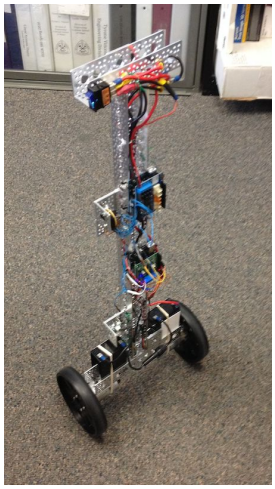
EENG498BC: Systems Exploration, Engineering, and Design Laboratory

Spring 2016

Balancing a Pendulum

Vibhuti Dave and Tyrone Vincent*

In the last document, we discussed a model for a balancing robot.



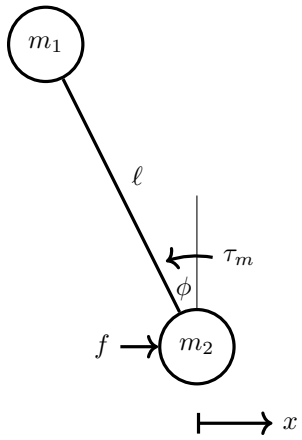


Figure 1: Idealized component diagram for pendulum robot

$$\begin{aligned}
\ddot{\phi} &= \frac{(g(m_1+m_2) - m_1\ell\cos\phi(\dot{\phi})^2)\sin\phi - \cos\phi b_x\dot{x} - \frac{(m_1+m_2)}{m_1\ell}b_\phi\dot{\phi} + \left(\frac{(m_1+m_2)}{m_1\ell} + \frac{\cos\phi}{r}\right)\tau_m}{\ell\left(m_2 + m_1\sin^2\phi\right)} \\
\ddot{x} &= \frac{gm_1\cos\phi\sin\phi - m_1\ell\sin\phi(\dot{\phi})^2 - b_x\dot{x} - \frac{\cos\phi}{\ell}b_\phi\dot{\phi} + \left(\frac{\cos\phi}{\ell} + \frac{1}{r}\right)\tau_m}{(m_2 + m_1\sin^2\phi)}
\end{aligned} \tag{1}$$

At this point, you should have done experiments to find the parameters for your robot.

We wish to design a control system that will stabilize the pendulum around $\phi = 0$. The nonlinear model is fairly complex to use for design. However, since the point of the control system is to regulate the system to stay near a specific operating point, we can use linear approximations (i.e. first order Taylor series) of the sine and cosine functions around this point. For $\phi \approx 0$ and $\dot{\phi} \approx 0$ this is $\sin(\phi) \approx \phi$, $\sin^2(\phi) \approx 0$ and $\cos(\phi) \approx 1$. In order to keep the model simple, we will also assume that the damping terms b_x and b_ϕ are small enough to be set to zero. (They can actually easily be added, but the resulting controller won't change that much, and it will make things a little simpler for you. Note that we will always go back an verify that the controller works using the full, nonlinear system model)

*Developed and edited by Tyrone Vincent and Vibhuti Dave

This gives the linearized dynamics

$$\ddot{\phi} = a\phi + c\tau_m$$
$$\ddot{x} = b\phi + d\tau_m$$

where

$$a = \frac{g(m_1 + m_2)}{\ell m_2}$$
$$c = \frac{1}{\ell m_2}\left(\frac{(m_1 + m_2)}{m_1 \ell} + \frac{1}{r}\right)$$
$$b = \frac{m_1}{\ell m_2}$$
$$d = \frac{1}{m_2}\left(\frac{1}{\ell} + \frac{1}{r}\right)$$

Taking the Laplace Transforms of both sides,

$$s^2\phi(s) = a\phi(s) + c\tau_m(s)$$
$$s^2 X(s) = b\phi(s) + d\tau_m(s)$$

We can solve these equations for the two transfer functions $\frac{\phi(s)}{\tau_m(s)}$ and $\frac{X(s)}{\tau_m(s)}$. This can easily done with $b \neq 0$, but for our robots, since we will be controlling the robot to be upright, it will turn out that the term $b\phi(s)$ will be small. Thus, we take the following as the transfer functions. (If you use Matlab, it isn't really a big deal to keep $b \neq 0$, and you can try the design both ways to see what effect there is)

$$\frac{\phi(s)}{\tau_m(s)} = \frac{c}{s^2 - a}$$
$$\frac{X(s)}{\tau_m(s)} = \frac{d}{s^2}$$

