From boundary lines to boundary surfaces for dynamic programming with final state constraints ⋆

Willy Cottin, Yuqi Liu, Guillaume Colin, Alain Charlet, Sébastien Houillé

From boundary lines to boundary surfaces for dynamic programming with final state constraints ⋆

W. Cottin [*,**] **Y. Liu** [**] **G. Colin** [**] **A. Charlet** [**] **S. Houillé** [*]

[*] *Advance Research Department, Stellantis, 78943 Vélizy-Villacoublay, France (e-mail: willy.cottin@stellantis.com).*
[**] *Univ. Orléans, PRISME EA4229, Orléans, France (e-mail: guillaume.colin@univ-orleans.fr)*

Abstract: This article addresses the generalization of the boundary lines method for Dynamic Programming, called here boundary surface. This method can be applied on any system with 2 states as long as there are border constraints (initial or final). The method has been applied to two different problems. The first one is a simplified convex acceleration problem with final constraints on speed and position. The second problem is the energy minimization of a parallel hybrid electric vehicle along a trip in a non linear and non convex formulation. Results show a non negligible accuracy improvement despite the increase in calculation time. This can yield more accurate optimal results than those found by any possible mesh grids on an average computer for a reasonable calculation time.

Keywords: Dynamic Programming (DP), Boundary Lines (BL), Boundary Surface (BdS), Optimal Control Problem (OCP), Energy Management Strategy (EMS)

1 Introduction

Vehicle powertrain electrification is considered an efficient solution to the problem of reducing CO_2 emissions from the automotive industry. Electrification requires multiple power sources in one vehicle, making a control to distribute the powers, called Energy Management Strategy (EMS), necessary. To evaluate the performance of strategies, criteria are mandatory. Moreover, an optimal control policy can help to correctly size a given system. Naturally in the automotive field, the first criterion to be minimized is the fuel consumption. However, with electrification, it is more appropriate to minimize energy consumption. These strategies can be written as Optimal Control Problems (OCP) and solved using optimal control methods. One method is Dynamic Programming (DP), a numerical method developed by R. Bellman in the 50s (Bellman, 1954). The OCP can also be written in its dual form and solved with the Pontryagin Maximum Principle (PMP) (Pontryagin, 1987).

Since PMP is an analytical method, the problem resolution is faster than with DP resolution. However, the main drawbacks of this method are constraints and non linearity considerations throughout the simulation, which cannot be easily handled with PMP but are straightforward with DP. This is mainly why DP is one of the reference methods to solve OCP. As DP is a numerical method, it explores all the possibilities and then determines the optimal control policy to apply to the dynamic system while minimizing a given criterion.

Because of the computation time and memory complexity of DP, research has been conducted to optimize it in order to handle increasingly complex systems. For example, Approximate Dynamic Programming (ADP) (Bertsekas, 2008) replaces the cost-to-go matrix in the backward step to avoid multiple interpolations. A second possibility is to use Iterative Dynamic Programming (IDP) (Luus, 2019). This solution aims to reduce time calculation by using a coarse mesh grid for all domains. Then, the same mesh grid is reused but domains are contracted around the previous solution. This provides an interesting accuracy for a reasonable calculation time. Another branch of research to reduce calculation time and calculation complexity has focused on Boundary Lines (BL) (Sundström et al., 2010). This consists in determining the extreme states that can be visited while satisfying final constraints. In the literature, a single state boundary line problem is well known. However the resolution of boundary lines for two states or more, called boundary surface (BdS) here, does not have an explicit solution. The study in (van Schijndel et al., 2014) attempted to estimate BdS for a two-states problem in order to mesh only in this area. This produces a more accurate result with the same mesh resolution.

In section 2, the Optimal Control Problem formulation is provided. Then in section 3, Dynamic Programming and the boundary surface method are explained. These methods are applied on two models in section 4. Section 5 concludes the paper.

⋆ This paper is supported by the Association Nationale de la Recherche et de la Technologie (ANRT) through a CIFRE contract. It is also developed under the collaborative framework OpenLab Energetics between Stellantis and PRISME laboratory.

