0.1 System types

Specifically, the position error constant k_p and velocity error constant k_v depend upon **the number of poles at the origin** in the open loop system. These correspond to roots in the denominator at $s = 0$, and represent a pure integration. Let's look more generally at the transfer function of our system and introduce the concept of system **type**. By factoring out any s terms from the denominator, we can write the transfer function in the following form:

$$G(s)H(s) = \frac{(s - z_1)(s - z_2)(s - z_3)...}{s^p(s - \sigma_1)(s - \sigma_2)(s - \alpha_k + j\omega_k)(s - \alpha_k - j\omega_k)...} \tag{1}$$

So, if there are p *poles at the origin*, the system is said to be a 'type p' system.

System type IS NOT THE ORDER OF THE SYSTEM!

For example, the servo motor can be expressed as:

$$\frac{\Theta_0(s)}{\Theta_i(s)} = \frac{k}{s(Is + f)} \tag{2}$$

which is a type 1 system. The electromagnet from lecture 2 can be expressed as:

$$\frac{I(s)}{V(s)} = \frac{1}{Ls + R} \tag{3}$$

which is a type 0 system. The mas spring damper system can be expressed as:

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + cs + k} \tag{4}$$

which is a type 0 system.

The system type quickly tells us the form of the SSE for our inputs without manipulation of block diagrams or converting to time domain. This gives an indication of the design of the controller.

0.1.1 System type - calculating error

Essentially, for a zero error we want k_p and k_v to be infinite, or at least be a constant for a finite error, depending on our requirements.

$$SSE = \frac{a}{1 + k_p}, \; SSVL = \frac{a}{k_v} \tag{5}$$

$$k_p = \lim_{s \to 0} G(s)H(s), \; k_v = \lim_{s \to 0} sG(s)H(s) \tag{6}$$

0.1.2 System type for unit feedback control systems - calculating error

The **position error constant** k_p for a *type p system* is given by:

- $p > 0, \; k_p = \lim_{s \to 0} G(s) = \infty$ - no steady-state position error
- $p = 0, \; k_p$ is finite - finite position error

The **velocity error constant** for a *type* p system is given by:

- $p > 1,\ k_v = \lim_{s\to 0} sG(s) = \infty$ - no velocity error
- $p = 1,\ k_v$ is finite - steady state velocity lag
- $p = 0,\ k_v = 0$ infinite lag (completely fails to track)

No. Integrators in denominator = system TYPE	Input type		
	Step $r(t) = a$ $R(s) = \frac{a}{s}$	Ramp $r(t) = at$ $R(s) = \frac{a}{s^2}$	Acceleration $r(t) = \frac{at^2}{2}$ $R(s) = \frac{a}{s^3}$
0	$e_{ss} = \frac{a}{1+k_p}$	$e_{ss} = \infty$	$e_{ss} = \infty$
1	$e_{ss} = 0$	$e_{ss} = \frac{a}{k_v}$	$e_{ss} = \infty$
2	$e_{ss} = 0$	$e_{ss} = 0$	$e_{ss} = \frac{a}{k_a}$

Table 1:

where ∞ means the system never settles. The more poles there are at the origin of the open-loop system, the better steady-state tracking performance. However, pure integrations have a highly destabilising effect on the control system!

0.1.3 Type 0 servo - spring return

Previously we have looked at servos with inertia and damping only, without and compliance/springs. This gives a type-1 transfer function. However, when a spring is added, the system becomes type-0.

$$\frac{\Theta_0(s)}{\Theta_i(s)} = \frac{k}{s(Is + f)} \tag{7}$$

The question arises, why would you add a spring? Some systems are not bidirectional e.g. a single acting piston in a hydraulic system. A spring is needed to return the piston to the correct position in the cycle. A spring is often added to servo actuators to return to a given position when system is off. HVAC systems use this for fire safety. The transfer function of this servo now becomes:

$$\sum T = I\frac{\mathrm{d}^2\theta_0}{\mathrm{d}t^2} \tag{8}$$

$$I\frac{\mathrm{d}^2\theta_0}{\mathrm{d}t^2} = T_s + T_f + T_m \tag{9}$$

where T_s and T_f oppose the motion of the motor. In time domain:

$$I\frac{\mathrm{d}^2\theta_0}{\mathrm{d}t^2} = -k_s\theta_0 - f\frac{\mathrm{d}\theta_0}{\mathrm{d}t} + k_m\theta_i \tag{10}$$

In Laplace:

$$Is^2\Theta_0(s) = -k\Theta_0(s) - fs\Theta_0(s) + k_m\Theta_i(s) \tag{11}$$

$$Is^2\Theta_0(s) + k_s\Theta_0(s) + fs\Theta_0(s) = k_m\Theta_i(s) \tag{12}$$

$$\Theta_0(s)\left(Is^2 + k_s + fs\right) = k_m\Theta_i(s) \tag{13}$$

Transfer function:

$$\frac{\Theta_0(s)}{\Theta_i(s)} = \frac{k_m}{Is^2 + fs + k_s} = G(s) \tag{14}$$

So the closed loop transfer function is now:

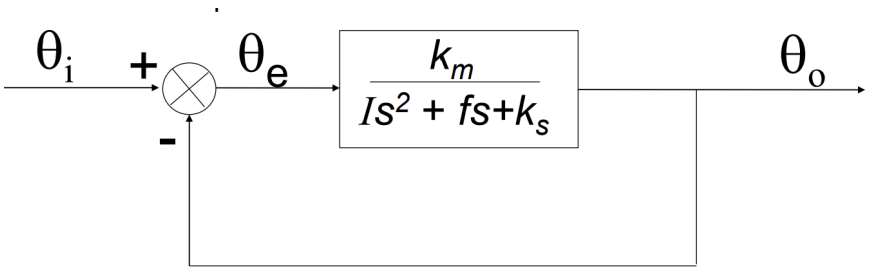


Figure 1:

$$F(s) = \frac{G(s)}{1 + G(s)} = \frac{k_m}{Is^2 + fs + k_s + k_m} \tag{15}$$

$$k_p = \lim_{s \to 0} G(s)H(s) = \frac{k_m}{Is^2 + fs + k_s} = \frac{k_m}{k_s} \tag{16}$$

$$k_v = \lim_{s \to 0} sG(s)H(s) = \frac{sk_m}{Is^2 + fs + k_s} = \frac{0}{k_s} \tag{17}$$

$$SSE = \frac{a}{1 + k_p} = \frac{ak_s}{k_s + k_m},\ SSVL = \frac{a}{k_v} = \infty \tag{18}$$

So now the servo will now only travel a fraction of the required distance for a step input, and will completely fail to track a ramp input, never settling to a steady state. We need an improved controller for these systems!

0.2 Proportional control

0.2.1 Proportional control - damping ratio trade off

We have seen the effect of damping ratio for our simple servo control system so far. We have two conflicting requirements:

- For transient response we want something around 0.6-0.8 to reduce overshoot and oscillations. $\zeta = \frac{f}{2\sqrt{kI}}$
- For steady state response, we want the damping ratio as low as possible at the expense of oscillations. $SSVL = \frac{af}{k} = \frac{2\zeta a}{\omega_n}$

For our type 1 servo, this trade off occurs if we want good ramp tracking, something we could possibly neglect in certain applications. However, the type 0 servo SSVL renders it close to unusable in most control applications (with the current controller). Commonly the challenge is to increase ζ, but with our current system, we do not have many options for adjustment.

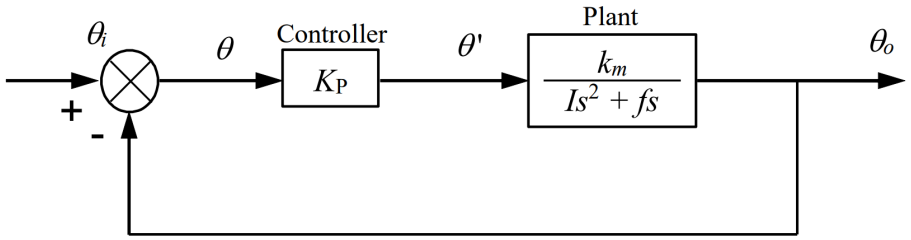


Figure 2:

We can either increase our friction f but this increases wear on the components, reducing the life of the system. So that leaves the controller gain k_p to adjust, we can reduce it at the expense of steady state performance. So we need a better controller!

0.3 Velocity feedback

0.3.1 Doing better - compensation methods

What the controller we have seen so far is known as a proportional controller, the control signal is directly proportional to the error signal, scaled by k_p. Compensation methods have been designed to improve **both** transient and steady state performance. One of the earliest and simplest compensation methods is to consider not just the position but the velocity too.

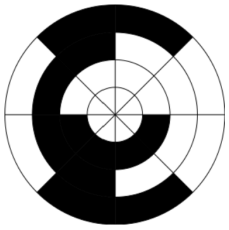


Figure 3:

If we take the position from the shaft encoder, and also record the change over time, we can measure the speed. The block diagram of this is just s - as this is the differential operator, we also include another gain here k_{vel} so we can choose the relative contributions of the two feedbacks. So we have the following block diagram.

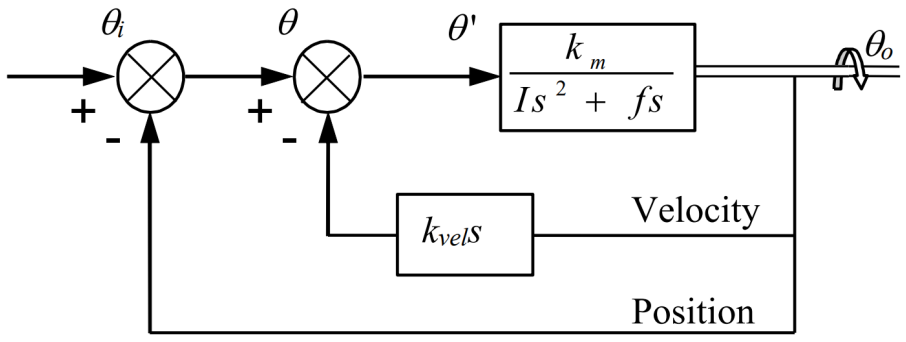


Figure 4:

Inner comparator:

$$\theta' = \theta - k_{vel} s\theta_0 \tag{19}$$

Outer comparator:

$$\theta = \theta_i - \theta_0 \tag{20}$$

If we calculate the new closed loop transfer function - work outwards!

$$\frac{\theta_0}{\theta} = \frac{G}{1+GH} = F(s) \tag{21}$$

$$F(s) = \frac{\frac{k_m}{Is^2+fs}}{1+\frac{k_m}{Is^2+fs}k_{vel}s} = \frac{k_m}{Is^2+fs+k_m k_{vel}s} \tag{22}$$

$$= \frac{k_m}{Is^2+(f+k_m k_{vel})\,s} \tag{23}$$

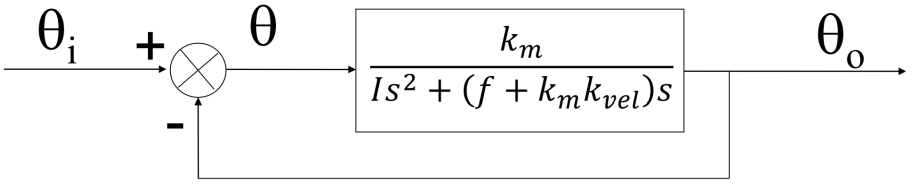


Figure 5:

