1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems

Géry Casiez, Nicolas Roussel, Daniel Vogel

1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems

Géry Casiez[1,2,3]**, Nicolas Roussel**[3] **& Daniel Vogel**[4]

[1]LIFL, [2]University of Lille & [3]Inria Lille, France
[4]Cheriton School of Computer Science, University of Waterloo, Canada
gery.casiez@lifl.fr, nicolas.roussel@inria.fr, dvogel@uwaterloo.ca

ABSTRACT

The 1€ filter ("one Euro filter") is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1€ filter has less lag using a reference amount of jitter reduction.

ACM Classification Keywords

H.5.2 [Information interfaces and presentation]: User interfaces - Input devices and strategies.

General Terms

Algorithms, Performance, Human Factors

Author Keywords

Signal; noise; jitter; precision; lag; responsiveness; filtering

INTRODUCTION

Noisy signals occur when an original time varying value undergoes undesirable and unpredictable perturbations. These may be caused by things like heat and magnetic fields affecting hardware circuitry, the limits of sensor resolution, or even unstable numerical computation. Noisy signals are a common problem when tracking human motion, particularly with custom sensing hardware and inexpensive input devices like the Kinect or Wiimote. In addition, even signals from established high-end sensing systems can become noisy when interaction techniques use large scaling effects. A common example is using a Vicon tracking system to implement ray casting with a wall display [6]: calibration problems and hand tremor add further perturbations to the ones amplified by the pointing technique.

Noise affects the quality of a signal in two primary ways [9]. It can reduce *accuracy*, by adding an *offset* between the observed values and the true ones. More often, it reduces *precision*, where repeated observations of values exhibit *jitter* – many different values are observed for a single true one. Jitter has a large effect on the way people perceive and act. For example noisy values are harder to read and unstable cursors hinder target acquisition [3, 7, 5]. One usually wants to filter noisy signals to reduce, and possibly remove, the unwanted parts of the signal. However, filtering inherently introduces time latency – commonly called *lag* – which reduces system *responsiveness*. Lag may not be an issue in domains like artificial perception and decision making, but with interactive feedback, it is very important. In fact, it is the combination of precision and responsiveness that are crucial: people can point accurately in spite of an offset, but only with minimal lag and jitter. The difficulty is that implementing and tuning a filter to minimize both jitter and lag is challenging, especially with little or no background in signal processing.

In this paper we describe the 1€ filter ("one Euro filter"), a tool to improve noisy signal quality with a tunable jitter and lag balance. It uses a low-pass filter, but the cutoff frequency changes according to speed: at low speeds, a low cutoff reduces jitter at the expense of lag, but at high speeds, the cutoff is increased to reduce lag rather than jitter. The intuition is that people are very sensitive to jitter and not latency when moving slowly, but as movement speed increases, people become very sensitive to latency and not jitter. We compare the 1€ filter to alternative techniques and show how it can reduce that same amount of jitter with less lag. It is also efficient and easy to understand, implement, and tune: the algorithm can be expressed in a few lines; it uses only basic arithmetic; and it has only two independent parameters that relate directly to jitter and lag. Other researchers and ourselves have already used variations of it in many projects. In fact, the "dynamic recursive low-pass filter" used by the third author in [6] established the basic principle, but it required four parameters and a fixed sample rate. The '1€' name is an homage to the $1 recognizer [10]: we believe that the 1€ filter can make filtering input signals simpler and better, much like the $1 recognizer did for gestures.

After a review of the jitter, lag, and alternative filtering techniques, we describe the 1€ filter in detail with an implementation, discuss tuning with different applications, and conclude with an illustrative comparison.

JITTER, LAG, AND FILTERING

Several studies show jitter and lag have a negative impact on performance. MacKenzie et al. found mouse movement times increased 16% with 75 ms lag, and up to 64% with 225

CHI'12, May 5–10, 2012, Austin, Texas, USA.

ms lag [3]. With 3D hand tracking, Ware and Balakrishnan found that only 50 ms lag reduced performance by more than 8% [7]. Pavlovych and Stuerzlinger found no performance degradation below 58 ms lag using a mouse or Wiimote, but increasing jitter from 4 to 8 pixels doubled error rates for small targets [5]. Assuming a 100 PPI screen, 4 pixels corresponds to 1mm of jitter mean-to-peak: close to the 0.4 mm of jitter they found with the established Optitrack system.

Although the precision of an input device may be very good, it does not take into account scaling effects introduced by interaction techniques. Device input is often scaled up, so people can cover more display distance with less device movement. For example, default operating system mouse transfer functions can be scaled up $12\times$ [1] and factors as high as $90\times$ have been used when ray casting on wall sized displays [6]. Regardless of native device precision, scaling amplifies even small sensing perturbations, increasing jitter.

