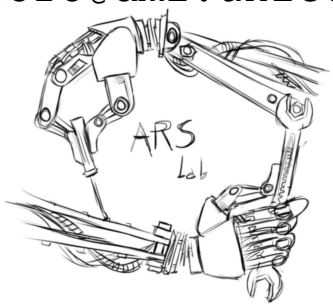
Sistemi di Controllo
Il Controllore Proporzionale-Integrale

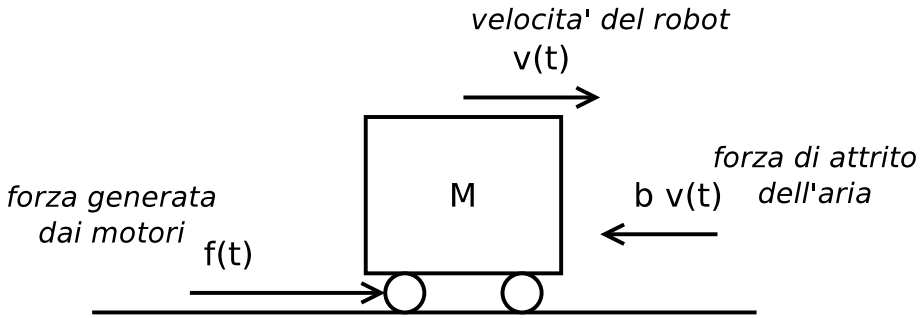
Corrado Santoro

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

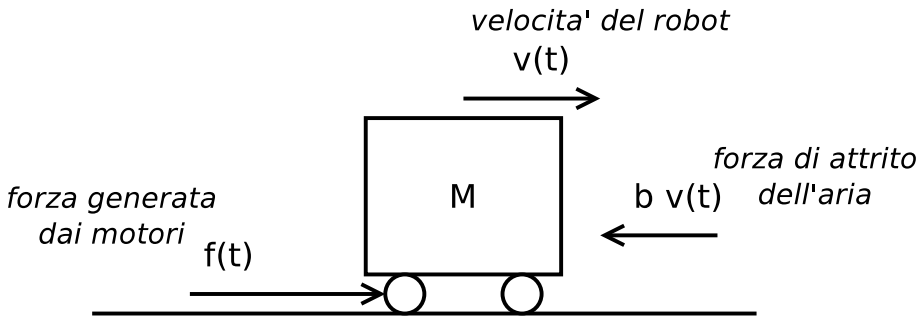


Nel sistema indicato, $f(t)$ è derivante dalla spinta dei motori, provocata a sua volta dall'applicazione di una certa tensione/corrente ai motori stessi.

Sappiamo che il sistema nel discreto è modellato come:

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

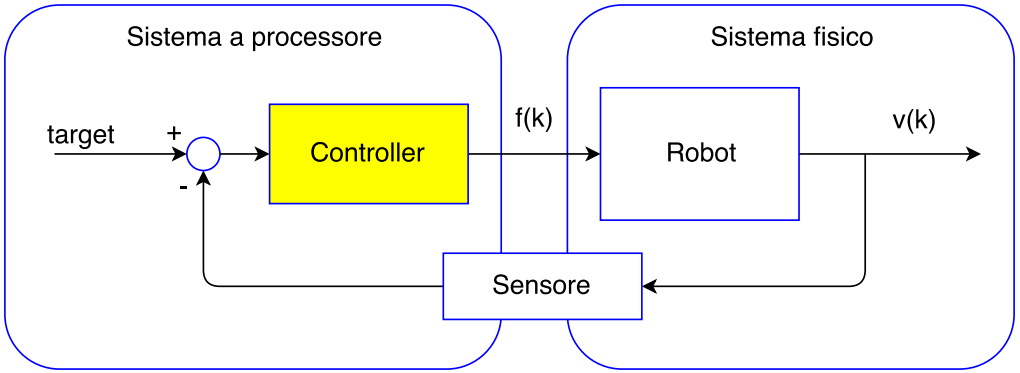
Il Problema del “Controllo”



Desideriamo trovare una $f(k)$ che consenta al robot di andare ad una velocità $\overline{v}$ specificata e di mantenerla

Il Problema del "Controllo"

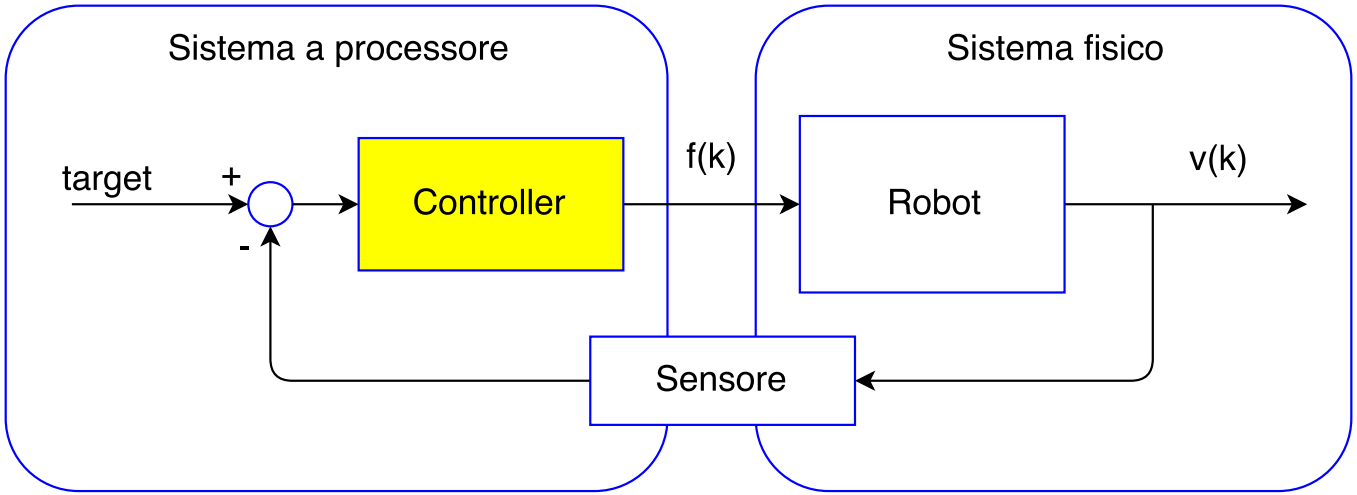
Desideriamo trovare una $f(k)$ che consenta al robot di andare ad una velocità $\overline{v}$ specificata e di mantenerla



