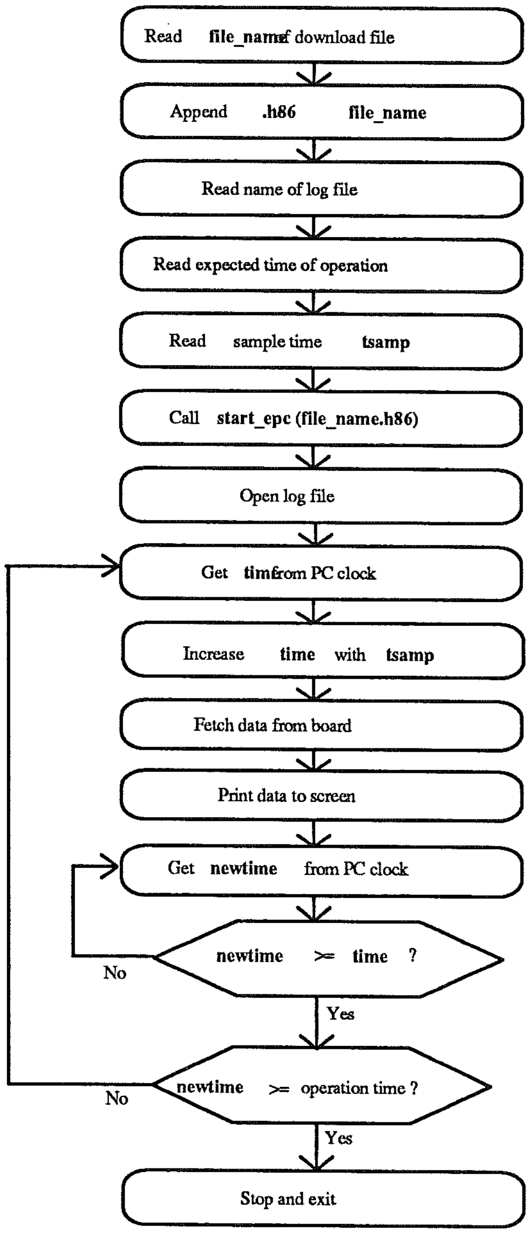


Figure 6.5: Flow chart for the control program BATMAN.EXE.

7. Conclusions

When designing a control system for an electric- or hybrid vehicle, the State of Charge (*SoC*) of the batteries is an important control variable. In most cases, integration of the load current (2.5) provides a sufficient estimate. It is however impossible to avoid a certain offset of the estimate due to measurement errors, and a certain drift due to losses within the battery. In a hybrid vehicle these errors become unacceptably large, because of long periods without calibration (fully charged/discharged).

To overcome these errors, I have developed a state observer, based on a non-linear model of the battery. Using traditional observer technique, it measures the load current, estimates the terminal voltage, and uses the voltage error to adjust the state derivatives (Figure 3.4).

The quality of such an estimate is of course dependant on the model accuracy. Errors in the modelled time constants do not effect the quality significantly, but unmodelled modes of the battery have a severe impact on the estimate. Measurement errors also influences the estimate, where the voltage measurement is more critical than the current measurement.

Through simulation, I have found a feedback gain vector which places the observer poles on an approximative half circle with the radius $\omega_0 \approx 1.4 \cdot 10^{-3} s^{-1}$. This placement makes the reconstruction error approach zero with a time constant of approximately 12 minutes, which is also confirmed through simulation. An attempt to make the observer faster, results in a less accurate estimate due to faster time constants in the battery, which are not modelled correctly.

I have also performed residual capacity tests, which show that the observer is capable of estimating the residual capacity with less than 5% error, practically regardless of initial state or discharge patterns.

The translation of the observer into C code has resulted in two general purpose interfaces towards the micro controller board EPC 166. These can be used to produce SoC-observers for other batteries, or for easy development of a complete battery management system.

8. Recommendations for future work

This project leaves several opportunities open for future development of a complete battery management system.

The first and most obvious continuation, is to use Volvo's other battery models and to develop *SoC*-observers for these batteries. With the program structure of my project, it is an easy task to build a library of *.H86 files, which all computes the State of Charge for each specific battery.

My second proposal is to integrate the C-code interfaces with the SIMULINK/C-code Generator[2], and investigate how the generator treats these source code files. I also suspect that a lot of work has to be put into the code generator environment in order to make it run smooth in Volvo's applications.

More work can be put into the actual modelling of the batteries. Not only simulation models should be developed, but also better linear models. Then perhaps adaptive control theory could be used to further enhance the quality the *SoC* estimate. Good models for predicting the remaining driving range, given the *SoC*, should also be developed.

