Internship second week report
7/06/25 - 11/07/25

SUPERVISORS :
Bruno QUOITIN
Aqeel AHMED

AUTHOR :
Maxime BARTHA

1 Notations

As explained in the first week report, I will be referring to each component of the whole system as :

- Device : arduino MKR1310 board in charge of transmitting LoRa frames.
- Transmitter : the computer in charge of scheduling the transmission of arduino devices
- Receiver : the computer in charge of scheduling and classifying the received LoRa frames

2 Initial objectives

My initial objectives for this second week were to :

- improve the time synchronisation method implemented during week1
- make it more robust to device crashing (I/O or port errors between transmitter and devices)
- make it working on large scale scenarios (with 10 or more devices)

3 Material and software used

Material :

- software defined radio : pluto, usrp, rtl
- LoRa transmitter : arduino MKR1310

Software :

- gnu radio companion : a block based programming software for signal processing.
- inspectrum : an open source spectrogram

4 Problems encountered

4.1 Captures with time synchronisation method

The machine learning algorithm we will be using will only need a sliced version of a LoRa frame corresponding to its preamble as in figure 1.

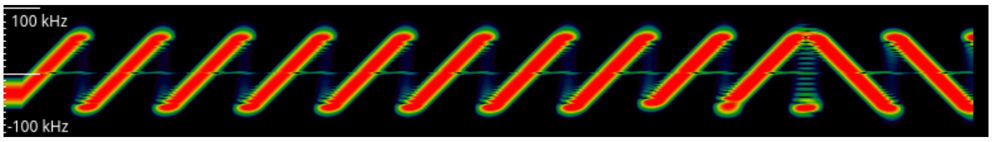

Figure 1: Example of capture needed for the machine learning algorithm

After making the time synchronisation method work and discussing with Aqeel about the resulting captures, a clear issue came to us : the frames were not captured with the same gap of samples each time.

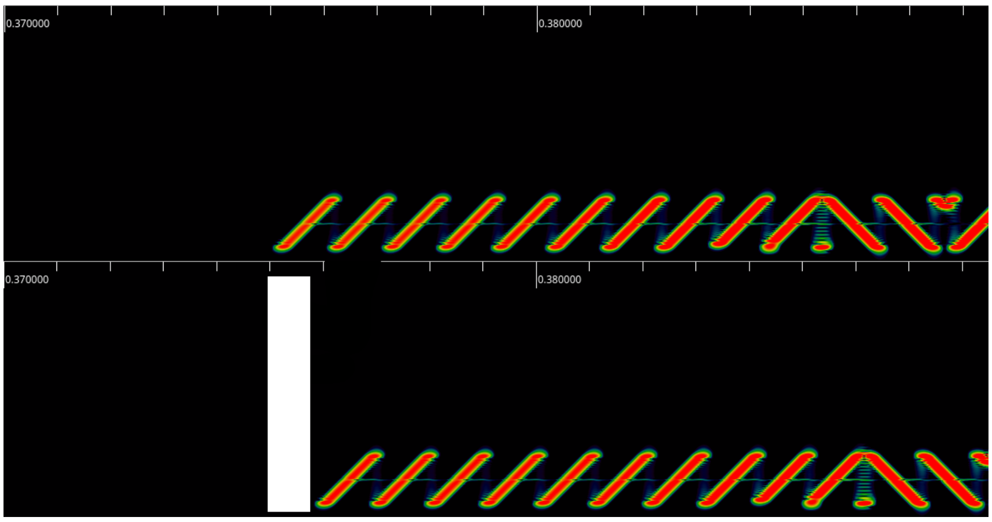

Figure 2: Two captured LoRa preamble from the same device. Sample gap difference shown in white

As shown in figure 2, in the captures made for a certain device, the start of the LoRa frame was not consistent. Due to this unconsistency, we wouldn't be able to post-process the captures correctly.

This was certainly due to random delays caused by :

1. The transmitter waking up from its sleep to send the arduino a sendFlag
2. The communication from the transmitter to arduino devices through Serial Ports
3. The arduino device receiving the sendFlag and sending the lora frame
4. The receiver switching from one device to another (implemented like a sleep)

So, by itself, the time synchronisation method wouldn't be enough for the Captures to be usable even with postprocessing.

4.2 LoRa detection

There were two issues with the last LoRa detection implementation:

1. The use of resampling before and after the detection
2. The gap in the beginning of captures

We decided to work on those issues to be able to have precise LoRa preamble captures for each frame.

