Robotics 311 : How to build robots and make them move

Prof. Elliott Rouse

GSI Yves Nazon MS

Fall 2022

Photo credit: Brenda Ahearn / U-M CoE

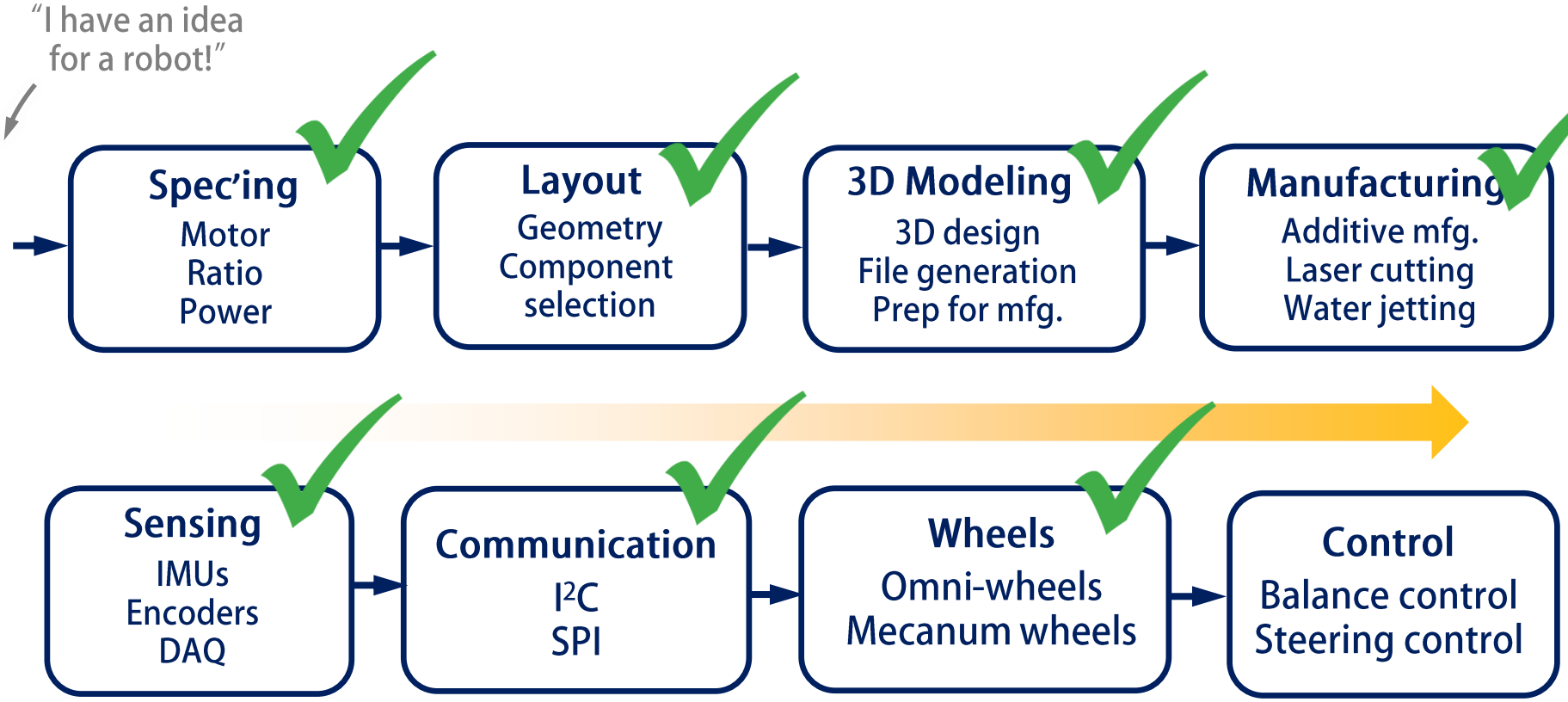
ROB 311 – Lecture 20

- Today we will go over control
- PID + MATLAB example / quiz

Announcements

- T-shirt form
- You can take / access your bins to work on the ball-bot
- Midterms will be graded next week
- Class grading