Inner loop transfer function:

$$F(s) = \frac{k_m}{Is^2+(f+k_m k_{vel})\,s} \tag{24}$$

Total closed loop transfer function:

$$F_T(s) = \frac{F}{1+F} \tag{25}$$

$$F_T(s) = \frac{k_m}{Is^2+(f+k_m k_{vel})s+k_m} \tag{26}$$

$$F_T(s) = \frac{\frac{k_m}{I}}{s^2+\frac{(f+k_m k_{vel})}{I}s+\frac{k_m}{I}} \tag{27}$$

$$\omega_n = \sqrt{\frac{k_m}{I}},\ \zeta = \frac{f+k_m k_{vel}}{2\sqrt{kI}} \tag{28}$$

So by adding the extra loop, we are able to keep the natural frequency the same, *but increase the damping ratio*. This means we can greatly improve the transient response compared to proportional control, where we can only scale the motor gain k_m through adjusting the parameter k_p. This is evident when considering the position of the poles of the system on the s-plane. Adjusting the motor gain using proportional control (k_p) can only move the poles closer to the real axis. But using velocity feedback (k_{vel}), it is possible to move the poles away from the imaginary axis, thus reducing the decay time.

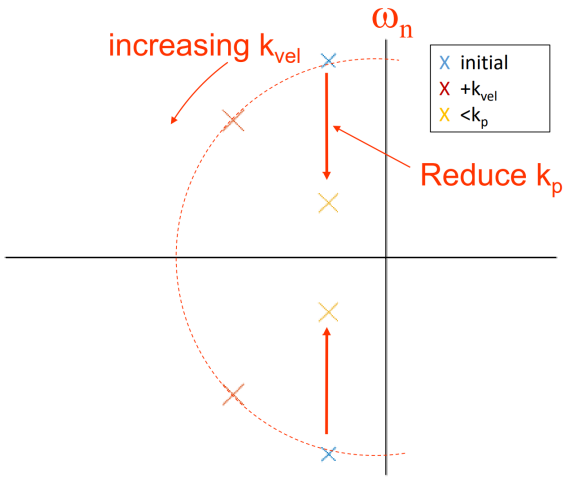


Figure 6:

In this example, the initial zeta was $\zeta = 0.3$, in order to achieve the practical estimate of $\zeta = 0.7$ with proportional control, we have to increase both the rise time and settling time. However, with velocity feedback we can achieve a vastly improved transient response.

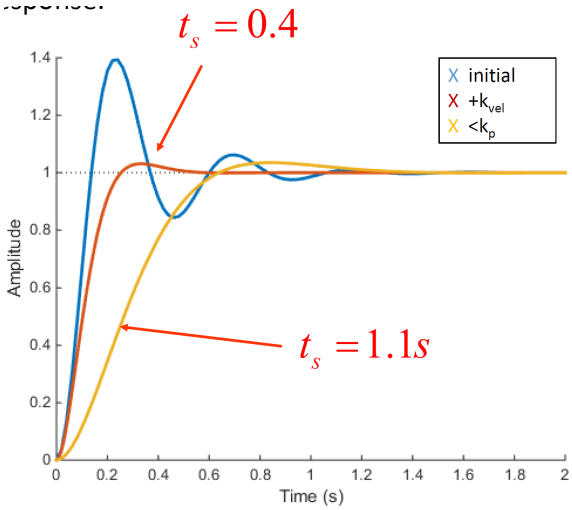


Figure 7:

0.3.2 Velocity feedback - disadvantages

This improved transient response comes at the expense of the steady state error, the expression for the SSVL becomes:

$$k_v = \lim_{s \to 0} sF(s) = \frac{sk_m}{Is^2 + (f + k_m k_{vel})\, s} = \frac{k_m}{f + k_m k_{vel}} \tag{29}$$

$$SSVL = a\frac{k_m}{f + k_m k_{vel}} \tag{30}$$

Compare this SSVL to the one we had previously $\left(\frac{af}{k}\right)$. Using this feedback system allows for the designer to trade off the transient and steady state response dependent

upon the requirements of the system. However, using the velocity in this manner is very sensitive to errors or noise from the sensor, which give large instantaneous changes in the error signal. To combat this, a low pass filter is normally employed in the feedback loop before the derivative term.

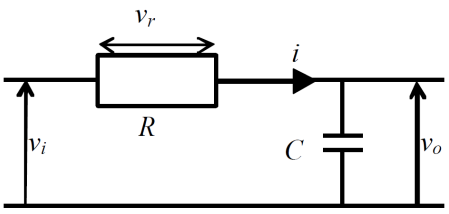


Figure 8:

$$= \frac{1}{RCs+1} \text{ see lecture 2} \tag{31}$$

This makes the response more complex and slightly reduces the improvements in transient response obtained.

0.3.3 Alternative controllers

The velocity feedback is rather specific to servomechanisms with a shaft encoder. It is not always possible to have a sensor or a measurement from which both the variable and the derivative can be acquired simultaneously.

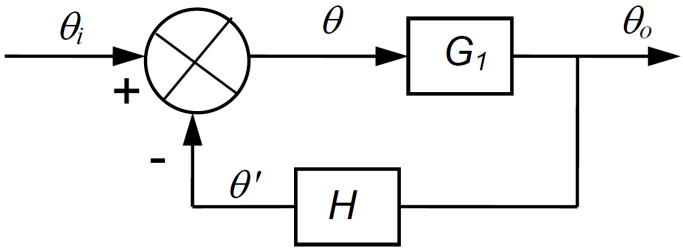


Figure 9:

Instead, we can consider what happens to the *error* across time - something necessarily present in all closed loop systems!