Soluzione a **Ciclo Chiuso**

- Usiamo un **sensore di velocità** e confrontiamo il valore con il target
- Qualora ci sia una **differenza**, usiamo questa differenza per determinare un'appropriata $f(k)$

Il Problema del “Controllo”

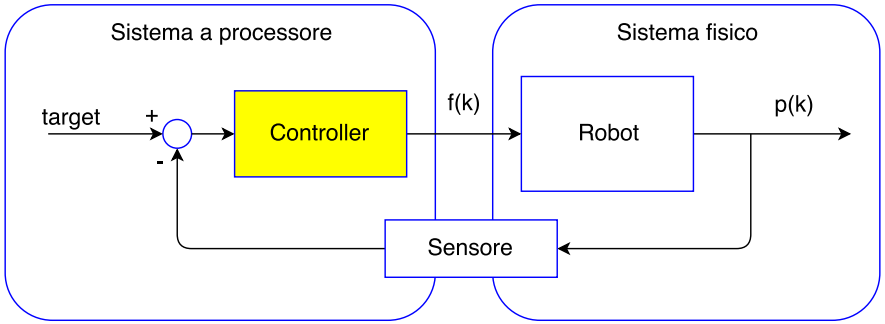


Possiamo usare anche in questo caso un **controllore proporzionale** ?

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

NO!

Rivediamo il Controllo in Posizione ...

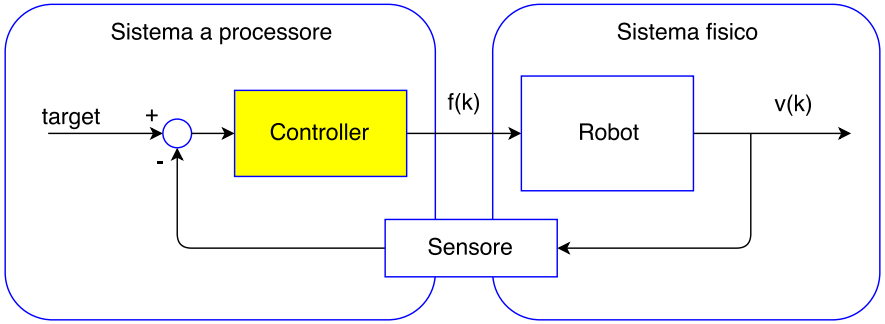


L'output "zero/non-zero" del controllore

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

- Quando il target **è raggiunto**, $p(k) = target$ il sistema deve essere **fermo**
- Pertanto è corretto che, in tali condizioni, l'output del controllore sia 0
- Quando il target **non è raggiunto**, il sistema deve essere **azionato** (output non-zero)

Applichiamo al Controllo in Velocità ...

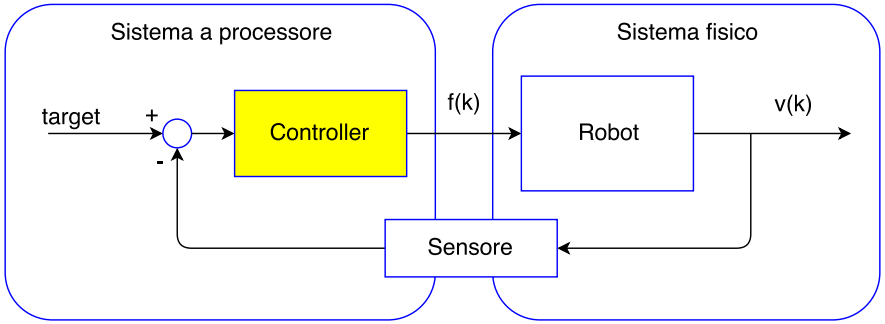


L'output "zero/non-zero" del controllore

$$f(k) = K_P\, e(k) = K_P\, (target - v(k))$$

- Quando il target **è raggiunto**, $v(k) = target$, il sistema deve essere **in moto a velocità costante**
- Ma se l'output del controllore è 0, il sistema **si fermerà**, pertanto un semplice controllo proporzionale **non va bene**
- Occorre un controllore che **continui a produrre una spinta** (costante) anche quando il target è stato raggiunto

Applichiamo al Controllo in Velocità ...