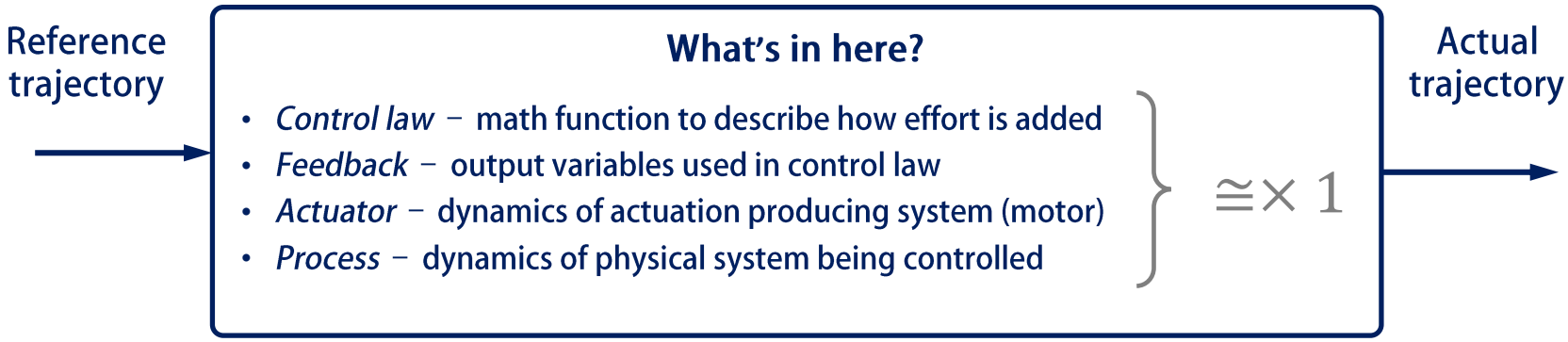
Where We Are



- We've learned much of the spec, design, and make processes
- Now, we need to learn how to make robots do what we want
- At the low level, this involves feedback control