Last, but not least, some work on the hardware should be carried out in order to increase the portability of the system.
First, an LCD-display with appropriate drive functions should be added. Then the possibility of detaching the micro controller from the evaluation board should be investigated. Placing the controller inside the signal conditioning unit opens the possibility of performing measurements on the vehicle during propulsion.

[2] Available from Jan -94 as "Beta-release"

9. References

[1] AB Volvo TU 06240, Modelling of Traction Batteries, Status report 1 and 2, 1993

[2] Larsson, P. & Andersson, U. Batteristatusmätare för El- och Hybridfordon, Examensarbete, Chalmers Tekniska Högskola, 1992.

[3] Lürkens, P. & Steffens,W. Ladezustandschätzung von Bleibatterien mit hilfe des Kalman-Filters, Rheinisch-Westfäliscen Technischen Hochschule, Aachen, 1986

[4] Giglioli, R. Charge and discharge fourth order dynamic model of the lead-acid battery, EVS-10, The 10th international Electric Vehicle Symposium in Hong Kong 1990

[5] Giglioli, R. et al. Experience on a battery state of charge observer, EVS-11, The 11th international Electric Vehicle Symposium in Florence (Italy) 1992

[6] Schöpe, F. Microcontroller based State of Charge indicator for NiCd batteries in electric vehicles, EVS-11, 1992

[7] Hank, B. BATTMAN - A battery management system, EVS-11, 1992

[8] Linden, D. Handbook of batteries & fuel cells, McGraw Hill, 1984

[9] Nielsen, L. Computer implementation of control systems, Departement for Automatic Control, Lund Institute of Technology, 1992

[10] Spanne, S. Lineära system, Matematiska institutionen vid Lund Tekniska Högskola, 1988

[11] Åström, K.-J., Wittenmark, B. Computer controlled systems, Prentice Hall, 1990

[12] Bilting, U., Skansholm, J. Vägen till C, Studentlitteratur, 1987

[13] KEIL Elektronik, C166 Professional developers kit, KEIL Elektronik, München, 1992

[14] Ertech, EPC 166 Evaluation board, Ertech GmbH, Erlangen 1992

[15] Siemens, SAB 80C166/83C166 User's manual, Siemens AG, 1990

[16] Widell J. Study of control stategies for a hybrid vehicle., master thesis at the Department for Automatic Control, LTH 1994. To be printed.

ECE-15 Power Profile for the JRC Reference Vehicle

The test cycle consists of two parts, one urban part which is repeated four times and one suburban part.

Total cycle length is 1180 seconds.
Urban part: 195 sec.
Suburban part: 400 sec.

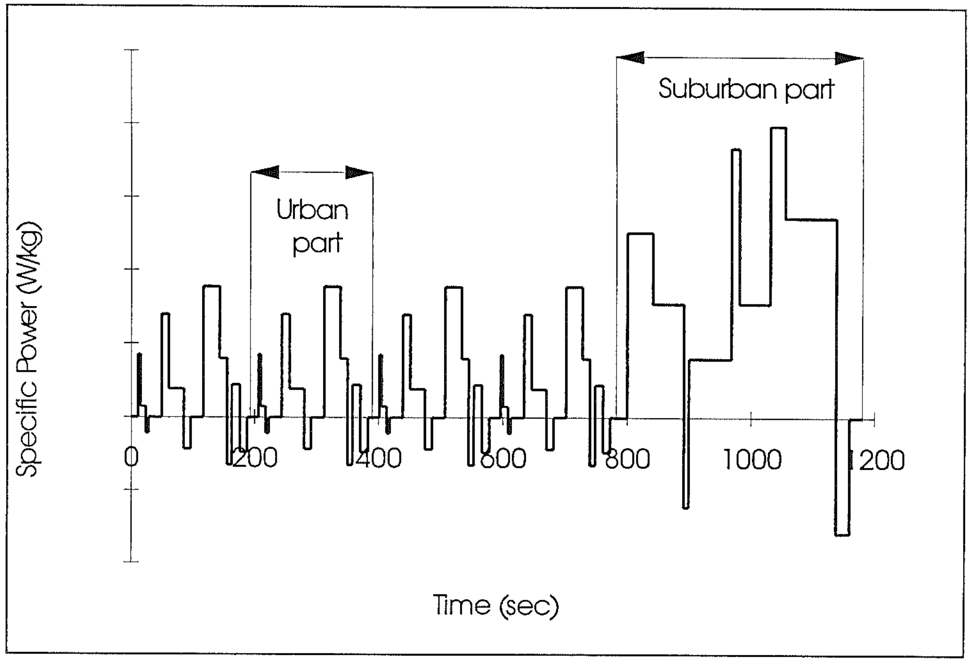
Termination Criteria

Terminate the discharge of the battery when either one of the following conditions are met.

1. The battery can not deliver minimum power demand. Minimum power demand is set by the power required in the time periode 275-358 seconds in the suburban part of the cycle.

