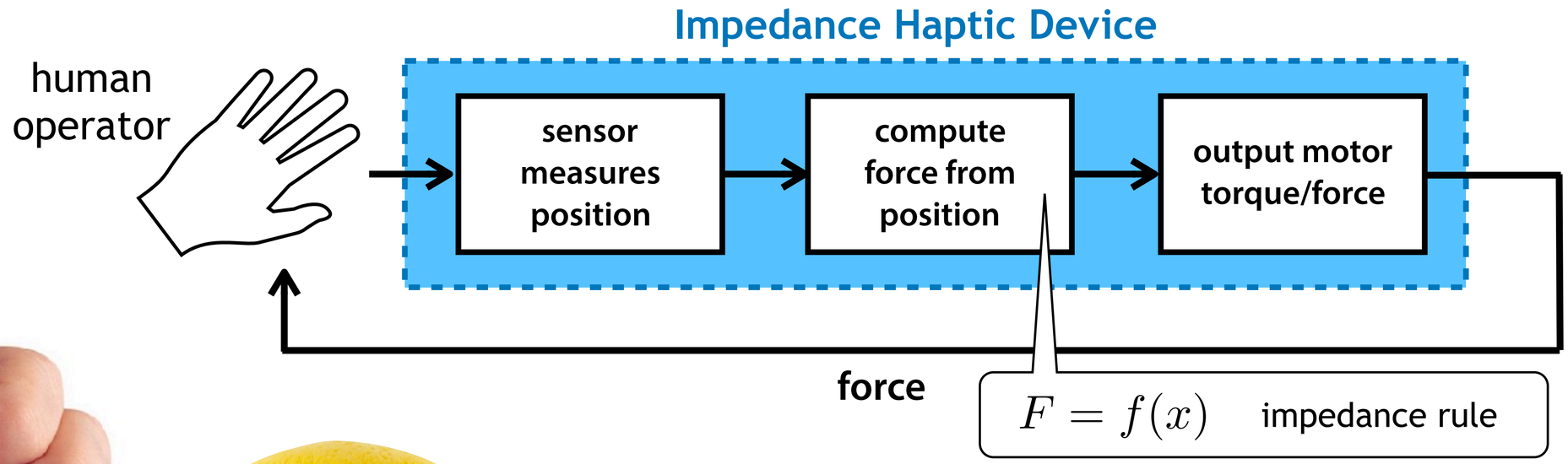
Estimating derivatives

ENGR E399/599
Prof Eatai Roth

Beyond springs



Technically, impedance devices compute force outputs from motion. That is, an impedance law may use position and its derivatives.

How can we mathematically describe the difference in sensation between a sponge and a racquetball? Is springiness enough?

Damping

$$\underbrace{kx}_{\text{spring}} + \underbrace{b\dot{x}}_{\text{damping}} + \underbrace{m\ddot{x}}_{\text{inertia}} = F$$

Whereas the spring term represents a force that is proportional to displacement, the damping term is a force that opposes motion.

A damping term that is linearly proportional to velocity is called *viscous damping*. This name alludes to the viscosity (thickness) of a fluid. Imagine dragging your finger through honey; what are the forces you feel?

Other kinds of damping are not necessarily linear. For example, aerodynamic drag is proportional to squared velocity.

Inertia

$$\underbrace{kx}_{\text{spring}} + \underbrace{b\dot{x}}_{\text{damping}} + \underbrace{m\ddot{x}}_{\text{inertia}} = F$$

Objects with mass have inertia. If an object is at rest it will stay at rest and if it's moving, it will continue to move with the same velocity. Unless an external force acts upon it.

The Hapkit naturally has damping from the friction in the bearing and in the motor. It also has inertia, both from the paddle and the motor.

