Brief Intro to Practical Control Design

Given a linear time-invariant (LTI) dynamic system, we can specify a set of coupled ordinary differential equations to describe its motion. Take for example the classic mass-spring system with a driving force. By summing forces, the dynamic equation can be determined to be

$$\begin{aligned} m\ddot{x} + c\dot{x} + kx &= u(t) \\ \ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n x &= \frac{u(t)}{m} \end{aligned} \tag{1}$$

where $\zeta = \frac{c}{2\sqrt{km}}$ is the damping coefficient, ω_n is the undamped resonant frequency in radians per second and $u(t)$ is the control force that is applied to the mass. The control problem reduces to the following

Find a function $u(t)$ such that $x(t)$ behaves as desired.

Really any function $u(t)$ that works is fine! So, this is a very open design problem. For example, lets say we'd like to have $x(t)$ settle at $x = 1$. From a static analysis, we know that when the system finally comes to rest $\ddot{x} = 0$ and $\dot{x} = 0$, so we're left with

$$\begin{aligned} kx &= u(t) \\ \therefore u(t) &= k * 1 \end{aligned}$$

Evidently, to achieve the goal we simply just need to apply a force of $1 \cdot k$ to the system. This approach is called an open-loop approach. It works well **if and only if** the system is inherently stable, the system has been painstakingly calibrated, and there are no unexpected external influences (e.g. wind, vibration, power fluctuations). The open-loop approach does not need to be a constant, really any time verifying function that does not depend on the dependent variable, x in this case, will do. For example, one could imagine achieving a faster response by applying an initial control of $3k$ and, after a few milliseconds, backing off to $1k$. In fact, one can even apply calculus-of-variations to define the *best* force trajectory to achieve some desired objective. However, any open loop approach fails when there are either unmodeled affects or disturbances to the system.

To achieve better results than open-loop control, one can design a closed-loop control system. The primary difference is a closed loop controller relies on the dependent variable, x in this case. The most basic closed-loop control approach is called bang-bang control. It defines the control as

$$u(t) = \begin{cases} +u_{max} & x(t) < x_{des} \\ 0 & x(t) = x_{des} \\ -u_{max} & x(t) > x_{des} \end{cases} \tag{2}$$

Crudely, it can be thought of as: If you're going too slow, slam on the gas; if your going too fast, slam on the breaks; if your going just right, coast. It turns out that this control approach is mathematically the most robust to uncertainties in the model and disturbances from outside forces. However, as your car-sick passenger might attest, it leads to a very jerky ride! Thus, we need a slightly more sophisticated approach to achieve a desirable smooth performance.

Linear Feedback: Time Response Design

Another approach, and the preferred method, is to implement a linear controller. With linear control, the function $u(t)$ is chosen as a linear combination of system's dependent variables. For the mass spring system above, the standard choice is

$$u(t) = -k_p x - k_d \dot{x} + u_{const} \tag{3}$$

Here u_{const} is the open-loop steady state force calculated above so that the final value is as desired, k_p is called the proportional gain and k_d is called the derivative gain, leading this to commonly be refered to as PD control. When this function is used the differential equation becomes

$$\begin{aligned} m\ddot{x} + c\dot{x} + kx &= k_p x + k_d \dot{x} + u_{const} \\ m\ddot{x} + (c + k_d)\,\dot{x} + (k + k_p)\,x &= u_{const} \\ \therefore \ddot{x} + \frac{(c + k_d)}{m}\dot{x} + \frac{(k + k_p)}{m}x &= \frac{u_{const}}{m} \end{aligned} \tag{4}$$

Given (1) and (4) we can solve for the steady-state, i.e. when $\ddot{x} = \dot{x} = 0$, and dynamic performance characteristics (natural frequency ω_n and damping ratio ζ)

$$\begin{aligned} x_{ss} &= \frac{k + k_p}{u_{const}} \\ \omega_n^2 &= \frac{(k + k_p)}{m} \\ 2\zeta\omega_n &= \frac{(c + k_d)}{m} \end{aligned}$$

Given a desired steady state value x_{ss}, a desired damping coefficient ζ, and a desired natural frequency ω_n, the controller gains are then chosen to be

$$k_p = m\omega_n^2 - k \tag{5}$$

$$k_d = 2\zeta\omega_n m - c \tag{6}$$

$$u_{const} = m\omega_n^2 x_{ss} = (k + k_p)\,x_{ss} \tag{7}$$

Here ω_n controls how fast the system converges, or equivalently, how fast of a sine wave it can follow; ζ controls how much overshoot the system will have to a step input; and x_{ss} is the final steady-state value for the function. Of course, we have only switched from having to pick k_p, k_d, and f_{const} to picking x_{ss}, ω_n, and ζ. The benefit is that the latter three terms are general to any second-order system and there exists design guidelines for selecting them. Take for instance the under-damped ($0 < \zeta < 1$) response to a step input in force:

$$x(t) = 1 - \frac{1}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n t} \sin\left(\omega_n\sqrt{1-\zeta^2}\,t + \arccos(\zeta)\right)$$

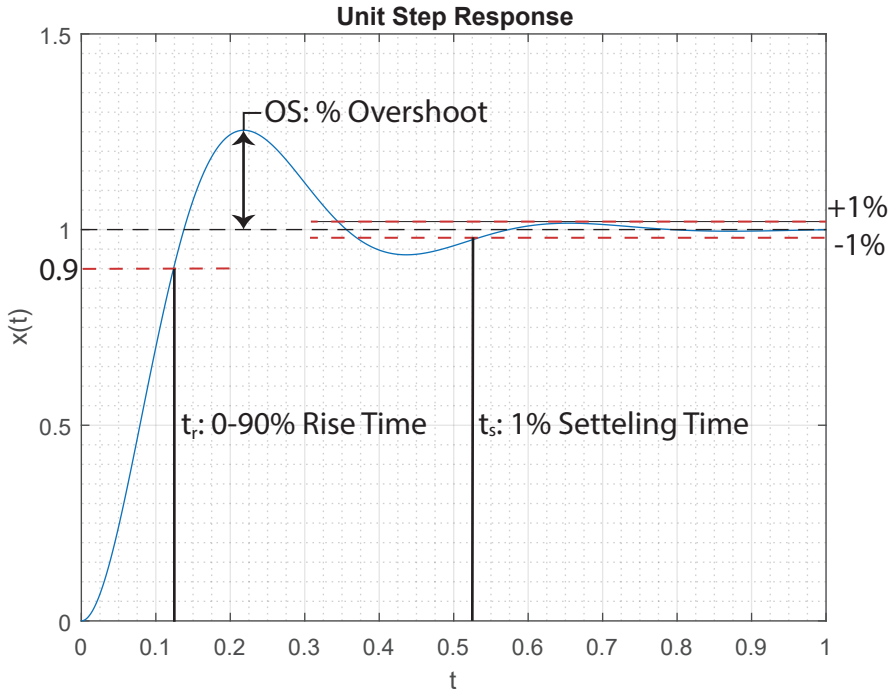
which can be used to show

$$\zeta = \sqrt{\frac{\ln(OS)^2}{\pi^2 + \ln(OS)^2}} \tag{8}$$

$$\omega_n = \frac{-\ln(1\%)}{\zeta t_{s,1\%}} \approx \frac{4.6}{\zeta t_{s,1\%}} \tag{9}$$

$$\omega_n = \frac{1.53 + 2.31\zeta^2}{t_{r,0-90\%}} \tag{10}$$

$$\omega_n = \frac{\pi}{t_p\sqrt{1-\zeta^2}} \tag{11}$$



Thus, given a desired step response profile including rise time, settling time, peak time, and percent overshoot, the necessary damping and natural frequency are determined and the gains can be calculated. Often, a choice of $\zeta = 0.707$ yields desirable performance with less than 5% overshoot. Given that, a choice of $\omega_n = \frac{5.8}{t_p}$ will yield a system that reaches its maximum overshoot at t_p seconds and proceeds to converge to steady state. Want faster response? Decrease t_p or increase ω_n. Want less overshoot? Increase ζ and recalculate ω_n.

Where To Start

Pick time to peak t_p based on your desired system response and steady state position x_{ss} based on your design objectives.