2 Optimal Control Problem formulation

The Optimal Control Problem formulation for a system consists in minimizing a given criterion while respecting constraints. The cost function $J(\mathbf{x}, \mathbf{u})$ is composed of the instantaneous cost $L(\boldsymbol{x}(t), \boldsymbol{u}(t))$ that is the consequence of the inputs applied to the system and states which the system passes through. The second element $\Phi(\boldsymbol{x}(t_f))$ is a penalty term applied as a function of the final system state. The OCP in its continuous form is the following one:

$$\begin{cases} \min\limits_{u} J(\mathbf{x}, \mathbf{u}) = \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t))dt + \Phi\left(\boldsymbol{x}\left(t_f\right)\right) \\ \text{w.r.t.:} \\ \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{x}(t) \in \boldsymbol{X}_t, \forall t \geq t_0 \\ \boldsymbol{u}(t) \in \boldsymbol{U}_t, \forall t \geq t_0 \end{cases} \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{u}$ are respectively the state and the input vectors. $\dot{\boldsymbol{x}}$ is the state derivative (the dynamics) which depends on the inputs and the current states. The input and state sets are defined as:

$$\begin{aligned} \boldsymbol{U} &= \{\boldsymbol{u} \in \mathbb{R}^n, \boldsymbol{u}_{\min}(t) \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{\max}(t)\} \\ \boldsymbol{X} &= \{\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{x}_{\min}(t) \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\max}(t)\} \end{aligned} \tag{2}$$

Since DP is a numerical method, the OCP will be written in discrete form which requires a grid of inputs, states and time, giving:

$$\begin{cases} \min\limits_{u} J(\mathbf{x}, \mathbf{u}) = \sum_{k=0}^{N-1} L(k, \boldsymbol{x}_k, \boldsymbol{u}_k) + \Phi(\boldsymbol{x}_N) \\ \text{w.r.t.:} \\ \boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \boldsymbol{x}_k \in \boldsymbol{X}_k, \forall k \geq 0 \\ \boldsymbol{u}_k \in \boldsymbol{U}_k, \forall k \geq 0 \end{cases} \tag{3}$$

Here, $\Phi(\boldsymbol{x}_N)$ is evaluated at each state mesh point at final time. There are several ways to treat this. One is to impose an infinite penalty for all final states that do not respect the constraint, as shown in equation (4).

$$\left\{ \begin{array}{c} \text{if } \boldsymbol{x}_N \in \boldsymbol{X}_N, \Phi(\boldsymbol{x}_N) = 0 \\ \text{else, } \Phi(\boldsymbol{x}_N) = Inf \end{array} \right\} \tag{4}$$

Instead of considering infinity for $\Phi(\boldsymbol{x}_N)$, it is also possible to have a sufficiently large number which will help the algorithm to find solutions. Another possibility is to consider some smooth penalty. This is done by using a given penalty function depending on final states. This approach will not lead to an infeasible solution but it has to be well tuned to converge to the target states.

3 Dynamic Programming

3.1 Dynamic Programming algorithm

Dynamic Programming is a numerical method to solve OCPs. To do so, the discrete dynamic programming principle is used:

$$V(k, \boldsymbol{x}_k) = \min_{u}(L(k, \boldsymbol{x}_k, \boldsymbol{u}_k) + V(k+1, \boldsymbol{x}_{k+1})) \tag{5}$$

where V represents the cost-to-go function evaluated at each point in the space domain. The algorithm is divided in two parts. The first one, called backward, is a time loop going from the end to the beginning of the simulation. During this loop, the model is run in forward form: $\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$ with $\boldsymbol{x}_k$ all the possible states and $\boldsymbol{u}_k$ the possible inputs according to the mesh grid. Then the instantaneous cost $L(k, \boldsymbol{x}_k, \boldsymbol{u}_k)$ is determined. However, to compute the whole cost-to-go matrix, an interpolation is mandatory to estimate the value for all possibilities. This interpolation can be either a nearest one, linear or any other interpolation method. Finally, equation (5) is solved for every state mesh node and the inputs associated to this optimal cost are stored in an optimal input matrix.

The second part, called forward, is also a time loop but going from the beginning to the end of the problem, where optimal inputs are determined by using results from the previous part. Moreover, the optimal policy is applied and the optimal cost is calculated at each time step. In the end, the whole system trajectory is determined as well as the global optimal cost.

