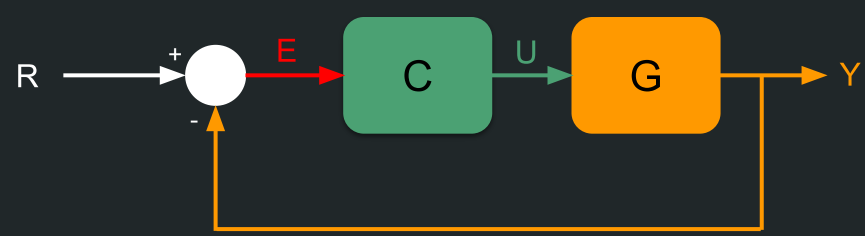
Digital Control Systems Design

Renan G. Maidana
renan.maidana@acad.pucrs.br

Porto Alegre, 2018

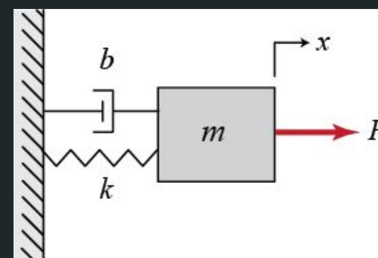
Recap

- So far, we have learned:
    - What are dynamic systems
    - What are feedback control systems



Recap

- So far, we have learned:
    - What are dynamic systems
    - What are feedback control systems
    - How to model a dynamic system
        - Differential Equations
        - Transfer Functions
        - System Identification



$$m\ddot{x} + b\dot{x} + kx = u$$

Recap

- So far, we have learned:
  - What are dynamic systems
  - What are feedback control systems
  - How to model a dynamic system
  - How to analyze a dynamic system’s performance
    - In time (Rise and Settling times)
    - In amplitude (Steady-State error)
    - In stability

Finally...

- Given a dynamic system we want to control, we can:
  - Model its Transfer Function (or approximate it through identification)
  - Define stability and performance parameters
  - Design a feedback control system which achieves our expectations

Finally…

- Given a dynamic system we want to control, we can:
  - Model its Transfer Function (or approximate it through identification)
  - Define stability and performance parameters
  - Design a feedback control system which achieves our expectations

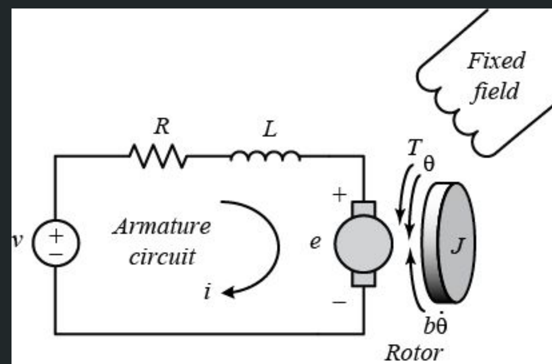
- The running example in this part will be the angular speed of a DC motor shaft
  - Very useful in robotics (motor speed -> robot motion)

Running Example

- The DC motor angular speed can be modelled with regards to the input voltage

$$\frac{\dot{\theta}}{V} = \frac{K}{(Js+b)(Ls+R)+K^2}$$

J = Moment of inertia
b = Viscous damping ratio
K = Electromotive force constant
R = Armature resistance
L = Armature inductance



Running Example

- Let us assume the following constructive parameters:
  - J = K = 0.01
  - b = 0.1
  - R = 1
  - L = 0.5
- Thus, the running example's model is:

$$G(s) = \frac{2}{s^2 + 12s + 20.02}$$

- Normally, the constructive parameters are unknown in a motor, since we can't open it up and measure its individual parts
  - Solution: Approximate a transfer function through IDENTIFICATION!

Control System Objectives

- We can start by analyzing the motor's **stability**
  - What are the poles and zeros of this system?

$$G(s) = \frac{2}{s^2 + 12s + 20.02}$$

No zeros

Poles in -10 and -2

Control System Objectives

- We can start by analyzing the motor's **stability**
  - What are the poles and zeros of this system?

$$G(s) = \frac{2}{s^2 + 12s + 20.02}$$

No zeros

Poles in -10 and -2

- Thus, the system is **stable**

Control System Objectives

- Now that we know it is stable, what is its closed-loop **steady-state response**?
  - What is the error to a step input?

$$e_{ss} = \lim_{s\to 0}\left(\frac{1}{1+G(s)}\right)$$

Control System Objectives

- Now that we know it is stable, what is its closed-loop **steady-state response**?
  - What is the error to a step input?

