Implementation of PID Control

A/Prof Lindsay Kleeman

Outline

- Definition continuous time real variables
- Gain benefits and behaviours
- Implementation: discrete time, integer variables
- How to tune
- How to implement
- Understanding what can go wrong.

Background

- PID control is the most common controller used in industry.
- Simple to implement – suitable for embedded systems
- Works robustly for most control problems
- Computational simple and well understood

Definitions

- Suppose we wish to control *x(t)* to some desired trajectory *x*(t)* with the input *u(t).*
- Define the error, *e(t) = x*(t)-x(t)*
- A Proportional Integral Differential (PID) Controller

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d\,e(t)}{dt}$$

Proportional Gain

Integral Gain

Differential Gain

Motivation for Gains

- Increasing K_p gives system greater reaction to errors
  - Decreases response time
  - K_p too high results in oscillatory response or instability

- K_p alone cannot remove steady state errors.

  $\Rightarrow$ Need for integral feedback K_i

  steady state errors wind up integral term, reducing error.

- Increasing K_i causes overshoot and oscillation.
  - Differential gain K_d damps overshoot by "predicting" errors using the current error derivative.

Getting the Sign Wrong!

- What happens when the feedback sign is wrong?

  – Eg Optical shaft encoder swapping ϕA and ϕB

  Eg Swapping DC motor voltage terminals

Digital Implementation

- Continuous time is mapped to discrete time at a *constant* sample period Δt
  - Variations in sample period generate controller errors
    - Eg scheduling latencies in RTOS
- Real variables approximated by finite precision *integer_bit.fraction_bits* or *i.f (see fixed point arithmetic notes)*
  - More controller errors.
    - Need to analyse effects.

Proportion Gain Implementation

- $K_p e(t)$ implemented directly with integer multiplication

Integral Gain Implementation

$$K_i \int_0^t e(\tau)d\tau \cong K_i \Delta t \sum_{k=0}^{k=n} e(k\Delta t)$$

$$\equiv K_i \Delta t \sum_{k=0}^{k=n} e_k$$

Discrete time error

$$\equiv K_i \Delta t \, S(n) = K_i \Delta t \left(S(n-1) + e_n \right)$$

Error Sum

Recursive formulation – one addition and multiply per sample

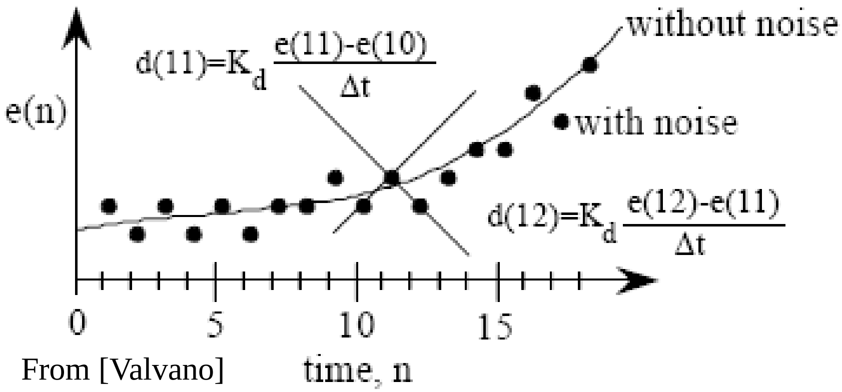
Integral Windup

- When steady state error cannot be corrected quickly, integral term increases indefinitely – called integral windup.
  - Controller saturates and cannot recover quickly
- Solutions:
  - limiting integral term, or
  - Limiting time period of integration
  - Disabling integral term outside controllable region

Differential Gain Implementation

$$K_d \frac{d\,e(t)}{dt} \cong K_d \frac{e_n - e_{n-1}}{\Delta t} = \frac{K_d}{\Delta t}\left(e_n - e_{n-1}\right)$$

Recursive formulation – one subtraction and multiply per sample



From [Valvano]

