EL2450 Hybrid and Embedded Control

Spring 2019

Dimos Dimarogonas
http://people.kth.se/~dimos/

Decision and Control Systems
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

EL2450 Hybrid and Embedded Control

- **Disposition**
  7.5 credits, 28h lectures, 28h exercises, 3 homework
- **Instructors**
  Dimos Dimarogonas, lecturer, dimos@kth.se
  David Umsonst, teaching assistant, umsonst@kth.se
  Dirk van Dooren, teaching assistant, dirkvd@kth.se
  Christos Verginis, teaching assistant, cverginis@kth.se

Course goal

Participants should gain insight into computer implementation of control algorithms, and realize the need to integrate control and computer engineering in the design of networked embedded control systems

Milestones

After the course, you should be able to

- Analyze, design and implement sampled-data control systems
- Characterize possibilities and limitations of real-time operating systems through mathematical models
- Appreciate flexibilities and compensate for uncertainties in networked control systems
- Apply hybrid systems modeling and analysis techniques to embedded systems

EL2450 Hybrid and Embedded Control

Lecture 1: Introduction

- Practical Information
- What is Hybrid and Embedded?
- Motivating example
- Course outline
- Review of sampled signals

Course Information

- All info available at

  `https://kth.instructure.com/courses/7591`

Material

- **Reading Material:** Book chapters and papers, reading per lecture found at course homepage
- **Lecture notes:** Available online after each lecture
- **Exercises:** Class room and home exercises
- **Homework:** 2 computer and one laboratory exercises
- **Software:** Matlab etc (see homepage)

Homework need to be submitted on time (no exceptions)

What is an Embedded System?

Computer system with the computer **embedded** in the application/physical process
Contrast with general-purpose and desk-top computers

Example: Design of VDC System

Vehicle dynamics control[1] (VDC) systems assist car driver in oversteering, under-steering and roll-over situations

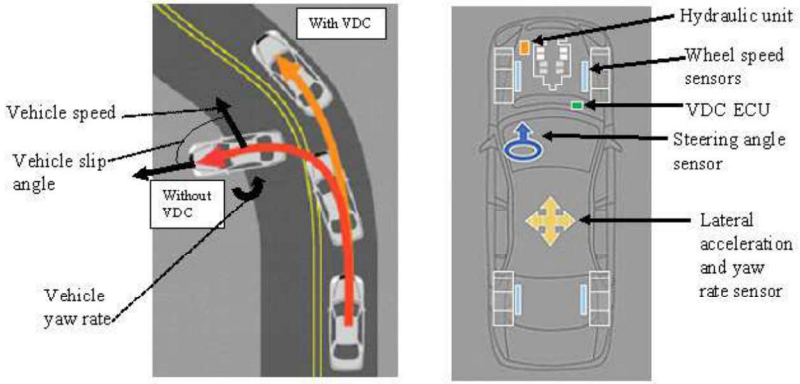
System design flow

1. Product specification: control objectives
2. Architecture definition: control structure, communication
3. Software development: control algorithms, filters
4. Physical implementation

[1] Also known as electronic stability program, dynamic stability control, or active yaw control.

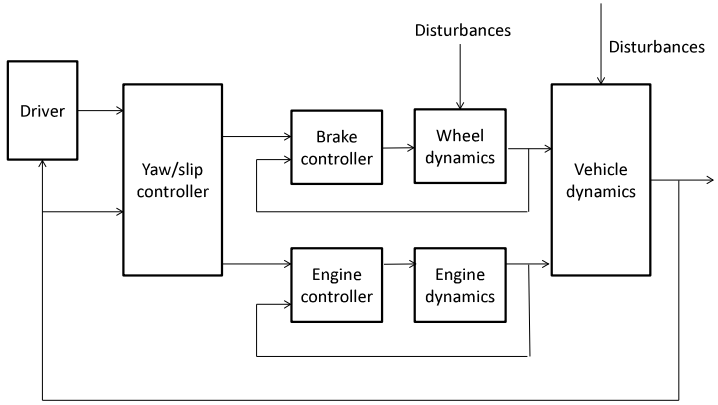
VDC Over-Steering

1. Friction limits reached for rear wheels
2. Car starts to skid, i.e., yaw rate (*girvinkel*) and side slip angle deviate from driver's intended
3. VDC detects emerging skidding and computes compensating torque
4. Applies braking force to outer front wheel



VDC Control Architecture

VDC is a cascade control structure with three controllers:

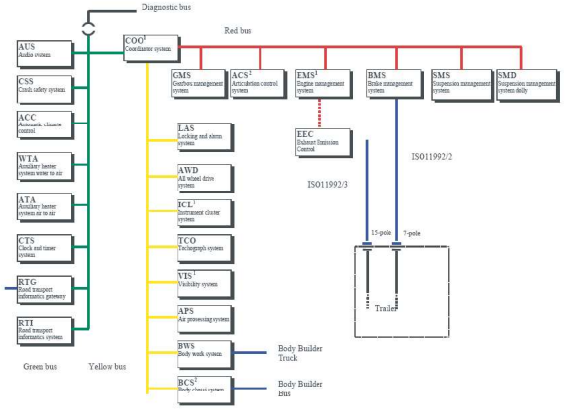


VDC Communication Architecture

- VDC is a networked embedded control system
- Utilizes the Controller Area Network (CAN) bus
- CAN is a communication bus that connects sensor, actuator and controller nodes
- A modern car has several such electronic control units

Scania Truck CAN

A Scania truck has three CAN buses



Characteristics of Embedded Systems

- Computational systems (but not a computer)
- Integrated with physical world via sensors and actuators
- Reactive (at the speed of the environment)
- Heterogeneous (mixed hw/sw architectures)
- Networked (share data and resources)

Relevance of Embedded Systems

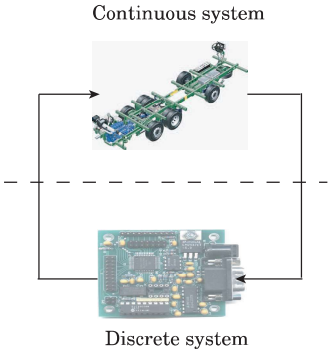
- Dominates computer market completely ($\approx 98\%$)
- Increasing interest in multi-disciplinary research fields

Examples:

- Transportation: modern car has several computers and networks
- Robotics: service robots, mobile manipulators
- Manufacturing and process industry
- Power generation and distribution

What is a Hybrid System?

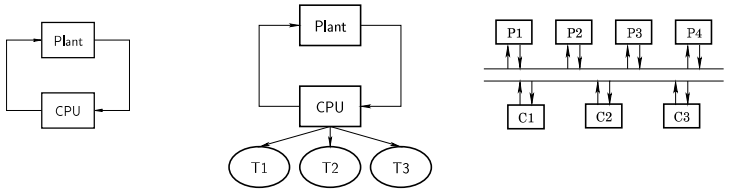
- A hybrid system is a mathematical model of an embedded system
- Dynamical system with interacting time-driven and event-driven dynamics



The Main Thread of the Course

Integrated design of computer-controlled systems

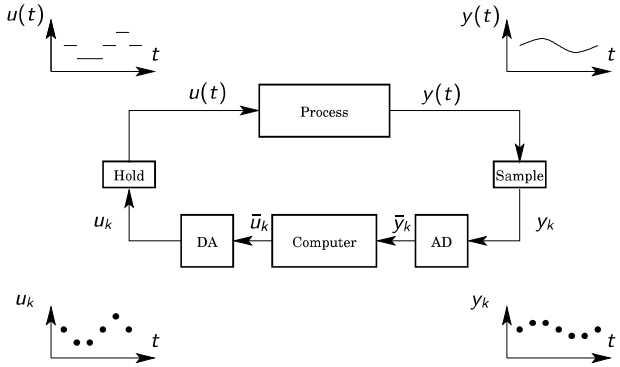
1. Single-task (time-triggered)
2. Multi-task (event-triggered)
3. Hybrid (networked)



Course Outline

- **Introduction:** course outline, motivating examples, review of sampled signals [L1]
- **Time-triggered control:** models, analysis, implementation [L2-L5]
- **Event-triggered control:** real-time operating systems, models of computations and software, scheduling [L6-L9]
- **Hybrid control:** modeling time-triggered and event-triggered systems, control and verification of hybrid systems [L10-L13]
- **Summary** [L14]

Computer-Controlled System



Signals in the loop have different characteristics

Signals

Continuous-valued and continuous-time signals:

- $u(t) \in \mathbb{R},\ t \in [0, \infty)$
- $y(t) \in \mathbb{R},\ t \in [0, \infty)$

Continuous-valued and discrete-time signals:

- $u_k \in \mathbb{R},\ k = 0, 1, \ldots$
- $y_k \in \mathbb{R},\ k = 0, 1, \ldots$

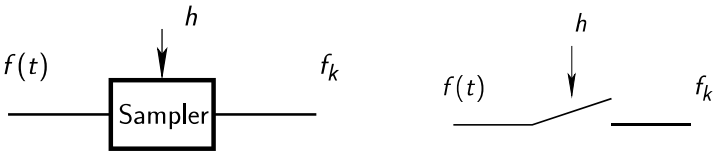
Discrete-valued and discrete-time signals:

- $\bar{u}_k \in \{\bar{u}^1, \ldots, \bar{u}^N\} \subset \mathbb{Q},\ k = 0, 1, \ldots$
- $\bar{y}_k \in \{\bar{y}^1, \ldots, \bar{y}^N\} \subset \mathbb{Q},\ k = 0, 1, \ldots$

Sampling

Uniform sampling with sampling period $h > 0$ generates

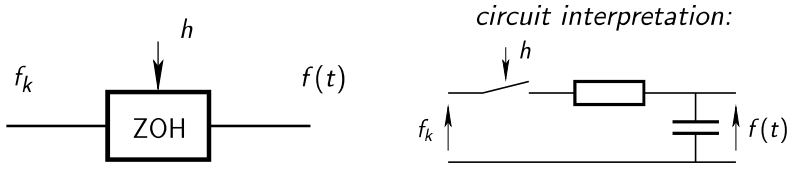
$$f_k = f(kh), \quad k = 0, 1, \ldots$$



Zero-Order Hold

ZOH *holds* the input constant over intervals h:

$$f(t) = f_k, \quad t \in [kh, kh + h)$$



"Zero": interpolation by polynomial of order zero.
Exist also first-order and higher-order holds

Fourier Transforms

$F(\omega)$ is Fourier transform of continuous-time signal $f(t)$:

$$F(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} f(t) dt, \qquad f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} F(\omega) d\omega$$

$F_s(\omega)$ is Fourier transform of discrete-time signal $f_k = f(kh)$:

$$F_s(\omega) = \sum_{k=-\infty}^{\infty} f_k e^{-i\omega k}, \qquad f_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\omega k} F_s(\omega) d\omega$$

Shannon's Sampling Theorem

A continuous-time signal $f(t)$ with Fourier transform

$$F(\omega) = 0, \quad \omega \notin (-\omega_0, \omega_0)$$

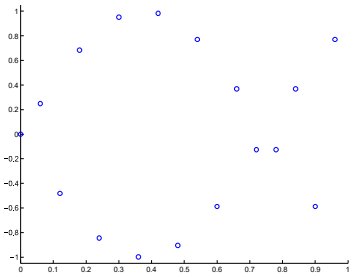
is uniquely reconstructed by its samples $f_k = f(kh)$,
if sampling frequency $\omega_s = 2\pi/h$ is larger than $2\omega_0$.
Then, $f(t)$ can be reconstructed from f_k as

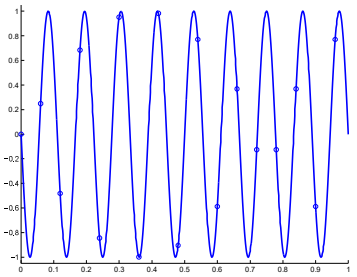
$$f(t) = \sum_{k=-\infty}^{\infty} f_k \operatorname{sinc} \frac{\omega_s(t - kh)}{2}$$

Note: Shannon reconstruction is not casual, ZOH practical alternative

Example: Sinusoid

Reconstruction possible if sampled at least twice per period:





Nyquist Frequency

$\omega_N = \omega_s/2$ is called the Nyquist frequency
Shannon's Sampling Theorem:
If $f(t)$ has no component above ω_N, then $f(t)$ is uniquely determined by its samples f_k
What happens when $\omega_s < 2\omega_0$? (equiv. $\omega_N < \omega_0$)

High frequencies of $f(t)$ are interpreted as low frequencies when not sampled sufficiently often

This is called aliasing or frequency folding

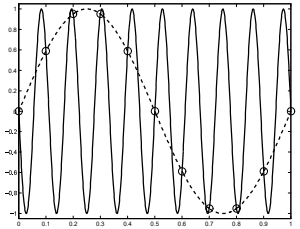
Aliasing

Components above ω_N in $f(t)$ cannot be distinguished in f_k

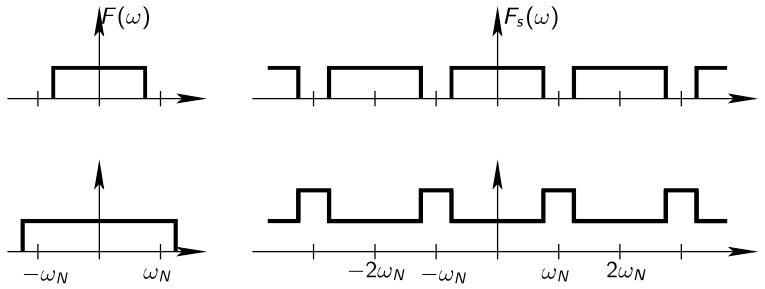
$$F_s(\omega) = \frac{1}{h} \sum_{k=-\infty}^{\infty} F(\omega + k\omega_s)$$

Example:

Frequencies $k\omega_s \pm \omega_1 > \omega_N$ of $f(t)$ are mapped into lower frequencies $0 \leq \omega_1 < \omega_N$ in the sampled signal.
$0 \leq \omega_1 < \omega_N$ is called *alias* of $k\omega_s \pm \omega_1$



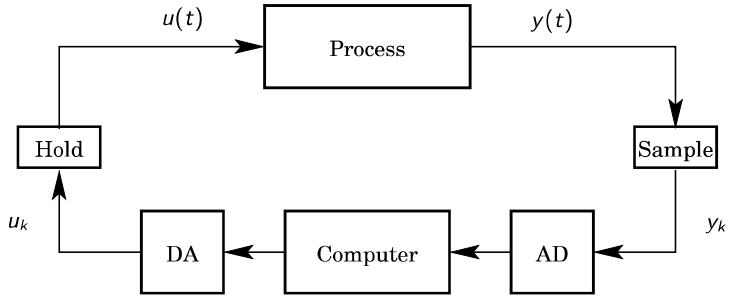
Aliasing: Frequency domain visualization



The original signal can be reconstructed from $F_s(\omega)$ in the first case with a low pass filter with appropriate cut-off frequency, since $F(\omega)$ does not contain frequencies over ω_N. This is in contrast to the second figure.

Discrete-Time Systems

We need tools to analyze discrete-time system $u_k \rightarrow y_k$:



Discrete-Time Models

Many results similar to continuous-time systems

- Input–output models: $y_{k+1} + ay_k = bu_k$
- State-space model: $x_{k+1} = ax_k + bu_k, \quad y_k = x_k$
- z-transform: $Y(z) = \sum_{k=0}^{\infty} y_k z^{-k}$
- (Pulse-)Transfer function $H(z)$ with $Y(z) = H(z)U(z)$

Please, revisit the Signals & Systems Course (or similar)

Next Lecture

Models of sampled systems

- Sampling of continuous-time systems
- State-space and input–output models
- Poles and zeros

EL2450 Hybrid and Embedded Control

Lecture 2: Models of sampled systems

- Sampling of continuous-time systems
- State-space and input–output models
- Poles and zeros

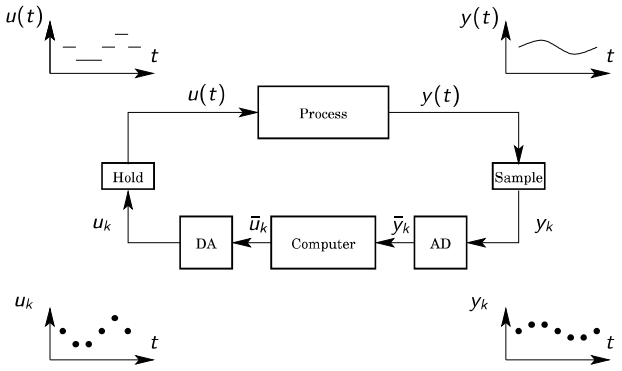
Today's Goal

You should be able to

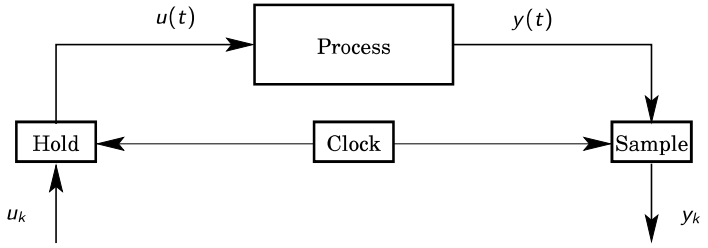
- Derive zero-order-hold sampling of systems (without and with delays in the loop)
- Relate SS and IO models
- Derive poles and zeros of discrete-time systems and relate them to their continuous-time counterparts

Computer-Controlled System

Consider the mapping from u_k to y_k:



Sampled Continuous-Time System



Process:

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

Sampling Continuous-Time System

For all $t \in [t_k, t_{k+1}]$

$$\begin{aligned} x(t) &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-\tau)}Bu(\tau)d\tau \\ &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-\tau)}d\tau Bu(t_k) \\ &= e^{A(t-t_k)}x(t_k) + \int_{0}^{t-t_k} e^{As}dsBu(t_k), \end{aligned}$$

$$\begin{aligned} x(t_{k+1}) &= \Phi(t_{k+1}, t_k)x(t_k) + \Gamma(t_{k+1}, t_k)u(t_k) \\ y(t_k) &= Cx(t_k) + Du(t_k) \end{aligned}$$

$$\Phi(t_{k+1}, t_k) = e^{A(t_{k+1}-t_k)}, \quad \Gamma(t_{k+1}, t_k) = \int_0^{t_{k+1}-t_k} e^{As}dsB$$

Note: Time-varying linear system

Uniform Sampling

Periodic sampling with $h > 0$ gives linear time-invariant system

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned}$$

where

$$\Phi = e^{Ah}, \quad \Gamma = \int_0^h e^{As}dsB$$

Example

Sampling the system

$$\dot{x} = ax + bu, \quad a \neq 0$$

gives

$$x(kh+h) = \Phi x(kh) + \Gamma u(kh)$$

with

$$\Phi = e^{ah}, \quad \Gamma = \int_0^h e^{as}dsb = \frac{b}{a}(e^{ah} - 1)$$

Discrete-Time Systems

Suppose now on (if not otherwise stated) that $h = 1$:

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

With initial condition $x(0)$, the solution is

$$\begin{aligned} x(k) &= \Phi x(k-1) + \Gamma u(k-1) \\ &= \Phi^2 x(k-2) + \Phi\Gamma u(k-2) + \Gamma u(k-1) = \dots \\ &= \Phi^k x(0) + \sum_{j=0}^{k-1} \Phi^{k-j-1}\Gamma u(j) \end{aligned}$$

Discrete-Time Systems

With initial condition $x(0)$, the solution is

$$x(k) = \Phi^k x(0) + \sum_{j=0}^{k-1} \Phi^{k-j-1} \Gamma u(j)$$

First part depends on $x(0)$ and the second one on the input sequence. Solution properties related to eigenvalues of Φ, given by its *characteristic equation*

$$\det(\lambda I - \Phi) = 0$$

Changing Coordinates

T nonsingular matrix and $z = Tx$. Then

$$z(k+1) = \tilde{\Phi} z(k) + \tilde{\Gamma} u(k)$$
$$y(k) = \tilde{C} z(k) + \tilde{D} u(k)$$

with $\tilde{\Phi} = T\Phi T^{-1}, \tilde{\Gamma} = T\Gamma, \tilde{C} = CT^{-1}, \tilde{D} = D$.
Key result:

$$\det(\lambda I - \Phi) = \det(\lambda I - \tilde{\Phi})$$

Coordinates can be chosen to give simple forms to system equations (revise canonical and special forms from basic control course).

Example: Diagonal Form

Φ has distinct eigenvalues $\lambda_1, \ldots, \lambda_n$. Then there exists a T such that

$$T\Phi T^{-1} = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}$$

Solution can be decoupled and has the form

$$z_i(k) = \lambda_i^k z_i(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} \beta_i u(j)$$

where $i = 1, \ldots, n$.

Example: Jordan Form

Φ has multiple eigenvalues, then generally not diagonalizable. There exists a T such that

$$T\Phi T^{-1} = \begin{bmatrix} L_{k_1}(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & L_{k_r}(\lambda_r) \end{bmatrix}$$

where $k_1 + \ldots + k_r = n$ and L_k is a $k \times k$ matrix with

$$L_k(\lambda) = \begin{bmatrix} \lambda & 1 & 0 & \ldots & 0 \\ 0 & \lambda & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \ldots & 0 & \lambda & 1 \\ 0 & \ldots & 0 & 0 & \lambda \end{bmatrix}$$

Pulse Response

For linear time-invariant systems, input-output relation has the general form

$$y(k) = \sum_{m=0}^{k} h(k-m)u(m) + y_p(k)$$

where y_p relates to initial conditions. For zero initial conditions, $h(k-m)$ gives the output at k of a unit pulse injected at m. It is called *pulse-response function* of the system.

Pulse Response

We can derive the pulse response from the state-space model

$$y(k) = C\Phi^k x(0) + \sum_{j=0}^{k-1} C\Phi^{k-j-1}\Gamma u(j) + Du(k)$$

Hence, the response to a unit pulse at $k = 0$ with $x(0) = 0$ is

$$h(k) = \begin{cases} 0, & k < 0 \\ D, & k = 0 \\ C\Phi^{k-1}\Gamma, & k > 0 \end{cases}$$

Pulse-Transfer Operator

Introduce forward- and backward- shift operator q: $qx(k) = x(k+1)$, $q^{-1}x(k) = x(k-1)$. Then,

$$qx(k) = \Phi x(k) + \Gamma u(k)$$

Hence,

$$y(k) = Cx(k) + Du(k) = \underbrace{\left[C(qI - \Phi)^{-1}\Gamma + D\right]}_{H(q)} u(k)$$

$H(q)$ is the *pulse-transfer operator*. $h(k), H(q)$ are also coordinate transformation invariant.

Pulse-Transfer Function

Similarly, with z-transform, $X(z) = \sum_{k=0}^{\infty} x(k)z^{-k}$, we obtain

$$z(X(z) - x(0)) = \Phi X(z) + \Gamma U(z)$$

so

$$\begin{aligned} Y(z) &= CX(z) + DU(z) \\ &= \underbrace{\left[C(zI - \Phi)^{-1}\Gamma + D\right]}_{H(z)} U(z) + C(zI - \Phi)^{-1}zx(0) \end{aligned}$$

$H(z) = \mathcal{Z}\{h(k)\}$ and is called the *pulse-transfer function*.

Note: $H(z)$ is equal to $H(q)$ with q replaced by z. Still they are two different mathematical objects. (Why?)

State-Space and Input–Output Models

Consider a state-space system of order n:

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$
$$y(k) = Cx(k) + Du(k)$$

Then, $y(k) = H(q)u(k) = \dfrac{B(q)}{A(q)}u(k)$, where

$$A(q) = q^n + a_1 q^{n-1} + \cdots + a_n, \quad B(q) = b_0 q^n + b_1 q^{n-1} + \cdots + b_n$$

or equivalently

$$y(k+n) + a_1 y(k+n-1) + \cdots + a_n y(k)$$
$$= b_0 u(k+n) + b_1 u(k+n-1) + \cdots + b_n u(k)$$

Sampled Double Integrator

What is $H(q)$ for a ZOH sampled $G(s) = 1/s^2$ with $h = 1$?

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

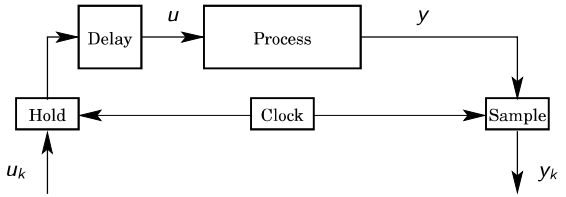
$$\Phi = e^{Ah} = I + A + A^2/2 + \cdots = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$
$$\Gamma = \int_0^h \begin{bmatrix} s \\ 1 \end{bmatrix} ds = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$$

$$H(q) = C(qI - \Phi)^{-1}\Gamma = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} q-1 & -1 \\ 0 & q-1 \end{bmatrix}^{-1} \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$$
$$= \frac{q+1}{2(q-1)^2}$$

Note: Table 2 on page 22 in [WAA] gives $H(q) \overset{\text{ZOH samp}}{\longleftrightarrow} G(s)$

Sampling System with Time-Delay

Delayed actuation with fixed delay $\tau \in (0, h)$



Then, $\dot{x}(t) = Ax(t) + Bu(t-\tau)$. $u(t)$ changes at $kh + \tau$, so

$$x(kh+h) = e^{Ah}x(kh) + \int_{kh}^{kh+h} e^{A(kh+h-s)}Bu(s-\tau)ds$$
$$= e^{Ah}x(kh) + \int_{kh}^{kh+\tau} e^{A(kh+h-s)}ds\, Bu(kh-h) + \int_{kh+\tau}^{kh+h} e^{A(kh+h-s)}ds\, Bu(kh)$$

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh-h)$$
$$\Phi = e^{Ah}, \quad \Gamma_0 = \int_0^{h-\tau} e^{As} ds B$$
$$\Gamma_1 = e^{A(h-\tau)} \int_0^{\tau} e^{As} ds B$$

Hence,

$$\begin{bmatrix} x(kh+h) \\ u(kh) \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x(kh) \\ u(kh-h) \end{bmatrix} + \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix} u(kh)$$

Note: Still LTI system, but of one order higher

Sampling System with Larger Time-Delay

What if the delay τ is larger than h?
Then, the previous approach still works, but needs a modification:
Example Suppose $\tau \in (h, 2h)$. Then,

$$\begin{bmatrix} x(kh+h) \\ u(kh-h) \\ u(kh) \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_1 & \Gamma_0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(kh) \\ u(kh-2h) \\ u(kh-h) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} u(kh)$$

for suitable Γ_1, Γ_0

Frequency Response

Recall that the frequency response of a continuous-time system $G(s)$ is given by $G(i\omega)$ for $\omega \in [0, \infty)$.
The frequency response of a sampled-data system $H(z)$ is given by $H(e^{i\omega h})$ for $\omega h \in [0, \pi]$.
Note that for $H(z)$ it suffices to study ω up to ω_N.

Poles and Zeros

- Poles are the zeros of $A(z)$
- Zeros are the zeros of $B(z)$

Equivalently

- Poles are the eigenvalues of Φ
- Zeros are $z \in \mathbf{C}$ such that $\det \begin{bmatrix} zI - \Phi & -\Gamma \\ C & D \end{bmatrix} = 0$

Physical interpretation:
Pole in p gives *mode* p^k in time response
Zero in a gives *blocking* of inputs a^k

Example: Blocking Zeros

$y(k+1) - ay(k) = u(k+1) - bu(k)$, $0 < a < 1 < b$, corresponds to

$$Y(z) = H(z)U(z) = \frac{z-b}{z-a}U(z)$$

$u(k) = b^k$ gives

$$Y(z) = \frac{z}{z-a} \overset{\mathcal{Z}^{-1}}{\to} y(k) = a^k$$

Hence, $y(k) \to 0$ and does not depend on $u(k)$.
The zero of $H(z)$ in b thus *blocks* the input $u(k) = b^k$.

Blocking Zeros

Consider

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$
$$y(k) = Cx(k) + Du(k)$$

A zero in a means *blocking* of inputs a^k. Then for the system above, the input a^k gives zero output for $z = a$ such that $\det \begin{bmatrix} zI - \Phi & -\Gamma \\ C & D \end{bmatrix} = 0$.

Pole Locations of Sampled System

Suppose

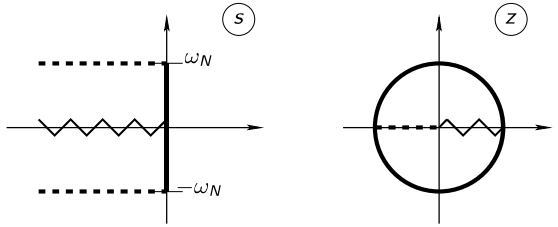
$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

has poles $\lambda_i(A)$, $i = 1, \ldots, n$. Then,

$$x(kh+h) = \Phi x(kh) + \Gamma u(kh)$$
$$y(kh) = Cx(kh) + Du(kh)$$

has poles $\exp(\lambda_i(A)h)$. Follows from $\Phi = \exp(Ah)$.

