École Centrale Lyon

UE PRO

Rapport de stage dit "d'application"

Élève :
Tristan Chevreau

Tuteur de Stage :
Daniele Botto
Tuteur-PCP :
Laurent Blanc

2 octobre 2023

Table des matières

Table des figures

Liste des tableaux

1 [FR] Introduction

Dans le cadre de mes études à Centrale Lyon, j'ai effectué un stage de niveau technicien dit "d'application". Je dois aussi effectuer une mobilité à l'étranger. Puisque j'ai décidé d'obtenir mon diplôme d'ingénieur sans césure ni double diplôme, mon cursus à Centrale va durer trois ans. Cela signifie que je peux compter sur ce stage pour valider la mobilité.

Je remercie Daniele Botto de m'avoir encadré durant ce stage avec l'aide de Serena Occhipinti, ainsi que Laurent Blanc pour m'avoir aidé à trouver ce stage et guider mes choix. Je remercie aussi toute l'équipe de DIMEAS pour leur accueil chaleureux et Paolo Neri qui m'a aidé à comprendre son travail sur la NHFA.

2 [FR] Contexte du stage et organisation de l'entreprise

2.1 La ville de Turin

Turin est une ville de la région du Piémont en Italie. Elle est chef-lieu de la ville métropolitaine de Turin (plus ou moins analogue à un département français). Le fonctionnement des divisions italiennes entre "aire urbaine" ou "métropole" étant largement différents de celui des villes françaises, il est difficile de lui associer une ville équivalente française. Néanmoins, on peut remarquer que Turin *intra-muros* contient environ autant d'habitants que la métropole de Bordeaux, c'est-à-dire 800 000.

Elle fut la capitale d'Italie pendant une courte période. Elle est connue comme étant une ville industrielle (*Fiat* en particulier, mais aussi *Caffarel, Lavazza, Martini,* ...), ce qui lui a valu d'être en partie bombardée pendant la seconde guerre mondiale. À la différence d'autres villes italiennes très concentrées comme Gênes ou Lucques, elle est composée de très larges avenues quadrillant toute la ville et faisant la part belle aux voitures individuelles. D'ailleurs, la plus grande avenue d'Europe est située à Turin : c'est « *Corso Francia* », longue de 11,6 km.

Turin est la quatrième ville d'Italie en termes de population, ce qui fait d'elle une ville très importante. En effet, l'Italie n'est pas aussi centralisée (aussi bien politiquement qu'en termes de population) autour de la capitale que la France, ce qui donne mécaniquement plus d'importance aux autres villes. Pour s'en convaincre, comparons le ratio entre la première et la quatrième commune d'Italie avec ce même ratio pour les unités urbaines françaises : Paris a 11 fois plus d'habitants que Lille, et Rome 3.5 fois plus d'habitants que Turin.

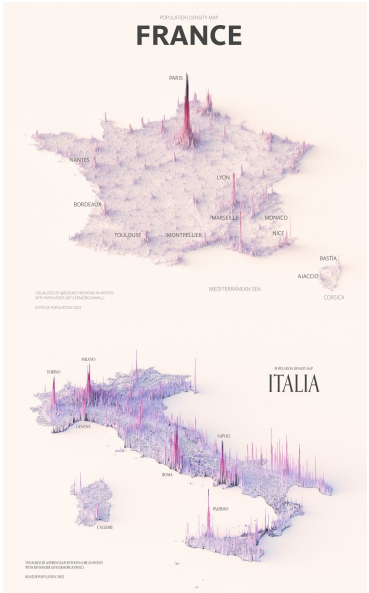


FIGURE 1 : Centralisation de la France versus l'Italie

Cet aspect décentralisé de l'Italie (figure 1) se ressent dans les conversations avec les collègues, mais aussi dans les informations et l'aspect très local des célébrations, des habitudes, de la culture et de la nourriture. Cela est aussi dû - d'après mes collègues - au fait que l'Italie en tant que pays est beaucoup plus jeune que la France.

2.2 *Politecnico di Torino* (*Polito*)

Politecnico di Torino (*Polito*) a été fondée en 1906, mais on peut remonter jusqu'en 1859 pour en trouver des traces. Elle héberge environ 35 000 étudiants, de la licence [1] au PhD.

Pour contenir ces 35 000 étudiants, de même que les 11 départements de l'école, *Polito* a plusieurs campus à Turin, mais d'autres aussi à Alexandrie, Bielle, Ivrée ... Le campus principal est situé dans le quartier *Crocetta*, et est tellement gigantesque (122 000 m^2) qu'il passe par-dessus l'avenue *Castelfidardo* (figure 2).

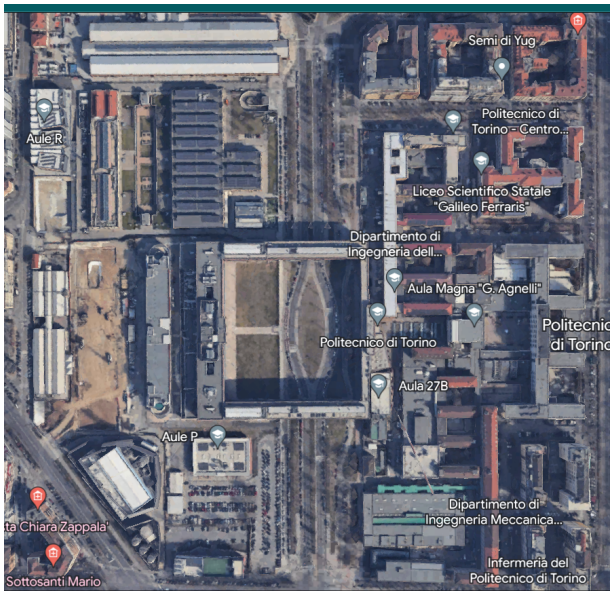


FIGURE 2 : Campus principal en vue aérienne

2.3 *DIMEAS - AERMEC* et sujet

J'ai fait mon stage dans le département DIMEAS (*Department of Mechanical and Aerospace Engineering*). Ce département est composé de 26 groupes de recherche, et j'ai effectué mon stage dans le groupe AERMEC (de nom complet *LAQ AERMEC for gas turbines and compressor blades*). Ce groupe de recherche s'intéresse particulièrement à l'étude de turbomachines (amortissement, fatigue, déformation ...).

Mon tuteur, *Daniele Botto*, est aussi le directeur de thèse d'une doctorante, *Serena Occhipinti*, qui travaille sur la méthode de *Digital Image Correlation* (DIC), c'est pourquoi cela était aussi mon sujet de stage. Cette méthode sera expliquée plus tard.

La figure 3 montre à quoi peut ressembler une des expériences présentes dans ce laboratoire.

[1] Le système scolaire italien est légèrement différent, car ils ont une année supplémentaire de lycée

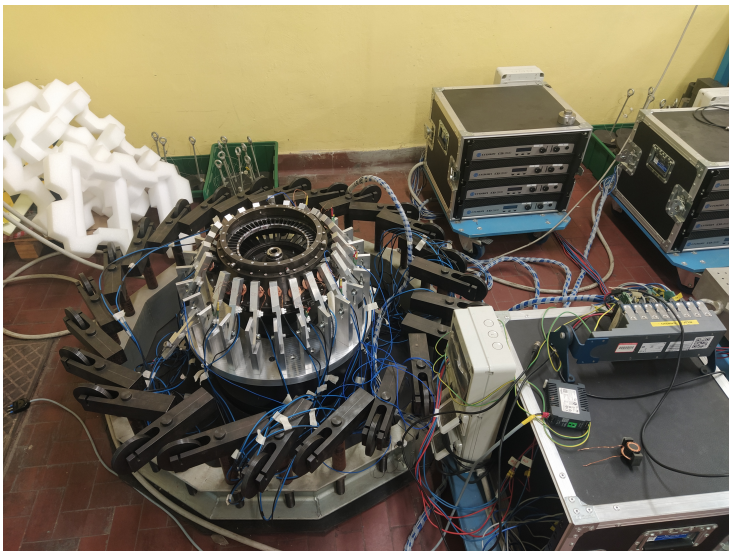


FIGURE 3 : Une des machines du laboratoire

3 [EN] Presentation and analysis of the undertaken engineer work

3.1 The context of the work: DIC with smartphones

3.1.1 DIC as a method

Digital Image Correlation (DIC) is a method using one or two cameras to measure changes between images, allowing to process 2D or 3D displacement fields. It usually works by tracking squares of pixels (subsets) that contain specific shapes or characteristics. To obtain this subset-specific shapes or characteristics, speckled patterns are often used on the object to analyse, as shown on figure 4.

