Module: Internet of Things **2024/2025**

Assignment: Aggregation Algorithm

1.1 Introduction

In this assignment you need to implement an aggregation algorithm to aggregate the sensor data based on the changes in the data (i.e. activity).

In other words, in times of high activity (i.e. more frequent changes in the data), the data should be less aggregated to capture the changes that has occurred in the data.

In times of low activity, more samples should be aggregated into one value to save communication bandwidth and save more energy on battery powered nodes.

1.2 Specification

There are three main parts:

- Storage
    - Read and store the meaningful light sensor data of a node continuously in a buffer at a rate of 2 readings per second.
    - The buffer size is set to 12. The buffer can be implemented as a First-In-First-Out (FIFO) Buffer.
- Activity Measurement
    - Measure changes of the data in the buffer using the standard deviation. Assume that a low deviation implies no interesting activities, and high deviation implies high level of activity (e.g. human movement causing changing of light levels).
    - Use reasonable threshold settings to classify activity levels.
    - The measurement may happen after collecting k readings. Set k=1 if you want to perform measurement for every reading for fast reaction. Set k=12 if you want to perform measurement after collecting all 12 readings. You may choose your own k setting. Note that the implementation of FIFO buffer will be necessary if k<12.
- Aggregation and Reporting
    - After the activity measurement, an appropriate data aggregation is performed before the reporting.
    - If there no interesting activity, you can aggregate the entire buffer of 12 data into a single average value.
    - In case there is some level of activity, aggregation of every 4 data into 1 should be used.
    - For high activity, no aggregation should be used.
    - The program reports the outcome by showing the buffers on the terminal.
- Symbolic Aggregate Approximation (SAX)
    - Transform the light sensor data in the buffer of length 12 into the strings of 4 fragments, using an alphabet of size 4, i.e., A B C D. One possible output, depending on the data, of course, can be similar to [B A D D].

You must follow the above specification.

For example, say we have collected readings to fill the entire buffer, B:

B = [0, 0, 3, 0, 2, 0, 0, 4, 0, 0, 3, 2]

The standard deviation is about 1.5 which is deemed no interesting activity. Therefore we produce the following output buffer, X, using 12-into-1 aggregation:

X = [1.1]

With some activity, we may get the following buffer:

B = [10, 2, 12, 2, 19, 1, 19, 100, 6, 4, 66, 33]

The standard deviation is about 29 which indicates some activity. Therefore we perform 4-into-1 aggregation to produce the following output buffer:

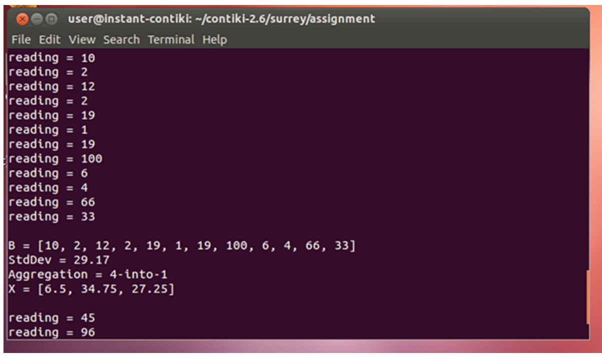
X = [6.5, 34.75, 27.25]

1.3 Screen snapshot of your program

When producing output to the screen, you should format the output as follows:

```
B = [10, 2, 12, 2, 19, 1, 19, 100, 6, 4, 66, 33]
StdDev = 29.17
Aggregation = 4-into-1
X = [6.5, 34.75, 27.25]
```

