Easier PD Controller Design

Design a controller that is just good enough
without tedious trial and error

UK Micromouse and Robotics Society

Peter Harrison
UKMARS
2022

Outline

- System Characterisation and Responses
- Closed Loop PD Control
- Tuning Controller Responses
- Simple Calculation gets you started

Characterise the Drive system

- Voltage => Speed
- Current => Torque
- Torque => Acceleration
- **Speed response is simple 1st Order**

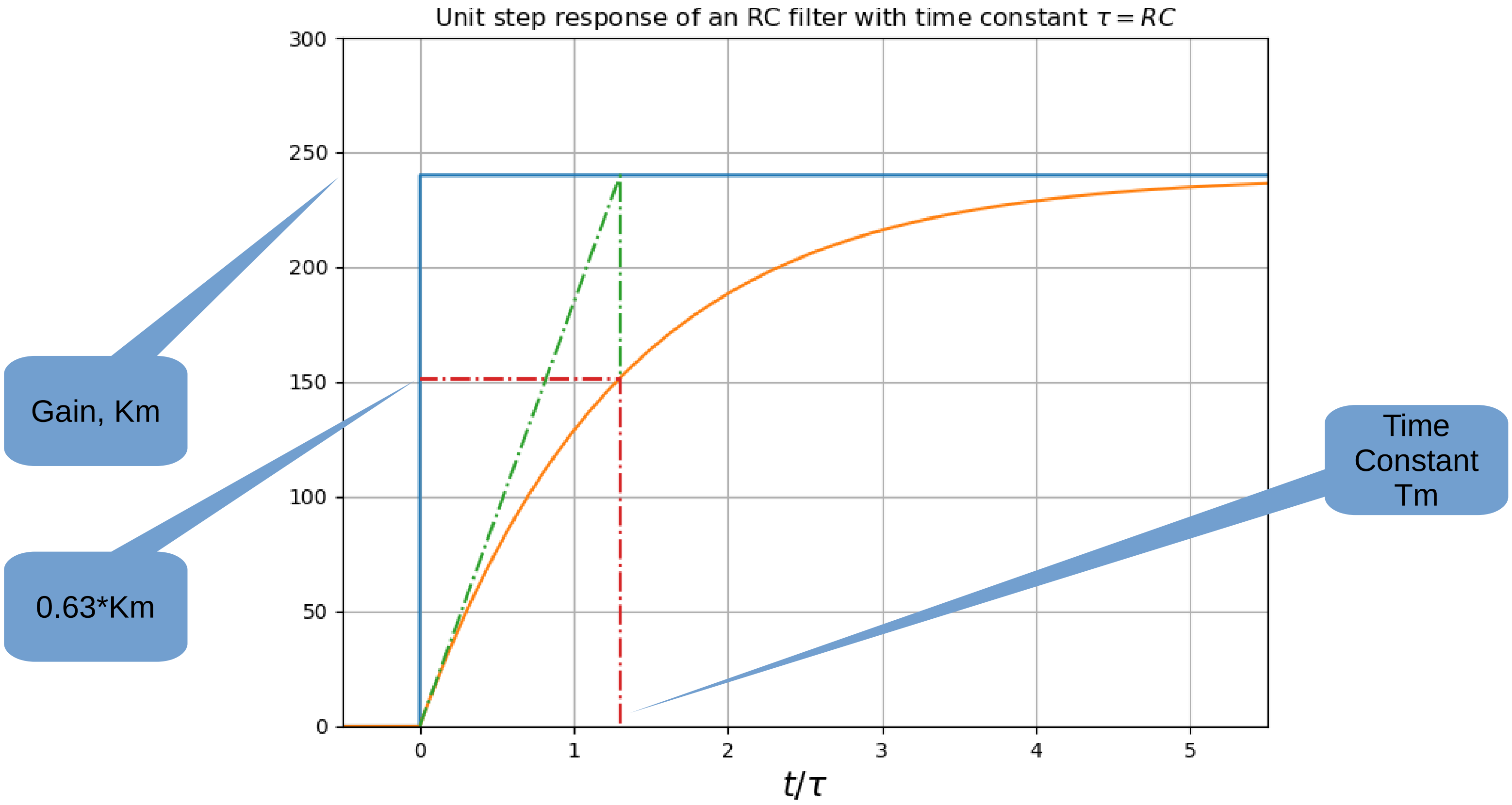
Drive Characteristics

- Two constants define the open loop response

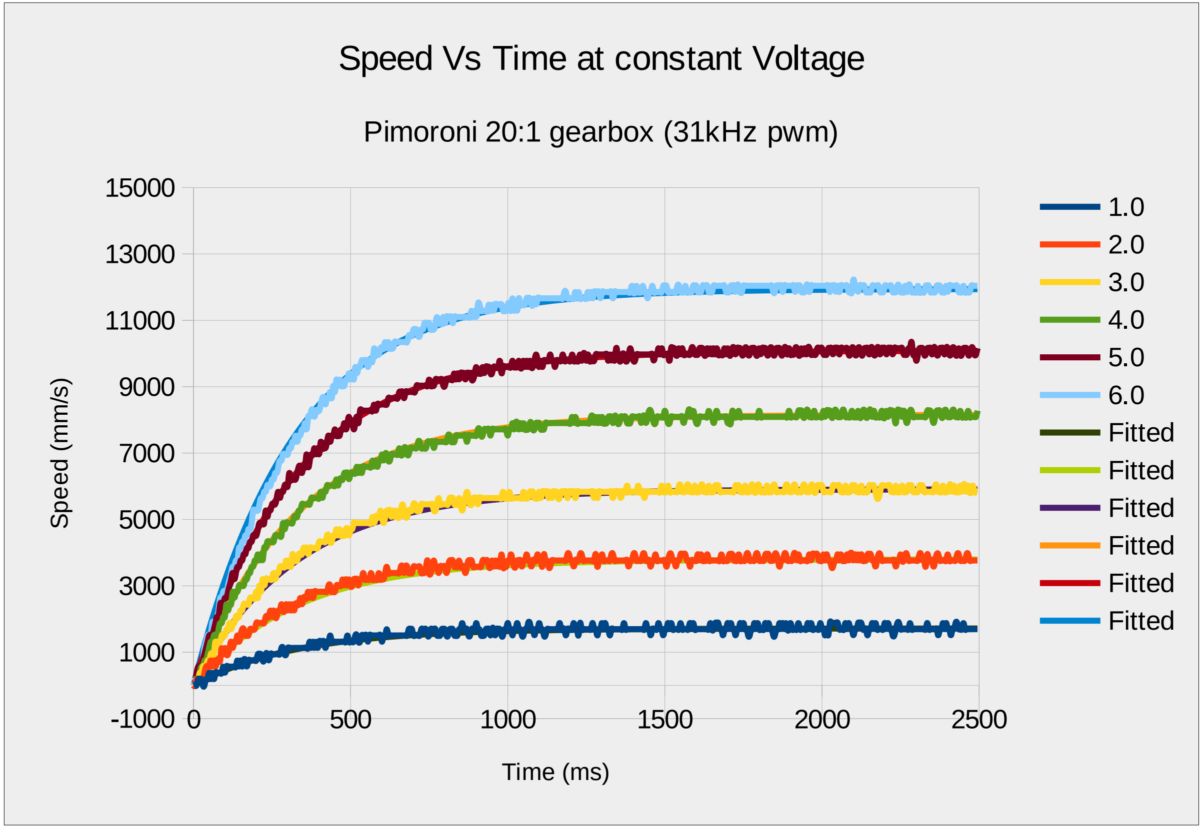
  K_m is the system gain in mm/s/Volt

  T_m is the system time constant

First Order Step Response

