

---

# Implementazione tramite microcontrollori di PID

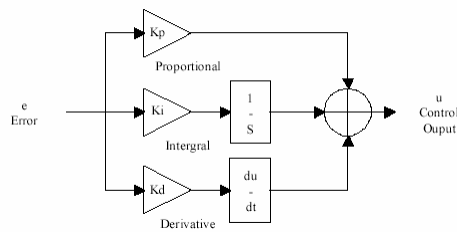
Pierfrancesco Foglia

---

---

## Introduzione

- Controllore PID: nel dominio del tempo



$$u_c(t) = k_p e(t) + k_i \int_0^t e(t) dt + k_d \frac{de}{dt}$$

## Algoritmo digitale: errore e termine proporzionale

- sia

$V_{ref}(t)$  = Tensione di Riferimento

$u_c(t)$  = Uscita del controllore

$u(t)$  = Uscita del sistema

- errore

$$e(t) = V_{ref}(t) - u(t) \Rightarrow e(k) = V_{ref}(k) - u(k)$$

- Termine proporzionale

$$u_{cp}(t) = K_p e(t) \Rightarrow u_{cp}(k) = K_p e(k)$$

## Termine integrale/I

$$u_{ci}(t) = K_i \int e(t) dt \Rightarrow$$

$$u_{ci}(0) = 0$$

$$u_{ci}(1) = K_i T \frac{1}{2} (e(0) + e(1)) \quad \text{regola dei trapezi}$$

$$\begin{aligned} u_{ci}(2) &= u_{ci}(1) + K_i T \frac{1}{2} (e(1) + e(2)) = K_i T \frac{1}{2} (e(0) + e(1)) + K_i T \frac{1}{2} (e(1) + e(2)) \\ &= K_i T \left( \frac{e(0)}{2} + e(1) + \frac{e(2)}{2} \right) \end{aligned}$$

$$u_{ci}(3) = K_i T \left( \frac{e(0)}{2} + e(1) + e(2) + \frac{e(3)}{2} \right)$$

## Termine integrale/II

$$u_{ci}(k) = K_i T \left( \frac{e(0)}{2} + e(1) + \dots + e(k-1) + \frac{e(k)}{2} \right)$$

tutti i termini seguono la regola "dei rettangoli" eccetto il primo e l'ultimo  
si approssima il valore a quello dei rettangoli (valido per T piccolo)

da cui

$$u_{ci}(k) = K_i T \sum_{i=0}^k e(i)$$

## Termine derivativo/formula complessiva

$$u_{cd}(t) = \frac{de(t)}{dt} \Rightarrow u_{cd}(k) = K_d \frac{(e(k+1) - e(k))}{T}$$

il rapporto incrementale si calcola come  $e(k) - e(k-1)$   
( $e(k+1)$  non è noto)

da cui

$$u_{cd}(k) = K_d \frac{(e(k) - e(k-1))}{T}$$

*complessivamente*

$$u_c(k) = K_p e(k) + K_i T \sum_{i=0}^k e(i) + K_d \frac{(e(k) - e(k-1))}{T}$$

## algoritmo/floating

```
Static float Sum_G, Old_error_G;
float PID_Control(float Error)
{
    // Termine proporzionale
    float Control_new = (PID_KP * Error);
    // termine integrale
    Sum_G += Error;
    Control_new += PID_KI * Sum_G; // nota: PID_KI è PID_KI/SAMPLE_RATE
    // termine Differenziale
    Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));
    // Control_new non può superare PID_MAX o essere inferiore a PID_MIN
    if (Control_new > PID_MAX)
    {
        Control_new = PID_MAX;
    }
    else
    {
        if (Control_new < PID_MIN)
            Control_new = PID_MIN;
    }
    // Memorizza l'errore
    Old_error_G = Error;
    return Control_new;
}
```

## Implementazione anti windup

- Quando l'attuatore lavora al max/min (ad esempio PWM con duty-cycle al 100% o 0%), non solo non è possibile aumentare/diminuire la risposta in uscita, ma non è opportuno continuare a sommare l'errore nell'integratore
- In tal caso ("saturando" l'integratore), si ottiene una migliore risposta del sistema