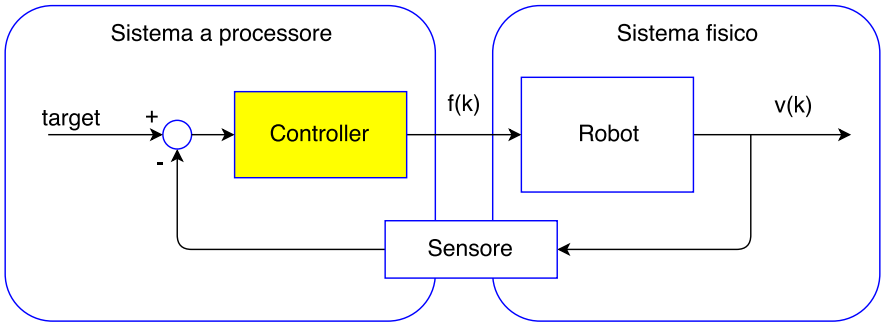
L'output "zero/non-zero" del controllore

- Occorre un controllore che **continui a produrre una spinta** (costante) anche quando il target è stato raggiunto
- Ad errore zero, la spinta **non deve variare**
- Allora mi serve un controllore che faccia **variare la spinta** in modo proporzionale all'errore, secondo un coefficiente Q

$$f(k) = f(k-1) + Q\, e(k)$$

- Il controllore deve essere un **"accumulatore di errore"**, in altri termini, un **integratore**

Controllore Integratore



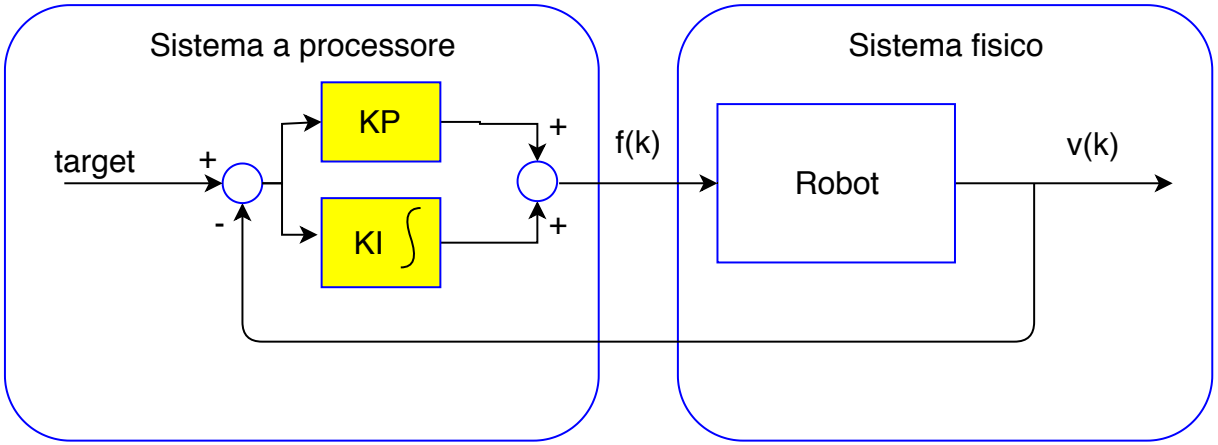
Controllore "I"

$$f(t) = K_I \int_0^t e(\tau) d\tau$$

Discretizzando:

$$f(k) = K_I \sum_{j=0}^{k} e(j) \Delta t$$

Controllore Proporzionale-Integrale



Controllore "PI"

Combinando il controllore proporzionale e il controllore integrale otteniamo:

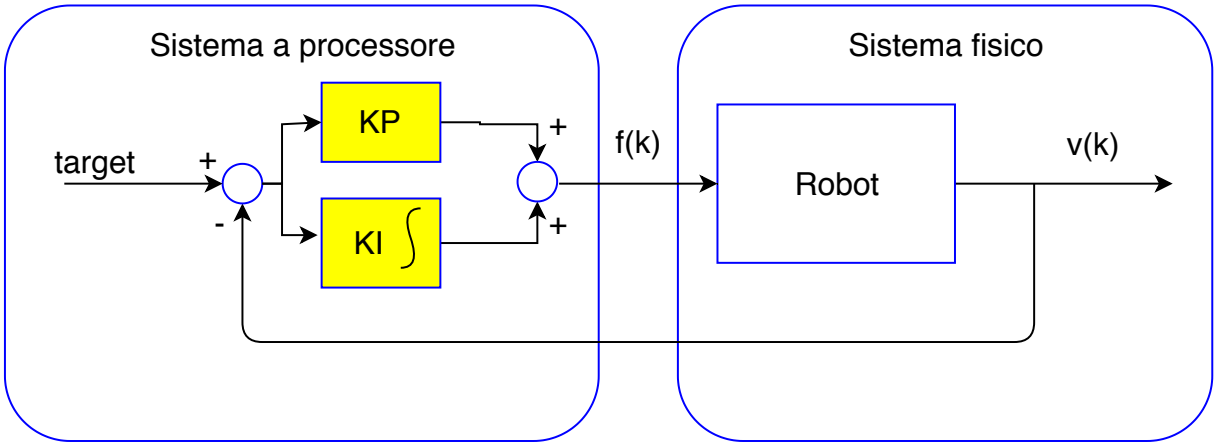
$$f(t) = K_P\, e(t) + K_I \int_0^t e(\tau) d\tau$$

