Chapter 7
Control

Part 1

7.1 Classical Control

Outline

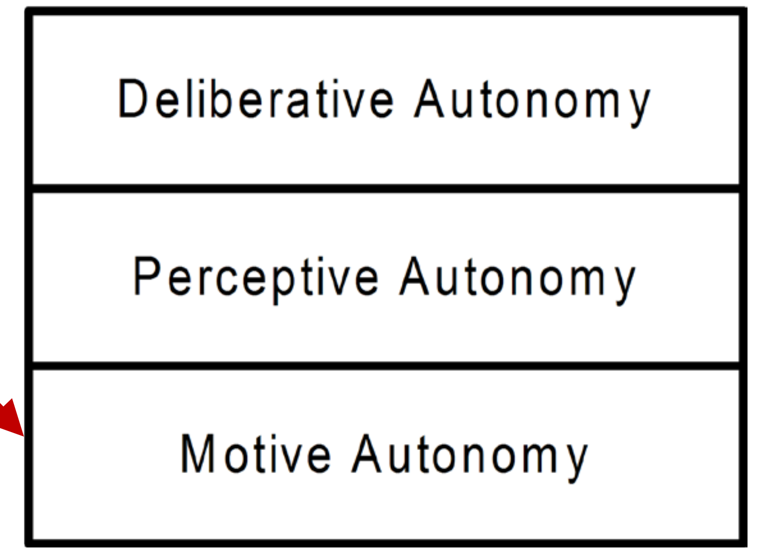
- 7.1 Classical Control
  - 7.1.1 Introduction
  - 7.1.2 Virtual Spring Damper
  - 7.1.3 Feedback Control
  - 7.1.4 Model Referenced and Feedforward Control
  - Summary

Outline

- 7.1 Classical Control
  - 7.1.1 Introduction
  - 7.1.2 Virtual Spring Damper
  - 7.1.3 Feedback Control
  - 7.1.4 Model Referenced and Feedforward Control
  - Summary

Hierarchy

- We are here now …
- Responsible for controlling the motion of the vehicle with respect to the environment.
- Requires feedback only of the motion state (position, heading, attitude, velocity) of the vehicle.
- Path following fits here.



7.1.1 Introduction to Control

- Controllers are a mapping:
  – from actuated variables (forces, power)
  – onto controlled variables (positions, velocities)

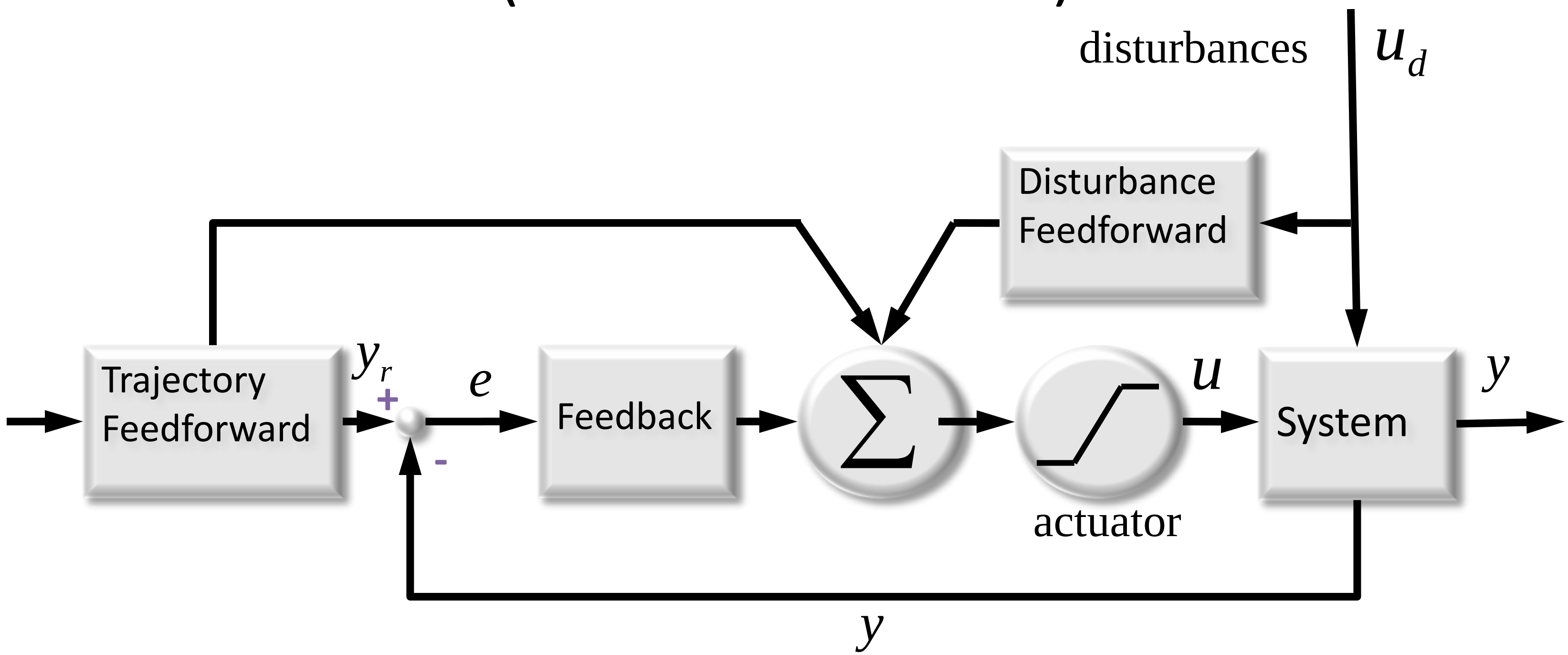
$$\dot{\underline{x}} = f(\underline{x}, \underline{u}, t)$$

Controlled Variables

Actuated Variables

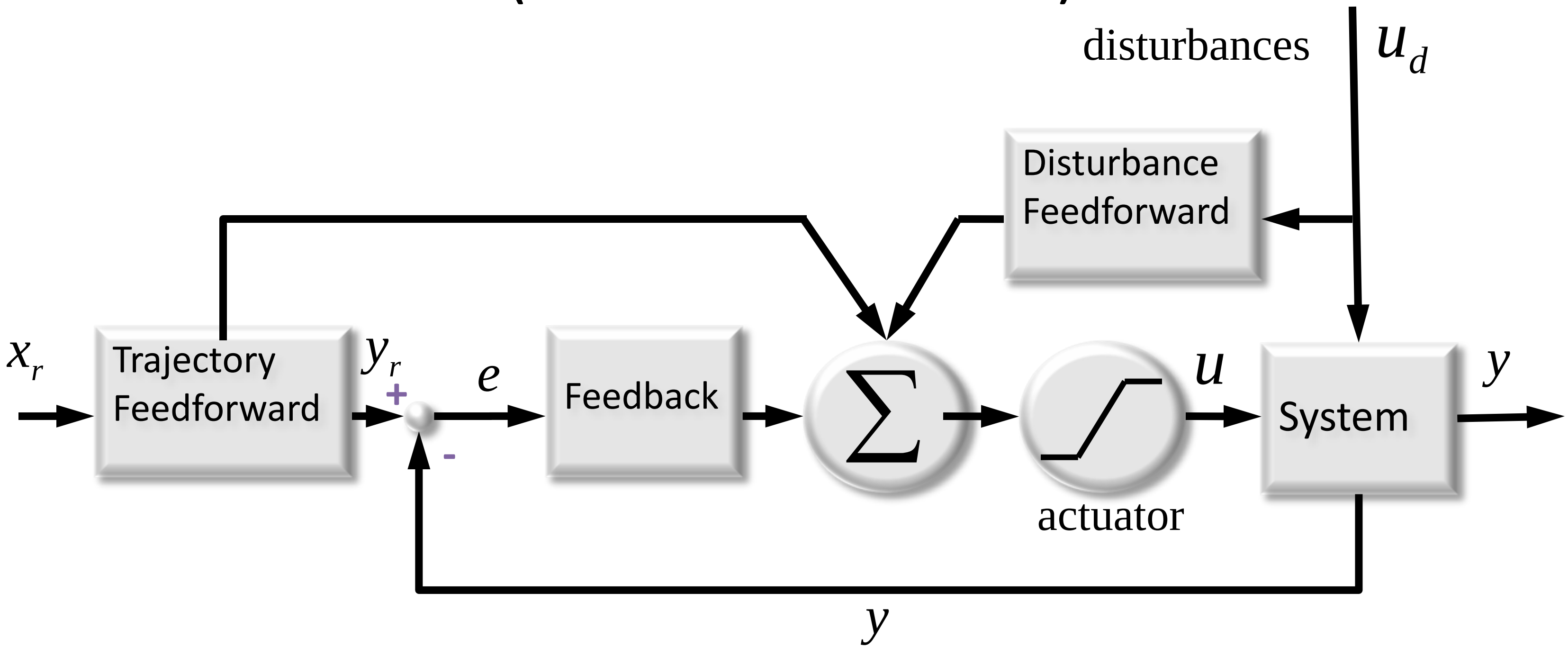
- Feedback alters the dynamics of a system to..
  – do what you want
  – do it in a useful (stable, convergent) way.

