EECS192 Lecture 11
Apr. 7, 2020

Notes:

1. Progress Report due Fri 4/10 at 6 pm on bcourses (no checkpoints after C9)

2. Round 1 4/14 and Round 2 4/28 during class time. Logistics to be arranged. Submit python code and we run.
   Python 3.7 (?)

3. Quiz 5 on 4/21 on bicycle model

4. HW #3 later this week (simulation with learning of track).

Better is the enemy of the good.

Optimism:

underestimate complexity+

overestimate ability

Topics

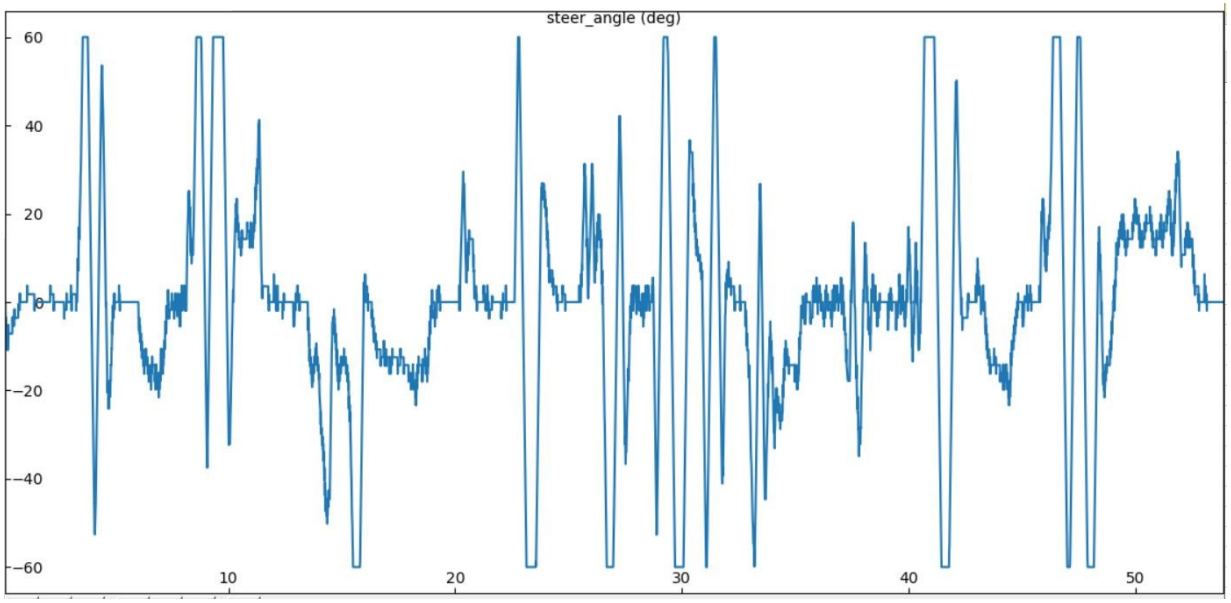
- Progress Report questions?
- HW2 notes
- Discrete Time control/timing
- Software Robustness

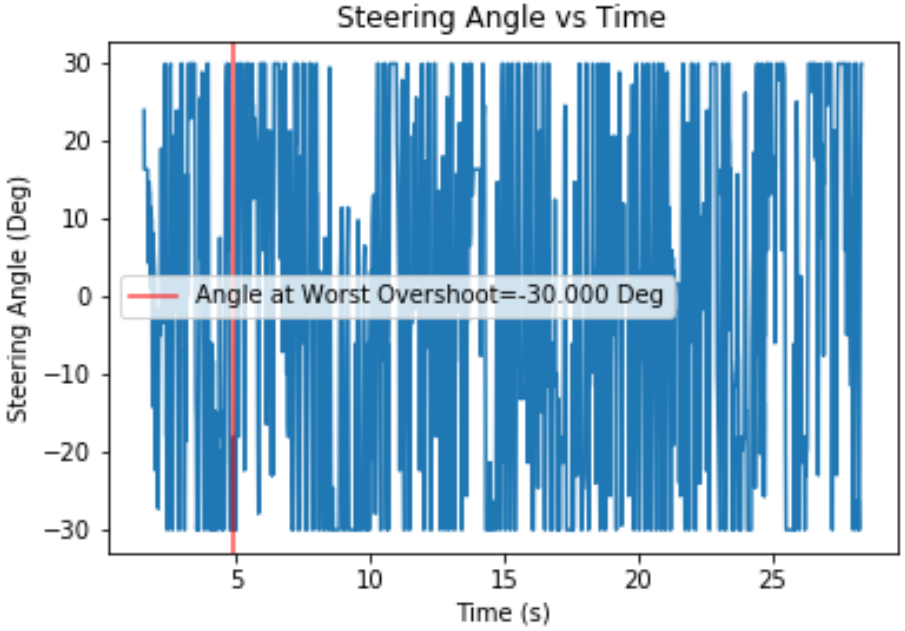
HW2 Notes- slew rate limit

Team	Speed (m/s)	Error
1	3	29 cm
2	2.7	32 cm
3	3.5	30 cm
4	3.3	30 cm
5	3	24 cm
6	2.7	27 cm
7	2.5	23 cm
8	2.7	27 cm
10	2.5	32 cm
11	2.6	13 cm

