Internet of Things Assignment Code

Aggregation Algorithm

Faiz Ahmad Khan

University of Surrey

2021/2022

Contents:

```c
/*------------------------- IOT Assignment --------------------------------*/

//----- Section-1: Libraries
#include "contiki.h"
#include "dev/light-sensor.h"
#include "dev/sht11-sensor.h"
#include <stdio.h>

/*-------------------------------------------------------------------------*/

//----- Section-2: Constants
#define Advance_Feature 1
/* 1- Run advance feature
   0- Skip advance feature */
#define BUFFER_SIZE 12
#define K 12
// For Cooja Sim
#define Light_Lower_Threshold 400
#define Light_Upper_Threshold 1000

/*-------------------------------------------------------------------------*/

//----- Section-3: Functions, Structures & Global Variables
static process_event_t Buffer_full_event; // Event for passing full Buffers

// Buffer Structure
typedef struct Buffer {
        float Array[BUFFER_SIZE];
        int Length;
        float Mean;
        float SD;
} Buffer;

// Read Temperature function
float getTemperature(void)
{
        int tempData;
        tempData = sht11_sensor.value(SHT11_SENSOR_TEMP_SKYSIM); // For Cooja Sim
        float so_temp = tempData;
        // transfer function
        float temp = (0.04 * so_temp) - 39.6;
        return temp;
}

// Read Light function
float getLight(void)
{
        float lightData = light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC);
        // transfer function
        float V_sensor = (1.5 * lightData) / 4096;
        float I = V_sensor / 100000;
        float light_lx = 0.625 * 1e6 * I * 1000;
        return light_lx;
}

// Calculate mean function
float getmean(float arr[],int length)
{
        float sum = 0;
        int i;
        for(i=0; i<length; i++)
        {
                sum += arr[i];
        }
        float mean;
        mean = sum / length;
        return mean;
}

// Calculate Standard Deviation function
float getSD(float arr[],int length,float mean)
{
        int i;
        float deviation;
        float sqdev = 0 ;
        for(i=0; i<length; i++)
        {
                deviation = arr[i] - mean;
                sqdev += deviation*deviation;
        }
        float SD;
        SD = sqrt(sqdev/length);
        return SD;
}

// Calculate square root function
float sqrt(float num)
{
        float sqrt = num/2;
        float t = 0;
        while(sqrt != t)
        {
                t = sqrt;
                sqrt = (num/t + t) / 2;
        }
        return sqrt;
}

// Convert Float to Int function
int d1(float f) // Integer part
{
        return((int)f);
}

unsigned int d2(float f) // Fractional part
{
        if (f>0)
        {
                return(100*(f-d1(f)));
        }
        else
        {
                return(100*(d1(f)-f));
        }
```

```
}

/*---------------------------------------------------------------------------*/

//----- Section-4: Start Processes
PROCESS(sensor_reading_process, "Sensor reading process");
PROCESS(aggregation_process, "Aggregation process");
AUTOSTART_PROCESSES(&sensor_reading_process,&aggregation_process);

/*---------------------------------------------------------------------------*/

//----- Section-5: Sensor Reading Process
PROCESS_THREAD(sensor_reading_process, ev, data)
{
        static struct etimer timer;

        PROCESS_BEGIN();

        // Set timer
        etimer_set(&timer, CLOCK_SECOND/2);

        // Activate sensors
        SENSORS_ACTIVATE(light_sensor);
        SENSORS_ACTIVATE(sht11_sensor);

        // Allocate global event number
        Buffer_full_event = process_alloc_event();

        // Buffers
        static struct Buffer Light_Buffer;
        static struct Buffer Temp_Buffer;

        static int counter = 0;
        while(1)
        {
                PROCESS_WAIT_EVENT_UNTIL(ev=PROCESS_EVENT_TIMER);

                // Take readings and print
                float temp = getTemperature();
                float light_lx = getLight();
                Light_Buffer.Array[counter] = light_lx;
                Temp_Buffer.Array[counter] = temp;

                printf("\nLight = %d.%u lux\n",d1(light_lx), d2(light_lx));
                printf("Temp = %d.%u C\n ",d1(temp), d2(temp));

                counter++;
                if (counter == K)
                {
                        // Get Length of Buffer
                        Light_Buffer.Length = sizeof(Light_Buffer.Array)/sizeof(Light_Buffer.Array[0]);
                        Temp_Buffer.Length = sizeof(Temp_Buffer.Array)/sizeof(Temp_Buffer.Array[0]);

                        int i;

                        // Print Light Bufffer
                        printf("\nLight Buffer = [ ");
                        for(i=0; i<K; i++)
                        {
                                printf("%d.%u",d1(Light_Buffer.Array[i]), d2(Light_Buffer.Array[i]));
                                if(i<(K-1))
                                {
                                        printf(", ");
                                }
                        }
                        printf(" ] ");

                        // Print Temperature Buffer
                        printf("\nTemperature Buffer = [ ");
                        for(i=0; i<K; i++)
                        {
                                printf("%d.%u",d1(Temp_Buffer.Array[i]), d2(Temp_Buffer.Array[i]));
                                if(i<(K-1))
                                {
                                        printf(", ");
                                }
                        }
                        printf(" ] \n");

                        // Get mean