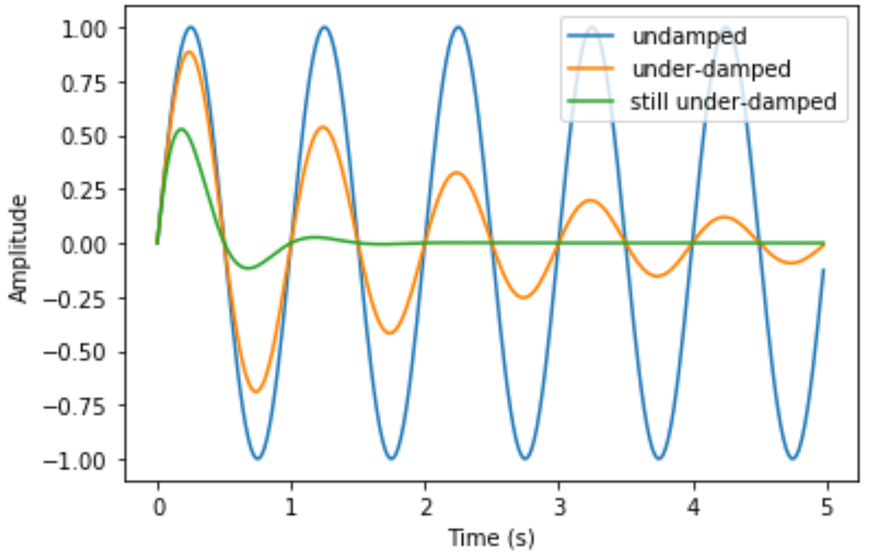
But we can change the apparent damping and inertia.

Note: the Hapkit is not naturally springy. The springiness is entirely simulated by the impedance law.

Why damping?

$$\underbrace{kx}_{\text{spring}} + \underbrace{b\dot{x}}_{\text{damping}} + \underbrace{m\ddot{x}}_{\text{inertia}} = F$$

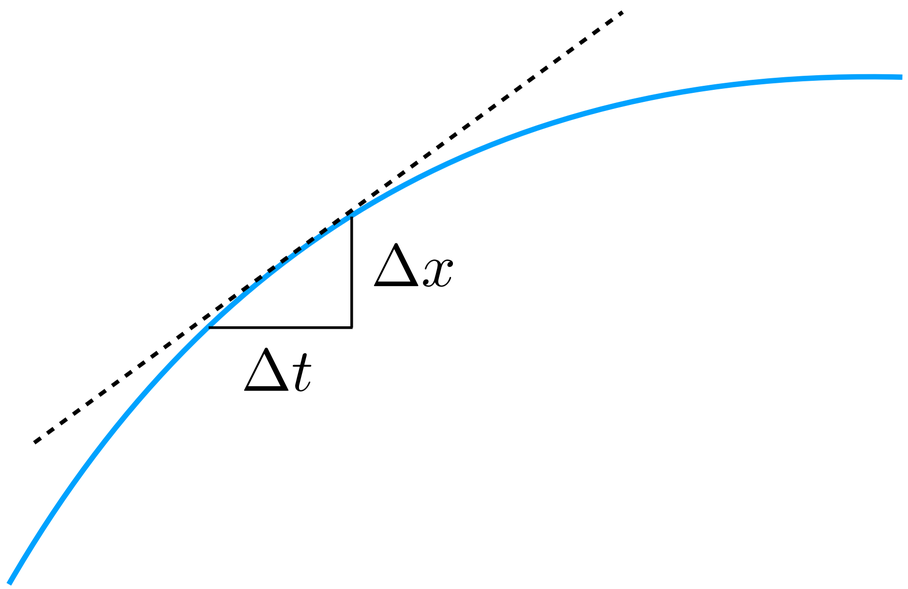
If a system is springy but undamped (no damping) or under-damped (not “enough” damping) it will oscillate.



We can introduce a damping term to mitigate some unwanted oscillations/vibrations in our haptic rendering.

Estimating derivatives of sampled data

$$\underbrace{kx}_{\text{spring}} + \underbrace{b\dot{x}}_{\text{damping}} + \underbrace{m\ddot{x}}_{\text{inertia}} = F$$



If we want to simulate damping or inertia, we'll need to estimate the derivatives of x (velocity and acceleration).