Control

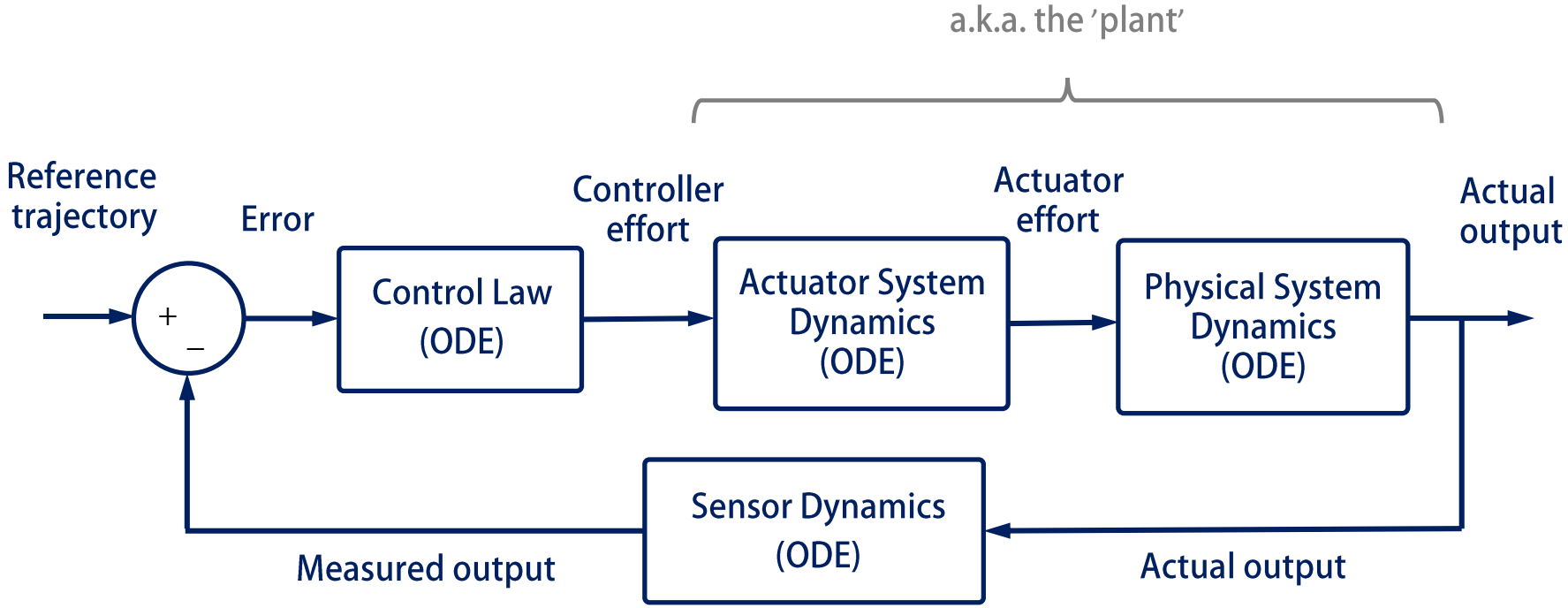
- Today, we will go over the basics of 'low-level' control
- What is the job of low-level control? Track a specific reference signal
- Examples
  - A robot needs to move its arm to follow a trajectory
  - An exoskeleton is providing torque or current assistance
  - Regulate the temperature in a house
  - Cruise control on your car
- The job of a controller? Track the reference / act like unity (so reference = actual)



Feedback Control Diagram

Human movement example

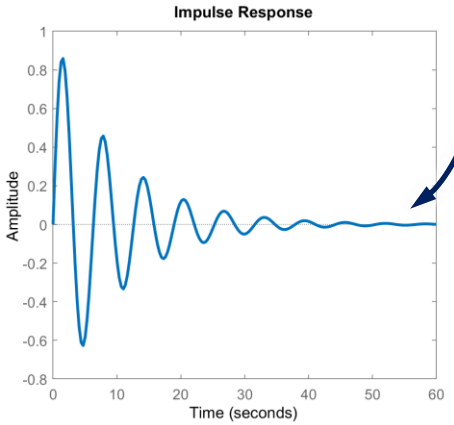
- Why is it hard to make a controller act like unity ($\times 1$)?
- Each one of these blocks is an ODE that maps the block input to the output
- Let's think of an example ODE for a physical system; where would it come from?
- Everyone's favorite: Newton's Second Law
- These systems may have 'dynamics,' which implies a derivative in their equation

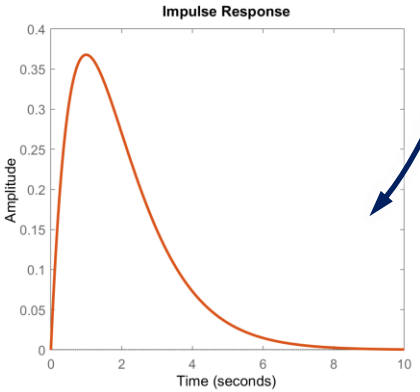


System Dynamics

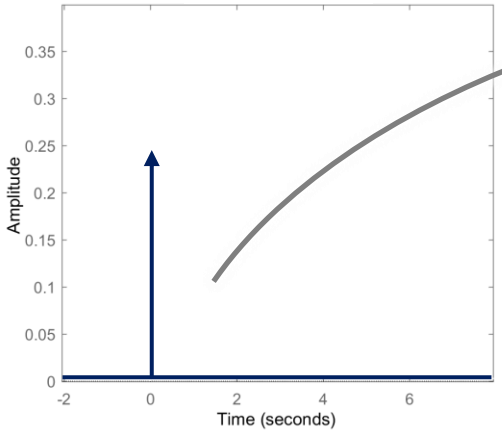
- If a system has dynamics / is an ODE, it will often include delays
- This usually slows the system down with potential oscillations
- We can view this behavior from the system's impulse response (IRF)
- This is the system response to a pure impulse
- Systems are fully described by their IRF
- Lets think about an example 2nd order system