7.1.1 Introduction
(General Controller)



- Controllers may
  - Map between signals of interest and those accepted by hardware.
  - Measure what system is doing in order to alter dynamics and/or reject disturbances
  - Elaborate terse goals into the required details.

7.1.1 Introduction
(General Controller)



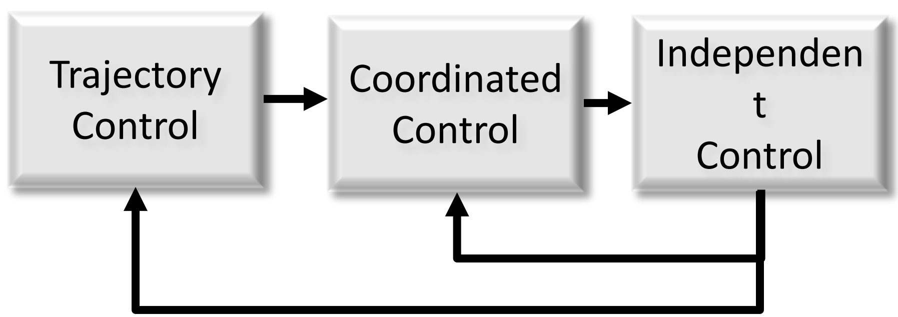
- y_r is the reference signal
- u is the input – the only way to really control the system
- u_d are the disturbances (friction, wind)
- Actuator symbol describes limited amplitude
- e is the error signal

7.1.1.2 Controller Elements

- Regulators try to achieve a specified fixed output (set point).
- Servos try to follow a reference signal.
- Feedback measures system response and it helps reduce the negative impact of
  - Parameter changes
  - Modelling error
  - Unwanted inputs (disturbances)
- Feedforward generates inputs that are independent of the present response.

7.1.1.3 Controller Hierarchy / Cascade

- A hierarchical arrangement of controllers is typical.
- Each layer generates reference signals for the layer below it.
- Each may generate composite feedback for layer above.



7.1.1.3.1 Independent Control Level

- Independent control level (SISO = single input, single output).
  - Control of actuators as independent entities.
  - Based on axis level feedback.
- React simply to the current (and past) error signal. Prediction is limited to computing error derivatives.
- Connected directly to actuators such as engine throttles, electric motors, and hydraulic valves.
  - calibration required of bias, scale etc.
  - basic kinematic transforms may occur.
- The methods of classical control are adequate to implement this layer.

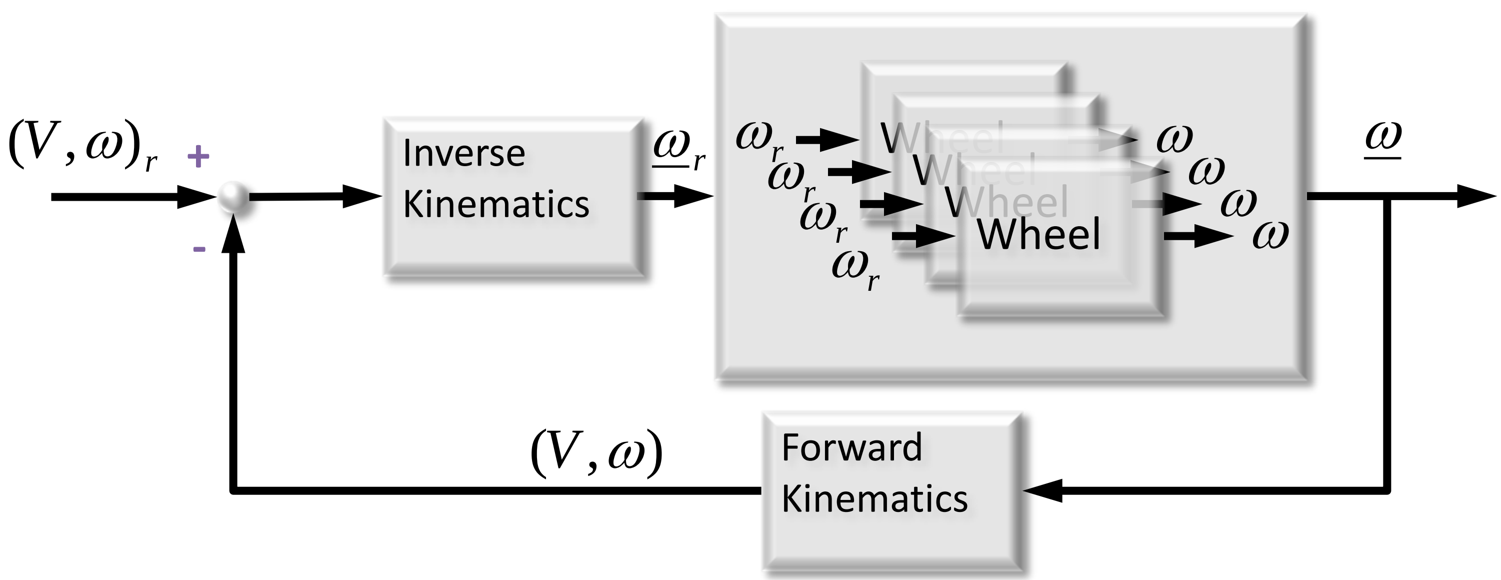
7.1.1.3.2 Coordinated Control Level

- All elements of the state vector are controlled as a unit. Individual axis response must be:
  - consistent: so that their net effect is what is desired.
  - synchronized: so that they have the right values at the right times.
- Based on composite feedback generated from several components.
- Modern state space control methods used here.

7.1.1.3.2 Coordinated Control Level

(Example WMR Coordinated Control)

- Control WMR wheel speeds to achieve a particular V and w.
- Convert wheel speed feedback to V and w.



7.1.1.3.3 Trajectory Control Level

- Considers the entire trajectory over a period of time.
- Normally relies on measurement and/or prediction of the motion of the robot with respect to the environment.
- Examples: driving to a designated pose, following a specified path, or following a lead vehicle or road.
- Much more common to use feedforward and optimal control methods in this layer.
- Layers above here are in perceptive autonomy

7.1.1.4 Controller Requirements

- Move a precise distance or to a precise location:
  - Position control
- Follow a path
  - Crosstrack and alongtrack control may be different
- Gross motion or move at a precise velocity
  - Velocity control

Outline

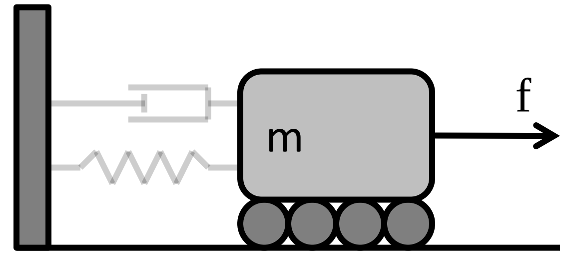
- 7.1 Classical Control
  - 7.1.1 Introduction
  - <u>7.1.2 Virtual Spring Damper</u>
  - 7.1.3 Feedback Control
  - 7.1.4 Model Referenced and Feedforward Control
  - Summary

Recall: Single Axis Control Loops

- Conduct no lookahead.
- React simply to the current (and past) error signal.
- Not coordinated with other servos that execute simultaneously.
- Connected directly to actuators such as engine throttles, electric motors, and hydraulic valves.
  - calibration required of bias, scale etc.
  - basic kinematic transforms may occur.

7.1.2 Virtual Spring Damper

- Mass is really governed by:



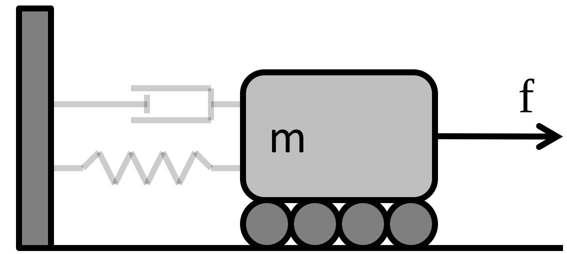
$$\ddot{y} = u(t)$$

- Not clear what u(t) will drive to a specific place y_{ss} for a constant input u_{ss}.
- A real mass-spring-damper will go to a specific place.
- Add measurements of position and speed and a computational spring and damper.

$$u(t) = \frac{f}{m} - \frac{c_c}{m}\dot{y} - \frac{k_c}{m}y$$

7.1.2 Virtual Spring Damper

- Substitute this for u(t):

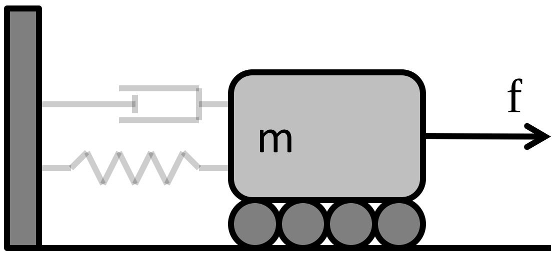


$$\ddot{y} + \frac{c_c}{m}\dot{y} + \frac{k_c}{m}y = \frac{f}{m}$$

- Now the mass behaves like there is a real spring and damper.
  - Goes to exactly the same place!
- This introduction of computational elements to alter system dynamics is the basic idea of control theory.

