Università degli Studi di Perugia

Facoltà di Ingegneria

Corso di Laurea Triennale in
Ingegneria Informatica ed Elettronica

Tesina di Ingegneria dei Sistemi di Controllo

PID: controllo del moto di una particella

Studente
David Polzoni

Professore
Paolo Valigi

Anno Accademico 2020-2021

0. Indice

1. PID: moto di una particella

Si vogliono evidenziare gli effetti dei singoli termini di un regolatore PID in uno scenario in cui è di interesse il controllo del moto di una particella unidimensionale, "governata" da forze, in un ambiente n-dimensionale. Il modello differenziale che descrive il processo viene ricavato a partire dalla legge fisica: $F = m\,a(t)$, dove F è la somma di tutte le forze applicate alla particella (comprese quelle esterne di disturbo). Ai fini della caratterizzazione del processo, sono di interesse le seguenti equazioni: $a(t) = \ddot{p}(t)$ e $v(t) = \dot{p}(t)$, dove $p(t)$ indica la posizione della particella. Si definisce il vettore di stato $x = [x_1,\ x_2]^T$ con $x_1 := p(t)$ e $x_2 := v(t)$, il segnale d'ingresso $u(t) := F$ (somma delle forze agenti sul processo) e infine, in uscita, si ha la posizione della particella: $y = x_1$. Di seguito, si riportano il modello differenziale e il modello in forma di spazio di stato.

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{m}\,u \quad \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = x_0 \\ y &= x_1 \end{aligned} \tag{1.1}$$

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u,\ x(0) = x_0,\ x \in \mathbb{R}^2,\ u \in \mathbb{R} \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x,\ x(0) = x_0,\ y \in \mathbb{R} \end{aligned} \tag{1.2}$$

Nell'implementazione realizzata, è previsto un auto-tuning del regolatore PID utilizzando l'algoritmo Twiddle [1, 2]. L'obiettivo è quello di minimizzare il CTE (Cross Track Error), ovvero l'errore di inseguimento della traiettoria desiderata. In `particle_trajectory.py` viene chiesto a una particella unidimensionale di seguire una traiettoria che cambia nel tempo; la traiettoria del set-point è data dalla funzione gradino unitario (da $t = 5$ s a $t = 15$ s). Quest'ultima può essere

modificata arbitrariamente agendo sul metodo `set_point(self, t)`. Si assume che la posizione della particella non possa essere controllata direttamente, ma che il regolatore PID emetta una forza di controllo, applicata al processo, in modo tale da controllarne (indirettamente) il moto. La figura 1.2 (a) illustra gli effetti dei diversi termini del regolatore PID sul controllo della traiettoria della particella. In blu è indicata la traiettoria del set-point impostato, in giallo l'effetto del regolatore P (termine proporzionale) sul controllo del moto, in verde viene presentato l'effetto del regolatore PD (termine proporzionale e derivativo); infine, in rosso è indicata la traiettoria della particella dopo l'auto-tuning del regolatore (PD). Il processo dinamico viene implementato ereditando `ctrl.Process` e sovrascrivendo i metodi indicati di seguito.

```
class MoveParticleProcess(ctrl.Process):

    def __init__(self, particle=ctrl.Particle(), pid=ctrl.PID()):
        super(MoveParticleProcess, self).__init__()
        self.particle = particle
        self.pid = pid

    # Return set-point position for particle to reach

    def set_point(self, t):
    ...

    # Sense particle position

    def sense(self, t):
    ...

    # Compute correction based on error

    def correct(self, error, dt):
    ...

    # Update particle position: the correction value 'u' as force
    acting on the particle; updates the motion equations by 'dt'

    def actuate(self, u, dt):
    ...
```

Listing 1.1: Caratterizzazione del processo dinamico.

Nel modo che segue è possibile simulare il processo dinamico.

```
process = MoveParticleProcess(
    particle=ctrl.Particle(x0=[0], v0=[0], inv_mass=1.),
    pid=ctrl.PID(kp=0.1, ki=0.0, kd=1.0) # PID controller
)
result = process.loop(tsim=100, dt=0.1)
plt.plot(result['t'], result['x']) # Plot trajectory over time
```

Listing 1.2: Simulazione del processo dinamico.

È inoltre possibile sperimentare diversi processi dinamici (e.g. aggiungendo forze esterne a `MoveParticleProcess`). Un esempio viene riportato di seguito.

```
def actuate(self, u, dt):
    self.particle.add_force(u)
    # External force actuates on process
    self.particle.add_force(np.array([-0.5 * self.particle.mass]))
    self.particle.update(dt)
```

Listing 1.3: Aggiunta di una forza esterna agente sul processo (disturbo).

In tal caso, rappresentato nella figura 1.2 (b), si nota che solo un regolatore PID che include un termine integrale permette agevolmente di inseguire la traiettoria desiderata[1]. L'effetto del regolatore PD non basta per rendere l'errore a regime nullo: la traiettoria della particella si stabilizza su una retta al di sotto del set-point. In ultimo, nella figura che segue, viene presentato lo schema a blocchi progettato per il controllo del moto della particella.

