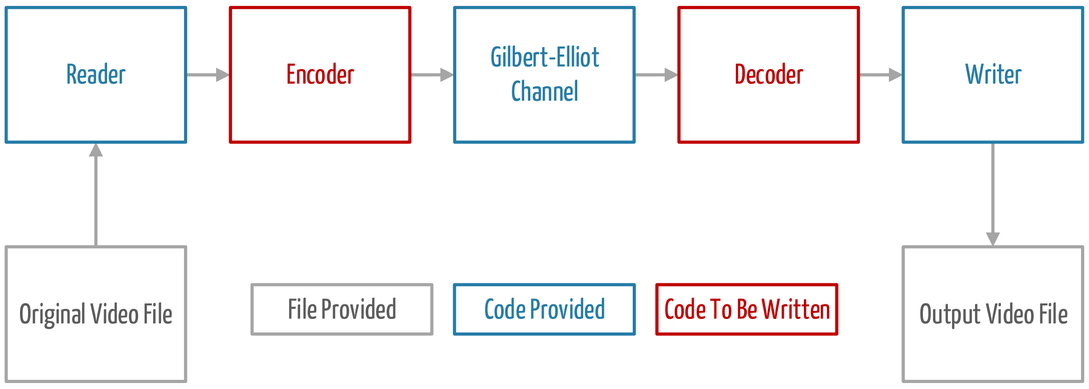
Multimedia Coding Pipeline

Introduction

In this project you are going to create a *codec* for sending multimedia files over a noisy channel. The codec contains an *encode* method, which prepares the data to be sent over the channel and a *decode* method that processes the data received from the channel. The pipeline looks as follows:



Your goal is to create a codec that is able to correct errors introduced by the channel. This should be done in a way that is sufficiently fast and does not require too much redundancy overhead.

The channel is a Gilbert-Elliot channel that introduces bit flips (0 into 1 and 1 into 0). It does **not** lose or reorder data. Whenever you send a byte to the channel there's a chance a variable number of bits will be flipped. The Gilbert-Elliot channel is also bursty, hence you could (but must not) implement your codec in a way that can deal with bursts.

Framework

We provide you with a framework implementation of the pipeline above. It implements an "identity" codec, i.e. it does not transform the data in any way before sending it to the channel. The codec is defined in the file `student.py` (in folder `mm_codecs`). There are two methods:

- `mm_encode(source)` which gets a Python generator as parameter `source`. By calling `next(source)` or iterating over the parameter, you will get the incoming data byte by byte. Your method outputs single bytes by calling `yield`.
- `mm_decode(source)` works the same as `mm_encode`, but collects data sent over the channel.

You will only have to change these methods to finish the project.

The file `pipeline.py` can be used to run the pipeline. The original video is read, sent through your encoder and transmitted over the channel. Afterwards the corrupted data is handed to your decoder and written back to a file. These files are eventually compared and statistics are printed out.

You are **not** allowed to use any third-party modules from the Python Package Index or other sources. Our reference implementation is able to fullfill all requirements just by importing the `struct` and `math` modules.

Evaluation

Using the provided video, you can also see how well your codec works by opening the resulting video. When using the template codec, the result should be full of artifacts.

If you want to test your codec with varying channel parameters, you can generate custom error patterns using the file `ge_pattern_generator.py`. Running the generator script using Python will create a new pattern with the current timestamp. You can also change the parameters (good and bad state reliability as well as the model's transition probability) when the ge object is created. The generation of additional patterns is completely optional.

Passing

To **pass** this project you have to provide a codec which

- is able to correct all errors for the benevolent channels.
- is able to correct most errors on more adversary channels.
- is able to correct some errors on bursty channels.
- does not add too much redundancy (at most 300%).
- does not make the pipeline run for more than 10 minutes per scenario.

You **may** provide a codec which is able to fulfill the following criteria:

- is able to correct all errors on more adversary channels.
- is able to correct all errors on bursty channels.
- uses little redundancy (our reference implementation has 100% redundancy, but it is possible to have less).
- is very fast.

To build a very fast codec, you might need third-party libraries (such as e.g. `numpy`), which will disqualify your submission. Nevertheless we are interested in how well you can do. Hence think about doing an official submission (without frameworks, considered for passing) and another inofficial submission via email (which will not be considered for grading).

Submission

We **require** you to hand in the `student.py` file, containing your `mm_encode` and `mm_decode` methods, using the **Course Management System**.