5 Solutions found

5.1 Captures with time synchronisation method

We decided to use detection in addition to time synchronisation. So the file saver block would save only the detected LoRa preamble and not the whole period.

5.2 LoRa detection

With the help of its creator, we managed to remove the hardcoded sample rate and pass it as a parameter.

An example of the last year's capture saving flowgraph can be found at the figure 3.
Compared to it at the figure 4 is the current capture saving flowgraph which doesn't need resampling.

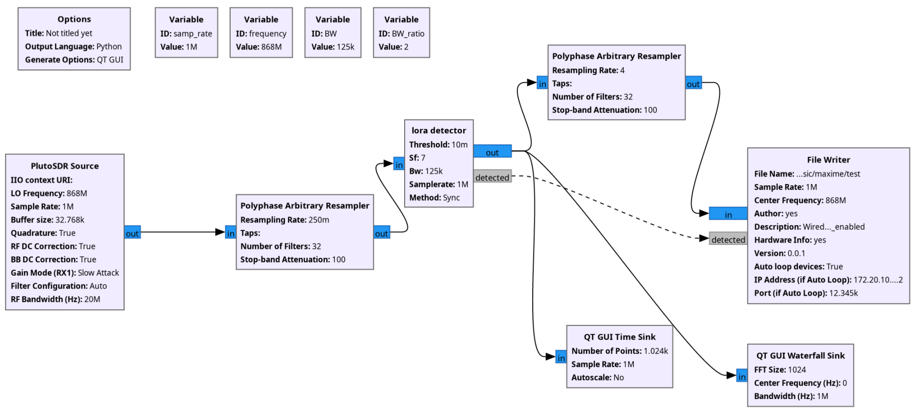

Figure 3: Last year's capture flowgraph

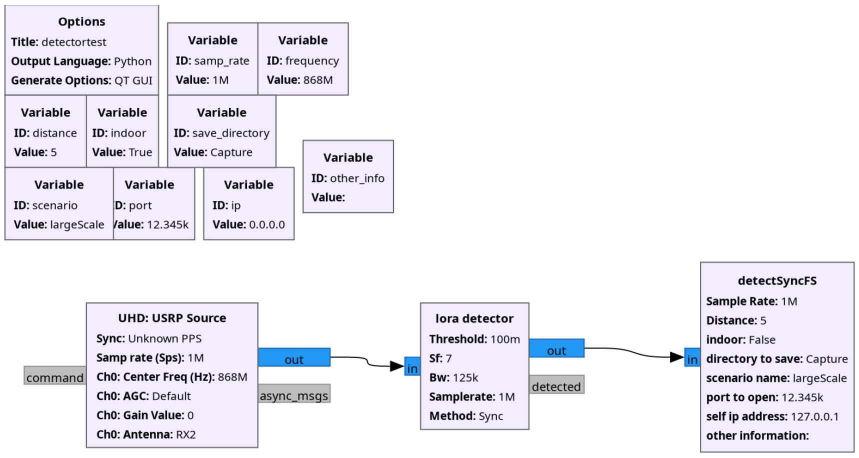

Figure 4: Current capture flowgraph

6 Summary

By changing the LoRa preamble detector, I've managed to implement a working detection + synchronisation method for capturing and classifying the received LoRa frames.

In addition, I made this detection + synchronisation method robust by adding real time socket communication in case of a falling device.

We found out that 2mm wide colson necklace were perfect for stabilizing the devices on the frame we will be using in large scale scenarios.

7 Detection + time synchronisation implementation

There is basically 3 parts to this implementation:

- The receiver side capturing the LoRa frames
- The transmitter side in charge of controlling which device should transmit
- The Arduino implementation handling the transmission

The transmitter is in charge of taking all the transmission parameters such as frequency, bandwidth, spreading factor, number of frames to transmit per devices (cycles) and synchronisation parameters such as phase and period.

The receiver is in charge of capturing the LoRa frames and classify them by device.

The arduino devices transmit LoRa frames when the transmitter tells them to.

7.1 Transmitter-arduino relation

Let's first take a look at the relation between the transmitter and the arduino devices :

the first job of the transmitter is to communicate all the transmission parameters to all the arduino devices connected to it by sending an InitFlag followed by the parameters.

Then when an arduino is initiated, it waits for a SendFlag for it to transmit a single LoRa frame. After Sending it, the arduino goes back to waiting for a SendFlag until it sent all the frames it had to.

A state machine showing this relation can be found in the figure 5.