Discretizzando:

$$f(k) = K_P\, e(k) + K_I \sum_{j=0}^{k} e(j) \Delta t$$

Dove K_P e K_I sono le **costanti di controllo**

Controllore Proporzionale-Integrale



Controllore "PI"

Da cui la *legge di aggiornamento*:

$$g(k) = g(k-1) + e(k)\Delta t$$

$$f(k) = K_P\, e(k) + K_I g(k)$$

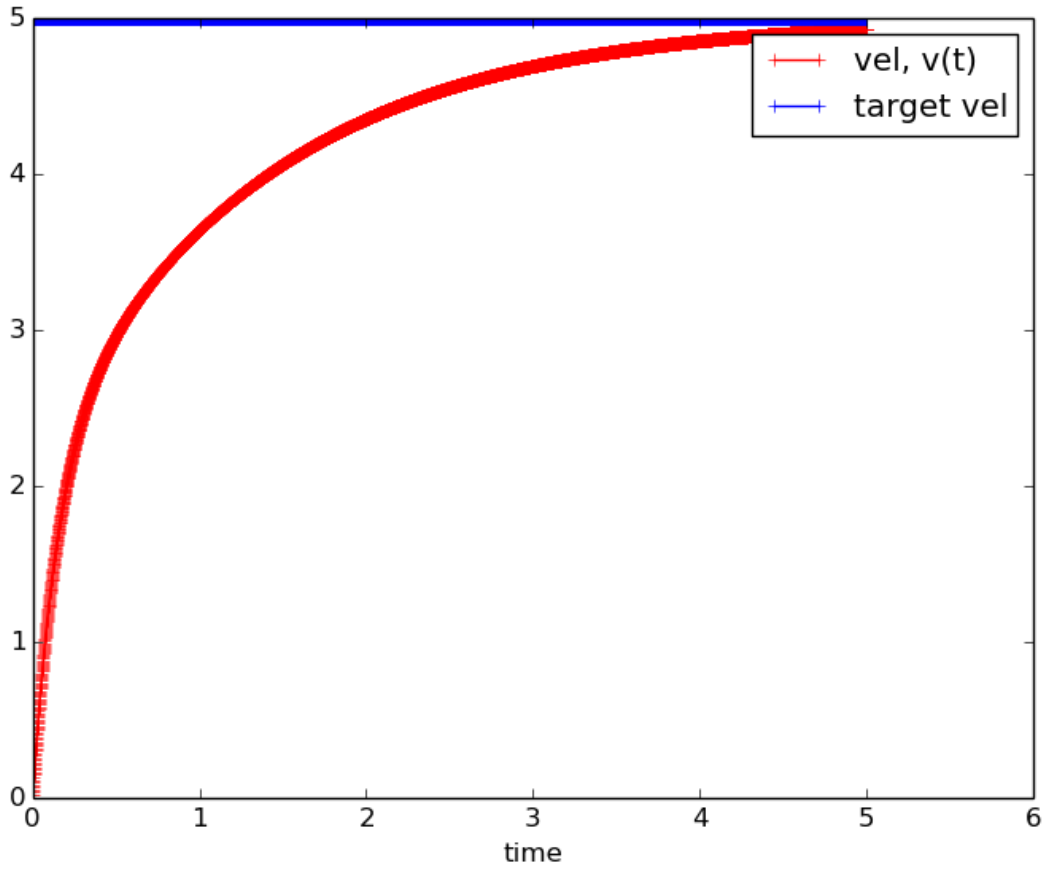
Dove $g(k)$ è il **fattore integrale** e K_P e K_I sono le **costanti di controllo**

Controllo Proporzionale/Integrale

```
class PI:

    def __init__(self, kp, ki):
        self.kp = kp
        self.ki = ki
        self.integral = 0

    def evaluate(self, target, current, delta_t):
        error = target - current
        self.integral = self.integral + error * delta_t
        output = self.kp * error + self.ki * self.integral
        return output
```