Potential outputs (depending on exact m, k, and c)

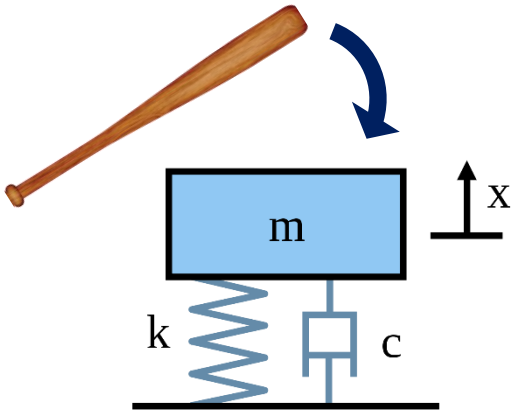




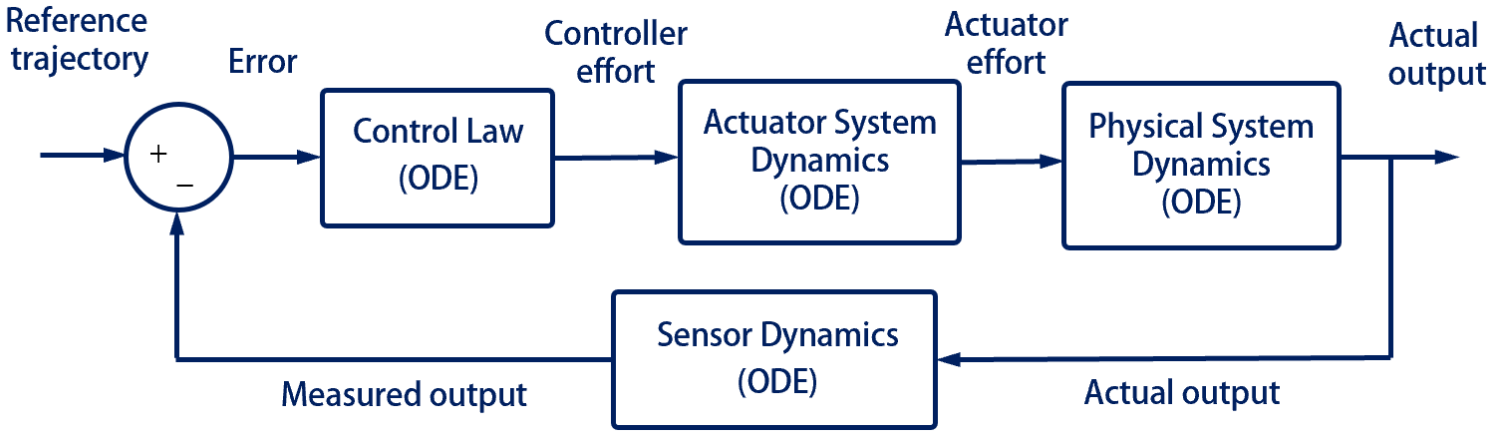
Force input (impulse function)



