FlyPT mover

Manual for version 2.9.3

INDEX

FILTERS

Filters might be the base of all we do in Mover.
A filter is a way of conditioning/transforming a value by performing operations over it.
In reality, I call it filters, because they change the value, but some are just operations like addition or multiplication.
In this chapter, I try to explain the filters. In the end, I will give some examples of filtering, but for motion, I will give examples on the motion chapter.

A filter is defined by a keyword, and a serie of parameters:

```
KEYWORD(PARAMETER1;PARAMETER2;PARAMETER3)
```

The keyword defines the type of filter you want to apply.
The number of parameters, depend on the filter you are using.
One of those parameters, the first one, is usually what we want to filter:

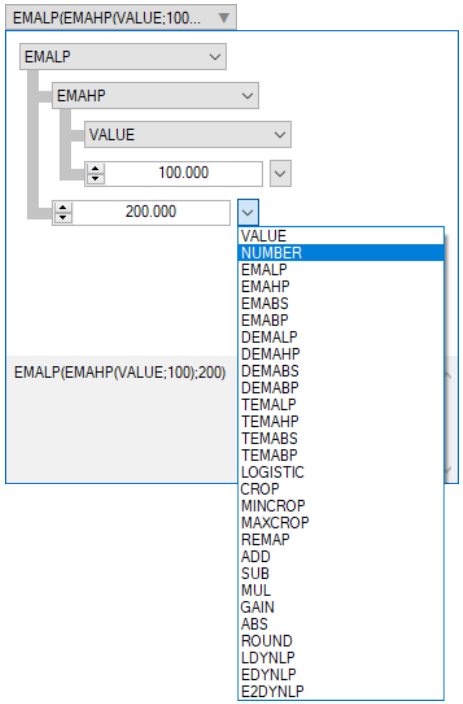
```
KEYWORD(VALUE;PARAMETER2;PARAMETER3)
```

The keyword VALUE represents the value we receive to be filtered.
For example if we are making a filter for sway and we are receiving the lateral acceleration, VALUE is the value received from the source/game.
But any parameter can be also a filter. So you can compose filters:

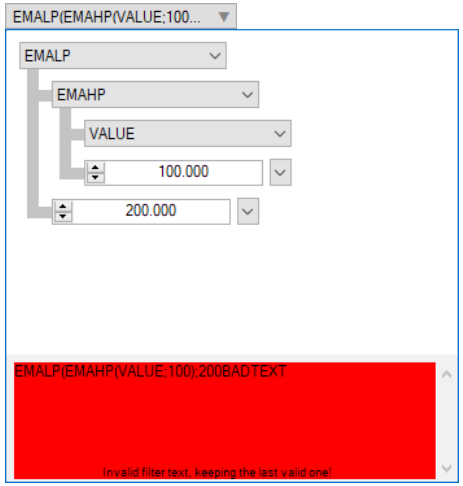
```
KEYWORD(KEYWORD(VALUE;PARAMETER1);PARAMETER2;PARAMETER3)
```

In this example, the first parameter is also a filter.
Offcourse, to calculate the main filter, Mover needs to calculate that “inner” filter in PARAMETER1 first.
So we can compose filters like a tree, and that's what we try to represent in the interface.

When editing a filter, we can use the drop down boxes and work on the tree or we can edit the text of the filter (in the gray area bellow).
When editing the filter if the current filter is not valid, the last valid filter will be the one in use and the text area will be red.

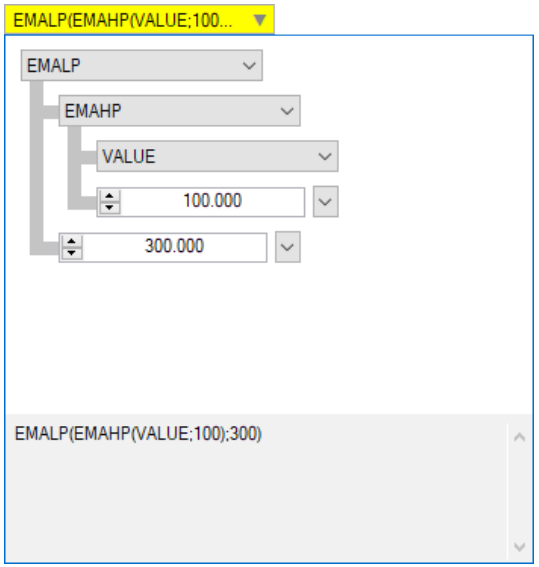


Example of a composed filter in Mover and it's tree



Error in the filter text

Another feature present in the filters is the filter transition.
Each time we change the filter, a transition between the old and the new filter is performed to make the change smooth.
While the transition is performed, the background of the filter control is yellow.
The time used in the transition, can be adjusted in the main menu of Mover, under options.

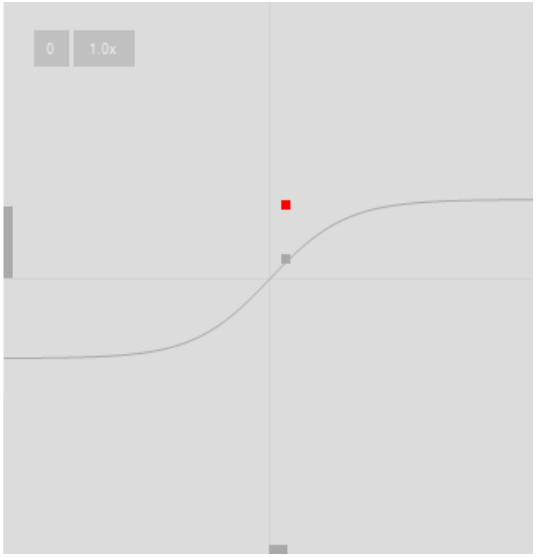


Yellow background while transition is performed

IN/OUT GRAPHIC

To make the filtering more visual, a graphic is shown with the filters.
That graphic shows the input value and respective output.
Input in the horizontal, and output in the vertical.
The shown line is the result of filters that are not time/sample based.
So for a specific input value, the grey dot shows the filtered value in the line, but that's without the time/sample based filters.
The result witht time/sample based filters is shown in the red dot.
Besides the points, in the left is shown a bar with the output value and bellow a bar with the input value.
You can move the graphic by draging it or zoom with the mouse wheel.
To reset position press the 0 button and to reset scale to 1, press the scale button.

All the filters marked with * in this manual are based on time/samples. They do not show on the in/out graphic.



AVAILABLE FILTERS

Let's now see all the available filters and respective keywords we have in Mover.
For now we look at the theory.
To start, let me clarify that when I say value/values, I mean a sequence of values that generates movement in the rig.
That sequence of values can be seen in the graphics viewer.
We are aplying the filters to that sequence of values or the last received value.
All the graphics shown in the tutorial are from sequences of values generated in Mover, so they are not real data from a game, but an easy way to show the results.
I encourage you to test the filters by using the calculated sources like we do here to show the filters.

VALUE

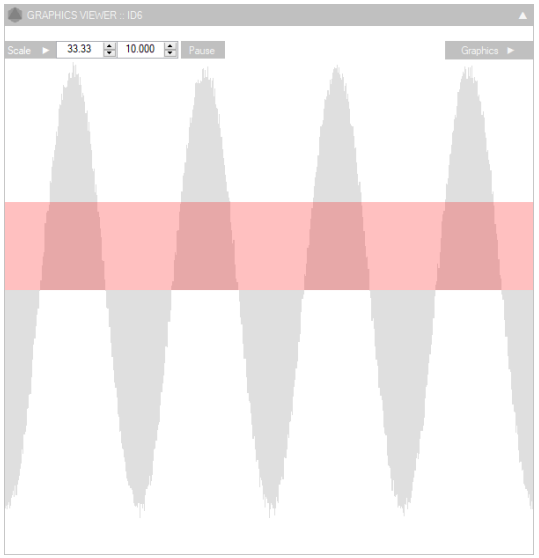
Like I said above, it represents the received value and has no parameters.
This is the default selection on the filters, and using it is the like not having a filter.

NUMBER

This keyword allows us to introduce a numeric value.
The keyword is visible only in the tree drop downs. You can't use it in the text.
To use a number in the text, just place the numeric value you want.

MINCROP(VALUE;100)

In this example, 100 is the numeric value.



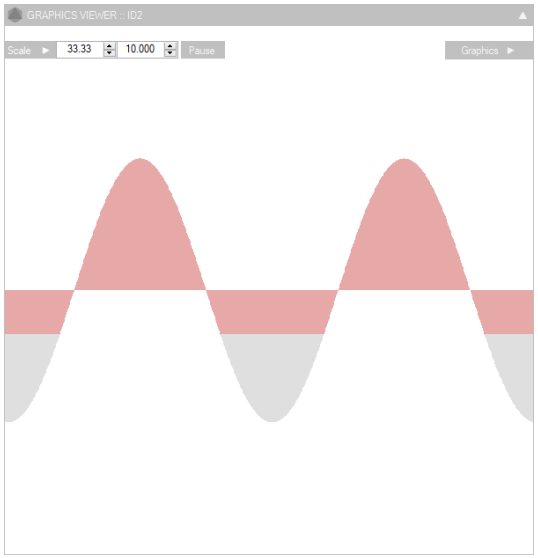
Here we can see in grey the received value, a noisy sine wave.
That's the VALUE.
In red we have afilter of type NUMBER with the value 10.
So the output is a constant 10.