Note cone is at: 15” = 38 cm

Steering saturation





Lesson: if tracking is good, steering angle change is small

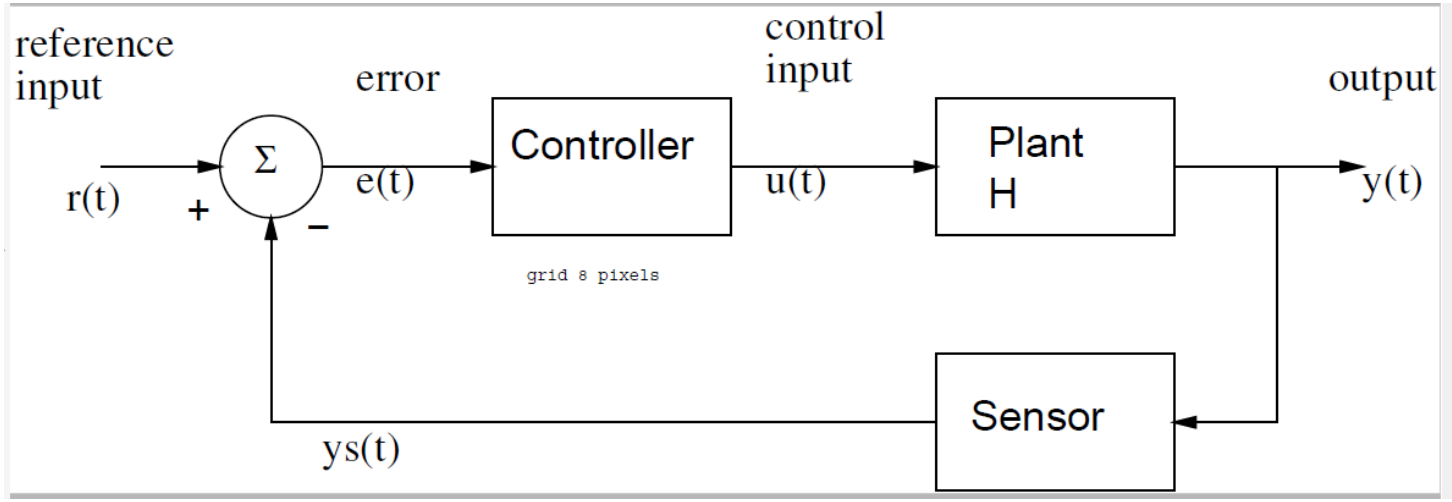
Simulation notes

- What are other line tracking errors in addition to 128 pixel quantization?
- What are some practical limits on steering control?

Topics

- Progress Report questions?
- HW2 notes
- Discrete Time control/timing
- Software Robustness

Control Synopsis



State equations: $$\dot{x}(t) = ax(t) + bu(t)$$

Output equations: $$y(t) = cx(t) + du(t)$$

Control Law (P): $$u(t) = k_p e(t) = k_p(r(t) - y(t)).$$

Control Synopsis

Control Law (P): $u(t) = k_p e(t) = k_p(r(t) - y(t)).$

New state equations:

$$\dot{x} = ax + bk_p e(t) = ax + bk_p(r - x) = (a - bk_p)x + bk_p r.$$

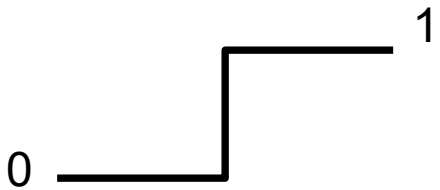
Zero Input Response (non-zero init condx, r(t)=0):

$$x(t) = x(0)e^{(a-bk_p)t} \quad for \quad t \geq 0.$$

a'=a – b k_p b' = b k_p

Total Response (non-zero init condx) by convolution:

$$x(t_o) = e^{a't_o}x(0) + \int_0^{t_o} e^{a'(t_o-\tau)}b'r(\tau)d\tau \ . \qquad (10)$$

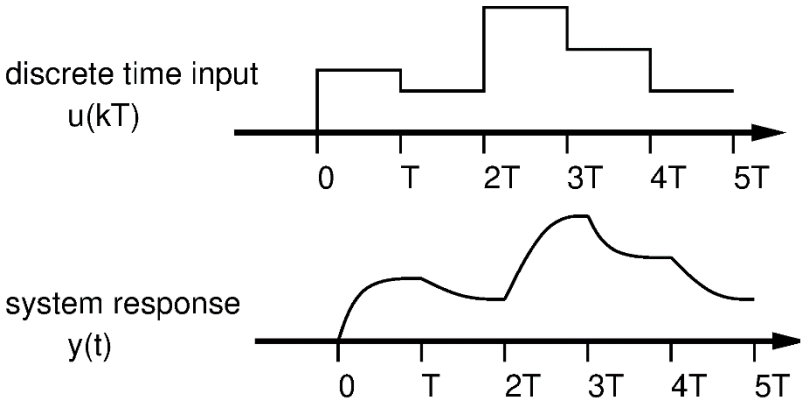


Step Response (zero init condx) by convolution:

$$x(t_o) = b' \int_0^{t_o} e^{a't_o}e^{-a'\tau}d\tau \ = \ \frac{-b'e^{a't_o}}{a'}e^{-a'\tau}|_0^{t_o} \ = \frac{b'}{a'}(1 - e^{-a't_o}) \ . \qquad (11)$$