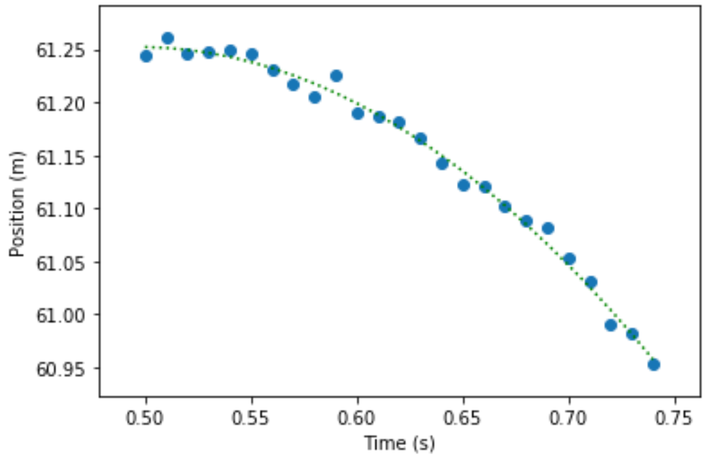
$$\dot{x} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

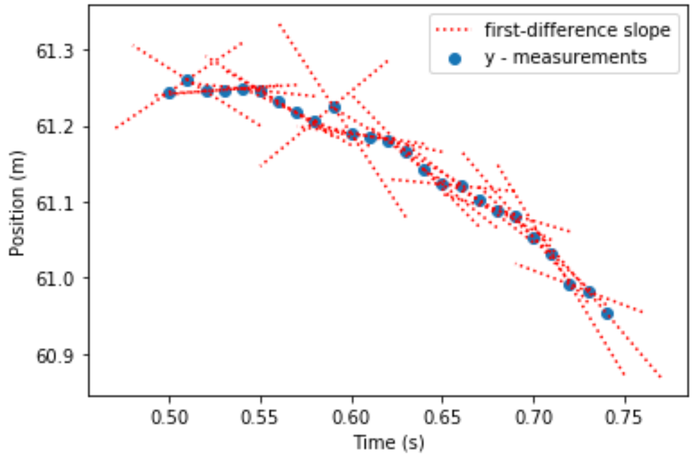
But, in reality we'll have something more like:

$$\dot{x}_{est} \approx \frac{(x(t) + \epsilon) - (x(t - \Delta t) + \epsilon)}{\Delta t + \delta}$$

Taking the derivative amplifies small errors.

Estimating derivatives of sampled data





If we want to simulate damping or inertia, we'll need to estimate the derivatives of x (velocity and acceleration).

$$\dot{x} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

But, in reality we'll have something more like:

$$\dot{x}_{est} \approx \frac{(x(t) + \epsilon) - (x(t - \Delta t) + \epsilon)}{\Delta t + \delta}$$

Taking the derivative amplifies small errors.

Low-pass filtering

One approach to dealing with noisy data is ***low-pass filtering***.

Low-pass filters smooth the data by keeping low frequency content (slow changes) while attenuating high frequency content (rapid changes).

Discrete exponential first-order low-pass filter

Suppose we want to filter a noisy measured signal x[k] to make a smoother (de-noised) signal estimate y[k]:

$$y[k] = \alpha \cdot x[k] + (1 - \alpha) \cdot y[k-1], \qquad 0 < \alpha < 1$$

Each new value of y[k] depends some on the new measurement x[k] and the last estimate y[k-1].

Low-pass filtering

Discrete exponential first-order low-pass filter

$$y[k] = \alpha \cdot x[k] + (1 - \alpha) \cdot y[k-1]$$

$$y[k] = \alpha(x[k] + (1 - \alpha) \cdot x[k-1] + (1 - \alpha)^2 \cdot x[k-2] + \cdots))$$

Alpha is called the smoothing factor. The smaller the value of alpha, the smoother the signal. Here is an example with alpha = 0.1.

$$y[k] = 0.1 \cdot x[k] + 0.09 \cdot x[k-1] + 0.081 \cdot x[k-2] + \cdots)$$

But there is a trade-off. As we decrease alpha, the output will be less responsive to fluctuations of input and will lag the true signal (like a delay).

Low-pass filtering in Z-Domain

Discrete exponential first-order low-pass filter

$$Y(z) = \frac{\alpha}{1-(1-\alpha)z^{-1}}X(z)$$

$$y[k] = \alpha \cdot x[k] + (1-\alpha) \cdot y[k-1]$$

Discrete exponential second-order low-pass filter

If we apply the first-order filter twice, we get a second-order filter. It's more smooth with a sharper frequency cut-off, but even more laggy.

$$Y(z) = \left(\frac{\alpha}{1-(1-\alpha)z^{-1}}\right)^2 X(z)$$

$$y[k] = \alpha^2 \cdot x[k] + 2 \cdot (1-\alpha) \cdot y[k-1] + (1-\alpha)^2 \cdot y[k-2]$$