Determining vehicle acceleration from noisy non-uniformly sampled speed data for control purposes

Edwin Solano-Araque, Guillaume Colin, Guy-Michel Cloarec, Ahmed Ketfi-Cherif, Yann Chamaillard

Preprints, 9th IFAC International Symposium on Advances in Automotive Control
Orléans, France, June 23-27, 2019

Determining vehicle acceleration from noisy non-uniformly sampled speed data for control purposes

Edwin Solano-Araque [*,**] **Guillaume Colin** [**]
Guy-Michel Cloarec [*] **Ahmed Ketfi-Cherif** [*]
Yann Chamaillard [**]

[*] *Renault S.A.S, France (e-mail: edwin.e.solano-araque@renault.com; guy-michel.cloarec@renault.com; ahmed.ketfi-cherif@renault.com)*
[**] *Univ. Orléans, PRISME, EA 4229, F45072, Orléans, France (e-mail: guillaume.colin@univ-orleans.fr; yann.chamaillard@univ-orleans.fr)*

Abstract: Vehicle acceleration is an important variable for many automotive control applications. In this paper, we present an approach to estimate vehicle acceleration from vehicle speed data coming from the CAN (Controller Area Network) bus. The proposed method, which can be seen as an extension of the Savitzky-Golay filter to non-uniformly sampled signals, re-samples them to a constant period while filtering noise coming from different sources and also provides a proper estimation of vehicle acceleration. We also consider the frequency response of the filtering effect of the method. Finally, some practical considerations for efficient implementation of the algorithm are given.

Keywords: Vehicle acceleration, Controller Area Network, Non-uniformly sampled signal processing, Local Polynomial Regression, Savitzky-Golay Filter, Computational complexity

1. INTRODUCTION

Vehicle acceleration has a major influence on many automotive applications. For example, it is a determining factor in drivability (Bellem et al. (2016)) and also in the energy consumption of electric vehicles (Mruzek et al. (2016), Solano-Araque et al. (2018)). Therefore, acquiring a vehicle acceleration signal from real-driving field tests is of great interest for control design.

These data can be obtained by installing acceleration sensors on the vehicle and obtaining this information directly. However, in some cases, this type of information may not be available; for example, when one wants to estimate the acceleration of the preceding vehicle for an automatic driving function, or when one must work in a given ECU of a car which has no access to the accelerometers but it has access to the vehicle speed. Another possibility is obtaining the vehicle speed signal, which is usually already present in the vehicle, and numerically deriving it in order to get an estimation of the acceleration. Nevertheless, given the fact that the vehicle's speed signal is usually very noisy, it can be hard to obtain an exploitable acceleration estimation from it.

In this work, we will consider the problem of getting an adequate acceleration estimation from the speed signal available on the CAN (Controller Area Network) bus of a commercial vehicle. This speed can correspond either to the controlled vehicle or the predecessor vehicle.

The CAN bus is widely used for automotive applications. This is due to its deterministic resolution of the contention (i.e. for a given information to be transmitted, the message to be sent is deterministically defined), low cost, and simple implementation (Di Natale et al. (2012)). Today, most all (if not all) of the vehicles produced in Europe are equipped with at least one CAN bus (Davis et al. (2007)).

However, CAN characteristics lead to some phenomena that can pollute the signal being sent. One effect of this pollution is the fact that the sample time can vary slightly between two transmissions or, worse, some messages can be lost during transmission. This results in CAN signals not being uniformly sampled. This can cause problems when one wants to estimate the derivative of such a signal (e.g. the derivative of a speed signal).

In this paper, we propose a technique allowing us to get an exploitable estimation of vehicle acceleration, suitable to be used for control design. This technique is based on Local Polynomial Regression and can also be seen as an extension of the Savitzky-Golay filter (Schafer (2011)) to non-uniformly sampled signals.

2. SIGNAL DISTORTIONS DUE TO CAN PROTOCOL

As already mentioned, the CAN protocol has some characteristics that may introduce noise in the signals. The main ones are :

- **Priority based arbitration:** The CAN arbitration protocol is priority-based, specifically it uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to determine access (Davis et al. (2007)). This means that if two nodes are trying to send a frame (sequence of bits) at the same time, the one

with the highest priority (with a numerically lower identifier) will circulate and the other one has to wait for the next opportunity to be transmitted. Hence, it may introduce a non-deterministic delay in the signals being sent.
- **Bit stuffing:** in order to avoid ambiguities in the frames being sent (Davis et al. (2007)), to allow nodes to synchronize their timing and to allow error detection (Watterson (2012)), some bits can be added to the frames. This operation affects the length of the frames and, therefore, it may introduce a slight delay in the transmission.
- **Limited signal resolution:** the maximum data length for a CAN frame is 8 bytes (Di Natale et al. (2012)). This means that the maximum resolution of a signal being transmitted through a single frame is 64 bits. And in practice, the resolution is lower due to the fact that automakers usually send many signals packaged in one frame.
- **Loss of messages:** On the CAN protocol there is no guarantee that the message that a node sends will be received by all of the others. Some messages may not be received by one or many of the concerned nodes. This would mean a loss of points in the transmitted signal.

