15th Triennial World Congress, Barcelona, Spain

# DETERMINATION OF THE FUEL-OPTIMAL TRAJECTORY FOR A VEHICLE ALONG A KNOWN ROUTE

**Frank Kirschbaum* Michael Back * Martin Hart ***

* DaimlerChrysler *AG – Research and Technology REM/EP, Stuttgart*

Abstract: Integration of topology data of the entire road ahead into the powertrain control unit in combination with hybrid electric powertrains offers a huge potential for optimising the control strategy. The intension of the present paper is to show a methodology for computing the maximum potential of fuel saving over a specified route. For this offline calculation a non-linear state-space model of the longitudinal dynamics is used to find the fuel optimal trajectory using Bellman's Dynamic Programming (DP). As the model is of third order with three control inputs and DP can only be applied to systems with a maximum sum of inputs and states of four, the iterative variant of DP is used. This special method described in this paper allows the solution of optimisation problems with a number of states and inputs up to six, without leading to insufferable computing times.

Keywords: non-linear system, dynamic optimisation, iterative dynamic programming, fuel-optimal operating, hybrid powertrain

## 1. INTRODUCTION

Hybrid electric powertrains are useful to reduce fuel consumption of cars. In order to effect an economy of fuel, a suitable control strategy is needed. The potential of fuel saving depends on the route properties, e.g. street-slope and desired velocity. Generally speaking, the electric hybrid powertrain gives a new degree of freedom (the battery as another energy storage and the electrical engine as a second torque source) and the telematics information allows us to use this degree of freedom in a optimal way.

The desire for fuel-optimal behaviour leads to a dynamic optimisation problem. If the velocity profile is given, the torque demand is directly calculable. As dynamic state only the SOC (state of charge) $q$ remains, and the system inputs are the ratio of combustion engine torque, and electrical engine torque and the gear $i$ [BaS02]. But if only a desired velocity is given, the vehicle speed $v$ and vehicle position $s$ are also dynamic states. Additionally the number of inputs increases, because the combustion engine torque and electrical engine torque are then independent from each other, and it is necessary to use two different inputs for the two different torque sources. In this case the inputs are combustion engine torque $T_{\mathrm{C}}$, electrical engine Torque $T_{\mathrm{E}}$ and gear $i$. The task now is to compute the optimal control inputs $T_{\mathrm{C}}^*$, $T_{\mathrm{E}}^*$ and $i^*$, which minimize the cost-criteria

$$J = c_1 \left(q(t_{\mathrm{f}}) - q_{\mathrm{f,d}}\right)^2 + c_2 \left(v(t_{\mathrm{f}}) - v_{\mathrm{f,d}}\right)^2 + \int_{t_0}^{t_{\mathrm{f}}} c_3 \zeta(n, T_{\mathrm{C}}) + c_4 \left(v - v_{\mathrm{d}}\right)^2 \,\mathrm{d}t \rightarrow \min \quad (1)$$

Index f means the final value or rather the value at the end of the optimisation horizon and index d means the desired value. The first two terms of equation 1, the so called Mayer-criteria, take consider the obtained final state, the third term,

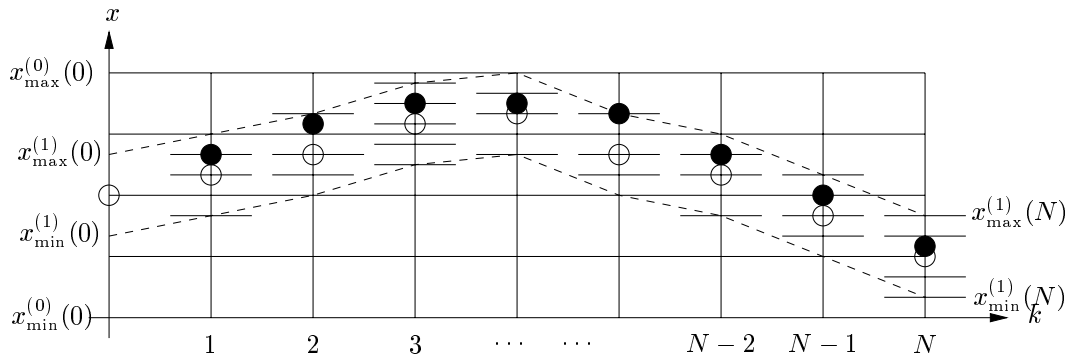


Fig. 1. Iterative Dynamic Programming – combination of Dynamic Programming with an adaptive search tube: example with one state

