Programming Intelligent Physical Systems

Lecture 1

Samarjit Chakraborty

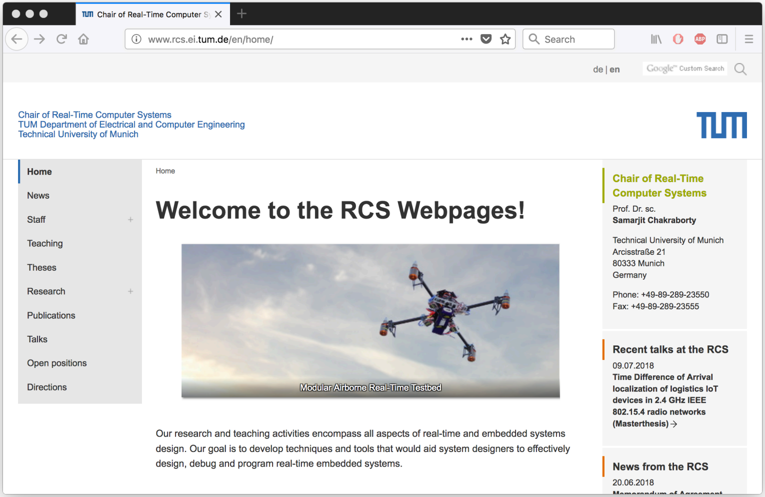
Technical University of Munich
Chair of Real-Time Computer Systems (RCS)

03 August, 2019

Lehrstuhl für
Realzeit-Computersysteme

Introduction

Who are we?



Chair of Real-Time Computer Systems
TUM Department of Electrical and Computer Engineering
Technical University of Munich

Welcome to the RCS Webpages!

Our research and teaching activities encompass all aspects of real-time and embedded systems design. Our goal is to develop techniques and tools that would aid system designers to effectively design, debug and program real-time embedded systems.

Chair of Real-Time Computer Systems

Prof. Dr. sc.
Samarjit Chakraborty

Technical University of Munich
Arcisstraße 21
80333 Munich
Germany

Phone: +49-89-289-23550
Fax: +49-89-289-23555

Recent talks at the RCS

09.07.2018
Time Difference of Arrival localization of logistics IoT devices in 2.4 GHz IEEE 802.15.4 radio networks (Masterthesis) →

News from the RCS

20.06.2018

RCS Activities

What do we work on?

- System-level design of real-time and embedded systems
- Embedded control systems (cyber-physical systems)

RCS Activities

Some application domains:

- Sensor systems: Body-area sensor networks, indoor navigation
- Power management of portable devices (e.g., projects with Intel, Google)
- Automotive electronics and software
- Electric vehicles
- Battery management systems
- Drone platforms
- Timing analysis of embedded systems

What will we study in this course?

- Programming Intelligent **Physical Systems**?
  - What is the difference between programming computers and programming physical systems?
  - Connections to control theory?
  - Extensions of control theory.
- Connections with Cyber-Physical Systems (CPS)
- Computer + Physical System = CPS

Role of control theory in this course

- What does control theory deal with?
- We will look at how to systematically *implement* controllers on distributed information technology platforms
- While control theory is concerned with *designing* control algorithms or control strategies, we will be concerned with the *science of implementing* control algorithms

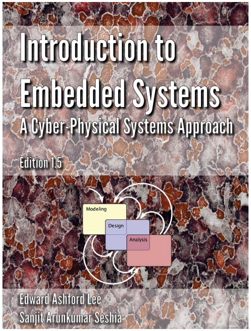
Difference between programming computers and programming physical systems

- Computers as data processing machines
- Computer program specifies how the input data is to be processed to generate output data
- But increasingly, we want to program a computer to make a robot **do** certain things, or a drone **to fly** in a certain manner, or a car to **drive** without colliding
  - In these cases, we need a model of the physical system
  - Then we need to actuate the system to enable it to behave in the desired fashion
  - Control theory provides us the tools for the above two steps
  - Techniques from cyber-physical systems (CPS) allow us to implement such controllers on computers or distributed embedded systems

What is a cyber-physical system?

E. Lee and S. Seshia, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach, 2014*. Online available at www.leeseshia.org.

"A cyber-physical system (CPS) is an integration of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa."



E. Lee, *Computing foundations and practice for cyber-physical systems: A preliminary report. 2007*

"..., we may think of cyber-physical systems to comprise embedded systems (the information processing part) and the physical environment."

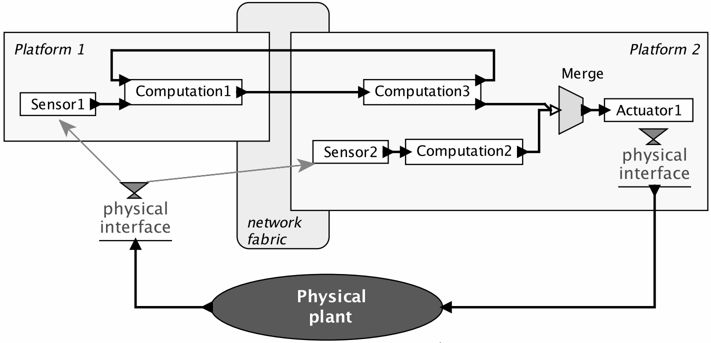


Figure: Common structure of an embedded/cyber-physical system.

Source: LeeSeshia14

Components of an embedded/cyber-physical system

- Physical plant is the physical part of the cyber-physical system consisting e.g. mechanical/electrical parts; biological/chemical processes.
- Cyber part consists of sensors, actuators, embedded computers, software and communication infrastructure.

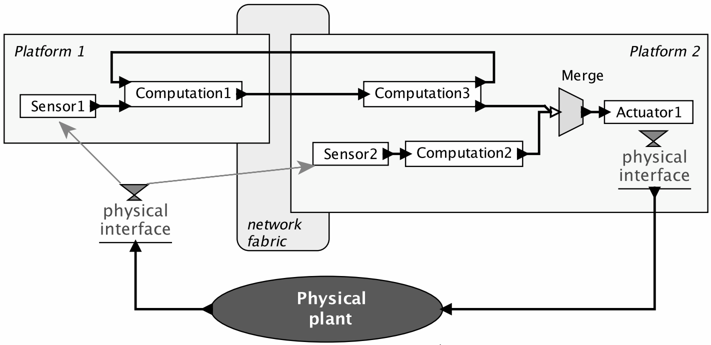
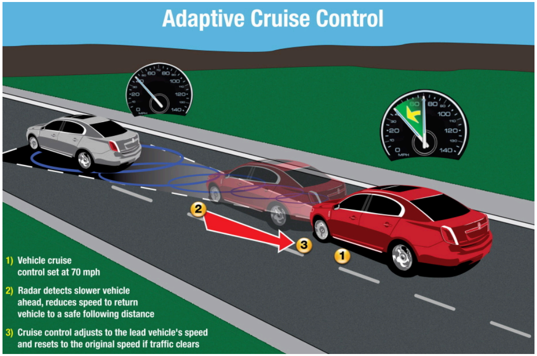


Figure: Common structure of an embedded/cyber-physical system.

Source: LeeSeshia14

Example: Adaptive Cruise Control

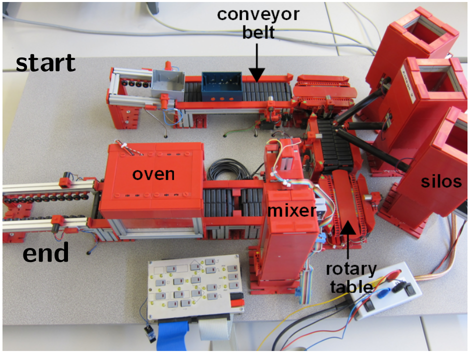

Adaptive Cruise Control
1) Vehicle cruise control set at 70 mph
2) Radar detects slower vehicle ahead, reduces speed to return vehicle to a safe following distance
3) Cruise control adjusts to the lead vehicle's speed and resets to the original speed if traffic clears
2
3
1
Image: http://blog.oakridgeford.com/

Different modeling concepts

- Ordinary differential equations are used to describe the vehicle dynamics.
- Finite state machines are used to model the different modes of the system, such as "vehicle ahead" or "no vehicle ahead".