Delta function gets smoothed / delayed

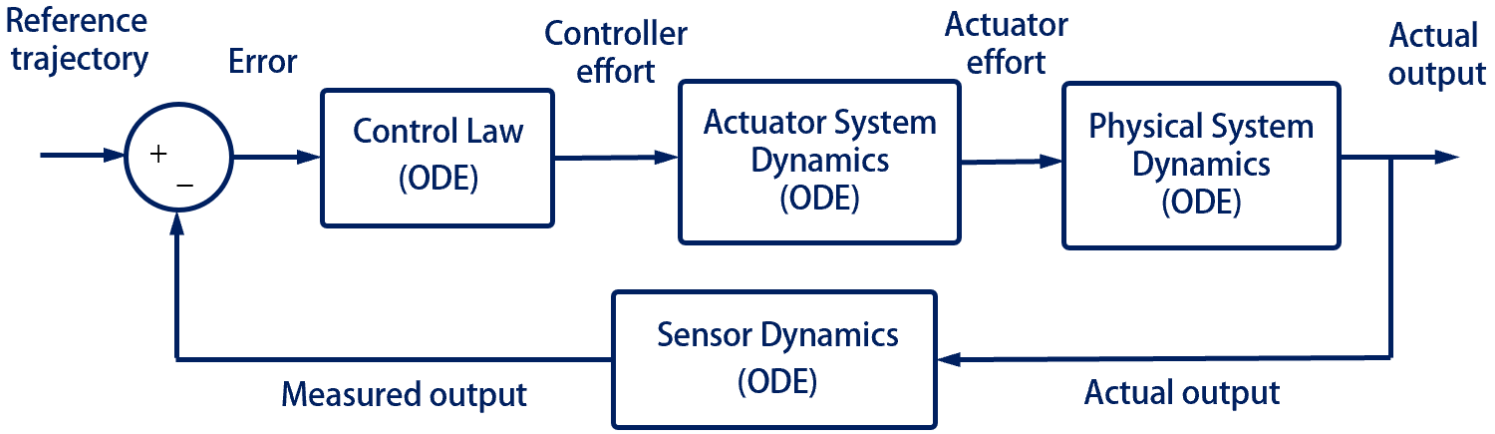


System Dynamics



- These dynamics / delays add challenge to tracking the reference
- Control Law – ODE comes from math instructions are provided, which often include derivatives
- Actuator – ODE that describes how the controller effort maps to actuator effort An example of this is the coupled electromechanical equations of a motor
- Physical system – ODE that describes how the physical system responds to the controller effort
- Sensor – ODE that describes how the output is measured. Most sensors have very little lag (high bandwidth)

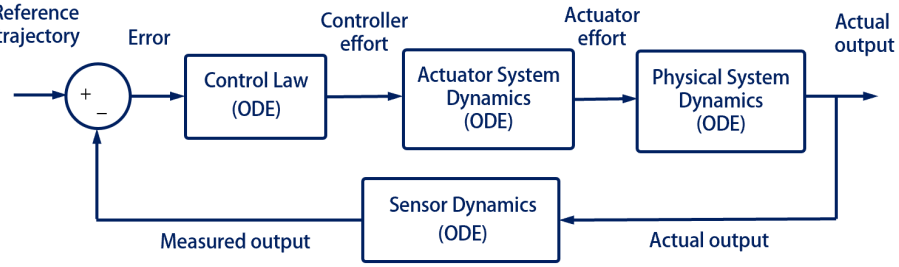
PID Control



- Objective: drive error to zero → if error is zero, system gain is unity
- Control laws are functions of the error signal $(e(t))$ – the difference between the reference and measured output
- We talked about proportional control—where the controller effort was just a gain on the error
- Most common control type: Proportional-Integral-Derivative (PID) control
- Controller effort is the sum of three terms
- Lets look at how the terms are created