3. METHOD FOR FILTERING AND DERIVING CAN SIGNALS

The method presented in this section makes it possible simultaneously to:

- re-sample CAN signals to a constant period, thus enabling classical signal processing techniques to be applied on them.
- reconstruct missing signal values during the transmission, thus compensating for messages losses.
- filter the signal to eliminate noise coming from different sources; the technique we propose performs particularly well on signals polluted with Gaussian noise.
- estimate the signal derivative, while compensating for the effect of noise and of loss of resolution due to CAN transmission.

Hence, the method we propose allows a filtered uniformly sampled signal and a proper estimation of its derivative to be obtained from a noisy non-uniformly sampled signal.

The method consists of several stages, in particular: 1) defining the output signal time vector and "creating" time-windows, 2) getting the signal points associated to each window, 3) (local) polynomial approximation of window points, 4) reconstructing the point at $t = t^{end}[k]$ and estimating the signal derivative at this point.

Each of these phases is now presented in turn.

3.1 Output signal definition and windows consideration

First, it is necessary to define the time instants $\underline{\mathbf{t^{end}}} \in \mathbb{R}^{N_{sign_1}}$, for which we want to reconstruct the raw signal, whose time vector is $\underline{\mathbf{t^{ini}}} \in \mathbb{R}^{N_{sign_0}}$. In order to get a uniformly-sampled signal, the output-signal time points must be uniformly distributed, with a sampling time τ_e. We will consider that the signal starts at $t_1^{ini} = t_0$ and it ends at $t_{N_{sign0}}^{ini} = t_f$.

Once the vector $\underline{\mathbf{t^{end}}}$ has been generated, we can consider the time-windows. First, we consider time-windows centered on each one of the points t_j^{end} and with a width τ_{win}. Such an algorithm can be used to obtain information from an experimental setting *a posteriori*.

We will introduce the notation we are using. win_j is the window corresponding to the i-th value of t_j^{end}. win_j starts at $t_{win_0}^j$ and ends at $t_{win_f}^j$; using equations 1 and 2 these values are calculated as a function of t_j^{end} and τ_{win}, for a centered window.

$$t_{win_0}^j = t_j^{end} - \tau_{win}/2, \tag{1}$$

$$t_{win_f}^j = t_j^{end} + \tau_{win}/2,. \tag{2}$$

In order to promote that each window has enough points to apply the method, we also introduce the following constraints:

$$t_1^{end} \geqslant t_0 + \frac{\tau_{win}}{2}, \tag{3}$$

$$t_{N_{sign1}}^{end} \leqslant t_f - \frac{\tau_{win}}{2}. \tag{4}$$

The following condition ensures a uniform sampling for the output signals:

$$t_{j+1}^{end} - t_j^{end} = \tau_e, \quad \forall j \in [1, ..., N_{sign_1} - 1]. \tag{5}$$

By respecting these conditions, we can get an appropriate vector $\underline{\mathbf{t^{end}}}$. For this paper, we have taken $t_1^{end} = t_0 + \frac{\tau_{win}}{2}$ and $t_{N_{sign1}}^{end} \leqslant t_f - \frac{\tau_{win}}{2}$. This is represented in Figure 1.

3.2 Obtaining the signal points associated to each window

From this point on, we are only working on the j-th window, win_j. Each of the following stages can be applied to each of the windows.

