Embedded Information Systems

Safety-critical systems

Nonconventional modelling

December 1, 2020

Safety-critical systems

Concepts and requirements of safety-critical systems

- **Risk analysis: Tolerable Hazard Rate** (THR)
  - In case of continuous operation: **the rate of hazardous error phenomena per hour**;
  - In case of discontinuous operation: **the probability of the hazardous error phenomena at the time of calling the function.**

SIL	Error of safety-critical function/hour
1	10^{-6}<THR<10^{-5}
2	10^{-7}<THR<10^{-6}
3	10^{-8}<THR<10^{-7}
4	10^{-9}<THR<10^{-8}

- **Categorization: Safety Integrity Level** (SIL)

1 year = 8760 hours.

Assuming SIL4: $10^8/8760 \cong 11415$ years without error.

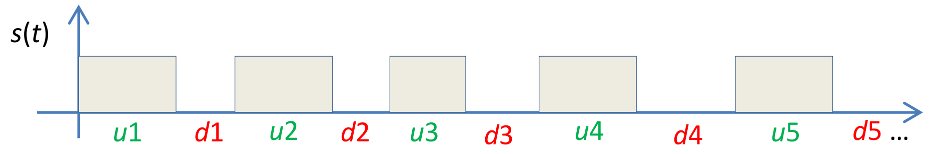
If the lifetime is 15 years, then during this time out of ~750 equipment the failure of one can be expected, since 15*750=11250.

Dependability requirements (Aspects of dependability):

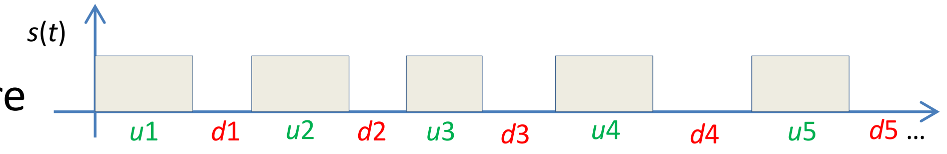
- **Reliability:** Probability of continuous correct service (until the first failure). E.g., “After departure the onboard control system shall function correctly for 12 hours”;
- **Availability:** Probability of correct service (considering repairs and maintenance). E.g., “Availability of the service shall be 95%”;
- **Safety:** Freedom from unacceptable risk of harm;
- **Integrity:** Avoidance of erroneous changes or alterations;
- **Maintainability:** Possibility of repairs and improvements;

Dependability metrics:

Basis: Partitioning the states of the system s(t). **Correct** (U, up) and **incorrect** (D, down) state partitions.



Mean values:



- **MTFF**=E{u1}	Mean Time to First Failure
- **MUT**=E{ui}	Mean Up Time
- **MTTF**	Mean Time To Failure (Same as previous)
- **MDT**=E{di}	Mean Down Time
- **MTTR**	Mean Time To Repair (Same as previous)
- **MTBF**=MUT+MDT	Mean Time Between Failures

Probability functions:

- **Reliability:** r(t)=P{s(t') ∈ U} $\forall t' < t$, system does not fail, →0.
- **Availability:** a(t)=P{s(t)∈ U} decreases with time, system may fail
- **Asymptotic availability**: A=$lim_{t\to\infty}$a(t) A=MTTF/(MTTF+MTTR)

Component attributes:

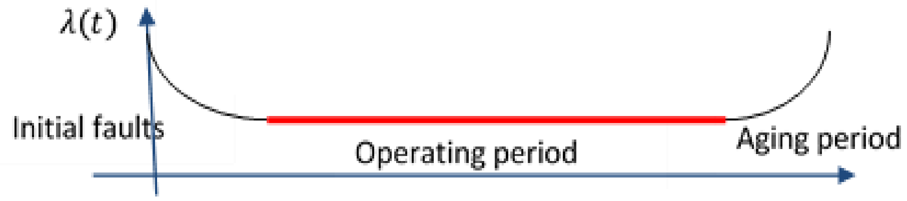
- **Fault rate**: λ(t) Probability that the component will fail in the interval Δt at time point t, given that it has been correct until *t,* is given by λ(t)Δt:

$$\boxed{\lambda(t)\Delta t = P\{s(t+\Delta t) \in D | s(t) \in U\},} \quad \Delta t \to 0.$$

- **Reliability of a component based on this definition**:

$$\lambda(t) = -\frac{1}{r(t)}\frac{dr(t)}{dt}, \text{ thus } r(t) = e^{-\int_0^t \lambda(t)dt}.$$

For electronic components:



In the operating period $\lambda(t) = \lambda$.
Assuming exponential distribution:

$$r(t) = e^{-\lambda t}$$

$$MTFF = E\{u1\} = \int_0^\infty r(t)dt = \frac{1}{\lambda}$$

Comment: Initial errors are filtered by testing at the end of the production lines.

Threats to dependability:

- **Fault:** Adjudged or hypothesized cause of an error.
- **Error:** State leading to the failure.
- **Failure:** The delivered service deviates from correct service.

Example of fault → error → failure chain:

Fault	Error	Failure
Bit flip in the memory due to a cosmic particle	Reading the faulty memory cell will result in incorrect value	The robot arm collides with the wall
The programmer increases a variable instead of decreasing	The faulty statement is executed, and the value of the variable will be incorrect	The result of the computation will be incorrect

Means to improve dependability:

- Fault prevention:
  - o Physical faults: Good components, shielding, ...
  - o Design faults: Good design methodology
- Fault removal:
  - o Design phase: Verification and corrections
  - o Prototype phase: Testing, diagnostics, repair
- Fault tolerance: Avoiding service failures
  - o Operational phase: Fault handling, reconfiguration
- Fault forecasting: Estimating faults and their effects
  - o Measurements and prediction

The development process: e.g. V-model, verification, validation, testing, ...

Organisation and independence of roles:

The roles:

Designer (analyst, architect, coder, unit tester) (DES);
Verifier (VER);
Validator (VAL);
Assessor (ASS);
Project manager (MAN);
Quality assurance personnel (QUA).

In case of **SIL0**:
DES, VER, VAL can be the same person,
ASS should be different person;

In case of **SIL1** and **SIL2**:
DES, VER-VAL, and ASS should be different persons;

In case of **SIL3** and **SIL4**:
MAN, DES, VER-VAL, and ASS should be different persons, or even VER and VAL.

Architecture design to avoid hazard

Fail-safe operation:
(1) fail-stop behaviour (Stopping (switch-off) is a safe state),
(2) fail-operational behaviour (Stopping (switch-off) is not a safe state, service is needed).

Fault tolerance is required.

Typical architectures for fail-stop operation

- Single channel architecture with built-in self-test;
- Two channels: (a) the same program with comparison,
  (b) not the same program with independent checker;

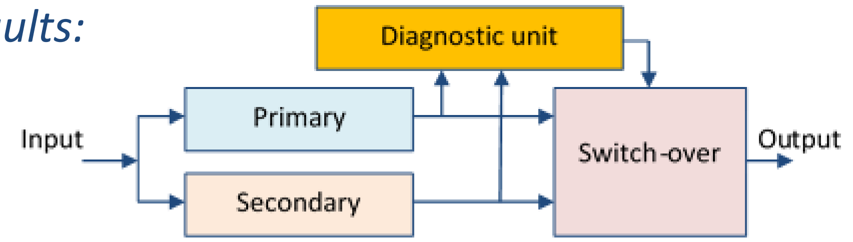
Fault tolerance: Providing (safe) service in case of faults (Fail-operational behaviour)
Extra resources: Redundancy: (1) Hardware, (2) Software, (3) Information, (4) Time.
Types of redundancy: cold (inactive in fault-free case), warm (operates with reduced load), hot (active in fault-free case).

How to use redundancy?

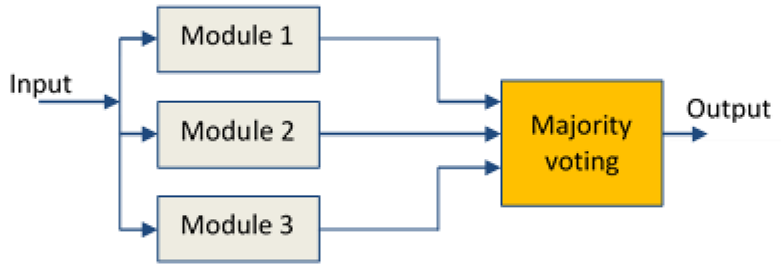
- Hardware design faults (<1%): hardware redundancy with design diversity.
- Hardware permanent operational faults (~20%): hardware redundancy, e.g. redundant processor.
- Hardware transient operational faults (~70-80%): time-redundancy (e.g. instruction retry); information redundancy (e.g. error correcting codes); software redundancy (e.g. recovery from saved state).
- Software design faults (~10): software redundancy with design diversity

Fault tolerance for hardware permanent faults:

Duplication with diagnostics:



Triple-modular redundancy (TMR):



Masking the failure by majority voting.
Voter is critical (but simple).

N-modular redundancy (NMR): Masking the failure by majority voting.
In mission critical systems surviving the mission time is of primary importance.
Following the mission repair is possible. Avionic on-board devices: 4MR, 5MR, or even 7MR.

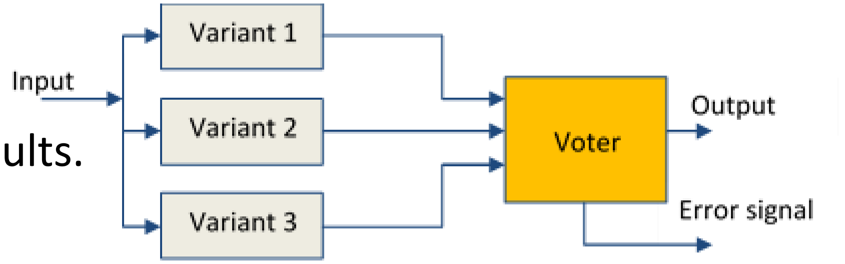
Fault tolerance for software faults:

Repeated execution is not effective for design faults. Redundancy with design diversity is required.

Application of variants:

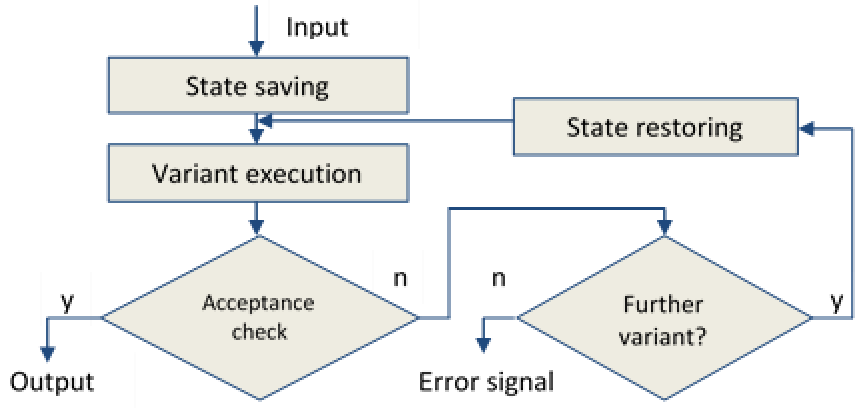
- redundant software modules with diverse algorithms and data structures,
- different programming language and development tools,
- separated development teams.

N-version programming: active redundancy, parallel execution, majority voting.
If output acceptance range is specified, the voter will check it.
The voter is a critical component (but simple).



Recovery blocks:

passive redundancy, activated only in case of faults.



(1) The primary variant is executed first;
(2) Acceptance check is performed at the output of the variants;

(If no acceptance test can be performed, then the method cannot be used.)

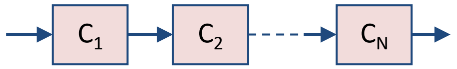
(3) In case of detected error another variant is executed.

Comparison:

Property/Type	N-version programming	Recovery blocks
Error detection	Majority voting, relative	Acceptance checking
Execution	Parallel	Serial
Execution time	Slowest variant or time-out	Depends on # of faults
Activation of redundancy	Always (active)	Only in case of fault (p)
# of tolerated faults	[(N-1)/2]	N-1
Error handling	Masking	Restoration

Reliability Block Diagram

1. Serial system: the components follow each other:



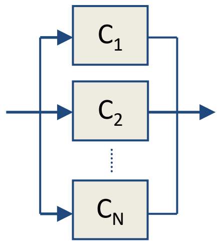
The system is faultless, if all the components are faultless.
The reliability of the system is the product of the reliability of the components:

$$\boxed{r_S(t) = \prod_{i=1}^{N} r_i(t).}$$

If the fault rate of the components is λ_i, then for the system

$$\boxed{MTFF = \frac{1}{\sum_{i=1}^{N} \lambda_i}.}$$

2. Parallel system: the components are parallelly connected:



The system is faulty, if all the components are faulty.
The probability of faulty behaviour is: (1 – reliability).

$$\boxed{1 - r_S(t) = \prod_{i=1}^{N} (1 - r_i(t)).}$$

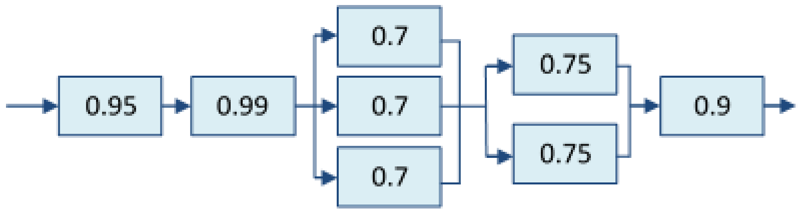
If the reliability of the components is the same: $r_C(t)$, then

$$\boxed{r_S(t) = 1 - (1 - r_C(t))^N.}$$

If the fault rate of the components is λ, then for the system

$$\boxed{MTFF = \frac{1}{\lambda} \sum_{i=1}^{N} \frac{1}{i}.}$$

3. Composite system: can be calculated component by component:

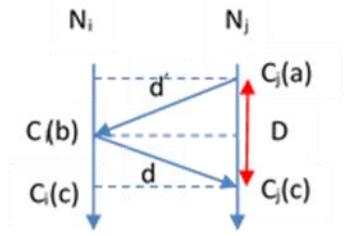


The availability of the system can be computed from the availability of the components:

$$A_S = 0.95 \cdot 0.99 \cdot [1 - (1 - 0.7)^3] \cdot [1 - (1 - 0.75)^2] \cdot 0.9 =$$
$$= 0.95 \cdot 0.99 \cdot 0.973 \cdot 0.9375 \cdot 0.9 = 0.77$$

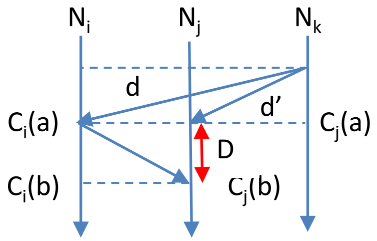
Embedded Information Systems: 2nd Mid-Term Exam 10.12.2019

1. In a distributed system the clocks are synchronized using a round-trip-communication based method. The time of the round-trip measured by the clock to be synchronized is $10\ ms$, while the specified minimum and the maximum of the message forwarding times within the complete network are $d_{min} = 3\ ms$ and $d_{max} = 6\ ms$, respectively. Compute the uncertainty of the clock synchronization (max. 3 points)! This uncertainty can be reduced by the reference broadcasting synchronization method. Why (max. 1 point)?

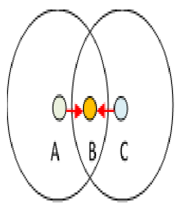


Since d' and d are in the ranges $4 \leq d' \leq 6$, and $4 \leq d \leq 6$, while $d' + d = 10$, $\boxed{\text{Therefore the uncertainty of the synchronization is } \pm 1\ ms.}$

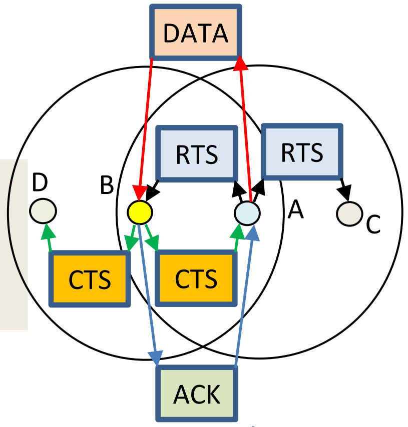
The uncertainty is reduced, because D good approximates d.

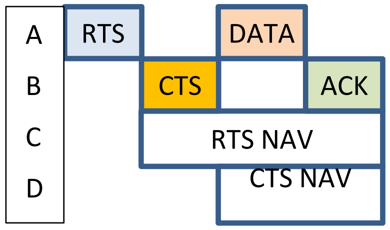


2. Please characterize the hidden terminal problem (max. 1 point)! How operates the RTS/CTS algorithm? What kind of information should contain the RTS and the CTS messages (max. 2 points)?



- A sends message to B
- C does not hear A!
- C also sends message to B
- B is unable to receive messages





Sender „A" sends RTS message to "B"
Receiver „B" replies with CTS message
Receiving CTS the sender "A" transmits data

The other nodes are not allowed to transmit after receiving RTS or CTS!

(NAV = Network Allocation Vector)

The address of the node and the length of the message.

3. Please compare the characteristic features of embedded operating systems with that of the desktop computers (max. 2 points)! What are the embedded real-time kernels for (max. 1 point)? How can the real-time operating systems be combined with conventional ones (max. 1 point)?

Desk-top OS-s are not suitable in embedded systems, because:
(1) the services are too extensive; (2) non-modular, non-fault tolerant, non-configurable, non-modifiable; (3) require large memory; (4) not optimized for power consumption; (5) are not designed for mission-critical application; (6) timing uncertainties are too large.

Embedded RTOS	Standard OS
application software	application software
middleware	middleware
device drivers	OS
real-time kernel	device drivers

The role of the kernel:
to provide parallel programming environment,
scheduling,
inter-task communication,
handling interrupts,
handling timers, and timing services,
(possibly) memory management

The **conventional operating systems** is one of the tasks of the real-time one.

4. Using pseudo-code, please explain how the Non-Blocking-Write (NBW) Protocol operates! What is this protocol for, and under what condition can it be applied (max. 3 points)?

```
Initialization: CCF:=0

Writer:
Start:      CCF_old:=CCF;
            CCF:=CCF_old+1;
            <write into data srtucture>
            CCF:=CCF_old+2;

Reader:
Start:      CCF_begin:=CCF;
            if CCF_begin=odd then goto Start;
            <read data structure>
            CCF_end:=CCF;
            if CCF_end ≠ CCF_begin then goto Start;
```

It is used if the host is event triggered.

5. In case of a communication channel what is the meaning of bit-length (max. 1 point)? What is called protocol efficiency (max. 1 point)?

The term ***bit length*** of a channel is used to denote the number of bits that can traverse the channel within one propagation delay.

For example, if the channel bandwidth is 100 Mbit and the channel is 200 m long, the bit length of the channel is 100 bits, since the propagation delay of this channel is 1 µs.

Then an upper bound for the data efficiency of any media access protocol in a bus system is given by:

$$\boxed{data\ efficiency < \frac{m}{m+bl}}$$

6. We have three hardware units in serial. Each has an availability factor of $K = 0.8$. Applying redundancy we would like to increase the overall availability of the system to a level of $K' \geq 0.8$. Please suggest at least one possible solution (max. 3 points)!

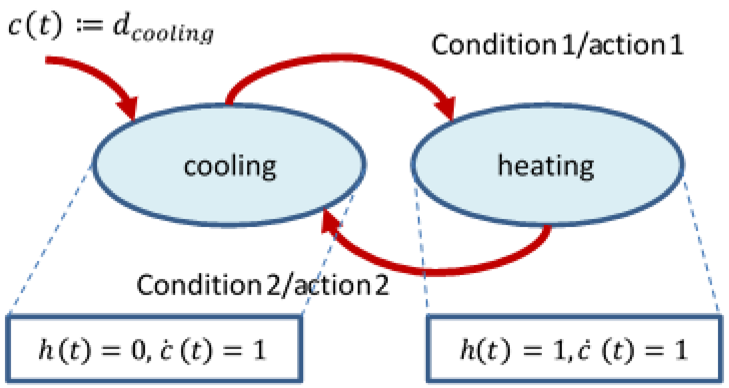
The system without redundancy has overall availability of $0.8 * 0.8 * 0.8 = 0.512$.

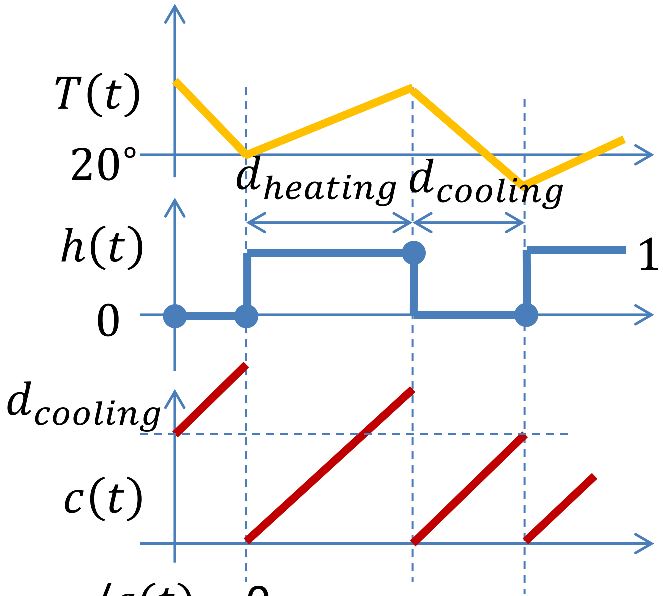
If we duplicate each unit then at unit level we have availability factor of

$$\boxed{1-(1-0.8)^2 = 0.96,}$$

and for the system level: $\boxed{0.96 * 0.96 * 0.96 = 0.8847 > 0.8}$

7. A timed automaton is applied to control heating and cooling. Please explain and characterize the operation by a state-transition diagram (max. 1 point), by defining condition/action steps (max. 1 point), and by the time-diagram of the operation (max. 2 points)!





Condition1/action1: $T(t) \leq 20 \wedge c(t) \geq d_{cooling} / c(t) = 0$.

Condition2/action2: $T(t) \geq 20 \wedge c(t) \geq d_{heating} / c(t) = 0$.

8. Time-Triggered Architecture (TTA) is applied. The nodes communicate via a field-bus. The bandwidth is $100 kbit/s$, the bandwidth utilization is 90%. During the cluster round, each node would like to send a single frame consisting of 50 bytes to the remaining nodes. Would it be possible to fit 10 nodes into the cluster, if the nodes would like to receive/refresh a frame at every $40\ ms$ (max. 2 points)?

10 nodes would send 400 bits at every $40\ ms$. $10 * \frac{400}{0.04} = 10^5 \frac{bit}{s} > 9 * 10^4 \frac{bit}{s}$

The answer is negative.

9. Please explain the polynomial regression problem for the case of a seemingly quadratic relation, if the number of measured values is N, and the model applies two tunable parameters! Derive the explicit formulae suitable to calculate the unknown parameters (max. 3 points)! Please explain why is advantageous to apply quadratic error criteria and models linear in their parameters together (max. 2 points)!

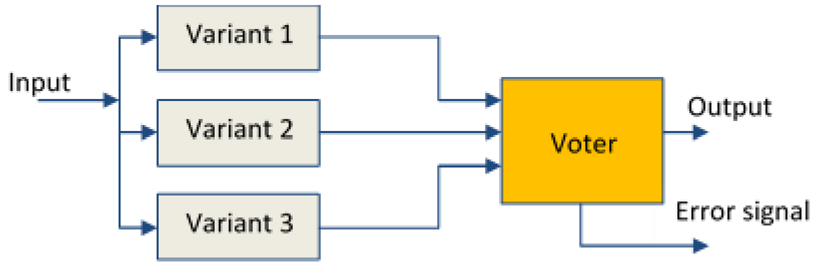
For the measured values we have $y_n = a_0 + a_2 u_n^2 + w_n$, where w_n denotes additive noise, $n = 0,1,\dots,N-1$. With vector notation: $z = Ua + w$. Using the LS method:

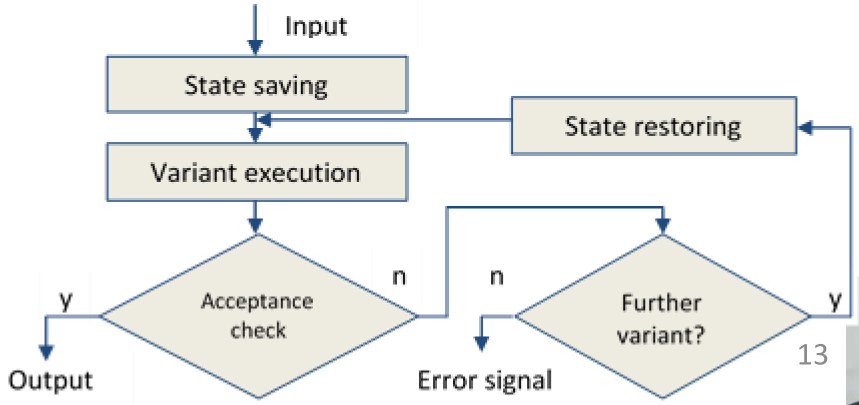
$$z = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & u_0^2 \\ 1 & u_1^2 \\ \vdots & \vdots \\ 1 & u_{N-1}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix} \qquad \left[U^T U \right] = \begin{bmatrix} N & \sum_{n=0}^{N-1} u_n^2 \\ \sum_{n=0}^{N-1} u_n^2 & \sum_{n=0}^{N-1} u_n^4 \end{bmatrix}, \quad U^T z = \begin{bmatrix} \sum_{n=0}^{N-1} y_n \\ \sum_{n=0}^{N-1} u_n^2 y_n \end{bmatrix}$$

$$\begin{bmatrix} \hat{a}_0 \\ \hat{a}_2 \end{bmatrix} = \frac{1}{\frac{1}{N}\sum_{n=0}^{N-1} u_n^4 - \left(\frac{1}{N}\sum_{n=0}^{N-1} u_n^2\right)^2} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} u_n^4 & -\frac{1}{N}\sum_{n=0}^{N-1} u_n^2 \\ -\frac{1}{N}\sum_{n=0}^{N-1} u_n^2 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} y_n \\ \frac{1}{N}\sum_{n=0}^{N-1} u_n^2 y_n \end{bmatrix}$$

Because results in linear set of equation.

10. Please explain the method of N-version programming and that of the recovery blocks (max. 2 points)! Compare the two methods concerning the way of error detection, program execution, way of redundancy activation, number of tolerated faults and error handling (max. 2 points)!





11. Please characterize the "function queue scheduling" method (max. 2 points)! Using this method: What will be the worst-case response time (max. 1 point)?

```
void interrupt A_Handler() { Handle_HW_A();
PutFunction(Service_A); }
void interrupt B_Handler() { Handle_HW_B();
PutFunction(Service_B); }
void interrupt C_Handler() { Handle_HW_C();
PutFunction(Service_C); }
void Service_A();
void Service_B();
void Service_C();
void main() {
        while (TRUE){
                while (IsFunctionQueueEmpty());
                        CallFirstFromQueue();
        }
}
```

maximum response time: execution time of the longest task + the execution time of the *i*th task.

hardware is handled with interrupt

inter-task communication with shared variables: no problem.

Between interrupt and task: pre-emption (shared variables may cause problems)

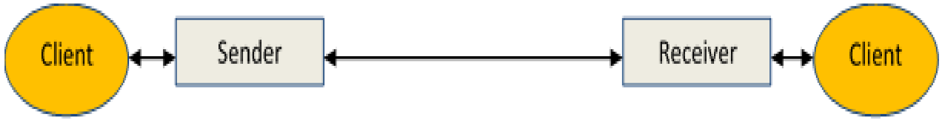
development: easy

The service order from the queue can be: (1) FIFO, (2) priority based

drawback: non preemptive

processor utilization:100%

12. By listing the steps of the corresponding program, please characterize the Positive Acknowledgement or Retransmission (PAR) protocol (max. 2 points)! How and when the communication error is detected (max. 1 points)?



(1) The **client** at the sender's site **initiates** the communication.

(2) The receiver has the authority to **delay the sender** via the bi-directional comm. channel.

(3) The communication error is detected **by the sender**, and not by the receiver.
The receiver is not informed when a communication error has been detected.

(4) **Time redundancy is used** to correct a communication error, thereby increasing the protocol latency in case of errors.

Program of the sender:

(1) The sender initializes a **retry counter to zero**.
(2) The sender **starts** a local time-out interval.
(3) The sender **sends** the message to the receiver.
(4) The sender **receives** an acknowledgement message from the receiver within the specified time-out interval.
(5) The sender **informs** its client about the successful transmission, and duly terminates.

If the sender does not receive a **positive acknowledgement** message from the receiver within the specified time-out interval:

(a) The sender **checks the retry counter** to determine whether the given maximum number of retries has already been exhausted.
(b) If so, **the sender aborts** the communication, and informs its client about the failure.
(c) If not, the sender **increments the retry counter** by one, and returns to (2).

Program of the receiver:

(1) If new message arrives at the receiver, **the receiver checks** whether this message has already been received.
(2) If not, the receiver **sends an acknowledgement** message to the sender, and **delivers** the message to its client.
(3) If yes, it just **sends another acknowledgement message** back to the sender.

(In this case the previous acknowledgement message has arrived at the sender out of the specified time-out interval or failed to arrive).

Nonconventional modelling

Hybrid systems: Hybrid systems combine continuous and discrete dynamics.

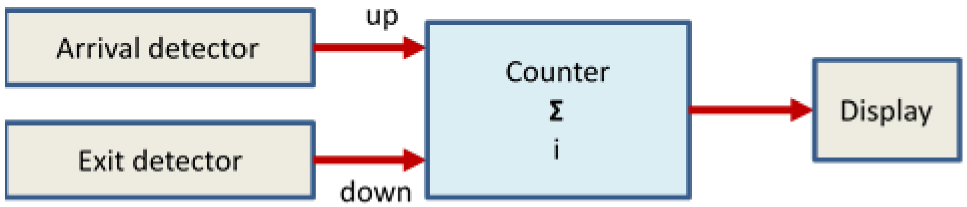
Sometimes they are called modal systems, because controlled by a Finite State Machine (FSM), they are switched into different modes of operation where they behave as continuous systems.

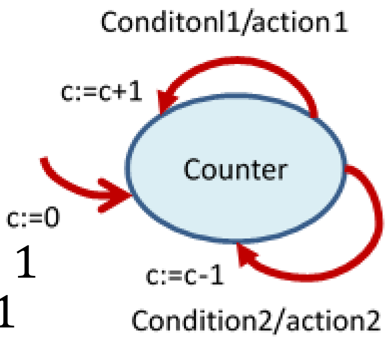
Concerning the mode changes, hybrid systems behave like discrete systems, but between these mode changes time dependency is present.

Discrete systems:

Example:

Number of cars in a parking house (max. *M*, which is the capacity of the house)

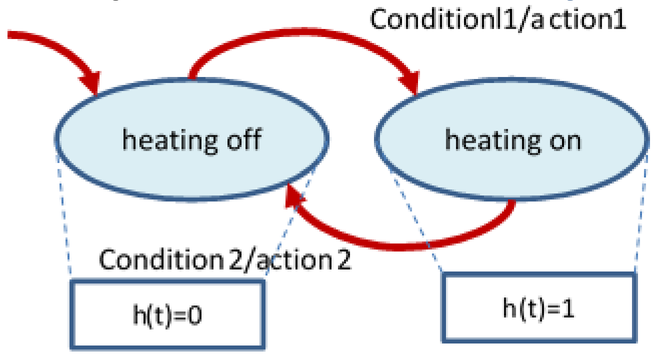




Condition1/action1: $up \wedge \neg down \wedge c < M / c + 1$

Condition2/action2: $down \wedge \neg up \wedge c > 0 / c - 1$

Example: Thermostat with hysteresis



Condition1/action1: $\boxed{\text{Temperature} \leq 18\ \text{C}^{o}/\text{heating on}}$

Condition2/action2: $\boxed{\text{Temperature} \geq 22\ \text{C}^{o}/\text{heating off}}$

System input: Temperature of the environment

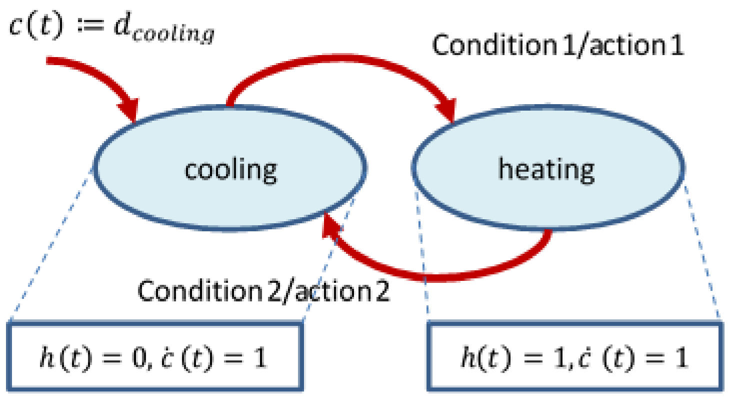
System output: heating on/heating off commands:

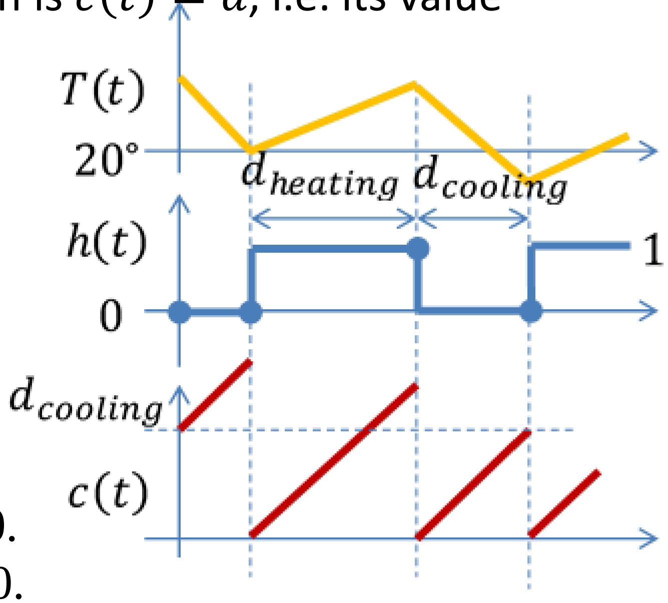
The corresponding **time functions:** $h(t) = 1$, $h(t) = 0$.

Timed automaton:

Example: ***Thermostat with timing instead of hysteresis:*** this is solved by the so-called **timed automaton**, which is the simplest nontrivial hybrid system.

These automata, behind their states measure the evolvement of time for a given value of duration: $\forall t \in d_m$, and the derivative of the clock function is $\dot{c}(t) = a$, i.e. its value changes with the evolvement of time.





Condition1/action1: $T(t) \leq 20 \wedge c(t) \geq d_{cooling} / c(t) = 0.$

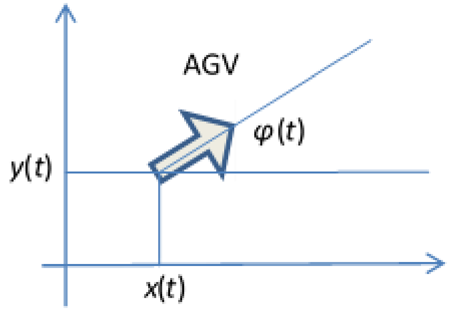
Condition2/action2: $T(t) \geq 20 \wedge c(t) \geq d_{heating} / c(t) = 0.$

Comments:

(1) $h(t)$ and $c(t)$ can be considered as tools of state refinement.They define some details of the operation (Modal systems).

(2) On the time diagram $T > 20\ C^{\circ}$. If it would be lower, then immediately the heating mode would start. This is served by the initial condition of the clock.

Example: ***Automated Guided Vehicle, AGV***



$$\dot{x}(t) = v(t)\cos(\varphi(t))$$
$$\dot{y}(t) = v(t)\sin(\varphi(t))$$
$$\dot{\varphi}(t) = \omega(t)$$

Two-level control: The AGV runs with a constant speed of 10 *km/h*. It has four operational mode: **left, right, straight, stop.**

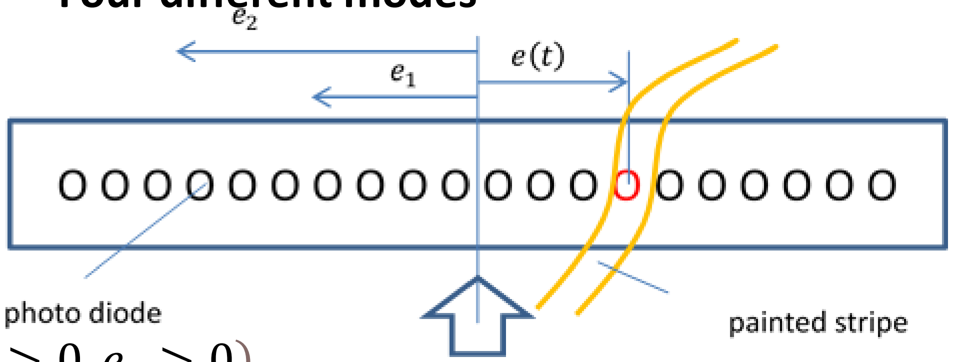
To every mode of operation, a separate differential equation is assigned.

original:	**straight:**	**left:**	**right:**	**stop:**
$\dot{x}(t) = v(t)\cos(\varphi(t))$	$\dot{x}(t) = 10\cos(\varphi(t))$	$\dot{x}(t) = 10\cos(\varphi(t))$	$\dot{x}(t) = 10\cos(\varphi(t))$	$\dot{x}(t) = 0$
$\dot{y}(t) = v(t)\sin(\varphi(t))$	$\dot{y}(t) = 10\sin(\varphi(t))$	$\dot{y}(t) = 10\sin(\varphi(t))$	$\dot{y}(t) = 10\sin(\varphi(t))$	$\dot{y}(t) = 0$
$\dot{\varphi}(t) = \omega(t)$	$\dot{\varphi}(t) = 0$	$\dot{\varphi}(t) = \pi$	$\dot{\varphi}(t) = -\pi$	$\dot{\varphi}(t) = 0$

Four different modes

The sensor of the AGV:
a set of photodiodes perpendicular to the direction of the movement.

Its output signal: $\boxed{e(t) = f(x(t), y(t)).}$



If $e(t) > 0$, then it deviates to the left,
if $e(t) < 0$, then to the right.

$(e_1 > 0, e_2 > 0)$

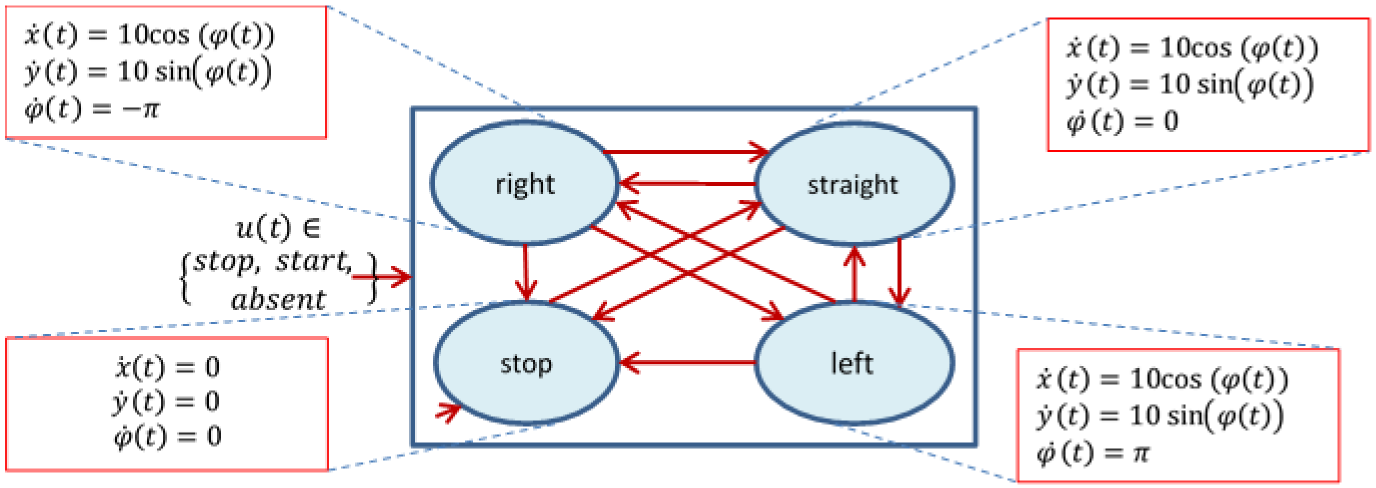
The control law of the AGV:

if $|e(t)| < e_1$, then go straight;
if $0 < e_2 < e(t)$, then go to right;
if $0 > -e_2 > e(t)$, then go to left.

The set of the input events: $u(t) \in \{stop, start, absent\}$.
Since *stop* and *start* are instantaneous events, *absent* gives the interpretation for other time instants.

State-transition generating conditions:

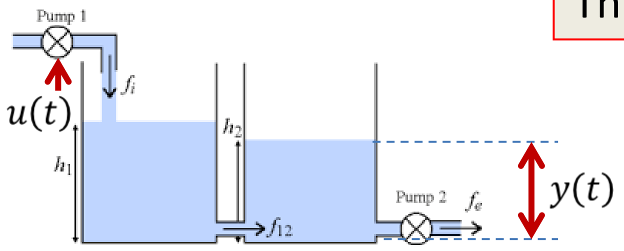
start = $\{(v(t), x(t), y(t), \varphi(t)) | u(t) = start\}$
go straight = $\{(v(t), x(t), y(t), \varphi(t)) | u(t) \neq stop, |e(t)| < e_1\}$
go right = $\{(v(t), x(t), y(t), \varphi(t)) | u(t) \neq stop, e_2 < e(t)\}$
go left = $\{(v(t), x(t), y(t), \varphi(t)) | u(t) \neq stop, -e_2 > e(t)\}$
stop = $\{(v(t), x(t), y(t), \varphi(t)) | u(t) = stop\}$


$\dot{x}(t) = 10\cos(\varphi(t))$
$\dot{y}(t) = 10\sin(\varphi(t))$
$\dot{\varphi}(t) = -\pi$
$\dot{x}(t) = 10\cos(\varphi(t))$
$\dot{y}(t) = 10\sin(\varphi(t))$
$\dot{\varphi}(t) = 0$
right
straight
$u(t) \in \{stop, start, absent\}$
stop
left
$\dot{x}(t) = 0$
$\dot{y}(t) = 0$
$\dot{\varphi}(t) = 0$
$\dot{x}(t) = 10\cos(\varphi(t))$
$\dot{y}(t) = 10\sin(\varphi(t))$
$\dot{\varphi}(t) = \pi$

Qualitative modelling and control I.

Example: The design of such a controller which keeps the level of the liquid in the second tank at a prescribed level.

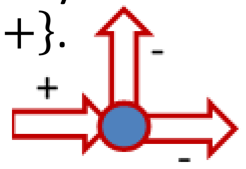
This is possible by setting *u(t)* at pump1 properly.



Problems of the quantitative model:
a) The physical limits are not modelled;
b) The equations are linearized;
c) Numerical values are inaccurate and change with time.

Qualitative Reasoning: Only the orientation of the quantities is considered.

Possible "values": $\{-, 0, +\}$.
The basic physical constraints are kept!



If at branching of a node the liquid flows out in two directions, then through the third tube the liquid should flow in.

The qualitative value of a quantity "*Q*" with respect to "*a*": $\boxed{[Q]_a}$
The qualitative value of the change of a quantity "*Q*" is the qualitative derivative:

$$[\delta Q]_a, [\delta^2 Q]_a, \ldots$$

Operations: $(invert\ A)$: Changes the sign.

$(vote\ A_1, A_2, \dots, A_n)$: Gives back the value in majority.

Qualitative control of the level of tank2: L_2 denotes the level relative to the desired value:

$[L_2] = +$:	higher than required.	$[\delta U] = +$:	increase pumping rate.
$[L_2] = 0$:	equals.	$[\delta U] = 0$:	pumping rate is appropriate.
$[L_2] = -$:	lower than required.	$[\delta U] = -$:	decrease pumping rate.

$[\delta U] = +$: a fixed amount of increase of the pumping rate: ΔU.

The qualitative values exist only at the sampling instants.
Between sampling instants there is no level detection.

$$\boxed{[L_2]_{(k)} = \left[actual\ level_{(k)} - required\ level_{(k)}\right]}$$

A very simple control law:

$$\boxed{Q1 \overset{def}{\triangleq} [\delta U]_{(k)} = (invert[L_2])_{(k)}}$$

Comment:

If ΔU is larger, then larger overshoot and oscillation can be expected, but the reaction is faster.
If ΔU is smaller, then the overshoot and the oscillation will be smaller, but also the reaction is slower.

Improved controllers: Quantities considered:

$$\left.\begin{array}{ll} \text{Level error of tank2:} & +, 0, - \\ \text{Speed of the level change of tank2:} & +, 0, - \\ \text{Speed of the level change of tank1:} & +, 0, - \end{array}\right\} 3 * 3 * 3 = 27$$

$$\boxed{Q2 \overset{def}{\triangleq} [\delta U]_{(k)} = \left(invert\left(vote\left(vote\left([L_2]_{(k)}, [\delta L_2]_{(k)}\right), [\delta L_1]_{(k)}\right)\right)\right)_{(k)}}$$

$$\boxed{Q3 \overset{def}{\triangleq} [\delta U]_{(k)} = \left(invert\left(vote\left([L_2]_{(k)}, [\delta L_2]_{(k)}, [\delta L_1]_{(k)}\right)\right)\right)_{(k)}}$$

Determination of $[\delta L_1]$: $\delta L_1 = \left(L_{2(k)} - L_{2(k-1)}\right) - \left(L_{2(k-1)} - L_{2(k-2)}\right) = \delta^2 L_2$

For the 27 combinations of the possible qualitative values the outputs of the three controllers can be summarized in the table below:

	$[L_2]$	$[\delta L_2]$	$[\delta L_1]$	$Q1$	$Q2$	$Q3$
1	+	+	+	-	-	-
2	+	+	0	-	-	-
3	+	+	-	-	0	-
4	+	0	+	-	-	-
5	+	0	0	-	-	-
...						
20	-	+	0	+	0	0
...						
27	-	-	-	+	+	+

Comments:

(1) A rule-based system was also elaborated for this problem.
It could not handle the case: Tank2 shows a constant value above the required level, and the level of Tank1 lowers.

(2) Setting sampling rate and the value of *ΔU* is a critical issue, and a crucial decision of the designer.