2014 IEEE Intelligent Vehicles Symposium (IV)
June 8-11, 2014. Dearborn, Michigan, USA

Approximate Dynamic Programming Methods Applied to Far Trajectory Planning in Optimal Control

Hans-Georg Wahl[1], Marc Holzäpfel[2] and Frank Gauterin[1]

Abstract— **There are several applications that need far trajectory planning within optimal control problems. One use case is the optimal predictive control of plug-in hybrid electric vehicles (PHEV). It is possible to find an optimal control with models of the vehicle and environment and a fast optimization algorithm that is capable to calculate over long distances within seconds so that dynamic information such as traffic can be recognized quickly. In this paper, several methods based on dynamic programming (DP) are combined to generate approximated optimal control trajectories with a reduced computational complexity to achieve close-to-real-time application. The resulting trajectories are transferred as strategic planning trajectory to subordinated vehicle controllers. Close-to-optimal trajectories are achieved with a large reduction in memory.**

I. INTRODUCTION

Dynamic programming (DP) is basically used to solve problems in sequential decision processes. Different communities use different vocabulary comprising similar concepts or algorithms [1]. In engineering, the field where DP is applied is often referred to as control theory.

One application in engineering is the optimal control of a vehicle, where the drive train is so complex that energy management systems (EMS) are required to guarantee drivability and an energy-optimal control.

Plug-in hybrid electric vehicles (PHEV) represent one of the most complex drive train set-ups. With two energy storages that can be recharged or refueled before the journey and the ability to convert and recover energy through recuperation during the journey, an optimal energy management is needed. With rising battery capacities and electric machine power, the strategic planning of driving modes becomes relevant for example, the decision of whether to drive in the electric mode or with the internal combustion engine. In addition, the chosen route as well as the dynamic behavior of traffic incidents have a huge impact on the optimal control that comprises predictive information. In the worst case, the information along the whole route have huge influence on individual decisions. Within model-predictive control, the local control decision is made based on a guiding control trajectory that comprises of global information beyond control but of local influence [2].

Model-predicted control approaches are applied successfully in hybrid electric vehicles (HEV) [3]. Digital maps and telematic information were used to estimate the requested power trajectory in [4] based on information on the future route and in [3] based on a driver model and fixed values for accelerations. The author of [5] applied a combination of driver assistance and energy management system in hybrid electric trucks.

In previous studies, methods to apply predictive cruise control in HEV were presented with a focus on complexity reduction through an iterative search of optimal trajectories [6]. In [7], a method to handle even high-dimensional optimization problems was introduced.

In this paper, we present several DP methods to approximate optimal control trajectories over a long distance with the main focus on memory reduction and savings in computing time.

This paper is organized as follows: Section 2 starts with the models used within the following sections. Section 3 starts with an introduction to discrete dynamic programming emphasizing the potentials that are given for memory and computing time improvements. A benchmark for comparison at a later stage is determined. Sections 4 to 6 present different variations of the DP and their improvement potentials. All methods are combined in Section 7 into an approximate dynamic programming method. Section 8 closes with a conclusion and suggests next steps.

II. MODELING

In [7], a method is proposed to solve the multidimensional optimization problem comprising the search for an optimal velocity and torque split control trajectory. The problem is split into two parts: The search for an optimal velocity trajectory resulting in power demand over distance and the optimal torque split for the needed power over distance expressed in a state-of-charge (SOC) trajectory.

For far trajectory planning, the route and traffic information uncertainty along the route from the actual position rises so that the precision due to splitting up of the optimization problem is sufficient. The author of [8] supports the assumption that "only the average information about the route is significant", especially in the case of a far-away destination.

The following methods are applied to the velocity trajectories, but can be transferred to the optimal torque split.

The models described in this paper are examples demonstrating the behavior of the presented methods. They can be replaced or modified by more precise or approximated elements.

A. Vehicle model

A quasi-static approach is chosen to model all relevant components of the drive train such as engine, decoupling

[1]H.-G. Wahl and F. Gauterin are with the Institute of Vehicle System Technology, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany hans-georg.wahl@kit.edu, frank.gauterin@kit.edu

[2]M. Holzäpfel is with the Dr. Ing. h.c. F. Porsche AG, 71287 Weissach, Germany marc.holzaepfel@porsche.de

clutch, gearbox, electric machine, and battery [9]. The vehicle model which is explained in more detail in [6] is extended to a PHEV with a larger battery and more powerful electric motor for exclusive electric driving.

