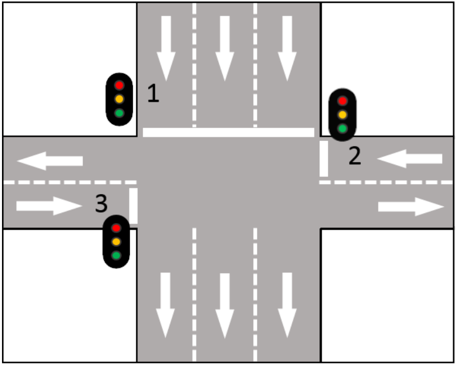
# Embedded systems
# Exercise session, 20/12
# Exam preparation

# Hybrid systems

# Problem 1 (cf. previous exercise session)

A smart traffic light system is installed at the intersection of two roads. There are three stop lights $\{1, 2, 3\}$ that can either be red, green, or orange. The state of 2 and 3 is identical at all times.



Traffic lights 2 and 3 are red and traffic light 1 is green as long as there are less than six cars waiting in front of 2 or 3. When this threshold is reached, stop light 1 becomes orange for 5 seconds before switching to red. At this time, stop lights 2 and 3 become green for 15 seconds. After that delay, they change to orange for 5 seconds and then switch to red as traffic light 1 becomes green again.

The incoming flows of cars at the three traffic lights are respectively $f_1 = 30$, $f_2 = 6$ and $f_3 = 3$ cars/minute. We also define the saturation flow of a traffic light as the rate of cars that are able to cross this light when it is green. The saturation flows of the three lights are respectively $s_1 = 1.5$, $s_2 = 0.5$ and $s_3 = 0.5$ car/second.

1. Construct a hybrid system that models this traffic management system. Initially, no cars are queueing in front of the traffic lights, and stop lights 2 and 3 are red while 1 is green.
2. Give the first 3 steps of the space-state exploration of this system.

Processes and variables:

- $P_1$: sequence of the lights, $x_1 =$ timer.
- $P_2$: cars incoming at 1, $x_2 =$ number of cars waiting.
- $P_3$: cars incoming at 2, $x_3 =$ number of cars waiting.
- $P_4$: cars incoming at 3, $x_4 =$ number of cars waiting.

Note: For the sake of simplicity, we assume the flows of cars to be continuous. To obtain a more precise model, we would need two variables for each traffic light:

- one for the (integer) number of cars waiting, and
- one for a (real) timer regulating the flow of cars.

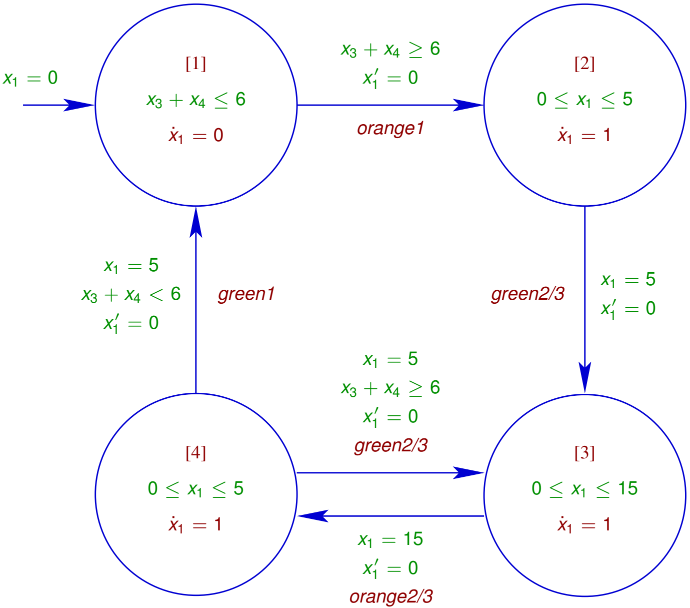
## Process 1

### Modes of operation:

- [1]: Light 1 is green, lights 2 and 3 are red.
- [2]: Light 1 is orange, lights 2 and 3 are red.
- [3]: Light 1 is red, lights 2 and 3 are green.
- [4]: Light 1 is red, lights 2 and 3 are orange.

### Communication:

- $x_3$ and $x_4$ provide the number of cars waiting at lights 2 and 3.
- Synchronization labels *green1*, *orange1*, *green2/3*, *orange2/3* inform the other processes of the lights state changes.


$x_1 = 0$
[1]
$x_3 + x_4 \leq 6$
$\dot{x}_1 = 0$
$x_3 + x_4 \geq 6$
$x_1' = 0$
*orange1*
[2]
$0 \leq x_1 \leq 5$
$\dot{x}_1 = 1$
$x_1 = 5$
$x_3 + x_4 < 6$
$x_1' = 0$
*green1*
*green2/3*
$x_1 = 5$
$x_1' = 0$
$x_1 = 5$
$x_3 + x_4 \geq 6$
$x_1' = 0$
*green2/3*
[4]
$0 \leq x_1 \leq 5$
$\dot{x}_1 = 1$
[3]
$0 \leq x_1 \leq 15$
$\dot{x}_1 = 1$
$x_1 = 15$
$x_1' = 0$
*orange2/3*