$$e_{ss} = \lim_{s \to 0} \left(\frac{1}{1 + G(s)} \right) \qquad e_{ss} = \frac{1}{1 + \frac{2}{20}} = 0.909$$

Control System Objectives

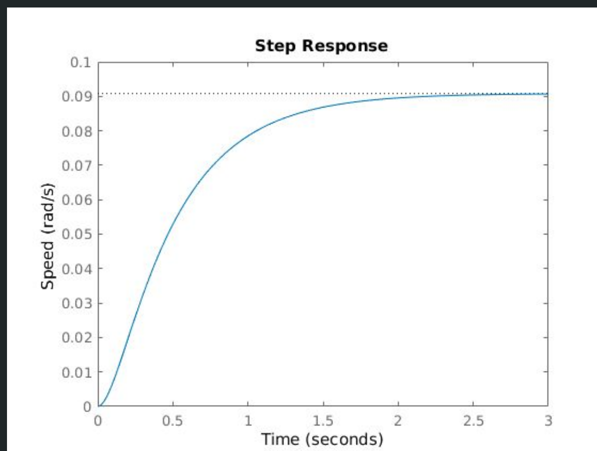
- Now that we know it is stable, what is its closed-loop **steady-state response**?
  - What is the error to a step input?

$$e_{ss} = \lim_{s \to 0} \left(\frac{1}{1 + G(s)} \right) \qquad e_{ss} = \frac{1}{1 + \frac{2}{20}} = 0.909$$

- The angular velocity is 0.091 rad/s for an input of 1 V
  - This is 90.9% off from our desired output of 1 rad/s per 1 Volt

Control System Objectives

- If we plot the closed-loop response, we can **visually verify both stability and steady-state error**



Control System Objectives

- Conclusion:
  - The system is stable, but it does not reach the desired steady-state response (1 rad/s per 1 V)
- How can we fix the steady-state error?

PID Control

- A **PID Controller** is a general class of controllers which combine three types of compensation
  - **Proportional (P)**
  - **Integral (I)**
  - **Derivative (D)**

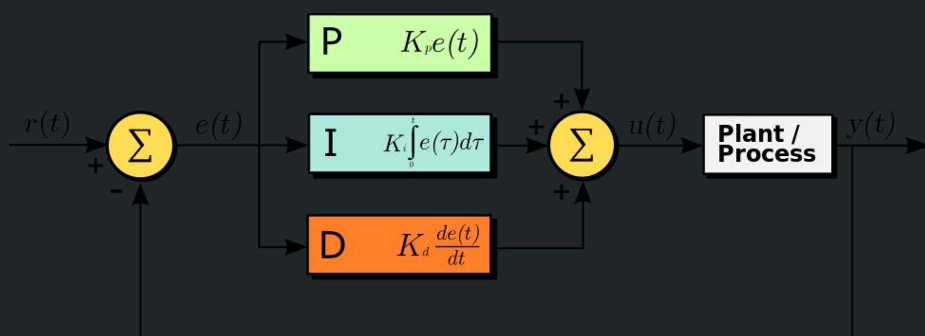
- The Transfer Function for a PID controller is:

$$\frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s$$

where Kp, Ki and Kd are the controller gains

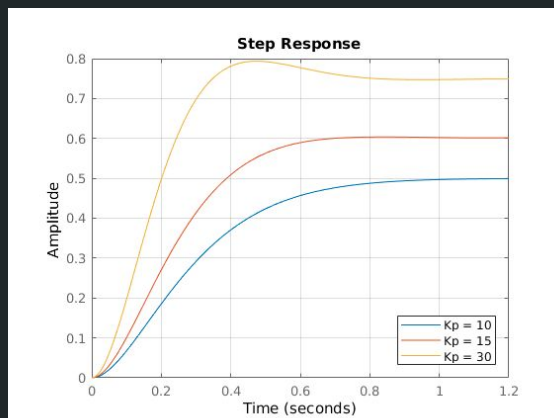
PID Control

- A **PID Controller** is a general class of controllers which combine three types of compensation
  - **Proportional (P)**
  - **Integral (I)**
  - **Derivative (D)**



