CS65K Robotics

Modelling, Planning and Control

Chapter 8: Motion Control

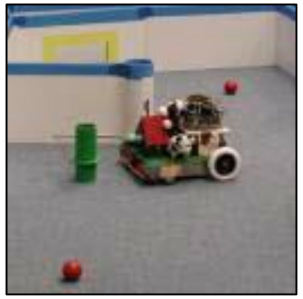
LECTURE 14: CONTROL LOOPS

DR. ERIC CHOU

IEEE SENIOR MEMBER

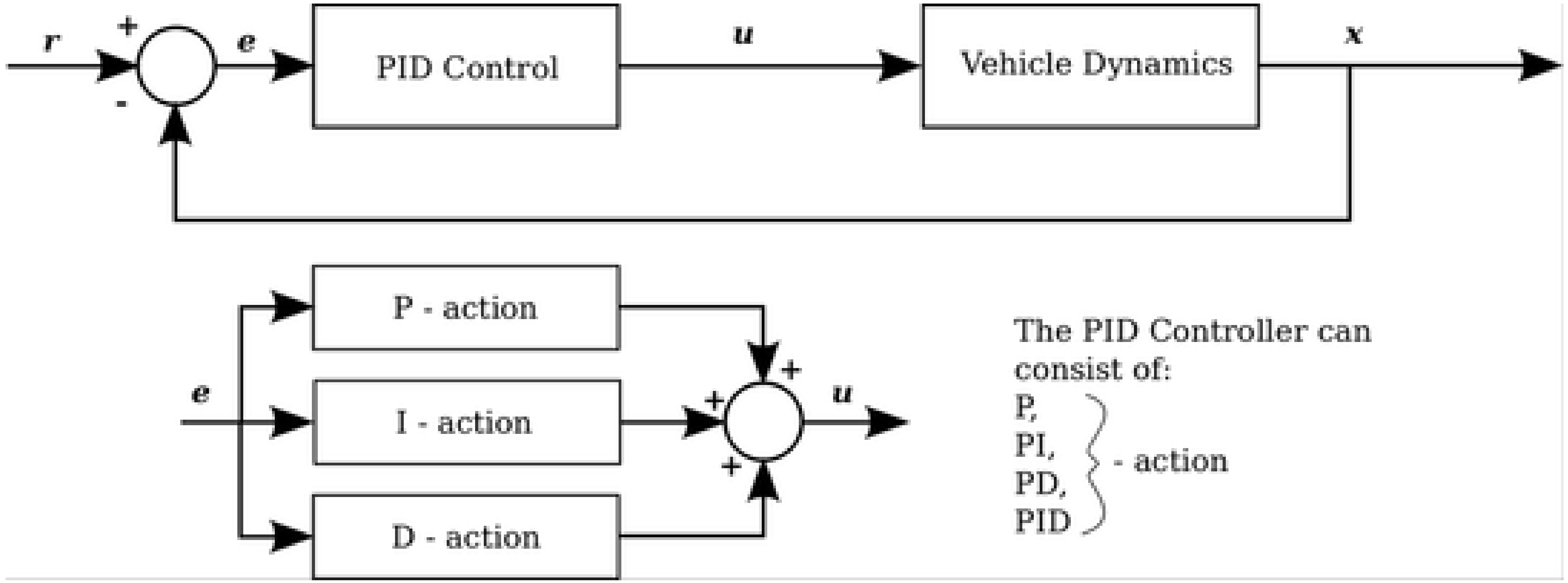
Objectives

- Basic Feedback Control
- High-level control system paradigms
    - Model-Plan-Act Approach
    - Behavioral Approach
    - Finite State Machine Approach
- **Low-level control loops**
    - PID controller for motor velocity
    - PID controller for robot drive system



Overview of Controllers

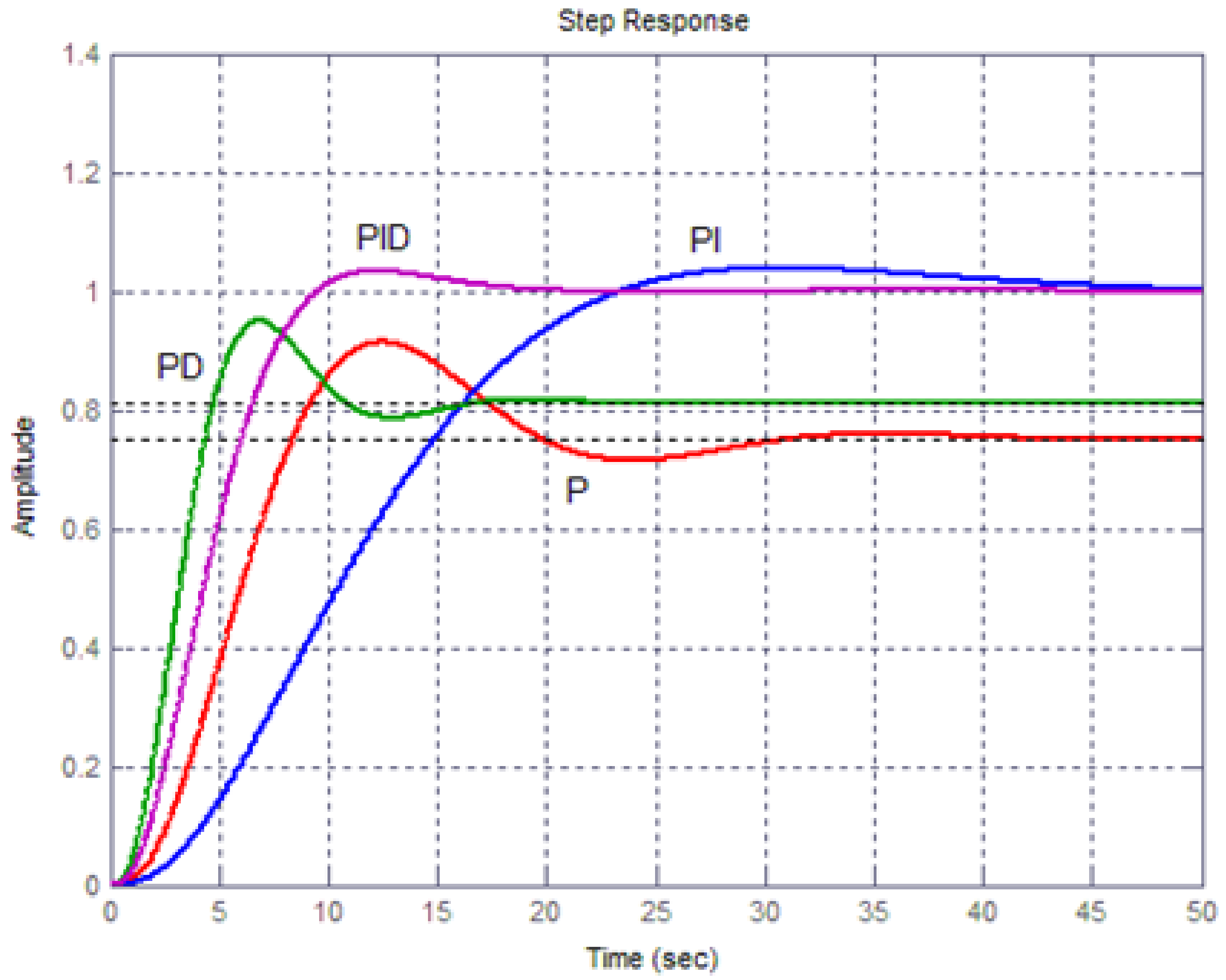
SECTION 1



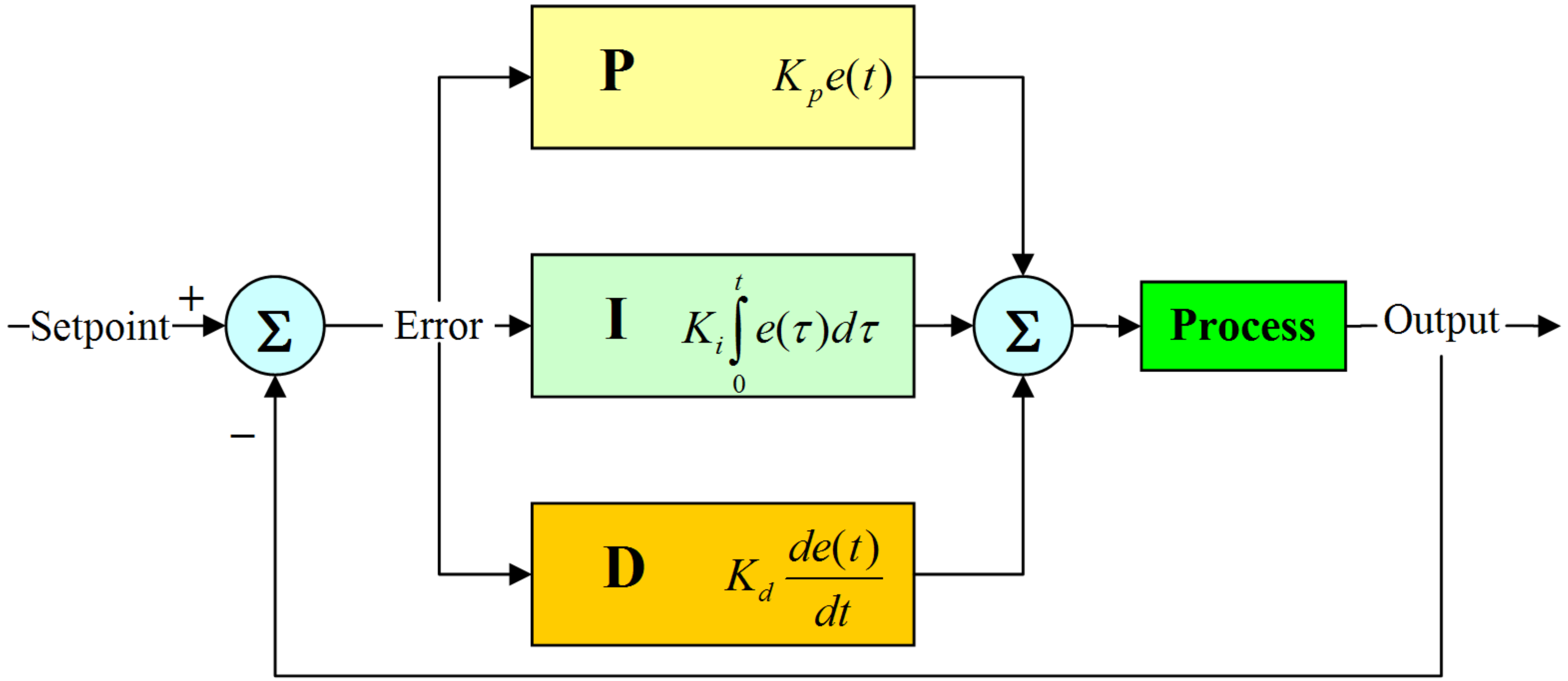
The PID Controller can consist of:

P,
PI,
PD,
PID

} - action



Step Response
PID
PI
PD
P
Amplitude
Time (sec)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
5
10
15
20
25
30
35
40
45
50



Setpoint
+
Σ
−
Error
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
Σ
Process
Output

Simple PID Simulator

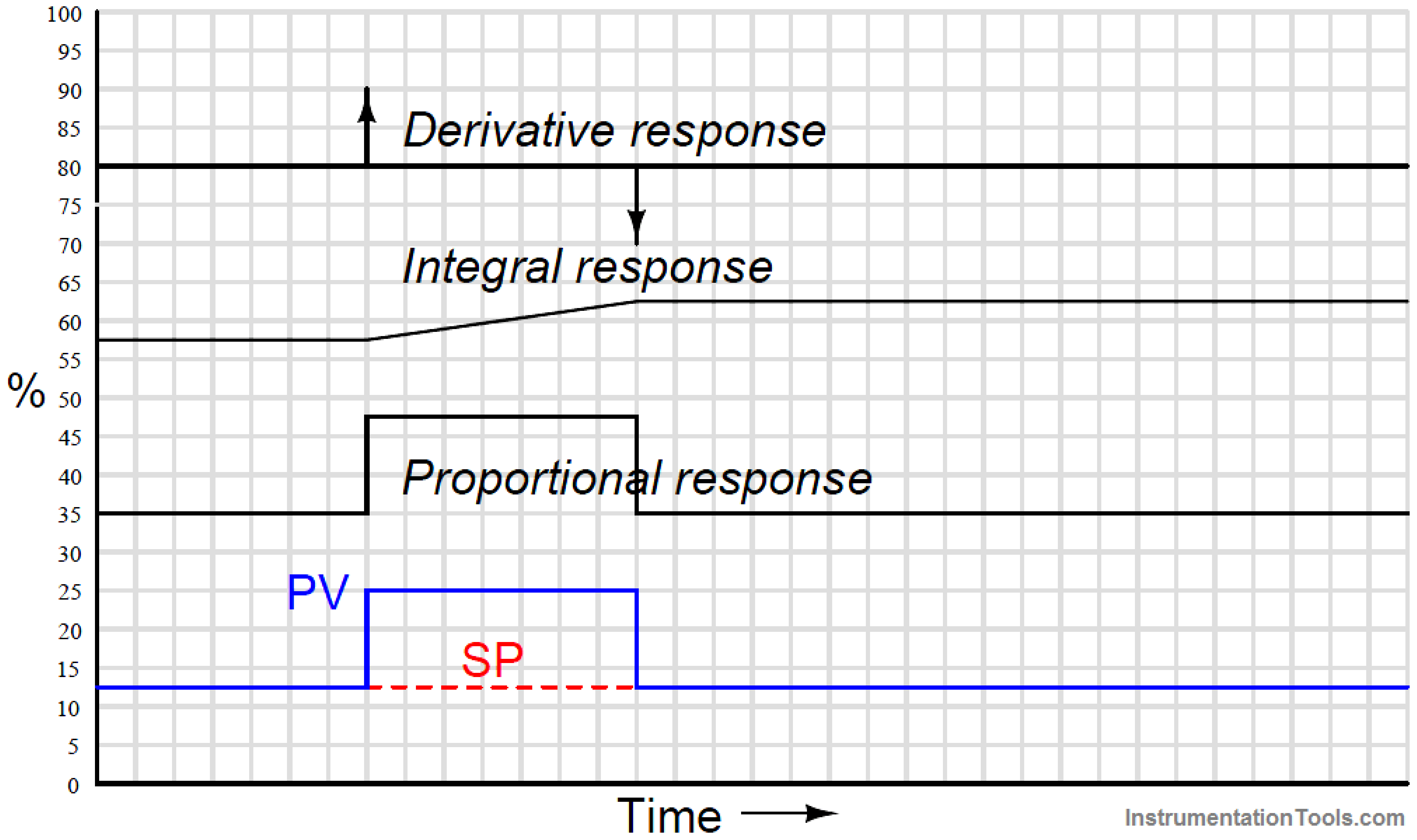
Demo Program: PID.py + test_pid.py

- A simple simulator.
- Used to see the basic characteristics of PID controllers.
- Specify I=0, D=0 for P-only controller.
- Specify D=0, for PI controller.
- Specify I=0, for PD controller.

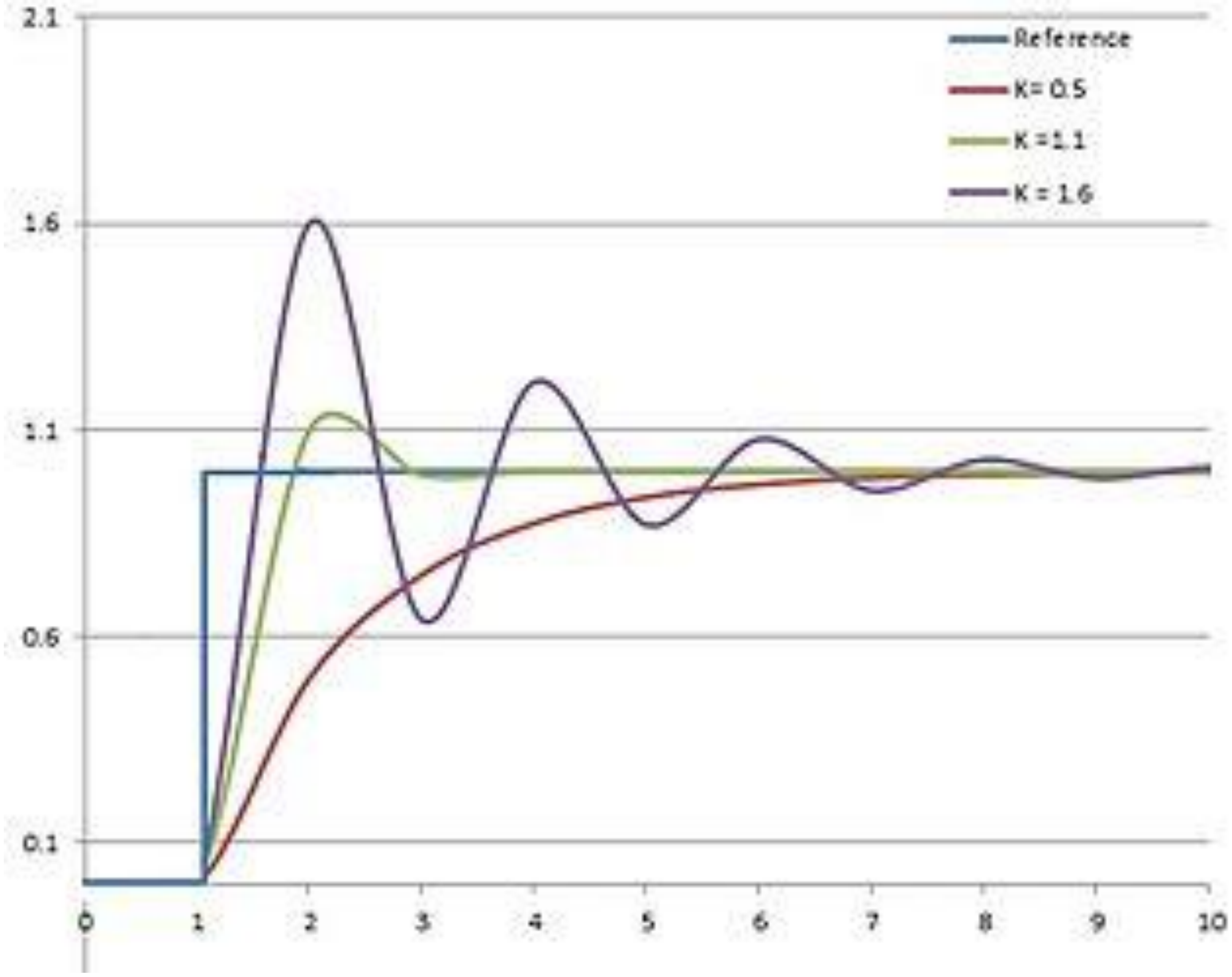
```python
def test_pid(P = 0.2,  I = 0.0, D= 0.0, L=100):
    pid = PID.PID(P, I, D)
    pid.SetPoint=0.0
    pid.setSampleTime(0.001)

    END = L
    feedback = 0
    feedback_list = []
    time_list = []
    setpoint_list = []

    for i in range(1, END):
        pid.update(feedback)
        output = pid.output
        if pid.SetPoint > 0:
            feedback += (output - (1/i))
        if i>9:
            pid.SetPoint = 1
        time.sleep(0.02)

        feedback_list.append(feedback)
        setpoint_list.append(pid.SetPoint)
        time_list.append(i)
```

```python
time_sm = np.array(time_list)
time_smooth = np.linspace(time_sm.min(), time_sm.max(), 300)

# feedback_smooth = spline(time_list, feedback_list, time_smooth)
# Using make_interp_spline to create BSpline
helper_x3 = make_interp_spline(time_list, feedback_list)
feedback_smooth = helper_x3(time_smooth)
```