0.4 PID control

PID = Proportional, Integral and Derivative controllers: These controllers consist of three terms, all applied to the error signal. They are *incredibly* common in engineering, accounting for nearly 90% of all use cases.

0.4.1 Working example: PID control for servo-mechanism

These can be modelled as shown below, with a DC motor rotating an inertial load, with a friction or damping resistance.

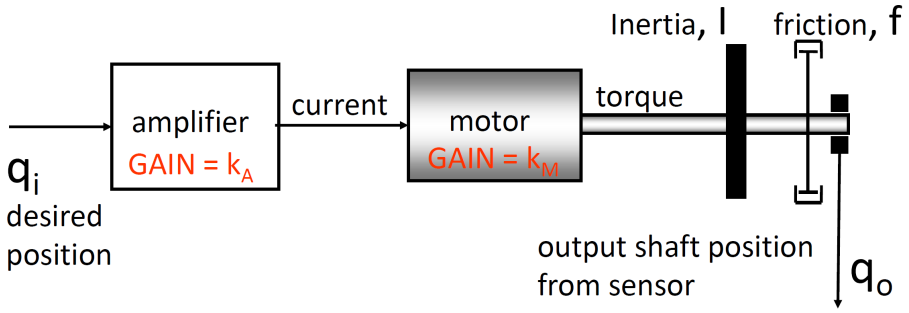


Figure 10:

0.4.2 Proportional error

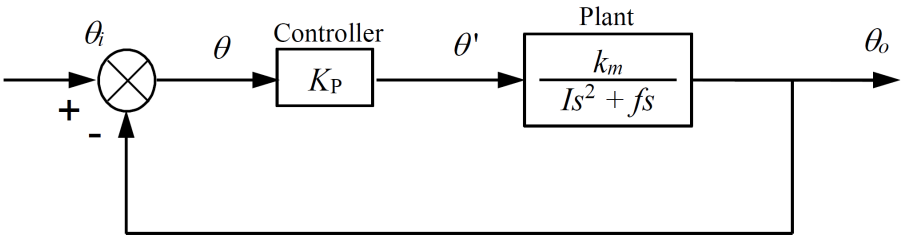


Figure 11:

$$F(s) = \frac{G(s)}{1+G(S)} = \frac{k_p k_m}{Is^2 + fs + k_p k_m} \tag{32}$$

We have already seen this in our previous analysis of second order systems, and from changing k_m from last last week. To summarise: low values of k_p give stable but slow responses, and high SSVL. High values reduce SSVL but response overshoots considerably. With proportional control the error *and thus the control signal* does not reach zero until the motor is at the desired position.

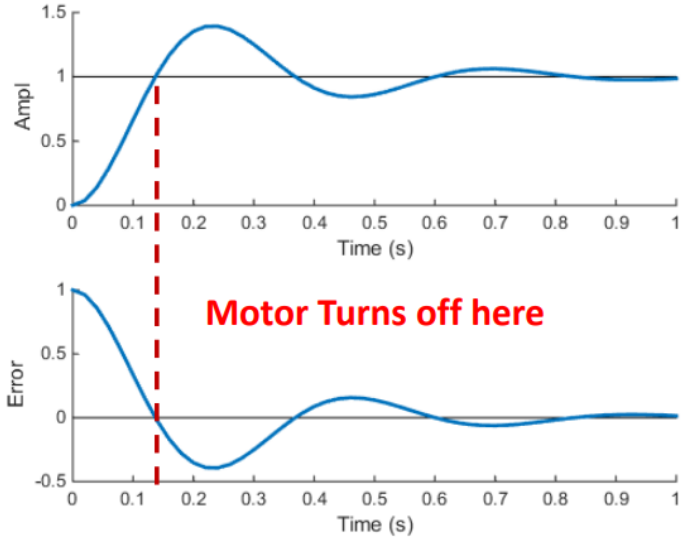


Figure 12:

However, due to the inertia for the system, the motor continues to move even without a control signal, resulting in an overshoot.

0.4.3 PID control

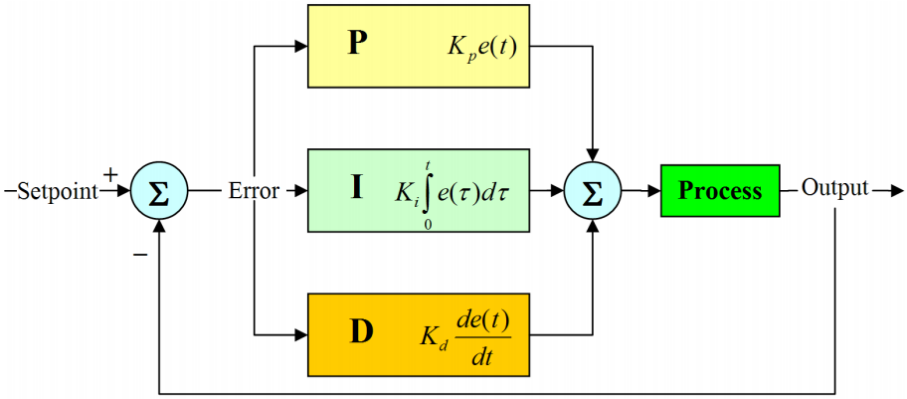


Figure 13:

0.4.4 Derivative error

We can replicate this by considering the **derivative** of the error.

