Discrete Differentiation

Nicholas Paine

5/2/12

Problem

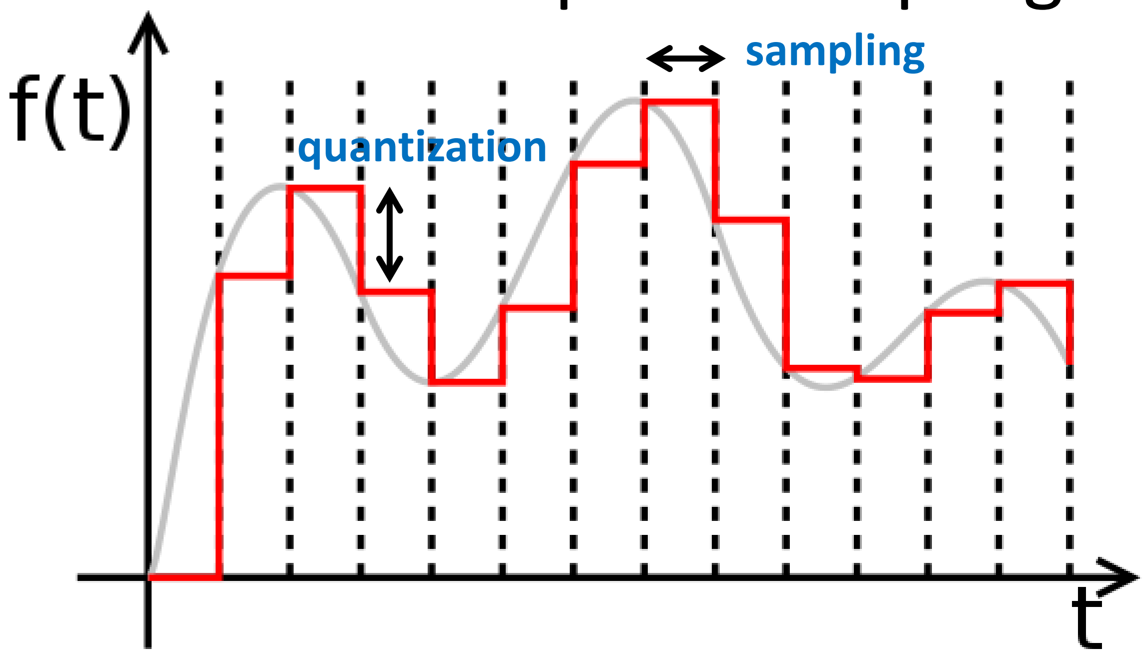
- Robots use encoders to measure joint angles
- Encoders measure angles (position)
- What if we want to know velocities or accelerations?

Common Approaches

1. Use velocity/acceleration sensors
   – Requires lots of sensors which is complicated and expensive
2. Perform **simple** signal processing to approximate time derivatives
   – In my experience this is what most people do
   – Simple and inexpensive
   – Noisy, especially after 1st derivative

Digital Measurements

Digital measurements have error from both quantization and temporal sampling



Both lead to “stair step” waveforms

Simple Differentiation Technique

A velocity approximation can be obtained as follows:

