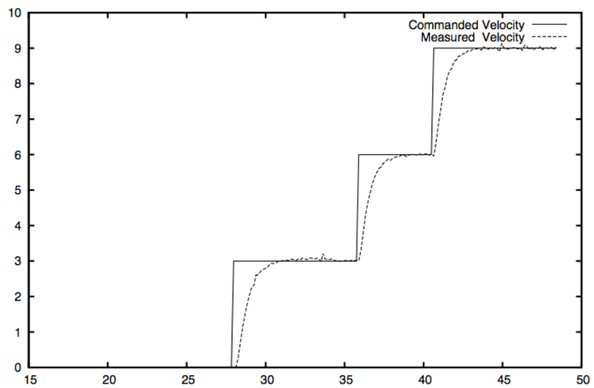
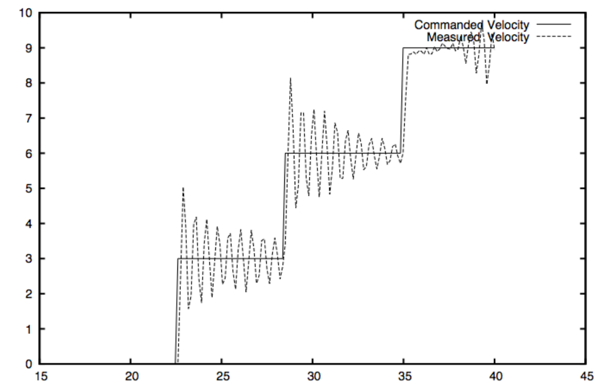



# Motor Control




RSS Lecture 3

Monday, 7 Feb 2011

Prof. Daniela Rus

(includes some material by Prof. Seth Teller)

Jones, Flynn & Seiger § 7.8.2

http://courses.csail.mit.edu/6.141/

# Today: Control

- Early mechanical examples
- Feed-forward and Feedback control
- Terminology
- Basic controllers:
  - Feed-Forward (FF) control
  - Bang-Bang control
  - Proportional (P) control
  - The D term: Proportional-Derivative (PD) control
  - The I term: Proportional-Integral (PI) control
  - Proportional-Integral-Derivative (PID) control
- Gain selection
- Applications

# The Role of Control

- Many tasks in robotics are defined by *achievement* goals
  - Go to the end of the maze
  - Push that box over here
- Other tasks in robotics are defined by *maintenance* goals:
  - Drive at 0.5m/s
  - Balance on one leg

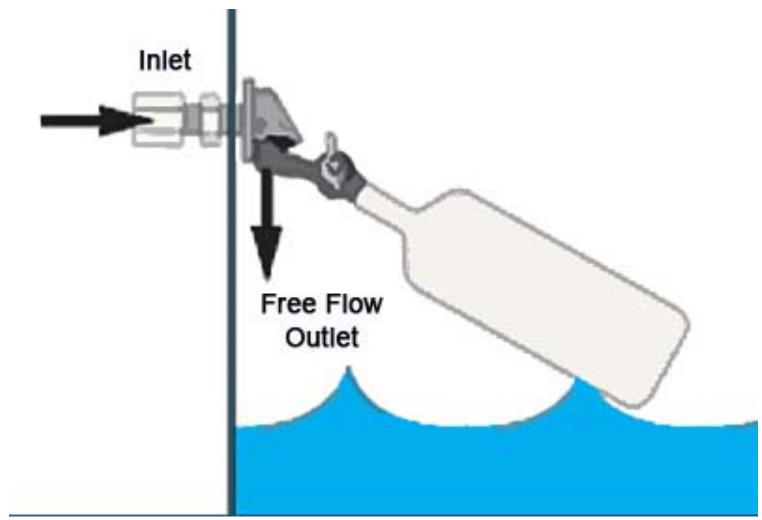
# The Role of Control

- *Control theory* is generally used for low-level maintenance goals
- General notions:
  - output = Controller(input)
  - output is control signal to actuator (e.g., motor voltage/current)
  - input is either goal state or goal state error (e.g., desired motor velocity)
- Controller is stateless

# What is the point of control?

- Consider any mechanism with adjustable DOFs*
(e.g. a valve, furnace, engine, car, robot…)

- Control is *purposeful variation* of these DOFs to achieve some specified *maintenance state*
  - Early mechanical examples: float valve, steam governor

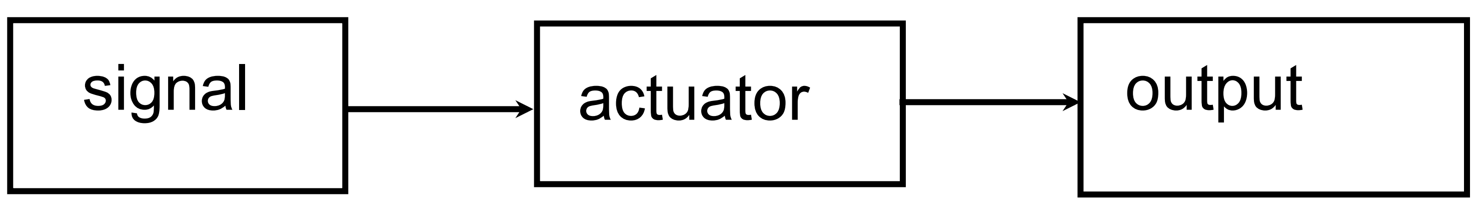



www.freshwatersystems.com

*DOFs = Degrees of Freedom

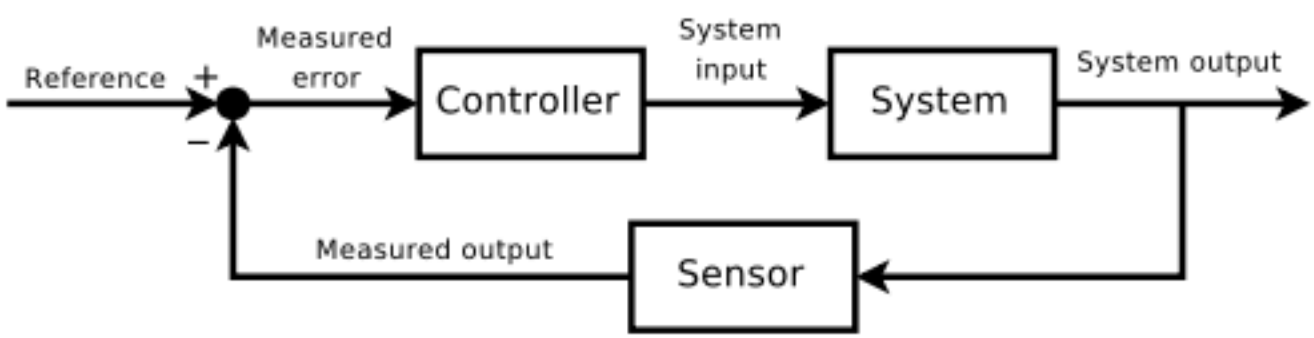
# Motor Control: Open Loop

- Give robot task with no concern for the environment
- Applications: ???
- Open loop: signal to action
- Not checked if correct action was taken
- Example: go forward for 15 secs, then turn left for 10 secs. Issues?



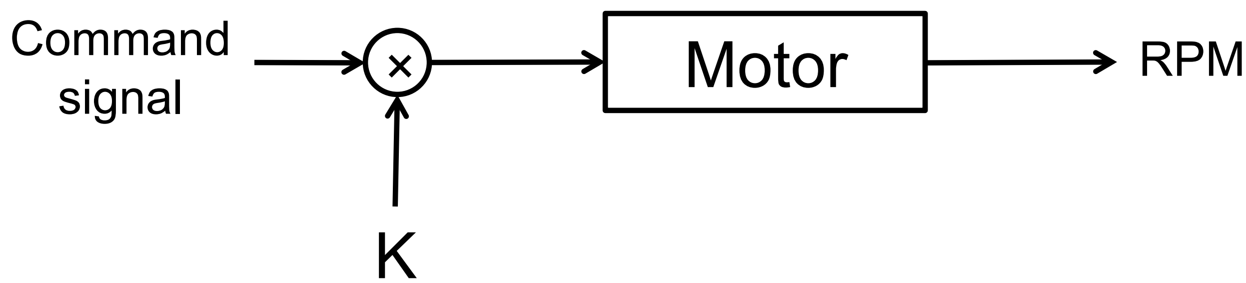
# Open loop (feed-forward) control

- Open loop controller:
  - output = FF(goal)
- Example.: motor speed controller (linear):
  - V = k * s
  - V is applied voltage on motor
  - s is speed
  - k is gain term (from calibration)
- Weakness:
  - Varying load on motor => motor may not maintain goal speed



# Feed-Forward (FF) Control

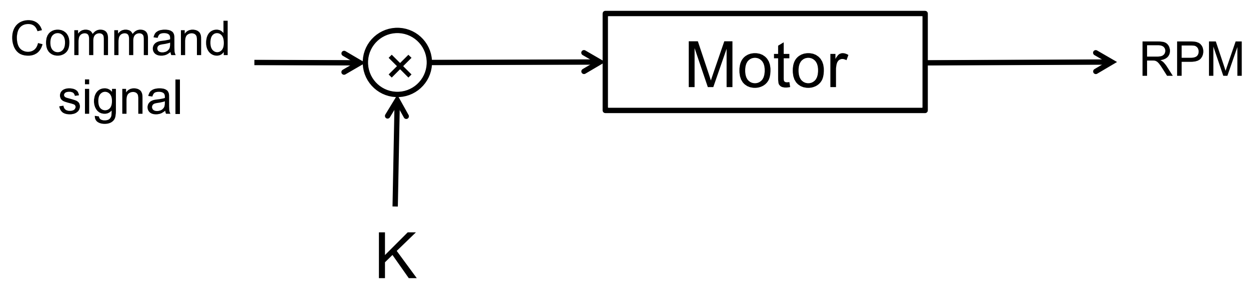
- Pass command signal from external environment directly to the *loaded element* (e.g., the motor)
- Command signal typically multiplied by a *gain* K