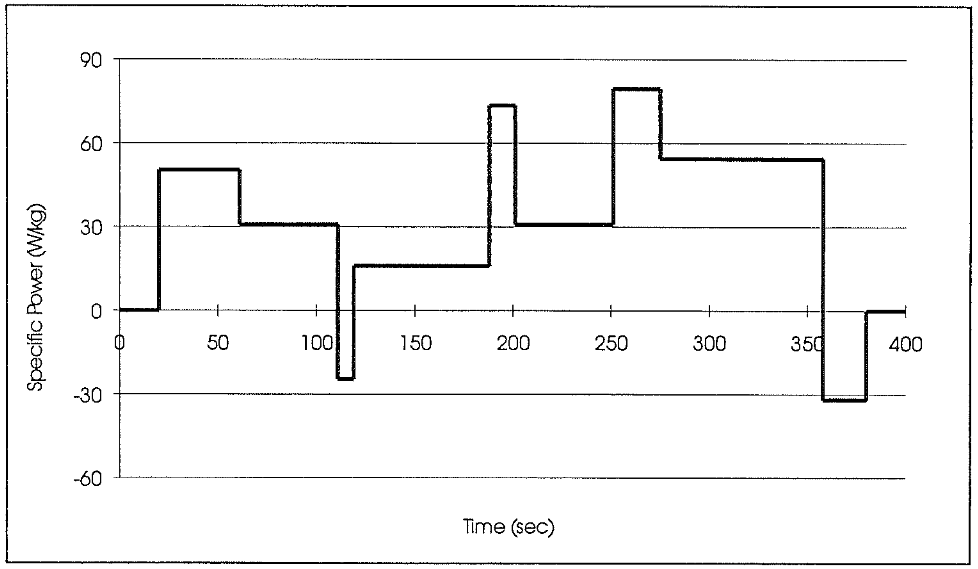
2. The minimum battery voltage is reached.

3. Any other termination criteria are met, i.e. maximum temperature.



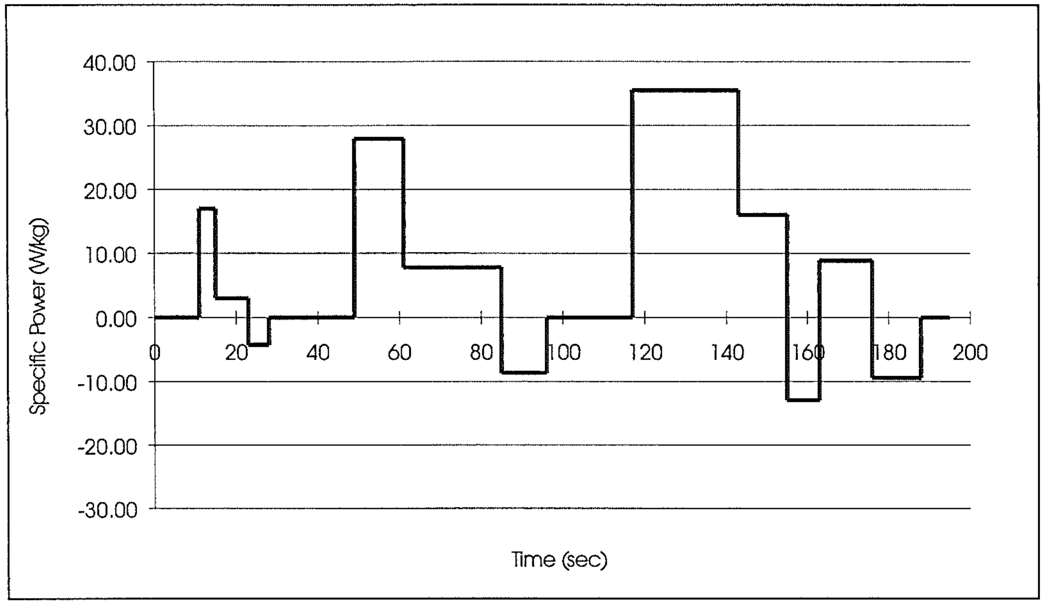
Suburban Part of ECE-15 for the JRC Reference Vehicle

Time (sec)	Sp. Power (W/kg)
0-20	0
20-61	50.3
61-11	30.9
111-119	-24.5
119-188	16.0
188-201	73.4
201-251	30.9
251-275	79.5
275-358	54.3
358-380	-31.8
380-400	0