the LAGRANGE-criteria, regards the transient behaviour. Here the velocity $v$ shall track the desired velocity $v_{\mathrm{d}}$ , and the fuel consumption

$$\int_{t_0}^{t_{\mathrm{f}}} \zeta(n, T_{\mathrm{C}}\,)\,\mathrm{d}t \tag{2}$$

shall be minimized. The function $\zeta(n, T_{\mathrm{C}}\,)$ is the instantaneous fuel consumption and is dependent on engine speed and combustion engine torque. With the weighting coefficients $c_1$ to $c_4$ the 'importance' of the different terms is adjustable.

Overrating the final SOC residue by choosing the co-efficient $c_1$ to a greater extent increases the fuel consumption and vice versa. The desired final SOC is here always 0.5, this means the battery should be half full at the end of the optimisation horizon.

A similar relation exists between the fuel consumption, valued by co-efficient $c_3$, and the velocity residue, valued by co-efficient $c_4$. In principle the second term of equation 1, valued by co-efficient $c_3$, is implicitly included in the fourth term. Co-efficient $c_2$ is a useful possibility to influence the numerical behaviour of the algorithm.

Additionally the input and the state signals have to satisfy the mathematical model of the hybrid power train:

$$\begin{bmatrix} \dot{s} \\ \dot{v} \\ \dot{q} \end{bmatrix} = \underline{f}\left(\begin{bmatrix} s \\ v \\ q \end{bmatrix}, \begin{bmatrix} T_{\mathrm{C}} \\ T_{\mathrm{E}} \\ i \end{bmatrix}\right) \tag{3}$$

As with many non-linear relationships like maps for Torque and Losses or the non-continuous behaviour of the gear box this model is non-linear.

The task is now accurately defined: solving a non-linear dynamic optimisation problem. A well known numerical methodology for calculating solutions of dynamic optimisation problems is BELLMAN's dynamic programming [BeD62]. This methodology suits both non-linear and time variant cases and is ideal for determining the global optimal solution. Unfortunately dynamic programming needs a lot of computing power with high memory requirements to sum up states and inputs of six. This problem can be reduced by using a iterative mutation of the dynamic programming, the so called iterative dynamic programming.

## 2. ITERATIVE DYNAMIC PROGRAMMING

The first step in finding a numerical solution is to convert the time-continuous and state-continuous problem to one with discrete time and discrete states. By using a numerical integration methodology, the time-continuous differential equations are transformed into difference equations

$$\begin{bmatrix} s_{k+1} \\ v_{k+1} \\ q_{k+1} \end{bmatrix} = \underline{g}\left(\begin{bmatrix} s_k \\ v_k \\ q_k \end{bmatrix}, \begin{bmatrix} \phi_k \\ T_{\mathrm{E,k}} \\ i_k \end{bmatrix}\right) \tag{4}$$

and the cost-criteria is converted into the form

$$\begin{aligned} J = {} & c_1\,(q(N) - q_{\mathrm{f,d}})^2 + c_2\,(v(N) - v_{\mathrm{f,d}})^2 \\ & + \sum_{0}^{N-1} c_3\zeta(n, T_{\mathrm{C}}) + c_4\,(v - v_{\mathrm{d}})^2\;\mathrm{d}t \;\rightarrow\; \min \end{aligned} \tag{5}$$

It is now possible to formulate the optimisation problem in form of the BELLMAN's recursive equation

$$\begin{aligned} S^*\,[\underline{x}(k), k] = \min_{\underline{u}(k)\,\in\,U} \Big\{ & h\,[\underline{x}(N), N] + \\ & + \sum_{w=k}^{N-1} f_0\,[\underline{x}(w), \underline{u}(w), w] \Big\} \end{aligned} \tag{6}$$

(valid for the discrete time steps $0, 1, 2, \ldots, k, \ldots, N$). Equation 6 can be evaluated backwards, starting with the final state at the end of the optimisation horizon. This procedure is called backward dynamic programming. Starting from a given final state its computes the optimal trajectories starting from an arbitrary start state. A complimentary procedure known as forward dynamic

programming, but this approach owns no relevance in control applications, because it solves optimisation problems with a given start state to an arbitrary end state.