The state $x_{vel,k}$ and control $u_{vel,k}$ vector for the optimal velocity trajectory is defined as:

$$x_{\text{vel},k} = \underline{v}, \quad u_{\text{vel},k} = \underline{F}_x \tag{1}$$

with $\underline{v}$ as position-dependent velocities and $\underline{F}_x$ being the total driving resistance force as the sum of rolling resistance, aerodynamic drag, grade resistance, acceleration force, and the effective longitudinal force in curves that are modeled with the single track model.

The state $x_{\text{soc},k}$ and control $u_{\text{soc},k}$ vector for the optimal SOC trajectory is defined as:

$$x_{soc,k} = \underline{s}, \quad u_{soc,k} = \begin{bmatrix} T_{\text{eng}} \\ T_{\text{mot}} \\ T_{\text{b}} \\ d_{\text{st}} \\ g \end{bmatrix} \tag{2}$$

where

$s =$ the current state of charge of the battery (SOC).
$T_{\text{eng}} =$ the engine torque.
$T_{\text{mot}} =$ the electric motor torque.
$T_{\text{b}} =$ the break torque.
$d_{\text{st}} =$ the status of the decoupling clutch.
$g =$ the current gear

The current gear is not part of the state vector and therefore not part of the optimization. The lowest possible gear for each velocity is chosen.

B. Route and traffic model

The route is given through a manual destination input over a navigation system. It is therefore fixed as long as no rerouting is done. Routing is not part of the optimization. If no input of the destination is provided, the destination can be estimated through historic data analysis of the GPS position and calendar time only [10].

In this paper, the digital map provides speed limit V_{leg} and altitude information along the route. The information is event-triggered and position-dependent to avoid redundancy, the distance steps in the optimization are therefore not equidistant. The optimization space is modeled such that only states are recognized that are reachable within comfortable driving conditions. This state space reduction is widely used in comparable optimization problems [3][7][14].

Average speed V_{avg} information as well as online traffic message channel (TMC) information V_{tmc} can be included by introducing an upper velocity limit V_{bound} where surpassing the boundary is penalized by high costs.

$$V_{\text{bound}} = \min\{V_{\text{leg}}, V_{\text{avg}}, V_{\text{tmc}}\} \tag{3}$$

The capability to predict the traffic information along the route to when it actually becomes relevant while passing this section has a major impact on the robustness of the results.

As test cycle, a route is chosen in the southwest of Germany with a length of 127 km and an altitude delta of 350 m (Fig. 2). 75% of the route are highways and the rest are country and city roads representing long-distance travel.

C. Costs

For the velocity trajectory, two criteria are considered in the costs for a state transition: The energy ξ_E and the average speed expressed by the deviation to V_{bound} as ξ_B. Both criteria are weighted by factors α and β and summed up to

$$\xi_{\text{vel},k} = \alpha \cdot \xi_E + \beta \cdot \xi_B \tag{4}$$

where

$$\begin{aligned} \xi_E &= F_x \cdot \Delta s + \Delta E_{\text{kin}} + \Delta E_{\text{pot}} \\ \xi_B &= \frac{\Delta s}{\Delta v} - \frac{\Delta s}{\Delta V_{\text{bound}}} \end{aligned} \tag{5}$$

For the SOC trajectory case, the costs for a state transition $\xi_{soc,k}$ consist in fuel consumption costs only for positive engine torque requests T_{eng} and rotational speed ω_{eng}:

$$\xi_{soc,k} = \begin{cases} f(T_{\text{eng}}, \omega_{\text{eng}}), & \text{if } T_{\text{eng}} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

III. Discrete Dynamic Programming

From [11][13], we assume that $\pi^* = \{\mu_0^*, \mu_1^*, \dots, \mu_{N-1}^*\}$ is the optimal policy for an N-stage optimization problem with optimal control actions μ_i^* and used up to stage i so that the current state vector is $\underline{x}_i$. Then the optimal policy for the sub-problem to minimize the cost $J^*_{\underline{x}_i}$ from stage i to N is $\pi_i^* = \{\mu_i^*, \mu_{i+1}^*, \dots, \mu_{N-1}^*\}$ with

$$J^*_{\underline{x}_i} = \min_{\pi \in \Pi} \left\{ \sum_{k=i}^{N-1} \xi_k(\underline{x}_k, \mu(\underline{x}_k)) + \xi_N(\underline{x}_N) \right\} \tag{7}$$

where $\xi_k(\underline{x}_k, \mu(\underline{x}_k))$ represents the cost-to-go function that values a state transition at stage k . With $\xi_N(\underline{x}_N)$, the costs for the last state can be valuated separately.