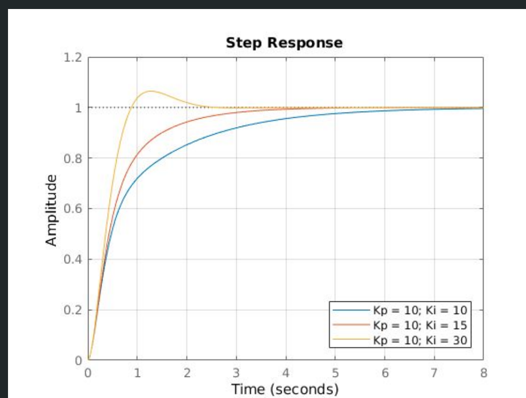
PID Control

- By increasing the proportional gain, we alter the system's **immediate response**, decreasing the rise time and the **steady-state error**



PID Control

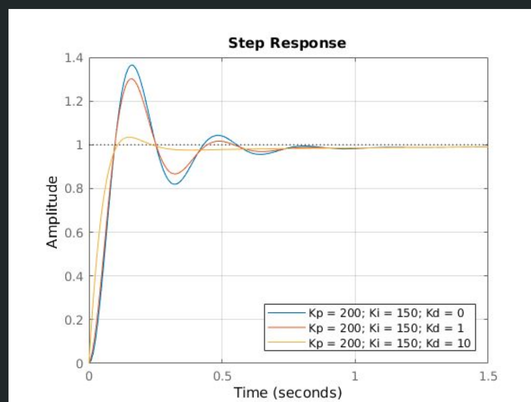
- Typically, proportional compensation is not enough to eliminate the steady-state error completely
  - The **integral effect** brings the steady-state error to 0



- An interpretation for the integral action is that it acts by **minimizing the amplitude error (in y)**

PID Control

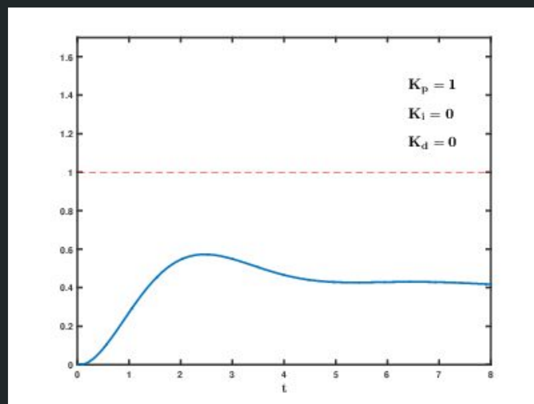
- The integrator may introduce unwanted oscillatory behavior in the system
  - The **derivative effect** eliminates oscillations



- An interpretation for the derivative action is that it acts by **minimizes the time error (in x)**

PID Control

- GIF:



https://upload.wikimedia.org/wikipedia/commons/3/33/PID_Compensation_Animated.gif

PID Control

- How do we implement a digital PID control loop?

```python
# C(s)
def PID():
    global I

    # Error
    e.append(r[-1] - y[-1])

    # Integral
    I += e[-1]*T

    # Derivative
    D = (e[-1]-e[-2])/T

    # P + I + D
    u = Kp*e[-1] + Ki*I + Kd*D
    return u
```

- To be applied to a properly discretized G(s) system

PID Tuning

- How do we choose the Kp, Ki and Kd gains?
    - Lots of guess work!

Effects of *increasing* a parameter independently[21][22]

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

- The search state-space for guessing the PID gains is huge, typically involving:
    - Altering one of the gains
    - Applying the tuned control
    - Recording the controlled system output
    - Seeing if the control achieved the desired closed-loop behavior

PID Tuning

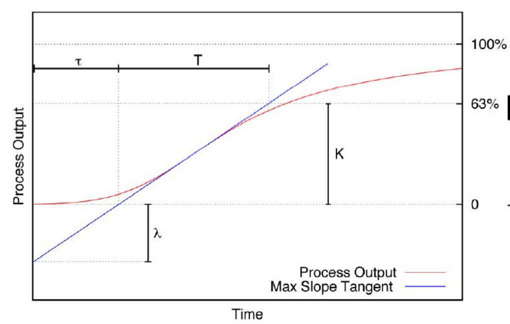
- Alternatively, there are tuning methods for PID controllers:
  - **Ziegler-Nichols**
  - Cohen-Coon

- In the Ziegler-Nichols method, we use the **open-loop** step response to approximate the expected response to a PID