Pole Mapping Through $z = \exp(sh)$



The mapping $s \mapsto \exp(sh)$ is not invertible. Several points of the s-plane are mapped into the z-plane.

The pole $z = 0$ has no counterpart in s-plane.

The left half plane of the s-domain is mapped into the unit disc of the z-plane. What might this imply?

Zero Locations of Sampled System

Zeros in s-domain is not easily mapped to z-domain.

Example:

$G(s) = \dfrac{1}{s^2}$ is sampled into $H(q) = \dfrac{q+1}{2(q-1)^2}$.

$G(s)$ has no zero, but $H(q)$ has zero in $z = -1$.

The sampling procedure gives more zeros.

Next Lecture

Analysis of sampled control

- Stability
- Reachability and observability
- Observers
- State and output feedback

EL2450 Hybrid and Embedded Control

Lecture 3: Analysis of sampled control

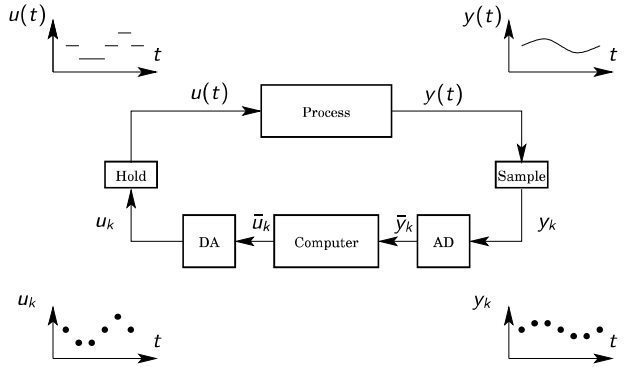
- Stability
- Reachability and observability
- Observers
- State and output feedback

Today's Goal

You should be able to

- Determine stability, reachability and observability for discrete-time systems
- Design state observers
- Design sampled-data control based on state and output feedback
- Design deadbeat controllers

Sampled Control System



To understand closed-loop behavior, we need tools to study properties of discrete-time control systems

Stability

The solution $x^*(k)$ of $x(k+1) = f(x(k))$ is *stable* if for all $\epsilon > 0$, there exists $\delta = \delta(\epsilon) > 0$ such that for all solutions $x(k)$

$$\|x(0) - x^*(0)\| < \delta \quad \Rightarrow \quad \|x(k) - x^*(k)\| < \epsilon, \quad k = 0, 1, \ldots$$

The solution is *asymptotically stable* if it is stable and

$$\|x(0) - x^*(0)\| < \delta \quad \Rightarrow \quad \|x(k) - x^*(k)\| \to 0, \text{ as } k \to \infty$$

Linear Systems A linear system $x(k+1) = \Phi x(k)$ is asymptotically stable if and only if $|\lambda_i(\Phi)| < 1$, $i = 1, \ldots, n$.

Stability-LTI Systems

For $x(k+1) = \Phi x(k)$, $x^*(k+1) = \Phi x^*(k)$ and $\tilde{x}(k) = x(k) - x^*(k)$, then $\tilde{x}(k+1) = \Phi\tilde{x}(k)$, $\tilde{x}(0) = x(0) - x^*(0)$.

If $x^*(k)$ is stable, all other solutions of $x(k+1) = \Phi x(k)$ stable. For LTI systems, stability is a property of the system and not of a particular solution.

Solution ($x(k) = \Phi^k x(0)$) can be transformed to linear combination of terms $p_i(k)\lambda_i^k(\Phi)$, where $p_i(k)$ polynomials in k of order one less than the multiplicity of $\lambda_i(\Phi)$. Thus asymptotic stability equivalent to $|\lambda_i(\Phi)| < 1$, $i = 1, \ldots, n$.

Lyapunov Function

A continuous function $V : \mathbb{R}^n \to \mathbb{R}$ is a *Lyapunov function* for $x(k+1) = f(x(k)), f(0) = 0$ if

1. $V(0) = 0$ and $V(x) > 0$, $\forall x \neq 0$
2. $\Delta V(x) := V(f(x)) - V(x) \leq 0$, $\forall x \neq 0$

Linear Systems $V(x) = x^T P x$, P positive definite, is a Lyapunov function for $x(k+1) = \Phi x(k)$, if (and only if)

$$\Phi^T P \Phi - P = -Q, \qquad Q \text{ positive semidefinite}$$

because
$\Delta V(x) = V(\Phi x) - V(x) = x^T(\Phi^T P \Phi - P)x = -x^T Q x \leq 0.$

Stability Test

The solution $x^*(k) = 0$ for $x(k+1) = f(x(k))$ is stable if there exists a Lyapunov function. It is asymptotically stable if, moreover, $\Delta V(x)$ is negative definite.

Linear Systems A linear system $x(k+1) = \Phi x(k)$ is asymptotically stable if (and only if) for any positive definite Q, there exists positive definite P such that

$$\Phi^T P \Phi - P = -Q$$

Reachability

$$x(k+1) = f(x(k), u(k))$$

is *reachable* if for any x_0, x_1 there exists a finite integer $N > 0$ and a control sequence $u(k)$, $k = 0, 1, \ldots, N-1$, such that $x(0) = x_0$ and $x(N) = x_1$.

Linear Systems

For $x(k+1) = \Phi x(k) + \Gamma u(k)$, we have

$$\begin{aligned} x(n) &= \Phi^n x(0) + \Phi^{n-1}\Gamma u(0) + \cdots + \Gamma u(n-1) \\ &= \Phi^n x(0) + \underbrace{\begin{bmatrix}\Gamma & \Phi\Gamma & \dots & \Phi^{n-1}\Gamma\end{bmatrix}}_{W_c} \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix} \end{aligned}$$

Hence, the system is reachable if and only if W_c is invertible.

Example

$$x(k+1) = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} x(k) + \begin{bmatrix}0\\ 1\end{bmatrix} u(k) \quad \Rightarrow \quad W_c = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}$$

Thus the system is not reachable.
Note that still, for any $x(0)$ there exist $u(0), u(1)$ such that $x(2) = 0$.

Observability

$$\begin{aligned} x(k+1) &= f(x(k)) \\ y(k) &= h(x(k)) \end{aligned}$$

is *observable* if there exists $N < \infty$ such that $x(0) = x_0$ can be determined from $y(0), \dots, y(N)$.

Linear Systems

$$\begin{aligned} x(k+1) &= \Phi x(k) \\ y(k) &= Cx(k) \end{aligned}$$

gives $y(0) = Cx(0)$, $y(1) = C\Phi x(0)$, $\dots$:

$$W_o x(0) := \begin{bmatrix} C \\ C\Phi \\ \vdots \\ C\Phi^{n-1} \end{bmatrix} x(0) = \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(n-1) \end{bmatrix}$$

Hence, the system is observable if and only if W_o is invertible.

Example

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x(k)$$
$$y(k) = \begin{bmatrix} 1 & 1 \end{bmatrix} x(k)$$

is not observable since

$$\text{rank } W_o = \text{rank} \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = 1 < 2$$

Note that both $x(0) = (0, 1)$ and $(1, 0)$ give $y(0) = 1$, $y(1) = 2$.

Observers

How obtain estimates of $x(k)$ from current and past y and u?
Direct calculations of $x(k)$ based on

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$
$$y(k) = Cx(k)$$

give

$$\underbrace{\begin{bmatrix} y(k-n+1) \\ y(k-n+2) \\ \vdots \\ y(k) \end{bmatrix}}_{Y_k} = W_o x(k-n+1) + \begin{bmatrix} 0 & 0 & \dots & 0 \\ C\Gamma & 0 & \dots & 0 \\ \vdots & \ddots & & \vdots \\ C\Phi^{n-2}\Gamma & C\Phi^{n-3}\Gamma & \dots & C\Gamma \end{bmatrix} \underbrace{\begin{bmatrix} u(k-n+1) \\ u(k-n+2) \\ \vdots \\ u(k-1) \end{bmatrix}}_{U_{k-1}}$$

Static Observer

This leads to the formula (multiply by W_o^{-1} and express $x(k)$ as function of $x(k-n+1)$)

$$x(k) = QY_k + RU_{k-1}$$

where the matrices Q, R depend only on Φ, Γ, C.

Note

- $x(k)$ is simply a linear combination of old y and u
- Method can be sensitive to noise and disturbances

Example

For ZOH-sampled $G(s) = 1/s^2$ with $h = 1$, we have

$$\Phi = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\begin{aligned} y(k) = x_1(k) &= x_1(k-1) + x_2(k-1) + u(k-1)/2 \\ &= y(k-1) + [x_2(k) - u(k-1)] + u(k-1)/2 \end{aligned}$$

$$\begin{aligned} x_1(k) &= y(k) \\ x_2(k) &= y(k) - y(k-1) + u(k-1)/2 \end{aligned}$$

- Disturbance in y hits x directly

Dynamic Observer

Denote by $\hat{x}(k+1|k)$ an estimate of $x(k+1)$ based on measurements $\ldots, y(k-1), y(k)$ up to time k.
A dynamic observer is given by

$$\hat{x}(k+1|k) = \Phi\hat{x}(k|k-1) + \Gamma u(k) + K[y(k) - C\hat{x}(k|k-1)]$$

The reconstruction error $\tilde{x}(k+1|k) = x(k+1) - \hat{x}(k+1|k) \to 0$ as $k \to \infty$ if K is chosen such that $|\lambda_i(\Phi - KC)| < 1$, because

$$\tilde{x}(k+1|k) = (\Phi - KC)\tilde{x}(k|k-1)$$

Example

Consider $G(s) = 1/s^2$,$h = 1$ again. Then, $\Phi - KC = \begin{bmatrix} 1-k_1 & 1 \\ -k_2 & 1 \end{bmatrix}$ with characteristic equation $z^2 + (k_1 - 2)z + 1 - k_1 + k_2 = 0$.
$k_1 = 1$, $k_2 = 1/4$ give zeros in $1/2$. Then,

$$\begin{bmatrix} \hat{x}_1(k+1|k) \\ \hat{x}_2(k+1|k) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_1(k|k-1) \\ \hat{x}_2(k|k-1) \end{bmatrix} + \begin{bmatrix} 1/2 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 1 \\ 1/4 \end{bmatrix} [y(k) - \hat{x}_1(k|k-1)]$$

Reduced-Order Observer

The dynamic observer has a unit delay from y to $\hat{x}$ (Check!)
This can be avoided by considering

$$\hat{x}(k+1|k+1) = \Phi\hat{x}(k|k) + \Gamma u(k) + K[y(k+1) - C(\Phi\hat{x}(k|k) + \Gamma u(k))]$$

where $y(k+1)$ is the current measurement.
The reconstruction error $\tilde{x} = x - \hat{x}$ fulfills

$$\tilde{x}(k+1|k+1) = (I - KC)\Phi\tilde{x}(k|k)$$

so K should be chosen such that $|\lambda_i[(I - KC)\Phi]| < 1$.

$$y(k+1) - C\hat{x}(k+1|k+1) = C\tilde{x}(k+1|k+1) = (I - CK)C\Phi\tilde{x}(k|k)$$

If we have p (independent) outputs (rank $C = p$), then K ($n \times p$ matrix) can be chosen such that $I - CK = 0$, where CK is a $p \times p$ matrix. Then,

$$C\hat{x}(k+1|k+1) = y(k+1)$$

hence, y is estimated with no error.
The exact knowledge of y can be used to reduce the order of the observer by p (see example). The resulting (reduced-order) filter is called a *Luenberger observer*.

Example

For the double integrator we get

$$\begin{bmatrix}\hat{x}_1(k+1|k+1)\\ \hat{x}_2(k+1|k+1)\end{bmatrix} = \begin{bmatrix}1-k_1 & 1-k_1\\ -k_2 & 1-k_2\end{bmatrix}\begin{bmatrix}\hat{x}_1(k|k)\\ \hat{x}_2(k|k)\end{bmatrix} + \begin{bmatrix}(1-k_1)/2\\ 1-k_2/2\end{bmatrix}u(k) + \begin{bmatrix}k_1\\ k_2\end{bmatrix}y(k+1)$$

$CK = I$ implies $k_1 = 1$. Then, the reduced-order observer becomes ($\hat{x}_1(k+1|k+1) = y(k+1)$ and)

$$\hat{x}_2(k+1|k+1) = (1-k_2)\hat{x}_2(k|k) + (1-k_2/2)u(k) + k_2(y(k+1) - y(k))$$

$k_2 = 1/2$ gives eigenvalue in $1/2$.

Feedback Control

Recall the main reasons for using feedback control:

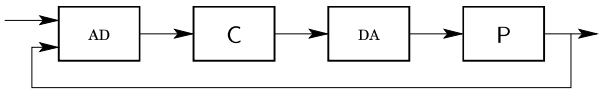
- Counteract process uncertainties and variations
- Track reference trajectories
- Attenuate disturbances
- Reduce influence of measurement noise

Disadvantages with feedback control include higher system design complexity (sensor, actuator, computer)

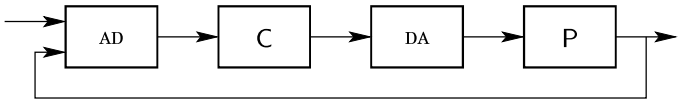
State Feedback and Output Feedback

$$x(k+1) = f(x(k), u(k))$$

Output feedback: $u(k)$ is based on the output $y(k) = h(x(k))$
State feedback: $u(k)$ is based on the state $x(k)$
Control objectives: Stabilization to equilibrium or tracking of certain $r(k)$



Example: Control of Double Integrator



Sampling

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

with $h = 1$ gives

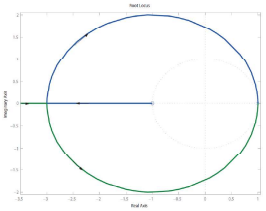
$$H(q) = \frac{q+1}{2(q-1)^2}$$

P Control

Consider proportional control based on output y:
$u(k) = K(r(k) - y(k))$, $K > 0$. Gives characteristic equation

$$z^2 + (K/2 - 2)z + 1 + K/2 = 0$$

which has zeros outside unit circle for all $K > 0$:



Hence, system not stabilizable with P control

PD Control

Consider proportional-derivative control:

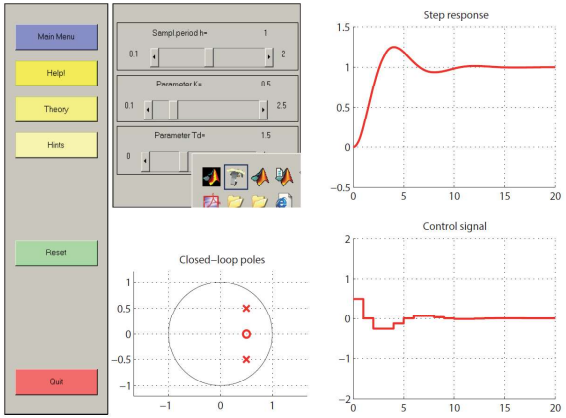
$$u(k) = K[r(k) - y(k) + 2T_d \frac{q-1}{q+1}(r(k) - y(k))]$$

Then,

$$y(k) = \frac{K/2[(1+2T_d)q + 1 - 2T_d]}{q^2 + (T_d K + K/2 - 2)q + 1 - T_d K + K/2} r(k)$$

where K, T_d can be chosen to give a stable closed-loop poles

$K = 0.5$, $T_d = 1.5$ give poles $0.5 \pm 0.5i$ and zero 0.5



From CCSDEMO

State Feedback

Consider the process $x(k+1) = \Phi x(k) + \Gamma u(k)$ under state feedback $u(k) = -Lx(k)$. Closed-loop dynamics is given by

$$x(k+1) = (\Phi - \Gamma L)x(k)$$

so L should be chosen such that $\lambda_i(\Phi - \Gamma L)$ are suitably placed inside unit circle.

Example For the double integrator, closed-loop characteristic polynomial equals $p(z) = z^2 + p_1 z + p_2$ if

$$L = \begin{bmatrix} \frac{1 + p_1 + p_2}{h^2} & \frac{3 + p_1 - p_2}{2h} \end{bmatrix}$$

Note that u becomes large if h is small.

Pole placement

Previous example indicates a design procedure for pole placement. Can be extended to higher order but calculations become tedius.

- More systematic methods exist.
- Ackermann's formula:

$$L = (0 \dots 0 \quad 1) W_c^{-1} p(\Phi)$$

  where $p(z)$ is the desired closed-loop characteristic polynomial.
- Use the formula to calculate gains in the previous example.

Deadbeat Control

If the desired closed-loop poles are put at the origin, then $p(z) = z^n$.
From Cayley-Hamilton it follows that $\Phi_c := \Phi - \Gamma L$ fulfills $\Phi_c^n = 0$.
Hence, for any $x(0)$

$$x(n) = \Phi_c^n x(0) = 0$$

This control strategy is called *deadbeat control*.
It drives the state into the origin in at most n steps.

Note There is no correspondence to deadbeat control in continuous time. What happens in continuous time if we try to drive the state to the origin in zero time? Can we drive the state in time nh?

Example

For the double integrator, closed-loop characteristic polynomial equals $p(z) = z^2$ if

$$L = \begin{bmatrix} \frac{1}{h^2} & \frac{3}{2h} \end{bmatrix}$$

Then,

$$u(0) = -\frac{1}{h^2} x_1(0) - \frac{3}{2h} x_2(0)$$
$$u(h) = -\frac{1}{h^2} x_1(h) - \frac{3}{2h} x_2(h)$$

This control strategy drives the state to the origin in two steps.

Output Feedback

If the state x is not measured, a dynamic observer

$$\hat{x}(k+1|k) = \Phi \hat{x}(k|k-1) + \Gamma u(k) + K[y(k) - C\hat{x}(k|k-1)]$$

can be used together with the control law

$$u(k) = -L\hat{x}(k|k-1)$$

With $\tilde{x} = x - \hat{x}$, the closed-loop system is

$$\begin{bmatrix} x(k+1) \\ \tilde{x}(k+1|k) \end{bmatrix} = \begin{bmatrix} \Phi - \Gamma L & \Gamma L \\ 0 & \Phi - KC \end{bmatrix} \begin{bmatrix} x(k) \\ \tilde{x}(k|k-1) \end{bmatrix}$$

Dynamics of order $2n$ determined by $\Phi - \Gamma L$ and $\Phi - KC$.

Duality

$$x(k+1) = \Phi x(k) + \Gamma u(k),\ \ y(k) = Cx(k)$$

with W_c, W_o. Then $\exists L$ s.t. $\Phi - \Gamma L$ has prescribed eigenvalues if (and only if) W_c is invertible.

Existence of K s.t. $\Phi - KC$ has prescribed eigenvalues equivalent to existence of K^T s.t. $\Phi^T - C^T K^T$ has prescribed eigenvalues. This happens if (and only if)

$$\underbrace{\begin{bmatrix} C^T & \Phi^T C^T & \dots & (\Phi^{n-1})^T C^T \end{bmatrix}}_{W_o^T}$$

is invertible.

Pulse-Transfer Function of Controller

The nth-order controller is given by

$$\begin{aligned} \hat{x}(k+1|k) &= (\Phi - KC - \Gamma L)\hat{x}(k|k-1) + Ky(k) \\ u(k) &= -L\hat{x}(k|k-1) \end{aligned}$$

which gives the input–output relation

$$u(k) = -L(qI - \Phi + KC + \Gamma L)^{-1} K y(k)$$

Internal model principle: note that the controller contains a model of the process (Φ, Γ, C) together with $2n$ tuning parameters in K and L.

Servo Problem

We can add a reference command r to the previous formulation:

$$u(k) = -L\hat{x}(k) + \ell_r r(k)$$

Then, the closed-loop system is

$$\begin{bmatrix} x(k+1) \\ \tilde{x}(k+1) \end{bmatrix} = \begin{bmatrix} \Phi - \Gamma L & \Gamma L \\ 0 & \Phi - KC \end{bmatrix} \begin{bmatrix} x(k) \\ \tilde{x}(k) \end{bmatrix} + \begin{bmatrix} \Gamma \ell_r \\ 0 \end{bmatrix} r(k)$$

$$y(k) = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \tilde{x}(k) \end{bmatrix}$$

On input–output form:

$$\begin{aligned} y(k) &= \begin{bmatrix} C & 0 \end{bmatrix} \left(qI - \begin{bmatrix} \Phi - \Gamma L & \Gamma L \\ 0 & \Phi - KC \end{bmatrix} \right)^{-1} \begin{bmatrix} \Gamma \ell_r \\ 0 \end{bmatrix} r(k) \\ &= C(qI - \Phi + \Gamma L)^{-1} \Gamma \ell_r r(k) \end{aligned}$$

Note

- Observer does not influence the input–output relation

Polynomial Representation of Servo Problem

We can do the previous calculations in polynomial form:

$$y(k) = \frac{B(q)}{A(q)}u(k), \qquad u(k) = \frac{S(q)}{R(q)}(\ell_r r(k) - y(k))$$

gives

$$(A(q)R(q) + B(q)S(q))y(k) = B(q)S(q)\ell_r r(k)$$

If the desired closed-loop system with respect to $r(t)$ is $y(k) = [B_m(q)/A_m(q)]r(k)$:

$$\frac{B(q)S(q)\ell_r}{A(q)R(q) + B(q)S(q)} = \frac{B_m(q)}{A_m(q)}$$

which gives the control parameters R, S, ℓ_r

Next Lecture

Computer realization of controllers

- Approximation of continuous-time designs
- Digital PID structures
- Choice of sampling time

EL2450 Hybrid and Embedded Control

Lecture 4: Computer realization of controllers

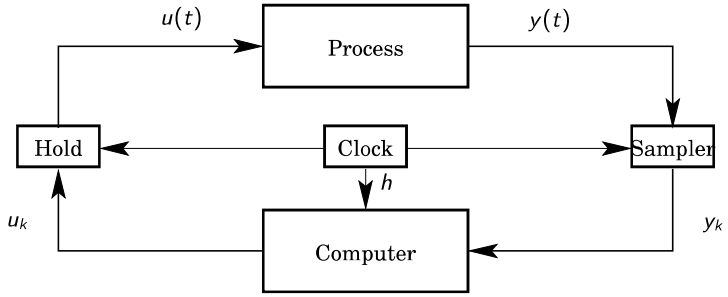
- Approximation of continuous-time designs
- Digital PID structures
- Realization of controllers
- Quantization
- Choice of sampling time

Today's Goal

You should be able to

- Transform continuous-time controller into discrete time
- Identify and select appropriate controller realizations
- Model and analyze quantization in computations and AD/DA converters
- Decide on reasonable sampling time

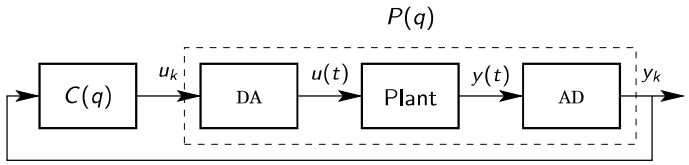
Time-Triggered Control System



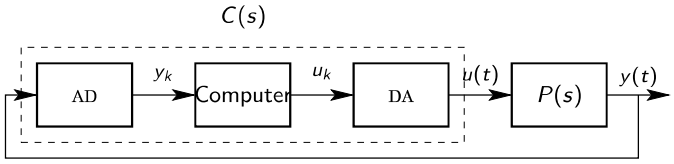
- **Q:** How apply continuous-time design methods to sampled control system?
- **A:** Make cuts at $u(t)$ and $y(t)$ (instead of at u_k and y_k)

Designs in Discrete or Continuous Time

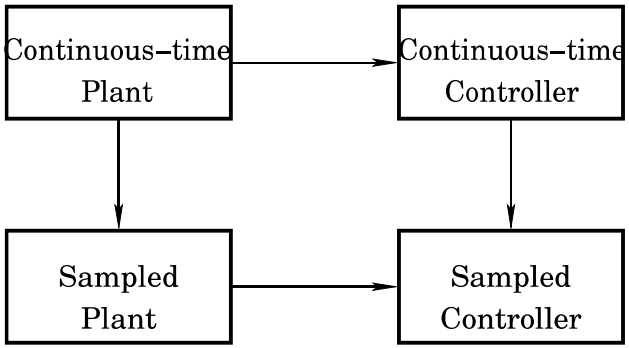
Sample plant, then derive a discrete-time controller:



Derive continuous-time controller, then transform to discrete-time algorithm:

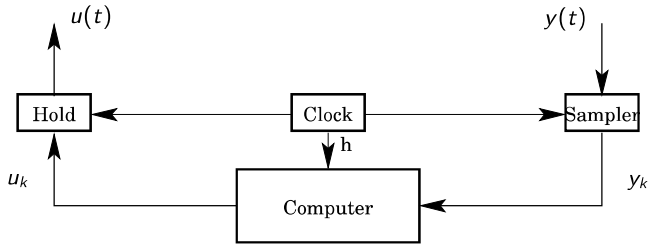


Designs in Discrete or Continuous Time



Approximating $C(s)$ by a Computer

Transform continuous-time design into discrete time:
Find $C(z)$ that approximates $C(s)$



Approximation of Continuous-Time Derivatives

Forward difference:

$$\frac{dx(t)}{dt} \approx \frac{x(t+h)-x(t)}{h} = \frac{q-1}{h}x(t)$$

Backward difference:

$$\frac{dx(t)}{dt} \approx \frac{x(t)-x(t-h)}{h} = \frac{q-1}{qh}x(t)$$

Tustin's approximation:

$$\frac{dx(t)}{dt} \approx \frac{2}{h}\cdot\frac{q-1}{q+1}x(t)$$

From $G(s)$ to $H(z)$

Pulse-transfer function $H(z)$ corresponding to transfer function $G(s)$ is

$$H(z) = G(s')$$

where

$$s' = \frac{z-1}{h} \qquad \text{(Forward difference)}$$

$$s' = \frac{z-1}{zh} \qquad \text{(Backward difference)}$$

$$s' = \frac{2}{h}\cdot\frac{z-1}{z+1} \qquad \text{(Tustin's approximation)}$$

Example

Discrete-time approximation of

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

gives pulse-transfer functions

$$H_1(z) = \frac{h}{z-1+h} \qquad \text{(Forward difference)}$$

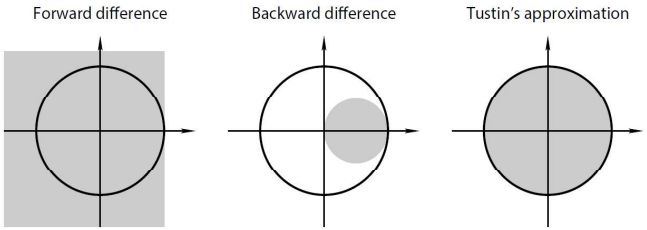
$$H_2(z) = \frac{h(1+h)^{-1}z}{z-(1+h)^{-1}} \qquad \text{(Backward difference)}$$

$$H_3(z) = \frac{h(2+h)^{-1}(z+1)}{z+(h-2)(h+2)^{-1}} \qquad \text{(Tustin's approximation)}$$

Note that H_1 is unstable for large h, but H_2 and H_3 are always stable.

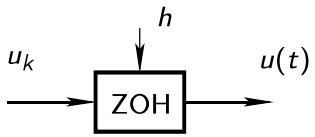
Stability Regions for $H(z)$

The stability region $\{s \in \mathbb{C} : \operatorname{Re} s < 0\}$ of continuous-time $G(s)$ is mapped to the regions below for discrete-time $H(z) = G(s')$:



- Forward difference may generate unstable $H(z)$ even if $G(s)$ is stable
- Backward difference may yield stable $H(z)$ even for unstable $G(s)$
- Tustin maps stable to stable

Transfer Function of a ZOH Circuit



Impulse response of $1/s$ is a step. Impulse response of e^{-s}/s is a delayed step. Hence, a ZOH circuit can be represented as

$$H(s) = \frac{1-e^{-sh}}{s}$$

since pulses can be represented as a series of impulse responses.

- For small $h > 0$: $H(s) \approx \dfrac{1-1+sh-s^2h^2/2+\ldots}{s} = h - \dfrac{sh^2}{2} + \ldots$
- Steady-state gain $H(0) = h$

Transfer Function of a Sampler

Recall the relationship between the Fourier transforms of the continuous-time and sampled signals:

$$F_s(\omega) = \frac{1}{h}\sum_{k=-\infty}^{\infty} F(\omega + k\omega_s)$$

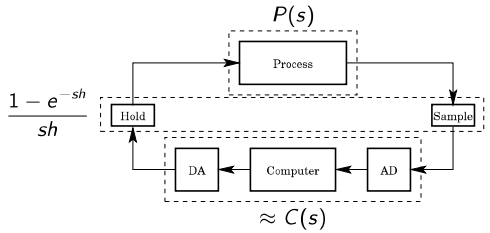
So there is a gain of $1/h$ between the frequency responses.