1. $\zeta \Leftarrow \sqrt{2}/2 = 0.707$
2. $\omega_n \Leftarrow 5.8/t_p$
3. $k_p \Leftarrow m\omega_n^2 - k$
4. $k_d \Leftarrow 2\zeta\omega_n m - c$
5. $u_{const} \Leftarrow m\omega_n^2 x_{ss} = (k + k_p)\, x_{ss}$
6. $u_{\text{total}} \Leftarrow -k_p x - k_d \dot{x} + u_{const}$

Laplace Space Design

The above design method works well, for a simple second order system in which both the state and its output are measured (full state feedback). However, as the system gets more complicated, the design goals get more strict, or only a single quantity is monitored, a more general approach is necessary. Take the second order system again, but this time lets look at its Laplace Transform.

$$\begin{aligned} m\ddot{x} + c\dot{x} + kx &= u(t) \\ \Rightarrow \left(ms^2 + cs + k\right) \mathcal{L}\left(x(t)\right) &= \mathcal{L}\left(u(t)\right) \\ \Rightarrow \left(ms^2 + cs + k\right) X(s) &= U(s) \end{aligned}$$

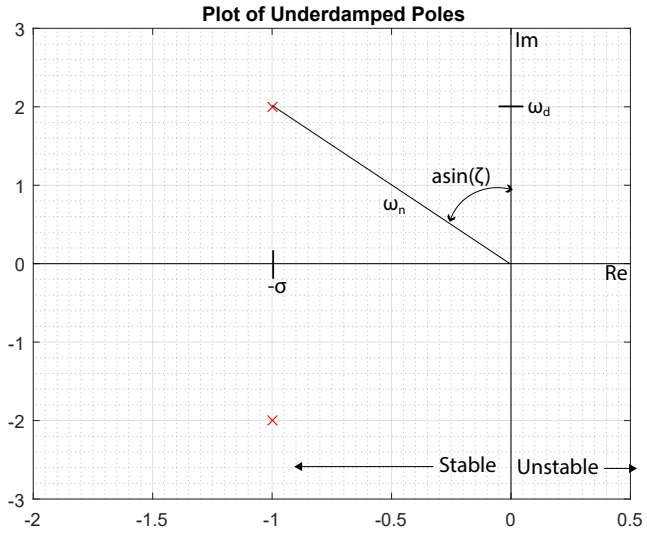
This can be transformed into a transfer function, an algebraic relationship between the forcing function and the output.

$$TF = \frac{X(s)}{U(s)} = \frac{1}{ms^2 + cs + k}$$

The denominator is called the characteristic polynomial, setting it to zero yields the ***characteristic equation***. The roots of the characteristic equation give what are interchangeably called the poles or the eigenvalues of the system. For an under-damped second order system system, the poles are the solutions to

$$\begin{aligned} ms^2 + cs + k &= 0 \\ s^2 + 2\zeta\omega_n s + \omega_n^2 &= 0 \\ \Rightarrow s &= -\omega_n\zeta \pm \boldsymbol{j}\omega_n\sqrt{1-\zeta^2} \\ &= -\omega_n\left(\zeta \pm \boldsymbol{j}\sqrt{1-\zeta^2}\right) \\ &= -\sigma \pm \boldsymbol{j}\omega_d \end{aligned}$$

Traditionally, these are plotted on the Real-Imaginary plane as x's.



Systems of higher order have higher order poles, but their qualitative characteristics are the same. The more negative the pole (eigenvalue) the faster its contribution to the transient response disappears, the larger the angle to the real axis, the more oscillatory the response from that pole. We refer to the ***dominant poles*** as the ones with least-negative (closest to positive) real part because they take the most time for their affect to dissipate. Often it is common to ignore poles that are five times ($5\times$) more negative in their real part than the most dominant pole, because these poles' affects on the transient settles out quicker than $1/5$ the time of the dominant pole. When designing a controller for a higher-order system using the above relationships to peak-time, etc, it is common to pick two-dominant poles based on the desired 2nd order response, then place the remaining poles at locations 3-$10\times$ more negative and commonly on the real axes to prevent high-frequency oscillation.

Filter Inspired Design

Reflecting on the intent of a control system–to track a desired input signal while rejecting environmental noise–we see a distinct similarity between the performance of a closed loop controlled system and an open-loop signal filter. Specifically, the closed loop control system should, in essence, behave like a filter on the reference signal. The signal processing community has spent significant effort developing filters optimized

to certain criteria. Possibly the most commonly used filter in industry for data collection, the Butterworth filter provides an optimally-flat pass-band–meaning that it preserves the Fourier (frequency domain) Power Spectrum amplitudes of the signal optimally. However, while the Butterworth filter does an excellent job of preserving the frequency-domain Power Spectrum of the recorded signal, it does distort the time-domain signal because it distorts the phase-shift of the signal's harmonic components. Although widely popular Butterworth filters are not the only optimal filter, the Bessel filter compromises on the pass-band flatness to optimize for a constant phase shift–meaning that the output signal is temporally as close to the input signal as possible given the desired filtering cutoff frequency. Considering this in a controls context, this means a step-response looks as much like the input step as possible–a desired trait for a control system. When designing a control system, we can re-cast the problem from specifying overshoot and settling times to choosing the poles of closed-loop transfer function to match the poles of an optimal low-pass filter, like a Bessel Filter. One benefit of pole-placement design based on optimal-filtering is that any order system can be accommodated without arbitrary placing poles in fast (negative) locations in the s-plane. A second benefit is the selection of system dynamic performance is reduced to only one parameter–the desired time to peak for a step response.

The pole locations can be computed directly from the Bessel polynomials, but practically this is facilitated nicely in Matlab using the *besself* function. For a for a second order system the time to peak is very nicely approximated from Matlab's cutoff frequency definition. For higher order systems, there is a scaling factor required, which is provided in the below table. To compute the desired pole locations in Matlab use:

$$[\tilde{},\text{Poles},\tilde{}] = \text{besself}(\text{N}, {}^{2\pi\gamma}/_{T_p})$$

Table 1: Time to Peak Scaling Factors

System Order (N)	Factor γ
1	1
2	0.998659
3	1.05177
4	1.15983
5	1.29069
6	1.42984
7	1.56765
8	1.72119
9	1.87101
10	2.02926

Block Diagrams

Control systems are often described using block diagrams. Because time-domain convolution is multiplication in the Laplace domain, this structure gives a graphical interpretation of what is happening. Generally, the physical system being controlled (also known traditionally as the *Plant* in controls) is represented by the transfer function $G(s)$ and the controller system is represented by the transfer function $H(s)$. The signals in a block diagram are: $y_d(s)$ the desired output, $y_m(s)$ the measured output, $e(s)$ the error $(y_d - y_m)$, $u(s)$ the control signal, and $d(s)$ a disturbance signal (something that affects your performance–but is outside of your control–like wind, unmodeled friction, or your lab partner messing with your setup). The full dynamics of the closed loop system are captured in the closed loop transfer function

$$T_{cl} = \frac{G(s)H(s)}{1 + G(s)H(s)} \tag{12}$$

Final Value Theorem

If you have verified that T_{cl} is a stable system, i.e. all of the roots to its characteristic equation have negative real part, then you can use the final value theorem to calculate the system's steady-state performance for an

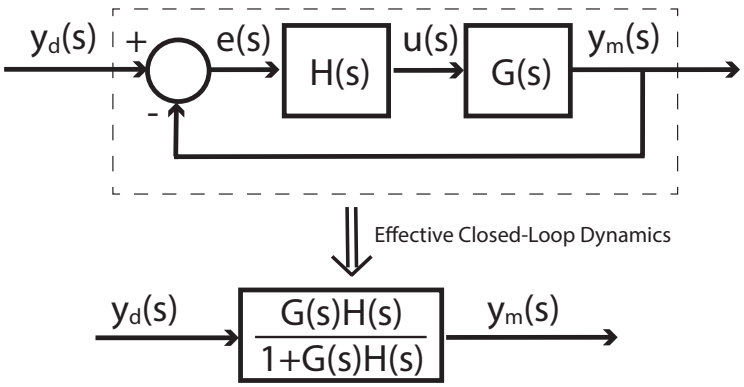


Figure 1: Traditional block diagram for closed loop control and resulting closed-loop system single-block equivalent.

input $I(s)$.

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} sT(s)I(s) \tag{13}$$

For a static input (or step response $I(s) = 1/s$), this reduces to:

$$\lim_{s\to 0} T(s)$$

Basically, if you want the system to have zero steady-state error then the open loop system gain

$$Gain_{steadyState} = \lim_{s\to 0} GH \tag{14}$$

defines the stiffness of the system to rejecting error at steady state. For that matter, if you want to have perfect tracking of any signal you want the gain for the open-loop transfer function (just the numerator GH) to go to infinity at some point. That is, for perfect constant-reference tracking, then we desire

$$\lim_{s\to 0} GH = \infty.$$

If we wish to track a sine wave, or any periodic motion, perfectly then we wish

$$\lim_{s\to j\omega} |GH| \Rightarrow \infty$$

where ω is the frequency in $^{\text{rad}}/_{\text{s}}$ of the periodic signal to follow. Thus, the controls problem becomes one of defining the polynomial filter $H(s)$ to place the zeros of the closed loop characteristic equation, $1 + G(s)H(s) = 0$, in the desired location and to set $\lim_{s\to j\omega} |GH|$ to be as large as possible at the desired frequency(ies).

