Understanding PID Loops

Skye Sweeney
Skye@fll-freak.com
May 20 2017

Taking a Shower

- From your experience with **your** shower, you pretty much know how your valve works to control temperature.

- Most of the time you put the valve in pretty much the right position on the first try.

- You only need a few nudges to get it just right.

Adjustment

- You have an idea where to put the valve.
- You have an idea how much to adjust the valve based on the temperature error.
- If you sense the temperature changing quickly, you may adjust even faster.
  - Hot water running out!
- When adjusting you need to account for the distance from valve to shower head
  - In my case it is about a 2.5 seconds lag

Caveats

- Varies due to
  - Season (cold water is really cold in winter)
  - Number of previous users
- All bets are off for the hotel shower!
  - PID controllers are tuned to a specific system.

So what are you doing?

- $T_{demanded}$ The temperature you would like
- T_{actual} The temperature the shower actually is.

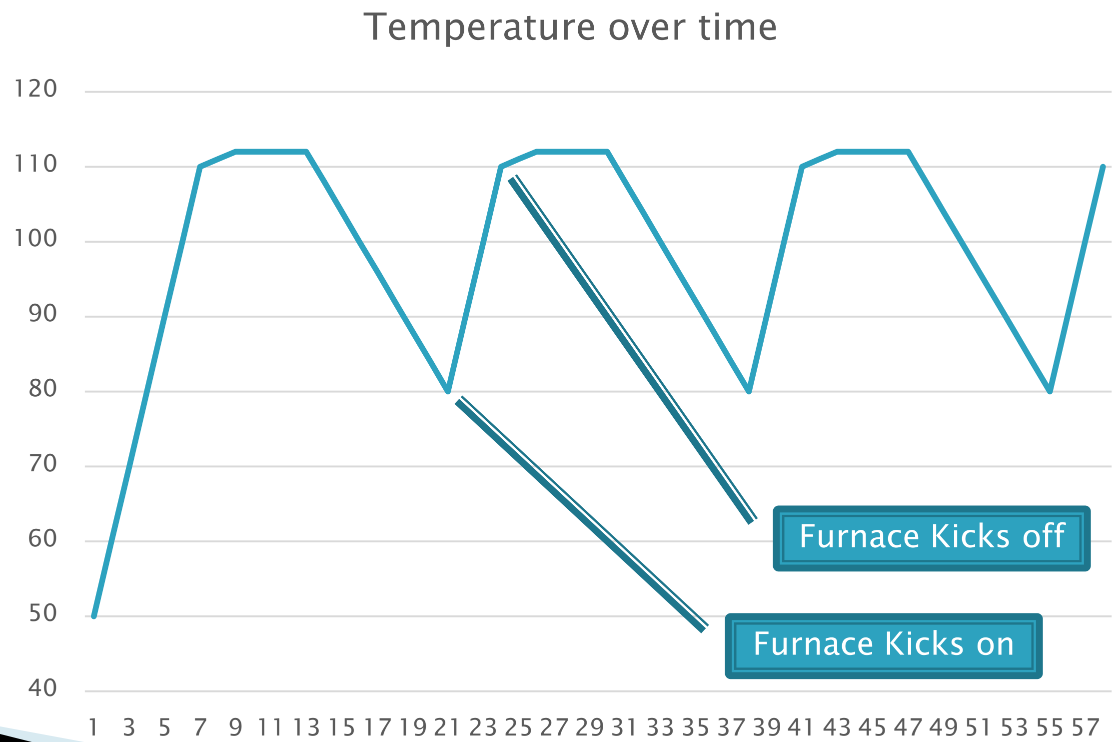
- Error = $T_{demanded} - T_{actual}$

- Want to drive Error to zero by adjusting the faucet valve.