ZOH together with sampler hence contribute to the loop with

$$\frac{1-e^{-sh}}{sh}$$

Continuous-Time Design of Sampled Controller

1. Design $C(s)$ based on $P(s)$ (or on $P(s)\dfrac{1-e^{-sh}}{sh}$ if h not small)
2. Derive discrete-time approximation of $C(s)$
3. Implement discrete-time algorithm



Continuous-time PID Controller

Continuous-time PID controller with $e = r - y$:

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int^t e(t) + T_d\frac{de(t)}{dt}\right]$$

Implementation of pure derivative not desirable since it yields large amplification of measurement noise. Approximation:

$$sT_d \approx \frac{sT_d}{1 + sT_d/N}$$

Practical continuous-time PID

Continuous-time PID controller:

$$\begin{aligned} U(s) &= K\left[bR(s) - Y(s) + \frac{1}{sT_i}\Big(R(s) - Y(s)\Big) - \frac{sT_d}{1 + sT_d/N}Y(s)\right] \\ &= P(s) + I(s) + D(s) \end{aligned}$$

Note the parameter $b \in [0, 1]$ and that r is not differentiated in the D-part. Using $p = \frac{d}{dt}$:

$$\begin{aligned} u(t) &= K\left[br(t) - y(t) + \frac{1}{T_i p}\Big(r(t) - y(t)\Big) - \frac{T_d p}{1 + T_d p/N}y(t)\right] \\ &= P(t) + I(t) + D(t) \end{aligned}$$

Discretization of P-part

$$P(kh) = K\big[br(kh) - y(kh)\big]$$

Discretization of I-part

$$I(t) = \frac{K}{T_i}\int^t e(s)ds = \frac{K}{T_i p}e(t)$$

Forward approximation gives

$$I(kh + h) = I(kh) + \frac{Kh}{T_i}e(kh)$$

Discretization of D-part

$$\frac{T_d}{N} \cdot \frac{dD(t)}{dt} + D(t) = -KT_d \frac{dy(t)}{dt}$$

Backward approximation gives

$$\frac{T_d}{Nh}[D(kh) - D(kh-h)] + D(kh) = -\frac{KT_d}{h}[y(kh) - y(kh-h)]$$

so

$$D(kh) = \frac{T_d}{T_d + Nh} D(kh-h) - \frac{KT_d N}{T_d + Nh}[y(kh) - y(kh-h)]$$

Discretized PID Control

With P, I, D as given above, then

$$u(kh) = P(kh) + I(kh) + D(kh)$$

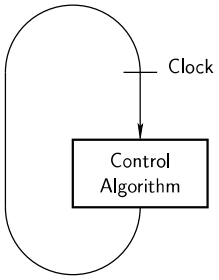
Note

- The given approximations (backward + forward) enable independent calculations of P, I, D
- Other approximations are possible (e.g., Tustin)
- Sometimes implementation on incremental form with output

$$\Delta u(kh) = u(kh) - u(kh-h)$$

Computer Code for Sampled-Data Controller

```
nexttime = getCurrentTime();
while (true) {
  AD_conversion();
  calculateOutput();
  DA_conversion();
  updateState();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```



- `calculateOutput` between `AD_conversion` and `DA_conversion` gives minimum computational time delay in control loop
- Controller states (such as observer state) are updated separately
- `sleepUntil` supposed to be supported by operating system

CalculateOutput and UpdateState

For a second-order controller on state space form

$$z(kh+h) = Az(kh) + By(kh)$$
$$u(kh) = Cz(kh) + Dy(kh)$$

`calculateOutput()` is essentially

```
u = c1*z1 + c2*z2 + d*y;
```

and `updateState()` is essentially

```
z1 = a11*z1 + a12*z2 + b1*y;
z2 = a21*z1 + a22*z2 + b2*y;
```

Java Code for PID Controller

```
public static void main(String[] args) {
  double uc, y, u;
  PID pid = new PID();        // Construct PID controller object
  long time = System.getCurrentTimeMillis(); // Get current time
  while (true) {
    y = readY();
    r = readR();
    u = pid.calculateOutput(r,y);
    writeU(u);
    pid.updateStates;
    time = time + par.pid.h*1000;
    Thread.waitUntil(time);                  // Wait until 'time'
  }
}
```



```
public double calculateOutput(double r, double y) {
  signals.r = r;
  signals.y = y;
  double P = par.K*(par.b*r-y);
  states.D = par.ad*states.D-par.bd*(y - states.yold);
  signals.v = P + states.I + states.D;
  if (signals.v < par.ulow) {
    signals.u = par.ulow;
  } else {
    if (signals.v > par.uhigh) {
      signals.u = par.uhigh;
    } else {
      signals.u = signals.v;
    }
  }
  return signals.u;
}