To respect the final constraints, the optimal cost-to-go matrix is initialized with some values. These values were explained in the previous section. If a hard penalty solution is used, some interpolation errors may occur due to the large values of penalties. Knowledge of the boundaries reduce/avoid contamination of the cost-to-go and improve DP accuracy.

3.2 Boundary Lines (BL)

Because the cost-to-go matrix can be contaminated with penalties, this may lead to a poor construction of the cost-to-go matrix and furthermore to a poor result. The role of boundary lines is to determine exactly where the frontier between the feasible and infeasible domain is. Thus, BL prevents this contamination from occurring.
BL operate with some assumptions. A first assumption is that all possible states existing in the boundary area are really feasible. This means that if a state is included in the area described by the boundary contour, then there is necessarily a solution to achieve the simulation in the final boundary area. In other words, it is impossible to have an area of infeasible points inside the boundary area. A second assumption is that the model used can be inverted. If not, it is impossible to do step 2.(a) of the following algorithm. BL are then determined by using the dynamic equations in the inverted form $\boldsymbol{x}_k = \boldsymbol{x}_{k+1} - f_k^{-1}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_k)$.

The algorithm used to determine BL is as follows:
1.Initialize the $\boldsymbol{x}_{end}$ with target boundary final states.
2.Backward loop in time for $k = N-1$ to 0:
(a) determine upper and lower boundary lines at step k from the inverted dynamic equations and the upper/lower positions at time $k+1$.
(b) store state solutions, associated input(s) and cost for upper and lower boundary.
If the state solutions at time k are outside the constraints, it is possible to enforce boundaries on these constraints and determine the associated inputs and costs. Lastly to determine boundaries and use the results during the backward step of DP, another time loop can be done before the backward one or it can be directly merged in the backward loop.

An example of the use of boundaries is shown in blue in Fig.1. From the point x_{k+1}^{i+1} it is possible to reach the blue square which has no cost-to-go information. In order to estimate this value, an interpolation with the two nearest meshed points is done. In this example, one of the two points is the lower boundary line. This interpolation can

be linear, nearest, etc. Thus the cost-to-go at the blue square with a linear interpolation is:

$$V(k+2, \boldsymbol{x}_{k+2}^{blue}) = V(k+2, \boldsymbol{x}_{k+2}^{i+2}) + (x_{k+2}^{blue} - x_{k+2}^{i+2}).\frac{V(k, \boldsymbol{x}_{k+2}^{bl_l}) - V(k+2, \boldsymbol{x}_{k+2}^{i+2})}{\boldsymbol{x}_{k+2}^{bl_l} - \boldsymbol{x}_{k+2}^{i+2}} \quad (6)$$

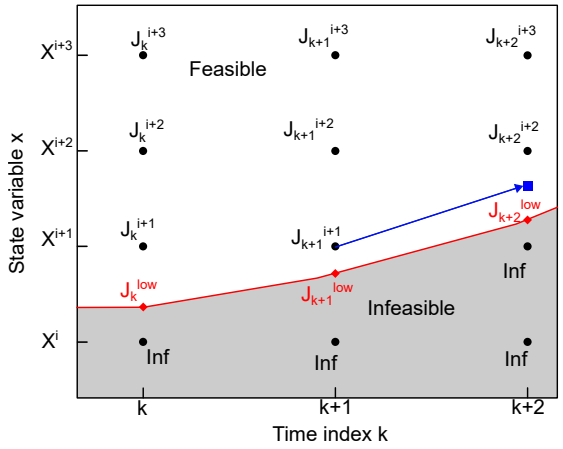

Fig. 1. Dynamic programming with cost-to-go of mesh and boundary lines scheme

Fig.1 shows the cost-to-go of meshed points (in space and time) whether in the feasible domain or not. Here, points outside the feasible zone have a cost-to-go set to infinite. Those zones are determined with boundary lines (the lower one in red here). In this scheme, all meshed points below the lower state at a certain time index are considered to be infeasible. Moreover, for each point belonging to the boundary line the cost-to-go is known and has been used to determine the cost-to-go for meshed points in the feasible domain.