These results highlight the importance of balancing jitter and lag. Jitter should be less than 1mm mean-to-peak, but lag should be below 60 ms. As we shall see, any filter introduces some lag and considering 40-50 ms of inherent system lag [5], that leaves less than 10-20 ms for the filter.

Moving average

By the Central Limit Theorem and reasonable assumptions, averaging enough successive values of a noisy signal should produce a better estimate of the true one [9]. As a result, a *moving average* of the last n data values is commonly used by computer scientists as a kind of filter. For example, Myers et al. [4] used one for laser pointers and reduced hand tremor jitter from ± 8 pixels to between ± 2 and ± 4 pixels using a 0.5s window ($n = 10$). Since all n values are weighted equally, this creates a lag up to n times the sampling period.

Low-pass filters and exponential smoothing

With human movements, noise typically forms high frequencies in the signal while actual limb movements have lower frequencies. A low-pass filter is designed to let these desired low frequency portions pass through, while attenuating high frequency signals above a fixed cutoff frequency. The *order* of a low-pass filter relates to how aggressively it attenuates each frequency: first order filters reduce the signal amplitude by half every time the frequency doubles, while higher order variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1 - \alpha)\, \hat{X}_{i-1} \quad (1)$$

The first term of the equation is the contribution of new input data value, and the second term adds inertia from previous values. As α decreases, jitter is reduced, but lag increases since the output responds more slowly to changes in input. Since the contribution of older values exponentially decreases, a low-pass filter will have less lag than a high n moving average filter.

Smoothing techniques used in business and economic forecasts are similar in approach to a low-pass filter. The equation for *single exponential smoothing* is very similar to Equation 1. As the name suggests, *double exponential smoothing* uses two of these equations to handle trends in the signal. Although not formally documented, the Microsoft Kinect skeleton filters appear to be a variant of this type of smoothing[1]. LaViola extended double exponential smoothing for predictive tracking [2], building on Equations 1 and 2 to predict positions τ time steps in the future (Equation 3):

$$\hat{X}_i^{[2]} = \alpha\, \hat{X}_i + (1 - \alpha)\, \hat{X}_{i-1}^{[2]} \quad (2)$$

$$P_{t+\tau} = \left(2 + \frac{\alpha\, \tau}{1 - \alpha}\right) \hat{X}_i - \left(1 + \frac{\alpha\, \tau}{1 - \alpha}\right) \hat{X}_i^{[2]} \quad (3)$$

Kalman filters

Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8]. When estimating the true position of a moving object, the process model is typically a linear function of the speed and the previous estimated position. With additional complexity, Extended and Unscented variants of Kalman filters can also model non-linear processes and observations [8].

In the frequent case where the process and measurement noise covariances are not known, one must determine them empirically. This task can be challenging, and an improperly tuned filter can increase and even degrade the signal [9], by creating artificial "overshooting" movements for example. Moreover, understanding Kalman filters requires mathematical knowledge beyond basic linear algebra such as statistics, random signals, and stochastic methods. Implementing them requires a language or library with matrix operations. And, as demonstrated by LaViola for predictive tracking, they can be considerably slower to compute than double exponential smoothing predictors (approximately $135\times$) with similar jitter and lag performance [2].

THE 1€ FILTER

The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed

[1] `http://cm-bloggers.blogspot.com/2011/07/kinect-sdk-smoothing-skeleton-data.html`

in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \quad (4)$$

$$\tau = \frac{1}{2\,\pi\,f_c} \quad (5)$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1 + \frac{\tau}{T_e}} \quad (6)$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \quad (7)$$

The sampling period T_e (or its inverse, the sampling rate) can be automatically computed from timestamps, so the cutoff frequency f_c is the only configurable parameter in equation 6. As with any low-pass filter, decreasing f_c reduces jitter, but increases lag. Finding a good trade-off between the two is difficult since people are more sensitive to jitter at low speeds, and more sensitive to lag at high speeds. This is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{\hat{X}}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving only two configurable parameters: the intercept $f_{c_{min}}$ and the slope β shown in Equation 7. Details of the algorithm are provided in the Appendix.

Tuning and Applications

To minimize jitter and lag when tracking human motion, the two parameters can be set using a simple two-step procedure. First β is set to 0 and $f_{c_{min}}$ to a reasonable middle-ground value such as 1 Hz. Then the body part is held steady or moved at a very low speed while $f_{c_{min}}$ is adjusted to remove jitter and preserve an acceptable lag during these slow movements. Next, the body part is moved quickly in different directions while β is increased with a focus on minimizing lag. Note that parameters $f_{c_{min}}$ and β have clear conceptual relationships: if high speed lag is a problem, increase β; if slow speed jitter is a problem, decrease $f_{c_{min}}$. Rotational input uses a similar tuning process, but rotation axis and angle are filtered separately.

Another application of the 1€ filter is displaying noisy numerical values, such as an unsteady frame rate used to monitor graphical application performance. The goal is to reduce jitter to make the numerical output legible while minimizing lag so the value remains timely. Tuning is similar to above: adjust $f_{c_{min}}$ until the text becomes stable, then increase β until just before the text become unstable.