Example: Production Lines

Baking line from the High-level Design Lab (RCS)



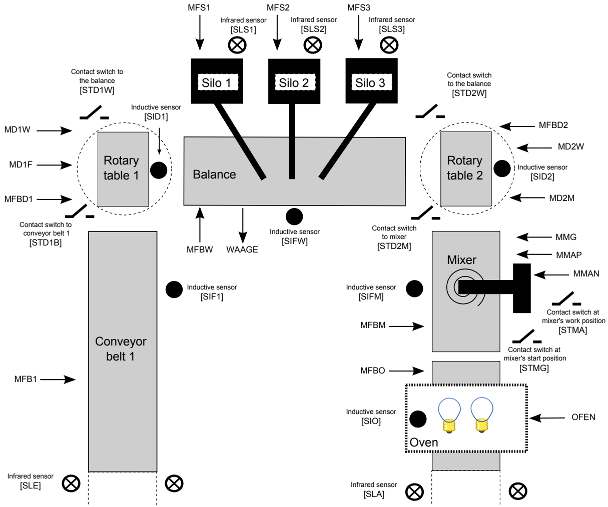
Components

- Boxes, oven, mixer, silos, conveyor belts, rotatory tables,...

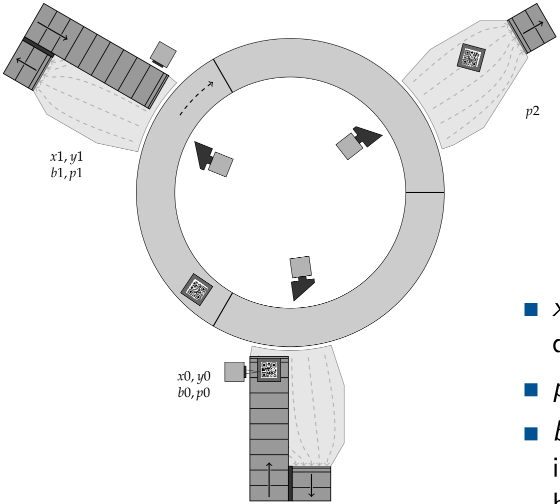
Description

- Boxes are moved from start position to end position.
- First stage: filling the boxes with material.
- Second stage: mixing the granulate material.
- Third stage: baking the material at a certain temperature in the oven.

Baking line: schematic overview with sensors and actuator signals



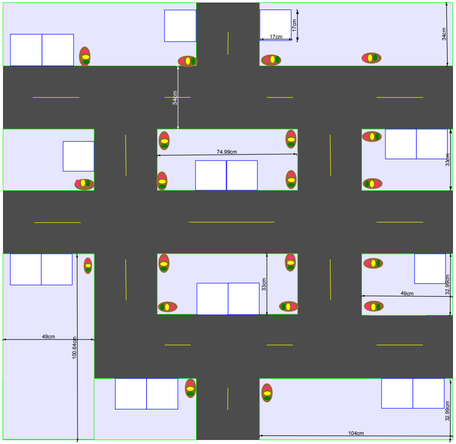
Rotational sorter



- x, y encode destination
- p activates the pusher
- b activates the incoming conveyor belt

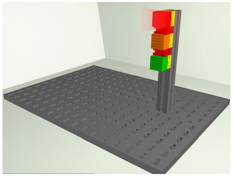
Example: Mobile Robots in an Urban Environment

Road network with parking lots



Mobile robots

Traffic lights



Embedded and cyber-physical systems are everywhere

- Industrial production systems
- Avionics
- Railway systems
- Automotive
- Mobile communication
- Medical devices
- Robotics
- Buildings and home applications
- ...

ACM Special Interest Group on Embedded Systems:
"..the potential for embedded computing quite literally everywhere is becoming a reality."

Your background

Please introduce yourself

- Name
- Program - Bachelors/Masters/PhD?
- Discipline - CS/EE/...?
- Have you done a course on control theory?
- Have you done a course on embedded systems?
- What is your understanding of CPS?

Coming up next ...

How to design a controller (based on classical control theory)?

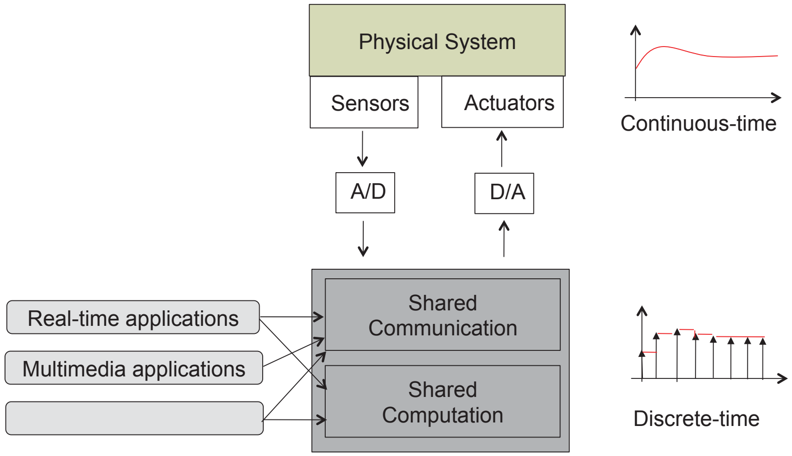
- Given a plant model, how to design a controller in the continuous time case
- How to design a controller in the discrete time case (since the controller will be implemented in software that will run on a digital platform)

Controller Design for Continuous-Time Systems

What is the meaning of designing a controller?

- Given a *plant*, we want to design a controller to enforce a desired behavior on the plant (actually on the combination of the plant and a controller)
- What are examples of such desired behaviors? Some state of the plant or its output approaches a specified reference with time
- For example, the temperature of a room approaches 22 deg C

Embedded Control Systems



Embedded platform: Control software

System Dynamics

Given physical system: Position control using DC motor

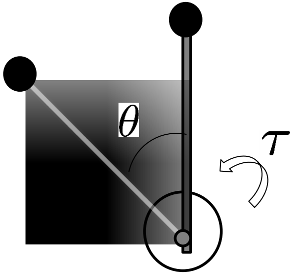


Figure: Inverted Pendulum

- In order to design the controller, we first need a mathematical model of the dynamics of the system
- $\theta =$ shaft angular position, $\tau =$ applied motor torque
- System dynamics: $\ddot{\theta} = 37\theta + 7.5\dot{\theta} + 6450\tau$

Laplace Transform

The Laplace transform of a time-domain function $f(t)$ is given by:

$$F(s) = L[f(t)] = \int_0^\infty f(t)\mathrm{e}^{-st}dt, t \geq 0$$

You may read L as "Laplace transform of" and s is a complex number given as $s = a + ib$
Some properties:

$$L\frac{d\int(f)}{dt} = [sF(s) - f(0^+)]$$
$$L\frac{d^2\int(f)}{dt^2} = [s^2F(s) - sf(0^+) - f'(0^+)]$$

System Models

- **Input-output model (Transfer function)** – suitable for *frequency domain analysis*
  - The relation between input $u(t)$ and output $y(t)$ (observed variable – e.g., position, temperature)
  - The analysis is done in frequency domain by taking Laplace transform

    $$G(s) = \frac{Y(s)}{U(s)}$$

    where s is the complex frequency
- **State-space model** – suitable for *time domain analysis*
  - Represent the system in terms of a number of internal states

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned}$$

Computing state-space model – Example

1. System dynamics: $\ddot{\theta} = 37\theta + 7.5\dot{\theta} + 6750\tau$
2. Input: $u = \tau$, and output: $y = \theta = x_1$
3. States: $x_1 = \theta; x_2 = \dot{\theta}$
4. State-space model:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = 37x_1 + 7.5x_1 + 6450u$$

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 37 & 7.5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 6450 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

Exercises

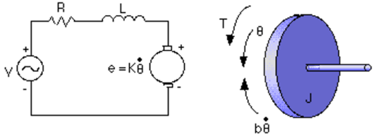
1. Find a state-space model for the following system:

$$\ddot{x} + 5\dot{x} + 6x = 8u$$

with output $y = x$ and input u.

2. Consider the follwing DC motor model:

$$J\ddot{\theta} + b\dot{\theta} = Ki$$
$$L\dot{i} + Ri = V - K\dot{\theta}$$



where θ is the shaft position, y the output, V is the terminal voltage and system input. J, b, K, L, R are constant motor parameters. Find a state-space model for this system.