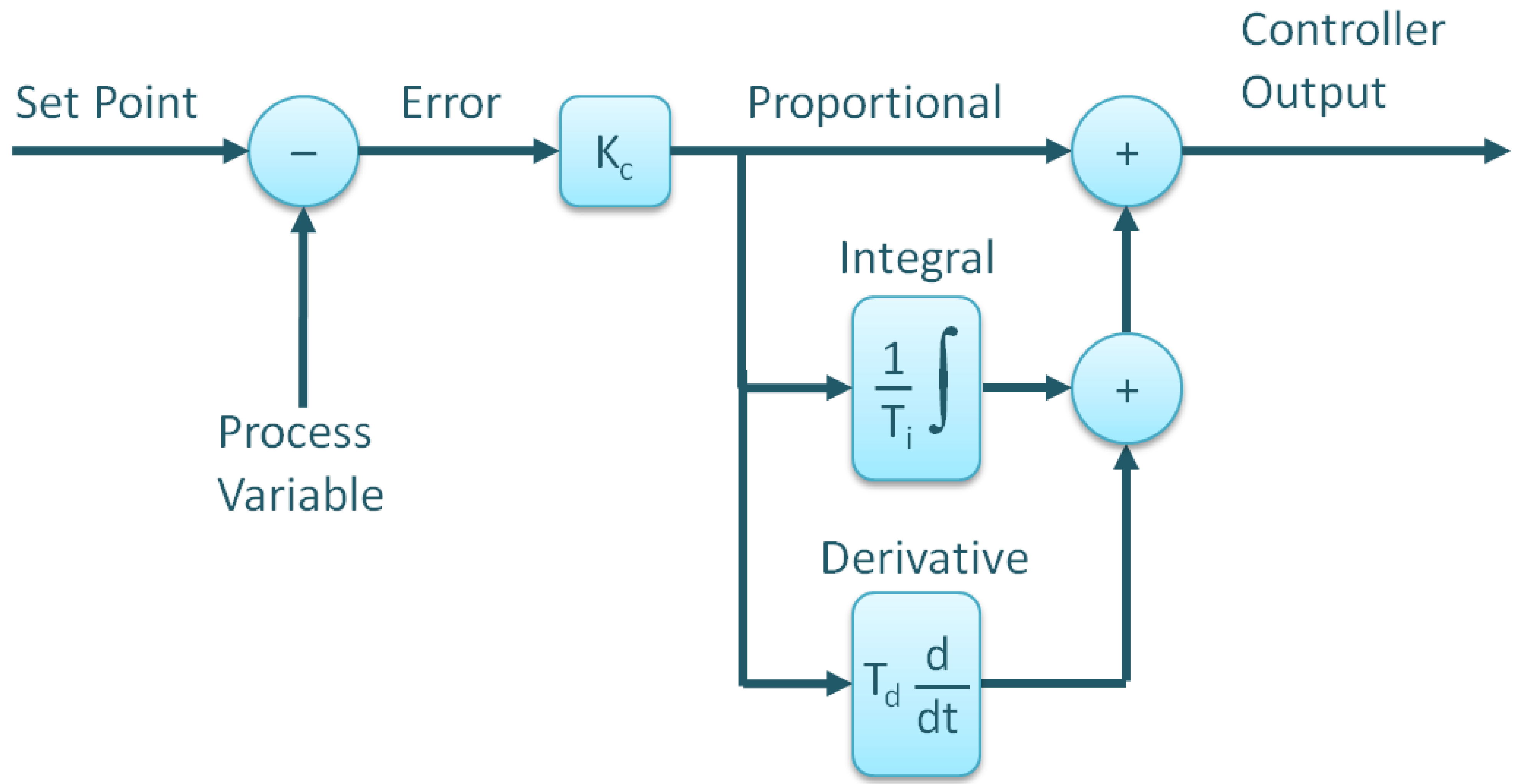
```
plt.plot(time_smooth, feedback_smooth)
plt.plot(time_list, setpoint_list)
plt.xlim((0, L))
plt.ylim((-0.2, 1.2))
plt.xlabel('time (s)')
plt.ylabel('PID (PV)')
plt.title('TEST PID')
plt.grid(True)
plt.show()
```

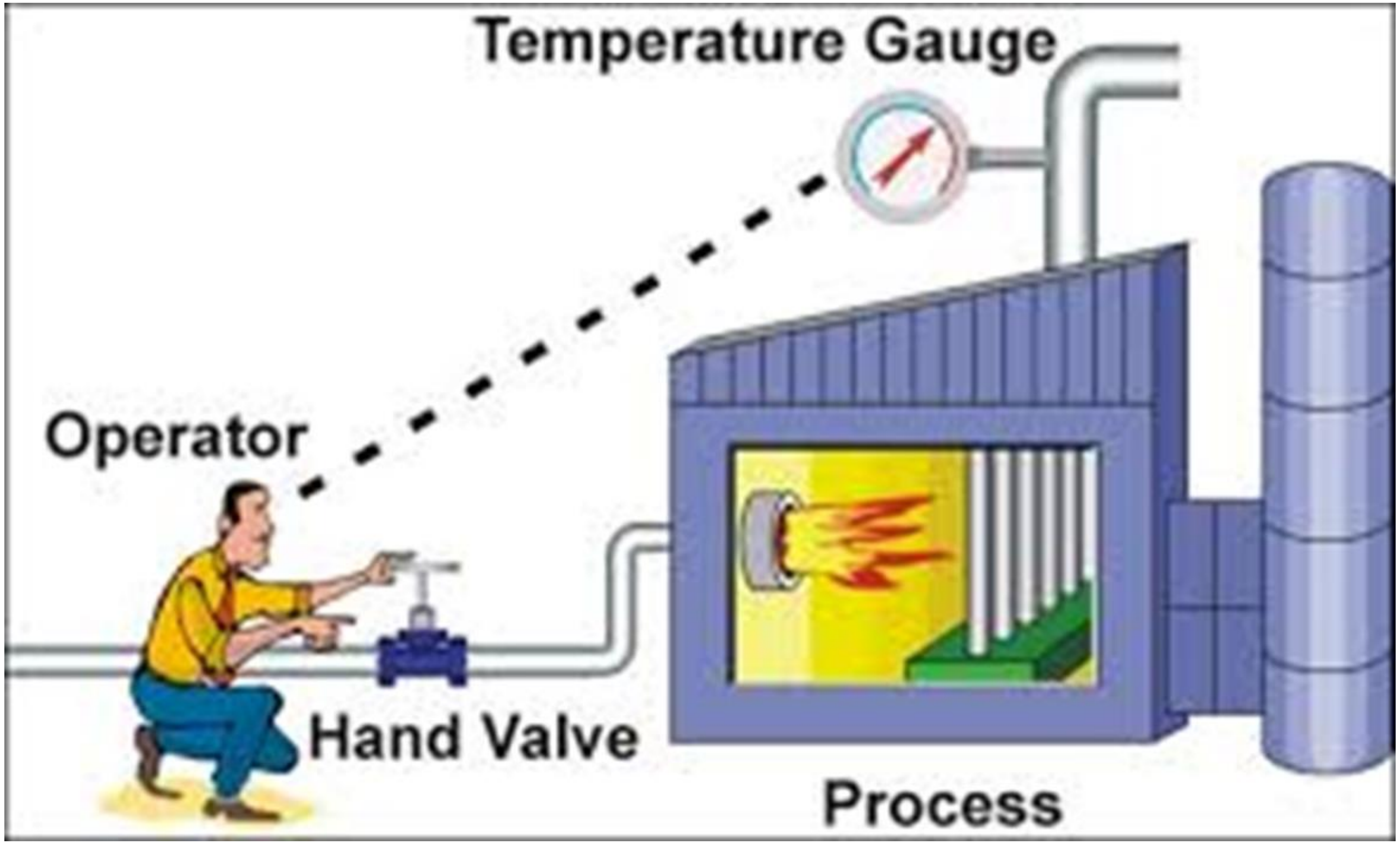


100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
%
Derivative response
Integral response
Proportional response
PV
SP
Time
InstrumentationTools.com



Reference
K= 0.5
K =1.1
K = 1.6
2.1
1.6
1.1
0.6
0.1
0
1
2
3
4
5
6
7
8
9
10


Set Point
−
Error
K_c
Proportional
+
Controller Output
Process Variable
Integral
$\frac{1}{T_i}\int$
+
Derivative
$T_d \frac{d}{dt}$


Temperature Gauge
Operator
Hand Valve
Process

Low Level Control Loop

SECTION 1

Control Loops

- Open Loops
- Closed Loops
- Proportional Controller
- Proportional-derivative Controller
- PID Controller

Problem: How do we set a motor to a given velocity?

Open Loop Controller

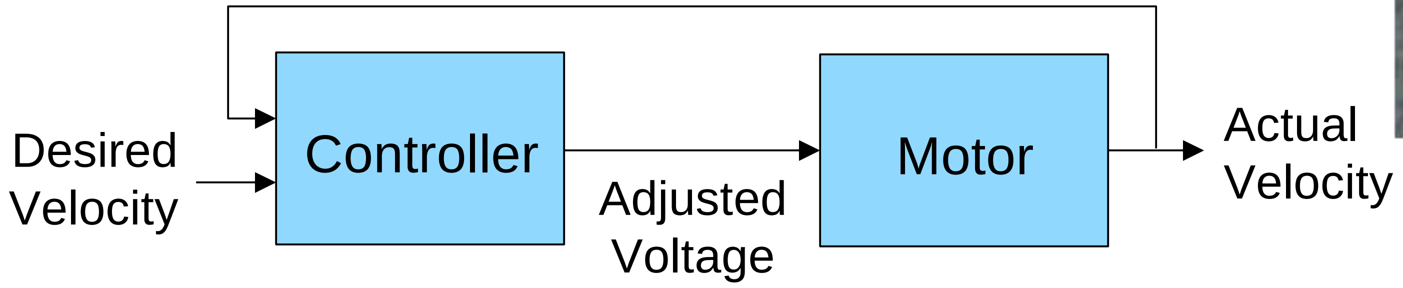
- Use trial and error to create some kind of relationship between velocity and voltage
- Changing supply voltage or drive surface could result in incorrect velocity

$$\begin{tikzcd}\text{Desired Velocity} \arrow[r] & \boxed{\text{Velocity To Volts}} \arrow[r] & \boxed{\text{Motor}} \arrow[r] & \text{Actual Velocity}\end{tikzcd}$$

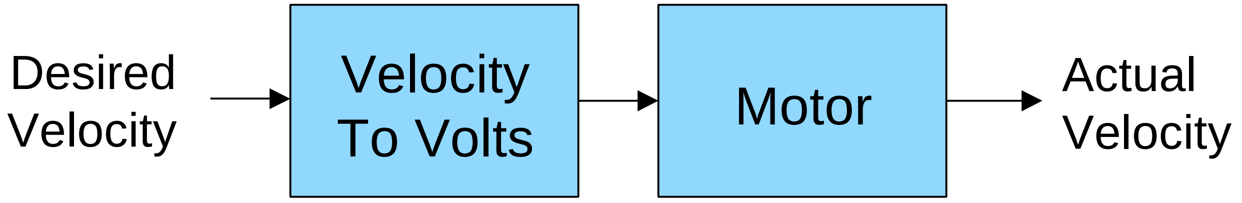
Problem: How do we set a motor to a given velocity?