PID Control

There are many possible ways of developing a $H(s)$ function, a standard choice is proportional integrative derivative control, or commonly PID for short. It works very well for second order systems and many systems we encounter are second order.

$$\begin{aligned} H_{\text{PID}} &= K_p + K_d s + K_i \frac{1}{s} \\ &= \frac{K_d s^2 + K_p s + K_i}{s} \end{aligned} \tag{15}$$

Which follows the following intuition: The proportional K_p term penalizes error and acts like a restoring spring, the derivative K_d term penalizes velocity and helps to dampen the motion, the integral $K_i$term

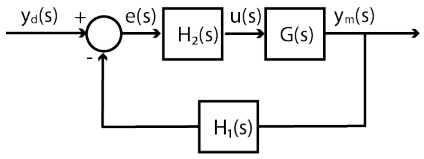


Figure 2: Split PID control to separate filtering from integration.

adapts to offsets and removes steady-state error (and can replace the u_{ss} from above). In fact, one can think of the integrator term as the first adaptive controller as it learns $u_{ss} \Leftrightarrow K_i \int edt$ as the system evolves.

However, when applied as presented in Fig. 1, the H_{PID} transfer function applies equally to the reference signal as it does the measurement, which is a potential issue for the derivative term. Whenever the reference changes abruptly or discontinuously, the derivative term requests significant (potentially infinite) control to be applied, but this is not meeting the intent of the design (damping velocity). To address this, the implementation of the system should split the block diagram into two separate parts: 1) a dynamic filter on the measurement, with a unit conversion K_u to ensure the reference and the measurement have the same units, and 2) a proportional error penalty and integrator on the error, as shown in 2 where

$$H_1 = K_u \left(1 + \frac{K_d}{K_p} s\right)$$
$$H_2 = K_p + \frac{K_i}{s}$$

One problem still remains though, the pure derivative still exists in H_1. This makes the system both unimplementable–you cant take a pure instantaneous derivative because you need to check the system at two separate times to see how its changing–and highly prone to noise. The solution is to treat H_2 as a filter where the measured value is low-pass filtered and differentiated in one step, this reduces the noise problem and introduces a time-delay (from the filter) so two times can be compared in the past. This is called ***bandwidth limited differentiation*** and is equivalent to ***lead-lag control*** which is often studied in undergraduate controls classes. It takes the form

$$H_1 = 1 + \frac{K_d}{K_p} \frac{s\alpha}{s + \alpha}$$
$$H_2 = K_p + \frac{K_i}{s}$$

Effectively, bandwidth limited differentiation limits the frequency content in any step input to be below $\approx 2\alpha$. By selecting α to be a frequency in $^{\text{rad}}/_{\text{s}}$ 2x higher than the desired maximum input frequency, the desired control will still be followed while the effects of noise are mitigated. Alternatively, α can be selected during pole placement as a free parameter to help reduce the gains K_p, K_d, and K_I while achieving the same response, as lower gains mean less control is necessary.

PID Based Pole Placement

The PID controller architecture provides 4 parameters (K_p, K_d, K_I, α) with which to tune the system, at the expense of adding two additional poles to the closed loop transfer function (one extra if either the integrator or derivative terms are ignored). When applied to a first or second order system, this allows exact pole placement by selecting the gains and filter coefficients appropriately since they are the same number of poles as tuning parameters. For higher-order plants, full pole placement can be achieved by adding additional higher-order bandwidth limited differentiation terms, e.g. $^{s^2\alpha^2}/_{(s+\alpha)^2}$. Take for example the second order plant

$$G = \frac{d}{(s + b)(s + c)} \tag{16}$$

The closed loop characteristic equation is $1 + GH_1H_2 = 0$ or specifically

$$0 = 1 + \frac{(K_i + s)\left(\alpha K_p + (K_p + K_d\alpha)\, s\right)}{s^2 + \alpha s} \frac{d}{(s+b)(s+c)} \tag{17}$$

$$= (s+b)(s+c)\left(s^2 + \alpha s\right) + d(K_i + s)\left(\alpha K_p + (K_p + K_d\alpha)\, s\right) \tag{18}$$

which is quadratic in s and bi-linear in the coefficients (K_p, K_d, K_I, α). Given four desired pole locations $\{\lambda_1 \cdots \lambda_4\}$, which can be chosen using the step-response criteria discussed above or selected to match the dynamic characteristics of some optimal-filter system, this can be reduced to four nonlinear equations

$$\lambda_1(\lambda_1+b)(\lambda_1+c)\left(\lambda_1+\alpha\right) + d(K_i+\lambda_1)\left(\alpha K_p + (K_p + K_d\alpha)\,\lambda_1\right) = 0$$
$$\lambda_2(\lambda_2+b)(\lambda_2+c)\left(\lambda_2+\alpha\right) + d(K_i+\lambda_2)\left(\alpha K_p + (K_p + K_d\alpha)\,\lambda_2\right) = 0$$
$$\lambda_3(\lambda_3+b)(\lambda_3+c)\left(\lambda_3+\alpha\right) + d(K_i+\lambda_3)\left(\alpha K_p + (K_p + K_d\alpha)\,\lambda_3\right) = 0$$
$$\lambda_4(\lambda_4+b)(\lambda_4+c)\left(\lambda_4+\alpha\right) + d(K_i+\lambda_4)\left(\alpha K_p + (K_p + K_d\alpha)\,\lambda_4\right) = 0$$

Another common situation is to apply PD control to second-order plant with at least one pole at zero (this for example arises from a DC motor with position feedback), rendering the integration term redundant. In this case, the resulting characteristic equation system is third order

$$0 = 1 + \frac{K_p\,(s+\alpha) + K_d s}{s+\alpha} \frac{c}{s(s+b)}$$
$$0 = s(s+\alpha)(s+b) + cK_p\,(s+\alpha) + cK_d s$$

and three poles can be placed as desired by picking K_p, K_d, and α, by minimizing the residual characteristic equation values at those desired pole locations. Although solutions do not always exist for any given set of λ's due to the non-linearity from cross terms (e.g. $K_p\alpha$), many designs can be achieved though algebraic solution or numerically by minimizing the residuals of the equations. Caution should be taken, if a set of λ's results in negative values for any parameter. This implies positive feedback, which can make the system unstable. Thus, it is considered undesirable to have negative gains and the desired closed-loop dynamics should be sped-up (λ's more negative in their real part) to achieve positive gains. If a slower response is desired in practice, this should be handled by crafting the reference trajectory to have the slower ramp-up performance characteristics desired.

Implementation: Discrete-Time Control

Roboticists and Engineers often implement their control designs not with analog electrical, kinetic, or hydraulic circuits, but rather with embedded micro-controllers and programming. This design choice leads to much more flexible system tuning as well as the implementation of *smarter* control loops with built-in error condition detection, control-limiting capability, and embedded trajectory design. However, the cost of this flexibility is that the systems operate in discrete intervals with, most commonly, a *zero-order-hold* on the control signals between control-decision changes, that is the control stays constant until a new decision is made. Think of it as walking through a night club with a strobe light. Each flash of the light gives you new information about the system around you, but you have to walk blindly during the dark periods. The faster the strobe, the easier it is to navigate; the slower the strobe the more care is necessary. The same is true for controller design. If the update frequency is exceptionally fast compared to the system dynamics (say $30\times$ faster), the gains calculated using the continuous methods above are just fine. However, as the update frequency becomes slower (and closer to the real part σ frequencies of the system's open loop or closed loop poles), the effect of holding the control constant begins to be problematic and can even make the system unstable.

The solution to this is to map the control from a continuous-time system to a discrete time system – using what is known as a *z-Transform*. The z-Transform yields an algebraic transfer function just like the Laplace (s) transform, but instead of it being in frequency domain, it operates in a discretized each-update domain. Although complicated to intuitively understand, implementation and use is actually quite straight

forward. To state it more simply, read z as next and z^{-1} as last, z^2 is the next-next, z^{-2} is the last-last, and z^0 is now, etc. Take the transfer-function encoded control law

$$H(z) = \frac{u(z)}{e(z)} = \frac{B(z)}{A(z)} = \frac{0.95z^{-1} + 0.8}{0.68z^{-1} + 1.0},$$

note how all terms have been expressed as negative (or zero) powers of z, this is because you cant know whats to come but you can know what has happened. To implement this we do some algebraic rearrangement,

$$\begin{aligned} u(z)A(z) &= e(z)B(z) \\ u(z)(0.68z^{-1} + 1.0) &= e(z)(0.95z^{-1} + 0.8) \\ 0.68u_{k-1} + 1.0u_k &= 0.95e_{k-1} + 0.8e_k \\ u_k &= \frac{0.8e_k + 0.95e_{k-1} - 0.69u_{k-1}}{1.0}, \end{aligned}$$

which can be read: the new control is: 0.8 times the new error, plus 0.95 times the prior error, minus 0.69 times the prior control, all divided by 1.0. Thus for this system, the micro-controller needs to keep track of the previous control and error between updates to calculate the new control with the new error. In a more general mathematical form

$$u_k = \frac{\sum_{i=0\ldots N} B_i e_{k-i} - \sum_{i=1\ldots N} A_i u_{k-i}}{A_0}, \tag{19}$$

where u_{k-i} is the control from i updates in the past and e_{k-i} is the error from i updates in the past. What is left is how to calculate the B and A terms given our desired continuous time control law $H(s)$.