- … Where does the gain value K come from?
- Under what conditions will FF control work well?
- You will implement an FF controller in Lab 2

# Feed-Forward (FF) Control

- Pass command signal from external environment directly to the *loaded element* (e.g., the motor)
- Command signal typically multiplied by a *gain* K




- … Where does the gain value K come from?
  - Calibration (example: PWM = 0, PWM = 255)
- Under what conditions will FF control work well?
  - When the presented load is uniform and known
- You will implement an FF controller in Lab 2

# Feedback Control

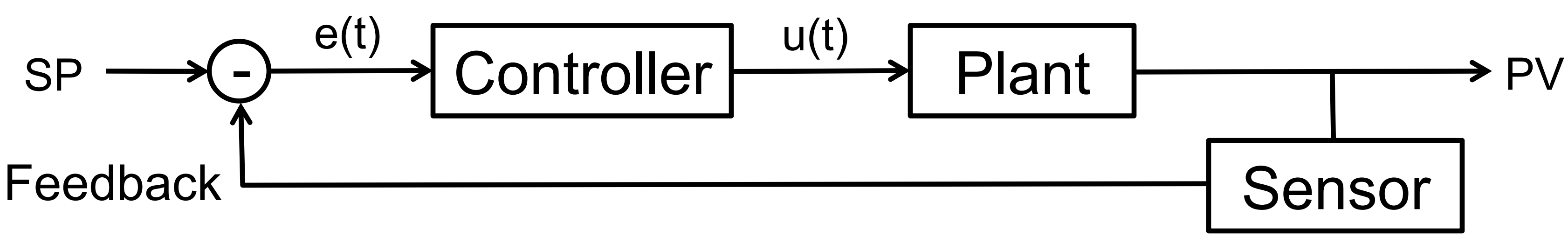
- Feedback controller:
  - output = FB(error)
  - error = goal state - measured state
  - controller attempts to minimize error
- Feedback control requires sensors:
  - Binary (at goal/not at goal)
  - Direction (less than/greater than)
  - Magnitude (very bad, bad, good)

# Example: Wall Following

- How would you use feedback control to implement a wall-following behavior in a robot?
- What sensors would you use, and would they provide magnitude and direction of the error?
- What will this robot's behavior look like?

# Feedback Control Terminology

- *Plant* **P**: process commanded by a *Controller*
- *Process Variable* **PV**: Value of some process or system quantity of interest (e.g. temperature, speed, force, …) as measured by a *Sensor*
- *Set Point** **SP**: Desired value of that quantity

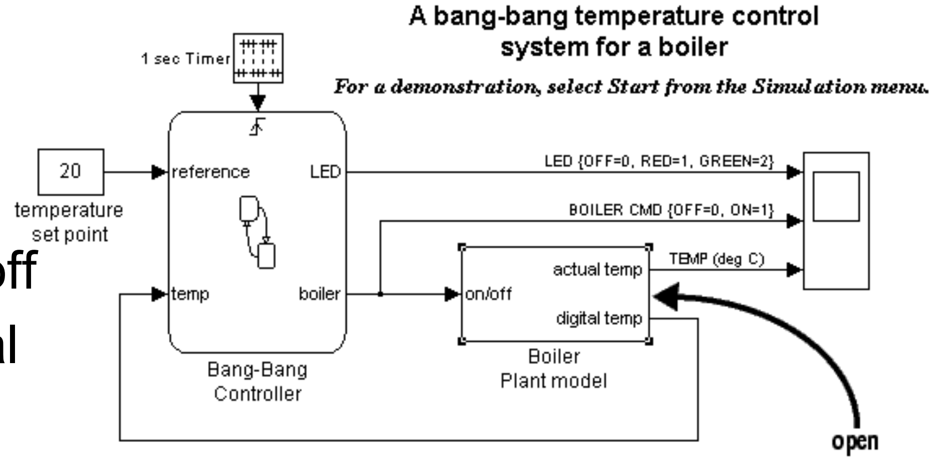


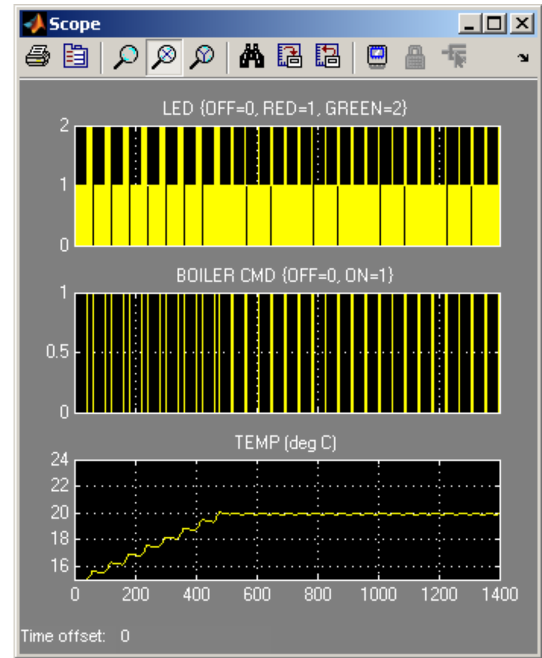

- *Error signal* **e(t)** = SP-PV: error in the process variable at time t, computed via *Feedback*
- *Control signal* **u(t)**: controller output (value of switch, voltage, PWM, throttle, steer angle, …)

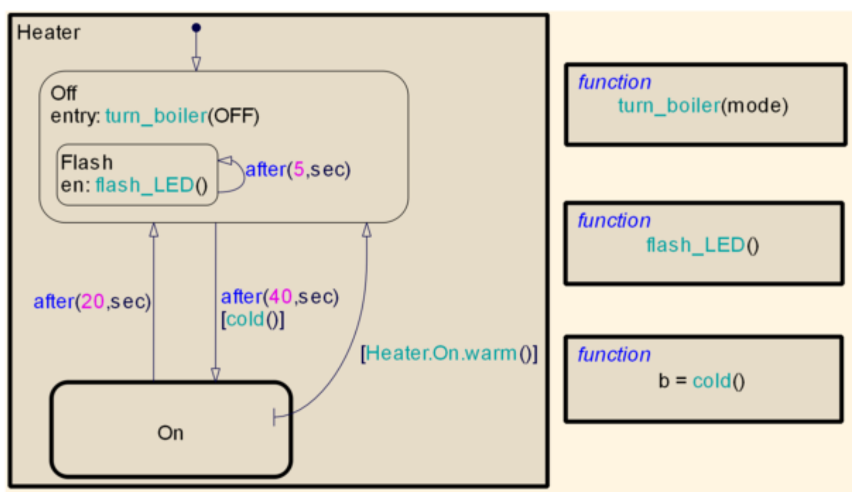
*Set point is sometimes called the "Reference"

# Bang-bang control

- Discrete on/off
- Furnace: goal temp = 70
- when temp < 70 BANG! Heat;
- when temp > 70 BANG! Stop the heat