All possible policies $\pi \in \Pi$ are defined as a sequence of actions μ_k that lead one path through the state space $S = \{\underline{x}_k, k = 0, 1, \dots N-1\}$. This transition from one state $\underline{x}_k$ to the next state $\underline{x}_{k+1}$ through the action μ_k is defined through the functional equation:

$$\underline{x}_{k+1} = \underline{g}_k(\underline{x}_k, \underline{u}_k) \quad \text{for} \quad k = 0, 1, \dots N-1 \tag{8}$$

In our case $\underline{g}_k$ is a white box model (see Section II) representing the vehicle in the environment.

For the DP optimization process three major parts are important

- The model g: Offers the basic information about possible states, actions and how to value them.
- The policies (trajectories) π: Represent all possible paths over k in a path matrix.

- The cost function J_π: Values each state through each policy over k in a cost matrix.

The latter two contain all necessary information to apply the principle of optimality and find the optimal policy but have the greatest impact on memory consumption.

A. *Curse of dimensionality*

In discrete dynamic programming (DDP) without stochastic influence, two curses of dimensionality can be found [1]:

1) The state space: If the state variable $\underline{x}_k$ has I dimensions, and if $\underline{x}_{ki}$ can take L possible values, then we have up to L^I different states and therefore the same amount of possible policies.
2) The action space: In DDP all following states are visited so that the number of possible numbers is equal to the number of possible states

With the growth of dimensions the number of states grows which makes DDP impracticable because of an exponential rise of computation time and needed memory especially for real-time application.

In the case of far trajectory planning, the number of stages P representing the precision and route information used has a linear impact so that the complexity of the computing time is:

$$P \cdot L_d^I \cdot L_d^I = P \cdot (L_d^I)^2 \in \mathcal{O}(n^2), \text{with } n = L_d^I \tag{9}$$

In the case of the memory, the complexity is linear to the states:

$$P \cdot L^I \in \mathcal{O}(n), \text{with } n = L^I \tag{10}$$

In Fig. 1, the optimal number of states L^1 for a one-

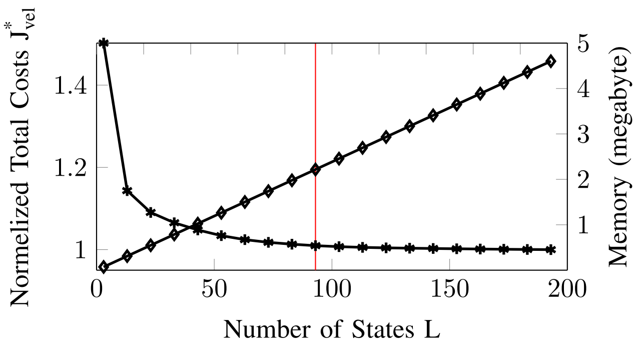

Fig. 1. Normalized total costs of optimal pathes of a variation of the number of states as search for an optimal benchmark

dimensional problem and the investigated route is plotted for DDP. Within one percent of the optimal total cost J^*, $L = 93$ states are chosen as DDP benchmark result with a memory need of 2.22 megabytes.

To gain computational performance and memory savings, several methods follow that allow improvements to a close real time application.

B. *Value and policy iteration*

Two types of algorithms in DP can be formulated for deterministic problems based on [1] to find the optimal policy. The value iteration (classic backward DDP) calculates cost values for getting to each possible state over k. It is the most widely used algorithm because it is simple to implement [1]. Each state in the optimization space is considered so that the maximum computation time and memory is needed but optimality is guaranteed within rounding errors. This algorithm therefore serves as a benchmark to the other one in this paper.

The policy iteration starts with one policy and tries to improve the summed cost value at stage N by exploring the optimization space, for example around found policies. With a good first guess, a fast convergence is possible.

In Figure 2, the two variants are compared in finding the optimal velocity for the test cycle.