Closed Loop Controller

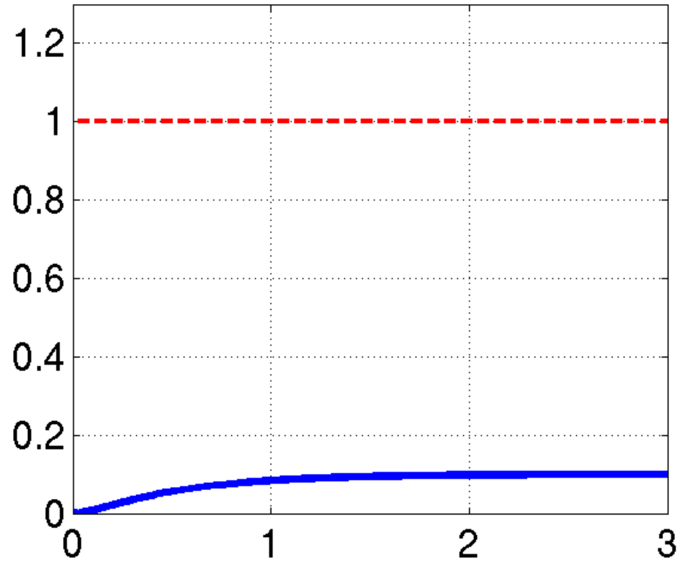
- Feedback is used to adjust the voltage sent to the motor so that the actual velocity equals the desired velocity
- Can use an optical encoder to measure actual velocity



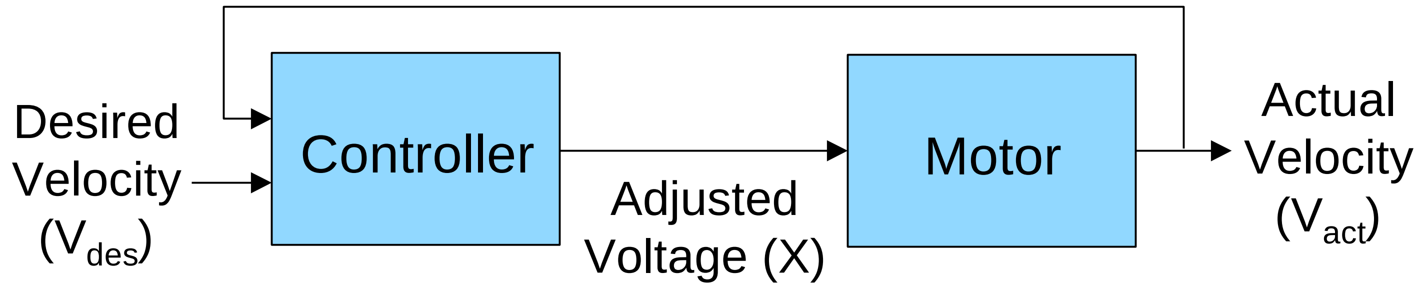
Step response with no controller



- Naive velocity to volts
- Model motor with several differential equations
- Slow rise time
- Stead-state offset

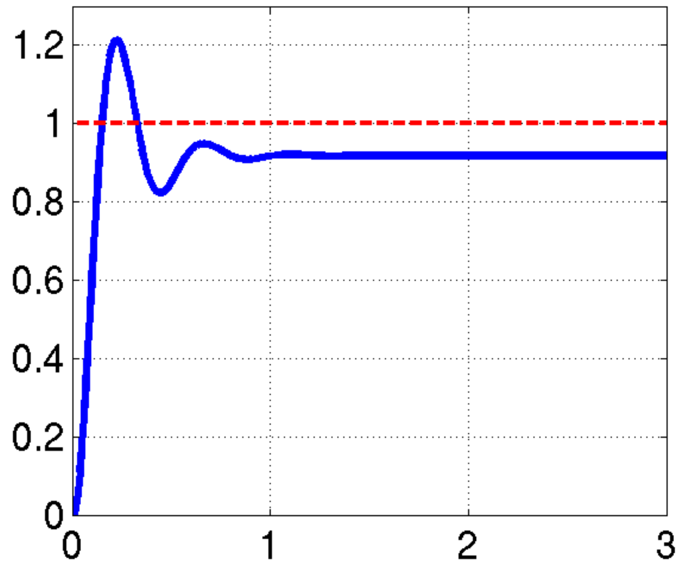


Step response with Proportional Controller

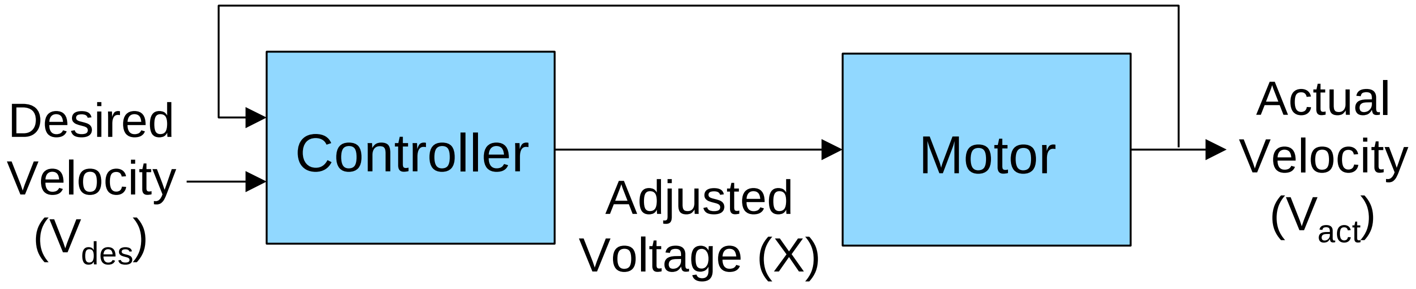


$$X = V_{des} + K_P \cdot \left(V_{des} - V_{act} \right)$$

- Big error big = big adj
- Faster rise time
- Overshoot
- Stead-state offset

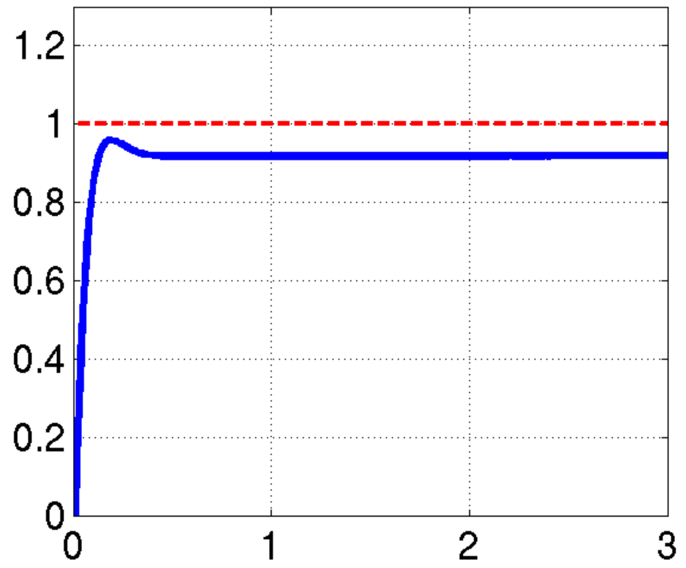


Step response with Proportional-derivative Controller

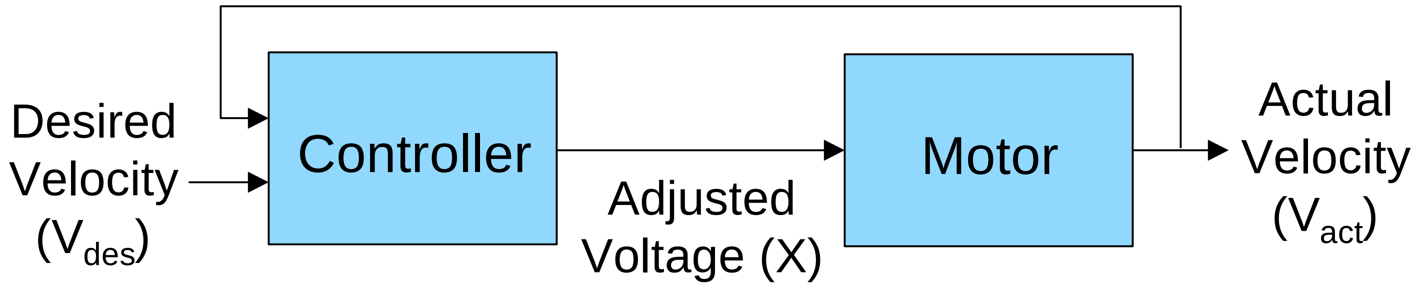


$$X = V_{des} + K_P e(t) - K_D \frac{de(t)}{dt}$$

- When approaching desired velocity quickly, de/dt term counteracts proportional term slowing adjustment
- Faster rise time
- Reduces overshoot

