PID explained

PID control is the best controller in a system without a model of the process…

$$u(t) = \mathrm{MV}(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,d\tau + K_\mathrm{d} \frac{de(t)}{dt}$$

Motor speed control:

The most important and common application of PID control for smart car is the motor speed control. You always need to **precisely control** the speed of your car at the desired speed in the following situations:

- **Turning**

  Due to the different turning radius of the inner and outer wheels, they are rotating in different speed while turning. Incorrect power given to the motors may cause the car to slip or turn in an undesired radius.

- **Moving on unsteady surface/slope**

  For moving in a constant speed, the power needed for the car to climb a slope has to be higher than that of moving on a flat surface.

- **Accelerating/decelerating**

  In some case, the speed has to be increased/decreased gradually to avoid any slipping.

However, how much power (pwm) you should give to your motor in order to achieve the **desired speed**?

In ideal case, if you know the inclination of the road, mass of the car, voltage level of the battery, efficiency of the motors, radius of the wheels, etc. at any instance, you can actually calculate how much power/pwm signal you should give.

But obviously, you won't be able to know them easily, and that's where the PID control comes in handy for you to control the system **without knowing those information**.

PID controller of motor speed:

All the information a PID controller needs is the current state of what you want to control, in this case, the **current speed** (*process variable*). Therefore, you need a **feedback system/sensor** providing you the current speed of the car. For the smart car, we use **encoder**, a sensor which measures the rotational speed. In PID control, **error** indicates the difference between the current state and the target state.

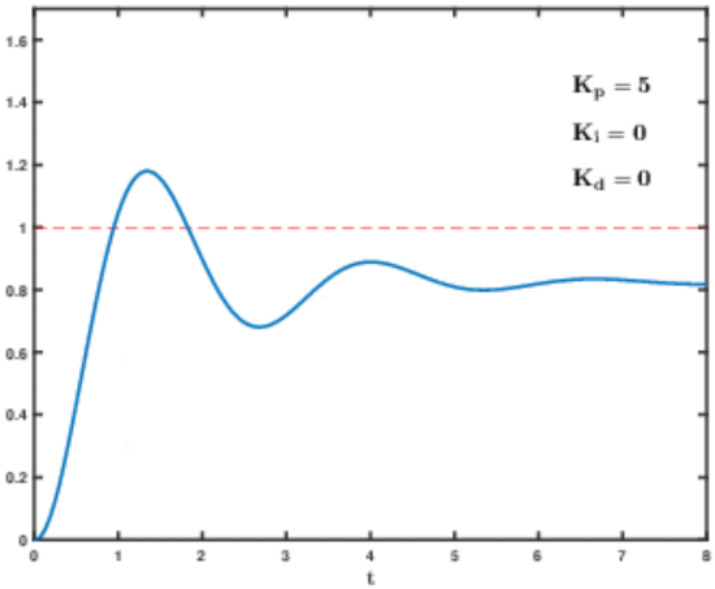
Error = desired speed - current speed

Remark: The speed given by the encoder may not be the speed of the car. Why? Besides, you may need **mean filter** or **median filter** for the encoder. What are the advantages and disadvantages of using them?

- **P for proportional**

P_{out} = K_p * currentError

$$P_{\text{out}} = K_{\text{p}} e(t)$$



The proportional term here means that the higher the error, the higher the output. When the desire speed is achieved, **P_{out} = K_p * Error** will be **zero**. Therefore, such proportional term alone cannot **sustain** the desired speed. It finally will reach an equilibrium that **K_p * Error** will output the power keeping the same current speed and thus the same error (*steady-state error*).