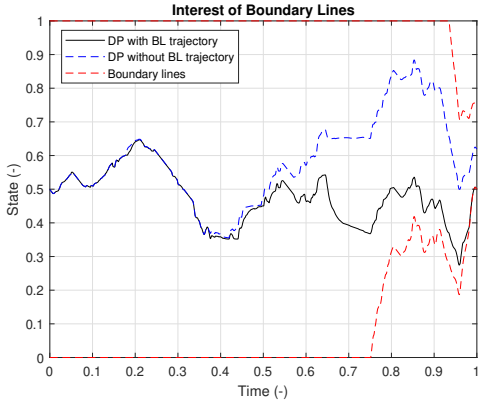

Fig. 2. Illustration of the interest of the boundary lines method

Fig. 2 shows the trajectory difference between a classic DP in a blue dashed line and DP with BL in black.Here the boundary lines are shown in red dashed lines and the same mesh is used for both DPs. At the beginning, the trajectories are similar. However, from the first half of the simulation,the black line drops below the blue one. The division appears because of the boundary method. Thanks to it, the cost-to-go matrix is now healthy in the area close to the lower boundary lines. This leads to a better result by decreasing the state in this example. At the end of the simulation, DP without BL arrives in the center of the target interval while DP with BL is led by the lower BL. Once more, DP without BL cannot find a solution at the border of the interval because of the proximity to infeasible zones.

3.3 Boundary Surfaces (BdS)

To generalize BL to more complex systems, borders have to be determined whatever the problem dimension is. However, when the system state dimensions are higher than one, determining the boundary borders is more difficult than for a single state problem. Because of the complexity of the system, the upper/lower boundary situation cannot be repeated since the borders are now continuous for a time step and not two points, as is the case with boundary lines.

In (van Schijndel et al., 2014), the reachable set of states is determined from the union of the previous backward reachable set and the convex set defined by the maximum estimated values of each state. This method gives an approximation of the reachable set of states which is used to refocus the meshing only in this space. Even if some infeasible spots exist in the reachable set the mesh resolution will naturally exclude them because of their results. However, because of the curse of dimensionality of DP, a solution that works with a coarse mesh is interesting. Thus in this article, the idea is to determine the reachable set of states using only the previous meshed set in backward. This makes it to work only with the initial mesh (coarse of fine) without any change over time.

Continuity of the borders involves the estimation of all possibilities from the boundary surface to determine a new boundary surface. The boundary surface method is used numerically such that non-analytical problems can be solved. The boundary $\boldsymbol{X}_k$ is the set of points at the edge of the feasible and infeasible zones. First of all, from the previous boundary set, all the inputs are applied to find all the points leading to the boundary P_k.

$$P_k = f^{-1}(\boldsymbol{X}_{k+1}, \boldsymbol{u}_k) \quad (7)$$

In this set, some points are outside the constraints O_k so they are removed $Q_k = P_k \backslash O_k$. Then, only points that are on the edge are kept to build the new boundary set (8). $\bar{Q}_k$ is the closure while $\overset{\circ}{Q}_k$ is the interior of the Q_k space.

$$\boldsymbol{X}_k = \bar{Q}_k \backslash \overset{\circ}{Q}_k \quad (8)$$

This is done from $k = N-1$ to 1 as in backward dynamic programming.

To limit the calculation time and memory resources required for boundary surfaces, it is possible to sample points belonging to this boundary. However, this sampling has to be done to conserve as much information as possible since points on the boundary could be massively placed in specific areas while some other areas could have very few points of information. Thus, if areas with fewer points are removed by the sampling, then there is a huge loss of information whereas if many points are removed from the massively populated areas, the loss of information is negligible. To perform the sampling, the perimeter of the boundary is divided into multiple parts. If the number of points in one part is higher than the maximum number

of points for a given part, then a sampling is done in between all the points existing in this precise part. With this solution, only parts with an excessive number of points are reduced to the maximum number of points available and parts with few points will keep all their points.

The boundary surface was determined here by using the boundary Matlab function. This function takes as input coordinates of the studied points along each dimension and returns indexes of points belonging to the boundary surface.

Fig.3 shows an example of an optimal trajectory with boundary surfaces on the same figure.