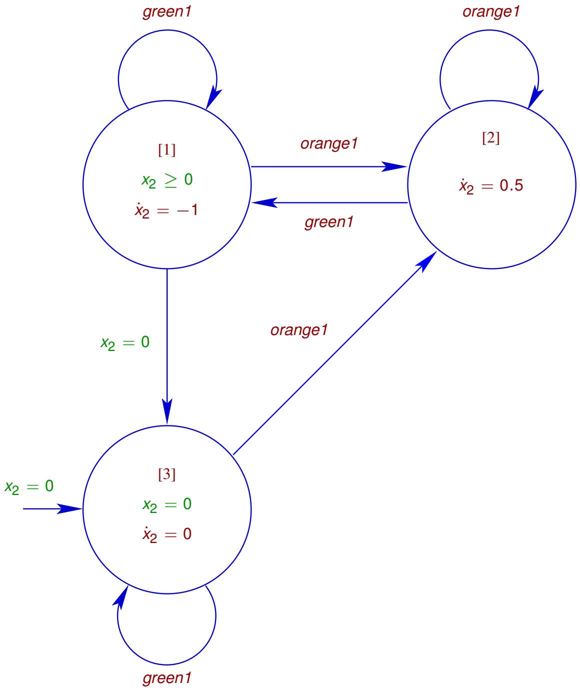
## Process 2

### Modes of operation:

- [1]: Light 1 is green.
  The value of $x_2$ changes at the rate of
  30 cars/min $-$ 1.5 cars/s $=$ $-1$ car/s.
- [2]: Light 1 is orange or red. The value of $x_2$ increases at the rate of 30 cars/min $=$ 0.5 car/s.
- [3]: Light 1 is green, and $x_2 = 0$.

### Communication:

- Provides the value of $x_2$.
- Reacts to the labels *green1* and *orange1*.


green1
orange1
[1]
$x_2 \geq 0$
$\dot{x}_2 = -1$
orange1
green1
[2]
$\dot{x}_2 = 0.5$
$x_2 = 0$
orange1
$x_2 = 0$
[3]
$x_2 = 0$
$\dot{x}_2 = 0$
green1

## Process 3

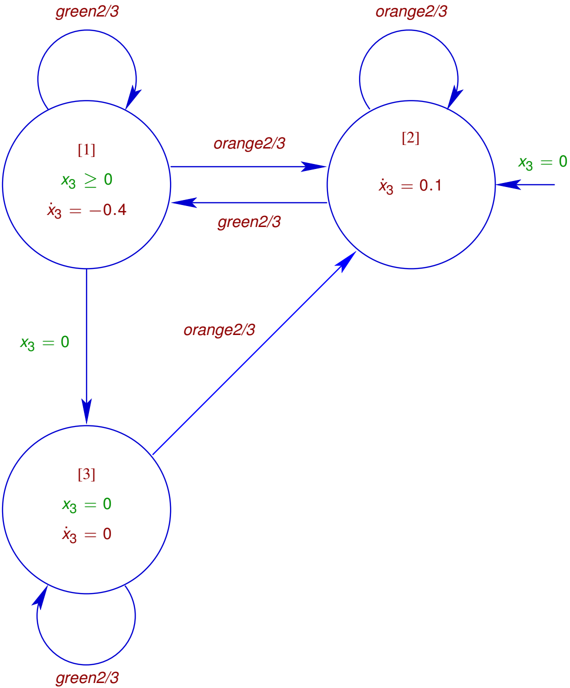
(Similar to Process 2)

### Modes of operation:

- [1]: Light 2 is green. The value of $x_3$ changes at the rate of 6 cars/min $-$ 0.5 car/s $= -$0.4 car/s.
- [2]: Light 2 is orange or red. The value of $x_3$ increases at the rate of 6 cars/min $=$ 0.1 car/s.
- [3]: Light 2 is green, and $x_3 = 0$.

### Communication:

- Provides the value of $x_3$.
- Reacts to the labels *green2/3* and *orange2/3*.


green2/3
orange2/3
[1]
$x_3 \geq 0$
$\dot{x}_3 = -0.4$
orange2/3
green2/3
[2]
$\dot{x}_3 = 0.1$
$x_3 = 0$
$x_3 = 0$
orange2/3
[3]
$x_3 = 0$
$\dot{x}_3 = 0$
green2/3

## Process 4

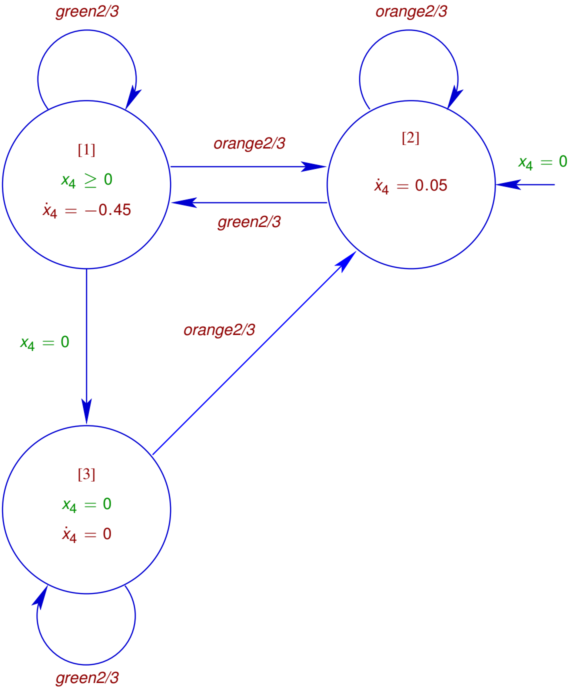
(Similar to Processes 2 and 3)

### Modes of operation:

- [1]: Light 3 is green. The value of $x_4$ changes at the rate of 3 cars/min $-$ 0.5 car/s $=$ $-$0.45 car/s.
- [2]: Light 3 is orange or red. The value of $x_4$ increases at the rate of 3 cars/min $=$ 0.05 car/s.
- [3]: Light 3 is green, and $x_4 = 0$.