7.1.2 Virtual Spring Damper

- Open loop system dynamics



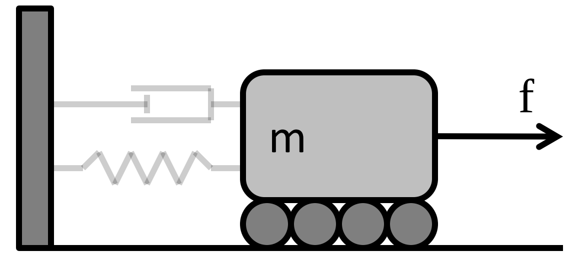
$$\ddot{y} = u(t)$$

- Closed loop system dynamics:

$$\ddot{y} + \frac{c_c}{m}\dot{y} + \frac{k_c}{m}y = \frac{f}{m} \qquad \textbf{Eqn A}$$

- Same as a real spring damper.

7.1.2.1 Stability



- Poles of the closed loop system are the same as the real MSD:

$$s = \omega_0(-\zeta \pm \sqrt{\zeta^2 - 1})$$

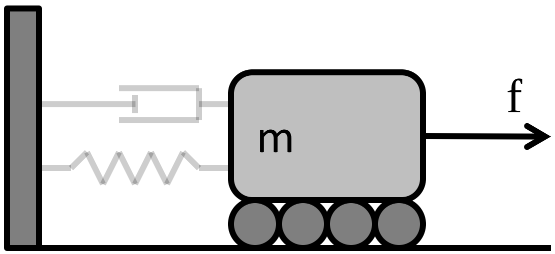
- General solution involves terms of the form:

$$e^{-st} = e^{-\sigma t}e^{-j\omega t} = e^{-\sigma t}[\cos(\omega t) - j\sin(\omega t)]$$

- Real part governs amplitude
- Imaginary part governs frequency
- Therefore stable if real parts are < 0.
  – Friction would always stabilize a real system.

7.1.2.2 Pole Placement

- Consider now changing the behavior of a real MSD system:



$$\ddot{y} + \frac{c}{m}\dot{y} + \frac{k}{m}y = u(t)$$

- Add sensors, compute a control:

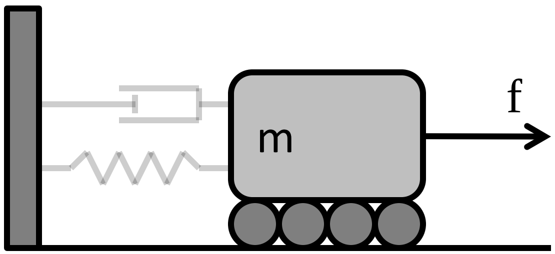
$$u(t) = \frac{f}{m} - \frac{c_c}{m}\dot{y} - \frac{k_c}{m}y$$

Feedback System can have ANY poles we desire!

- Substitute back:

$$\ddot{y} + \frac{(c + c_c)}{m}\dot{y} + \frac{(k + k_c)}{m}y = \frac{f}{m}$$

7.1.2.3 Error Coordinates



- Define the error signal:

$$e(t) = y_r(t) - y(t)$$

- Substitute for y in Eqn A:

$$[\ddot{y}_r - \ddot{e}] + \frac{c_c}{m}[\dot{y}_r - \dot{e}] + \frac{k_c}{m}[y_r - e] = \frac{f_r}{m}$$

- For a constant reference input

$$\ddot{y}_r = \dot{y}_r = 0$$

- Move y_r to RHS:

$$[-\ddot{e}] + \frac{c_c}{m}[-\dot{e}] + \frac{k_c}{m}[-e] = \frac{f_r}{m} - \frac{k_c}{m}[y_r]$$

- But $k_c y_r = f_r$ so:

$$\ddot{e} + \frac{c_c}{m}\dot{e} + \frac{k_c}{m}e = 0$$

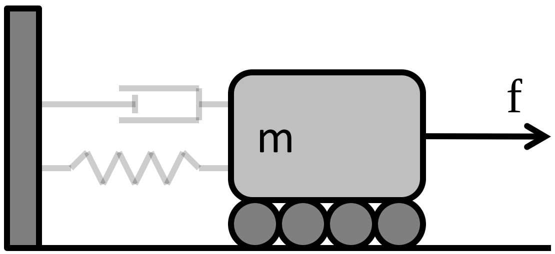
Error dynamics are the same as that of a damped oscillator

7.1.2.3 Error Coordinates

(Control in Error Coordinates)

- Last result suggests this control:

$$u(t) = \frac{c_c}{m}\dot{e} + \frac{k_c}{m}e$$



- Substitute into Eqn A:

$$\ddot{y} = \frac{c_c}{m}\dot{e} + \frac{k_c}{m}e = \frac{c_c}{m}[\dot{y}_r - \dot{y}] + \frac{k_c}{m}[y_r - y]$$

- But $\dot{y}_r = 0$ and $k_c y_r = f_r$ so this is:

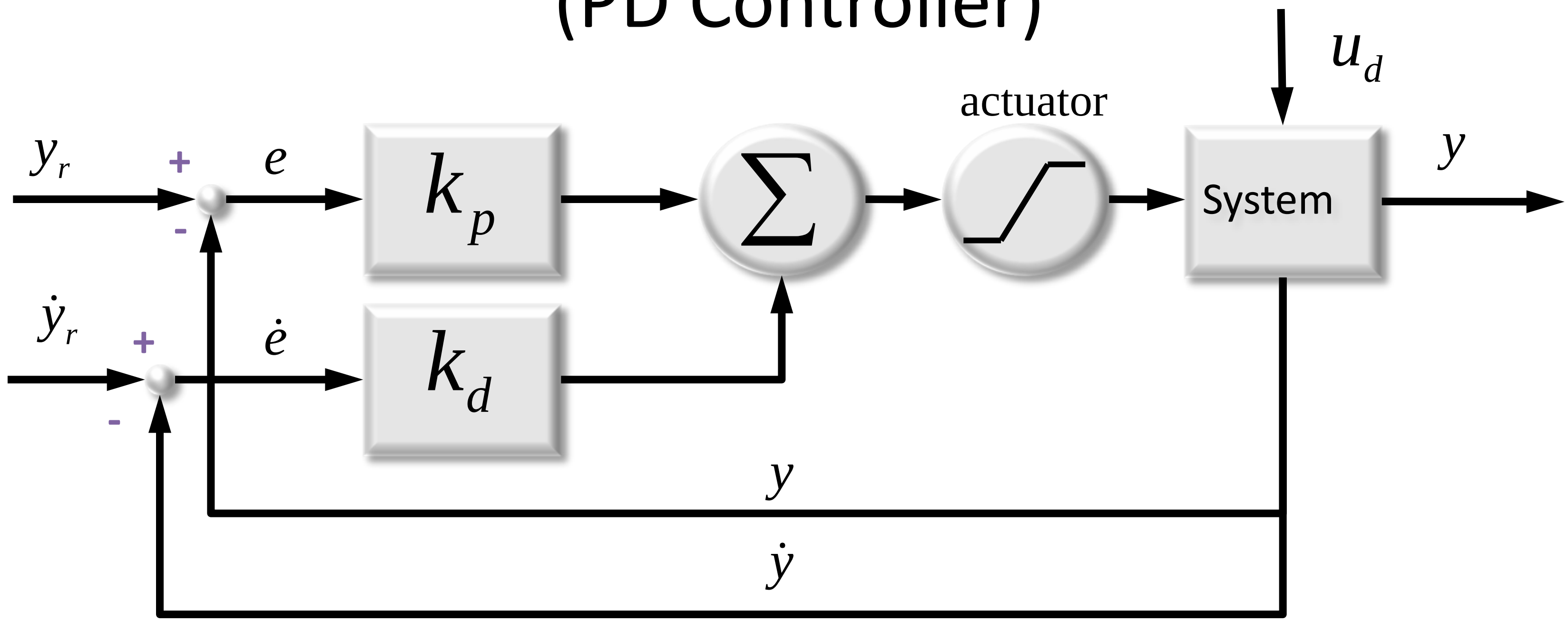
$$\ddot{y} + \frac{c_c}{m}\dot{y} + \frac{k_c}{m}y = \frac{f_r}{m}$$

Controlling the error dynamics like a MSD makes the system behave like a MSD

Outline

- 7.1 Classical Control
  - 7.1.1 Introduction
  - 7.1.2 Virtual Spring Damper
  - <u>7.1.3 Feedback Control</u>
  - 7.1.4 Model Referenced and Feedforward Control
  - Summary