PID Tuning

- In the Ziegler-Nichols method, we use the **open-loop** step response to approximate the expected response to a PID

$$k_p = k \qquad k_i = \frac{k}{T_i} \qquad k_d = kT_d$$



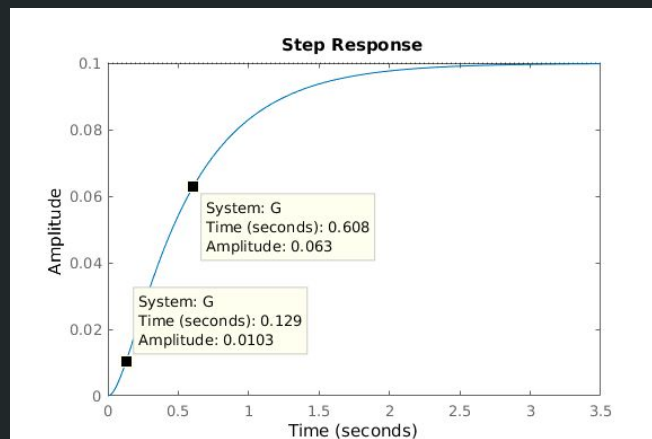
	k	T_i	T_d
P	$1/\lambda$		
PI	$0.9/\lambda$	3τ	
PID	$1.2/\lambda$	2τ	$\tau/2$

- Applied to the **OPEN LOOP** system, that is:
    - u -> G(s) -> y

PID Tuning

- Ziegler-Nichols
  - First, we find T as the difference between the time for 63% and the time for 10%:

T = 0.608 - 0.129 = 0.479 s

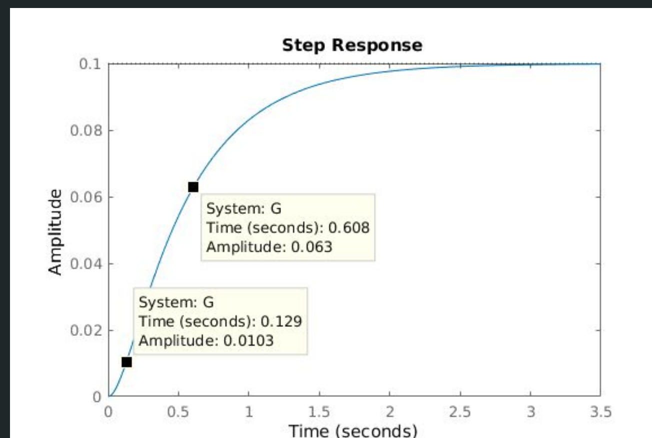


PID Tuning

- Ziegler-Nichols
  - First, we find T as the difference between the time for 63% and the time for 10%:

  T = 0.608 - 0.129 = 0.479 s

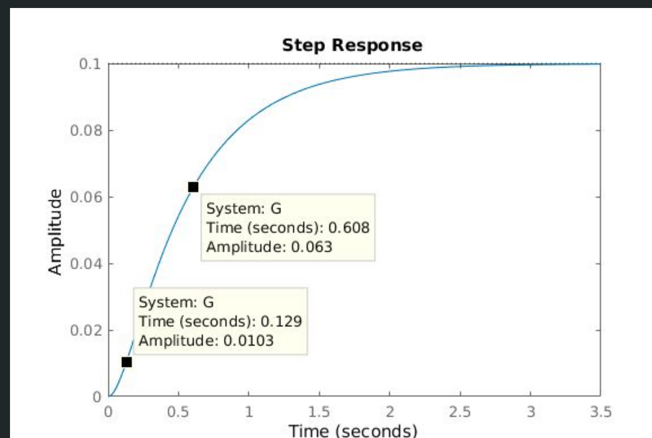
- The dead-time (τ) is the time for the response to achieve 10%, in this case, τ = 0.129 s



PID Tuning

- Ziegler-Nichols
  - The slope λ is the amplitude difference between the upper and lower times, divided by the time elapsed (in this case, T)

$$\lambda = \frac{0.063 - 0.0103}{0.479} = 0.11$$



- The slope also gives us the declivity of the time response, or how fast it changes in time