Figure 4: Speckle patterns, made using paint bombs (*the bottom one was done by me*)

DIC techniques are in constant evolution, in terms of algorithms, tools and setting up. It is also a very active research topic as shown by figure 5. They can be used in mechanics, kinetics, material science, biology ...

The principle is more or less always the same (figure 6), that is, analysing the difference between the movement of subsets of pixels to get strain and displacement fields. DIC is often used when very fine and precise results are needed.

With one camera, you can do 2D displacement field analysis, and with two cameras, 3D is possible (we will ignore the existence of colour spectrum, mirroring and other advanced techniques to do 3D analysis with one camera). The main problem when trying to do 3D-DIC is to take simultaneous images from the object. Usually, two high speed and high resolution cameras are used, with a common external trigger. Those are expensive (generally hundreds or even thousands of euros) and need specific communication ports and software to work correctly and allow precise external triggering.

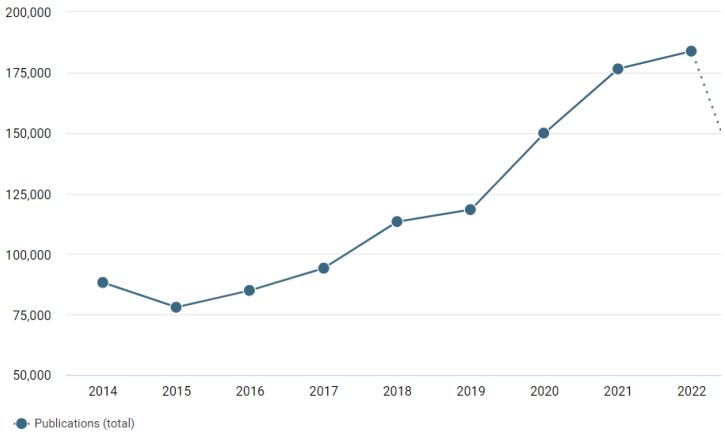


Figure 5: Number of publications containing the words "digital image correlation", according to `app.dimensions.ai`

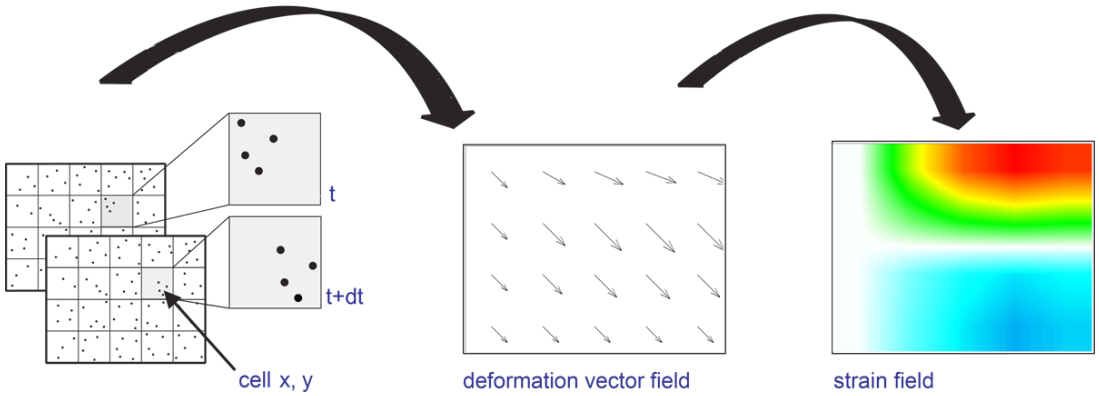


Figure 6: Principle of DIC analysis (from `lavision.de`)

3.1.2 DIC with smartphones

The fact that DIC cameras are very expensive and require specific software to work is a limiting factor to its widespread use. That's why it would be interesting to use smartphones as the DIC cameras : smartphones are widespread, easy to use and some of them have very good camera resolutions. In this use case, the DIC analysis would be of a lesser quality but would allow using it in an educational context (students experiments, for example), where results quality is less important.

But one problem arises : when using smartphones, it is very difficult to trigger photos simultaneously. My internship's goal was to test and begin preliminary work on some methods to obtain synchronized images from two smartphones. I tested mainly three methods :

- Method 1: taking videos with both smartphones and using a digital clock in the field of view of those videos to synchronize the images
- Method 2: taking photos synchronously using a "selfie stick" trigger

- Method 3: taking both videos independently and expressing the displacement signal of each video as a function to obtain more points than the sampling rate, thus not needing to synchronize them.

Those methods will be explained later. For testing all of these methods, we used a plastic L-beams with around a 28 Hz resonant frequency installed on a shaker (figure 7). These plastic beams had a 'speckle' pattern installed on it. When needed, two powerful LED lights were available.

Figure 7: Setup common to the 3 methods (image extracted from a test)

NB: we found some existing apps and tools for doing this, but they were all obsolete, and/or usable for only a few smartphone models. Furthermore, they needed to be compiled using *Android Studio* and then uploaded to the phone in developer mode as they were not available on the app store, making them almost useless in an educational context.

3.2 Documenting the work

Because of the short, diverse and prospective nature of the work done, if I wanted my work to be useful it needed to be well documented. I choose to document each task as *mardown* files [2]. These files are a big part of my work and can be found in the annex section.

When the reader wants to know more information about a specific paragraph, he will find that the corresponding annex will have a more detailed explanation in terms of methodology, tools and results, as well as the specific sources for this task.

I wrote all my scripts in *Python* (3.10) and documented and commented them, trying to follow the PEP8 style guide on coding for better readability [3]. For every important script, there are examples of input and output data and test samples.

My internship supervisor, Daniel Botto, received a folder containing a very short summary of my work (see annex page 32), pointing to numbered folders containing additional files, scripts, and documentation for each task.

[2] This is just a markup language and is easily human-readable and pdf convertible `https://www.markdownguide.org/`

[3] `https://peps.python.org/pep-0008/`

This internship report is only an additional text to present my work in a more global, imaged and academic way. This is why there is redundancy between the body of the report and the annexes. Because this report is supposed to contain a maximum of 30 pages without annexes, some parts of the work are only shallowly explained.

3.3 Thorough description of the tested methods

3.3.1 Method 1: digital Clock in the field of view

General description

During my internship, I developed a tool called Frame-Coupler, which serves as a utility to identify common frames among multiple videos of the same object from different points of view. To achieve this, a digital chronometer is required to be present within the field of view of all the videos. This clock is then digitally read, and its readings are used to attribute a date to each frame in the videos. This allows for the identification of common frames even when the videos have different start and end times, frame rates, and other variations. This can also compensate for frame rate inconsistencies and missing frames, which are fairly common in smartphone cameras.

The Frame-Coupler program expects a specific model of digital chronometer for synchronization [4]. This can be displayed on a computer as most screens have a 60 fps frame rate. The videos of the object need to have this screen in the field of view. MPEG-4 is the preferred file type for these videos, as they contain timestamp information for each frame. These timestamps are a precise way to measure the relative time in between the frames of the same videos. The problem is, these timestamps are just a reference to the first frame ; the absolute time of these frames is known using the digital chronometer, and allows synchronizing the videos as a whole with each other.

Key Steps

Frame-Coupler operates in four key steps. The workflow of the program (figure 8) is divided into four steps, each of which can be executed independently using their respective *python* files. Alternatively, the entire process can be run in one go using the 'main.py' file.

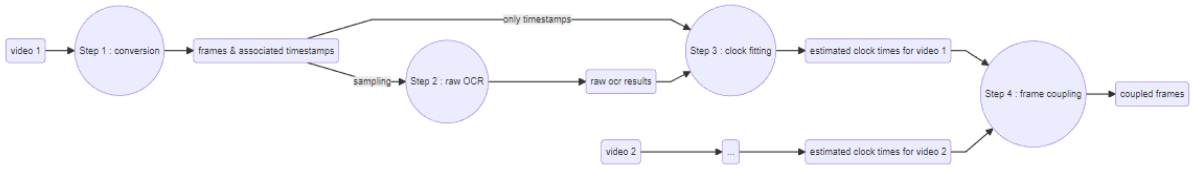


Figure 8: Workflow of the frame-coupling program

Step 1: Conversion

In this step, MPEG-4 video files (.mp4) are converted into gray JPEG frames (.jpg). This process also generates a file containing the timestamps for each frame. The converted frames are saved for further processing. This step is very time-consuming, and can create a lot of files when the analysed videos are long.

Step 2: Raw OCR (Optical Character Recognition)

OCR stands for optical character recognition. OCR pieces of software are commonly

[4] `https://www.youtube.com/watch?v=rf2Lmfqi5ZM`

used in archives to digitalize a paper text document into computer-compatible text to allow proper post-processing and a lighter file size. In our use case, OCR is performed on selected frames to read the digital chronometer. The user defines a region of interest for chronometer detection, and the *Tesseract* OCR API [5] *pytesseract* is then used to read it. This OCR software is very reliable as it is maintained by *Google*. The raw OCR results are saved in a file alongside the frames.