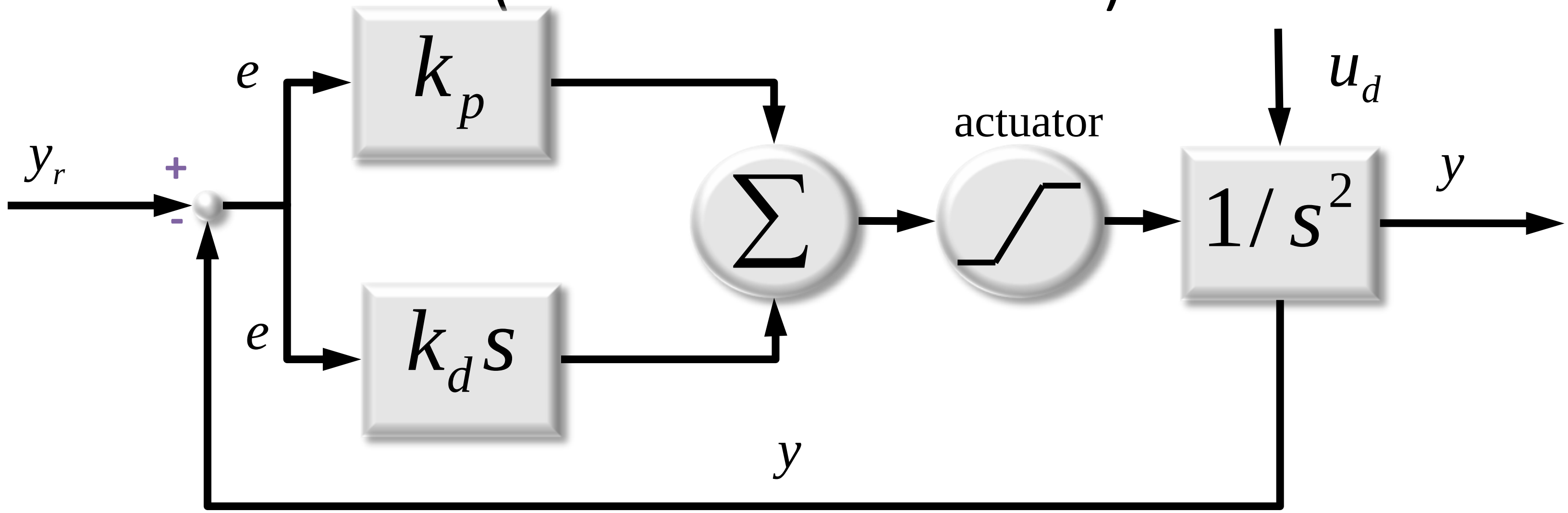
7.1.3 Feedback Control
(PD Controller)



- Functions like a MSD
- Steady state response is y_r.
  - Goes where you tell it to go.

7.1.3 Feedback Control

(PD Transfer function)



- Based on that block diagram trick:

$$T(s) = \frac{H}{1+GH} = \frac{(1/s^2)(k_d s + k_p)}{1+(1/s^2)(k_d s + k_p)} = \frac{k_d s + k_p}{s^2 + k_d s + k_p}$$

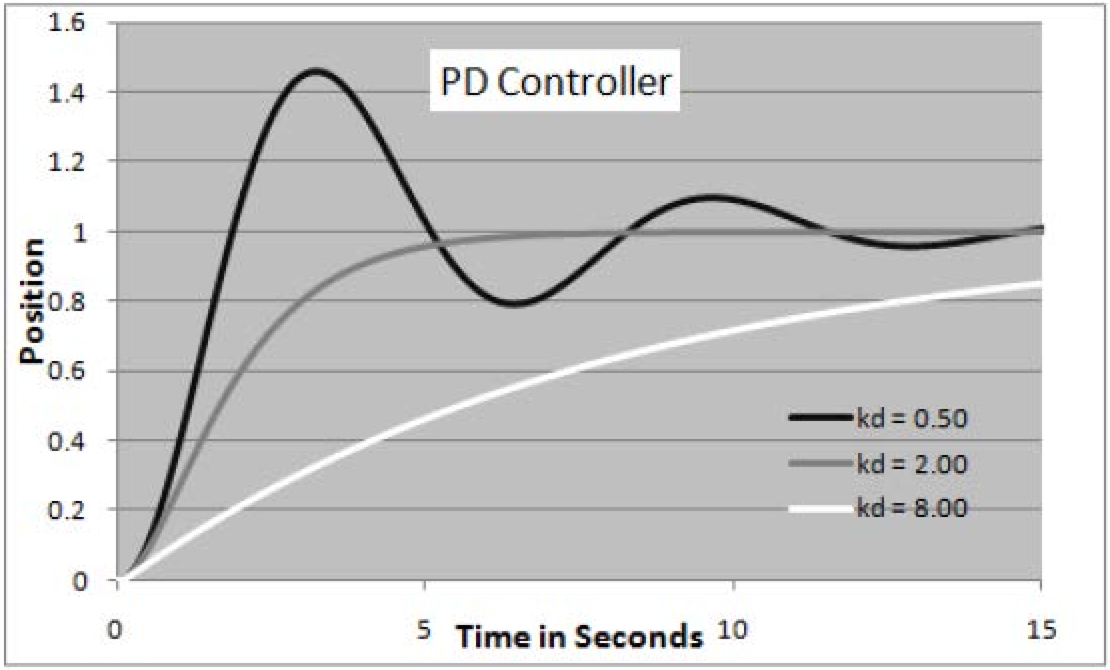
- For a unit mass:

$$k_d = 2\zeta\omega_0 \qquad k_p = \omega_0^2$$

- Close loop poles $s = -\zeta\omega_0 \pm \omega_0\sqrt{(\zeta^2 - 1)} = -\frac{k_d}{2} \pm \frac{1}{2}\sqrt{(k_d^2 - 4k_p)}$

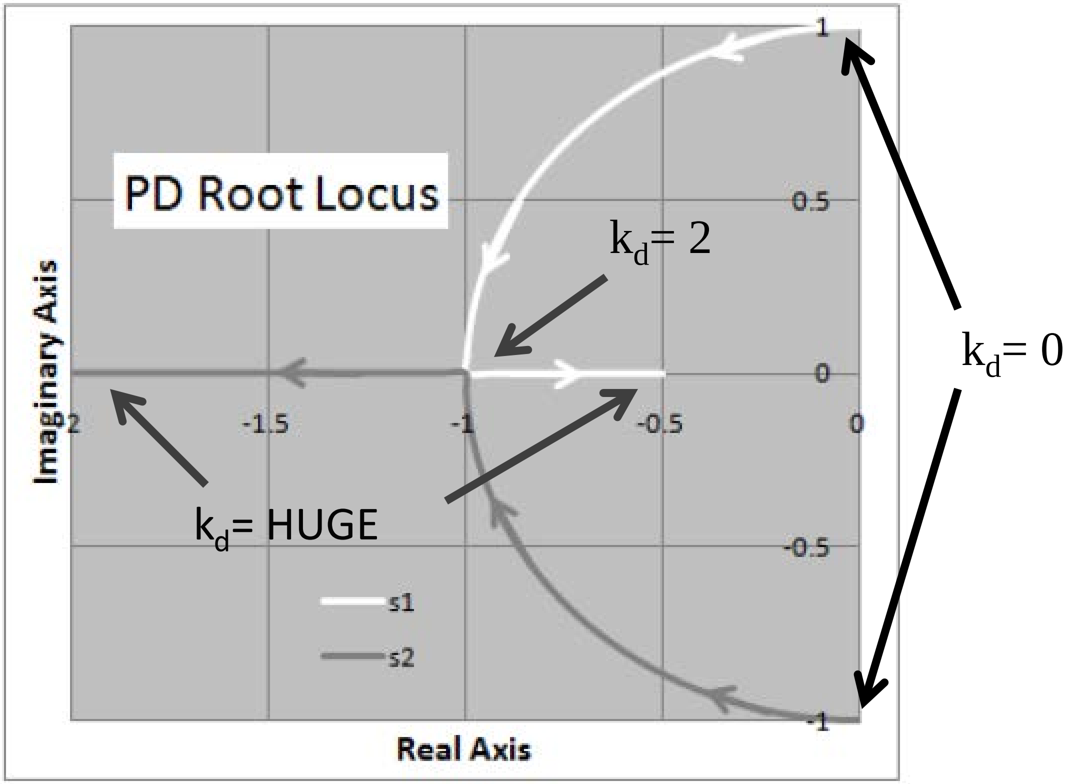
7.1.3 Feedback Control

(PD Loop Response (k_p = 1))



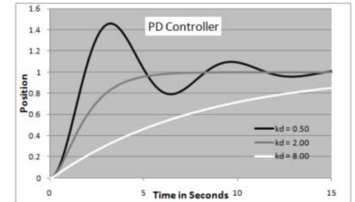
- Critically damped when $k_p = 1$, $k_d = 2$.
- Poles determine damping, oscillation, stability
- Input determines *where it goes* but the poles decide <u>how it gets there</u>.

7.1.3.1 PD Root Locus



- Plot poles as function of some gain.
  - "Dance of the poles" is a common behavior

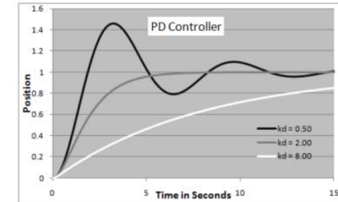
7.1.3.2 Performance Metrics



- 90% rise time
  - time required to achieve 90% of final value.
  - 1.7, 3.9, 18.2 for three responses above
  - time constant is the 63% rise time.
- Percent overshoot:
  - Overshoot amplitude / final value
  - 45.7% for 1st above, 0 for others.