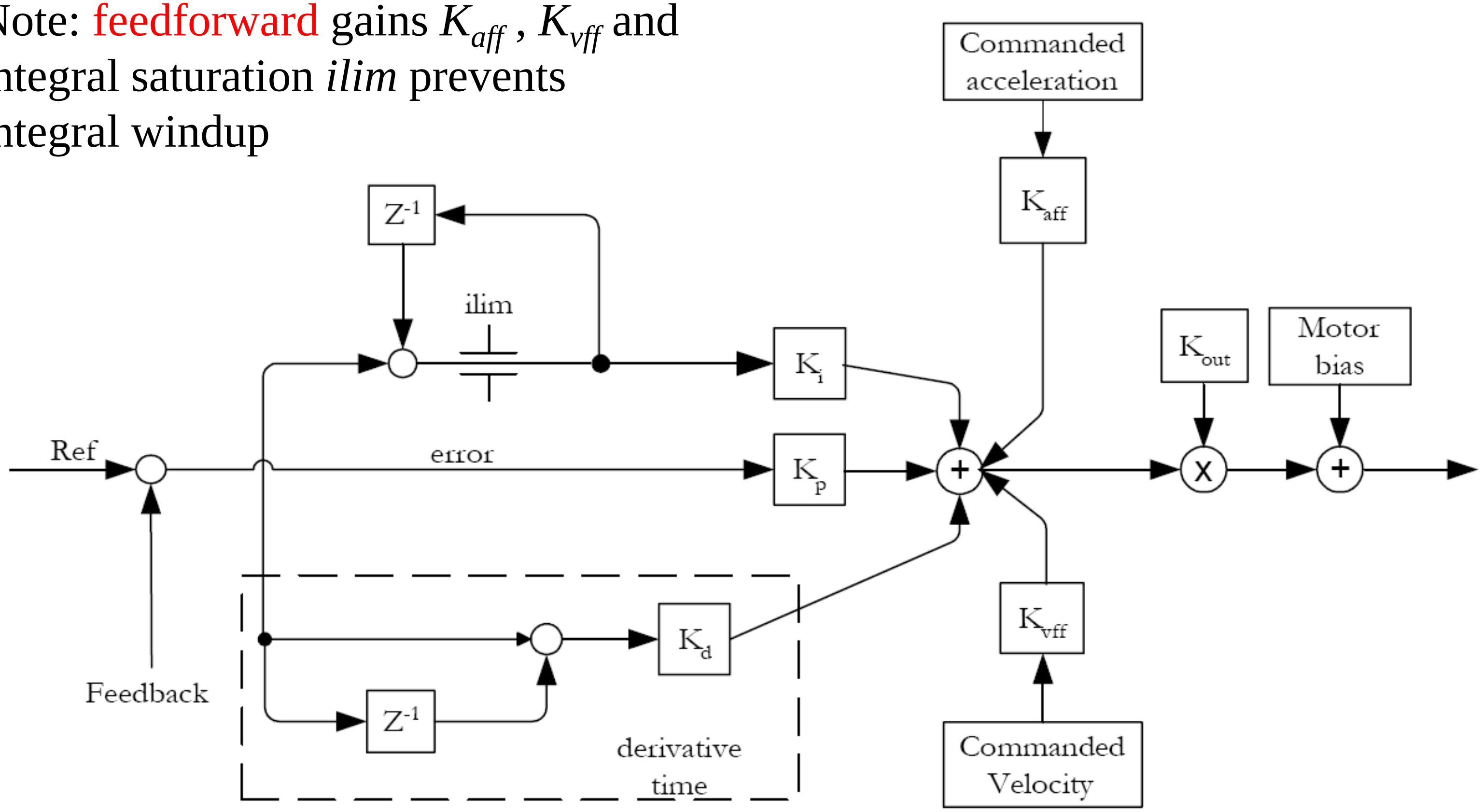
Error Difference can be noisy: average below better:

$$K_d \frac{d\,e(t)}{dt} \cong K_d\left(\frac{1}{2}\left[\frac{e_n - e_{n-3}}{3\Delta t}\right] + \frac{1}{2}\left[\frac{e_{n-1} - e_{n-2}}{\Delta t}\right]\right)$$

$$= \frac{K_d}{6\Delta t}\left(e_n + 3e_{n-1} - 3e_{n-2} - e_{n-3}\right)$$

Commercial Implementation

Note: feedforward gains K_{aff} , K_{vff} and integral saturation *ilim* prevents integral windup



http://www.pmdcorp.com/downloads/app_notes/servoLoop.pdf

Pendulum Lab Implementation

$$MotorVolt_n = \left(K_p e_n + K_d^1 (e_n - e_{n-1}) + (128 + K_i^1 \sum_{j=0}^{n} e_j)/256 + 128 \right)/256$$

- $K'_d = K_d/\Delta t$ and $K'_i = K_i * \Delta t$, so that K_p , K'_d and K'_i scale as *1 1/Δt* and *Δt*
- K_p and K'_d 8.8 bit representation (8 integer + 8 fraction bits)
- K'_i 0.16 representation
- Rounding is used when removing fractional bits from results
- *Δt* approximately 10 msec.
  - If *Δt too small* =>
    - error difference mostly 0 and rarely 1, rendering K_d useless,
    - integral winds up since error ~constant
  - If *Δt too large* => poor control and slow.
    Want *Δt* << time constant of motor response.