MINCROP

This filter cuts all values under a certain limit.
Only values above or equal to that limit are allowed.

```
MINCROP(parameter1;parameter2)
```

parameter1 is what we want to filter
parameter2 is the minimum value we allow to pass



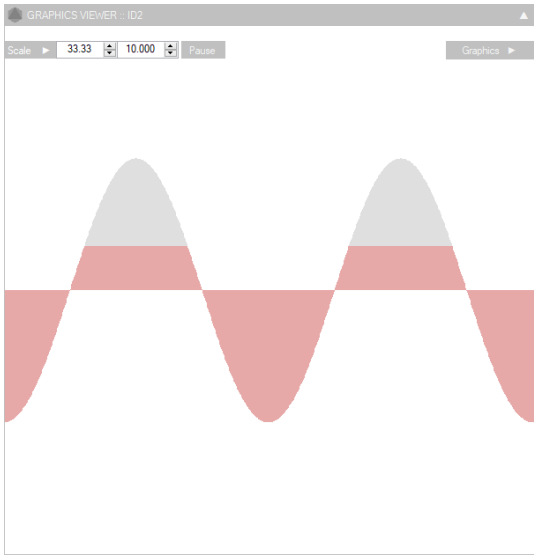
This is the result of a MINCROP(VALUE;-5).
All values under -5 are removed.
(In gray, the original values, and in red the filtered values)

MAXCROP

This filter cuts all values above a certain limit.
Only values under or equal to that limit are allowed.

```
MAXCROP(parameter1;parameter2)
```

parameter1 is what we want to filter
parameter2 is the maximum value we allow to pass



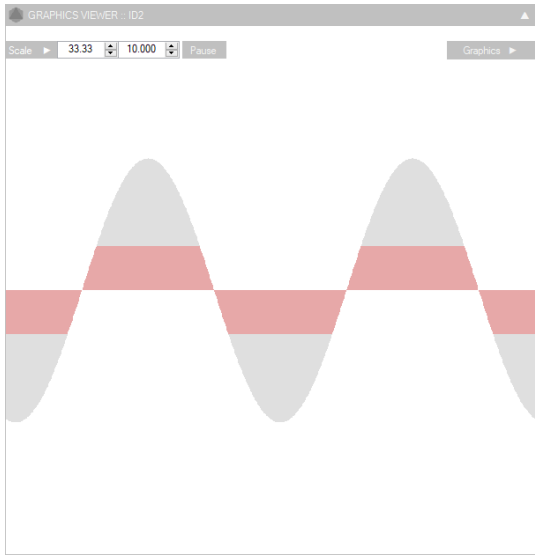
Result of a MAXCROP(VALUE;5)

CROP

This filter cuts all values under the lowest limit defined
and above the highest limit defined.

```
CROP(parameter1;parameter2;parameter3)
```

parameter1 is what we want to filter
parameter2 is the minimum value we allow to pass
parameter3 is the maximum value we allow to pass



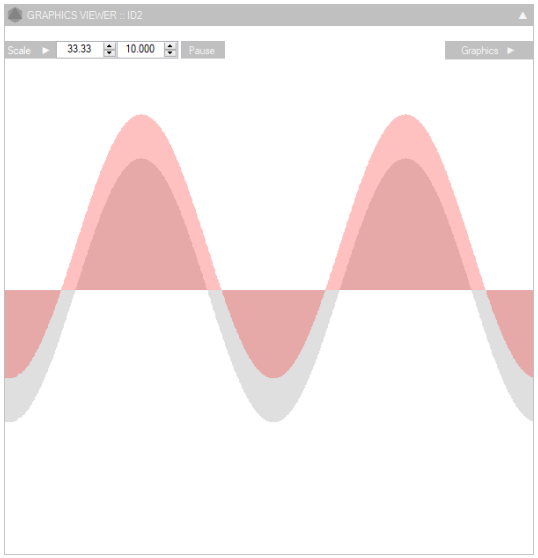
Result of CROP(VALUE;-5;5)

ADD

Use to increase a value.

```
ADD(parameter1;parameter2)
```

parameter1 and parameter2 are the values we want to add.



Result of ADD(VALUE;5)

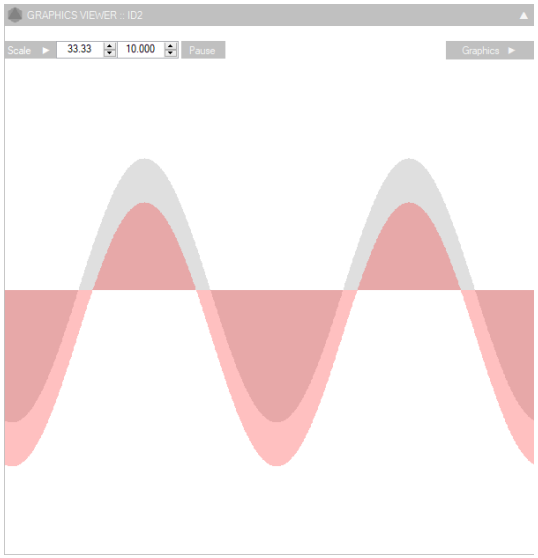
SUB

Use to decrease a value

```
SUB(parameter1;parameter2)
```

parameter1 is the value we want to be transformed.
parameter2 is the amount to subtract to parameter1.

Result is parameter1 – parameter2



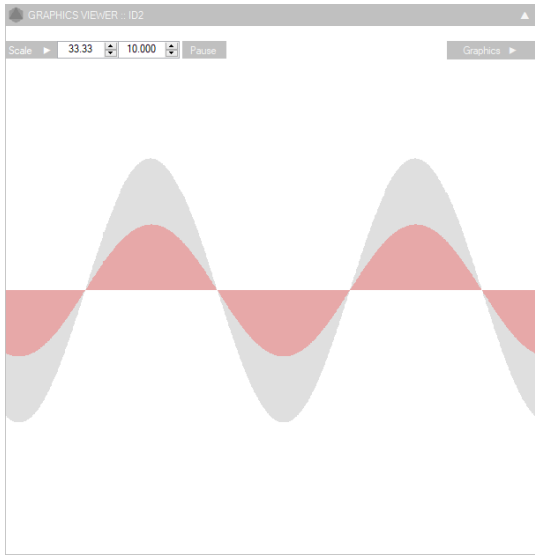
Result of SUB(VALUE;5)

GAIN or MUL

Use to multiply two values or scale a value.

```
GAIN(parameter1;parameter2)
MUL(parameter1;parameter2)
```

Gain and MUL are the same, just with different keywords, where MUL means multiply.
parameter1 and parameter2 are the values we want to multiply.



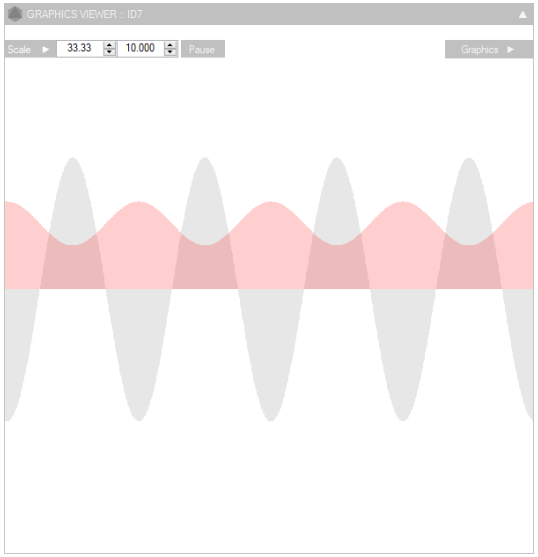
Result of GAIN(VALUE;0.5)

REMAP

Remap is used to translate and scale a value.
Or like the name says, remap a value from one range into another one.

```
REMAP(parameter1;parameter2;parameter3;parameter4;parameter5)
```

parameter1 is the value we want to remap.
parameter2 and parameter3 define the range of the parameter1.
parameter4 and parameter5 are the new range we want.



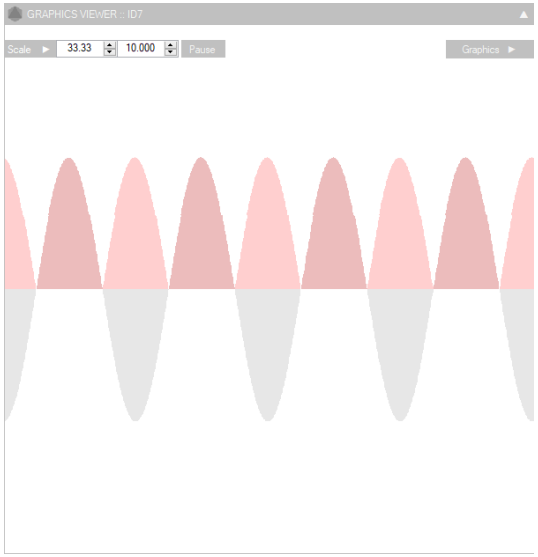
Result of REMAP(VALUE;-15;15;10;5)

ABS

Abs is used to make all values positive.
With this filter we get absolute values.

```
ABS(parameter1)
```

parameter1 is the value we want to remap.



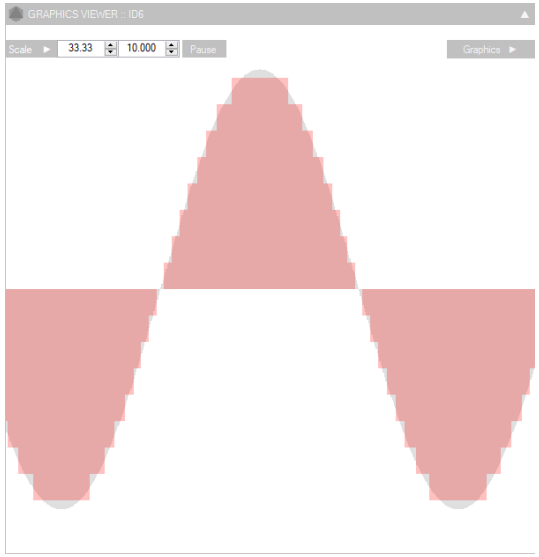
Result of ABS(VALUE)

ROUND

Round to nearest value

```
ROUND(parameter1;parameter2)
```

parameter1 is the value we want to remap.
parameter2 is the multiple we want to round to.



Result of ROUND(VALUE;3)

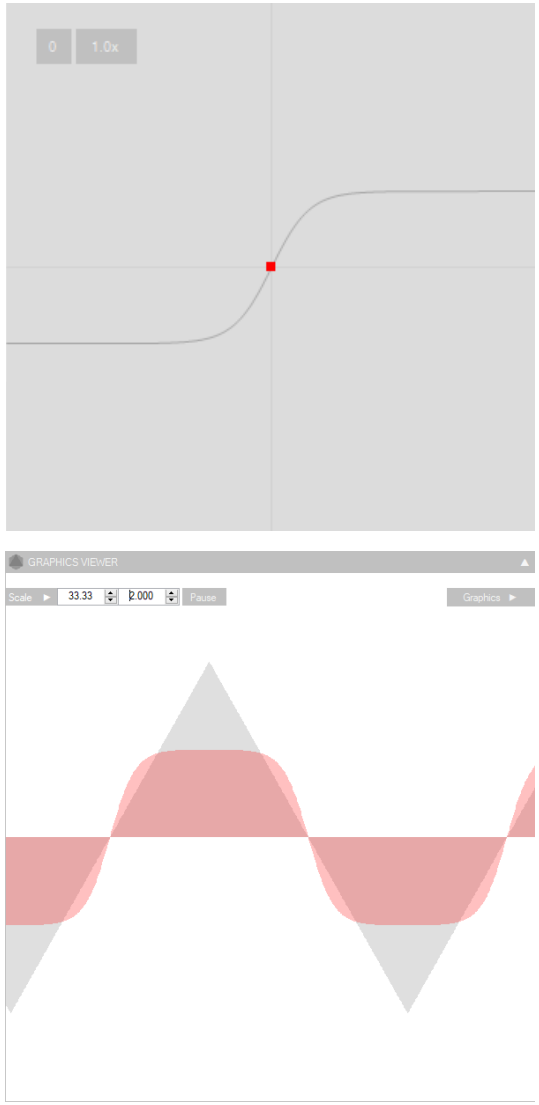
LOGISTIC

The logistic filter works as a limit filter, but closer the value get's to the limit, slower it goes there.
If above the limit, greater the value, even slower it goes. So it's an infinitesimal aproach to the limit without never reaching it.

```
LOGISTIC(parameter1;parameter2;parameter3)
```

parameter1 is the value we want to limit.
parameter2 is the range that defines the minimum/maximum value.
parameter3 is how fast we go to the limit.