public void updateStates() {
  states.I = states.I + par.bi*(signals.r - signals.y)
           + par.ar*(signals.u - signals.v); // Integral part
  states.yold = signals.y;
}
```


Comments on PID Computer Code

- Controller state update is separated from controller output calculations
- Controller parameters are pre-calculated (e.g., `par.bi`, `par.ad`)
- A *thread* (e.g., `Thread.waitUntil`) is a real-time task, and supposed to be supported by the operating system

Well-Conditioned Realizations

Consider controller

$$u(k) = \frac{b_o + b_1 q^{-1} + \cdots + b_m q^{-m}}{1 + a_1 q^{-1} + \cdots + a_n q^{-n}} y(k)$$

This can be realized in different forms, which influence

- Sensitivity to parameter and state quantization
- Number of storage elements and parameters (memory)
- Parameter range

Sensitivity Analysis

Consider characteristic polynomial for $i \in \{1, \ldots, n\}$:

$$A(z, a_i) = z^n + a_1 z^{n-1} + \cdots + a_n = (z - p_1) \ldots (z - p_n)$$

Suppose perturbation $a_i + \delta a_i$ gives $p_k + \delta p_k$. Then,

$$\begin{aligned} 0 &= A(p_k + \delta p_k, a_i + \delta a_i) \\ &= A(p_k, a_i) + \frac{dA(p_k, a_i)}{dz} \delta p_k + \frac{dA(p_k, a_i)}{da_i} \delta a_i + \ldots \end{aligned}$$

For small perturbations,

$$\delta p_k \approx -\frac{dA/da_i}{dA/dz}(p_k, a_i) \delta a_i$$

$$\frac{dA(p_k, a_i)}{da_i} = p_k^{n-i}, \qquad \frac{dA(p_k, a_i)}{dz} = \prod_{j \neq k} (p_k - p_j)$$

Hence, if $p_j \neq p_k$,

$$\delta p_k \approx -\frac{p_k^{n-i}}{\prod_{j \neq k}(p_k - p_j)} \delta a_i$$

Note

- Sensitive to parameter quantization if poles are close and/or close to one
- For $|p_k| < 1$, largest perturbation is obtained for $i = n$, i.e., sensitivity largest for perturbations in a_n

Bad Implementation I: Direct Form

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

- Not minimal form: $m + n$ variables but only n states
- Parameters in implementation equal to characteristic polynomial: sensitive to computational errors if n large and poles close to one or close with each other

Bad Implementation II: Canonical Form

Canonical forms, such as observable canonical form:

$$x(k+1) = \begin{bmatrix} -a_1 & 1 & 0 & \ldots & 0 \\ -a_2 & 0 & 1 & \ldots & 0 \\ \vdots & & & \ddots & \vdots \\ -a_{n-1} & 0 & 0 & \ldots & 1 \\ -a_n & 0 & 0 & \ldots & 0 \end{bmatrix} x(k) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} y(k)$$

$$u(k) = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \end{bmatrix} x(k)$$

- Similar to in direct form: parameters in implementation equal to characteristic polynomial, so sensitive

Good Implementation

Controller with n_r distinct poles and n_c complex pairs:

$$z_i(k+1) = \lambda_i z_i(k) + \beta_i y(k), \qquad i = 1, \ldots, n_r$$

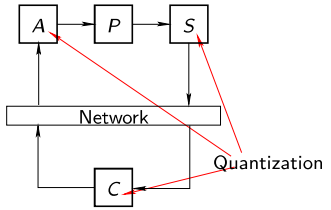
$$v_i(k+1) = \begin{bmatrix} \sigma_i & \omega_i \\ -\omega_i & \sigma_i \end{bmatrix} v_i(k) + \begin{bmatrix} \gamma_{i1} \\ \gamma_{i2} \end{bmatrix} y(k), \qquad i = 1, \ldots, n_c$$

$$u(k) = \sum_{i=1}^{n_r} \alpha_i z_i(k) + \sum_{i=1}^{n_c} \delta_i^T v_i(k)$$

- Robust to perturbations in parameters and states
- Parallel form implementation, as a cascade of first and second-order filters (see exercise 5.1)

Quantization

- Quantization in AD converters
- Quantization of controller parameters
- Roundoff, overflow, and underflow in operations (addition etc.)
- Quantization in DA converters



Quantization in AD and DA Converters

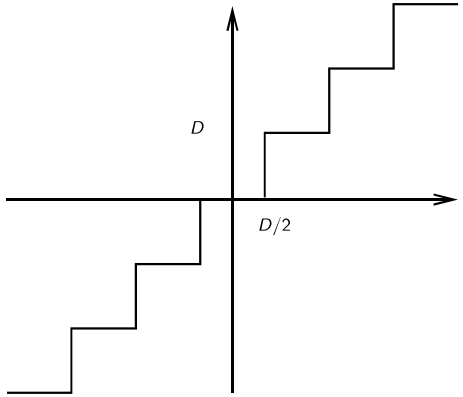
AD Converter

- Typical accuracy of 8, 10, 12, and 14 bits
- 8 bits correspond to $1/2^8 \approx 0.4\%$ resolution
- 14 bits correspond to $1/2^{14} \approx 0.006\%$ resolution

DA Converter

- Typical accuracy of 10 bits

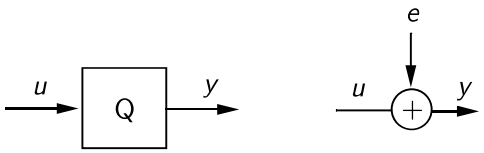
Uniform Quantization



Linear Model of Quantization

Model quantization error as a uniformly distributed stochastic signal e independent of u with

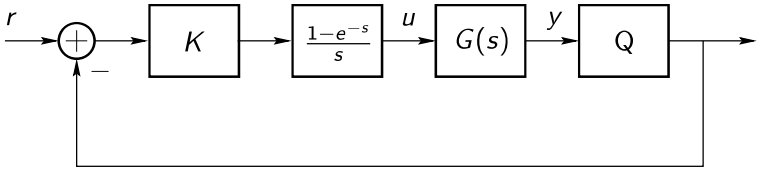
$$\text{Var}(e) = \int_{-\infty}^{\infty} e^2 f_e \, de = \int_{-D/2}^{D/2} \frac{e^2}{D} \, de = \frac{D^2}{12}$$

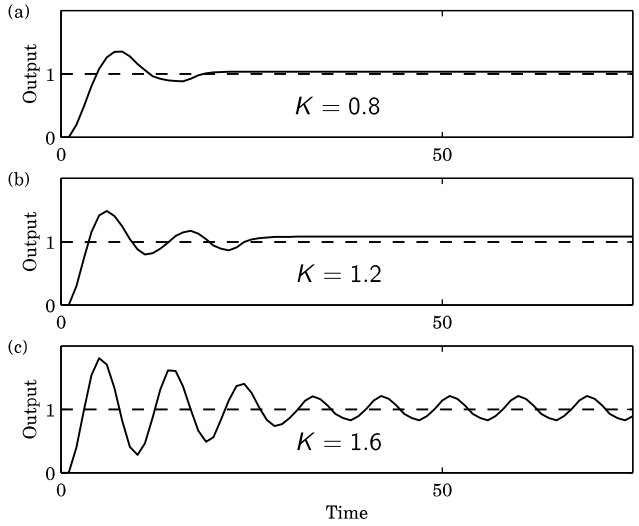


Works if D is small compared to the variations in u

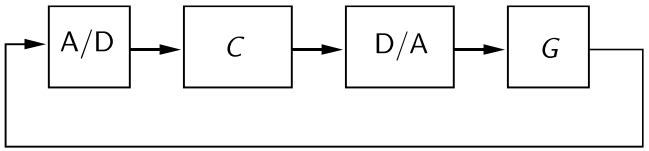
Example: Sensor Quantization

- Quantization of process output with $D = 0.2$
- Quantizer generates stable oscillation for $K = 1.6$
- Can be predicted using nonlinear control methods

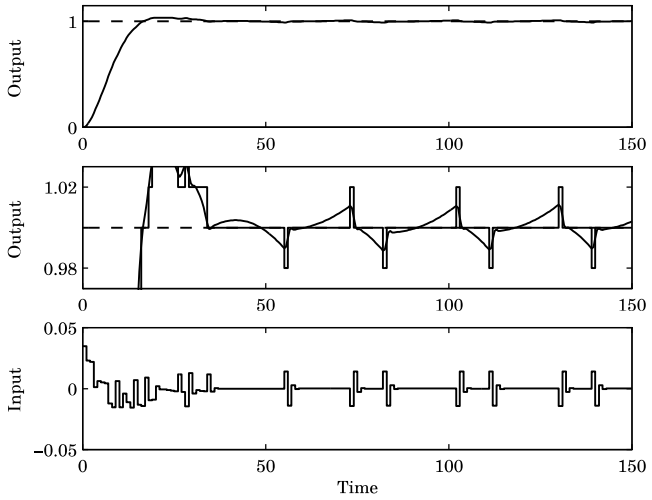




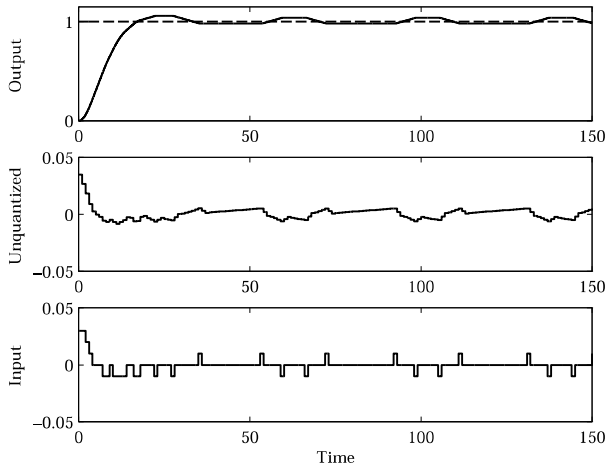
Example: Quantization in AD or DA



Quantization $D = 0.02$ in AD converter:



Quantization $D = 0.01$ in DA converter:



Choosing Sampling Time

Sampling time $h > 0$ might be

- Considered as an independent design parameter
- Fixed by the application or implementation platform
- Uncertain due to asynchronous sampling and hold
- Time varying due to communication and computation variations

Note

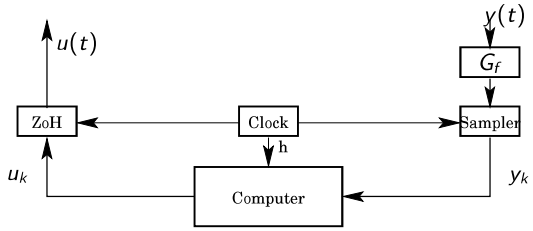
- Sampling time might heavily influence closed-loop performance
- Sometimes overlooked in the design
- Lack of systematic methods for choosing sampling time
- Use computer simulations off-line

Sampling Time Rule of Thumb

- Choose h such that there will be 4 to 10 samples per rise time

For a second-order system with natural frequency ω_0 this gives
$0.2 < \omega_0 h < 0.6$
The sampling frequency ω_s should thus fulfill $10\omega_0 < \omega_s < 30\omega_0$, so the sampling frequency for control is much higher than the frequency $2\omega_0$ needed for reconstructing a sampled signal of frequency ω_0
The rule of the thumb is built on extensive experience, but not much theory

Another Sampling Time Rule of Thumb



Choose h such that the zero-order-hold (ZoH) and anti-aliasing filter (G_f) give phase margin decrease of 5 to 15 deg
With cross-over frequency ω_c for the continuous-time plant, this results (under certain assumptions) in

$$h\omega_c \approx 0.05 \quad \text{to} \quad 0.14$$

Why Not $h \to 0$?

If a plant

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

is sampled fast ($h \approx 0$), then the discrete-time dynamics

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$

looks approximately like integrators to the controller because

$$\Phi = e^{Ah} \approx I + Ah$$

It will then require high numerical accuracy in the controller (computer) to be able to distinguish the difference.

Next Lecture

- Quantization in state feedback
- Network delays and data drops
- Compensation for delay variations

EL2450 Hybrid and Embedded Control

Lecture 5: Implementation aspects

- Modeling and compensation for jitter, delay, loss
- Quantization and packet losses in state feedback

Today's Goal

You should be able to

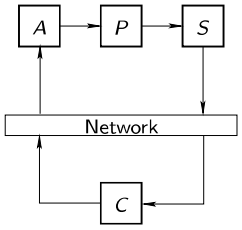
- Derive models for delay, jitter and loss
- Modify controllers to compensate for known and unknown delays
- Model and analyze packet losses and quantization effect in state feedback

Implementation Aspects

Computations and communications introduce imperfections, e.g.,

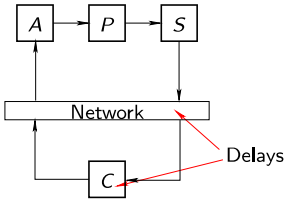
- Delays
- Packet Losses
- Quantization

Where do they appear in the control loop:



Time Delays

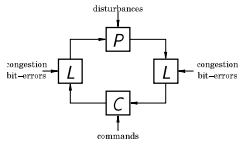
- Delays τ in communications and computations
- Delays are bad for control loops (avoid if possible)
- Delays can be known or unknown (influence control)
- Delay variation is denoted *jitter*
- Data loss (e.g., lost packet) can be interpreted as $\tau = \infty$



Communication Influence on Control Loop

Communication impose uncertainties

- Transmission delays
  - Data are delayed due to buffering and propagation delays
  - Delays are varying due to varying network load
- Data drops
  - Data are lost due to network protocol
  - Bit-errors in wireless links
  - Sudden loss of connection



Control Systems with Unknown Delays

Nyquist Criterion: Control system with phase margin φ_m at ω_c can have maximum (fixed) time delay

$$\tau < \varphi_m/\omega_c$$

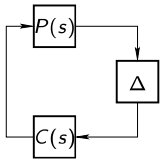
Example $P(s) = 1/s^2$ with $C(s) = K(1 + T_d s)$, $K = 1$, $T_d = 1.4$, gives phase margin $\varphi_m = 1.13 = 65$ deg at $\omega_c = 1.54$. Then,

$$\tau < \varphi_m/\omega_c = 0.73$$

Stability under Unknown Time-Varying Delay

Theorem: Consider linear feedback system with Δ representing a delay $0 \leq \tau(t) \leq \tau_{\max}$. Closed-loop system stable if

$$\left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \right| < \frac{1}{\tau_{\max}\omega}, \ \forall \omega \in [0, \infty]$$



Proof is based on small gain theorem (see [L, paper 3])

Relations to Nyquist Criterion

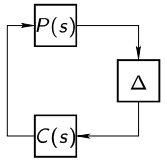
At $\omega = \omega_c$,

$$\left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \right| = \frac{1}{|1 - e^{i\varphi_m}|} \approx \frac{1}{\varphi_m}$$

Hence, closed-loop stability if

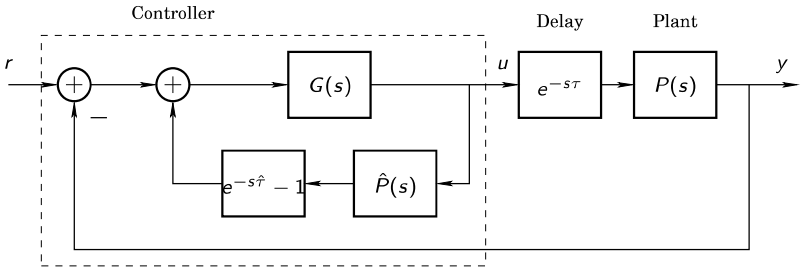
$$\frac{1}{\varphi_m} < \frac{1}{\tau_{\max}\omega_c}$$

Corresponds to Nyquist criterion for constant $\tau(t) = \tau_{\max}$



Control Systems with Known Delays

Known time delays can be compensated with **Smith predictor**:



Closed-loop system with $\hat{P} = P$ and $\hat{\tau} = \tau$: $y = \frac{PG}{1+PG} e^{-s\tau} r$
Design controller as if there were no time delay and then implement structure above

Example

Consider control design for

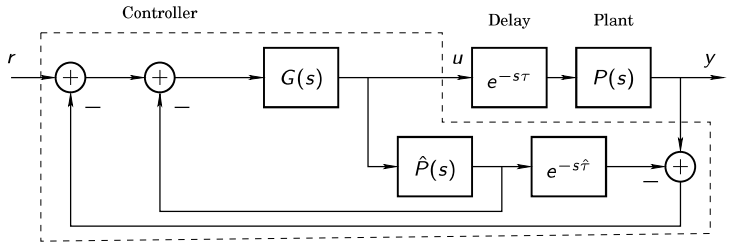
$$P(s)e^{-s\tau} = \frac{e^{-s\tau}}{s^2}$$

$G(s) = K(1 + T_d s)$, $K = 1$, $T_d = 1.4$, gives performance worse than the Smith Predictor.
Controller with Smith predictor (from u to $r - y$) is then given by

$$\begin{aligned} C(s) &= \frac{G(s)}{1 - P(s)G(s)(e^{-s\tau} - 1)} \\ &= \frac{Ks^2(1 + T_d s)}{s^2 + K(1 + T_d s) - K(1 + T_d s)e^{-s\tau}} \end{aligned}$$

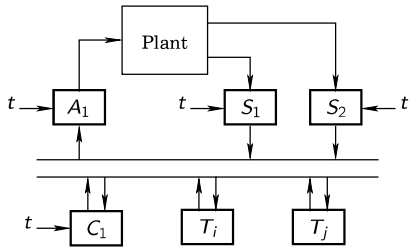
Interpretation of Smith Predictor

Compared to conventional PID control, the Smith predictor estimates old responses y. Alternative block diagram of Smith predictor:



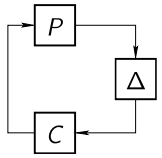
Time Stamps

Delays can be estimated from **time stamped data**, each node transmits data together with sampling time, e.g., $(y(t), t)$
If the receiving node is synchronized, it can determine and compensate time delay



Time Stamped Sensor Measurements

Suppose sensor measurements are delayed unknown and varying time $\tau(t)$
If sensor data $(y(t_s), t_s)$ is received at controller at time $t = t_c$, the current delay is $\tau(t) = t_c - t_s$, which can then be used in the control algorithm (cf., Smith predictor)



Important to take *all* delays into account (buffering, computation, propagation etc.)

Compensating Delays in State Feedback

Consider plant with state feedback

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

Sensor node sends $x(kh)$ to control node.
Suppose delay $\tau_k = \tau(kh) < h$ in Δ.
Controller has $x(kh)$ available and derives (draw time axis)

$$\bar{x}(kh + \tau_k) = e^{A\tau_k} x(kh) + \int_{kh}^{kh+\tau_k} e^{A(kh+\tau_k-s)} Bu(s)ds$$

and then

$$u(kh + \tau_k) = -L\bar{x}(kh + \tau_k)$$

Compensating Delays in Output Feedback

Consider plant

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

Sensor node sends $y(kh)$ to control node with transmission delay $\tau_k = \tau(kh) < h$.
Perform estimation and control in the following order:

$$\bar{x}(kh) = \hat{x}(kh) + K[y(kh) - C\hat{x}(kh)]$$

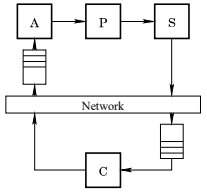
$$\bar{x}(kh + \tau_k) = e^{A\tau_k} \bar{x}(kh) + \int_{kh}^{kh+\tau_k} e^{A(kh+\tau_k-s)} Bu(s)ds$$

$$u(kh + \tau_k) = -L\bar{x}(kh + \tau_k)$$

$$\hat{x}(kh + h) = e^{A(h-\tau_k)} \bar{x}(kh + \tau_k) + \int_{kh+\tau_k}^{kh+h} e^{A(kh+h-s)} Bu(s)ds$$

Delays Larger Than h

- Similar scheme can be applied for a large (known) delay $\tau(t) > h$, by extending the state of the estimator (cf., sampling of systems with delay in Lecture 2)
- Buffers can be introduced to handle out-of-order delivery
- It is possible to use late data to adjust old estimates
- But buffers may introduce time delay

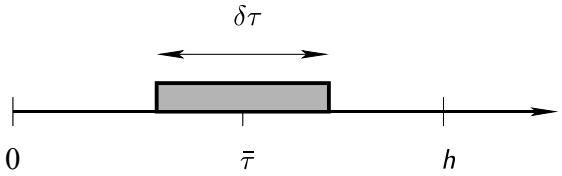


Jitter

Jitter δt is max deviation in time delay

$$\delta\tau = \tau_{\max} - \tau_{\min}$$

Introduce mean $\bar{\tau} = (\tau_{\max} + \tau_{\min})/2$



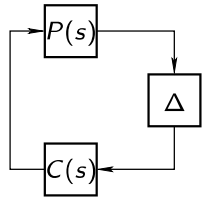
Stability with Jitter

Theorem: Consider linear feedback system with Δ representing a delay

$$0 \le \bar{\tau} - \delta\tau/2 \le \tau(t) \le \bar{\tau} + \delta\tau/2$$

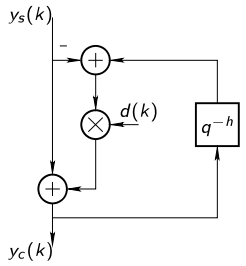
Closed-loop system stable if

$$\left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)e^{-i\omega\bar{\tau}}} \right| < \frac{\sqrt{2}}{\delta\tau \cdot \omega}$$



Data Loss Model

Let $d(k)$ be binary a stochastic variable, $y_s(k)$ the data packet transmitted at sensor node, and $y_c(k)$ received data packet $d(k) = 1$ corresponds to packet loss and $d(k) = 0$ to no loss



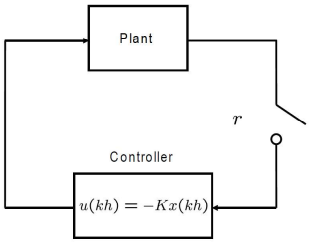
Observer with Loss

An observer handling data loss:

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) + \begin{cases} K[y(k) - C\hat{x}(k)], & d(k) = 0 \\ 0, & d(k) = 1 \end{cases}$$

Similar adjustments can be made to the (stochastic) Kalman filter, in which $K = K(k)$ is time varying

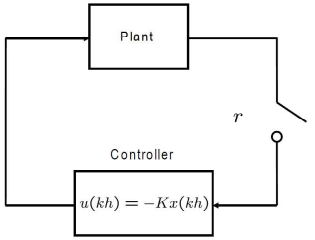
State Feedback Stability with Packet Losses



Plant dynamics: $x((k+1)h) = \Phi x(kh) + \Gamma u(kh)$

Controller: $u(kh) = -K\bar{x}(kh)$, where $\bar{x}(kh) = x(kh)$ if packet is transmitted, $\bar{x}(kh) = \bar{x}((k-1)h)$ otherwise.

State Feedback Stability with Packet Losses



Theorem Suppose that the closed-loop system without packet losses is stable, ie, the eigenvalues of $\Phi - \Gamma K$ are inside the unit circle. Let r be the successful packet reception rate. Then

- if the open-loop system is marginally stable, then the system is exponentially stable for all $0 < r \leq 1$.

State Feedback Stability with Packet Losses (cont.ed)

- if the open-loop system is unstable, then the system is exponentially stable for all

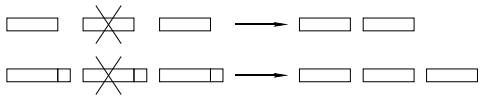
$$\frac{1}{1-\gamma_1/\gamma_2} < r \leq 1\,,$$

where $\gamma_1 = \log[\lambda^2_{\max}(\Phi - \Gamma K)]$, $\gamma_2 = \log[\lambda^2_{\max}(\Phi)]$

- Proofs in [ZBP]

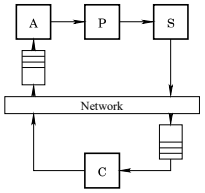
Error Correction

- Data drops are naturally handled through coding
- By introducing parity bits (redundancy), it is possible to reconstruct data at the receiver node even if some packets are lost
- The amount of redundancy $r(k)$ should be minimized in order to maximize data transmission, but still large enough to counteract data drops
- Redundancy $r(k)$ can be controlled based on feedback information on varying network load and other operating conditions



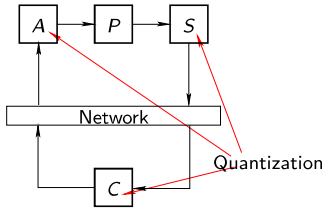
Delay or Loss

- There is a floating boundary between delay and drop
- TCP considers a packet to be lost after a specified time denoted `timeout`
- It can be better to drop data, than use old information for control
- Feedback is forgiving in the sense that communication drops and noise is often attenuated by controller



Quantization

- Quantization in AD converters
- Quantization of controller parameters
- Roundoff, overflow, and underflow in operations (addition etc.)
- Quantization in DA converters



State Feedback Stability with Quantization

- Quantization affects stability properties of nominal closed loop system.
- Can be analyzed in certain cases using Lyapunov techniques.
- Quantization induces error: compensated by feedback in some cases.

Review Lyapunov functions for continuous systems

A differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ is a *Lyapunov function* for $\dot{x} = f(x), f(0) = 0$ if

1. $V(0) = 0$ and $V(x) > 0$, $\forall x \neq 0$
2. $\dot{V}(x) := \dfrac{\partial V}{\partial x} f(x) \leq 0$, $\forall x \neq 0$

Linear Systems $V(x) = x^T P x$, P positive definite, is a (quadratic) Lyapunov function for $\dot{x} = Ax$, if (and only if)

$$A^T P + PA = -Q, \qquad Q \text{ positive semidefinite}$$

because $\dot{V}(x) = x^T(A^T P + PA)x = -x^T Q x \leq 0$.

Stability Test

The solution $x^*(t) = 0$ for $\dot{x}(t) = f(x(t))$ is stable if there exists a Lyapunov function. It is asymptotically stable if, moreover, $\dot{V}(x)$ is negative definite.

Linear Systems A linear system $\dot{x}(t) = Ax(t)$ is asymptotically stable if (and only if) for any positive definite Q, there exists positive definite P such that

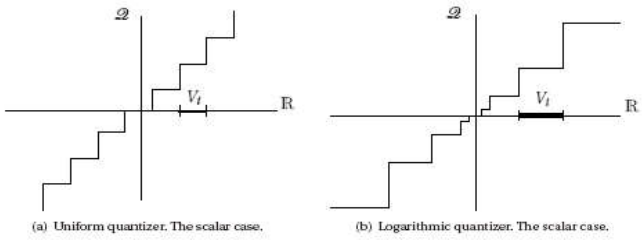
$$A^T P + PA = -Q$$

Stability LTI systems with errors

- Consider $\dot{x} = Ax + Bu$. Assume that $u = Kx$ is a stabilizing controller.
- Now assume measurement errors in the feedback, so that control input is $u = K(x + e)$.
- **Fact**: There exists a quadratic Lyapunov function V and $a, b > 0$ for which

$$\dot{V} \leq -a||x||^2 + b||x||||e||$$

Quantization Modelling

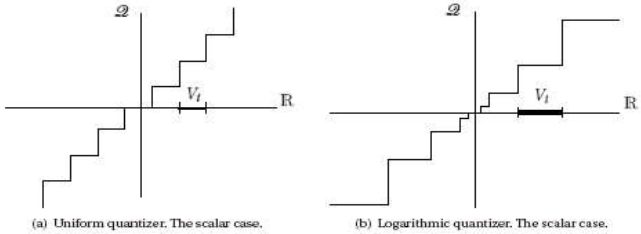


(a) Uniform quantizer. The scalar case. (b) Logarithmic quantizer. The scalar case.

In general, a quantizer $q : \mathbb{R}^n \to \mathcal{Q}$.

- Quantization regions are the sets $\{z \in \mathbb{R}^n | q(z) = i \in \mathcal{Q}\}$.
- Quantization range: highest value of signal that can be mapped to the quantizer.

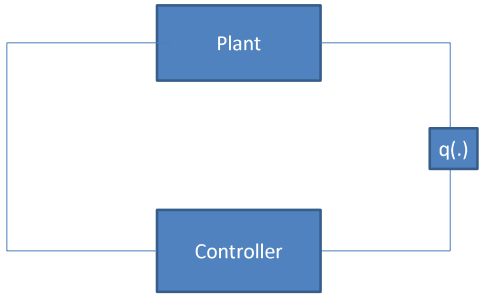
Quantization Error Modelling



(a) Uniform quantizer. The scalar case. (b) Logarithmic quantizer. The scalar case.

- Quantization error: $e(x(t)) = q(x(t)) - x(t)$.
- Uniform quantization: $||q(x(t)) - x(t)|| \leq \delta_u$.
- Logarithmic quantization: $||q(x(t)) - x(t)|| \leq \delta_l ||x(t)||$.

State Feedback Stability with Quantization



Here we consider state quantization. Input and output quantization is also possible.

State Feedback Stability with Quantization

- Plant: $\dot{x} = Ax + Bu$. Assume that $u = Kx$ is a stabilizing controller.
- Controller with quantization: $u = Kq(x)$. ("certainty equivalence controller").

Theorem Suppose that the closed-loop system without quantization is asymptotically stable. Then there exist sufficiently small δ_u, δ_l for which the quantized closed-loop system (i)is asymptotically stable, for the case of logarithmic quantizers, (ii) converges to a region around the equilibrium point, whose size depends on δ_u, for the case of uniform quantizers.
Note: proof based on Fact of slide 30.

State Feedback Stability with Quantization

- Can be extended to input ($u = q(Kx)$) and output quantization.
- Worst-case approach:maximum error is considered in the analysis.
- Alternative to consider stochastic approach (cf. Lecture 4).

Computer Arithmetics

- Control design and analysis are mainly based on high resolution and large range (floating-point) arithmetics: x, y, u supposed to be real valued
- Microcomputers in embedded systems may have fixed-point arithmetic

Analysis and design problems:

- What are the influences of limited word lengths (16 or 32 bits)?
- Is special attention needed in computations (overflow, roundoff)?
- Word length can sometimes be a choice, e.g., special-purpose VLSI circuits in consumer electronics

Floating-Point Arithmetics

IEEE 754 Standard: Numbers represented as

$$\pm a \cdot 2^b$$

where $0 \leq a < 2$ is the *significand*, and b the *exponent*

- Short real: 32 bits (Java/C: *float*)
  - 1 sign + 8 exponent + 23 significand
  - Range 2^{-126}–2^{128}
- Long real: 64 bits (Java/C: *double*)
  - 1 sign + 11 exponent + 52 significand
  - Range 2^{-1022}–2^{1024}

Supports infinity and NaN. Used by most processors, except some digital signal processors (DSP's)

Fixed-Point Arithmetics

- Word given in binary format
- Typical word lengths are 8, 16, and 32 bits
- 16 bits correspond to $\{-2^{15}, \ldots, 2^{15}-1\} = \{-32768, \ldots, 32767\}$

Computation properties:

- Result depends on order of computations
- Overflow risk
- Put attention to overflow characteristics (saturation)

Next Lecture

Event-based control and real-time systems

- Event-based control
- Real-time systems and scheduling

EL2450 Hybrid and Embedded Control

Lecture 6: Event-based control and real-time systems

- Event-based control
- Real-time systems

Today's Goal

You should be able to

- Design appropriate sampling times for event-based control
- Describe the main features of a real-time operating system
- Formulate the real-time scheduling problem

Event-based control

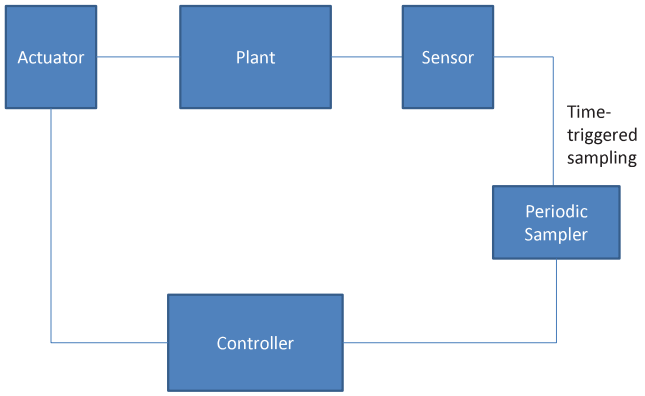
- Periodic, time-triggered sampling at pre-specified instants: does not take into account optimal resource usage
- Sampling time choise: heuristics, rules of thumb
- Unsettling not to formally derive sampling time
- A strategy considering better resource usage: event-triggered control
- Aperiodic sampling based on event-based rule
- Improves energy usage and processor time usability

Stability LTI systems with errors

- Consider $\dot{x} = Ax + Bu$. Assume that $u = Kx$ is a stabilizing controller.
- Now assume measurement errors in the feedback, so that control input is $u = K(x + e)$.
- **Fact**: There exists a quadratic Lyapunov function V and $a, b > 0$ for which

$$\dot{V} \leq -a||x||^2 + b||x||||e||$$

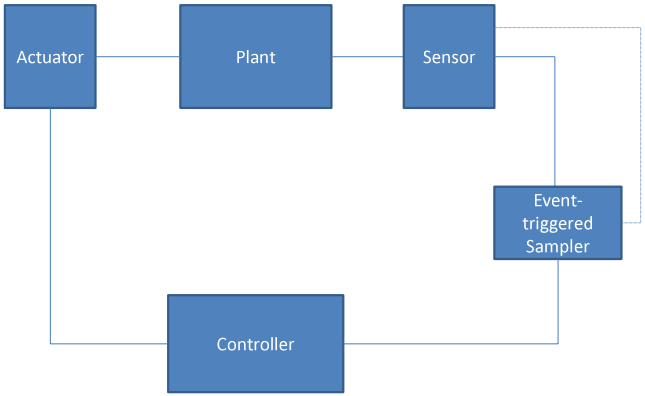
Periodic sampled control



Periodic sampled control

- Plant: $\dot{x} = Ax + Bu$.
- Assume that $u = Kx$ is a stabilizing controller without sampling.
- Controller with periodic sampling and ZOH:
  $u(t) = Kx(kh), t \in [kh, kh + h)$.

Event-based control



Event-based control

- Event-based control takes into account state or output feedback in order to sample as less as possible in an aperiodic fashion.
- Controller with aperiodic sampling and ZOH:
  $u(t) = Kx(t_k), t \in [t_k, t_{k+1})$.
- Question: how can we choose the sequence $\{t_k\}, k = 0, \ldots$ so that the stability properties are maintained and the sampling is as rare as possible?
- Answer: this is held by using *feedback* information from the plant in the sampling process!

State feedback stability with event-based control

State error: $e(t) = x(t_k) - x(t), t \in [t_k, t_{k+1})$. The control law can be written as
$u(t) = Kx(t_k) = K(x(t) - x(t) + x(t_k)) = K(x(t) + e(t))$.
The closed loop system can be written as
$\dot{x}(t) = (A + BK)x(t) + BKe(t)$.
Fact: since $A + BK$ is stable, there exists a quadratic Lyapunov function V for which

$$\dot{V} \leq -a||x||^2 + b||x||||e||$$

State feedback stability with event-based control

Idea: the choice of t_k affects the value e. Maintain $\dot{V}$ negative by choosing e appropriately. If $||e(t)|| \leq \sigma||x(t)||$ and $-a + b\sigma = -c < 0$, then $\dot{V} \leq -c||x||^2$ and asymptotic stability is maintained.
Event-triggering rule: If the last sample was at t_k, the next sample is taken at

$$t_{k+1} = \inf_t \{t > t_k | ||e(t)|| = \sigma||x(t)||\}$$

State feedback stability with event-based control

Theorem The event-based rule

$$t_{k+1} = \inf_t \{t > t_k | ||e(t)|| = \sigma||x(t)||\}$$

maintains the asymptotic stability of the original system for appropriate choice of $\sigma > 0$. Furthermore, the inter-sampling times $t_{k+1} - t_k$ are lower bounded by a positive constant τ^* for all k,$\sigma > 0$.

Inter-sampling times lower bound

Proof sketch After an event at t_k we have $||e(t_k)||/||x(t_k)|| = 0$. Next event happens when $||e(t)||/||x(t)|| = \sigma$. Time it takes for $||e||/||x||$ to go from zero to σ is lower bound by a strictly positive constant τ^*. This is due to that solutions of differential inequality

$$\frac{d}{dt}\frac{||e||^2}{||x||^2} \leq 2||A+BK||\frac{||e||}{||x||}+2(||BK||+||A+BK||)\frac{||e||^2}{||x||^2}+2||BK||\frac{||e||^3}{||x||^3}$$

need a minimum τ^* to go from 0 to σ^2.

Time delays

In case of a (small enough) fixed delay δ, we can compute a more conservative $0 < \sigma' < \sigma$ to compensate for the delay. Control law implemented with a delay δ:

$$u(t) = Kx(t_k), t \in [t_k + \delta, t_{k+1} + \delta)$$

Nonlinear systems

- Plant: $\dot{x} = f(x, u)$.
- Assuming that $u = k(x)$ is a stabilizing controller may not be enough. Need *assumption* on properties of controller with error measurements.
- Controller with aperiodic sampling and ZOH: $u(t) = k(x(t_k)), t \in [t_k, t_{k+1})$.
- Closed loop system is $\dot{x} = f(x, k(x + e))$.
- Existence of V similar to Fact of slide 4 not a given for nonlinear case.

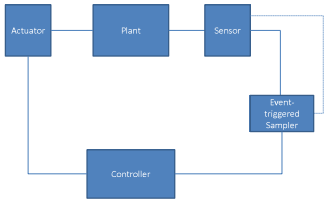
Nonlinear systems

- Assume V and strictly increasing functions $a(||x||), \gamma(||e||), a(0) = 0, \gamma(0) = 0$ such that

$$\frac{\partial V}{\partial x} f(x, k(x + e)) \leq -a(||x||) + \gamma(||e||)$$

- Then the trigger rule $\gamma(||e||) \leq \sigma a(||x||)$, $0 < \sigma < 1$ maintains the stability properties of the nominal closed-loop system (verify).
- $t_{k+1} = \inf_t \{t > t_k | \gamma(||e(t)||) = \sigma a(||x(t)||)\}$
- Robustness properties of closed-loop system need to be explored and not a given as in linear case.

Event-based control



- How can we avoid continuous monitoring of state or output feedback?
- Idea: use prediction of the state evolution. This can be done for linear systems (self-triggered control).
- Another idea: use periodic samples of state and implement control in an event-based fashion (periodic event-based control).

Periodic event-based control

- Plant: $\dot{x} = Ax + Bu$.
- Sample uniformly with period h, yealding

$$x(kh + h) = \Phi x(kh) + \Gamma u(kh)$$

where

$$\Phi = e^{Ah}, \quad \Gamma = \int_0^h e^{As} ds B$$

- h can be chosen as the minimum implementable period of the digital platform, since the interest is in less frequent controller updates.

Periodic event-based control

- Assume $u(kh) = Lx(kh)$ is a stabilizing controller.
- Then, there exists P such that

$$(\Phi + \Gamma L)^T P(\Phi + \Gamma L) - P = -Q$$

with P, Q positive definite.

- Idea: implement sampled controller in an event-based fashion. Controller given by

$$u(kh) = Lx(k_i h), k \in [k_i, k_{i+1}).$$

- Event sequence is now $\{k_0 h, k_1 h, \ldots\} \subset \{0, h, 2h, \ldots\}$.

State feedback stability with periodic event-based control

- State error:

$$e(kh) = x(k_i h) - x(kh), k \in [k_i, k_{i+1})$$

The control law can be written as $u(kh) = L(x(kh) + e(kh))$.

- Then $x(kh + h) = (\Phi + \Gamma L)x(kh) + \Gamma L e(kh)$.
- One can now show that $V = x^T P x$ satisfies

$$V(x(kh + h)) - V(x(kh)) \leq -a||x(kh)||^2 + b||e(kh)||^2$$

for some $a, b > 0$.

State feedback stability with periodic event-based control

If $||e(kh)|| \leq \sqrt{\sigma}||x(kh)||$ and $-a + b\sigma = -c < 0$, then

$$V(x(kh + h)) - V(x(kh)) \leq -c||x(kh)||^2$$

and asymptotic stability is maintained.

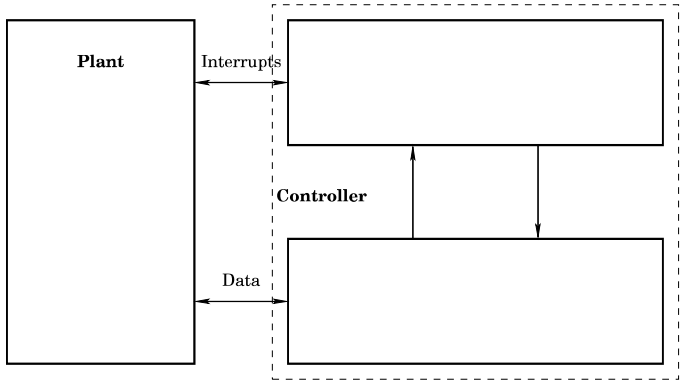
- This yields a triggering rule similarly to continuous-time case: next trigger time $k_{i+1} h$ chosen based on first next violation of $||e(kh)|| \leq \sqrt{\sigma}||x(kh)||$.
- Avoids continuous state measurements.

Comments on event-based control

- Choice of σ represents tradeoff between sampling frequency and decay rate of V.
- Minimum inter-sampling time τ^* can be used for periodic implementations.
- Delays and other dynamics can be considered.
- Trend: distributed event-based control.

Event-Triggered Control System

Need real-time operating system to handle multiple events and concurrency



What is a Real-Time Computer System?

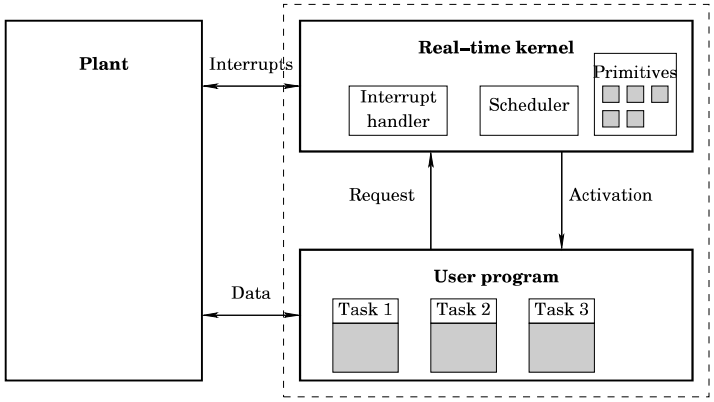
Real-time systems are defined as those computer systems for which correctness depends not only on the logical properties of the computations, but also on the time at which the results are computed.

Properties of Real-Time Systems

A real-time system is a computer system with

- Timing requirements (hard or soft)
- Time-triggered and event-triggered actions
- Concurrent activities
- Digital and analogue signals

Real-Time Operating System



Hard and Soft Real-Time Systems

- A **hard real-time system** is a system where it is absolutely necessary that the responses occur within the required deadline
- A **soft real-time system** is a system where deadlines are important but the system still functions if deadlines are occasionally missed

- Safety critical systems are often hard real-time systems
- Most hard real-time systems are control systems
- Most real-time control systems are not hard
- Real-time does not mean high-speed computations

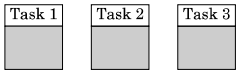
Events and Tasks

Real-time systems must respond to events:

- Periodic and aperiodic events
- External and internal events

A **task** is the work that has to be done to service an event.

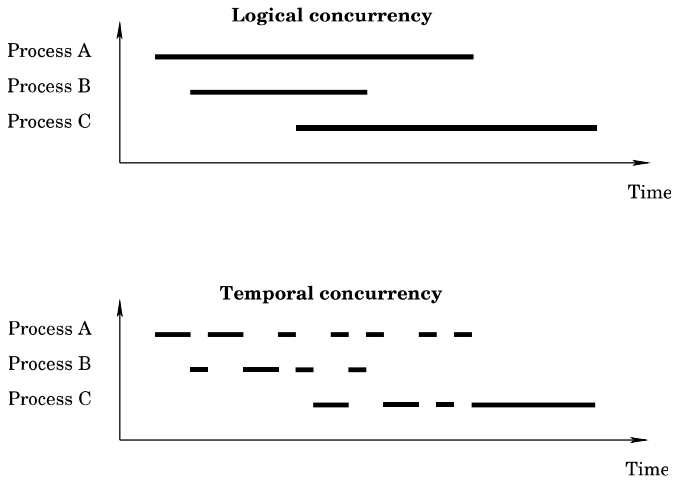
- A control task is an example of a task
- Tasks are often handled independently during design



Concurrency

- Real-time systems are concurrent
- Events may occur at the same time
- Processes multiplex their execution on a single CPU
- Lead to concurrent programming
- Requires scheduling and interrupts

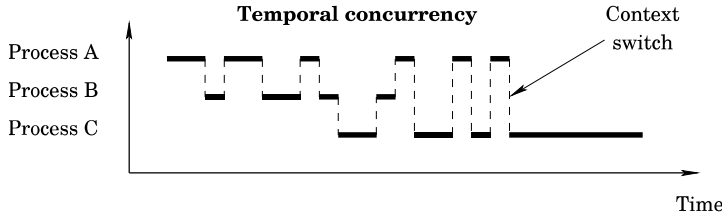
Logical and Temporal Concurrency



Context Switch

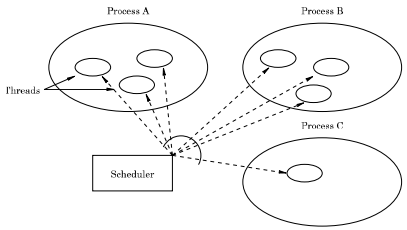
A **context switch** occurs when the system changes the running process

The context of the previously running process is stored and the context of the new process is restored



Processes and Threads

- A **process** is an executed program with its own allocated memory
- A process consists of **threads**, which implement the computations
- Threads are the basic unit of work handled by the scheduler



Example `PID` can be a process executing a PID controller, while `AD_conversion` and `updateState` can be threads inside `PID`.

Scheduling

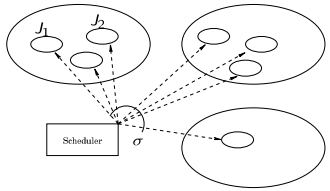
- **Clock-driven scheduling**
  - Decides on which process to execute based on the *time*
  - Schedule is computed off-line and stored for use at run time
  - Appropriate for hard real-time systems
- **Priority-driven scheduling**
  - Decides on which process to execute based on the process *priority*
  - Scheduling is an on-line activity, often hard to predict
  - CPU idle only when no processes need execution
  - Many scheduling strategies can be implemented through priorities

Here we mainly discuss priority-driven scheduling.

Schedule

For a set of tasks $J = \{J_1, \ldots, J_n\}$, a **schedule** is a map $\sigma : \mathbb{R}^+ \mapsto \{0, 1, \ldots, n\}$ assigning a task at each time instant t:

$$\sigma(t) = \begin{cases} k \neq 0, & \text{CPU should execute } J_k \\ 0, & \text{CPU is idle} \end{cases}$$

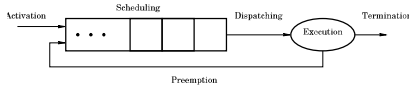


- σ is **feasible** if J can be completed according to specified constraints
- J is **schedulable** if there exists a feasible σ

Scheduling Algorithms

A **scheduling algorithm** sets task execution order (defines σ)
A scheduling algorithm is

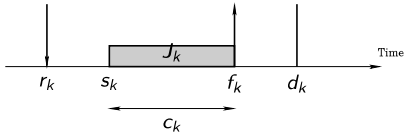
- **preemptive** if the running task can be arbitrarily suspended at any time (otherwise **non-preemptive**)
- **static** if scheduling decisions are based on fixed parameters assigned prior to activation (otherwise **dynamic**)
- **off-line** if σ is generated off-line and stored in a table (otherwise **on-line**)



Timing Constraints

A task J_k can be characterized by the following parameters:

- **Release time** r_k is the time at which J_k becomes ready for execution
- **Computation time** c_k is the time necessary for the CPU to execute J_k without interruption
- **Deadline** d_k is the time before which J_k should be completed
- **Start time** s_k is the (actual) time at which J_k starts executing
- **Finishing time** f_k is the (actual) time at which J_k finishes executing



Independent Periodic Tasks

We mainly focus on scheduling independent periodic tasks.
Tasks are

- **independent** if there are no precedence relations and no resource constraints
- **periodic** if they are activated at a constant rate

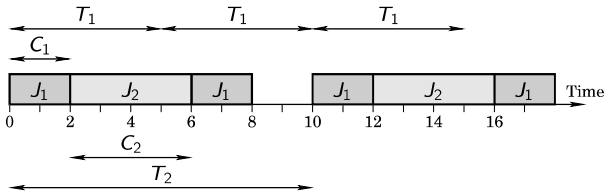
Periodic Tasks

Each periodic task J_k is characterized by its

- **Period** T_k
- **Worst-case computation time** C_k
- **Relative deadline** D_k (deadline relative to current release time)

Scheduled tasks have

- **Worst-case response time** R_k (largest time between release and termination)
- **Phase** ϕ_k (release time of the first task instance)



Schedule Length and Feasibility

The length of a schedule σ is equal to

$$\text{lcm}(T_1, \ldots, T_n)$$

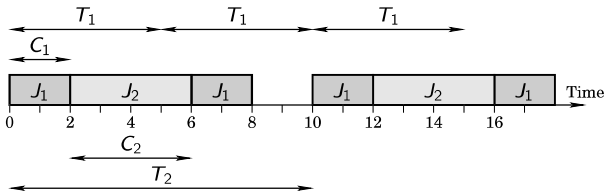
which is the least common multiplier of the task periods.

σ is feasible if all deadlines are met, i.e.,

$$R_k \leq D_k, \qquad \forall J_k \in J$$

Example

J_1 has $T_1 = 5$ and $C_1 = 2$
J_2 has $T_2 = 10$ and $C_2 = 4$
Deadlines $D_1 = T_1 = 5$ and $D_2 = T_2 = 10$ are met with schedule below, since $R_1 = 3 < D_1$ and $R_2 = 6 < D_2$.
Schedule length is $\text{lcm}(T_1, T_2) = \text{lcm}(5, 10) = 10$



Utilization Factor

The **utilization factor** U of a periodic task set J is the fraction of processor time spent in the execution of the task set.
Since C_i/T_i is the fraction for J_i, we have

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Example The previous task set gives

$$U = \frac{2}{5} + \frac{4}{10} = 0.8$$

Hence, CPU used at most 80% of the time.

Properties of Utilization Factor

It is desirable to find a scheduling algorithm that gives a feasible schedule

- If utilization factor $U > 1$, then the task set J is not schedulable
- Even if $U \leq 1$, it might be hard to find a feasible schedule
- Scheduling algorithms have various other properties (see next lecture)

Next Lecture

Real-Time Scheduling

- Scheduling algorithms
- Schedulability analysis

EL2450 Hybrid and Embedded Control

Lecture 7: Real-time scheduling

- Scheduling periodic and aperiodic tasks
- Schedulability analysis

Today's Goal

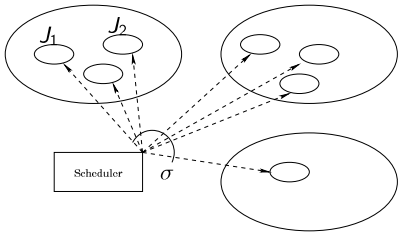
You should be able to model and analyze

- scheduling problems
- earliest deadline first scheduling
- rate monotonic scheduling
- deadline monotonic scheduling
- polling server

Scheduling

For a set of tasks $J = \{J_1, \ldots, J_n\}$, a **schedule** is a map $\sigma : \mathbb{R}^+ \mapsto \{0, 1, \ldots, n\}$ assigning a task at each time instant t:

$$\sigma(t) = \begin{cases} k \neq 0, & \text{CPU should execute } J_k \\ 0, & \text{CPU is idle} \end{cases}$$

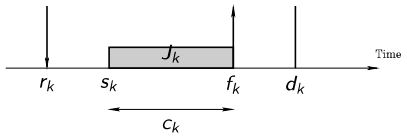


- σ is **feasible** if J can be completed according to specified constraints
- J is **schedulable** if there exists a feasible σ

Timing Constraints

A task J_k can be characterized by the following parameters:

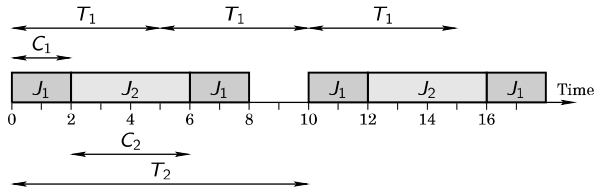
- **Release time** r_k is the time at which J_k becomes ready for execution
- **Computation time** c_k is the time necessary for the CPU to execute J_k
- **Deadline** d_k is the time before which J_k should be completed
- **Start time** s_k is the (actual) time at which J_k starts executing
- **Finishing time** f_k is the (actual) time at which J_k finishes executing



Independent Periodic Tasks

Suppose all tasks J_k are independent and periodic with

- **Period** T_k
- **Worst-case computation time** C_k
- **Relative deadline** D_k (deadline relative to current release time; often $D_k \equiv T_k$)
- **Worst-case response time** R_k (largest time between release and termination)
- **Phase** ϕ_k (release time of the first task instance)



Schedule Length and Feasibility

For independent and periodic tasks J, the length of a schedule σ is equal to

$$\text{lcm}(T_1, \ldots, T_n)$$

σ is feasible if all deadlines are met, i.e.,

$$R_k \leq D_k, \qquad \forall J_k \in J$$

Utilization Factor

The **utilization factor** U of a periodic task set J is the fraction of processor time spent in the execution of the task set. Since C_i/T_i is the fraction for J_i, we have

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- If $U > 1$, then the task set J is not schedulable
- Even if $U \leq 1$, it might be hard to find a feasible schedule
- U is independent of the scheduling algorithm

Scheduling Problem

The scheduling problem of finding a feasible σ for a set of independent periodic tasks $J = \{J_1, \ldots, J_n\}$ can be formulated as

$$\begin{aligned} &\text{Find} && \sigma \\ &\text{such that} && R_k \leq D_k \\ & && U \leq 1 \end{aligned}$$

We will consider the following potential solutions

- Earliest deadline first scheduling
- Rate monotonic scheduling
- Deadline monotonic scheduling

Earliest Deadline First Scheduling

Earliest deadline first (EDF) scheduling algorithm assigns **dynamic priorities** to the tasks based on their absolute deadlines:

Execute task with shortest time to deadline d_k

Note

- Priorities are set dynamically
- Works also for aperiodic tasks

EDF Schedulability

A set of periodic tasks $J = \{J_1, \ldots, J_n\}$ with $D_k = T_k$, $k = 1, \ldots, n$, is schedulable with EDF if and only if

$$U \leq 1$$

Note

- Processor can be fully utilized with EDF.
- A similar result holds even if $D_k \neq T_k$.
- If J can be scheduled by any algorithm, then EDF can schedule J. Equivalently, for $U > 1$, no algorithm can produce a feasible schedule.

Proof Sketch

(Only if) The total demand of computation time by all tasks between $t = 0$ and $t = T_1 T_2 \ldots T_n$ is

$$\frac{T_1 T_2 \ldots T_n}{T_1} C_1 + \frac{T_1 T_2 \ldots T_n}{T_2} C_2 + \ldots + \frac{T_1 T_2 \ldots T_n}{T_n} C_n$$

If this exceeds the available processor time $t = T_1 T_2 \ldots T_n$, ie,

$$\frac{T_1 T_2 \ldots T_n}{T_1} C_1 + \frac{T_1 T_2 \ldots T_n}{T_2} C_2 + \ldots + \frac{T_1 T_2 \ldots T_n}{T_n} C_n > T_1 T_2 \ldots T_n$$

or if $\sum_{i=1}^{n} \frac{C_i}{T_i} = U > 1$ then J is not schedulable with EDF (or any other scheduling algorithm).

Proof Sketch

(If) Shown by contradiction. Suppose that $U \leq 1$ and J is not schedulable. Let t_2 be the instant of time-overflow (deadline of an unfulfilled request). Let $[t_1, t_2]$ be the longest interval of continuous utilization, such that only tasks with deadline less than or equal to t_2 are executed in $[t_1, t_2]$. Then, t_1 is the release time of some periodic task. The total computation time demanded by periodic tasks in $[t_1, t_2]$ is

$$C_p(t_1, t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

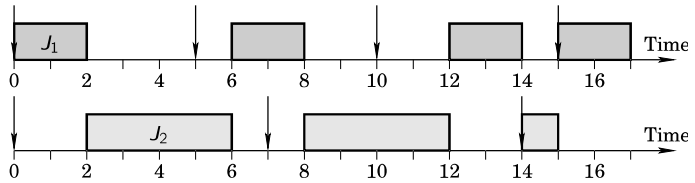
where $\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor$ is the total number of periods of J_i entirely contained in $[t_1, t_2]$. Then,

$$C_p(t_1, t_2) \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1) U$$

Since there is no processor idle period, $C_p(t_1, t_2) > t_2 - t_1$, we get the contradiction $U > 1$.

Example: EDF Scheduling

J_1 has $T_1 = D_1 = 5$, $C_1 = 2$
J_2 has $T_2 = D_2 = 7$, $C_2 = 4$
Since $U = \frac{2}{5} + \frac{4}{7} = 0.97 \leq 1$, the tasks are schedulable with EDF.



Rate Monotonic Scheduling

Rate monotonic (RM) scheduling algorithm assigns **fixed priorities** to tasks, such that $T_i < T_j$ implies that J_i gets higher priority than J_j.

Note

- Provides a way to set fixed priorities for a set of tasks
- Fixed priorities might otherwise often be set heuristically

RM Schedulability

A set of periodic tasks $J = \{J_1, \ldots, J_n\}$ is schedulable with RM if

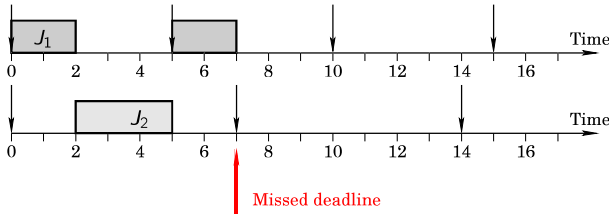
$$U \leq n(2^{1/n} - 1)$$

Note

- Not a necessary condition, so there might exist an RM schedule even if U does not fulfill the inequality
- $n(2^{1/n} - 1) \to \ln 2 \approx 0.69$, as $n \to \infty$, so RM can always schedule J if the total process utilization is less than 0.69
- A maximum utilization of 0.69 is often used as a rule of thumb for RM

Example: RM Scheduling

Try to schedule the previous example with RM. Since $T_1 < T_2$, RM gives higher priority to J_1 than J_2.
RM does not give a feasible schedule!



Note that $U = 0.97 > 2(2^{1/2} - 1) \approx 0.83$

RM is Optimal

If a set of periodic tasks are not schedulable by RM, then the set is not schedulable by any other **fixed priority** scheduling algorithm.

- RM is in this sense the best fixed priority algorithm
- RM is not good when $D_i \ll T_i$ (rare but urgent tasks)

Deadline Monotonic Scheduling

Deadline monotonic (DM) scheduling algorithm assigns **fixed priorities** to tasks, such that $D_i < D_j$ implies that J_i gets higher priority than J_j.

Note

- At any instant, the task with shortest relative deadline is executed
- Fixed priority schedule since relative deadlines are constant
- For tasks with deadlines less than periods
- Works for rare but urgent tasks
- DM=RM if $D_i \equiv T_i$

Worst-Case Response Time Calculation

Suppose the tasks $J_1, \ldots, J_i$ are ordered by decreasing fixed priority. Worst-case response time R_i for J_i is the largest time between release and termination. It can be derived as the smallest positive solution to

$$R_i = C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j$$

Note

- R_i appears on both sides of the equation
- $\sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j$ represents the preemption by higher-priority tasks

Example

Consider tasks (from previous examples):
J_1 has $T_1 = 5$, $C_1 = 2$, high priority
J_2 has $T_2 = 7$, $C_2 = 4$, low priority
Worst-case response times are then $R_1 = 2$ and $R_2 = 8$, because:

$$R_1 = C_1 = 2, \qquad R_2 = C_2 + \lceil \frac{R_2}{T_1} \rceil C_1 = 4 + \lceil \frac{R_2}{5} \rceil 2$$

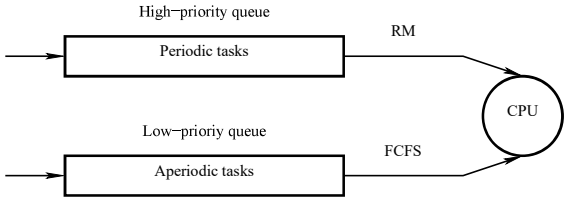
Iterate over R_2^k with $R_2^0 = 0$:

$$R_2^1 = 4 + \lceil \frac{R_2^0}{5} \rceil 2 = 4, \qquad R_2^2 = 4 + \lceil \frac{4}{5} \rceil 2 = 6$$
$$R_2^3 = 4 + \lceil \frac{6}{5} \rceil 2 = 8, \qquad R_2^4 = 4 + \lceil \frac{8}{5} \rceil 2 = 8 = R_2^3$$

Scheduling Periodic and Aperiodic Tasks Together

Background Scheduling

- Schedule aperiodic tasks in the background (when CPU would be idle)
- May lead to long response time for aperiodic requests



Polling Server Scheduling

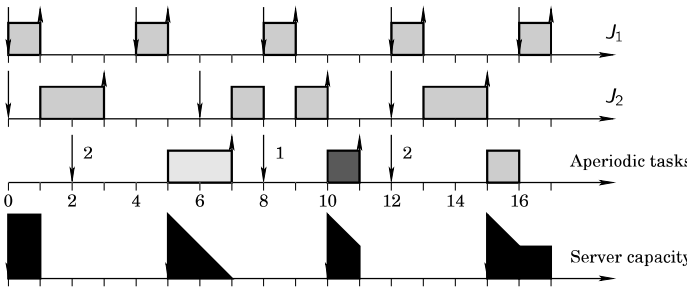
- A *polling server* is a periodic task that serves aperiodic tasks
- Gives guaranteed CPU utilization also for the aperiodic tasks

Polling Server

- A polling server task J_S is characterized by a period T_S and a server capacity C_S, as any other periodic task
- The polling server is scheduled by the algorithm for periodic tasks
- Once activated, the server starts serving the pending aperiodic requests within the limit of its capacity
- Several scheduling strategies possible for the aperiodic requests

Example: RM Scheduling and Polling Server

Periodic task J_1: $T_1 = 4$, $C_1 = 1$
Periodic task J_2: $T_2 = 6$, $C_2 = 2$
Server task J_S: $T_S = 5$, $C_S = 2$



Subtask Scheduling

- It is often suitable to divide tasks into subtasks, e.g., control tasks
- May create dependency, so it is in general harder to design schedule

Control Tasks

Each control task J_k is dived into four subtasks:

J_k^{AD} AD conversion

J_k^{CO} Calculate controller output

J_k^{DA} DA conversion

J_k^{US} Update state