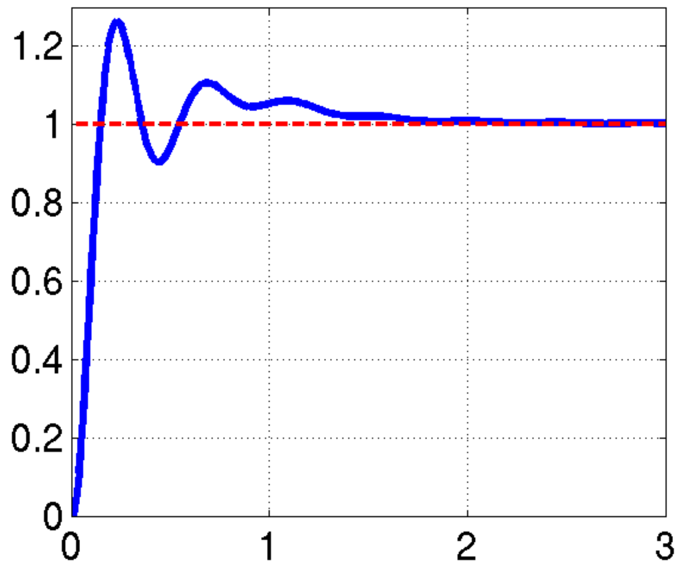


Step response with Proportional-Integral Controller

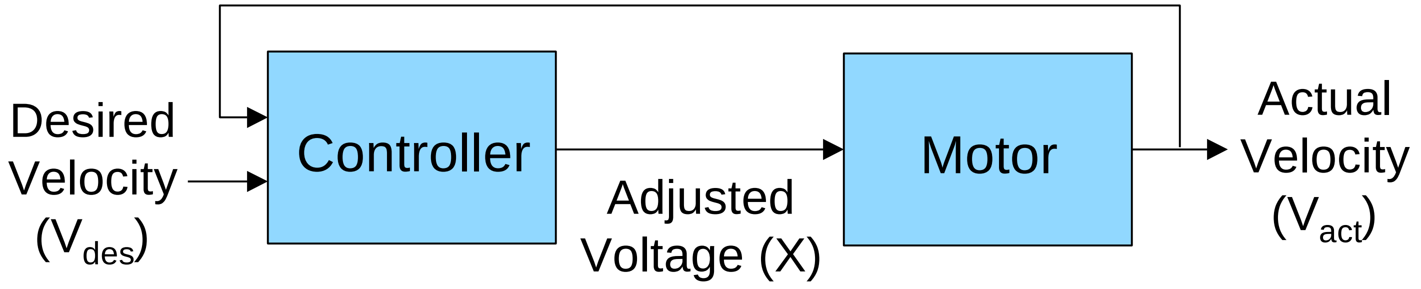


$$X = V_{des} + K_P e(t) - K_I \int e(t)\ dt$$

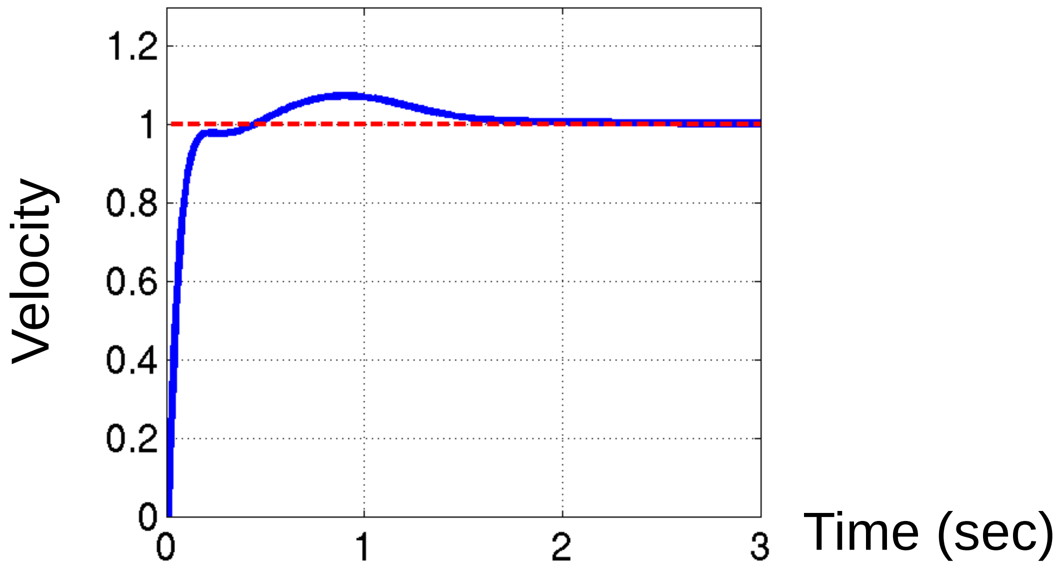
- Integral term eliminates accumulated error
- Increases overshoot



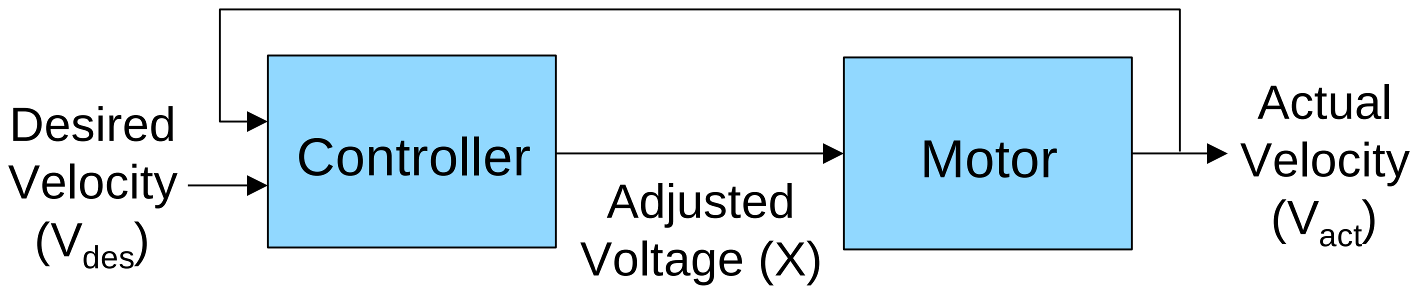
Step response with PID controller



$$X = V_{des} + K_P e(t) + K_I \int e(t)\, dt - K_D \frac{de(t)}{dt}$$



Choosing and tuning a controller

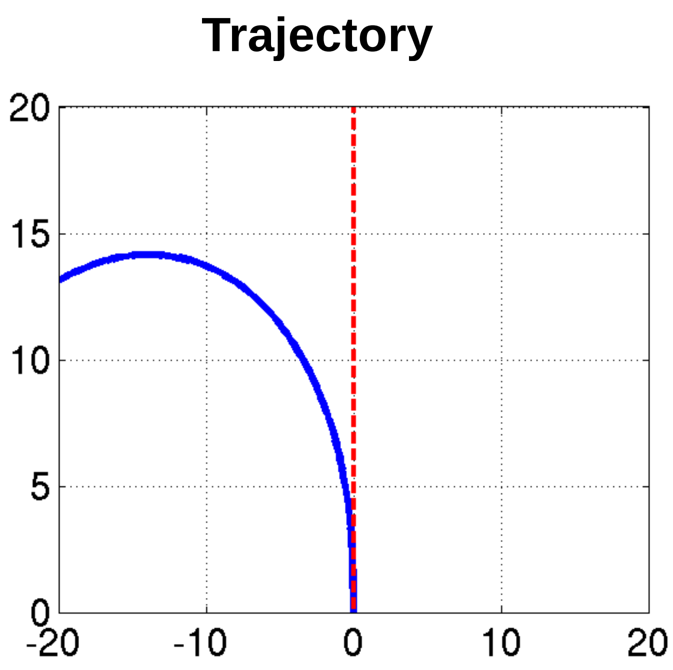


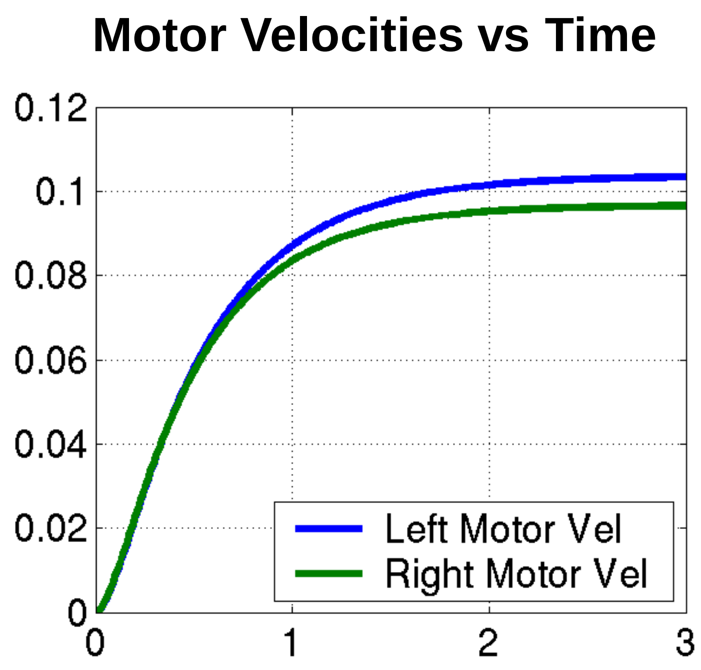
- Use the simplest controller which achieves the desired result
- Tuning PID constants is very tricky, especially for integral constants
- Consult the literature for more controller tips and techniques

Keep in a Straight Line

SECTION 1

Problem: How do we make our robots go in a nice straight line?

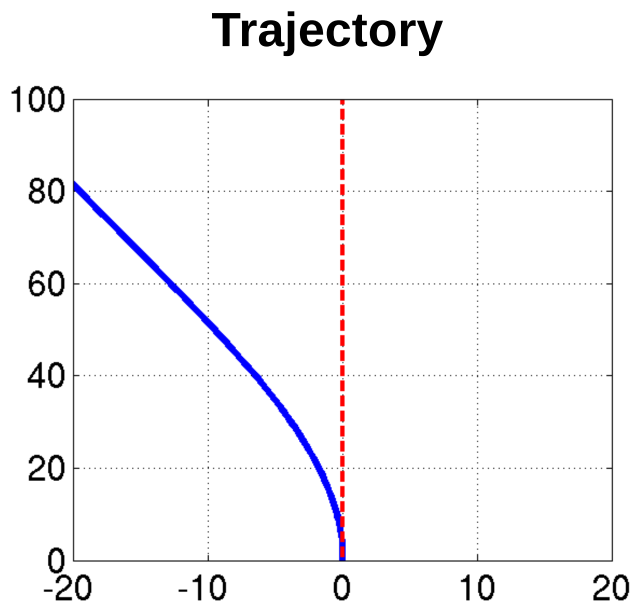


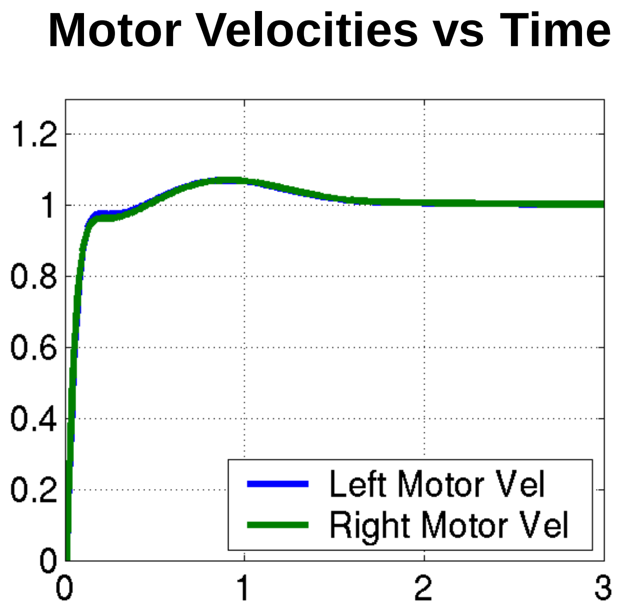


Model differential drive with slight motor mismatch

- With an open loop controller, setting motors to same velocity results in a less than straight trajectory

Problem: How do we make our robots go in a nice straight line?