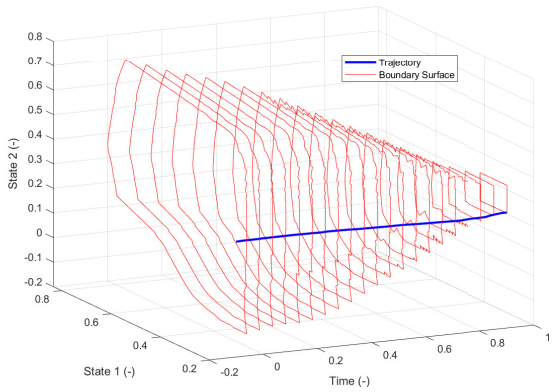

Fig. 3. BdS illustration with optimal trajectory in state space over time

4 Applications

4.1 Example 1: Acceleration problem with speed and position constraints

The first model used to illustrate the benefits of using BdS is a simple convex problem with one command and two states. This model (9) considers a vehicle trying to reach a fixed speed v_f and position p_f at a precise timing t_f. Thus the system input is the acceleration $u(t) = a(t)$ and states are speed $v(t)$ and position $p(t)$.

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{v} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{9}$$

Here, the vehicle has to reach the given position $p_f = 200$ m and speed $v_f = 100$ km/h at a given time $t_f = 10$ s. The optimization criteria is the integral of the square of the instantaneous acceleration, which leads to : $L(\boldsymbol{x}(t), \boldsymbol{u}(t)) = a(t)^2$.

4.1.1 Optimal Control Problem resolution with Pontryagin Maximum Principle

Since the problem is convex, it is possible to use PMP to solve the OCP and have a reference to compare DP with several solutions: with or without BdS and with smooth penalty.

In this case the Hamiltonian H can be written as:

$$\begin{aligned} &H(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = L(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^T \dot{\boldsymbol{x}} \\ \implies &H(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = u^2(t) + \lambda_v(t).u(t) + \lambda_p(t).v(t) \end{aligned} \tag{10}$$

where $\lambda_p(t)$ and $\lambda_v(t)$ are the co-states to be found.

As H is convex, the optimal input $u^*(t)$ can be found as follows:

$$\frac{\partial H(\boldsymbol{x}(t), \boldsymbol{u}(t), t)}{\partial u^*(t)} = 0 \implies u^*(t) = \frac{-\lambda_v(t)}{2} \tag{11}$$

while the co-state dynamics are defined by Eq. (12).

$$\dot{\boldsymbol{\lambda}}_i(t) = \frac{-\partial(H(\boldsymbol{x}(t), \boldsymbol{u}(t), t))}{\partial \boldsymbol{x}_i} \tag{12}$$

$$\text{w.r.t.:} \int_{t_0}^{t_f} u^*(t)dt = v_f - v_0, \int_{t_0}^{t_f} v^*(t)dt = p_f - p_0 \tag{13}$$

From (13), it is possible to have analytical expressions of the two co-states $\lambda_v(t)$ and $\lambda_p(t)$ and then include them in expressions of optimal acceleration, speed and position to fully resolve the problem.

4.1.2 Optimal Control Problem resolution with Dynamic Programming

Dynamic programming is also used to solve this simplified acceleration problem with constraints on speed and position. The objective is to compare DP with and without the BdS and also DP with smooth penalty (DPSP) to PMP. Table 1 sums up the main results. In Table 1, penalty

Methods	PMP	DPSP	DP w/o BdS	DP w/ BdS
Accel. Mesh (-)	-	201	201	201
Speed Mesh (-)	-	201	201	201
Position Mesh (-)	-	201	201	201
Time step (s)	-	0.1	0.1	0.1
Cost (-)	122	128.5	175.5	124.5
Final Speed (km/h)	100	100	110	101
Final position(m)	200	200	208	200

Table 1. Comparison of results between PMP and DP with/without BdS and DPSP

$\Phi(\boldsymbol{x}_N) = 10^{20}$ if $x_N \notin \boldsymbol{X}_N$ otherwise, $\Phi(\boldsymbol{x}_N) = 0$. For the smooth penalty DP, $\Phi(\boldsymbol{x}_N) = \sum_{i=1}^{2} C^i.(\boldsymbol{x}_N^i - \boldsymbol{x}_f^i)$ with $\boldsymbol{x}^1 = v$, $\boldsymbol{x}^2 = p$ and $C^{(1,2)} = 500$. This smooth penalty aims to have both states as close as possible to their targets. To achieve this coefficients were tuned by hand. Results show that DP without BdS has the highest cost and is the farthest away from the final constraints. DPSP satisfies the constraints and has a much lower cost than DP without BdS. However, the best DP solution is the one with BdS.

