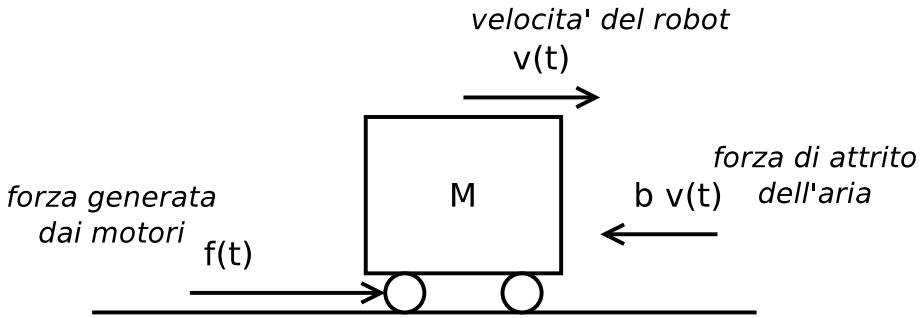
Sistemi di Controllo
Il Controllore Proporzionale

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Programmazione Sistemi Robotici

Controllo del Robot

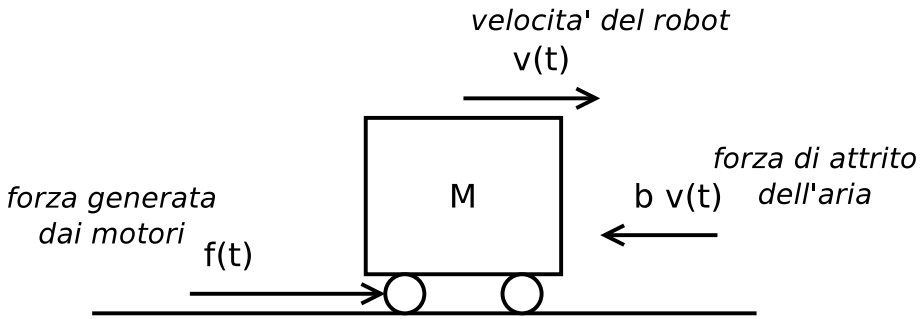


Nel sistema indicato, $f(t)$ è derivante dalla spinta dei motori, provocata a sua volta dall'applicazione di una certa tensione/corrente ai motori stessi.

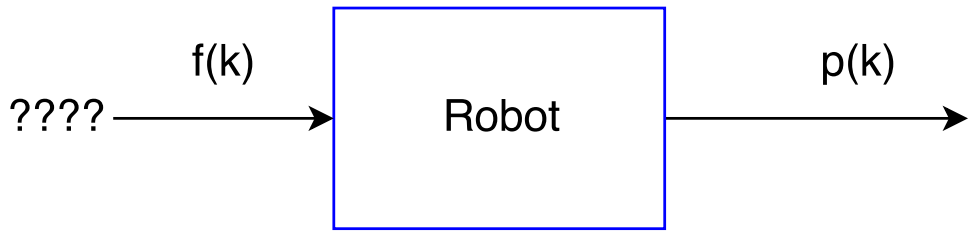
Sappiamo che il sistema nel discreto è modellato come:

$$\begin{bmatrix} v(k+1) \\ p(k+1) \end{bmatrix} = \begin{bmatrix} -\frac{b\Delta t}{M} + 1 & 0 \\ \Delta t & 1 \end{bmatrix} \begin{bmatrix} v(k) \\ p(k) \end{bmatrix} + \begin{bmatrix} \frac{\Delta t}{M} \\ 0 \end{bmatrix} f(k)$$

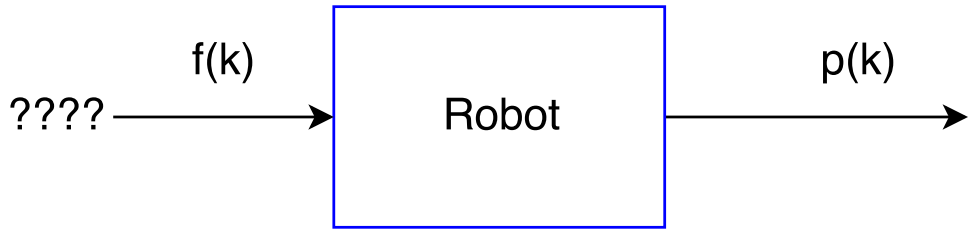
Il Problema del "Controllo"