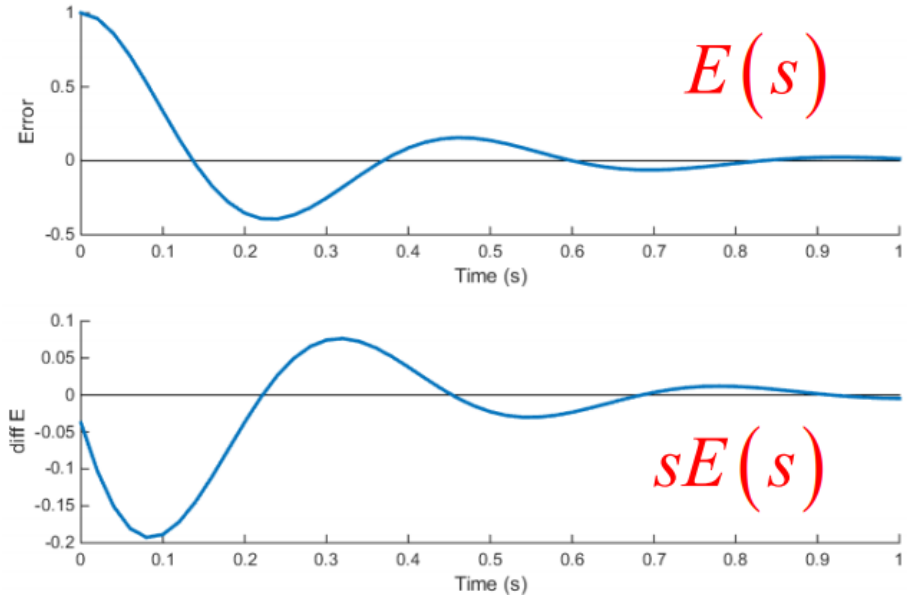
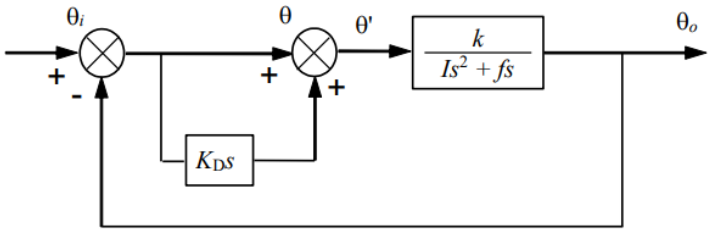


Figure 14:

This is negative as the servo approaches the target, so offers a way of restraining the servos forward motion. Lets consider a derivative controller with unity proportional gain.



$$\theta' = \theta + K_D s\theta \qquad \frac{\theta'}{\theta} = 1 + K_D s \qquad \theta = \theta_i - \theta_o$$

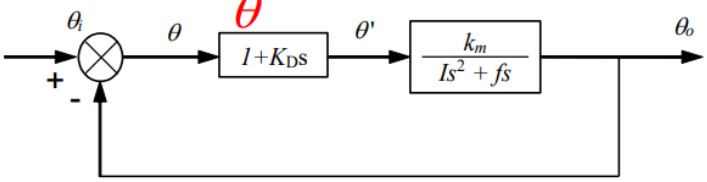


Figure 15:

The open loop transfer function now becomes type 1:

$$G' = \frac{(1 + K_D s)\, k_m}{Is^2 + fs} \tag{33}$$

Giving a *closed loop transfer function* of:

$$F(s) = \frac{G'(s)}{1 + G'(s)} = \frac{(1 + K_D s)\, k_m}{Is^2 + fs + (1 + K_D s)\, k_m} \tag{34}$$

$$F(s) = \frac{k_m K_d s + k_m}{Is^2 + (f + k_m K_D)\, s + k_m} \tag{35}$$

System is still second order but now there is a zero in the numerator. The effect of this is subtle compared to the change in the poles. More importantly, lets consider the SSVL of this system:

$$k_v = \lim_{s \to 0} sG'(s) = \frac{s\,(1 + K_D s)\, k_m}{s\,(Is + f)} = \frac{k_m}{f} \tag{36}$$

So, unlike velocity feedback, the SSVL is unchanged by derivative error. So we can improve transient response without compromising steady state error. The effect on the step response is similar to that of the minor loop velocity feedback.

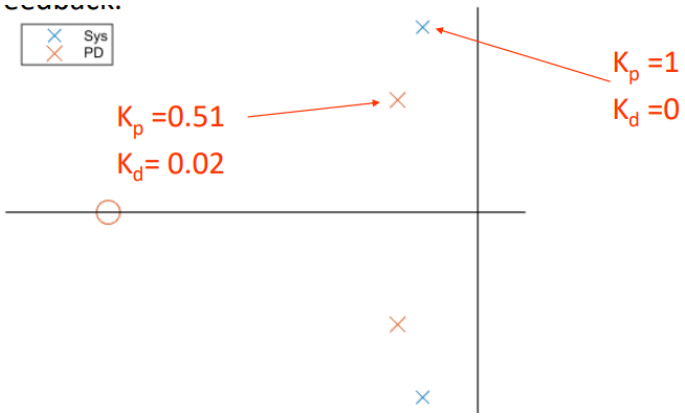


Figure 16:

The poles of the system become less oscillatory and decay quicker. This combined with a reduced proportional gain, gives an improved transient response. This is clear when looking at the step response, and the relative contributions of the two error terms.

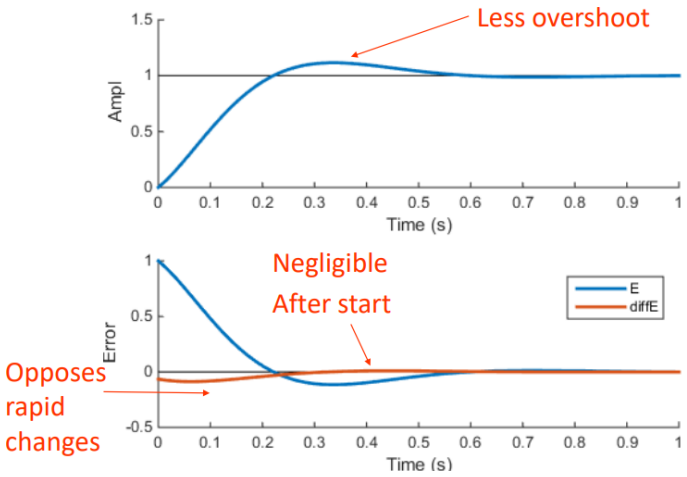


Figure 17:

At the start, the derivative term is significant *and in the opposite direction to* the proportional error term, but becomes negligible as the system settles.

