Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

System Modeling & Control

Presented by

Prof. Amit Kumar Sahoo
CUTM, BBSR

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Module VII
Controllers

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Lesson - 35

Controllers

Introduction

Suppose you have a system that needs to be controlled

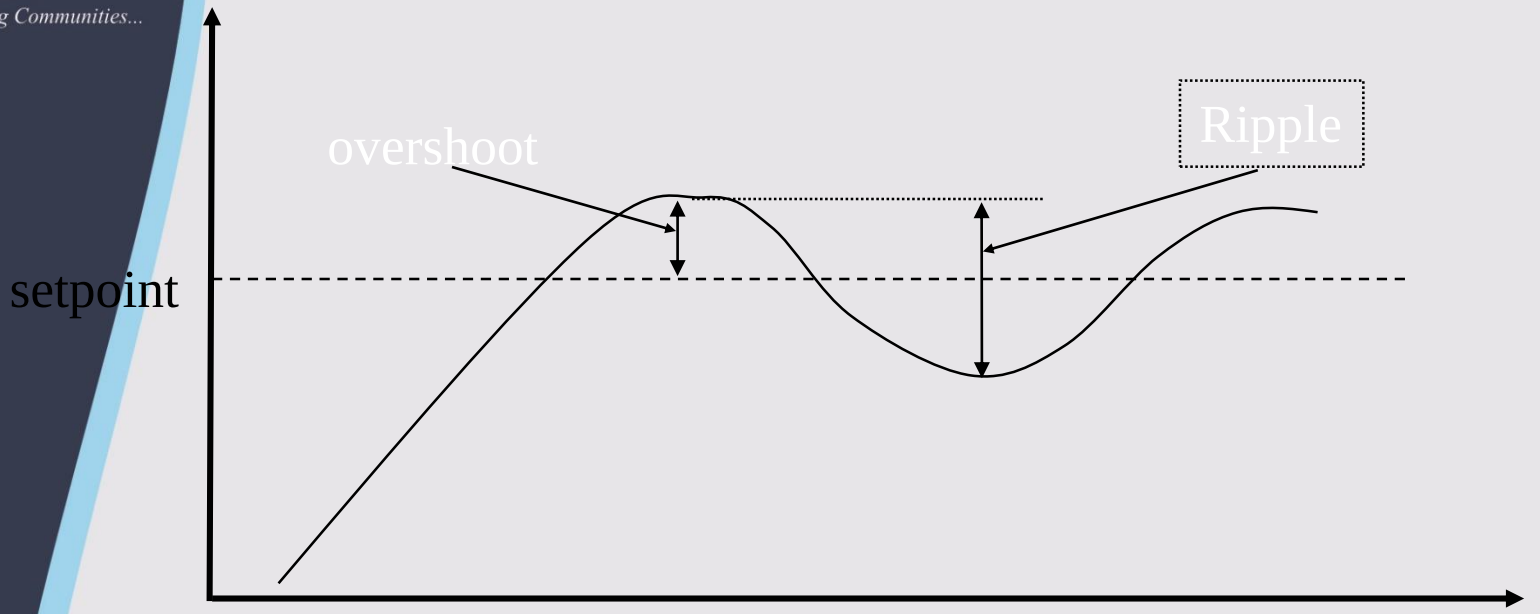
Your software gives commands, the system responds to it

- Turn x degrees to the right
- Move forward 15 wheel rotations

Can you always trust your commands will be executed accurately?

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Problem example



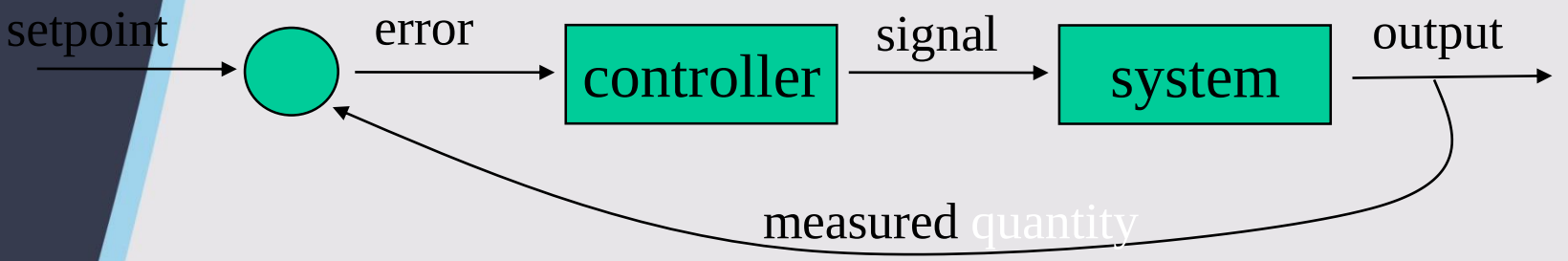
Increase the quantity until you get to the setpoint

Temperature, angle, speed, etc

If too much, reduce the quantity, until the setpoint

Centurion University
Shaping Lives...
Empowering Communities...

Closed loop controller



- closed loop because it has feedback
- output is measured at a certain frequency
- signal is generated at a certain frequency
- which frequency is not smaller?

On-off control

For some systems, on-off signaling is sufficient

For example, a thermostat, when the heater is either on or off, and early cruise-control systems

- Could do airflow or speed control also
- More modern systems do it

Depending on the frequency of control, overhead of on-off, etc, this could cause overshoots and undershoots (ripples)

- Oscillation is a common behavior in control systems
- Need to avoid it at all costs… well, almost all costs

Proportional control

- Good alternative to on-off control: more “control” ☺
- Signal becomes proportional to the error
  - *P (setpoint – output)*
  - Example, car speed for cruise control
- Need to find out value of constant P
  - Tuning the controller is a **hard** job
  - If P is too high, what happens?
  - If P is too low, what happens?
- Typically a prop cntrl decreases response time (quickly gets to the setpoint) but increases overshoot

Adding derivative control

To avoid (or reduce) overshoot/ripple, take into account how fast are you approaching the setpoint

- If very fast, overshoot may be forthcoming: reduce the signal recommended by the proportional controller
- If very slow, may never get to setpoint: increase the signal
- In general: *D (current measure – previous measure)*

PD controllers are slower than P, but less oscillation, **smaller** overshoot/ripple

Integral control

- There may still be error in the PD controller
  - For example, the output is close to setpoint
    - P is very small and so is the error, discretization of signal will provide no change in the P controller
    - D controller will not change signal, unless there is change in output
- Take the sum of the errors over time, even if they're small, they'll eventually add up
  - I * sum_over_time (setpoint — output)
- Again the main problem is the value of I
- Can we let sum grow to infinity?

Summary

Different types of controllers

PID hardest task is tuning

Controller	Response time	Overshoot	Error
On-off	Smallest	Highest	Large
Proportional	Small	Large	Small
Integral	Decreases	Increases	Zero
Derivative	Increases	Decreases	Small change

Tips for Designing a PID Controller

1. Obtain an open-loop response and determine what needs to be improved
2. Add a proportional control to improve the rise time
3. Add a derivative control to improve the overshoot
4. Add an integral control to eliminate the steady-state error
5. Adjust each of Kp, Ki, and Kd until you obtain a desired overall response.

Lastly, please keep in mind that you do not need to implement all three controllers (proportional, derivative, and integral) into a single system, if not necessary. For example, if a PI controller gives a good enough response (like the above example), then you don't need to implement derivative controller to the system.

Keep the controller as simple as possible.