Control Synopsis- Discrete Time

Superposition of Step Responses



$$x((k+1)T) = e^{a(k+1)T}x(0) + e^{a(k+1)T}\int_0^{(k+1)T} e^{-a\tau}bu(\tau)d\tau \ . \qquad (15)$$

$$x(kT) = e^{akT}x(0) + e^{akT}\int_0^{kT} e^{-a\tau}bu(\tau)d\tau \ . \qquad (14)$$

$$x((k+1)T) = e^{aT}x(kT) + e^{a(k+1)T}\int_{kT}^{(k+1)T} e^{-a\tau}bu(\tau)d\tau \ = e^{aT}x(kT) + \int_0^{T} e^{a\lambda}bu(kT)d\lambda \ , \ (16)$$

Control Synopsis- Discrete Time

$$G(T) \equiv e^{aT} \quad \text{and} \quad H(T) \equiv b \int_0^T e^{a\lambda} d\lambda \ . \tag{17}$$

State equations:

$$x((k+1)T) = G(T)x(kT) + H(T)u(kT) \tag{18}$$

Output equations:

$$y(kT) = Cx(kT) + Du(kT) \ . \tag{19}$$

Total Response (non-zero init condx) by convolution:

$$x(k) = G^k x(0) + \sum_{j=0}^{k-1} G^{k-j-1} H u(j) \ . \tag{23}$$

Control Synopsis- Discrete Time

Control Law (P):

$$U(kT) = k_p\,[r(kT) - x(kT)]$$

New state equations:

$$x((k+1)T) = G(T)x(kT) + H(T)k_p(r(kT) - x(kT)) = [G - Hk_p]x(kT) + Hk_p r(kT)\ . \quad (24)$$

$$x((k+1)T) = [e^{aT} + \frac{k_p}{a}(1 - e^{aT})]x(kT) + Hk_p r(kT) \ = \ G'x(kT) + Hk_p r(kT) \quad . \qquad (25)$$

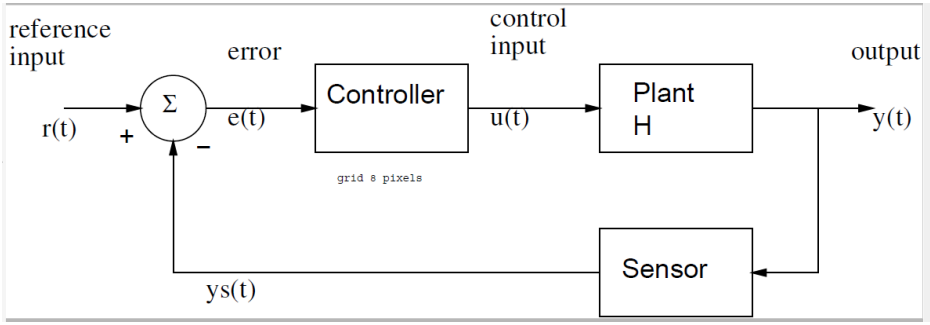
For stability:

$$|e^{aT} - \frac{k_p}{a}(e^{aT} - 1)| < 1. \qquad (26)$$

Notes: stability depends on gain **and** T!