Desideriamo trovare una $f(k)$ che consenta al robot di raggiungere una posizione $\overline{p}$ specificata; una volta che la posizione è raggiunta, essa deve essere <u>mantenuta</u>



Il Problema del "Controllo"



Desideriamo trovare una $f(k)$ che consenta al robot di raggiungere una posizione $\overline{p}$ specificata; una volta che la posizione è raggiunta, essa deve essere mantenuta

Soluzione 1: **Ciclo Aperto**

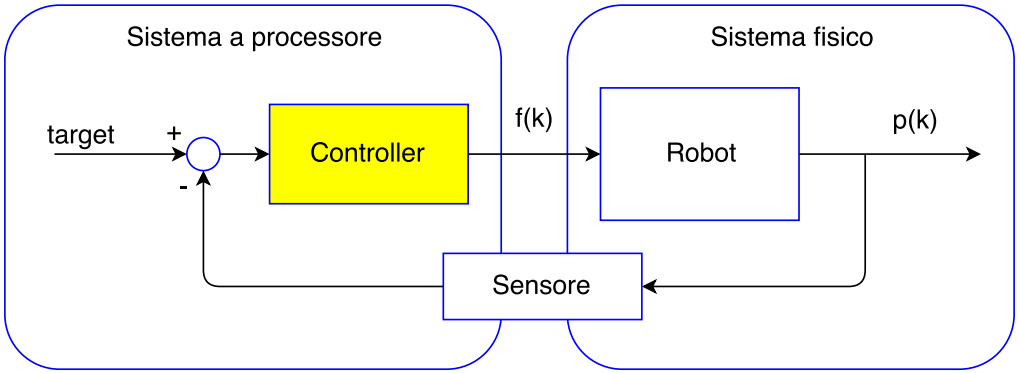
Invertiamo analiticamente il modello e determiniamo la $f(k)$ a partire dalla $\overline{p}$

Non funziona!

- Non è detto che il modello sia (facilmente) invertibile
- Il modello è sempre un'approssimazione della realtà

Il Problema del "Controllo"

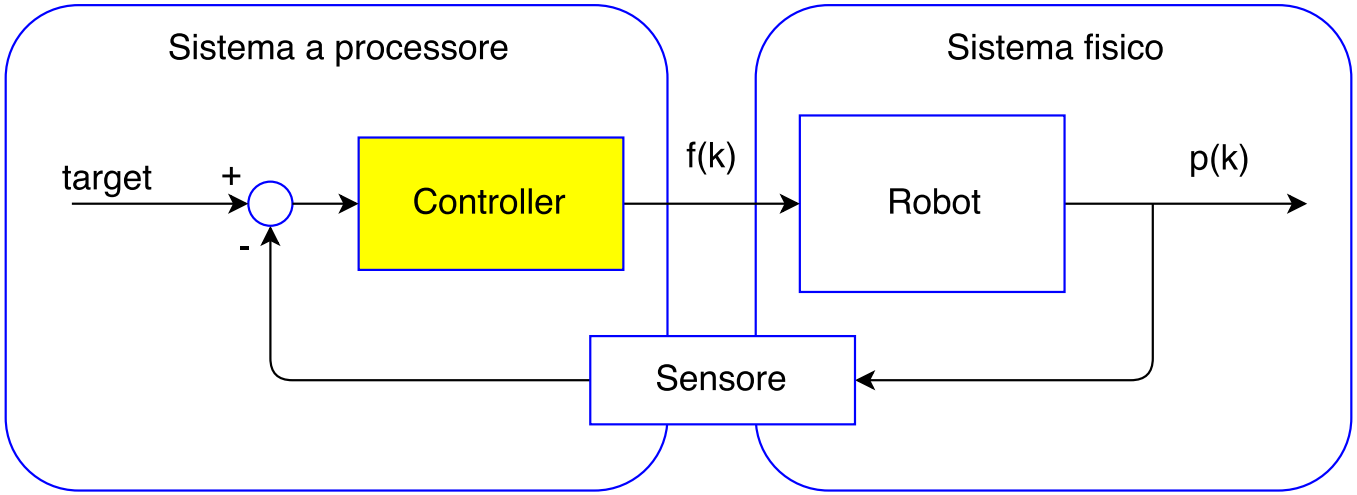
Desideriamo trovare una $f(k)$ che consenta al robot di raggiungere una posizione $\overline{p}$ specificata; una volta che la posizione è raggiunta, essa deve essere mantenuta