To convert from the continuous-time Laplace-space control law $H(s)$ to a z-transform, we take advantage of a relation ship that

$$\text{step} = \frac{1}{s} \Rightarrow \frac{z-1}{z} \tag{20}$$

$$t = \frac{1}{s^2} \Rightarrow \frac{Tz}{(z-1)^2} \tag{21}$$

$$e^{-at} \Rightarrow \frac{1}{s+a} \Rightarrow \frac{z}{z - e^{-aT}} \tag{22}$$

$$\text{delay T} = 1 - e^{-Ts} \Rightarrow \left(1 - z^{-1}\right) \tag{23}$$

$$\text{z.h.o.} = \frac{1 - e^{-Ts}}{s} \Rightarrow \left(1 - z^{-1}\right) \frac{z-1}{z} \tag{24}$$

where T is the discrete update period in seconds, and z.h.o is a zero-order-hold of T seconds. Given the continuous time transfer function $H(s)$, the discretized version becomes

$$\begin{aligned} H(z) &= \mathcal{Z}\left\{\frac{1 - e^{Ts}}{s} H(s)\right\} \\ &= \mathcal{Z}\left\{1 - e^{Ts}\right\} \mathcal{Z}\left\{\frac{H(s)}{s}\right\} \\ &= \left(1 - z^{-1}\right) \mathcal{Z}\left\{\frac{H(s)}{s}\right\} \end{aligned} \tag{25}$$

We then need to use a partial-fraction expansion to convert $H(s)/s$ into a series of $C/(s+a)$ polynomials. That is find the coefficients C and frequencies a such that $H(s)/s$ can be expanded to a sum

$$\frac{H(s)}{s} = \sum_{i=0\ldots N} \frac{C_i}{s + a_i},$$

Given this, the z-transform is

$$H(z) = \left(1 - z^{-1}\right) \sum_{i=0\ldots N} C_i \frac{z}{z - e^{-a_i T}}.$$

Take, for example, the bandwidth limited PID control law from above, we need to discretize the two filters H_1 and H_2, so they can be implemented in a micro-controller with an update happening every T seconds. To do this we'll start with H_2 and use the identities from (20) to (24) to assist with the conversion.

$$\begin{aligned} H_2\{z\} &= \mathcal{Z}\left\{\frac{1-e^{Ts}}{s}H_2(s)\right\} \\ &= \left(1-z^{-1}\right)\mathcal{Z}\left\{\frac{K_p}{s}+\frac{K_i}{s^2}\right\} \\ &= \left(1-z^{-1}\right)\left(K_p\frac{z}{z-1}+K_i\frac{Tz}{(z-1)^2}\right) \\ &= \frac{K_p+\left(TK_i-K_p\right)z^{-1}}{1-z^{-1}} \end{aligned}$$

Next, we follow the same process to discretize H_1

$$\begin{aligned} H_1\{z\} &= \left(1-z^{-1}\right)\mathcal{Z}\left\{\frac{1}{s}+\frac{K_d}{K_p}\frac{\alpha}{s+\alpha}\right\} \\ &= \left(1-z^{-1}\right)\left(\frac{z}{z-1}+\frac{K_d}{K_p}\frac{\alpha z}{z-e^{-\alpha T}}\right) \\ &= \frac{\left(1+\frac{K_d}{K_p}\alpha\right)-\left(\frac{K_d}{K_p}\alpha+e^{-\alpha T}\right)z^{-1}}{1-e^{-\alpha T}z^{-1}} \end{aligned}$$

Finally, given these transforms, we can implement the code by introducing an intermediate variable y which is the output of the H_1 filter given the measurements m. The control is then implemented by following the block-diagram math in Fig. (2) from measurement to control-input

Given: m_k (measurement), r_k (reference)
First Time Initialize: $m_{k-1} \leftarrow m_k$, $y_{k-1} \leftarrow m_k$, $e_{k-1} \leftarrow 0$, $u_{k-1} \leftarrow 0$
$y_k \leftarrow \left(1+\frac{K_d}{K_p}\alpha\right)m_k - \left(\frac{K_d}{K_p}\alpha+e^{-\alpha T}\right)m_{k-1} + e^{-\alpha T}y_{k-1}$
$e_k \leftarrow r_k - y_k$
$u_k \leftarrow K_p e_k + \left(TK_i - K_p\right)e_{k-1} + u_{k-1}$
$y_{k-1} \leftarrow y_k$
$e_{k-1} \leftarrow e_k$
$u_{k-1} \leftarrow u_k$
$m_{k-1} \leftarrow m_k$
Apply Control: u_k

Algorithm 1: Discrete Time PID Implementation

In the case of PD control this simplifies to:

Given: m_k (measurement), r_k (reference)
First Time Initialize: $m_{k-1} \leftarrow m_k$, $y_{k-1} \leftarrow m_k$
$y_k \leftarrow \left(1+\frac{K_d}{K_p}\alpha\right)m_k - \left(\frac{K_d}{K_p}\alpha+e^{-\alpha T}\right)m_{k-1} + e^{-\alpha T}y_{k-1}$
$u_k \leftarrow K_p\left(r_k - y_k\right)$
$y_{k-1} \leftarrow y_k$
$m_{k-1} \leftarrow m_k$
Apply Control: u_k

Algorithm 2: Discrete Time PD Implementation

State-Space Control Design

This section is given at a very high-level. For those interested in learning more, consider taking EENG-417 Modern Control Design

Controllers

An alternative method to Laplace-Space or root-locus design is to use a state space method. This approach has been greatly facilitated by numerical packages like Matlab. The first step in this process is to convert your ODE into a State Space (matrix) representation of coupled first order ODEs. Given

$$m\ddot{z} + c\dot{z} + kz = u(t)$$

we first define a change of variables to assist in the transformation. Let $x_1 = z$ and $x_2 = \dot{z}$, given this we have a system of ODEs

$$\begin{aligned} \dot{x}_1 &= x_2 \\ m\dot{x}_2 + cx_2 + kx_1 &= u(t) \end{aligned}$$

which can be rewritten as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{c}{m}x_2 - \frac{k}{m}k_1 + u(t) \end{aligned}$$

This can be rewritten in matrix form as

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + Bu(t) \\ z &= C\mathbf{x} + Du(t), \\ \mathbf{x} &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ A &= \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 \end{bmatrix} \\ D &= 0 \end{aligned}$$

where C has been chosen to reflect the original output of the single second order ODE.

The homogeneous equation is just $\dot{\mathbf{x}} = A\mathbf{x}$ and has the solution

$$\mathbf{x}(t) = e^{At}\mathbf{x}_0$$

where this is a matrix exponential not an element-wise exponential, i.e.

$$e^{At} = I + At + \frac{t^2}{2}A^2 + \frac{t^3}{6}A^3 + \sum_{k=4}^{\infty}\frac{t^k}{k!}A^k$$

where I is an identity matrix of the same size as A. Using an eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

we find that

$$\begin{aligned} e^{At} &= VV^{-1} + V\Lambda V^{-1}t + \frac{t^2}{2}V\Lambda V^{-1}V\Lambda V^{-1} + \frac{t^3}{6}V\Lambda V^{-1}V\Lambda V^{-1}V\Lambda V^{-1} + \dots \\ &= V\left(I + \Lambda t + \frac{t^2}{2}\Lambda^2 + \frac{t^3}{6}\Lambda^3 + \sum_{k=4}^{\infty}\frac{t^k}{k!}\Lambda^k\right)V^{-1} \\ &= Ve^{\Lambda t}V^{-1} \end{aligned}$$

Since Λ, the matrix of eigenvalues, is diagonal this is simply

$$e^{At} = V \begin{bmatrix} e^{\lambda_1 t} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & e^{\lambda_n t} \end{bmatrix} V^{-1}.$$

Thus, the dynamics of the system are determined by the eigenvalues of the matrix A with the eigenvectors just mix and matching these time response. We then define the control problem as:
Find a matrix K such that the eigenvalues of $A - BK$ ($A\mathbf{x} - BK\mathbf{x} \Rightarrow (A - BK)\mathbf{x}$) yields the close loop system dynamics desired.

We can also explore if the system can be controlled, that is can we specify a input $u(t)$ to specify a system state (x_1, x_2). Note: you can only independently control as many states as their are input u's available. This question can be answered by looking at the rank of the controllability matrix

$$\mathcal{C} = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix},$$

where n is the number of states. It turns out, by the Cayley-Hamilton theorem, that a matrix satisfies its own characteristic equation. That is, if the eigenvalues are given by the roots of the characteristic equation

$$\prod_{k=1}^{n} (\lambda_k - s) = 0,$$

then the equation

$$\prod_{k=1}^{n} (\lambda_k I - A) = 0,$$

where λ_k's are the eigenvalues of the A matrix, is also true. This can be used to calculate the feedback gains necessary to place the poles as desired:

$$K^\top = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix} \mathcal{C}^{-1} \left(\prod_{k=1}^{n} (\lambda_k I - A) \right),$$

where here λ_k are the *desired* closed loop eigenvalues and $\mathcal{C}$ is the controllability matrix discussed above. This equation is known as Ackermann's Formula and more information including the derivation is provided on its Wiki page. This is of course implemented in Matlab

K = acker(A,B,P)

where P is a vector of your desired closed loop poles. This works for a single-input (one u) system. If you have a multiple input multiple output system then the above method does not work. In that case you would use the Matlab command

K = place(A,B,P)

How do you determine the locations of the closed-loop poles? You use the design formula provided above for the step response! For your system to behave like a second-order system, there should be a greater than 5 times separation gap between the second least negative pole and the remaining poles. That is, if your system has four poles with $\mathrm{Re}(p_1) \geq \mathrm{Re}(p_2) \geq \mathrm{Re}(p_3) \geq \mathrm{Re}(p_4)$, then the design relations above will hold if $\mathrm{Re}(p_3) \leq 5\mathrm{Re}(p_2)$. The bigger the gap the better the relationships hold.