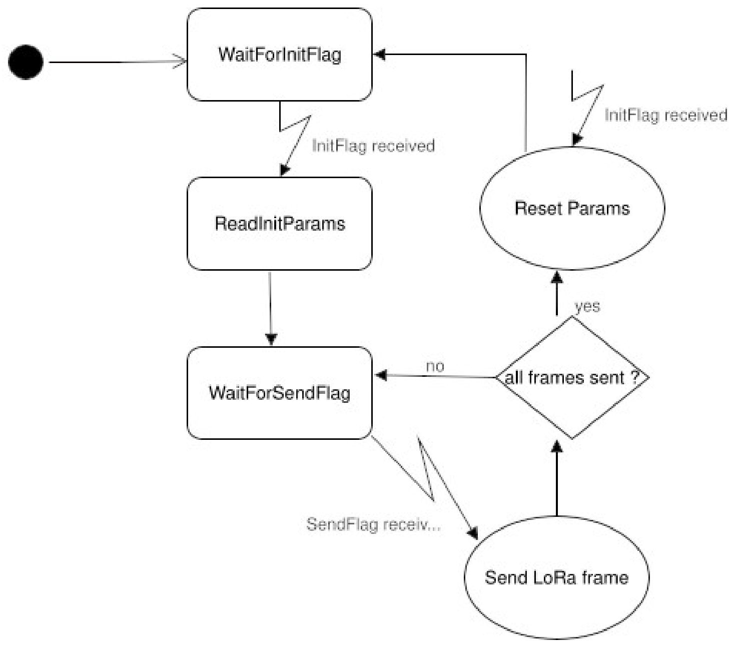

Figure 5: arduino state machine

In the WaitingForInitFlag state, the arduino's builtin LED blinks every seconds.
In the WaitingForSendFlag state, the arduino's builtin LED doesn't blink.
During the transmission of a LoRa frame, the arduino's builtin LED is lit up.

The behavior of the builtin LED then permits us to know in which state the arduino is at any time.

7.2 Transmitter-receiver relation

First, the receiver sets itself as a socket server on a certain port. The transmitter connects to this socket using the port and the receiver's IP address. Then, when the connection is done, the transmitter sends all the synchronisation parameters to the receiver such as :

- the starting time of transmission
- a list of the devices ordered by device id
- the period between each transmission

Then, after the starting time is reached :
The transmitter sends a LoRa frame from the first device in the list and cycles the device's transmission each period.
The receiver waits for a detected LoRa frame to come from the LoRa detector and cycles the device's identifiers each period.

7.3 cycling devices and frame classification

For the receiver to rightfully classify the detected LoRa frames per device and frame count, I implemented a separate thread in charge of cycling those identifiers.

I had to thread it because the work function of the file saver wasn't running continuously, it only ran when there were incoming signals (so when LoRa frames were detected).

7.4 Saving structure

A saving directory and scenario name have to be given. The structure can be seen in figure 6

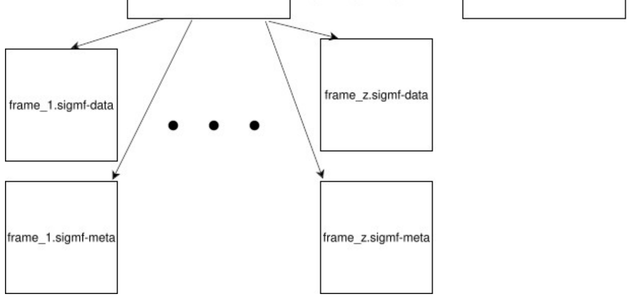

Figure 6: saving structure

Each .sigmf-meta files contains :

- the data type (c32_le)

- the frequency
- the sample rate
- the capture time
- a description taking the distance, outdoor/indoor and other possible parameters

7.5 Arduino crash handling

If at any point, a device is unreachable from the transmitter, the goal is to remove it from sending and receiving device lists. This will have the effect of it being ignored completely gaining the time it would have been taking.

When a device is detected as unreachable, a "remove" message is sent by the transmitter to the receiver followed by the device id to be removed, the restart time (for synchronisation purpose) and the device id to restart the reception/transmission on.

This permits us to remove any crashing device on both the receiver and transmitter.

To implement this method, the TCP connection between the receiver and transmitter had to be threaded to be able to react in real time.

7.6 Improvements

Some improvements can be made in this implementation :

- use implicit header for the LoRa frame
- thread the file saving (it takes a lot of time and may be a treshold for minimum transmission period)

8 Conclusion

After testing on a larger scale (10 devices), the implementation works even with crashing devices. The goal for the last two weeks of this Internship is to be able to capture and classify 15 devices in different scenarios. Some improvements can be made to make those captures faster but most of the work is done.