Computing transfer function – Example

1. System dynamics: $\ddot{\theta} = 37\theta + 7.5\dot{\theta} + 6450\tau$
2. Input and Output: $u = \tau$ (input motor torque); $y = \theta$ (position)
3. Take Laplace transform with zero initial condition:

$$s^2\theta(s) = 37\theta(s) + 7.5s\theta(s) + 6450\tau(s)$$

4. Transfer Function:

$$\frac{\theta(s)}{\tau(s)} = \frac{6450}{s^2 - 7.5s - 37}$$

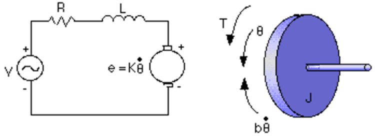
Exercises

1. Find the transfer function between x and u:

$$\ddot{x} + 5\dot{x} + 6x = 8u$$

2. Consider the following DC motor model:

$$J\ddot{\theta} + b\dot{\theta} = Ki$$
$$L\dot{i} + Ri = V - K\dot{\theta}$$



where θ is the shaft position, y the output, V is the terminal voltage and system input. J, b, K, L, R are constant motor parameters. Find the transfer function between θ and V.

State-space to Transfer Function

State-space model

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

Laplace transform and $x(0) = 0$

$$sX(s) = AX(s) + BU(s)$$
$$Y(s) = CX(s)$$
$$X(s) = (sI - A)^{-1}BU(s)$$
$$Y(s) = C(sI - A)^{-1}BU(s)$$
$$\frac{Y(s)}{U(s)} = C(sI - A)^{-1}B$$

$$\boxed{G(s) = C(sI - A)^{-1}B}$$

Poles and Zeros

- Given transfer function $G(s) = \frac{N(s)}{D(s)}$, the system poles are roots of the polynomial $D(s)$ and the system zeros are roots of polynomial $N(s)$
- Example:

$$G(s) = \frac{(s+0.2)(s+1)}{(s-1)(s-2)(s-3)}$$

  Zeros: $-0.2, -1$; Poles: $1, 2, 3$

- Example: Double Integrator:

$$G(s) = \frac{1}{s^2}$$

  Zeros: none; Poles: $0, 0$

Eigenvalues of a matrix

A number λ is an *eigenvalue* of an $n \times n$ matrix A if there is a nonzero n vector x such that

$$Ax = \lambda x$$

The corresponding vector x is the *eigenvector* of the matrix A.

Eigenvalues of a matrix

For a given λ, the eigenvector equation

$$Ax = \lambda x$$

is equivalent to the linear homogeneous equation

$$[A - \lambda I]x = 0$$

Such an equation possesses a nonzero solution if and only if the determinant of the coefficient matrix vanishes.
Hence, for λ to be an eigenvalue of the matrix A, the following should hold:

$$det[A - \lambda I] = 0$$

Characteristic equation

- Alternatively, system poles can also be computed from the state-space model by computing the *Characteristic Equation* which is given by:
$$det(\lambda I - A) = 0$$
- System *poles* are the *roots of the characteristic polynomial*
$$det(\lambda I - A)$$
- System *poles* are the *eigenvalues of A*
- Example: Double Integrator
$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$
$$\lambda I - A = \begin{bmatrix} \lambda & -1 \\ 0 & \lambda \end{bmatrix}$$
$$det(\lambda I - A) = \lambda^2$$

$\Rightarrow$ Poles at $0, 0$

Stability

There are various notions

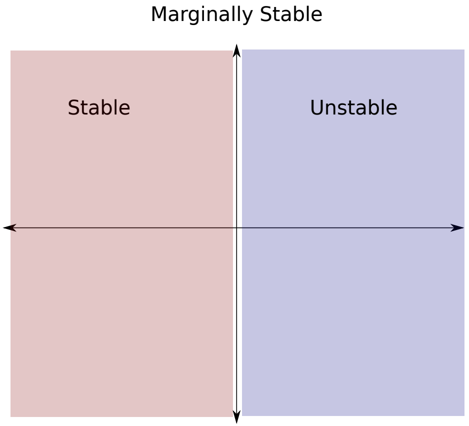
- Bounded-Input Bounded-Output (BIBO) stability
- Stability in the sense of Lyapunov
- Asymptotic stability
- Exponential stability
- etc.

Our lecture is mainly confined to the following aspects:

1. $|x(t)|, |y(t)|, |u(t)| < \infty$ (all signals are bounded),
2. $y(t) \to 0; t \to \infty$ (asymptotic stability)

Stability condition: continuous-time case

- Stable system: All poles should have negative real part
- Marginally stable system: One or multiple poles are on imaginary axis and all other poles have negative real parts
- Unstable system: One or more poles with positive real part.



Stability

- Example:

$$G(s) = \frac{(s+0.2)(s+1)}{(s-1)(s-2)(s-3)}$$

  Poles at $1, 2, 3 \Rightarrow$ Unstable!

- Example:

$$G(s) = \frac{1}{s^2}$$

  Poles at $0, 0 \Rightarrow$ Marginally stable

- Example:

$$G(s) = \frac{1}{s^2+4}$$

  Poles at $\pm 2i \Rightarrow$ Marginally stable

Summary

System dynamics: $\ddot{\theta} = 37\theta + 7.5\dot{\theta} + 6450\tau$

State Space

$$\ddot{x} = \begin{bmatrix} 0 & 1 \\ 37 & 7.5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 6450 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

$$A = \begin{bmatrix} 0 & 1 \\ 37 & 7.5 \end{bmatrix}$$

$$\lambda I - A = \begin{bmatrix} \lambda 1 & \\ -37 & \lambda - 7.5 \end{bmatrix}$$

$$det(\lambda I - A) = \lambda^2 - 7.5\lambda - 37$$

$\Rightarrow$ Poles at $10.9, -3.4$

Transfer Function

$$\frac{\theta(s)}{\tau(s)} = \frac{6450}{s^2 - 7.5s - 37}$$

$\Rightarrow$ Roots of $s^2 - 7.5s - 37$

$\Rightarrow$ Poles at $10.9, -3.4$

$\boxed{\Rightarrow \text{Unstable!}}$

Summary

How to compute system poles?

1. Roots of $det(\lambda I - A)$
2. Eigenvalues of system matrix A
3. Solution of system characteristics equation $det(\lambda I - A) = 0$
4. Roots of $D(s)$ for the transfer function $G(s) = \frac{N(s)}{D(s)}$

- Stable system: All poles with negative real part
- Marginally stable system: One or multiple poles are on imaginary axis and all other poles have negative real parts
- Unstable system: One or more poles with positive real part.

Summary

What have we learnt?

1. What are system dynamics, input-output model and transfer function
2. Stability is governed by system poles
3. Depending on the location of the poles, the system can be stable, unstable or marginally stable

Controller Design Problem

- We have a linear system given by the state-space model

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

- For n-dimensional system Single-Input-Single-Output (SISO) systems

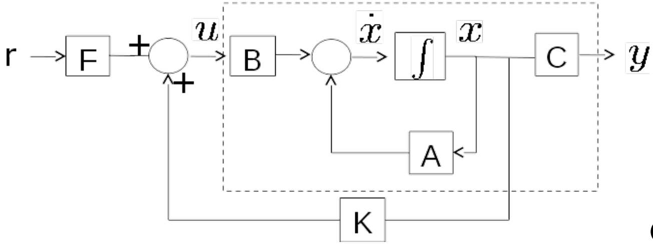
$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}'$$
$$A \in \mathbb{R}^n \times \mathbb{R}^n, B \in \mathbb{R}^n \times 1, C \in 1 \times \mathbb{R}^n$$

- Objective

$$y \to r; t \to \infty$$

- u = ?

State feedback



Open-loop system, i.e. with $u = 0 \Rightarrow \dot{x} = Ax$

Closed-loop system with state-feedback control: $u = Kx + Fr$

$$\dot{x} = (A + BK)x + BFr$$
$$y = Cx$$

Controller Design

- Control law

$$u = Kx + Fr$$

  r: reference, K: feedback gain, F: static feedforward gain
- How to design K?
- How to design F?

