

## Lecture 10

[These slides]

- Program Specifications
- Implementation of Controller Processes
- Program termination
- Debugging real-time programs
- Embedded Systems & Numerics

1

## Lectures in Study Period 2

- 29/10 - Integrated Control and Scheduling – Anton Cervin
- 5/11 - Setpoint Handling
- 12/11 - Industrial Control Systems - IEC 61131, CORBA, OPC
- 19/11 - Real-Time Communication
- 26/11 - Formal methods, Petri Nets, Hybrid Systems
- 3/12 - Course Summary
- 10/12 - Project Demonstrations

Lecture Time: Wednesdays 15.15-17.00 in E:B

Two problem-solving exercises in week 2 and 3.

2

## Program Specification

- Module/Package Structure
- Real-Time Structure
- Control Structure
- User Interface
- Implementation Plan

3

## Module/Package Structure

- In Java, code is decomposed into packages
  - For small applications, the anonymous (default) package
  - Objects as code modules
- Interface Specification
  - Informally, i.e., specifying the methods that the objects should implement and their interfaces (Class skeletons)
  - Abstract classes
  - Java interfaces
  - Graphically, e.g., the notation used in the exercises or UML notation

4

## Abstract classes for interface specifications

```
public abstract class RegulSpecification {

    public static final int OFF = 0;
    public static final int BEAM = 1;
    public static final int BALL = 2;

    // Start threads with specified priority
    public abstract void start(int prio);

    // Terminates threads in a graceful manner
    public abstract void terminate();

    ...
}
```

Require that the Regul class extends RegulSpecification.

5

## Java Interfaces for interface specifications

```
public interface RegulSpecification {
    public static final int POSITION = 0;
    public static final int ANGLE = 1;

    public void start(int prio);

    public void terminate();

    ...
}
```

Require that the Regul class implements RegulSpecification

6

In real-time applications it is in general not enough to only specify the interface.

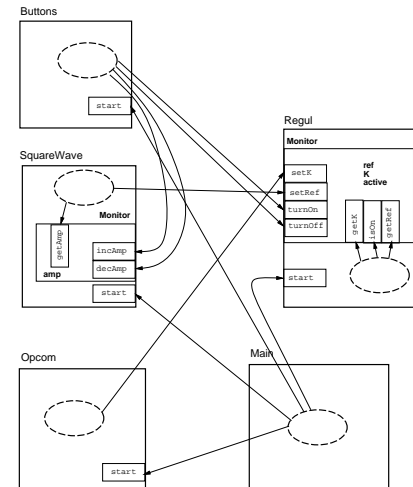
Additional information needed:

- are there any threads involved and if so, what are their priorities?
- are the methods blocking or non-blocking?
- if the methods return references or take references as arguments, then how should the associated memory be handled
  - code examples later

Parts of this is covered by the notation used in the exercises.

7

## Buttons Exercise



8

## Real-Time Structure

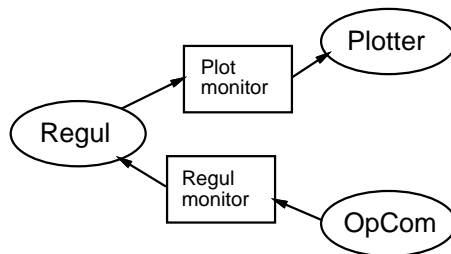
- Parallel activities:
    - control loops
    - operator communication (input)
    - data presentation
- 
- communication
  - dialogue handling
  - monitoring & supervision
  - ...

9

- Thread descriptions
  - Regul
    - \* periodic
    - \* high priority
  - Refgen
    - \* often periodic
    - \* often lower priority and frequency than Regul
  - Plotter
    - \* event based
    - \* waits on plot data
  - OpCom
    - \* waits on keyboard, mouse input
    - \* In Java/Swing, a single event-dispatching thread
    - \* calls a method (e.g., `ActionPerformed`) in the object listeners of the object issuing the event

10

- Communication & Synchronization
  - how do the threads communicate - monitors, mail-boxes, ..
  - e.g.



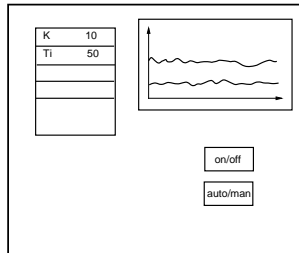
11

## Control Structures

- Control principles (e.g., PID, state feedback, ..., continuous time design, discrete time design)
- Operating modes
  - auto – manual
  - startup – shutdown
  - position – angle
  - position – velocity
- Safety Shutdown
- Alarms
- Command Structure
- Logic
  - interlocks
  - sequences

12

## User Interface



- Screen layout
- Command, subcommands
- Menus
- Data presentation
- Ergonomics

13

## Implementation Plan

- Order
  - plan
  - deadlines
- Who does what?
- Testability!