$$v[n] = \frac{x[n] - x[n-1]}{T_s}$$

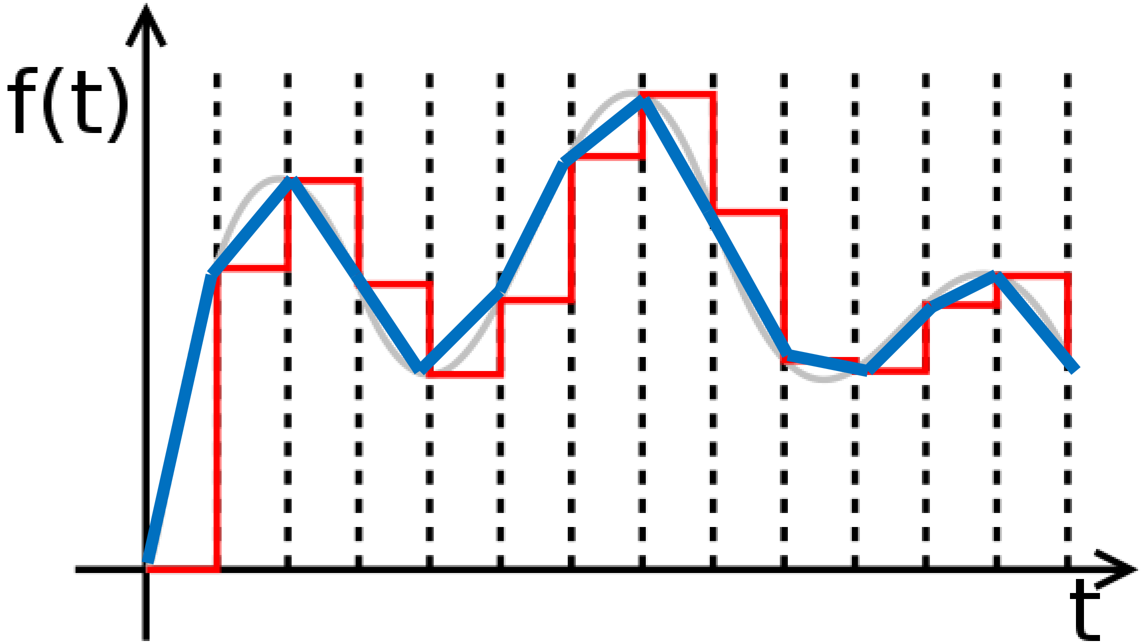
The same applies for acceleration:

$$a[n] = \frac{v[n] - v[n-1]}{T_s}$$

Any problem with this?