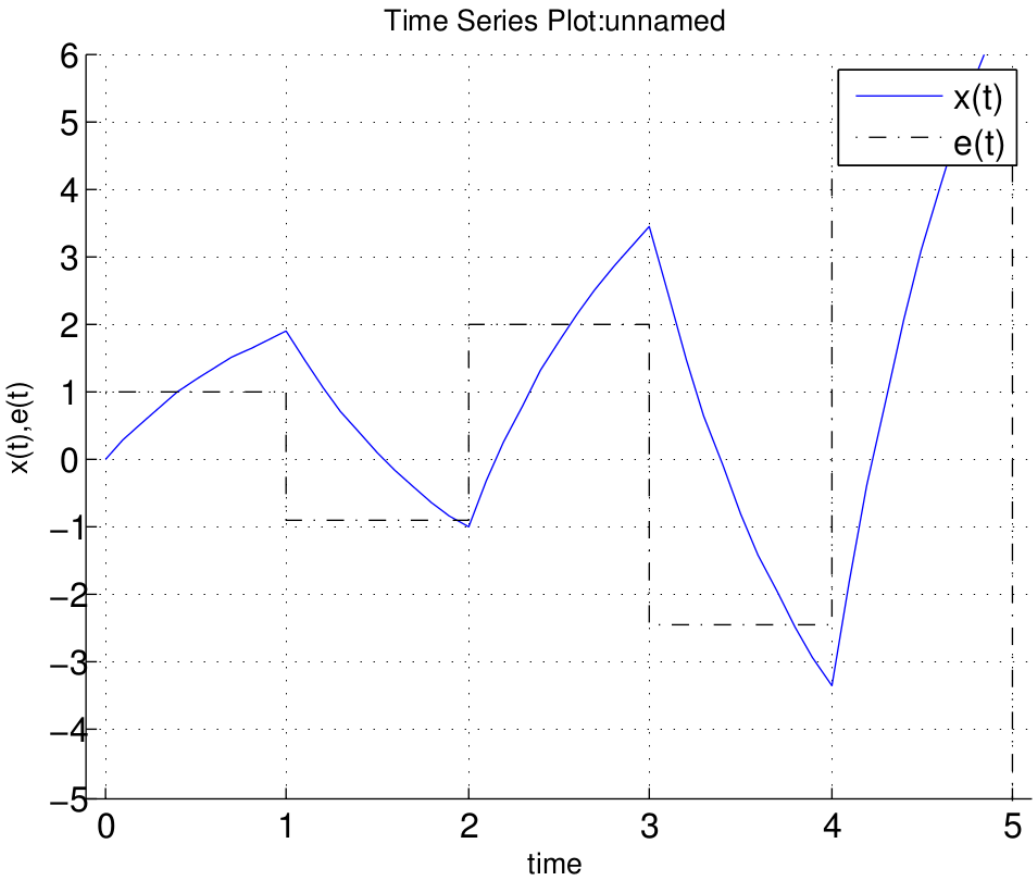
Discrete Time Control

e(t), u=kp e(t)



u[k] = kp*(r[k]-x[k])

Let x[k] = y[k]



Topics

- Progress Report questions?
- HW2 notes
- Discrete Time control/timing
- Software Robustness

10 Questions to Consider when Reviewing Code

Jacob Beningo

Embedded Systems Conference -2017

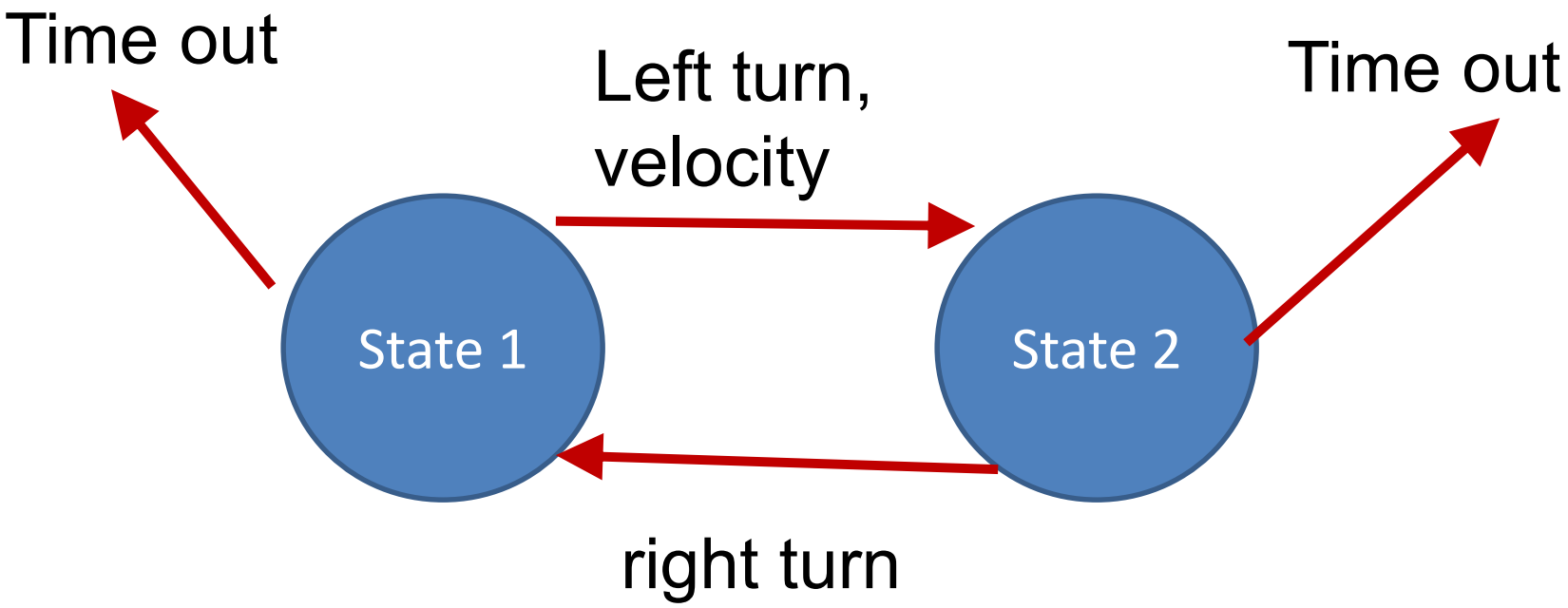
https://www.designnews.com/electronics-test/10-questions-consider-when-reviewing-code/143583201956491?cid=nl.x.dn14.edt.aud.dn.20170329