At win_j, we will get all the points between ${t_{win_0}}^j \leqslant t \leqslant {t_{win_f}}^j$, each of which will be denoted ${x_k}^j, k = 1, ..., m_j$. We will note $\underline{\mathbf{x^j}}$ the vector with all the points belonging to win_j, as presented in (6).

$$\begin{aligned} \underline{\mathbf{x^j}} = [{x_1}^j(t_1), \ldots, {x_k}^j(t_k), \ldots, {x_{m_j}}^j(t_{m_j})]^T \\ \mid \quad \exists {x_k}^j(t_k) \in \underline{\mathbf{x^j}}, \quad \forall t_k \in T_j \\ T_j = \{t \mid \quad t \in \underline{\mathbf{t^{ini}}}, \quad {t_{win_0}}^j \leqslant t \leqslant {t_{win_f}}^j\} \end{aligned} \tag{6}$$

3.3 (Local) Polynomial approximation of the points within the window

Once we have gotten $\underline{\mathbf{x^j}}$, it is possible to define a function $f(t)$ to approximate the values of the points ${x_k}^j({t_k}^j)$. For the method we propose we use a polynomial function, $f_{n_p}(t)$, as presented in (7), where n_p is the order of the polynomial, $\underline{\mathbf{P_{n_p}}(t)} = [1, t, t^2, \ldots, t^{n_p}]^T$ and $\underline{\theta} = [\theta_0, \theta_1, \ldots \theta_{n_p}]^T$.

$$f(t) = \sum_{i=0}^{n_p} \theta_i . t^i = \underline{\mathbf{P_{n_p}}(t)}^T \underline{\theta} \tag{7}$$

In order to simplify the notation, from now on, we will omit the super-index j from most of the equations. $f(\underline{\theta}, t)$ is a linear function w.r.t. $\underline{\theta}$, thus we can consider $f(t)$

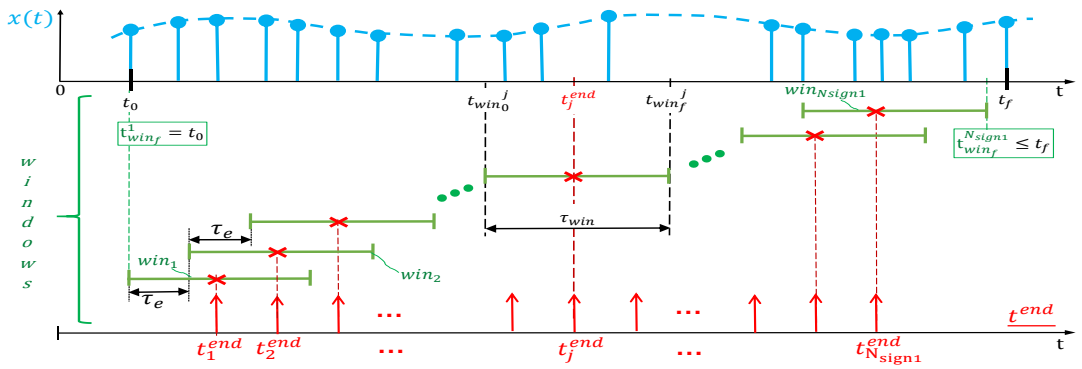

Fig. 1. Time-windows for the proposed method

as a *linear measurement model* (Boyd and Vandenberghe (2004)). Hence, by considering ν_k as the error between the measurement x_k and the estimation $f(t_k)$, where $t_k \in \underline{\mathbf{t}}$, we get:

$$x_k(t_k) = \underline{\mathbf{P_{n_p}}(\mathrm{t_k})}^T . \underline{\theta} + \nu_k \tag{8}$$

By considering all the m points in $\underline{\mathbf{x}}$, we find (9), where $\underline{\mathbf{x}} \in \mathbb{R}^m$ are the points within the window, $T_p = [\underline{\mathbf{P_{n_p}}(\mathrm{t_1})}, \underline{\mathbf{P_{n_p}}(\mathrm{t_2})}, \ldots, \underline{\mathbf{P_{n_p}}(\mathrm{t_m})}]^T \in \mathbb{R}^{m \times n_p}$ is a matrix where each row corresponds to the $0^{th} - {n_p}^{th}$ powers of t_k, $\underline{\theta} \in \mathbb{R}^{n_p}$ is a vector with the parameters of $f(t)$, that we want to estimate, and $\underline{\nu} \in \mathbb{R}^{m_j}$ is the estimation error for each point. We will assume that $m_j \geqslant n_p$.

$$\underline{\mathbf{x}} = T_p . \underline{\theta} + \underline{\nu} \tag{9}$$

By considering that a function of the form $f(\underline{\theta}^\star, t)$, such as the one given in 7, can "perfectly" (i.e. goodly enough for the application purposes) represent the signal behavior within the considered window, and that the difference between the measurements and the estimations corresponds to Gaussian noise (i.e. ν follows a Gaussian distribution), finding the parameters that best represent the points within the window becomes a least-squares problem (Boyd and Vandenberghe (2004)). Therefore, the optimal value for the parameters of $f(t)$ for the window, $\underline{\theta}^\star$, are given by (10).

$$\underline{\theta}^\star = \left(T_p^T . T_p\right)^{-1} . T_p^T . \underline{\mathbf{x}} \tag{10}$$

3.4 Signal Reconstruction at $t = t_j^{end}$ and signal derivative estimation

To find the filtered signal value associated to win_j, we need to find the value of $f(\underline{\theta}^\star, t)$ at $t = t_j^{end}$, this value will be associated to the output signal. This operation is presented in Figure 2.