Removing steady state error and tracking dynamic trajectories: Designing K to give the closed loop system the desired dynamics does not help with any control point other than $\mathbf{x} = 0$. Instead of $\mathbf{x}$ as the state, look at the error $\mathbf{e} = \mathbf{x}_d - \mathbf{x}$, where $\mathbf{x}_d$ is the desired state.

$$\begin{aligned} \dot{\mathbf{e}} &= \dot{\mathbf{x}}_d - A\mathbf{x} - B\mathbf{u} \\ \dot{\mathbf{e}} &= \dot{\mathbf{x}}_d - A(\mathbf{x}_d - \mathbf{e}) - B\mathbf{u} \\ \text{Let:} \mathbf{u} &= \mathbf{u}_{ss} + K\mathbf{e} \quad \text{(steady state + error correction)} \\ \dot{\mathbf{e}} &= A\mathbf{e} + \dot{\mathbf{x}}_d - A\mathbf{x}_d - B(\mathbf{u}_{ss} + K\mathbf{e}) \\ \therefore \dot{\mathbf{e}} &= (A - BK)\mathbf{e} + \dot{\mathbf{x}}_d - B\mathbf{u}_{ss} - A\mathbf{x}_d \end{aligned}$$

at steady state, when $\mathbf{e} \to 0$ (hopefully) and $\dot{\mathbf{e}} \to 0$, this becomes

$$\begin{aligned} \mathbf{0} &= \dot{\mathbf{x}}_d - B\mathbf{u}_{ss} - A\mathbf{x}_d \\ B\mathbf{u}_{ss} &= \dot{\mathbf{x}}_d - A\mathbf{x}_d \\ \therefore \mathbf{u}_{ss} &= \left(B^\top B\right)^{-1} B^\top \left(\dot{\mathbf{x}}_d - A\mathbf{x}_d\right) \quad (26) \\ &= B_{ss}\left(\dot{\mathbf{x}}_d - A\mathbf{x}_d\right) \quad (27) \end{aligned}$$

Thus, the stabilizing feedback control law that achieves zero steady-state error (assuming a perfect model) is given by:

$$\mathbf{u} = \underbrace{K\left(\mathbf{x}_d - \mathbf{x}\right)}_{\text{Error Correction}} + \underbrace{B_{ss}\left(\dot{\mathbf{x}}_d - A\mathbf{x}_d\right)}_{\text{Steady-State Offset \& Dynamic Trajectory Correction}} \quad (28)$$

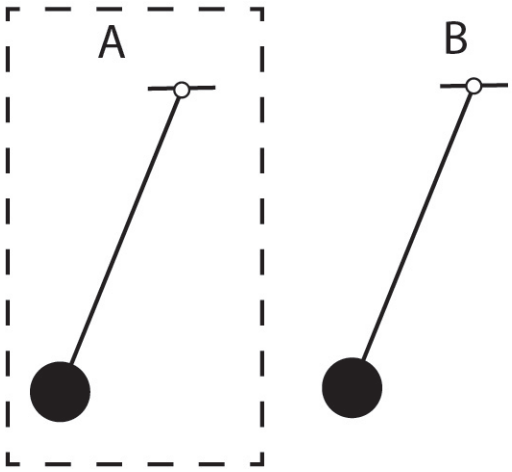
where K is selected to achieve the desire dynamic response properties of the closed loop system. Given this, the closed loop system becomes:

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B\left(K\left(\mathbf{x}_d - \mathbf{x}\right) + B_{ss}\left(\dot{\mathbf{x}}_d - A\mathbf{x}_d\right)\right) \\ &= (A - BK)\mathbf{x} + B(K - B_{ss}A)\mathbf{x}_d + B_{ss}\dot{\mathbf{x}}_d \\ &= A_{cl}\mathbf{x} + B_{cl}\mathbf{x}_d + \mathbf{u}_{dynamic} \end{aligned}$$

where A_{cl} is the effective closed loop system, describes how the desired state affects the system, and $\mathbf{u}_{dynamic}$ describes the control input necessary to track changes in the desired state.

Observers

The challenge with state space design is not in calculating the gains, as with Matlab that is fairly trivial. The challenge is with requiring knowledge of *all* of the states! With Laplace design, we only needed to know the output from our sensor. With state space design, we still only get the output from our sensor, but we need to know all of the sates (positions, velocities, etc) in order to do control. This task is called state estimation, and we do this with an observer (for those in the know, a Kalman Filter is just a type of observer).



Lets say we have two identical pendulums as shown in the figure above. Now if we have been careful about their construction, we can start pendulum A and B with the same initial conditions and they'll go on ticking exactly the same forever—this is how clocks work of course. Now what we care about is the angle and speed of Pendulum A, but we can only place a position sensor on it (for what ever reason). We can, however, place a position and speed sensor and a motor on Pendulum B. Then we can compare the position of A with the position of B and apply torque to pendulum B to help it keep time with pendulum A (accelerating if its falling behind decelerating if its going too fast). By doing this, we can also correct for dissimilarities in the Pendulum's construction and initial conditions. Now, if we take this one step further and instead of

constructing a second pendulum, we can simulate it numerically by solving its differential equation. The benefit of doing the observation numerically is that we can adjust both its position and velocity based on the data coming from Pendulum A.

For the observer we look at the state estimate $\tilde{\mathbf{x}}$ and how it evolves over time

$$\dot{\tilde{\mathbf{x}}} = A\tilde{\mathbf{x}} + B\mathbf{u}_c + \mathbf{u}_o,$$

here the matrices A and B have been selected to match the actual system's dynamics as closely as possible (equivalently to fabricating Pendulum B as closely to A as possible), $\mathbf{u}_c$ is the control input to the system, and $\mathbf{u}_o$ is a control input we choose to force the observer to track the actual system. Let's look at the error between the actual state and the observed state:

$$\begin{aligned}
\mathbf{e} &= \mathbf{x} - \tilde{\mathbf{x}} \\
\dot{\mathbf{e}} &= A\mathbf{x} + B\mathbf{u}_c - A\tilde{\mathbf{x}} - B\mathbf{u}_c - \mathbf{u}_o \\
\dot{\mathbf{e}} &= A\mathbf{e} - \mathbf{u}_o
\end{aligned}$$

if we choose (since we can choose it to be anything we want) $\mathbf{u}_0 = LC\mathbf{e}$, we get

$$\begin{aligned}
\dot{\mathbf{e}} &= A\mathbf{e} - LC\mathbf{e} \\
\dot{\mathbf{e}} &= (A - LC)\mathbf{e}.
\end{aligned}$$

If we choose the eigenvalues of $A-LC$ to be negative real part (stable), we guarantee the error will converge to zero! This is accomplished by picking the gain matrix L such that the observer poles are in desired locations. This problem is (almost) the same as the control problem. If we take the transpose of this equation:$\dot{\mathbf{e}}^\top = (A^\top - C^\top L^\top)\mathbf{e}^\top$, it is still completely valid and has the same eigenvalues. Thus, we need to find $L^\top$such that $A^\top - C^\top L^\top$has the eigenvalues desired, which mathematically is identical to finding K such that $A - BK$ has the desired eigenvalues! Thus, we get the observability matrix:

$$\mathcal{O} = \begin{bmatrix} C^\top & A^\top C^\top & \cdots & \left(A^\top\right)^N C^\top \end{bmatrix},$$

which, if it is full rank, tells us if we can see estimate all of the system's states based on the measurements coming in. To calculate the gain L we use the same Matlab functions as we did for control (remembering to transpose the terms and the result, as appropriate) L = acker(A',C',P)' for a MIMO system L = place(A',C',P)' where the ' indicates the transpose in Matlab. Now, we just need to decide which poles P we wish to have.

An observer acts like a filter. The slower the poles (less negative), the more filtering (averaging) will happen. If our sensors have low noise, we can make P faster (more negative) than our control system because we can trust the readings. If the sensors have a lot of noise, we may need to put P slower than our system (less negative) to smooth out the noise. The later choice will affect the dynamic response (you cant make something move fast if you cant track it fast enough). Another approach to picking gains is to draw inspiration from known filter designs, like the Butterworth, Bessel, Chebyshev, etc. Thus, you create an observer that has the frequency response characteristics of these optimized designs while simultaneously supplying the other states. For example, if you wish to have a 20rad/s cut-off frequency with Butterworth characteristics you'd use the following in Matlab:

[~,P,~] = butter(size(A,1),20,'s');

L = place(A',C',P)'.