```
nexttime = getCurrentTime();
while (true) {
  AD_conversion();
  calculateOutput();
  DA_conversion();
  updateState();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```


Design Control Task Schedule

- Set $D^{US} = T$ for all tasks
- Minimize D^{CO} for all tasks

Next Lecture

Models of computation

- Discrete-event systems
- Transition systems

EL2450 Hybrid and Embedded Control

Lecture 8: Models of Computation

- Discrete-event systems
- Transition systems

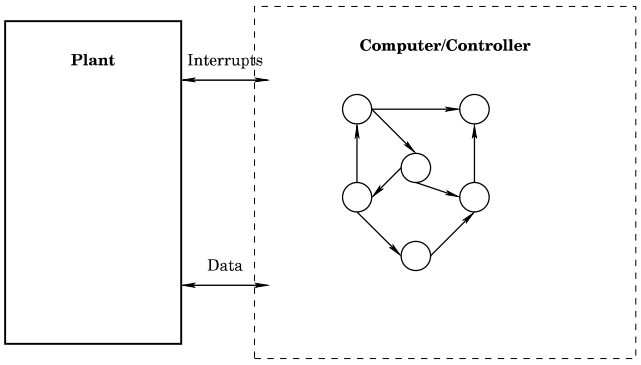
Today's Goal

You should be able to

- model and analyze automata
- deadlock, livelock, blocking, state-space minimization
- model and analyze transition systems
- do reach set computations
- do verification of safety and invariance properties

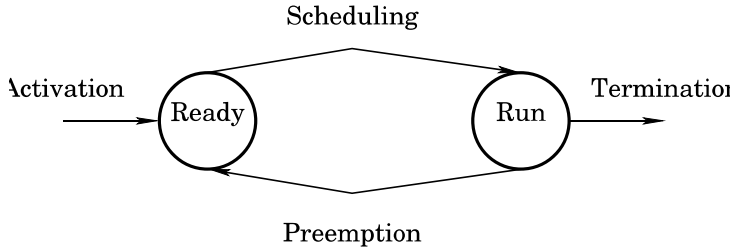
How Model Control Computations?

- Need mathematical models for analysis, design and verification
- Models should capture real-time and discrete-event features



Example: Scheduling

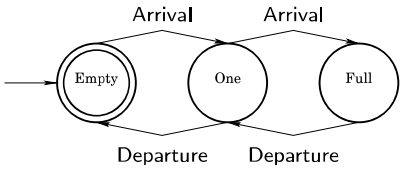
State machine model of task scheduling:



- Nodes represent states
- Edges represent transitions between states

Example: Queue

Model of a small queue with two positions:



- Nodes represent states (number of elements in the queue)
- Edges represent transitions between states
- Transitions are taken at events "Arrival" and "Departure"

Automaton

A deterministic automaton A is a five-tuple

$$A = (Q, E, \delta, q_0, Q_m)$$

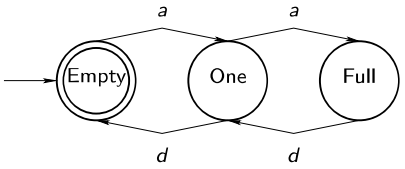
- Q is a finite set of states
- E is a finite set of events
- $\delta : Q \times E \mapsto Q$ is a transition function
- $q_0 \in Q$ is the initial state
- $Q_m \subseteq Q$ is the marked (or final) states

$q' = \delta(q, e)$ means that there is a transition labeled by event e from state q to q'.

Example: Queue Automaton

$A_Q = (Q, E, \delta, q_0, Q_m)$

- $Q = \{\text{Empty}, \text{One}, \text{Full}\}$, $E = \{a, d\}$
- $\delta(\text{Empty}, a) = \text{One}$, $\delta(\text{One}, a) = \text{Full}$, $\delta(\text{Full}, d) = \text{One}$, $\delta(\text{One}, d) = \text{Empty}$
- $q_0 = \text{Empty}$
- $Q_m = \text{Empty}$



Arrow without label indicates initial state and circle indicates marked (final) state

Events

ϵ denotes the empty string (no event): $\delta(q, \epsilon) = q, \forall q \in Q$.
E^* denotes all finite strings of elements of E together with ϵ.
$q = \delta(q_0, s)$, $s \in E^*$, denotes the state reached after executing the string $s = e_1 e_2 \dots e_k$, $e_i \in E$, i.e.,

$$q = \delta(\delta(\dots(\delta(q_0, e_1), \dots, e_k)$$

where all transitions are supposed to be well-defined.

Example: If $E = \{a, d\}$ then $E^* = \{\epsilon, a, d, aa, ad, dd, \dots\}$ and, for example, $\delta(q_0, ad) = \delta(\delta(q_0, a), d)$

Languages

A **language** L defined over an event set E is a set of finite strings formed from the events in E, so $L \subseteq E^*$.

Example
Let $E = \{a, d\}$. Then $L_1 = \{a\}$, $L_2 = \{a, aad\}$, and $L_3 = \{\epsilon, a, d\}$ are languages.

Operations on Languages

For $L, L_1, L_2 \subseteq E^*$, the **concatenation** of L_1, L_2 is defined as

$$L_1 L_2 = \{s \in E^* : \ (s = s_1 s_2) \text{ and } (s_i \in L_i, i = 1, 2)\}$$

The **Kleene-closure** of L is defined as

$$L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots$$

Example
Let $E = \{a, d\}$. For $L_1 = \{a\}$, $L_2 = \{a, aad\}$, $L_1 L_2 = \{aa, aaad\}$ and $L_2^* = \{\epsilon, a, aad, aa, aaad, aada, aadaad, \ldots\}$.

Generated and Marked Languages

A language **generated** by an automaton A is defined as

$$L(A) = \{s \in E^* : \ \delta(q_0, s) \text{ is defined}\}$$

A language **marked** by an automaton A is defined as

$$L_m(A) = \{s \in L(A) : \ \delta(q_0, s) \in Q_m\}$$

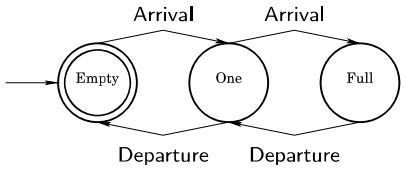
- $L(A)$ represents all directed paths in the state transition diagram starting from the initial state
- $L_m(A)$ represents the subset of these paths that ends in the marked states
- $L_m(A) \subseteq L(A)$

Example

The marked language of the queue automaton A_Q is

$$L_m(A_Q) = \{(a(ad)^* d)^*\}$$

It represents all arrival/departure sequences that start and end with an empty queue.

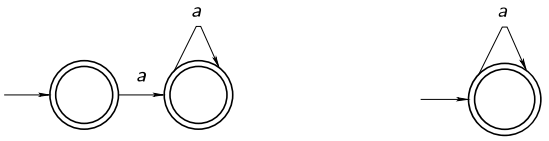


Note that $L(A_Q) \neq E^*$. Why?

Equivalent Automata

A_1 and A_2 are **equivalent** if $L(A_1) = L(A_2)$ and $L_m(A_1) = L_m(A_2)$.

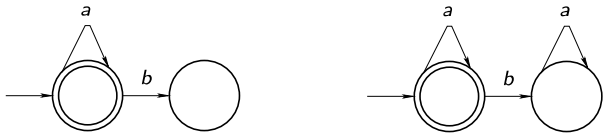
Example
The following automata are equivalent:



Deadlock and Livelock

A **deadlock** is a situation when an unmarked state can be reached and from which no further event can be executed
A **livelock** is a situation when a subset of unmarked states can be reached, but no transition is going out from the subset

Example

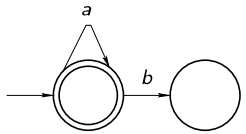


Prefix-Closure

$$\overline{L_m(A)} = \{s \in E^* : \exists s' \in E^*, ss' \in L_m(A)\}$$

is the prefix-closure of $L_m(A)$. Includes all prefixes of all strings in $L_m(A)$.
Note that $L_m(A) \subseteq \overline{L_m(A)} \subseteq L(A)$ (why?)

Example



$L(A) = \{a^*, a^*b\}, \quad L_m(A) = \{a^*\}, \quad \overline{L_m(A)} = L_m(A)$

Deadlock/Livelock and Prefix-Closure

If **deadlock** happens then $\overline{L_m(A)}$ is a proper subset of $L(A)$. This follows from that any string in $L(A)$ that ends at a deadlock state q cannot be a prefix of a string in $L_m(A)$.

If **livelock** happens then $\overline{L_m(A)}$ is a proper subset of $L(A)$. This follows from that any string in $L(A)$ that reaches the "absorbing" set of unmarked states cannot be a prefix of a string in $L_m(A)$ (since there is no way out of the "absorbing" set).

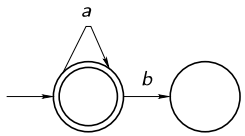
Blocking is used to denote these two cases.

Blocking Automaton

Definition
Automaton A with $\overline{L_m(A)} \subset L(A)$ is called **blocking**.

Definition
Automaton A with $\overline{L_m(A)} = L(A)$ is called **nonblocking**.

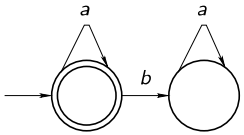
Example: Blocking and Deadlock



$$L(A) = \{a^*, a^*b\}, \quad L_m(A) = \{a^*\}, \quad \overline{L_m(A)} = L_m(A)$$

Since $\overline{L_m(A)} \subset L(A)$, the automaton is blocking. Note the deadlock.

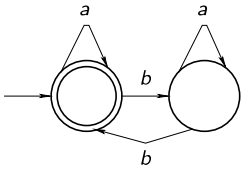
Example: Blocking and Livelock



$$L(A) = \{a^*, a^*ba^*\}, \quad L_m(A) = \{a^*\}, \quad \overline{L_m(A)} = L_m(A)$$

Since $\overline{L_m(A)} \subset L(A)$, the automaton is blocking. Note the livelock.

Example: Nonblocking



$$L(A) = E^*, \quad L_m(A) = \{a^*, (a^*ba^*b)^*\}, \quad \overline{L_m(A)} = E^*$$

Since $\overline{L_m(A)} = L(A)$, the automaton is nonblocking.

Nondeterministic Automaton

A nondeterministic automaton A is a five-tuple

$$A = (Q, E \cup \{\epsilon\}, \delta, q_0, Q_m)$$

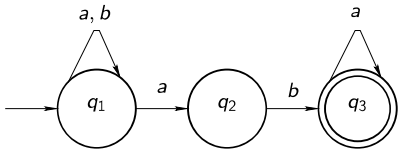
where

- Q is a finite set of states
- $E \cup \{\epsilon\}$ is a finite set of events
- $\delta : Q \times E \cup \{\epsilon\} \mapsto 2^Q$ is a transition function
- $q_0 \subseteq Q$ is the set of initial states
- $Q_m \subseteq Q$ is the marked (or final) states

Note: The differences to a deterministic automaton is that

- δ maps to a set (2^Q denotes the set of all subsets of Q)
- q_0 is a set

Example



What happens when more than one transition is possible?

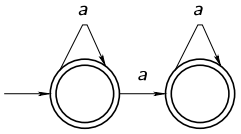
- The machine "splits" into multiple copies
- Each branch follows one possibility
- Together, branches follow *all* possibilities.

Example
What happens if the automaton receives the sequence *aba* ?

State-Space Minimization

There is no unique way to construct an automaton that marks a given language.
It is sometimes desirable to find the automaton with smallest number of states (lowest cardinality of Q) that marks a language.

Example
Minimize the state-space of the following automaton:

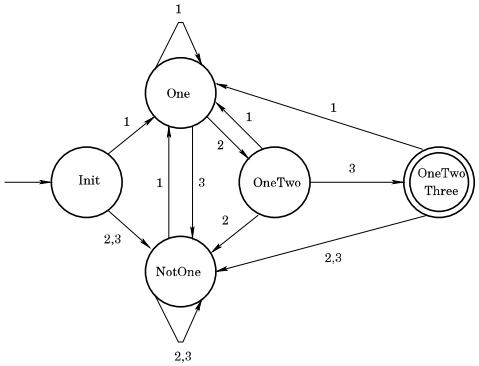


Example: A Digit Sequence Detector

Design an automaton A for a machine that reads digits from $\{1, 2, 3\}$ and detects (i.e., marks) any string that ends with the substring "123".
A should generate E^*, with $E = \{1, 2, 3\}$, since it should accept any input digit at any time. A should mark

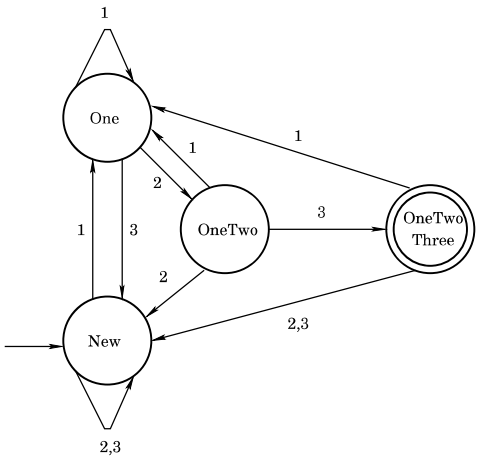
$$L = \{ss' \in E^* : \ s \in E^*, s' = 123\}$$

A non-minimal solution:



Note that $\delta(\text{Init}, e) = \delta(\text{NotOne}, e)$ for all e.

A minimal solution in which Init and NotOne are aggregated:



Summary: Discrete-Event Systems

- An automaton is a mathematical model of a discrete-event system
- Useful to analyze blocking of the system, automata interconnections possible
- Also possible to optimize criteria and check other properties

- Automata are defined only for *finite* sets of states and events
- Transition systems: generalize to possibility of *infinite* sets of states and/or events

Transition Systems

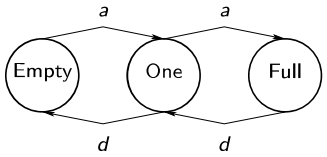
A **transition system** is a tuple $T = (S, \Sigma, \rightarrow)$ where

- S is a set of states
- Σ is a set of generators (or actions)
- $\rightarrow \subset S \times \Sigma \times S$ is a transition relation

- We use the notation $s \xrightarrow{\sigma} s'$ to denote $(s, \sigma, s') \in \rightarrow$

- We may include initial and final states, $S_0 \subseteq S$ and $S_F \subseteq S$, respectively. Then we denote $T = (S, \Sigma, \rightarrow, S_0, S_F)$.

Discrete Event System as a Transition System



$T_{\text{DES}} = (S, \Sigma, \to)$ where

- $S = \{\text{Empty}, \text{One}, \text{Full}\}$ is a set of states
- $\Sigma = \{a, d\}$ is a set of generators
- $\to = \{(\text{Empty}, a, \text{One}), (\text{One}, a, \text{Full}), (\text{Full}, d, \text{One}), (\text{One}, d, \text{Empty})\}$ or with the other (more intuitive) notation:

$$\text{Empty} \xrightarrow{a} \text{One}, \quad \text{One} \xrightarrow{a} \text{Full}, \quad \text{Full} \xrightarrow{d} \text{One}, \quad \text{One} \xrightarrow{d} \text{Empty}$$

Differential Equation as a Transition System

Consider the ordinary differential equation $\dot{x} = f(x)$, $x \in \mathbb{R}$, and let $\phi(t)$ denote its solution
The ODE defines the transition system $T_{\text{ODE}} = (S, \Sigma, \to)$ where

- $S = \mathbb{R}$
- $\Sigma = \text{Time} = \{t : \ t \geq 0\}$
- $x \xrightarrow{t} y$ provided that $x = \phi(0)$ and $y = \phi(t)$

Predecessor and Successors

For transition system $T = (S, \Sigma, \to)$ define the **predecessor** operator for $P \subset S$ as

$$\text{Pre}(P) = \{s \in S : \ \exists \sigma \in \Sigma, \exists p \in P, s \xrightarrow{\sigma} p\}$$

the **successor** operator as

$$\text{Post}(P) = \{s \in S : \ \exists \sigma \in \Sigma, \exists p \in P, p \xrightarrow{\sigma} s\}$$

Recursively, we define

$$\text{Post}^0(P) = P, \qquad \text{Post}^k(P) = \text{Post}(\text{Post}^{k-1}(P))$$

and similarly for Pre^k

Example

For the discrete event system,

$$\text{Pre}(\text{Empty}) = \text{One}, \quad \text{Pre}(\{\text{Empty}, \text{One}\}) = \{\text{Empty}, \text{One}, \text{Full}\}$$

For the ODE $\dot{x} = -x$:

$$\text{Pre}([-\epsilon, \epsilon]) = \begin{cases} \mathbb{R}, & \epsilon > 0 \\ 0, & \epsilon = 0 \end{cases}$$

$$\text{Post}([-\epsilon, \epsilon]) = [-\epsilon, \epsilon]$$

Pre_σ and Post_σ

For specific $\sigma \in \Sigma$, the $\mathrm{Pre}_\sigma(P)$ and $\mathrm{Post}_\sigma(P)$ are

$$\mathrm{Pre}_\sigma(P) = \{s \in S : \exists p \in P, s \xrightarrow{\sigma} p\}$$

$$\mathrm{Post}_\sigma(P) = \{s \in S : \exists p \in P, p \xrightarrow{\sigma} s\}$$

Recursively,

$$\mathrm{Pre}^0_\sigma(P) = P \qquad \mathrm{Pre}^k_\sigma(P) = \mathrm{Pre}_\sigma(\mathrm{Pre}^{k-1}_\sigma(P))$$

and similarly for Post_σ

Reach Set

$\mathrm{Reach}(S_0)$ is the set of states that can be reached from S_0 by a sequence of transitions, i.e.,

$$\mathrm{Reach}(S_0) = \bigcup_{k=0,1,\ldots} \mathrm{Post}^k(S_0)$$

Examples

$$\mathrm{Reach}_{T_{\mathrm{DES}}}(\mathrm{Empty}) = \{\mathrm{Empty}, \mathrm{One}, \mathrm{Full}\}$$

For the differential equation $\dot{x} = ax$, $a > 0$,

$$\mathrm{Reach}_{T_{\mathrm{ODE}}}(x_0) = [x_0, \infty), \text{if } x_0 > 0$$

$$\mathrm{Reach}_{T_{\mathrm{ODE}}}(x_0) = (-\infty, x_0], \text{if } x_0 < 0$$

$$\mathrm{Reach}_{T_{\mathrm{ODE}}}(x_0) = 0, \text{if } x_0 = 0$$

Safety

For a transition system $T = (S, \Sigma, \rightarrow)$ with initial state in S_0, let $B \subset S$ denote a "bad" set, i.e., a set of states that we don't want the system to enter. T is **safe** if

$$\mathrm{Reach}(S_0) \cap B = \emptyset$$

Remark

- B encodes the property to *verify*
- Verification is about verifying that the system fulfills its specification, i.e., $\mathrm{Reach}(S_0) \cap B = \emptyset$

Validating Designs

Embedded system designs can be validated with various degrees of rigour (higher is better):

- By construction: property is inherent
- By verification: property is provable syntactically
- By simulation: check behaviour for all inputs
- By intuition: property is true, I just know it is
- By assertion: property is true, wanna make something of it?
- By intimidation: don't even try to doubt whether it is true

[Pappas,2005]

Verification

- **Prove** that a system fulfill certain specification
- Based on a mathematical model and a computational tool

Decidability

- A verification problem is **decidable** if there exists an algorithm that solves it and terminates in a finite number of steps
- A verification problem is **semi-decidable** if the algorithm may not terminate, but if it does, it produces the correct result
- Verifying **safety** for a finite transition system is decidable
- Verifying safety can be decidable or semi-decidable for some more general transition systems, but is in most cases undecidable
- Verifying **safety** can be solved by computing Reach

Reach Set Computation

Reachability Algorithm to compute $\text{Reach}(S_0)$
$\text{Reach}_{-1} := \emptyset$, $\text{Reach}_0 := S_0$, $i := 0$
`while` $\text{Reach}_i \neq \text{Reach}_{i-1}$ `do`
    $\text{Reach}_{i+1} := \text{Reach}_i \cup \text{Post}(\text{Reach}_i)$
    $i := i + 1$
`end`

- If the algorithm terminates, then $\text{Reach}(S_0) := \text{Reach}_i$
- If the state space is finite, the algorithm terminates in a finite number of steps
- The algorithm does not necessarily terminate for general transition systems
- $s \xrightarrow{e} s'$ is simple, but $s \xrightarrow{t} s'$ in general hard

Invariance

Consider $T = (S, \Sigma, \rightarrow)$

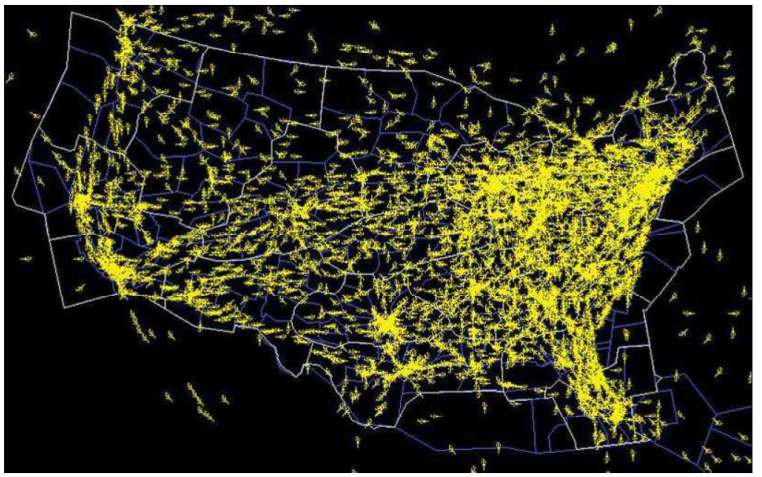
- $S_0 \subset S$ is invariant if $\text{Reach}(S_0) = S_0$, ie, all states starting from S_0 remain in S_0
- Reachability algorithm may help in searching for invariant sets
- Finding invariant sets might help in Reach computation

Over-Approximation of Reach Set

- It is often hard to calculate Reach exactly
- Compute an over-approximation $A \supset$ Reach instead
- Note that $A \cap B = \emptyset$ implies that Reach $\cap B = \emptyset$, so safety is guaranteed if the algorithm based on over-approximation terminates

Example
Derive an over-approximation of Reach for $\dot{x} = f(x)$. Assume that B is associated to instability.