### Communication:

- Provides the value of $x_4$.
- Reacts to the labels *green2/3* and *orange2/3*.


green2/3
orange2/3
[1]
$x_4 \geq 0$
$\dot{x}_4 = -0.45$
orange2/3
green2/3
[2]
$\dot{x}_4 = 0.05$
$x_4 = 0$
$x_4 = 0$
orange2/3
[3]
$x_4 = 0$
$\dot{x}_4 = 0$
green2/3

Note: We have assumed that no car passes an orange light.

State-space exploration:

$$([1],[3],[2],[2]) : x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 0.$$

$$\begin{array}{lll}
\overset{\leq 40}{\Longrightarrow} & ([1],[3],[2],[2]) & : \quad x_1 = 0,\ x_2 = 0,\ x_3 = 0.1\,t \\
 & & \phantom{:} \quad x_4 = 0.05\,t,\ 0 \leq t \leq 40. \\
\xrightarrow{\mathit{orange1}} & ([2],[2],[2],[2]) & : \quad x_1 = 0,\ x_2 = 0,\ x_3 = 4,\ x_4 = 2. \\
\overset{\leq 5}{\Longrightarrow} & ([2],[2],[2],[2]) & : \quad x_1 = t,\ x_2 = 0.5\,t,\ x_3 = 4 + 0.1\,t, \\
 & & \phantom{:} \quad x_4 = 2 + 0.05\,t,\ 0 \leq t \leq 5. \\
\longrightarrow & \ldots &
\end{array}$$

# Problem 2

The principle of a telemeter is to measure the time elapsed between the emission of an ultrasonic signal and the reception of an echo sent back by a target. If this time is equal to *t* seconds, then the distance to this target is estimated to be 170 *t* meters.

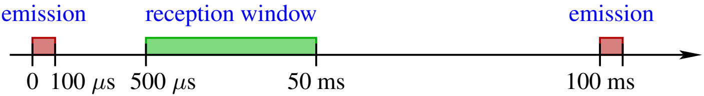
The signals are emitted every 100 ms and last 100 $\mu$s each. The received signals are considered valid if they meet the following conditions:

- they last between 50 and 150 $\mu$s,
- they are received between 500 $\mu$s and 50 ms after the emission of a signal (this delay is computed between the first transition of the signals).

The other signals are discarded.

1. Considering a simple environment where a signal can be received at any moment, describe a hybrid system modelling a telemeter.
2. Give the first 3 steps of the space-state exploration of this system.

## Problem digest:



(not to scale)

## Processes and variables:

- $P_1$: Emission cycle, $x_1$ = time since beginning of last emission, $x_2$ = emitted signal.
- $P_2$: Receiver, $x_3$ = received signal, $x_4$ = timer, $x_5$ = measured distance.
- $P_3$: Environment.

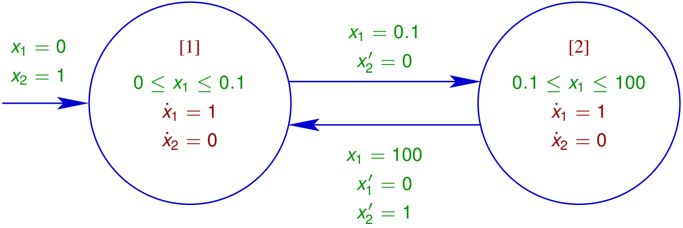
## Process 1

### Modes of operation:

- [1]: Emitting.
- [2]: Waiting.

### Communication:

- Provides $x_1$ (that helps determine reception window).
- Provides $x_2$ (representing the emitted signal: 1 = on, 0 = off).


$x_1 = 0$
$x_2 = 1$
[1]
$0 \leq x_1 \leq 0.1$
$\dot{x}_1 = 1$
$\dot{x}_2 = 0$
$x_1 = 0.1$
$x_2' = 0$
$x_1 = 100$
$x_1' = 0$
$x_2' = 1$
[2]
$0.1 \leq x_1 \leq 100$
$\dot{x}_1 = 1$
$\dot{x}_2 = 0$

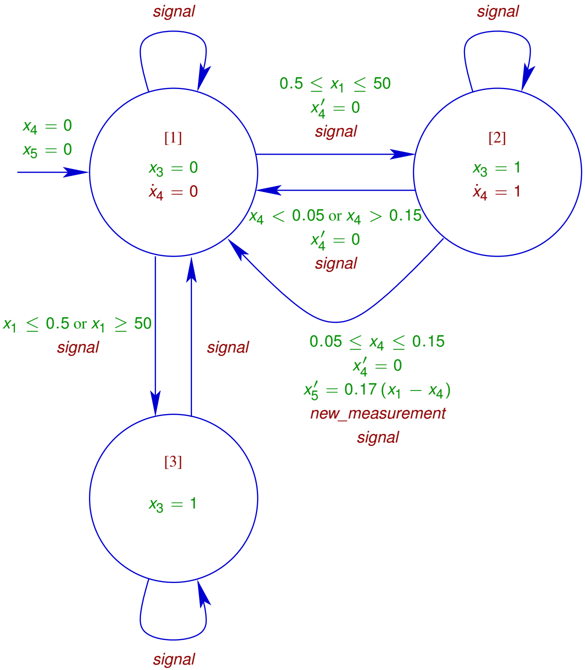
# Process 2

## Modes of operation:

- [1]: Waiting to receive.
- [2]: Measuring the duration of the received signal (in the reception window).
- [3]: Receiving a signal outside of the reception window.

## Communication:

- Reads the received signal $x_3$ (1 = on, 0 = off).
- Synchonizes on labels *new_measurement* and *signal*.
- Provides the length measurement $x_5$.