0.4.5 PID control

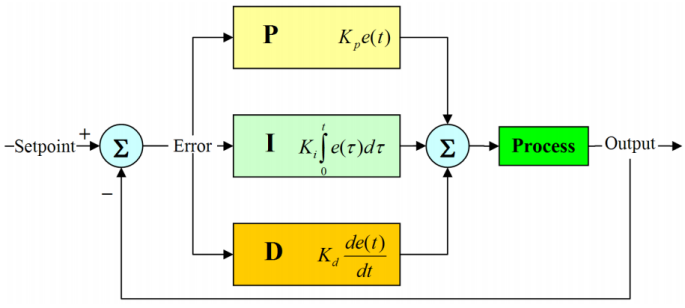


Figure 18:

0.4.6 Integral error

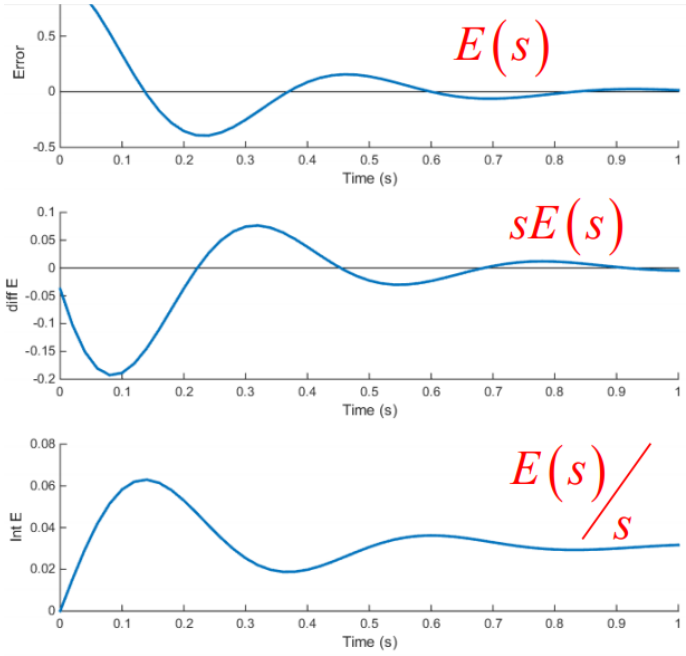
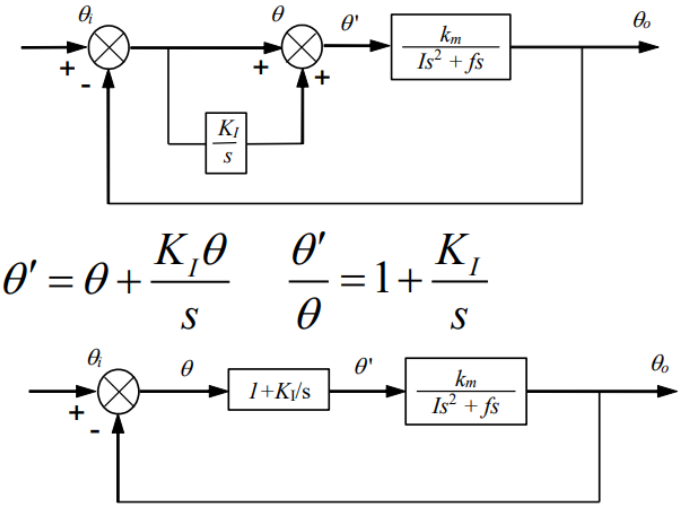


Figure 19:

Further, if we consider the **integral** or total error over time. This gradually increases over time, and can be used to magnify the control signal for small errors, and improve SSE. The integral of the error is used for correcting steady state errors, as it adds a pole at zero in the transfer function. This increases type.



$$\theta' = \theta + \frac{K_I \theta}{s} \qquad \frac{\theta'}{\theta} = 1 + \frac{K_I}{s}$$

Figure 20:

$$G' = \frac{\left(1 + \frac{K_I}{s}\right) k_m}{Is^2 + fs} = \frac{\frac{1}{s}\left(s + K_I\right) k_m}{s\left(Is + f\right)} \tag{37}$$

$$G' = \frac{k_m s + k_m K_I}{s^2\left(Is + f\right)} \tag{38}$$

With a closed loop transfer function of:

$$F(s) = \frac{G'(s)}{1 + G'(s)} = \frac{k_m s + k_m K_I}{s^2 (Is + f) + k_m s + k_m K_I} \tag{39}$$

$$F(s) = \frac{K - ms + k_m K_I}{Is^3 + fs^2 + k_m s + k_m K_I} \tag{40}$$

Eq.40 is a third order transfer function. Increasing the system type improves the steady state response:

$$k_v = \lim_{s \to 0} sG'(s) = \frac{s (k_m s + k_m K_I)}{s^2 (Is + f)} \tag{41}$$

$$k_v = \frac{k_m K_I}{0} = \infty \tag{42}$$

$$\text{SSVL} = \frac{a}{k_v} = 0 \tag{43}$$

We now have no velocity lag! The effect of an extra pole close to the origin - which would make our system *very slow* - is largely cancelled out by the nearby zero.