1. Does the program build without warnings?
2. Are there any blocking functions?
3. Are there any potential infinite loops?
4. Should this function parameter be const?
6. Has extern been limited with a liberal use of static?
7. Do all if … else if … conditionals end with an else?
8. Are assertions and/or input/output checks present?
9. Are header guards present? The guard prevents double inclusion of the #include directives.
10. *Is floating point mathematics being used?*

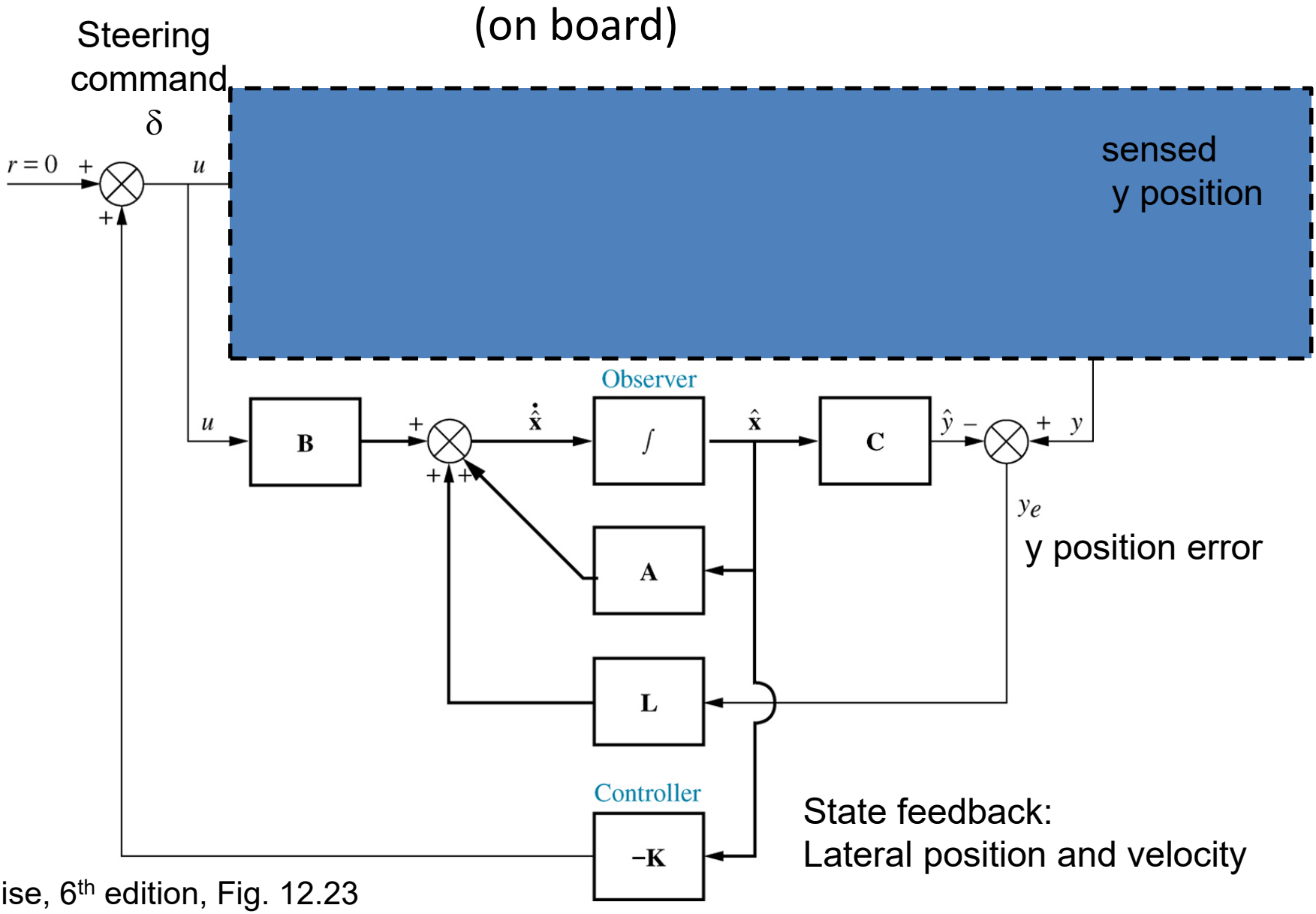
Software Robustness

- Checksums for bit rot
- Lost track detection
- Autocalibration at startup
  - (sanity check for steering angle vs line error)
  - AGC
- State Observer/estimator
- Discrete State observer
- Watch dog timer/computer operating properly COP

FSM Recognizer (generalized WDT)