Soluzione 2: **Ciclo Chiuso**

- **Leggiamo** con un sensore la posizione **effettiva**
- **Confrontiamola** con il nostro valore desiderato $\overline{p}$ (target)
- Qualora ci sia una **differenza**, usiamo questa differenza per determinare un'appropriata $f(k)$
- Ripetiamo questi tre step **continuamente**, o meglio, **periodicamente**

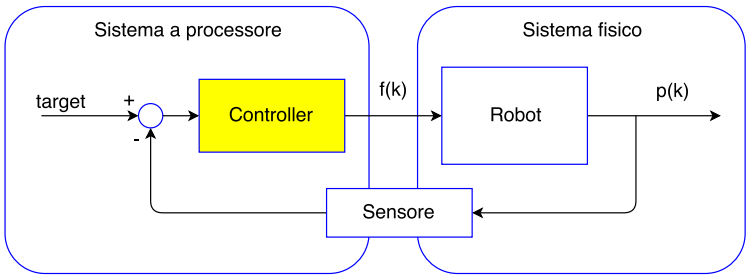
Il Problema del “Controllo”



`doPositionControl(`*target*`)`

while *True* **do**
    On each ΔT;
    $current \leftarrow read\_position\_by\_sensor()$;
    $error \leftarrow target - current$;
    $f \leftarrow controller(error)$;
    $drive\_motors(f)$;
end

Il Problema del "Controllo"



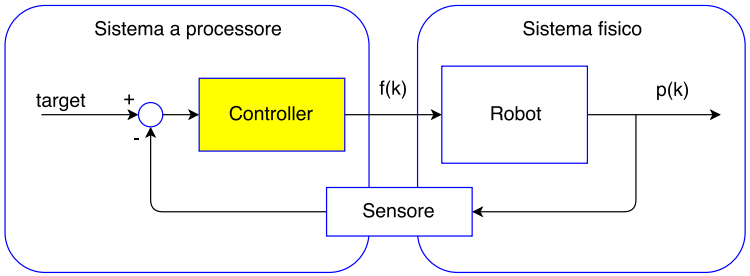
doPositionControl

while *True* **do**
    On each ΔT;
    $current \leftarrow read\_position\_by\_sensor()$;
    $error \leftarrow target - current$;
    $f \leftarrow controller(error)$;
    $drive\_motors(f)$;
end

Funziona!!!!

- Non si basa su un modello teorico ma usa un **dato vero** letto da un sensore
- Non richiede la conoscenza del modello teorico e funziona **su tutti i sistemi**
- Agisce **continuamente** sul sistema e legge l'effetto delle azioni
- In caso di variazioni della posizione **interviene subito** riportandola al target

Il Problema del "Controllo"



doPositionControl

while *True* **do**
    On each ΔT;
    $current \leftarrow read\_position\_by\_sensor()$;
    $error \leftarrow target - current$;
    $f \leftarrow controller(error)$;
    $drive\_motors(f)$;
end