COMPARISON WITH OTHER FILTERS

To compare the 1€ filter with other techniques, we created a Python application that periodically samples the XY position of the system cursor, adds noise, and displays filtered cursor positions. Each filter can be tuned interactively and all filters can be shown simultaneously making it possible to visually compare jitter reduction and lag across parameter settings and filters. Once tuned, timestamped positions can be logged for the system cursor (with and without noise) and filtered positions of all filters. We used a MacBook Pro with a 1440 × 900 pixel display (109 PPI).

In our comparison, we used independent Gaussian white noises for X and Y with a 50 dB SNR[2], a public implementation of the Kalman filter[3], and custom implementations of a moving average, single exponential, and LaViola's double exponential smoothing. We tuned moving average first and used its performance as a baseline. We found that averaging more than 14 data values did not reduce jitter further and only increased lag, so we used n=14. Then we interactively tuned the other filters to primarily match the jitter reduction of moving average, and secondarily attempting to reduce lag.

Tuning single exponential smoothing to match the reference jitter requires a low alpha value (α=0.11) which introduces lag. This highlights the difficulty of tuning with only a single parameter. For LaViola's double exponential smoothing filter, the reference jitter is obtained with a lower alpha value (α=0.06) and with lower lag. However, this causes overshooting when the pointer abruptly decelerates. For the Kalman filter, we set the measurement noise covariance to the variance of the introduced noise (18.06) as in [2], and adjusted the process noise covariance until we obtained the reference jitter reduction (at a value of 0.3). The amount of lag for this setting was comparable to the moving average and single-exponential. For the 1€ filter, we matched the reference jitter and optimized lag using the tuning procedure described above. In the first tuning step, setting $f_{c_{min}}$ = 1 Hz and β = 0 matched the reference jitter and lag was similar to single exponential smoothing. In the second tuning step, increasing β to 0.007 made the lag almost imperceivable yet maintained the reference jitter when stationary or moving slowly. A supplementary video demonstrates this tuning process and visualizes filter performance.

For a quantitative comparison, we logged the system cursor at 60 Hz for about 1 hour during regular desktop use, then added white noise and applied the filters using the settings above. Figure 1 shows the distance from each filtered cursor position to the true one, binned into four speed intervals. Note that since we tuned the filters to match a reference jitter when not moving, the error between filtered position and noiseless position is primarily due to lag when moving. With higher speeds, the filtered position lags farther and farther behind, increasing this distance (the small distances in the 0 mm/s interval are likely due to offset or overshooting). All filters introduce a similar amount of lag except for the 1€ filter which has less lag across all speed intervals.

As an overall comparison, we computed the Standard Error of the Mean (SEM) in mm for each filter for this data

[2]This signal-to-noise ratio was estimated from Gametrak data using a zero phase shift filter and is consistent with numbers in [2]

[3]`http://greg.czerniak.info/node/5`

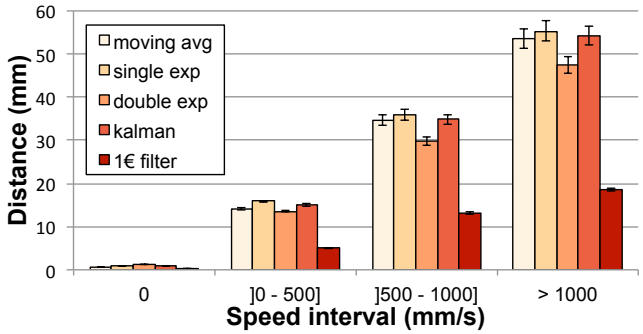

Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

CONCLUSION

Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES

1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. of UIST '05*, ACM (2005), 33–42.
7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

APPENDIX A - 1€ FILTER

Algorithm 1: 1€ filter

EXT: First time flag: *firstTime* set to *true*
Data update rate: *rate*
Minimum cutoff frequency: *mincutoff*
Cutoff slope: *beta*
Low-pass filter: *xfilt*
Cutoff frequency for derivate: *dcutoff*
Low-pass filter for derivate: *dxfilt*
IN : Noisy sample value: *x*
OUT: Filtered sample value

```
if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

EXT: First time flag: *firstTime* set to *true*
IN : Noisy sample value : *x*
Alpha value : *alpha*
OUT: Filtered value

```
if firstTime then
    firstTime ← false
    hatxprev ← x
end
hatx ← alpha * x + (1 - alpha) * hatxprev
hatxprev ← hatx
return hatx
```

Algorithm 3: Alpha computation

IN : Data update rate in Hz: *rate*
Cutoff frequency in Hz: *cutoff*
OUT: Alpha value for low-pass filter

```
tau ← 1.0 / (2*π*cutoff)
te ← 1.0 / rate
return 1.0 / (1.0 + tau/te)
```