- Definitions:
  - Setpoint — What you want. $T_{demanded}$ in this case.
  - Process Variable — What you have. T_{actual} in this case.
  - Control Variable — What you control. The valve in this case

My In-laws Shower

- There “Endless Hot Water” system allows the temperature to oscillate significantly.



Let's Fix the Shower!

- Temperature sensor in shower head
- Servo to drive valve
- Human interface to set temperature
- Arduino to control the whole mess
- What could possibly go wrong?

- But how to write control algorithm?

PID as the control algorithm

- A very popular control system algorithm. Often used in things like cruise controls, autopilots, factory controls. Best used for systems that are fairly linear.
- The name comes from
  - Proportional
  - Integral
  - Differential
- Also P, I, D, PI, PD, ID controllers

The Most Popular!

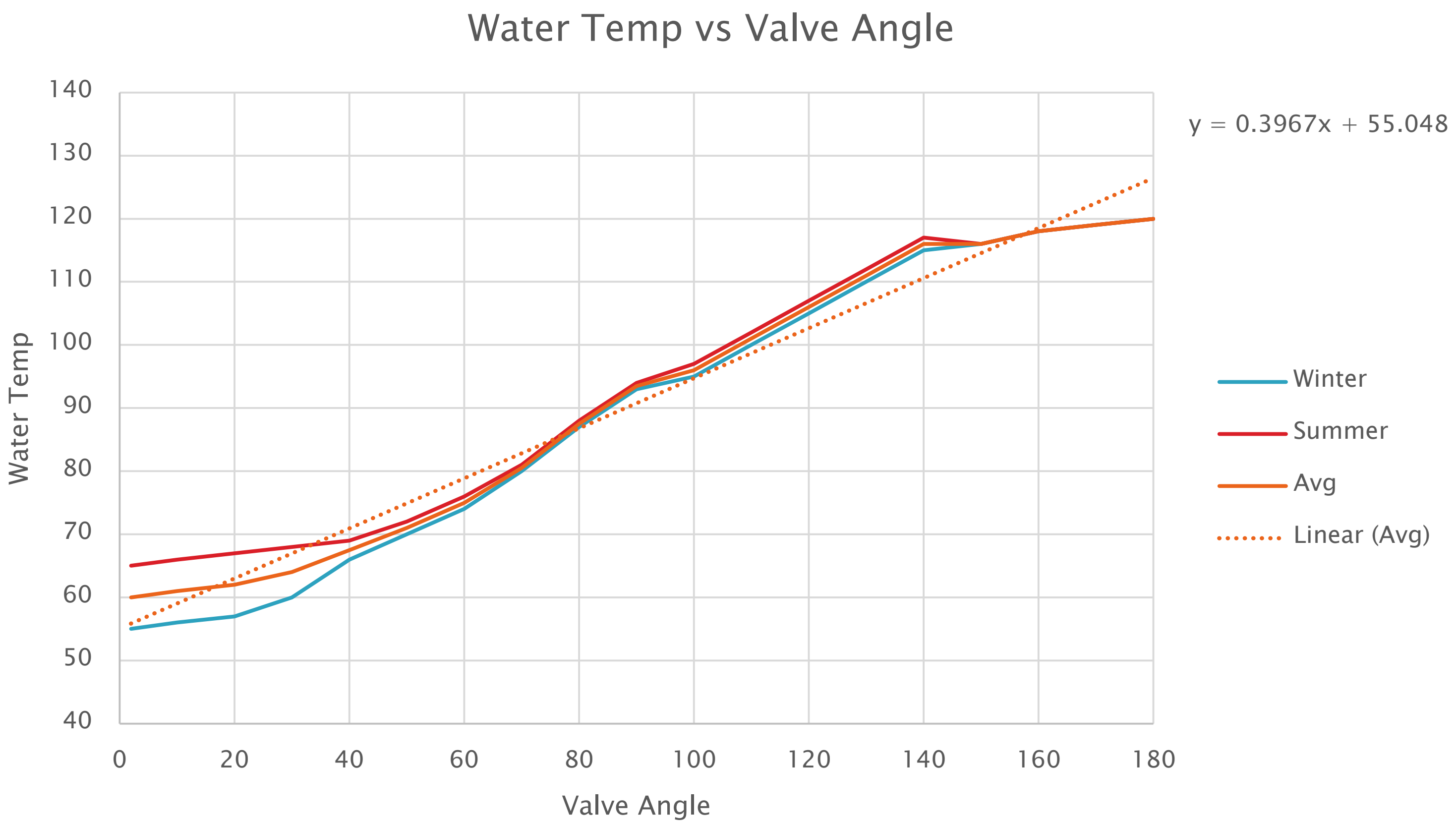
Software

```
loop(void)
{
   Actual      = getTemp();
   Demanded = getFromHuman();
   Error        = Actual-Demanded;


   valveAngle =        Some kind of magic


   setValveAngle(valveAngle);
   delay(period);
}
```