                        Light_Buffer.Mean = getmean(Light_Buffer.Array, K);
                        Temp_Buffer.Mean = getmean(Temp_Buffer.Array, K);

                        // Get Standard Deviation
                        Light_Buffer.SD = getSD(Light_Buffer.Array, K, Light_Buffer.Mean);
                        Temp_Buffer.SD = getSD(Temp_Buffer.Array, K, Temp_Buffer.Mean);

                        // Print Standard Deviation
                        printf("\nStandard Deviation of Light = %d.%u ",d1(Light_Buffer.SD), d2(Light_Buffer.SD));
                        printf("\nStandard Deviation of Temperature = %d.%u \n",d1(Temp_Buffer.SD), d2(Temp_Buffer.SD));

/*---------------------------------------------------------------------------*/

                        //----- Section-6: Advance Feature
                        /* To find Linear Regression using Least Squares
                        and the Pearson Correlation Coefficient.
                        y = mx + c
                        y -> Dependant Variable (Temperature)
                        x -> Independant Variable (Light)
                        m -> Slope
                        c -> Intercept

                        r = Covariance(x,y) / SD(x) * SD(y)
                        Covariance(x,y) -> Mean(x*y) - Mean(x) * Mean(y)
                        r -> Pearson correlation coefficient */

                        if(Advance_Feature == 1)
                        {
                                printf("\nAdvance Feature\n");
                                // Linear Regression
                                float m;
                                float c;
                                float num = 0; // numerator
                                float den = 0; // denominator
                                for(i=0; i<K; i++)
```

```
                                {
                                        num += (Light_Buffer.Array[i] - Light_Buffer.Mean) * (Temp_Buffer.Array[i] - Temp_Buffer.Mean);
                                        den += (Light_Buffer.Array[i] - Light_Buffer.Mean) * (Light_Buffer.Array[i] - Light_Buffer.Mean);
                                }
                                m = num / den;
                                c = Temp_Buffer.Mean - m * Light_Buffer.Mean;
                                printf("Slope (m) = %d.%u \n",d1(m), d2(m));
                                printf("Intercept (c) = %d.%u \n",d1(c), d2(c));

                                // Pearson correlation coefficient
                                float r;
                                float xy[K];
                                for(i=0; i<K; i++)
                                {
                                        xy[i] = Light_Buffer.Array[i] * Temp_Buffer.Array[i];
                                }
                                float mean_xy; // Mean of Product of Vectors
                                mean_xy = getmean(xy,K);
                                float cov; // Covariance
                                cov = mean_xy - (Light_Buffer.Mean * Temp_Buffer.Mean);
                                r = cov / (Light_Buffer.SD * Temp_Buffer.SD);
                                printf("Pearson correlation coefficient (r) = %d.%u \n",d1(r), d2(r));
                        }

/*---------------------------------------------------------------------------*/

                        // Send Buffer to Aggregation Process
                        process_post(&aggregation_process, Buffer_full_event, &Light_Buffer);
                        counter = 0;
                }
                etimer_reset(&timer);
        }
        PROCESS_END();
}

/*---------------------------------------------------------------------------*/

//----- Section-7: Aggregation Process
PROCESS_THREAD(aggregation_process,ev,data)
{
        PROCESS_BEGIN();

        while(1)
        {
                PROCESS_WAIT_EVENT_UNTIL(ev == Buffer_full_event); //Wait for Buffer to fill
                Buffer Light_Buffer = *(Buffer *)data; // Cast to a Buffer pointer

                // High Activity No Aggregation
                if(Light_Buffer.SD > Light_Upper_Threshold)
                {
                        printf("\nAggregation = No Aggregation \n");
                        // Print Output Buffer
                        printf("Light_X = [ ");
                        int i;
                        for(i=0; i<K; i++)
                        {
                                printf("%d.%u",d1(Light_Buffer.Array[i]), d2(Light_Buffer.Array[i]));
                                if(i<(K-1))
                                {
                                        printf(", ");
                                }
                        }
                        printf(" ] \n");
                }

                // Some Activity 4-into-1 Aggregation
                else if(Light_Buffer.SD > Light_Lower_Threshold && Light_Buffer.SD <Light_Upper_Threshold)
                {
                        printf("\nAggregation = 4-into-1 \n");
                        // For K less than 4, aggregate entire buffer
                        if(K<4)
                        {
                                printf("4-into-1 Aggregation not possible!");
                                printf("\nAggregation = K-into-1 \n");
                                float Output_Buffer[1];
                                Output_Buffer[0] = Light_Buffer.Mean;
                                // Print Output Buffer
                                printf("Light_X = [ ");
                                printf("%d.%u ] \n",d1(Output_Buffer[0]), d2(Output_Buffer[0]));
