Control Theory: Introduction and PID

Robotics – COMS4045A / COMS7049A

Benjamin Rosman

Over the next few weeks…

- You will learn how to:
  - determine where the robot is (forward kinematics)
  - determine where you'd like it to be (inverse kinematics)
  - determine how it may move (dynamics)

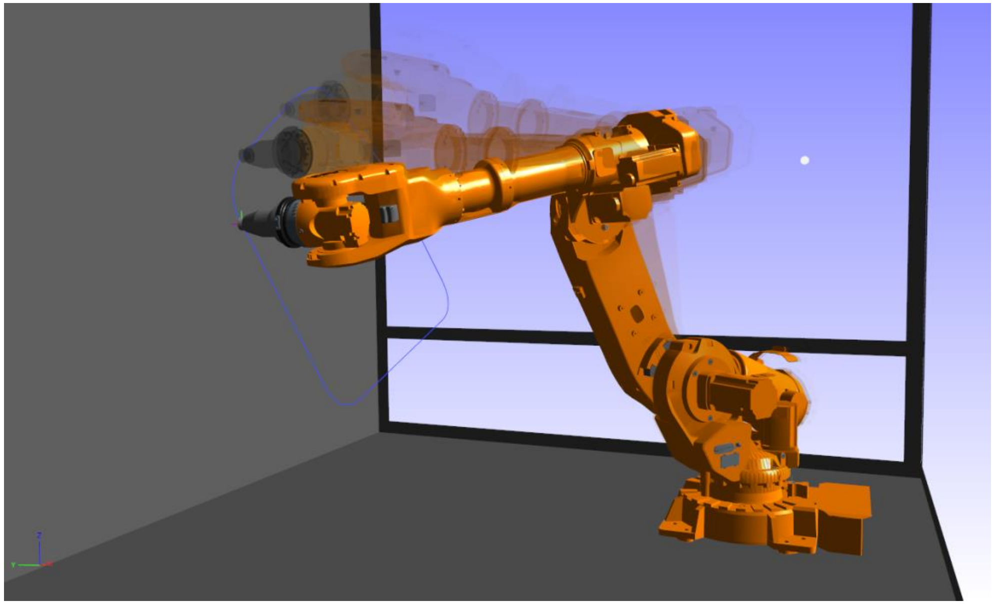
- But now:

  How to get it to move

  From where it is

  To where I want it

  (Control theory)

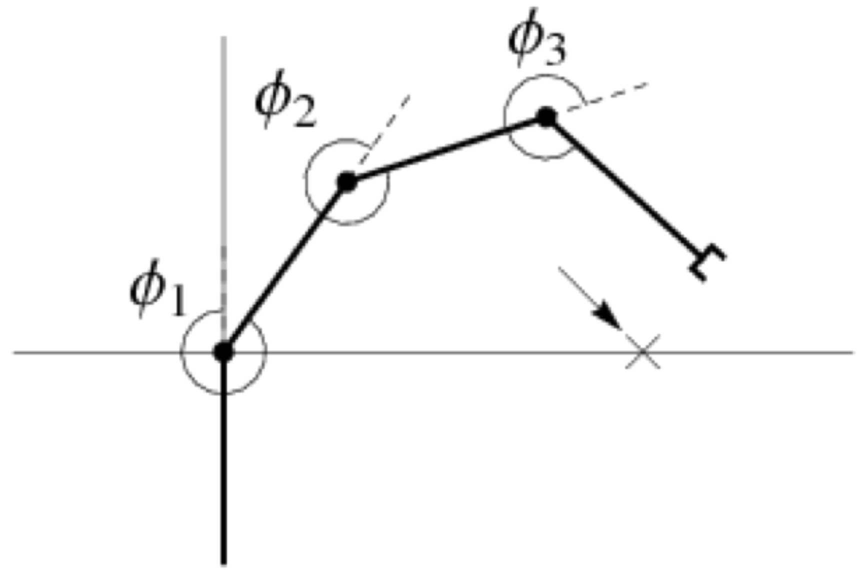


States

- State = robot's representation of the configuration of the world (and itself)
  - Ideally unique
  - Typically don't model things that don't change (e.g. obstacles)
- Why states?
  - We want to know (or learn) what actions (a or u) to take at each state (x or s or q)
  - In continuous systems, state is a function of time:
    - $x = x(t)$
  - Then we get velocity as a derivative:
    - $x'(t) = \dot{x}(t) = \frac{dx}{dt}$

Examples of state

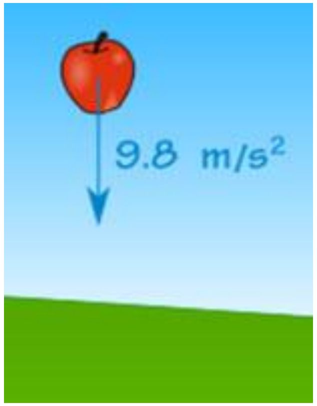
- $x = \{\phi_1, \phi_2, \phi_3\}$
- $x = \{\phi_1, \phi_2, \phi_3, \dot{\phi}_1, \dot{\phi}_2, \dot{\phi}_3\}$
- $x = \{\phi_1, \phi_2, \phi_3, isHandOpen\}$



- $x = \{xPos, yPos\}$
- $x = \{xPos, yPos, direction\}$
- $x = \{xPos, yPos, colour\}$
- $x = \{xPos, yPos, hasKey\}$
- $x = \{\boldsymbol{xAll}, \boldsymbol{yAll}\}$

System dynamics

- System dynamics describe how the state changes:
  - Modelled as differential equations
  - Passively: $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$
  - With time: $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)$
  - Controlled: $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, u)$



- Examples:
  - Newton's 2nd law: $\ddot{x}(t) = F/m$
  - Falling under gravity: $\dot{x}(t) = gt$
  - Population growth: $\dot{x}(t) = Ax(t)(1 - \frac{x(t)}{B})$

Aside: systems of DEs

- General trick:
  - Any higher-order DE can be written as a first-order SYSTEM
- E.g.
  - $\ddot{x} = 3x$
  - Now let $x_1 = x$, and $x_2 = \dot{x}$
  - So: $\dot{x}_1 = x_2$
  - Then: $\dot{x}_2 = 3x_1$
  - As a system: $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- In general: $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$
  - Question: what about $x''' = 2x'' - x + 1$?