Example of a LOGISTIC(VALUE;50;1) in a triangle with 100 amplitude. The value stays inside the -50/50 range and aproaches the limit slower and slower as it get's near or surpasses the limit.



Example of a LOGISTIC(VALUE;50;2) in the same triangle loop. The value aproaches the limit faster

EMALP*

EMA is an exponential moving average.

EMA is a way of calculating an average value.
Usually, an average is calculated by adding values and then dividing by the amount of values added.
The exponential moving average does the same, but the old values have less wheight on that average, while the most recent ones have an higher wheight on the average.
The wheight of a value in the average, varies exponentially. That's why we call it exponential moving average.
The advantage of this filter is that it's fast to calculate, and follows closelly the received values.

In the top figure we have an example.
We are receiving a signal (in grey) full of noise. By using a EMA with the last 200 values, we can remove the noise and we get the almost clean sinusoid that was hidden inside that signal.
The EMA as smoothned the signal, by using 200 values, where the oldest value as almost zero wheight in the average and the last one as full wheight.

LP is low pass.

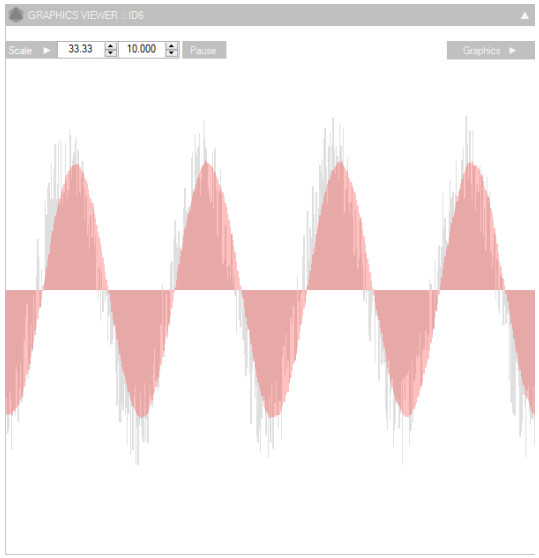
This means we only allow low frequency to pass.
If you look at the example given, by aplying the EMA, we are removing the spikes, that are high frequency signals.
So the EMA is in reality a EMALP.

EMALP is a low pass filter, where only the low pass frequencies are allowed and the high frequencies are removed.
The strenght of the smoothening depends on the amount of samples used.

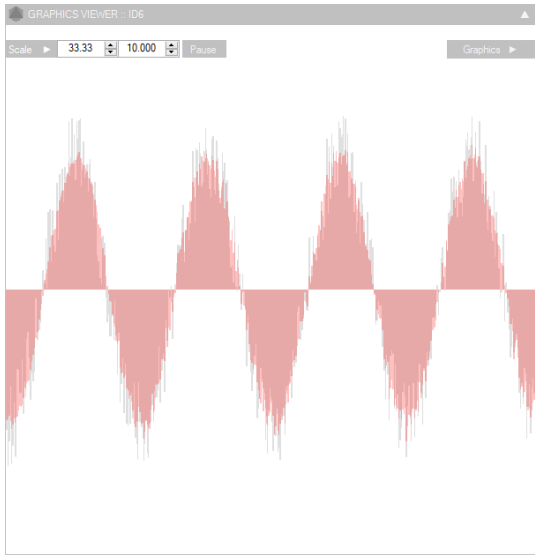
To use it in Mover we do:

```
EMALP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the EMA.



EMA with 200 samples



Result of EMALP(VALUE;10)
Using 10 samples

To better understand the behaviour of the filter, we can look at the filter in the frequency domain.

In the low pass filter we are requesting to just allow through the filter, low frequency signal.
Cutoff frequency is the frequency that defines that point.

But? Where is frequency? How do I define the frequency where I want to stop receiving values for this low pass filter?

Well, with Mover, we don't have the "frequency", we have the number of samples used to calculate the average.
Like I said above, more samples, more spikes removed, softer the signal.

We have the frequency, but it depends on the elapsed time between each sample received.
To achieve fast calculations and coherant results, Mover tries to keep a steady calculation time in each loop. So Mover ignores frequency. There's no need here to look for the value of frequency.
By default, the calculation time in Mover is 2 ms. You can change it in the Main window menu, under options.
So when compairing results between users, you should always say the filter parameters and the calculation rate, because results are different for different calculation rates.

Be also aware that the cutoff frequency is not all in all out.

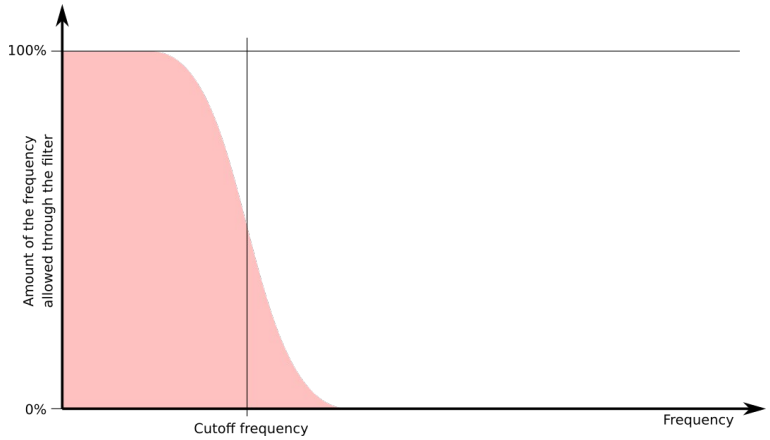
The figure shows a low pass filter.
From left to right we have frequency increasing.
Vertically, we have how much of the frequency passes on the filter.
For the lowest frequency, all values pass, and when we start reaching the cutoff frequency, the amount starts decreasing and will reach zero after the cutoff frequency.
So cutoff frequency is not all in all out. It has a transition.
That's what happens on our filters. There's no clear definition between what passes and what doesn't pass.

So that 200 that appears on the definition of the filter, is the number of samples used by the filter.
It's the way to define the cutoff frequency in Mover.

If you use one sample, that's the equivalent to infinit value for cutoff frequency. Everything passes.
You are calculating an average using one value with the last received value. Result is the received value.

If you use 200, then you calculate the average of 200 values. You are lowering the cutoff frequency as you increase the number of samples.
Please retain this:

More samples in the filter = Lower cutoff frequency
1 sample = Infinite cutoff frequency
More samples in EMALP = Smoother



The EMALP in the frequency domain

DEMALP* and TEMALP*

DEMA is a double exponential moving average
TEMA is a triple exponential moving average

Why this?
Because EMA has a small problem. It introduces lag. Look at the bellow figure. It's perfectlly visible the introduced lag in the noisy square wave.

To make it better, we can use the DEMA and TEMA filters. They reduce significantlly the lag, but they are slower (not to much) and they add an undesired feature, what we call the overshoot.

Look at the examples for DEMALP and TEMALP, the result responds faster, but we get an overshoot on fast changes.

To use DEMALP t in Mover we do:

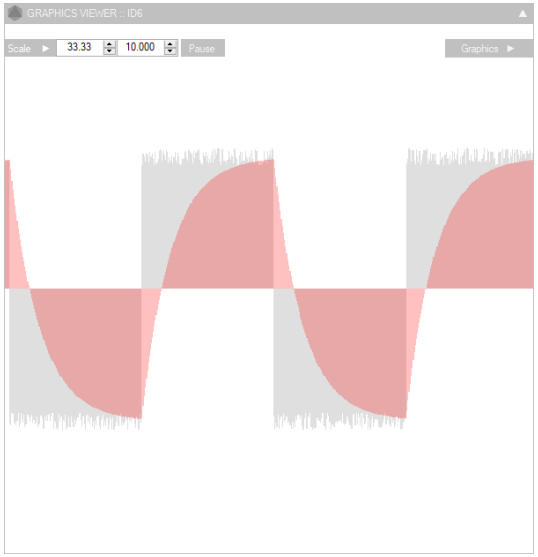
```
DEMALP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the DEMA.

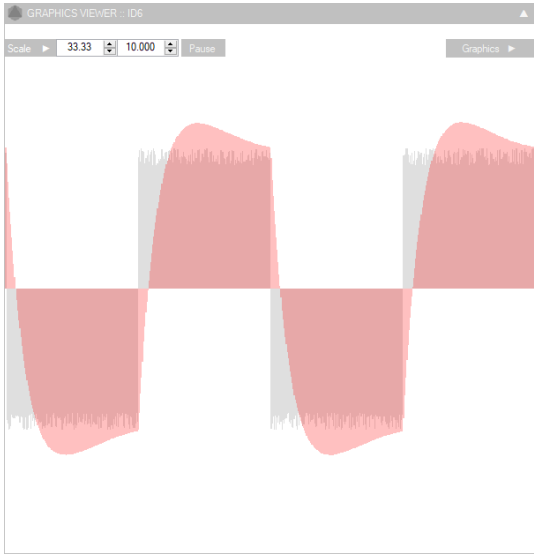
To use TEMALP in Mover we do:

```
TEMALP(parameter1;parameter2)
```

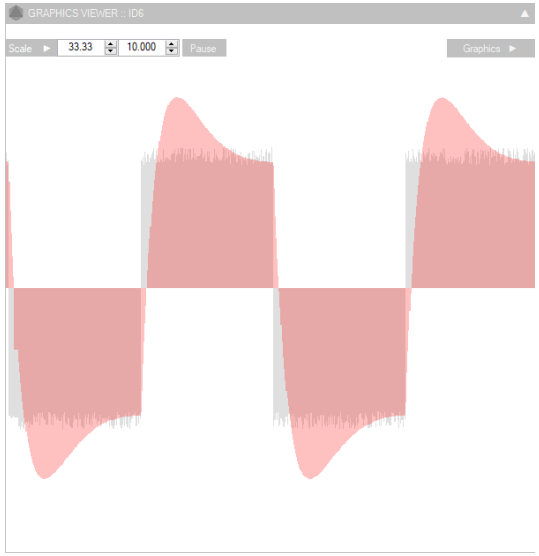
parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the TEMA.



Result of EMALP(VALUE;1000)



Result of DEMALP(VALUE;1000)



Result of TEMALP(VALUE;1000)

EMAHP*

HP of high pass.
So while in low pass we smooth the values, in high pass we get the spikes.