Feedback Gain: Ackermann's Formula

- Choose the desired closed-loop poles at

$$\begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n\end{bmatrix}$$

- Using Ackermann's formula we get:

$$\begin{aligned}K &= -\begin{bmatrix}0 & 0 & \cdots & 1\end{bmatrix}\gamma^{-1}H(A)\\ \gamma &= \begin{bmatrix}B & AB & A^2B & \cdots & A^{n-1}B\end{bmatrix}\\ H(A) &= (A-\alpha_1 I)(A-\alpha_2 I)(A-\alpha_3 I)\cdots(A-\alpha_n I)\end{aligned}$$

- Poles of $(A+BL)$ are at $\begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n\end{bmatrix}$

Static Feedforward Gain

$$u = Kx + Fr$$

K: pole placement; F: static feedforward gains are calculated as follows

$$\begin{aligned}
\dot{x} &= (A + BK)x + BFr \\
y &= Cx \\
\rightarrow X(s) &= (sI - A - BK)^{-1} BFR(S) \\
\rightarrow Y(s) &= CX(s) = C(sI - A - BK)^{-1} BFR(S) \\
\rightarrow G_{cl} &= \frac{Y(s)}{R(s)} = C(sI - A - BK)^{-1} BF
\end{aligned}$$

F should be chosen such that $y(t) \rightarrow r; t \rightarrow \infty$
Using final value theorem: $lim_{s \rightarrow 0} sY(s) = r; F = \frac{1}{C(-A-BK)^{-1}B}$

Summary of Overall Design

- Given System:

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

Objectives:

1. Place system poles
2. Achieve $y \rightarrow r; t \rightarrow \infty$
3. Design K and F

- Control Law : $u = Kx + Fr$

1. Check controllability of (A, B). γ must be invertible.

$$\gamma = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

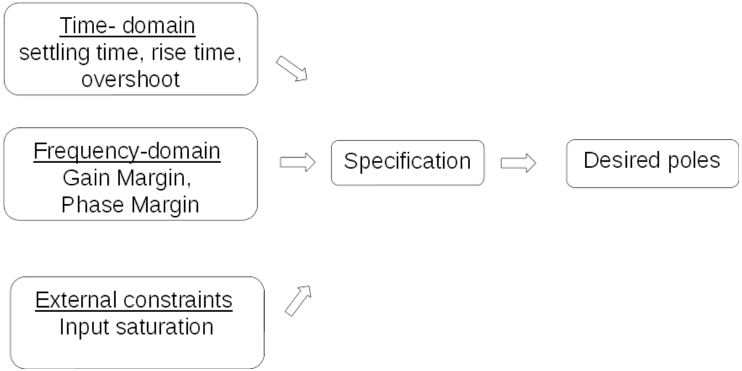
2. Apply Ackermann's formula

$$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(A)$$

3. Feedforward gain: $F = \frac{1}{C(-A - BK)^{-1}B}$

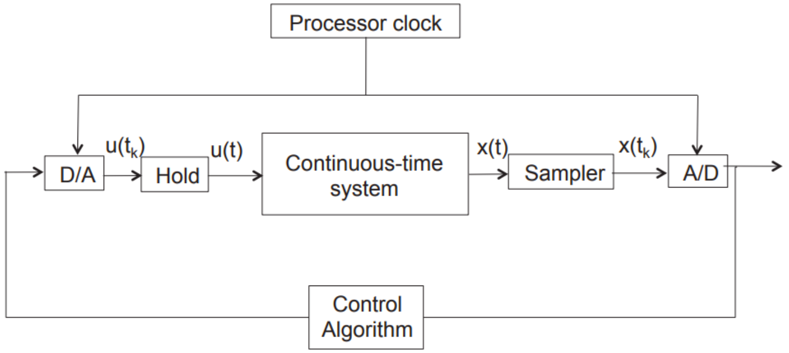
Summary: Pole Placement

It is reasonable to represent system performance by its poles.



Controller Design for Discrete-Time Systems

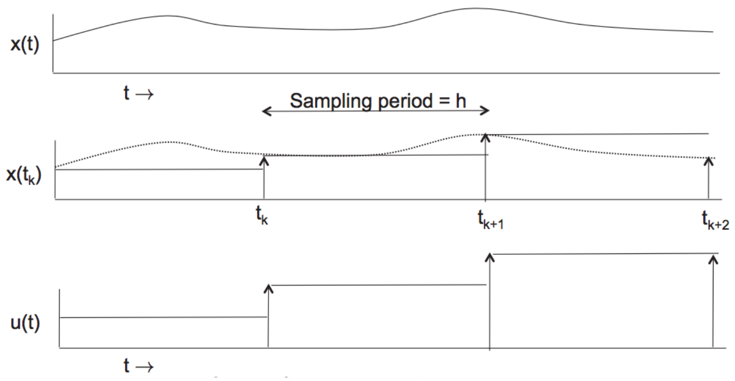
Digital Platform:Sample and Hold



D/A → digital-to-analog converter
A/D → analog-to-digital converter

- Input u(t) is piecewise constant
- Look at the sampling points

ZOH Sampling



piecewise constant

$$u(t) = u(t_k); t_k \leq t \leq t_{k+1}$$

Continuous to Discrete Case

The solution to the matrix equation

$$\frac{d\phi}{dt} = A\phi$$

where ϕ is an $n \times n$ matrix, given by

$$\phi(t) = e^{At}, \text{if } \phi(0) = I$$
$$e^{At} = I + At + \frac{A^2t^2}{2!} + \frac{A^3t^3}{3!} + \ldots$$

Discretization

- Continuous-time state-space model

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned}$$

- Taking Laplace transform

$$sX(s) - x(0) = AX(s) + BU(s)(sI - A)X(s) \quad = x(0) + BU(s)X(s)$$

- Taking inverse Laplace transform

$$\begin{aligned} \mathcal{L}^{-1}[X(s)] =& \mathcal{L}^{-1}[(sI - A)^{-1}]x(0) + \mathcal{L}^{-1}[(sI - A)^{-1}BU(s)] \\ x(t) =& e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \end{aligned}$$

Discretization: Basic Maths

- Periodic sampling: $t_{k+1} - t_k = h$ (sampling period is constant)
- We have

$$x(t_{k+1}) = e^{A(t_{k+1}-t_k)}x(t_k) + \int_0^{t_{k+1}-t_k} e^{As}Bds.u(t_k)$$

- Replacing $(t_{k+1} - t_k)$ with sampling period h in

$$x(t_{k+1}) = e^{Ah}x(t_k) + \int_0^{h} e^{As}Bds.u(t_k)$$

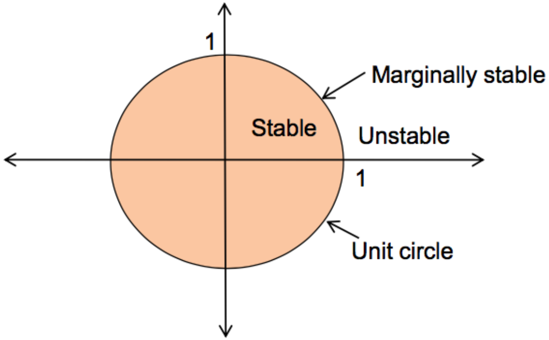
- We obtain $x(t_{k+1}) = \phi x(t_k) + \Gamma u(t_k)$
  where

$$\phi = e^{Ah}$$

$$\Gamma = \int_0^{h} e^{As}Bds$$

Discrete-time System Stability

- Stable system
  ⇒ Absolute values of all poles lesser than unity
- Marginally stable system
  ⇒ Absolute values of one or multiple poles are unity
- Unstable system
  ⇒ Absolute values of one or more poles are greater than unity



Design: Step 1 (Discretization)

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

ZOH periodic sampling with period = h

$$x[k+1] = \phi x[k] + \Gamma u[k]$$
$$y[k] = Cx[k]$$

where:

$$\phi = e^{Ah}$$
$$\Gamma = \int_0^h e^{As} B ds$$

$$e^{Ah} = I + Ah + \frac{A^2 h^2}{2!} + \frac{A^3 h^3}{3!} + \ldots$$

Design: Step 2 (Controller Design)

- Given System:

$$x[k+1] = \phi x[k] + \Gamma u[k]$$
$$y[k] = Cx[k]$$

Objectives:

1. Place system poles
2. Achieve $y \rightarrow r$ as $t \rightarrow \infty$
3. Design K and F

- Control Law :
  $u[k] = Kx[k] + Fr$

1. Check controllability of $(\phi,\Gamma) \rightarrow$ must be controllable. γ must be invertible.

$$\gamma = \begin{bmatrix} \Gamma & \phi\Gamma & \phi^2\Gamma & \cdots & \phi^{n-1}\Gamma \end{bmatrix}$$