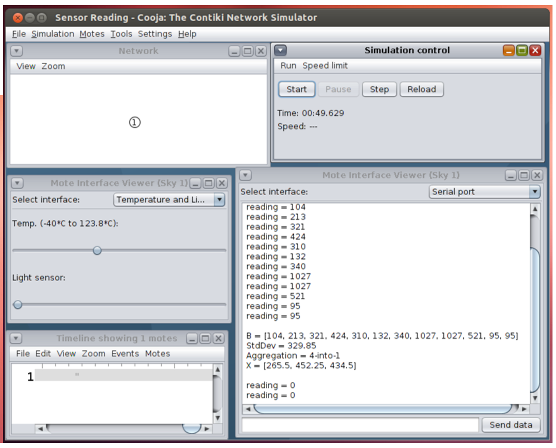
See below for some samples from the terminal (if you are using a mote) or Cooja (if you are using Cooja simulation):


user@instant-contiki: ~/contiki-2.6/surrey/assignment
File Edit View Search Terminal Help

```
reading = 10
reading = 2
reading = 12
reading = 2
reading = 19
reading = 1
reading = 19
reading = 100
reading = 6
reading = 4
reading = 66
reading = 33

B = [10, 2, 12, 2, 19, 1, 19, 100, 6, 4, 66, 33]
StdDev = 29.17
Aggregation = 4-into-1
X = [6.5, 34.75, 27.25]

reading = 45
reading = 96
```


Sensor Reading - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start
Pause
Step
Reload
Mote Interface Viewer (Sky 1)
Select interface:
Temperature and Li...
Temp. (-40°C to 123.8°C):
Light sensor:
Timeline showing 1 motes
File Edit View Zoom Events Motes
Serial port
Send data

```
Time: 00:49.629
Speed: ---
```

```
reading = 104
reading = 213
reading = 321
reading = 424
reading = 310
reading = 132
reading = 340
reading = 1027
reading = 1027
reading = 521
reading = 95
reading = 95

B = [104, 213, 321, 424, 310, 132, 340, 1027, 1027, 521, 95, 95]
StdDev = 329.85
Aggregation = 4-into-1
X = [265.5, 452.25, 434.5]

reading = 0
reading = 0
```

1.4 Submission Materials (on Week 9)

You must submit the following materials:

- Your source code in a single PDF file.
    - We will read your source code (rather than running it).
    - If you submit multiple files, we'll randomly pick one to read and ignore others.
    - If your file is not in PDF format, we'll open with our default application and read from the application. The formatting may look ugly which will affect the quality of the code presentation and readability.
    - You may use color or add line numbers they can improve readability.
    - Make your code readable. Some good practices are: use of meaningful variable names, adequate but not excessive commenting, consistent indentation, apply don't repeat yourself (DRY) principle, etc.

- Three screenshots in a single PDF file.
  - One screenshot for no aggregation (i.e. high activity)
  - One screenshot for 4-into-1 aggregation (i.e. some activity)
  - One screenshot for 12-into-1 aggregation (i.e. low activity)
- A one page description of any advanced feature.
  - The description should be clear and concise to help any reader understand the challenge and the effort involved.
  - We will assess advanced feature based on the challenges and output quality of the feature, rather than the writing of the description.
  - You should include screenshots to show evidence of your implementation. They should be provided after the 1-page description outside of the 1-page description. If you need to explain the screenshots, the explanation must be given within the 1-page description. We will ignore any text description after the first page.

Do not zip your submission. We expect 2 files (or 3 files if you include the 1-page advanced feature description).

1.5 Code Demonstration (on Week 10)

Additional to your submission, we also ask you to demonstrate the running of your code. This will happen in Week 10 during the 2-hour lab session.

On Week 10, you should:

- Attend the lab on time as usual
- Bring your lab materials (mote if you have collected one and your code). You are not allowed to participate without the materials
- Download the instruction from SurreyLearn
  - It will become visible at the beginning of the lab session
  - It contains a few additional tasks asking you to modify your code to produce some additional outputs
- Complete the tasks within the 2-hour session
- Submit the screen snapshots to SurreyLearn by the end of the session
- You may seek for clarification during the session
- You must apply for EC if you cannot participate

You must complete the code demonstration individually using your submitted code. Demonstrators can help to clarify the instruction, but they cannot assist you with coding or debugging.