14

## Success Criteria

- Performance
  - control
  - real-time
- Structure
  - modules
  - abstract data types/objects
  - procedures/methods
- General (extendible)
  - no hard coding
  - extensions imply small modifications
  - difficult
- Readable code
  - comments
  - name choices
  - programming style

15

## Implementation of Controller Processes

Example: PID

$$U(s) = K(E(s) + \frac{1}{sT_i}E(s) + \frac{sT_d}{1 + \frac{sT_d}{N}}(-Y(s))) \quad (1)$$

Discrete Form:

$$\begin{aligned} 1: \quad d(k) &= \frac{T_d}{T_d + Nh}d(k-1) - \frac{KT_dN}{T_d + Nh}(y(k) - y(k-1)) \\ v(k) &= K(y_r(k) - y(k)) + i(k) + d(k) \\ u(k) &= \text{sat}(v(k), \text{UMIN}, \text{UMAX}) \end{aligned}$$

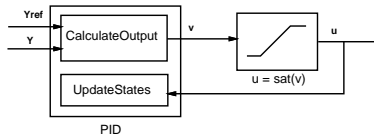
$$\begin{aligned} 2: \quad i(k+1) &= i(k) + \frac{Kh}{T_i}e(k) + \frac{h}{T_r}(u(k) - v(k)) \\ y(k) &= y(k-1) \end{aligned}$$

1 - calculateOutput

16

## Why 2 procedures?

- Minimize computational delay
  - Computational delay  $\approx$  time delay
  - Reduces phase margin, may lead to instability
- Code structuring reasons



- u not available until the saturation block has been calculated

17

## Minimize Computations

$$dp_1 := \frac{T_d}{T_d + Nh}$$

$$dp_2 := \frac{KT_dN}{T_d + Nh}$$

..

..

$$d(k) := dp_1 d(k-1) - dp_2 (y(k) - y(k-1))$$

- precalculated parameters
- $dp_1$  and  $dp_2$  only recalculated when any of the parameters are changed
- similar for the I-part

18

## Data Structures

- Parameters ( $K, T_i, \dots$ )
- States ( $i, d, yold$ )
  - three states (one more than necessary)
  - increases readability
- Modes (Auto, Manual, ...)
- AuxVars ( $v, yref, \dots$ )
  - yref
  - bumpless parameter changes when  $\beta \neq 1$
  - Needed to determine  $P_{old}$  and  $P_{new}$

$$I_{new} = I_{old} + K_{old}(\beta_{old} y_{sp} - y) - K_{new}(\beta_{new} y_{sp} - y)$$

We can assume that  $y_{sp} = y$ , i.e.,

$$I_{new} = I_{old} + y_{sp}(K_{old}(\beta_{old} - 1) - K_{new}(\beta_{new} - 1))$$

19

## Methods

### Procedures

- setParameters(PIDParams)
- setMode
- double calculateOutput(yref, y) // returns v
- updateState(u)
- Constructor

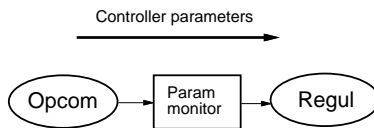
20

## Parameter passing: Opcom to Regul

Goals:

- Regul should be blocked as little as possible
- As little jitter as possible for Regul
- Regul should use the same parameters in calculateOutput and in updateState
- Regul should use consistent parameter values, i.e., it should be possible to change multiple parameters in one update
- Separation between the controller algorithm, e.g., the basic algorithm, i.e., PID, and its usage in a real-time thread, i.e. Regul.

First: Assume that a monitor is used for communication.