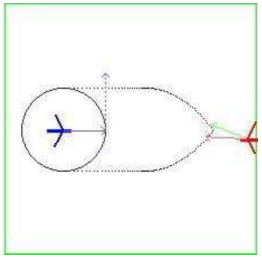
Example: Air Traffic Control [Tomlin]

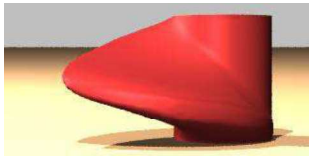


Reachability Analysis

- Autopilots of modern jets are highly automated and have hundreds of flight modes
- Important to verify safe operation of autopilot
- Enables future architectures for free flight

Reach set computations for collision avoidance:





Beyond safety and invariance

- Current trend: define more complicated specifications using formal languages
- Linear Temporal Logic(LTL), Computational Tree Logic(CTL) define specification as a formula φ
- Basic verification problem

$$T \models \varphi$$

- Approach to solution: Model Checking

Specifications through LTL

- Assign atomic propositions (Π) to states S through labeling functions $L : S \to 2^{\Pi}$
- Π represent things that can happen/observed at each state
- Transition system expanded to $T = (S, \Sigma, \to, \Pi, L, S_0, S_F)$
- LTL expresses specifications along sequences of Π using temporal (eg, always, eventually) and boolean (and, or) operators
- Example spec: "eventually go to Region A and always avoid Region B"

Specifications through LTL

- An LTL formula φ over the set of APs Π is defined inductively as follows: (i) every $\pi \in \Pi$ is a formula; (ii) if φ_1, φ_2 are formulas, then $\varphi_1 \vee \varphi_2$ (or), $\varphi_1 \mathrm{U} \varphi_2$ (until), $\neg\varphi_1$ (negation), $\mathrm{X}\varphi_1$ (next), $\mathrm{G}\varphi_1$ (always), $\mathrm{F}\varphi_1$ (eventually) are formulas
- LTL formulas have also automata-based reprentations (ref. Lec 13)
- Combination with transition system T representing the plant reveals which formulas can be fulfilled by T

LTL Model Checking

Given **finite** transition system T and LTL formula φ, determine if T fulfills φ (ie, whether $T \models \varphi$)

- Model checker returns a counterexample if there exists one
- Computations based on automata-based representation of φ (product operator with T)
- Otherwise it returns that the spec is verified
- Off-the-self model checkers exist
- Can be used for control synthesis using $\neg\varphi$

Next Lecture

Hybrid automata

- Motivate hybrid systems
- Hybrid automata as models of hybrid systems
- Dynamical properties of hybrid automata

EL2450 Hybrid and Embedded Control

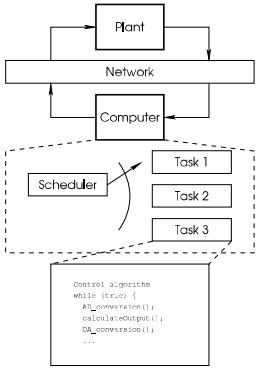
Lecture 9: Hybrid automata

- Hybrid automata as models of hybrid systems
- Dynamical properties of hybrid automata

Today's Goal

You should be able to

- specify a hybrid automaton
- define system evolution semantics
- analyze existence, uniqueness and Zenoness of solutions



← Time-triggered system

← Event-triggered system

← Event-triggered system

Hybrid Dynamics

- Dynamics are essential in reach set computations and verification process: finiteness is lost
- Need to consider all causes of system evolution!
- Mixing time- and event-triggered dynamics lead to **hybrid** dynamics
- What are the dynamics within the discrete states? What might cause discrete transitions?
- Hybrid systems are a particular class of transition systems

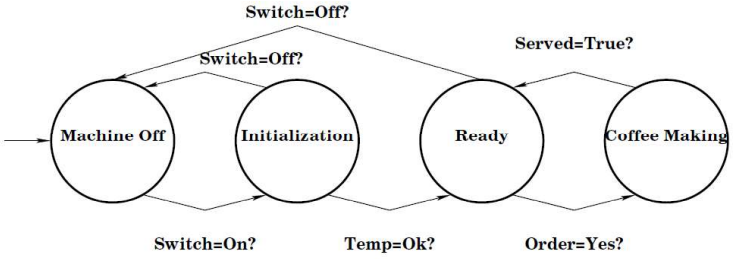
Espresso Machine Example

Task: Design the control system for an automatic espresso machine

Input and Outputs

- What are the relations between inputs and output?

Discrete Event System



- High-level abstraction
- What is in each discrete state?

Initialization

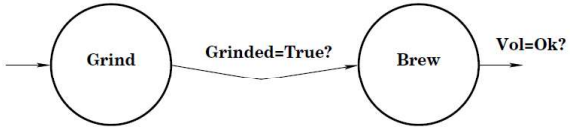
- Start initialization state when machine is switched on
- End the initialization state either
  - When machine is switched off, or
  - When the water temperature T is over a suitable temperature T^*

Ready

- Control T around T^*
- Wait for further a coffee order or that the machine is switched off

Coffee Making

Coffee making can be split into two states:



- In Grind state, coffee beans are prepared for brewing
- In Brew state, the steam is passed through the coffee
  - Pressure P and temperature T are continuously controlled
  - State terminated when coffee volume V is over a desired volume V^*

Automatic Gearbox Control Example

Task: Design the control system for an automatic gearbox

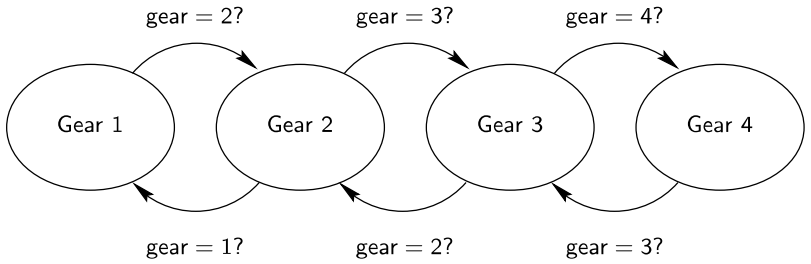
x_1 is the longitudinal position of the car and x_2 its velocity
The dynamics (for a normalized car) can be written as

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \alpha_{\text{gear}}(x_2)u$$

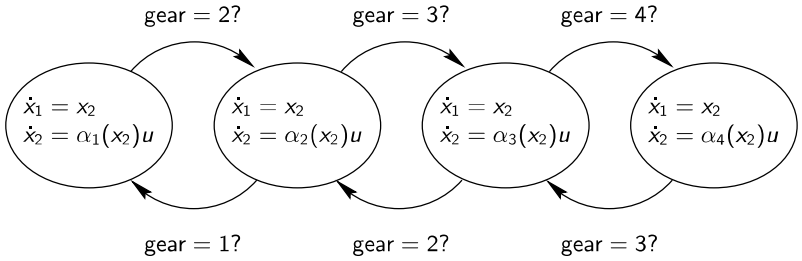
where u corresponds to the throttle position and $\alpha_{\text{gear}}(\cdot)$ to the efficiency of a specific gear

- $u \in [0, u_{\text{max}}]$ is a real-valued control
- gear $\in \{1, 2, 3, 4\}$ is an integer-valued control

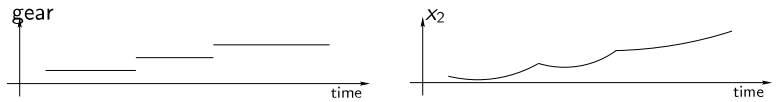
Discrete Event Model



Hybrid System Model



Typical solutions:

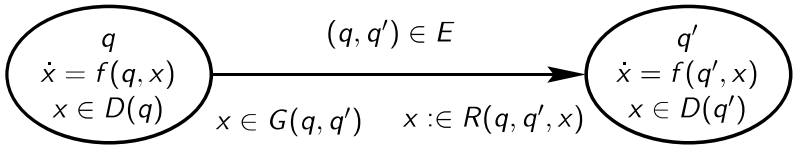


How choose controls u and gear in a suitable way?

Hybrid Automaton

- **Hybrid automaton** is a formal model of a hybrid system
- It defines the evolution of the hybrid system state

Hybrid Automaton $H = (Q, X, \mathsf{Init}, f, D, E, G, R)$

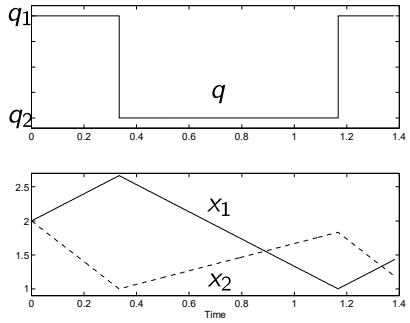


- Q discrete state space and X continuous state space
- $\mathsf{Init} \subseteq Q \times X$ initial states
- $f : Q \times X \to X$ vector fields
- $D : Q \to 2^X$ domains
- $E \subset Q \times Q$ edges
- $G : E \to 2^X$ guards
- $R : E \times X \to 2^X$ resets

Solution of Hybrid Automaton

A **solution** $\chi = (\tau, q, x)$ of H consists of

- **Time trajectory** τ: time line on which the solution is defined
- **State trajectory** (q, x): state evolution (defined on τ) of the hybrid automaton



Time Trajectory τ

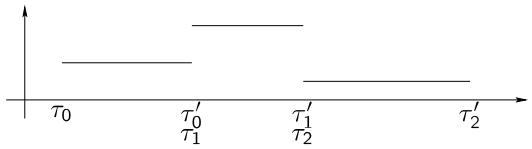
A sequence of (time) intervals

$$\tau = \{I_i\}_{i=0}^{N}$$

such that

- $I_i = [\tau_i, \tau_i')$ for all $i < N$;
- if $N < \infty$, then either $I_N = [\tau_N, \tau_N']$, or $I_N = [\tau_N, \tau_N')$; and
- $\tau_i \leq \tau_i' = \tau_{i+1}$ for all i.

Notation: $\langle \tau \rangle = \{0, 1, \ldots, N\}$

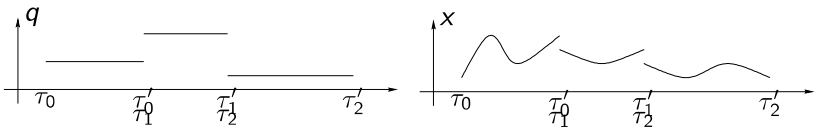


Solution $\chi = (\tau, q, x)$

Solutions accepted by H:

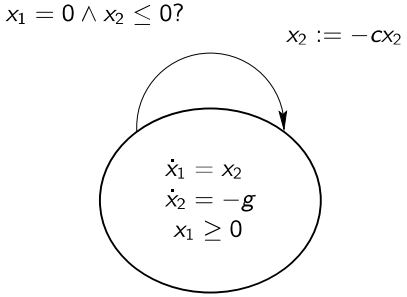
$\tau = \{I_i\}_{i=0}^{N}$, $q : \langle \tau \rangle \to Q$, $x = \{x^i : i \in \langle \tau \rangle\}$, $x^i : I_i \to X$ such that

- **Initialization:** $(q(0), x^0(0)) \in \mathsf{Init}$,
- **Time-driven:** for all $t \in [\tau_i, \tau_i')$, $\dot{x}^i(t) = f\big(q(i), x^i(t)\big)$ and $x^i(t) \in D(q(i))$
- **Event-driven:** for all $i \in \langle \tau \rangle \setminus \{N\}$, $e = (q(i), q(i+1)) \in E$, $x^i(\tau_i') \in G(e)$, and $x^{i+1}(\tau_{i+1}) \in R(e, x^i(\tau_i'))$



Example: Bouncing Ball

A ball that loses a fraction of its energy at each bounce

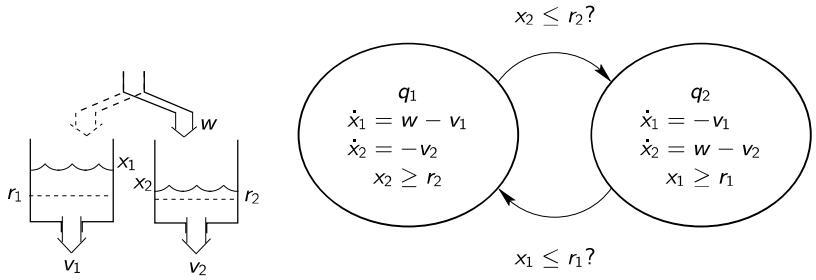


Examples: Differential Equation and Automaton

- A continuous-time system $\dot{x} = f(x)$ can be represented as a hybrid automaton with a single discrete state
- A discrete-event system can be represented as a hybrid automaton with no continuous dynamics

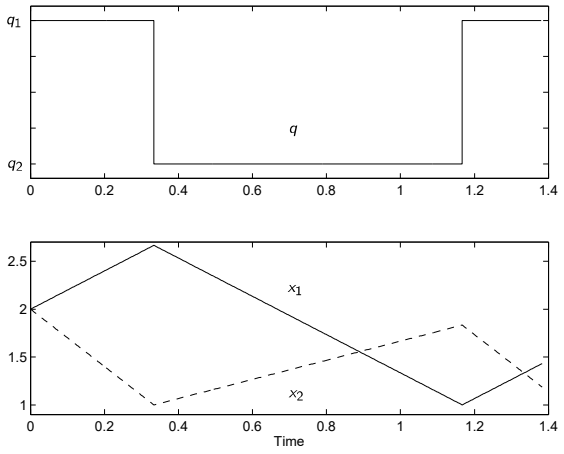
Example: Water Tank System

Control objective is to keep water volumes above r_1 and r_2 by switching the inflow



$$H = (Q, X, \mathsf{Init}, f, D, E, G, R)$$

- $Q = \{q_1, q_2\}$;
- $X = \mathbb{R}^2$;
- $\mathsf{Init} = Q \times \{x \in \mathbb{R}^2 : x_1 \ge r_1 \wedge x_2 \ge r_2\}$;
- $f(q_1, x) = (w - v_1, -v_2)$ and $f(q_2, x) = (-v_1, w - v_2)$;
- $D(q_1) = \{x \in \mathbb{R}^2 : x_2 \ge r_2\}$ and $D(q_2) = \{x \in \mathbb{R}^2 : x_1 \ge r_1\}$;
- $E = \{(q_1, q_2), (q_2, q_1)\}$;
- $G(q_1, q_2) = \{x \in \mathbb{R}^2 : x_2 \le r_2\}$ and $G(q_2, q_1) = \{x \in \mathbb{R}^2 : x_1 \le r_1\}$;
- $R(q_1, q_2, x) = R(q_2, q_1, x) = \{x\}$.

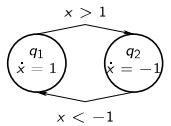


Hybrid Automaton as a Transition System

A hybrid automaton is a transition system $T_H = (S, \Sigma, \rightarrow)$ with interacting event-driven and time-driven evolution:

- $S = Q \times X$ and $(q, x) \in S$ denotes the state
- $\Sigma = \{g\} \cup \mathsf{Time}$ ($\mathsf{Time} = \{t : t \ge 0\}$) where the generators $\{g\}$ cause the discrete jumps and Time the continuous evolution
- $(q, x) \rightarrow (q', x')$ defines the event-driven and time-driven transitions

Example: $S = Q \times X$, with $Q = \{q_1, q_2\}$ and $X = \mathbb{R}$
$\Sigma = \{g_1, g_2\} \cup \mathsf{Time}$, g_1 corresponds to the event $x > 1$ and g_2 to $x < -1$



Transition Relation for Hybrid Automaton

- To each discrete state $q \in Q$, we associate a differential equation $\dot{x} = f_q(x) = f(q, x)$. Let $\phi_q(t)$ denote its solution.
- To each generator g (linked to an edge $e \in Q \times Q$), we associate a guard $G : Q \times Q \to 2^X$

The transition relation $\to$ of $T_H = (S, \Sigma, \to)$ then consists of two parts:
Time-driven: $(q, x) \xrightarrow{t} (q, y)$ provided that $x = \phi_q(0)$ and $y = \phi_q(t)$
Event-driven: $(q, x) \xrightarrow{g} (q', x')$ provided that $x \in G(q, q')$ and $x' \in R(q, q', x)$

Example: Time-driven dynamics is given by $f_{q_1} = 1$ and $f_{q_2} = -1$
Event-driven dynamics is given by $G(q_1, q_2) = \{x > 1\}$ and
$G(q_2, q_2) = \{x < -1\}$

Properties of Hybrid Automata

Liveness	For all $(q_0, x_0) \in$ Init, there exists at least one (infinite) solution from (q_0, x_0)
Determinism	For all $(q_0, x_0) \in$ Init, there exists at most one solution starting from (q_0, x_0)
Zenoness	τ infinite sequence and finite execution time: $\tau_\infty = \sum_{i=1}^{\infty}(\tau_i' - \tau_i) < \infty$
Stability	Stability of equilibria and other invariant sets
Reachability	Reachable states Reach $\subset Q \times X$

Liveness

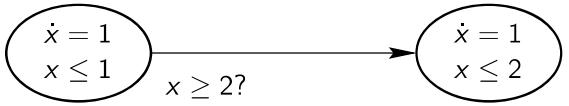
Definition
For all $(q_0, x_0) \in$ Init, there exists **at least one** (infinite) solution from (q_0, x_0)

Fact
A hybrid automaton is live if for all reachable states for which continuous evolution is impossible, a discrete transition is possible

- It is reasonable to expect that models for physical systems should be live
- If a hybrid automaton is not live, it can be due to over-simplifications in the model

Example

Let Init $= (q_1, 0)$. Then the following hybrid automaton is not live (blocking):



Determinism

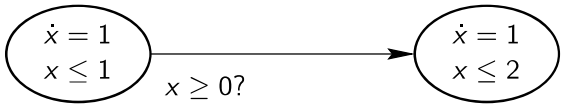
Definition
For all $(q_0, x_0) \in \mathsf{Init}$, there exists **at most one** solution starting from (q_0, x_0)

Fact
A hybrid automaton is deterministic if there is

- no choice between continuous evolution and a discrete transition, and
- a discrete transition can never lead to multiple destinations

Example

Let $\mathsf{Init} = (q_1, 0)$. Then the following hybrid automaton is non-deterministic:



Determinism (formally)

Let

$$\mathrm{Out}_H = \{(q, x) \in Q \times X : \forall \epsilon > 0, \exists t \in [0, \epsilon), \phi_q(0) = x, \phi_q(t) \notin D(q)\}$$

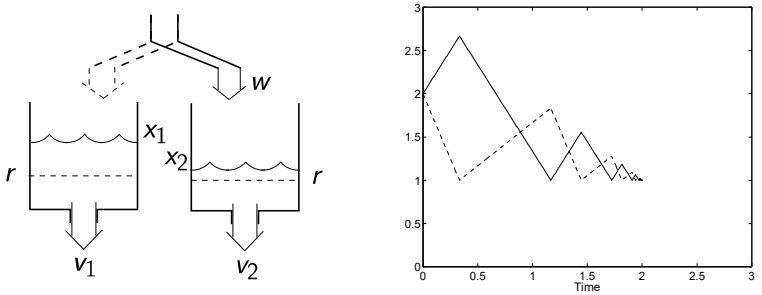
denote the set of states where continuous evolution is impossible.

Fact H is deterministic if and only if for all reachable (q, x)

- if $x \in G(q, q')$ for some $(q, q') \in E$, then $(q, x) \in \mathrm{Out}_H$
- if $(q, q'), (q, q'') \in E$ with $q' \neq q''$, then $G(q, q') \cap G(q, q'') = \emptyset$
- if $(q, q') \in E$ and $x \in G(q, q')$, then $|R(q, q', x)| \leq 1$

Zeno Solution of Hybrid Automaton

A solution $\chi = (\tau, q, x)$ is Zeno if $\tau_\infty = \sum_{i=1}^{\infty} (\tau_i' - \tau_i) < \infty$
Example—Water tank system: If $\max\{v_1, v_2\} < w < v_1 + v_2$ then $\tau_\infty = (x_1(0) + x_2(0) - 2r)/(v_1 + v_2 - w) < \infty$



Execution is not defined for $t > \tau_\infty$

Zeno of Elea (490–430 B.C.)

- Born in southern Italy
- Met Socrates in Athens 449 B.C.
- Went back to Elea and into politics
- Tortured to death

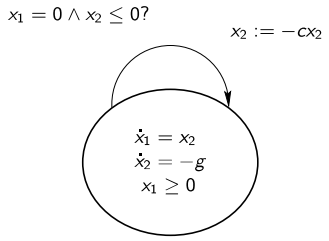
- Paradoxes "proved" that motion and time are illusions
- Led to mathematical problems not solved until 19th century

Zeno

- A solution is Zeno if it exhibits infinitely many discrete jumps in finite time
- Zeno is a truly hybrid phenomenon: it cannot be formulated for a purely discrete system without the notion of continuous time
- Zeno is due to that the model does not reflect reality with sufficient detail
- Fact: a hybrid automaton has Zeno solutions only if (Q, E) is a cyclic graph (a graph with a loop)

Example: Bouncing Ball

A ball that loses a fraction of its energy at each bounce

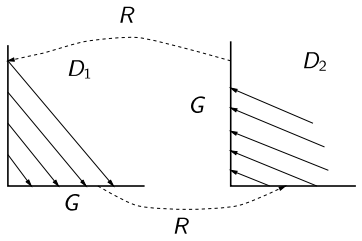


For which values of c are the solutions of the bouncing ball hybrid automaton Zeno?

Zeno State

The convergence point of a Zeno solution is denoted **Zeno state**
Zeno states lie on the intersection of guards

Example—Water tank system:



Next Lecture

Stability of hybrid systems

- Stability of hybrid systems
- Stability criteria for hybrid systems

EL2450 Hybrid and Embedded Control

Lecture 10: Stability of hybrid systems

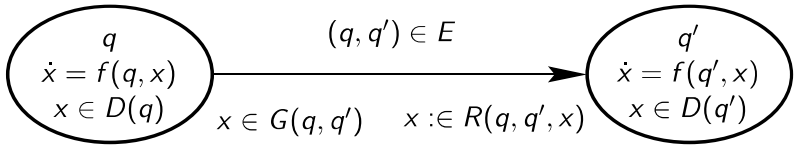
- Stability of hybrid systems
- Stability criteria for hybrid systems

Today's Goal

You should be able to

- define switched systems as a subclass of hybrid systems
- analyze stability of switched systems
- distinguish between arbitrary and constrained switching

Hybrid Automaton $H = (Q, X, \text{Init}, f, D, E, G, R)$



- Q discrete state space and X continuous state space
- $\text{Init} \subseteq Q \times X$ initial states
- $f : Q \times X \to X$ vector fields
- $D : Q \to 2^X$ domains
- $E \subset Q \times Q$ edges
- $G : E \to 2^X$ guards
- $R : E \times X \to 2^X$ resets

Switched Systems

Abstracted model of hybrid systems where continuous part is highlighted more

- $f_q, q \in Q$ family of functions
- $\dot{x} = f_q(x), q \in Q$ family of systems
- Example: $f_q(x) = A_q x$ linear switched systems

Switching signals

- Switched system $\dot{x}(t) = f_{\sigma(t)}(x(t))$ defined based on *switching signal* $\sigma : [0, \infty) \to Q$
- Discontinuities of $\sigma(t)$: switching times
- $\sigma(t) = \lim_{\tau \to t^+}$ for all $t \geq 0$.

Switching can be state- or time-dependent

Switched Systems

Let Ω_q, $q = 1, \ldots, m$ denote a partition of the continuous state space $\mathbb{R}^n$. A state-dependent switched system is given by

$$\dot{x} = f_q(x), \qquad x \in \Omega_q$$

Example
$x \in \mathbb{R}^2$, Ω_q quadrant q, $q = 1, \ldots, 4$, and

$$\dot{x} = A_q x$$
$$x \in \Omega_q$$

Switched System as Hybrid Automaton

$$\dot{x} = f_q(x), \qquad x \in \Omega_q$$

corresponds to the hybrid automaton

- $Q = \{1, \ldots, m\}$, $X = \mathbb{R}^n$, Init $\subset \{q\} \times \Omega_q$
- $f(q, x) = f_q(x)$
- $D(q) = \Omega_q$
- $(q, q') \in E$ if $D(q)$ to $D(q')$ are "neighbors" (i.e., $\overline{D(q)} \cap \overline{D(q')} \neq \emptyset$) and there are solutions that go from $D(q)$ to $D(q')$
- $G(q, q') = \overline{D(q)} \cap \overline{D(q')}$
- $R(q, q', x) = \{x\}$

Stability

A solution x^* of a **switched system** is stable if for all $\epsilon > 0$, there exists $\delta = \delta(\epsilon) > 0$ such that for all solutions x

$$\|x(0) - x^*(0)\| < \delta \quad \Rightarrow \quad \|x(t) - x^*(t)\| < \epsilon, \quad \forall t > 0$$

The solution is *asymptotically stable* if it is stable and

$$\|x(0) - x^*(0)\| < \delta \quad \Rightarrow \quad \|x(t) - x^*(t)\| \to 0, \text{ as } t \to \infty$$

- Can be generalized to hybrid automata
- Assumes existence of solutions that expand to infinite time (Zeno is excluded)

Lyapunov's Second Method

Let $x^* = 0$ be an equilibrium point of $\dot{x} = f(x)$. If there exists a function $V : \mathbb{R}^n \to \mathbb{R}$ such that

$$V(0) = 0$$
$$V(x) > 0, \ \ \forall x \in \mathbb{R}^n \setminus \{0\}$$
$$\dot{V}(x) \leq (<) 0, \ \ \forall x \in \mathbb{R}^n,$$

then x^* is (asymptotically) stable

Lyapunov Function for Linear System

Real $\lambda_i(A) < 0$ for all i if and only if for every positive definite $Q = Q^T$ there exists a positive definite $P = P^T$ such that

$$PA + A^T P = -Q$$

A Lyapunov function for a linear system

$$\dot{x} = Ax$$

is given by

$$V(x) = x^T P x$$

In particular,

$$\dot{V}(x) = x^T P \dot{x} + \dot{x}^T P x = x^T (PA + A^T P) x = -x^T Q x < 0$$

Example

$$\dot{x} = A_1 x = \begin{pmatrix} -1 & 10 \\ -100 & -1 \end{pmatrix} x$$

Then,

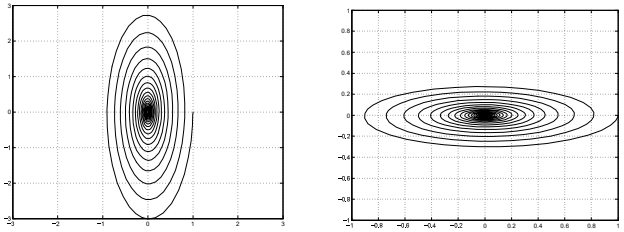
$$P = [\text{lyap in Matlab}] = \begin{pmatrix} 0.2752 & -0.0225 \\ -0.0225 & 2.7478 \end{pmatrix}$$

solves the Lyapunov equation $A_1 P + P A_1^T = -I$. Then, $V = x^T P x$ fulfills the three conditions in the Lyapunov theorem (check!). Hence, $x^* = 0$ is stable.

Note that $\lambda(A_1) = -1 \pm i10\sqrt{10}$

$$\dot{x} = A_1 x = \begin{pmatrix} -1 & 10 \\ -100 & -1 \end{pmatrix} x, \qquad \dot{x} = A_2 x = \begin{pmatrix} -1 & 100 \\ -10 & -1 \end{pmatrix} x$$

have the following phase portraits:

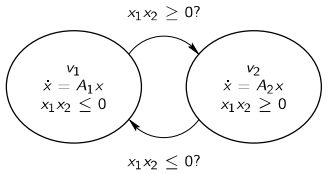


Note

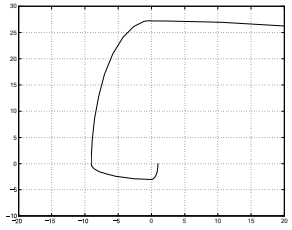
- Both systems are stable with $\lambda(A_j) = -1 \pm i10\sqrt{10}$, $j = 1, 2$
- $\exists$ Lyapunov function for each system
- What can we say about stability of linear switched system based on stability of individual subsystems?
- Answer: not much...

Example: Stable+Stable=Unstable

Consider switched system corresponding to hybrid automaton:

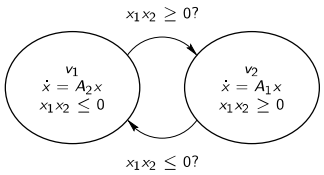


Even if A_1 and A_2 are stable, the switched system is unstable:

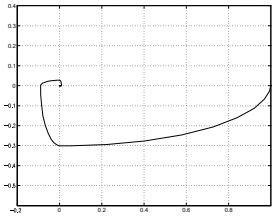


Example: Stable+Stable=Stable

Let A_1 and A_2 change place:



Then, also the switched system is stable:



Some Conclusions

- Arbitrary switching ($\sigma(t)$ can be chosen arbitrary) may destabilize a switched system even if all subsystems are stable
- Even if all subsystems are unstable, it may be still possible to stabilize the switched system by constraining the switching sequence appropriately
- Stability under arbitrary switching
- Stability under constrained switching. How to define switching sequence appropriately?

Arbitrary Switching

- In some cases σ can be chosen arbitrary and still stabilize the system, ie, the switched system is stable for any σ
- Assumption: All subsystems (asymptotically) stable. Necessary condition, why?
- Can be analyzed through Lyapunov method
- Idea: consider a single energy function for the whole system and for all possible switching sequences

Common Lyapunov Function

Consider the system

$$\dot{x} = A_\sigma x$$

where $\sigma : [0, \infty) \to \{1, \ldots, m\}$ is an arbitrary switching sequence. If there exists $P, Q > 0$ (positive definite), such that

$$PA_q + A_q^T P \leq -Q, \qquad q = 1, \ldots, m$$

then the origin is asymptotically stable

- $V(x) = x^T Px$ is a common quadratic Lyapunov function for all systems $\dot{x} = A_q x$
- Equivalent condition: there exists $P, Q_q > 0$ such that $PA_q + A_q^T P = -Q_q, \qquad q = 1, \ldots, m$

Commuting System Matrices

Consider the system $\dot{x} = A_\sigma x$, where $\sigma : [0, \infty) \to \{1, \dots, m\}$ is an arbitrary switching sequence. If all A_q are stable and

$$A_k A_\ell = A_\ell A_k, \qquad k, \ell \in \{1, \dots, m\}$$

then the origin is stable.

Proof for $m = 2$: If $A_1 A_2 = A_2 A_1$ then $\exp A_1 \exp A_2 = \exp A_2 \exp A_1$ (why?). Then, for time trajectory τ and $t \in [\tau_i, \tau_i']$,

$$\begin{aligned} x(t) &= \exp[A_1(t - \tau_i)] \exp[A_2(\tau_{i-1}' - \tau_{i-1})] \cdots \exp[A_1(\tau_0' - \tau_0)] x_0 \\ &= \exp\left[A_1[(t - \tau_i) + \cdots + (\tau_0' - \tau_0)]\right] \\ &\quad \times \exp\left[A_2[(\tau_{i-1}' - \tau_{i-1}) + \cdots + (\tau_1' - \tau_1)]\right] x_0 \end{aligned}$$

Stability follows from that A_1 and A_2 are stable.

Switched Nonlinear Systems

Consider the system

$$\dot{x} = f_\sigma(x)$$

where $\sigma : [0, \infty) \to \{1, \dots, m\}$ is an arbitrary switching sequence and $f_q(0) = 0, q \in \{1, \dots, m\}$. If there exists a positive definite $V : \mathbb{R}^n \to \mathbb{R}$, such that

$$\frac{\partial V}{\partial x} f_q(x) < 0, \forall x \neq 0, \forall q \in \{1, \dots, m\}$$

then the origin is asymptotically stable

Commutation Relations for Switched Nonlinear Systems

- The Lie bracket of two vector fields f_1, f_2 is defined as

$$[f_1, f_2](x) := \frac{\partial f_2(x)}{\partial x} f_1(x) - \frac{\partial f_1(x)}{\partial x} f_2(x)$$

  We say that the vector fields commute if their Lie bracket is identically zero.
- Assume that $f_q, q \in \{1, \dots, m\}$ is a finite set of commuting vector fields and all subsystems have an asymptotically stable equilibium at the origin. Then the switched system $\dot{x} = f_\sigma(x)$ is asymptotically stable for any switching sequence.

Constrained Switching

- Stability under arbitrary switching does not hold in general (see earlier example).
- Choice of σ is constrained due to state and/or time dependencies. How to examine stability?
- How to derive appropriate σ that guarantees stability?

Multiple Lyapunov Functions

Suppose $x^* = 0$ is an equilibrium of each mode $q = 1, \dots, m$ of the switched system

$$\dot{x} = f_\sigma(x), \quad \sigma : [0, \infty) \to \{1, \dots, m\}$$

If there exist functions $V_1, \dots, V_m$ such that

$$V_q(0) = 0, \quad V_q(x) > 0, \ \ \forall x \in \mathbb{R}^n \setminus \{0\}$$
$$\dot{V}_q(x(t)) \leq 0, \quad \text{whenever } \sigma(t) = q$$

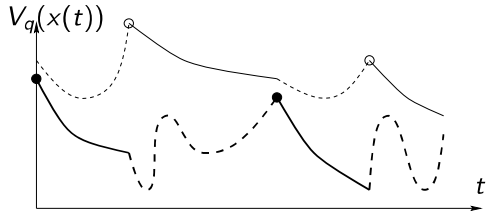
and the sequences $\{V_q(x(\tau_{i_q}))\}$, $q = 1, \dots, m$ are non-increasing, where τ_{i_q} are the time instances when mode q becomes active, then x^* is stable.

Example

Let the origin be a stable equilibrium point for

$$\dot{x} = f_q(x), \qquad x \in \Omega_q, \qquad q = 1, 2$$

Below, $V_1(x(t))$ and $V_2(x(t))$ are shown. The active parts are solid. The sequences $\{V_q(x(\tau_{i_q}))\}$, $q = 1, 2$, are indicated

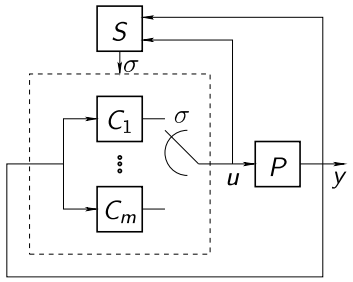


Notes on Multiple Lyapunov Functions

- Strict inequalities imply asymptotic stability for switched linear systems.
- State-dependent switching: Multiple Lyapunov functions that have the same value on the switching surface can be used.
- Alternatively, one can search for one Lyapunov function that behaves appropriately in regions of interest (ie, decreases at regions where corresponding subsystem is active and remains continuous at switching surfaces). OBS!! NOT a common Lyapunov function.
- Computational methods for both exist.

Supervisory Control

- How choose switching $\sigma = \sigma(t)$ such that $\dot{x} = f_\sigma(x)$ has desired property?
- Let a **supervisor** decide on which controller should be active through the switching signal $\sigma : [0, \infty) \to \{1, \dots, m\}$



A Stabilizing Switching Sequence

Suppose there exist $\mu_q \geq 0$, $q \in Q$ and $\sum_{q=1}^{m} \mu_q = 1$, such that $A = \sum_{q=1}^{m} \mu_q A_q$ is stable. Then, a stabilizing switching sequence $\sigma : [0, \infty) \to Q := \{1, \ldots, m\}$ for

$$\dot{x} = A_\sigma x,$$

is given by

$$\sigma(x(t)) = \arg\min_{q \in Q} \{x^T(t)(A_q^T P + P A_q)x(t)\}$$

where $P > 0$ is the solution to $A^T P + PA = -I$.

Proof: Follows from that $\sum_{q=1}^{m} \mu_q x^T (A_q^T P + P A_q)x < 0$ and $\mu_q \geq 0$, which gives $x^T (A_{\sigma(x)}^T P + P A_{\sigma(x)})x < 0$ for any $x \neq 0$.

Notes

- Intuition: if $\exists$ stable convex combination of A_q then we can always find a mode for which the energy is decreasing at the current state.
- $V = x^T P x$ not a Common Lyapunov Function for the previous result. Why?
- Example application: Apply the previous result to a (networked) control system where either no control or a state-feedback control is applied, so that $\dot{x} = Ax$ corresponds to the open-loop system and $\dot{x} = (A - BL)x$ corresponds to the closed-loop system.

Next Lecture

Control of hybrid systems

- Supervisory control
- Stabilizing switching control

EL2450 Hybrid and Embedded Control

Lecture 11: Control of hybrid systems

- Nonholonomic control
- Supervisory control
- Multi-agent control

Today's Goal

You should be able to

- identify necessary conditions for continuous feedback stabilization
- design a hybrid controller for a class of nonholonomic systems
- design supervisory controllers for linear systems
- design controllers for multi-agent systems under limited sensing and communication

Motivations for hybrid Control

Some motivations for hybrid control

- Systems that are not stabilizable by continuous feedback (e.g., nonholonomic systems)
- Systems with large modelling uncertainty (e.g., linear systems with parametric uncertainty)
- Systems with sensing and communication constraints (e.g., quantized control, large scale systems with sensing limitations)

Brockett's necessary condition for continuous stabilization

Let $\dot{x} = f(x,u)$, $x \in \mathbb{R}^n, u \in \mathbb{R}^m$ and suppose $u = k(x), k(0) = 0$ is continuous and makes $x = 0$ (locally) asymptotically stable for $\dot{x} = f(x, k(x))$. Then the image of every neighborhood of $(0,0)$ in $\mathbb{R}^n \times \mathbb{R}^m$ under $(x,u) \mapsto f(x,u)$ contains some neighborhood of 0 in $\mathbb{R}^n$.

- Intuition: starting near zero and applying small controls, we must be able to move in all directions
- Systems with certain constraints in motion do not satisfy this condition

Nonholonomic systems

Systems of the form

$$\dot{x} = f(x,u) = \sum_{i=1}^{m} g_i(x)u_i = G(x)u$$

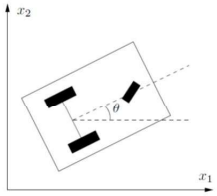
where $G \in \mathbb{R}^{n\times m}$ and $m < n$ are called nonholonomic: constraints involving both position and velocity.

Corollary of Brockett's condition: nonholonomic systems with rank $G(0) = m < n$ cannot be asymptotically stabilized by continuous feedback.

Unicycle model

Unicycles are a popular model for wheeled mobile robots

$$\begin{aligned}\dot{x}_1 &= u_1 \cos\theta\\ \dot{x}_2 &= u_1 \sin\theta\\ \dot{\theta} &= u_2\end{aligned}$$



- u_1, u_2 linear and angular velocities, θ orientation wrt x_1 axis
- The robot cannot move sideways (nonholonomic constraint)
- Asymptotic stabilization equivalent to parking at the origin and align with x_1 axis

Nonholonomic integrator

Using the transformation (that preserves the origin)

$$\begin{aligned}x &= x_1\cos\theta + x_2\sin\theta\\ y &= \theta\\ z &= 2(x_1\sin\theta - x_2\cos\theta) - \theta(x_1\cos\theta + x_2\sin\theta)\\ u &= u_1 - u_2(x_1\sin\theta - x_2\cos\theta)\\ v &= u_2\end{aligned}$$

we get the equivalent form

$$\begin{aligned}\dot{x} &= u\\ \dot{y} &= v\\ \dot{z} &= xv - yu\end{aligned}$$

known as Brockett's *nonholonomic integrator*

A further transformation

With a further transformation

$$\begin{aligned}x &= r\cos\psi\\ y &= r\sin\psi\\ u &= \bar{u}\cos\psi - \bar{v}\sin\psi\\ v &= \bar{u}\sin\psi + \bar{v}\cos\psi\end{aligned}$$

we get (in cylindrical coordinates)

$$\begin{aligned}\dot{r} &= \bar{u}\\ \dot{\psi} &= \frac{\bar{v}}{r}\\ \dot{z} &= r\bar{v}\end{aligned}$$

which holds in this case only for $x^2 + y^2 = r^2 \neq 0$

A hybrid controller for the nonholonomic integrator

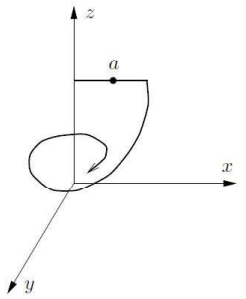
For $r(0) \neq 0$ it can be shown that if $\bar{u} = -r^2$, $\bar{v} = -z$, then $r, z \to 0$ and that $r(t) \neq 0$ for all t. Thus $x, y, z \to 0$ and thus $x_1, x_2, \theta \to 0$.

What happens when $r(0) = 0$? Apply switching logic as follows:

- Apply control that takes state away from z-axis (e.g., $u = v = 1$ or $u = v = z(0)$)
- At certain time T, switch to $\bar{u} = -r^2$, $\bar{v} = -z$
- Two discrete states and the switch between them happens at T
- Then based on the previous result the states go to the origin

Typical trajectory

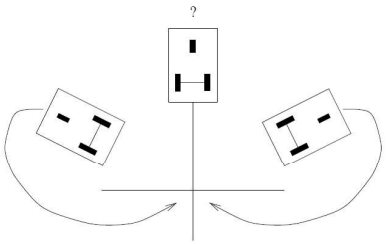
- A typical trajectory of the switching logic is on the right
- At point a either discrete node can be active



Intuition with respect to parking example

Unicycle stabilization related to parking at the origin.

- Ambiguity at $r(0) = 0$ leads to discontinuous controllers
- Related to need for logical choice at certain initial conditions
- How to rotate when starting at z-axis?

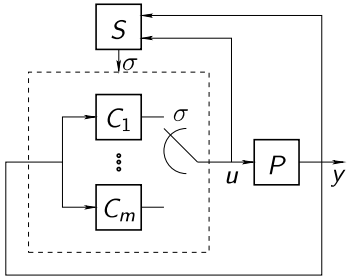


A hybrid controller for the nonholonomic integrator

- Truly hybrid controller since it depends both on state value and discrete event
- Different switching controllers for nonholonomic systems exist
- Alternative to discontinuous feedback: time-varying feedback
- Previous result can be robustified by replacing $r(0) = 0$ condition with $|r(0)| \leq \epsilon$ for some $\epsilon > 0$

Supervisory Control

- How choose switching $\sigma = \sigma(t)$ such that P has desired property?
- Let a **supervisor** decide on which controller should be active through the switching signal $\sigma : [0, \infty) \to \{1, \dots, m\}$



Supervisory Control

- Motivated by applications where parametric uncertainty of plant/controller belongs to a compact or finite set.
- In contrast to adaptive control where parametric uncertainty evolves in a continuum.
- Example: supervisory control for linear systems with parametric uncertainty. Consider the plant

$$\dot{x} = A_{p^*} x + B_{p^*} u$$
$$y = C_{p^*} x$$

where $A_p, B_p, C_p, p \in \mathcal{P}$ a given finite family of matrices, and the real $p^* \in \mathcal{P}$ unknown.

Linear supervisory Control

Define for each p the family of observers

$$\dot{x}_p = (A_p + K_p C_p) x_p + B_p u - K_p y$$
$$y_p = C_p x_p$$

and corresponding controllers $u_p = F_p x_p$, $p \in \mathcal{P}$. Key assumption: assume that K_p, F_p can be chosen such that $A_p + K_p C_p$ and $A_p + B_p F_p$ are stable.

Switching logic can be related to estimation errors of each observer, $e_p = y_p - y$, $p \in \mathcal{P}$. Define $\mu_p(t) = \int_0^t |e_p(t)|^2 dt$.

Switching logic

Design $\sigma : [0, \infty) \to \mathcal{P}$ for the control $u(t) = F_{\sigma(t)} x_{\sigma(t)}$.

Hysteresis switching logic:

- Set $\sigma(0) = \arg\min_{p \in \mathcal{P}} \mu_p(0)$
- For current value of σ equal to q, keep σ constant until

$$\min_{p \in \mathcal{P}} \mu_p(t) + h \leq \mu_q(t)$$

- Then set $\sigma(t) = \arg\min_{p \in \mathcal{P}} \mu_p(t)$

Switching logic

Results

- Result: there exists a time T where $\sigma(t) = q^*$, for all $t \geq T$ and both x and x_{q^*} go to zero.
- Proof relies on that μ_p nondecreasing and that e_{p^*} goes to zero irrespective of u, and thus μ_{p^*} remains bounded.

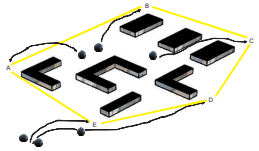
Notes on supervisory control

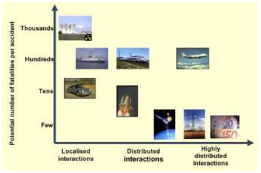
- Previous example rather special
- More advanced methods exist
- Hysteresis constant h reduces switching frequency and excludes Zeno
- Known limitations of standard adaptive control can be overcome by supervisory control

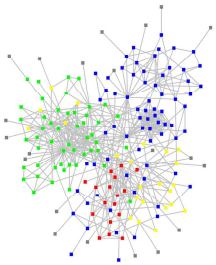
Multi-agent systems: motivation

- Many applications involve distributed control over different entities ("agents")
- Multi-robot/vehicle coordination
- Sensor networks
- Social networks
- Bio-inspired coordination
- Local control specs arise due to limitations in sensing and/or communication

Some nice figures





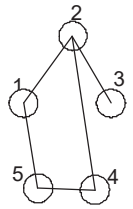


Limited Sensing and Communication aspects

- Limited Sensing: Vision based sensors, range sensors (sonars, laser scanners,...)
- Limited Communication: communication channel, bandwith, coding,...
- Limitations in communication/sensing do now allow each agent to communicate with everyone else

Graph theoretic approach

- Modelling of limitations through graphs



$G = (V, E)$

- Agents are the vertices $V = \{1, \ldots, N\}$
- Edges $E \subset V \times V$ are pairs of agents that can communicate
- Neighboring set for agent i: $N_i = \{j \in V | (i, j) \in E\}$
- Connected graph: has a path between each pair of its vertices

Switching graphs

- What happens when the communication graph loses/adds edges over time?
- Time-varying sets: $N_i(t) = \{j \in V | (i, j) \in E(t)\}$
- Different $N_i(t)$ correspond to different graphs
- Different graphs correspond to different discrete nodes of a hybrid system!

The adjacency matrix and the degree matrix

- We want to associate matrices with graphs.
- Neighboring set: $N_i = \{j \in V | (i, j) \in E\}$
- Adjacency matrix

$$A = A(G) = [a_{ij}], a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if otherwise.} \end{cases}$$

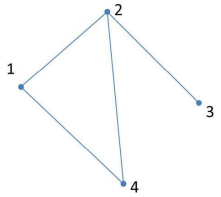
- Degree matrix

$$\Delta = \Delta(G) = \text{diag}(d_1, \ldots, d_N), d_i = \sum_j a_{ij} = |N_i|$$

The Laplacian matrix and its eigenvalues

- $L = L(G) = \Delta(G) - A(G)$
- Symmetric and positive semi-definite matrix
- Eigenvalues $0 = \lambda_1(G) \leq \lambda_2(G) \leq \ldots \leq \lambda_N(G)$
- For a connected G, $L(G)$ has a simple zero eigenvalue with the corresponding eigenvector $\mathbf{1} = [1, \ldots, 1]^T$.
- Thus $\lambda_2(G) > 0$ for a connected graph.

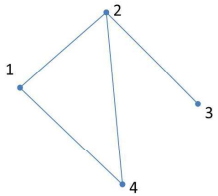
Example



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

$$\Delta = \mathrm{diag}(2, 3, 1, 2) \; L = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix}$$

Example



Laplacian eigenvalues: $0, 1, 3, 4$. If the edge between $2, 3$ is deleted, then the graph becomes disconnected, with new Laplacian $L^* = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix}$

The eigenvalues of L^* are $0, 0, 3, 3$.

Introducing interaction: the agreement paradigm

- Consider a multiple sensor network measuring temperature
- Goal: reach the best estimate of temperature given the available limited information
- It seems that the average of all measurements is the optimal choice
- Problem: design an algorithm that converges to the average for all sensors given *relative* information exchange

Agreement algorithms for static graphs

- N agents with $\dot{x}_i = u_i, i \in V = \{1, \ldots, N\}$
- Neighboring set: $N_i = \{j \in V | (i,j) \in E\}$
- Available information for i: $(x_i - x_j), j \in N_i$
- Agreement algorithm: $u_i = -\sum_{j \in N_i} (x_i - x_j)$
- $\dot{x} = -Lx$, where L is the Laplacian matrix of the graph $G = (V, E)$
- Stack vector notation $x = [x_1, \ldots, x_N]^T$

Convergence and Performance

Theorem: If G is connected then

$$x \to \mathcal{A} \triangleq \{x \in \mathbb{R}^N | x_i = x_j, \forall i, j\},$$

and the agreement point is equal to the initial average of the agents. Moreover, the worst-case convergence rate is equal to $\lambda_2(G)$.

Proof sketch

- Proof: Performance analysis using the disagreement vector
- Let $a(t) = \frac{1}{N} \sum_i x_i(t)$ denote the average of the agents' states. Then:

$$\dot{a} = \frac{1}{N} \sum_i \dot{x}_i = -\frac{1}{N} \sum_i \sum_{j \in N_i} (x_i(t) - x_j(t)) = 0$$

- Then $a(t) = a(0) = \frac{1}{N} \sum_i x_i(0) \equiv a$

Disagreement dynamics

- Define $\delta_i = x_i - a$ for each $i \in V$. δ_i represents the *disagreement* of i with respect to the desired average value.
- Decomposition of x: $x(t) = a\mathbf{1} + \delta(t)$
- Disagreement dynamics: $\dot{\delta} = -L\delta$
- Using $V = \frac{1}{2}\delta^T\delta$ as a Lyapunov function candidate it can be shown that

$$||\delta(t)|| \leq ||\delta(0)|| e^{-\lambda_2 t}$$

Proof sketch (cont.)

For any connected graph G:

$$\min_{z\neq 0:\mathbf{1}^T z=0} \frac{z^T L z}{||z||^2} = \lambda_2(G)$$

- From $\mathbf{1}^T\delta = 0$, we have $\delta^T L\delta \geq \lambda_2(G)||\delta||^2$, for all $\delta \neq 0$
- Then $\dot{V} \leq -2\lambda_2(G)V$
- Applying Comparison lemma we get $||\delta(t)|| \leq ||\delta(0)||e^{-\lambda_2 t}$

Switching graphs

- Switching control law $u_i = -\sum_{j\in N_i(t)} (x_i - x_j)$
- $G(t) \in G_c^N, \forall t \geq 0$: active graph at time t.
- G_c^N: all connected graphs with N vertices
- The system switches between possible connected graphs with N vertices. $G(t)$ remains constant between consecutive topology changes due to edge addition or loss.

Switching graphs

- We have $\dot{x}(t) = -L(G(t))x(t)$ and $a(t) = a(0) = \frac{1}{N}\sum_i x_i(0) = a$
- Then $\dot{\delta} = -L(G(t))\delta$ and $V = \frac{1}{2}\delta^T\delta$ can be used as a common Lyapunov function for the switched system (why?)
- It can be shown that

$$||\delta(t)|| \leq ||\delta(0)||e^{-\min_{G\in G_c^N}\{\lambda_2(G)\}t}$$

Relation to supervisory control

- Assume that there exists a supervisor that can activate/deactivate certain edges
- When connectivity is lost, the supervisor tries to restore it by activating appropriate new edges

Next Lecture

Verification of hybrid systems

- Reachability of hybrid systems
- Bisimulatations of hybrid systems

EL2450 Hybrid and Embedded Control

Lecture 12: Verification of hybrid systems

- Reachability for hybrid automata
- Bisimulations of hybrid systems

Today's Goal

You should be able to

- do reachability analysis for hybrid automata
- state when transitions systems are bisimilar
- find a bisimulation quotient for a transition system

Recap: Verification

- **Prove** that a system fulfills certain property
- Based on a mathematical model and a computational tool

Recap: Safety of Transition Systems

For a transition system $T = (S, \Sigma, \rightarrow, S_0)$, let $B \subseteq S$ denote a "bad" set, i.e., a set of states that we don't want the system to enter. T is **safe** if

$$\text{Reach}(S_0) \cap B = \emptyset,$$

where $\text{Reach}(S_0)$ is the set of states that can be reached from S_0 by a sequence of transitions, i.e.,

$$\text{Reach}(S_0) = \bigcup_{k=0,1,\ldots} \text{Post}^k(S_0).$$

- There is an algorithm for reach set computation
- The algorithm is guaranteed to terminate if the transition system is finite

Safety of Hybrid Automata

For a hybrid automaton $H = (Q, X, \mathsf{Init}, f, D, E, G, R)$, let $B \subseteq Q \times X$ denote a "bad" set, i.e., a set of states that we don't want the system to enter. H is **safe** if

$$\mathsf{Reach}_H(\mathsf{Init}) \cap B = \emptyset,$$

where $\mathsf{Reach}_H(\mathsf{Init}) \subseteq Q \times X$ is the set of states that can be reached by a solution of H from Init, i.e.,

$$(\bar{q}, \bar{x}) \in \mathsf{Reach}_H(\mathsf{Init})$$

if and only if there exists a solution $\chi = (\tau, q, x)$ of H such that

- $(q(0), x^0(0)) \in \mathsf{Init}$, and
- $(q(t), x^i(t)) = (\bar{q}, \bar{x})$ for some $t \in [\tau_i, \tau_i') \in \tau$.