PID Control

Controller effort: $u(t)$
Error: $e(t)$
Input: $r(t)$
Output: $y(t)$



- Proportional gain (k_p)
  - Simplest term – error scaled by k_p
  - Acts like a virtual spring (for position control)

$$u_p(t) = k_p e(t)$$

- Integral gain (k_i)
  - Integral of error scaled by k_i
  - Used to remove steady state error
  - Whatever is below reference must be subtracted by what is above reference

$$u_i(t) = k_i \int_{t_0}^{t} e(\tau)\, d\tau$$

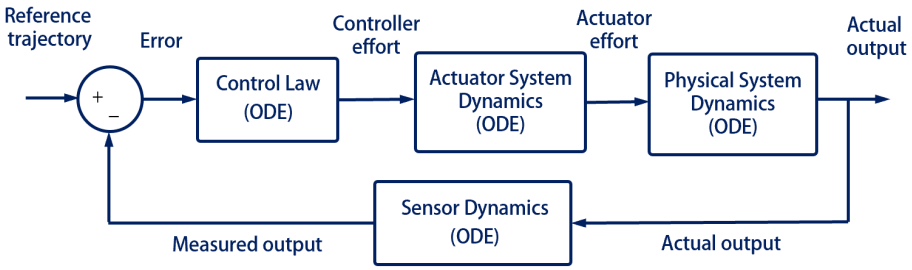
- Derivative gain (k_d)
  - Derivative of error scaled by k_d
  - Virtual damper
  - Never used alone / susceptible to noise