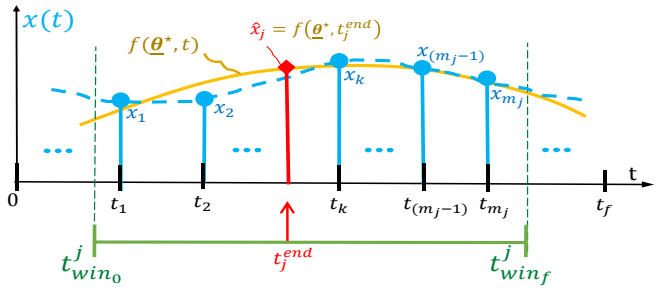

Fig. 2. Output signal reconstruction

We can find an estimate of the signal derivative at $t = t_j^{end}$ by applying (11):

$$\frac{dx}{dt}(t_j^{end}) = [0, 1, 2, \ldots, n_p] \,\mathrm{diag}\,(\underline{\theta}^\star) \begin{bmatrix} 0 \\ (t_j^{end})^0 \\ (t_j^{end})^1 \\ \vdots \\ (t_j^{end})^{n_p-1} \end{bmatrix} \tag{11}$$

Once we have done so, we only have to apply the method to each of the windows to get a filtered signal and an estimate of its derivative.

3.5 Causal implementation of the method: "left-oriented" window

The method we have just presented, based on a centered window, is non-causal, which means that it is not possible to use it for a real-time implementation. However, one can easily get a causal version of this algorithm by using a window that only considers past values of the signal; it is equivalent to saying that $t_j^{end} = t_j^{win_f}$. We may call such a window a *left-oriented window*. In that case, the values of $t_{win_0}^j$ and $t_{win_f}^j$ can be calculated by applying equation (12) and the condition $t_j^{win_f} = t_j^{end}$.

$$t_{win_0}^j = t_j^{end} - \tau_{win}. \tag{12}$$

The conditions to promote having enough points to apply the method will become:

$$t_1^{end} \geqslant t_0 + \tau_{win}, \tag{13}$$

$$t_{N_{sign1}}^{end} \leqslant t_f. \tag{14}$$

When one directly applies the polynomial approximation presented in section 3.3, the frequency response of the filter that one can get would not be acceptable for an application. This is due to the fact that a polynomial regression tends to be closer to the central values of the signal and yields greater errors for the corners of the approximation. However, a way to compensate for this fact is to weight the signal points by the absolute value of the difference between t_j^{end} , and each point time, t_k^{ini}. It is equivalent to minimizing the value of $\underline{\nu}^T W_1 \underline{\nu}$, where $W_1 = \mathrm{diag}\,(\underline{\mathbf{w}})$, $w_k^1 = |t_j^{end} - t_k^{ini}|$, which is a weighted least-squares problem.

In order to prioritize even more the points that are closer to the time at which one would like to reconstruct the signal,

one can generalize the previous method by considering the magnitude of the l-th power of each gap. It is expressed by equation (15), which is a variant without loss of generality. The fact of dividing all the numbers by the l-th power of the maximum value of $|t_j^{end} - t_k^{ini}|$ permits to normalize the values of w_k to 1, in order to avoid numerical problems.

$$W_l = \text{diag}\left(\underline{\mathbf{w}}\right), \quad w_k^l = \frac{\left|t_j^{end} - t_k^{ini}\right|^l}{\left(\max\limits_{r=1,\ldots,m_j} \left|t_j^{end} - t_r^{ini}\right|\right)^l}. \quad (15)$$

The solution of the weighted least-squares problem is given by:

$$\underline{\theta}^{\star} = \left(T_p^T . W_l . T_p\right)^{-1} . T_p^T . W_l . \underline{\mathbf{x}} \quad (16)$$

4. FREQUENCY RESPONSE OF THE FILTERING AND CHOICE OF THE METHOD PARAMETERS

An important characteristic of a filter is its frequency response. In this section we present the frequency response of the proposed methods, which can be viewed as interpolation filters, obtained numerically by applying it to sinusoidal signals of different frequencies and comparing the amplitude of the input and the output signals in order to get the gain. It is important to note that, given the fact that the proposed methods are not necessarily linear filters, it is possible that this representation does not give a comprehensive representation of the filter behavior.