                        }
                        // For K equal to 4 or greater
                        else
                        {
                                int Q = K / 4; // Quotient
                                int R = K % 4; // Remainder
                                float sum;
                                float mean;
                                int j = 0;
                                int i;
                                // For values of K, divisible by 4
                                if(R == 0)
                                {
                                        float Output_Buffer[Q];
                                        while(j<K)
                                        {
                                                // Mean of 4 values set to appropriate slot in output buffer
                                                sum = Light_Buffer.Array[j]+Light_Buffer.Array[j+1]+Light_Buffer.Array[j+2]+Light_Buffer.Array[j+3];
                                                mean = sum / 4;
                                                Output_Buffer[j/4] = mean;
                                                j = j+4;
                                        }
                                        // Print Output Buffer
                                        printf("Light_X = [ ");
                                        for(i=0; i<Q; i++)
                                        {
                                                printf("%d.%u",d1(Output_Buffer[i]), d2(Output_Buffer[i]));
                                                if(i<(Q-1))
                                                {
                                                        printf(", ");
                                                }
                                        }
                                        printf(" ] \n");
                                }
                                // For values of K, not divisible by 4
                                else
```

```
                                {
                                        // Adjust Output Buffer to include extra readings within Light Buffer
                                        Q = Q + 1;
                                        float Output_Buffer[Q];
                                        while(j<(K-R))
                                        {
                                                // Mean of 4 values set to appropriate slot in output buffer
                                                sum = Light_Buffer.Array[j]+Light_Buffer.Array[j+1]+Light_Buffer.Array[j+2]+Light_Buffer.Array[j+3];
                                                mean = sum / 4;
                                                Output_Buffer[j/4] = mean;
                                                j = j+4;
                                        }
                                        if(R == 1)
                                        {
                                                // Insert final value of Light Buffer into Output Buffer
                                                Output_Buffer[Q-1] = Light_Buffer.Array[K-1];
                                        }
                                        else if(R == 2)
                                        {
                                                // Insert mean of last 2 values of Light Buffer into Output Buffer
                                                sum = Light_Buffer.Array[K-1] + Light_Buffer.Array[K-2];
                                                mean = sum / R;
                                                Output_Buffer[Q-1] = mean;
                                        }
                                        else if(R == 3)
                                        {
                                                // Insert mean of last 3 values of Light Buffer into Output Buffer
                                                sum = Light_Buffer.Array[K-1] + Light_Buffer.Array[K-2] + Light_Buffer.Array[K-3];
                                                mean = sum / R;
                                                Output_Buffer[Q-1] = mean;
                                        }
                                        // Print Output Buffer
                                        printf("Light_X = [ ");
                                        for(i=0; i<Q; i++)
                                        {
                                                printf("%d.%u",d1(Output_Buffer[i]), d2(Output_Buffer[i]));
                                                if(i<(Q-1))
                                                {
                                                        printf(", ");
                                                }
                                        }
                                        printf(" ] \n");
                                }
                        }
                }

                // No Activity K-into-1 Aggregation
                else
                {
                        printf("\nAggregation = K-into-1 \n");
                        float Output_Buffer[1];
                        Output_Buffer[0] = Light_Buffer.Mean;
                        // Print Output Buffer
                        printf("Light_X = [ ");
                        printf("%d.%u ] \n",d1(Output_Buffer[0]), d2(Output_Buffer[0]));
                }
                printf("-------------------------------------------\n");
        }
        PROCESS_END();
}

/*---------------------------------------------------------------------------*/
```