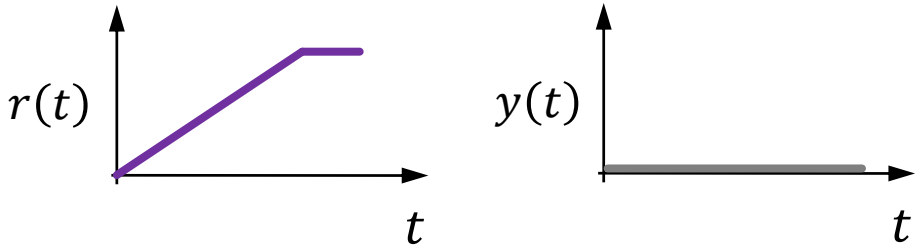
$$u_d(t) = k_d \frac{de(t)}{dt}$$

Controller effort: $u(t)$ is the sum of these terms

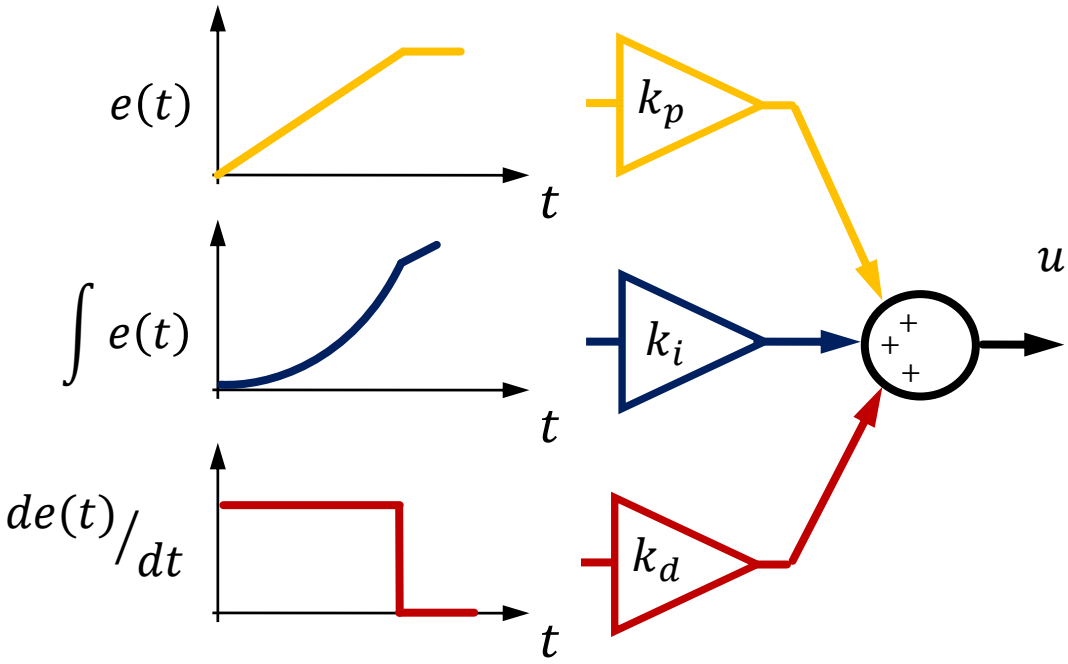
PID Control

Controller effort: $u(t)$
Error: $e(t)$
Input: $r(t)$
Output: $y(t)$

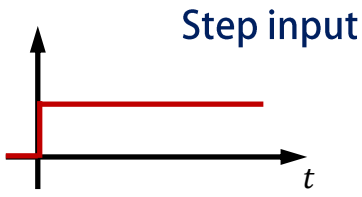




Combined controller output
$(k_p = k_i = k_d = 1)$

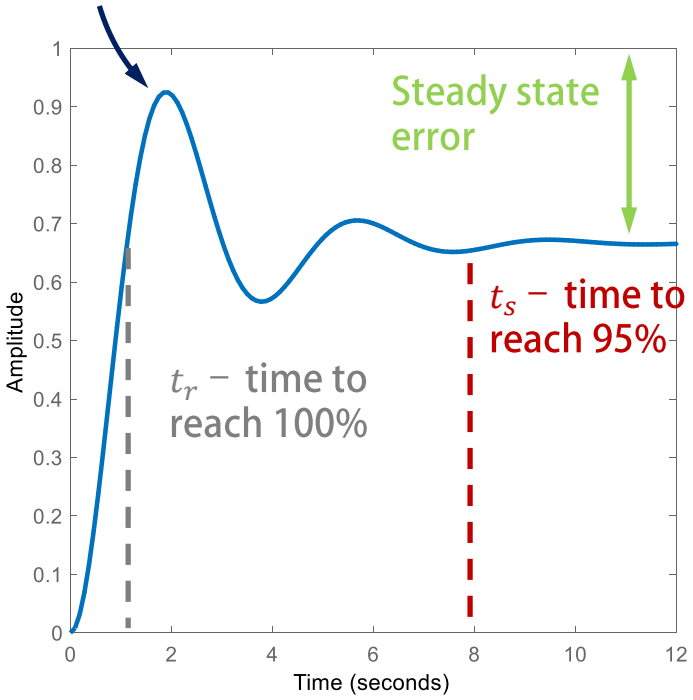


PID Control

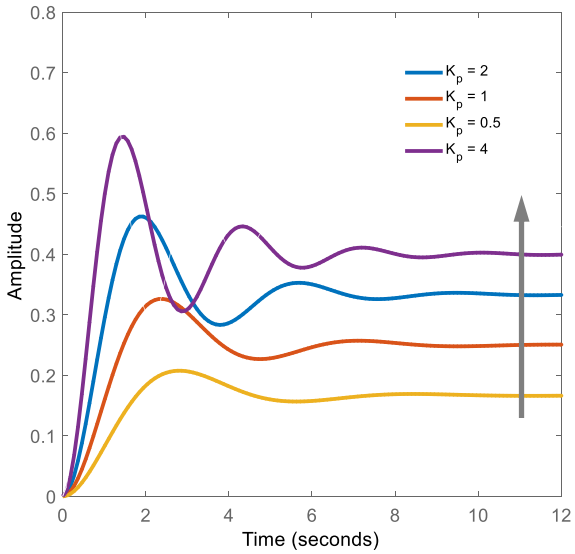


- Coefficients must be tuned, often using a step response
- Done by iteratively by trying different controller parameters – 'tuning'
- Assessed often with time domain parameters
  - Rise time, settling time, percent overshoot, steady state error