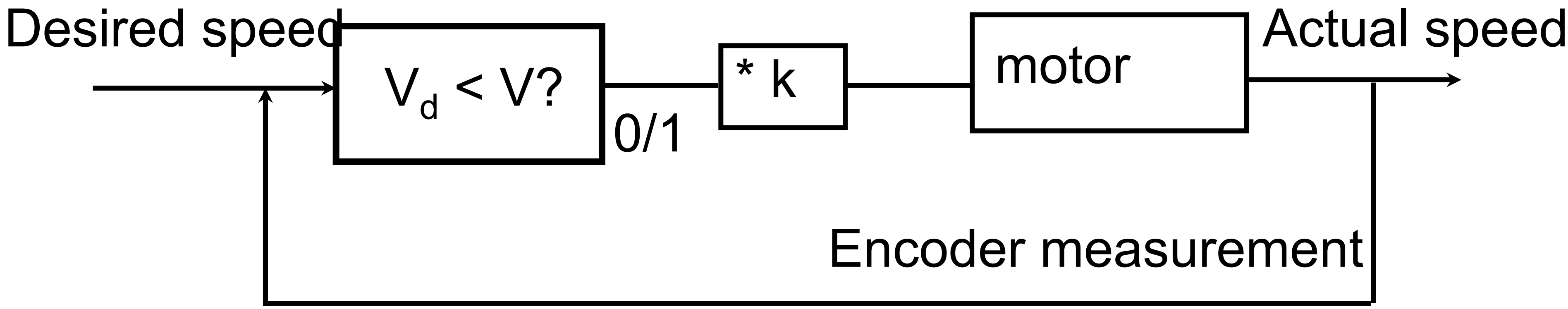


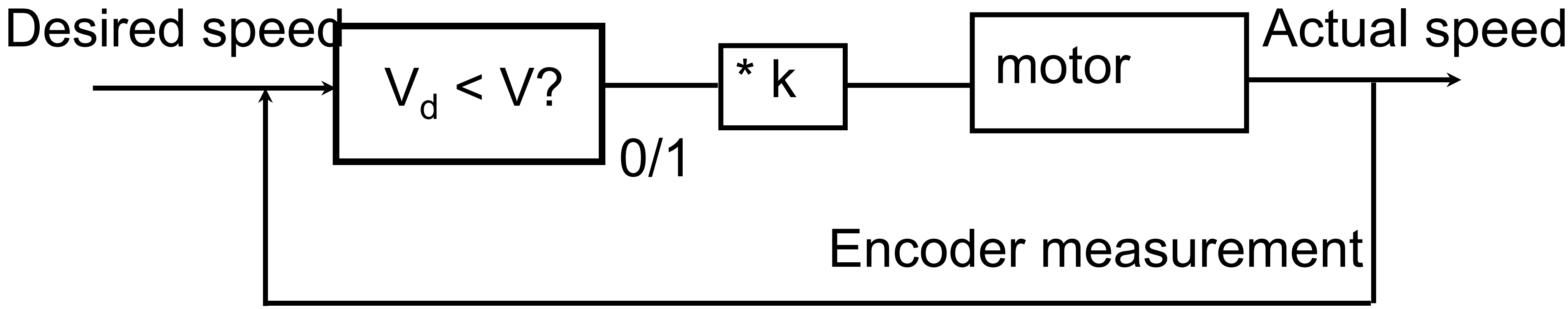

Example source: Mathworks

# Bang-bang control




O(t) =
O(t) =

# Bang-bang control

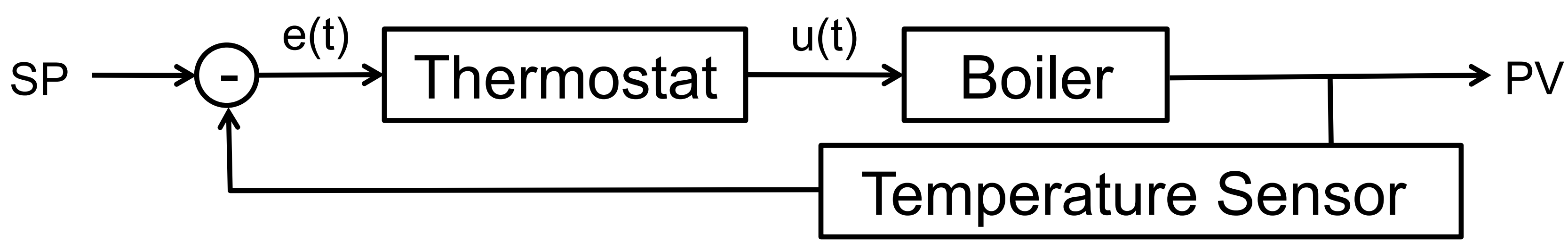



$O(t) = k$ if $v(t) < V_d$

$O(t) = 0$ otherwise

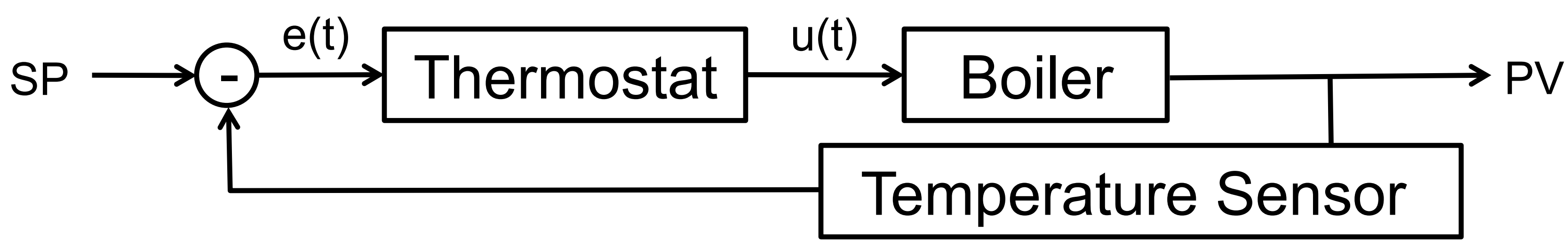
# Example: Home Heating System

- *Plant* P:
- *Process Variable* PV:
- *Controller:* *Sensor:*
- *Set Point* SP:
- *Control signal*:



# Example: Home Heating System

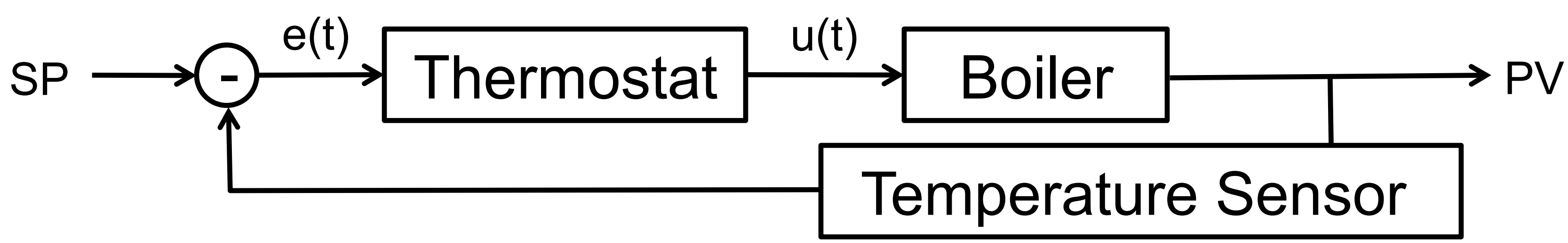
- *Plant* P: Boiler with on-off switch (1 = all on ; 0 = all off)
- *Process Variable* PV: Current home temperature
- *Controller:* Thermostat   *Sensor:* Thermometer
- *Set Point* SP: Thermostat setting (desired temp.)
- *Control signal*: Boiler on-off switch $u(t) \in \{0, 1\}$




How could the function **u(t)** be implemented?

# Example: Home Heating System

- *Plant* P: Boiler with on-off switch (1 = all on ; 0 = all off)
- *Process Variable* PV: Current home temperature
- *Controller:* Thermostat *Sensor:* Thermometer
- *Set Point* SP: Thermostat setting (desired temp.)
- *Control signal*: Boiler on-off switch $u(t) \in \{0, 1\}$

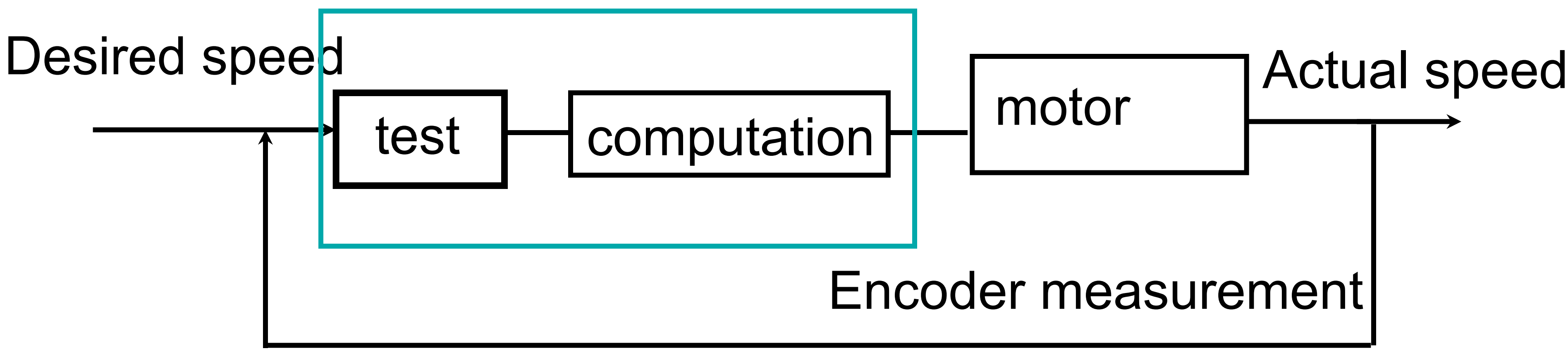



How could the function **u(t)** be implemented?

u(t) = **1** if e(t) > 0 [i.e., if SP > PV], **0** otherwise

# Motor Control: closed loop

- A way of getting a robot to achieve and maintain a goal state by constantly comparing current state with goal state.
- Use sensor for feedback

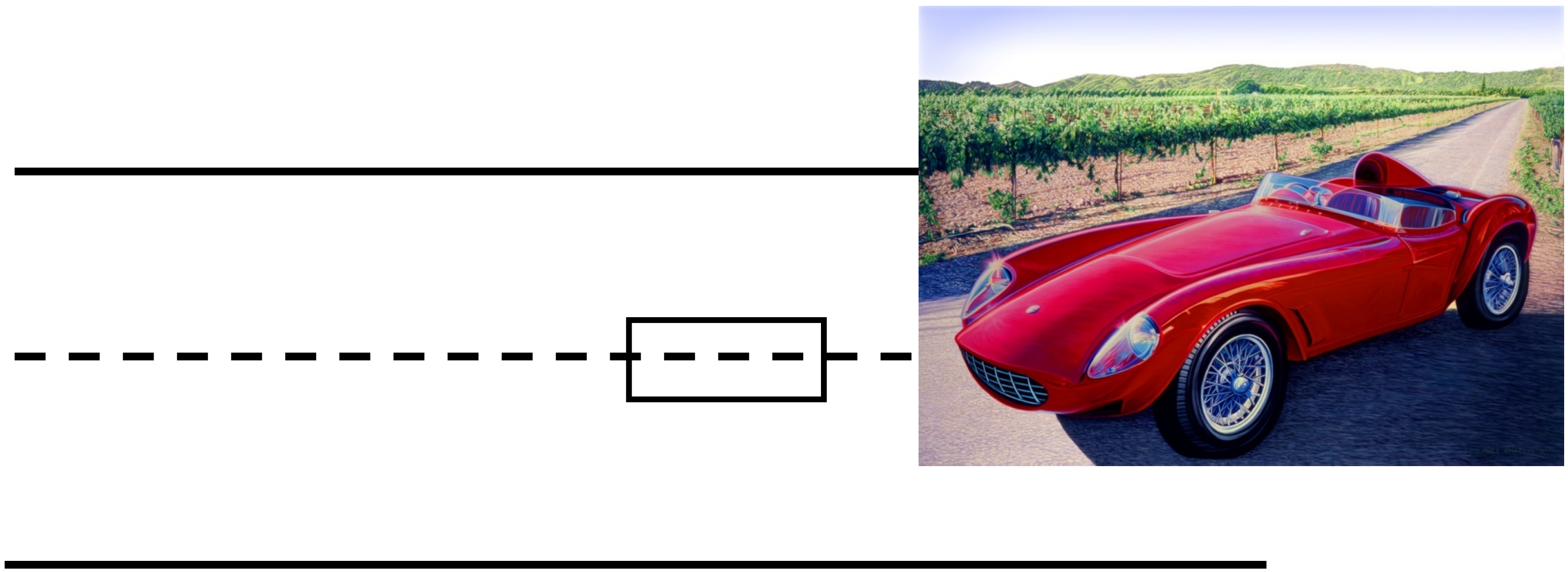


# Motor Control: PID

- *Control theory is the science that studies the behavior of control systems*
- *CurrentState - DesiredState = Error*
- Main types of simple linear controllers:
  - P: proportional control
  - PD: proportional derivative control
  - PI: proportional integral control
  - PID: proportional integral derivative control