Simple Differentiation Technique

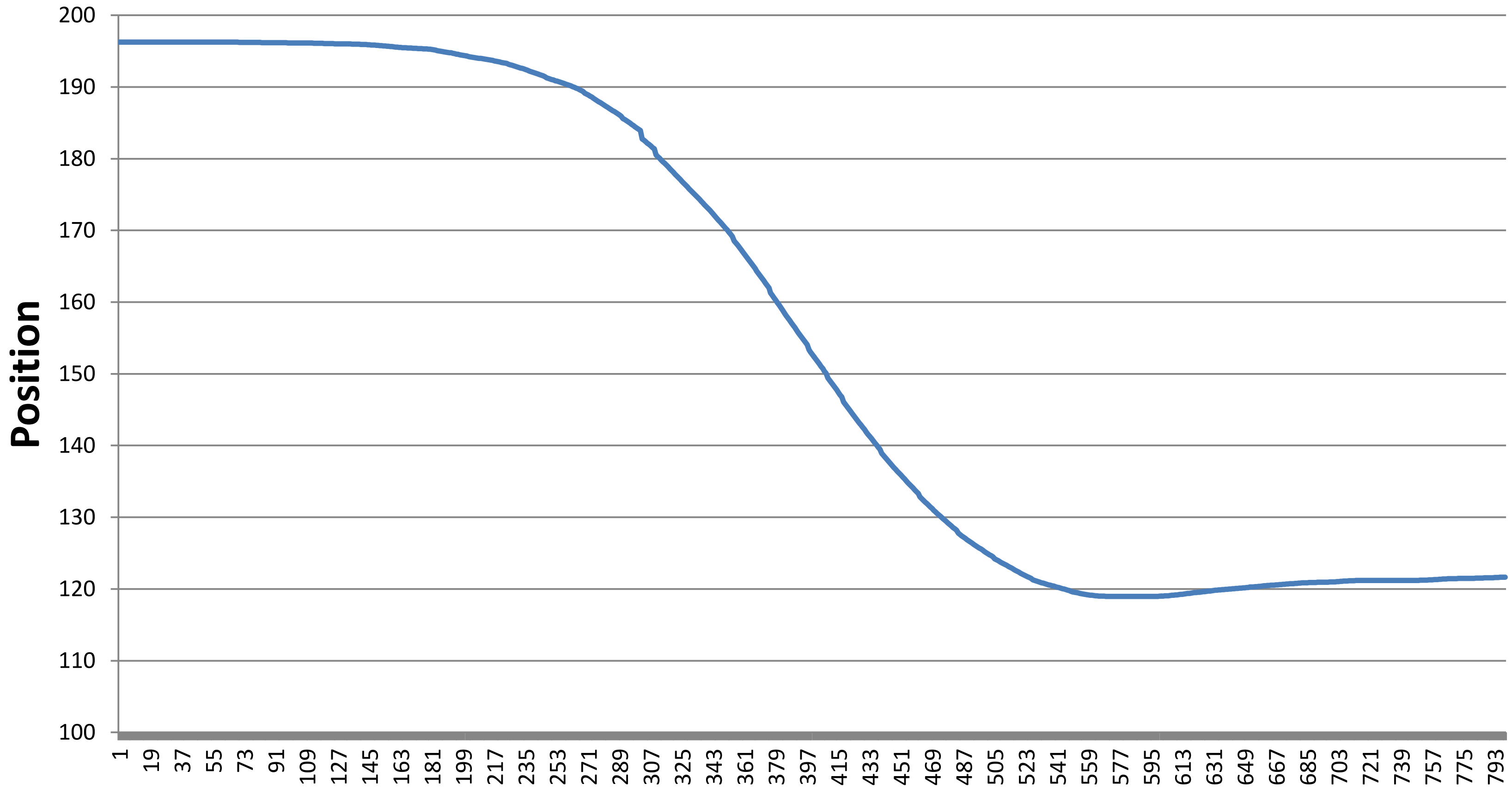
Quantization and temporal sampling lead to instantaneous changes in measured position