Proportional



Proportional

- So a good initial guess is:
- Temp = 0.3967 * ValveAngle + 55.048
- ValveAngle = (Temp–55.048)/0.3967
- ValveAngle = 2.52*Temp – 138.8
- For every 1F temp change, you need to turn the handle by about 2.52 degrees.

Software

```
loop(void)
{
   Actual       = getTemp();
   Demanded = getFromHuman();
   Error         = Actual-Demanded;


   P = Error * Kp;  // Kp ≅2.5F/ValveDeg


   analogWrite(P);
   delay(period);
}
```

Can’t ever get there!

- If the error is 0, then the valve is set to 0.
  - This makes the shower COLD!
- If the error is large, then the valve moves towards hot.

- So you reach a steady state, but it is NOT the temperature you want. ☹

Integral

- The Integral term will tweak the solution.
  - If too warm make colder
  - If too cold make warmer
- Each time we run through the loop, we will add or subtract a bit to valveAngle to move the valve in the right direction.

(In math, continuous adding is known as ‘taking the integral’)

Integral

- Compute the error
  - $E = T_{demanded} - T_{actual}$

- Now perform the 'integral'
  - I = I + E * Ki; (Ki is a positive magic constant; more later)

- If we are too cold, $T_{demanded} > T_{actual}$ so the error will be positive.

- 'I' will increase each time the shower is too cold.
- 'I' will decrease each time the shower is too warm.
- 'I' will change more the worse the error is.

Software

```
loop(void)
{
   Actual     = getTemp();
   Demanded = getFromHuman();
   Error      = Actual-Demanded;


   P = Error * Kp;

   I  = I + Error * Ki;


   setValveAngle(P + I);
   delay(period);
}
```

90% of time this is all you need!

Differential

- Rarely used in practice.
  - Error rate term is often very noisy
- Acts based on how fast the error is changing.
- Most often used to dampen the system
  - Think of a car shock absorber.
  - The K term will often be negative to oppose PI

Software

```
loop(void)
{
   Actual       = getTemp();
   Demanded = getFromHuman();
   Error          = Actual–Demanded;
   ErrorRate    = (LastError – Error);

   P = Error * Kp;
   I  = I + Error * Ki;
   D = ErrorRate * Kd;
   LastError = Error;

   setValveAngle(P + I + D);
   delay(period);
}
```

Feed Forward

- Used when you have good system knowledge.
- If you can predict where the control should be, start there!
- Does not rely on having to wait for I to integrate up.
- For the shower we have a good system model.
  - ValveAngle = 2.52*Temp – 138.8

Software

```
loop(void)
{
   Actual     = getTemp();
   Demanded = getFromHuman();
   Error      = Actual–Demanded;
   ErrorRate    = (LastError – Error);

   P = Error * Kp;
   I  = I + Error * Ki;
   D = ErrorRate * Kd;
   LastError = Error;
   F = (Demanded * 2.52 – 138.9);

   setValveAngle(P + I + D + F);
   delay(period);
}
```