Percent overshoot



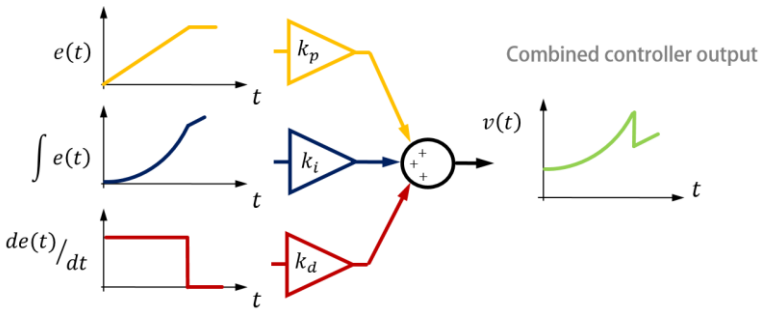
$$u(t) = k_p e(t) + k_i \int_{t_0}^{t} e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$



$\uparrow k_p$

PID Control

- The coefficients in the control law can be adjusted to achieve certain performance
- Ziegler-Nichols has developed different tuning strategies (more info here)
- ① Increase k_p as high as comfortable
- ② Reduce k_p and add k_i to remove steady state error
- ③ Add k_d to remove oscillations
- By driving error to zero, the controller behaves like unity

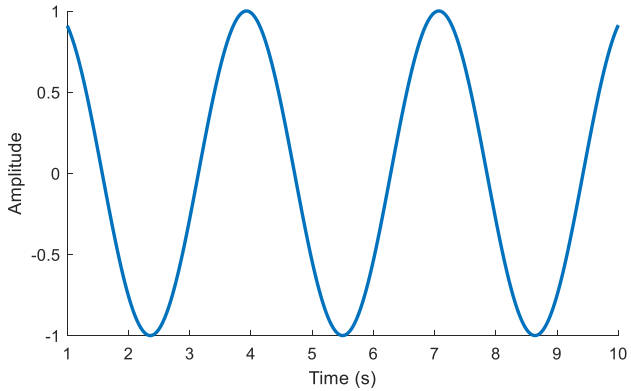


$$u(t) = k_p e(t) + k_i \int_{t_0}^{t} e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$

Parameter Increase	Rise time	Overshoot	Settling Time	S.S. Error
k_p	↓	↑	Small ↑	↓
k_i	↓	↑	↑	↓↓
k_d	Small change	↓	↓	Small change

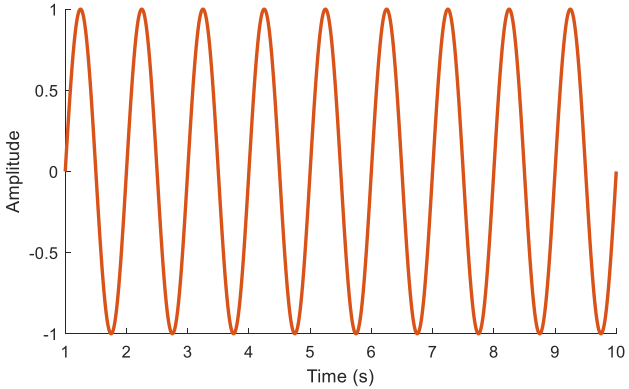
Reference Trajectories

- The trajectory of the reference has a significant impact on controller performance
- Oscillating references are harder to track
- The faster the frequency the harder it will be for the controller



Easier to track

Harder to track



- Some references are unchanging (static) – these systems are called regulators (e.g. thermostat)
- What does our reference look like? Is it high frequency?
- The output should closely match the reference, but at high frequencies, the amplitude will begin to attenuate and the output sine wave will be delayed

PID Control

- MATLAB demonstration of P, PI, and PID controllers
- Then you will tune the PID in MATLAB as today's quiz
- Once tuned, create a PDF of your step response and submit on Canvas

Controller for Z-Axis Torque

- Now we will add the z-axis torque from button commands you choose
- Triggers provide continuous values and buttons provide binary values
- Create a torque function using the button presses and add to the torque commands (or use the demos)
- You will need to set the z-axis torque to the torque value from the PS4 controller