Which lead to instantaneous changes in velocity and acceleration

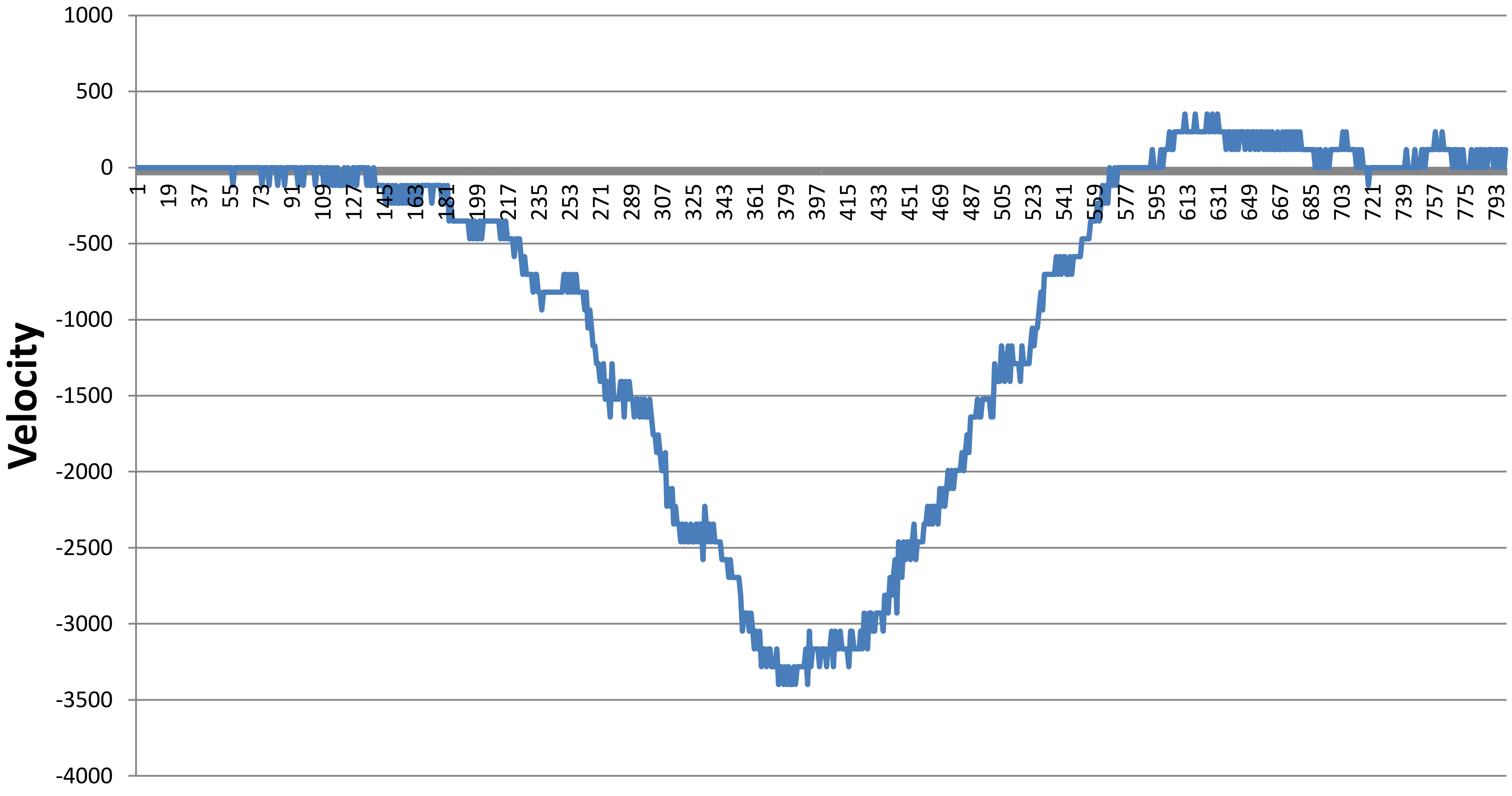
Simple Differentiation Technique

What does this actually look like?