In order to evaluate the recursive equation (6) it is necessary to make the state space discrete. Then an n-dimensional grid approximates the n-dimensional state space. With bigger discretisation rates (and that means with more discrete grid lines) the accuracy, but also the needed computing power increases. To reduce this problem BELLMAN has already developed the basic idea of the Iterative Dynamic Programming (IDP) [BeD62]. The approach is to start with a coarse grid, in order to get a first, but at this time not very accurate solution. To increase the accuracy, the grid is constricted about the solution of the last computing cycle. A new appliance of the dynamic programming will now provide a new better solution with an increased accuracy. Figure 1 demonstrates this procedure for a problem with only one state. There are only five discrete state values and after the first use of the dynamic programming a solution can be found, marked by the non-filled circles. In the next step at the central points of the new grid will be shifted to the non-filled circles, and additionally the distance between two grid lines will be reduced. Then it is possible to restart the dynamic programming to get a better solution. From iteration step to iteration step the accuracy will increase, because the grid width decreases during the same procedure.

Generally speaking the grid width of the state-grid $\Delta\underline{x}$ and the grid width of the input-grid $\Delta\underline{u}$ will be contracted by iteration:

$$\Delta\underline{x}^{(w+1)} = \underline{\gamma}_{\underline{x}} \cdot \Delta\underline{x}^{(w)}\,, \tag{7}$$

$$\Delta\underline{u}^{(w+1)} = \underline{\gamma}_{\underline{u}} \cdot \Delta\underline{u}^{(w)} \tag{8}$$

The exponent $^{(i)}$ means the $i$-th iteration and $\underline{\gamma}_{\underline{x}}$ and $\underline{\gamma}_{\underline{u}}$ are the contraction matrices

$$\underline{\gamma}_{\underline{x}} = \begin{bmatrix} 0<\gamma_{x_1}<1 & 0 & \cdots & 0 \\ 0 & 0<\gamma_{x_2}<1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0<\gamma_{x_n}<1 \end{bmatrix} \tag{9}$$

$$\underline{\gamma}_{\underline{u}} = \begin{bmatrix} 0<\gamma_{u_1}<1 & 0 & \cdots & 0 \\ 0 & 0<\gamma_{u_2}<1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0<\gamma_{u_m}<1 \end{bmatrix} \tag{10}$$

with

$$n = \dim\{\underline{x}\}\,, \qquad m = \dim\{\underline{u}\} \tag{11}$$

as the dimensions of state-space and input-space. From iteration step to iteration step also the central points of the new grid are shifted to the solution of the step before:

$$\frac{\underline{x}_{\max}^{(w+1)}(k) - \underline{x}_{\min}^{(w+1)}(k)}{2} = \underline{x}^{*(w)} \tag{12}$$

$$\frac{\underline{u}_{\max}^{(w+1)}(k) - \underline{u}_{\min}^{(w+1)}(k)}{2} = \underline{u}^{*(w)} \tag{13}$$

The indices $_{\max}$ and $_{\min}$ Are marking the upper and lower limit of the grids.

In simple words, the iterative dynamic programming is just the combination of dynamic programming with a 'adaptive search tube' [Sch66].