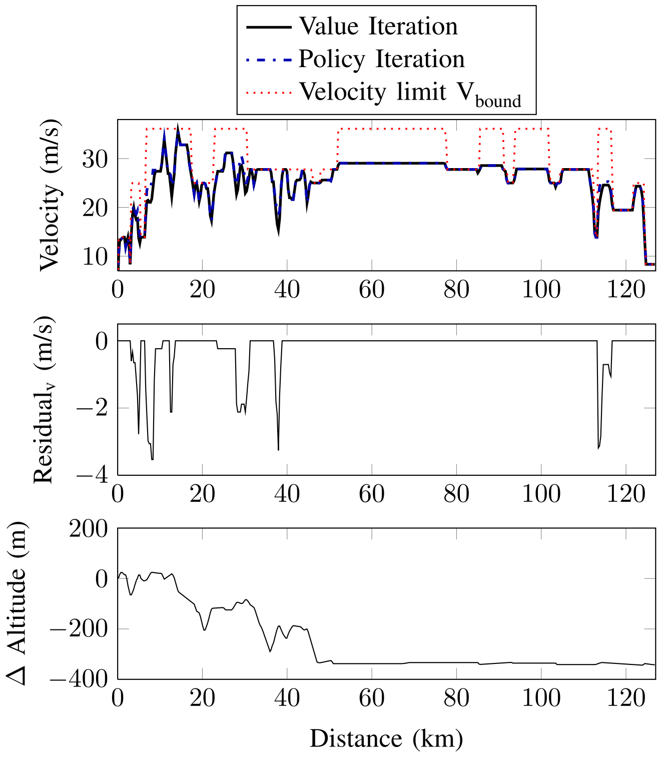

Fig. 2. Value versus policy iteration for the optimal velocity trajectory on the investigated real test cycle containing a mixed highway and countryroad scenario in the southwest of Germany

Combining both strategies results in having the advantage of both sides: Fast close-optimal policies by a fast improvement and convergence in a few iterations.

In Fig. 3, the different variants (value, policy and combined value with policy iteration) are compared. The policy iteration starts with the best costs due to a good initial guess and reaches the optimal costs within one percent by 22 iterations (right vertical line) and 1.21 seconds. A reduction of approximately 82% in computing time is achieved for the test cycle. The combination of both methods in this case leads to less iterations (left vertical line) with a higher demand in computing time with 2.39 seconds still more than halving the benchmark of a classical value iteration DDP.

IV. Iterative Dynamic Programming

Another method that reduces computing time and memory is iterative dynamic programming (IDP). It has been success-

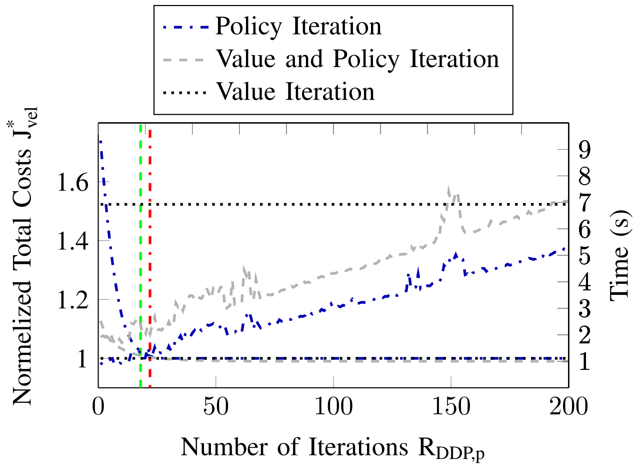

Fig. 3. Convergence of policy, value and combined iteration methods and their computing time on an 2.5 Gigahertz machine with 1534 DMIPS based on a non-optimised Dhrystone benchmark. The total costs are normalized to the total cost of the value iteration $J^*_{vel} = 1$ (left). The absolute computing time over the number of policy iterations is plotted to the right axis, while the value iteration is represented with $t = 6.9s$

fully applied by R. Luus [14] in chemical engineering as well as in former works [6].

The basic idea is to start with a coarse state space and to apply the DP value iteration method to find a policy. The policy found is used to adjust the grid around the policy with a reduction factor. After adjusting the grid and applying DP iteratively, a convergence leads to the optimal policy.

The reduction factor and the number of states used at each stage are adjustable parameters to save computing time and memory.