## Codice anti windup

```
float PID_Control(float Error)
{
    float Control_new = (PID_KP * Error);
    Sum_G += Error; Control_new += PID_KI * Sum_G;
    Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));

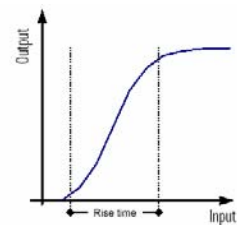
    if (PID_WINDUP_PROTECTION) // protezione wind-up opzionale
    {
        if ((Control_new > PID_MAX) || (Control_new < PID_MIN))
            Sum_G -= Error; // Don't increase Sum...
    }

    if (Control_new > PID_MAX)
        Control_new = PID_MAX;
    else
        if (Control_new < PID_MIN)
            Control_new = PID_MIN;
    Old_error_G = Error;
    return Control_new;
}
```

## Scelta della frequenza di campionamento

- Nel DSP si utilizza solitamente il criterio di Nyquist
- Nei sistemi di controllo, un criterio usato è quello del rise-time:
  - Valori di 6, 20, 40 volte sono utilizzati
  - Se rise time = 0,1 s, si prende  $f_s = 400$  Hz

$$\text{Sample frequency} = \frac{40}{\text{Rise time}}$$



## Complessità computazionale e problemi di controllo

- Sono richieste 4 moltiplicazioni, 4 addizioni e 1 sottrazione
- In fp, sono circa 2000 istruzioni se il microcontrollore non dispone di hw fp
- Sono necessari circa 2 msec in un 8051 a 12Mhz
  
- Ciò è sufficiente per la maggior parte dei problemi di controllo, in cui sono tipicamente richiesti intervalli di campionamento di 100 msec o più
  
- In caso tale prestazioni non sono sufficienti, si può
  - Utilizzare un microcontrollore con hw fp
  - Prendere un microcontrollore che supporta clock più elevati
    - Fino a 50Mhz per l'8051
  - Prendere un microcontrollore più potente
    - 16-32 bit

## Il tuning

- Algoritmo “empirico”, basato sul metodo di Ziegler-Nichols ma molto usato in pratica
  - Porre a zero i termini integrale e differenziale
  - Partendo da  $K_p$  basso, incrementare  $K_p$  lentamente fino ad ottenere oscillazioni (fino all'instabilità)
  - Ridurre  $k_p$  alla metà del valore precedente
  - Se necessario, introdurre piccoli valori di  $k_d$  per “tagliare” la risposta
  - Se necessario, introdurre piccoli valori di  $k_i$  per portare a zero errori in regime stazionario
    - Iterare varie volte negli ultimi due passi
  - Se  $K_i \neq 0$ , utilizzare sempre la protezione wind-up

## Altri algoritmi di tuning

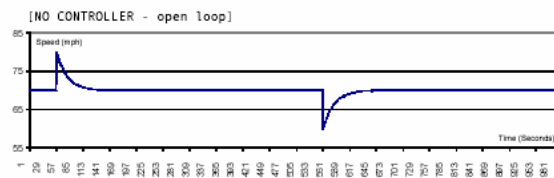
- Esistono algoritmi “formali”, basati sulla risposta ad anello aperto o chiuso del sistema e formule per ricavare da queste i parametri
  - Modelli basati sulla modellazione matematica del sistema ed il calcolo dei parametri della risposta
  - Modelli sperimentali - Esempio: metodo di Ziegler-Nichols

## Esempio di Tuning: Cruise Control in una macchina: calcolo dei parametri del PID

- Non disponendo del sistema, si può procedere per via numerica tramite simulazione.
- Si integra l'equazione del moto tramite la formula di Eulero
  - Throttle è il segnale di controllo, con cui si agisce sul motore
  - Fric si oppone al moto in modo proporzionale alla velocità
  - Mass è la massa del mezzo
  - L'accelerazione è data da:
    - $Accel = (float)(Throttle * ENGINE\_POWER - (FRIC * Old\_speed)) / MASS;$
  - La velocità, in un intervallo DT, è pari a
    - $Speed = Old\_speed + Accel * (1.0f * DT);$
  - L'intervallo DT può essere pari alla frequenza di campionamento o, meglio, inferiore
    - Si migliora l'algoritmo di integrazione

## Sollecitazione e risposta a ciclo aperto

- Si suppone che la velocità impostata sia di 70MH
- Si suppone che a  $t=50$  s e a  $t=550$ s la velocità vari per effetto del vento (accelera e decellera)
- Ecco la risposta a ciclo aperto