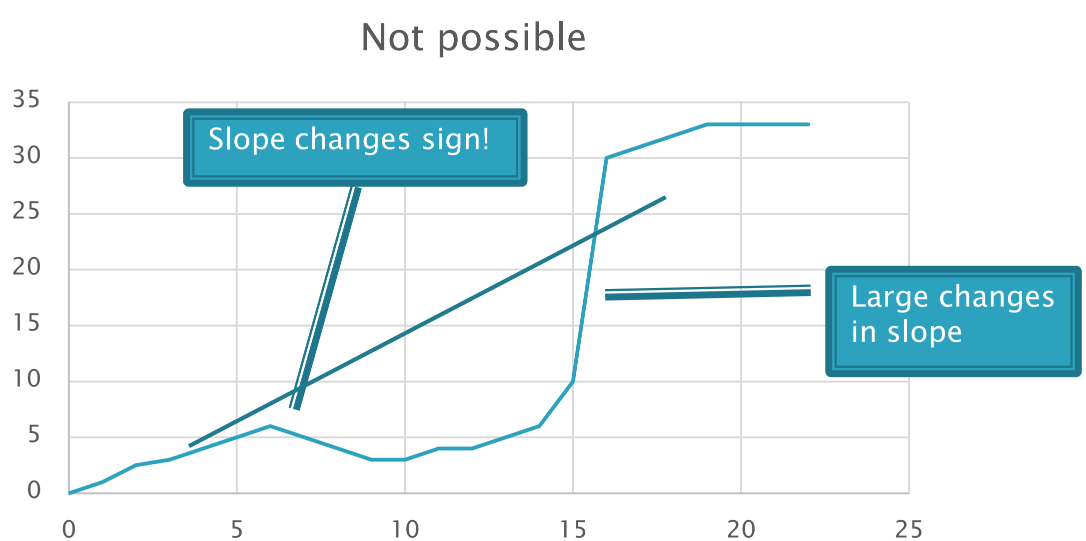
Software Considerations

- **Must** call the PID control loop at a regular rate.
- Ensure that you limit variables to within legal ranges.
  - Limit P, I, D, F, and sum terms independently.
    - P = …
    - P = constrain(P, lowP, highP);
    - I = …
    - I = constain(I, lowI, highI);
    - …

When can they be used?

- When a control directly affects the output.
  - Can’t control car speed by changing radio volume!

- When the system is monotonic and linear.