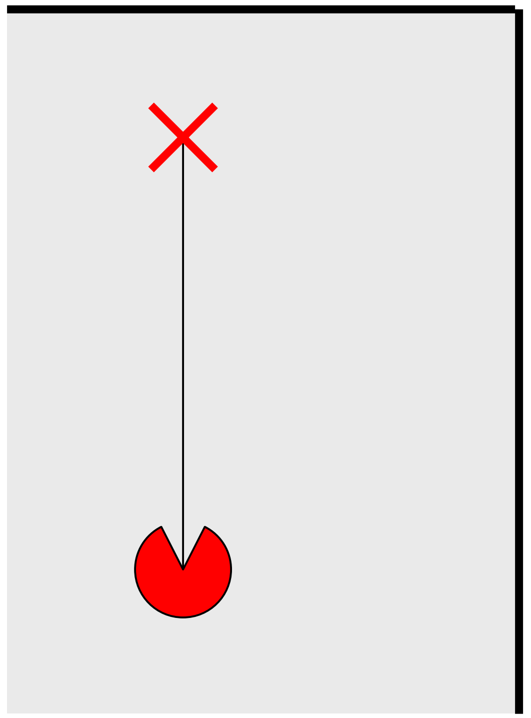




- With an independent PID controller for each motor, setting motors to same velocity results in a straight trajectory but not necessarily **straight ahead**!

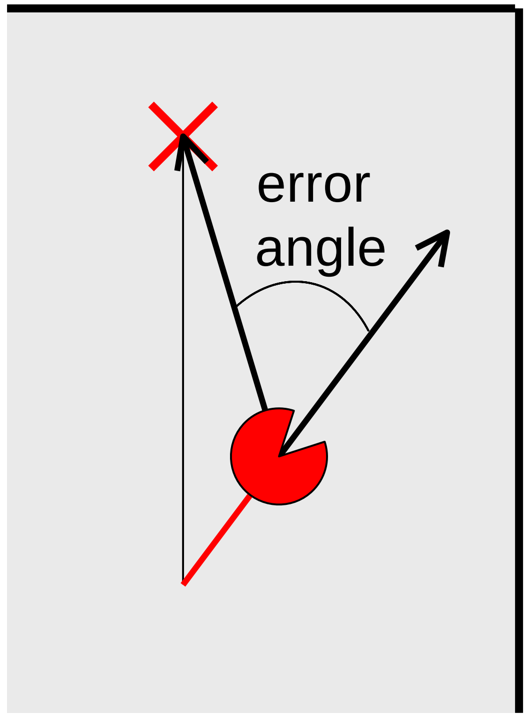
Problem: How do we make our robots go in a nice straight line?

- Need to couple drive motors
  - Use low-level PID controllers to set motor velocity and a high-level PID controller to couple the motors
  - Use one high-level PID controller which uses odometry or even image processing to estimate error



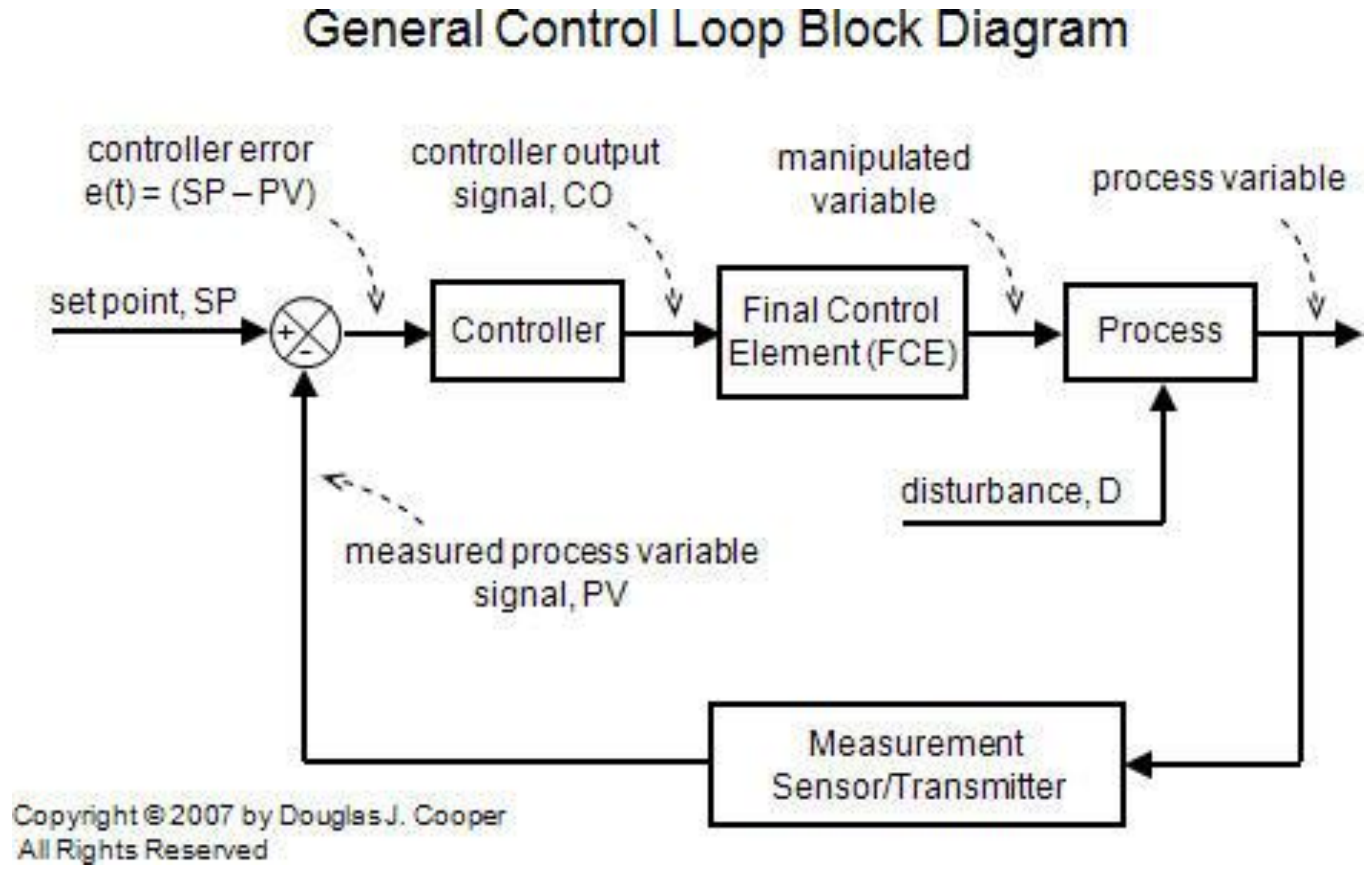
Problem: How do we make our robots go in a nice straight line?

- Need to couple drive motors
  - Use low-level PID controllers to set motor velocity and a high-level PID controller to couple the motors
  - Use one high-level PID controller which uses odometry or even image processing to estimate error



Proportional-only Control

SECTION 1


General Control Loop Block Diagram
controller error
e(t) = (SP – PV)
controller output
signal, CO
manipulated
variable
process variable
set point, SP
+
-
Controller
Final Control
Element (FCE)
Process
disturbance, D
measured process variable
signal, PV
Measurement
Sensor/Transmitter
Copyright © 2007 by Douglas J. Cooper
All Rights Reserved

General Control Loop

A sensor measures and transmits the current value of the process variable, PV, back to the controller (the 'controller wire in')

- Controller error at current time t is computed as set point minus measured process variable, or e(t) = SP – PV
- The controller uses this e(t) in a control algorithm to compute a new controller output signal, CO
- The CO signal is sent to the final control element (e.g. valve, pump, heater, fan) causing it to change (the 'controller wire out')
- The change in the final control element (FCE) causes a change in a manipulated variable
- The change in the manipulated variable (e.g. flow rate of liquid or gas) causes a change in the PV

P-only Control Algorithm

The P-Only controller computes a CO action every loop sample time T as:

$$CO = CO_{bias} + Kc \cdot e(t)$$

Where:

- CO_{bias} = controller bias or null value
- Kc = controller gain, *a tuning parameter*
- e(t) = controller error = SP – PV
- SP = set point
- PV = measured process variable

P-only Control

- A variation of Proportional Integral Derivative (PID) control is to use only the proportional term as a P-only control. The value of the controller output *u(t)* is fed into the system as the manipulated variable input. (PV: Process Variable, SP: Set Point)

$$u(t) = u_{bias} + K_c\,(SP - PV) = u_{bias} + K_c\,e(t)$$

P-only Control

- The u_{bias} term is a constant that is typically set to the value of $u(t)$ when the controller is first switched from manual to automatic mode. This gives "bumpless" transfer if the error is zero when the controller is turned on. The one tuning value for a P-only controller is the controller gain, K_c. The value of K_c is a multiplier on the error and a higher value makes the controller more aggressive at responding to any error away from the set point. The set point (SP) is the target value and process variable (PV) is the measured value that may deviate from the desired value. The error from the set point is the difference between the SP and PV and is defined as $e(t)=SP-PV$.