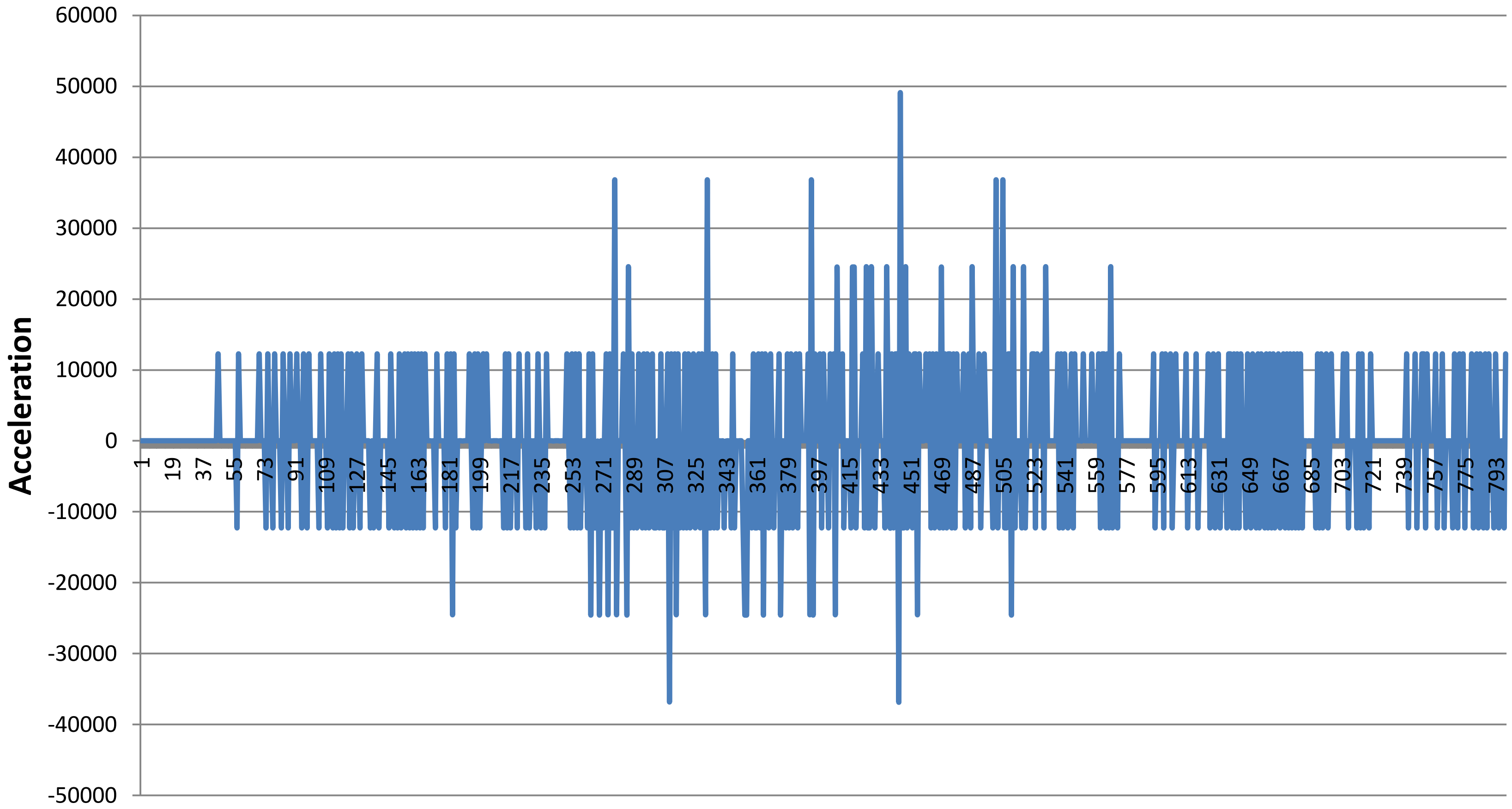
Simple Differentiation Technique

What does this actually look like?