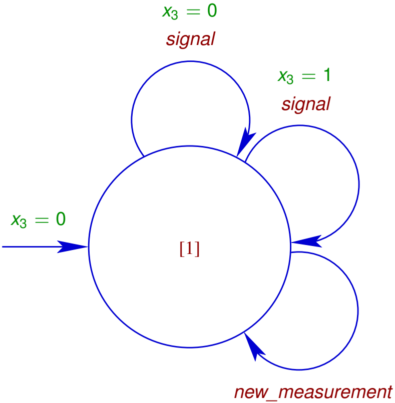

signal
signal
$0.5 \leq x_1 \leq 50$
$x_4' = 0$
signal
$x_4 = 0$
$x_5 = 0$
[1]
$x_3 = 0$
$\dot{x}_4 = 0$
[2]
$x_3 = 1$
$\dot{x}_4 = 1$
$x_4 < 0.05$ or $x_4 > 0.15$
$x_4' = 0$
signal
$x_1 \leq 0.5$ or $x_1 \geq 50$
signal
signal
$0.05 \leq x_4 \leq 0.15$
$x_4' = 0$
$x_5' = 0.17\,(x_1 - x_4)$
new_measurement
signal
[3]
$x_3 = 1$
signal

## Process 3

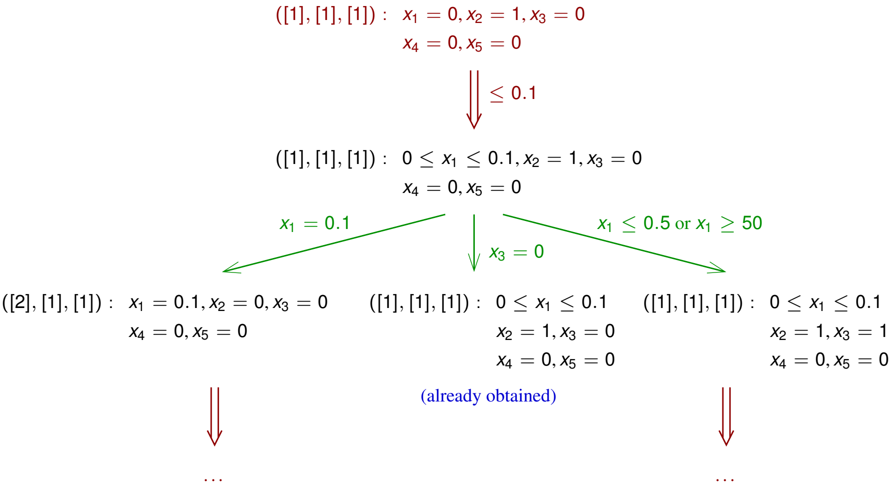
Modes of operation: Only one: [1].

Communication:

- Emits (non-deterministically) the received signal $x_3$ as well as the synchronization label *signal*.
- Reacts to *new_measurement*.


$x_3 = 0$
signal
$x_3 = 1$
signal
$x_3 = 0$
[1]
new_measurement

# State-space exploration:



$([1],[1],[1]):\ x_1 = 0, x_2 = 1, x_3 = 0$
$x_4 = 0, x_5 = 0$

$\leq 0.1$

$([1],[1],[1]):\ 0 \leq x_1 \leq 0.1, x_2 = 1, x_3 = 0$
$x_4 = 0, x_5 = 0$

$x_1 = 0.1$

$x_3 = 0$

$x_1 \leq 0.5$ or $x_1 \geq 50$

$([2],[1],[1]):\ x_1 = 0.1, x_2 = 0, x_3 = 0$
$x_4 = 0, x_5 = 0$

$([1],[1],[1]):\ 0 \leq x_1 \leq 0.1$
$x_2 = 1, x_3 = 0$
$x_4 = 0, x_5 = 0$

(already obtained)

$([1],[1],[1]):\ 0 \leq x_1 \leq 0.1$
$x_2 = 1, x_3 = 1$
$x_4 = 0, x_5 = 0$

$\ldots$

$\ldots$

# Problem 3

A quadcopter is able to fly autonomously by automatically adjusting its altitude with respect to the ground using a distance sensor. The nominal altitude *a* is a parameter provided by the pilot, and remains constant during flight. When the quadcopter is flying, it constantly measures the difference between its actual altitude (measured by the sensor) and *a*. If the absolute value of this difference exceeds 20 cm, then the aircraft adjusts its altitude at a speed between 0.9 and 1.1 m/s on the vertical axis, in the appropriate direction. When the measured altitude is 20 cm or less from *a*, the quadcopter keeps its vertical speed between $-0.1$ and 0.1 m/s.

The quadcopter is equipped with a 2400 mAh battery. During flight, its motors and embedded electronics drain a current between 2 and 10 A. As soon as the charge level of the battery drops below 600 mAh, the quadcopter immediately enters an automatic landing procedure, during which it decreases its altitude at a constant rate of 2 m/s until it touches the ground. It then automatically shuts off. The aircraft also shuts off if its battery becomes totally depleted, which corresponds to an abnormal situation that should be avoided.

Initially, we assume the quadcopter to be on the ground with its motors turned on and a fully charged battery.

1. Model the behavior of this quadcopter with a hybrid system.
2. Explain how this hybrid system can be used for checking whether an abnormal situation can be reached for a given value of the parameter *a*. Illustrate your answer by carrying out in detail the first three steps of the procedure.

Processes and variables:

- $P_1$: altitude control, $x_1 =$ altitude in m.
- $P_2$: battery, $x_2 =$ remaining capacity in Ah.