In Figure 3, we compare the frequency response of both the proposed non-causal and causal methods with the response of a classic technique, namely a Butterworth filter. The parameters used for the method are: $\tau_{win} = 200ms$ and $n_p = 1$ for the non-causal method, and $\tau_{win} = 850ms$, $n_p = 1$ and $l = 2$, which lead quite close to a cutoff frequency at 6dB of 2.997Hz (i.e. the frequency at which the output signal amplitude is attenuated by 50.12% w.r.t. the input signal). The Buttherworth filter has been designed such as to get the same cutoff frequency.

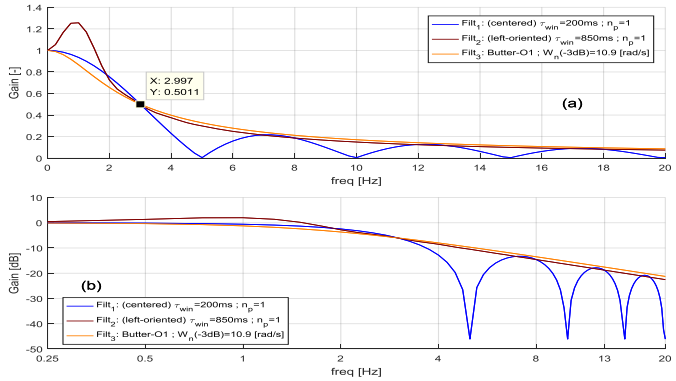

Fig. 3. Frequency response of the proposed method and of a linear interpolation filter; (a) linear-scale, (b) logarithmic scale

We can observe that the non-causal method preserves more of the signal energy below the cutoff frequency and, at the same time, it attenuates the signal better for frequencies above the cutoff frequency. Concerning the causal method, we can see that for a certain range of frequencies there is a gain greater than 1, which means that those frequencies will be amplified by the filter. Nonetheless, the filter preserves (or amplifies) a great amount of signal energy in the range of interest, while better attenuating the noise on higher frequencies.

5. PRACTICAL CONSIDERATIONS FOR OFFLINE ALGORITHM IMPLEMENTATION

When implementing the proposed method, the computational burden can be a problem. Probably, the main way of reducing it is by improving the way one solves the least-squares problem for the polynomial approximation. However, another interesting way of improving the complexity is to reduce the burden associated to getting the points within each one of the time-windows. This is particularly true for long signals because, as is shown below, this is a task whose complexity scales quite rapidly as the input or output size increases. In this section, we present an appropriate algorithm to recover the points within the window while limiting the computational burden.

First we will consider a naive algorithm to get the point within the windows, presented in Algorithm 1. It will take as an input the original signal time vector, $\underline{\mathbf{t}}$, the number of inputs of the output signal, m, the sampling time of the output signal τ_e and the size of the windows, τ_{win}. The algorithm returns, for each window, a vector containing the indexes of the points within the window.

Algorithm 1 Determining the window points - naive version

Input:
Original signal : $\underline{\mathbf{t}} \in \mathbb{R}^n$;
Number of windows (output signal size) : m ;
Method parameters : τ_e , τ_{win}

Output:
m vectors $\underline{\mathbf{v}}^j$ with the indexes of $\underline{\mathbf{t}}$ belonging to the window j

```
1: begin
       Create m void vectors v^j = ∅ ,   j = 1, ..., m
2:     for each value of the signal, i, do
3:         for time window, j, do
4:             if t[i] is contained by the window j then
5:                 Append the value i at the end of v^j
6:             end if
7:         end for
8:     end for
9: end
```

In order to analyze the complexity of the naive algorithm, we will use the pseudo-code of Algorithm 2, for which some programming choices have been done, including the use of dynamic memory allocation. The blue comments show the computational complexity of each task. To calculate the computational burden of a loop, one must add the cost of every task within the loop and, then, multiply it by the number of iterations in the loop, in "big $\mathcal{O}$" notation.

For the naive algorithm, the main complexity is associated to the two nested loops, whose complexity is $\mathcal{O}(n.m)$, which means that the computational burden increases linearly with the input and the output signal sizes n and m, respectively. Given the fact that for a constant sampling time τ_e, n and m are closely related, one can roughly consider it as a quadratic growth.