- Effettuare il Tuning del PID

## Esercizio

- Svolgere il Tuning del PID dell'esercizio precedente (Cruise Control System), utilizzando come "chiusura dell'anello" il modello approssimato della soluzione dell'equazione del moto.
- Viene fornito il codice C
- Il grafico si può costruire con excel

## Altre implementazioni di PID/A

- L'algoritmo implementativo di PID visto genera un segnale nullo in caso di errore nullo.
- In molti problemi di controllo, non si deve generare un segnale nullo, ma un segnale che “non varia” in caso di errore nullo
- Nel caso precedente, si ottiene, nel Tuning, un errore a regime non nullo, che si elimina tramite componente integrale
- Tale componente si può eliminare sommando anche l'uscita del PID al passo precedente
  - Ciò equivale in qualche modo a mettere un integratore aggiuntivo in serie

## Implementazione con “integratore aggiuntivo in serie”

```
Static float Sum_G, Old_error_G;
float PID_Control(float Error, float Control_old)
{
    // Termine proporzionale
    float Control_new = Control_old + (PID_KP * Error);
    // termine integrale
    Sum_G += Error;
    Control_new += PID_KI * Sum_G; // nota: PID_KI è PID_KI/SAMPLE_RATE
    // termine Differenziale
    Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));
    // Control_new non può superare PID_MAX o essere inferiore a PID_MIN
    if (Control_new > PID_MAX)
    {
        Control_new = PID_MAX;
    }
    else
    {
        if (Control_new < PID_MIN)
            Control_new = PID_MIN;
    }
    // Memorizza l'errore
    Old_error_G = Error;
    return Control_new;
}
```

Termine Integrale  
“aggiuntivo”

## Implementazione intera

- Richiede meno istruzioni (è più veloce) ma richiede attenzione per il calcolo delle costanti, per evitare di perdere in precisione
- 2 soluzioni:
  - Si moltiplica per una costante, in modo da aumentare la risoluzione
  - Si lavora con  $1/X$  o  $1/(X+1)$ , per valori di  $K_i$ ,  $K_d$ ,  $K_p < 1$ 
    - Come è fatto il grafico di tali funzioni?

## Implementazione 1

```
int Sum_G, Old_error_G; // interi ad 8 bit
int PID_KP; // memorizzato come kp*8
int PID_KI; // memorizzato come ki*TS*256
int PID_KD; // memorizzato come (kd/TS)*256

int PID_Control(int Error, int Control_old)
{
    // Termine proporzionale
    int tmp_x = PID_KP * Error;
    tmp_x = tmp_x >> 3; // dividi per 8
    int Control_new = Control_old + tmp_x; // accumula nell'uscita

    // termine integrale
    Sum_G += Error; // accumula l'errore
    tmp_x = PID_KI * Sum_G;
    tmp_x = tmp_x >> 8; // dividi per 256
    Control_new += tmp_x; // accumula nell'uscita

    // termine Differenziale
    tmp_x = PID_KD * (Error - Old_error_G);
    tmp_x = tmp_x >> 8; // dividi per 256
    Control_new += tmp_x; // accumula nell'uscita
}
```

# Implementazione 1

```
// Control_new è limitata da PID_MAX e PID_MIN che sono ora 0 e 255 per int a 8 bit
if (Control_new > 255)
{
    Control_new = 255;
}
else
{
    if (Control_new < 0)
        Control_new = 0;
}
// Memorizza l'errore
Old_error_G = Error;
return Control_new;
} // end_of_PID_CONTROL
```

- Nota: si può aggiungere il codice per la gestione del wind-up

# Implementazione 2

```
int Sum_G, Old_error_G; // interi ad 8 bit
int PID_KP; // memorizzato come 1/kp
int PID_KI; // memorizzato in modo che  $TS*ki=1/(1+PID\_KI)$ 
// nota:  $PID\_KI=1/(TS*ki)-1$ 
// per valori <1 di  $TS*ki$  è grande ma tende a INF per  $ki \rightarrow 0$ 
// occorre un controllo if ( $ki=0 \rightarrow PID\_KI=0$ )
int PID_KD; // memorizzato in modo che  $kd/TS=1/(1+PID\_KD)$ 
// nota:  $PID\_KD=TS/kd-1$ 
// per valori <1 di  $TS/ki$  è grande ma tende a INF per  $kd \rightarrow 0$ 
// occorre un controllo if ( $kd=0 \rightarrow PID\_KD=0$ )

int PID_Control(int Error, int Control_old)
{
    // Termine proporzionale
    int Control_new = Control_old + Error/PID_KP;

    // termine integrale
    // calcolato se  $PID\_KI \neq 0$ 
    if (PID_KI)
    {
        Sum_G += Error;
        Control_new += Sum_G/(1+PID_KI);
    }
}
```