And now…

- What we really want is to **tell the robot how to function in a specified manner**
  - When this would not happen naturally
  - System subject to perturbations – maintain stability
- This is control theory
  - In general, find control input u to drive the system to desired states x
  - $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x}, \boldsymbol{u})$
- Much of the core of robotics

Issues to think about: the system

- System may be **linear**, which makes the maths easy:
  - $\dot{\boldsymbol{x}}(t) = A(t)\boldsymbol{x}(t) + B(t)\boldsymbol{u}(t)$
- System may be **time invariant**:
  - $A(t) = A; \quad B(t) = B$
- System is not always **observable**. Often we only have a partial view of the state *x* through some observables *y*:
  - i.e. it can’t see everything (very common)
  - $\boldsymbol{y}(t) = C\boldsymbol{x}(t) + D\boldsymbol{u}(t)$
- Does the system have **delays** in updates, sensing, or control?
  - $\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t - 10)$

Issues to think about: stability

- It is worth considering the **steady state behaviour** of the system
  - i.e. what is the asymptotic behaviour: $\boldsymbol{x}(t \to \infty)$
- This relates to the notion of **stability**:
  - Informally, a system is stable if the behaviour converges (or in a weaker case, stays similar)
- In linear systems, consider the **eigenvalues** of A:
  - $Re(\lambda_A) < 0$?

Controllability

- **Controllability**: is it *possible* to fully control the system (affect each of the *n* state variables of the system)?
  - E.g. $\boldsymbol{x} = [x, \dot{x}, \ddot{x}]^T$ (position, velocity, acceleration)
  - $\boldsymbol{u} = [u, 0, 0]^T$ (we can directly affect the position)
  - $\boldsymbol{u} = [0, 0, u]^T$ (we can directly affect the acceleration)
- Consider continuous linear time-invariant system:
  - $\dot{x}(t) = Ax(t) + Bu(t)$
  - $y(t) = Cx(t) + Du(t)$
  - Compute controllability matrix:
    - $R = [B\ AB\ A^2B\ \ \ldots A^{n-1}B]$
  - System is controllable if full rank: $rank(R) = n$

Intuition: Does repeatedly applying the control policy allow you to change every state independently?

Example

- $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \overset{A}{\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \overset{B}{\begin{pmatrix} 0 \\ 1 \end{pmatrix}} u\ ;\ y = \overset{C}{\begin{pmatrix} 1 & 1 \end{pmatrix}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- Controllable?

$R = [B\ AB\ A^2B\ \ldots A^{n-1}B]$

- $R = \begin{pmatrix} 0 & 0 \\ 1 & 3 \end{pmatrix}$: $rank(R) = 1 < n$ $(\mathrm{n} = 2)$: No!

- $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u\ ;\ y = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- Controllable?
- $R = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$: $rank(R) = 2 = n$: Yes!

Observability

- **Observability**: is it *possible* to fully observe the system (observe each state variable)?
  - E.g. $\boldsymbol{x} = [x, \dot{x}, \ddot{x}]^T$
  - $\boldsymbol{x} = [x, \dot{x}, \ddot{x}]^T, \boldsymbol{y} = x$ (we only observe position)
  - $\boldsymbol{x} = [x, \dot{x}, \ddot{x}]^T, \boldsymbol{y} = \ddot{x}$ (we only observe acceleration)
- Consider continuous linear time-invariant system:
- Compute observability matrix:
    - $O = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix}$
  - System is observable if full rank: $rank(O) = n$

Intuition: Does repeatedly observing the system allow you to see every state variable?

Example

- $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \overset{A}{\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \overset{B}{\begin{pmatrix} 0 \\ 1 \end{pmatrix}} u \,;\, y = \overset{C}{\begin{pmatrix} 0 & 1 \end{pmatrix}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- Observable?
- $O = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$: $rank(O) = 1 < n \ (n = 2)$: No!

$$O = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

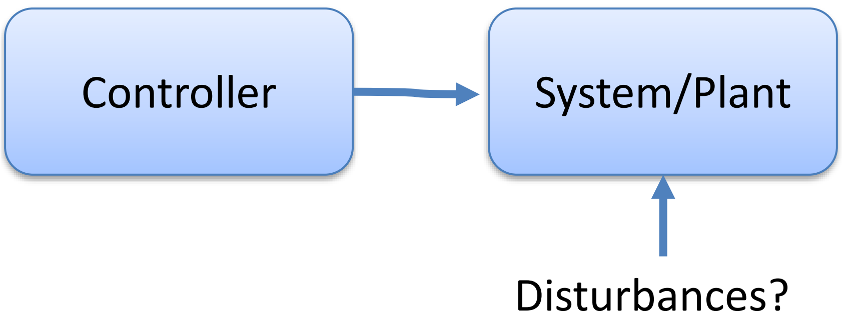
- $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \,;\, y = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- Observable?
- $O = \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix}$: $rank(O) = 2 = n$: Yes!

Issues to think about: the controller

- What kind of control strategy to use?
- Is the strategy **efficient**?
  - Optimal control theory (later)
- Can the controller **sense** some aspect of the system? Can it **respond** to changes?
  - Open loop vs closed loop control
- Can the controller make **predictions** about the environment/system?
  - Model free vs model based control

Open loop control

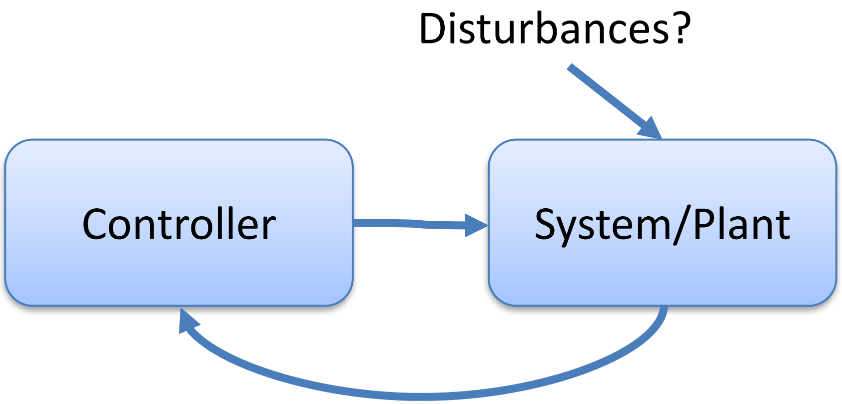
- Open loop control:
  - Policy set and pre-determined (function of time, NOT state)



- Pros:
  - Cheap, simple
  - Fast (feedback and sensing may be too slow)
  - Can be calibrated
- Cons:
  - Policy doesn’t take disturbances into account

Closed loop (feedback) control

- Feedback control:
  - Measure disturbance (error) and use this to adjust control