To do it, we first smooth the values and then we take the difference between the unsmoothed value and the smoothed ones.
Please compare the two top figures. Left is a low pass and right the high pass, both using 100 samples.

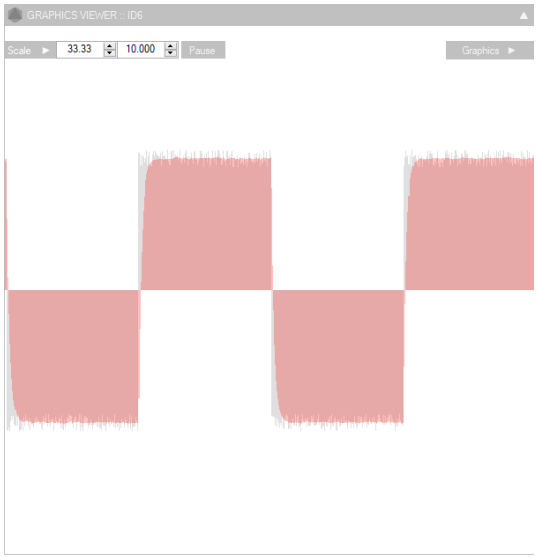
The result of this filter is the spikes on the signal we are receiving.
Looking carefully, you can see the values vary around zero, because we are getting just the spikes.
So the high pass washes always to zero.

To use EMAHP t in Mover we do:

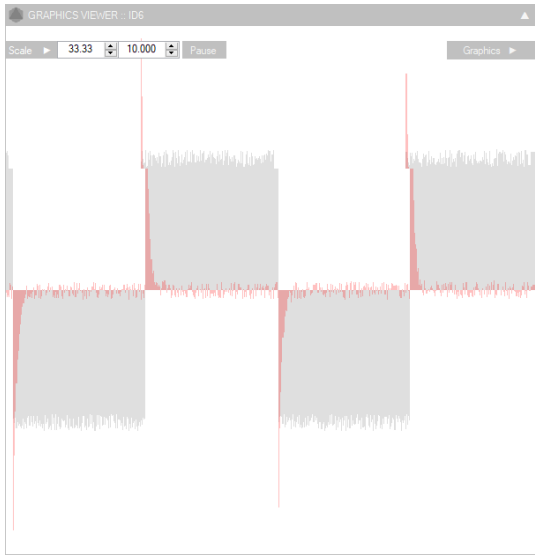
```
EMAHP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the EMA.

Please take into account that the number of samples is for the low pass since high pass is original minus low pass.

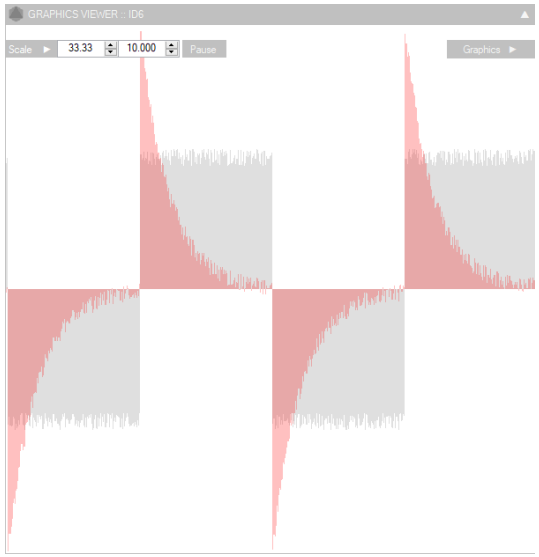


The result of EMALP(VALUE;100)
Here to compare with the EMAHP



The result of EMAHP(VALUE;100)
If you look carefully you can see that's equal to VALUE-EMALP(VALUE;100)

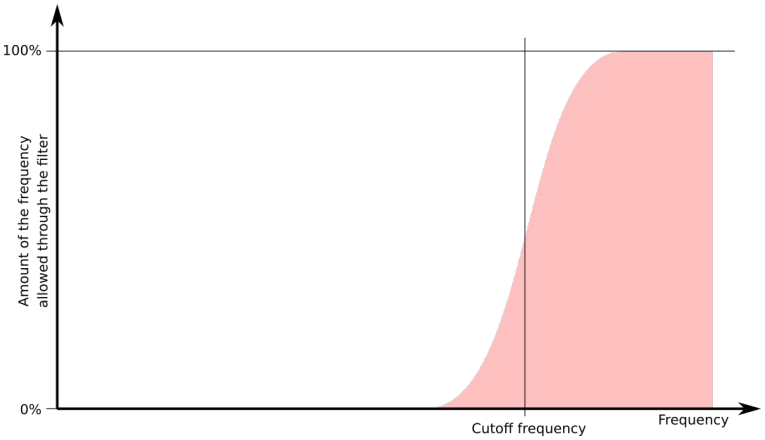
The result of EMAHP(VALUE;10)



The result of EMAHP(VALUE;1000)

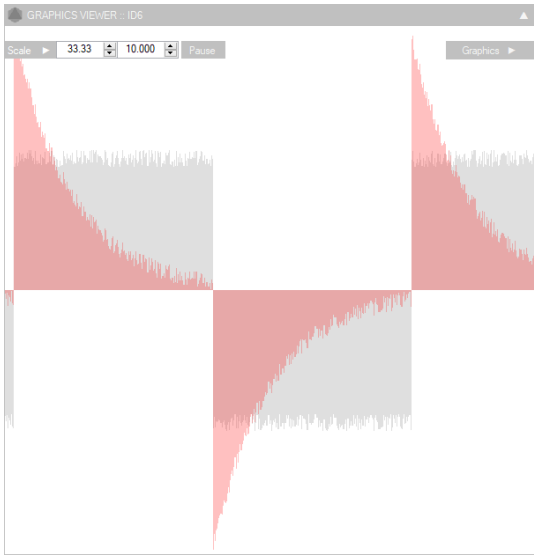
Looking at the EMAHP in the frequency domain we can see that:

More samples in the filter = Lower cutoff frequency
1 sample = Infinite cutoff frequency
More samples in EMAHP = Becomes smoother preserving spikes

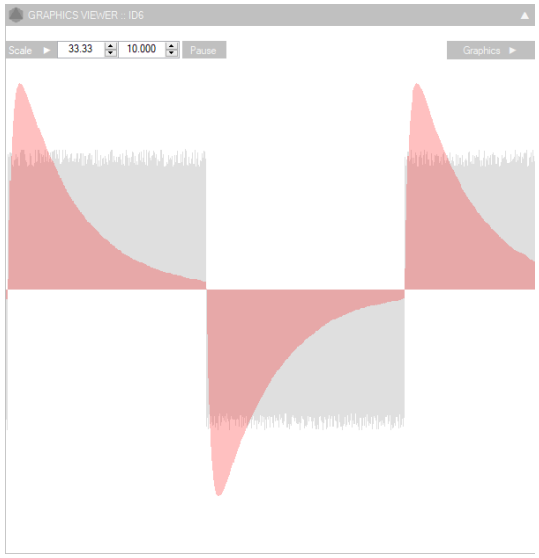


The EMAHP in the frequency domain

This is a good filter to washout the constant values and keep the noise.
In the example of the left image we have an EMAHP(VALUE;2000) making the washout. It keeps the small details of the noise.
But we can filter it again with a low pass to smooth it.
That's what we see in the right image example.



The result of EMAHP(VALUE;2000)



The result of EMALP(EMAHP(VALUE;2000);150)

DEMAHP* and TEMAHP*

So those are the original value less the DEMALP or TEMALP.

They look to unstable for use because of the overshoot that causes a strange behaviour.

In the images we can compare the EMAHP, DEMAHP and TEMAHP.

To use DEMAHP t in Mover we do:

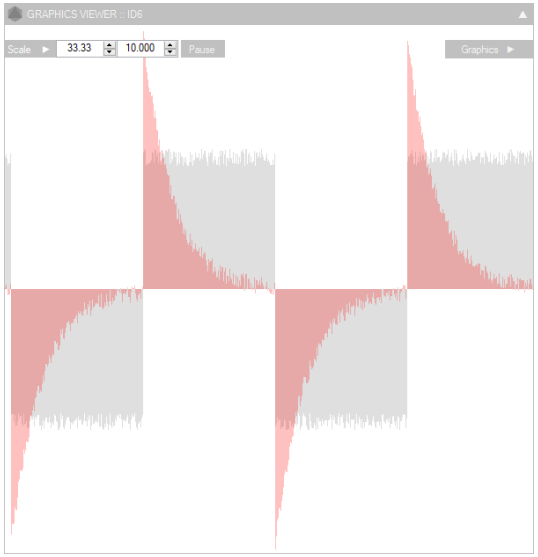
```
DEMAHP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the DEMA.

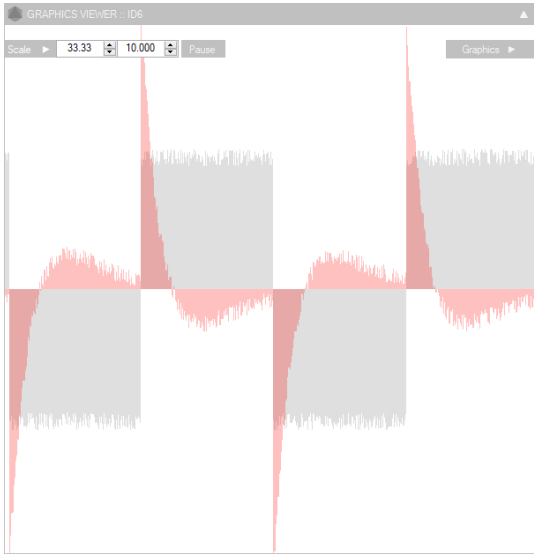
To use TEMAHP in Mover we do:

```
TEMAHP(parameter1;parameter2)
```

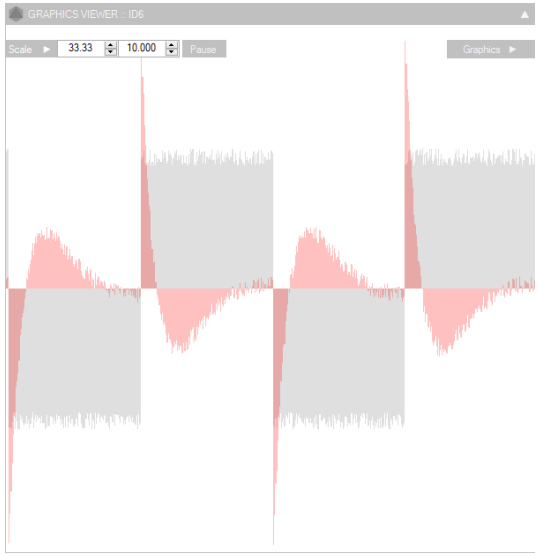
parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the TEMA.