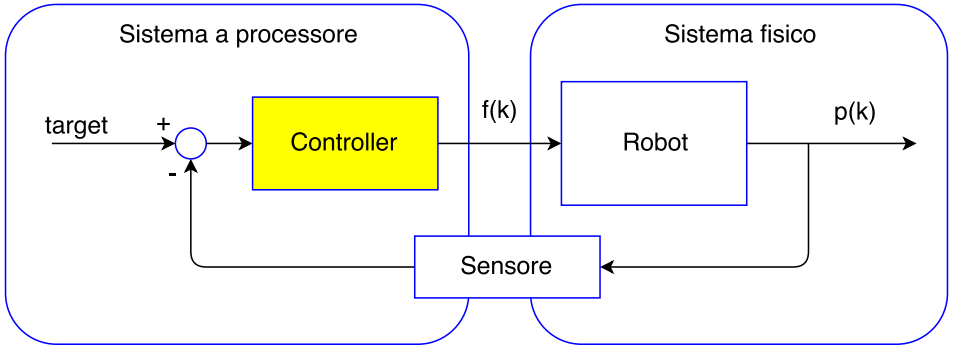
Problematiche da risolvere

- La complessità si sposta però dal modello alla **progettazione del blocco di controllo**
- Il controller è comunque un **sistema dinamico** che occorre sintetizzare (progettare) e successivamente implementare
- **Com'è fatto il controller?**

Un Semplice Controllore per la Posizione

Esempio di controllo in posizione

Controllo in Posizione (semplificato)



Soluzione 1

- Se l'errore è diverso da zero, usiamo una $f(k)$ costante (positiva o negativa a seconda del segno dell'errore)
- Quando l'errore diventa 0, applichiamo $f(k) = 0$, così fermiamo il sistema

Non Funziona!!

- Il sistema (robot) ha una **dinamica**: quando lo stimolo è 0 il sistema **non si ferma istantaneamente!**
- Il target verrà "superato" provocando dunque una "retromarcia"
- Il comportamento a regime sarà **oscillatorio**

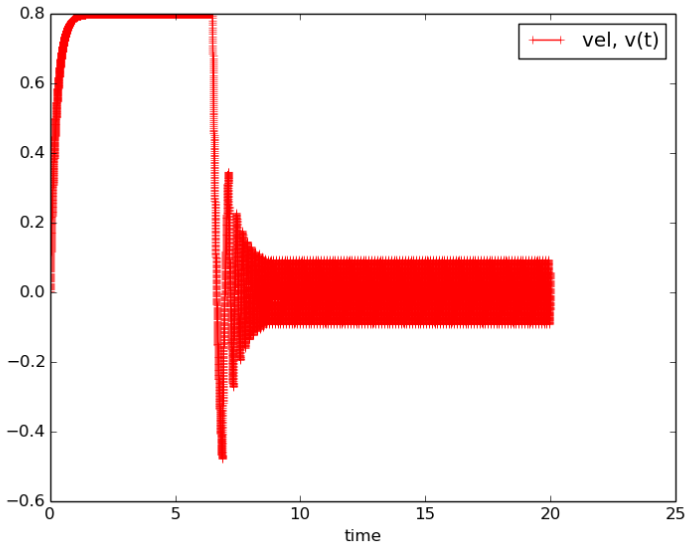
Controllo On-Off

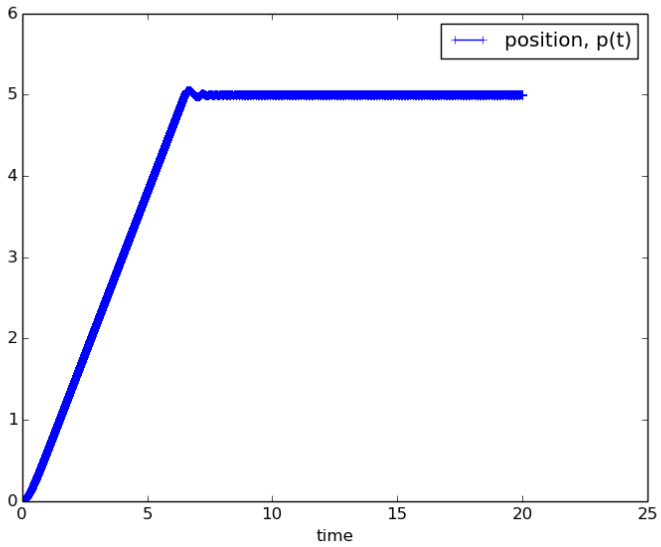
```
class Massa:

    def __init__(self, _M, _b):
        self.M = _M
        self.b = _b
        # variabili di stato, p = posizione, v = velocita'
        self.p = 0
        self.v = 0

    def evaluate(self, _input, dt):
        self.p = self.p + delta_t * self.v
        self.v = (1 - self.b * dt/self.M) * self.v + dt / self.M * _input

    def get_position(self):
        return self.p

    def get_speed(self):
        return self.v
```