## Implementazione 2

```
// termine Differenziale
// calcolato se PID_KI !=0
if (PID_KD)
{
    Control_new += (Error - Old_error_G)/(1+PID_KD);
    Old_error_G = Error;
}

// Control_new non può superare PID_MAX o essere inferiore a PID_MIN ma sono 0 e 255
if (Control_new > 255)
{
    Control_new = 255;
    Sum_G -= Error; // protezione di windup
}
else
    if (Control_new < 0)
    {
        Control_new = 0;
        Sum_G += Error; // protezione di windup
    }

return Control_new;
}
```

## Altre implementazioni di PID/B

- Si possono utilizzare differenti algoritmi numerici per migliorare la stabilità numerica o l'errore
- Si può utilizzare una formulazione "ricorsiva", utile per il passaggio dal funzionamento manuale a quello automatico
- Esempi:
  - Algoritmo di velocità
  - Termine derivativo migliorato

## Algoritmo di velocità

$$u_c(k) = K_p e(k) + K_i T \sum_{i=0}^k e(i) + K_d \frac{(e(k) - e(k-1))}{T}$$

$$u_c(k-1) = K_p e(k-1) + K_i T \sum_{i=0}^{k-1} e(i) + K_d \frac{(e(k-1) - e(k-2))}{T}$$

$$u_c(k) - u_c(k-1) = K_p (e(k) - e(k-1)) + K_i T e(k) + K_d \frac{(e(k) - 2e(k-1) + e(k-2))}{T}$$

da cui

$$u(k) = u(k-1) + A_1 e(k) + A_2 e(k-1) + A_3 e(k-2)$$

con

$$A_1 = K_p + K_i T + K_d / T; A_2 = -(K_p + 2K_d / T); A_3 = K_d / T$$

- Utilizzo: per il passaggio dalla modalità manuale a quella automatica

## Termine derivativo migliorato

- Migliora i disturbi di misura

$$\frac{de}{dt} = \frac{1}{6} (e(k) + 3e(k-1) - 3e(k-2) - e(k-3)) \text{ media a quattro punti}$$

da cui

$$u(k) = A_1 e(k) + A_2 e(k-1) + A_3 e(k-2) + A_4 e(k-3) + S(k)$$

con

$$A_1 = K_p + K_d / 6T; A_2 = K_d / 2T; A_3 = -K_d / 2T; A_4 = -K_d / 6T$$

$$S(k) = \text{termine integrativo solito}$$

## Quale implementazione? Fp vs int32 vs int 16

- Vantaggi <<
  - Aumento della precisione
    - Minori errori di
      - Quantizzazione
      - Arrotondamento
  - Svantaggi
    - Maggiore potenza di calcolo richiesta
      - Controllori più potenti (costosi) o algoritmi più lenti
    - Maggiore lunghezza del codice
    - Maggiore dimensione dell'area dati
  - Osservazioni:
    - la maggior parte dei problemi di controllo richiedono frequenze di campionamento superiore al msec – decine di msec
    - I tempi di calcolo del PID devono essere inferiori a tale ordine
- In generale, la scelta del tipo di implementazione dipende dal particolare problema, dalle prestazioni richieste (in termini di risposta) e dal costo
  - Occorre valutare sperimentalmente la soluzione

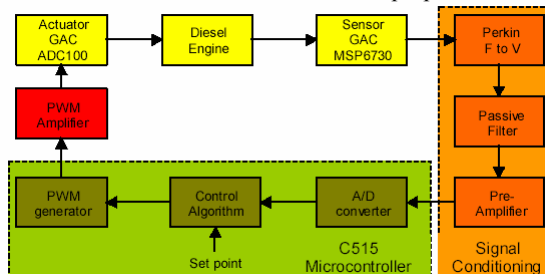
## Caso di Studio: il controllo di un motore diesel