Concerning the memory cost of the algorithm, the algorithm requires generating m vectors, $\underline{\mathbf{v}}^j$, with a mean size $\tilde{v}$, roughly constant (in fact, $\tilde{v}$ depends on different parameters of the method, and it is indirectly associated

Algorithm 2 Determining the window points - naive version pseudo-code

Input:
Original signal : $\underline{\mathbf{t}} \in \mathbb{R}^n$;
Number of windows (output signal size) : m ;
Method parameters : τ_e , τ_{win}
Output:
m vectors $\underline{\mathbf{v}}^j$ with the indexes of $\underline{\mathbf{t}}$ belonging to the window j

1: **begin**
2: Create m void vectors $\underline{\mathbf{v}}^j = \emptyset\,, \quad j = 1, ..., m$ ▷$\mathcal{O}(m)$
3: $t_{win0} = 0$; $t_{winf} = 0$ ▷$\mathcal{O}(1)$
4: **for** $i := 1 : n$ **do** ▷*loop:* $\mathcal{O}(n)$
5: **for** $j := 1 : m$ **do** ▷*loop:* $\mathcal{O}(m)$
6: $t_{win0} := (j-1) * \tau_e$ ▷$\mathcal{O}(1)$
7: $t_{winf} := (j-1) * \tau_e + \tau_{win}$ ▷$\mathcal{O}(1)$
8: **if** $\underline{\mathbf{t}}[i] \geqslant t_{win0}$ **and** $\underline{\mathbf{t}}[i] \leqslant t_{winf}$ **then** ▷$\mathcal{O}(1)$
9: Append i at the end of $\underline{\mathbf{v}}^j$ ▷$\mathcal{O}(1)$ *(big)*
10: **end if**
11: **end for**
12: **end for**
13: **end**

to m, but here we are neglecting this dependency). This represents a memory cost $\mathcal{O}(\tilde{v}.m) = \mathcal{O}(m)$.

The improved algorithm we propose is presented in Algorithm 3. It takes the same inputs as the naive algorithm, but it only returns two vectors of size m: $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}}$ and $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}$, which respectively contain the initial and the final indexes of the points associated to each one of the windows.

Algorithm 3 Determining the window points - proposed algorithm

Input:
Original signal: $\underline{\mathbf{t}} \in \mathbb{R}^n$;
Number of windows (output signal size): m ;
Method parameters: τ_e , τ_{win}
Output:
2 vectors with the initial and final indexes for each window, respectively $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}}$ and $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}$

1: **begin**
2: Create and initialize the auxiliary scalars I_{ini}, I_{end} I_{ini_1}, t_{sign}, t_{win0}, t_{win0}, t_{winf}
3: Create and initialize the vectors $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}}$ and $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}$
4: **for** each signal value, i (starting at $i = 2$) **do**
5: Save I_{ini}'s last value in I_{ini_1}
6: Find the first window I_{ini} whose $t^{I_{ini}}_{win_0} \leqslant t[i]$
7: Find the first window I_{end} whose $t^{I_{end}}_{win_f} > t[i]$
8: **for** k **from** $Idx_{ini\_1} + 1$ **to** Idx_{ini} **do**
9: **if** { $t[i] \leqslant t_{win^k_f}$ and a point belonging to the window have not been found yet } **then**
10: i is the first index of the window k i.e. $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}}[k] := i$
11: **end if**
12: **end for**
13: **for** k **from** I_{end} **to** I_{ini} **do**
14: i is the last-until-now valid index of the signal within the window k i.e. $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}[k] := i$
▷ note: if in future iterations another index greater than i is found to belong to the window, $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}[k]$ will be overwritten
15: **end for**
16: **end for**
17: **end**

In order to analyze the complexity of the proposed algorithm, we will use the pseudo-code of Algorithm 4, for which also some programming choices have been done; however, this time we have chosen to use static memory allocation, as we know the output size *a priori*.

Algorithm 4 Determining the window points - proposed algorithm pseudo-code

Input:
Original signal: $\underline{\mathbf{t}} \in \mathbb{R}^n$;
Number of windows (output signal size): m ;
Method parameters: τ_e , τ_{win}
Output:
2 vectors with the initial and final indexes for each window, respectively $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}}$ and $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}}$