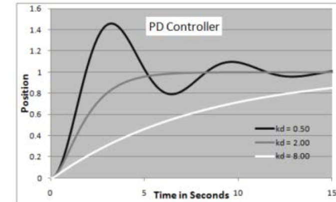
Carnegie Mellon
THE ROBOTICS INSTITUTE

7.1.3.2 Performance Metrics



- 2% settling time
  - time required to settle within 2% of final value.
  - typically 4 time constants
- Steady state error:
  - Error after all transients have faded

7.1.3.3 Derivative Term Issues

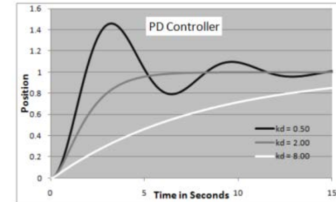


- Derivatives magnify noise.
  - Hence its best not to differentiate the position feedback.
- Alternatives
  - Filter out high frequencies before differentiating.
  - Use measurements of velocity. Works because:

$$e(t) = y_r(t) - y_r(t)$$

$$\dot{e}(t) = \dot{y}_r(t) - \dot{y}(t)$$

7.1.3.4 PID Control



- In PD we have:
  - proportional (now)
  - derivative (future)
- Is integral (past) of any use?
- You betcha. The default answer in industry:

$$u(t) = k_d \dot{e} + k_p e + k_i \int e(t) dt$$

Integral Gain

7.1.4.3 PID Control

- Suppose we have friction in the system. If so:

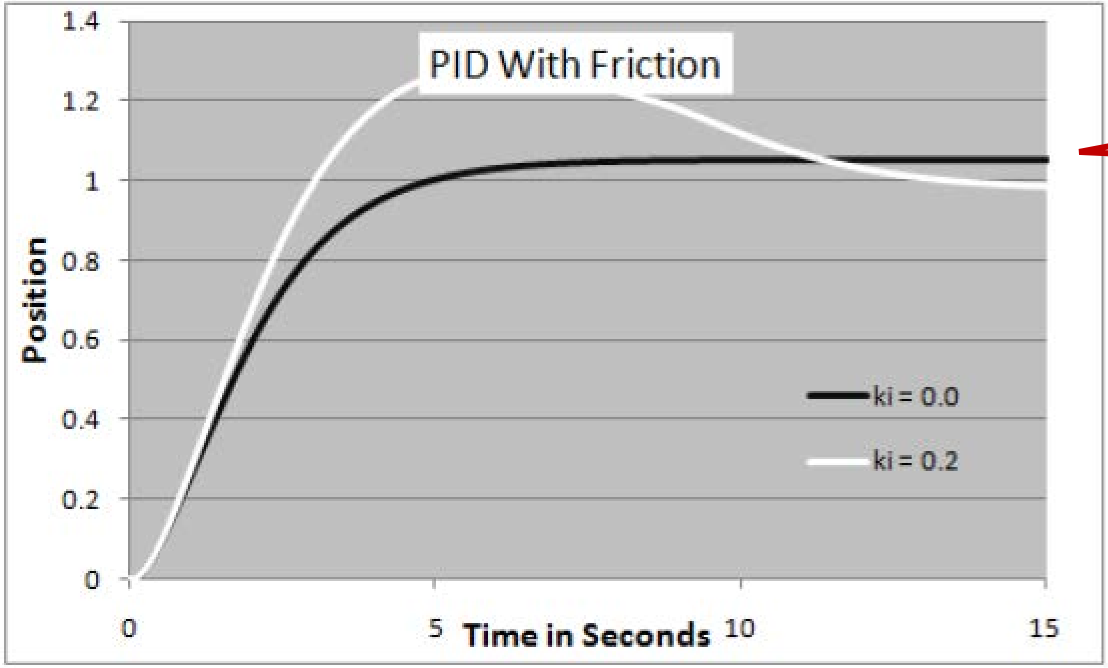
$$\ddot{y} + \frac{c_c}{m}\dot{y} + \frac{k_c}{m}y = \frac{f_r + f_s}{m}$$

- So, steady state solution is:

$$y_{ss} = \left(\frac{f_r + f_s}{k_c}\right)$$

- It does not go to the right place.
- However, the integral gain in the PID removes this error!