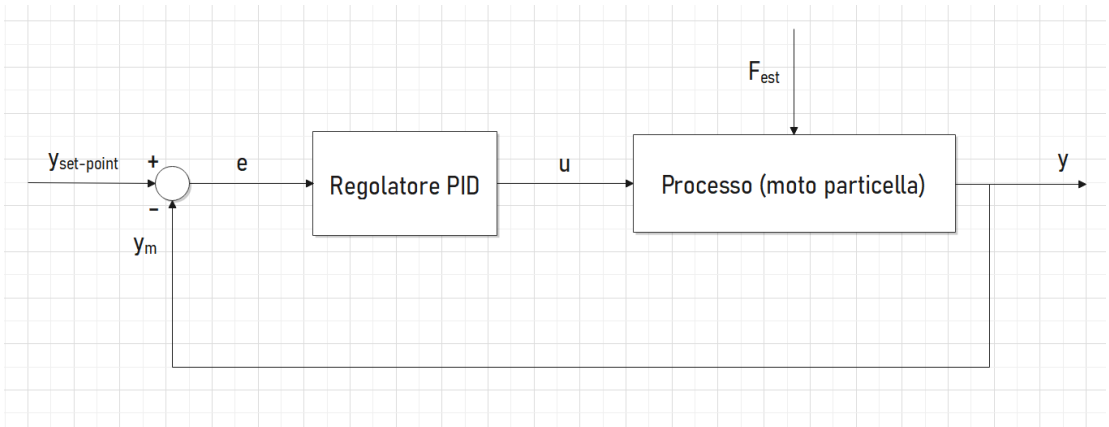
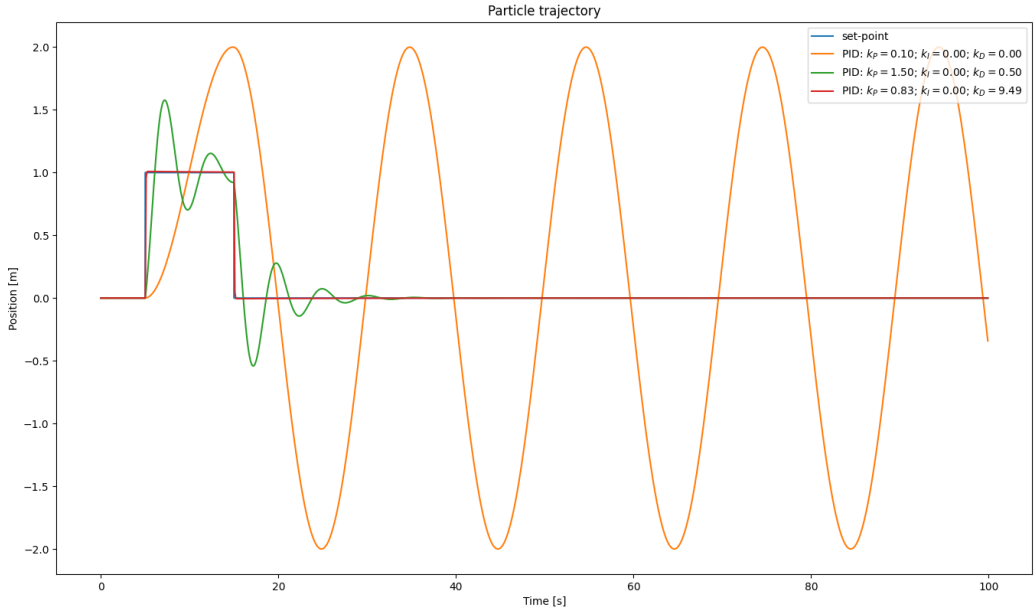
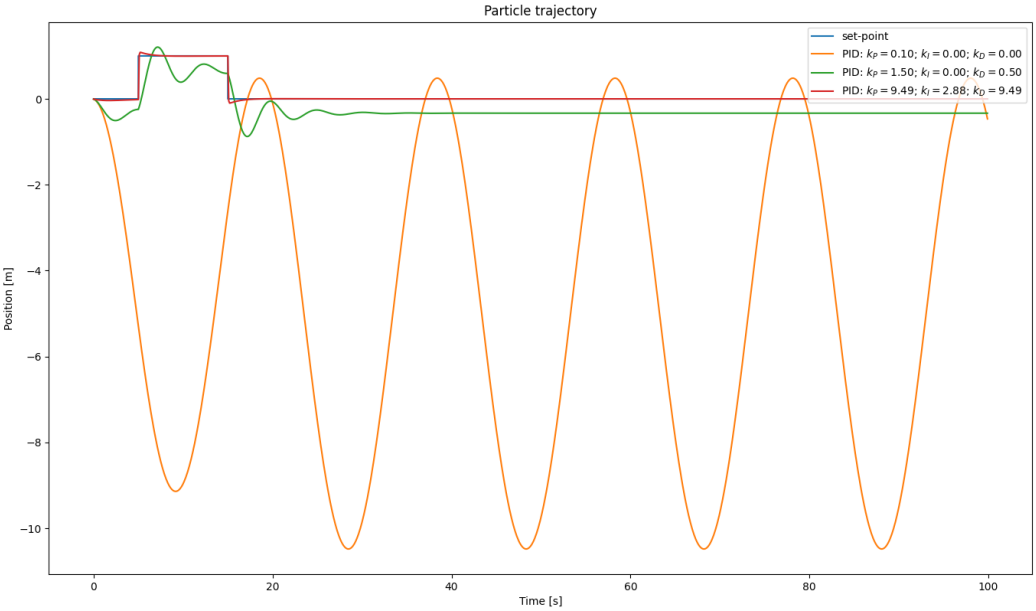


Figura 1.1: Schema a blocchi: controllo del moto di una particella.

[1] Il termine integrale garantisce un errore a regime nullo per riferimenti e disturbi a gradino.



(a) Traiettoria della particella: PD.



(b) Traiettoria della particella: PID.

Figura 1.2: PID: controllo del moto di una particella.

2. Bibliografia

[1] Martin Thoma (06/09/2014), "The Twiddle Algorithm", https://martin-thoma.com/twiddle/;

[2] Sebastian Thrun - Udacity (19/03/2012), "Twiddle - Artificial Intelligence for Robotics", https://www.youtube.com/watch?v=2uQ2BSzDvXs&ab_channel=Udacity.