Another valid option is a Bessel filter, which is designed to have constant phase lag. The result of this is that the output signal has the same (as close as possible) temporal shape as the input signal given the reduction in high-frequency content, e.g., it keeps square-waves as square possible. For this filter you'd type:

[~,P,~] = besself(size(A,1),20);

L = place(A',C',P)'.

The frequency selected is chosen based on what frequency content (form a Fourier perspective) the system and signal should have. Information at frequencies below this value are considered physically meaningful, information at frequencies above this value are considered noise. If you resulting filter is too-slow increase the cutoff frequency, if it is too noisy decrease the cutoff frequency.

Discrete Time State-Space Systems

Discrete time systems describe how a system state evolves at one time stamp to the next, that is

$$\begin{aligned}\mathbf{x}_{k+1} &= A_d\mathbf{x}_k + B_d\mathbf{u} \\ \mathbf{y}_k &= C\mathbf{x}_k + D\mathbf{u},\end{aligned}$$

instead of discussing the rate of change of states, we discuss what the next state we'll measure is. Mathematically, this is the same form as the continuous time case. The only difference is that the eigenvalues need to be in the range $(-1, 1)$ for the system to be stable. To find the control gains K or observer gains L we use the same acker and place commands as above. To convert a continuous time system to a discrete time system we do the following

$$\begin{aligned}A_d &= e^{A\Delta T} \\ B_d &= A^{-1}\left(A_d - I\right)B \\ C_d &= C \\ D_d &= D,\end{aligned}$$

where $e^{A\Delta T}$ is a matrix exponential (in Matlab use the command expm not exp). This approach can break down if A is not invertible. A more robust method is to calculate A_d and B_d simultaneously:

$$\begin{bmatrix}A_d & B_d \\ 0 & \mathbf{1}\end{bmatrix} = e^{\begin{bmatrix}A & B \\ 0 & 0\end{bmatrix}\Delta t}$$

which works by implicitly assuming a zero-order-hold on the control input $\mathbf{u}$, that is $\mathbf{u}$ is held constant between updates. This is my preferred method for discretization. Note that this is directly related to the z-transform discussed earlier. Just as the Laplace space transform can be recovered by

$$G\{s\} = C\left(sI - A\right)^{-1}B + D$$

the z-transform can be recovered by

$$G\{z\} = C\left(zI - A_d\right)^{-1}B_d + D,$$

either of which can be calculated efficiently using a symbolic mathematics package (Matlab has one, check it out!).

To convert poles, just multiply by the update period and take the exponential

$$P_{d,i} = e^{P_{c,i}\Delta T},$$

where $P_{c,i}$ is the ith desired continuous time pole and the exponential is the standard version. **Design the system dynamics in continuous time, convert it to discrete time, calculate the gains, and implement.**

Calculating gains for Discrete Observers: One difference from the continuous time case is caused by sampling. The observer is trying to estimate the current state x_k based on the last state estimate $\tilde{x}_{k-1}$ the last control $\tilde{u}_{k-1}$and the current measurement y_k. Thus, we first need to estimate our current state based on the system dynamics, then compare that to our measurement, to correct our current estimate. That is:

$$\begin{aligned}\tilde{\mathbf{x}}^{\dagger}_{k+1} &= A_d\tilde{\mathbf{x}}_k + B_d\mathbf{u}_k \quad \textit{a priori estimate} \\ \tilde{\mathbf{x}}_{k+1} &= \tilde{\mathbf{x}}^{\dagger}_{k+1} + L\left(y_{k+1} - C\tilde{\mathbf{x}}^{\dagger}_{k+1}\right) \quad \textit{a posteriori estimate}\end{aligned}$$

Combining the two steps we find

$$\tilde{\mathbf{x}}_{k+1} = \left(A_d - LCA_d\right)\tilde{\mathbf{x}}_k + \left(I - LC\right)B_d\mathbf{u}_k + Ly_{k+1}$$

Looking at the special case of no input u and a measurement at the origin $y = 0$, the dynamic system is

$$\tilde{\mathbf{x}}_{k+1} = (A_d - LCA_d)\,\tilde{\mathbf{x}}_k$$

Thus we need to use

$$L = \text{place}\left(A_d^\top,\, (C_d A_d)^\top,\, P_d\right)^\top$$

for the discrete system.

Combined Z-Transform Implementation of the Controller-Observer System:

The control law, then has the form

$$\begin{aligned}
\mathbf{u}_{k+1} &= K\mathbf{e}_{k+1} \\
&= K\left(\mathbf{r}_{k+1} - \tilde{\mathbf{x}}_{x+1}\right) \\
&= K\left(\mathbf{r}_{k+1} - (A_d - LCA_d)\,\tilde{\mathbf{x}}_k - (I - LC)\,B_d\mathbf{u}_k - Ly_{k+1}\right)
\end{aligned}$$

which can be expressed with y as an input control and an augmented state $\tilde{\mathbf{x}}$ and $\mathbf{u}$

$$\begin{bmatrix}\tilde{\mathbf{x}}_k \\ \mathbf{u}_k\end{bmatrix} = \begin{bmatrix}(A_d - LCA_d) & (I - LC)\,B_d \\ K\,(A_d - LCA_d) & -K\,(I - LC)\,B_d\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{x}}_{k-1} \\ \mathbf{u}_{k-1}\end{bmatrix} + \begin{bmatrix}\mathbb{O} & L \\ K & -KL\end{bmatrix}\begin{bmatrix}\mathbf{r}_k \\ y_k\end{bmatrix}$$

which can be turned into a z-transform of the resulting controller design according to

$$\mathbf{u}\{z\} = \begin{bmatrix}\mathbb{O} & \mathbb{I}\end{bmatrix}\left(\mathbb{I} - z^{-1}\begin{bmatrix}(A_d - LCA_d) & (I - LC)\,B_d \\ K\,(A_d - LCA_d) & -K\,(I - LC)\,B_d\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbb{O} & L \\ K & -KL\end{bmatrix},$$

which will have a z-transform for $\mathbf{u}$ that is a function of both the desired states $\mathbf{r}$ and the measurements $\mathbf{y}$, which can be collected into a single larger z-transform for the full system with multiple inputs (desired states r and measurements y) and one output u.

Discrete Controller & Observer Implementation:

This need not be implemented through a z-transform. A controller can be directly implemented in a micro-controller using the matrix form:

$$\begin{aligned}
\tilde{\mathbf{x}}^\dagger_{k+1} &= A_d\tilde{\mathbf{x}}_k + B_d\mathbf{u}_k \quad \textit{a priori estimate} \\
\tilde{\mathbf{x}}_{k+1} &= \tilde{\mathbf{x}}^\dagger_{k+1} + L\left(y_{k+1} - C\tilde{\mathbf{x}}^\dagger_{k+1}\right) \quad \textit{a posteriori estimate} \\
&= (A_d - LCA_d)\,\tilde{\mathbf{x}}_k + (I - LC)\,B_d + Ly_{k+1} \quad \textit{equilvalent} \\
&= A_{d,o}\tilde{\mathbf{x}}_k + B_{d,o}\mathbf{u}_k + Ly_{k+1} \quad \textit{equilvalent simplified} \\
\mathbf{u}_{k+1} &= K\left(\mathbf{x}_{k+1,des} - \tilde{\mathbf{x}}_{k+1}\right) + \mathbf{u_{ss}} \\
\mathbf{u}_{k+1} &= K\left(\mathbf{x}_{k+1,des} - \tilde{\mathbf{x}}_{k+1}\right) + B_{ss}\left(\dot{\mathbf{x}}_{k+1,des} - A\mathbf{x}_{k+1,des}\right).
\end{aligned}$$

Note: $\mathbf{u}_{ss}$and B_{ss} are calculated with the continuous time matrices (A not A_d and B not B_d), states, and desired state derivatives.