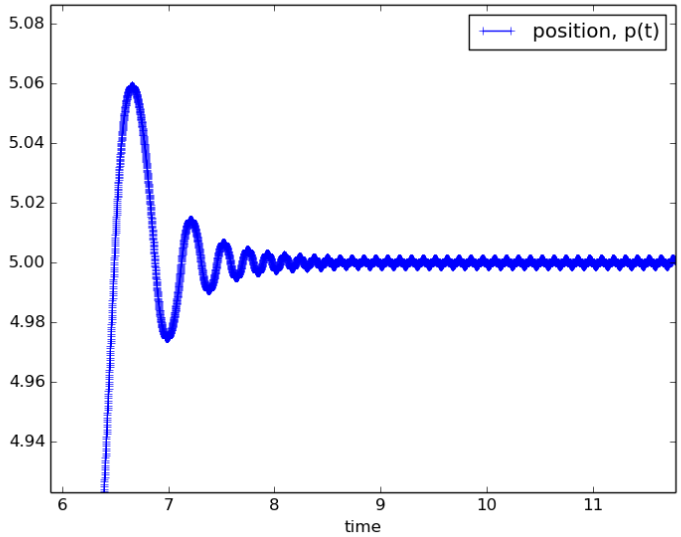
Controllo On-Off

```
delta_t = 1e-3 # 1 ms

robot = Massa(6.0, 25.0)

target_pos = 5  # 5 metri

t = 0.0

while t < 20:  # 20 secondi
    current_pos = robot.get_position()

    error = target_pos - current_pos
    if (error > 0):
        f = 25
    elif (error < 0):
        f = -25
    else:
        f = 0

    robot.evaluate(f, delta_t)

    t = t + delta_t
```

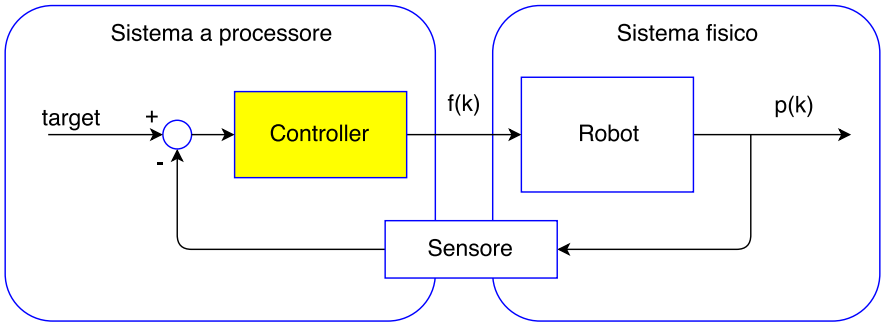
Velocità e Posizione del Controllo On-Off







Controllo in Posizione (semplificato)



Soluzione 2

- Per cercare di rispettare la dinamica, usiamo una $f(k)$ **proporzionale all'errore**
- Se l'errore è grande, lo stimolo applicato sarà grande, facendo sì che il target venga "avvicinato prima possibile"
- Se siamo in prossimità del target applichiamo un stimolo piccolo