P-only Control

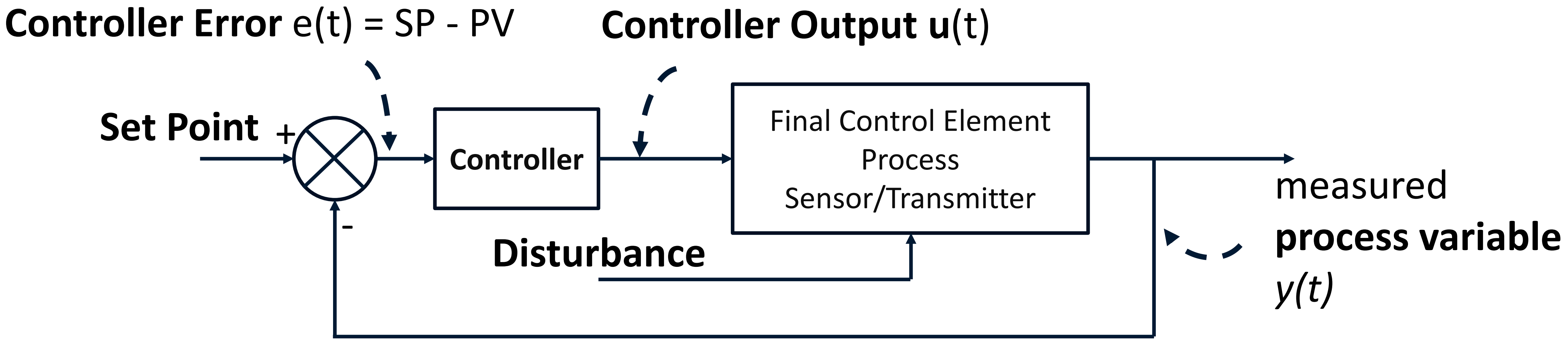
- P-only control is needed for integrating processes (e.g. tank level control with no outlet flow). If used on non-integrating processes there may be persistent offset between the desired set point and process variable with a P-only controller. Integral action is typically used to remove offset (see **PI Control**).

The process Designer’s “Process”

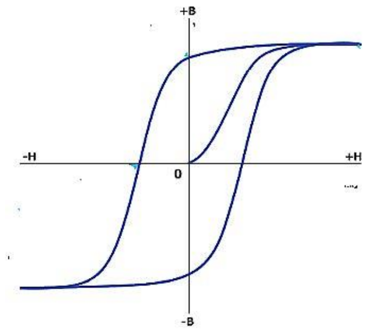
- For new installations, the dynamic behaviors of valves and sensors should be carefully considered prior to purchase.
- If undesirable behavior in an existing loop can be traced to certain equipment, then consider relocation or replacement.
- A final control element and sensor should start to respond quickly (add little dead time) and complete the response quickly (have a small time constant)
- The qualifier “quickly” is relative to the overall time constant of the process (a θ_p of 9 min is large relative to a τ_p of 10 min but small relative to a τ_p of 1000 min)

Redefining “process” for Controller Design/Tuning

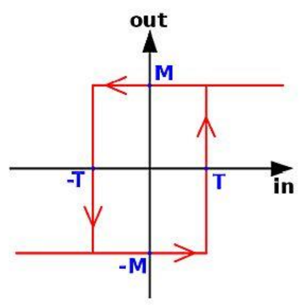
- The controller sends a signal out on one wire and receives the result as a change in measurement on another wire.
- From the controller’s view, the individual gains, time constants and dead time all lump into one overall dynamic behanior
- Here, “process dynamics” refers to these combined behaviors



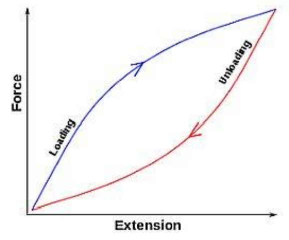
Hysteresis curves
Soft magnetic material



Schmitt trigger behavior

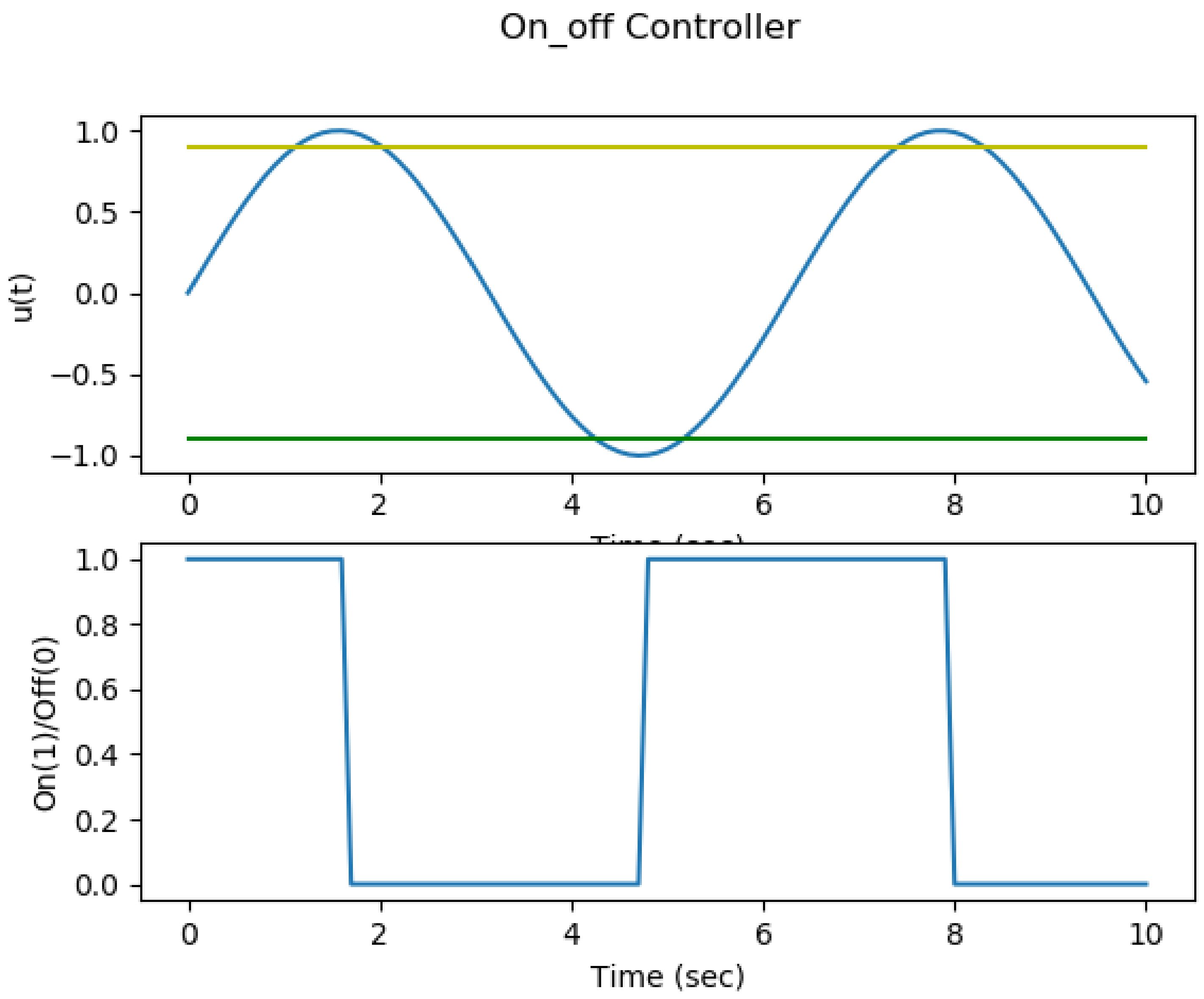


Rubber band behavior



On/Off – The Simplest Controller

- For on/off control, the final control element is either completely open/on/maximum or closed/off/minimum.
- To protect the final control element from wear, a dead band or upper and a lower set point is used.
- As long as the measured variable remains between these limits, no changes in control action are made.


On_off Controller
u(t)
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
Time (sec)
On(1)/Off(0)
1.0
0.8
0.6
0.4
0.2
0.0
Time (sec)

```python
from pylab import *
t = linspace(0, 10, 101)

def f(a, cL, cU):
    b = []
    y = 0
    for i in range(len(a)-1):
        x = a[i+1]-a[i]
        if (x < 0):
            if a[i]>=cU: y=0
            b.append(y)
        else:
            if a[i]<=cL: y=1
            b.append(y)
    return b
```

```python
u = [sin(t[i]) for i in range(len(t))]
s = [u[i] for i in range(len(t))]
s.insert(0, -1)
p = f(s, -0.9, 0.9)
cUU = [0.9 for i in range(len(t))]
cLL = [-0.9 for i in range(len(t))]
fig, axs = plt.subplots(2)
fig.suptitle('On_off Controller')
axs[0].plot(t, u)
axs[0].plot(t, cUU, 'y')
axs[0].plot(t, cLL, 'g')
axs[1].plot(t, p)
axs[0].set_xlabel("Time (sec)")
axs[1].set_xlabel("Time (sec)")
axs[0].set_ylabel("u(t)")
axs[1].set_ylabel("On(1)/Off(0)")
show()
```

Usefulness of On/Off Control

- On/Off with dead band is useful for home appliances such as furnaces, air conditioners, ovens, and refrigerators.
- For most industrial applications, on/off is too limiting (think about riding in a car that has on/off cruise control.)

Intermediate Value Control and PID

- Industry requires controllers that permit tighter control with less oscillation in the measured process variable
- These algorithms:
  - compute a complete range of actions between full on/off
  - Require a final control element that can assume intermediate positions between full on/off
- Example final control elements include process valves, variable speed pumps and compressors, electronic heating elements.