Basically it is possible to implement the methodology in MatLab-Code. Because of the MatLab overhead this leads to insufferable high computing times. As a consequence, the methodology was implemented with the programming language `C`. The developed software `BALU` has no restrictions in the number of states and inputs, but in the actual version only continues states are supported [KiS00]. A version without this restriction is under development. The procedure to solve an optimisation problem with `BALU` is

- defining the cost-criteria in `BALU.C`
- defining the number and limitations of inputs and states in `BALU.C`
- defining the number of discrete inputs and states as well as the number of time steps and the sampling time in `BALU.C`
- using the Real-Time-Workshop of Matlab/Simulink or the AutoCoder of MATRIXx/SystemBuild to generate `C`-Code of your model
- merging of `BALU.C` and the generated model source code
- compiling, linking and running the generated software

The next chapter presents some results produced by `BALU`. All sampling times for the following results are 0.5 seconds. The modelled vehicle is based on a real existing super mini car with a DIESEL-engine and Common-Rail Direct Injection (CDI), a synchronous electrical engine and a NiMH-Battery. The maximum electrical engine torque is approximately a third of the maximum combustion engine torque.

## 3. RESULTS

### 3.1 *Route with a given slope-profile*

In the first case the optimal solution for a real existing test route is in demand. This 19510 m long route is formed from a surrounding road of Stuttgart. The street slope $\alpha$ is given along the route(see figure 2).

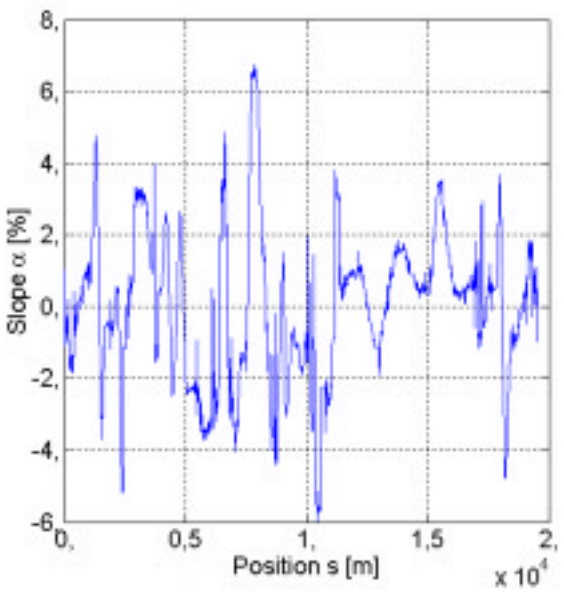


Fig. 2. Slope-profile of the 'Kernen-Route'

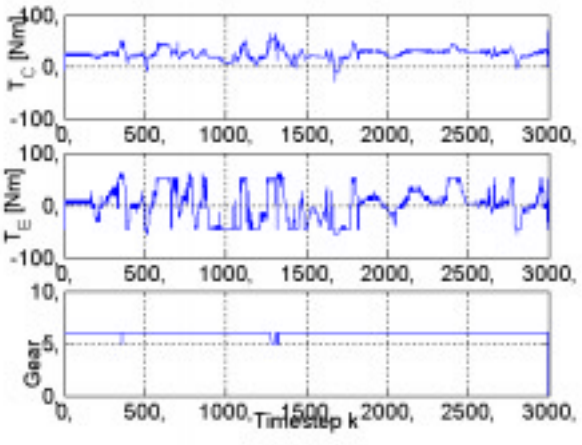


Fig. 3. Optimal inputs $T_{\mathrm{C}}^*$, $T_{\mathrm{E}}^*$ and $i^*$ for the test route

Figure 3 shows the optimal input signals for this route and the cost-criteria 1: The combustion engine torque $T_{\mathrm{C}}^*$, the optimal electrical engine torque $T_{\mathrm{E}}^*$ and the optimal gear shift $i^*$. The desired velocity is constantly 50 km/h. Both recuperation and load shift are used by the algorithm to reduce the fuel consumption. Figure 4 shows the dedicated optimal trajectories: the vehicle position $s^*$, vehicle velocity $v^*$, and state of charge $q^*$. For reducing the fuel consumption it is useful to undershoot the desired velocity. The size of the undershoot depends on the value of the coefficient $c_4$ in the cost-criteria 1. With the chosen values the reducing of fuel consumption is now considered 'more important' than reaching the desired velocity.