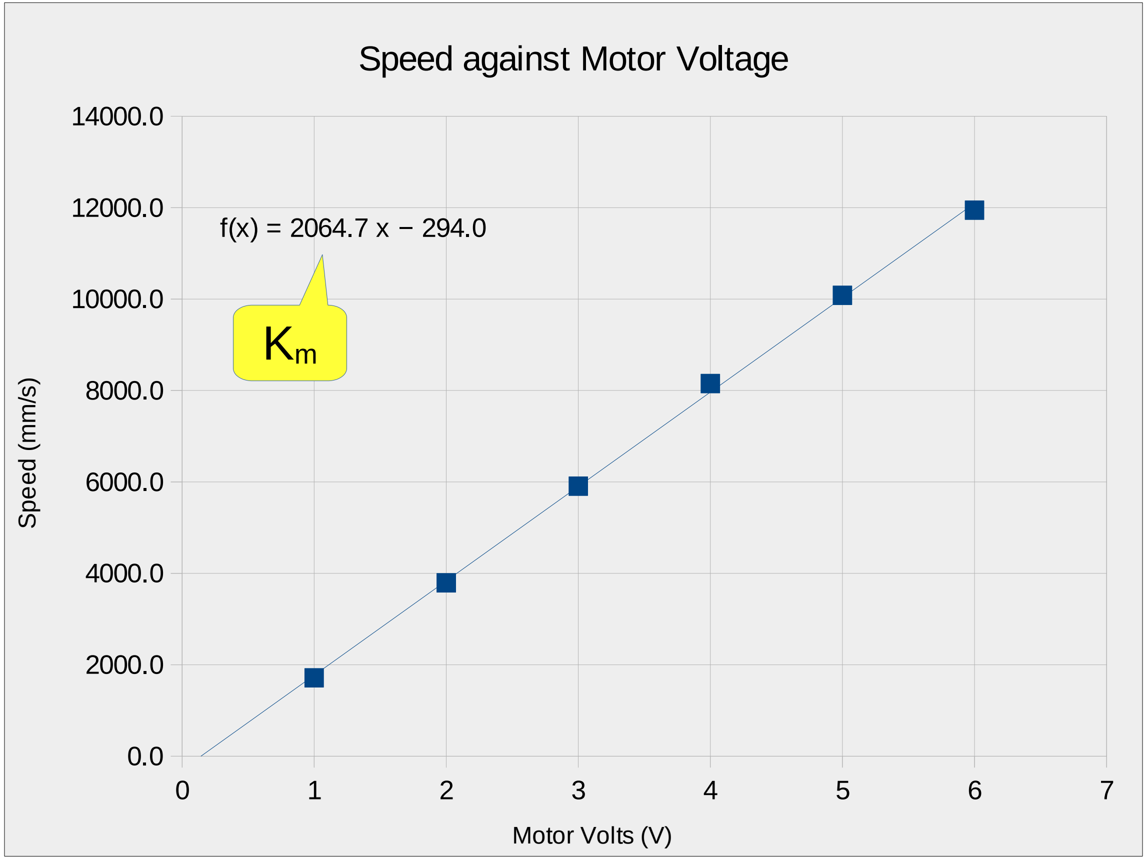


Plot Open Loop Voltage Response



$$V(t) = V_{max}\left(1 - e^{-\left(\frac{t}{T_m}\right)}\right)$$

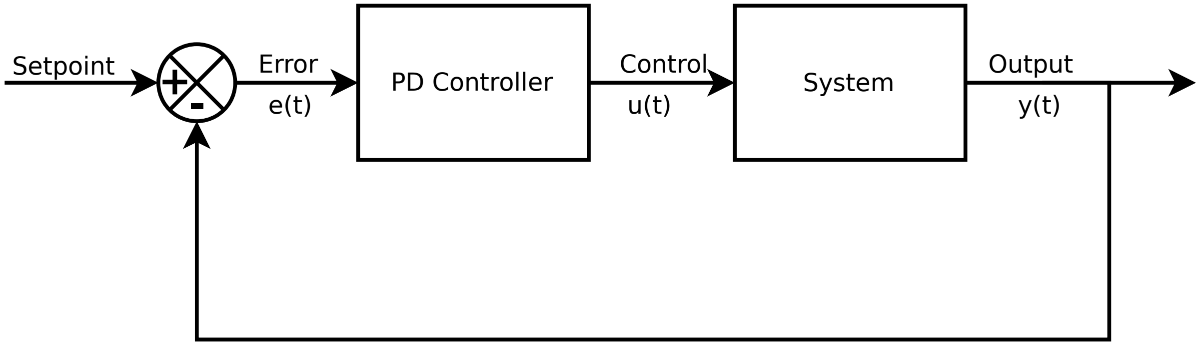
Obtain System Gain Km



Motor Lab 1
Open Loop Response

Closed Loop PD Controller

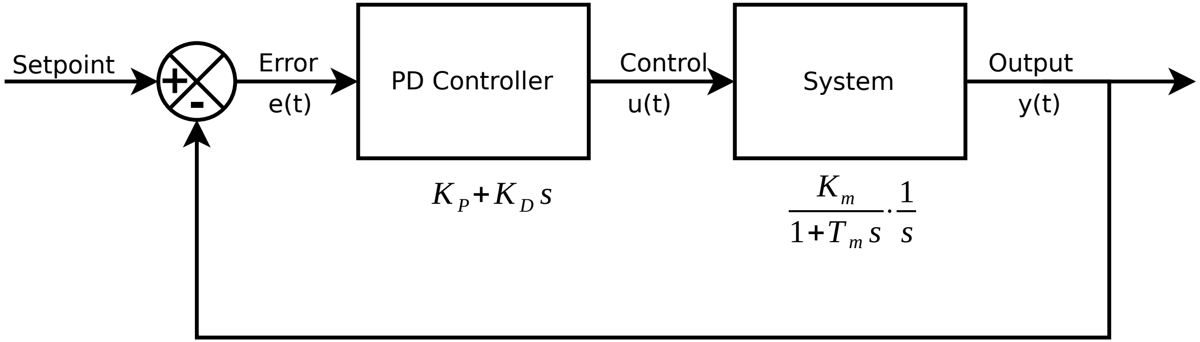
- System Block Diagram



  – Error compensation
  – Disturbance rejection