All solutions are shown in Fig.4 where the red dashed curve is the PMP solution, the blue dashed curve is DP without BdS, the green dashed curve is DP with smooth penalty and the black curve is DP with BdS. All accelerations have the same global behavior, i.e. full acceleration at the beginning followed by a linear decrease until the end. The PMP gives a pure linear acceleration while the three DP solutions show some fluctuations on the input linked to the numerical resolution of the problem. It is also important to note that the fluctuations are higher and occur throughout the simulation for DP without BdS, which leads to a sub optimal solution. The behavior of DPSP is close to that of DP with BdS. However, fluctuations at the end are higher, leading also to a sub-optimality compared to DP with BdS. Thanks to boundary surfaces, fluctuations

are lower and appear at the end of the simulation. This behavior appears because the solution lies on or close to the boundaries. Nonetheless the fluctuations remains in the feasible domain throughout the simulation. At the beginning in time of the boundary surface DP, the acceleration is much smoother because of the use of boundary surfaces during the backward part to improve DP accuracy.

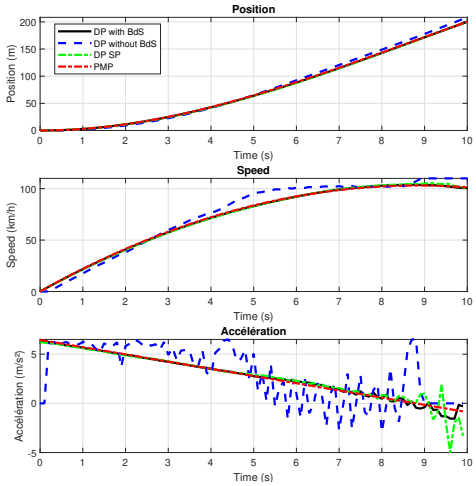

Fig. 4. Acceleration problem resolution with PMP, DP without BdS and DP with BdS

4.2 Example 2: Parallel Hybrid Electric Vehicle

The second model used to apply the methodology is a non convex, non linear model which corresponds to a parallel hybrid electric vehicle (De Jager et al., 2013). Here, the vehicle powertrain is composed of a battery, a motor and an irreversible power source (engine, turbine or other). The irreversible power source is assumed to give only the exact required power instantaneously. To calculate the required power from the powertrain to follow the cycle, Newton's second law is used. Fig. 5 shows a block scheme of the studied system. To propel the vehicle at a certain speed, the powertrain has to provide wheel power P_W. This power can be provided by the engine P_E and/or the motor P_M. To drive the motor power, the torque motor C_M is used. The battery State Of Charge (SOC_B) is the ratio of the actual capacity compared to the maximum capacity of the battery. Moreover, to regulate the motor temperature T_M with the coolant mass flow $\dot{m}_c$, the battery provides power P_P to the coolant system. The battery also provides power to the motor to ensure the motor power. The battery has to ensure the sum of these two powers P_K expressed in eq.(14).

$$\begin{cases} P_K = \eta_M . P_M \text{ if } P_M < 0 \\ P_K = \frac{P_M}{\eta_M} \text{ else} \\ \text{with: } \eta_M = \frac{C_M(t).\omega_M(t)}{C_M(t).\omega_M(t) + R.\left(\frac{C_M(t)}{k}\right)^2} \end{cases} \tag{14}$$

For more information on the modeling, see (Cottin et al., 2022). The 2 inputs 2 outputs system can be modeled as follows:

$$\boldsymbol{x} = \begin{pmatrix} SOC_B \\ T_M \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} C_M \\ \dot{m}_c \end{pmatrix} \tag{15}$$