One complication of the used test route is, that there are not many parts where recuperation is possible. Figure 5 shows how the algorithm uses load shifting for saving fuel. Represented are the combustion engine efficiency on this route for a conventional vehicle; the combustion engine efficiency for the hybrid vehicle and the maximal possible efficiency. Most of the time the combustion engine load is shifted to a better efficiency.

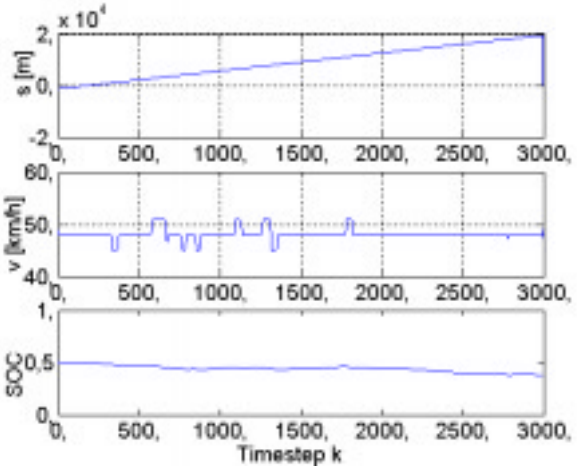


Fig. 4. Optimal states $s^*$, $v^*$ and $q^*$ for the 'Kernen-Route'

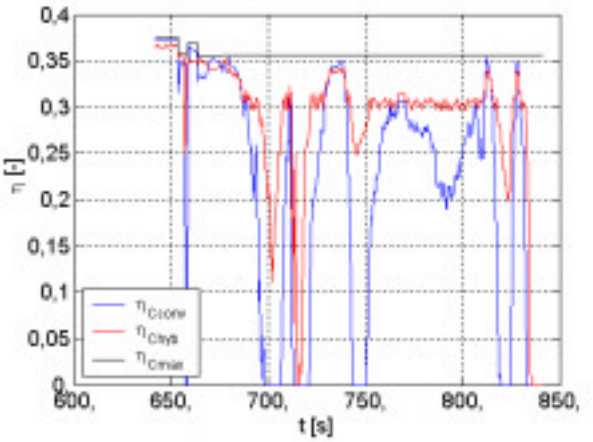


Fig. 5. Load shifting

Particularly the input signals (see figure 3) are impinged with noise. One source of it is quantisation noise. Another source is the measurement noise of the slope profile in figure 2. For the first two inputs, (the torques of the combustion engine and the electrical engine) it is possible to smooth the signals without any loss of optimality. This does not apply to the gear input, as this input is discrete in reality. To reduce this noise it would be necessary to establish a further state and to value it in the cost-criteria.

### 3.2 *Route with a given velocity-profile – European Drive Cycle*

In the second example the street slope is constant and the velocity along the route is given. This means, that in the cost-criteria 1 the desired velocity $v_{\mathrm{d}}(s)$ is a function of the vehicle position. Figure 6 shows the desired velocity for the European Drive Cycle (EDC); the figures 7 and 8, similar to the case of subsection 3.1, shows the results of the optimisation process. The desired velocity of the EDC is well fitted (see figure 8),

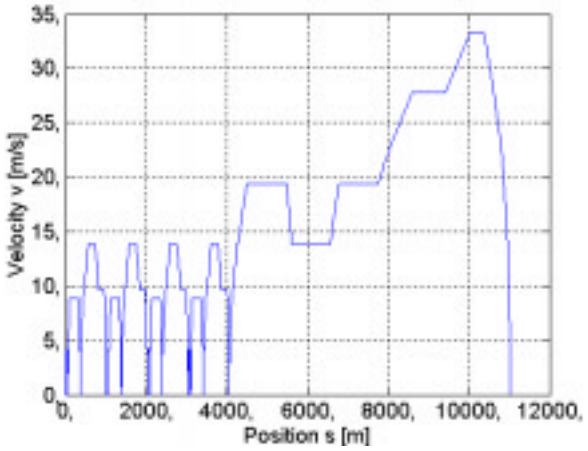


Fig. 6. Desired velocity for the European Drive Cycle (ECD)

