Wireless Project – Report

Name1 Last-Name1
Name2 Last-Name2

May 21, 2025

Contents

1 Project Context

This project aims to implement a receiver / decoder of a simplified 5G NR signal.

2 Project architecture

2.1 File tree

```
├── report/
├── topic/
├── data/
│   ├── tfMatrix.csv
│   ├── tfMatrix_2.csv
│   └── tfMatrix_3.csv
├── code/
│   ├── main.py
│   ├── src/
│   │   ├── binary_transformation.py
│   │   ├── crc.py
│   │   ├── decode.py
│   │   ├── demod.py
│   │   ├── hamming748.py
│   │   └── utils.py
│   └── tests/
│       ├── tests_hamming.py
│       ├── tests_modulation.py
│       └── utils.py
└── README.sh
```

Figure 1: Project file structure

2.2 Files description

The folder `data/` contains the matrix. The matrix 1 and 3 are correct and should be correctly decoded, but the matrix 2 had a very bad transmission and cannot be recovered.

The folder `code/` contains the implementation.

The main file is `code/main.py`. It contains a very simple command line parser to run the code with the wanted matrix. The usage is detailed in the readme file (and with the `-h` flag).

The file `code/src/utils.py` contains general functions to manage matrix (getting from file, display, flatten index, ...)

The file `code/src/demod.py` implements multiple demodulation functions (`bpsk`, `qpsk`, `16qam`).

The file `code/src/decode.py` implements the decoding of a matrix (for a given user identifier). It uses the other files to do so. It is implemented using the object oriented paradigm.

The folder `code/tests/` defines unit tests on hamming and modulation.

The file `code/tests/utils.py` useful functions for the unit tests.

3 How does it work ?

3.1 Matrix loading and shortening

We load the matrix using the function `get_matrix(fn: str)` from `code/src/utils.py`

This function reads the CSV file, and makes a matrix of complex numbers.

As is, the matrix contains 1024 columns, corresponding to the size of the FFT. But in the 5G NR, only 624 symbols are used, in the range described in the subject.

The function also removes the unused parts.

3.2 PBCH decoding

Once we get the complex matrix, it is given to the class `DecodeMatrix` (in the file `code/src/decode.py`).

First we remove the part before the start of the PBCH (the two first lines / symbols), and we flatten the matrix (done in the method `retrieve_PBCH`).

Then we use the method `decode_PBCH_user(user_ident: int)` that first demodulates and decodes the PBCH header (retrieving `cell_ident` and the number of users).

For the first matrix, we get 18 users and a cell identifier of 12345

We note that the PBCH is modulated using bpsk, and this is because it is the most reliable modulation, with only two symbols possible. It is needed to use a robust one here (even if less information can be transmitted in the same amount of time) because the informations contained in the PBCH are important for the rest of the signal.

Then, this method will walk in the linearised complex matrix (by steps of 48) to search for the corresponding `user_ident`. To check this, the method `is_user_at_block(user_idx: int, user_ident: int) -> bool` is used.

When the corresponding user is found (the one with the `user_ident` we selected), the method `extract_PBCH_user_data(user_idx: int) -> dict[str, int]`.

Now we have, for the selected user, the position and modulation used for the corresponding PDCCHU.

3.3 PDCCHU decoding

With the information provided by the PBCHU, we can get the right portion of the matrix corresponding to the PDCCHU: it starts at line `symbol_start` and column `rb_start`.

However, it is not very clear how the length of the block should be recovered. The current fix is the following: if `mcs` is 2, then the length of the block is of 3 resource blocks, otherwise (`mcs` is 0), the length is 6 resource blocks. This is because bpsk keeps the same length, and qpsk double the length when demodulating.

Then, with the value of `mcs`, the block is demodulated and decoded (see the function `demod_decode_block(block: list[complex128], mcs: int) -> list[int]`)

Then the data is recovered by segmenting the result and converting to decimal.

3.4 Getting the payload

Now we can get the payload. Using the informations in the PDCCHU, we have the position of the start and the length of the block.

The demodulation and the decoding is implemented in the `demod_decode_PDSCH_block(block : list[complex128], mcs: int) -> list[int]` function.

Once demodulated and decoded, the integrity of the data is checked with the CRC.

There is one thing that is not implemented here: the padding is not removed, so when converting to a string, there is some garbage at the end.

Also, for some users, the CRC is not correct.

3.5 Getting the message

To convert the payload to an ASCII string, we use the function `payload_to_str(payload : list[int], user_ident: int) -> str`. It uses functions from the given file `code/src/binary_transformation.py` to convert to bytes and then to decrypt (Caesar).

4 Result

Our implementation mostly works: it is able to retrieve the message for most users for matrix one and three (and of course not for the second matrix that contains way too much noise).

But the CRC is wrong for some users (this might be a bug in our implementation).

Also, the padding is not removed, so the message strings contains garbage in their ends.

We tried to make the implementation as general as possible, using variables when needed instead of hard-coding values.

It is possible to select the wanted user and matrix directly from the command line using the simple parser (as described in the README).

Output examples:

```
$ python3 code/main.py data/tfMatrix.csv 3 -v

PBCHU: {'user_ident': 3, 'mcs': 2, 'symb_start': 5, 'rb_start': 9, 'harq
    ': 0}
PDCCHU: {'user_ident': 3, 'mcs': 7, 'symb_start': 11, 'rb_start': 39, '
    rb_size': 27, 'crc_flag': 0}

```

```
User 3: your key is 37
Never gonna give you up
Never gonna let you down
 ...
Ne
```

```
$ python3 code/main.py data/tfMatrix_3.csv 3 -v

PBCHU: {'user_ident': 3, 'mcs': 0, 'symb_start': 5, 'rb_start': 33, 'harq
    ': 0}
PDCCHU: {'user_ident': 3, 'mcs': 26, 'symb_start': 11, 'rb_start': 9, '
    rb_size': 24, 'crc_flag': 2}

User 3: your key is 36
Never gonnY?e
```

5 Conclusion

In this project, we implemented a program able to decode a simplified version of a 5G NR signal. This hands-on project bridges theoretical concepts and practical implementation, enhancing our understanding of how modern cellular networks manage user-specific data and ensure reliable communication.