$$f(k) = K_P\ e(k) = K_P\ (target - p(k))$$

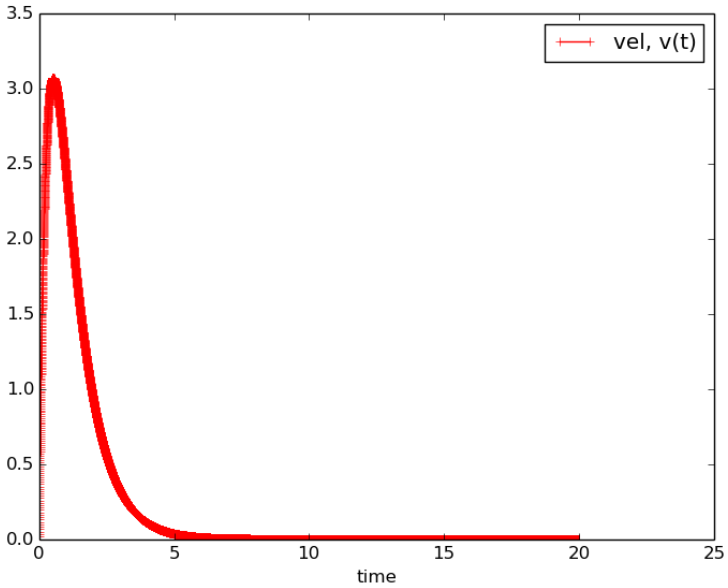
Funziona!!

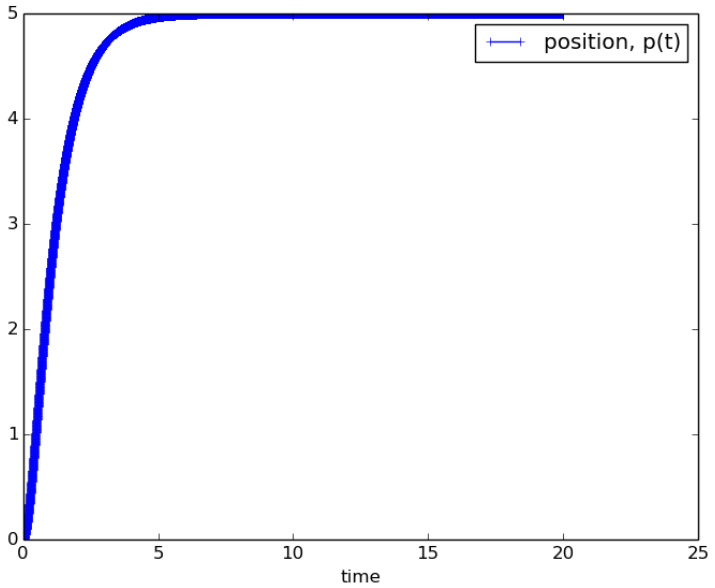
- ... ma occorre "calcolare" o "scegliere" il valore di K_P

Controllo Proporzionale

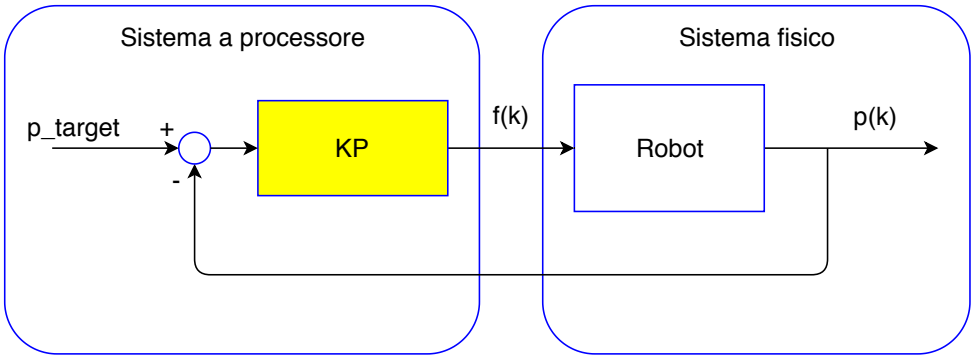
```
delta_t = 1e-3 # 1 ms

robot = Massa(6.0, 25.0)

target_pos = 5  # 5 metri

t = 0.0

kp = 10

while t < 20:  # 20 secondi
    current_pos = robot.get_position()

    error = target_pos - current_pos

    f = kp * error

    robot.evaluate(f, delta_t)

    t = t + delta_t
```