2. Apply Ackermann's formula

$$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(\phi)$$

3. Feedforward gain: $F = \frac{1}{C(I-\phi-\Gamma K)^{-1}\Gamma}$

Step 2

- Given:

$$x[k+1] = \phi x[k] + \Gamma u[k]$$
$$y[k] = Cx[k]$$
$$\phi \in \mathbb{R}^n \times \mathbb{R}^n, \Gamma \in \mathbb{R}^n \times 1, C \in 1 \times \mathbb{R}^n$$

- The control input $u[k] = Kx[k]$ such that closed-loop poles are at

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \end{bmatrix}$$

- Using Ackermann's formula:

$$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(\phi)$$

where

$$\gamma = \begin{bmatrix} \Gamma & \phi\Gamma & \phi^2\Gamma & \cdots & \phi^{n-1}\Gamma \end{bmatrix}$$
$$H(\phi) = (\phi - \alpha_1 I)(\phi - \alpha_2 I)(\phi - \alpha_3 I)\cdots(\phi - \alpha_n I)$$

Continuous Vs Discrete Time

Continuous-time $\dot{x} = Ax + Bu$, $y = Cx$	**ZOH periodic sampled** $x[k+1] = \phi x[k] + \Gamma u[k]$, $y[k] = Cx[k]$
Input: $u = Kx + Fr$	Input: $u[k] = Kx[k] + Fr$
Controllability matrix: $\gamma = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$	Controllability matrix: $\gamma = \begin{bmatrix} \Gamma & \phi\Gamma & \phi^2\Gamma & \cdots & \phi^{n-1}\Gamma \end{bmatrix}$
$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(A)$	$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(\phi)$
$F = \frac{1}{C(-A-BK)^{-1}B}$	$F = \frac{1}{C(I-\phi-\Gamma K)^{-1}\Gamma}$

What's Next?

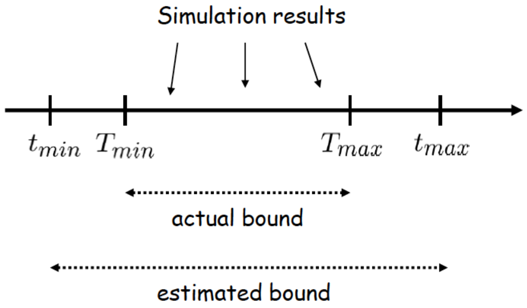
- Although we have assumed discrete sampling with ZOH, we have also assumed that the control input is available exactly at the same time when the system state has been sampled.
- In many real systems, it might take non-negligible time to compute the control input and also to communicate the control signal.
- As a result, the system dynamics will be different from what we have considered.
- How should the controller be designed in such a case?

We will next discuss how to perform timing analysis of distributed embedded platforms. In particular, we will look at:

- Worst-Case Execution Time Analysis
- Schedulability Analysis

Basics of timing analysis: Worst Case Execution Time (WCET) analysis of programs running on a single processor

WCET estimation

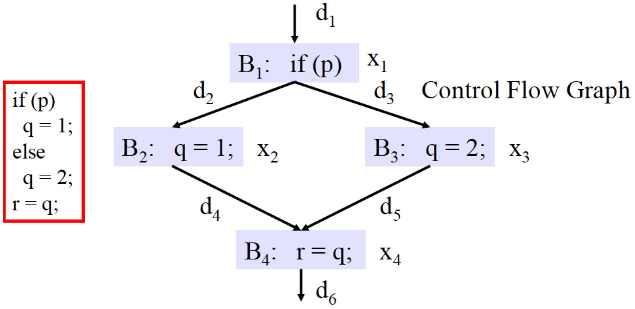


- Estimated bounds should enclose the actual bounds (be *safe*)
- The goal is to obtain bounds that are as tight as possible, i.e., t_{min} is almost equal to T_{min} and t_{max} is almost equal to T_{max}

WCET estimation: constrained optimization problem

- Execution time of a program stems from the time it takes to execute the various instructions in the program
- Hence, total execution time of a program $E = \sum_{i=1}^{N} c_i x_i$, where c_i is the execution time of instruction i and x_i is the number of times the instruction i is executed. N is the number of different instructions in the program
- WCET $= \max E$ subject to certain constraints on how many times the different instructions can be executed
- Because of the constraints, not all possible values of x_is are feasible (because not all program paths might be feasible)

What are the constraints?



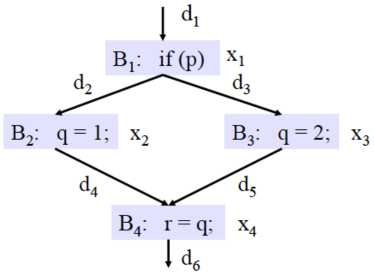
```
if (p)
  q = 1;
else
  q = 2;
r = q;
```

- x_i is the number of times the basic block i is executed
- d_i is the number of times a particular edge is followed

What is a basic block?

- It is a block of code with only one entry and one exit point
- Once a code enters a basic block then all the instructions in the basic block are executed
- There are no conditional branches inside a basic block
- The execution counts of all instructions in a basic block are the same
- Hence, total execution time of a program $E = \sum_{i=1}^{N} c_i x_i$, where c_i is the execution time of basic block i and x_i is the number of times the basic block i is executed. N is the number of different basic blocks in the program. By using basic blocks instead of instructions, we reduce the number of variables in the optimization problem $\text{WCET} = \max E$

Structural constraints



$$x_1 = d_1 = d_2 + d_3$$
$$x_2 = d_2 = d_4$$
$$x_3 = d_3 = d_5$$
$$x_4 = d_4 + d_5 = d_6$$

8 structural constraints

- The structural constraints stem from the control flow in the program
- These constraints can be automatically derived by analyzing the control flow graph of the program

Logical constraints

```
x1   i = 0;
x2   while (i < 100)
x3     {if (ok)
x4         j++;
         else
x5       { k++;
           ok = 1;}
x6       i++ }
```

- Logical constraints arise from the logical flow in the program
- Examples: $x5 \leq 1$ and $x4 \geq 99$ (do you understand why?)
- These constraints might not always be automatically derivable

WCET as constrained optimization

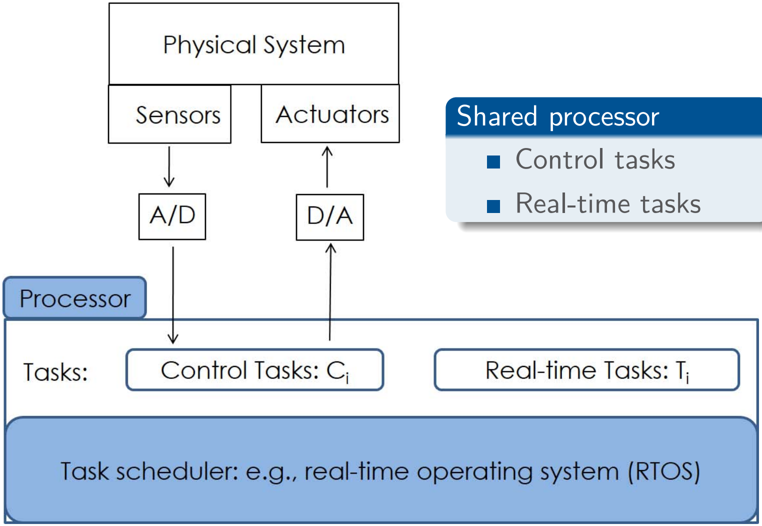
- WCET $= \max E$, where $E = \sum_{i=1}^{N} c_i x_i$, and c_i is the execution time of basic block i, x_i is the number of times the basic block i is executed, and N is the number of different basic blocks in the program, subject to the structural and logical constraints
- Since the objective function is linear, all the constraints are linear, and we are only interested in integer valuations of the x_is, this is an integer linear programming problem (ILP) and can be solved by an ILP solver (see: `http://www.gurobi.com/resources/switching-to-gurobi/open-source-solvers`)
- Examples of WCET analyzers: `www.absint.com`

Microarchitecture modeling for WCET estimation

- So far we assumed that the execution time of an instruction (or a basic block) is constant
- But this is not true. For example, the execution time of an instruction will be different depending on whether or not it is in the cache
- The ILP formulation we saw can be extended to incorporate the effects of caches, pipelines, and speculative execution (like branch prediction)
- Current WCET analyzers use a combination of ILP, abstract interpretation, and model checking
- See: Ravindra Metta, Martin Becker, Prasad Bokil, Samarjit Chakraborty, R. Venkatesh: *TIC: a scalable model checking based approach to WCET estimation*. 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems (LCTES), extended version to appear in Software Tools for Technology Transfer, `https://arxiv.org/abs/1802.09239`

Schedulability analysis for a single processor platform

Implementation on a shared processor



Shared processor

- Control tasks
- Real-time tasks

Problem description

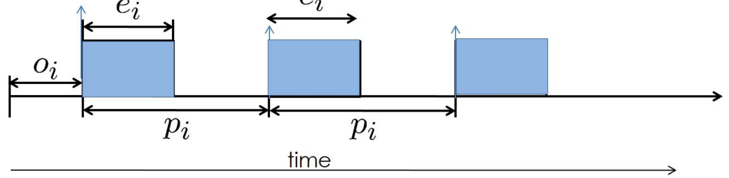
- Multiple real-time tasks are running on the processor. The real-time tasks are denoted by T_i.
- There are one or more control applications (control tasks) running on the same processor. These tasks are denoted by C_i.
- All tasks are executed on a single processor. The processor is the shared computational resource.
- A task scheduler chooses which task to execute at a given time.
- This is a very common setup where a control application has to share the computational resource with other real-time tasks.
- The problem is that the control applications and real-time applications have different types of requirements (will be explained in the coming slides). This need to be taken into account in the design process.