  – Improved performance

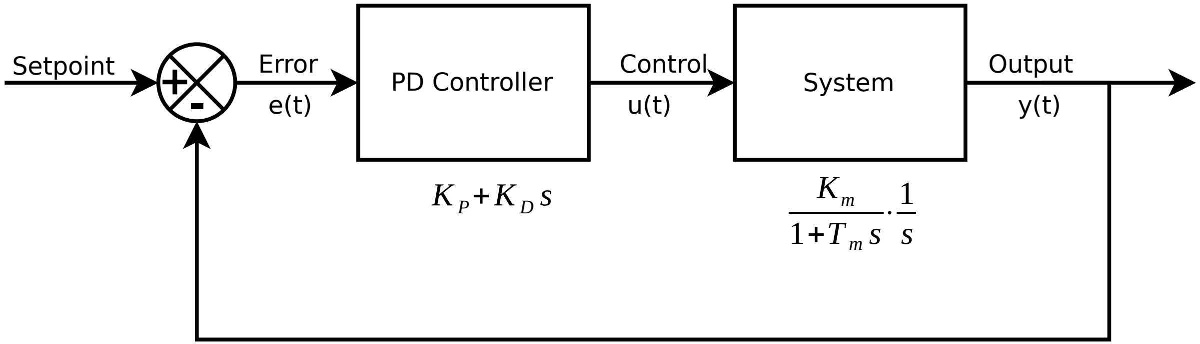
Closed Loop PD Controller

- System Block Diagram



Closed Loop PD Controller

- System Block Diagram

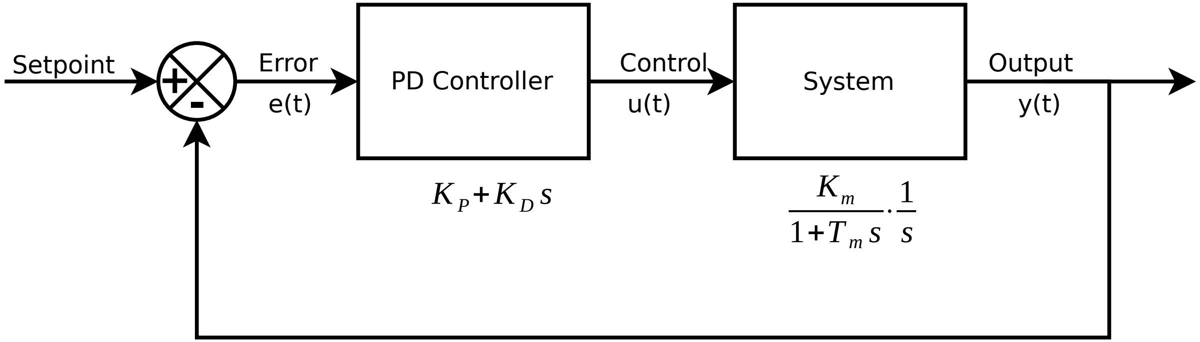


- Closed Loop Transfer Function

$$\frac{K_m}{T_m}\cdot\frac{K_D s+K_P}{s^2+\left(\frac{K_D*K_m+1}{Tm}\right)s+\frac{K_P K_m}{Tm}}$$