21

## Example 1

- Execution of control algorithm inside monitor
- Synchronized methods: setParameters, calculateOutput, updateState

```
while (doIt) {
    y = ychan.get();
    v = calculateOutput(y);
    u = saturate(v,umin,umax);
    uchan.set(u);
    updateState(u);
    sleep(..);
}
```

Problem:

- a change of parameters can be performed between the two parts of the controller algorithm
- Regul may be blocked if the execution of setParameters takes long time
- No separation between Regul and PID

22

## Example 1: Regul and PID separation

- The PID class implements the monitor using synchronized methods (calculateOutput, updateState, setParameters)
- Contained within Regul

```
while (doIt) {
    y = ychan.get();
    v = pid.calculateOutput(y, yref);
    u = saturate(v,umin,umax);
    uchan.set(u);
    pid.updateState(u);
    sleep(..);
}
```

Problem:

- a change of parameters can be performed between the two parts of the controller algorithm
- Regul may be blocked if the execution of setParameters takes long time

23

## Example 1: Indivisible control calculations

- Use a synchronized block that synchronizes upon pid

```
while (doIt) {
    y = ychan.get();
    synchronized (pid) {
        v = pid.calculateOutput(y, yref);
        u = saturate(v,umin,umax);
        uchan.set(u);
        pid.updateState(u);
    }
    sleep(..);
}
```

Problem:

- Regul may be blocked if the execution of setParameters takes long time

24

## Example 1: Minimize setParameters

Performs as much as possible of the the time-consuming parameter updating outside the monitor

Synchronized methods:

- `getParameters`: Returns a reference to the currently used parameters.
- `setParameters(Params newParams)`: Sets new parameters.

```
public synchronized Params getParameters() {
    return p;
}

public synchronized void setParameters(Params newParams) {
    p = newParams;
}
```

25

REGUL

```
while (doIt) {
    y = ychan.get();
    synchronized (pid) {
        v = pid.calculateOutput(y,yr);
        u = saturate(v,umin,umax);
        uchan.set(u);
        pid.updateState(u);
    }
    sleep(...);
}
```

OPCOM

(within the PID context)

```
oldp = getParameters();
p = new Params;
use p and oldp to update
parameters outside
monitor;
setParameters(p);
```

26

## Example 2: PID calculations outside monitor

- Execution of control algorithm outside monitor
- Synchronized methods:

```
setParameters
getParameterK
getParameterTi
...
```

- `calculateOutput` and `updateState` non-synchronized methods belonging to Regul

```
while (doIt) {
    p.K = getParameterK;
    p.Ti = getParameterTi;
    y = ychan.get();
    v = calculateOutput(P,y);
    u = saturate(v,umin,umax);
    uchan.set(u);
    updateState(P,u);
    sleep(...);
}
```

27

Problem:

- risk for inconsistent controller parameters
- Regul may be blocked by the execution of `setParameters`
- sampling jitter for Regul

28

## Example 2: A single getParameters

- Execution of control algorithm outside monitor
- Synchronized methods:

```
setParameters  
getParameters  
...
```

- calculateOutput and updateState local methods in Regul

```
LOOP  
  y = ychan.get();  
  v = calculateOutput(P,y);  
  u = saturate(v,umin,umax);  
  uchan.set(u);  
  updateState(P,u);  
  P = getParameters;  
  sleep(...);  
END;
```

29

## Comments:

- all parameters fetched in a single access – no inconsistency
- getParameters after the execution of the controller – less risk for jitter
- Regul can still be blocked by the execution of setParameters, unless the technique in Example 1 is used.
- separation between Regul and PID more difficult to achieve

30

## Program Termination

How make sure that the program terminates in an orderly fashion?

The rough way:

- System.exit(0)
- Called from a listener in Opcom
- As far as we know the only way to terminate the Swing event-dispatching thread, i.e., it should be called
- Need ways of terminating the other threads in an orderly fashion, i.e., make sure that uchan.set(0.0) is called

31

## Code Example

```
public class Main {  
  static void main(String[] argv) {  
  
    // Start the application  
  
    opCom.waitForExit(); // Take (wait) on a semaphore  
                        // that is signalled from a listener  
  
    regul.stop();  
    refGen.stop();  
    opCom.stop();  
    System.exit(0);  
  }  
}
```

32

### Code Example, continued

- Regul and Refgen:

- run() method

```
while (doIt) {  
    // perform activity  
    sleep(...);  
}
```

- Plotter thread within OpCom

- run() method

```
while (doIt) {  
    plotData = buffer.get();  
    // Plots the data  
}
```

- stop() method

- sets doIt to false (in Regul also uchan.set(0.0))

33

### Code Example, continued

Does not work as intended.

The threads will only check the doIt flag when they have finished sleeping or received new plot-data

The threads will most likely terminate due to the System.exit(0) call and not because of termination of their run methods.

The doIt flag is used by multiple threads without synchronization.