1.6 Advanced Feature on Data Processing

The advanced feature **should not** replace the basic specification. If needed, you can maintain two source files, one for the required specified in the previous subsection, and another one contains the advanced feature. You should then print them in a single PDF file for submission.

In the advanced features, you need to get readings from the light sensor and temperature sensor and save them into two separate buffers (which can be viewed as vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ with X_i and Y_i representing the i-th measurement from the light and temperature sensor, respectively). Then, You are required to implement the following features:

1. Manhattan Distance of Two Vectors

2. Correlation of Two Vectors

4. Short-Time Fourier Transform (STFT) of vector $\boldsymbol{X}$

5. Spectral Entropy of vector $\boldsymbol{X}$

Some hints to compute these functions can be found below.

1. Manhattan Distance of Two Vectors

The Manhattan distance D between two vectors X and Y is defined as:

$$D(X,Y) = \sum_{i=1}^{n} |X_i - Y_i|$$

where n is the number of elements in each vector.

2. Correlation of Two Vectors

The correlation r between vectors X and Y is given by:

$$r(X,Y) = \sum_{i=1}^{n} \frac{(X_i - \mu_X)(Y_i - \mu_Y)}{\sigma_X \sigma_Y}$$

where μ_X and μ_Y are the means of vectors X and Y, and σ_X and σ_Y are their standard deviations.

3. Short-Time Fourier Transform (STFT)

The Short-Time Fourier Transform (STFT) is useful for analyzing non-stationary signals by providing time-localized frequency information.

We will split the n-elements vector $\boldsymbol{X}$, with $n = 12$, into $M = 5$ overlapped chunks, each containing $Z = 4$ samples, with the chunks overlapping by $L = 2$ samples and with hop size $\delta = Z - L$. The STFT of the signal $\boldsymbol{X}$ with a rectangular window is defined as:

$$S[m,k] = \sum_{z=0}^{Z-1} X[z + m\ \delta]\ e^{\left\{-j\left(\frac{2\pi kz}{N}\right)\right\}},$$

for $m = \{0,1,2,3,4\}$ and $k = \{0,1,2,3\}$.

4. Spectral Entropy

Spectral entropy quantifies the amount of information present in the frequency domain and helps to understand the complexity and randomness of the signal.

To compute spectral entropy, follow these steps:

1. Calculate the Power Spectrum:

$$P[m,k] = |S[m,k]|^2$$

2. Average Power Spectrum Over Time Segments:

$$\bar{P}[k] = \left(\frac{1}{M}\right) \sum_{m=0}^{M-1} P[m,k]$$

3. Calculate the Total Average Power:

$$\bar{P}_{total} = \sum_{k=0}^{Z-1} \bar{P}[k]$$

4. Calculate the Probability Density Function (PDF):

$$\bar{p}_k = \frac{\bar{P}[k]}{\bar{P}_{total}}$$

5. Calculate the Spectral Entropy:

$$H = -\sum_{k=0}^{Z-1} \bar{p}_k \log_2 \bar{p}_k$$

1.7 Marking Scheme

The assignment accounts for 20%. The following is the breakdown:

- #1 Implementation & Code Readability...6%
  - 6%: You have accurately implemented all specified features. Your code structure and its presentation are professional.
  - 5%-4%: Your implementation and your code presentation are good. However, you either missed some features, inaccurately implemented some features, or did not present your code professionally.

    - 3%-2%: There are obvious flaws in your implementation.
    - 1%-0%: You did not submit your code or there is no evidence that your code has accurately implemented any feature.
- #2 Output (screen snapshot)...2%
    - 2%: The outputs show accurate implementation.
    - 1%: You did not provide all necessary outputs or the format of the outputs are wrong.
    - 0%: You did not provide your screenshots or your screenshots contain no relevant information.
- #3 Advanced Features...6%
    - 6%: All features are implemented professionally.
    - 5%-3%: Somewhat challenging features are well implemented.
    - 2%-1%: Simple feature with straightforward implementation.
    - 0%: No implementation of advanced feature.
- #4 Two-Hour Code Demonstration...6%