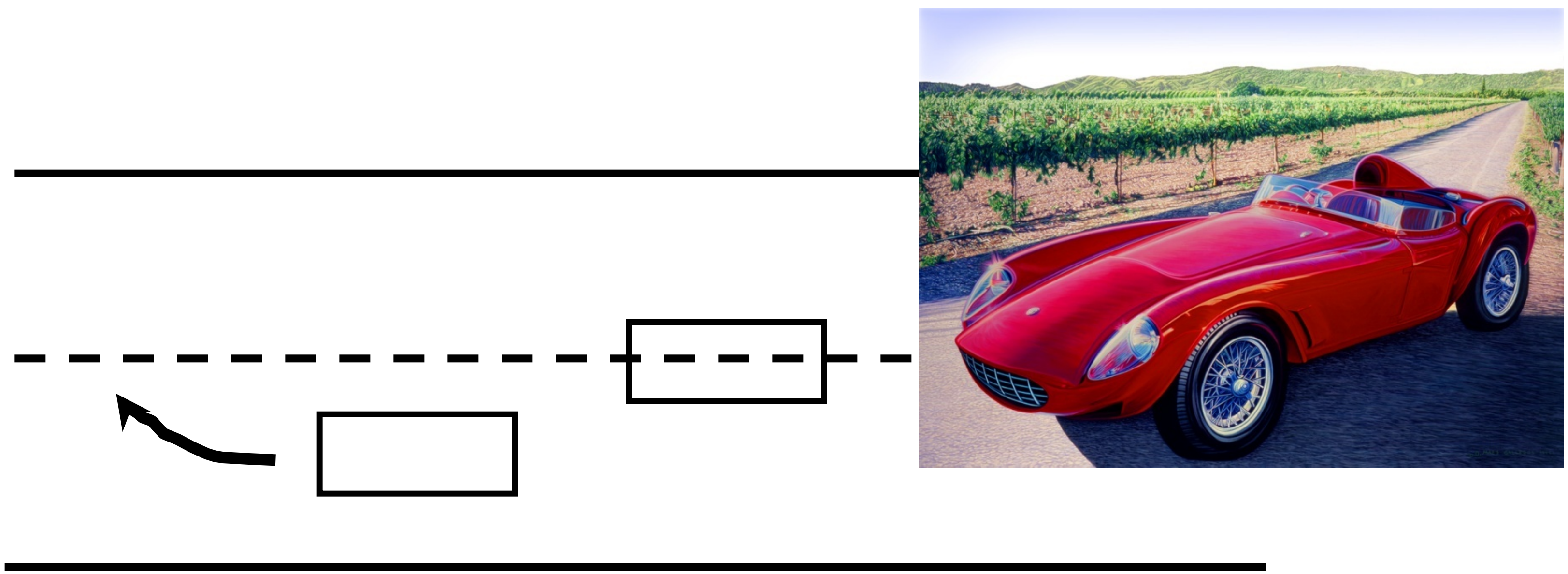
# Example: driving

- Steer a car in the center of a lane



# Example: driving

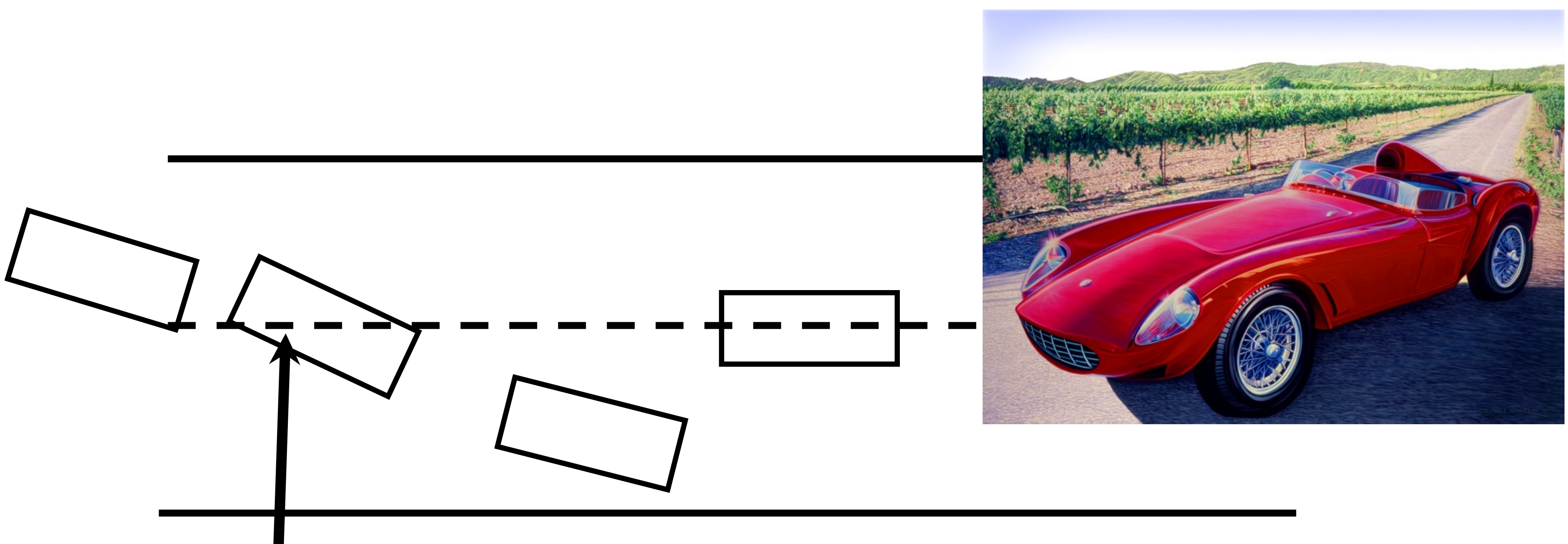
- Steer a car in the center of it lane



Observed error: distance off from center line

# Example: driving

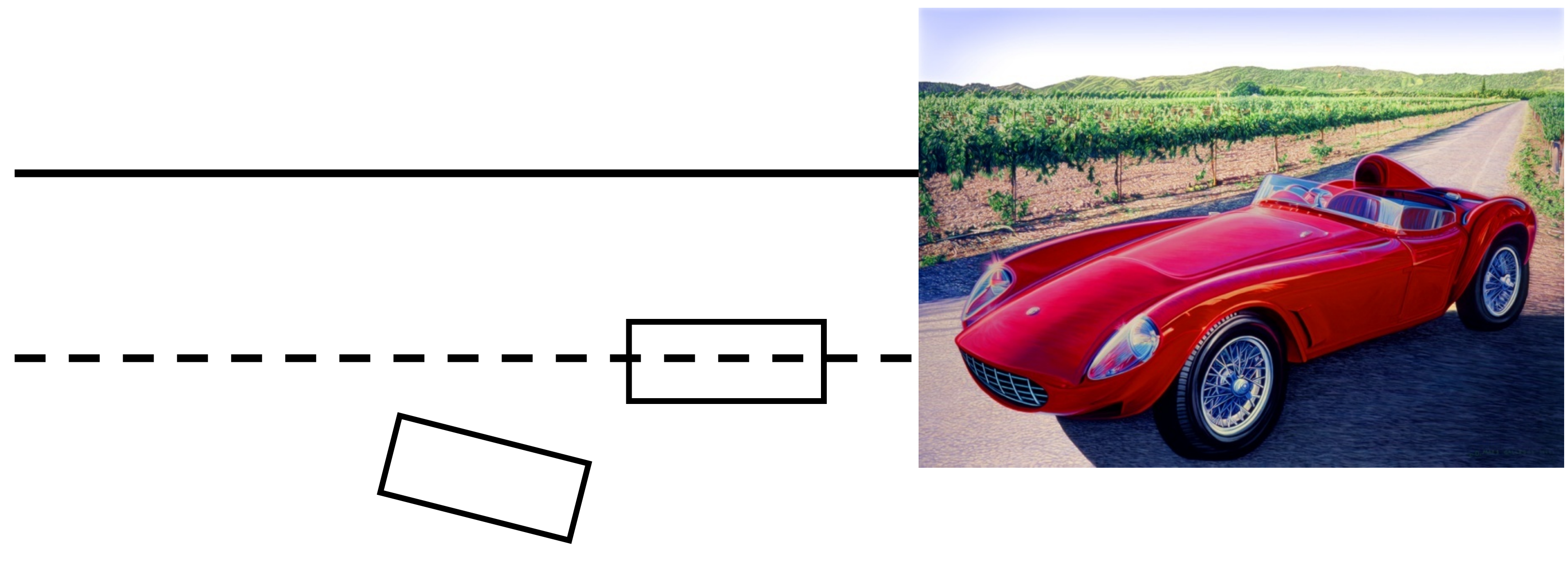
- Steer a car in the center of it lane



Error is zero but how is the car pointed?
What will this do to the car?
P controller is happy on line independent of orientation!