- OK since the flag is a boolean and the read and write operations are atomic
- However, the flag should be declared as volatile to make sure that a write operation is seen by the other threads without delay

```
private volatile boolean doIt = true;
```

34

### Code Example, continued

Alternative approach:

- let the main thread wait for acknowledgment semaphores before it calls System.exit(0)

```
opCom.waitForExit();  
regul.stop();  
refGen.stop();  
opCom.stop();  
opCom.stoppedAcknowledgement();  
regul.stoppedAcknowledgement();  
refgen.stoppedAcknowledgement();  
System.exit(0);  
}  
}
```

35

- Modify the run methods

```
while (doIt) {  
    // perform activity  
    sleep(...);  
}  
signalAcknowledgement();
```

Still does not solve the problem with plotter thread, which will not terminate if it does not receive any plot data from the regul thread.

36

### Code example, continued

Modify the stop method in Opcom

```
plotThread.interrupt();
```

Causes the throwing of an InterruptedException

Catch this exception in the run method of the plotter thread

```
while (doIt) {  
    try {  
        plotData = buffer.get();  
    } catch (InterruptedException e) {  
        doIt = false;  
    }  
}
```

Also necessary for the get() method of the Buffer class to throw this exception.

37

### Debugging

- Debugging real-time programs is in general a difficult issue.
  - Event-driven and non-deterministic execution.
- Debugging realtime control applications even harder
  - Difficult to distinguish algorithm errors (e.g., badly initialized parameters, logical algorithm faults, wrong parameter values, ...) from real-time errors.

38

### Run-Time Problems

- Check your program to make sure you have no busy-wait loops that may cause starvation. Use the green-thread model to check this.
- Check that your thread priorities are correct.
- Try to isolate your fault by only starting one thread at a time.
- Use comments to gradually comment out more and more of your code. ("divide and concur")
- Use debugging printouts, but make sure that you do not try to write debugging outputs at a too high frequency. In that case the output itself will cause real-time problems.
- When using short sampling intervals, do not update the plotters every sample.
- If you suspect signal problems, borrow a voltage meter and measure.
- Apply the performance tricks from Lecture 9.

39

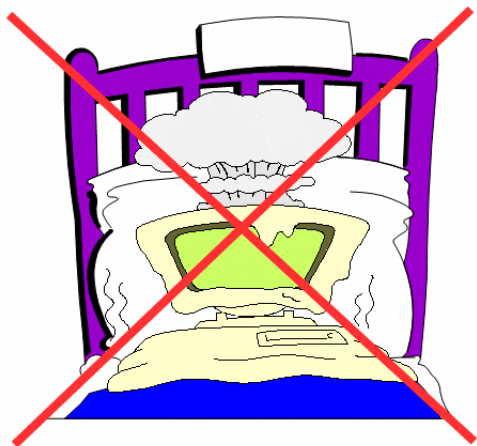
### Embedded Systems

Many of the real-time systems that have hard real-time constraints are embedded systems.

Many control systems are also embedded.

40

## What is an embedded system?



41

## Embedded Systems

- Embedded systems (ES) = information processing systems embedded into a larger product
- Main reason for buying is **not** information processing
- Increased emphasis due to the pervasive/ubiquitous computing trend ("embedded everywhere")
- Mass-marketed products subject to hard economic constraints



42

## Importance of Embedded Software and Embedded Processors

"... the New York Times has estimated that the average American comes into contact with about 60 micro-processors every day..."  
[Camposano, 1996]

Latest top-level BMWs contain over 100 micro-processors  
[Personal communication]



43

## Views on embedded software (1)

.. the next ten years may be as revolutionary for embedded system design as the last 10 years have been for IC design [Wayne Wolf, 1996]



... it is now common knowledge that more than 70% of the development cost for complex systems such as automotive electronics and communication systems are due to software development  
[A. Sangiovanni-Vincentelli, 1999]

44

## Views on embedded software (2)

... dramatic growth of the number of embedded applications and the size and the complexity of the software used in these applications.

For many products in the area of consumer electronics the amount of code is **doubling every two years**.

[Fritz Vaandrager in: Rozenberg, Vaandrager (eds.): Lectures on Embedded Systems, LNCS, Vol. 1494, 1998]

45

## Views on embedded software (3)

It is estimated that each year embedded software is written five times as much as 'regular' software

The vast majority of CPU-chips produced world-wide today are used in the embedded market ... ; only a small portion of CPU's is applied in PC's

... the number of software-constructors of Embedded Systems will rise from 2 million in 1994 to 10 million in 2010;

... the number of constructors employed by software-producers 'merely' rises from 0.6 million to 1.1 million.