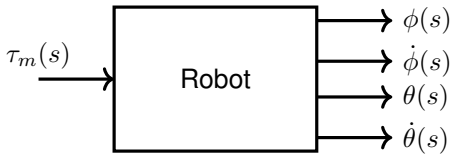
Only one problem: we can't directly measure ϕ with a gyroscope, we can only measure $\dot{\phi}$. However, consider the following trick: instead of stabilizing ϕ at 0, what if we try and stabilize $\dot{\phi}$ at 0? Generally, controlling $\dot{\phi} = 0$ would ensure ϕ is a constant, but we would not necessarily know what that constant is. However, for this system, because of gravity there is no way that we can achieve $\dot{\phi} = 0$ *without* being at one of the equilibrium points of $\phi = 0$ or $\phi = \pi$. To consider $\dot{\phi}$ as the output of the system, we will multiply our transfer function by s:

$$\frac{\dot{\phi}(s)}{\tau(s)} = \frac{cs}{s^2 - a}.$$

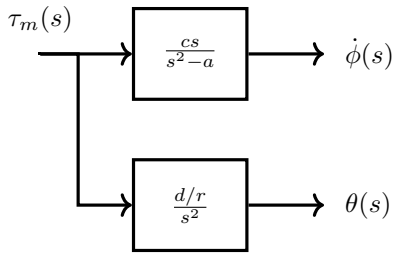
We can also scale x by r to recover the wheel rotation θ

$$\frac{\theta(s)}{\tau(s)} = \frac{d/r}{s^2}.$$

We now have two different models that we can use. One is the more accurate, nonlinear, model given by the Simulink subsystem



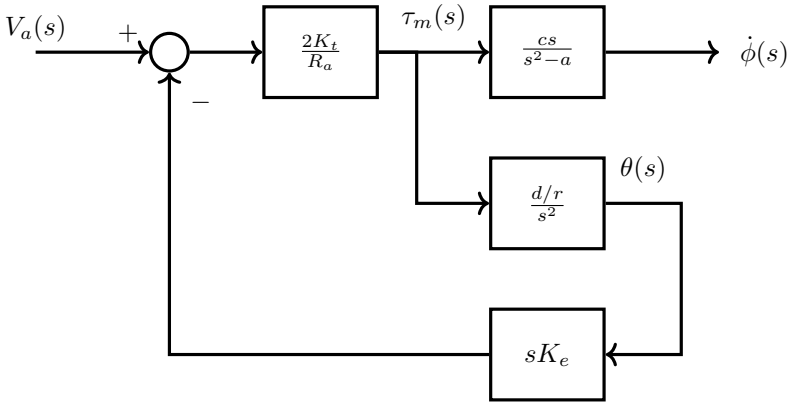
The other is an approximate, linear model that only keeps track of $\dot{\phi}$ and $\theta(s)$



This linear model can be used if the following hold:

- ϕ is small
- $\dot{\phi}$ is small
- the damping terms are negligible.

You can use the simpler model for designing the feedback control system. First, let's add the other components to the robot, starting with the motor. In the following block diagram, we have (again, for simplicity) assumed the motor inductance $L_a = 0$. The motor voltage is $V_a(s)$.



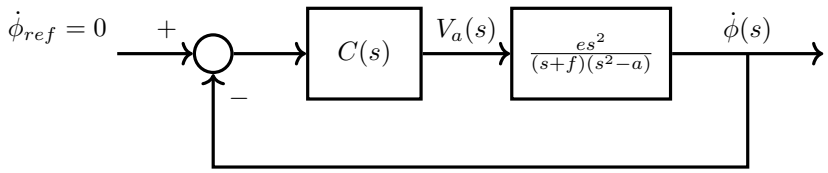
Simplifying the block diagram, we get the following transfer function with motor voltage as the input:

$$\begin{aligned}\frac{\dot{\phi}(s)}{V_a(s)} &= \frac{\frac{2K_t}{R_a}}{1+\frac{2K_tK_ed}{R_ars}}\frac{cs}{s^2-a} \\ &= \frac{es^2}{(s+f)(s^2-a)}\end{aligned}$$

where

$$\begin{aligned}e &= \frac{2K_tc}{R_a} \\ f &= \frac{2K_tK_ed}{R_ar}\end{aligned}$$

In the next section, we will investigate the design of a controller $C(s)$ that operates within the following unity-gain feedback control system to stabilize $\dot{\phi}$.



