Continuous vs discrete-time Kalman filter, approximations

© 2026, Antonio Sala Piqueras, Universitat Politecnica de Valencia, Spain. All rights reserved.

Presentations in video [English]

https://personales.upv.es/asala/YT/V/medex2EN.html [continuous-time 2nd-order and discrete-time

3rd-order Kalman filters, comparison, finite difference estimation variance]

https://personales.upv.es/asala/YT/V/medex3redEN.html [2nd-order discrete-time Kalman filters]

Presentaciones en vídeo [Español]

https://personales.upv.es/asala/YT/V/medex2.html [fliltro Kalman continuo (orden 2), discreto (orden 3)]

https://personales.upv.es/asala/YT/V/medex3red.html [filtro Kalman discreto orden 2]

Objective: understanding the "low-level sensing" behaviour, comparing a continuous-time Kalman Filter with a discrete-time one assuming measurement comes as $y_{meas} = \frac{1}{T_s}\int_0^{T_s}(y(\tau) + measurementnoise)\, d\tau$ with some PSD for the measurement noise.

Table of Contents

Running prior code, to get variables in workspace....1
Comparison with discrete-time Kalman filter....3
Converging to the continuous-time Kalman filter....4
Approximating to a discrete-time Kalman for 2nd-order process....4
Approximation 1....5
Approximation 2....5
Acquisition period NOT equal to sampling period?....6

Running prior code, to get variables in workspace

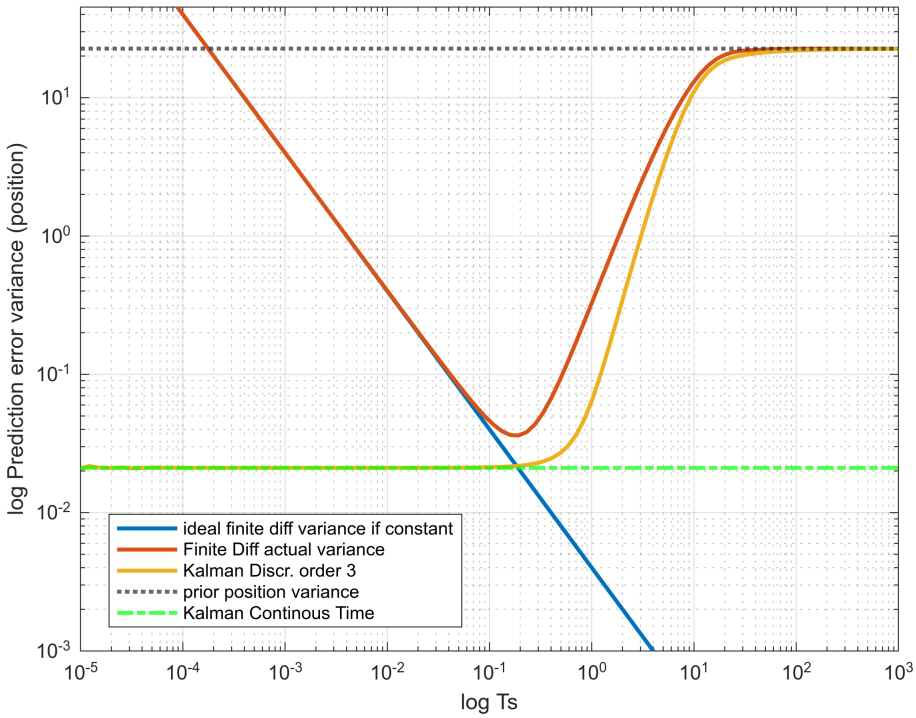
```
run("SensorTSTKlmVsDifFinPart1.mlx")
```

```
ans = 2×1 complex
  -0.1768 + 0.1768i
  -0.1768 - 0.1768i
ans =

            1
  ------------------------
  (s^2 + 0.3535s + 0.0625)
```

```
Continuous-time zero/pole/gain model.
Model Properties
ans = 1×3 complex
  0.999999960000001 + 0.000000000000000i  0.999646450012332 - 0.000353431795885i ···
Wd = 3×3
   0.000008000992239   0.000000000002032   0.000000001332388
   0.000000000002032   0.000000002665256   0.000001998586216
   0.000000001332388   0.000001998586216   0.001998586299899
ans = 3×3
   0.000008000000000                   0                   0
                   0                   0                   0
                   0                   0   0.002000000000000
P = 3×3
10^6  ×
    6.3997    0.0001   -0.0000
    0.0001    0.0000         0
   -0.0000         0    0.0000
ans = 2×2
   22.6276         0
         0    1.4142
Pred = 2×2
   22.6276         0
         0    1.4142
bigP = 6×6
10^6  ×
    6.3997    0.0001   -0.0000    6.3997    0.0001   -0.0000
    0.0001    0.0000         0    0.0001    0.0000   -0.0000
   -0.0000         0    0.0000   -0.0000    0.0000    0.0000
    6.3997    0.0001   -0.0000    6.3997    0.0001   -0.0000
    0.0001    0.0000    0.0000    0.0001    0.0000         0
   -0.0000   -0.0000    0.0000   -0.0000         0    0.0000
FinDiff = 1×2
  -500   500
ans = 2.0028
z2disc = 3×3
    0.0000    0.0000    0.0000
    0.0000    0.0211    0.0555
    0.0000    0.0555    0.3137
ans = 0.3348
L = 3×1
    1.0105
    5.2834
   13.7963
z2x = 3×3
    0.0000    0.0000    0.0000
    0.0000    0.0211    0.0555
    0.0000    0.0555    0.3137
Lc3 = 3×1
10^5  ×
    0.6325
    3.3331
    8.7821
Lc2 = 2×1
    5.2699
   13.8858
z2reduced = 2×2
    0.0211    0.0555
    0.0555    0.3137
ans = 0.3347
Warning: The [G 0;H I]*[Qn Nn;Nn' Rn]*[G 0;H I]' matrix should be positive semi-definite. Type
"help kalman" for more information.
Warning: The [G 0;H I]*[Qn Nn;Nn' Rn]*[G 0;H I]' matrix should be positive semi-definite. Type
"help kalman" for more information.
Warning: The [G 0;H I]*[Qn Nn;Nn' Rn]*[G 0;H I]' matrix should be positive semi-definite. Type
"help kalman" for more information.
```

```
Tsbest = 0.1918
```