Software Robustness: Observer



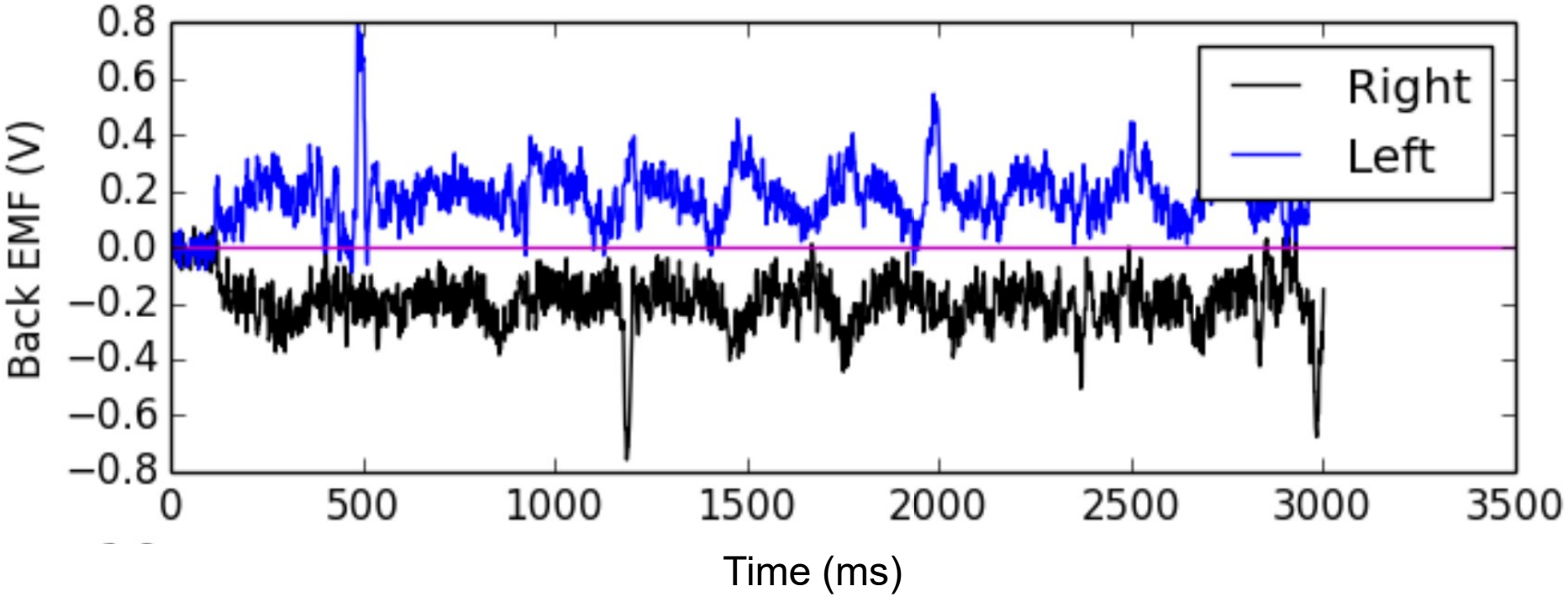
N. Nise, 6th edition, Fig. 12.23

Digital Filtering

- Moving average
  - y1[n] = (y[n-2]+y[n-1]+y[n])/3
- Median filter (outlier rejection)
- Notch filter (mechanical vibration)
  - y[n] = (x[n-2]+2x[n-1]+x[n])/4
- Model based filtering (or Kalman filter)