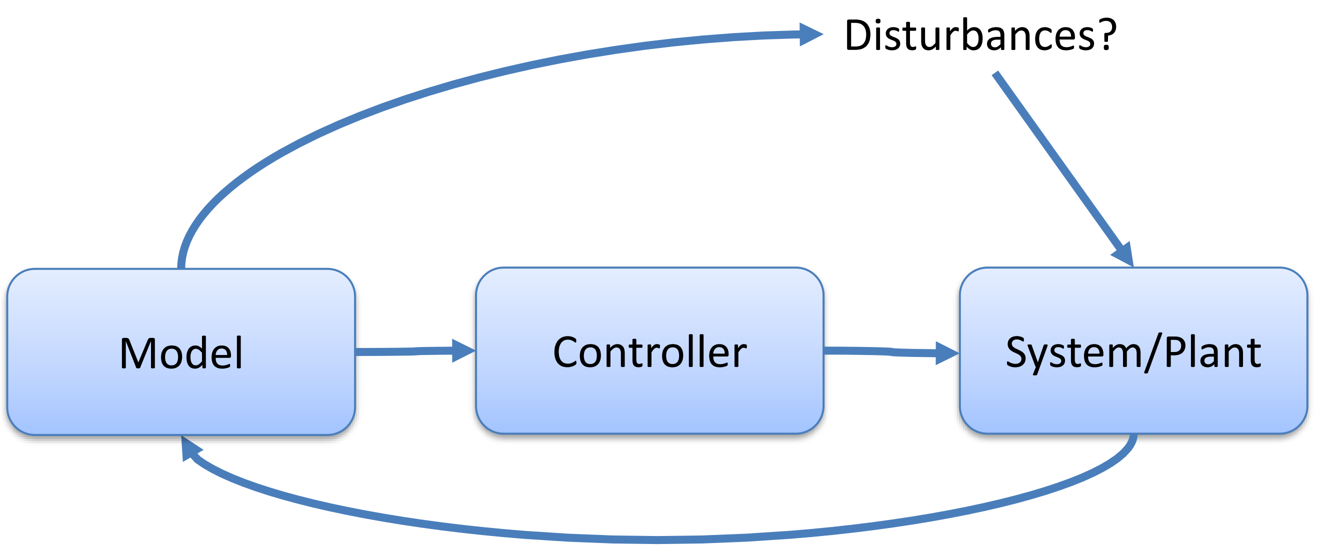


Closed loop (feedback) control

- Pros:
  - Simple
  - Doesn’t require process model
  - Robust output for unpredictable disturbances
- Cons:
  - Requires sensors to measure output
  - Requires tuning: low gain slow, high gain unstable
  - Delays in feedback produce oscillations

Model based control

- Model predictive control:
  - Maintain a process model, make predictions and act accordingly



Model based control

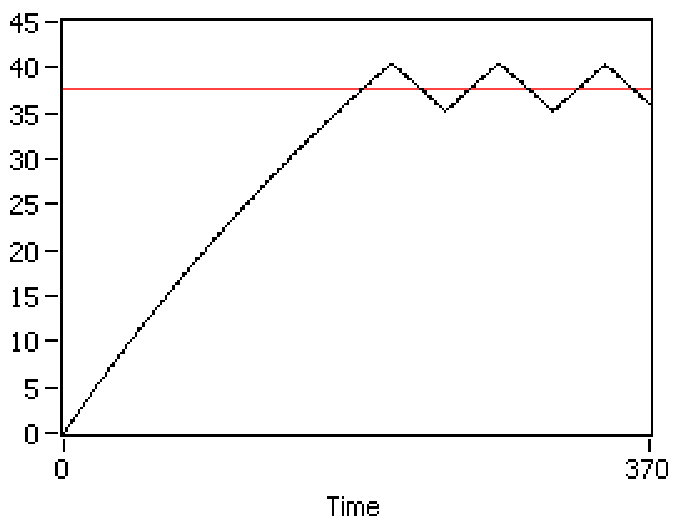
- Pros:
  - Anticipate disturbances and effects of actions, plan to account for these
  - Plan long action sequences
  - Prevent potential problems
  - Robust responses to disturbances
- Cons:
  - Slow
  - Difficult to acquire and maintain a model
    - Traditionally provided by expert
    - Learning system (more on this later)

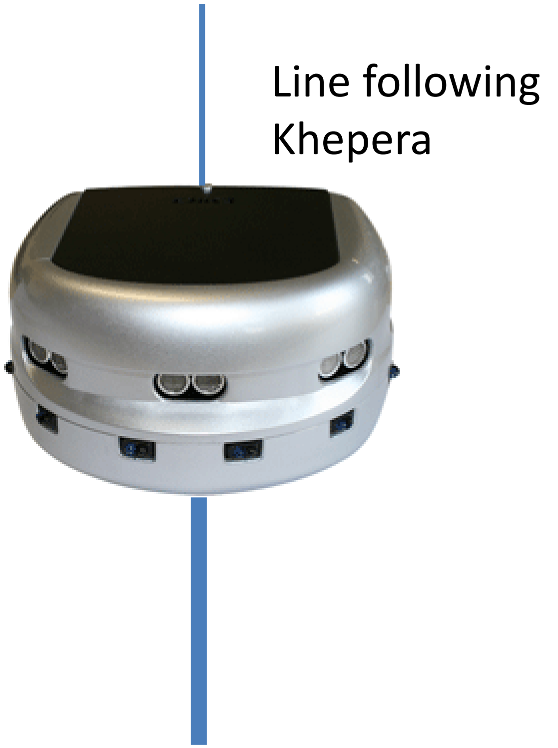
e.g. Room heating

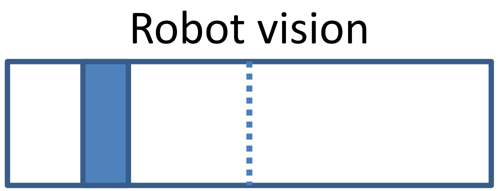
- Heater to increase room temperature
- Open loop:
  - Switch heater on, after some pre-defined time, switch off
- Feedback:
  - Use thermometer in room to switch off at desired temperature
- Model based:
  - Make predictions based on the number of people in the room, outside temperature, doors and windows

Example – line following

- Bang-bang control (on-off control)
  - Switch between extremes
- Controller:
  - Turn left if line is to the left
  - Turn right if line is to the right