Comparison with discrete-time Kalman filter

We assume we "measure" the integrator state with basically "zero" variance every Ts:

```
sysd=ss(Ad,eye(3),[1 0 0],0,Ts);
[~,L,~,~,z2disc]=kalman(sysd,Wd,1e-12);
```

The posterior error variance of the discrete-time Kalman filter is:

```
format long
z2disc
```

```
z2disc = 3×3
   0.000000000001000   0.000000000005256   0.000000000013807
   0.000000000005256   0.021081532560776   0.055546856666116
   0.000000000013807   0.055546856666116   0.313671634407982
```

```
trace(z2disc)
```

```
ans =
   0.334753166969758
```

```
format short
```

The discrete-time Kalman filter gain is:

```
L
```

```
L = 3×1
    1.0105
    5.2834
```

```
    13.7963
```

Converging to the continuous-time Kalman filter

The continuous-time Kalman filter with position sensor contaminated with, of course, white noise with same PSD, entering to other "electronics" given by the KF instead of "averaging integrator":

```
[~,Lc2,z2reduced]=kalman(sys_reduced,W_process,W_ad_sensor);
Lc2
```

```
Lc2 = 2×1
    5.2699
   13.8858
```

```
format long
z2reduced
```

```
z2reduced = 2×2
   0.021079540020250   0.055543375933163
   0.055543375933163   0.313662035773152
```

```
trace(z2reduced)
```

```
ans =
   0.334741575793402
```

```
format short
```

Approximating to a discrete-time Kalman for 2nd-order process

We might think on the 2nd-order process, where I measure position... well, position sort of "in the middle of the integration interval", say position-velocity*Ts/2. As we are integrating, we are "measuring" $p(T_s/2)\cdot T_s$, sort of. It is equivalent to measure position with $T_s/2$ delay, of course, multiplying noise standard deviation times $1/T_s$, i.e., multiplying variance by $1/T_s^2$.

