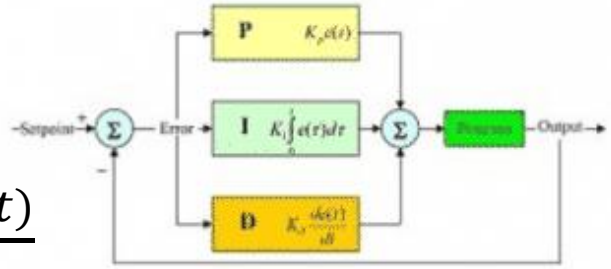
SISTEMAS EMBARCADOS

PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

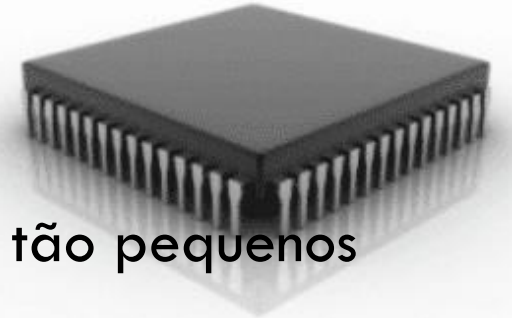
PID: IMPLEMENTAÇÃO



No domínio contínuo: $u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$

ou em termos do tempo integral τ_i e do tempo derivativo τ_d

$$u(t) = K_p \left(e(t) \frac{1}{\tau_i} \int_0^t e(\tau)d\tau + \tau_d \frac{de(t)}{dt} \right)$$



Equivale ao caso analógico, ou seja, os intervalos de tempo são tão pequenos quanto se queira (grandeza analógica)!

Na implementação em microprocessador/microcontrolador, as grandezas são **discretizadas** de acordo com determinado período de amostragem T_s e os termos integrativo e derivativo precisam ser **aproximados para implementação em computador/microcontrolador**

PID: IMPLEMENTAÇÃO

Definições: $u(t)$: sinal de controle/comando/MV: variável manipulada

$y(t)$: sinal de saída/PV: variável de processo

K_p, K_i, K_d: ganhos proporcional, integrativo, derivativo

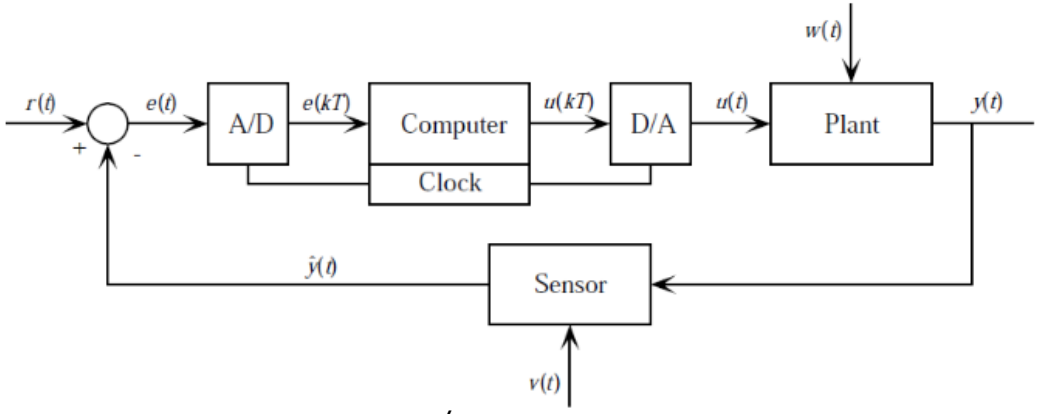
$e(t)$: erro de rastreamento (referência (setpoint) – y(t)): off-set

Ação	Efeito
proporcional	Reduz oscilações, tornando o sistema mais estável, mas não garante erro (off-set) nulo, age com uma maior amplitude de correção a fim de manter a estabilidade
integrativa	Elimina o desvio de off-set (erro), fazendo com que a PV permaneça próximo ao setpoint mesmo após um distúrbio, quanto maior for o tempo de permanência do erro, maior o valor da ação integral
derivativa	Fornece ação antecipativa evitando previamente que o erro se torne maior quando o processo se caracteriza por ter uma resposta lenta em relação à velocidade da variação do erro. Com o erro detetado, aumenta a velocidade da resposta. Para lentos (temperatura, ok), para rápidos (velocidade, líquido, próprio sistema corrige rápido, parte derivativa pode ser anulada)

PID: EFEITO INDIVIDUAL AÇÕES (INCREMENTO)

Resposta	Tempo de subida	Overshoot (%)	Tempo de estabilização	Erro regime permanente
kp	reduz	aumenta	Pouca alteração	reduz
ki	reduz	aumenta	aumenta	elimina
kd	Pouca alteração	reduz	reduz	Pouca alteração

Em sistemas microprocessados/microcontrolados, existem disponíveis várias bibliotecas com implementações do PID: https://playground.arduino.cc/Code/PIDLibrary/. O sinal analógico de saída da planta/sistema é adquirido via conversor A/D, processado, e então enviado para a planta via PWM ou conversor D/A. Se o sensor da PV é digital, não há necessidade do A/D.



https://www.embarcados.com.br/controlador-pid-digital-parte-1/

PID: APROXIMAÇÕES

$$P(t) = K_p \times e(t) \leftrightarrow K_p \times e[n], n = k \times T_s, com\ k\ \in N$$

As k amostras são espaçadas por Ts

- Parte derivativa (lembrar Matemática II!)

$$\frac{df(t)}{dt} = lim_{\Delta t \to 0} \frac{f(t) - f(t - \Delta t)}{\Delta t}, seja\ \Delta t = T_s.$$

como cada amostra é lida a cada Ts, pode-se aproximar no caso discreto por uma simples diferença

$$K_d \frac{de(t)}{dt} = K_d \times (e[n] - e[n-1])$$

No caso da integral, é a somatório:

$$\int_t^{t+\Delta t} f(\tau)d\tau = lim_{\Delta t \to 0} \sum_{n=t}^{\Delta t + t} f(n) \times \Delta t = \Delta t \times \{f(n) + f(\Delta t + t)\}$$

PID: APROXIMAÇÕES

$$K_i \times \int_0^t e(\tau)d\tau = K_i \times (f[n] + f[n-1])$$

Logo,

$$u[n] = K_p \times e[n] + K_d \times (e[n] - e[n-1]) + K_i \times (f[n] + f[n-1])$$

onde,

$$K_i = \frac{K_p \times T_s}{\tau_i}, K_d = \frac{K_p \times \tau_d}{T_s}$$

Na prática, a ação integral deve ser limitada, pois pode crescer indefinidamente (efeito wind-up), sendo necessário um RESET ao superar um valor máximo IMAX. A saída de controle também deve ser limitada (SATURAÇÃO) ao valor máximo possível (UNO: 5V, ESP: 3,3 V etc.)

Código de controle PID em C.

```c
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}
```

```
void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}
```

Analisar linha a linha código: https://playground.arduino.cc/Code/PIDLibrary/.

PID_v1.h e PID_v1.cpp

Existem outras implementações mais precisas, como esta:
https://www.embarcados.com.br/controlador-pid-digital-parte-2/