Result of EMAHP(VALUE;1000)



Result of DEMAHP(VALUE;1000)



Result of TEMALP(VALUE;1000)

EMABP*

BP is band pass.
This means we receive values inside a specific band of frequencies.

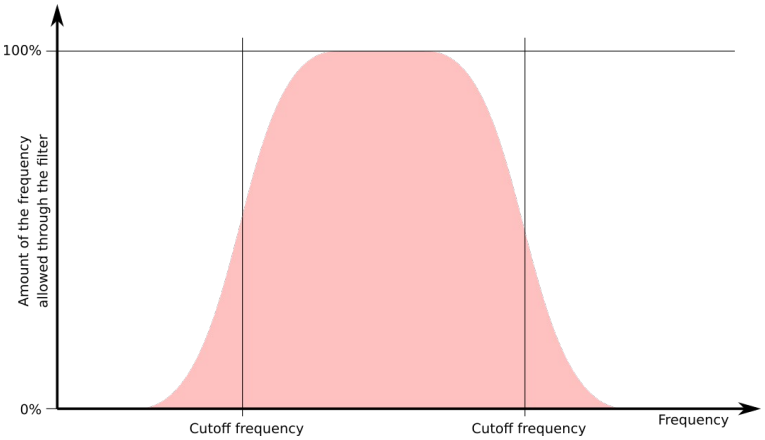
In the image we can see the filter in the frequency domain, showing the band of frequecies defined by the two cutoff frequencies, where we receive data.

To use it in Mover, do:

```
EMABP(parameter1;parameter2;parameter3)
```

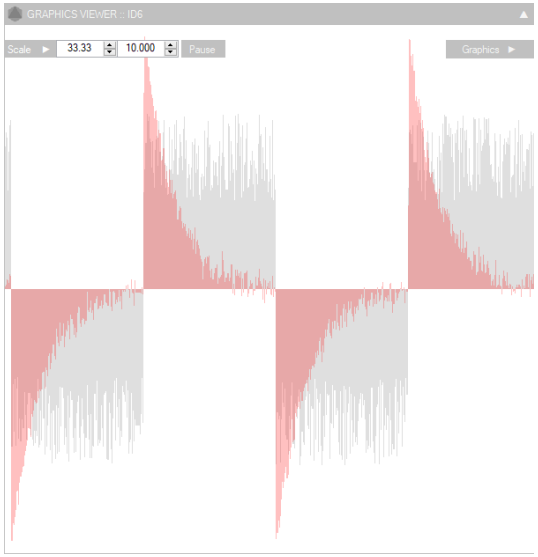
parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the EMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the EMA for the other cutoff frequency.

You can swap parameter2 and parameter3, Mover will always pass the band between the smaller and the bigger frequency.

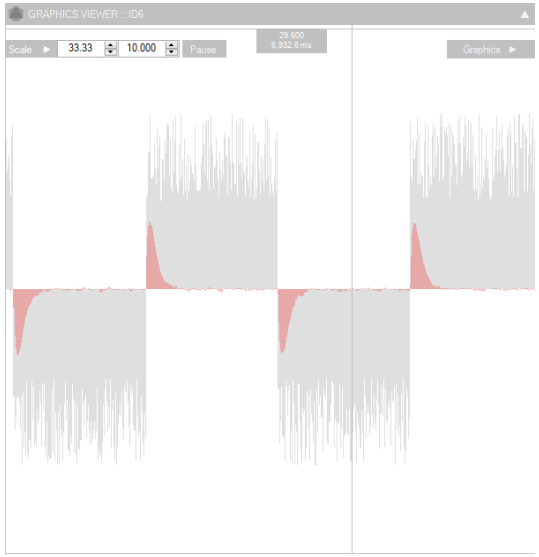


The EMAPB in the frequency domain

The use of this filter is a good alternative for washout.
Instead of using an EMALP over a EMAHP, we can use just the EMABP.
They are not the same. But with the EMABP, adjusting the lower value is easier to get more or less spikes, while the higher value defines how fast we go to zero in the washout.



Example of EMABP(VALUE;10;1000)



Example of EMABP(VALUE;100;200)
Going faster to zero with less spikes

DEMABP* and TEMABP*

The double and triple band passes look to unstable for use because of the overshoot that causes the strange behaviour.
They can be used, but in practise they look to strange for good results.

In the images we can compare the EMABP, DEMABP and TEMABP.

To use DEMABP t in Mover we do:

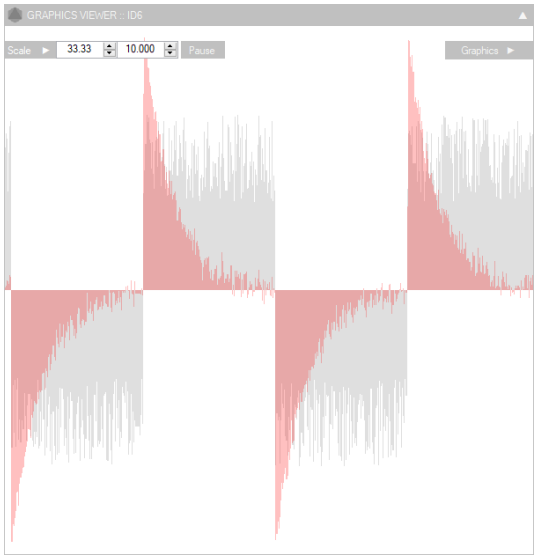
```
DEMABP(parameter1;parameter2;parameter3)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the DEMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the DEMA for the other cutoff frequency.

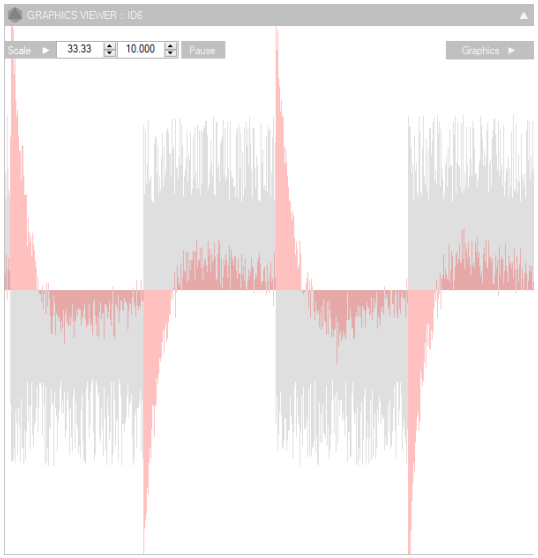
To use TEMABP in Mover we do:

```
TEMABP(parameter1;parameter2;parameter3)
```

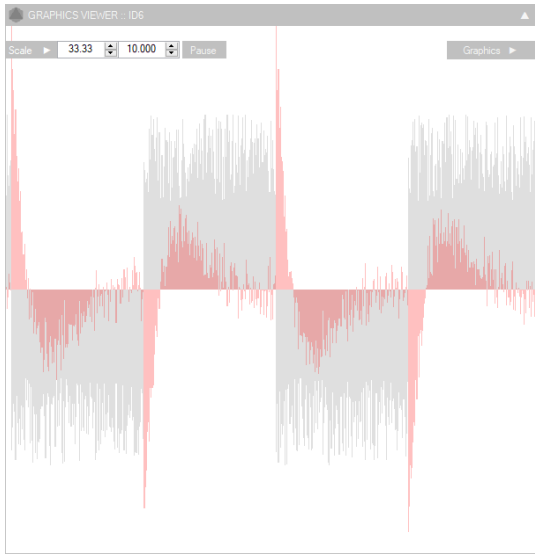
parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the TEMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the TEMA for the other cutoff frequency.



Example of EMABP(VALUE;10;1000)



Example of DEMABP(VALUE;10;1000)



Example of TEMABP(VALUE;10;1000)

EMABS*

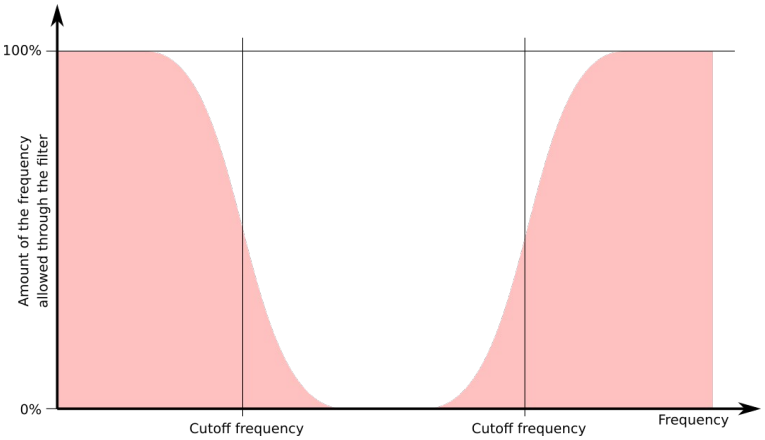
BS is band stop.
This means we don't receive values inside a specific band of frequencies.
In the image we can see the filter in the frequency domain, showing the band of frequecies defined by the two cutoff frequencies, where we stop receiving data.
To use it in Mover, do:

```
EMABS(parameter1;parameter2;parameter3)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the EMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the EMA for the other cutoff frequency.

You can swap parameter2 and parameter3, Mover will always remove the band between the smaller and the bigger frequency.

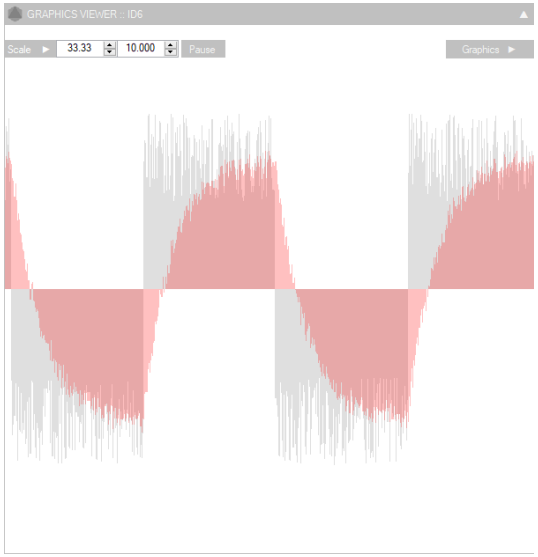
Note that the EMABS = ORIGINAL - EMABP



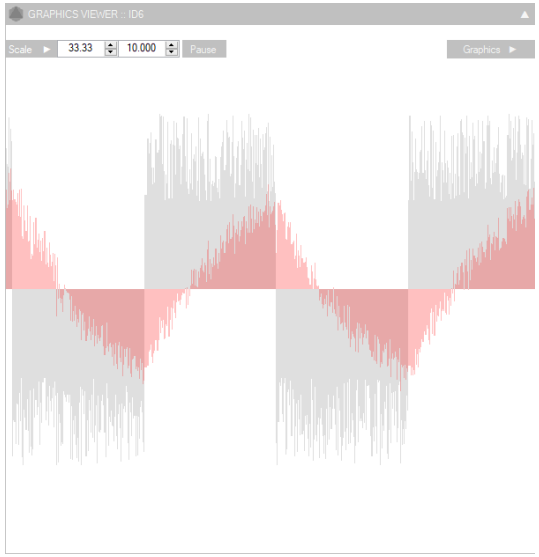
The EMABS in the frequency domain

The EMABS is good to smooth a value and get the high frequencies.
It's like the opposite to washout.
It's washing in to the current value.