Measurement noise also adds up, with Bd(1,1). Of course, this is "approximate": the true sensing dynamics is in the 3rd-order process. There is correlation between process and measurement noise in Bd.

As $Bd(1,1)\approx PSD_{measurement\ noise}\cdot T_s$, for small Ts, we get measurement noise variance $\approx PSD_{measurement\ noise}\cdot\frac{1}{T_s}$, which appears in textbooks and "accounting for half-a-sample delay" as a rule of thumb for controller design, i.e., "add $(1-\frac{T_s}{2}s)$ to the transfer function".

These rules of thumb stem from the "precise" computations of the augmented 3rd-order process, and apply, in principle, only when this integrating-type sensing is active. *Well, ZOH discretization in control also gives rise to a similar rule of thumb, unrelated to the topics under discussion here.

Approximation 1

The 1st approximation from our augmented system discretization is:

```
A_d_reduced=expm(Areduced*Ts)
```

```
A_d_reduced = 2×2
    1.0000    0.0020
   -0.0001    0.9993
```

```
sysd2=ss(A_d_reduced,eye(2),[1 -Ts/2]*Ts,0,Ts);
size(sysd2)
```

```
State-space model with 1 outputs, 2 inputs, and 2 states.
```

```
format long
Wd
```

```
Wd = 3×3
   0.000008000992239   0.000000000002032   0.000000001332388
   0.000000000002032   0.000000002665256   0.000001998586216
   0.000000001332388   0.000001998586216   0.001998586299899
```

```
format short
[~,L2b,~,~,z2b]=kalman(sysd2,Wd(2:3,2:3),Wd(1,1),Wd(1,2:3)');
L2b
```

```
L2b = 2×1
    5.2834
   13.7963
```

```
z2b
```

```
z2b = 2×2
    0.0211    0.0555
    0.0555    0.3137
```

Approximation 2

A second approximation with that 1/Ts times PSD is, for small sampling period:

```
sysd2x=ss(A_d_reduced,eye(2),[1 -Ts/2],0,Ts);
W_state_continuous=Greduced*W_process*Greduced'
```

```
W_state_continuous = 2×2
     0     0
     0     1
```

```
[~,L2bx,~,~,z2bx]=...
    kalman(sysd2x,W_state_continuous*Ts,W_ad_sensor/Ts);
L2bx
```

```
L2bx = 2×1
    0.0106
    0.0276
```

```
L2bx/Ts %this compares to the above
```

```
ans = 2×1
```

```
    5.2837
   13.7972
```

```
z2bx
```

```
z2bx = 2×2
    0.0211    0.0555
    0.0555    0.3147
```

These things are approximate, but incorrect/suboptimal with the "real thing" which is assumed to be the 3rd-order system.

Acquisition period NOT equal to sampling period?

In some applications, your data acquisition may average things every T_{acq}, but you post-processing may use data every $T_s \geq T_{acq}$.

Then, you should discretize, if T_{acq} is "small" so that Euler-Maruyama approximation is good enough with:

- **MEASUREMENT NOISE** variance: `W_ad_sensor/Tacq`
- **PROCESS NOISE** variance: depends on T_s because it is independent of what happens with T_{acq}!.

If T_s is, too, small enough, use `Greduced*W_process*Greduced'*Ts`, otherwise use the exact integration `int(expm(A*t)*G*W*G'*expm(A'*t),0,Ts)`

This is why some applications change T_s but do not change the "measurement noise variance": it is assumed that T_{acq} does NOT change.