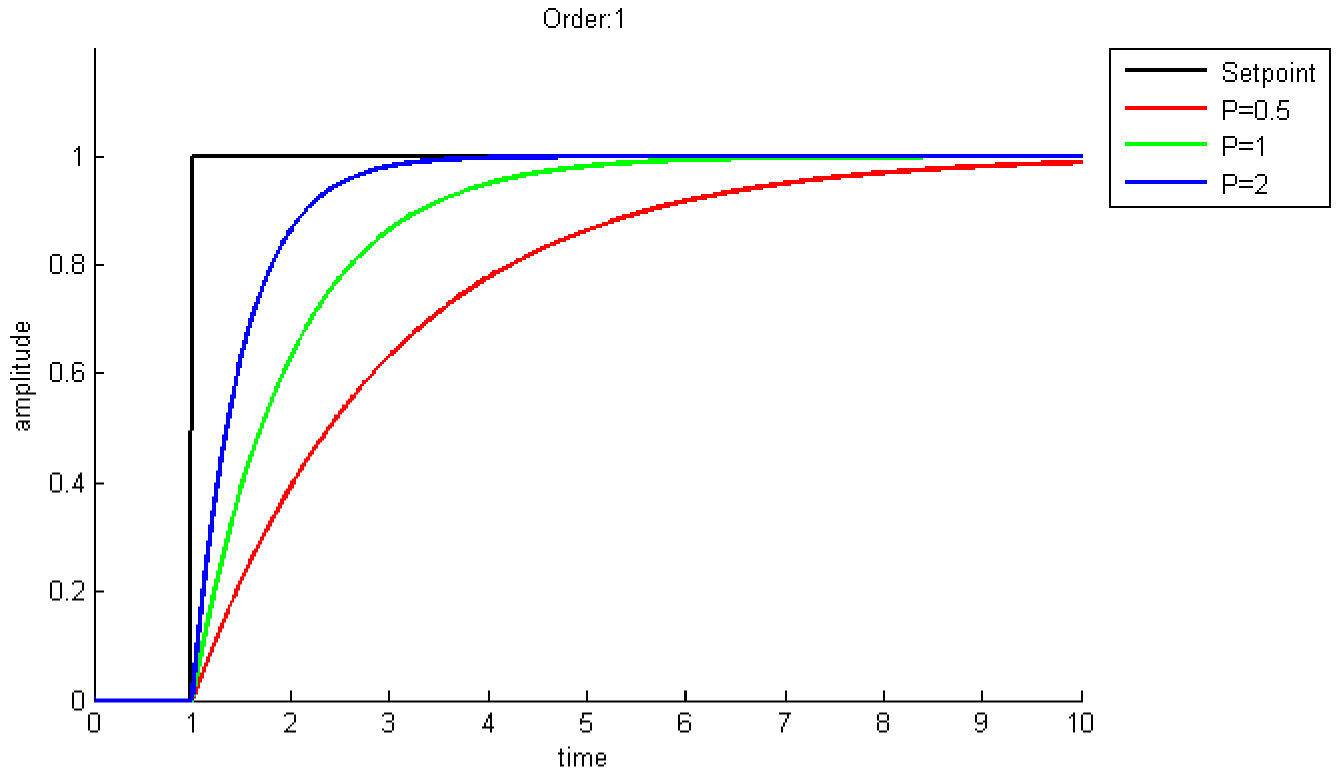


Proportional error control

- Idea: move proportionally left or right
  - Proportional control
- Let the desired state (setpoint) be x_{goal}
- Then, error $e(t) = x_{goal} - x(t)$
- Response output: $K_P e(t)$
- K_P is the proportional gain
  - Too low: slow response
  - Too high: overshoot (unstable)
  - Problem: undershoots (steady state error) when there is a steady disturbance!