Step 3: Clock Fitting

The raw OCR results are processed to correct errors and convert the clock readings into dates in milliseconds. Common errors include "8" seen as "B" or misplaced spaces. These dates are then used to fit a linear curve using the frame timestamps, resulting in an association of a precise date with every frame. Misread points are deleted using the residual-iterative method. The residual iterative method consists of fitting a curve to a set of points, deleting all the points which have a residual higher that a chosen threshold, and then fitting back a curve, and repeating the process with lower and lower thresholds. After this step, we have a function that can link the timestamps of the frames to an absolute reference : the digital chronometer. Figure 9 shows an example of a straight line fitting ; we can see that some values will be deleted as they are obviously incorrect. Because the chronometer has a refresh rate of only 60 fps and the videos can have a frame rate of up to 240 fps, two consecutive frames can have the same chronometer value. But because of the fitting, we can have a better precision than the chronometer.

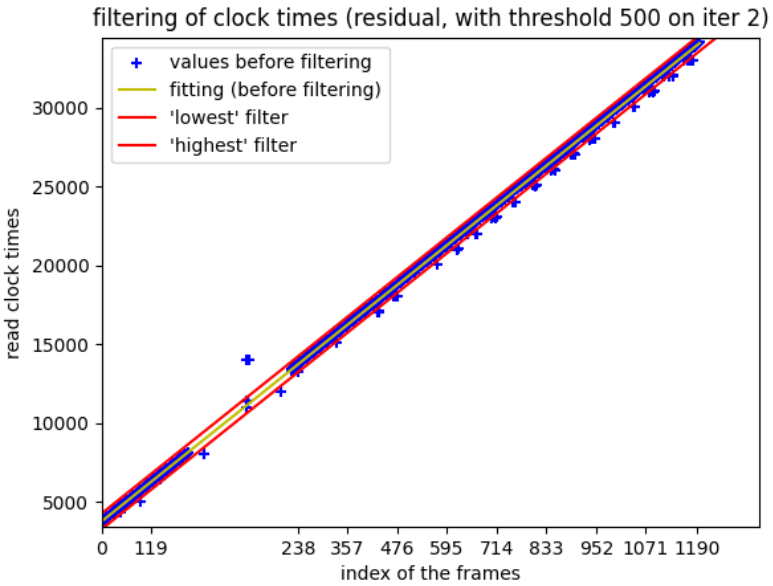


Figure 9: An example of fitting

Step 4: Frame Coupling

In the final step, the Frame-Coupler program identifies common frames among videos by comparing the dates associated with each frame's timestamps. Frames are

[5] An API is a way for a software to communicate with another software. In our case, *Tesseract* can communicate with *python*.

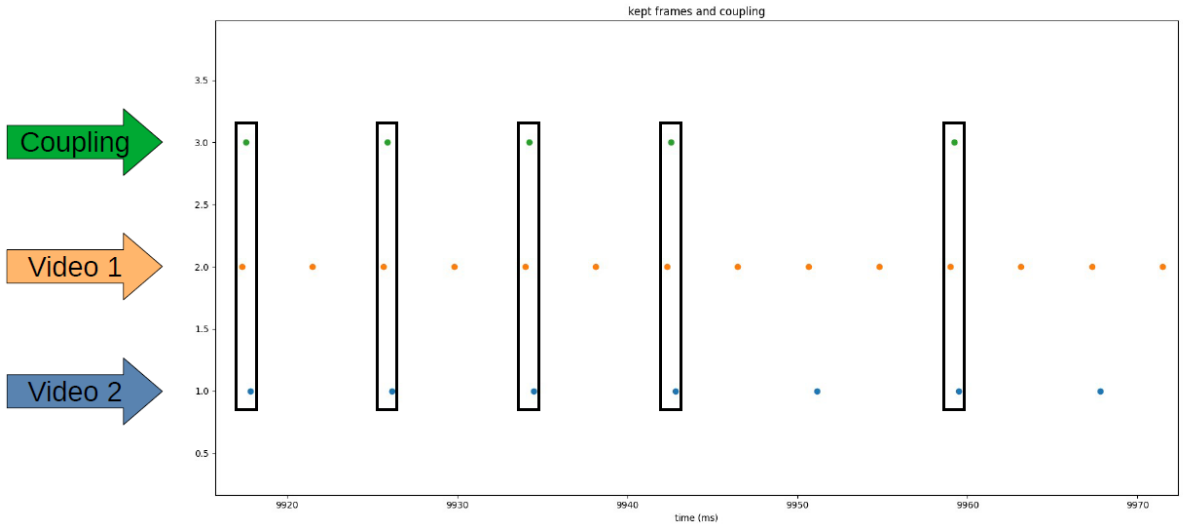


Figure 10: Selection of common frames (dots represent the date of a frame)

considered common when the difference in their dates falls within a user-chosen threshold. The results include the coupled frames in separate folders and a CSV file containing the synchronized timestamps. This process is shown on figure 10.

Figure 11 shows two videos placed in the real-time world : we can see that the points are not evenly spaced on the Y-axis, showing that the videos have different frame rates. We can also see that the slope of the lines are not the same : this demonstrates that the timestamps only serve a purpose for a relative time reference. The graph also shows that videos were neither started nor stopped at the same time.

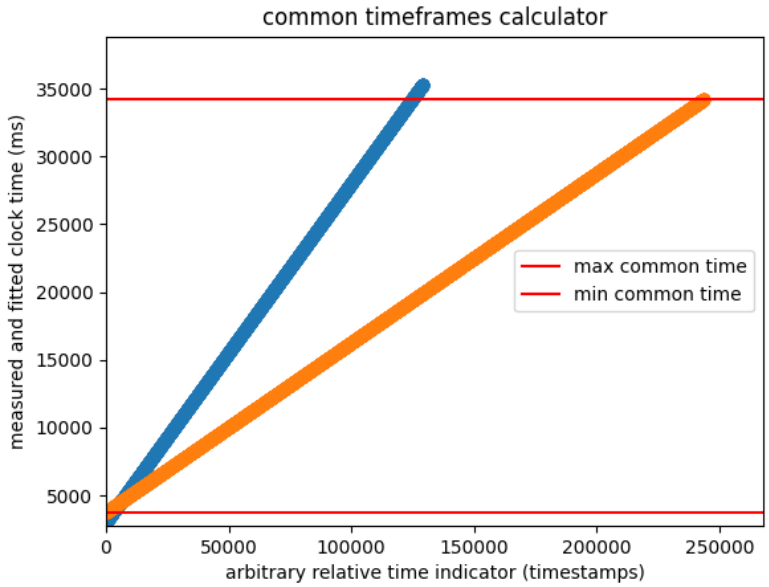


Figure 11: Visualization of two videos in the real-time world

As intuition would dictate, the higher the threshold for considering that two frames are "in sync", the more common frames we find. If the threshold is higher than the delay

in between two frames, we can even find that on frame is "in sync" with two or more frames of the other video, as seen on figure 12 with more than 100 % of kept frames.

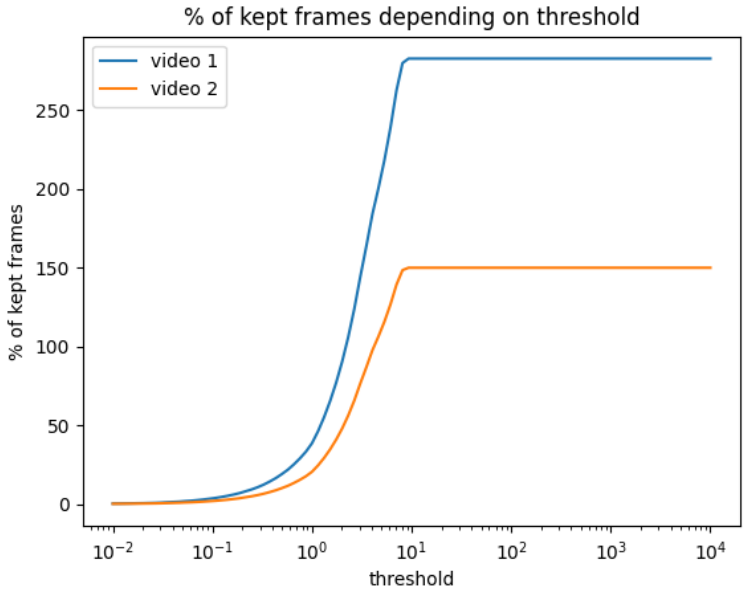


Figure 12: Kept frames in relation to the chosen threshold

Because the frame rates are different and not consistent throughout all the videos, there is a "beating" phenomenon. This means that for a given syncing threshold, we can have extended periods of time during which there is no common frames, and other periods where a lot of common frames are found. We can see this phenomenon on the figure 13. Each blue dot is a kept frame : the length of the green line represents the delay in between the frames (varying from 0 seconds to the chosen threshold) in the form of a standard deviation.