Velocità e Posizione del Controllo P





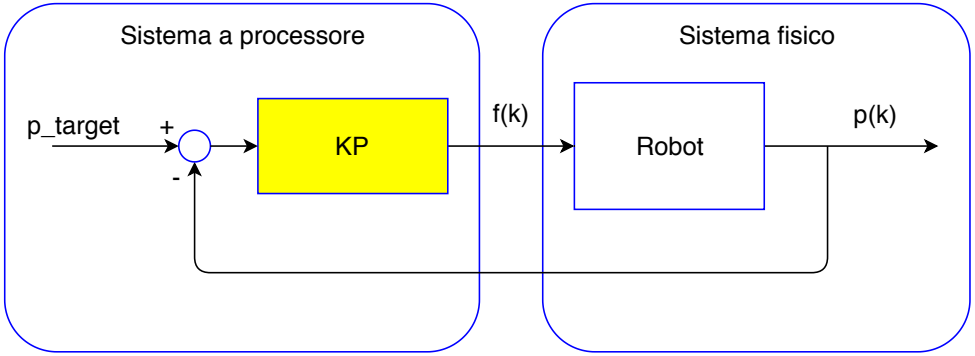
Controllo Proporzionale



Controllo “P”

- Lo schema studiato è noto come **controllo proporzionale** o semplicemente **controllo P**
- Il principio è quello di applicare un “comando” che sia **proporzionale** all’errore nella grandezza da controllare

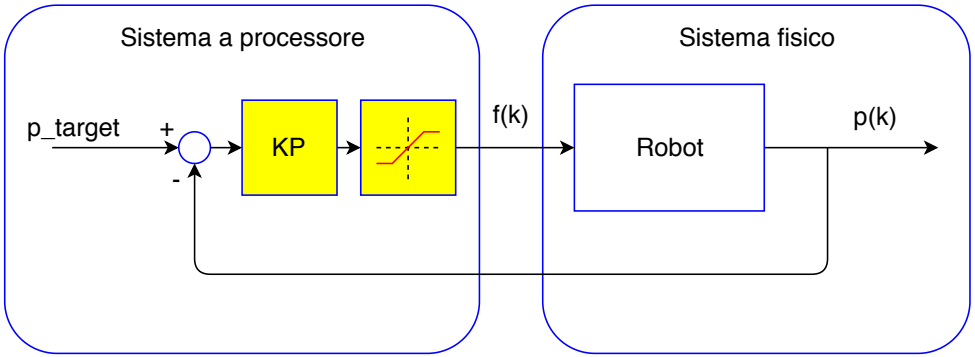
Controllo Proporzionale



Controllo “P” e Saturazione

- Tuttavia, quando l’errore è molto grande, può accadere che il comando generato sia **troppo grande**
- Ogni sistema fisico ha sempre un **limite massimo** che non occorre superare, pena (a volte!) la “distruzione” del sistema stesso o parte di esso
- Esempio:
  - Un motore elettrico ha una tensione di lavoro e, a quella tensione, genera la sua coppia massima; se pilotiamo il motore con una tensione **maggiore** rischiamo di bruciarlo

Controllo Proporzionale



Controllo “P” e Saturazione

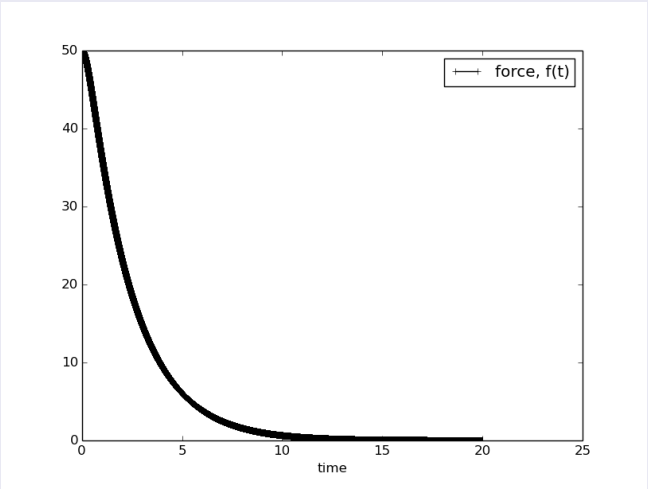
- Per ovviare al problema, le uscite dei controllori sono sempre **“saturate”** ...
- ... cioè limitate nell’intervallo $[-MAX, MAX]$
- Il blocco posto in cascata al controllore P è detto **saturatore**
- Esso si implementa con dei semplici “if”