Moving Average vs. Median Filter

Example: motor brush noise, back EMF measurement



{0,2,-1,4,0,2,1,1,20,1,0,2} ➔

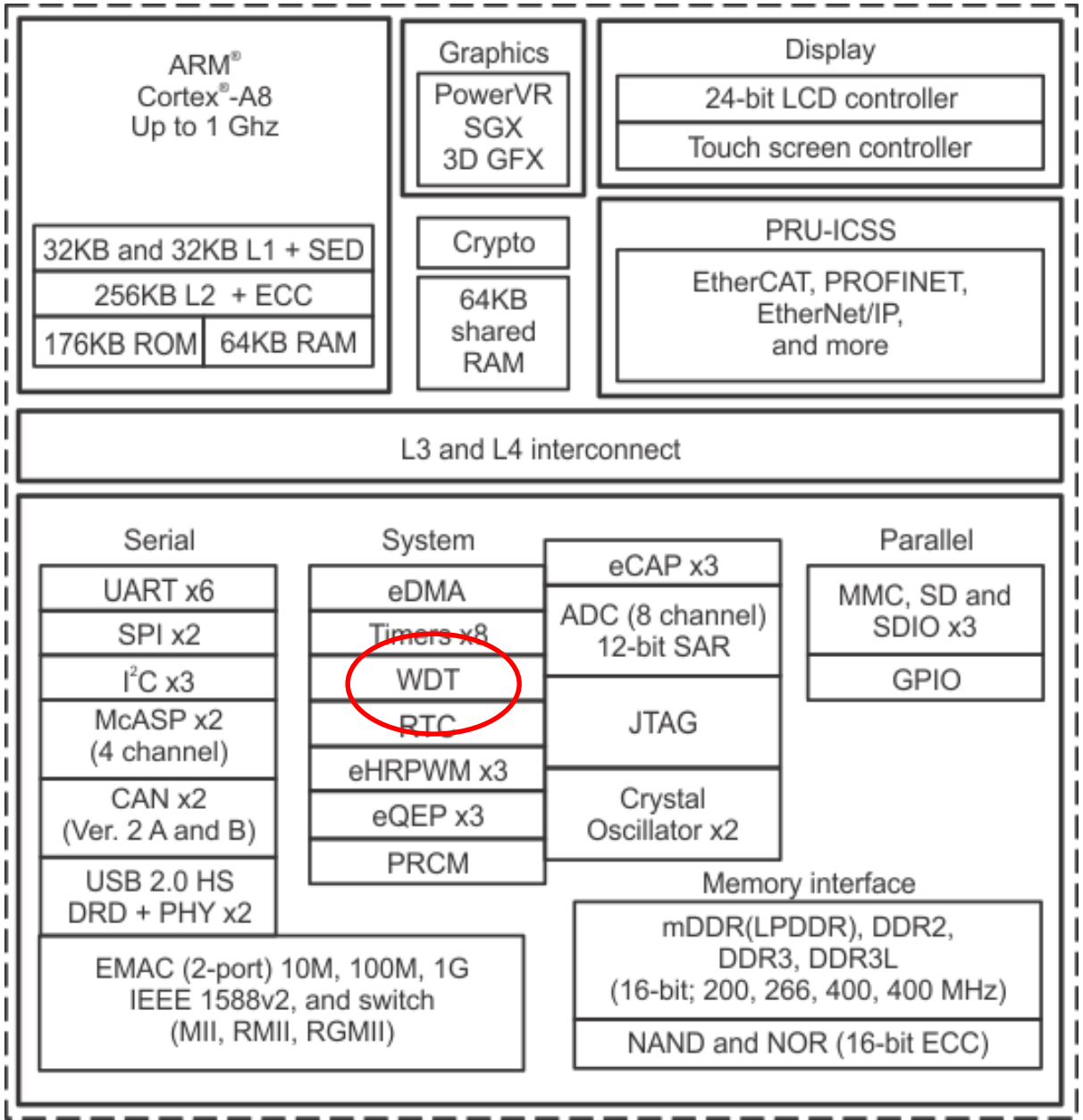
{0,2,-1,2,0,1,1,1,1,1} 3 element median filter

{0,2, 0.3,1.7, 2, 1,1.3, 7.3, 7.3, 7,1…} 3 elem MA

C.O.P. Watchdog timer

- Despite extensive software and hardware testing, faults will still occur in real devices. Even momentary noise spikes on a power supply can lock up a processor occasionally. Such events will occur on the power grid several times a year. Watchdog timers provide a last line of defense to prevent system failure with minimal hardware cost.
- https://developer.mbed.org/cookbook/Watch Dog-Timer