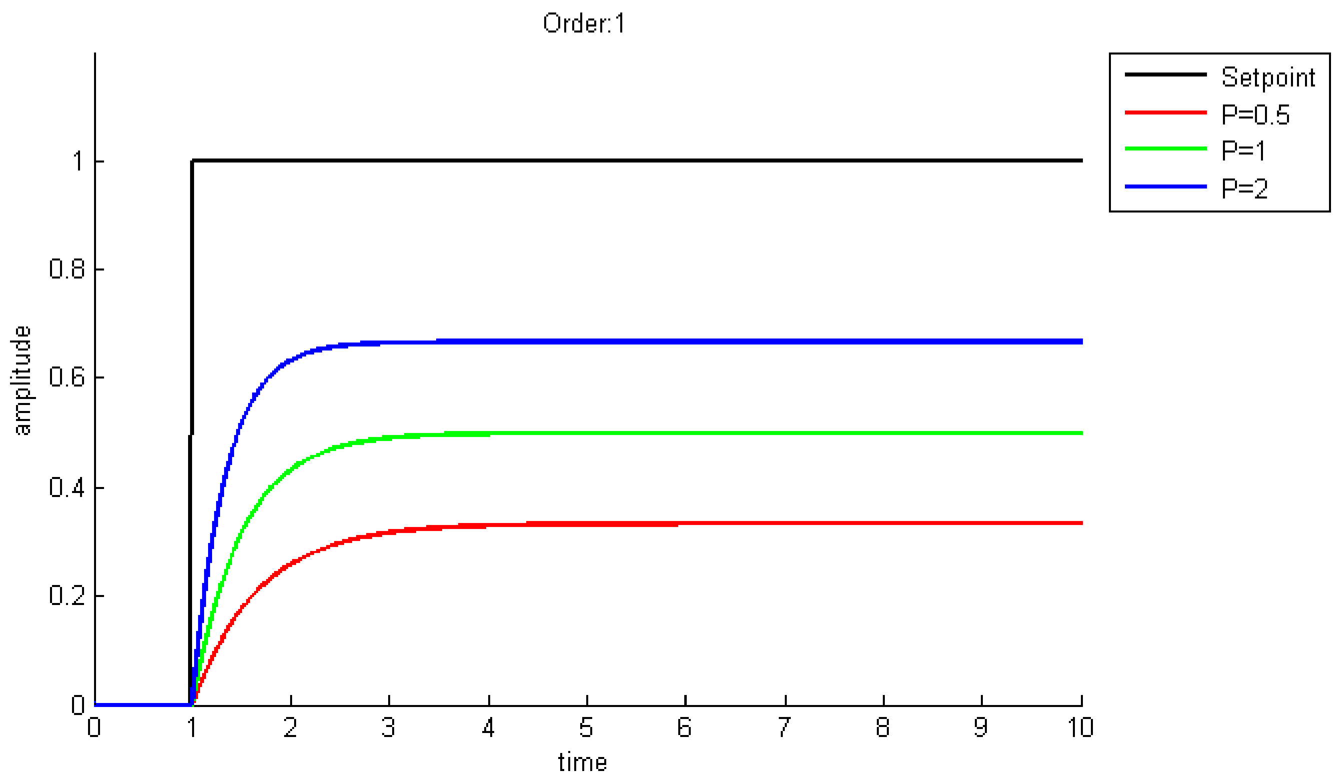
Proportional error control

- Simple first order system



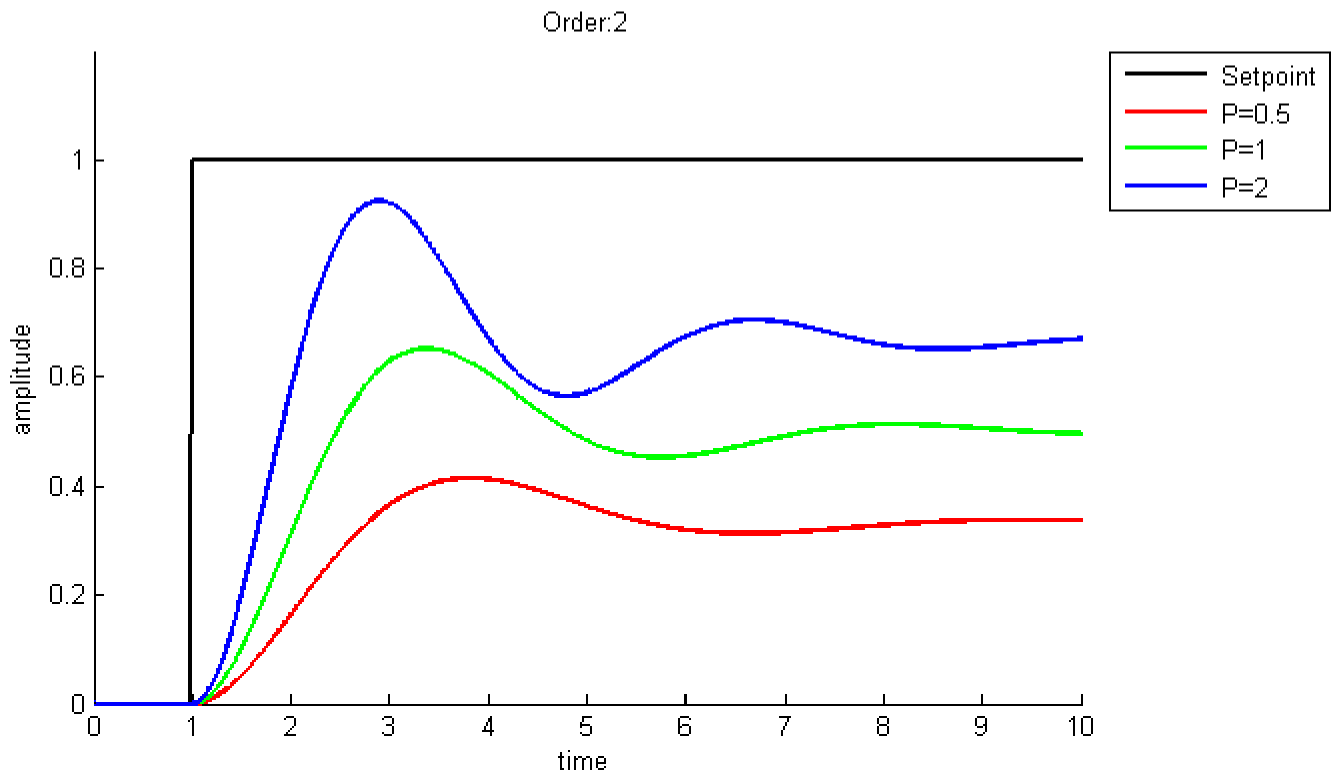
Proportional error control

- Steady constant disturbance (droop problem)
  - (think of "wind")



Proportional error control

- Second order system (with damping)
- Droop and oscillations

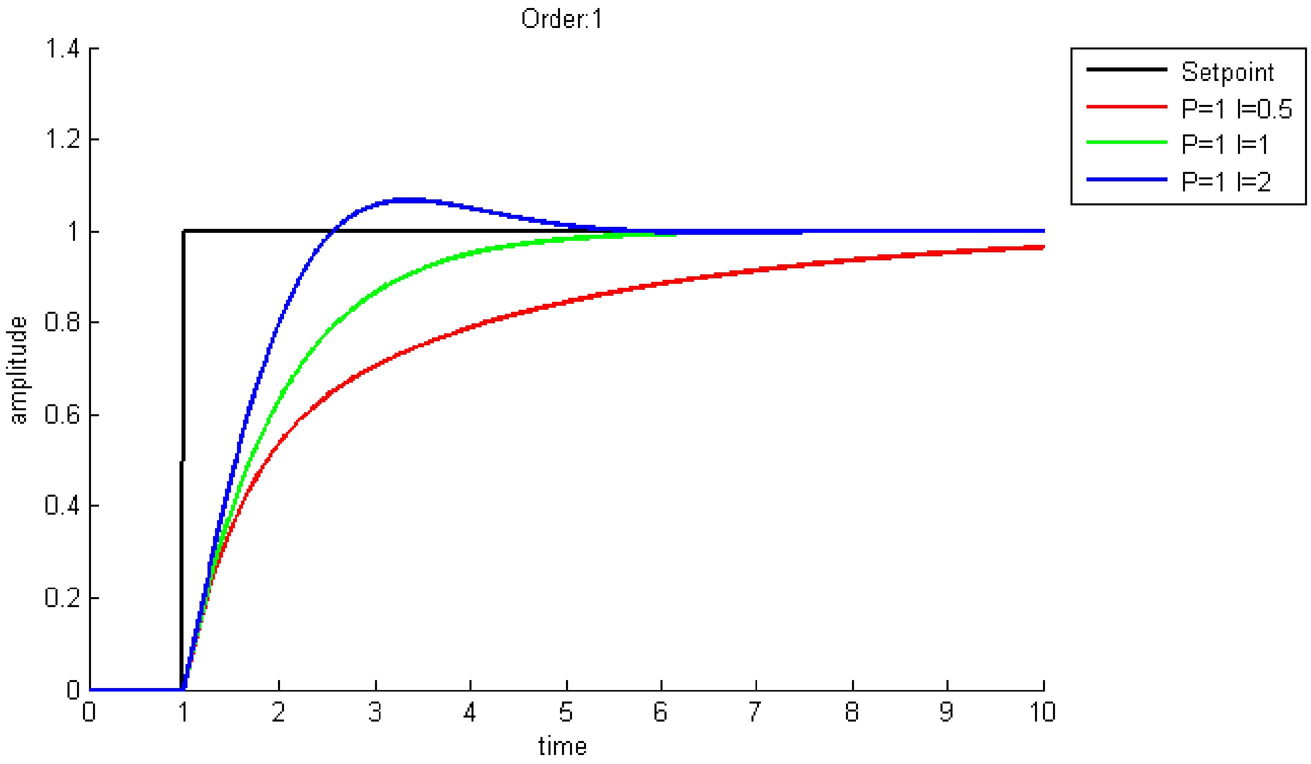


Proportional integral control

- Account for undershoot of P controller?
- Accumulate history of error
  - Increase response at history increases
- Then, error history $\int_0^t e(\tau)d\tau$
- Response output: $K_I \int_0^t e(\tau)d\tau$
- K_I is the integral gain
  - Tends to overshoot and cause oscillatory behaviour!