Urban Part of ECE-15 for the JRC Reference Vehicle

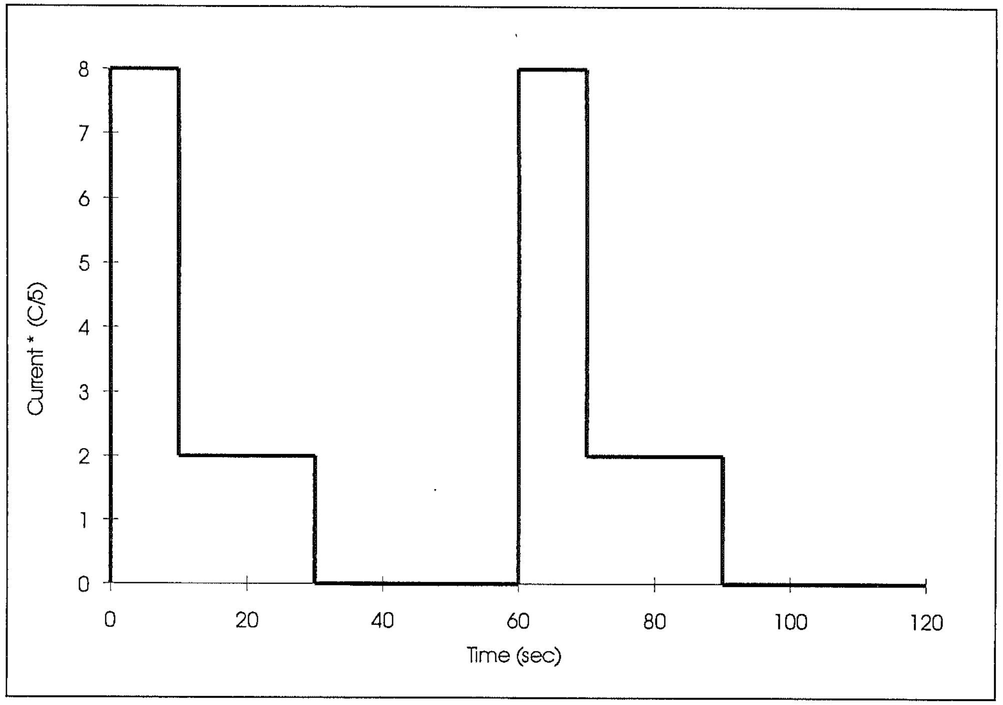
Time (sec)	Sp. Power (W/kg)
0-11	0
11-15	17.0
15-23	3.0
23-28	-4.3
28-49	0
49-61	27.9
61-85	7.8
85-96	-8.6
96-117	0
117-143	35.5
143-155	16.0
155-163	-13.0
163-176	8.9
176-188	-9.4
188-195	0