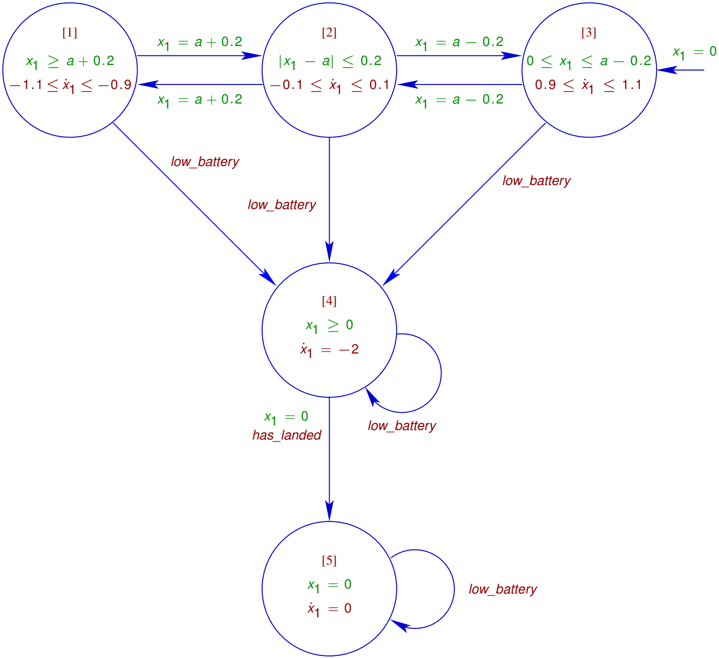
## Process 1

### Modes of operation:

- [1]: $x_1 \geq a + 0.2$.
- [2]: $a - 0.2 \leq x_1 \leq a + 0.2$.
- [3]: $x_1 \leq a - 0.2$.
- [4]: Descending for landing.
- [5]: Landed.

### Communication:

- Provides $x_1$.
- Reacts to label *low_battery* to know when to initiate landing.
- Synchronizes with *has_landed* to signal when ground is reached.


[1]
$x_1 \geq a + 0.2$
$-1.1 \leq \dot{x}_1 \leq -0.9$
[2]
$|x_1 - a| \leq 0.2$
$-0.1 \leq \dot{x}_1 \leq 0.1$
[3]
$0 \leq x_1 \leq a - 0.2$
$0.9 \leq \dot{x}_1 \leq 1.1$
$x_1 = 0$
$x_1 = a + 0.2$
$x_1 = a + 0.2$
$x_1 = a - 0.2$
$x_1 = a - 0.2$
*low_battery*
*low_battery*
*low_battery*
[4]
$x_1 \geq 0$
$\dot{x}_1 = -2$
*low_battery*
$x_1 = 0$
*has_landed*
[5]
$x_1 = 0$
$\dot{x}_1 = 0$
*low_battery*

Notes:

- We assume $a > 0.2$.
- After reaching [2], it is impossible to stay in locations [1] and [3] for a non-zero time.

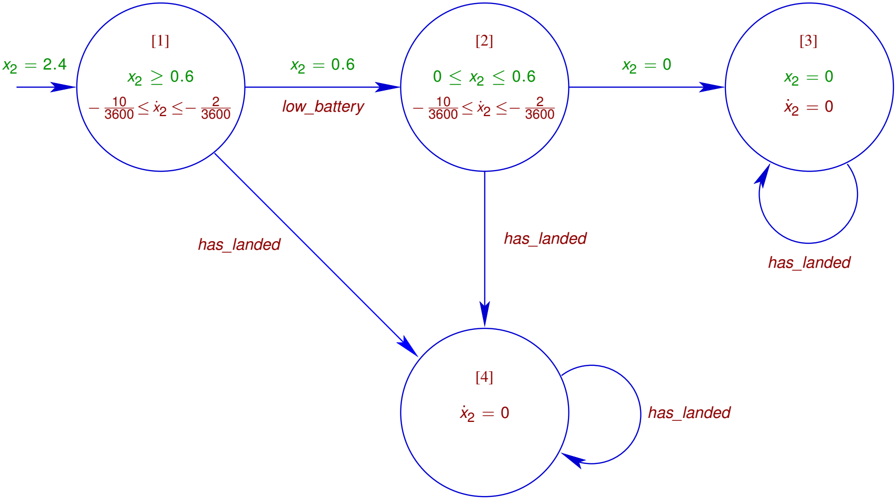
# Process 2

## Modes of operation:

- [1]: Normal mode, $x_2 \geq 0.6$.
- [2]: Emergency mode, $x_2 \leq 0.6$.
- [3]: Battery fully depleted, $x_2 = 0$.
- [4]: Turned off, $\dot{x}_2 = 0$.

## Communication:

- Provides $x_2$.
- Emits the synchronization label *low_battery* to initiate automatic landing procedure.
- Reacts to the label *has_landed* to turn the quadcopter off.


$x_2 = 2.4$
[1]
$x_2 \geq 0.6$
$-\frac{10}{3600} \leq \dot{x}_2 \leq -\frac{2}{3600}$
$x_2 = 0.6$
low_battery
[2]
$0 \leq x_2 \leq 0.6$
$-\frac{10}{3600} \leq \dot{x}_2 \leq -\frac{2}{3600}$
$x_2 = 0$
[3]
$x_2 = 0$
$\dot{x}_2 = 0$
has_landed
has_landed
has_landed
[4]
$\dot{x}_2 = 0$
has_landed

## State-space exploration:

Checking whether an abnormal situation is possible amounts to checking whether Location [3] is reachable in Process 2.