Closed Loop PD Controller

- System Block Diagram



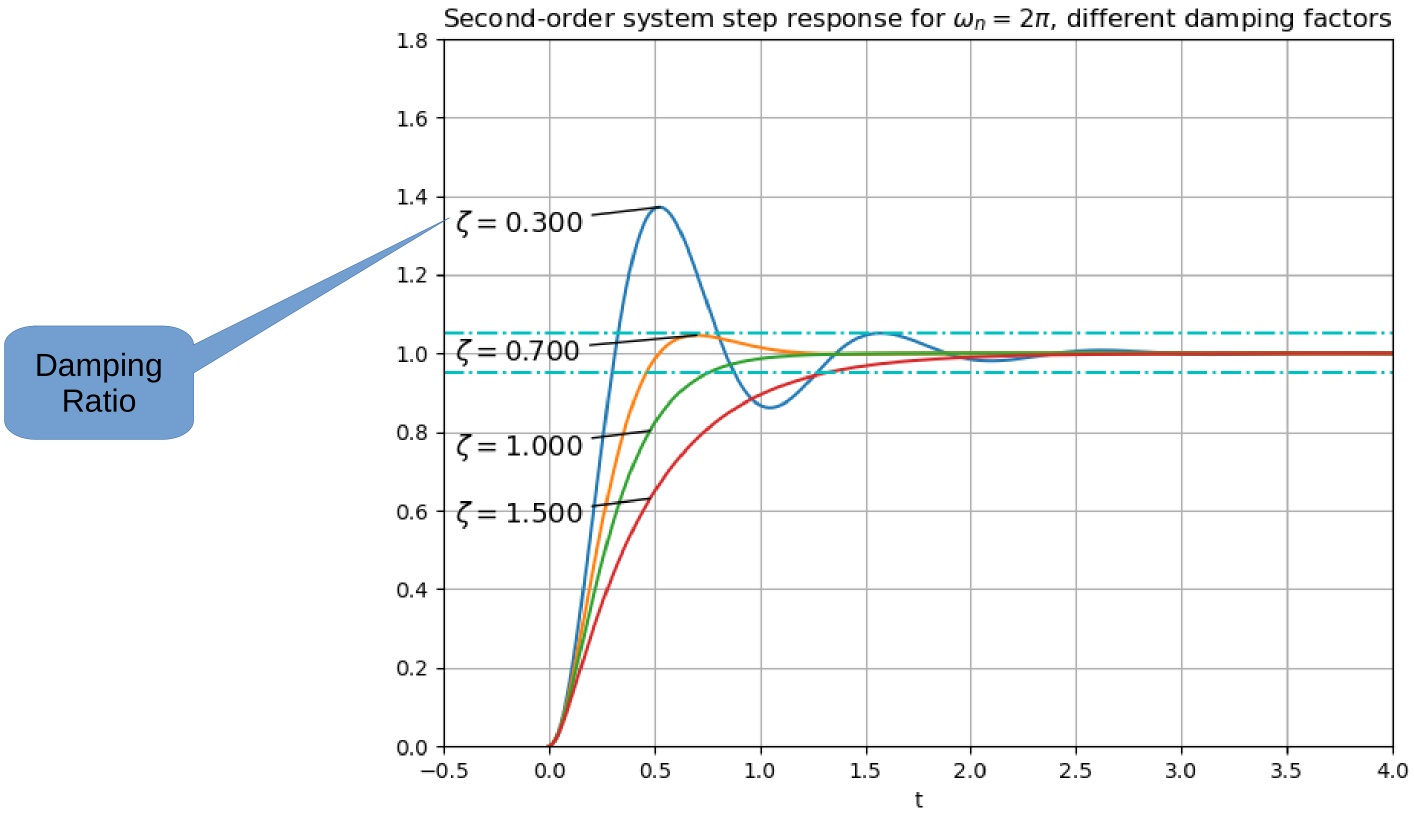
- Closed Loop Transfer Function

$$\frac{K_m}{T_m}\cdot\frac{K_D s+K_P}{s^2+\left(\frac{K_D*K_m+1}{Tm}\right)s+\frac{K_P K_m}{Tm}}$$