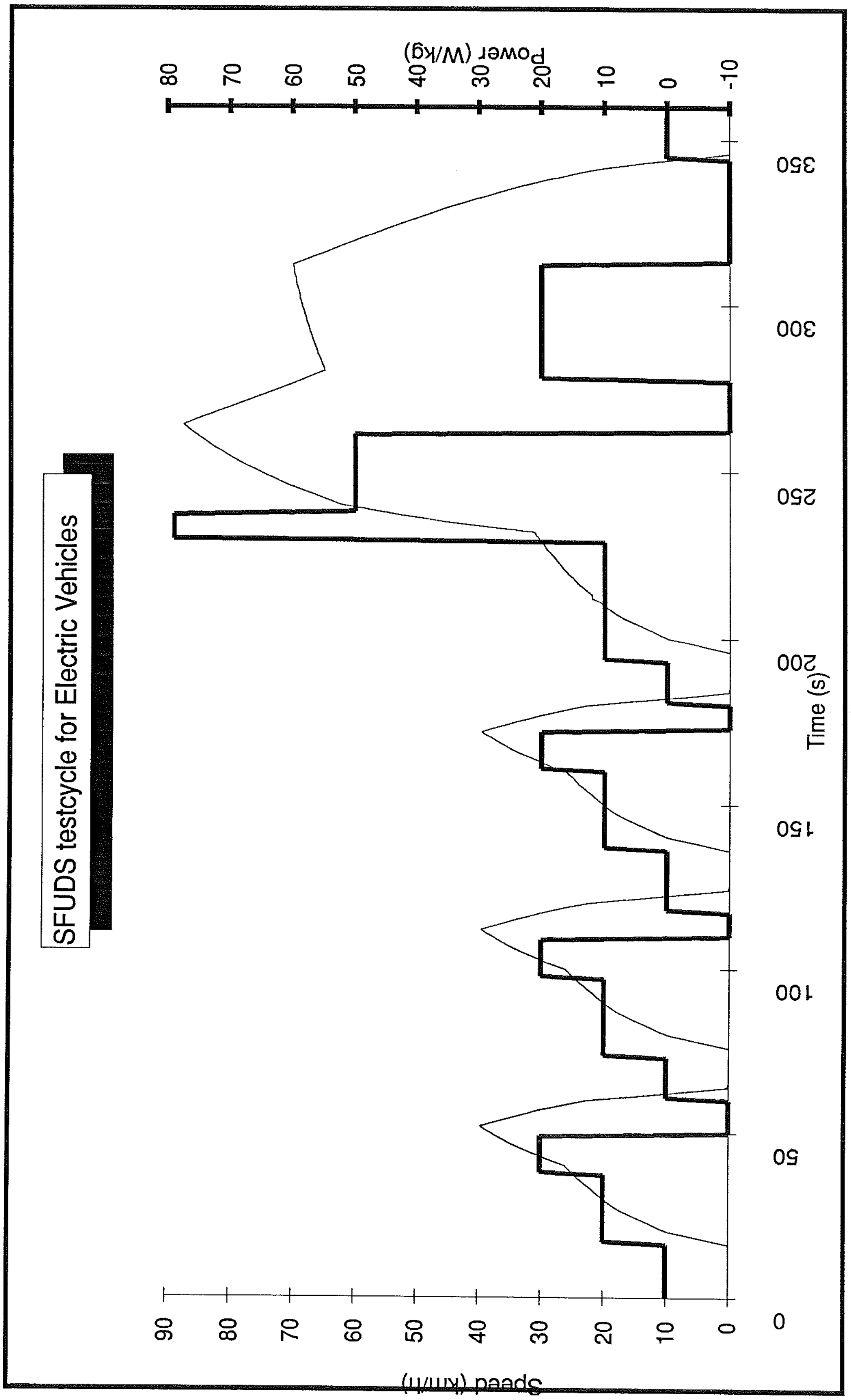


Current Profile for two cycles of the TC69/WG3

Time (sec)	Current
0-10	8 * C/5
10-30	2 * C/5
30-60	0

The cycle will be repeated until Cut-off-Voltage specified by the battery manufacturer has been reached.





SFUDS testcycle for Electric Vehicles
Speed (km/h)
0
10
20
30
40
50
60
70
80
90
Power (W/kg)
-10
0
10
20
30
40
50
60
70
80
Time (s)
0
50
100
150
200
250
300
350