Note: We assume (realistically) that $a$ is such that the first ascent and altitude stabilization do not deplete the battery below 600 mAh.

$$([3],[1]) : x_1 = 0,\ x_2 = 2.4.$$

$$\begin{array}{llll}
\xRightarrow{\le \frac{a-0.2}{0.9}} & ([3],[1]) & : & 0.9\, t_1 \le x_1 \le 1.1\, t_1, \\
 & & & 2.4 - \dfrac{t_1}{360} \le x_2 \le 2.4 - \dfrac{t_1}{1800}, \\
 & & & 0 \le t_1 \le \dfrac{a-0.2}{0.9} \\
\xrightarrow{x_1 = a-0.2} & ([2],[1]) & : & x_1 = a - 0.2, \\
 & & & 2.4 - \dfrac{t_1}{360} \le x_2 \le 2.4 - \dfrac{t_1}{1800}, \\
 & & & 0 \le t_1 \le \dfrac{a-0.2}{0.9}
\end{array}$$

$$\begin{array}{lll}
\overset{\leq 3240}{\Longrightarrow} & ([2],[1]) & : \quad a - 0.2 \leq x_1 \leq \min(a - 0.2 + 0.1\, t_2, a + 0.2), \\
& & \quad \max\left(2.4 - \dfrac{t_1 + t_2}{360}, 0.6\right) \leq x_2 \leq 2.4 - \dfrac{t_1 + t_2}{1800}, \\
& & \quad 0 \leq t_1 \leq \dfrac{a - 0.2}{0.9} \\
& & \quad 0 \leq t_2 \leq 3240 - t_1 \\
\longrightarrow & \ldots &
\end{array}$$

# Software architectures

# Problem 1

In a cooling system, a microcontroller controls the opening of a cold water valve in order to keep the temperature of a certain element constant. It performs the following tasks:

- sending the value of an angle to the valve 10 times per second. This operation lasts 1 ms.
- communicating with a sensor array 1 time per second. This operation requires 1 ms.
- computing a new angle whenever a variation of more than 2% is observed in the measurements. This computation lasts 2 seconds.
- monitoring an alert button 25 times per second. This takes a negligible amount of time.

If the alert button is pressed, a new angle has to be computed as soon as possible using predetermined parameters and previous measurements. The alert button must be pressed again to revert to the previous computation method for the angle.

1. What is the best software architecture for this system? (Carefully justify your answer.)
2. Using pseudocode, give the global structure of this embedded software.

## List of tasks:

- $\tau_1$: Valve command: period = 100 ms, exec. time = 1 ms.
- $\tau_2$: Sensor communication: period = 1000 ms, exec. time = 1 ms.
- $\tau_3$: Angle computation: period = whenever possible, exec. time = 2 s.
- $\tau_4$: Button monitoring: period = 40 ms, exec. time = $\varepsilon$.

## Can some tasks be performed by interrupt routines?

Yes: $\tau_4$. (The other tasks take too long.)

## Is preemption needed?

Yes! ($\tau_1$ and $\tau_2$ over $\tau_3$.)

$\rightarrow$ RTOS.

## Task priorities?

$P(\tau_1) > P(\tau_2) > P(\tau_3)$. Task $\tau_4$ performed in an interrupt routine.

(Note: $\tau_4$ could also be implemented by a periodic task.)

## Tasks communication:

- Global variables for the valve angle, the sensor measurements, and the alert mode.
- Binary semaphores controlling concurrent access to those variables.
- Possibility for $\tau_4$ of terminating $\tau_3$ in the case of an alert.

```
#include <rtos.h>
#include <rtos-semaphores.h>
#include "datastruct.h"

static volatile double valve_angle;
static volatile sensors_data measurements;
static volatile int in_alert_mode = 0;
static semaphore angle_sem, measurements_sem;

static void task1(void)  /* Valve command */
{
  double a;

  wait(angle_sem);
  a = valve_angle;
  signal(angle_sem);

  !! send angle a to valve
}
```

```
static void task2(void)  /* Sensor communication */
{
  sensors_data d;

  !! acquire sensors data in d

  wait(measurements_sem);
  measurements = d;
  signal(measurements_sem);
}

static void task3(void)  /* Angle computation */
{
  double a;
  sensors_data d;

  for (;;)
    {
      wait(measurements_sem);
      d = measurements;
      signal(measurements_sem);
      !! depending on d and the current value of in_alert_mode,
      !! compute a new angle in a

       wait(angle_sem);
       valve_angle = a;
       in_alert_mode = 0;
       signal(angle_sem);
    }
}

interrupt void task4(void)  /* Button monitoring */
{
  !! read button, and update in_alert_mode

  if (in_alert_mode)
     {
       kill(task3);
       create_one_shot_task(task3, 1);
     }
}
```

```
void main(void)
{
  !! initialize OS
  !! initialize data structures

  create_periodic_task(task1, 100, 3);
  create_periodic_task(task2, 1000, 2);
  create_one_shot_task(task3, 1);

  !! configure timer for triggering an interrupt every 40 ms,
  !! calling task4

  angle_sem = create_binary_semaphore(1);
  measurements_sem = create_binary_semaphore(1);

  enable();  /* User-programmed interrupts */

  !! start tasks sequencing
}
```

# Problem 2

A quadcopter contains a microcontroller that controls its four motors. This microcontroller is responsible for stabilizing the spatial position and orientation of the aircraft during flight, and for processing the orders sent by the pilot via a remote control. In order to do this, it performs the following tasks:

- Reading, processing and filtering data received from various sensors such as accelerometers and gyroscopes. This task has to be performed at a rate of 200 Hz, and takes 2 ms.
- Implementing a control loop. This task has to be performed at a rate of 100 Hz, and takes 1 ms.
- Communicating with the remote control. This task has to be performed at a rate of 50 Hz, and takes 0.2 ms.
- Writing a flight log in flash memory. This task has to be performed at a rate of 10 Hz, and takes 15 ms. The operations carried out by the task essentially amount to waiting for the flash memory component to trigger an interrupt signalling the end of the write operation.

1. What is the best software architecture for this system? (Carefully justify your answer.)
2. Using pseudocode, give the global structure of this embedded software.