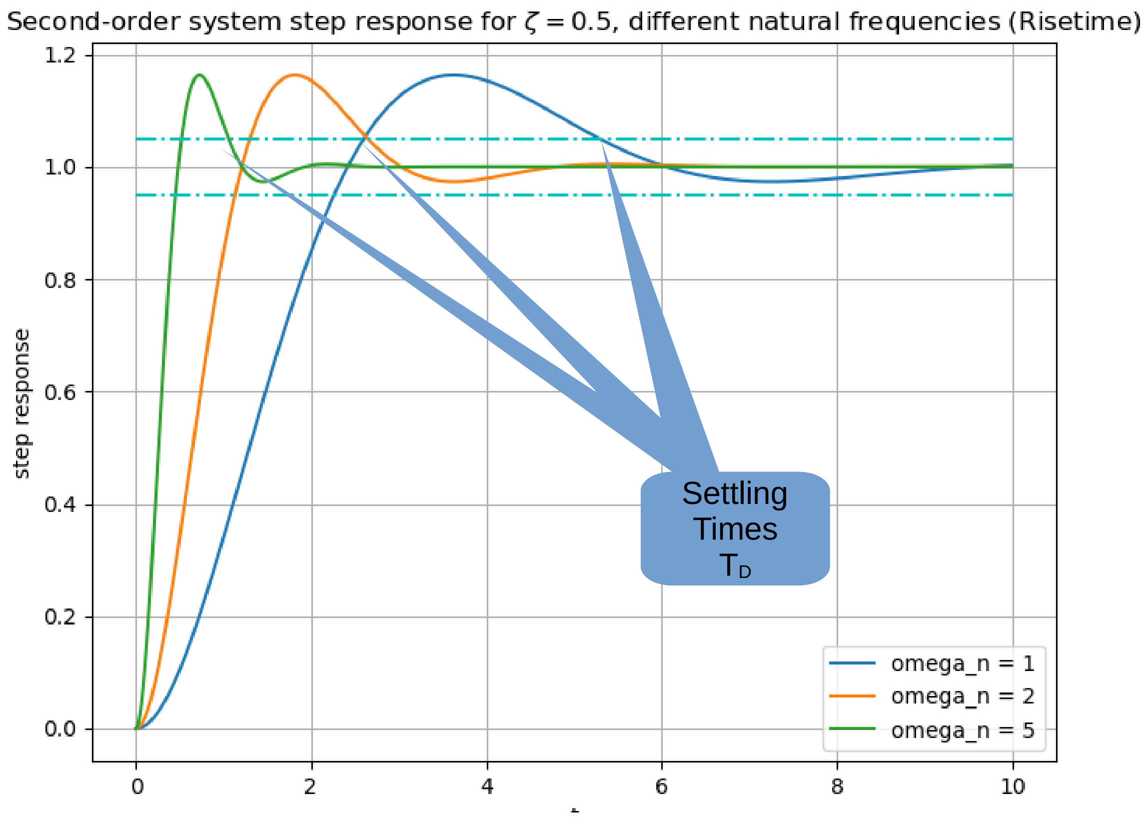
Second Order System Response

- Well understood
- Many techniques
- Standardised responses in terms of
  - Damping Ratio, ζ (zeta)
  - Natural Frequency, ω_n (equivalent to rise time)

2^{nd} Order Step Response Damping



2^{nd} Order Response Settling Time



Closed Loop Step Response Test

- No good for characterisation
- Too many unknowns
- Controller saturation with high gains
- Unpredictable outputs
- **Real robots track changing profiles**

Controller Tuning

- Multiple Techniques
  - Mathematical
  - Empirical
- Skill/Experience needed
- Intimate knowledge of system required
- Domain specific

Motor Lab 2
Manual Tuning

Don’t Give Up

- Recall that feed forward
  - is easy
  - improves for poor controllers
- So – how to make a ‘*good enough*’ controller without tedious tuning?