The program

Executable files were generated using *cx_freeze*, to allow using this program without any Python installation, as well as third-party libraries. The project and the documentation are published on *GitHub* [6]. These executables can be found in the 'releases' tab of the project.

All the user interaction is done through a command line interface (CLI). The program can also display some graphs and diagrams as shown in the key steps. This is less user-friendly, but also a lot faster and more robust. An example of the output of the program can be found in the annexes, on page 37.

A tutorial and other explanation on how to run the program is present, as is customary, in the *README.txt* file at the root of the *GitHub* deposit. This file can be found in the annexes, on page 34.

Testing

[6] `https://github.com/trischevreau/Frame-Coupler`

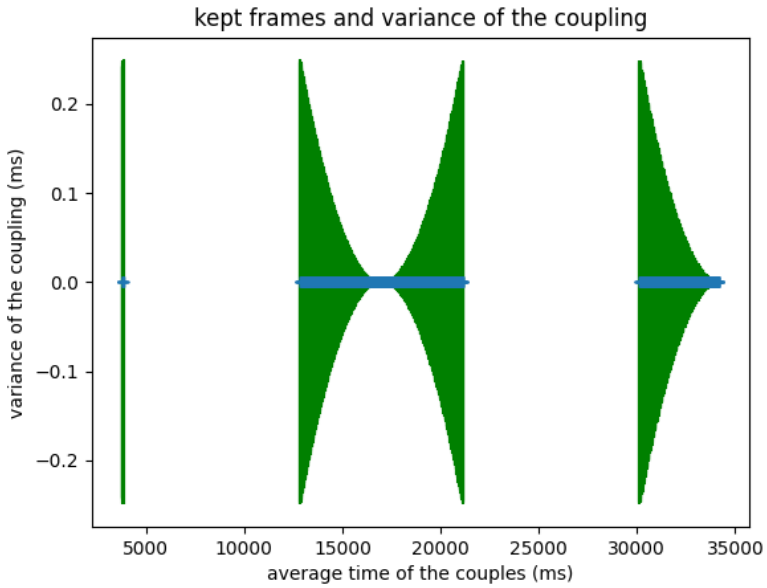


Figure 13: A visualization of the kept frames

The program was tested using two videos of a beam with the characteristics from table 1.

video number	frame rate	resolution	length (number of frames)
1	60	1920 x 1080	3881
2	120	1920 x 1080	7313

Table 1: Characteristics of the videos

With different thresholds, the number of found common frames varied. Table 2 aggregates the results from this experiment. We can see that we always managed to have some common frames ; nonetheless, because of the beating phenomenon, it was not uniformly distributed throughout the lengths of the videos (as seen on figure 13). This can be a problem, because maybe the parts of the videos where we have synchronized frames is not interesting (For example, if the first seconds of the video are calibration frames and that we don't have synchronized frames if this part).

threshold (ms)	common frames	% of frames from video 1	% of frames from video 2
0.5	697	18.0	9.5
1	1542	39.7	21.1
2	3536	91.1	48.4

Table 2: Results of the frame coupling program

3.3.2 Method 2: "selfie stick" trigger

This method uses an electronic circuit to simulate a selfie stick that triggers two smartphones cameras simultaneously.

How does a selfie stick work ?

Note: the corresponding annex can be found on page 40

Most camera applications allow for triggering a photo with a selfie stick (figure 14). These gadgets are passive electronic circuits that use the headphones jack port to 'communicate' with the smartphone.

Figure 14: A common selfie stick (from *Amazon*)

Jack ports have up to four pins: Microphone, Ground, Left and Right channels (figure 15). Most phones check the resistance in between the Microphone and Ground pins : when the resistance is right (about $5k\Omega$), the phone knows that there is a microphone. But when the resistance gets low enough (about 400Ω), this triggers functions on the phone, such as volume up/down or pause. This is also how headsets with buttons and a microphone on the cable work. The camera app then detects these functions, and triggers the photo. The phone must first detect a microphone before starting to look for button presses.

To emulate a selfie stick for only one phone, the circuit was very easy, as it just had to have two resistors and a push-button as seen on figure 16.

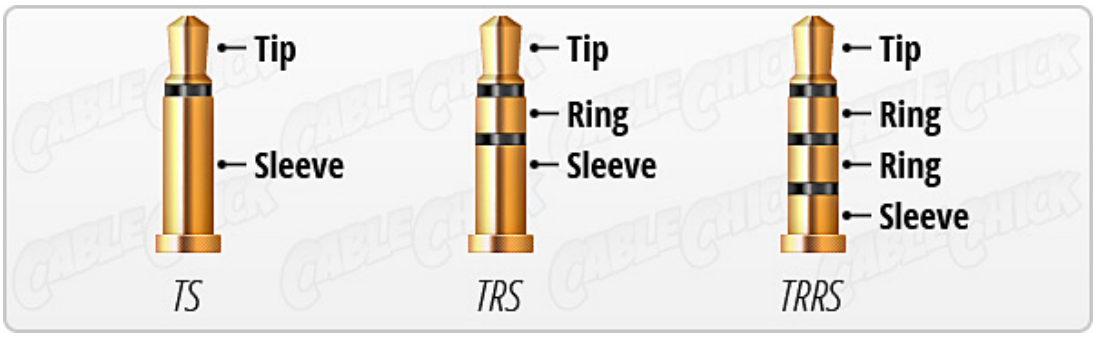


Figure 15: 4-pin jack connector

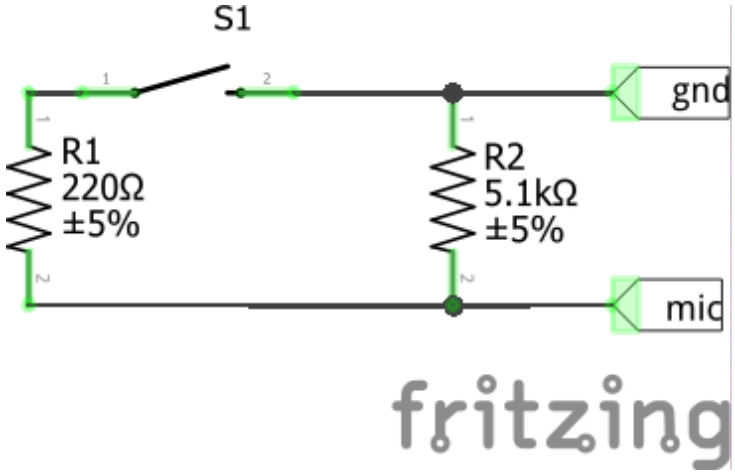


Figure 16: The simple circuitry of a one-phone selfie stick

For two phones, it was more complicated, as I couldn't connect the grounds of the phones with each other without risking damaging the phones. Furthermore, I wanted the circuit to be controlled with a signal generator to ensure maximum precision. I needed to make a circuitry that had one input : the external trigger, and two outputs : two headphone jacks to connect to the smartphone. The outputs needed to have the same exact signal but electronically independent (to avoid short-circuits and mutual interaction). When the input voltage is zero, the output has a resistance of a microphone ; when it is around 3V, the output has a resistance corresponding to the volume up function. Figure 17 shows the scheme I designed.

This scheme uses opto-couplers. Opto-couplers are electronic components used to make circuits electronically independent. When a voltage is applied to them by the first circuit, their resistance in the second circuit drops. This allows to transmit information without transmitting electrons. They can be made simply by taping a LED to a photoresistor (see figure 18); industrially produced opto-couplers exists and are not expensive, such as the 5N35.

With this circuit, and a function generator on the input, it was possible to trigger photos on two phones without risking destroying them. According to the official datasheet of the 5N35, it should also be almost instantaneous in our use case ($< 1ms$). Figure 19 shows a prototype on a breadboard of this circuit. The push-button can be replaced with a pulse signal generator.

