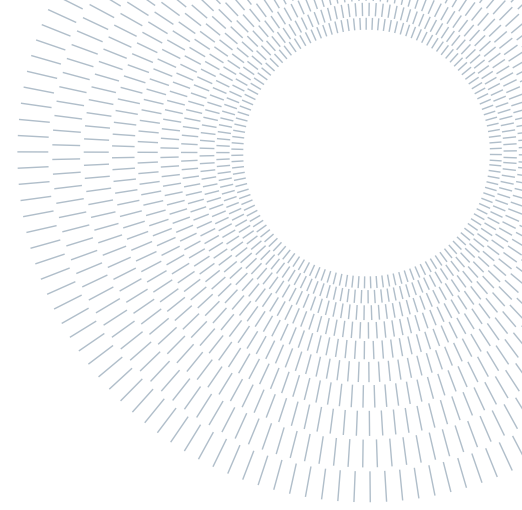




POLITECNICO
MILANO 1863

**SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE**



CPA/Template attacks output viewer

Project for the Embedded Systems course
Computer Science and Engineering

Leonardo Gargani, 10569221
Olivier Wartique, 10844629

Supervisor:
Giuseppe Massari

Co-supervisor:
Michele Piccoli

Academic year:
2021-2022

Abstract: The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main research questions that have been addressed in your thesis. It needs to summarize the motivations and the adopted approach as well as the findings of your work and their relevance and impact.

1. Introduction

...

2. Implementation

...

3. Faced problems and solutions

During the development of the application, we had to face up some problems, mainly due to the size of the dataset to handle. We think it can be useful to share here some insights about our decision-making process as the development was moving forward, and the solutions we came up with.

Starting from the very beginning, our goal is plotting the content of a .csv file, while maintaining the possibility of moving around inside the plot (i.e., zoom-in, zoom-out, change the X-range).

Our initial approach was plotting all the file at once.

The problem that immediately arised was that handling over 7.5 million points (every .csv file has about 30 thousand rows and 256 columns) was definitely too resource-demanding. That led to an evidently unusable GUI due to its poor performance, not only when moving around, but also for the loading time of the plot itself.

Therefore, we started experimenting looking for some possible improvements to load data into memory in an efficient way using Python.

A first available option we found was converting the .csv file into a .hdf5 one. HDF5 is a file format built for fast I/O processing and storage, whose potential can be fully exploited through a handy Python library. However, the result we obtained was still very poor and the HDF5 option had to be discarded.

Since we handled all the data as numpy arrays, we also came across the .npy file format. It is a simple format for saving numpy arrays, in an optimized way, with no loss of information. After converting the .csv file into a .npy one, we plotted it in the same way as before. The loading time was significantly lower and the usability of the GUI improved, but still far from an acceptable result.

Having understood that .npy was a far better choice over .csv and .hdf5, we decided to keep that conversion step and start tweaking around to gain as much performance as possible, this time in the PyQtGraph plotting process.

A special tweak worth mentioning is the way we subsample data in order to be displayed in the plot. PyQtGraph provides three downsampling modes: subsample, mean, and peak.

Despite being the slowest one, we picked the last one ("peak") since it downsamples by drawing a saw wave that follows the min and max of the original data. This is the only way to make sure that the peak of our plot does not get lost while displaying the points.

Back to the fact that the GUI responsiveness was far from being acceptable, we came to the conclusion that showing all the 7.5 M points at the same time was not feasible in any way.

At this point we tried to change our approach. Here was (and currently is) our new one.

Since every file contains 256 rows, and each row corresponds to a curve in the plot, we can preprocess every file and identify for it which is the peak and what row it shows up in. Then, save the results in a new file. When the user opens a plot, now the program reads the content of such a file to display by default only the curve with the peak. In addition, each one of the other curves can be shown/hidden in the plot by using a dedicated checkbox.

This new approach leads to a very low loading time and to a GUI which is not only usable, but which has high performance in every zoom/scroll operation.

Despite the high performance increase intrinsic to the new way of displaying data, we still decided to keep the conversion step from .csv to .npy. In fact, the same file stored as .npy takes more than 40% less space on disk than if stored as .csv. As we will cover in more detail in the next section, this leads to both faster loading times and lower memory usage at runtime.

4. Experimental results

...

5. Conclusions

...

6. Guide (to be removed)

Figures, Tables and Algorithms have to contain a Caption that describes their content, and have to be properly referred in the text.

6.1. Figures

Include figures (.png, .jpg, .eps) with the following command:

```
\includegraphics[options]{filename.xxx}
```



Figure 1: Caption of the Figure.

Thanks to the `\subfloat` command, a single figure, such as Figure 1, can contain multiple sub-figures with their own caption and label, e.g. Figure 2a and Figure 2b.



Figure 2: Caption of the Figure.

6.2. Tables

Example of Table (optional)

	column1	column2	column3
row1	1	2	3
row2	α	β	γ
row3	alpha	beta	gamma

Table 1: Caption of the Table.

How to insert itemized lists:

- first item;
- second item.

	column1	column2	column3	column4	column5	column6
row1	1	2	3	4	5	6
row2	a	b	c	d	e	f
row3	α	β	γ	δ	ϕ	ω
row4	alpha	beta	gamma	delta	phi	omega

Table 2: Highlighting the columns

How to write numbered lists:

1. first item;
2. second item.