Intermediate Value Control and PID

- Popular intermediate value controller is PID (proportional-integral-derivative) controller
- PID computes a controller output signal base on control error:

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_i} \int e(t)dt - K_c \tau_d \frac{dy}{dt}$$

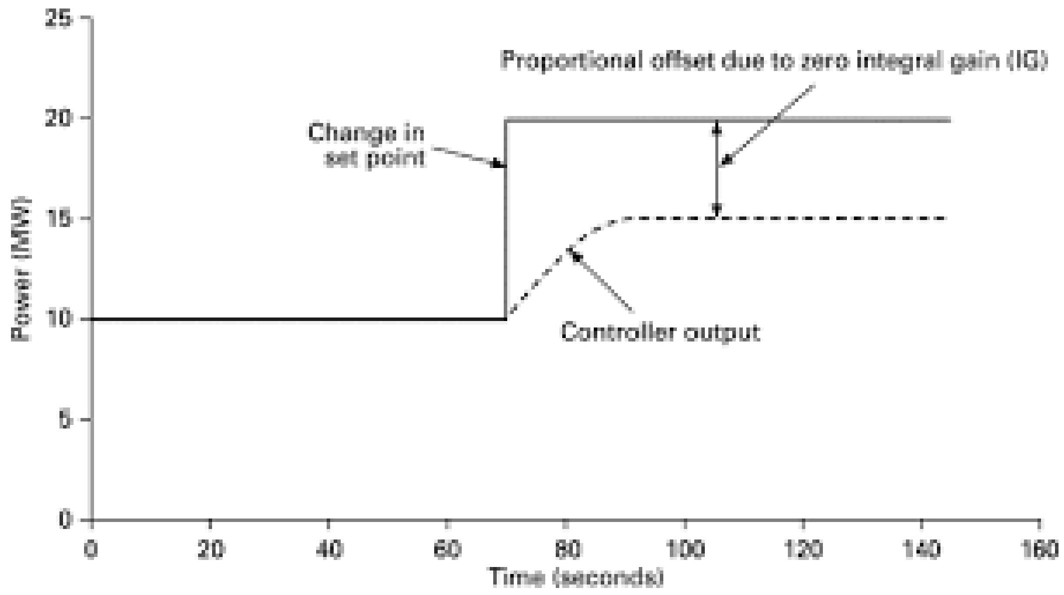
Where:

$y(t)$ = measured process variable
$u(t)$ = controller output signal
u_{bias} = controller bias or null value
$e(t)$ = controller error = $y_{setpoint} - y(t)$
K_c = controller gain (a stunning parameter)
t_i = controller reset time (a tuning parameter)
t_d = controller derivative time (a tuning parameter)

P-Only: The Simplest PID Controller

The proportional Controller

- The simplest PID controller is proportional or P-Only control
- It can computer the Intermediate Control value between 0-100%



The Gravity Drained Tanks Control Loop

- Measurement, computation and control action repeat every loop sample time:
  - A sensor measures the liquid level in the lower tank
  - This measurement is subtracted from the set point level to determine a control error; $e(t) = y_{setpoint} - y(t)$
  - The controller computes an output based on this error and it is transmitted to the valve, causing it to move
  - This causes the liquid flow rate into the top tank to change, which ultimately changes the level in the lower tank.
- The goal is to eliminate the controller error by making the measured level in the lower tank equal the set point level.

P-Only Control

- The controller computes an output signal every sample time:

$$u(t) = u_{bias} + K_c\, e(t)$$

Where

$u(t)$	= controller input
u_{bias}	= the controller bias
$e(t)$	= controller error = set point – measurement = $y_{setpoint} - y(t)$
K_c	= controller gain (a tuning parameter)

- Controller gain, $K_c \neq$ steady state process gain, K_p
- A larger K_c means a more active controller
  - Units of K_c are units of the controller (e.g., %) divided by the units
  - For gravity-drained tank, units of K_c are %/m
  - Remember that for this example, units of K_p are m/%

Design Level of Operation

- A controller is designed for a particular process behavior (or particular values of the FOPDT parameters K_p, τ_p, and θ_p)
- Real processes are nonlinear, so their behavior changes as operating level changes.
- Thus, a controller should be designed for a specific level of operation.

Collect Process Data at the Design Level

- The design value for the measured process variable is where the set point will be set during normal operation
- The design value for the important disturbance variables are their typical or expected levels during normal operation.
- Perform the dynamic test as near practical to the design level of the measured process variable when the disturbances are quiet and near their typical values.

Controller Gain, Kc, From Correlations

- P-Only control has one adjustable or tuning parameter, Kc

$$u(t) = u_{bias} + K_c\, e(t)$$

- Kc sets the activity of the controller to changes in error, e(t)
  - If Kc is small, the controller is sluggish
  - If Kc is large, the controller is aggressive

To determine K_c, use this controller design procedure:

- Generate dynamic process data at the design of operation
- Fit FOPDT dynamic model to this data
- Use the FOPDT model parameters in a correlation to compute initial estimates of K_c

Controller Gain, K_c, from Correlations

- Integral of time-weighted absolute error (ITAE) tuning correlations:
- If set point tracking (servo control) is the objective:

$$K_c = \frac{0.20}{K_p}\left(\frac{\tau_p}{\theta_p}\right)^{1.22}$$ Set point tracking

- If disturbance rejection (regulatory control) is the objective:

$$K_c = \frac{0.50}{K_p}\left(\frac{\tau_p}{\theta_p}\right)^{1.08}$$ Disturbance tracking

Controller Gain, K_c, from Correlations

- Correlations provide an initial guess or starting point only.
- Final tuning requires online trial and error because:
  - The designer may desire performance different from that provided by the correlation
  - The FOPDT model used for tuning may not match the actual dynamic behavior of the plant
  - Performance must be balanced over a range of nonlinear operation
  - Performance must be balanced for set point tracking and disturbance rejection.
- Note:
  - The designer defines “best” control performance
  - It is conservative to start with a small K_c value.

Understanding Controller Bias, u_{bias}

- Thought Experiment:
  - Consider P-Only cruise control where $u(t)$ is the flow of gas
  - Suppose velocity set point = measured velocity = 70 kph
  - Since $y(t) = y_{setpoint}$ then $e(t)=0$ and P-only controller is
  - $u(t) = u_{bias} + 0$
  - If u_{bias} were set to zero, then the flow of gas to the engine would be zero even though the car is going 70 kph
- If the car is going 70 kph, there clearly is a baseline flow of gas.
- This baseline controller output is the bias or null value.

Reverse Acting, Direct Acting and Control Action

- If K_p is positive and the process variable is too high, the controller decreases the controller output to correct the error.

  $\rightarrow$ The controller action is the reverse of the problem

  - When K_p is positive, the controller is reverse acting
  - When K_p is negative, the controller is direct acting

- Since K_c always has the same sign as K_p, then
  - K_p and K_c positive $\rightarrow$ reverse acting
  - K_p and K_c negative $\rightarrow$ direct acting

Offset – The Big Disadvantage of P-Only Control

- Big Advantage of P-Only Control:
  - Only one tuning parameter so it’s easy to find “best” tuning
- Big disadvantage:
  - the controller permits offset

Offset – The Big Disadvantage of P-Only Control

- Offset occurs under P-Only control when the set point and/or disturbances are at values other than that used as the design level of operation (that used to determine u_{bias})

$$u(t) = u_{bias} + K_c\, e(t)$$

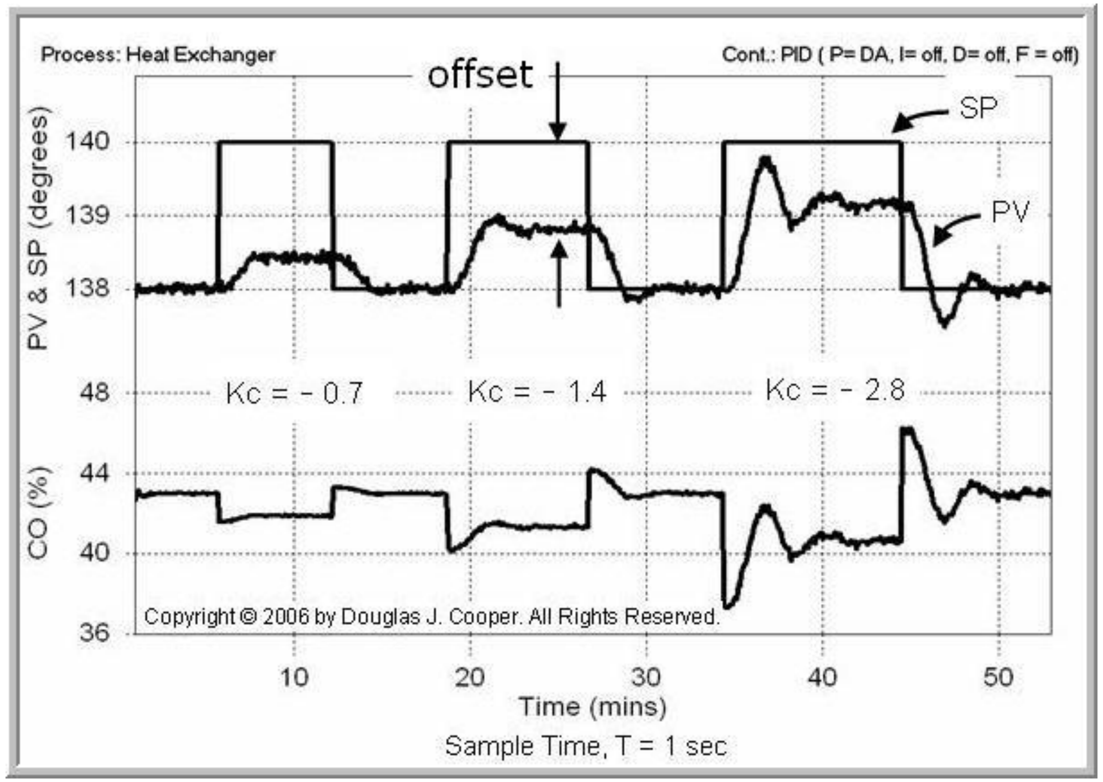
- How can the P-Only controller compute a value for $u(t)$ that is different from u_{bias} at steady state?
- The only way is if $e(t) \neq 0$

How do you get rid of Offset?

- Operate at the design conditions
- Use a more advanced control algorithm
    - PI instead of P-Only

Offset and Kc

- As Kc increases: → Offset decrease → Oscillatory behavior increases



Bumpless Transfer to Automatic

- A "bumpless transfer" achieves a smooth translation to closed loop by automatically:
  - Setting u_{bias} equal to the current controller output value
  - Setting the set point equal to the current measured process variable value
- So, when put in automatic, there is no controller error and the bias is properly set to produce no offset.
- As a result, no immediate control action is necessary that would "bump" the measured process variable.

Tuning Correlations

- A common tuning correlation for P-only control is the ITAE (Integral of Time-weighted Absolute Error) method. Different tuning correlations are provided for disturbance rejection (also referred to as *regulatory* control) and set point tracking (also referred to as *servo* control).

  - $K_c = \frac{0.20}{K_p} \left(\frac{\tau_p}{\theta_p}\right)^{1.22}$ Set point tracking

  - $K_c = \frac{0.50}{K_p} \left(\frac{\tau_p}{\theta_p}\right)^{1.08}$ Disturbance tracking

- The parameters K_c, τ_p, and θ_p are obtained by fitting dynamic input and output data to a **first-order plus dead-time (FOPDT) model**.