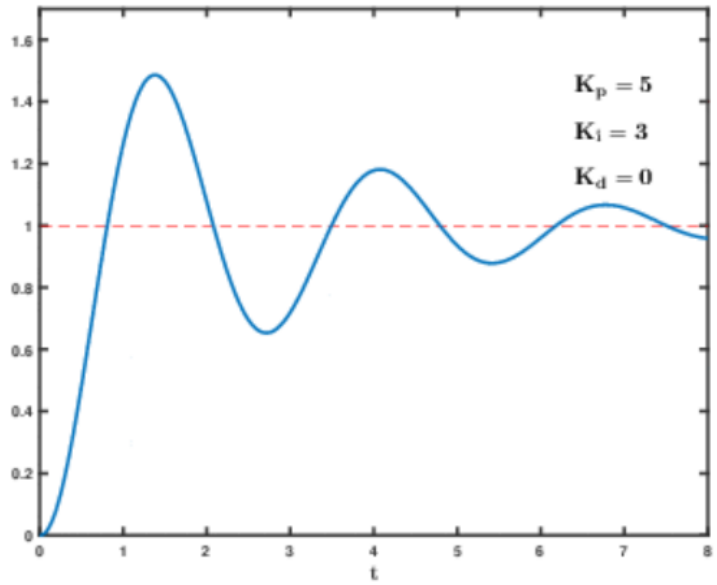
Remark: There exists a P controller (not PID controller) which have similar implementation with this but an extra constant term is added in order to maintain the output when the error is zero. But it is not applicable in the speed control for smart car. Do you know why?

- **I for integral**

I_{out} += K_i * currentError **or** **I_{out} = K_i * (sumError + currentError)**

$$I_{\text{out}} = K_{\text{i}} \int_0^t e(\tau)\, d\tau$$

The integral term is like an **incremental** version of the proportional term where the error will **sum up** over time to generate the output. Therefore, the integral term can cause the current speed to reach the desired speed and **sustain** the output when the **error** and the **proportional term** are **zero**. In other words, the integral term eliminates the steady-state error. Although the integral term can let us achieve the desired speed, problems like **overshoot** and **oscillation** still occur.



Remark: The integral term alone can already let the **process variable** to reach the **setpoint**, but very often, the proportional term will be used together with the integral term to be a PI controller (not PID controller). Then, what is the purpose of using the proportional term while the target can be reached with only the integral term? Also, what will happen if K_i is too large?

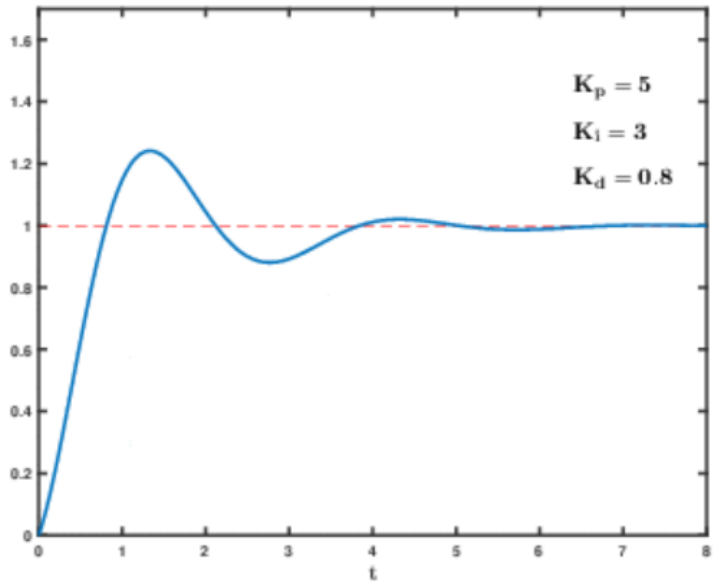
- **D for derivative**

D_{out} = K_d * (currentError - previousError)/(timeInterval)

$$D_{\text{out}} = K_{\text{d}} \frac{de(t)}{dt}$$

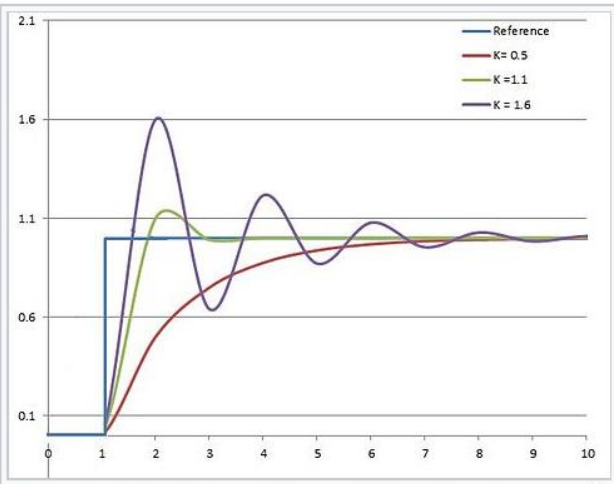
Although we can reach the setpoint with the PI controller, there is still some problems which are **overshoot** and **oscillations**. Therefore, the derivative term is needed to handle this issue.

