A Simple Low-Pass Filter Example

Wyatt Newman
October, 2014

Sensor signals can be noisy, motivating filtering. This is particularly true of sensor values in drcsim for the wrist and feet force/torque sensors, as well as the joint effort sensors.

An example of implementing a simple, 1st-order low-pass filter appears in the class code under …/examples/example_filter. The node "foot_wrench_filter" subscribes to the robot's sensor values on the atlas/atlas_state topic. From this topic, only the r_foot and l_foot force/torque values are examined. These are modified within the callback by the following example line of code:

```
g_r_foot[2] = (1 - a_factor) *g_r_foot[2]+  a_factor * states.r_foot.force.z;
```

This expression is equivalent to:
x(t+dt) = (1-a)*x(t) + a*u(t+dt)
where x(t) is the filtered output, being stimulated by input (sensor samples) u(t). The value of "dt" is determined by the callback update rate, which will synchronize with the 1kHz publication rate of atlas/atlas_state.

Rearranging algebraically, we have:

$$x(t+\Delta t)-x(t)=a(u(t+\Delta t)-x(t))$$

or,

$$\frac{\Delta x}{\Delta t}=\frac{a}{\Delta t}(u(t+\Delta t)-x(t))$$

Treating delta-t in the the above as small, we have an (approximate) ordinary, linear differential equation. To visualize the solution, consider a step input for u(t). If u(0) is u0, and if the initial condition is x(0) = x0, then the solution to the differential equation is:

$$x(t)=u_0-(u_0-x_0)e^{-(a/\Delta t)t}$$

We can see that x(t) approaches u0 with an error that decays exponentially with time constant $\tau=\frac{\Delta t}{a}$. Because the exponent is negative, the error converges to zero. However, this does constrain choices of the parameter "a" to positive values (else the algorithm will not converge).

There is also a limit on the maximum size of "a". As a approaches unity, the algorithm degenerates to simply:
x(t+dt) = (1-a)*x(t) + a*u(t) = u(t).

That is, there is no filtering for a=1. Rather, the raw samples, u(t), are copied identically to the output x(t).

We can interpret the time constant in terms of a low-pass filter cut-off frequency:

$$\omega_{lpf} = 1/\tau$$

For input frequencies greater than this low-pass filter cut-off frequency, the input oscillations will be attenuated in the output (filtered) signal. For input oscillations well below this frequency, the output will contain the unattenuated stimulus.

In terms of visualizing the influence of this filter, a low-frequency cut-off filter (i.e., small value of a) performs heavy filtering, resulting in very clean outputs. However, this also means that the filtered output is slow to respond to sudden stimuli. If the low-pass filter is too slow, then it introduce difficulties in feedback. Equivalently, a feedback signal that is "late" can result in a corrective action that is inappropriate (e.g. in the wrong direction), ultimately leading to instability. Choosing a low-pass filter cut-off frequency should be done in coordination with choosing feedback gains.

The example code in the example_filter package, node foot_wrench_filter, performs low-pass filtering on the foot force/torque signals. The result is republished on the topics: r_foot_filtered and l_foot_filtered. The influence of filtering can be seen by plotting the "raw" signals and the filtered signals on the same plot, e.g. using:

rqt_plot atlas/atlas_state/r_foot/force/z /r_foot_filtered/wrench/force/z

or:

rqt_plot atlas/atlas_state/r_foot/torque/y /r_foot_filtered/wrench/torque/y

In the source code, the values a_factor and b_factor may be changed to see the influence of these terms on the low-pass filtered result.