[Department of Trade and Industry/ IDC Benelux BV: Embedded software research in the Netherlands. Analysis and results, 1997]

46

## Resource Constraints

- Product-level constraints (\$\$, size, connectability, ...) generate platform-level resource constraints:
  - Computing speed
  - Word length, amount of memory and chip size
    - 8, 16 bit processors
    - Small flash memory → small applications
    - Small RAM memory → small amount of variables
    - None or limited support for RTOS/RT-kernels
  - Communication bandwidth
  - Power consumption
  - ...
- True in spite of the rapid development of computing hardware

47

## Computer Arithmetics

Control analysis and design assumes floating point arithmetics (i.e. high range and resolution)

Hardware-supported on modern high-end processors (e.g., floating point ALUs (Arithmetic-Logic Units))

Representation:

$$\pm f \times 2^{\pm e}$$

- $f$ : mantissa, significand, fraction
- 2: radix or base
- $e$ : exponent

48

## IEEE 754 Standard

Used by almost all floating-point processors (except certain DSPs)

Single precision (Java/C float):

- 32-bit word divided into 1 sign bit, 8-bit exponent, and 23-bit mantissa
- Range:  $2^{-126} - 2^{128}$

Double precision format (Java/C double):

- 64-bit word divided into 1 sign bit, 11-bit exponent, and 52-bit mantissa.
- Range:  $2^{-1022} - 2^{1024}$

Supports infinity and NaN

49

## Floating-point emulation

Emulate floating-point arithmetics in software

Approaches:

- compiler supported
- manually
  - e.g., floating point variables represented as C structs
  - floating point operations in the form of a library

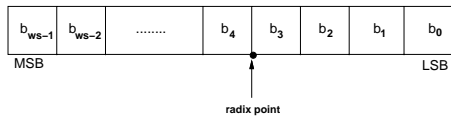
Problems:

- Code size becomes too large
- Slows down execution speed
- Non-trivial

50

## Fixed Point Arithmetics

Use the binary word directly for representing numbers



- MSB - Most significant bit
- LSB - Least significant bit
- ws - word-size

Unsigned versus signed

51

## Fixed Point Arithmetics

Integer arithmetics:

- radix point to right of LSB
- 16 bits signed integer gives range  $-32768 \leq \hat{x} \leq 32767$   
( $(-2^{15}) - (2^{15} - 1)$ )

Fractional arithmetics:

- radix point to right of MSB (signed)
- 0.10011001

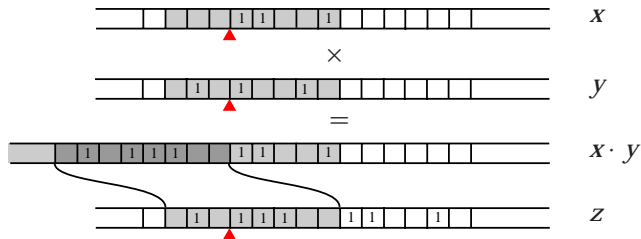
Generalized fixed point arithmetics:

- application-defined radix point
- 1101.0110
- Scaling:  $x = \hat{x}/2^4$  - shifting the radix point

52

## Fixed Point Calculations

Fixed point multiplication involves quantization



Fixed-point addition is error-free

Quantization (truncation or rounding)

- modeled as “noise”

Overflow (wrap-around or saturation)

53

## Example: Scalar products

Many controllers and filters involve calculations of scalar products, e.g.,

$$u = -Lx = -[I_1 I_2 I_3][x_1 x_2 x_3]^T = -I_1 x_1 - I_2 x_2 - I_3 x_3$$

Consider the vectors

$$\begin{aligned} a &= (100 \ 1 \ 100) \\ b &= (100 \ 1 \ -100) \end{aligned}$$

The true scalar product is 1

When computed in fixed point representation using a precision corresponding to three decimal places, the result will be 0 (100 × 100 + 1 × 1 is rounded to 10000)

The result depends on the order or the operations.

To avoid this it is common to use higher resolution in the accumulator and round to a smaller resolution afterwards.

54

## Fixed-Point Arithmetics Problems

- Quantization
  - Fixed-point values are rounded or truncated.
    - Coefficient Quantization: Poles and zeros end up somewhere else
    - Signal Quantization:
      - \* Noise is added in each operation
      - \* Quantization may cause signal bias
      - \* Quantization may cause limit cycles. Either in the output only (LSB) or in the entire system through feedback.
- Overflow
  - Adding/Multiplying two sufficiently large numbers can produce a result that does not fit into the representation.
    - Scaling important both of variables and of coefficients.
    - Overflow characteristics. Saturation or wrap-around? Hardware supported overflow detection or not.