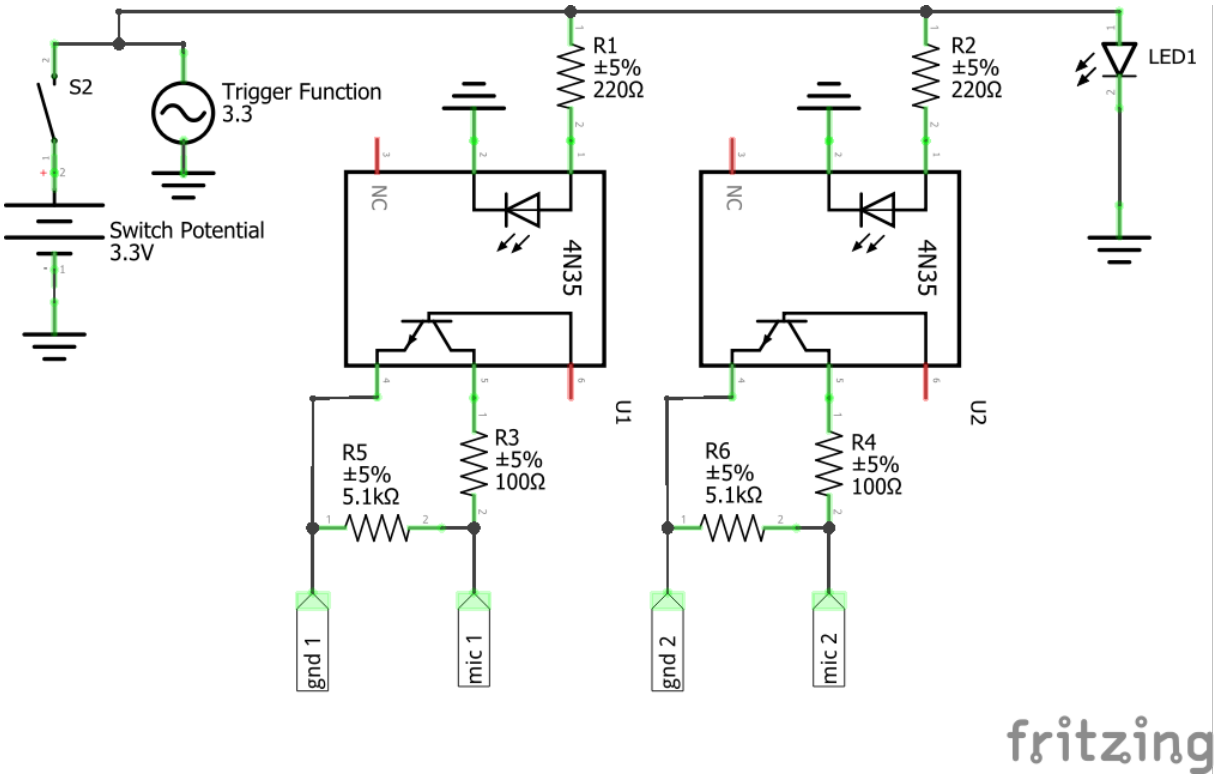


Figure 17: The circuit for two phones and an external trigger

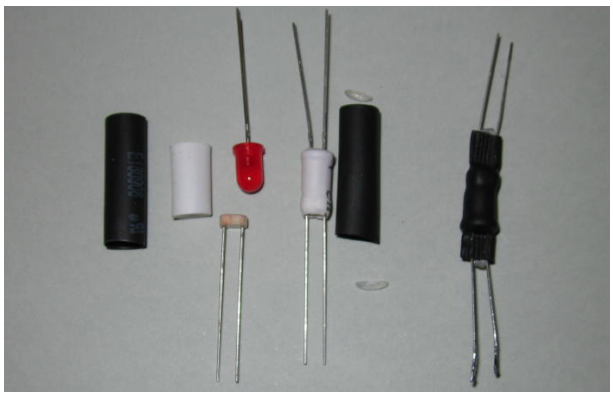


Figure 18: A DIY optocouple that can replace a 4N35 (from *Eliott Sound Products*)

Methodology for measuring the efficiency of the circuit

Note: the corresponding annex can be found on page 47

To test this contraption, I installed two (identical and factory reset) phones in the direction of a computer displaying a 60 fps digital chronometer (see the setup of the experiment on figure 20). Then I triggered periodically the camera using the 'selfie-stick' circuitry and an external function generator in pulse mode. To ensure that each measure of the delay is independent of the others, the period of the function generator was set to $2s$.

For each photo, I read the digital chronometer using text recognition (*pytesseract*, see previous part), and the delay in between the smartphones' photos could be measured

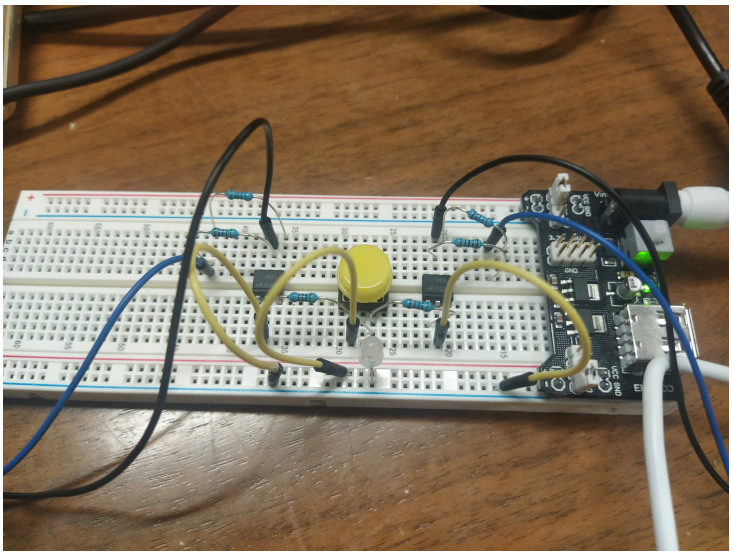


Figure 19: One prototype of the circuit on a breadboard.

Figure 20: The setup of the experiment

by subtracting those two times.

Because of the methodology, the minimum delay I could measure was $17ms$. I expected it to be a problem, but the selfie-stick system was not performing as good as I expected, so this was precise enough.

I chose to test the contraption with 3 different conditions (see table 3). I varied the following parameters :

- the number of pixels of the image (in megapixels)
- the ISO value of the camera, which is a measure of its sensibility to light

The following notable parameters were constant throughout the experiment :

- All the variables regarding the external function generator triggering the photos through the 'selfie stick' circuitry
- The shutter speed of the camera, which is the length during which it is exposed to light. I kept it as short as possible ($1/4000s$) to avoid increasing latency or having a blurred digital chronometer.
- The number of samples (meaning, pairs of photos) was around 140.

Experiment number	Megapixels	ISO value
1	13	400
2	13	1600
3	6	400

Table 3: The different experiments

Results and interpretation

The image 21 is an example of the results obtained for experiment number 3.

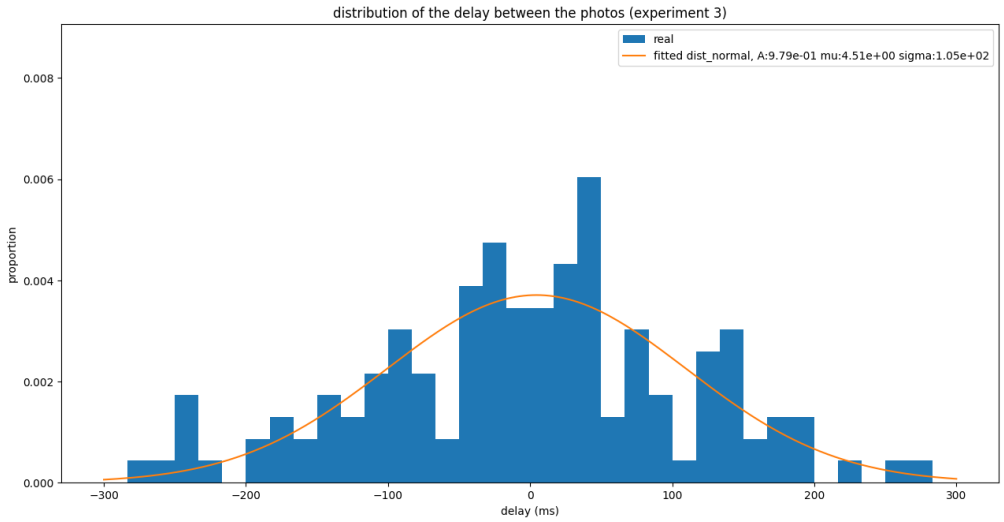


Figure 21: Distribution of the delays of experiment 3

We chose to model the distribution of the delay with a normal curve. If we aggregate all the results, we get table 4.

Experiment number	standard deviation (ms)	mean delay (ms)
1	33.3	1.2
2	30.1	1.8
3	105	4.5

Table 4: Results of the delay measurement and model for each experiment

From these results we get following conclusions (most important first) :

1. This method of triggering cannot be used in the context of dynamic DIC because the studied phenomenon would have to be very slow ($1Hz$ for a maximum error of 10 % in the best-case scenario 2)
2. Counterintuitively, a higher resolution does not mean a bigger delay.
3. The mean value of the delay is so low ($< 5ms$) that we can guess that no phone was less responsive to the trigger than the other.

