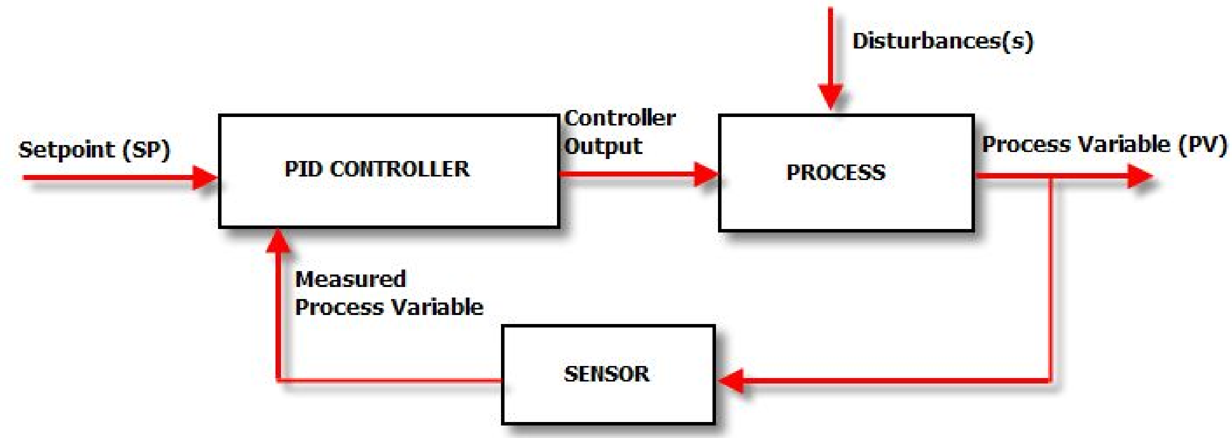
Session 7: PID Controller

Objectives

- Learn the basics of PID control

What is a Controller?

(Ref: https://www.csimn.com/CSI_pages/PIDforDummies.html)



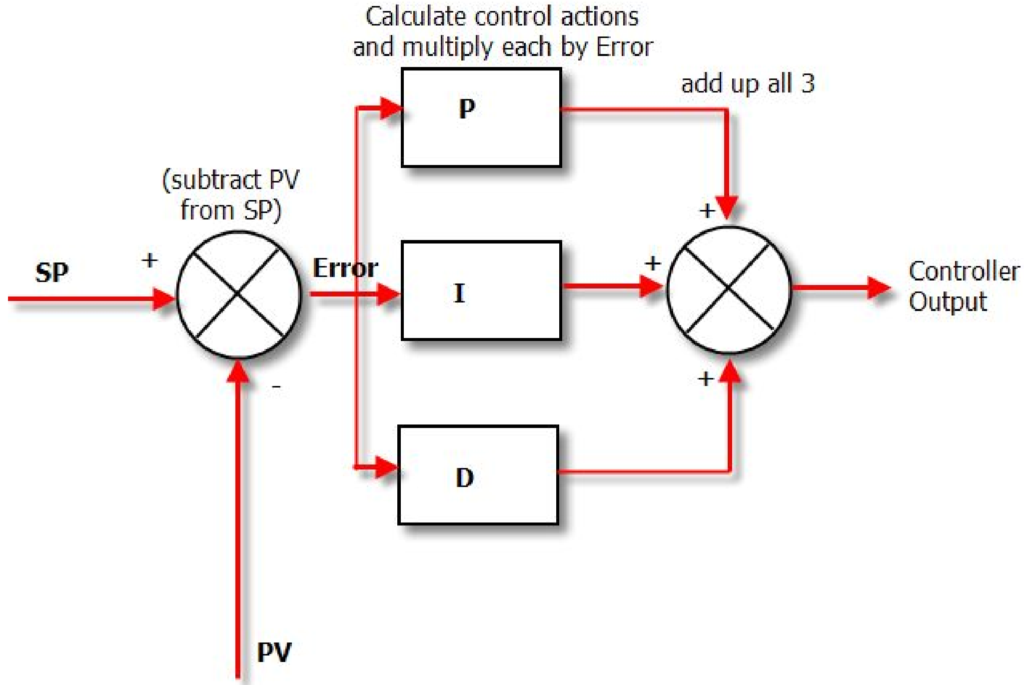
The Setpoint (SP) is the value that we want the process to be. For our robocar, we'll be using the PID controller for steering adjustment to try to keep the robocar on the line. Hence the Setpoint is set to zero.

The PID controller looks at the setpoint and compares it with the actual value of the Process Variable (PV).

If the SP and the PV are the same – then the controller is a very happy little box. It doesn't have to do anything, it will set its output to zero.

However, if there is a disparity between the SP and the PV we have an error (CTE) and corrective action is needed. In our robocar application, the corrective action is a steering command that steers the robocar back to the middle.