Given this, the effective closed loop system with a constant reference becomes

$$\begin{bmatrix}\mathbf{x} \\ \tilde{\mathbf{x}}\end{bmatrix}_{k+1} = \begin{bmatrix}A_d & -B_dK \\ LCA_d & (A_d - B_dK - LC\,(A_d - B_dK))\end{bmatrix}\begin{bmatrix}\mathbf{x} \\ \tilde{\mathbf{x}}\end{bmatrix}_k + \begin{bmatrix}B_d(K - (B^\top B)^{-1}B^\top A) \\ B_d(K - (B^\top B)^{-1}B^\top A)\end{bmatrix}\mathbf{x}_{k,des}$$

$$y_k = \begin{bmatrix}C & -DB_dK\end{bmatrix}\begin{bmatrix}\mathbf{x} \\ \tilde{\mathbf{x}}\end{bmatrix}_k + DB_d(K - (B^\top B)^{-1}B^\top A)\mathbf{x}_{k,des}$$

Discrete Time Integrator

For signals with little change, that is $\dot{\mathbf{x}}_{k+1,des} = 0$ nominally, we can use an integrator to remove steady state error by estimating $\mathbf{u_{ss}}$without needing to calculate it directly. To calculate the control system, we augment the sate with the error integration, that is

$$A_{d,i} \Rightarrow \begin{bmatrix} 1 & C \\ 0 & A_d \end{bmatrix}$$
$$B_{d,i} \Rightarrow \begin{bmatrix} 0 \\ B_d \end{bmatrix}.$$

Note that the augmented system is controllable, but not observable. *Nota Bene:* The augmented system is unobservable because the compounding error does not affect the actual measured dynamics. That is, any amount of compounded error won't change the current position or velocity is of the actual system (until we act on it) simply because its not a real state (no energy storage). This is not a problem, because the integration is of our own design. We'll keep track of the compounding error ourselves, and if the value is not exactly correct, it does not really matter—it's just a book keeping method to get rid of steady-state error anyway.

Given this augmented system we can calculate the controller gain K the same way as before

$$K = place(A_{d,i}, B_{d,i}, P_d).$$

If we add the additional pole on the real axis faster 5x faster than our previous system, the dynamics don't appreciably change, but the steady state error is eliminated. The observer does not change, the integration is done simply by adding up the residual error each time step. That is, implementation becomes

$$\begin{aligned}
\tilde{\mathbf{x}}^{\dagger}_{k+1} &= A_d\tilde{\mathbf{x}}_k + B_d\mathbf{u}_k \quad \textit{a priori estimate} \\
\tilde{\mathbf{x}}_{k+1} &= \tilde{\mathbf{x}}^{\dagger}_{k+1} + L\left(y_{k+1} - C\tilde{\mathbf{x}}^{\dagger}_{k+1}\right) \quad \textit{a posteriori estimate} \\
e_{int} &= e_{int} + C\left(\mathbf{x}_{k+1,des} - \tilde{\mathbf{x}}_{k+1}\right) \quad \textit{integrator update} \\
\mathbf{u}_{k+1} &= K\left(\begin{bmatrix} 0 \\ \mathbf{x}_{k+1,des} \end{bmatrix} - \begin{bmatrix} e_{int} \\ \tilde{\mathbf{x}}_{k+1} \end{bmatrix}\right) + \mathbf{u}_{ss},
\end{aligned}$$

note that we no-longer need to add in $\mathbf{u}_{ss}$ explicitly because the integrator will adjust for that automatically, but if we do the integrator has less work to do and the performance will be better.

Examples

Given the state space system:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -20 & -7 & -3 \end{bmatrix}$$
$$B = \begin{bmatrix} 0 \\ 0 \\ 23 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$
$$D = 0$$

The eigenvalues, step response, and frequency response of the open-loop system are:

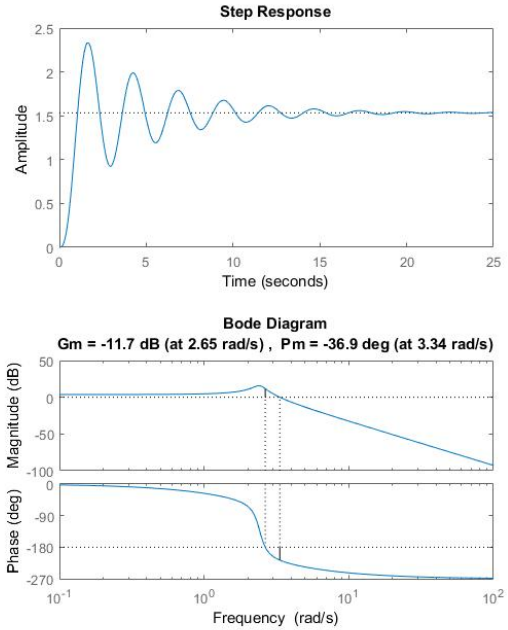
```
A = [0 1 0; 0 0 1; -15 -7 -3;];
B = [0;0;23];
C = [1 0 0];
D = 0;
ol_poles = eig(A)
ol_sys = ss(A,B,C,D);
ol_step_info = stepinfo(ol_sys)
figure(1)
subplot(2,1,1)
step(ol_sys)
subplot(2,1,2)
margin(ol_sys)
```

Algorithm 3: Matlab Code for open-loop system response

$$\text{ol\_poles} = \begin{bmatrix} -2.5568 + 0.0000i \\ -0.2216 + 2.4120i \\ -0.2216 - 2.4120i \end{bmatrix}$$

ol_step_info =

RiseTime: 0.6035
SettlingTime: 16.1477
SettlingMin: 0.9244
SettlingMax: 2.3335
Overshoot: 52.1828
Undershoot: 0
Peak: 2.3335
PeakTime: 1.6571



From these plots we can see that the system has a lot of ringing in the step response and does not go to a value of 1, moreover there is a resonance in the frequency sweep, which may be undesirable.

Continuous Time Controller

Lets say we'd like to have:

Rise Time 0.5 seconds

Overshoot 3%

Steady-state value 1

Using (8) above:

$$\zeta = \sqrt{\frac{\ln(0.03)^2}{\pi^2 + \ln(0.03)^2}} = 0.7448$$

using (10) above:

$$\omega_n = \frac{1.53 + 2.31 \cdot 0.7448^2}{0.5} = 5.623^{\text{rad}/\text{s}}$$

Thus we should place two poles at:$P = -5.623\left(0.7448 \pm i\sqrt{1 - 0.7448^2}\right)$. The third pole should be faster, to keep it looking like a second order response, lets place it at $\sim -5\zeta\omega_n$. Thus:

$$\text{cl\_poles} = \begin{matrix} -20.9396 \\ -4.1879 - 3.7520i \\ -4.1879 + 3.7520i \end{matrix}$$

given this, we can calculate the gain matrix K

$$\begin{aligned} K &= \text{place}(A, B, \text{cl\_poles}) \\ &= \begin{bmatrix} 49.0165 & 13.9061 & 1.44547 \end{bmatrix} \end{aligned}$$

The steady state input becomes:

$$\begin{aligned} \mathbf{u}_{ss} &= -\left(B^\top B\right)^{-1} B^\top A \mathbf{x}_d \\ &= -\begin{bmatrix} -0.6522 & -0.3043 & -0.1304 \end{bmatrix} \mathbf{x}_d \\ &= 0.6522 \quad \text{for a unit step} \end{aligned}$$

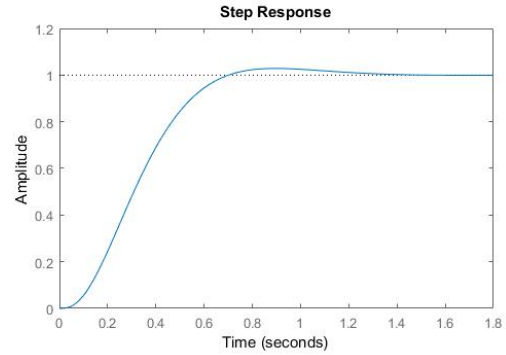
zeta = 0.7448

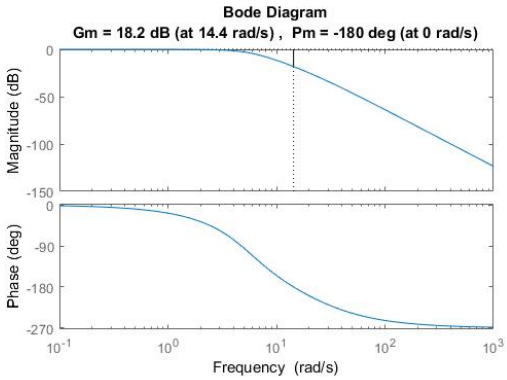
wn = 5.6229

cl_poles = -20.9396 + 0.0000i
-4.1879 - 3.7520i
-4.1879 + 3.7520i

K = [28.1322 8.6958 1.1442]

cl_step_info =
RiseTime: 0.4193
SettlingTime: 1.0747
SettlingMin: 0.9016
SettlingMax: 1.0286
Overshoot: 2.8602
Undershoot: 0
Peak: 1.0286





Discrete Time Controller

Assuming this will be implemented in a discrete time system with a 0.05 second update rate we can repeat the above:

```
A = [0 1 0; 0 0 1; -15 -7 -3;];
B = [0;0;23];
C = [1 0 0];
D = 0;
%% Continuous Time Design
zeta = sqrt(log(0.03)^2/(pi^2+log(0.03)^2))
wn = (1.53+2.31*zeta^2)/0.5
cl_poles = [-5*wn*zeta;-wn*(zeta+[1;-1]*sqrt(1-zeta^2)*1j)]
K = place(A,B,cl_poles)
% Closed Loop System Model
A_cl = A-B*K;
B_cl = B*(K-(B'*B)^-1*B'*A)*[1;0;0]; % the [1;0;0] reflects a unit step in state 1
cl_sys = ss(A_cl,B_cl,C,D);
cl_step_info = stepinfo(cl_sys)
figure(1)
subplot(2,1,1)
step(cl_sys)
subplot(2,1,2)
margin(cl_sys)
```