Pre and Post for Hybrid Automata

The **predecessor operator** $\mathsf{Pre}(P), P \subseteq Q \times X$ for a hybrid automaton is

$$\mathsf{Pre}_H(P) = \{(q_p, x_p) : \exists (q, x) \in P, (q_p, x_p) \xrightarrow{t} (q, x) \text{ or } (q_p, x_p) \xrightarrow{e} (q, x)\}.$$

The **successor operator** $\mathsf{Post}(P)$ for a hybrid automaton is

$$\mathsf{Post}_H(P) = \{(q_p, x_p) : \exists (q, x) \in P, (q, x) \xrightarrow{t} (q_p, x_p) \text{ or } (q, x) \xrightarrow{e} (q_p, x_p)\}.$$

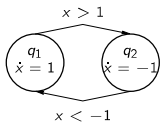
$$\mathsf{Reach}_H(\mathsf{Init}) = \bigcup_{k=0,1,\ldots} \mathsf{Post}_H^k(\mathsf{Init}).$$

Hybrid Automaton as a Transition System

A hybrid automaton is a transition system $T_H = (S, \Sigma, \rightarrow, S_0 = \mathsf{Init})$ with interacting event-driven and time-driven evolution:

- $S = Q \times X$ and $(q, x) \in S$ denotes the state
- $\Sigma = \{g\} \cup \mathsf{Time}$ where the generators $\{g\}$ causes the jumps and Time the continuous evolution
- $(q, x) \rightarrow (q', x')$ defines the event-driven and time-driven transitions

Example: $S = Q \times X$, with $Q = \{q_1, q_2\}$ and $X = \mathbb{R}$,
$\Sigma = \{g_1, g_2\} \cup \mathsf{Time}$, g_1 corresponds to the event $x > 1$ and g_2 to $x < -1$,
$S_0 = \mathit{Init} = \{(q_1, x) \mid x \geq 0\}$



Reach Set for Hybrid Automata

Reach set for a hybrid automaton H can be computed in the transition system T_H

$$\mathsf{Reach}_H(\mathsf{Init}) = \mathsf{Reach}(S_0)$$

- T_H can be infinite and thus the reach set computation algorithm does not have to terminate
- **Idea:** To simplify T_H while preserving all information about its behaviors

Example

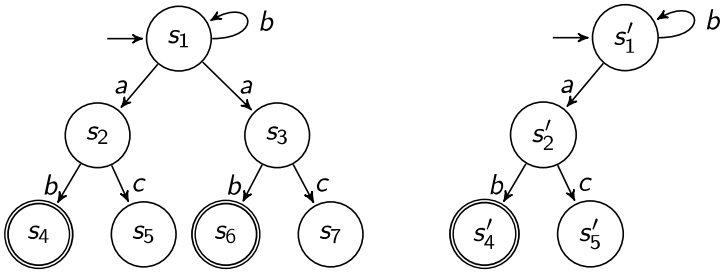


Figure: The transition systems $T_1 = (S, \Sigma, \rightarrow, S_0, S_F)$ (left) and $T'_1 = (S', \Sigma, \rightarrow', S'_0, S'_F)$ (right).

- How formalize that T_1 (left) and T'_1 (right) above have similar behaviour?

Relation

Given two sets A and B, a (binary) **relation** R from A to B is a subset of $A \times B$. We write $a\,R\,b$ if $(a, b) \in R$.

Example
$=$ is a relation from $\mathbb{N}$ to $\mathbb{N}$.

Simulation Relation

Given transition systems $T = (S, \Sigma, \rightarrow, S_0, S_F)$ and $T' = (S', \Sigma, \rightarrow', S'_0, S'_F)$. A relation $\sim \subseteq S \times S'$ is a **simulation relation** if

1. $\forall s \in S_0\ (\exists s' \in S'_0.\ s \sim s')$
2. $s \sim s' \wedge s \in S_F \Rightarrow s' \in S'_F$
3. $\forall \sigma \in \Sigma\ (s \sim s' \wedge s \xrightarrow{\sigma} r \Rightarrow \exists r' \in S'$ such that $s' \xrightarrow{\sigma} r'$ and $r \sim r')$

We say that T' **simulates** T, i.e. that T is simulated by T', denoted by $T \sim T'$

Example: Simulation

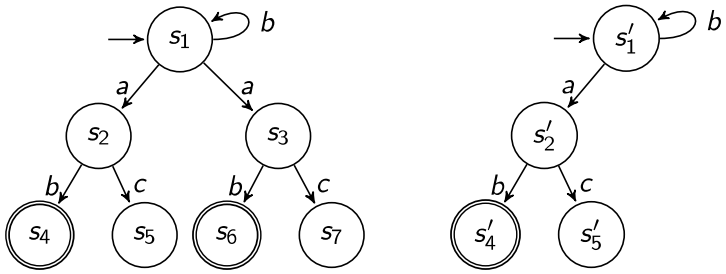


Figure: The transition systems T_1 (left) and T'_1 (right).

Derive a simulation relation for T_1 and T'_1

Example: Simulation

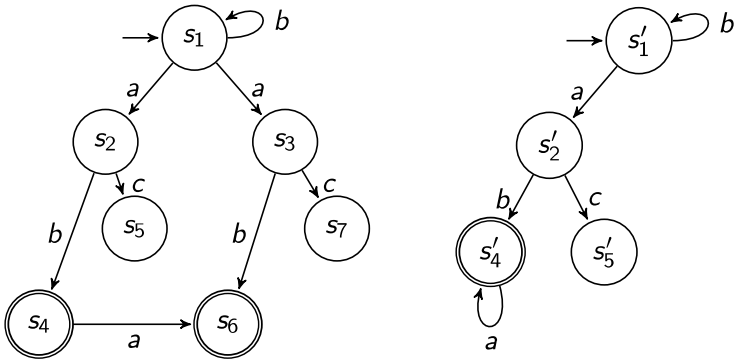


Figure: T_2 (left), T_2' (right).

Derive a simulation relation for T_2 and T_2'

Bisimulation Relation

If

- $\sim \subseteq S \times S'$ is a simulation relation from T to T' and
- $\sim' = \{(s', s) : (s, s') \in \sim\} \subseteq S' \times S$ is a simulation relation from T' to T,

then $\sim$ is a **bisimulation relation**.

- The existence of a bisimulation relation between two transition systems indicates that they are equivalent in some sense
- We say that T and T' are **bisimilar**

Examples: Bisimulation Relation

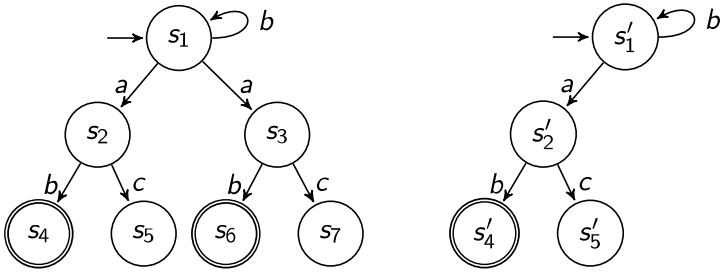


Figure: The transition systems T_1 (left) and T_1' (right) are bisimilar.

Examples: Bisimulation Relation

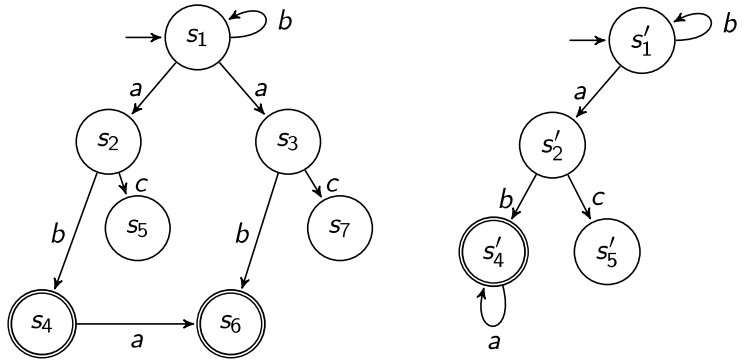


Figure: T_2 (left), T_2' (right) are not bisimilar, because T_2' simulates T_2, but T_2 does not simulate T_2'.

Examples: Bisimulation Relation

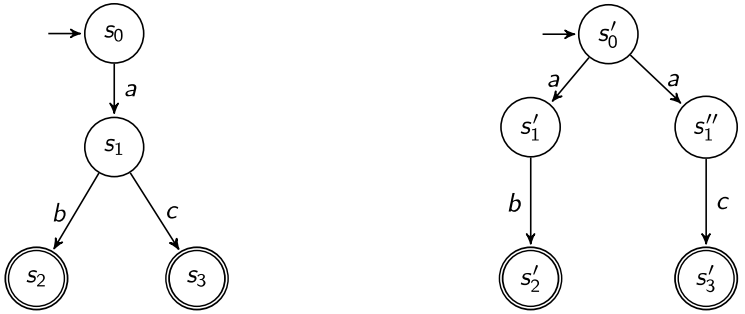


Figure: T_3 (left), T'_3 (right).

Are T_3 and T'_3 bisimilar?

Equivalence Relation

A relation $\equiv \subset S \times S$ is an **equivalence relation** if for all $s, s', s'' \in S$

1. $s \equiv s$ (reflexive)
2. $s \equiv s' \Rightarrow s' \equiv s$ (symmetric)
3. $s \equiv s'$ and $s' \equiv s'' \Rightarrow s \equiv s''$ (transitive)

Example
$=$ is an equivalence relation

Equivalence Class

Let $\equiv \subseteq S \times S$ be an equivalence relation. The **equivalence class** of $r \in S$ is defined as $[r] = \{s \in S \mid s \equiv r\}$.

Note
The equivalence classes constitute a partition of S, i.e., a collection of states $S/_{\equiv} = \{S_i\}_{i \in I}$ such that

$$S_i \cap S_j = \emptyset, \text{ for all } i \neq j$$

and

$$\bigcup_{i \in I} S_i = S$$

Quotient Transition System

Given a transition system $T = (S, \Sigma, \rightarrow, S_0, S_F)$ and a partition $S/_{\equiv} = \{S_i\}_{i \in I}$, the **quotient transition system** $\hat{T} = (\hat{S}, \Sigma, \hat{\rightarrow}, \hat{S}_0, \hat{S}_F)$ is defined as

1. $\hat{S} = S/_{\equiv}$
2. $\hat{s} \xrightarrow{\sigma} \hat{s}'$ if $\exists s, s' \in S, s \in \hat{s}, s' \in \hat{s}'$, $s \xrightarrow{\sigma} s'$
3. $\hat{s} \in \hat{S}_0$ if $\exists s \in \hat{s}$, $s \in S_0$
4. $\hat{s} \in \hat{S}_F$ if $\exists s \in \hat{s}$, $s \in S_F$

Can we find a *finite* partition such that T and $\hat{T}$ are bisimilar?

Quotient Transition System

Given an equivalence relation $\equiv \subseteq S \times S$, the relation $\sim \subseteq S \times S/_{\equiv}$ such that $\sim = \{(s, [s]) | s \in S\}$ is a bisimulation relation between $T = (S, \Sigma, \rightarrow, S_0, S_F)$ and its quotient transition system $\hat{T}$ when:

1. S_F is a union of equivalence classes.
2. For each $P \subseteq S$ that is a union of equivalence classes, $\text{Pre}_\sigma(P)$ is a union of equivalence classes, for all $\sigma \in \Sigma$.

Thus, if T and $\hat{T}$ are bisimilar, all the information related to T can be derived from the evolution in $\hat{T}$

Bisimulation Quotient Algorithm

1. initialize $S/_{\equiv} = \{\{s \mid s \in S \setminus S_F\}, \{s \mid s \in S_F\}\}$
2. while $\exists P, P' \in S/_{\equiv}, \sigma \in \Sigma$, s.t. $\emptyset \neq P \cap Pre_\sigma(P') \neq P$
   $P_1 := P \cap Pre_\sigma(P')$
   $P_2 := P \setminus Pre_\sigma(P')$
   $S/_{\equiv} := (S/_{\equiv} \setminus \{P\}) \cup \{P_1, P_2\}$
   end while

If the algorithm terminates, it computes the *coarsest* quotient.
If T is infinite, the algorithm is not guaranteed to terminate.

Example: Bisimulation Relation

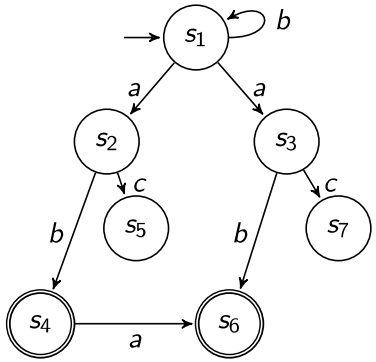


Figure: T_3

Find $\hat{T}_3$

Properties of Quotient Transition Systems

- For finite state systems an algorithm that always finds the coarsest bisimilar quotient system exists and always terminates
- Even if a transition system has infinite state space, its corresponding quotient transition system can be finite
- For time-triggered continuous-time systems (and hybrid systems) we cannot always find a finite partition

Reachability for Bisimilar Transition Systems

Given $T = (S, \Sigma, \rightarrow, S_0, S_F)$, the question whether

$$\text{Reach}(S_0) \cap S_F = \emptyset$$

in T is equivalent to the question whether

$$\text{Reach}(\hat{S}_0) \cap \hat{S}_F = \emptyset$$

in the bisimulation quotient transition system $\hat{T}$.

Safety Verification for Hybrid Automata

- A hybrid automaton $H = (Q, X, \text{Init}, f, D, E, G, R)$ and a bad set $B \subseteq Q \times X$ can be captured as a transition system $T_H = (S, \Sigma, \rightarrow, S_0, S_F = B)$, and the question whether

  $$\text{Reach}_H(\mathit{Init}) \cap B = \emptyset$$

  is equivalent to the question whether

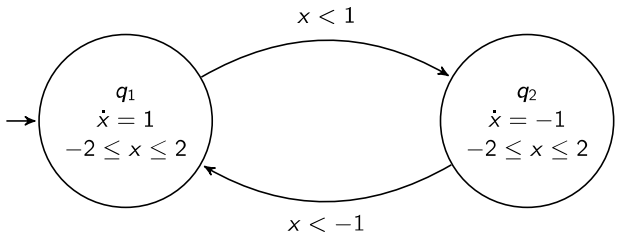
  $$\text{Reach}(S_0) \cap S_F = \emptyset$$

  in T_H, which is equivalent to the question whether

  $$\text{Reach}(\hat{S}_0) \cap \hat{S}_F = \emptyset$$

  in the bisimulation quotient transition system $\hat{T}_H$.
- If $\hat{T}_H$ is finite, then the Reach Set Computation algorithm (ref. Lec. 9) terminates in a finite number of steps.

Example: Safety Verification of a Hybrid Automaton



Initial set: $\mathit{Init} = \{(q_1, 0)\}$
Bad set: $B = \{(q_2, x) \mid x \in [1, 2]\}$

Next Lecture

- Which classes of hybrid systems admit a finite bisimulation quotient transition system?
  - Reachability for timed automata, multi-rate automata, rectangular automata
- Over-approximations

EL2450 Hybrid and Embedded Control

Lecture 13: Hybrid systems bisimulations

- Definition of timed, multi-rate, and rectangular automata
- Over-approximations of reachable sets
- Model-checking of transition systems over LTL specifications

Today's Goal

You should be able to

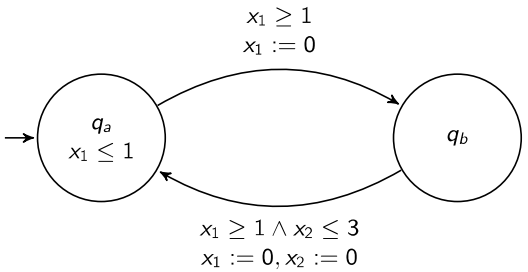
- model a timed automaton
- translate a multi-rate and rectangular automaton to a timed automaton
- understand the concept of reachability analysis for hybrid systems through reach set over-approximation
- understand the model checking process for FTS under LTL specifications

Timed Automata

Timed automata are a subclass of hybrid automata
$TA = (Q, X, \mathrm{Init}, f, D, E, G, R)$

- $Q = \{q_1, \ldots, q_m\}$, $X = \mathbb{R}^n_+$, $\mathsf{Init} \subseteq Q \times \{0\}^n$
- $f(q, x) = (1, \ldots, 1)$: "clock" dynamics
- $E \subseteq Q \times Q$
- $D(q), G(e)$ are *rectangular* sets, i.e., they are finite boolean combinations of constraints of the form $x_i \bowtie a_i$, where $\bowtie \in \{<, \leq, =, \geq, >\}$, and a_i is a positive integer.
- $R(e, x) = \{y\}$, where $y_i = 0$ or $y_i = x_i$ for all $1 \leq i \leq n$.

Example: Timed Automaton

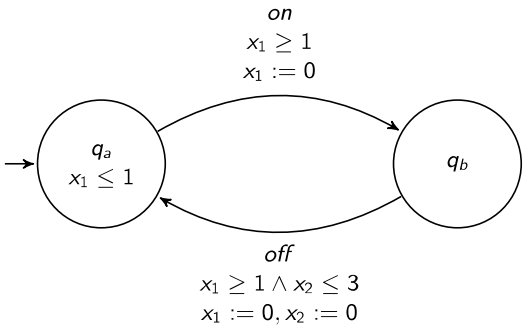


Labeled Timed Automata

Labeled timed automata are a subclass of hybrid automata $TA = (Q, X, \text{Init}, f, Act, D, E, G, R)$

- $Q = \{q_1, \ldots, q_m\}$, $X = \mathbb{R}^n_+$, $\text{Init} \subseteq Q \times \{0\}^n$
- $f(q, x) = (1, \ldots, 1)$: "clock" dynamics
- Act **is the set of events**
- $E \subseteq Q \times Act \times Q$
- $D(q), G(e)$ are *rectangular* sets, i.e., they are finite boolean combinations of constraints of the form $x_i \bowtie a_i$, where $\bowtie \in \{<, \leq, =, \geq, >\}$, and a_i is a positive integer.
- $R(e, x) = \{y\}$, where $y_i = 0$ or $y_i = x_i$ for all $1 \leq i \leq n$.

Example: Labeled Timed Automaton



Timed Automata as Transition Systems

A timed automaton $TA = (Q, X, \text{Init}, f, Act, D, E, G, R)$ can be interpreted a transition system $T_{TA} = (S, \Sigma, \rightarrow, S_0 = \text{Init})$:

- $S = Q \times X$ and $(q, x) \in S$ denotes the state
- $\Sigma = Act \cup \text{Time}$ where the generators Act are the event names and Time the continuous evolution
- $(q, x) \xrightarrow{\sigma} (q', y)$ for $\sigma \in Act$ if
  - there exists $(q, \sigma, q') \in E$, and
  - x satisfies the guard $G(q, \sigma, q')$, $\{y\} = R((q, \sigma, q'), x)$, and y satisfies the domain $D(q')$.
- $(q, x) \xrightarrow{\text{Time}} (q, x')$ if $x' = x + \text{Time}$, and x' satisfies $D(q)$.

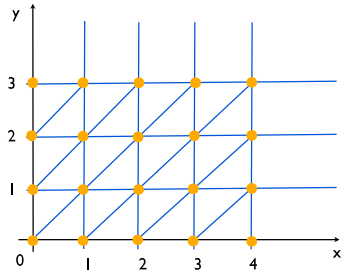
Region Equivalence for One Variable

Let $\equiv$ be an equivalence, where $x \equiv x'$ iff

- $x > M \wedge x' > M$, where M is the largest value associated with x that appears in TA, or
- $x \leq M_i, x' \leq M$, $\lfloor x \rfloor = \lfloor x' \rfloor$, and $frac(x) = 0 \Leftrightarrow frac(x') = 0$

$[x]_{\equiv} = \{x' \mid x \equiv x'\}$

Region Equivalence for Two Variables

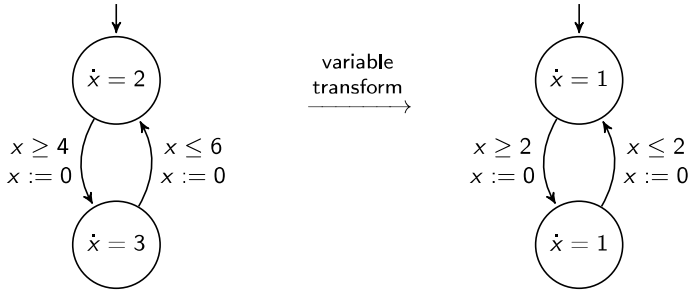


- Equivalence classes: points (orange circles), line segments (blue lines), open sets (between lines)
- $M_x = 4$, $M_y = 3$

Reachability for Timed Automata

- Determine equivalent regions
- Find the quotient transition system of T_{TA}, as $\hat{T}_{TA}$
- Compute reachable set of $\hat{T}_{TA}$
- Same reachability for T_{TA}

Multi-Rate Automata

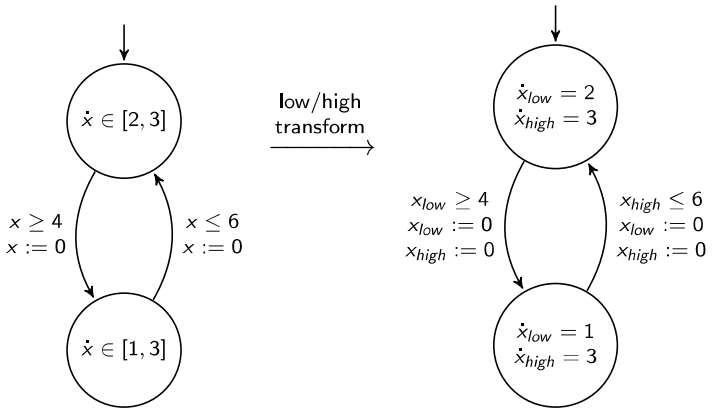


Reachability for Multi-Rate Automata

- Translate into timed automata
- Multi-rate automaton has to be **initialized**, i.e., if a rate of a variable changes along an edge, then the variable has to be reset along the edge

Example: Remove $x := 0$ from the previous example

Rectangular Automata



Reachability for Rectangular Automata

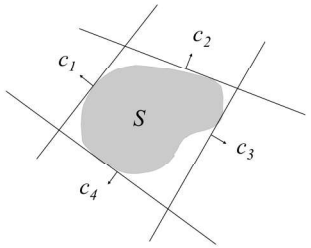
- Translate into multi-rate and then into timed automata
- Rectangular automaton has to be **initialized**, i.e., if a rate of a variable changes along an edge, then the variable has to be reset along the edge

Example: Remove $x := 0$ from the previous example

Over-Approximation of Reach Set

- It is often hard to calculate Reach exactly
- Compute an over-approximation $A \supset$ Reach instead
- Note that $A \cap B = \emptyset$ implies that Reach $\cap B = \emptyset$, so safety is guaranteed if the algorithm based on over-approximation terminates

Approximate a Set with a Polyhedron



- Choose normal vectors $c_1, \ldots, c_n$
- Wrap the set in the polyhedron
- Solve an optimization problem

Figure by R. Alur

Flowpipe Approximation

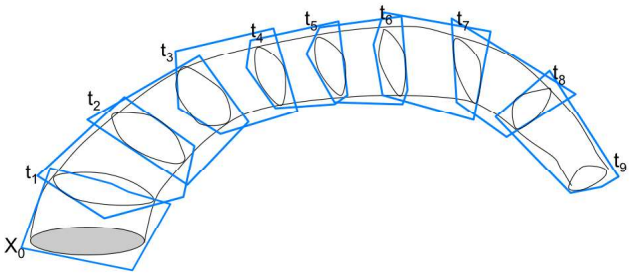


Figure by R. Alur

- Divide Reach(S_0) into $[t_k, t_k + 1]$ segments
- Enclose each segment with a convex polytope
- Reach(S_0) is a union of polytopes

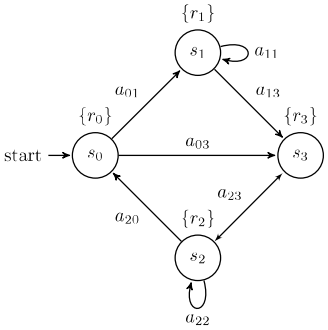
Beyond Reachability Verification

- More complex properties can be expressed and verified,
- Temporal logics, such as LTL, CTL.
- Verification or synthesis of a system and its specification

Model-checking: system model

System modeled as *labeled* finite transition systems (FTS) $\mathcal{T} = (S, \Sigma, \Pi, \rightarrow, L, S_0)$:

- S is a set of states;
- Σ is a set of admissible actions;
- Π is a set of atomic propositions;
- $\rightarrow \in S \times \Sigma \times S$ is a set of transitions;
- $S_0 \subseteq S$ is the set of initial states;
- $L : S \rightarrow 2^{\Pi}$ s.t. $L(s) \subseteq \Pi$ is the set of propositions satisfied by $s \in S$.



Note: the FTS would be weighted by adding costs on the transitions.

Model-checking: specification

Specification given as linear temporal logic (LTL) formulas over Π:

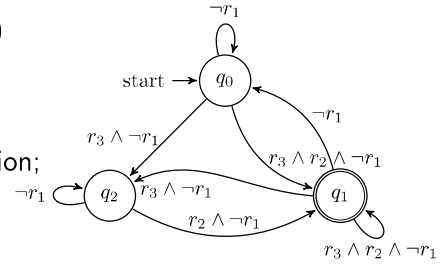
$$\varphi ::= \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi_1 \mid \mathsf{X}\,\varphi_1 \mid \varphi_1\,\mathsf{U}\,\varphi_2 \mid \mathsf{F}\,\varphi_1 \mid \mathsf{G}\,\varphi_1,$$

see Lecture 9 for details.

Control tasks: Safety: $\mathsf{G}\neg\varphi_1$. Order: $\mathsf{F}(\varphi_1 \wedge \mathsf{F}(\varphi_2 \wedge \mathsf{F}\varphi_3))$. Response: $\varphi_1 \rightarrow \varphi_2$. Liveness: $\mathsf{GF}\varphi_1$.

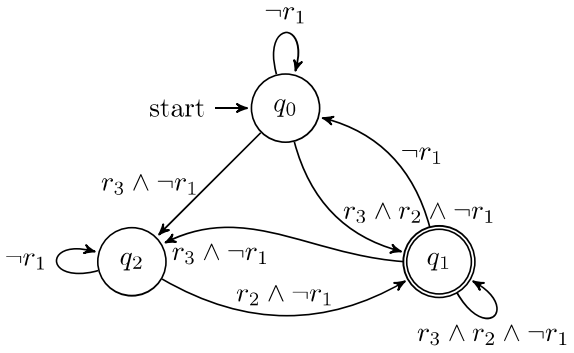
Büchi automaton $\mathcal{A}_\varphi = (Q, 2^{\Pi}, \delta, Q_0, F)$

- Q is a finite set of states;
- 2^{Π} is an input alphabet;
- $\delta : Q \times 2^{\Pi} \rightarrow 2^{Q}$ is a transition relation;
- Q_0, $F \subseteq Q$ are initial and accepting states.



Model-checking: Büchi Automaton example

Example: $\varphi = (\mathsf{G}\,\mathsf{F}\,r_2) \wedge (\mathsf{G}\,\mathsf{F}\,r_3) \wedge (\mathsf{G}\,\neg r_1)$.

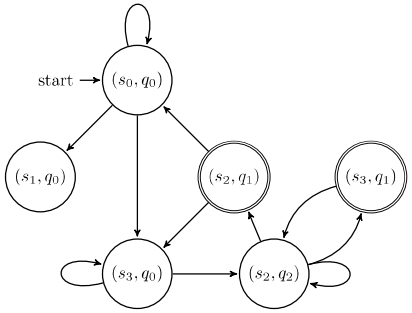


Software: LTL2BA, LTL2dstar, SPOT...

Model-checking: product automaton

The **product** automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_\varphi = (Q_\mathcal{P},\, \Sigma,\, \delta_\mathcal{P},\, Q_{\mathcal{P},0},\, F_\mathcal{P})$,

- $Q_\mathcal{P} = S \times Q$;
- $\delta_\mathcal{P} \subseteq Q_\mathcal{P} \times \Sigma \times Q_\mathcal{P}$.
  $((s,\, q),\, \sigma,\, (s',\, q')) \in \delta_\mathcal{P}$
  if $(s, \sigma,\, s') \in \longrightarrow$
  and $q' \in \delta(q,\, L(s))$;
- $Q_{\mathcal{P},0} = \{(s_0,\, q_0) \mid q_0 \in Q_0\}$;
- $F_\mathcal{P} = \{(s,\, q_f) \mid s \in S,\, q_f \in F\}$.

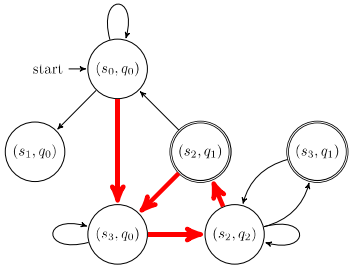


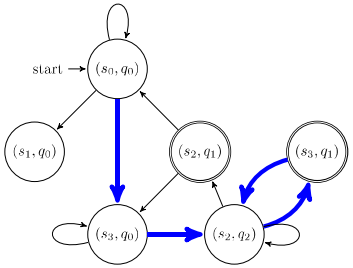
Idea: intersection between language generated by $\mathcal{T}$ and the one accepted by $\mathcal{A}_\varphi$.

Model-checking: graph-search

Graph search for plan prefix and suffix:

- Plan prefix: a path from an initial state to an accepting state;
- Plan suffix: a cycle containing at least an accepting state;
- Combine the plan prefix and suffix (which can be done in different ways).





Next Lecture

Summary

- Summary of the course
- What is in the exam?