Processor

- A task T_i is a piece of code (e.g., c, Java language code) which implements a specific functionality.
- Usually, multiple tasks run on a processor.
- A processor runs a real-time operating system (RTOS) that manages the execution of the tasks according to their schedules (task schedulers).
- Scheduling
  - Choice of which task to execute at a given time.
  - The tasks can be scheduled either in time-triggered or in event-triggered fashion depending on the RTOS and the scheduler.

Time-triggered tasks

- The time-trigged tasks are often periodic.



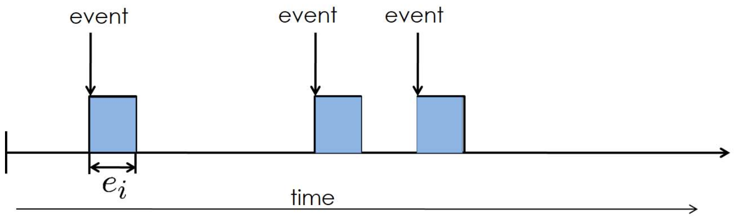
- Schedule for a task T_i: $\{o_i, e_i, p_i\}$
  - o_i: the task offset
  - e_i: the worst-case execution time
  - p_i: the task period
- An instance of the task is called a job.

Time-triggered tasks

- A periodic dispatcher is used to trigger the tasks.
- The start time of the k-th job of a periodic time-triggered task T_i is given by:
$$t_i(k) = o_i + k \times p_i.$$
- The finish time of the k-th job of a periodic time-triggered task T_i is given by:
$$\widehat{t_i}(k) = o_i + k \times p_i + e_i.$$
- Time-predictable.
- Inflexible since the task schedules are pre-defined. This results in poor resource utilization.

Event-triggered tasks

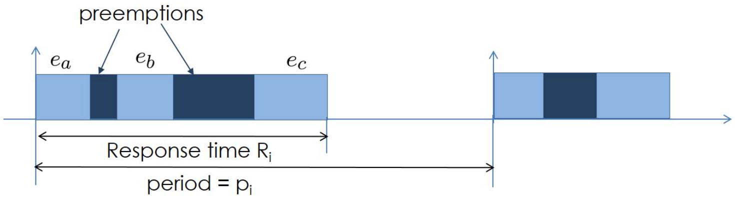
- The task is triggered in an event-driven fashion.
- Event can be a processor interrupt or an external signal (e.g., voltage pulse, switch).



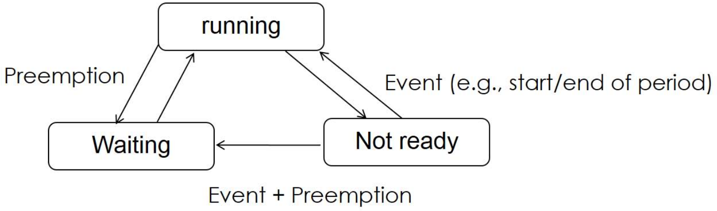
- T_i: e_i.
- Better resource utilization and flexible.
- Not time-predictable – real-time properties need to be verified and guaranteed by the designer.

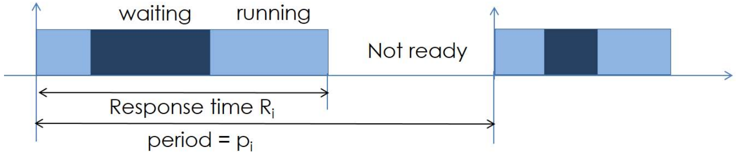
Preemptions and response time

- The worst-case execution time is the sum of all execution segments
$$e_i = e_a + e_b + e_c + \cdots .$$
- The worst-case response time is the longest time taken by the processor to allow e_i time of execution for the task T_i.
- When preempted: the processor either runs another higher priority task or be idle (for some other reason such as heating and thermal problems).
- A scheduler can either be preemptive or non-preemptive.

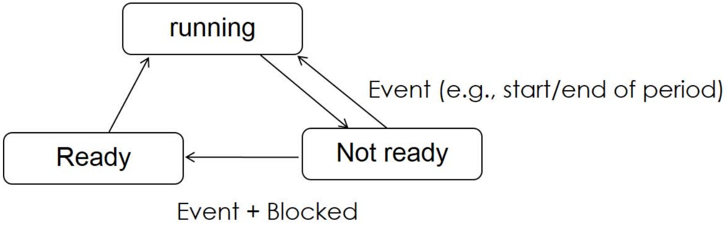


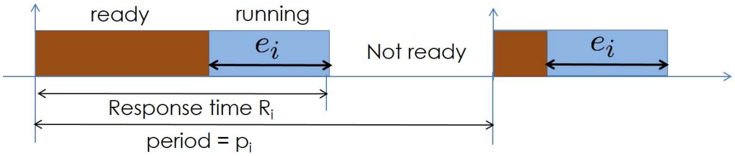
Preemptive tasks





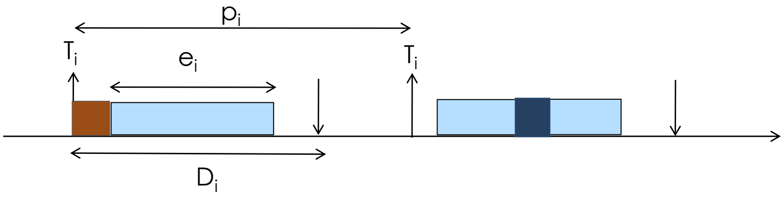
Non-preemptive tasks





Real-time task model

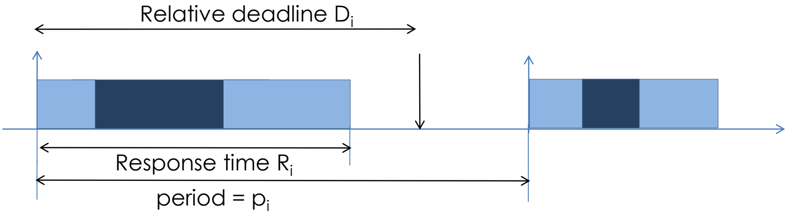
- Period p_i
- Relative Deadline D_i
- WCET e_i



Schedulability

A task set is schedulable if and only if all jobs of all tasks meet their relative deadline D_i, i.e.

$$R_i \leq D_i, \forall i$$



Real-time tasks – schedulability

- For hard real-time tasks, the deadlines must always be met.
- A system is not schedulable if the scheduler cannot find a way to switch between the tasks such that the deadlines are met.
- The test is sufficient if, when it answers "Yes", all deadlines will be met.
- The test is necessary if, when it answers "No", there really is a situation where deadlines could be missed.
- The test is exact if it is both sufficient and necessary.
- A sufficient test is an absolute requirement and one likes it to be as close to necessary as possible (known as tightness of an analysis).