Adjusting the lower value gives you more or less spikes, while the higher value defines how fast we go to the current value (washin).



Example of EMABS(VALUE;1,5;1000)



Example of EMABS(VALUE;2;3000)

DEMABS* and TEMABS*

The double and triple band stops look to unstable for use because of the overshoot that causes the strange behaviour.
They can be used, but in practise they look to strange for good results.

In the images we can compare the EMABS, DEMABS and TEMABS.

To use DEMABS t in Mover we do:

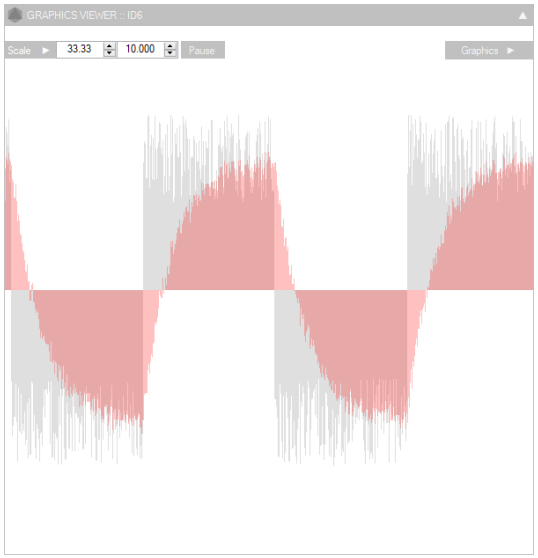
```
DEMABP(parameter1;parameter2;parameter3)
```

parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the DEMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the DEMA for the other cutoff frequency.

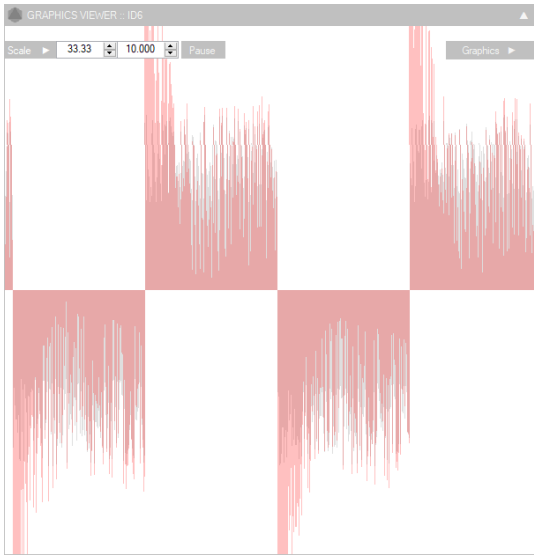
To use TEMABS in Mover we do:

```
TEMABP(parameter1;parameter2;parameter3)
```

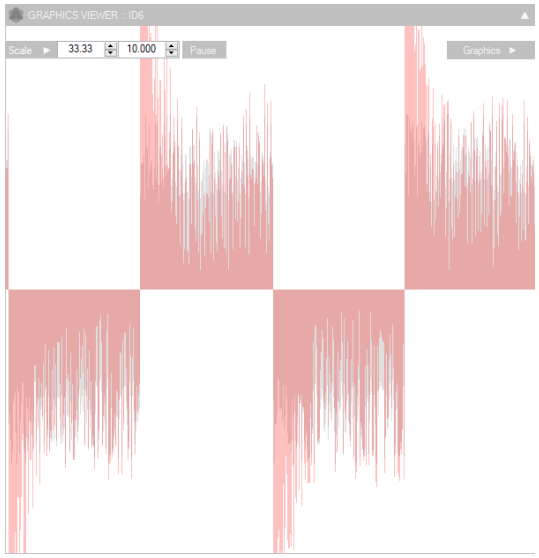
parameter1 is the value we want to filter.
parameter2 is the amount of samples/values used in the TEMA for one of the cutoff frequencies.
parameter3 is the amount of samples/values used in the TEMA for the other cutoff frequency.



Example of EMABS(VALUE;1,5;1000)



Example of DEMABS(VALUE;1,5;1000)



Example of TEMABS(VALUE;1,5;1000)

SPIKE*, EDYNLP* and LDYNLP*

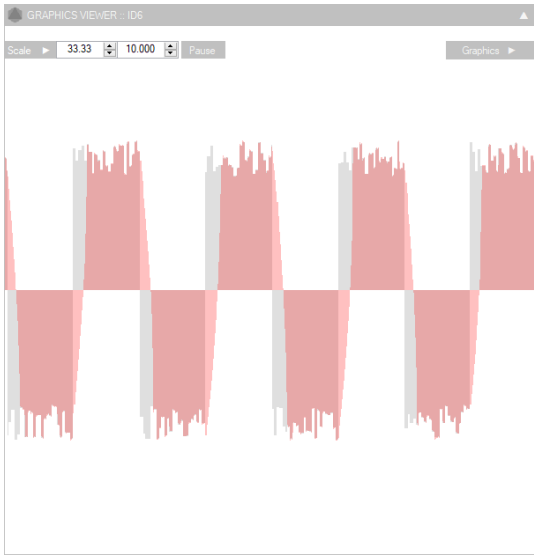
EDYNLP and LDYNLP are specific SPIKE cases. They are dynamic EMALP filters used to remove spikes.

To use it in Mover do:

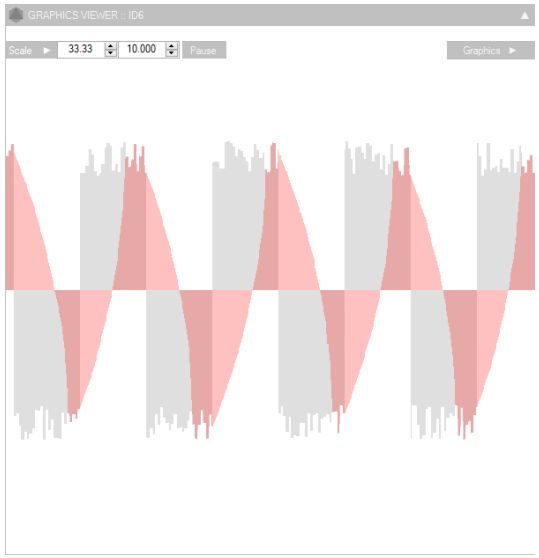
```
SPIKE(parameter1;parameter2;parameter3;parameter4)

EDYNLP(parameter1;parameter2;parameter3)

LDYNLP(parameter1;parameter2;parameter3)
```

parameter1 is the value we want to filter.
parameter2 is the window where the filter has no effect.
parameter3 is the strenght of the spike removal.
parameter4 is the exponential degree.

In EDYNLP, parameter4 is 2 and in LDYNLP it has a value of 1.



Example of SPIKE(VALUE;1)

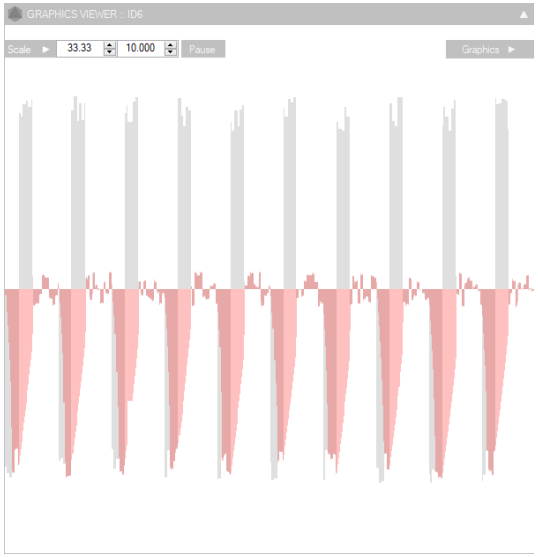


Example of SPIKE(VALUE;4)
Transitions becoming smoother.

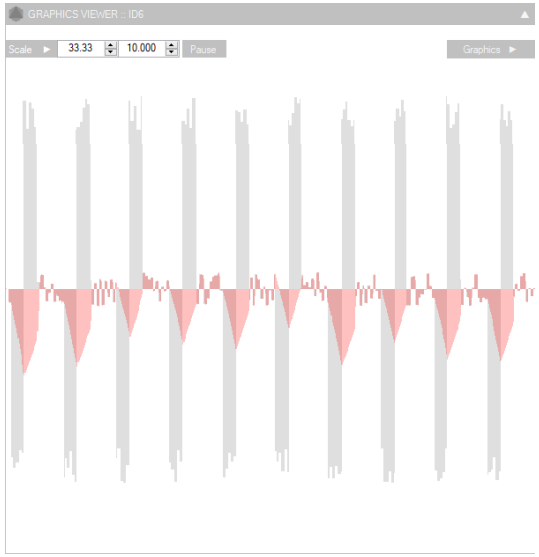
The filter acts by aplying a stronger filter for larger variances of value.
How much the value increases, depends on the exponential degree and strenght.

Filter strongness = strenght x variance^{degree}

Parameter2 defines a window of variance where the filter doesn't act.
So if window is 10, the filter starts to work above 10 keeping the received value original if under or equal to 10.



Example of SPIKE(VALUE;1)
Removing some spikes and smoothing others.