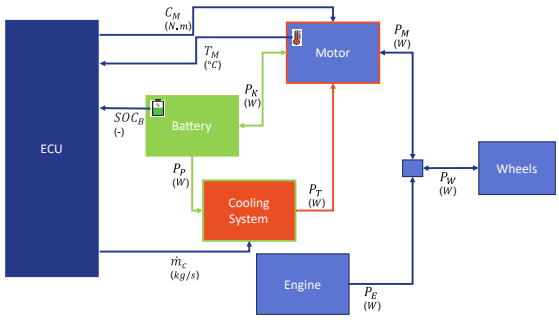

Fig. 5. Parallel Hybrid Electric Vehicle block scheme

with the following dynamics:

$$\dot{SOC}_B = (\theta \cdot \dot{m}_c(t) \cdot Cp_c \cdot (T_M(t) - T_c) + R \cdot \left(\frac{C_M(t)}{k}\right)^2 + C_M(t) \cdot \omega_M(t)) . \frac{-1}{Q \cdot (E + e \cdot SOC_B(t))} \tag{16}$$

$$\dot{T}_M = \frac{R \cdot \left(\frac{C_M(t)}{k}\right)^2 - \dot{m}_c(t) \cdot Cp_c \cdot (T_M(t) - T_c)}{m_m \cdot Cp_m} \tag{17}$$

Eq. (16) describes the battery State Of Charge SOC_B variation over time. It depends on the power spent to use the cooling system $\theta \cdot \dot{m}_c(t) \cdot Cp_c \cdot (T_M(t) - T_c)$, the motor losses $R \cdot (\frac{C_M(t)}{k})^2$ and the propulsion $C_M(t) \cdot \omega_M(t)$. The first power varies over time since it depends on the mass flow input $\dot{m}_c(t)$ and the motor temperature state $T_M(t)$. The cooling temperature T_c, thermal capacity Cp_c and the thermal to electric power coefficient θ are constants. Power losses are defined by the constant motor resistance R and its intensity which is function of the torque motor input $C_M(t)$. The required power to propel the vehicle is a function of the motor torque and speed$\omega_M(t)$. These two variables vary in time. In order to have an intensity, electrical powers are divided by the battery voltage which is modeled here by a constant voltage E plus a voltage function of the battery State Of Charge $e \cdot SOC_B(t)$. Thus this voltage is a function of time. Lastly, this intensity is divided by the maximum capacity of the battery Q.
Eq. (17) which captures the motor temperature dynamic over time, is a thermal power balance. It describes the dynamic function of the heating power, losses here, and the power removed by the cooling system.

In this optimal control problem, the objective is to minimize the overall vehicle energy consumption while managing the battery state of charge and the motor temperature. The trip to achieve here is the WLTC cycle. The variables' constraints are expressed in (18).

$$\begin{cases} 0 \leq SOC_B(t) \leq 1 \\ 20 \leq T_M(t) \leq 60 \\ -100 \leq C_M(t) \leq 100 \\ 0 \leq \dot{m}_c(t) \leq 0.05 \end{cases} \tag{18}$$

The instantaneous criterion is expressed in (19). The objective here is to ensure a trip repeatability while optimizing the motor temperature.

$$L(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \sqrt{P_E^2(\boldsymbol{u}(t)) + f(T_M) \cdot P_B^2(\boldsymbol{u}(t))} \quad \text{with } f(T_M) = aT_M^2 + bT_M + c \tag{19}$$

Coefficients a, b and c are representative of the electrical efficiency of the powertrain.

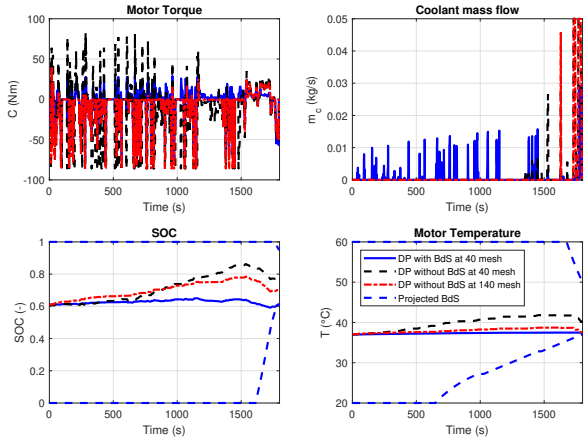

Fig. 6. Parallel hybrid electric vehicle optimal solution with boundary surface

Fig.6 presents the solutions given by DP without BdS with 100 mesh points for each variable in dashed red and with 20 mesh points for each variable in dashed black. DP with BdS is also shown with blue lines while the BdS projected in one state space and time is presented as a dashed blue line. The solution found by DP with BdS uses the mass flow coolant during all the trip compared to DPs without BdS. Thus the motor temperature is maintained at the right temperature with respect to the constraints. Moreover, the distribution of powertrain power seems to be more efficient to achieve the trip at the minimum acceptable final SOC_B. The results is a lower energy consumption since there is no unnecessary battery filling.