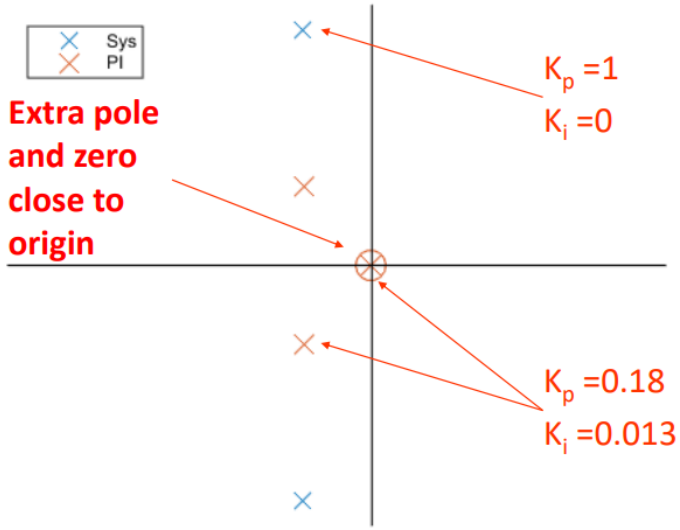


Figure 21:

This means the system is broadly similar to a proportional controller, with the exception of improved steady state performance.

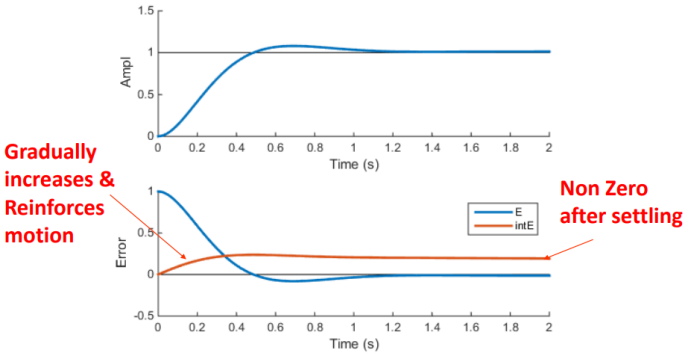


Figure 22:

We can see that the integral term has enabled proper tracking of a ramp input.

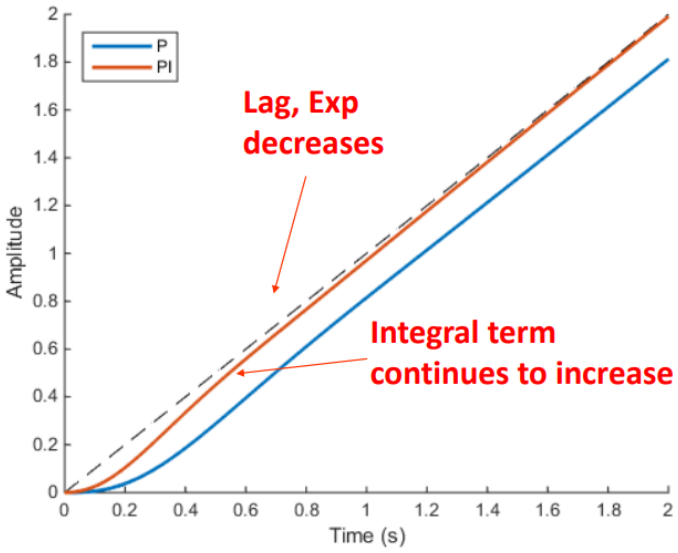


Figure 23:

This is useful for overcoming steady state errors arising from nonlinearities not included in the model which prevent proper tracking in reality.

0.4.7 PID control

Putting all three of these controllers together gives the complete PID.

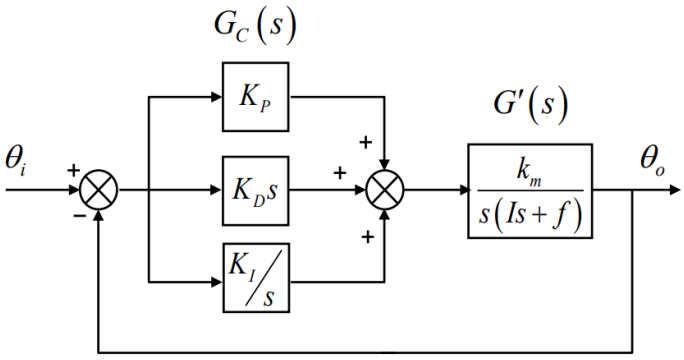


Figure 24:

$$G_c(s) = K_P + K_D s + \frac{K_I}{s} = \frac{K_D s^2 + K_P s + K_I}{s} \tag{44}$$

Open loop gain:

$$G'(s) = \frac{k_m \left(K_D s^2 + K_P s + K_I\right)}{s^2 \left(I s + f\right)} \tag{45}$$

The closed loop transfer function is then:

$$F(s) = \frac{k_m \left(K_D s^2 + K_P s + K_I\right)}{s^2 \left(Is + f\right) + k_m \left(K_D s^2 + K_P s + K_I\right)} \tag{46}$$

$$F(s) = \frac{k_m K_D s^2 + k_m K_P s + k_m K_I}{Is^3 + fs^2 + k_m K_D s^2 + k_m K_P s + k_m K_I} \tag{47}$$

$$F(s) = \frac{k_m K_D s^2 + k_m K_P s + k_m K_I}{Is^3 + \left(f + k_m K_D\right) s^2 + k_m K_P s + k_m K_I} \tag{48}$$

Thus, by choosing the appropriate values of K_P, K_D and K_I, it is possible to design a controller with improved transient response **and** decreased/no steady state error (in theory). There is no unique solution for the settings of the three gains, and so the values are dependent upon the specific system and the application.