Attuatore di velocità:

- Agisce sugli iniettori
- È controllato da una PWM

senore di velocità:

- Produce un segnale di frequenza variabile
- È proporzionale alla velocità



Condizionamento:

- Convertitore F-V
- Filtro passivo antialiasing (a 50hz)
- Preamplificatore (10db)

microcontrollore:

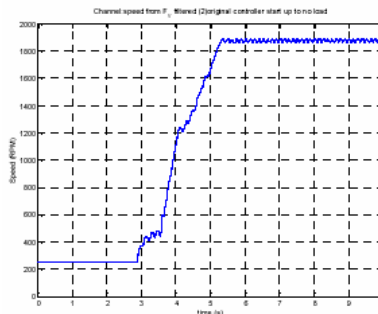
- Convertitore A/D 10 bit
- CU 8051
- Modulatore PWM

## Che implementazione adottare?

- Si può adottare fp, int32 o int16
  - Nel passaggio >> si perde in precisione, ma si diminuisce il tempo di esecuzione
  - La maggior potenza a disposizione potrebbe essere usata per
    - Aumentare la frequenza di campionamento, compensando gli errori e migliorando la risposta
    - Svolgere altri task
  - In alternativa, si può usare un microcontrollore meno potente (più economico)
- La scelta si fa in base alle prestazioni: si valuta il comportamento nelle varie implementazioni

## Scelta della frequenza di campionamento

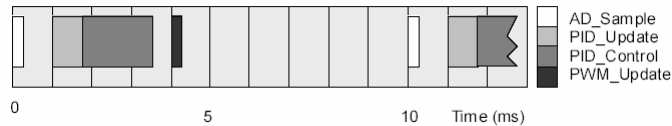
- Il motore deve essere controllato nel range di giri fra 1500 rpm e 1800 rpm
- Si utilizza il criterio del rise time



- Rise\_time=2-2,5 sec
- Si sceglie SAMPLE\_TIME=10msec (circa rise\_time/200: elevato, molto più del dovuto)

## Organizzazione in task

- Quanto di scheduling: 1 msec
- Tutti i task sono stati presi periodici, con periodo di 10msec
  - Task:
    - AD\_Sample: lettura dal convertitore A/D
    - PID\_UPDATE-PID\_CONTROL: implementazione del PID
    - PWM\_UPDATE: generazione del segnale PWM
    - L'attivazione dei task avviene come in figura (sono utilizzati dei ritardi per evitare sovrapposizioni)



Pierfrancesco

Implementazione tramite microcontrollori di PID

0

5

10 Time (ms)

31

## Prestazioni: fp vs int32 vs int 16

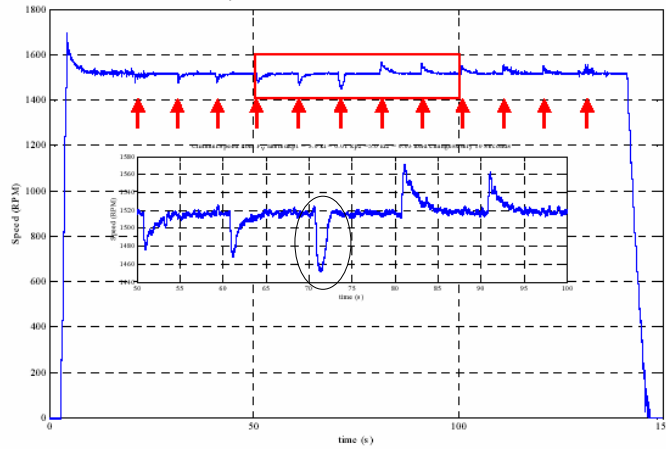
- Vengono implementate 3 versioni dell'algoritmo:
  - Fp
  - INT32
  - INT16
- Viene misurata la velocità del sistema nei 3 casi, applicando un carico al motore, incrementato di 10 KW da 0 a 60 KW, e poi riportato a 0
- Parametri:  $K_p=5$ ,  $K_i=0.03$ ,  $K_d=0.01$