1: **begin**
2: $I_{ini} = 1$; $I_{end} = 1$; $I_{ini\_1} = 0$ ▷$\mathcal{O}(1)$
3: $t_{sign} = 0$; $t_{win0} = 0$; $t_{winf} = 0$ ▷$\mathcal{O}(1)$
4: $\underline{\mathbf{v}}^{\mathbf{ini}}_{\mathbf{idx}} = [1\,;\, \underline{\mathbf{0}}_{m-1}]$; $\underline{\mathbf{v}}^{\mathbf{end}}_{\mathbf{idx}} = \underline{\mathbf{0}}_m$ ▷$\mathcal{O}(1)$
5: **for** $i := 2 : n$ **do** ▷*loop:* $\mathcal{O}(n)$
6: $t_{sign} := \underline{\mathbf{t}}[i]$ ▷$\mathcal{O}(1)$
7: $I_{ini\_1} := I_{ini}$ ▷$\mathcal{O}(1)$
8: **while** { $(t_{sign} \geqslant Idx_{ini} * \tau_e)$ **and** $(t_{sign} < m * \tau_e)$ } **do** ▷*loop:* $\mathcal{O}(\widetilde{m}_1)$
9: $I_{ini} := I_{ini} + 1$ ▷$\mathcal{O}(1)$
10: **end while**
11: **while** { $\left(t_{sign} > (I_{fin} - 1) * \tau_e + \tau_{win}\right)$ **and** $(I_{end} \leqslant I_{ini})$ } **do** ▷*loop:* $\mathcal{O}(\widetilde{m}_2)$
12: $I_{fin} := I_{end} + 1$ ▷$\mathcal{O}(1)$
13: **end while**
14: **for** $k := (I_{ini\_1} + 1) : I_{ini}$ **do** ▷*loop:* $\mathcal{O}(\widetilde{m}_1)$
15: **if** { $t_{sign} \leqslant (k-1) * \tau_e + \tau_{win}$ **and** $V_{idx\,ini}[k] == 0$ } **then** ▷$\mathcal{O}(1)$
16: $V_{idx\,ini}[k] := i$ ▷$\mathcal{O}(1)$
17: **end if**
18: **end for**
19: **for** $k := (I_{end}) : I_{ini}$ **do** ▷*loop:* $\mathcal{O}(\widetilde{m}_2)$
20: $V_{idx\,end}[k] := i$ ▷$\mathcal{O}(1)$
21: **end for**
22: **end for**
23: **end**

As we can see on Algorithm 4, after the initialization phase there is a loop containing four independent internal loops. Concerning the computational burden, the initialization tasks (lines 2-4) can be executed in constant time $\mathcal{O}(1)$; the external loop is executed $\mathcal{O}(n)$ times, and each one of the internal loops is executed in a mean time $\mathcal{O}(\widetilde{m}_l), l = 1, 2$, where $\widetilde{m}_l$ does not depend directly on the number of windows m. Therefore, by adding these results, we will get an amortized computational complexity $\mathcal{O}(n(2\widetilde{m}_1 + 2\widetilde{m}_2) + 1) = \mathcal{O}(n(\widetilde{m}_1 + \widetilde{m}_2))$, with $(\widetilde{m}_1 + \widetilde{m}_2) \ll m$ when m is big.

Concerning the memory cost of the algorithm, we know exactly the amount of memory that the algorithm requires. We need six scalar variables (Idx_{ini}, Idx_{end}, Idx_{ini_1}, t_{sign}, t_{win0}, t_{winf}) and two m-sized vectors ($V_{idx\,ini}$ et $V_{idx\,end}$). Hence, the memory cost associated to the algorithm is $\mathcal{O}(2m + 6) = \mathcal{O}(m)$.

Table. 1 presents a synthesis of the complexity results detailed above.

Algorithm	Time	Memory
Naive	$\mathcal{O}(n.m)$	$\approx \mathcal{O}(m)$ (dynamic)
Proposed	$\approx \mathcal{O}(n)$	$\mathcal{O}(m)$ (static)

Table 1. Comparison of naive and proposed algorithms in time and memory complexity

6. ESTIMATION OF VEHICLE ACCELERATION

Finally, we applied *a posteriori* the proposed method to the vehicle speed signal obtained through sensors installed on each of the wheels, and transmitted by a CAN bus

Figure 4 presents the raw and processed signals of vehicle speed and acceleration. For the acceleration estimate, the following estimation methods were compared: applying Euler's method to the raw signal, applying Euler's method to the speed signal processed by a Buttherworth O1 filter, filtering with a Buttherworth filter the derivative of the raw signal, applying the proposed method with a centered window (non-causal method) and applying it with a left-oriented window (causal method). In order to fairly compare the methods, we have use the same parameters of the filters shown in Figure 3. In order to apply the linear filters, we have linearly interpolated the signals in order to get an uniformly-sampled signal (which introduces a distortion in the signal as it behaves as a low-pass filter).