Algorithm 4: Matlab Code for closed-loop system response

```
A = [0 1 0; 0 0 1; -15 -7 -3;];
B = [0;0;23];
C = [1 0 0];
D = 0;
%% Continuous Time Design
zeta = sqrt(log(0.03)^2/(pi^2+log(0.03)^2))
wn = (1.53+2.31*zeta^2)/0.5
cl_poles = [-5*wn*zeta;-wn*(zeta+[1;-1]*sqrt(1-zeta^2)*1j)]
%% Discrete Time Design
dt = 0.05;
Ad = expm(A*dt)
Bd = A^-1*(Ad-eye(size(A)))*B
cl_poles_disc = exp(cl_poles*dt)
Kd = place(Ad,Bd,cl_poles_disc)
%% Closed Loop System Model
Ad_cl = Ad-Bd*Kd;
Bd_cl = Bd*(Kd-(B'*B)^-1*B'*A)*[1;0;0]; % the [1;0;0] reflects a unit step in state 1
cl_sys = ss(Ad_cl,Bd_cl,C,D,dt);
cl_step_info = stepinfo(cl_sys)
figure(1)
subplot(2,1,1)
step(cl_sys)
subplot(2,1,2)
margin(cl_sys)
```

Algorithm 5: Matlab Code for closed-loop system response with a discrete system

zeta = 0.7448
wn = 5.6229
cl_poles = -20.9396 + 0.0000i
-4.1879 - 3.7520i
-4.1879 + 3.7520i

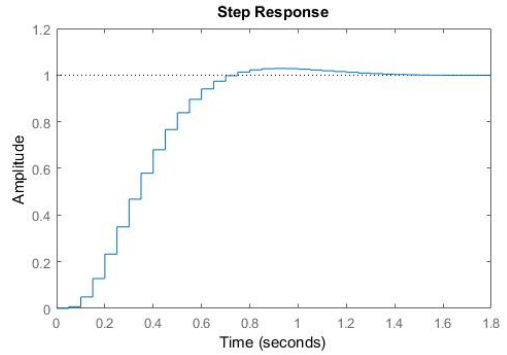
Ad =
0.9997 0.0499 0.0012
-0.0178 0.9914 0.0463
-0.6944 -0.3419 0.8525

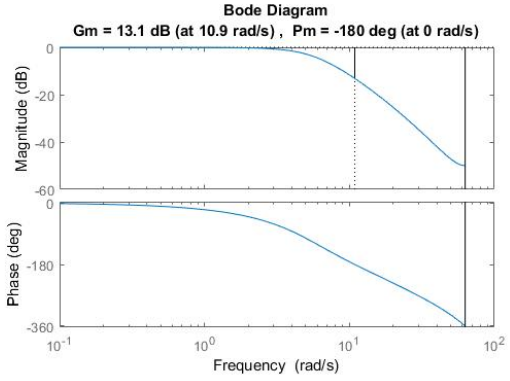
Bd =
0.0005
0.0273
1.0647

cl_poles_disc =
0.3510 + 0.0000i
0.7968 - 0.1513i
0.7968 + 0.1513i

Kd = [14.9663 5.0400 0.7084]

cl_step_info =
RiseTime: 0.4500
SettlingTime: 1.1000
SettlingMin: 0.9409
SettlingMax: 1.0286
Overshoot: 2.8610
Undershoot: 0
Peak: 1.0286
PeakTime: 0.9000





Observer Design

The system is being sampled at $T_s = 0.05$ seconds, this makes the Nyquist frequency: $f_{ny} = \frac{\pi}{T_s} = 62.8^{\text{rad}}/_{\text{s}}$. From the above frequency response we know the controller cannot follow any input signal faster than around 2 rad/s, so a observer functioning at a 10rad/s cutoff would be faster than our system and less than the Nyquist frequency. So, we'll design the observer to act like a Butterworth filter with a 10rad/s cutoff frequency.

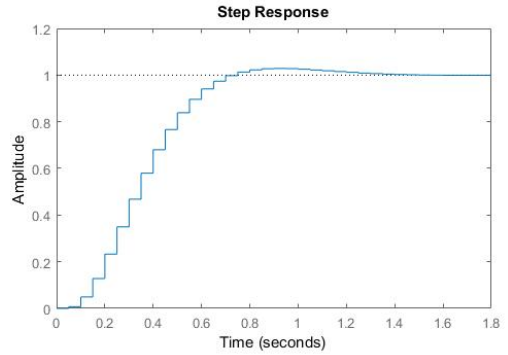
```
A = [0 1 0; 0 0 1; -15 -7 -3;];
B = [0;0;23];
C = [1 0 0];
D = 0;
%% Continuous Time Design
zeta = sqrt(log(0.03)^2/(pi^2+log(0.03)^2));
wn = (1.53+2.31*zeta^2)/0.5;
cl_poles = [-5*wn*zeta;-wn*(zeta+[1;-1]*sqrt(1-zeta^2)*1j)];
%% Discrete Time Design
dt = 0.05;
Ad = expm(A*dt);
Bd = A^-1*(Ad-eye(size(A)))*B;
cl_poles_disc = exp(cl_poles*dt);
Kd = place(Ad,Bd,cl_poles_disc);
%% Observer Design
w_filter = 10; % 10 rad/s cut-off frequency
[~,P,~] = butter(size(A,1),w_filter,'s');
obs_poles = exp(P*dt)
Ld = place(Ad',(C*Ad)',obs_poles)'
%% System Model
Ad_cl_obs = [Ad -Bd*Kd; Ld*C*Ad Ad-Bd*Kd-Ld*C*(Ad-Bd*Kd)];
Bd_cl = Bd*(Kd-(B'*B)^-1*B'*A)*[1;0;0]; % the [1;0;0] reflects a unit step in state 1
Bd_cl_obs = [Bd_cl;Bd_cl];
cl_sys_obs = ss(Ad_cl_obs,Bd_cl_obs,[C zeros(size(C))],D,dt);
cl_step_info = stepinfo(cl_sys_obs)
figure(1)
subplot(2,1,1)
step(cl_sys_obs)
subplot(2,1,2)
margin(cl_sys_obs)
```

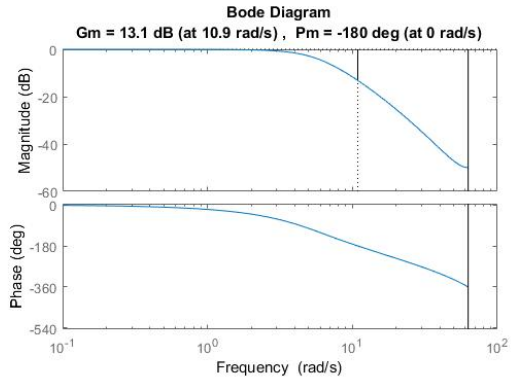
Algorithm 6: Matlab Code for discrete observer design

obs_poles =
0.7069 + 0.3268i
0.7069 - 0.3268i
0.6065 + 0.0000i

Ld =
0.5726
4.6947
14.0926

cl_step_info =
RiseTime: 0.4500
SettlingTime: 1.1000
SettlingMin: 0.9313
SettlingMax: 1.0251
Overshoot: 3.0729
Undershoot: 0
Peak: 1.0251
PeakTime: 0.9000





Filter Implementation

Suggested Exercise: Redo the above examples but use a Bessel filter to pick the poles. For the controller start with a $\omega_n = {}^{2\pi}/_{T_r}$to calculate the poles. For the observers start with these same pole locations. How does the performance compare to the above plots?

Initialize
x_est ⇐ 3x1 float zeros (persistent quantity)
u_last ⇐ 1x1 float zeros (persistent quantity)
Ad ⇐ 3x3 float from Matlab output (constant quantity)
Bd ⇐ 3x1 float from Matlab output (constant quantity)
C ⇐ 1x3 float from Matlab output (constant quantity)
K ⇐ 1x3 float from Matlab output (constant quantity)
L ⇐ 3x1 float from Matlab output (constant quantity)
B_ss ⇐ 1x3 float from Matlab output (constant quantity, i.e., -(B'*B)^-1B'*A)
Every 0.05 seconds DO:
Observer Portion
y ⇐ take measurement
x_est ⇐ *matrixMultiply*(Ad,x_est) + *matrixMultiply*(Bd,u_last)
y_est ⇐ *matrixMultiply*(C,x_est)
meas_err ⇐ y-y_est
x_est ⇐ x_est + *matrixMultiply*(L,meas_err)
Controller Portion
xd ⇐ get desired state
state_error ⇐xd-x_est
u_dyn ⇐ *matrixMultiply*(K,state_error)
u_ss ⇐ *matrixMultiply*(B_ss,xd)
u_last ⇐ u_dyn+u_ss
Control Application
applyControl(u_last)
End DO

Algorithm 7: Pseudo Code for Controller Implementation