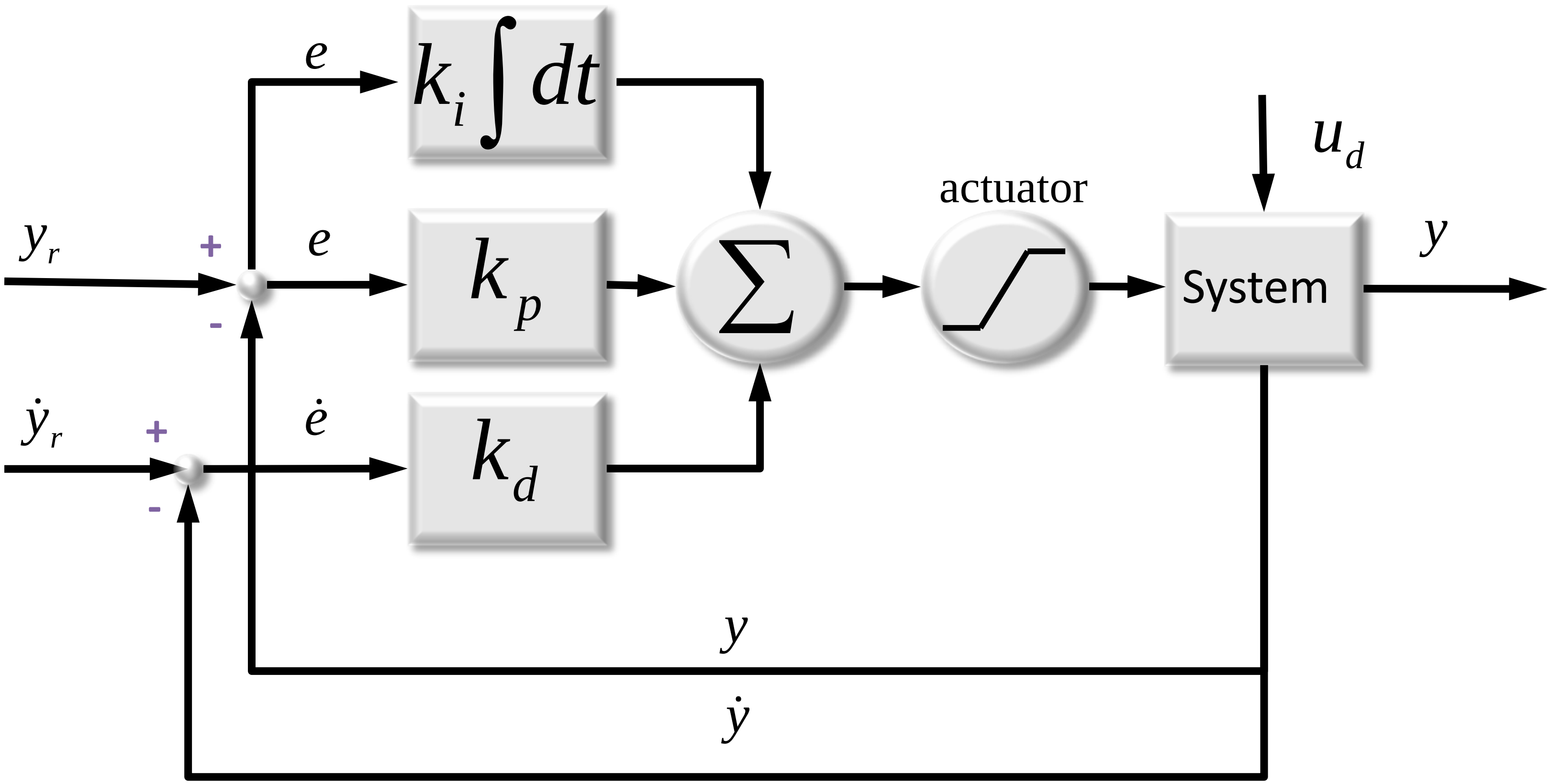
7.1.3.4 PID Control



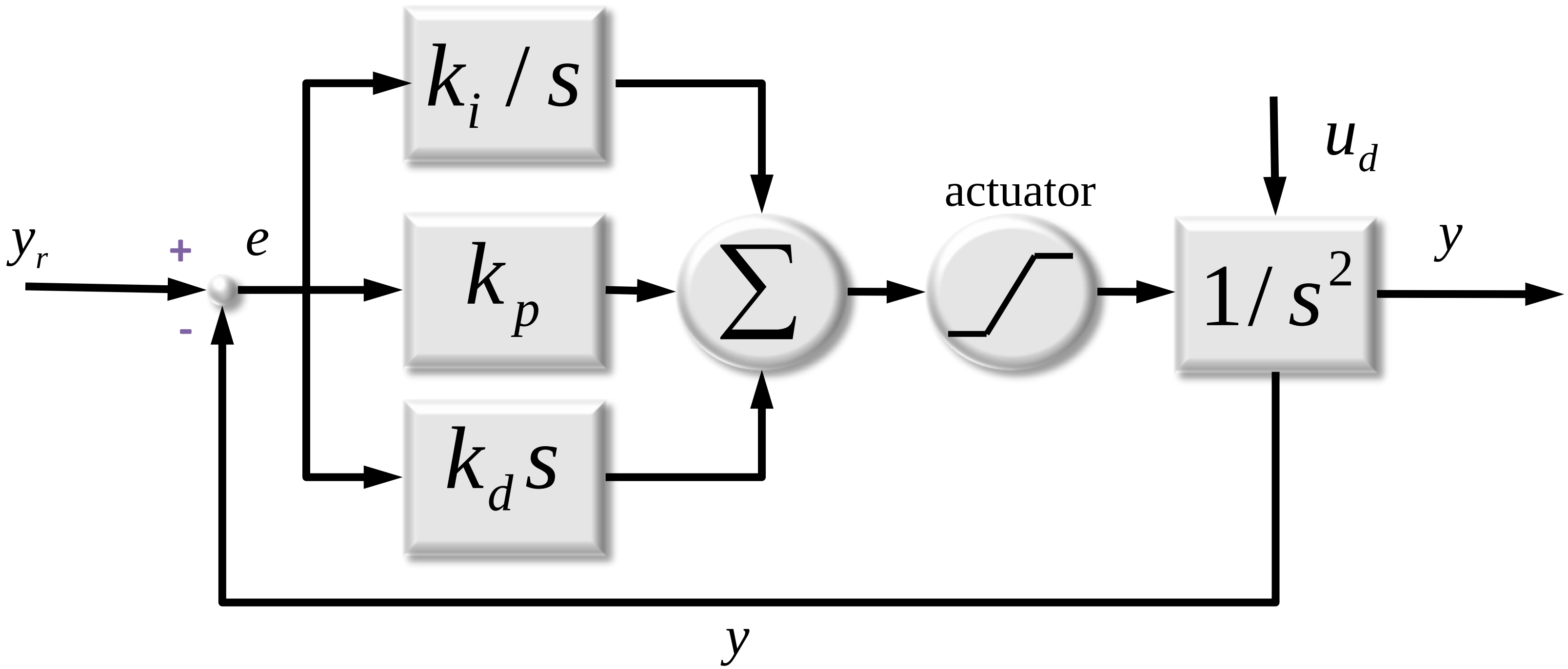
Steady State Error

- The I term builds up for persistent errors.

PID Block Diagram (time domain)



PID Block Diagram (s domain)



$$T(s) = \frac{H}{1+GH} = \frac{(1/s^2)(k_d s + k_p + k_i/s)}{1+(1/s^2)(k_d s + k_p + k_i/s)} = \frac{k_d s^2 + k_p s + k_i}{s^3 + k_d s^2 + k_p s + k_i}$$

7.1.3.5 Integral Term Issues

- Growth of the integral term is called windup.
  - It can be disasterous.
- Has the capacity to output maximum control effort for an extended period of time.
- Moral:
  - Enforce a threshold on its magnitude.
  - Clear it when you can detect that the loop has been opened.

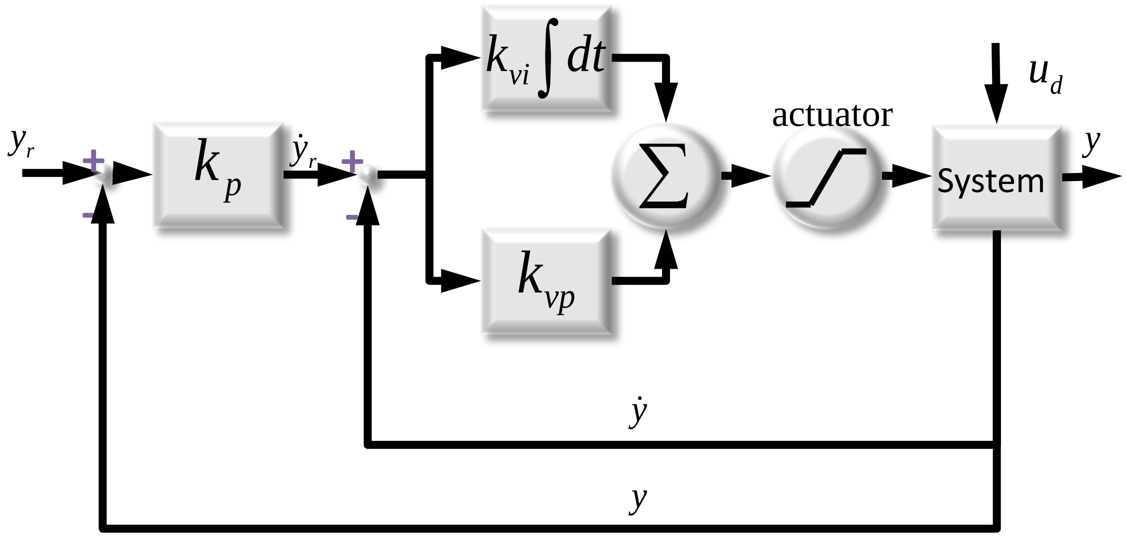
PID Loops (Summary)

- Proportional Term
  - Corrects for the present value of the error.
  - Kp is often called servo “stiffness”.
- Integral Term
  - Corrects for persistent (average) errors [also known as dc offset].
- Derivative Term:
  - Corrects for predicted future errors
  - Predictive element

PID Loops

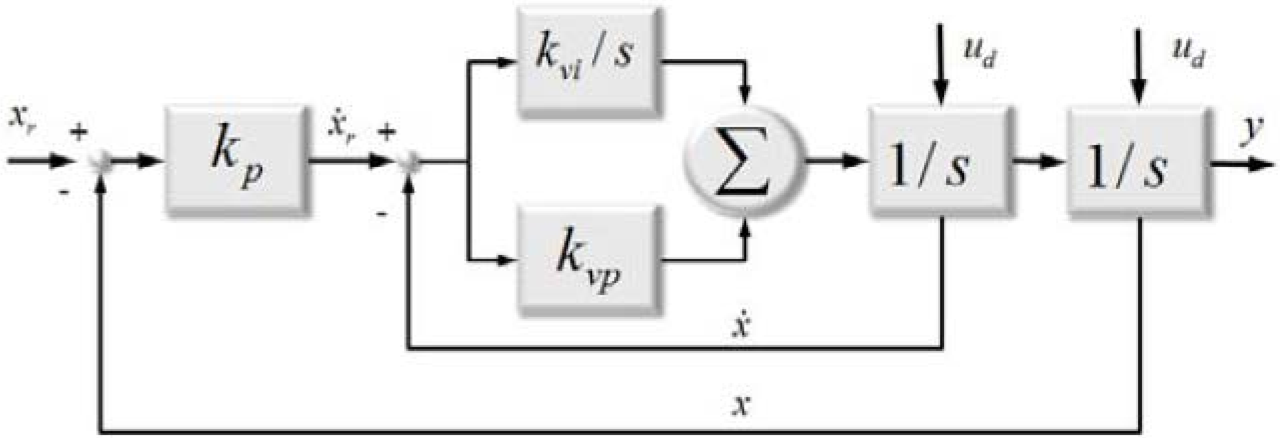
- Can adjust process inputs based on the error, error history and error rate
  - which gives more accurate and stable control
- Can be used to control any measurable variable which can be affected by manipulating some other process variable.

7.1.3.6 Cascade Control



- Position loop around a velocity loop.
- Maybe 2nd most common in industry.
- May be forced on you by e.g. a motor.
- Inner loop tries to remove velocity error quickly.
- Because it's a hierarchy of loops, it applies in abstract way to all robots.

7.1.3.6 Cascade Control



- Inner loop:

$$T_v(s) = \frac{H_v}{1+G_vH_v} = \frac{(1/s^2)(k_{vp}s+k_{vi})}{1+(1/s^2)(k_{vp}s+k_{vi})} = \frac{k_{vp}s+k_{vi}}{s^2+k_{vp}s+k_{vi}}$$

- Outer loop (using inner as the system)

$$T(s) = \frac{H}{1+GH} = \frac{(1/s)k_pT_v(s)}{1+(1/s)k_pT_v(s)} = \frac{k_p[k_{vp}s+k_{vi}]}{s^3+k_{vp}s^2+[k_{vi}+k_pk_{vp}]s+k_pk_{vi}}$$

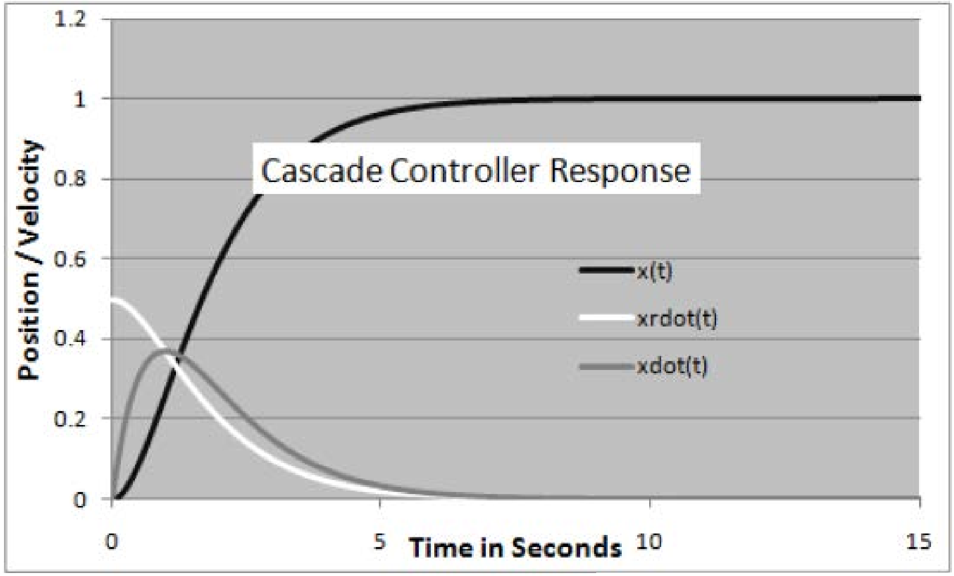
- For $k_{vi}=0$

$$T(s) = \frac{k_pk_{vp}}{s^2+k_{vp}s+k_pk_{vp}}$$

Denom is same as PD

7.1.3.6 Cascade Control

(Cascade Control Response)



- Repeated pole at -1 when $k_{vp} = 2\,,\, k_p = 0.5$
- This configuration responds like a PD.
- Still takes 5 seconds to get there. Hmmm.