The PID Controller

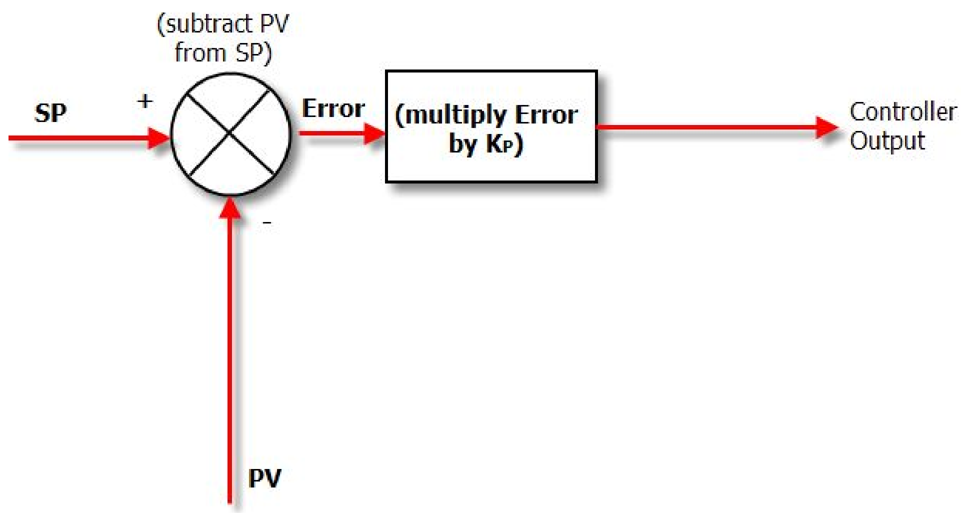


The PV is subtracted from the SP to create the Error. The error is simply multiplied by one, two or all of the calculated P, I and D actions (depending which ones are turned on). Then the resulting “error x control actions” are added together and sent to the controller output.

The PID controller consists of a linear combination of three controllers:

- Proportional
- Integral
- Derivative

For each controller, we simply scale the error by a constant. For example, the block diagram for the Proportional control looks like this:

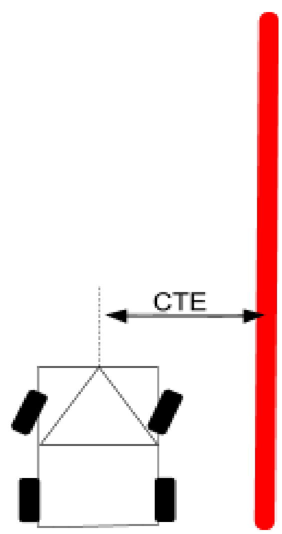


We need to tune the three controller constants Kp, Ki and Kd before we can use the PID controller for the robocar.

Proportional Control

The proportional control applies a corrective action that is directly proportional to the error.

$$\alpha = K_p \times CTE$$



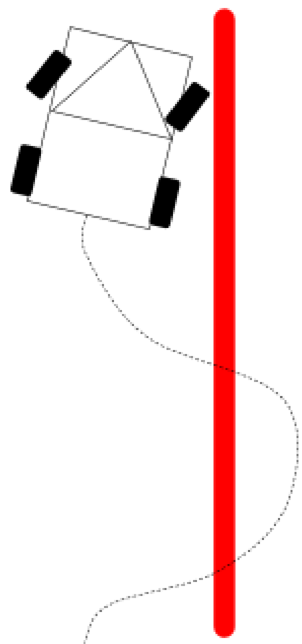
TO-DO:

- Open the `pid.py` file in the `camcontrol` node of Ex-7-1-PID_Control.

- Edit function `step()` so that the proportional control is active.
- Open `camcontrol_node.py` and edit the value of the proportional gain `Kp`. Experiment with different values and make observations.

Proportional + Derivative Control

With only the proportional control active, the robocar might end up oscillating around the line as it overshoots the line each time. This is especially if the proportional control gain is set too high.



To minimize oscillations, we add a derivative control that compensates for high rate of change in the CTE, i.e. the derivative of CTE.

$$\alpha = K_p \cdot CTE + K_d \cdot \frac{d}{dt} CTE$$

Since our time steps (duration of each update) are the same, we can simplify the derivative term by a difference and adjust the gain accordingly.

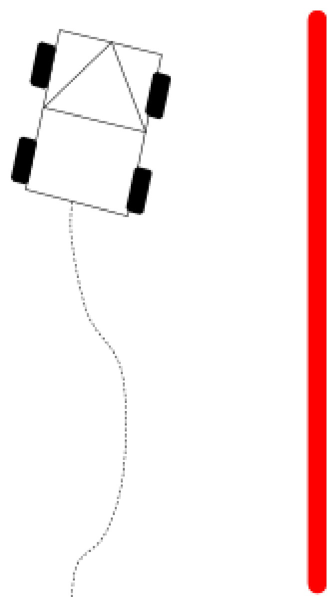
$$\frac{d}{dt} CTE_i \approx CTE_i - CTE_{i-1}$$

TO-DO:

- Open the `pid.py` file in the `camcontrol` node of Ex-7-1-PID_Control.
- Edit function `step()` to add derivative control.
- Open `camcontrol_node.py` and edit the value of the derivative gain `Kd`. Experiment with different values of `Kp` and `Kd` and make observations.

Proportional + Integral + Derivative Control

What if we still see a persistent bias with the PD controller, e.g. if the robocar is not set up / calibrated properly?



That's where the integral component comes in. An integral term increases action in relation not only to the error but also the time for which it has persisted, i.e. we want to keep track of all the previous CTEs.

$$\alpha = K_p \cdot CTE + K_i \cdot \int_0^t CTE \cdot dt' + K_d \cdot \frac{d}{dt} CTE$$

Just like the derivative term, we'll simplify our implementation of the integral term with a sum of the CTEs.

$$\int_0^t CTE \cdot dt' \approx \sum_{i=0}^{t} CTE_i$$

TO-DO:

- Open the `pid.py` file in the `camcontrol` node of Ex-7-1-PID_Control.
- Edit function `step()` to add integral control.
- Open `camcontrol_node.py` and edit the value of the derivative gain `Ki`. Experiment with different values of `Kp, Ki` and `Kd` and make observations.
- Try driving around the test track while tuning the PID coefficients, Once you are statisfied with the performance, record the values of the PID parameters for later sessions.