In Fig. 4, the number of states and iterations are varied and the total cost is plotted and normalized to the total cost of the benchmark DDP III so that the value one means an identical policy of IDP and DDP

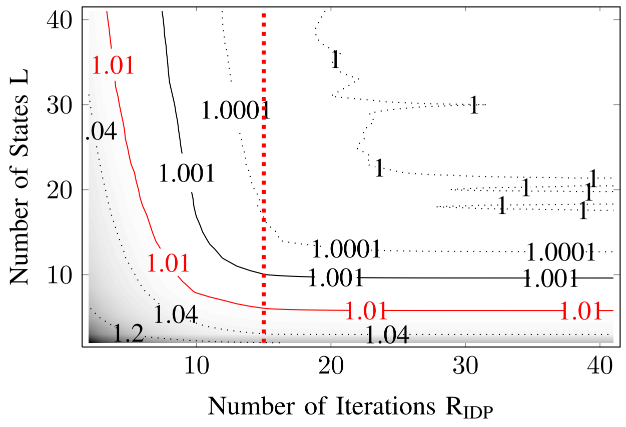

Fig. 4. Total costs of optimal paths over number of iterations and states computed by IDP and normalized by the optimal total cost of a DDP solution

On the Pareto front of one percent precision and with a reduction factor of 0.35, a computing time and memory optimal solution is found within 15 iterations and 6 states. With a computing time of 2.42 seconds and a memory need of 0.143 megabytes the time is reduced by 65% while the memory is reduced by 93% compared to the benchmark values with value iteration in DDP. For a one-per-mill precision, 10 states are needed within the same number of iterations. The computing time rises to 3.24 seconds (53%) and 0.238 megabytes (89%) memory.

V. Continuous Dynamic Programming

In DDP the major problem from a memory point of view is the dependency of the precision on the discretization of the represented state space. A more detailed state space involves high memory and computing time requirements. A coarse state space leads to interpolation effects and errors. In [7], a continuous state space is introduced that separates the calculation precision from the needed memory by defining interval boxes instead of discrete states. Within this box, the best local transition decides on the state and the corresponding value is saved and used as starting node at the next stage. This method is called continuous dynamic programming (CDP).

CDP allows to separate the state space from the action space, which means that the state space memory can be reduced while the number of transitions rises, or vice versa. Because the action space size can differ from the state space the complexity changes, and the second curse remains [1]:

The control vector $\underline{u}_k = \mu_k(\underline{x}_k)$ has K dimensions. $\underline{u}_{ki}$ can take on N outcomes, we have up to N^K outcomes. The complexity in computing time changes compared to DDP and declines if $L_c^I \cdot N_c^K < (L_d^I)^2$:

$$P \cdot L_c^I \cdot N_c^K \in \mathcal{O}(n), \text{with } n = L_c^I \cdot N_c^K \tag{11}$$

In this paper, no local preselection of the transition as well as no heuristics for the sizes of the intervals were performed. Both the state boxes and the chosen transitions are equidistant.

VI. One-Step Model Predictive Control

All found policies π contain important information in DP. They are saved in a path matrix containing the predecessor state for each state at stage k. Therefore, the memory size is $P \cdot L^I$. For example, a one-dimensional state space $L = 1$ with a discretization of $I = 100$ values over $P = 1000$ stages with integer data type needs 98 kilobytes in memory.

One way to replace the path matrix that is needed to reconstruct the optimal path is a method called one-step model predictive control (OSMPC).

The author of [8] used the method to save calculation time when applying DP on predefined routes. There, the cost matrix is calculated one time for a given drive cycle and the OSMPC is used as fast and robust online controller that has good results even if the cycle behavior changes.

In this paper, the OSMPC method uses the cost matrix $J^M(\underline{x}_k)$ containing all summed-up costs J_π for all states and policies as one-step look ahead guidance to reconstruct the optimal policy, fast. Starting at stage N with the found optimal state x_N^*, a backward value iteration is calculated at stage $k = N - 1$ with all μ_k resulting in x_N^*. The optimal action μ_k^* is chosen by finding the best cost-to-go J_k^+:

$$J_k^+ = \min\left\{\xi_k(\underline{x}_k, \mu(\underline{x}_k)) + J_k^M(\underline{x}_k)\right\} \tag{12}$$

With Eq. 8 and μ_k^* the optimal state x_k^* is found and the procedure is repeated until $k = 0$. Because the global information about the optimal total cost is stored in $J^M(\underline{x}_k)$, a local optimization in a one stage action ahead $J^M(\underline{x}_k)$ leads to the global action.