Online Tuning of PID Loops

- 1) Set K_i and K_d to zero.
- 2) Increase K_p until the output oscillates.
- 3) Increase K_i until oscillation stops.
- 4) Increase K_d until the loop is acceptably quick to reach its reference.
- A fast PID loop usually overshoots slightly to reach the setpoint more quickly.

Offline Tuning

- Use system identification techniques to determine the coefficients of the differential equations in the system model.
- Then there are formulas for the optimal gains.
- Can also just play around in simulation in this case.

Performance Issues

- PIDs etc. respond violently to a step input.
  - This creates momentum which causes overshoot.
  - This mean gains must be kept low to maintain stability.
  - That means sluggish response.

7.1.4.2 Limitations of Feedback-Only Controls

- Delayed response to errors
  - must wait for errors to occur before removing them.
  - Yet, sometimes they can be predicted
- Coupled Response
  - Ideally manipulate the response to the reference differently from errors.

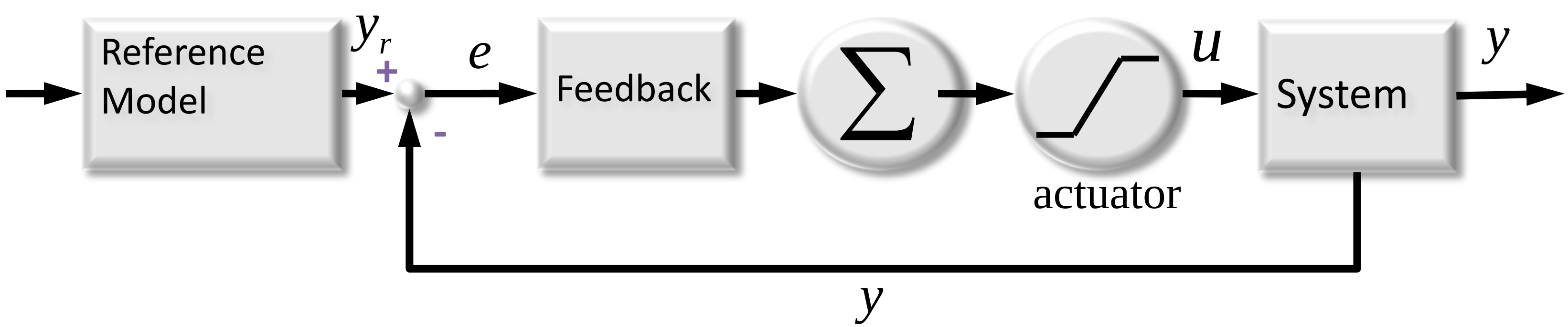
Outline

- 7.1 Classical Control
  - 7.1.1 Introduction
  - 7.1.2 Virtual Spring Damper
  - 7.1.3 Feedback Control
  - <u>7.1.4 Model Referenced and Feedforward Control</u>
  - Summary

7.1.4.1 Model Referenced Control

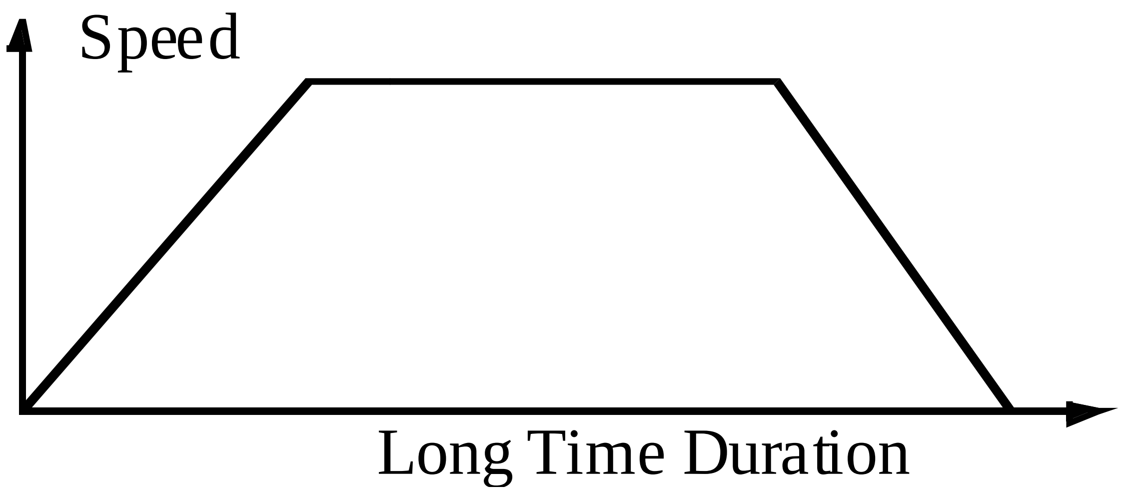
- Is there a better way? Yes.
  - Generate a feasible trajectory to the goal.
  - Use that as the reference.
- Don’t trust the PID to generate the trajectory.
- Tell the controller…
  - Not only where to go but…
  - how to get there.

7.1.4.1 Model Referenced Control



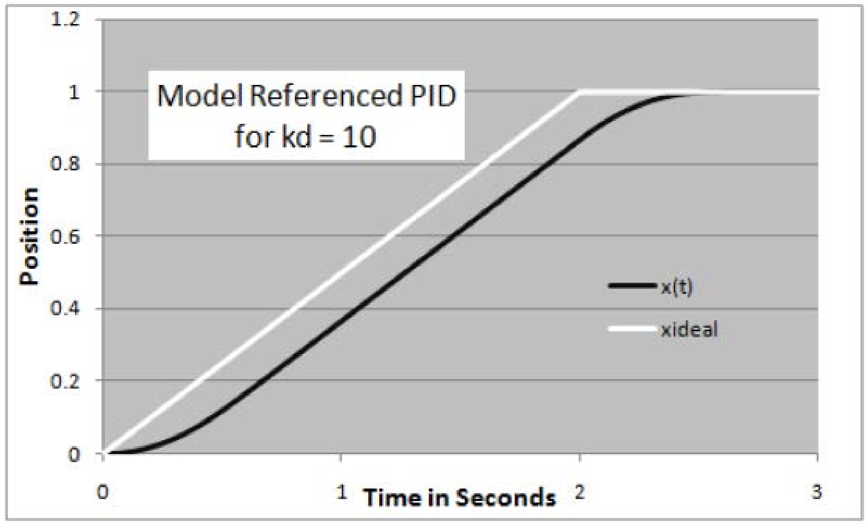
- Makes it possible to:
  - Raise the gains
  - Improve response
- MUST measure errors wrt new reference trajectory.

Common Velocity Reference Trajectory



- Integrate or differentiate as necessary to get the other signals.
- Two views:
  – A reference model generates the trajectory.
  – A less infeasible trajectory is generated however.

Model Referenced Control Response



- Twice as fast to the goal.
- Much higher gain.
- Turns out…. STILL not optimal.

Carnegie Mellon THE ROBOTICS INSTITUTE

7.1.4.2 Limitations of Feedback

- Delayed response to errors.
  - Literally waits for them to happen and then responds.
  - Even though components due to changes in reference signal are totally predictable.
- Coupled response
  - Response to reference and errors uses same mechanism – error feedback.

7.1.4.3 Feedforward Control

(Open Loop Bang-Bang Control)

- Open loop is bad right? Only sorta.
- Position for constant force input.

$$y(t) = \frac{1}{2}\left(\frac{f}{m}\right)t^2$$

- Time required to travel to position yr is:

$$t = \sqrt{2\frac{m}{f_{max}}y_r}$$

- However, it will arrive at high speed and overshoot.