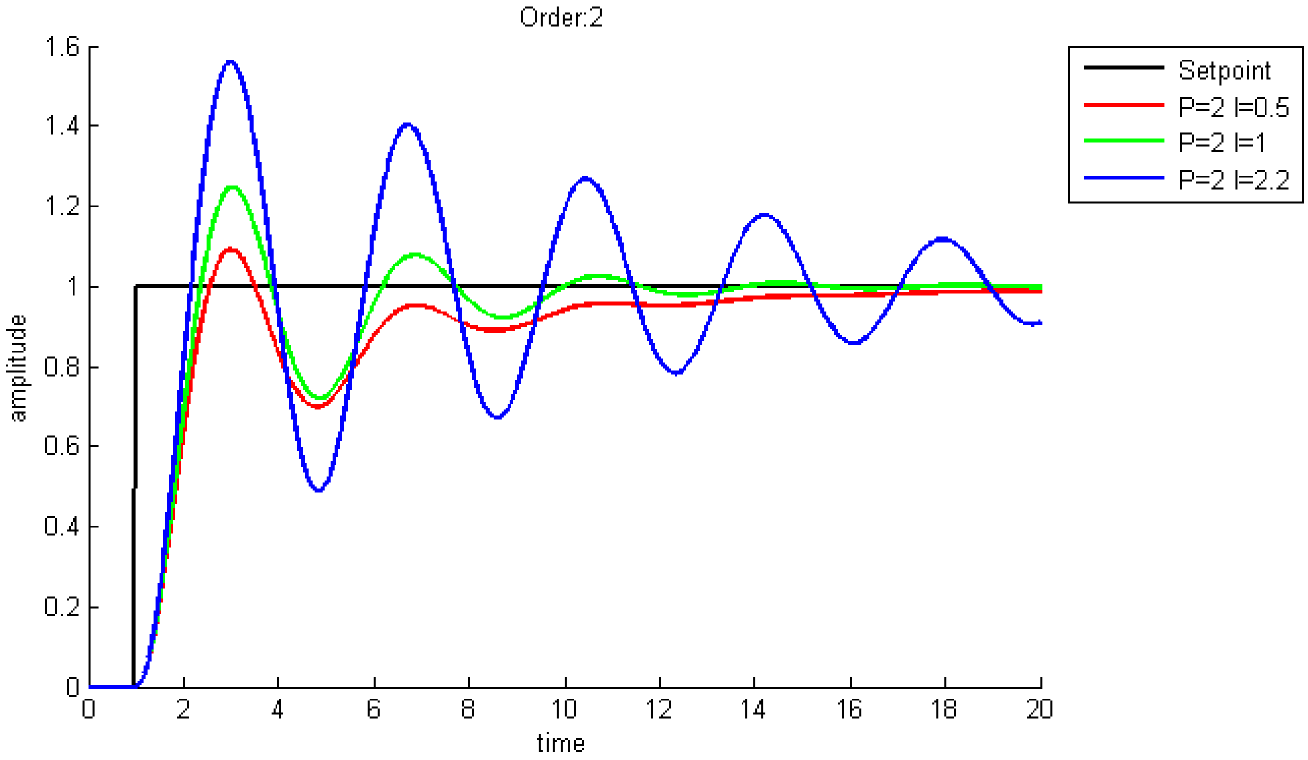
Proportional integral control

- Accounts for steady state error



Proportional integral control

- Introduce or amplify oscillations

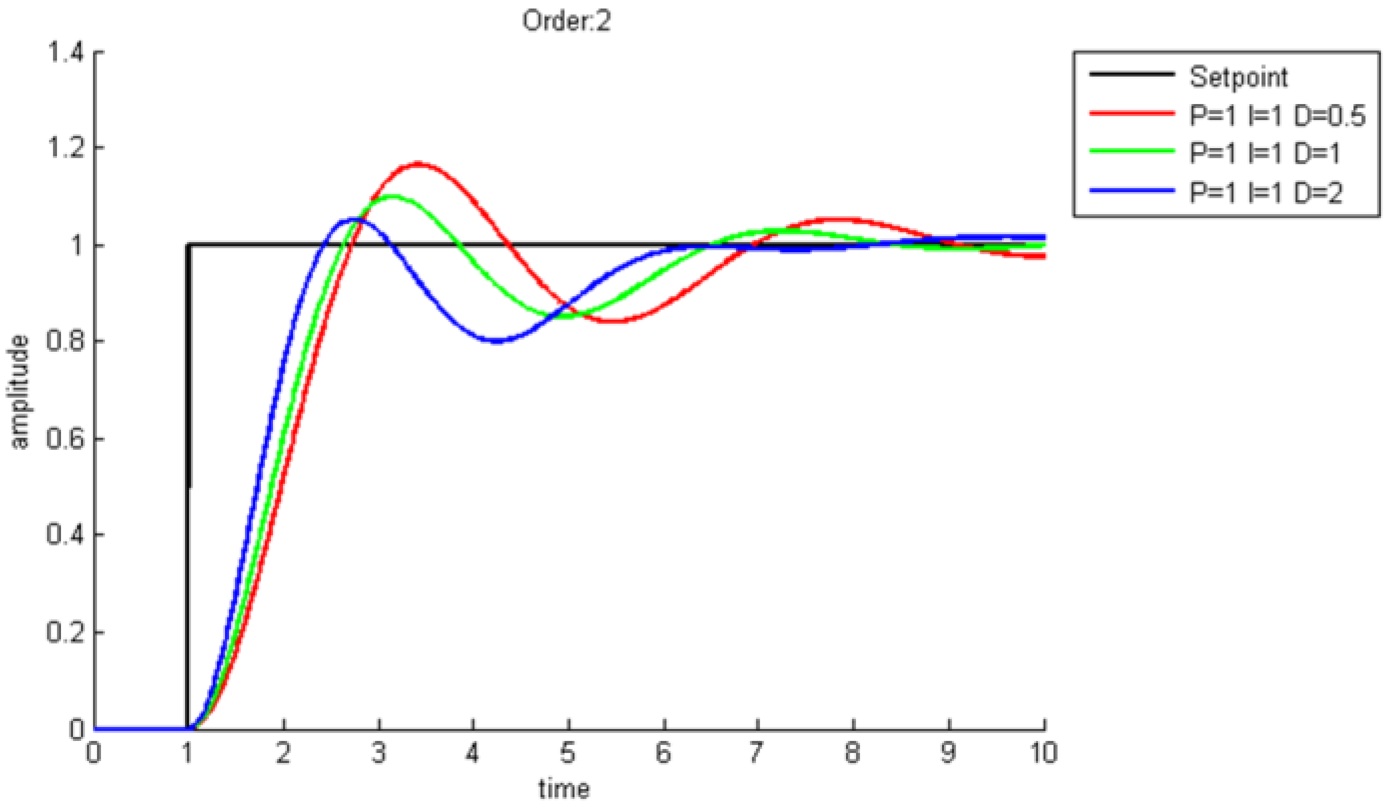


Proportional derivative control

- Correct for (damp) oscillations?
  - Often caused by inertia, delays, and rapid overcorrecting
- Want to resist change: the faster the system changes, the harder it resists (artificial friction)
  - "Predictive" corrections
- Rate of change of error $\frac{d}{dt}e(t)$
- Response output: $K_D\frac{d}{dt}e(t)$
- K_D is the derivative gain
  - Improves settling time and stability
  - Can amplify noise, so not always used
  - Can slow control