Calculate the Constants

- Standard well understood transfer function

$$u = G\frac{\dot{\omega}_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

- Compare terms and approximate to get

$$K_p = \frac{T_m}{K_m} \cdot \frac{16}{\zeta^2 T_D^2}$$

$$K_d = \frac{8T_m - T_D}{T_D K_m} \qquad \mathit{for} \quad T_D < 8T_m$$

Calculate the Constants

- Standard well understood transfer function

$$u = G\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

- Compare terms and approximate to get

$$K_p = \frac{T_m}{K_m} \cdot \frac{16}{\zeta^2 T_D^2}$$

$$K_d = \frac{8T_m - T_D}{T_D K_m}$$

Motor Lab 3
Choosing Response Parameters

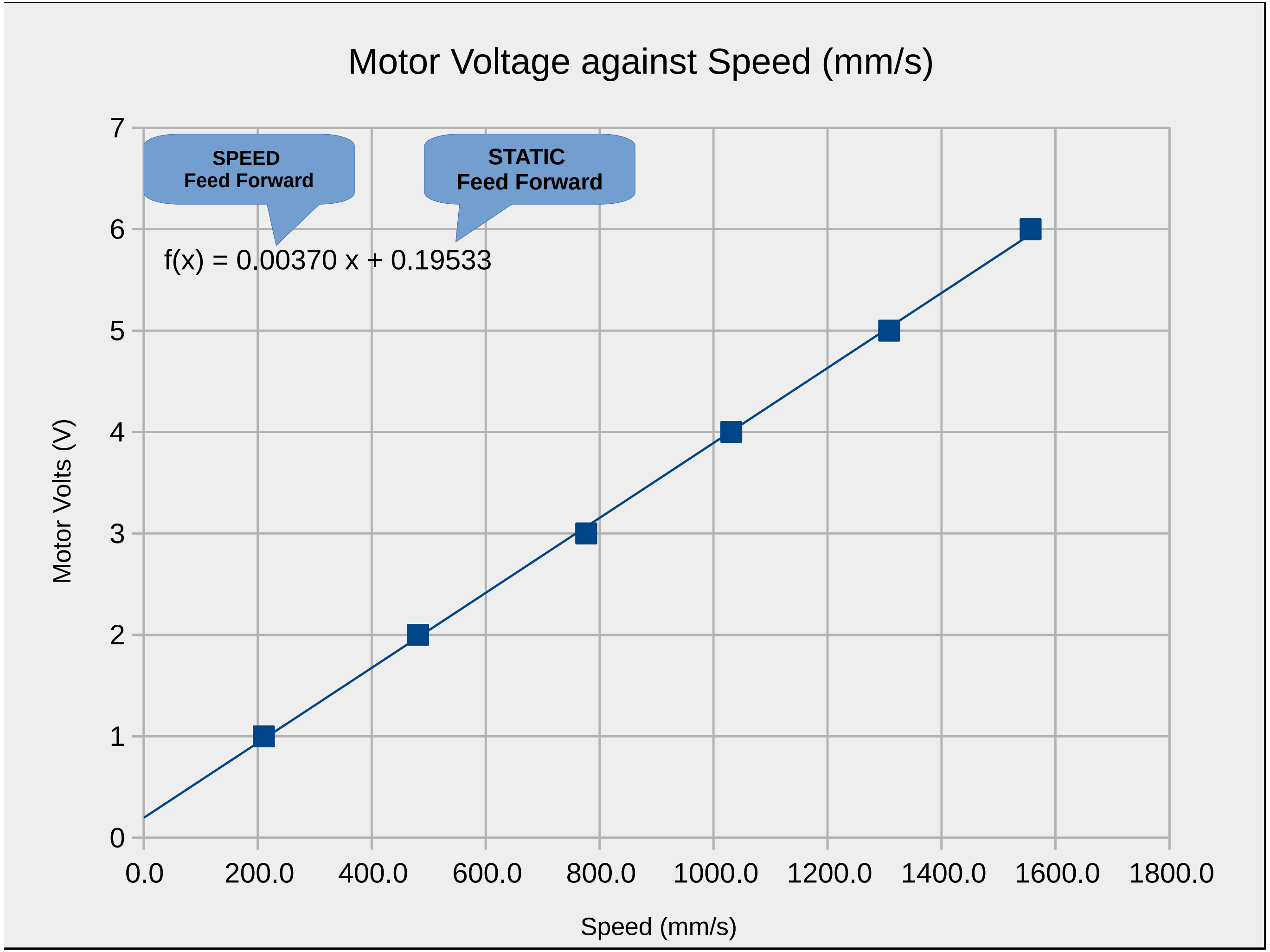
Summary

- Feedback controllers are essential for accuracy
- But can be hard to tune
- Simple intuitive calculations give a ‘good enough’ controller
- Add feed forward for the icing on the cake

“I would have written a shorter letter, but I did not have the time.”