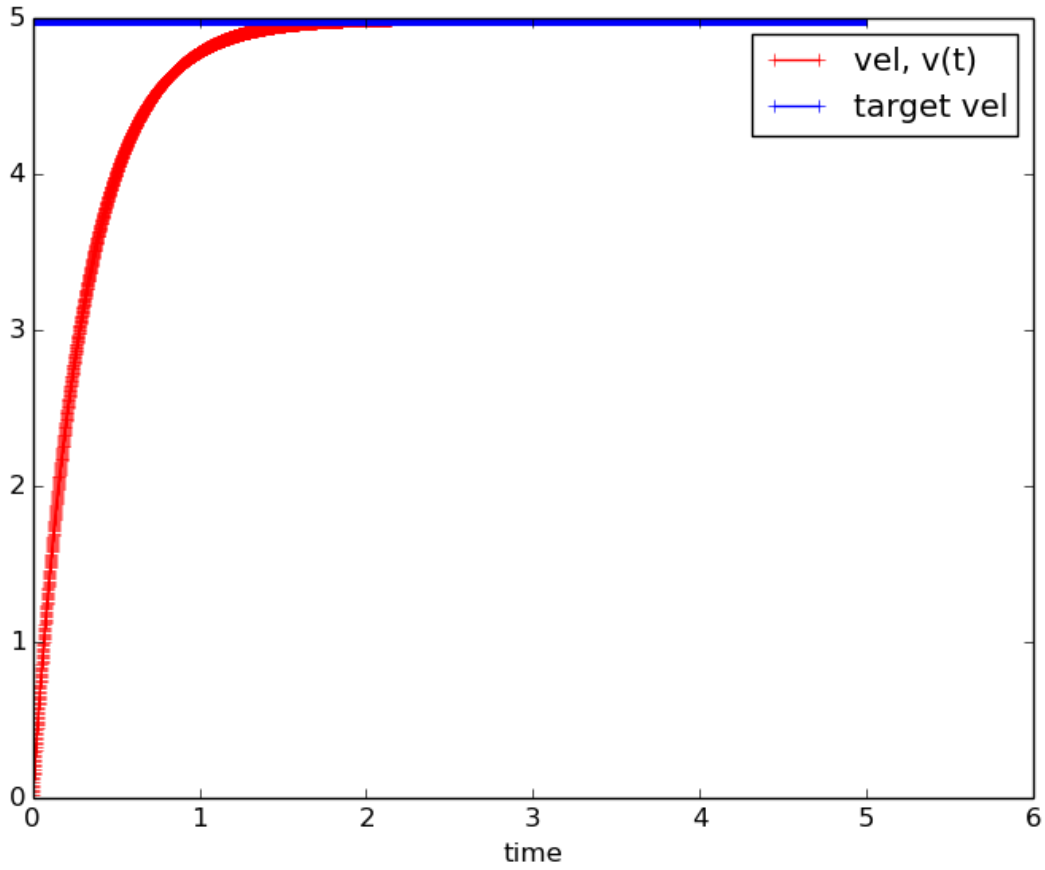
Esempio di Risposta al gradino

$$target = 5m/s, M = 6kg, b = 25Ns^2/m$$
$$K_P = 20, K_I = 10$$



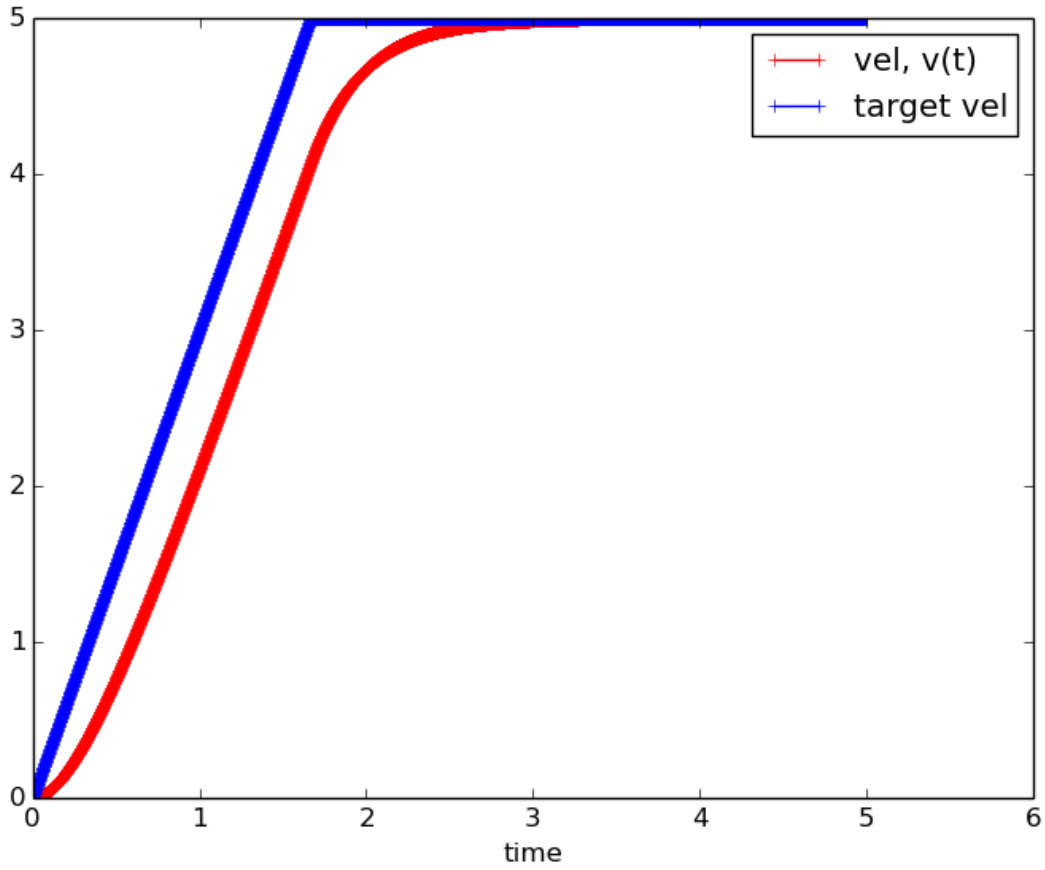
Esempio di Risposta al gradino

$$target = 5m/s, M = 6kg, b = 25Ns^2/m$$
$$K_P = 20, K_I = 80$$



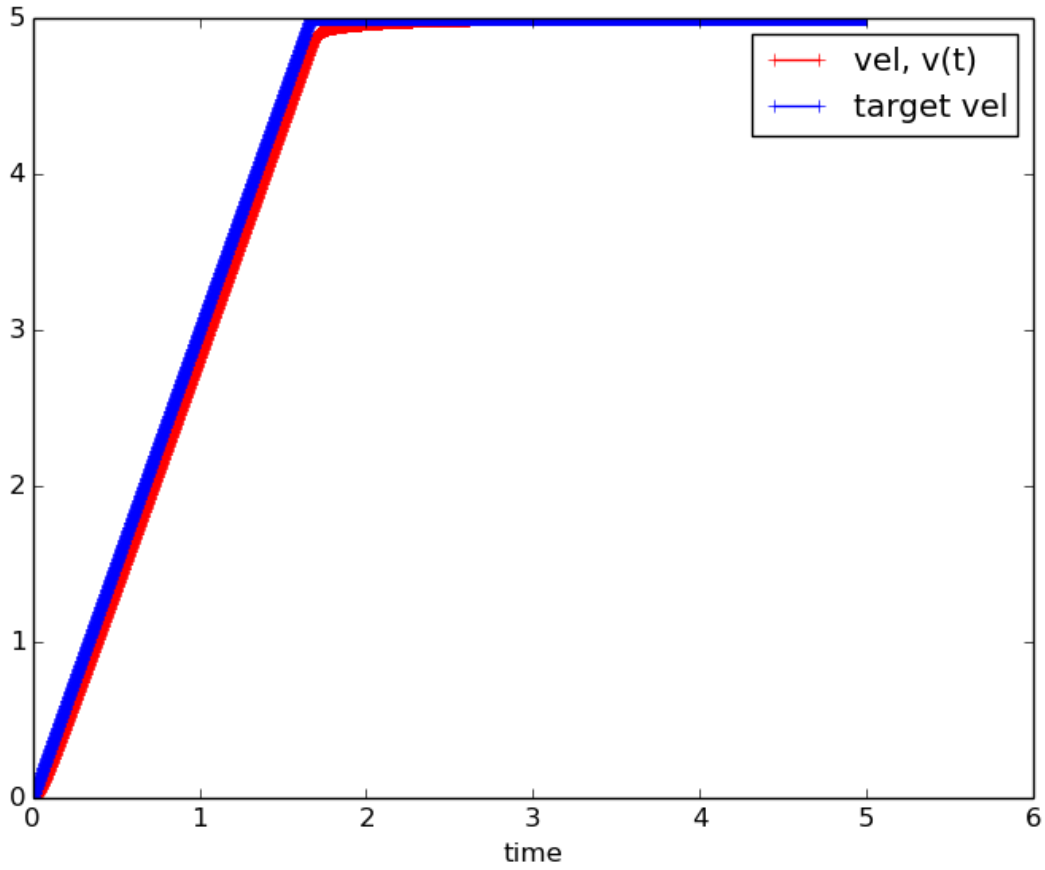
Esempio di Risposta alla rampa

$$accelerazione = 3m/s^2, M = 6kg, b = 25Ns^2/m$$
$$K_P = 20, K_I = 80$$

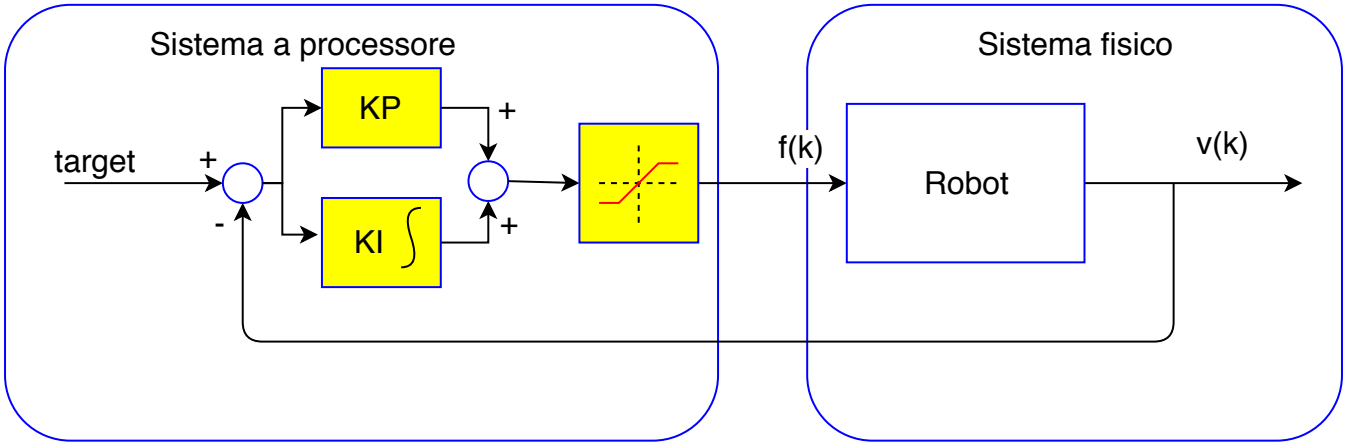


Esempio di Risposta alla rampa

$$accelerazione = 3m/s^2, M = 6kg, b = 25Ns^2/m$$
$$K_P = 150, K_I = 400$$



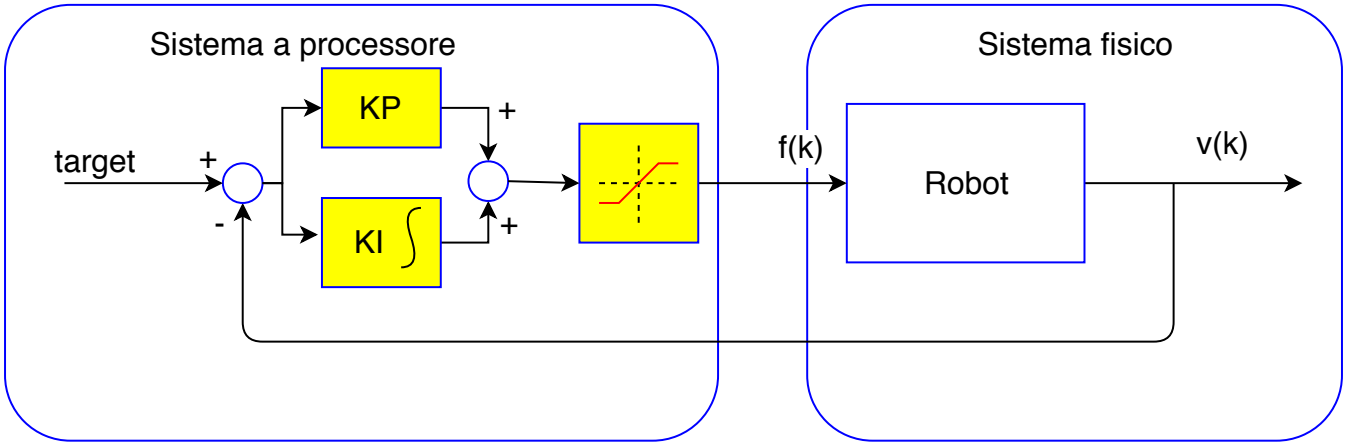
Controllo PI con Saturazione



Controllo "PI" e Saturazione

