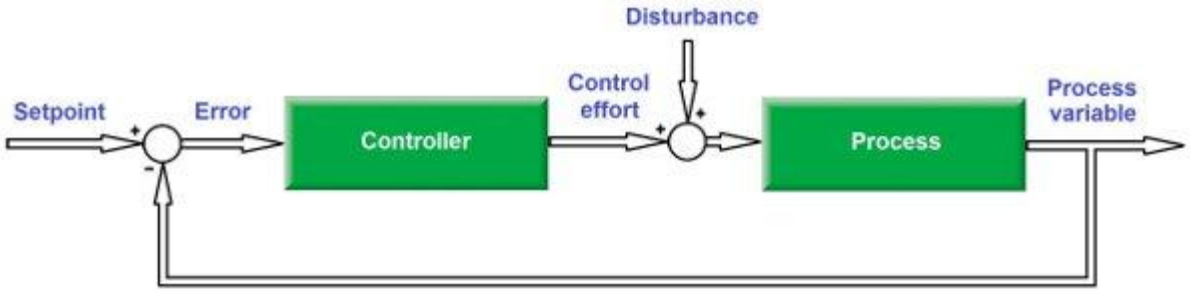
Disturbance-Rejection vs. Setpoint-Tracking Controllers

Designing a feedback control loop starts with understanding its objective as well as the process's behavior.

Vance VanDoren, Ph.D., P.E.
09/26/2011

The principal objective of a feedback controller is typically either *disturbance rejection* or *setpoint tracking*. A controller designed to reject disturbances will take action to force the process variable back toward the desired setpoint whenever a disturbance or *load*on the process causes a deviation.

A car's cruise controller, for example, will throttle up the engine whenever it detects a drop in the car's speed during an uphill climb. It will continue working to *reject* or overcome the extra load on the car until the car is once again moving as fast as the driver originally specified. Disturbance-rejection controllers are best suited for applications where the setpoint is constant and the process variable is required to stay close to it.



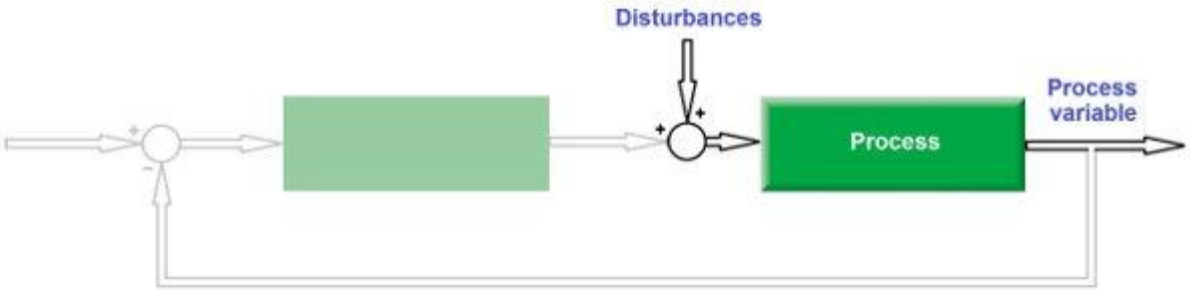
In contrast, a setpoint-tracking controller is appropriate when the setpoint is expected to change frequently and the controller is required to raise or lower the process variable accordingly. A luxury car equipped with an automatic temperature controller will *track*a changing setpoint by adjusting the heater's output whenever a new driver calls for a new interior temperature.

Disturbance-rejection and setpoint-tracking controllers can each do the job of the other (a cruise controller can increase the car's speed when the driver wants to go faster, and the car's temperature controller can cut back the heating when the sun comes out), but *optimal* performance generally requires that a controller be designed or tuned for one role or the other. To see why, consider the feedback loop shown in the Control Loop diagram and the effects

of an abrupt disturbance to the process or an abrupt change in the process variable's setpoint.

