PID Control

Nathan Sprague

Example System...

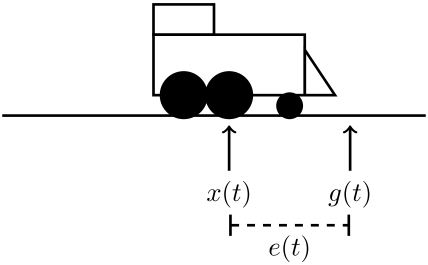


Figure 1: Train

- Goal: move the locomotive to the goal location
- $e(t)$ is the error at time t

$$e(t) = g(t) - x(t)$$

Proportional Control

- Controller function:

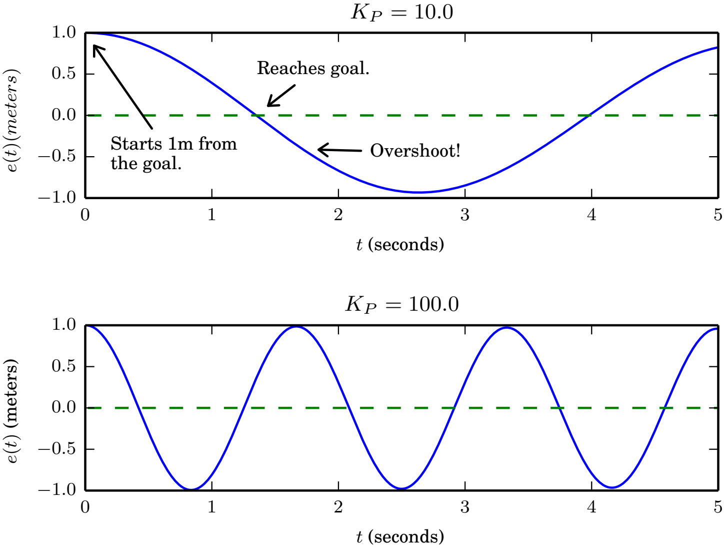
$$u(t) = K_p e(t)$$

- $u(t)$ - control signal at time t
- K_p - gain

Python pseudocode:

```python
def p_controller(train, g, K_P):
    while True:
        e = g - train.x
        u = K_P * e
        train.throttle(u)
```

(Disappointing) Result:



Problems

- Real systems have:
  - momentum
  - friction/drag
  - outside forces (e.g. gravity)

PD Control

- Fix: Incorporate rate of change in the error:
  - If error is going down quickly, ease off on the control signal
  - If error is going up quickly, increase the control signal

derivative term

$$u(t) = K_p e(t) + \boxed{K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}}$$

In practice, we will approximate the derivative...

$$\frac{\mathrm{d}e(t)}{\mathrm{d}t} \approx \frac{e(t) - e(t - \Delta t)}{\Delta t}$$

Python pseudocode:

```python
def pd_controller(train, g, K_P, K_D):
    e_prev = g - train.x
    while True:
        e = g - train.x
        dedt = (e - e_prev) / train.dt
        u = K_P * e + K_D * dedt
        train.throttle(u)
        e_prev = e
```

PD Result

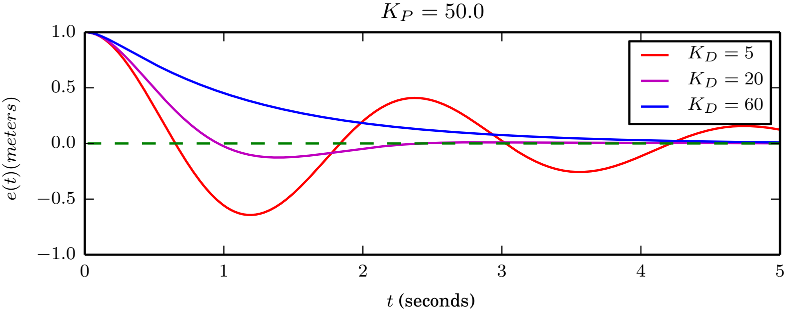


Figure 3: PD controller result

What about this situation?

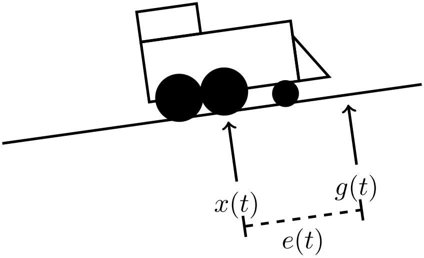


Figure 4: Train on a Hill

What about this situation?

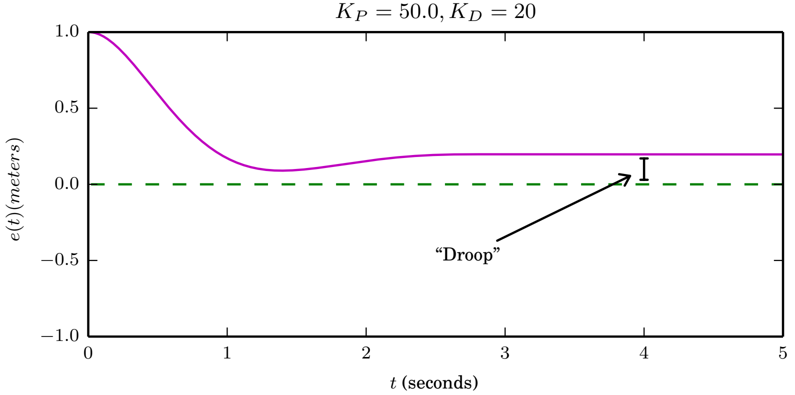


Figure 5: Droop

PID Control

- Next idea: Add a term that is proportional to the amount of error seen in the past
  - If there has been a lot of error in the past, increase control signal

integral term

$$u(t) = K_P e(t) + \boxed{K_I \int_0^t e(\tau) d\tau} + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

Approximating Integral

In practice, we will approximate the integral as a summation...

\# of steps before time t

$$\int_0^t e(\tau) d\tau \approx \sum_{i=0}^{t/\Delta t} e(i\Delta t) \Delta t$$

Error at time step i

PID in Python

```python
def pid_controller(train, g, K_P, K_I, K_D):

    e_prev = g - train.x
    e_sum = 0              # accumulator for integral term
    while True:
        e = g - train.x
        e_sum = e_sum + e * train.dt
        dedt = (e - e_prev) / train.dt
        u = K_P * e   +   K_I * e_sum   +   K_D * dedt
        train.throttle(u)
        e_prev = e
```

PID Result

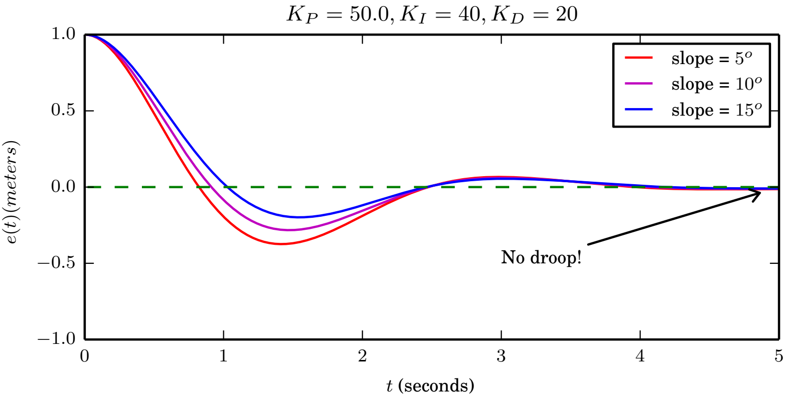


Figure 6: Good