- Anche con i controllori PI si usa **"saturare"** l'uscita in un intervallo $[-MAX, MAX]$
- Si aggiunge dunque in cascata un blocco **saturatore**
- La saturazione, quando c'è un termine integrale, va tuttavia trattata con cura

Controllo PI con Saturazione e Anti-Windup



Controllo "PI" e Saturazione e Anti-Windup

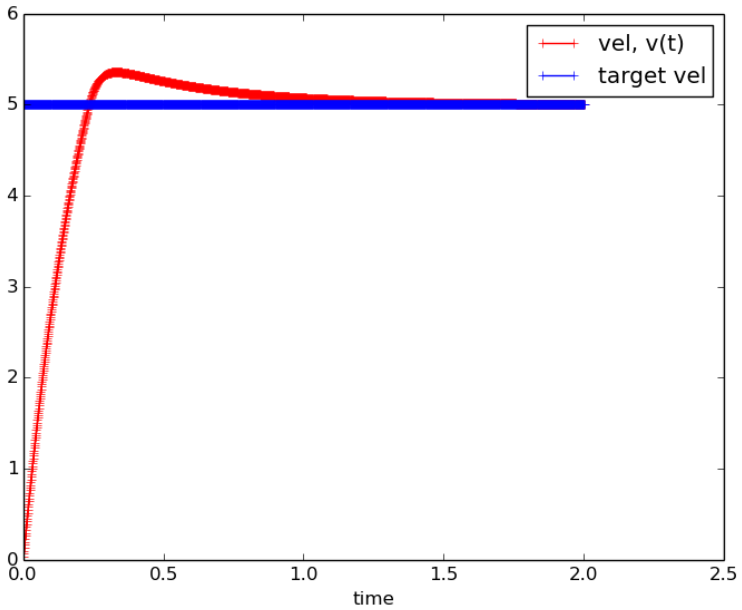
- Quando il sistema è saturato, il sistema è **ai limiti**, e l'errore non potrà mai ridursi secondo quanto ci si aspetta
- Poichè l'integratore **accumula** l'errore, quando siamo in saturazione è **inutile** accumulare errore che **non potrà ridursi**
- Per tale motivo, in *condizioni di saturazione*, si preferisce **non calcolare il termine integrale**