7.1.4.3 Feedforward Control

(Open Loop Bang-Bang Control)

- Need to reverse force at the halfway point.

$$t_{mid} = \sqrt{\frac{m}{f_{max}} y_r}$$

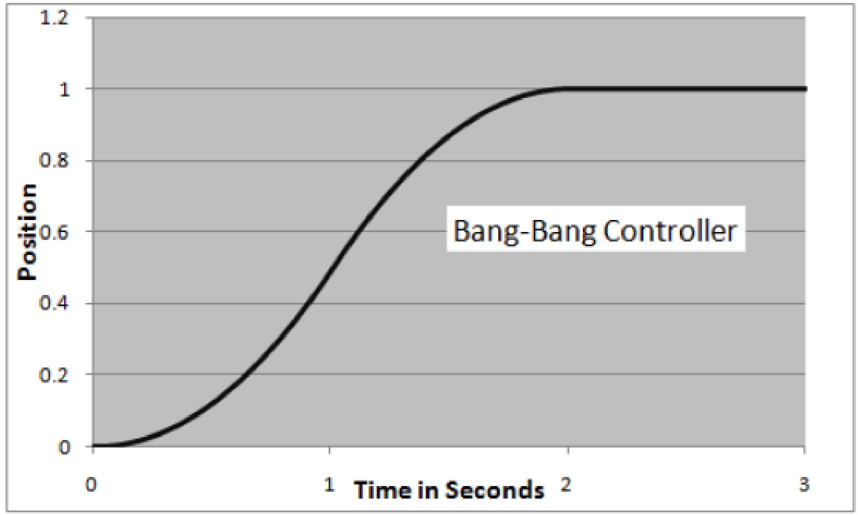
- Now, we have defined this control law:

$$u_{bb}(t) = \begin{bmatrix} f_{max} \text{ if } t < t_{mid} \\ -f_{max} \text{ if } t_{mid} < t < 2t_{mid} \\ 0 \text{ otherwise} \end{bmatrix}$$

- Any control that switches between extremes like this is called a bang-bang controller.

7.1.4.3 Feedforward Control

(Bang-Bang Response)



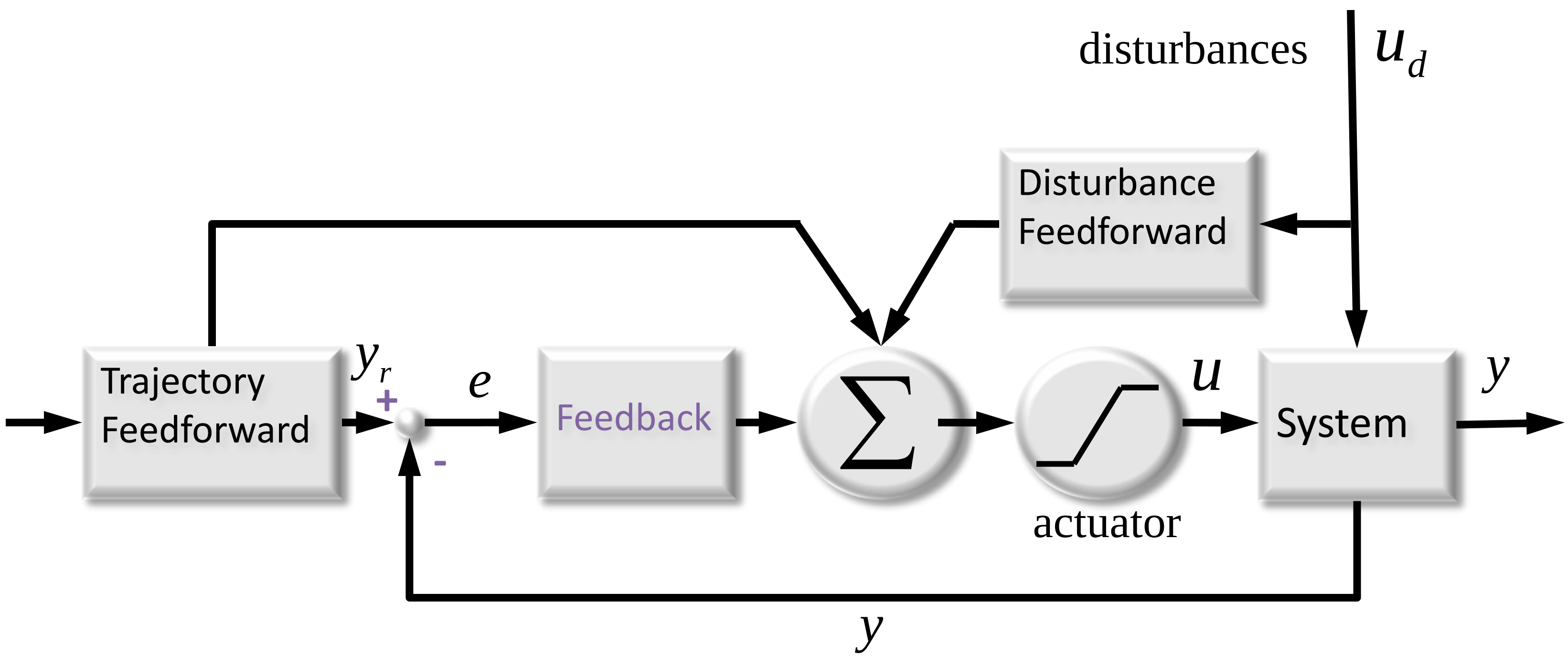
- Gets there in 2 seconds flat. That is the minimum possible.
- No overshoot.

The end of feedback?

- Feedback:
  - Underperforms
  - Does not remove errors quickly (gains too low)
  - Is potentially unstable
  - Requires expensive finicky sensors.
- Thy name is a swear word.
- Thou art abolished.

The Rebirth of Feedback

- Feedback does remove errors.
- Feedforward is not even aware of them.
- Can we combine them? Yes.



7.1.4.4 Feedforward with Feedback Trim

- For the bang-bang control, the reference trajectory is:

$$y_r(t) = \int_0^t \int_0^t u_{bb}(t)dtdt$$

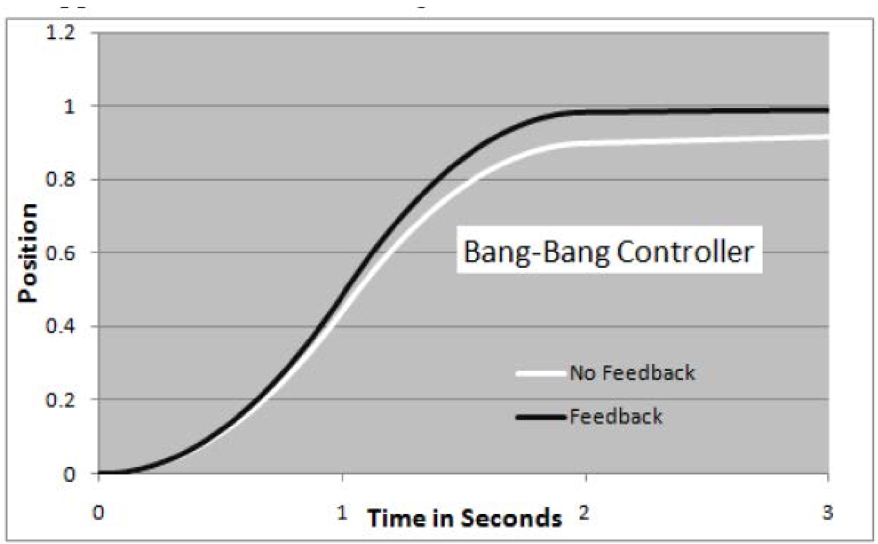
- Use this for <u>computing errors</u>.
- The adjoined control is:

$$u(t) = u_{bb}(t) + u_{fb}(t) = \begin{bmatrix} f_{max} \text{ if } t < t_{mid} \\ -f_{max} \text{ if } t_{mid} < t < 2t_{mid} \\ 0 \text{ otherwise} \end{bmatrix} + k_i \int_0^t e(t)dt + k_d \dot{e}(t)$$

- Use this for computing the force.

7.1.4.4 Feedforward with Feedback Trim

(Response of Composite Controller)



- Even for a massive 10% friction disturbance.
- Gets to the right place.
- Gets there in (almost) record time.
  - Friction adds slight delay

7.1.4.5 Trajectory Generation Problem

- Solve this problem:

$$y_r(t_f) = \int_0^{t_f}\int_0^{t_f} \ddot{y}(u(t), t)\,dt\,dt = y_f$$

$$\dot{y}_r(t_f) = \ddot{y}_r(t_f) = 0$$

- For an input trajectory u(t) and terminal time t_f.

7.1.4.6 Feedback versus Feedforward

	Feedback	Feedforward
Removes Unpredictable Errors and Disturbances	(+) YES	(-) NO
Removes Predictable Errors and Disturbances	(-) NO	(+) YES
Removes Errors and Disturbances Before They Happen	(-) NO	(+) YES
Requires Model of System	(+) NO	(-) YES
Affects Stability of System	(-) YES	(+) NO

Outline

- 7.1 Classical Control
  - 7.1.1 Introduction
  - 7.1.2 Virtual Spring Damper
  - 7.1.3 Feedback Control
  - 7.1.4 Model Referenced and Feedforward Control
  - Summary

Summary

- There are many forms of mobile robot controls
- They can be arranged in a rough hierarchy.
- There is a kind of generic PID loop that covers alot of cases.
- Feedback and feedforward both have their merits.
  - Doing both at once is a good idea.