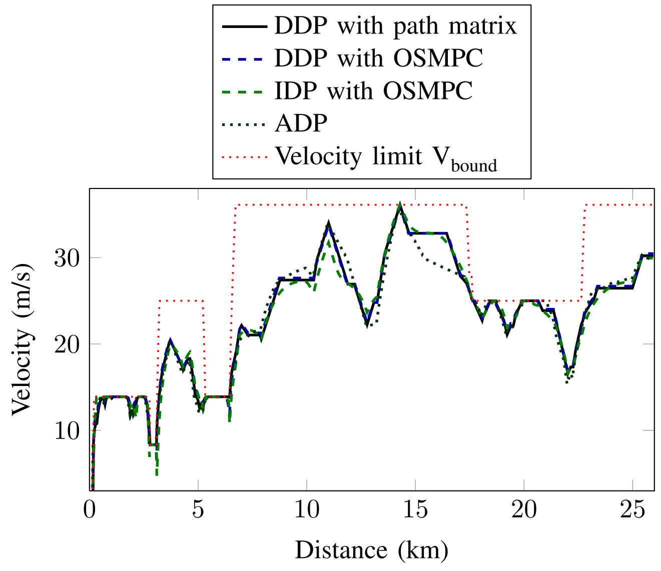

Fig. 5. Comparison of a policy reconstructed by the path matrix and the OSMPC with different investigated methods on the first 25 km of the test cycle

A policy reconstructed by OSMPC and a policy reconstructed by the path matrix are shown in Fig. 5. The OSMPC policy is very close to the policy reconstructed by a path matrix. The total cost of the OSMPC policy is 2.75% higher than that of the compared policy. The extra computing time is significantly low. At a distance of 3 and 6.5 kilometers, respectively, the OSMPC undershoots because of the short information of only one stage at a critical acceleration phase along the V_{bound}. A two- or three-step model-predictive controller can reduce these effects.

With the OSMPC method, the memory is reduced because no path matrix is needed anymore. The method can also be combined with IDP (See Fig. 5).

VII. Approximate Dynamic Programming

Approximating the cost function is called approximate dynamic programming (ADP) [1][11]. The idea is to reduce the complexity by approximating information needed for DP. Global optimality is not guaranteed, but can be approximated through different techniques.

The cost matrix still needs memory and plays an important role in DP. Because of its convex and steady shape it can be easily fitted by polynomials. The order of the polynomials has a great impact on the resulting precision but can also lead to a unstable behavior with rising number.

In our case, the shape of the cost function is fitted at each stage with linear regression and a polynomial of small order. The linear regression problem can be defined as [1]:

$$\min_{\Theta_i} \sum_{m=1}^{n} \left(J_i^m - (\Theta_{i0} + \sum_{j=1}^{L} \Theta_{i,j} v_j^m) \right)^2 \tag{13}$$

where c^n is the observation of our n predicted variables which represent the costs. $v_1^n, v_2^n, \ldots, v_L^n$ are the observations or values for each state. The optimal parameter vector Θ^* is found in a batch routine:

$$\Theta^* = \left[(X^n)^T X^n\right]^{-1} (X^n)^T Y^n \tag{14}$$

where

$$X_i^n = \begin{pmatrix} v_0^1 & v_1^1 & \cdots & v_L^1 \\ v_0^2 & v_1^2 & \cdots & v_L^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_0^n & v_1^n & \cdots & v_L^n \end{pmatrix} \tag{15}$$

is the n by $L+1$ matrix and

$$Y_i^n = \begin{pmatrix} J_i^1 \\ J_i^2 \\ \vdots \\ J_i^n \end{pmatrix} \tag{16}$$

is the vector of observations.