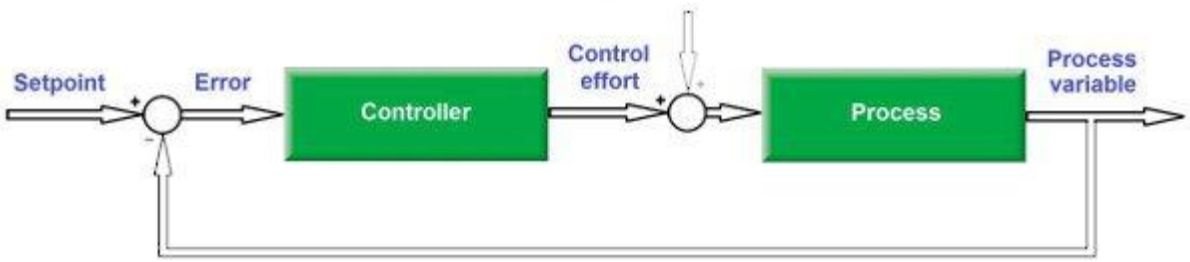
Open-loop operations

First, suppose that the feedback path is disabled so that the controller is operating in open-loop mode. After a disturbance, the process variable will begin to change according to the magnitude of the load and the physical characteristics of the process. In the cruise control example, the sudden resistance added by the hill will start to decelerate the car according to the hill's steepness and the car's inertia.



Note that an open-loop controller doesn't actually play any role in determining how the process reacts to a disturbance, so the controller's tuning is irrelevant when feedback is disabled. In contrast, a setpoint change will pass through *both* the controller *and*the process, even without any feedback. See the Open-Loop Operations diagram.

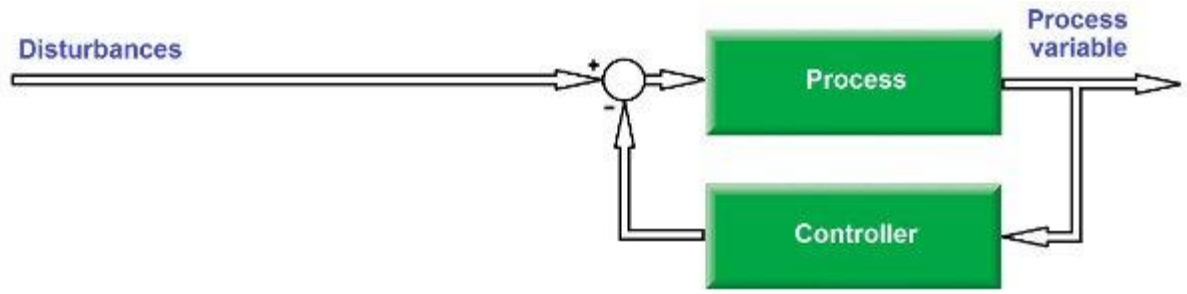
As a result, the mathematical inertia of the controller combines with the physical inertia of the process to make the process's response to a setpoint change slower than its response to an abrupt disturbance. This is especially true when the controller is equipped with integral action. The I component of a PID controller tends to *filter* or average-out the effects of a setpoint change by introducing a time lag that limits the rate at which the resulting control effort can change.



In the car temperature control example, this phenomenon is evident when the controller starts turning up the heat upon receiving the driver's request for a warmer interior. The car's heater will in turn begin to raise the car's temperature at a rate that depends on how aggressively the controller is tuned and how quickly the interior temperature reacts to the heater's efforts. A direct disturbance such as a burst of sunshine would typically raise the car's temperature at a much faster rate because the effects of the disturbance would not depend on the controller ramping up first.