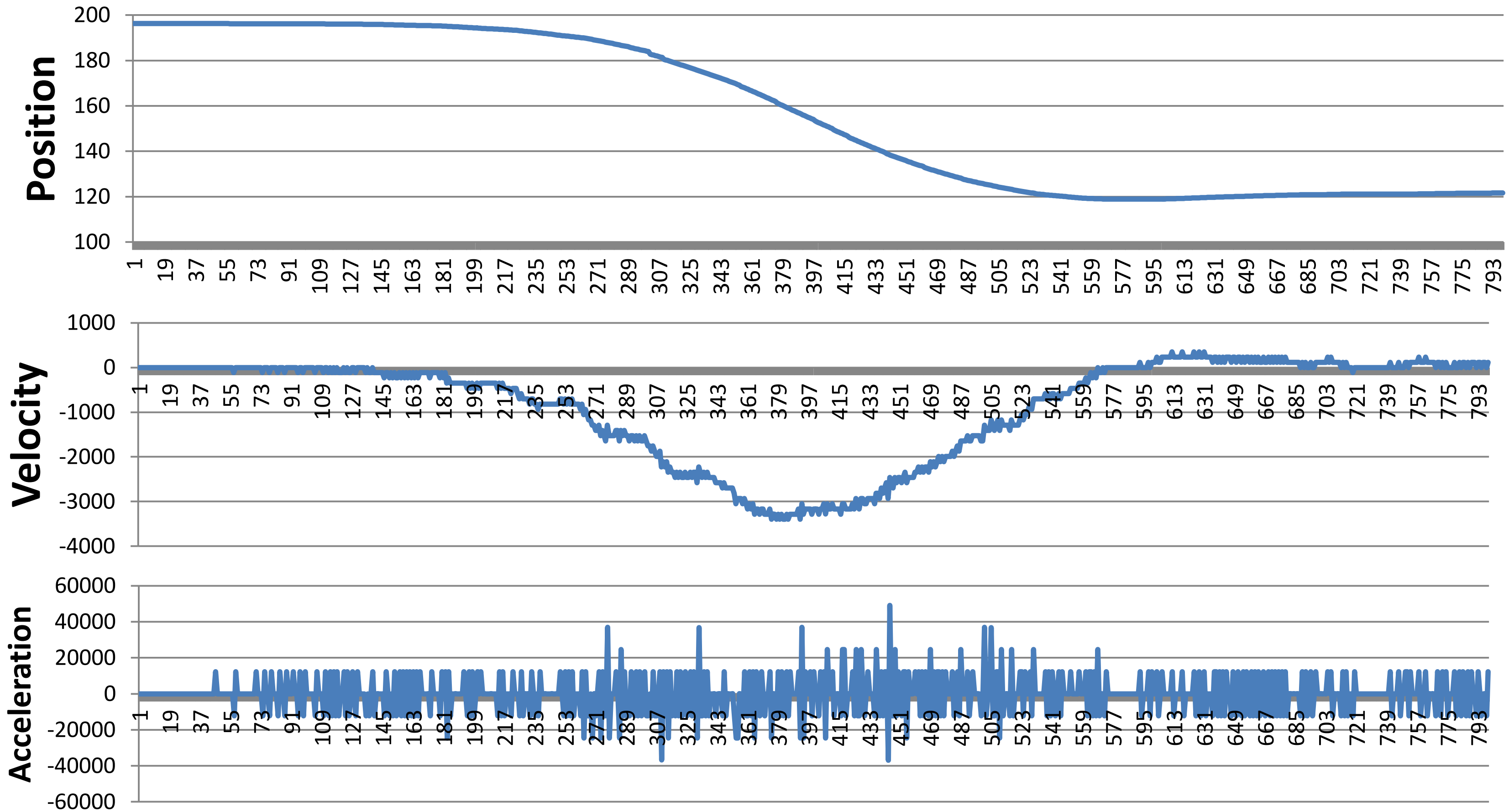
Simple Differentiation Technique

What does this actually look like?



Simple Differentiation Technique

What does this actually look like?



The ‘s’ operator

- ‘s’ is the continuous time derivative operator
- We approximated this in discrete time with the following:

$$s \leftrightarrow \frac{x(n) - x(n-1)}{T_s} \leftrightarrow \frac{1 - z^{-1}}{T_s}$$

- Where x(n) is the current sample, and x(n-1) is the previous sample and T_s is the sample period

Tustin Transform

- The Tustin Transform improves on our simple approximation by dividing by an averaging term

$$s \leftrightarrow \frac{1 - z^{-1}}{T_s} \cdot \frac{2}{1 + z^{-1}}$$

- The Tustin Transform does a better job of preserving the transfer function than our simple method

Velocity transfer function

- If we use the s operator with a low pass filter, we can produce a higher quality time derivative signal

$$\frac{v(s)}{x(s)} = \frac{s}{\left({s}/{\omega_c} \right)^2 + 1.4142\left({s}/{\omega_c} \right) + 1}$$

Acceleration transfer function

- We can do the same thing for acceleration

$$\frac{a(s)}{x(s)} = \frac{s^2}{\left(s/\omega_c \right)^2 + 1.4142\left(s/\omega_c \right) + 1}$$