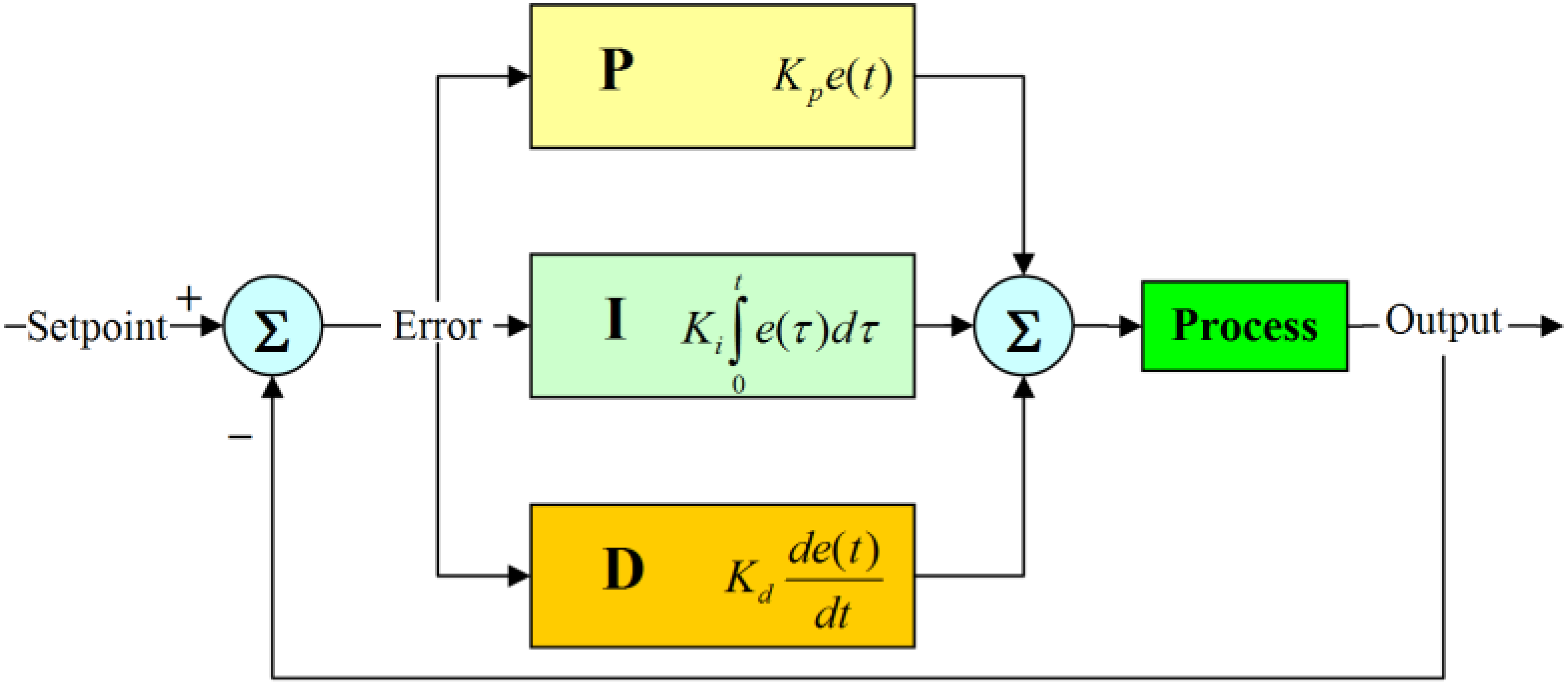
PID control

- Oscillation damping

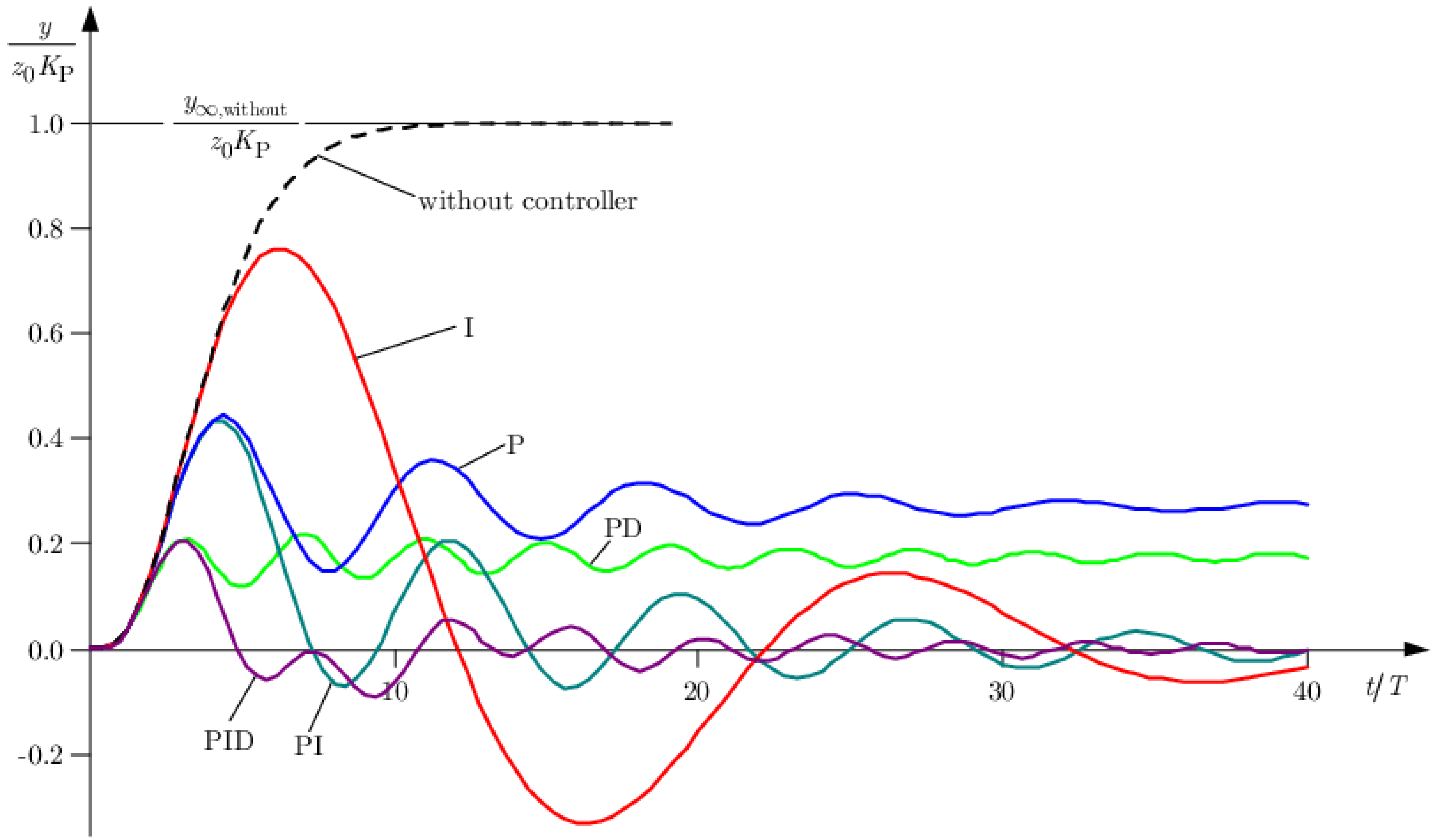


PID control

- Combine: PID control
- Control law: $U = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t)$



PID control



$\frac{y}{z_0 K_P}$
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
$\frac{y_{\infty,\text{without}}}{z_0 K_P}$
without controller
I
P
PD
PID
PI
10
20
30
40
t/T

PID control

- No knowledge of the plant dynamics
  – Only: current and desired behaviours
- Factors to tune for:
  – Responsiveness
  – Setpoint offset
  – Oscillation
- Tuning is often a fine art
- Different parameters may be needed in different regions of state space (tracking)
- D-term not often used in practice (noise amplifying)