Investigating the cost matrix J^M in Fig. 6 results in the choice of a second-order polynomial:

$$J_{fit,i}^n = \Theta_i(0) + \Theta_i(1) \cdot v_{fit,i}^n + \Theta_i(2) \cdot (v_{fit,i}^n)^2 \tag{17}$$

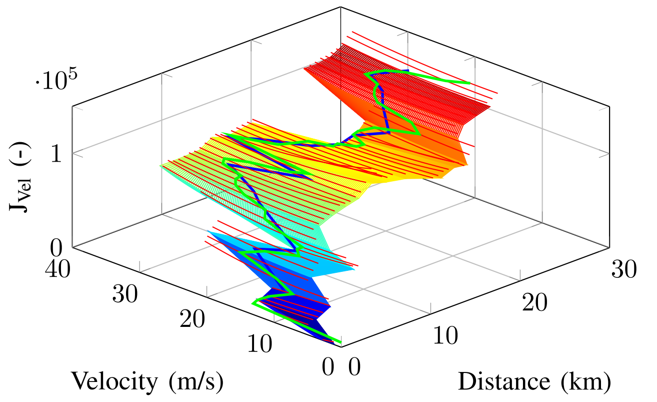

Fig. 6. The cost matrix J^M is overlaid by the cost function J_{fit} (red) represented by polynomials at each stage. The optimal policies are plotted over the corresponding costs. The optimal policy with DDP (blue) and the optimal policy with ADP (green) are shown.

A. Combining the methods

ADP is applied in a continuous state space (CDP), with iterative reduction of the boundaries (IDP), with linear regression fit of the cost matrix and a replacement of the path matrix by the OSMPC (see Fig. 5). The OSMPC is adjusted so that a reconstruction of the policy is possible on the basis of an approximated cost function J_{fit}. The cost-to-go function in Eq. 12 is modified:

$$J_k^+ = \min \left\{ \xi_k(\underline{x}_k, \mu(\underline{x}_k)) + J_{fit,k}(\underline{x}_k) \right\} \tag{18}$$

For the investigated test cycle, the linear regression is applied at each stage $i = 1 \dots N$ with a set of observations with the minimal costs instead of saving the results in the cost matrix.

In Fig. 7, samples of optimal fits are shown at six distances ($d_i, i = 5, 8, 11, 17, 23, 29$). The x axis represents the values v normalized to the investigated value range while the y axis represents the cost function $J_{fit,i}$ at stage i.

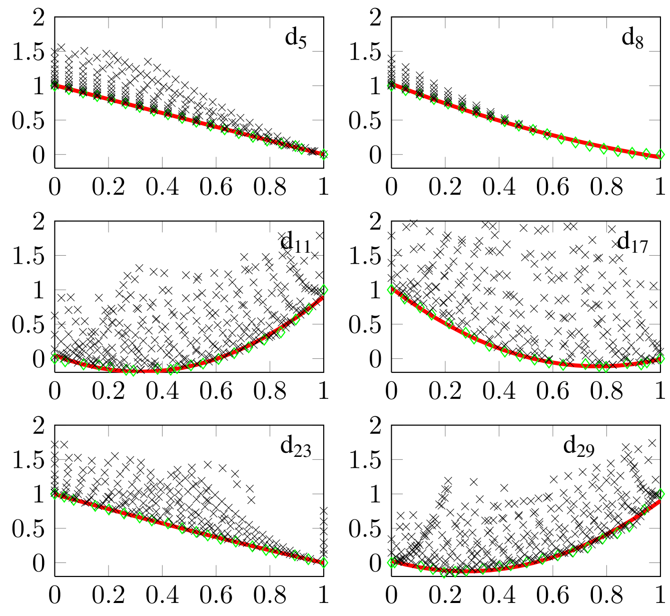

Fig. 7. Samples of the linear regression fit of the cost function at six different stages. Cost values (black crosses) are plotted over the normalized state spaces. The Pareto-optimal costs (green crosses) are fitted (red) at each stage.

With three iterations, 20 velocity boxes representing the continuous state space, the computing time is 1.478 seconds. A reduction by 78% compared to the benchmark is achieved. The memory is reduced by over 99% to 15 kilobytes with a cost function precision of 7%.

The optimal velocity policies are used as requested power over distance to find the optimal torque split. The effects of the reduced precision in case of the ADP method are shown in Fig. 8. The summed delta costs between the two SOC trajectories calculated based on v^*_{DDP} and v^*_{ADP} result in a 0.97% deviation. This proofs again the assumption that a reduced precision in the optimal velocity trajectory (7%) has a minor impact on the optimal SOC trajectory (1%). An approximation is therefore acceptable.