## List of tasks:

- $\tau_1$: Acquiring and processing data: period = 5 ms, exec. time = 2 ms.
- $\tau_2$: Control loop: period = 10 ms, exec. time = 1 ms.
- $\tau_3$: Remote control: period = 20 ms, exec. time = 0.2 ms.
- $\tau_4$: Logging: period = 100 ms, duration = 15 ms, exec. time = $\varepsilon$.

## Can some tasks be performed by interrupt routines?

Yes: $\tau_4$. (The time-consuming work of that task is performed by a peripheral.)

## Is preemption needed?

No. Invoking Tasks $\tau_1$, $\tau_2$ and $\tau_3$, and starting $\tau_4$, can follow a fixed timing.

$\rightarrow$ Round-robin with interrupts.

## Task/task and task/interrupt routine communication:

Global variables:

- Counter for a single timer with a 2.5 ms period.
- Flag for signaling new value of counter.
- Current sensors data (written by $\tau_1$ and read by $\tau_2$ and $\tau_4$).
- Current mode of operation (modified by $\tau_3$ and read by $\tau_2$ and $\tau_4$).

```
#include "types.h"
#include "datastruct.h"

static volatile UINT4 counter = 0;
static volatile BOOL counter_flag = 0;
static sensors_data current_data;
static mode_of_operation current_mode;

static void task1(void)  /* Data acquisition and processing */
{
  !! acquire data, process it, and write
  !! the result in current_data
}

static void task2(void)  /* Control loop */
{
  !! read current_data and current_mode, and
  !! perform control computations
}
```

```
static void task3(void)  /* Remote control */
{
  !! check for incoming data from the remote
  !! connection, and update current_mode
}

static void task4(void)  /* First operations of logging */
{
  !! read current_data and current_mode, and
  !! start writing in flash memory
}

interrupt void timer1_isr(void)  /* 2.5 ms timer */
{
  counter++;
  counter_flag = 1;
}

interrupt void flash_isr(void)  /* End of flash write */
{
  !! finish the log writing operations
}
```

```
void main(void)
{
  UINT4 c;

  !! initialize global data structures

  !! configure timer1 interrupt (calling timer1_isr()),
     with period 2.5 ms

  !! enable flash memory interrupt, calling flash_isr()

  enable(); /* Global interrupts */

  for (;;)
    {
      if (!counter_flag)
         continue;

      counter_flag = 0;

      disable();
      c = counter;
      enable();
      if (!(c % 2))
        task1();

      if (c % 4 == 1)
        task_2();

     if (c % 8 == 3)
       task_3();

     if (c % 40 == 7)
       task_4();
    }
}
```

# Problem 3

A nanosatellite in charge of taking pictures of the earth is controlled by an embedded system. This system is equipped with a radio transceiver that receives telecommand data packets from a ground station. Upon receiving a data packet, the radio transceiver sends an interrupt request to the onboard microcontroller. Such requests are always separated by a delay of at least one second.

The microcontroller has to perform the following tasks:

- $\tau_1$ fetches and processes the data packets communicated by the radio transceiver.
- $\tau_2$ performs computations for estimating the position of the satellite, every 10 ms.
- $\tau_3$ regulates a DC/DC power converter, every 2 ms.
- $\tau_4$ acquires an image from a camera, after telecommand data requesting to perform this operation has been received from the ground station. It then performs image processing operations.

Tasks $\tau_1$ and $\tau_3$ execute in less than 0.1 ms; task $\tau_2$ needs 1 ms. The execution of task $\tau_4$ may require up to 500 ms of CPU time, depending on the image processing operations that must be carried out.

1. What is the best software architecture for this system? Justify.
2. Using pseudocode, give the global structure of this software.

## List of tasks:

- $\tau_1$: Receiving and processing packets: triggered by an interrupt, period $\geq$ 1000 ms, exec. time $<$ 0.1 ms.
- $\tau_2$: Position computation: period = 10 ms, exec. time = 1 ms.
- $\tau_3$: DC/DC control loop: period = 2 ms, exec. time $<$ 0.1 ms.
- $\tau_4$: Image acquisition and processing: period = ? ($>$ 1000 ms, triggered by data received by $\tau_1$), exec. time $\leq$ 500 ms.

## Can some tasks be performed by interrupt routines?

Partially: $\tau_1$ is triggered by an interrupt, but some of its operations (processing) are not urgent.

## Is preemption needed?

Yes! ($\tau_1$, $\tau_2$ and $\tau_3$ over $\tau_4$.)

$\rightarrow$ RTOS.

## Task priorities?

$P(\tau_3) > P(\tau_2) > P(\tau_1) > P(\tau_4)$.

## Tasks communication:

- Message queue between the urgent and non-urgent parts of $\tau_1$, for the data to be processed.
- Global variable for the current mode of operation (updated by $\tau_1$ and read by $\tau_4$).
- Binary semaphore for controlling concurrent access to this variable.
- Binary semaphore for $\tau_1$ to wake up $\tau_4$ in order to take an image.

```
#include <rtos.h>
#include <rtos-semaphores.h>
#include <rtos-queues.h>
#include "datastruct.h"

static volatile mode_of_operation current_mode;
static semaphore mode_sem, take_image_sem;
static queue packet_queue;

static void task1(void)  /* Processing packets */
{
  data_packet p;

  for (;;)
    {
      p = queue_receive(packet_queue, BLOCKING);

      !! process data packet in p, and update
      !! current_mode

      if (!! image must be taken)
        signal(take_image_sem);
    }
}
```

```
interrupt void com_isr(void)  /* Urgent part of task1 */
{
  data_packet p;

  !! receive packet into p

  queue_send(packet_queue, p, NON_BLOCKING);
}

static void task2(void)  /* Position computation */
{
  !! compute position
}

static void task3(void)  /* DC/DC control loop */
{
  !! control DC/DC variables
}
```

```
static void task4(void)  /* Image acquisition and processing */
{
  for (;;)
    {
      wait(take_image_sem);

      !! take image
      !! process image
    }
}

void main(void)
{
  !! initialize OS
  !! initialize data structures

  create_one_shot_task(task1, 2);
  create_periodic_task(task2, 10, 3);
  create_periodic_task(task3, 2, 4);
  create_one_shot_task(task4, 1);

  !! configure communication interrupt, calling com_isr()
  mode_sem = create_binary_semaphore(1);
  take_image_sem = create_binary_semaphore(0);
  packet_queue = create_message_queue(sizeof(data_packet));

  enable();  /* User-programmed interrupts */

  !! start tasks sequencing
}
```