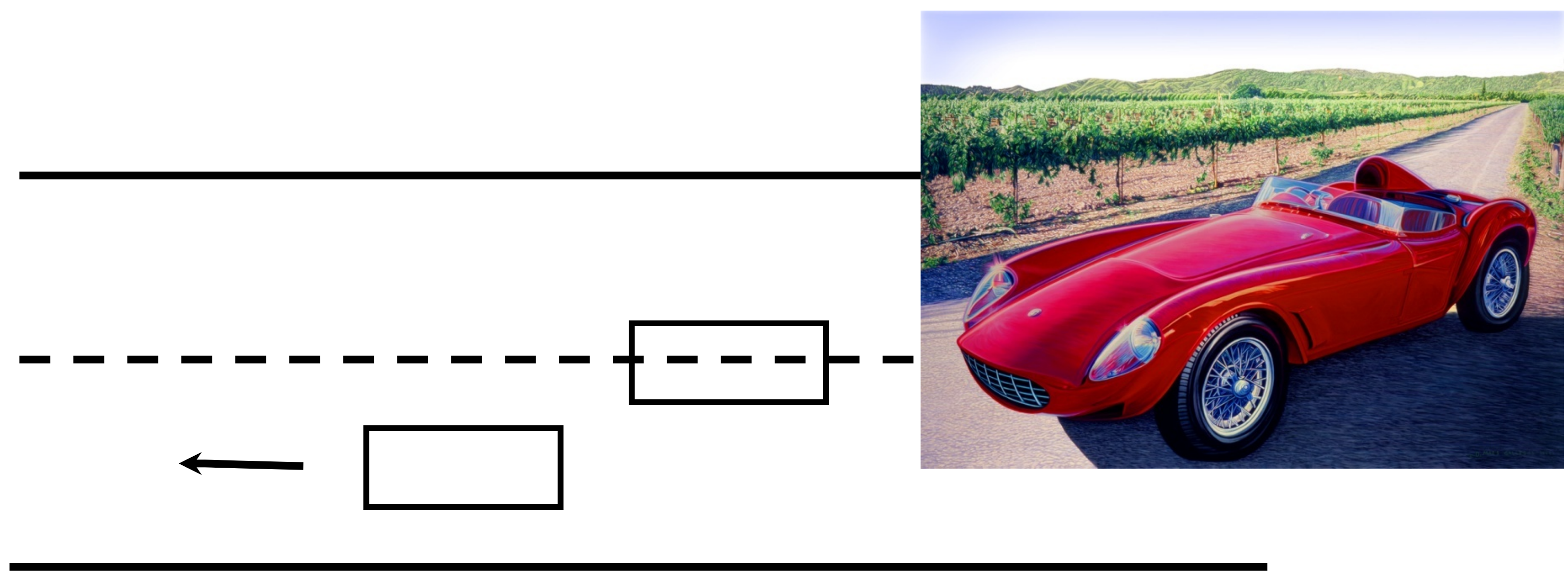
# What if respond ~ rate of change ?

- Steer a car in the center of it lane



# Example: driving

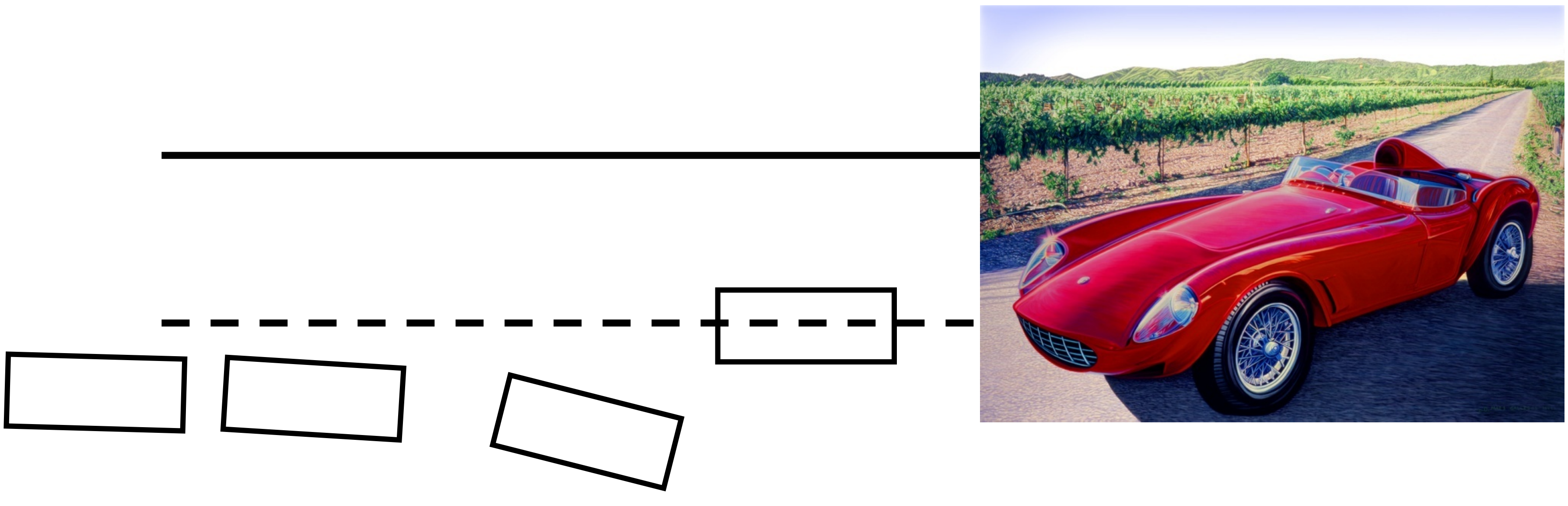
- Steer a car in the center of it lane



What is the observed rate of error?
Other error?

# What if respond ~ rate of change ?

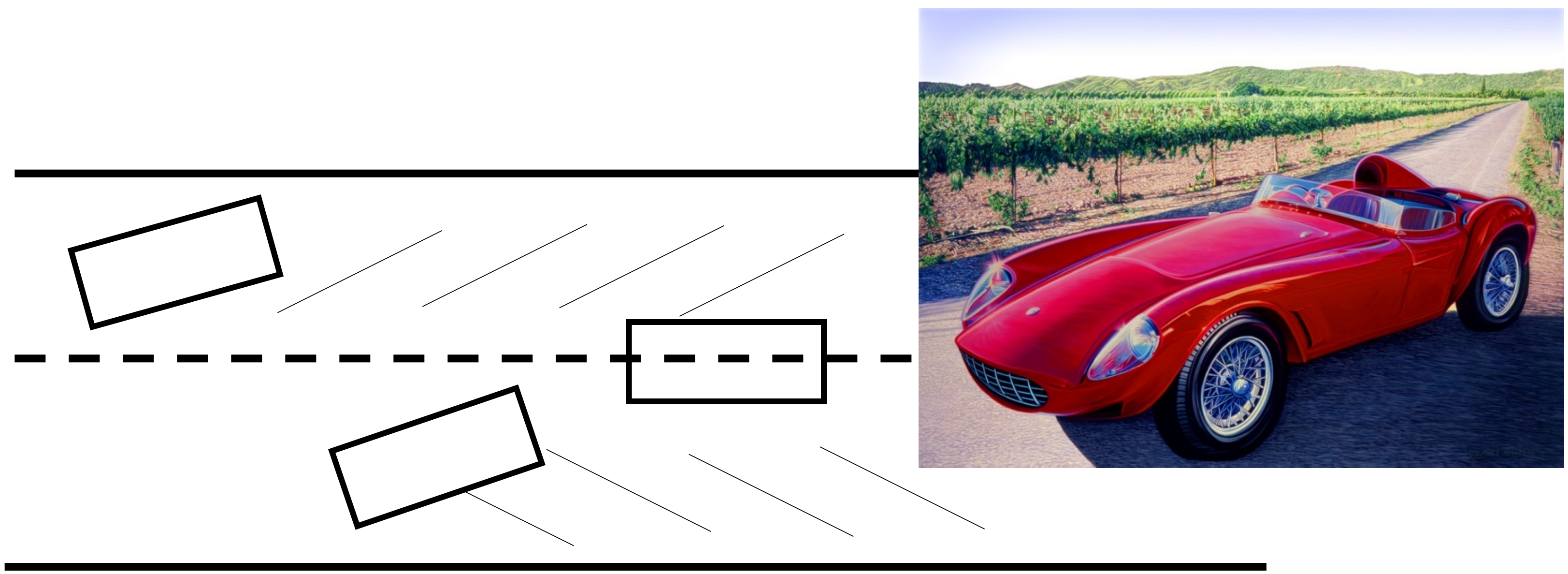
- Steer a car in the center of it lane



D controller is Happy on any parallel line!

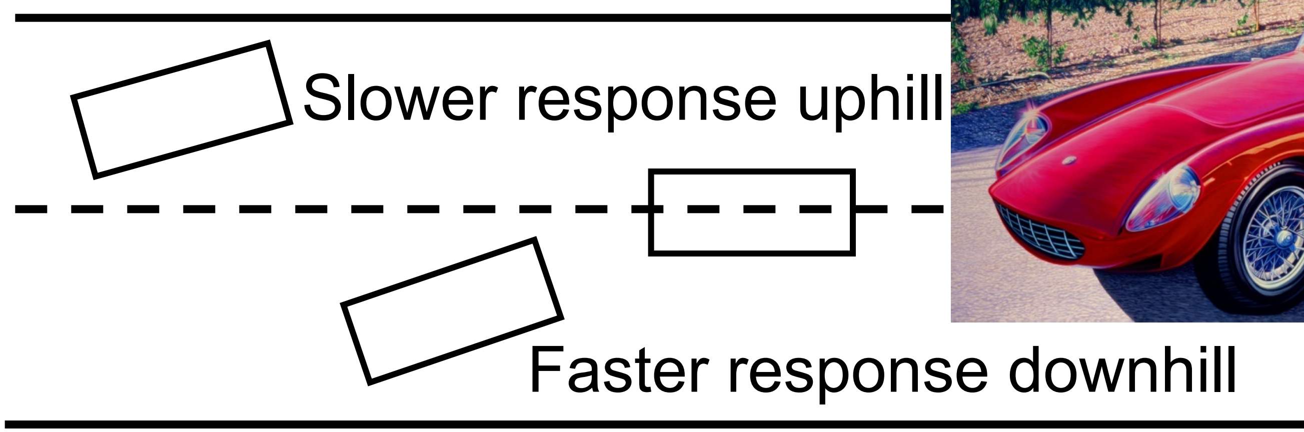
# What if Road Sloped ?