Example of SPIKE(VALUE;4)

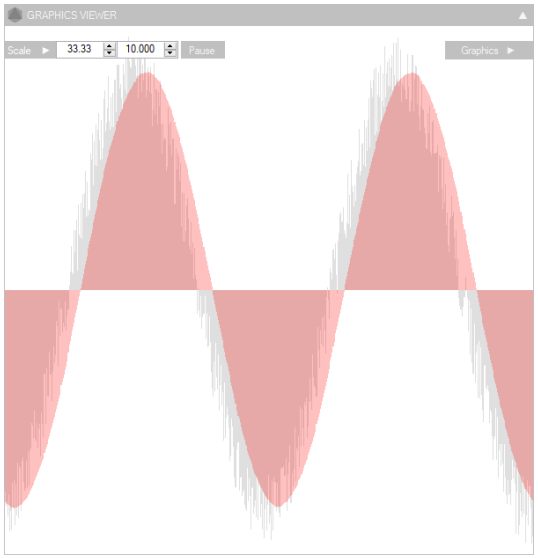
MALP*

This is a simple median average.
To use it do:

```
MALP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the number of samples used.

Compared to EMALP, this filter requires more memory.



Example of MALP(VALUE;500)

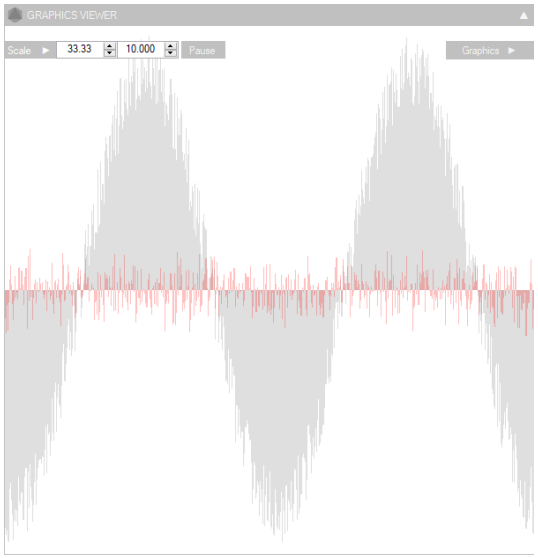
MAHP*

This is a simple median average.
But the result is equivalent to an high pass.
We subtract the value we want to filter with the MALP.
To use it do:

```
MAHP(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the number of samples used.

Compared to EMAHP, this filter requires more memory.



Example of MAHP(VALUE;500)

SIGN

Returns 1 or -1 depending on the signal of the value received.
To use it do:

```
SIGN(parameter1)
```

parameter1 is the value we want to filter.

Example of SIGN(VALUE)

CUBIC

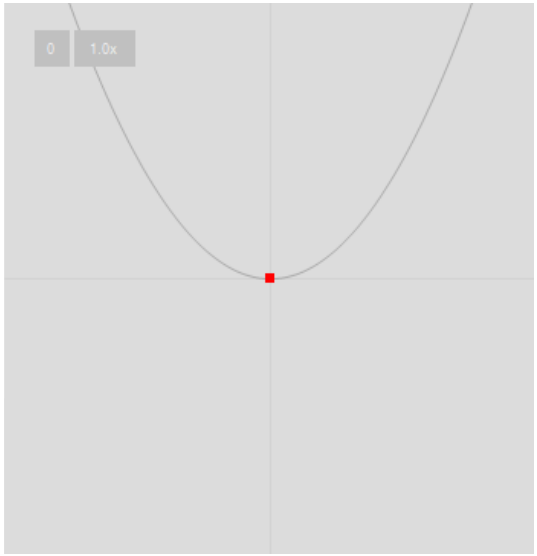
It's the result of a cubic function: ax^3+bx^2+cx+d

To use it do:

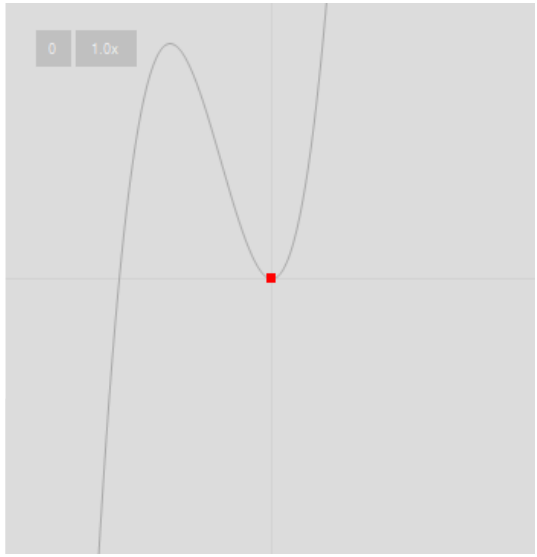
```
CUBIC(parameter1;parameter2;parameter3;parameter4;pa
rameter5)
```

parameter1 is the value we want to filter.
parameter2 is the a of the function.
parameter3 is the b of the function.
parameter4 is the c of the function.
parameter5 is the d of the function.

Used specially for traction loss and deadzones.
See the CUBIC3 which is faster and is enough for that task.
Other variations might have applications, up to your imagination.



Example of CUBIC(VALUE;0;0.01;0;0)



Example of CUBIC(VALUE;0.001;0.1;0;0)

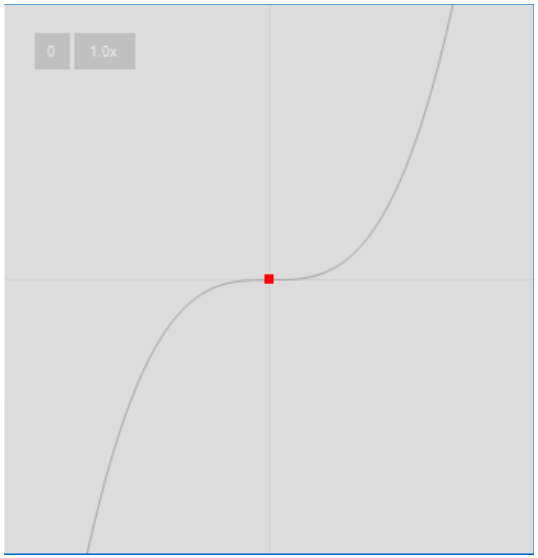
CUBIC3

Simplification of the CUBIC filter, to use specially in traction loss and deadzone definition.
To use it do:

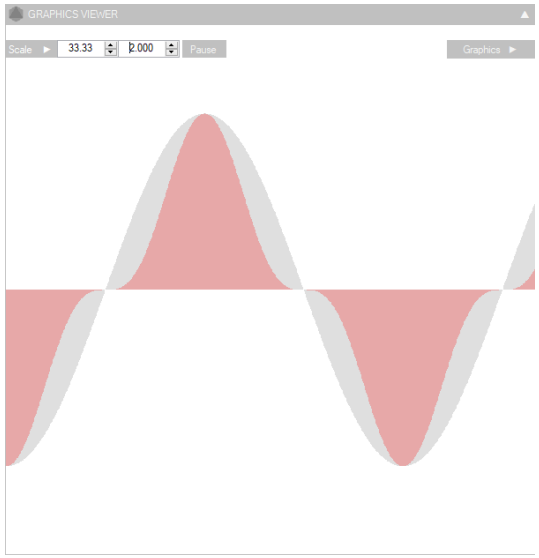
```
CUBIC3(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the a of the cubic function.

Compared to the CUBIC, all other parameters are zero.
Lower parameter3 increases the deadzone.



The in/out graphic of the CUBIC3(VALUE;0.0001)
Here we can see how CUBIC3 works as a deadzone and with exponential growth. Good for traction loss from rotation.



Example of CUBIC3(VALUE;0.001)

DIF*

Diferential of the received value.
To use it do:

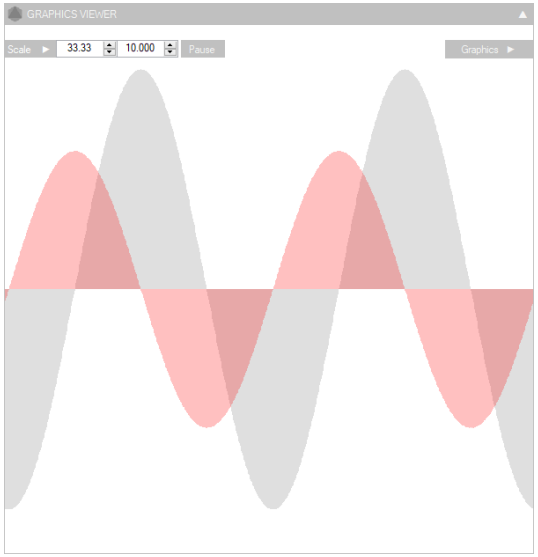
```
DIF(parameter1)
```

parameter1 is the value we want to filter.

This allows us to go from speed to aceleration for example. Or other physical values:

▼Abserk
▼Abseleration
▼Absity
▼Absement
▼Displacement
▼Speed
▼Acceleration
▼Jerk
▼Jounce
▼Flounce
▼Pounce

It's recommended the use of high definition calculation rate.



Example of DIF(VALUE)

INT*

Integral of the received value.
To use it do:

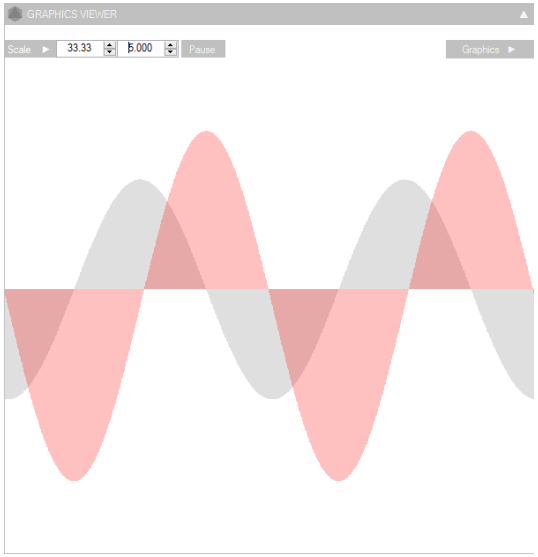
```
INT(parameter1)
```

parameter1 is the value we want to filter.

This allows us to go from acceleration to speed for example. Or other physical values:

▼Pounce
▼Flounce
▼Jounce
▼Jerk
▼Acceleration
▼Speed
▼Displacement
▼Absement
▼Absity
▼Abseleration
▼Abserk

Results change depending on when you start the filter.



Example of INT(VALUE)

NOISE*

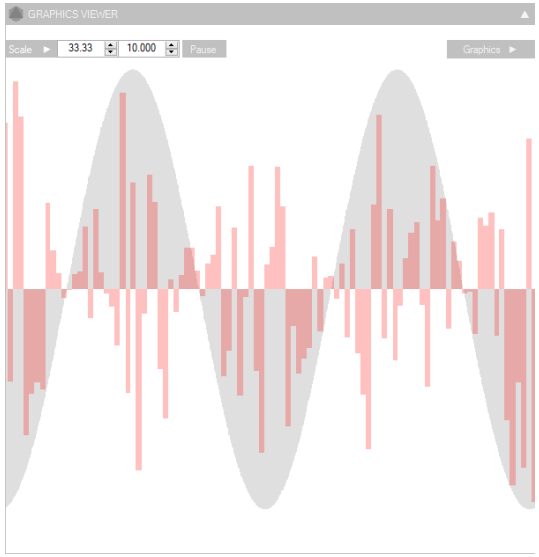
Generates noise with a maximum range of the received value
To use it do:

```
NOISE(parameter1,parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the frequency we update the noise.

The frequency is not time, but calculation cycles.