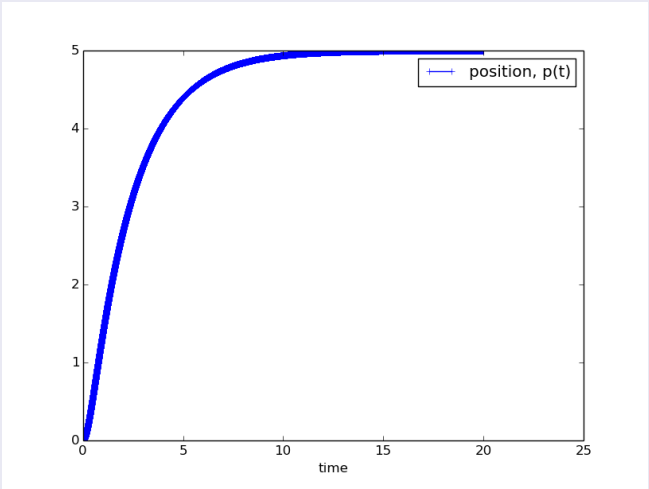
Controllo Proporzionale con Saturazione

```
delta_t = 1e-3 # 1 ms

robot = Massa(6.0, 25.0)

target_pos = 5  # 5 metri

t = 0.0

kp = 10

while t < 20:  # 20 secondi
    current_pos = robot.get_position()

    error = target_pos - current_pos

    f = kp * error
    # saturazione nell'intervallo [-15, 15]
    if (f > 15):
        f = 15
    elif (f < -15):
        f = -15

    robot.evaluate(f, delta_t)

    t = t + delta_t
```

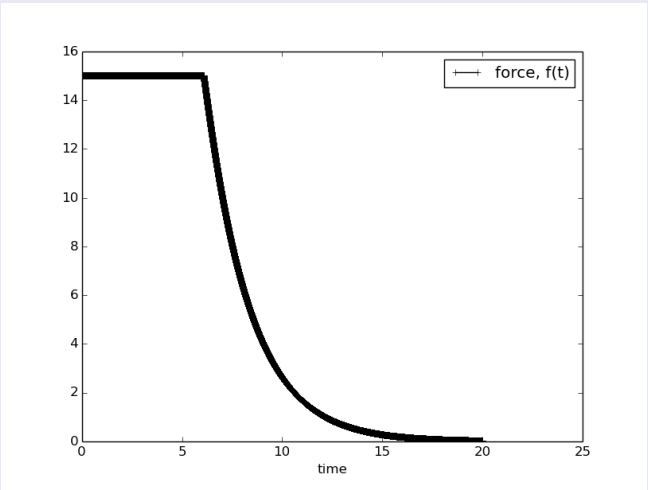
Confronto con e senza Saturazione

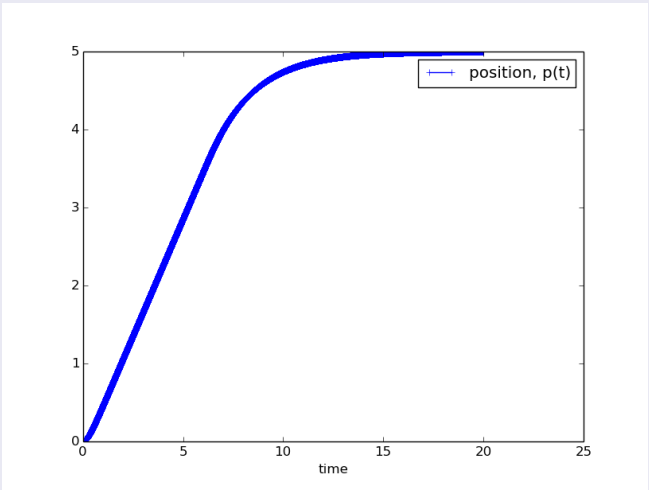
Senza saturazione





Con saturazione





Sistemi di Controllo
Il Controllore Proporzionale

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Programmazione Sistemi Robotici