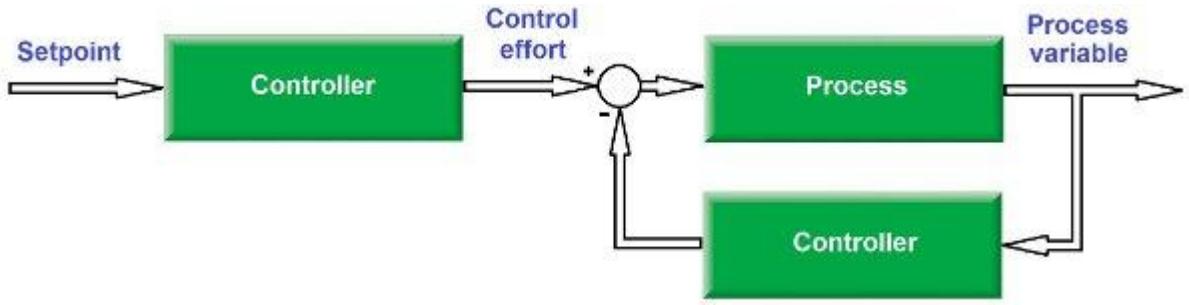
Closed-loop operations

Of course an open-loop controller can't really reject disturbances nor track setpoint changes without feedback, so it makes sense to ask, "What happens to that extra setpoint response time when the feedback is enabled?" Usually, nothing. Unless the controller happens to be equipped with *setpoint filtering*, the setpoint response will remain slower than the disturbance response by exactly the same amount as in the open-loop case. See the Closed-Loop Operations diagram.

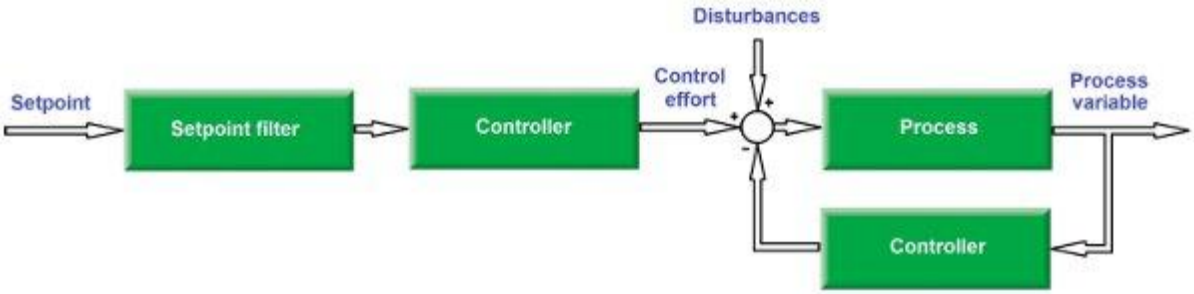


But since that difference in response times is attributable entirely to the time lag of the controller, one might wonder if it would still be possible to design a setpoint-tracking controller that is just as fast as its disturbance-rejection counterpart by tuning it to respond instantaneously to a setpoint change.

That won't work either. Eliminating the controller's time lag would require disabling its integral action, and that would prevent the process variable from ever reaching the setpoint. For more on this *steady-state offset* phenomenon, see "The Three Faces of PID,"*Control Engineering*, March 2007.



On the other hand, the controller's mathematical inertia can be minimized without completely defeating its ability to eliminate errors between the process variable and the setpoint. A fast setpoint-tracking controller would require particularly aggressive tuning, but that shouldn't be a problem so long as the controller never needs to reject a disturbance. But if an unexpected load ever does disturb the process abruptly, a setpoint-tracking controller will tend to overreact and cause the process variable to oscillate unnecessarily.



Conversely, a controller tuned to reject abrupt disturbances will typically be relatively slow about implementing a setpoint change. Fortunately, a typical feedback control loop in an industrial application will operate for extended periods at a constant setpoint, so the only time that a disturbance-rejection controller normally experiences a delay because of a setpoint change is at start-up.

Caveats

Unfortunately, that's not the end of the disturbance-rejection vs. setpoint-tracking story. To this point we have assumed that the process is subject to abrupt disturbances such as when a car with cruise control suddenly encounters a steep hill. Many if not most feedback control applications involve much less dramatic disturbances—rolling hills rather than steep inclines, for example.

When the physical properties of the process limit the rate at which disturbances can affect the process variable, the disturbance response will sometimes be *slower* than the setpoint response, not faster. In such cases, more aggressive tuning would be appropriate for a disturbance-rejection controller than for its setpoint-tracking counterpart. The key is determining which scenario applies to the process at hand and which objective the controller is required to achieve.

Source:

http://www.controleng.com/single-article/disturbance-rejection-vs-setpoint-tracking-controllers/fe8b0cf50a2f0000c3b647c7e36326af.html