VIII. CONCLUSIONS

In this paper, several methods to reduce computing time and memory within DP were presented for optimal control trajectories in PHEVs. Through iteration and state space adjustments, memory-consuming matrices can be reduced while iteration leads to the needed precision. All methods can be combined within ADP so that the memory can be reduced by 99% percent and the computing time by 72% percent compared to classic DP in this test scenario. The approximation leads to a precision within 7% of the

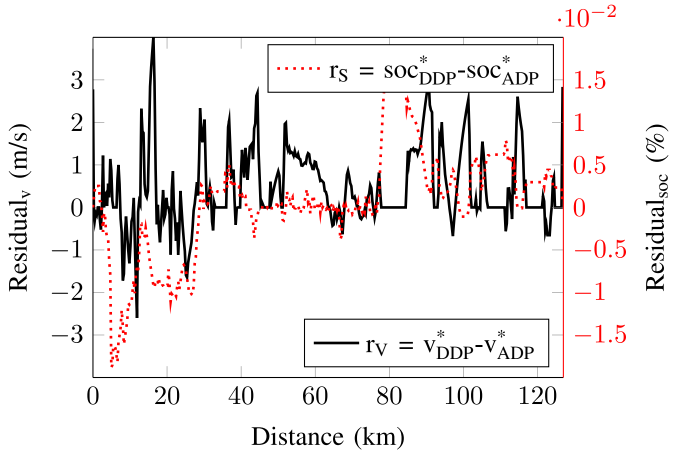

Fig. 8. The residual r_v of the approximated velocity is shown on the left side (black). The residuum of the approximated SOC r_s is shown on the right side (red).

compared results. Therefore, a close-to-real-time application to find optimal control trajectories even on long distances is possible. The next steps will consist in the reduction of noise effects as well as the investigation of the needed range which has an impact on the policies.

ACKNOWLEDGEMENT

The author would like to thank Philipp Köhler and Jakob Wiebe for their support.

REFERENCES

[1] Powell, W.B. (2007) 'Approximate Dynamic Programming - Solving the Curses of Dimensionality', *Wiley, New Jersey*, Vol. 1.
[2] Rawlings, J. (2000) 'Tutorial Overview of Model Predictive Control', *IEEE Control Systems Magazine*,
[3] Back, M. (2005) 'Prädiktive Antriebsregelung zum energie optimalen Betrieb von Hybridfahrzeugen', *PhD Thesis*, Karlsruhe Institute of Technology.
[4] Gong, Q. et al. (2007) 'Optimal Power Management of Plug-in HEV with Intelligent Transportation System', *International Conference on Advanced Intelligent Mechatronics*, Vol. 4, No. 7, pp.1–6.
[5] Van Keulen, T. et al. (2009) 'Predictive Cruise Control in Hybrid Electric Vehicles', *World Electric Vehicle Journal*, Vol. 3.
[6] Wahl, H.-G. et al. (2013) 'An Iterative Dynamic Programming Approach for the Global Optimal Control of Hybrid Electric Vehicles under Real-time Constraints', *IEEE Intelligent Vehicles Symposium 2013*, June 2013.
[7] Wahl, H.-G. et al. (2013) 'A Real-Time Capable Enhanced Dynamic Programming Approach for Predictive Optimal Cruise Control in Hybrid Electric Vehicles', *16th International IEEE Conference on Intelligent Transportation Systems*, October 2013.
[8] Johannesson, L. et al. (2007) 'Assessing the Potential of Predictive Control for Hybrid Electric Vehicles using Stochastic Dynamic Programming', *IEEE Transactions on Intelligent Transportation Systems*, Vol. 8. No. 1, March 2007, pp.71–83.
[9] Guzzella, L. et al. (2013) 'Vehicle propulsion systems', *Springer*, Vol. 3.
[10] Larsson, V. et al. (2012) 'Benefit of Route Recognition in Energy Management of Plug-in Hybrid Electric Vehicles', *Proceedings of the 2012 American Control Conference*, pp.1314–1320.
[11] Bertsekas, D. (2012) 'Dynamic programming and optimal control', *Athena Scientific, Belmont, Mass.*, Vol. 2., 4th edition.
[12] Bellman, R. (1957) 'Dynamic Programming', *Princeton University Press*,
[13] Bellman, R. et al. (1962) 'Applied Dynamic Programming', *Princeton University Press*,
[14] Luus, R. (2000) 'Iterative Dynamic Programming', *Chapman and Hall CRC*,