Gain/Sample Time Tuning

Design controller (Ziegler and Nichol)

$\Delta\mathbf{T}$ is the time step for the digital controller.

run P and PI controllers with $\Delta\mathbf{T} = 0.1\mathbf{L}$,

run a PID controller $\Delta\mathbf{T} = 0.05\mathbf{L}$.

Proportional Controller

$\mathbf{K_P} = \Delta\mathbf{U}/(\mathbf{L}*\mathbf{R})$

Proportional-Integral Controller

$\mathbf{K_P} = 0.9\ \Delta\mathbf{U}/(\mathbf{L}*\mathbf{R})$

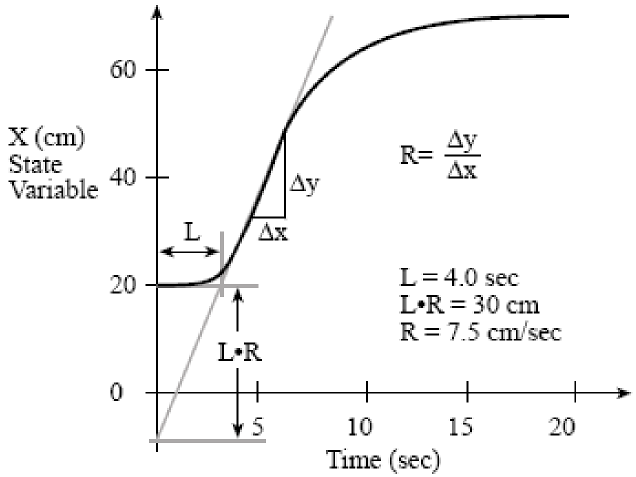
$\mathbf{K_I} = \mathbf{K_P}/(3.33\mathbf{L})$

Proportional-Integral-Derivative Controller

$\mathbf{K_P} = 1.2\ \Delta\mathbf{U}/(\mathbf{L}*\mathbf{R})$

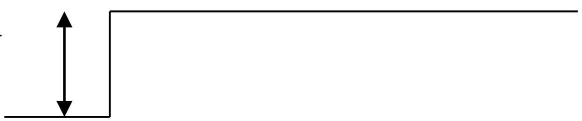
$\mathbf{K_I} = 0.5\ \mathbf{K_P}/\mathbf{L}$

$\mathbf{K_D} = 0.5\ \mathbf{K_P}\ \mathbf{L}$



From [Valvano]

Input step ΔU
At Time=0



Not suitable for DC motors with angle feedback and voltage control – why?

Ziegler–Nichols Tuning

- K_i *and* K_d are first set to zero.
- K_p increased to "critical gain" *Kc* where starts to oscillate.
- *Kc* and the oscillation period *Pc* are used to set the gains as shown:

Ziegler–Nichols method			
Control Type	K_p	K_i	K_d
P	$0.5 \cdot K_c$	-	-
PI	$0.45 \cdot K_c$	$1.2K_p / P_c$	-
PID	$0.6 \cdot K_c$	$2K_p / P_c$	$K_pP_c / 8$

Discrete Time Version	*Kp*	*Ki'*	*Kd'*
P	*0.5 Kc*	-	-
PI	*0.45 Kc*	*1.2Δt Kp/Pc*	-
PID	*0.6 Kc*	*2 Δt Kp/Pc*	*KpPc/(8Δt)*

http://en.wikipedia.org/wiki/PID_controller

Manual Tuning

Requires experience and trial and error

Effects of *increasing* parameters				
Parameter	**Rise Time**	**Overshoot**	**Settling Time**	**S.S. Error**
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small Decrease	Decrease	Decrease	None

http://en.wikipedia.org/wiki/PID_controller

Summary

- PID is a low cost, popular & effective control strategy
- Tuning and servo sample time dependent on system dynamics
- Need good understanding of implementation issues:
  - Integral windup
  - Output saturation
  - Over/under sample rates
  - Benefits of feedforward: reduced errors, faster response.