1. Pendulum Control

The controller that we will design, $C(s)$, will provide voltage command, v_a, which is calculated based on measurements of the angular velocity $\dot{\phi}$, compared to the desired reference, 0 rad/s. Since the system under control is unstable, the control design is somewhat different than the standard methods in EENG307. However, we can use the root locus to help us with our design.

- Calculate the parameters c, f and a for your robot.
- Check that proportional control will not work by plotting the root locus of $\frac{es^2}{(s+f)(s^2-a)}$. Verify that there is an unstable closed loop pole for any value of K when $C(s) = K$.
- Now, try a controller of the form

$$C(s) = K\frac{(s+\alpha)(s+\beta)}{(s+\gamma)(s-\delta)}$$

  Where α, β, γ and δ are positive constants to be found. Note that the term $(s-\delta)$ is not a typo - we are going to include an unstable pole with this controller. Choose $\alpha = \sqrt{a}$, $\beta \approx 1$, γ a small value, and δ a little smaller than $\sqrt{a}$, and plot the root locus of $\frac{(s+\alpha)(s+\beta)}{(s+\gamma)(s-\delta)}\frac{es^2}{(s+f)(s^2-a)}$ (don't forget the negative sign out front). Vary the values of α, β, γ, and δ to see the effect of each. You will probably need to zoom in to the origin on your root locus to see the interesting part of the plot.
- From the root locus, choose a value of K so that the closed loop poles will be well damped. Plot the impulse response (this is what you would see if the pendulum was bumped). You may want to re-adjust K, and/or α, β, γ or δ depending on what you see in the impulse response.
- Using Simulink, simulate the closed loop behavior with your controller applied to the nonlinear model. Some discussion of this is in the next section.

2. Signs and scaling

When implementing your controller, you will need to be careful about the signs of the actuator and sensor signals, and the scaling associated with them. For the design above, we have assumed that for a positive motor current, the clockwise torque will act on the wheels, creating a counter clockwise torque on the pendulum. Also, we have assumed that the motion $\dot{\phi}$ is measured with counter-clockwise motion as positive. Depending on how you mounted and configured your motors and gyro, this convention may or may not hold. You will want to verify that a positive motor voltage will tend to cause a positive angular velocity. If not, change the sign of either the gyro or the motor voltage (but not both!)

Another thing to watch out for is the required scaling in order to implement your controller. Note that the controller designed above outputs a signal in units of Volts and has an input with units of rad/s. However, in the Arduino, you will create a output voltage (or really, PWM signals) by choosing a specific integer, and likewise the gyro reading will be an integer representing some voltage and will definitely not have units of rad/s. Make sure you do the right conversions.

3. Control implementation on a micro-controller

When implementing the PID controller we used the discrete approximation for the integral and derivative. The derivative approximation was

$$e(t) = \frac{dy(t)}{dt} \approx \frac{y(t) - y(t-T_s)}{T_s},$$

where T_s is the sample time. Using discrete time signal notation this becomes

$$e[k] = \frac{y[k] - y[k-1]}{T_s},$$

and solving for $y[k]$ gives us

$$y[k] = y[k-1] + T_s e[k].$$

We can use this same derivative approximation to convert an arbitrary first order differential equation to a recursive equation. For example, if we have a desired controller of the form

$$\frac{Y(s)}{E(s)} = K\frac{as+1}{bs+1}$$

where a and b are constants, then

$$\begin{aligned}(bs+1)Y(s) &= K(as+1)E(s),\\ bsY(s) + 1Y(s) &= KasE(s) + KE(s),\end{aligned}$$