- Tale ottimizzazione è detta **anti-windup**

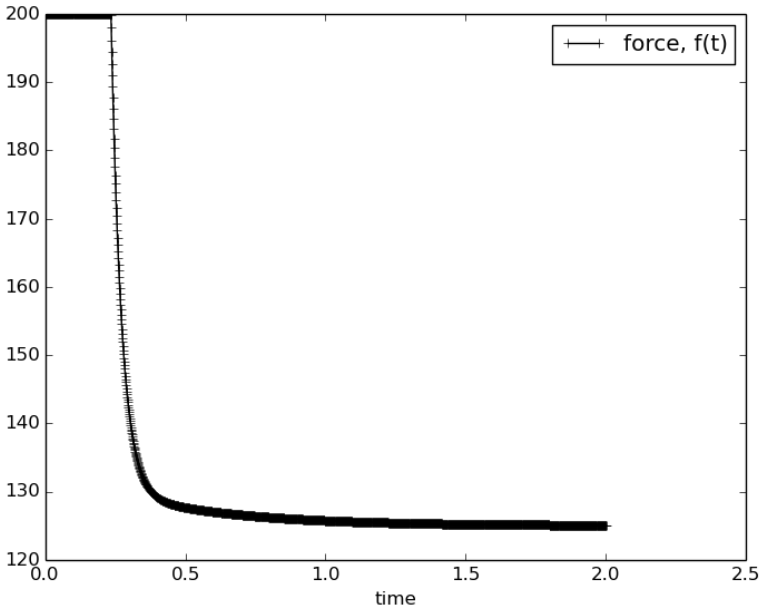
Controllo Proporzionale/Integrale con Saturazione

```
class PISat:

    def __init__(self, kp, ki, sat):
        self.kp = kp
        self.ki = ki
        self.saturation = sat
        self.integral = 0

    def evaluate(self, target, current, delta_t):
        error = target - current
        self.integral = self.integral + error * delta_t
        output = self.kp * error + self.ki * self.integral
        if output > self.saturation:
            output = self.saturation
        elif output < -self.saturation:
            output = -self.saturation
        return output
```

