Name: Elena Mihalas
Course: Computer Vision

ASSIGNEMENT 1

```
/*I have defined the two processes: one for setting the threshold by averaging the first 10
measurements, one for veryfing the temperature and light values and for sending an alarm
notification in case of fire detection. The monitoring of temperature and light was set to be variable
in order to save energy. I have also tried to implement the request of re-calibration through the
button pressing but I could not understand why it did not work  */

#include "contiki.h"
#include "dev/light-sensor.h" /*For the light sensor*/
#include "dev/sht11-sensor.h" /*For the temperature sensor*/
#include "dev/leds.h"         /*For leds management*/
#include <stdio.h>            /*For printf()*/
#include "etimer.h"           // For timer management
#include "lib/random.h"       /*For the library dealing with random numbers*/
//#include "dev/button-sensor.h" /*For the user button driver*/



/*Digits before decimal point*/
unsigned short ip(float f)
{
  return((unsigned short)f);
}

/*Digits after decimal point*/
unsigned short fp(float f)
{
  return(1000*(f-ip(f)));
}

/************************************************/

PROCESS(set_threshold, "Calibration stage");

PROCESS(fire_alarm, "Fire Alarm");

/*************************************************/

AUTOSTART_PROCESSES(&set_threshold,&fire_alarm);

static process_event_t event_data_ready;
//static process_event_t event_button;

  static float buffer[2];  // I declare "buffer" as a global variable in order to be able to use it inside the
                                                                                              second Process_Thread

/************************************************/
```

```
PROCESS_THREAD(set_threshold, ev, data)
{
  static struct etimer timer;

/*************************/
  PROCESS_BEGIN();

 event_data_ready = process_alloc_event();  /* Allocates the an event in "event_data_ready" in order
                                                                  to post it later to fire_alarm process*/


  SENSORS_ACTIVATE(light_sensor);
  SENSORS_ACTIVATE(sht11_sensor);
  //SENSORS_ACTIVATE(button_sensor);



  float temp, lx;
  static int i;

  for(i=0; i<10; i++ )
          {
           etimer_set(&timer, CLOCK_SECOND*1);
            PROCESS_WAIT_EVENT_UNTIL(ev=PROCESS_EVENT_TIMER); // when taking into
                                                                account the button-sensor it has
                                                                   to be added the condition
                                                                        "ev==event_button"

           temp = temp+(0.04*sht11_sensor.value(SHT11_SENSOR_TEMP)-39.6);
          /*Obtain the sum of temperature values after 10sec*/

           float V_sensor = 1.5*light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC)/4096;
           /*Get the voltage value from the light sensor*/
           float I = V_sensor/1e5;
           /*Convert the voltage into current*/
           lx = lx+(0.625*1e6*I*1e3);
           /*Obtain the sum of lux values after 10sec*/

           etimer_reset(&timer);
          }
          buffer[0]=temp/10;
          buffer[1]=lx/10;

  printf("temperature threshhold: %u.%u\n", ip(buffer[0]), fp(buffer[0]));
  printf("light threshhold: %u.%u\n", ip(buffer[1]), fp(buffer[1]));

   process_post(&fire_alarm, event_data_ready, &buffer);  // Post an event to the  fire_alarm process
                                                                     and pass a pointer to buffer


  PROCESS_END();
}
```

```
/*******************************************************/
/*******************************************************/

PROCESS_THREAD(fire_alarm,ev,data)
{
 static struct etimer et;

/**************************/

 PROCESS_BEGIN();

 event_button = process_alloc_event();  // Allocates the event of the button-sensor changed value

 SENSORS_ACTIVATE(light_sensor);
 SENSORS_ACTIVATE(sht11_sensor);
 //SENSORS_ACTIVATE(button_sensor);
 leds_off(LEDS_ALL);

 PROCESS_WAIT_EVENT_UNTIL(ev==event_data_ready);      /* Wait for the data being sent by
                                                                    the set_threshold process*/

      float delay=5*((float)random_rand()/RANDOM_RAND_MAX)+5;
       etimer_set(&et, CLOCK_SECOND*delay); /*Set a 5 to 10sec delay in order to reduce energy
                                                                              consumption*/

 while (1)  // case: button_sensor ->  while (ev != sensors_event && data != &button_sensor)
    {
                                       //printf("The data are monitored\n"); -> can be used in order
                                                          to check if the while loop works properly


        PROCESS_WAIT_UNTIL(etimer_expired(&et)); // Start the process after  the delay period


        float temperature=0.04*sht11_sensor.value(SHT11_SENSOR_TEMP)-39.6;
       /*Read the temperature value in Celsius -> the value
        takes 14bits of memory and is taken at 3V*/

        float V_sensor=1.5*light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC)/4096;
        /*Get the voltage value from the light sensor*/
        float I=V_sensor/1e5;
        /*Convert the voltage into current*/
        float light_lx=0.625*1e6*I*1e3;
        /*Obtain the lux value of the light sensor*/

            if(temperature > (buffer[0]+50) && light_lx > (buffer[1]+250))  // Set the limits for the
                                                                              alarm notification
      {
             printf("---ALARM! FIRE!---\n");   /*Write on the screen ALARM!*/

             leds_on(LEDS_ALL);
```

```
                 leds_toggle(LEDS_RED);        /*Set the blinking of the leds*/
               clock_delay((unsigned int)1e9); /*I apply a very high value in order to make visible the
                                                                                  blinking of the leds*/
                 leds_toggle(LEDS_RED);

                 leds_toggle(LEDS_GREEN);
                 clock_delay((unsigned int)1e9);
                 leds_toggle(LEDS_GREEN);


          }
          leds_off(LEDS_ALL);
          etimer_reset(&et);
      }

         /*if (ev == sensors_evenT && data == &button_sensor)
         {
         process_post(&set_threshold, event_button, &button_sensor);
         // Post an event to the  set_threshold process and pass a pointer to data
         process_start(&set_threshold, NULL);
         }*/

         PROCESS_END();
}

/*************************************************/
```

ASSIGNEMENT 2

```
/* This program implemets an algorithm to aggregate the sensor data data based on activity
changes. The sensors readings are stored continously in a buffer and then their activity is evaluted
by computing the standard deviation of the measurements. If there is a low deviation all the values
are summed up to one value, otherwise the data is aggregated any 4 values, namely the buffer's size
becomes three. */


#include "contiki.h"
#include "etimer.h"
#include <stdio.h>
#include "dev/light-sensor.h"

#define size 12

/*Digits before decimal point*/
unsigned short ip(float f)
{
  return((unsigned short)f);
}

/*Digits after decimal point*/
unsigned short fp(float f)
```

```
{
 return(1000*(f-ip(f)));
}


/*Square root function obtained through the Newton's Method*/

/*Newton's method assumes the function f to have a continuous derivative.