Looking at the bellow example, we can see that the generated value is inside the range given by the value. So for a value of 10, the noise is generated between -10 and +10.



Example of NOISE(VALUE;100)

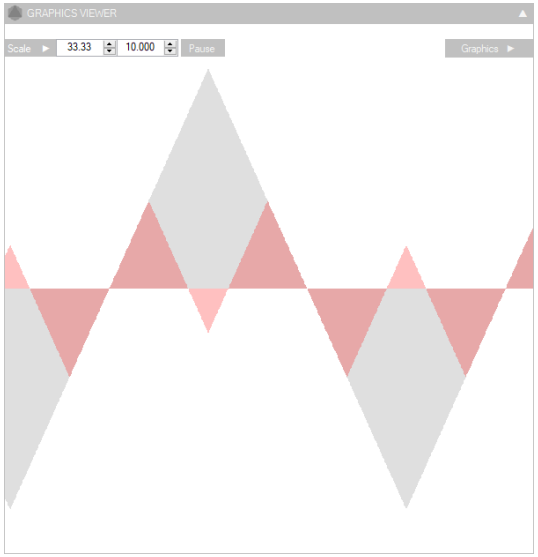
ROLLOVER

Makes a value roll between a positive/negative rollover value when the value goes above or bellow the rollover value.
To use it do:

```
ROLLOVER(parameter1;parameter2)
```

parameter1 is the value we want to filter.
parameter2 is the value where we rollover.

Used by some for aircrafts roll, to go back to zero when the rig has limited roll.



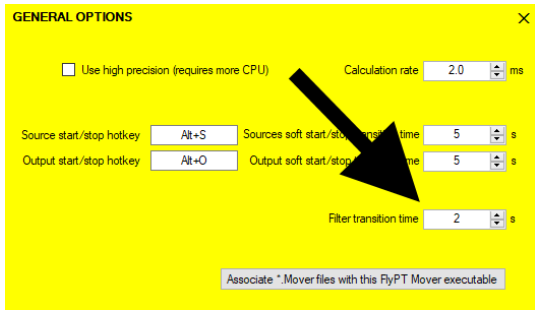
Example of ROLLOVER(VALUE;10)
In the example, the value goes from -25 to +25 in grey.
The filter result is between -10 and 10.

FILTER TRANSITION

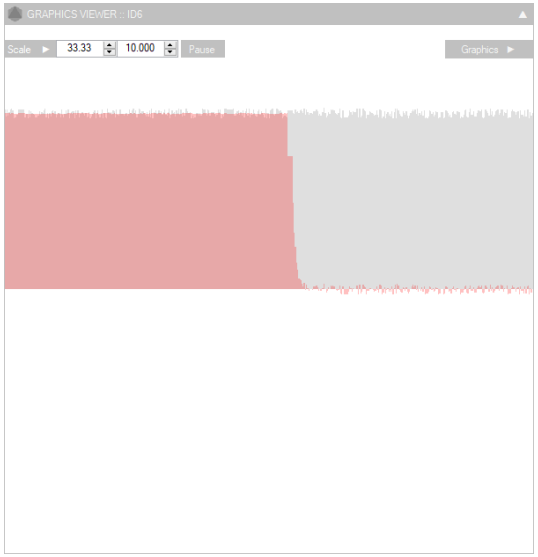
Ok, so now we know the filters, why do we have transition between two filters?

Imagine we have a low pass filter running and the current value is around 20.
If I change the filter to a high pass filter, we sudenlly change the value from 20 to around 0. This causes a sudden move on the rig.
So what the transition does is start with 100% low pass and 0% high pass and transition to 0% low pass and 100% high pass, in the selected time.
The change is not linear, it's an s curve to make it softer.

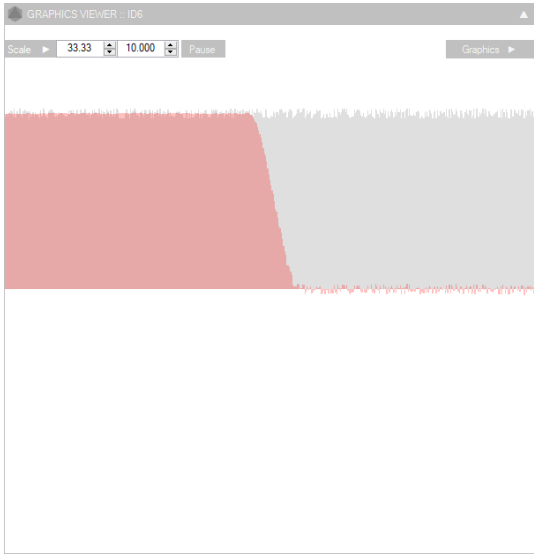
The transition time is set on the options of the main window.
Right click to open the menu and press options to get the general options panel.



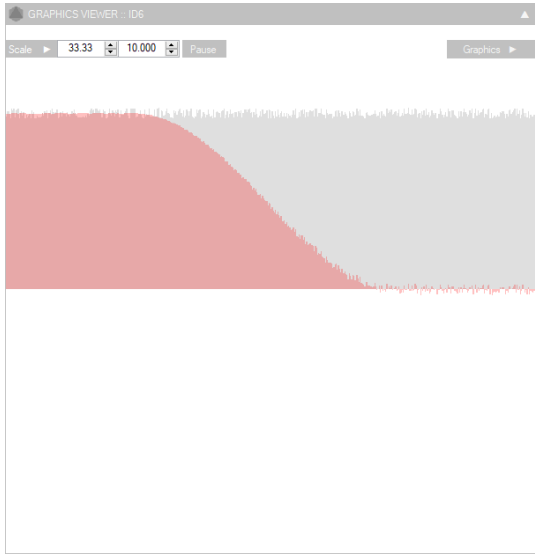
The general options panel



Transition with time = 0s
(note: the step is caused by the interface update)



Transition with time = 2s



Transition with time = 10s

PRACTICAL EXAMPLES

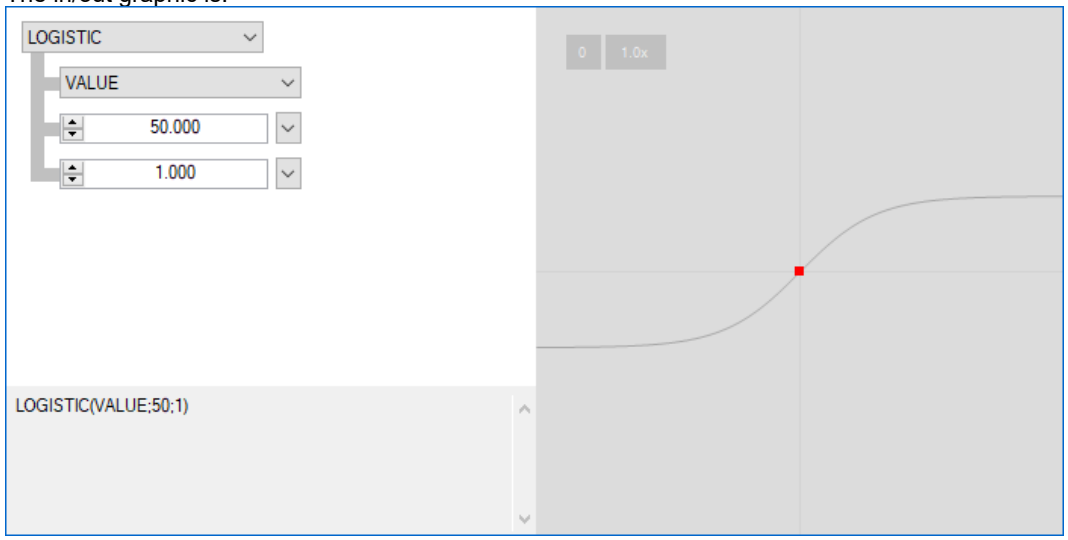
Example 1: Make actuator move inside limits and decelerate when reaching the limit.

Here's a good example to use the logistic filter.
Imagine we have an actuator with 100 mm travel, or 50 mm range.
To use the logistic to limit the actuator to the 50 mm range we can do:

LOGISTIC(VALUE;50;1)

By using this on the actuator filter, even if the value is more than 50 or less than -50 we never go beyond the -50/+50 limit.
And when reaching the limit, the actuator decelerates.

The in/out graphic is:



Example 2: Get the road noise on the suspension of a car

So we receive the suspension travel of a car in the suspension module, but we want just the noise.
To get the noise, we want just the spikes. For that we use a high pass filter:

EMAHP(VALUE;10)

With this filter we are guetting just the highest frequencies and the the values stay around zero.
So we are guetting the road noise on the suspension and not the full travel.

Example 3: Get the road noise on the suspension of a car, but without huge spikes

So in the last example, we are still getting huge spikes from the suspension, because they are high frequency, how to reduce them?
Use a logistic over the high pass:

LOGISTIC(EMAHP(VALUE;10);10;1)

With this, we get just the noise of the road and we limit the value to a range between -10 and 10.
Huge spikes are now removed.

Example 4: Heave in the rig is too violent we want it softer

To make the movement softer we can use a low pass filter:

EMALP(VALUE;150)

We are softening the values received.

Example 5: After smoothing the heave, travel is still to big althought smoother

To reduce the travel, we can use gain to lower the amplitude:

GAIN(EMALP(VALUE;150);0.5)

So we are aplying 0.5 gain to the low pass result. We get half the travel we had.

Example 6: We want surge to represent the accelerations/brakings with all the details of high frequencies, but we want to return surge to zero when the acceleration is constant

Here's a good chance to use the EMABP:

EMABP(VALUE;10;1000)

The result is that we keep the high frequency variations and the more stable/constant values are washed, making the surge go back to zero.

Example 7: Since in example 6 we go back to zero in surge, let's use the constant values of longitudinal acceleration in pitch.

So we want to replace constant accelerations with pitch to make us feel the pressure in the back.
So since we are filtering surge with an EMABP, let's use a EMABS to get the values we bin in the surge. So what we remove to washout is in the EMABS the opposite to EMABP:

EMABS(VALUE;10;1000)

Example 7: Ok, but we still have some high frequency in the pitch and it feels strange

Ok, an EMALP over the EMABS:

EMALP(EMABS(VALUE;10;1000);75)

And now we have a soft pitch to replace constant accelerations.

Example 8: How to get traction loss

To get traction loss, we should use rotation speed as the source value
Use a CUBIC3 to allow for a deadzone around zero. What this means is that for small speed rotations, the rig rotation is low or zero.
But if the car starts to rotate fast due to traction loss, we go out of the deadzone and the rotations increases exponentially.
So we would use :

CUBIC3(VALUE;0.001)

Lower the value, greater the deadzone.
But we can also limit the rotation for example to 15º. For that we use a LOGISTIC:

LOGISTIC(CUBIC3(VALUE;0.001);15;1)

The input output graphic is going to be: