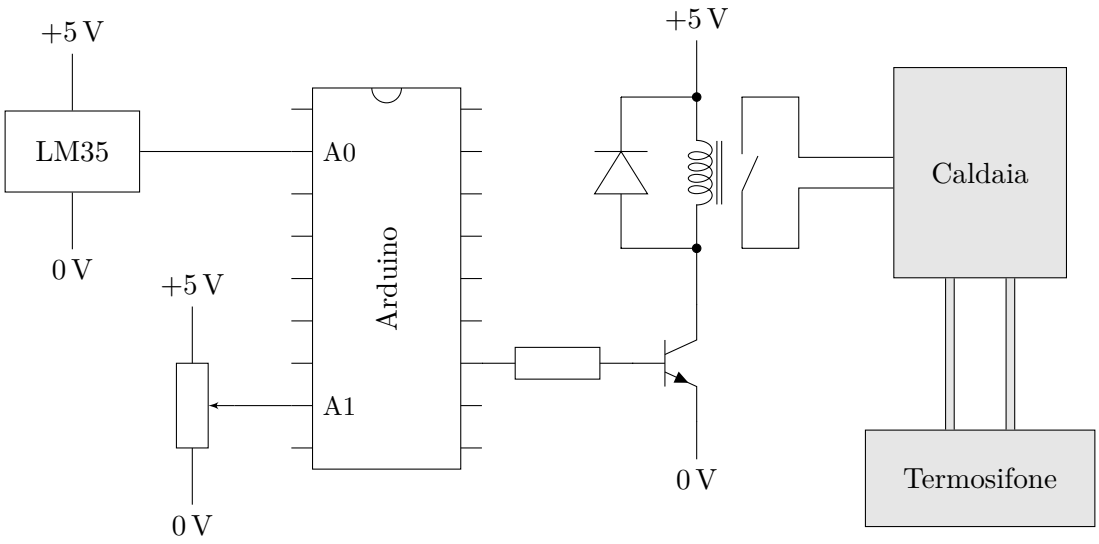
Azzolini Riccardo 2020-12-01

Controllo di processo

1 Scenario: riscaldamento di una stanza

Si supponga di voler usare Arduino per regolare la temperatura in una stanza, che è riscaldata da una caldaia mediante un termosifone. Oltre ovviamente alla scheda Arduino, gli elementi hardware fondamentali del sistema da realizzare sarebbero:

- un modo per permettere ad Arduino di accendere e spegnere la caldaia (ad esempio un relè);
- un sensore di temperatura (come ad esempio un LM35);
- una qualche interfaccia utente per l'impostazione della temperatura desiderata, che potrebbe essere un semplice potenziometro, un display numerico con pulsanti di aumento e diminuzione, uno schermo touch, un modulo Wi-Fi per il comando da smartphone, ecc.



Il sensore di temperatura permette di implementare un **controllo di processo** a **retroazione** (**feedback**):[1] quando Arduino comanda l'accensione della caldaia perché la

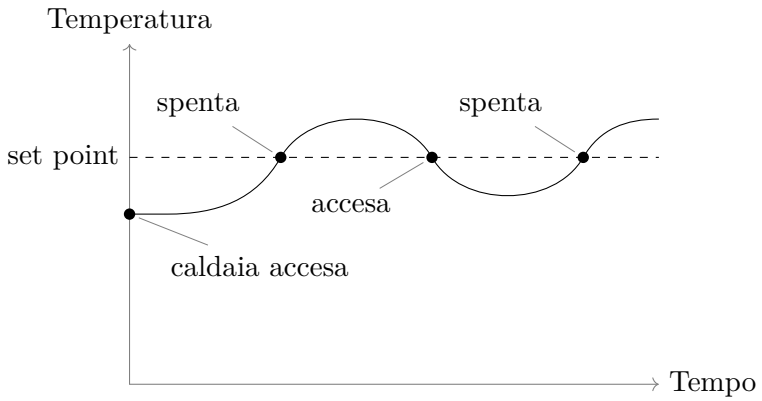
[1] L'opposto del controllo a retroazione è il controllo ad *anello aperto* (*open loop*), nel quale il controller non riceve informazioni sullo stato del processo.

temperatura era troppo bassa, il sensore rileva il risultante aumento di temperatura e lo rimanda ad Arduino, che può quindi regolare istante per istante l'azione di riscaldamento, in un modo tanto più "intelligente" quanto più è avanzata la logica implementata nel software.

La logica di controllo più semplice consiste nell'accendere la caldaia quando la temperatura della stanza è inferiore a quella impostata dall'utente, che in gergo prende il nome di **set point**, e poi spegnere la caldaia quando la temperatura della stanza raggiunge il set point. Questa tecnica, però, non tiene conto del **tempo di risposta** del sistema:

- quando la caldaia viene accesa, essa deve innanzitutto scaldare l'acqua nei tubi, che poi deve scaldare il termosifone, e solo a questo punto la temperatura nella stanza inizia ad aumentare;
- quando la caldaia viene spenta, il termosifone è ancora caldo, quindi la temperatura nella stanza continua a salire ancora per un po'.

Si ottiene allora un effetto del genere:



Uno schema di controllo più avanzato potrebbe tenere conto del tempo di risposta accendendo e spegnendo la caldaia "in anticipo", così da mantenere la temperatura richiesta con maggiore precisione.

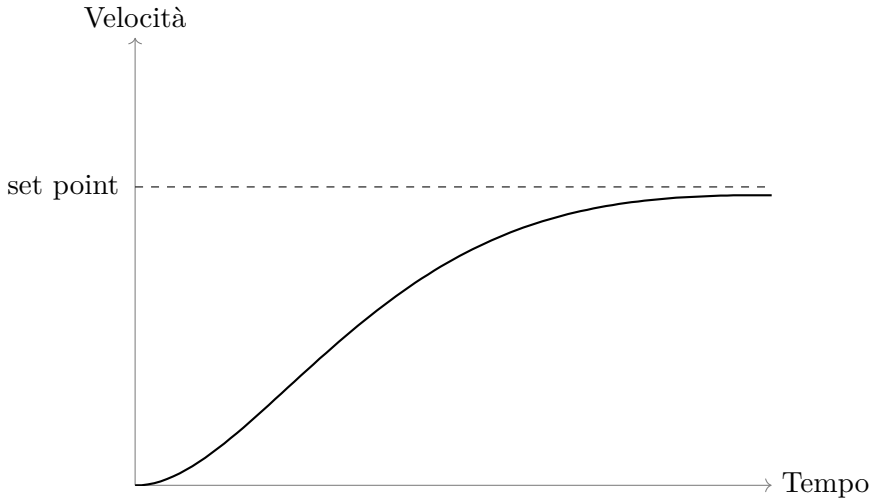
2 Scenario: velocità di un motore elettrico

Un'altra applicazione comune del controllo di processo è la regolazione della velocità di un motore elettrico, ad esempio quello di una lavatrice. Questo scenario è utile per illustrare uno schema di controllo più avanzato di quello visto prima, perché un motore elettrico può facilmente essere comandato con un segnale analogico, ad esempio in PWM (a differenza della maggior parte delle caldaie, che accettano solo un comando acceso/spento).

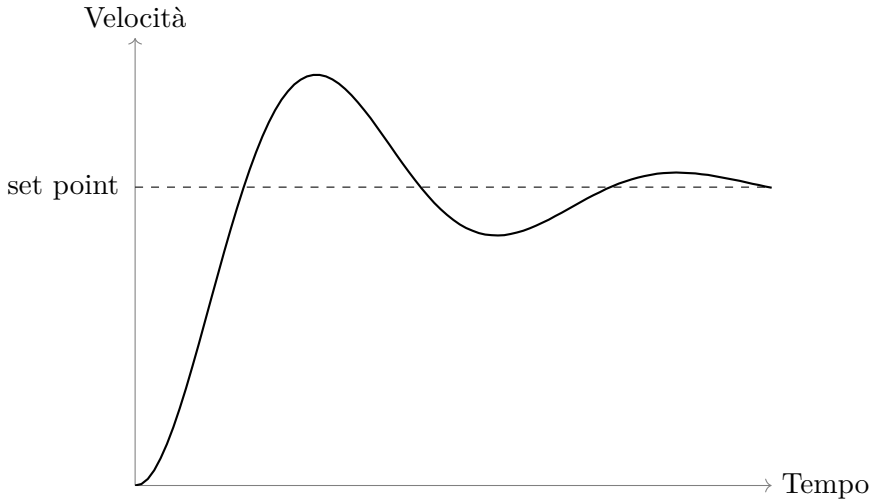
Per implementare il controllo con retroazione della velocità di un motore, è necessario che questo sia munito di un sensore di velocità. La differenza tra la velocità misurata da tale sensore e il set point (la velocità desiderata) prende il nome di **errore**.

Un primo metodo per controllare la velocità del motore è mandare a esso un segnale **proporzionale** all'errore, calcolato moltiplicando l'errore per una qualche costante K_p. Così, il motore accelera (o rallenta) tanto più rapidamente quanto più è "lontano" dalla velocità desiderata, ma ciò comporta alcuni effetti indesiderati:

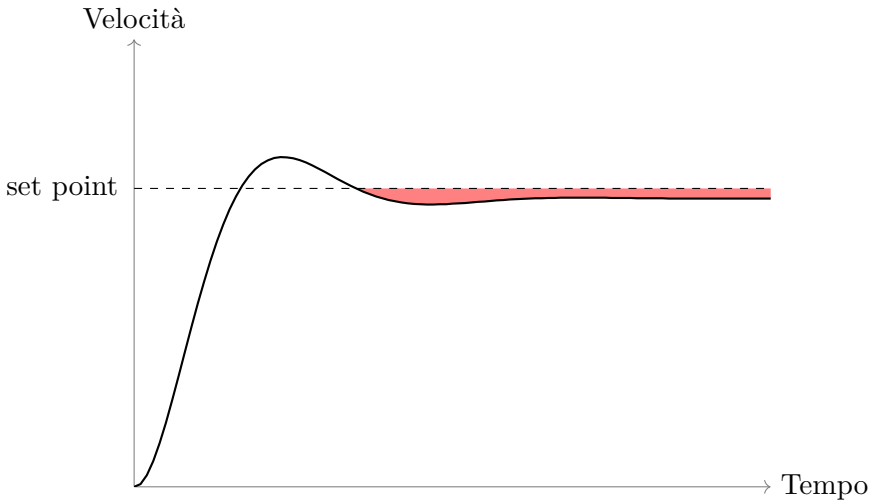
- se K_p è piccola, anche con un errore grande il motore accelera lentamente, quindi impiega molto tempo ad arrivare a regime:



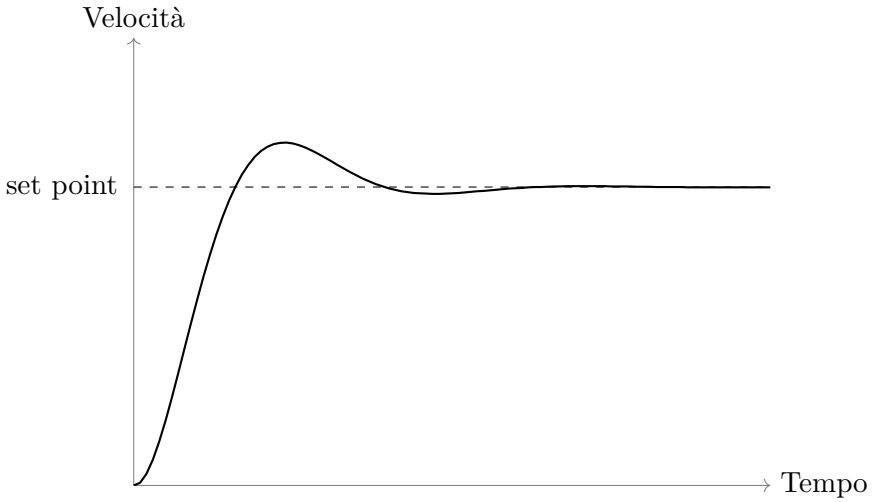
- se K_p è grande, il motore accelera molto rapidamente, quindi tende a superare il set point (*overshoot*) e provocare *oscillazioni*:



- quando l'errore è piccolo, anche il segnale di controllo proporzionale è piccolo, dunque non si riesce a eliminare completamente l'errore:



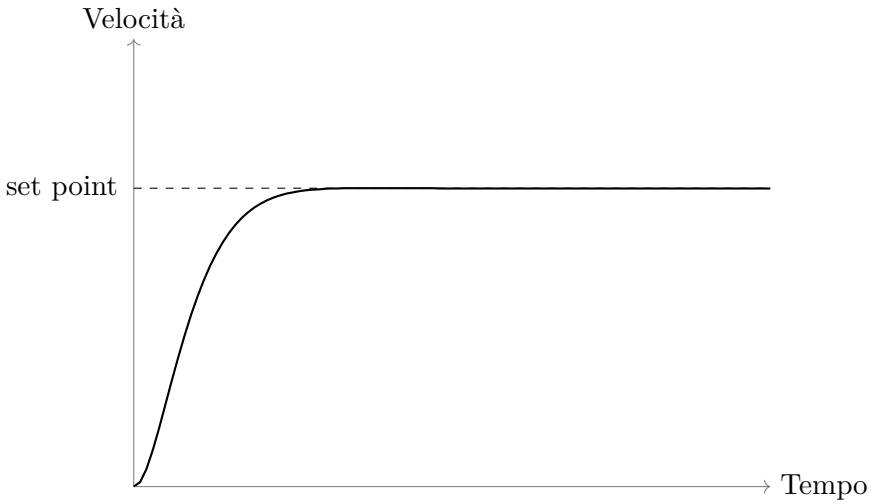
Per eliminare l'errore rimanente, si somma al segnale di controllo l'**integrale** dell'errore nel tempo, moltiplicato per una costante K_i. Esso aumenta gradualmente fino ad azzerare completamente l'errore:



In compenso, proprio perché può cambiare solo gradualmente, l'integrale rallenta il tempo di risposta del sistema, aumentando la tendenza all'overshoot.

L'ultimo passo è aggiungere all'output anche un elemento **derivativo**, la derivata dell'errore, moltiplicata per una costante K_d. Esso cerca di contrastare i cambiamenti rapidi della velocità del motore, facendolo avvicinare al set point in modo più graduale anche

se K_p è grande, riducendo ("smorzando") così l'overshoot senza aumentare di troppo il tempo necessario per arrivare a regime (come invece avverrebbe se si riducesse K_p):



Lo schema appena descritto prende il nome di **controllo PID** (proporzionale, integrale, derivativo), ed è molto usato perché è efficace e semplice da implementare. Uno dei pochi accorgimenti necessari, quando si implementa il controllo PID su un microcontrollore a bassa potenza, è che i calcoli avvengano più rapidamente della velocità a cui cambia lo stato del processo da controllare.