 Newton's method may not converge if started too far away from a root.
 However, when it does converge, it is faster than the bisection method, and is usually quadratic*/

static float sqrt(float x) {

    if(x < 0)    // Checks if the value is negative
     {
      return -1;
     }
    if(x == 0 || x == 1) // Doesn't change the number when it is NULL or ONE
      {
       return x;
      }

    float guess = 1.0;   // Guesses the solution -> by default I give it value 1

    float diff =  guess*guess - x;   // Checks if we get closer to the solution: root*root-x=0

         while(diff*diff > 0.0000001)   //  As diff is a negative number I take the square of it,therefore I
                                                                  increase the precision of my solution
                                                                                                          {

        guess = guess - (guess*guess - x)/(2*guess);  // First iteration of Newton's method
        diff= guess*guess - x;    // Redefines the diff value

        /* guess' value updates with each iteration */
       }

    return guess;
   }


/*******************************************************/

PROCESS(aggregation, "Aggregation Algorithm");

AUTOSTART_PROCESSES(&aggregation);
/*******************************************************/

PROCESS_THREAD(aggregation,ev,data)
{
static struct etimer timer;

/****************/
```

```
PROCESS_BEGIN();

  SENSORS_ACTIVATE( light_sensor );    // Activates the light sensor


  static int i=0, j=0;        // Declares all the local variables
  static float buffer[size];
  static float buffer_dev[size];
  static float activity[3];
  static float sum=0.0, sum_dev=0.0, sum1=0.0;
  static float lux, average=0.0, dev_std=0.0;


while(1)
{
  etimer_set(&timer,CLOCK_CONF_SECOND);  // Sets the timer
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

     /*Get the voltage value from the light sensor*/
     float V_sensor=1.5*light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC)/4096;
     /*Convert the voltage into current*/
     float I=V_sensor/1e5;
     lux=0.625*1e6*I*1e3;

     buffer[i]=lux;  // Inserts the values inside a buffer of size 12
     sum = sum + lux;      // Sums up the first 10 values

     i= i + 1;

     //printf("value_lux: %u.%u\n", ip(lux), fp(lux) );

  if (i==size)     // Once the buffer is fill up the activity level starts to be computed
     {
                                                                  //printf("Buffer content \n");
          average=sum/size;   // Computes the average of the buffer

          for (j=0; j<size; j++)
          {
                    buffer_dev[j]=(buffer[j]-average)*(buffer[j]-average);  // Substractes the average
                                                            from each element inside the buffer
                    sum_dev+=buffer_dev[j];    // Sums up the new values inside the buffer
          }

          float arg=sum_dev/size;     // Averages the sum_dev

                                                  //printf("value_arg: %u.%u\n", ip(arg), fp(arg) );

/* Standard Deviation value __> any inaccuracy in the value of the square root is due to the
representation functions "ip" and "fp" and not to the sqrt function. For example: 3.007 will be
represented as 3.7 */
```

```
        dev_std = sqrt(arg) ;


        printf("Standard deviation value: %u.%u\n", ip(dev_std), fp(dev_std) );


        if (dev_std<=10)    // Analysing the 2 cases of activity level
      {
        printf("The activity is low. The buffer holds the value: %u.%u\n", ip(average), fp(average) );
        activity[0]=average;  // If the activity is low, the average of buffer is printed
      }
        else {
                for(i=1; i<=3; i++){
                  for(j=0; j<i*4; j++){
                        sum1+=buffer[j];
                    }
                  activity[i]=sum1/4;
                  sum1=0;

                  // If the activity is high, the values are aggregated and the buffer's size becomes 3
                  printf("The activity is high. The buffer holds the values: %i -> %u.%u\n",
                                             i, ip(activity[i]), fp(activity[i]) );
                }
            }
        i=0;
        sum_dev=0;
        sum=0;

    }
  etimer_reset(&timer);
}

PROCESS_END();

}

/***************************************************************************/
```