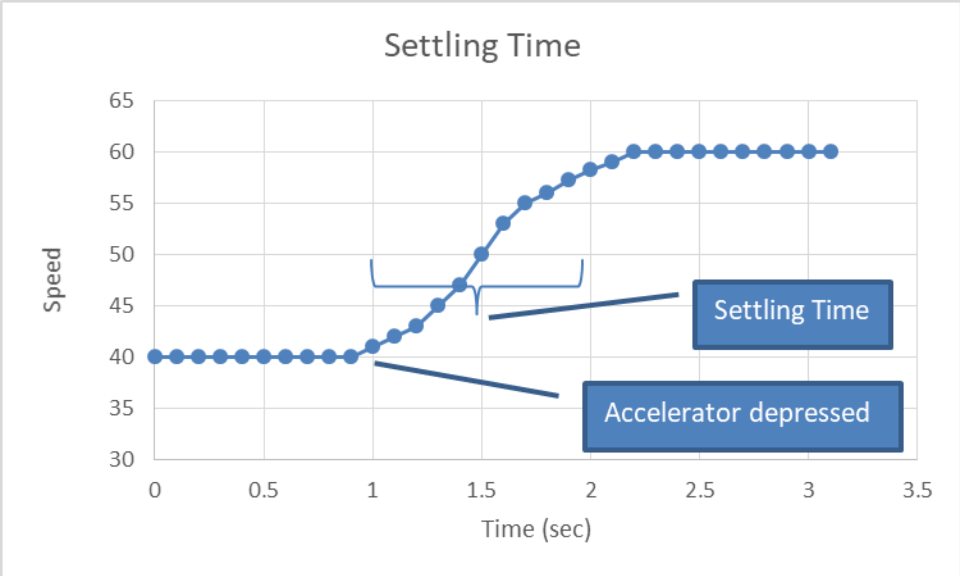
Monotonic: Having the same sign slope

Dynamic Considerations

- How often to call the PID loop?
  - How fast does the system respond?
  - Blast furnace can run slower than Segway.
  - How fast does sensor change?
  - Rule of Thumb:
    - Run the PID loop at about 10 to 100 times faster then the settling time.

Settling Time

- The time it takes the natural system to return to equilibrium after a disturbance.
  - Shower: The time it takes for the water to move from the valve to your skin and reheat the pipes.
  - Cruise control: The time it takes the car to reach its new speed when you make a sudden accelerator change.



Types of Systems

- Self-regulating
  - KI is the key term for stability
- Integrating
  - KP is the key term for stability

Self Regulating Samples

- Temperature control
  - Gas flow to control oven temperature
- Speed control
  - Accelerator to control speed.
- Power Supply
  - PWM duty cycle to control voltage
- Servo control
  - Pulse width to control output position

Integrating Examples

- Fluid level
  - Valve filling slowly draining tank
    - Swimming pool with evaporation
- Heading
  - Steering wheel controlling direction

Tuning

- How to pick kP, kI, and kd?

- Trial and error
  - Adjust kP till you get oscillations
  - Back kP off by 25-50%
  - Adjust kI to get good following
  - Only use kD if nothing else works.

- Heuristic methods like Zeiggler Meyers
  - Requires the ability to run specific tests and record data.
  - Data is then processed to compute the kP, kI, and kD terms.
  - Often leads to aggressive tuning that must be tweaked.

Things to look out for

- Symmetry
  - Heat only temperature control vs Peltier cooler
- Linear discontinuities
  - May need different gains for different gears
- Sensor placement
  - Minimize dead time (latency)
- Sensor accuracy, update rate, and noise
  - You get what you pay for!
- Control accuracy
  - 8 bit PWM only gives you 256 steps.

Improvements

- Windup removal
  - Limit I to some range.
- Lockouts
  - Freeze loop when some known transient event occurs like opening the oven door.
- Derivative filtering
  - Add small filter to clean up error rate noise
    - FiltErrorRate = k * FiltErrorRate + (1.0–k)*ErrorRate

Bumpless operation

- When setpoint changes, the error value will change suddenly.
- This can cause a significant bump (shock) to the system.
- So we want to create a way to smooth over the transition.

Bumpless calculations

- $CV_1 = e_1 * K_p + I_1 * K_i + \dot{e}_1 * K_d$
- $e_1 = SP_1 - PV$
- $CV_2 = e_2 * K_p + I_2 * K_i + \dot{e}_2 * K_d$
- $e_2 = SP_2 - PV$
- Want $CV_1 \cong CV_2$ when $SP_1 \rightarrow SP_2$

- $e_1 * K_p + I_1 * K_i + \cancel{\dot{e}_1 * K_d} = e_2 * K_p + I_2 * K_i + \cancel{\dot{e}_2 * K_d}$

- $I_2 = I_1 + K_p / K_i * (SP_1 - SP_2)$

Example 1

- Wall following robot
- Sensor is an IR distance sensor
  - Like a Sharp
- Control is the difference in motor speeds.
- This is an Integrating system

Example 2

- Motor speed control
- Sensor is an optical encoder (90 cpr)
- Speed control is via PWM (255 counts)
- This is a self regulating system