Processor utilization

Task	period	deadline	WCET
T_1	p_1	D_1	e_1
T_2	p_2	D_2	e_2
T_3	p_3	D_3	e_3
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Total processor utilization: $U = \sum_i \frac{e_i}{p_i}$

- The worst-case utilization is very important performance metric for schedulability
- Often, a design criteria is $U \leq U_{limit}$
- For 100% processor utilization $U = 1.0$
- For 100% processor utilization implies that the processor has no idle time

Processor Load

Tasks	period	deadline	WCET
T_1	p_1	D_1	e_1
T_2	p_2	D_2	e_2
T_3	p_3	D_3	e_3
$\dots$	$\dots$	$\dots$	$\dots$

Demand bound function: $h(t) = \sum_i^n max(0, \lfloor \frac{t-D_i}{p_i} + 1 \rfloor) \times e_i$

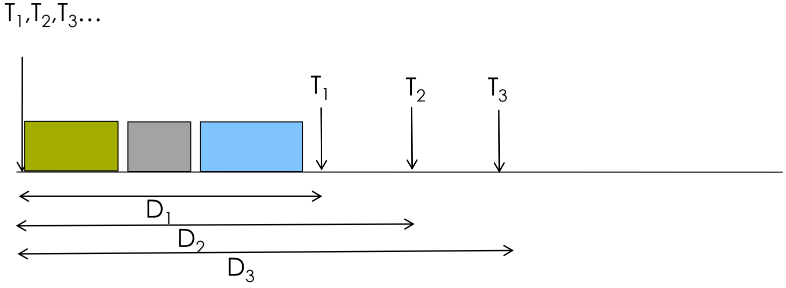
Processor Load: $\max_{\forall t}(\frac{h(t)}{t})$

$\boxed{\text{For schedulability in the uni-processor case: Load} \leq 1, \forall t}$

The critical instant

- It can be shown that, in the single processor case, the worst situation, from a schedulability perspective, occurs when all tasks want to start their execution at the same time instant
- This is known as the critical instant.
- If we can show that the task set is schedulable in this situation, it will also be schedulable in other situations.
- If we can show that the task set is schedulable for the worst case execution times, then the task set will also be schedulable if the actual execution times are shorter.
- Hence, all single processor (uni-processor) scheduling analysis only needs to check for this case.

Tasks	period	deadline	WCET
T_1	p_1	D_1	e_1
T_2	p_2	D_2	e_2
T_3	p_3	D_3	e_3
...	...	...	...



Scheduling the tasks onto a processor

- Fixed priority
  - Deadline monotonic (DM)
  - Rate monotonic (RM)
  - ...
- Dynamic priority
  - Earliest Deadline First (EDF)
  - ...

Fixed priority preemptive

- Each task has a fixed priority
- The task dispatcher selects the task with the highest priority
- All tasks are preemptive: if a higher priority task arrives while a lower priority task is running, the lower priority task will be stopped and higher priority task will be executed
- There are different ways to assign priority to the tasks (priority assignment problems). Well-known schemes are:
  - Deadline monotonic
  - Rate monotonic

Rate monotonic (fixed priority)

- n periodic tasks $T_i : p_i, D_i, e_i$
- Priorities are set monotonically to the periods – a task with a shorter period is assigned higher priority and preemptive
- Processor utilization: $U = \sum_{i=1}^{n} \frac{e_i}{p_i}$
- $D_i = p_i$, i.e. Deadline = period
- Schedulability test (sufficient condition):

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq n(2^{1/n} - 1)$$

where n is the number of scheduled tasks. The above test is conservative compared to response time analysis.

Example 1

Consider the given fixed priority and preemptive task set running on a processor. The priorities are assigned using a rate monotonic scheme. Perform a schedulability test if all the tasks are going to meet their deadline under a rate monotonic scheme.

Tasks	p_i (ms)	D_i (ms)	e_i (ms)	priority
T_1	15	15	3	1 (highest)
T_2	20	20	8	2
T_3	30	30	12	3

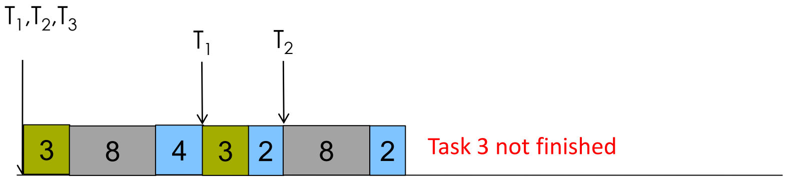
$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} = \frac{3}{15} + \frac{8}{20} + \frac{12}{30} = 1 > n(2^{1/n} - 1) = 3 \times (2^{1/3} - 1) = 0.78$$

$\Rightarrow$ Not schedulable!

Example 1

Tasks	p_i (ms)	D_i (ms)	e_i (ms)	priority
T_1	15	15	3	1 (highest)
T_2	20	20	8	2
T_3	30	30	12	3

Timing at the critical instant



$\Rightarrow$ Not schedulable!

Example 2

Consider the given fixed priority and preemptive task set running on a processor. The priorities are assigned using rate monotonic scheme. Perform a schedulability test if all the tasks are going to meet their deadline under a rate monotonic scheme.

Tasks	p_i (ms)	D_i (ms)	e_i (ms)	priority
T_1	15	15	3	1 (highest)
T_2	20	20	8	2
T_3	30	30	4	3

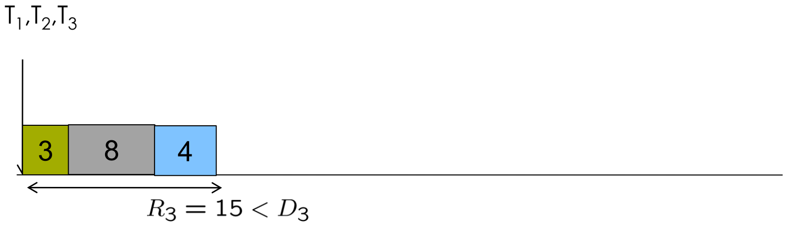
$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} = \frac{3}{15} + \frac{8}{20} + \frac{4}{30} = 0.73 < n(2^{1/n} - 1) = 3 \times (2^{1/3} - 1) = 0.78$$

$\Rightarrow$ Schedulable!

Example 2

Tasks	p_i (ms)	D_i (ms)	e_i (ms)	priority
T_1	15	15	3	1 (highest)
T_2	20	20	8	2
T_3	30	30	4	3

Timing at the critical instant



$\Rightarrow$ Schedulable!

Deadline monotonic (fixed priority)

- n periodic tasks $T_i : \{p_i, D_i, e_i\}$
- Priorities are set monotonically to the deadline – a task with a shorter deadline is assigned a higher priority.
- Preemptive.
- Processor utilization

$$U = \sum_{i=1}^{i=n} \frac{e_i}{p_i}$$

- Deadline $D_i <$ period p_i
- Schedulability test (sufficient condition)

$$\sum_{i=1}^{i=n} \frac{e_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$$

where n is the number of scheduled tasks. The above test is conservative compared to response time analysis.

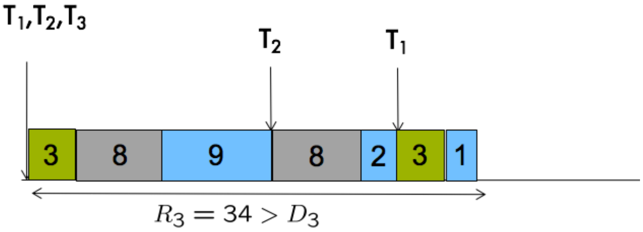
- Example 3
  Consider the given fixed priority and preemptive task set running on a processor. The priorities are assigned using deadline monotonic scheme. Perform a schedulability test if all the tasks are going to meet their deadlines.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority
T_1	30	15	3	1
T_2	20	20	8	2
T_3	40	30	12	3

$$\begin{aligned}\sum_{i=1}^{i=3} \frac{e_i}{D_i} &= \frac{3}{15} + \frac{8}{20} + \frac{12}{30} \\ &= 1.0 \quad \Rightarrow \text{Not schedulable!} \\ &> 3 \times (2^{\frac{1}{3}} - 1) > 0.78\end{aligned}$$

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority
T_1	30	15	3	1
T_2	20	20	8	2
T_3	40	30	12	3