Esempio di Risposta al gradino con saturazione

$$target = 5m/s^2, M = 6kg, b = 25Ns^2/m$$
$$K_P = 150, K_I = 400, Sat = 200N$$



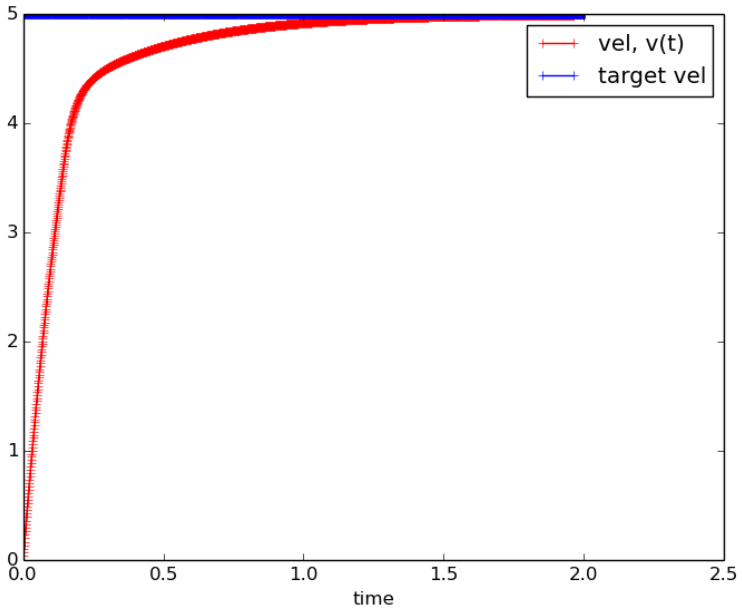


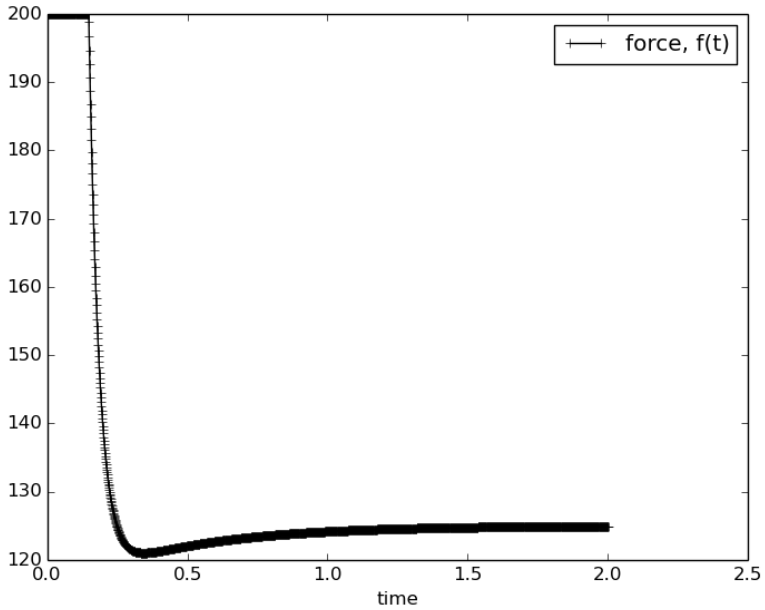
Controllo PI con Saturazione e Anti-Windup

```
class PISat:

    def __init__(self, kp, ki, sat):
        self.kp = kp
        self.ki = ki
        self.saturation = sat
        self.integral = 0
        self.saturation_flag = False

    def evaluate(self, target, current, delta_t):
        error = target - current
        if not(self.saturation_flag):
            self.integral = self.integral + error * delta_t
        output = self.kp * error + self.ki * self.integral
        if output > self.saturation:
            output = self.saturation
            self.saturation_flag = True
        elif output < -self.saturation:
            output = -self.saturation
            self.saturation_flag = True
        else:
            self.saturation_flag = False
        return output
```

Risposta al gradino con saturazione e Anti-Windup

$$target = 5m/s^2, M = 6kg, b = 25Ns^2/m$$
$$K_P = 150, K_I = 400, Sat = 200N$$

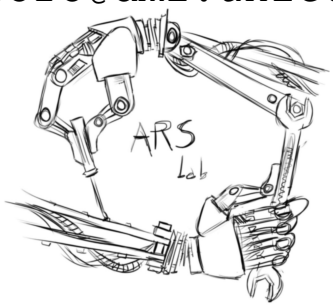




Sistemi di Controllo
Il Controllore Proporzionale-Integrale

Corrado Santoro

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



Programmazione Sistemi Robotici