- Steer a car in the center of its lane



# What if Road Sloped ?

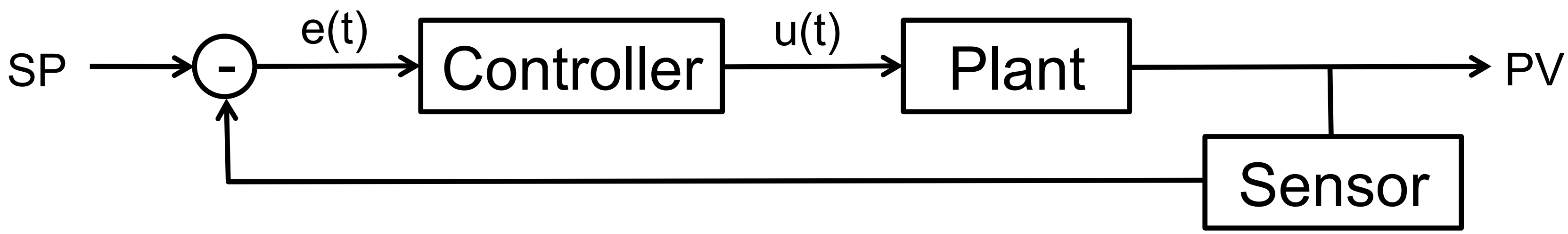
- Steer a car in the center of it lane




Gravity contributes a steady-state error

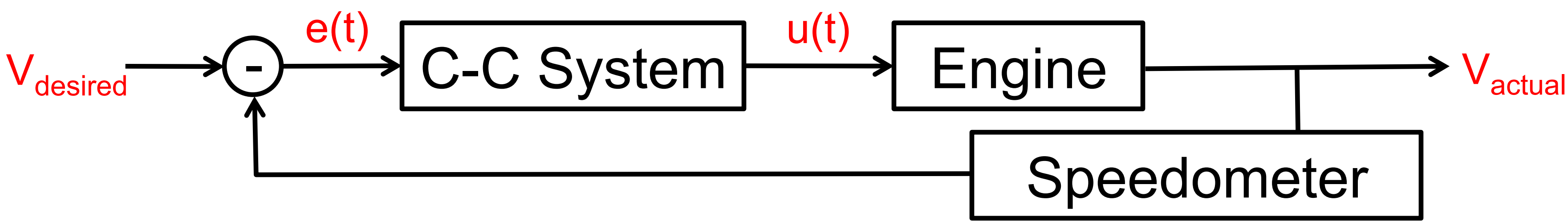
# Proportional Control

- Suppose plant can be commanded by a *continuous*, rather than discrete, signal
  - Valve position to a pipeline or carburetor
  - Throttle to an internal combustion engine
  - PWM value to a DC motor
- What's a natural thing to try?
  - *Proportional* (P) C*ontrol*: make the command signal a scalar multiple of the error term: $u(t) = K_P \times e(t)$



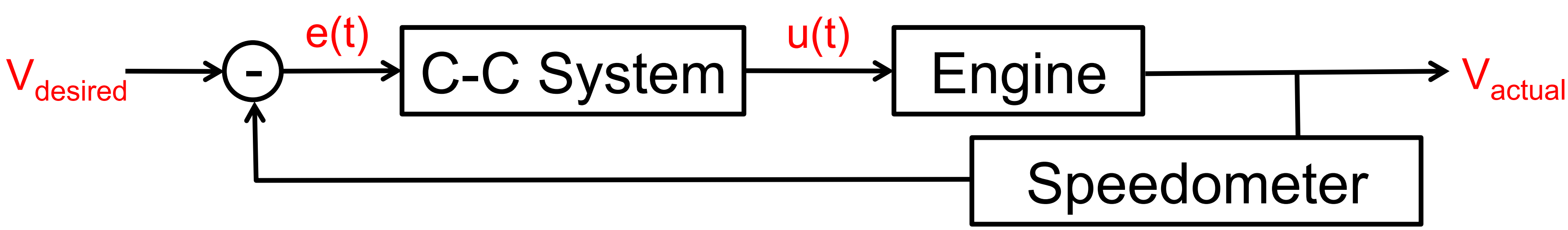
# Example: Cruise Control (CC) System

- *Plant* P:
- *Process Variable* PV:
- *Controller:* *Sensor:*
- *Set Point* SP:
- *Control signal:*



# Example: Cruise Control (CC) System

- *Plant* P: Engine with throttle setting u ∈ [0..1]
- *Process Variable* PV: Current speed $V_{actual}$
- *Controller:* C-C system *Sensor:* Speedometer
- *Set Point* SP: Desired speed $V_{desired}$
- *Control signal*: Continuous throttle value u ∈ [0..1]




Define $e(t) = V_{desired} - V_{actual}$, $u(t) = K_P \times e(t)$, clipped to [0..1]

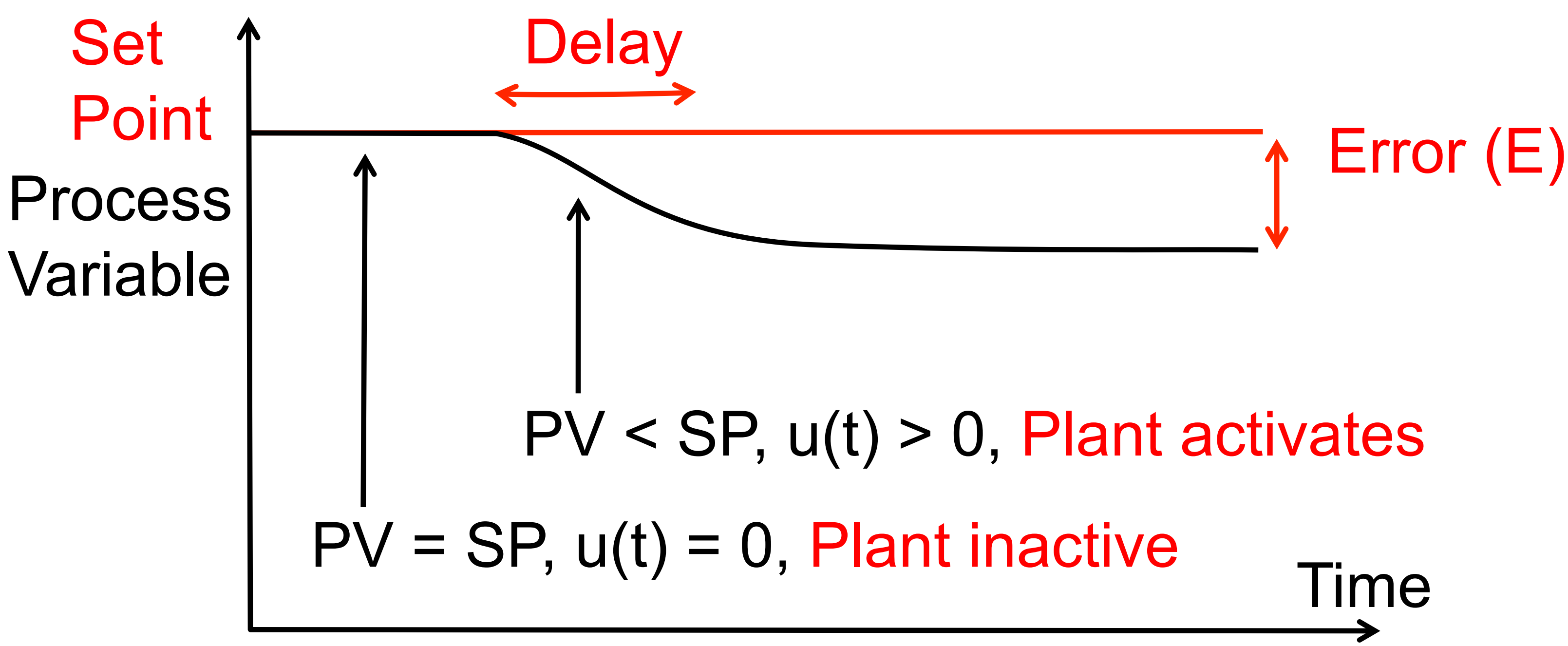
i.e. Throttle = $K_P \times (V_{desired} - V_{actual})$

Does this controller "settle" at the desired speed?

No; it exhibits **error** (E).

# Proportional Control: Why E?

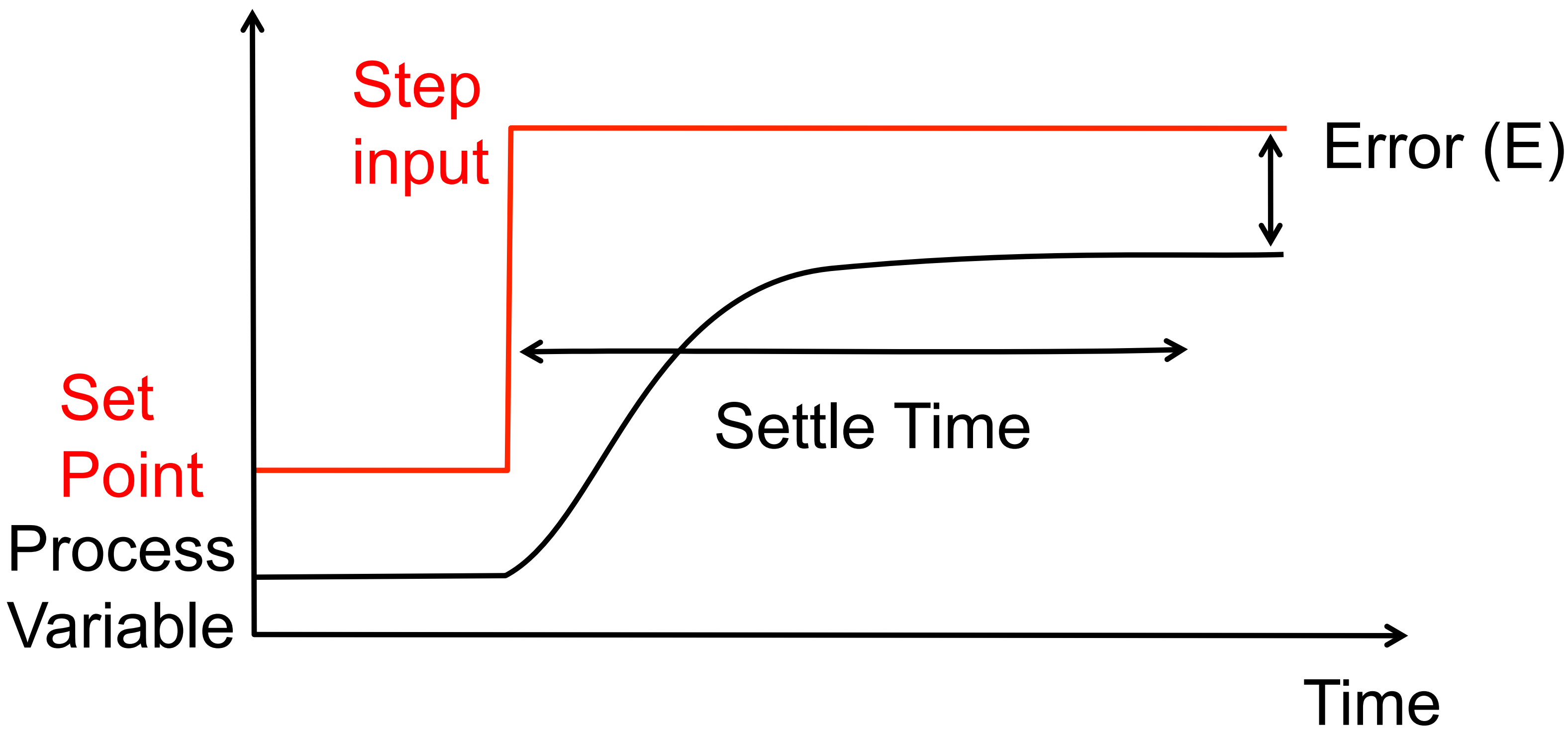
- Suppose $e(t) = 0$. Then $u(t) = K_P * e = 0$ (Plant inactivated)
- Process Variable *deviates* from Set Point, activating plant
- But any real physical system has a *delayed response*
- Deviation, sustained over delay interval, yields error