55

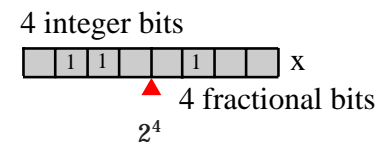
## Example: Coefficient Quantization

An example controller

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.5776}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184}$$

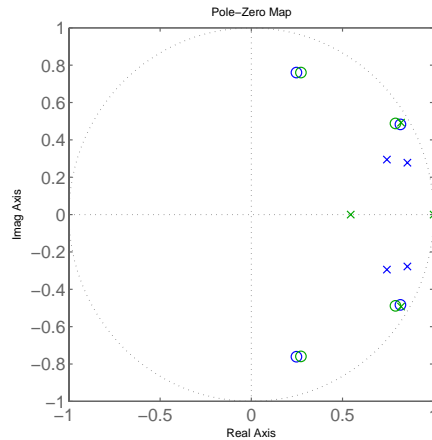
8-bit fixed point coefficients with  $x = \hat{x}/2^4$ , so

$$x \in [-8.0 \dots 7.9375]$$



56

### Example



### Issues: Realization of Digital Controllers

A digital controller

$$u(k) = H(q^{-1})y(k) = \frac{b_0 + b_1q^{-1} + \dots + b_mq^{-m}}{1 + a_1q^{-1} + a_2q^{-2} + \dots + a_nq^{-n}}y(k)$$

can be realized in a number of different ways with equivalent input-output behavior (different choice of state variables)

Issues:

- number of storage elements (memory)
- number of non-zero non-one coefficients
- coefficient range
- sensitivity towards coefficient quantization
- sensitivity towards state quantization
  - order of computations matters

### Direct and Companion Forms

$$u(k) = \sum_{i=0}^m b_i u(k-i) - \sum_{i=1}^n a_i y(k-i)$$

Not minimal ( $n + m$  states)

Companion forms (e.g., observable canonical form or controllable canonical form):

$$x(k+1) = \begin{pmatrix} -a_1 & 1 & 0 & \dots & 0 \\ -a_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_{n-1} & 0 & 0 & \dots & 1 \\ -a_n & 0 & 0 & \dots & 0 \end{pmatrix} x(k) + \begin{pmatrix} b_1 \\ \vdots \\ b_{m-1} \\ b_m \\ 0 \end{pmatrix} y(k)$$

$$u(k) = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix} x(k)$$

Minimal

Coefficients in the characteristic polynomial are the coefficients in the realization. Sensitive to computational errors if the systems are of high order and if the poles or zeros are close to each other.

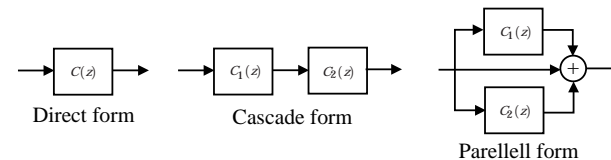
### Example

A linear system can be rewritten in many ways:

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.5776}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184}$$

$$= \left( \frac{z^2 - 1.635z + 0.9025}{z^2 - 1.712z + 0.81} \right) \left( \frac{z^2 - 0.4944z + 0.64}{z^2 - 1.488z + 0.64} \right)$$

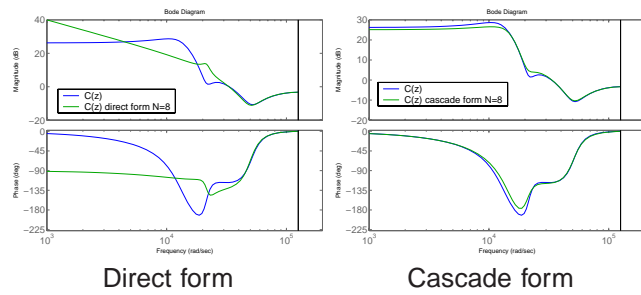
$$= 1 + \frac{-5.396z + 6.302}{z^2 - 1.712z + 0.81} + \frac{6.466z - 4.907}{z^2 - 1.488z + 0.64}$$



## Well-conditioned realizations

Parallel (diagonal/Jordan) and cascade (series) forms have normally the best numerical properties.

If poles (zeroes) are far apart, direct form is usable.



Direct form

Cascade form