We can notice that all the considered methods produce a much less noisy signal than the estimation based on the raw signal. However, the linear method introduces a delay in the speed signal.

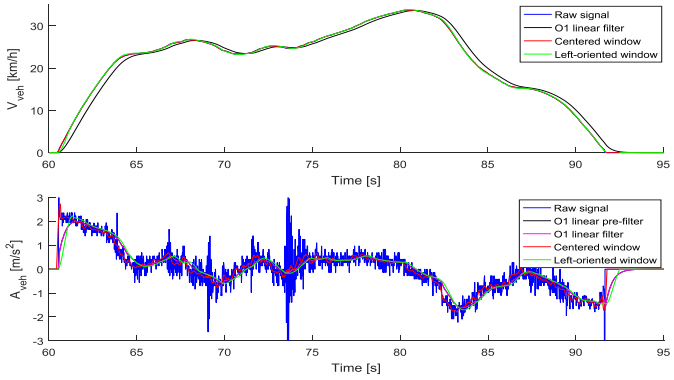

Fig. 4. Vehicle Acceleration Estimation

Figure 5 presents a zoomed view of the velocity and accelerations. We can see that there is a real improvement in the acceleration due to the processing. However, all the causal methods (the proposed causal method and the linear ones) introduce a delay in the estimation; however, by using the left-oriented window, one gets a less noisy signal. Nonetheless, the proposed causal method also amplifies the signal values at some points.

Also, there is a pronounced jump in the raw estimations at the beginning of the vehicle maneuver. This is due to the fact that commercial vehicle speed sensors are usually not able to produce a reliable estimation at low speed. Therefore, the speed signal has a value 0 when the vehicle speed is below a certain level. This produces a jump in the signal which has quite a strong effect on the acceleration estimation. We can notice that, when the non-causal proposed method is applied, this phenomenon practically disappears without introducing a delay in the signal.

7. CONCLUSIONS AND DISCUSSION

The proposed non-causal method provides an adequate estimate of vehicle acceleration without having an accelerometer, which could be used for many *a posteriori* applications. However, it is hard to say if the causal proposed method really improves the acceleration estimation; it could be the case when one needs to get a non-noisy rough estimate of the acceleration.

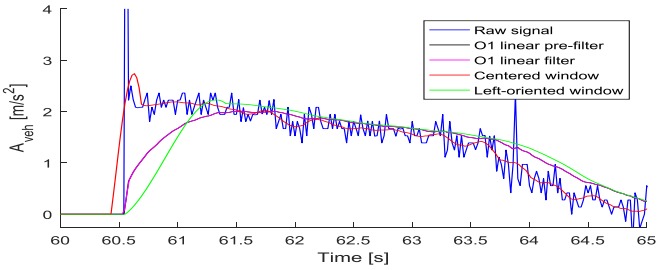

Fig. 5. Vehicle Acceleration Estimation - Zoomed view

Also, an efficient way to implement a crucial step of the method has been proposed. This method could also be applied to some other problems where finding whether some given points belong to an equally spaced set of overlapping intervals is required.

In this work, we have considered a speed signal based on wheel sensors. Another possibility that could be applied to single-gear electric vehicles would be to estimate acceleration based on the electric machine speed; this is a more noisy signal but, at the same time, it does not have the problem of a low dead-zone and it could have a better resolution. The proposed method could properly filter such a signal in order to better exploit the information to get a relevant acceleration estimate.

REFERENCES

Bellem, H., Schönenberg, T., Krems, J.F., and Schrauf, M. (2016). Objective metrics of comfort: developing a driving style for highly automated vehicles. *Transportation research part F: traffic psychology and behaviour*, 41, 45–54.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press, New York.

Davis, R.I., Burns, A., Bril, R.J., and Lukkien, J.J. (2007). Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3), 239–272.

Di Natale, M., Zeng, H., Giusto, P., and Ghosal, A. (2012). *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media.

Mruzek, M., Gajdáč, I., Kučera, L., and Barta, D. (2016). Analysis of parameters influencing electric vehicle range. *Procedia Engineering*, 134, 165–174.

Schafer, R.W. (2011). What is a savitzky-golay filter?[lecture notes]. *IEEE Signal processing magazine*, 28(4), 111–117.

Solano-Araque, E., Colin, G., Cloarec, G.M., Ketfi-Cherif, A., and Chamaillard, Y. (2018). Energy analysis of eco-driving maneuvers on electric vehicles. *IFAC-PapersOnLine*, 51(31), 195–200.

Watterson, C. (2012). Controller area network (can) implementation guide. *Appl. Note-1123 Analog Devices Inc.*