3.3.3 Method 3: displacement signal as a function

General description

For the two first methods, we wanted to obtain synchronized images, compute the 2D-displacement fields of these images to obtain data and use this data to have a 3D-DIC result. These two last steps were done internally by the DIC pieces of software. This workflow is represented by figure 22

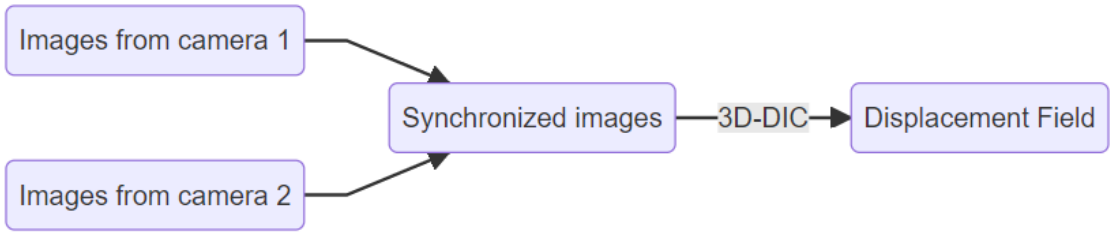


Figure 22: The classic workflow for the two first methods *(NB : The focus of my work was not the 3D-DIC itself, but the getting of synced images.)*

For this last method, we completely change our point of view (figure 23). Instead of wanting to obtain synchronized images from the two cameras, we try to obtain synchronized functions representing the 2D-displacement fields seen by both cameras separately. These functions take the date as an input and a displacement field (for a specific camera) as an output. Then, we can do a 3D-DIC by calling these functions for any value (any date) we want. This means, we don't have to get all the frames synchronized.

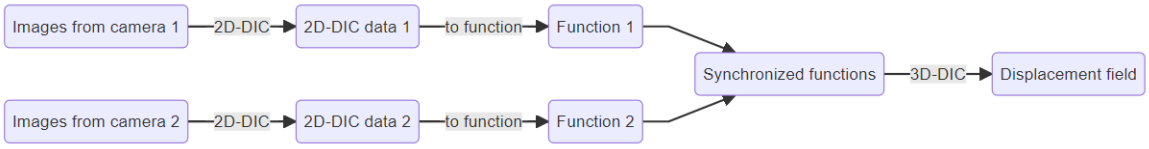


Figure 23: The new workflow for the last tested method

Because my knowledge of DIC still stays limited, I focused primarily on getting the displacement field function from videos taken with a smartphone.

The output of the function and Non-Harmonic Fourier Analysis (NHFA)

Because the output of the function needs to be a matrix representing the displacement field, we chose to transform each value of the matrix into a function. To do that, we can use NHFA.

NHFA is a way to analyse a signal by making a few assumptions on its shape, especially its frequency spectrum. NHFA tests a set of frequencies (for example a linear space of 50 points from 10 to $50Hz$) and returns the frequency with the highest amplitude if it was in the signal. Then, this frequency is deleted from the signal, and we can do the processing again. The result of this analysis is an amplitude, a phase, and a frequency for n iterations of the algorithm. Each iteration gives a new probable harmonic of the signal.

NHFA is especially useful as the input data doesn't need to respect Shannon's theorem. In fact, NHFA works for randomly-sampled signals, which is the case of most videos from smartphones. As explained previously, videos taken with smartphone cameras have a tendency to skip some frames on the videos and to have variable frame rates, meaning we can't really use a sampling frequency as a value as we do for classical Fourier analyses.

To implement a NHFA algorithm into my program, I used Paolo Neris'[7] work as well as his help.

The program

Note: the corresponding annex can be found on page 52

The program I coded in *Python* was meant to transform into functions results from 2D-DIC done using *Digital Image Correlation Engine (DICe)*[8], which is a commonly used free software to do DIC.

To make testing faster and easier, I started by doing the NHFA analysis only on one point of the beam. This meant the function had only two values as an output : the X and Y coordinate of the point. The input was still the same : a value representing the date at which the point was at this position. Then, the program was extended to each point. Because of this, the program has two modes, corresponding to the modes of DICe : "tracking" for one point analysis, and "subset-based" for multiple points analysis.

The found functions are then saved into a CSV file containing the frequency, amplitude, and phase of each harmonic. Because we have a CSV file for each point of the beam, and two coordinates, we have $2n$ CSV files for n analysed points.

The code also contained other functionalities, such as a way to transform results from DICe into videos.

Results

I tested this method with a beam excited at $30Hz$ by the shaker. The base frame rate of the video was $28Hz$. This lower frame rate allows for a higher resolution of the frames. I tracked the end of the beam using the DICe software in 'tracking' mode. The resulting data for the Y axis coordinate of the point of the end of the beam is presented in figure 24.

On this data, we can see that the resonant frequency of the studied beam cannot be seen intuitively because we do not respect Shannons' theorem. To compute the NHFA algorithm and find those harmonics, the assumptions on the signal were as follows

- The signal contains frequencies from 10 to $400Hz$
- We test 4000 frequencies from this range
- The signal has an offset

[7] `http://dx.doi.org/10.1016/j.jsv.2015.06.048`
[8] `https://github.com/dicengine/dice/`

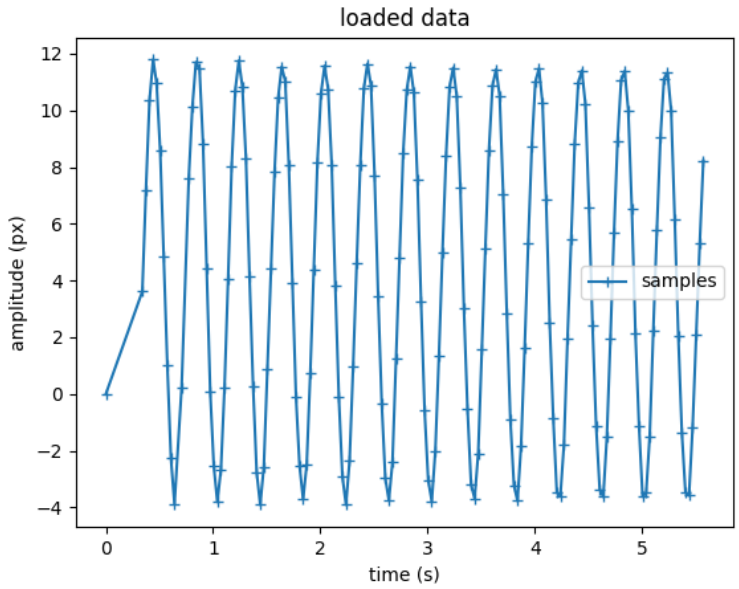


Figure 24: Data resulting from the DICe program (Y axis coordinate of the end of the beam)

The results of the iterative NHFA method are presented in the table 5. We can see that the NHFA algorithm managed to find the resonant frequency. But we can see some artefacts (non-existing frequencies are found) in what the algorithm finds in the next iterations, probably because the input data was only a simple sinusoid.

Offset	4.118	*	*
Harmonic	Amplitude	Frequency	Phase
1	11.695	30.015	-0.978
2	9.454	30.003	2.940
3	7.686	32.518	-1.968
4	7.644	32.531	-2.211
5	7.593	32.505	-1.725

Table 5: Results of the NHFA algorithm on the test data

Visually, the results can be seen on figure 25.

If we recreate the signal using the results for 1 iteration (figure 26) and for 10 iterations (figure 27), we can see that the artefacts have a significant impact on the results. But we can also see that for the recreated signal for 1 iteration fits well with the data points.

Next steps

The next things to do with this would be to test ways to synchronize the functions. We have multiple solutions :

- Assumptions on the signal (such as the timing of the minimum or the maximum)

- Giving a short pulse easily seen on the functions as a synchronizing point
- Intercorrelation techniques

  Then, we would be able to extract 3D-DIC information from that.