Why not just introduce constant term, $u(t) = A + K_P * e(t)$ ?

# Proportional Control Step Response

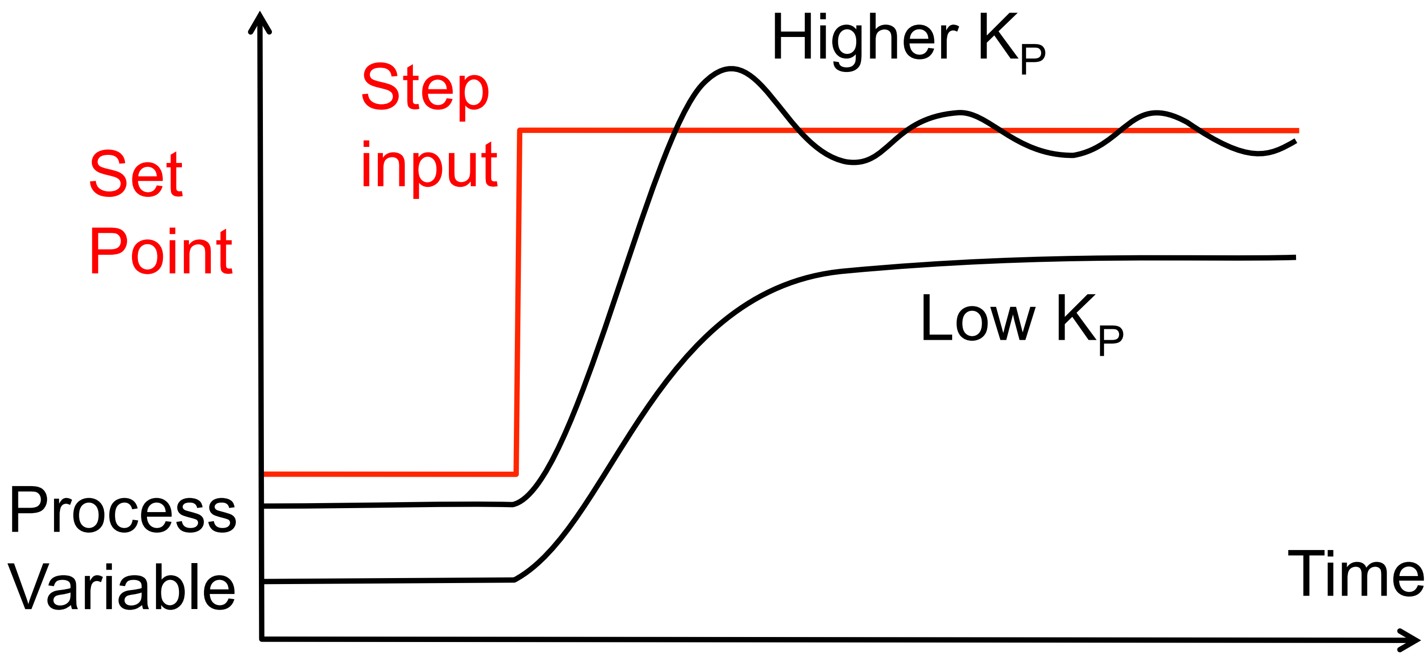
Notional plot and terminology:




Is E constant over time? No; it depends on *load.*

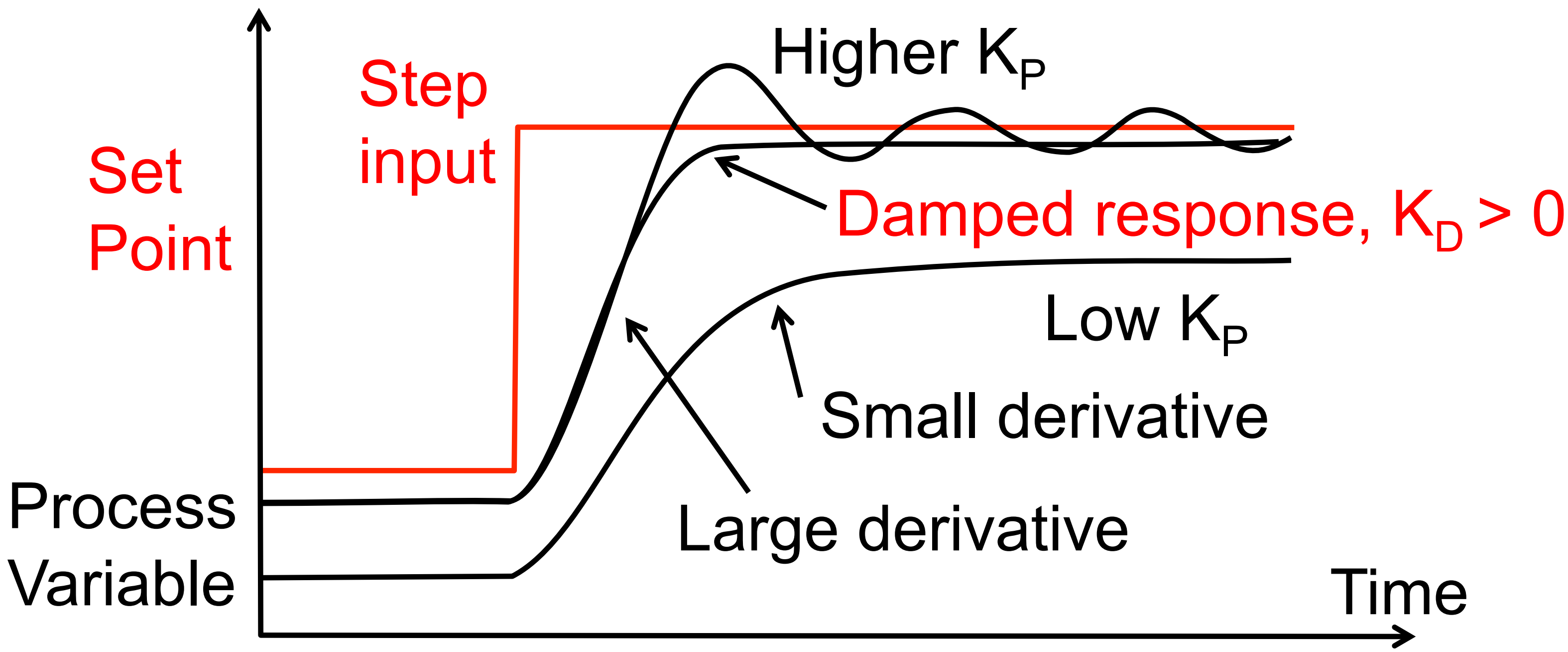
# Proportional Control and Error

- Can combat E by increasing $K_P$ ("the P gain")
- This gives a faster response and lower E!
- But increasing the gain too much leads to overshoot and instability



# Combating Overshoot: The D Term

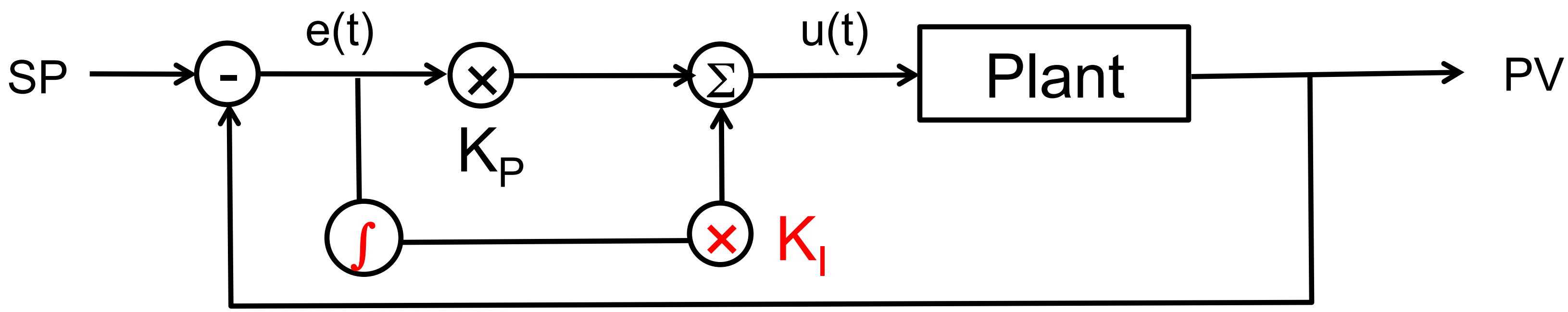
- Note the *derivative* of error in responses below
- *Subtract* it from output to counteract overshoot
- Then $u(t) = K_P \times e(t) + K_D \times d\,[e(t)]\,/\,dt$
  - $K_D$ the “derivative” or “damping” term in PD controller




- … But still haven’t eliminated steady-state error!