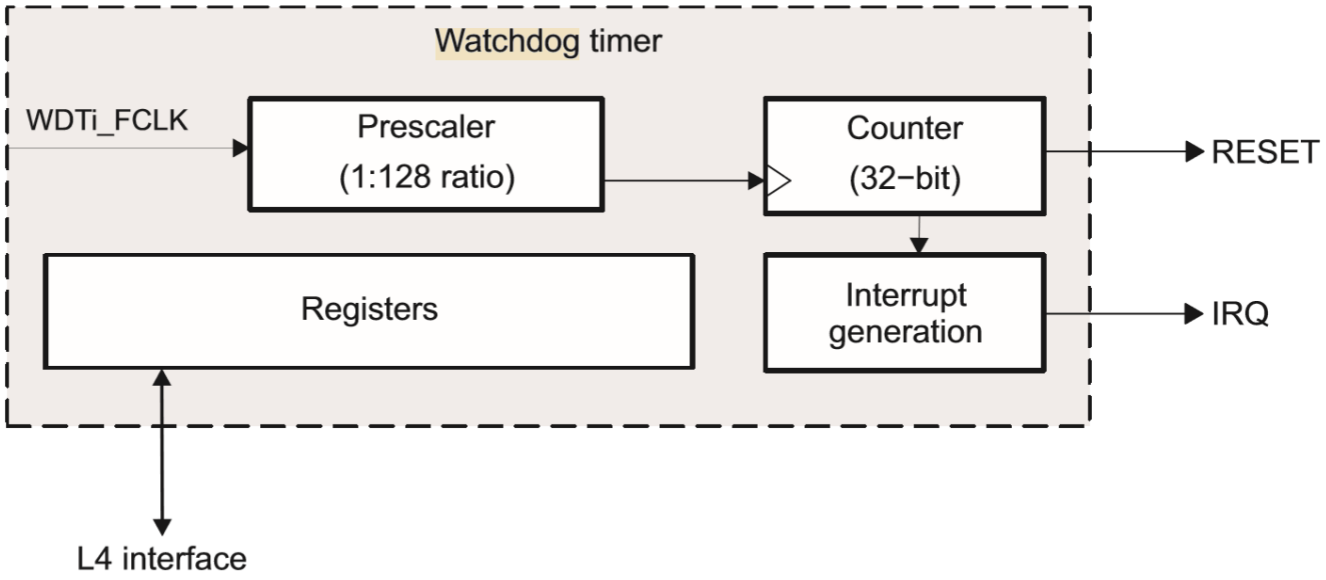
ARM Cortex A8 Overview



Beaglebone Blue
TI **Sitara™ AM335x Processors**
ARM Cortex A8
4GB Flash
512 MB RAM
32 bit ARM 7 core
1 GHz
A/D
3x SPI
Timers
WiFi
USB
microSD card

Watch Dog Timer

Figure 20-94. 32-Bit Watchdog Timer Functional Block Diagram



Watchdog reset

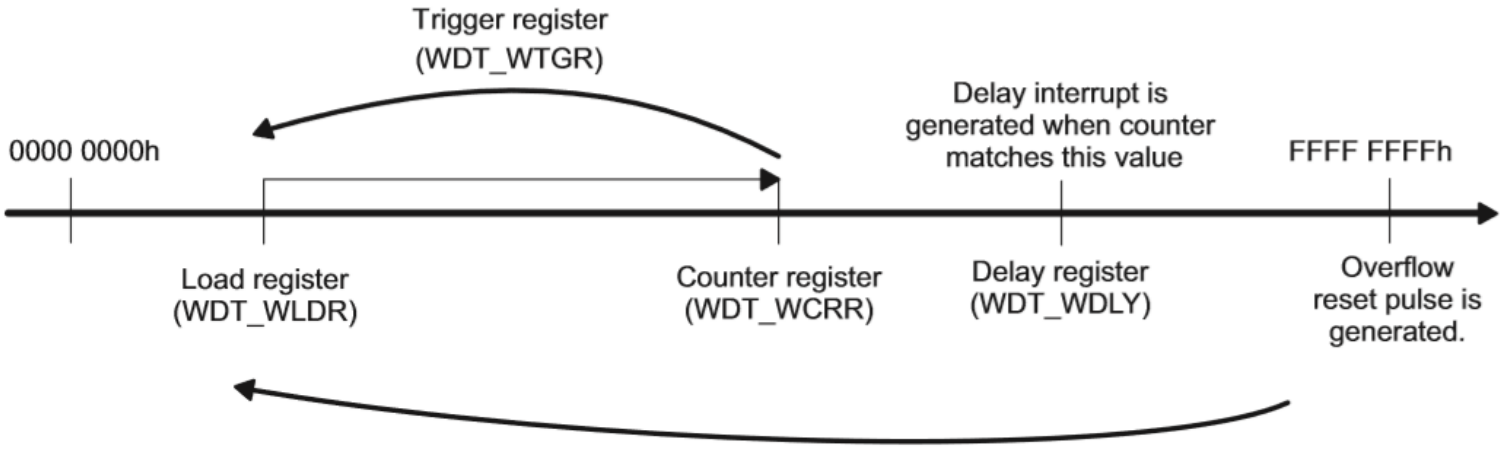
20.4.3.5 Overflow/Reset Generation

When the watchdog timer counter register (WDT_WCRR) overflows, an active-low reset pulse is generated to the PRCM module. This RESET pulse causes the PRCM module to generate global WARM reset of the device, which causes the nRESETIN_OUT pin to be driven out of the device. This pulse is one prescaled timer clock cycle wide and occurs at the same time as the timer counter overflow.

After reset generation, the counter is automatically reloaded with the value stored in the watchdog load register (WDT_WLDR) and the prescaler is reset (the prescaler ratio remains unchanged). When the reset pulse output is generated, the timer counter begins incrementing again.

Figure 20-95 shows a general functional view of the watchdog timers.

Figure 20-95. Watchdog Timers General Functional View



20.4.3.8 Start/Stop Sequence for Watchdog Timers (Using the WDT_WSPR Register)

To start and stop a watchdog timer, access must be made through the start/stop register (WDT_WSPR) using a specific sequence.
To disable the timer, follow this sequence:
1. Write XXXX AAAAh in WDT_WSPR.
2. Poll for posted write to complete using WDT_WWPS.W_PEND_WSPR.
3. Write XXXX 5555h in WDT_WSPR.
4. Poll for posted write to complete using WDT_WWPS.W_PEND_WSPR.

To enable the timer, follow this sequence:
1. Write XXXX BBBBh in WDT_WSPR.
2. Poll for posted write to complete using WDT_WWPS.W_PEND_WSPR.
3. Write XXXX 4444h in WDT_WSPR.
4. Poll for posted write to complete using WDT_WWPS.W_PEND_WSPR.

All other write sequences on the WDT_WSPR register have no effect on the start/stop feature of the module.