Appendix C: Source Code listings

```
/********************************************************** */
/*              EPCCON.H                                              */
/* Add the following line to your application:                      */
/*                                                                     */
/*          #include "epccon.h"                                       */
/*                                                                     */
/* when using the EPC-166 card for measurement- or control applications. */
/* Follow the instructions below on how to stucture your application.    */
/* Look at SOCOBS.C as an example of how to use EPCCON.                */
/***********************************************************/

struct clock {  /* structure of the clock record    */
  unsigned int   day;   /* day                  */
  unsigned int   hour;  /* hour                 */
  unsigned int   min;   /* minute               */
  unsigned int   sec;   /* second               */
  unsigned int   msec;  /* milliseconds         */
};

struct input_data_struct {
  struct clock   mtime;
  char           nr_of_values;
  float          value[10];
  char           signal[16];
};

struct output_data_struct {
  unsigned int   port;  /* Word transfer to external unit via P1 */
};

struct pc_data_struct {
  struct clock     mtime;
  unsigned long    sample_number;
  unsigned int     nr_of_measurements, nr_of_results;
  int              m_result[10]; /* Max 10 measurement results */
  int              c_result[8];  /* Max 8 calculated results */
};


extern void init_epc (char nr_of_an_in, unsigned int tsamp, char pc_en);

  /* Initializes the epc card to perform measurements on
    "nr_of_an_in" (max 10) different analog input channels.
    "tsamp" is the time (min 10 milisecs) between two consecutive
    samples.
    When "pc_en" is set, epccon will make an interrupt request to
    the PC on IRQ3. The values in the structure pointed out by
    "pc_ptr", can then be fetched from EPC memory by a PC interrupt
    routine.
    A call to this function with appropriate parameters
    MUST be done from your application program before the "run"
    command is executed.
  */
```

```
extern void run (void (*user) ());

        /* Tells the EPC drive program in which function the
            actual computing takes place.

          Example:      run (calculate);

          where the parameter "calculate" is a user defined function
          which computes the control signal at a given point of time.
          The "bit" declared return parameter is used to signal that
          an unexpected error has occured during computation. When this
          bit is set, the EPC will stop further execution.

          The user defined function must have the structure below:


          Example:   void calculate(input_data_struct  *input_ptr,
                                    output_data_struct *output_ptr,
                                    pc_data_struct   *pc_ptr )

          ALL access to input-data obtained from external devices
          (A/D-converter etc) is done via the input pointer. The results
          are sent back to the EPC via the the output structure and the
          output pointer.
          The PC structure is used to send values to a PC.
        */
```

```
/*************************************************************/
/*                PCCON.H                                     */
/* This is a user interface between the EPC166 evaluation card and an IBM PC.   */
/* It contains functions which loads and starts executable files,             */
/* Fetches data from the card, fetches and translates error messages,         */
/* and stops the card if requested.                                           */
/*                                                                            */
/* The interface is designed for use together with applications              */
/* devellopped on basis of the EPCCON.H standard, and should be used         */
/* to control and to monitor data from such an application.                  */
/*                                                                            */
/* In your application program, add   #include "pccon.h"                     */
/*                                                                            */
/*************************************************************/


struct clock {  /* structure of the clock record        */
  unsigned int   day;    /* day                         */
  unsigned int   hour;   /* hour                        */
  unsigned int   min;    /* minute                      */
  unsigned int   sec;    /* second                      */
  unsigned int   msec;   /* milliseconds                */
};


struct pc_data_struct {    /* Results from EPC166 will be obtained    */
                           /* on this format                          */
  struct clock   mtime;
  unsigned long            sample_number;
  unsigned int             nr_of_measurements;
  unsigned int             nr_of_results;
  int                      m_result[10]; /* Max 10 measurement results */
  int                      c_result[8];  /* Max 8 calculated results   */
};

void start_epc (char *hex_file);
/* Loads and starts the executable file "hex_file"        */
/* (created with H166) into the EPC, and starts it.       */

void stop_epc (void);
/* Stops the execution of the program currently running on the EPC.  */

struct pc_data_struct fetch_data (void);
/* Fetches data from the part of extended memory that is mapped      */
/* into the EPC memory, and returns it in a "pc_data_struct".        */
/* Prints error messages from the EPC.                               */
/* Logs data from the EPC to file.                                   */

void open_log_file (char *log_file);
/* Opens a file for logging of data from the EPC card. Logging is    */
/* done automatically as fetch_data is called.                       */

void close_log_file (void);
/* Closes the log file, if it is open.                               */
```

```
/************************************************************/
/*                    EPC166.h                                 */
/* Contains useful functions for the PC-programmer, who wants to */
/* communicate with the EPC-166 evaluation board               */
/*                                                             */
/* Insert this line in your EPC application :  #include "epc166.h"  */
/*                                                             */
/************************************************************/

#define EPC_BUS_MODE    0x0002      /* 16 bit multiplexed bus mode */
#define EPC_RESET       0x0010      /* 0 --> reset */
#define SECURITY_CLK    0x0020      /* clock for security-function */
#define SECURITY_DIS    0x0040      /* 1 --> disable sec.-function */
#define TRAP_NMI        0x0080      /* non maskable interrupt */
#define PC_NOINT        0x0E00      /* no interrupt */
#define SGTDIS          0x1000      /* 0 --> enable segmentation */
#define BOARD_ENABLED   0x8000
#define HWR_INT_OR      0x0200      /* or-mask to enable IRQ 3 in HWR   */
#define HWR_INT_AND     0xF1FF      /* and-mask to disable IRQ 3 in HWR  */
#define ESC             27
#define BACKSPACE       8

/***  shared function prototypes  ***/

unsigned int far _cdecl ExtPeekW   (unsigned long Adr);
/* Reads one word from extended memory          */
/* Example: value = ExtPeekW (address);         */

void       far _cdecl ExtPokeW   (unsigned long Adr, unsigned int Val);
/* Writes one word into extended memory.        */
/* Example: ExtPokeW (address, value);          */

unsigned int far _cdecl ReadHWR   (void);
/* Reads the EPC166 hardware register at        */
/* BaseAddress + 10000h                         */
/* Example: hwr = ReadHWR;                      */

unsigned int far _cdecl WriteHWR  (unsigned int Value);
/* Writes into the EPC166 hardware register at  */
/* BaseAddress + 10000h                         */
/* Example: new_hwr = WriteHWR (value);         */

int far _cdecl UnlockBoard(void);
/* Unlocks the EPC166 security function.        */

void far _cdecl LockBoard  (void);
/* Activates the EPC166 security function.      */

void far _cdecl ResetBoard (int on);
/* Clears/sets the RESET-bit in the HWR         */

int far _cdecl LoadHEX   (char *Filename);
/* Loads a hex-file (.h66) into the EPC166 board.   */
```

ECE-15 Power Profile for the JRC Reference Vehicle

The test cycle consists of two parts, one urban part which is repeated four times and one suburban part.

Total cycle length is 1180 seconds.
Urban part: 195 sec.
Suburban part: 400 sec.