# Combating Steady-State Error: I Term

- Idea: apply correction based on *integrated* error
  - If error persists, integrated term will grow in magnitude
  - Sum proportional and integral term into control output

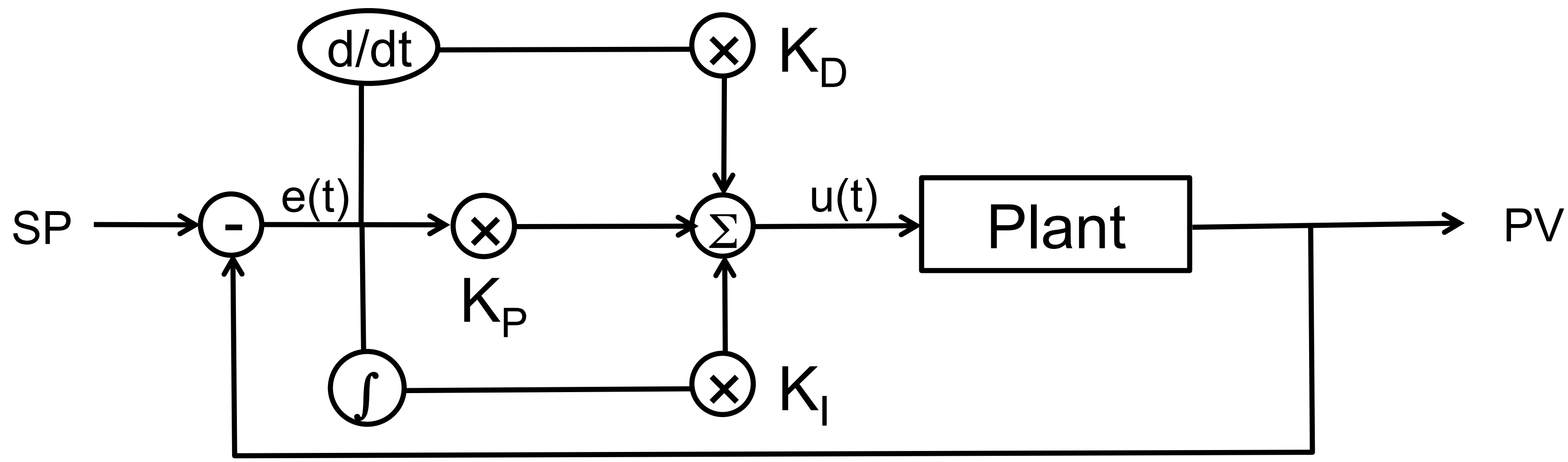



Then $u(t) = K_P \times e(t) + K_I \times \int e(t)$ (where the integral of the error term is taken over some specified time interval)
This produces a *proportional-integral* (PI) controller

Incorporating the I term eliminates SSE by modulating the plant input so that the *time-averaged error is zero*.

# Putting it All Together: PID Control

- Incorporate P, I and D terms in controller output
  - Combine as a weighted sum, using gains as weights




Then $u(t) = K_P \times e(t) + K_I \times \int e(t) + K_D \times d\,[e(t)]\,/\,dt$

This is a "proportional-integral-derivative" or *PID controller*

# Implementation Issues

- How do we approximate $K_i * \int err(t)\, dt$ to implement an I controller?

- How do we approximate $K_d * derr/dt$ to implement a D controller?

# PID control

- PID control combines P and D control:

  $o = K_p * i + K_i * \int i(t)\, dt + K_d * di/dt$

  $o = K_p * err + K_i * \int err(t)\, dt + K_d * derr/dt$

- P component combats present error
- I component combats past (cumulative) error
- D component combats future error
- Gains must be tuned together

# Ziegler-Nichols Tuning Method

Exploration: set the plant under P control and start increasing the Kp gain until loop oscillates

Note critical gain $K_C$ and oscillation period $T_C$

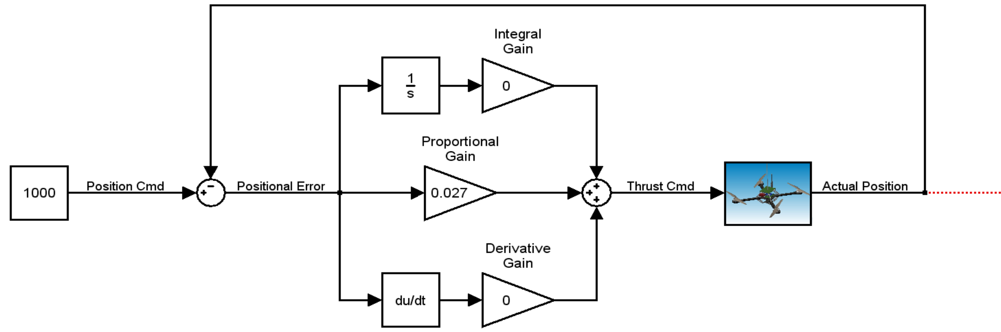
| | $K_P$ | $K_I$ | $K_D$ |
|---|---|---|---|
| P | $0.5K_C$ | | |
| PI | $0.45K_C$ | $1.2K_P/T_C$ | |
| PID | $0.5K_C$ | $2K_P/T_C$ | $K_PT_C/8$ |

Z & N developed rule using Monte Carlo method

Rule useful in the absence of models

# Example: Quadrotor Control Using Ziegler-Nichols Method

- Integral and Derivative gains are set to zero
- Proportional gain is increased until system oscillates in response to a step input
- This is known as the critical gain $K_C$, and the system oscillates with a period $P_C$

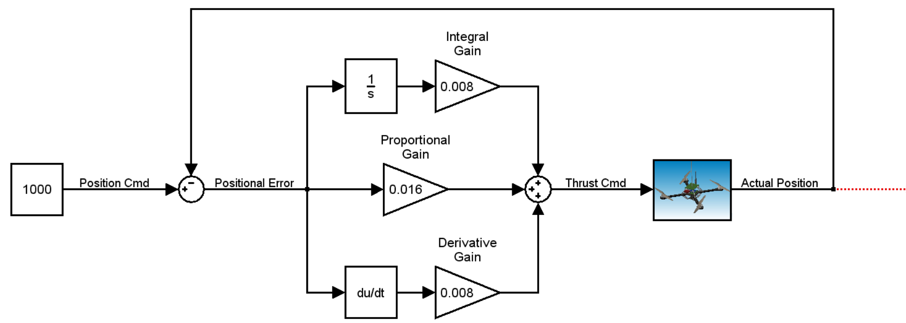



# Calculating PID parameters

- $K_P = 0.6*K_C$
- $K_I = 2*K_P / T_C$
- $K_D = 0.125*K_P*T_C$

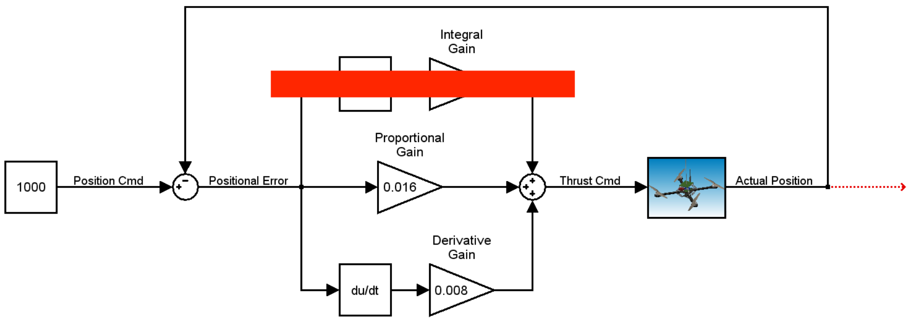
- $T_C = 4$

- The Ziegler-Nichols Method is a guideline for experimentally obtaining 25% overshoot from a step response.



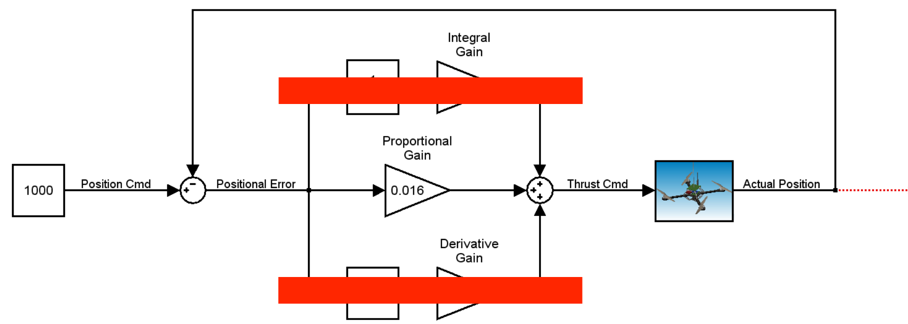

# What does $K_I$ do?

- For this system, $K_I$ is crucial in eliminating steady state error primarily caused by gravity.
- The PD system overshoots and damping is still reasonable.




# What does $K_D$ do?

- Reasonable $K_D$ helps minimize overshoot and settling time.
- Too much $K_D$ leads to system instability.
- P system has unacceptable overshoot, settling time, and steady state error. However, it is stable.




# Control summary

| | Control type | Feedback | Pro/Con |
|---|---|---|---|
| Bang-bang | discrete | yes | Simple/ Discrete |
| Open loop | Control law | no | Simple/may be unrepeatable |
| Closed loop | P, I, D | yes | Continuous/ Tune Gains |