and finding the inverse Laplace transform gives us

$$b\frac{d}{dt}y(t) + y(t) = Ka\frac{d}{dt}e(t) + Ke(t).$$

Using the approximation $\frac{dy(t)}{dt} \approx \frac{y(t)-y(t-T_s)}{T_s}$ and $\frac{de(t)}{dt} \approx \frac{e(t)-e(t-T_s)}{T_s}$, and converting to discrete time signals results in

$$b\frac{y[k] - y[k-1]}{T_s} + y[k] = Ka\frac{e[k] - e[k-1]}{T_s} + Ke[k].$$

We can solve this equation for $y[k]$ to give our final recursive formula

$$y[k] = \frac{1}{b+T_s}\left(by[k-1] + K(a+T_s)e[k] - Kae[k-1]\right).$$

This can be extended to higher order transfer functions by introducing intermediate variables. For example, suppose we have the controller

$$C(s) = \frac{U(s)}{E(s)} = K\frac{(as+1)(bs+1)}{(cs+1)(ds+1)}$$

We want to calculate the signal u, based on measurements of e. We can introduce the intermediate variable v, defined via

$$\frac{V(s)}{E(s)} = K\frac{(as+1)}{(bs+1)}.$$

Then we have

$$\frac{U(s)}{V(s)} = \frac{(cs+1)}{(ds+1)}.$$

The controller can be implemented by calculating v from e, and then calculating u from v.

4. Anti-windup for unstable terms

For the integral term in the PID control, we implemented an anti-windup scheme that re-calculated the integral term when actuator saturation occured. Anti-windup is necessary for any unstable term in a controller, for example if either $b < 0$ or $d < 0$. It is most convenient if the unstable term is the last one, so let's suppose that $d < 0$. The recursive formula would be

$$u[k] = \frac{1}{d+T_s}\left(du[k-1] + K(c+T_s)v[k] - Kcv[k-1]\right).$$

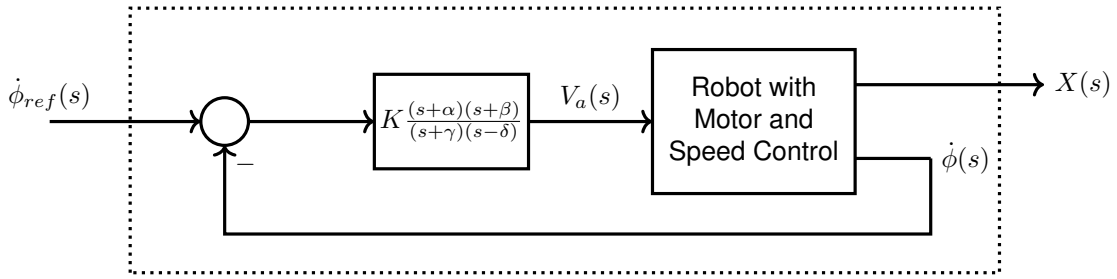
Suppose the actuator is saturated when $|u| = u_{max}$. In this case, if we find that u calculated using the controller had magnitude greater than u_{max}, we set it so that its magnitude is exactly u_{max}. The code would look something like this:

```
⋮
u = (1/(d+Ts))*(d*u_past + K*(c+Ts)*v - K*c*v_past);
if (abs(u)>umax) { u=sgn(u)*umax; }
u_past = u;
v_past = v;
⋮
```

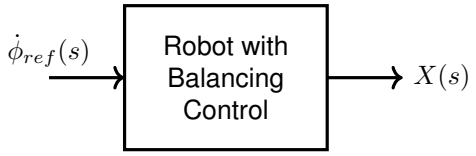
where `sgn` is the signum function that returns 1 if the argument is positive and -1 if it is negative. This is not a native function for the Arduino, but is easily implemented.

5. Further Hierarchical Control Loops

The controller designed above can be applied to the nonlinear model.



Via simulation, you will find that with $\dot{\phi}_{ref} = 0$, this controller will successfully balance the robot, in that $\dot{\phi} = 0$ and ϕ will also be close to 0, even if the initial condition $\phi(0) \neq 0$. However, you will also find that the controller will cause x to vary quite a bit. In order to have a functional robot, we need to also control the horizontal position. However, now we have yet another level of encapsulation:



Can you design a controller that will regulate x around a desired value, using $\dot{\phi}$ as an actuator?