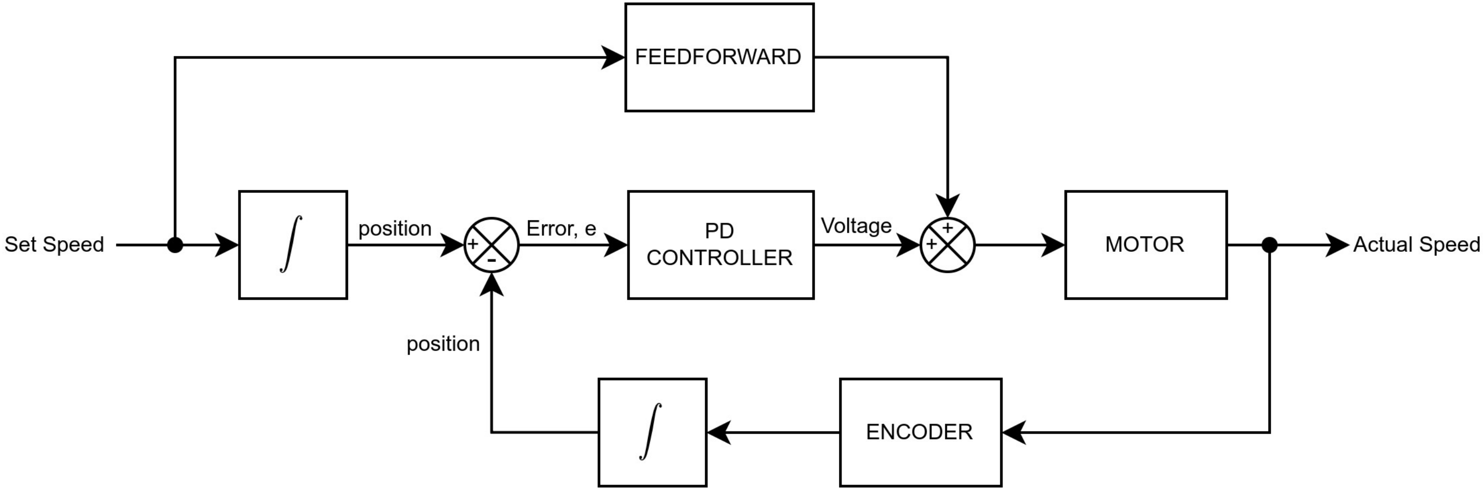
Blaise Pascal

Thank You



Motor Voltage against Speed (mm/s)
SPEED
Feed Forward
STATIC
Feed Forward
f(x) = 0.00370 x + 0.19533
Motor Volts (V)
0
1
2
3
4
5
6
7
0.0
200.0
400.0
600.0
800.0
1000.0
1200.0
1400.0
1600.0
1800.0
Speed (mm/s)

Feedback + Feed Forward



PD Controller Implementation

$$u(t) = K_p e(t) + \frac{K_d(e(t) - e(t-1))}{dt}$$

```
delta_e = error - old_error
old error = error

ctrl_out = Kp * error + Kd * delta_e
```

Note that Kd must be divided by the loop time
Do this at configuration, not in the loop

Feedforward function complete

```
// For the right wheel
float get_right_feedforward(float speed) {
  static float old_speed = 0;
  float acc = (speed - old_speed) * LOOP_FREQUENCY;
  old_speed = speed;
  if (speed > 0) {
    right_FF = ACC_FF * acc + SPEED_FF * speed + BIAS_FF;
  } else {
    right_FF = ACC_FF * acc + SPEED_FF * speed - BIAS_FF;
  }
  return rightFF;
}
```

Implementation - UKMARSBOT(ish)

```
void update_motor_controllers() {
  pos_output = position_controller(); // the PD Controller
  left_output = pos_output;
  right_output = pos_output;
  v_fwd = forward_speed();
  v_left = v_fwd - (PI / 180.0) * MOUSE_RADIUS * v_rot;
  v_right = v_fwd + (PI / 180.0) * MOUSE_RADIUS * v_rot;
  left_output += SPEED_FF * v_left + BIAS_FF;
  right_output += SPEED_FF * v_right + BIAS_FF;
  set_right_motor_volts(right_output);
  set_left_motor_volts(left_output);
}
```

This is a simplification of the actual code

Acceleration Feed Forward Constant

- Response $V(t)=V_{max}\left(1-e^{-\left(\frac{t}{\tau}\right)}\right)$
- Acceleration $A(t)=\frac{V_{max}}{\tau}e^{-\left(\frac{t}{\tau}\right)}$
- At t = 0: $A_o=\frac{V_{max}}{\tau}$
- Plot Voltage against Acceleration
- Slope is Acceleration Constant $K_{acc}=K_{ff}\,\tau$

Position Controller

Implementing a PD controller is not hard

```
float position_controller() {
  s_fwd_error += forward.increment() - robot_fwd_increment();
  float diff = s_fwd_error - s_old_fwd_error;
  s_old_fwd_error = s_fwd_error;
  float output = FWD_KP * s_fwd_error + FWD_KD * diff;
  return output;
}
```