Master's Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

UNSUPERVISED MACHINE LEARNING ALGORITHMS FOR EDGE NOVELTY DETECTION

Supervisors

Prof. Marcello CHIABERGE

Dott. Umberto ALBERTIN

Dott. Gianluca DARA

Candidate

Ariel PRIARONE

03 2024

Acknowledgements

I would like to thank the PoliTO Interdepartmental Centre for Service Robotics (PIC4Ser) for giving me the opportunity to work on this project. The guidance and infrastructure provided by the centre have been invaluable during the development of this work.

To my parents, who have given me everything.
Thank you for always making me believe that anything is possible.
Ai miei genitori, che mi hanno dato tutto.
Grazie per avermi sempre fatto credere che tutto sia possibile.

Ariel

Table of Contents

List of Tables

List of Figures

Chapter 1

Embedded implementation

In ??, an overview of the framework developed in `python` was given, relying on the MongoDB database. This chapter will focus on the implementation of the embedded system. The first big difference is that the embedded system is written in `C`, which is not an object-oriented language. The second big difference is that the embedded system does not use a database, but it relies only on the variables stored in the RAM. Because of the memory constraints, the training phase relies upon the communication with a PC for storing the heavy data. Once the model has been trained, the model is stored in the embedded program and the novelty detection is performed in real time. The general structure is shown in **figure 1.1**.

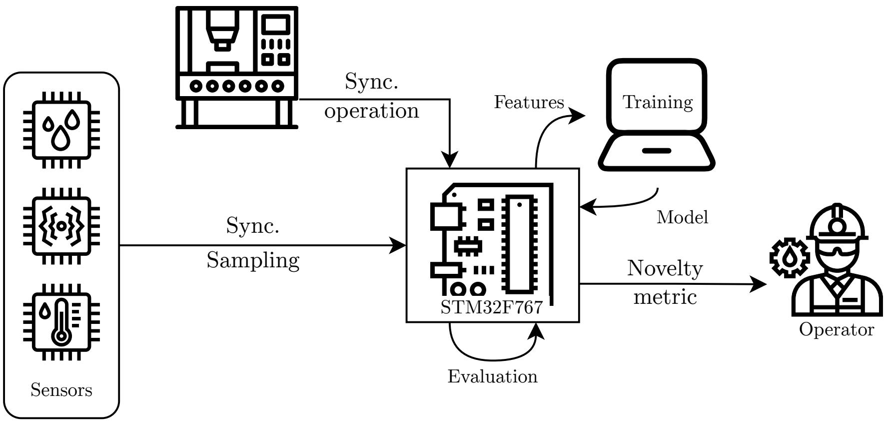

Figure 1.1: *Embedded system overview*

1.1 Hardware

The hardware used for the implementation is the STM32F767ZI board. The characteristics of the board are resumed in **table 1.1**.

Table 1.1: *Hardware characteristics of STM32F767ZI board*

Feature	Description
Microcontroller	STM32F767ZI
Architecture	ARM Cortex-M7
Clock Speed	Up to 216 MHz
Flash Memory	2 MB
SRAM	512 KB
EEPROM	No
GPIO	Up to 176
Timers	3 x 12-bit, 12 x 16-bit, 2 x 32-bit
ADC	3 x 12-bit
DAC	3 x 12-bit
Communication Interfaces	USART, UART, SPI, I2C, CAN, Ethernet, USB
Operating Voltage	1.7V - 3.6V
Operating Temperature	$-40\,°C$ to $150\,°C$

Similarly to what has been done for the `python` implementation, the parameters of the algorithm are configurable. To avoid the reading of files during the operation, the configuration is held in global variables defined in a header file. The configurable parameters are the usual: depth of the wavelet three, number of features, sampling frequency, time-series length etc.

1.2 Software

The code consists of a main loop, that is continuously running. It is responsible for executing the state machine behaviour, that manages the different phases of operation. The phases of operation are the same as described for the `python` implementation, except for the training phase, in which the microcontroller performs the sensor polling and the feature extraction and then sends the data to the PC using serial communication. The PC is responsible for the training phase. This part is developed again in `python`, but the final model is then formatted as a `model.h` file that can be directly included in the embedded code. The model is then stored in the flash memory of the microcontroller, together with the rest of the program.

The hardware configuration has been done using the IDE (STM32cubeIDE tool), which is a graphical interface that allows the configuration of the microcontroller and generates the initialization code.

1.2.1 Sensor polling

The microcontroller comes with a Hardware Abstraction Layer (HAL) which acts as an intermediary layer between the hardware and software. It simplifies interaction with the microcontroller's peripherals, such as GPIO, UART, and timers, by providing standardized functions and APIs. The HAL library enhances code reusability across different STM32 microcontroller families, streamlining the development process and enhancing the scalability of embedded systems projects.

To sample the data at a precise sampling frequency, two options are available:

- Use the Direct Memory Acces (DMA) capability of the microcontroller. This approach allows sampling the GPIO and storing the result in the memory accessible by the CPU, without using CPU time. The DMA is then configured to trigger an interrupt at the end of the transfer, and the interrupt is used to signal the end of the sampling and to start the feature extraction. It is suitable for high sampling frequencies and in fact, even downscaling the clock frequency linked to the DMA there is a lower bound of obtainable sampling frequencies.

- Use the Timer peripheral of the microcontroller. The timer is configured to trigger an interrupt at a precise frequency, and the interrupt causes the CPU to poll the sensor data. This approach is suitable for sampling frequencies that are not too high, and it is the one used in this work (for frequency in the order of kHz). If too many interrupts are generated, the CPU may not be able to execute them instantly, so the actual sampling may shift from the desired frequency, however, in this implementation the only interrupt used is the one for the sampling, so the CPU is not overloaded and the sampling frequency is precise.

1.2.2 Feature extraction

The features available to be extracted are the same as the ones described in **??**. The time-domain features are coded directly in a function that is responsible for extracting them. The frequency-domain features are computed by another function that relies on the `C` library `wavelib` for the wavelet transform [1]. The power of the wavelet coefficients is then computed and appended to the feature vector. The feature vector is then stored in the RAM, and it is used for novelty detection. The features are then standardized using the same mean and standard deviation used for the training phase.

1.2.3 Evaluation

When the microcontroller is in the evaluation phase, the feature vector is processed to compute the novelty metric. The model cluster centroids and radiuses were saved in the code in the training phase, so now it is possible to run the **??**.

1.2.4 Custom `C` functions

The `C` main loop, which executes all the behaviours that in the `python` implementation were executed by the various agents, relies on the library functions as well as on the custom functions resumed in **table 1.2**.

Table 1.2: *Custom function implemented in* `C`

Method	**Description**
setRTCclock	Set the clock of the microcontroller to the current time
get_time	Get the current time from the RTC clock
acquireSnapshot	Acquire a snapshot from the sensor
calcSnapDistanceError	Calculate the novelty metric based on the model centroids, radiuses and the current features vector
std_sclr	Standardize the features vector
snapReadyHandler	Handle the snapshot ready event (the interrupt of the Timer)
norm2	Compute the norm of a vector
packetCoeff	Perform the Wavelet packed decomposition and compute the norm of the coefficients, it relies on the *wavelib* library [1]
featureExtractor	Extract the features from the snapshot (both time domain and frequency domain)
eucDist	Compute the Euclidean distance between two vectors

Bibliography

[1] Rafat Hussain. *wavelib*. `https://github.com/rafat/wavelib.git`. Dec. 2014 (cit. on pp. 3, 4).