- Not optimal

Tuning the parameters

- Manually:
  1. Set K_I and K_D to 0
  2. Increase K_P until oscillations
  3. Halve K_P
  4. Increase K_I until any offset corrected sufficiently fast (too high will be unstable)
  5. Increase K_D until quick recovery from perturbation (too high will overshoot)

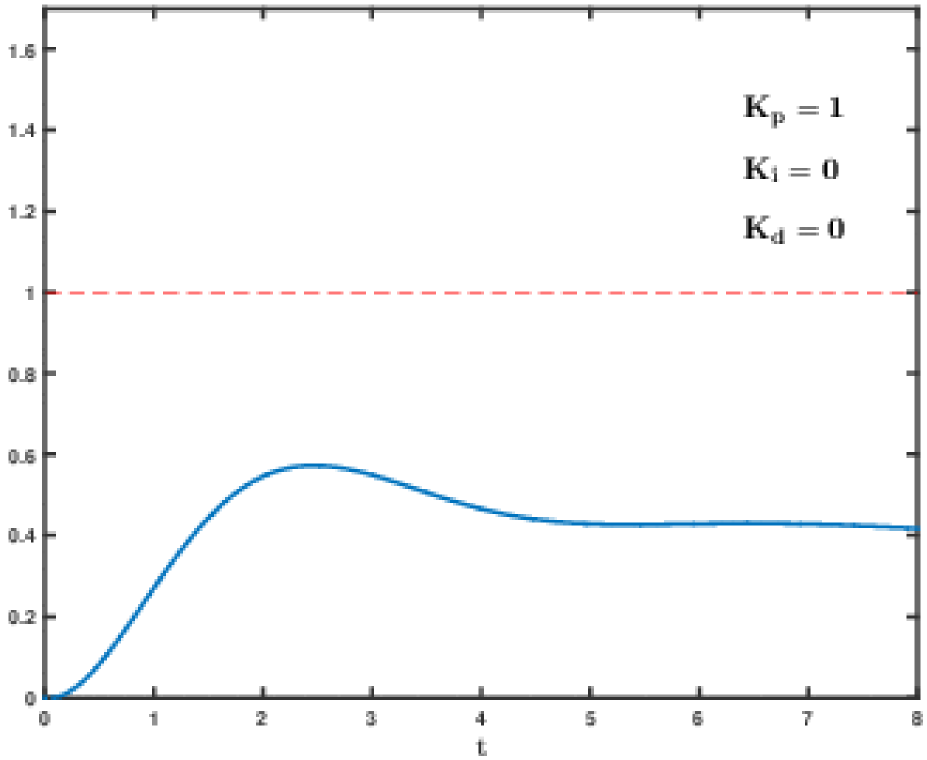
Tuning the parameters

- Ziegler-Nichols method:
  1. Set K_I and K_D to 0
  2. Increase K_P until oscillations (at ultimate gain K_U) with oscillation period P_U
  3. Then set:

Ziegler–Nichols method

Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$1.2K_p/P_u$	-
PID	$0.60K_u$	$2K_p/P_u$	$K_pP_u/8$

Illustrated parameter tuning



https://en.wikipedia.org/wiki/PID_controller#/media/File:PID_Compensation_Animated.gif