Example 2

- What is the settling time?
  - Run step test and find about 1 second

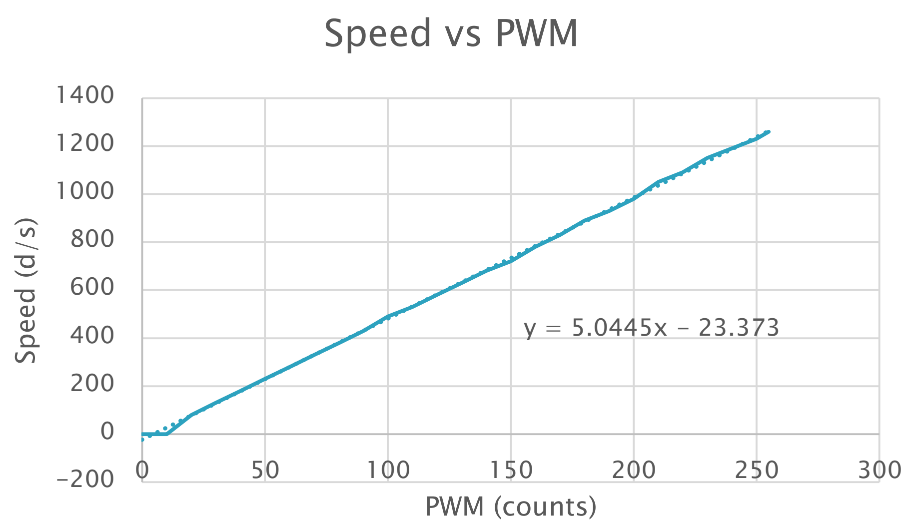
- What should be the PID update rate?
  - Settling time/(10 to 100) is 100ms to 10ms

Example 2

- How accurately can we measure speed?
  - At full speed, each stripe goes by in 2.5ms
  - Arduino micros gives answer to +- 4 usec
  - This equates to a 0.16% error.
- How fast is our sensor update rate?
  - At the low speed end each stripe takes 20ms!
- How accurately can we set speed
  - At full speed we get 1600 deg/sec
  - At full speed our PWM count is 255
  - So each count is about 6.25 deg/sec

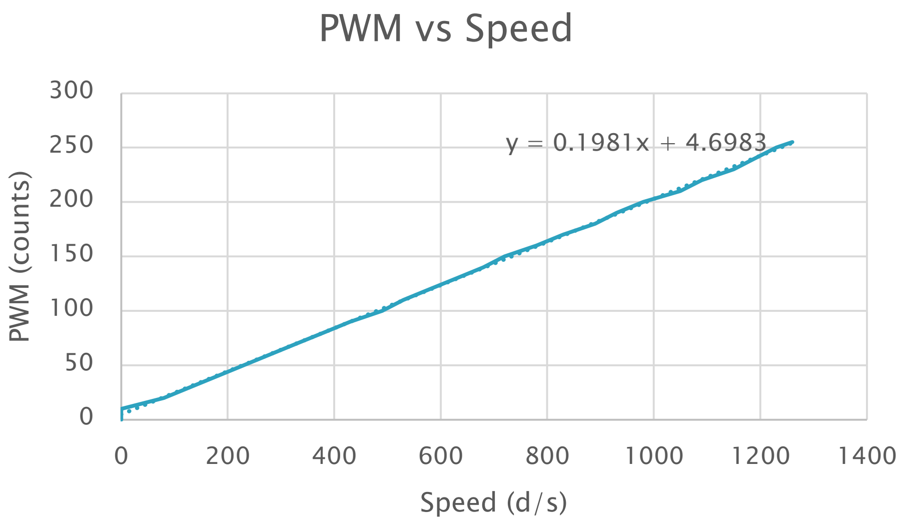
Example 2

- Can we guess what KP should be?
- Get speed at each PWM count



Example 2

- Swap axes to determine slope
  - You need ~0.2 PWM counts per deg/sec
  - That is a good place to start with KP!



Demo

Questions?