- Timing at the critical instant



Not schedulable

- Example 4
  Consider the given fixed priority task and preemptive set running on a processor. The priorities are assigned using deadline monotonic scheme. Perform a schedulability test if all the tasks are going to meet deadline under deadline monotonic scheme.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority
T_1	30	15	3	1
T_2	20	20	4	2
T_3	40	30	4	3

$$\sum_{i=1}^{i=3} \frac{e_i}{D_i} = \frac{3}{15} + \frac{4}{20} + \frac{4}{30}$$

$$= 0.5333 \quad \Rightarrow \text{Schedulable!}$$

$$\leq 3 \times (2^{\frac{1}{3}} - 1) \leq 0.78$$

- Example 5

  Consider the given fixed priority and preemptive task set running on a processor. The priorities are assigned using deadline monotonic scheme. Perform a schedulability test if all the tasks are going to meet deadline under deadline monotonic scheme.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority
T_1	30	15	3	1
T_2	20	20	4	2
T_3	40	30	4	3
T_4	50	40	5	4

$$\sum_{i=1}^{i=4} \frac{e_i}{D_i} = \frac{3}{15} + \frac{4}{20} + \frac{4}{30} + \frac{5}{40} = 0.6583 \quad \Rightarrow \text{Schedulable!}$$

$$\leq 4 \times (2^{\frac{1}{4}} - 1) \leq 0.7568$$

Earliest Deadline First (dynamic scheduling)

- Dynamic dispatcher: all scheduling decisions are made online. Optimal schedule and 100% utilization is possible when $D_i = p_i$
- The task with the shortest relative deadline runs
- Preemptive
- Task model $T_i : \{p_i, D_i, e_i\}$
- Deadline $D_i =$ period p_i
- Schedulability test (necessary and sufficient test):

$$U = \sum_i \frac{e_i}{p_i} \leq 1$$

- Example 6

  Consider the given fixed priority and preemptive task set running on a processor. The tasks are running under EDF scheme. Perform a schedulability test if all the tasks are going to meet deadline under EDF.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)
T_1	30	30	3
T_2	20	20	8
T_3	40	40	12

$$U = \sum_{i=1}^{i=3} \frac{e_i}{p_i} = \frac{3}{30} + \frac{8}{20} + \frac{12}{40}$$

$$= 0.8 \quad \Rightarrow \text{Schedulable!}$$

$$\leq 1$$

- Example 7

  Consider the given fixed priority and preemptive task set running on a processor. The tasks are running under EDF scheme. Perform a schedulability test if all the tasks are going to meet deadline under EDF.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)
T_1	30	30	3
T_2	20	20	4
T_3	40	40	4
T_4	50	50	5

$$U = \sum_{i=1}^{i=4} \frac{e_i}{p_i} = \frac{3}{30} + \frac{4}{20} + \frac{4}{40} + \frac{5}{50}$$
$$= 0.5 \quad \Rightarrow \text{Schedulable!}$$
$$\leq 1$$

Response time analysis

- A task set $\{T_i\}$.
- Each task is represented as $T_i \sim \{p_i, D_i, e_i\}$.
- Each task has a fixed unique priority (DM or RT or any other scheme).
- All tasks are preemptive.
- For each task T_i: $D_i \leq p_i$.
- Response time of a task T_i is denoted as R_i.
- Our objective is to make sure that for each task T_i: $R_i \leq D_i$.
- Hence, we need to compute the response time R_i for each task.

Response time analysis

- Response time with fixed priority preemptive scheduling for the given task set is given by,

$$R_i = e_i + \sum_{\forall j \in hp(i)} \lceil \frac{R_i}{p_j} \rceil e_j$$

where $hp(i)$ is the set of tasks of higher priority than task T_i. Ceiling function $\lceil x \rceil$returns the smallest integer greater than x.

- Recurrence relation needs to be solved iteratively using

$$R_i^{n+1} = e_i + \sum_{\forall j \in hp(i)} \lceil \frac{R_i^n}{p_j} \rceil e_j.$$

Start with $R_i^0 = 0$.

- Example 8

  Consider the given fixed priority task set running on a processor. Compute response time for all the tasks. Check if real-time tasks meeting their deadlines and control task meets its design constraint.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority	Remark
T_1	30	15	3	2	Real-time task
T_2	20	12	8	1(highest)	Real-time task
T_c	$h = 30$	$D_c = 30$	12	3	control task

- Note that the control task T_c does not have a deadline. However, the design constraint is that $R_c \leq D_c$.

$R_2^0 = 0, R_2^1 = 8, R_2^2 = 8 \rightarrow R_2 = 8$
Clearly, $R_2 < D_2 \rightarrow$ deadline meet

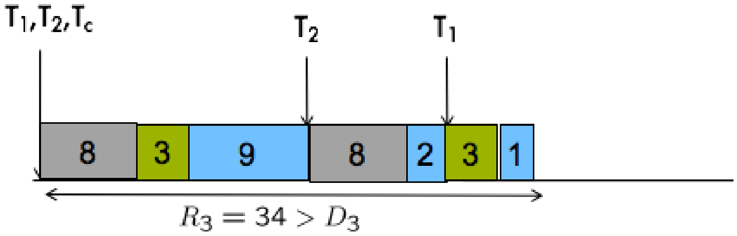
$R_1^0 = 0, R_1^1 = 3, R_1^2 = 3 + \lceil \frac{R_1^1}{p_2} \rceil e_2 = 11,$
$R_1^3 = 3 + \lceil \frac{R_1^2}{p_2} \rceil e_2 = 11, \rightarrow R_1 = 11$
Clearly, $R_1 < D_1 \rightarrow$ deadline meet

$R_c^0 = 0, R_c^1 = 12,$
$R_c^2 = 12 + \lceil \frac{R_c^1}{p_1} \rceil e_1 + \lceil \frac{R_c^1}{p_2} \rceil e_2 = 23,$
$R_c^3 = 12 + \lceil \frac{R_c^2}{p_1} \rceil e_1 + \lceil \frac{R_c^2}{p_2} \rceil e_2 = 31,$
$R_c^4 = 12 + \lceil \frac{R_c^3}{p_1} \rceil e_1 + \lceil \frac{R_c^3}{p_2} \rceil e_2 = 34, \rightarrow R_c = 34$
$R_c > D_c$ violation of design constraints

- The task set is not schedulable under the given priority since control task violates the design constraint $R_c > D_c$.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority	Remark
T_1	30	15	3	2	Real-time task
T_2	20	12	8	1(highest)	Real-time task
T_c	$h = 30$	$D_c = 30$	12	3	control task

- Timing at the critical instant



Not schedulable!

- Example 9
  Consider the given fixed priority task set running on a processor. Compute response time for all the tasks. Check if real-time tasks meeting their deadlines and control task meets its design constraint.

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority	Remark
T_1	30	15	3	2	Real-time task
T_2	20	12	8	1(highest)	Real-time task
T_c	$h = 50$	$D_c = 50$	12	3	control task

$R_2 = 8 < D_2 \rightarrow$ deadline meet

$R_1 = 11 < D_1 \rightarrow$ deadline meet

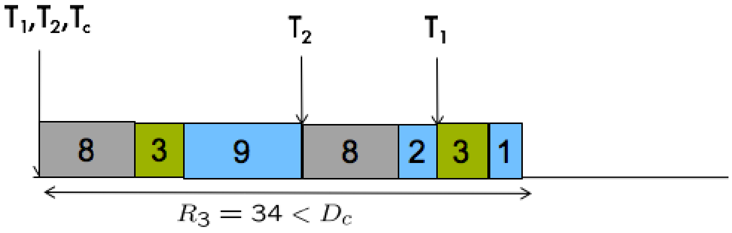
$$R_c^0 = 0, R_c^1 = 12,$$
$$R_c^2 = 12 + \lceil \frac{R_c^1}{p_1} \rceil e_1 + \lceil \frac{R_c^1}{p_2} \rceil e_2 = 23,$$
$$R_c^3 = 12 + \lceil \frac{R_c^2}{p_1} \rceil e_1 + \lceil \frac{R_c^2}{p_2} \rceil e_2 = 31,$$
$$R_c^4 = 12 + \lceil \frac{R_c^3}{p_1} \rceil e_1 + \lceil \frac{R_c^3}{p_2} \rceil e_2 = 34, \rightarrow D_c = 34$$

$R_c < D_c$ meeting design constraints

- The task set is schedulable under the given priority and control task also meeting deadline. The control task with sampling period h=30ms is not feasible on the processor (as we have seen in Example 8). When we choose a longer sampling period h=50ms, the design becomes feasible (as we could see in Example 9).

Tasks	p_i(ms)	D_i(ms)	e_i(ms)	priority	Remark
T_1	30	15	3	2	Real-time task
T_2	20	12	8	1(highest)	Real-time task
T_c	$h = 50$	$D_c = 50$	12	3	control task

- Timing at the critical instant



Schedulable!

Next lectures:

- Accounting for platform timing properties in controller design
- Timing analysis for communication architectures