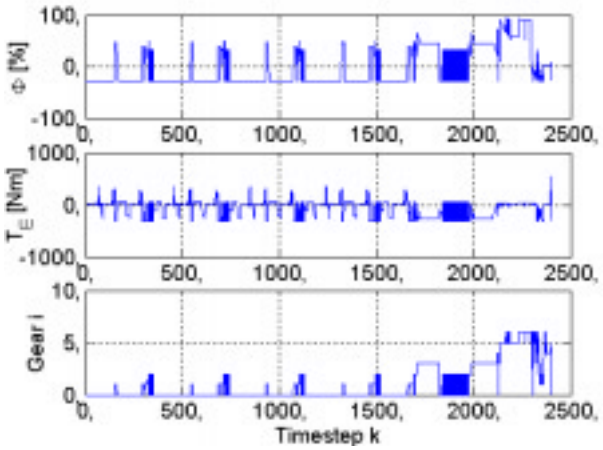


Fig. 7. Optimal inputs $\phi^*$, $T_E^*$ and $i^*$ for the EDC

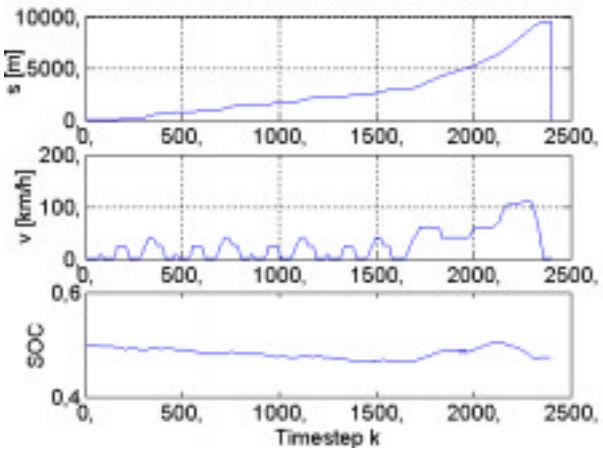


Fig. 8. Optimal states $s^*$, $v^*$ and $q^*$ for the EDC

and the state of charge at the end of the cycle remains in a allowed range. The variation to a desired final value or the deviation to the desired velocity is determined by the constants $c_1$, $c_2$, and $c_4$.

## 4. CONCLUSION

To find out the energy-optimal behaviour of hybrid powertrains, it is necessary to solve a nonlinear dynamic optimisation problem. One methodology to handle non-linear and time-variant optimisation problems is the dynamic programming technique. Because of higher sums of inputs and states it is sensible to use an iterative variant on dynamic programming. This combination of Dynamic Programming with an adaptive search tube reduces the needed computing time and the memory requirement as well. The methodology is suitable for the cases street slope is given, velocity profile is given and its combination. For a proper choose of time discretisation and state space quantisation, the methodology computes the global optimal control, thereby it is useful to determine the maximum potential of fuel-saving for a given powertrain driving along a given route.

## 5. REFERENCES

[BaS02] Back, M.; Simons, M.; Kirschbaum, F.; Krebs, V.: *Predictive Control of Powertrains*, submitted to IFAC International Federation of Automatic Control, 15th IFAC World Congress 2002, July 21-26, Barcelona 2002.

[BeD62] Bellman, R.E.; Dreyfus, S.E.: *Applied Dynamic Programming*, Princeton University Press, Princeton 1962.

[KiS00] Kirschbaum, F.; Stöhr, G.; Back, M.: *Suboptimal Robust Control of Fast Electro-magnetic Actuators*, IFAC International Federation of Automatic Control, Mechatronics 2000, Darmstadt 2000.

[Sch66] Schulze, K.-K.: *Methode des adaptiven Suchschlauchs zur Lösung von Variationsproblemen mit Dynamic-Programming-Verfahren*, Elektronische Datenverarbeitung 8, 1966.