PID Tuning

- Ziegler-Nichols
  - Finally, the gains for the PID can be found using the table

k = 1.2/0.11 = 10.9
Ti = 2 x 0.129 = 0.258
Td = 0.129/2 = 0.0645

Kp = k = 10.9
Ki = k/Ti = 42.248
Kd = k x Td = 0.703

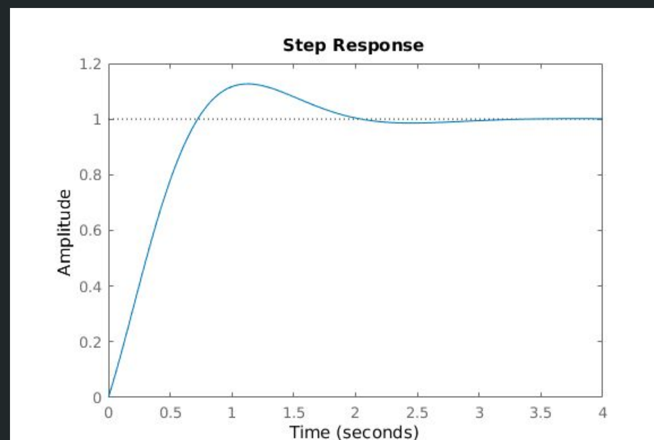
	k	T_i	T_d
P	$1/\lambda$		
PI	$0.9/\lambda$	3τ	
PID	$1.2/\lambda$	2τ	$\tau/2$

$k_p = k \qquad k_i = \dfrac{k}{T_i} \qquad k_d = kT_d$

PID Tuning

- Ziegler-Nichols

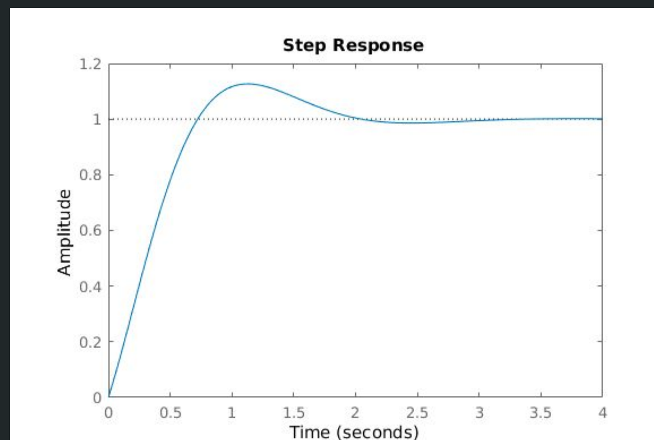
Kp = k = 10.9
Ki = k/Ti = 42.248
Kd = k x Td = 0.703



PID Tuning

- Good response!
  - Steady-state error is zero
  - Stable

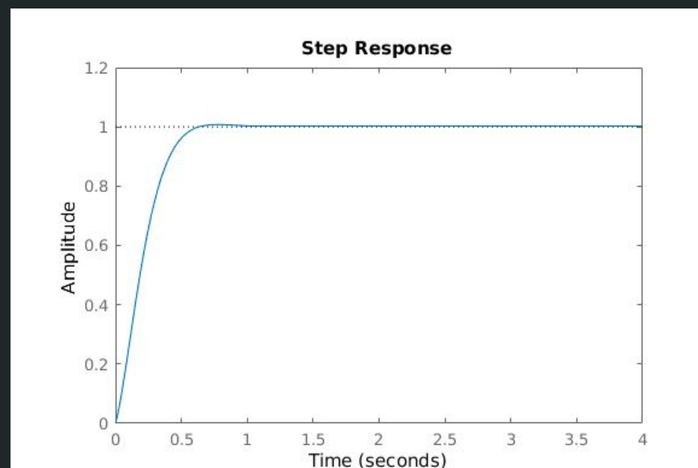
- The rise time and oscillatory behavior are still a bit lacking
  - **Ziegler-Nichols provides only an approximation of the ideal gains!!**
  - **Some manual tuning is still required**



PID Tuning

- Ziegler-Nichols (after manual fine-tuning)

Kp = 25.9
Ki = 48.2
Kd = 0.803



- The desired rise time, overshoot and steady-state error depend on the control system designed
    - **Manually fine-tune your PID to whatever you want the system behavior to be**

PID Tuning

- There are many tuning methods out there
  - The Cohen-Coon method for example:

		k	T_i	T_d
Cohen–Coon	P	$\frac{1}{K}\left(1+\frac{0.35\theta}{1-\theta}\right)\frac{T}{\tau}$		
	PI	$\frac{0.9}{K}\left(1+\frac{0.92\theta}{1-\theta}\right)\frac{T}{\tau}$	$\frac{3.3-3.0\theta}{1+1.2\theta}\tau$	
	PID	$\frac{1.35}{K}\left(1+\frac{0.18\theta}{1-\theta}\right)\frac{T}{\tau}$	$\frac{2.5-2.0\theta}{1-0.39\theta}\tau$	$\frac{0.37(1-\theta)}{1-0.81\theta}\tau$