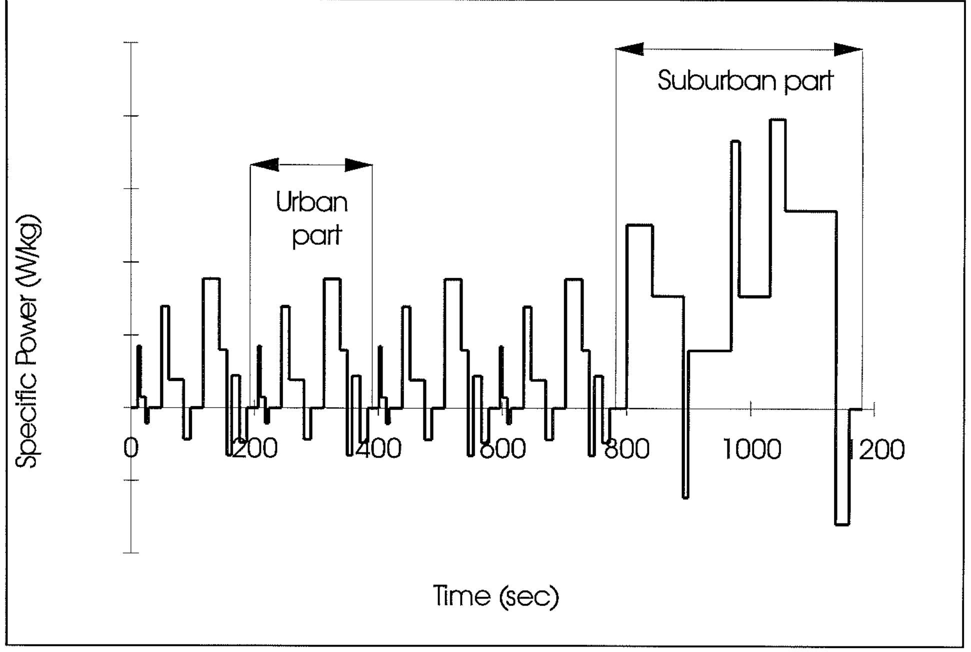
Termination Criteria

Terminate the discharge of the battery when either one of the following conditions are met.

1. The battery can not deliver minimum power demand. Minimum power demand is set by the power required in the time periode 275-358 seconds in the suburban part of the cycle.

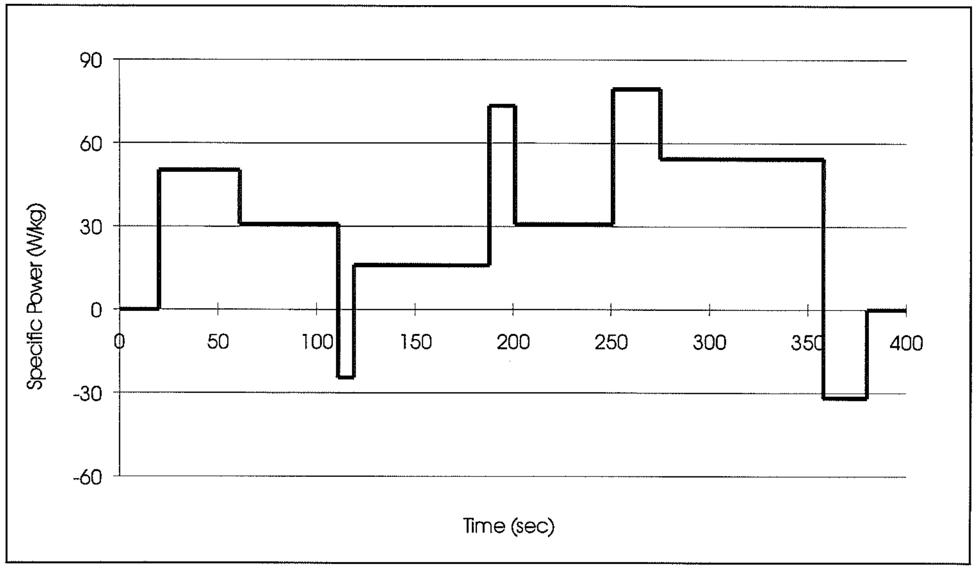
2. The minimum battery voltage is reached.