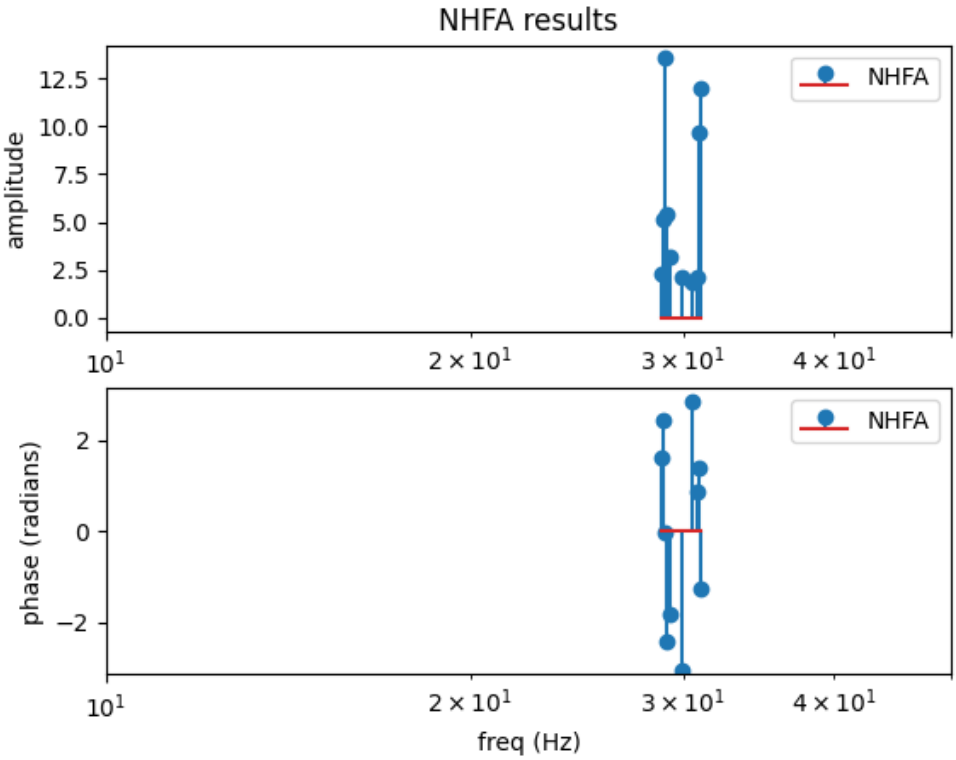


Figure 25: Caption

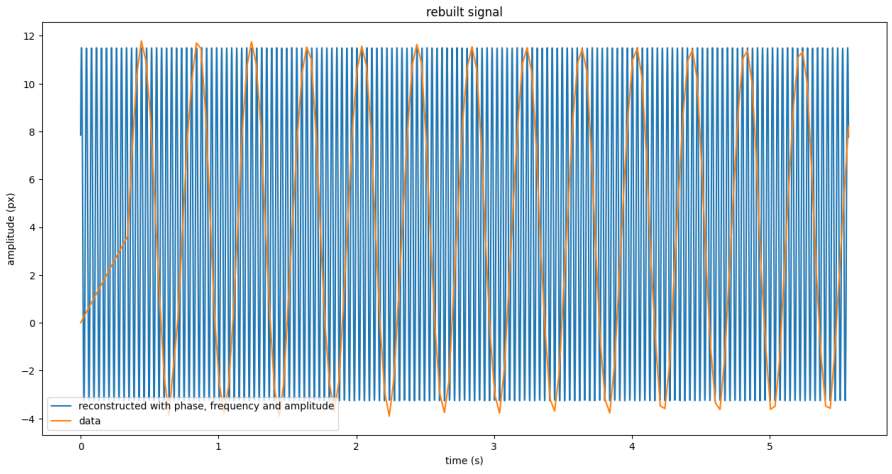


Figure 26: Recreated signal for 1 iteration of the NHFA algorithm

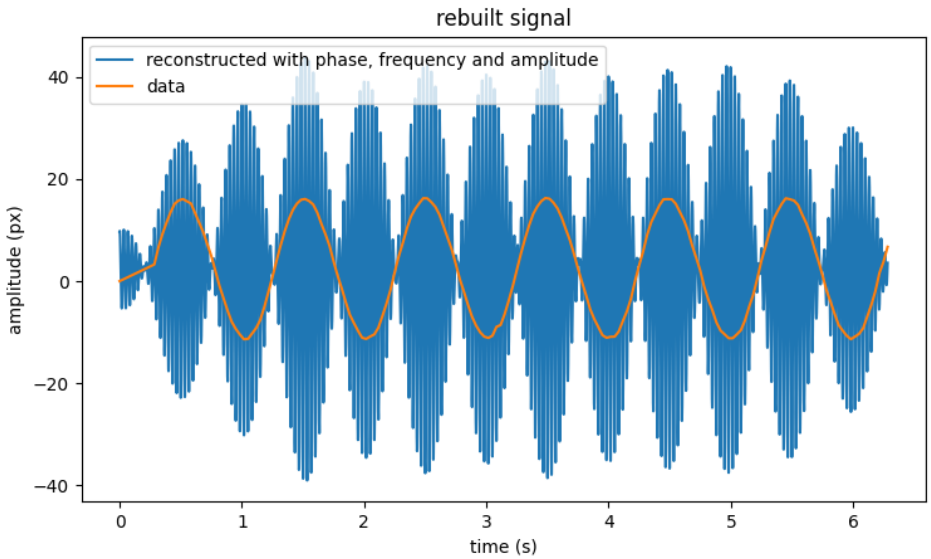


Figure 27: Recreated signal for 10 iterations of the NHFA algorithm

3.3.4 Conclusion on the tested methods

The comparison between the two first methods is presented on figure 28. Method 3 is not included in this comparison as it is unfinished. To my opinion, method 1 is promising. Furthermore, it is already usable.

Criterion	The selfie stick trigger	The video frame coupler
Speed	Fast (no processing)	Slow (heavy post processing)
Accuracy	Low (~100ms)	High (~10ms)
Setting Complexity	High (circuitry, function generator ...)	Medium (python third-party library and Tesseract setting)

Figure 28: Comparison of method 1 and 2

3.4 Other work

During my internship, I also wrote tutorials for the future trainees. For example, I wrote a tutorial on how to use SFTP (SSH File Transfer Protocol) between phones and a computer to transfer large video files during experiments. This tutorial can be found on page 56 of the annexes.

I also tested the use of the LED lights of the lab in 'stroboscope' mode (we could potentially test a fourth synchronizing method using them). These tutorials can be found on pages 59 and 61 of the annexes.

4 [FR] Place de ce stage dans la construction de mon projet professionnel

4.1 L'international

J'ai beaucoup apprécié le fait de vivre dans une nouvelle ville, et le fait d'être confronté à d'autres manières de travailler et de s'organiser, mais aussi de s'amuser. Néanmoins, ce stage a confirmé mon envie de ne pas poursuivre une carrière à l'international. La proximité (relative) avec ma famille, la facilité d'évoluer dans un système que je connais et comprends mieux et l'intérêt que j'ai pour la France sont autant de facteurs qui ne me donnent pas envie d'évoluer à l'étranger.

Je suis aussi très reconnaissant des aides financières et du soutien que j'ai pu recevoir de la région, de l'école, du programme Erasmus et d'autres. En effet, ce stage m'a fait réaliser que certaines choses me paraissent évidentes, mais ne sont pas acquises, même dans un pays aussi proche que nous et aussi développé que l'Italie. Par exemple, certaines conversations à propos de la sécurité sociale italienne m'ont très étonné, et mes critiques envers le système de santé français seront maintenant bien plus modérées.

À l'inverse, certaines choses du quotidien sont bien plus pratiques. Par exemple à Turin, le tri des déchets se fait dans des poubelles au pied des immeubles et comprend une poubelle tout-venant, de verre et métal, de compost, de cartons et de plastiques. Des amendes sont distribuées en cas de tri insatisfaisant. Cela fait que globalement, le tri paraît assez respecté. Il ne faut pas, comme en France, courir après les conteneurs (bac jaune, bac à verre, composteurs en ville très éloignés).

Pour autant, dans le cas de l'Italie, j'ai eu l'impression que les différents systèmes publics et usages ne correspondraient pas à mes besoins si j'y habitais.

4.2 Le monde académique

Ce stage a fait évoluer mon avis sur le monde académique, même si je reste conscient que l'ambiance et l'état d'esprit sont extrêmement variables en fonction du groupe de travail. Néanmoins, le fait d'être autonome, de devoir expérimenter, mais aussi de se documenter et de discuter avec des chercheurs passionnés par leurs projets de recherche était enthousiasmant. L'envie de savoir et un certain amour pour la science et la technologie qui régnait à la fois dans le travail, mais aussi dans les discussions personnelles, sont des choses que je ne pense pas retrouver à cette échelle dans un travail d'ingénieur "conventionnel". Probablement, dans les entreprises classiques dans lesquelles je me vois chercher du travail, la science est plus un moyen qu'un but. Ceci est peut-être moins vrai pour les startups qui développent des technologies peu explorées.