Pierfrancesco Foglia

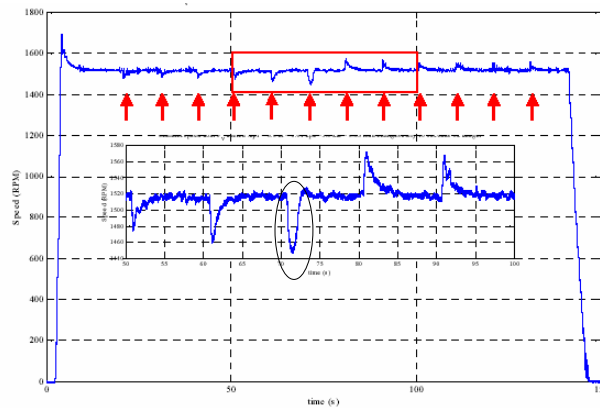
Implementazione tramite microcontrollori di PID

32

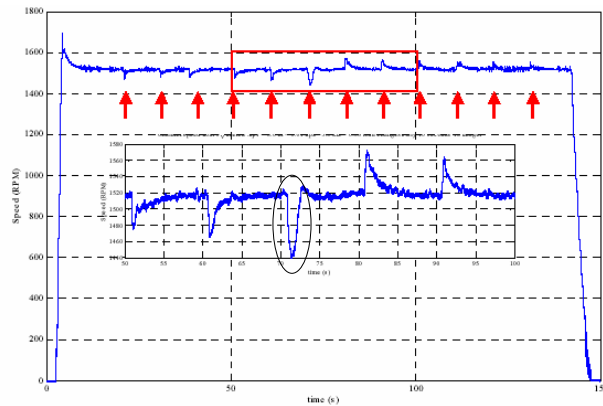
fp



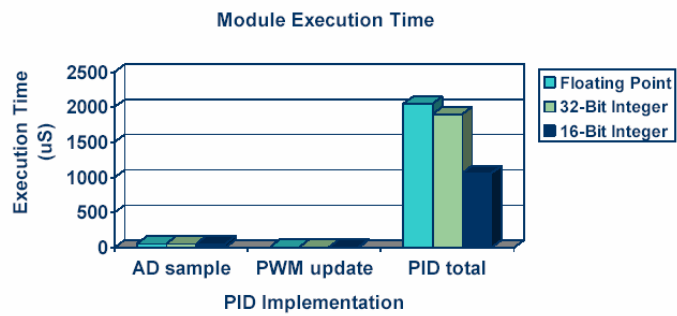
INT32



# INT16



# Tempo di esecuzione

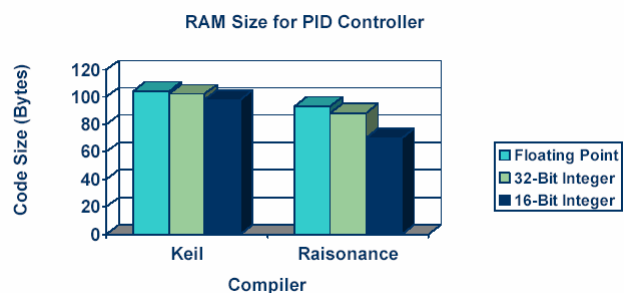
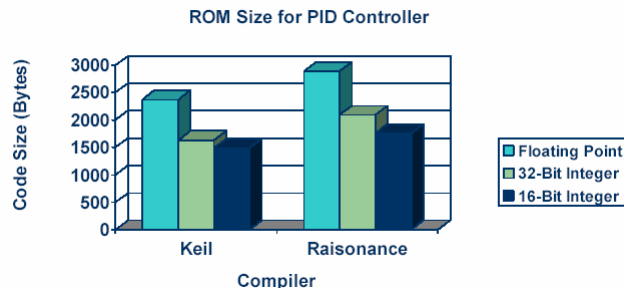


Nota: sono tempi medi

Il tempo di PID varia, in base agli input, per i float

(il best case è migliore degli int32)

## Dimensione del codice e dei dati



Pierfrancesco Foglia  
Implementazione tra

37

## conclusioni

- Per questo problema, si può adottare una implementazione int16
  - Si può aumentare ulteriormente la frequenza di campionamento
  - Oppure si ha a disposizione tempo per svolgere altre funzioni
- Microcontrollori a basse prestazioni (8 bit) sono adeguati per tali classi di problemi
- Controllori a prestazioni più elevate sono indicati per gestire interazioni con l'utente complesse

Pierfrancesco Foglia  
Implementazione tramite microcontrollori di PID

38