$$\theta = \frac{\tau}{\tau + T}$$

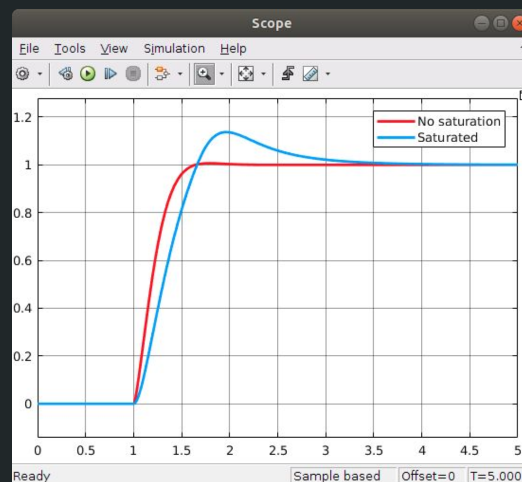
- If a certain method is not working, maybe another will do the trick

- Other examples:
  - AMIGO
  - Ziegler-Nichols with Critical Gain
  - Etc

Anti-Windup

- The **actuator saturation** is a commonly ignored effect in the PID control systems

- When the controller output is outside of what the actuator can handle, the effects of the integral portion accumulate
    - This is known as **wind-up**



- **The integral effect piles up because the controller output is not able to bring the steady-state error to 0 in a timely manner**
    - Even with positive error, **the accumulated integral takes a while to go back to normal**, introducing overshoot effects

Anti-Windup

- To avoid this, we limit the integrator influence if the actuator is saturated

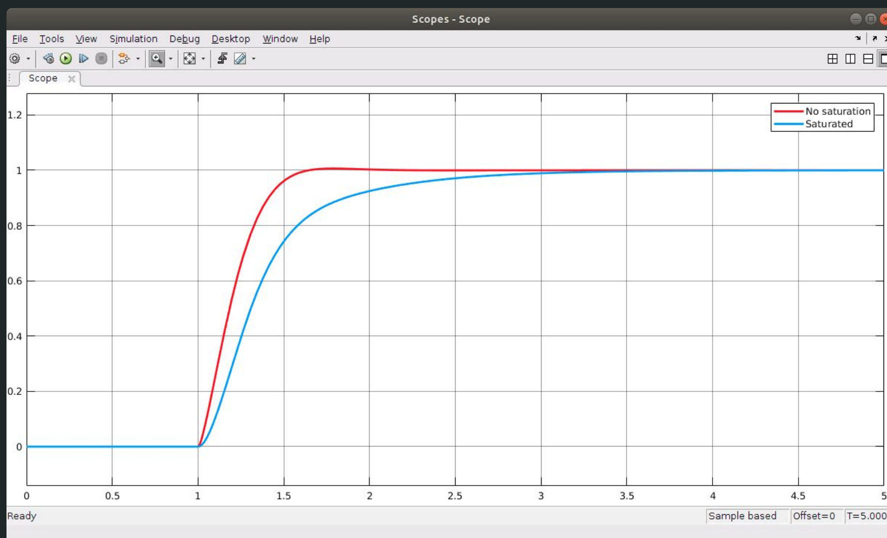
```
# C(s) - Last u is parameter as workaround for antiwindup in python
def PID(u):
    global I

    # Error
    e.append(r[-1] - y[-1])

    # Integral action with anti-windup
    within_saturation = (lsat < u and u < usat)
    upper_exceeded_but_negative_error = (u > usat and e < 0)
    lower_exceeded_but_positive_error = (u < lsat and e > 0)
    if within_saturation or upper_exceeded_but_negative_error or lower_exceeded_but_positive_error:
        I += e[-1]*T

    # Derivative
    D = (e[-1]-e[-2])/T

    # P + I + D
    u = Kp*e[-1] + Ki*I + Kd*D
    return u
```

Anti-Windup logic:

- If actuation is within limits
  - We can integrate
- If actuation has exceeded upper saturation limit
  - But error is negative
    - Integral effect will be subtracted, so we can integrate
  - Else, do not integrate
- If actuation has exceeded lower saturation limit
  - But error is positive
    - Integral effect will be added, so we can integrate
  - Else, do not integrate

If the controller output is saturated, the integral effects (positive and negative) will pile up, causing the controller to have an unaccounted inertia

Anti-Windup



Scopes - Scope
File Tools View Simulation Debug Desktop Window Help
Scope
No saturation
Saturated
1.2
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Ready
Sample based
Offset=0
T=5.000