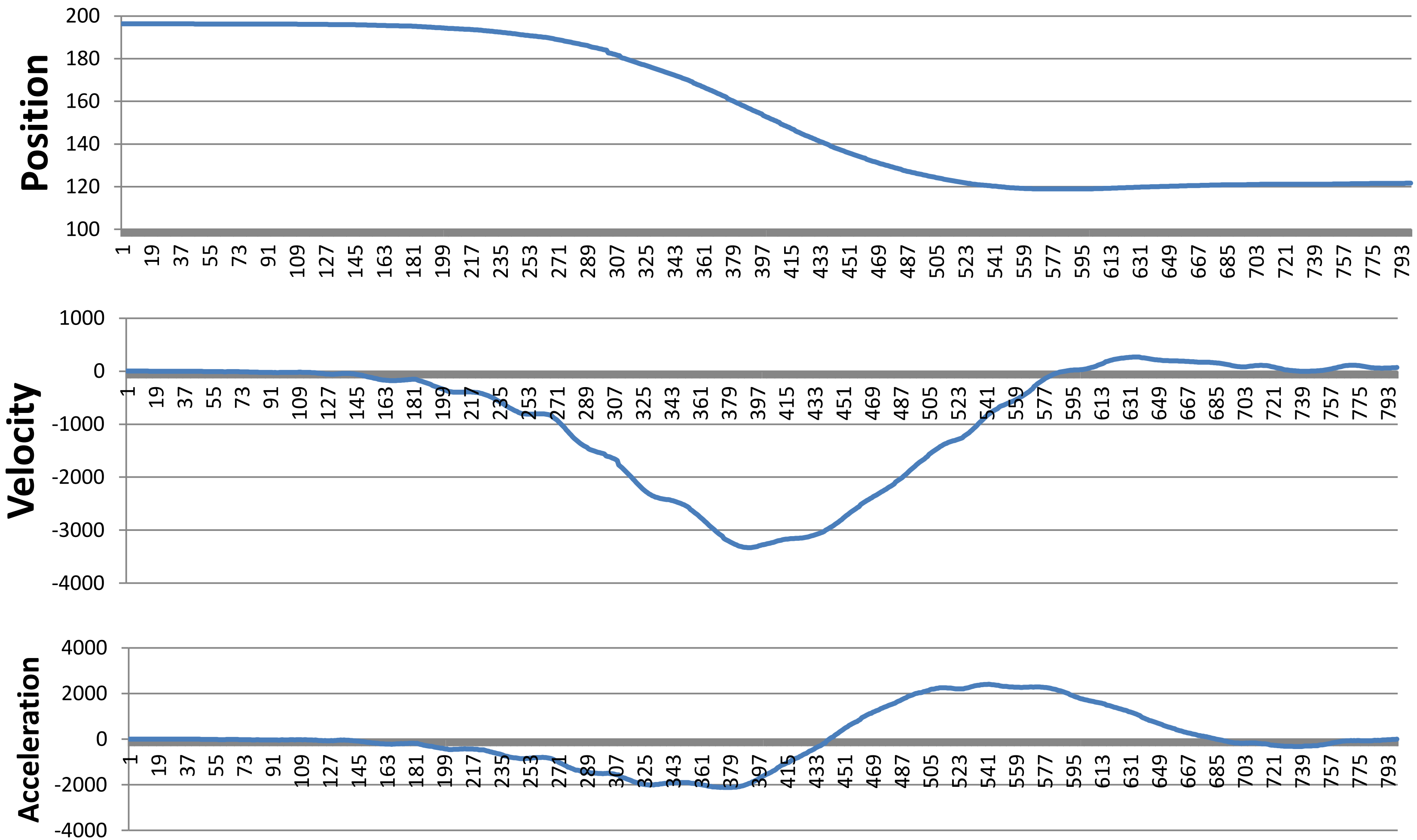
Discretization

- Both of these transfer functions can be discretized using the Tustin Transform so that they may be implemented on a computer as a digital filter
- See appendix