The derivative term here behaves similar with the air resistant which is proportional to the velocity. That means the higher the **rate of change** of the error, the larger the effect of the derivative term. Apart from air resistant, you may also treat it as the **damping effect** of the mass-spring-damper model.

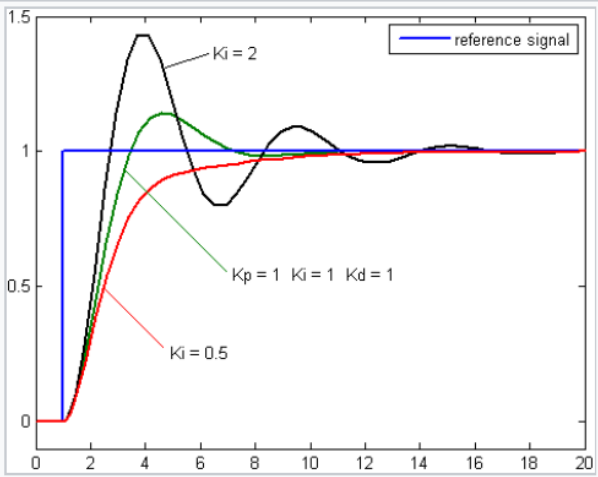


Remark: The sensor reading may be noisy or fluctuating. What is the effect of this to the derivative term and how to solve it? Besides, the derivative term is to reduce oscillation, but it will also cause oscillation. Do you know why? Also, the derivative term behaves oppositely to the proportional term and integral term. What do you think about the sign of K_d comparing to K_p and K_i?

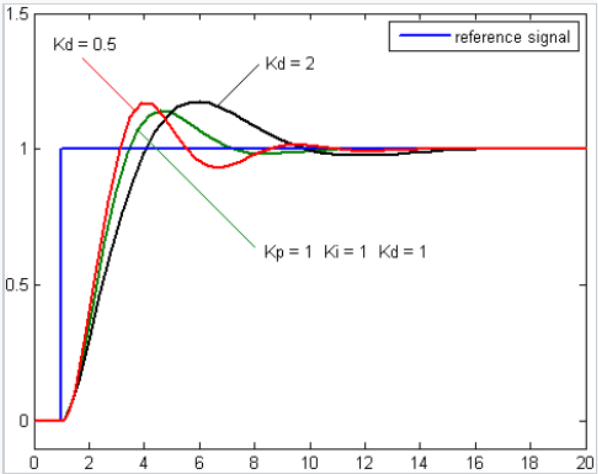
Effect of K_p, K_i and K_d:



Response of PV to step change of SP vs time, for three values of K_p (K_i and K_d held constant)



Response of PV to step change of SP vs time, for three values of K_i (K_p and K_d held constant)



Response of PV to step change of SP vs time, for three values of K_d (K_p and K_i held constant)

Tuning PID:

After knowing the functions and properties of each terms, what you have to do is to tune the three constants K_p, K_i and K_d. Very often, you have to **decrease K_p, K_i and increase K_d to reduce overshoot and oscillation**. Tuning a new set of PID, you may follow the procedure from Wikipedia:

"One tuning method is to first set K_i and K_d values to zero. Increase the K_p until the output of the loop oscillates, then the K_p should be set to approximately half of that value. Then increase K_i until any offset is corrected in sufficient time for the process. However, too much K_i will cause instability. Finally, increase K_d."

Noted that you need a **real time interface** to plot the values of the process variable so that you can observe the effect of changing the three terms while tuning.

Pseudocode:

```
previous_error = 0
integral = 0
loop:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto loop
```

Miscellaneous:

Integral windup:
https://en.wikipedia.org/wiki/Integral_windup
Setpoint step change:
https://en.wikipedia.org/wiki/PID_controller#Setpoint_step_change
Median filter:
https://en.wikipedia.org/wiki/Median_filter
Servo PID (the integral term is unnecessary):
https://www.youtube.com/watch?v=4Y7zG48uHRo