3. Any other termination criteria are met, i.e. maximum temperature.



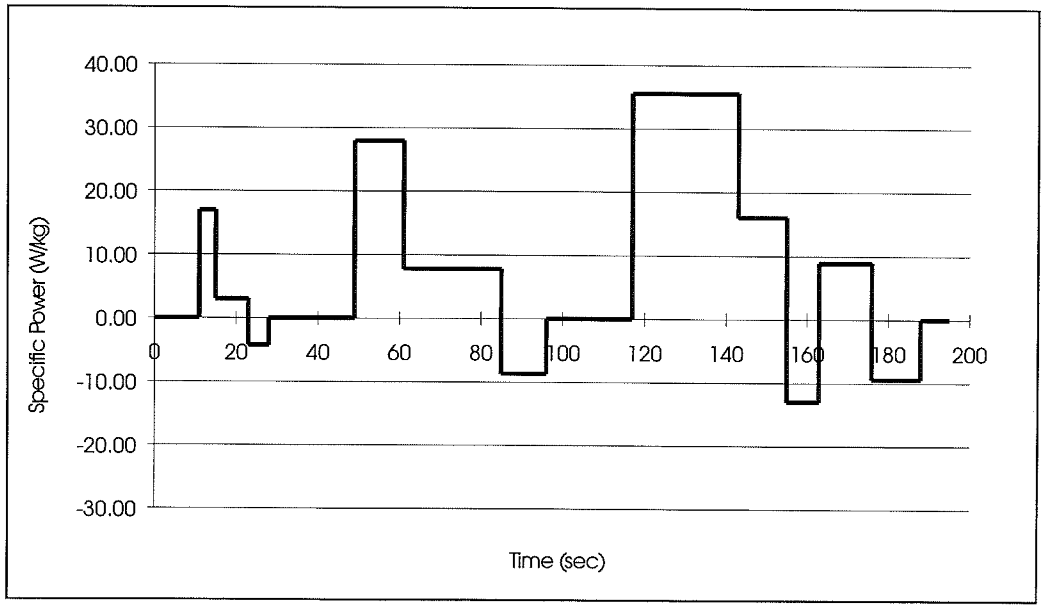
Suburban Part of ECE-15 for the JRC Reference Vehicle

Time (sec)	Sp. Power (W/kg)
0-20	0
20-61	50.3
61-11	30.9
111-119	-24.5
119-188	16.0
188-201	73.4
201-251	30.9
251-275	79.5
275-358	54.3
358-380	-31.8
380-400	0



Urban Part of ECE-15 for the JRC Reference Vehicle

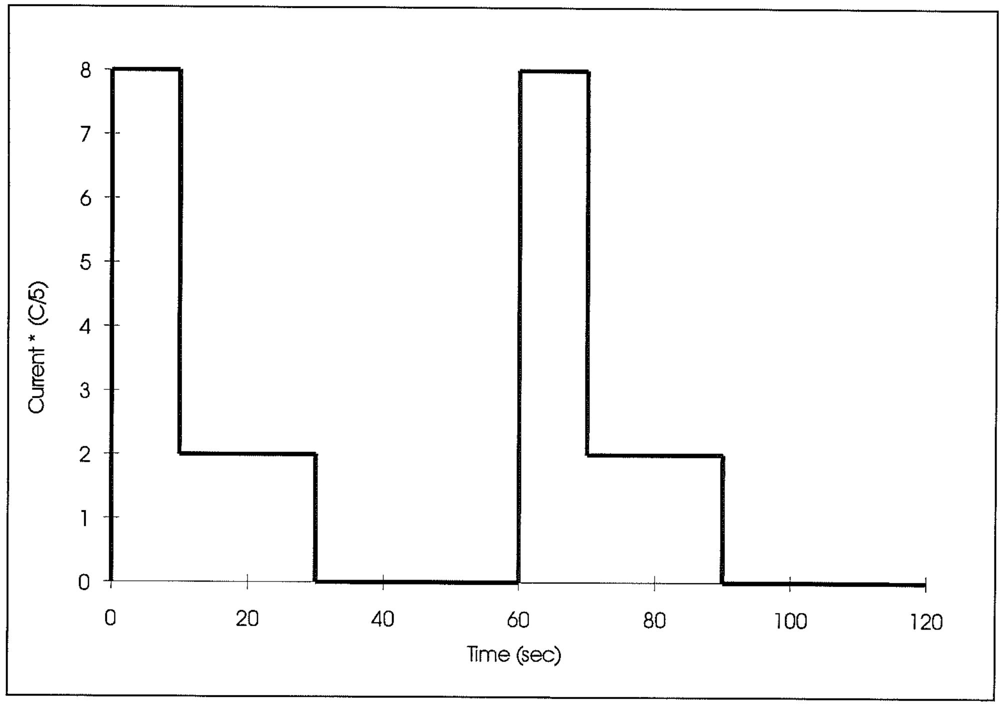
Time (sec)	Sp. Power (W/kg)
0-11	0
11-15	17.0
15-23	3.0
23-28	-4.3
28-49	0
49-61	27.9
61-85	7.8
85-96	-8.6
96-117	0
117-143	35.5
143-155	16.0
155-163	-13.0
163-176	8.9
176-188	-9.4
188-195	0



Current Profile for two cycles of the TC69/WG3

Time (sec)	Current
0-10	8 * C/5
10-30	2 * C/5
30-60	0

The cycle will be repeated until Cut-off-Voltage specified by the battery manufacturer has been reached.


Current * (C/5)
Time (sec)