Even with the small number of meshes for the BdS, the management of the final constraints $SOC_B \geq 0.61$ and $T_M \geq 37$°C is much better. It can be seen that BdS is closer than classical DP to the minimum final constraint value, leading to a reduced sub-optimality. The results are shown in Table 2.

Methods	DP w/o BdS	DP w/o BdS	DP w/ BdS
Mesh points (-)	40	140	40
Cost (MJ)	11.1	8.3	6.8
$T_M(t_f)$ (°C)	40.2	37.5	37.4
$SOC_B(t_f)$ (-)	0.78	0.7	0.61
Calc. Time (s)	336	55 168	55 566

Table 2. Result comparison between DP without and with BdS

The above results show the final system states for three different simulations shown in Fig. 6. The computer used is an Intel CORE i7 processor at 3.6 GHz with 96 Go of RAM. It shows the utility of the BdS through the expected cost, which is the smallest one despite the allocation of resources. However, for the same mesh grid, the total calculation time is much higher because of the interpolation function while considering the boundary surface points. To achieve the same optimal result, BdS required less time than DP without BdS, making it a valuable option. On top of this, it might be possible to reduce this calculation time and thus obtain even greater benefit from the BdS.

5 Conclusion

This article presents a solution to extend the Boundary Lines method for one-state problems to a Boundary Surface method for two-states problems. Usage has been shown on two problems: one convex problem solved with PMP and DP and one non-convex problem solved only with DP with/without the BdS. The idea was to estimate the usefulness of the BdS for multi-state systems in the resolution of the optimal control problem.

For a simplified convex problem, a comparison between PMP and DP with/without BdS and DP with smooth penalty was done. It shows that for the same mesh grid, if the smooth penalty is well tuned, BDSP can satisfy final constraints with a lower cost than DP without BdS. However, BdS offers results much closer to the reference set by PMP than DP without BdS or with smooth penalty.

The boundary surface solution was then tested on a convex and a non-linear and non convex system to compare the results of DP with and without BdS. In this case, a comparison with PMP is no longer possible. Results show that the calculation time is much higher with BdS for the same mesh grid but that the gain in accuracy is very high. In such that it is possible to have a better solution than the finest mesh grid possible with an average computer for a reasonable calculation time.

Based on the way this BdS works, it is possible to use the proposed method on systems with more than 2 states. For systems with 3 states, the boundary will be edges of a volume and for more than 3 states, the feasible area will be an hypervolume. Since the calculation time with BdS can be high, future work will focus on reducing the time needed for the BdS method.

References

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6), 503–515.

Bertsekas, D.P. (2008). Approximate dynamic programming.

Cottin, W., Colin, G., Charlet, A., and Houillé, S. (2022). On the use of frequency analysis tools to minimize the calculation complexity of optimal control problems. *IFAC-PapersOnLine*, 55(16), 424–429.

De Jager, B., Van Keulen, T., and Kessels, J. (2013). *Optimal control of hybrid vehicles*. Springer.

Luus, R. (2019). *Iterative dynamic programming*. Chapman and Hall/CRC.

Pontryagin, L.S. (1987). *Mathematical theory of optimal processes*. CRC press.

Sundström, O., Ambühl, D., and Guzzella, L. (2010). On implementation of dynamic programming for optimal control problems with final state constraints. *Oil & Gas Science and Technology–Revue de l'Institut Français du Pétrole*, 65(1), 91–102.

van Schijndel, J., Donkers, M., Willems, F., and Heemels, W. (2014). Dynamic programming for integrated emission management in diesel engines. *IFAC Proceedings Volumes*, 47(3), 11860–11865.