# Scheduling problems

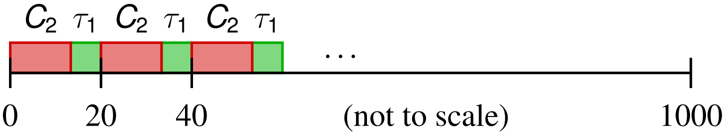
# Problem 1

Let $\tau_1$ and $\tau_2$ be two periodic tasks for which the execution times and periods are respectively $C_1, C_2$ and $T_1, T_2$.

If $T_1 = 1$ ms and $T_2 = 20$ $\mu$s, under which conditions is this pair of tasks schedulable ? Justify.

$T_1$ : 1000 $\mu$s
$T_2$ : 20 $\mu$s

$$P_2 > P_1$$



$T_1$ is an integer multiple of $T_2$

$$\Rightarrow \quad C_2 \leq 20 \ \mu\text{s}$$

$$C_1 + 50 C_2 \leq 1000 \ \mu\text{s}$$

# Problem 2

Consider the following set of periodic tasks $\tau_i = (C_i, T_i)$:

$$\{\tau_1 = (3, 7),\ \tau_2 = (\alpha, 6),\ \tau_3 = (1, 10)\},$$

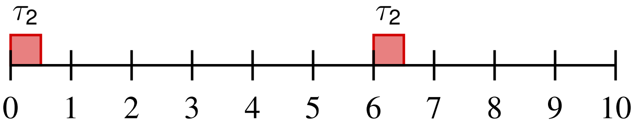
where $\alpha$ is a parameter.

1. Compute the maximum value of $\alpha$ for this set of tasks to be schedulable.
2. Verify your answer with a graphical simulation.

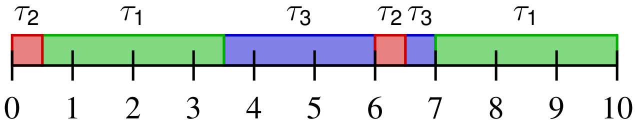
$$\begin{array}{llll} T_1: & 7 & C_1: & 3 \\ T_2: & 6 & C_2: & \alpha \\ T_3: & 10 & C_3: & 1 \end{array}$$

$$P_2 > P_1 > P_3$$

Case 1: $\tau_2$ finishes before $t = 10$ ($\alpha \leq 4$):



## Case 1.1: $\tau_2$ finishes before $t = 7$ ($\alpha \leq 1$):

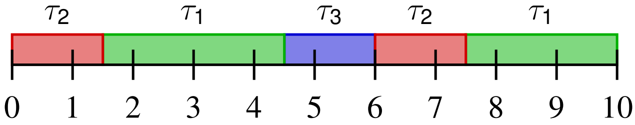


$$C_3 = 1 = 7 - 2 \times \alpha - 1 \times 3$$

$$\Rightarrow \; \alpha = \frac{3}{2}$$

(Contradiction!)

## Case 1.2: $\tau_2$ finishes at or after $t = 7$ ($1 \leq \alpha \leq 4$):

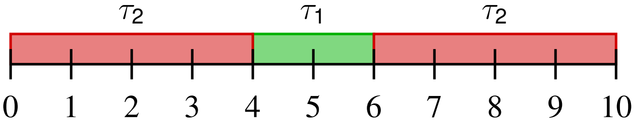


$$C_3 = 1 = 6 - \alpha - 3$$

$$\Rightarrow \ \alpha = 2$$

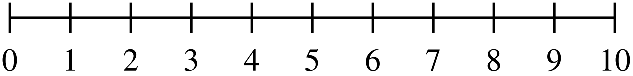
(OK)

Case 2: $\tau_2$ finishes at or after $t = 10$ ($\alpha \geq 4$):

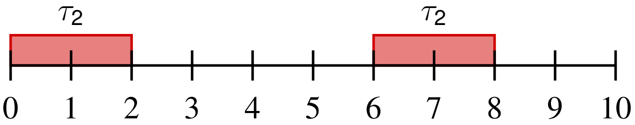


Impossible! (Not enough time left for $\tau_1$ and $\tau_3$.)

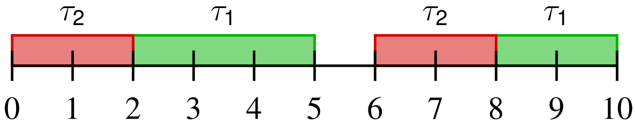
Simulation with $\alpha = 2$:



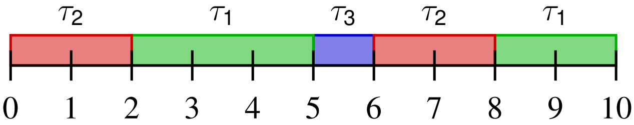
Simulation with $\alpha = 2$:



Simulation with $\alpha = 2$:



Simulation with $\alpha = 2$:


$\tau_2$
$\tau_1$
$\tau_3$
$\tau_2$
$\tau_1$
0
1
2
3
4
5
6
7
8
9
10