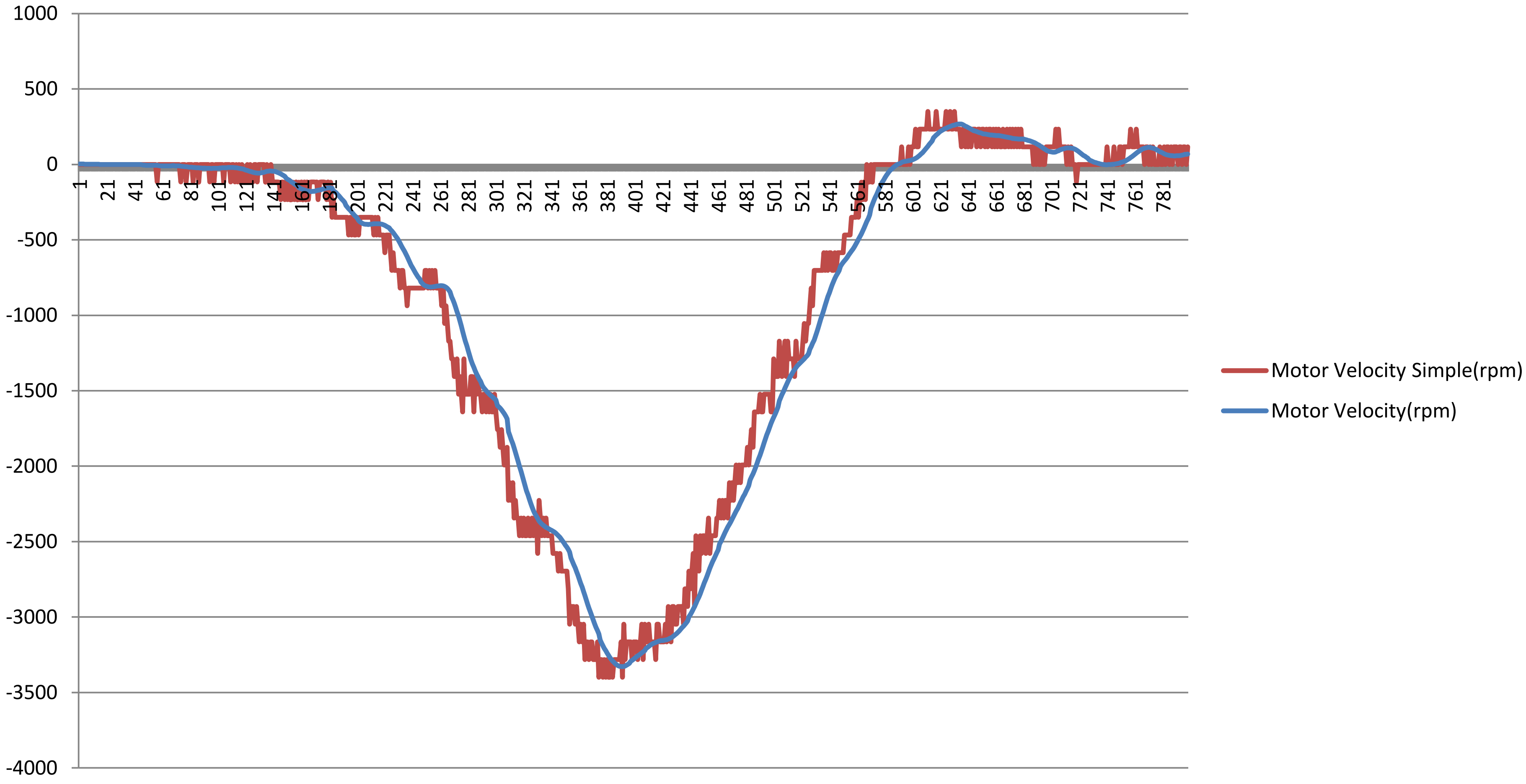
Matlab code

```
%generates digital filter coefficients for acceleration calculation

clear all
format compact;
format long g;

A_fc = 5; %LPF cutoff freq (hz)
Ts = 0.001; %sampling period

[an,ad] = butter(2,A_fc*2*pi,'s'); %generate LPF coeffs
Q_A = tf(an,ad); %LPF tfr function
Afilt = tf([1 0 0],1) * Q_A; %multiply by s^2

%%print digital filter coefficients
Hd = c2d(Afilt,Ts,'tustin');
[num,den] = tfdata(Hd);
num = cell2mat(num);
den = cell2mat(den);
for i = 1:length(num)
    inCoeffs(i) = num(i);
end
for i = 2:length(den)
    outCoeffs(i-1) = -den(i);
end
disp('input coefficients * (1 n-1 n-2 ... n-m)')
inCoeffs
disp('output coefficients * (n-1 n-2 ... n-m)')
outCoeffs
disp('a(n) = IN_COEFF1*x(n) + IN_COEFF2*x(n-1) + IN_COEFF3*x(n-2) + OUT_COEFF1*a(n-1) + OUT_COEFF2*a(n-2)')
```

Tustin/LPF Technique



Velocity Comparison



Appendix

- Output of the Tustin Transform is of the form:

$$P(z) = \frac{az^2 + bz + c}{z^2 + dz + e}$$

Which is just a discrete time transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{az^2 + bz + c}{z^2 + dz + e}$$

Appendix

$$\frac{Y(z)}{X(z)} = \frac{az^2 + bz + c}{z^2 + dz + e}$$

Cross multiplying yields:

$$Y(z)(z^2 + dz + e) = X(z)(az^2 + bz + c)$$

Then dividing by z^2 and rearranging:

$$Y(z) = X(z)(a + bz^{-1} + cz^{-2}) - Y(z)(dz^{-1} + ez^{-2})$$

Appendix

$$Y(z) = X(z)(a + bz^{-1} + cz^{-2}) - Y(z)(dz^{-1} + ez^{-2})$$

- This equation represents what is called a digital filter (an IIR filter to be precise)
- The output (Y) is a function of the current and previous inputs (X) and of the previous outputs
- Remember the z^-1 operator represents the previous sample
- We can write code for this

Appendix

$$Y(z) = X(z)(a + bz^{-1} + cz^{-2}) - Y(z)(dz^{-1} + ez^{-2})$$

Code:

```
y[0] = a*x[0] + b*x[1] + c*x[2]
     - d*y[1] – e*y[2];
x[2] = x[1];
x[1] = x[0];
y[2] = y[1];
y[1] = y[0]
return y[0];
```