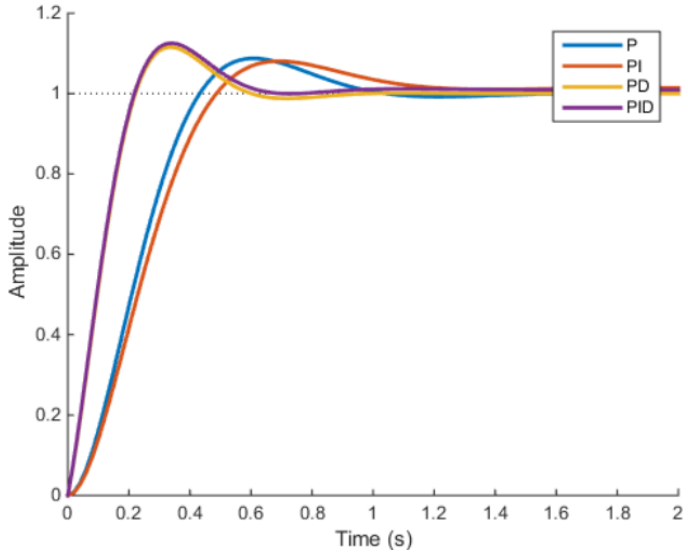


Figure 25:

Selecting these parameters is known as **tuning**, and it was (and still is in machine learning circles) a very active area of research. These controllers are used everywhere, and can even be applied with acceptable results with little or no knowledge of the system being controlled!

If we set one of the gains to zero, then we remove that term from the controller, e.g.

$$K_I \to 0 \text{ PID becomes PD controller} \tag{49}$$

Qualitatively, the three terms can be thought of as follows:

- Proportional - tries to reach target as soon as possible
- Derivative - resists overshooting
- Integral - Corrects for steady state errors

Consider our servo model, each controller changes the step response in the following ways:

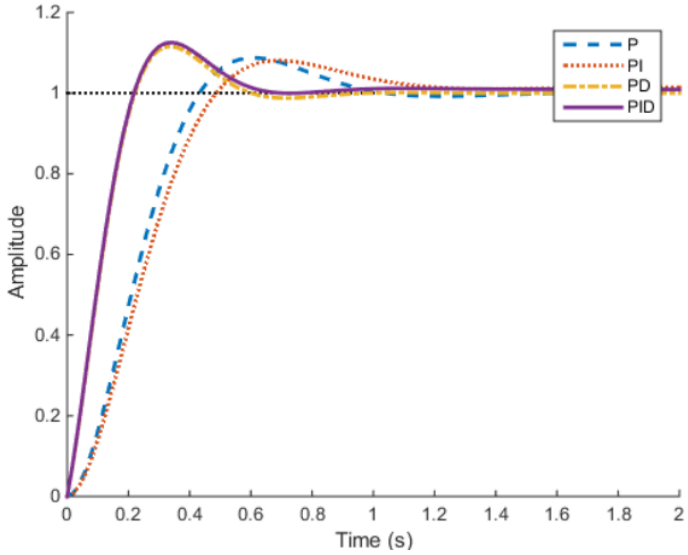


Figure 26:

P - Initially chosen to give ζ close to 0.7 for acceptable transient response. PI - I term has little effect on the transient response for this **TYPE 1** system. Steady state error technically 0, but may improve in practice. Open loop transfer function:

$$G(s)H(s) = \frac{(s-z_1)(s-z_2)(s-z_3)\ldots}{s^p(s-\sigma_1)(s-\sigma_2)(s-\alpha_k+j\omega_k)(s-\alpha_k-j\omega_k)\ldots} \tag{50}$$

No. Integrators in denominator = system TYPE	Input type		
	Step $r(t)=a$ $R(s)=\frac{a}{s}$	Ramp $r(t)=at$ $R(s)=\frac{a}{s^2}$	Acceleration $r(t)=\frac{at^2}{2}$ $R(s)=\frac{a}{s^3}$
0	$e_{ss}=\frac{a}{1+k_p}$	$e_{ss}=\infty$	$e_{ss}=\infty$
1	$e_{ss}=0$	$e_{ss}=\frac{a}{k_v}$	$e_{ss}=\infty$
2	$e_{ss}=0$	$e_{ss}=0$	$e_{ss}=\frac{a}{k_a}$

Table 2:

PD - D term improves transient response as enabling higher P weighting to be used, but with a reduced overshoot. Steady state unchanged. PID - Again, transient response largely unchanged, but potential benefits in steady state.

0.4.8 Choosing the weightings

The above figure was generated using a transfer function of a model of a servo system, and MATLAB's implementation of some commonly used "tuner" algorithms, where err on the side of caution.

THESE ARE ONLY AS GOOD AS YOUR MODEL

0.4.9 PID control

Implementing a basic PID controller is a comparatively simple task compared to that of analysing the system and choosing the coefficients. An example Arduino

code is showing, assuming **Setpoint** determined elsewhere and **Input** receives the data from the feedback path.

```
  void Compute()
{
  /*How long since we last calculated*/
  unsigned long now = millis();
  double timeChange = (now - lastTime);

  /*Compute all the working error variables*/
  double error = Setpoint - Input;

  errSum = errSum+ (error * timeChange);

  double dErr = (error - lastErr) / timeChange;

  /*Compute PID Output*/
  Output = kp * error + ki * errSum + kd * dErr;

  /*Remember some variables for next time*/
  lastErr = error;
  lastTime = now;
}
```

The process above is to get dT, calculate the error $E(s)$, update dE $\left(\frac{E(s)}{s}\right)$, est $\frac{\mathrm{d}E}{\mathrm{d}t}$ $sE(s)$. We also know:

$$G_c(s) = K_P + K_D s + \frac{K_I}{s} \tag{51}$$

The new control signal **Output** is then sent to the system. This code could be used for a vast majority of simple applications, with the **input** and **output** and **set point**, being application specific. However, selecting the correct setpoint is easier said than done! In fact much of the field of robots could be seen (very cleverly) deciding *what* the next set point should be.