Néanmoins, il subsiste en Italie (comme en France) des magouilles et autres arrangements propres apparemment au monde académique et qui sont des choses qui me déplaisent. Par exemple, l'un de mes collègues italiens m'avait expliqué que pour être sûr d'être le seul candidat à sa thèse, lui et son tuteur avaient dû déposer l'offre sur un site d'un autre département qui n'avait rien à voir et à demander des prérequis qui correspondaient exactement au cursus de ce collègue. Je sais que ce genre de choses est monnaie

courante aussi en France, et je trouve que c'est malhonnête, car je ne peux m'empêcher de penser au candidat mal informé qui s'embête à monter un dossier alors qu'il n'a aucune chance et qui ne comprendra pas pourquoi il est refusé.

Après ce stage, je n'exclus pas l'idée de faire une thèse. J'étais en effet dans le bureau des thésards, et j'ai eu l'impression qu'ils étaient heureux de ce qu'ils faisaient (même s'ils n'étaient pas payés cher vis-à-vis du travail fourni).

4.3 Le travail d'ingénieur

Durant ce stage, j'ai été amené à concevoir et à développer des circuits électroniques et informatiques. Ensuite, il fallait valider des exigences avec des tests et rédiger une documentation. Ce processus était très satisfaisant pour moi et j'espère que je pourrai retrouver cela dans mon futur travail.

C'est pourquoi je pense que les domaines de la recherche et développement et de l'innovation / entrepreneuriat sont ceux auxquels je dois m'intéresser. Dans les domaines du conseil ou de la gestion des risques, je ne retrouverai pas ce processus sous cette forme.

5 [FR] Conclusion

Ce stage m'a permis d'appliquer mes connaissances techniques au travers de travaux variés durant lesquels j'ai développé mon autonomie. J'ai aussi dû utiliser mes capacités de recherche et de documentation. J'ai pu comprendre et utiliser les techniques de *Digital Image Correlation* et tester des cas d'utilisation innovants. J'ai aussi dû apprendre à montrer les résultats de mes travaux de manière plus spontanée, moins scolaire et en anglais. En outre, l'analyse détaillée des trois méthodes testées, a enrichi ma boîte à outils en tant qu'ingénieur.

Ce stage m'a aussi offert l'opportunité de m'immerger dans un environnement international et académique, améliorant ainsi ma compréhension de ces deux aspects de ma carrière.

6 [EN] Annexes

Tristan Chevreau, Summer 2023
tristan.chevreau@gmail.com
tristan.chevreau@ecl21.ec-lyon.fr
https://www.linkedin.com/in/tristan-chevreau-ingenieur/
+33 6 89 50 69 11

Dear Reader,

This folder contains the work I did during this summer internship at Polito. I tried to make it as clear as possible, so I separated my work into numerated folders.

Footnotes are the names of the folders containing documents in relation to each work. Each folder contains additional READMEs or files.

Here is a quick summary of what I did :

The subject of my internship was to find ways to make 3D - Digital Image Correlation using two smartphones. For that, I needed to have synchronized images of the object I wanted to do DIC on. I tested mainly 3 methods :

1) Taking high frame rate videos and synchronizing the images using a digital clock display in the video field of view. I used a script that I called 'Video Frame Coupler'[1], published on GitHub and with a downloadable EXE file in the 'releases' panel.

2) Taking photos in a synchronous way using an electronic circuitry mimicking a selfie stick[2]. I then measured the delay in between the photos from each phone and saw that it was not usable for dynamic DIC[3].

3) Taking both videos independently to do DIC on. I used the DICe software (https://github.com/dicengine/dice/releases). I then recreated the displacement signal using the 'Non Harmonic Fourier Analysis' (NHFA)[4], to make these signals into sums of sine functions. The goal after that would be to combine these two functions into 3D displacement fields without any sampling constraints.

I conclude that Methods 1) and 3) should be further investigated, and that method 2) does not seem promising.

1 Video Frame Coupler (https://github.com/trischevreau/Frame-Coupler)
2 Selfie Stick Circuitry for Android Smartphones
3 Delay Between Photos Taken with a Smartphone Using an External Trigger
4 DICe and NHFA

Other works I did that I think could be interesting to someone else:

- To transfer the files from the phone to the computer I used the SFTP protocol which was easier than Bluetooth.[5]
- I modelized a parametric beam using *FreeCAD* to 3D-print it[6]. Because it is parametric, you can fine-tune it to a correct theoritical modal frequency.
- I tried to obtain synchronized images using stroboscopes but I did not go very deep into it. I have made some manuals and study about the 'strobo' mode of the *Stratus Lights 200 W LED Module*[7]. I abandonned it as it defeats the purpose of only using smartphones if you need a stroboscope.
- I tried to do an implementation of muDIC (https://mudic.readthedocs.io/en/latest/) and added some interfaces around it to make it easier to use. But I then used only DICe because it is more common and functional.[8]

I also added a presentation I did to show my work to M Botto and M Neri in the middle of my internship[9].

I hope it was clear ! Thanks for this great internship.

5 Using SFTP to Transfer Files from a Phone to a Computer
6 3D Beam
7 Strobo with a Stratus Lights 200 W LED Module
8 Python DIC
9 Two Sync Techniques

Frame-Coupler

Short description

This program selects common images from a set of videos, using the display of a digital clock in the field of view to synchronize their times.

Long description

This program is used to find common frames between two or more videos. To use it, you need to display a digital clock in the field of view of both videos. This digital clock will be read digitally and will be used to have a precise date associated to every image of the videos. This allows to film something with different devices, different start and end times for the video, different frame rates, etc. and get common frames between this videos anyway.

This common frames are not evenly spaced throughout time. For a set of two videos if the frame rates of the devices are close but not exactly the same, there will be a 'beating' phenomenon. If the frame rates are not consistent, the spacing between common frames will feel random.

This is especially useful as it is often hard to synchronize the start of videos filmed using two or more different devices ; furthermore, depending on the device, the frame rate can be inconsistent and there could be frames missing.

This digital clock can have a refresh rate that is under the one of the cameras filming it as the fitting will allow to discriminate the frames anyway.

These videos should be MPEG-4 files, which will be converted to gray JPEG frames. Each frame of a MPEG-4 file is associated with a time stamp.

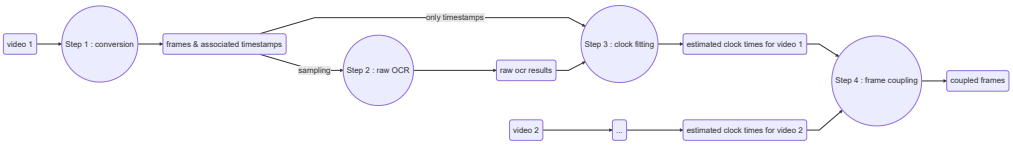
The digital clock is supposed to be of this model : https://www.youtube.com/watch?v=rf2Lmfqi5ZM

Releases

You can find executable files generated with cx_Freeze in the 'releases' tab.

Work flow

Here is a flowchart to understand how it works :



The process is divided into 4 steps. These steps can be executed independently using their own python files, or the full process can be executed in one shot using the *main.py* file.

- step 1 : the MPEG-4 (.mp4) video is converted into gray JPEG (.jpg) frames and a pickle file containing the timestamps of each frame.
  - .png would probably be better than .jpg, however *tesseract* seems to prefer .jpg files.
  - each extractable frame is saved.

- step 2 : OCR (Optical Character Recognition) is performed on some of theses frames. The user chooses a zone were we should find the digital clock. The *tesseract* API *pytesseract* is then used to try and read this clock. The raw results of this process are saved in a *python pickle* file in the folder of the frames.
- step 3 : The raw results of the preceding step are processed to clean OCR errors and get a date in milliseconds (for example, 00:01:3@:I33 is transformed to 00:01:30:133 and then transformed to 90133 milliseconds). This milliseconds readings are then used to fit a linear curve using the timestamps of the frames. After this step, we have associated a date in milliseconds to every frame using their (even those were we couldn't read the digital clock). The results of this process are saved in another *python pickle* file in the folder of the frames.
- step 4 : This last step can be done when steps 1 to 3 were done to at least 2 videos. During this last steps, the dates in milliseconds associated to every frames are read to find common frames between the two videos. A frame is deemed 'common' to both videos when the difference of their dates is under a certain threshold. The results of this process is the coupled frames in different folders and a .csv file containing the dates.

How to install the program

The program was tested and coded using *python 3.10*. Any equal or superior version *python* install should work fine.

You can use 'verify_librairies.py' to see which imports are missing and if your install went well.

You should install *Tesseract-OCR* from *Google* as a program (https://github.com/tesseract-ocr) to use the *pytesseract* library. You can find setup programs here : https://tesseract-ocr.github.io/tessdoc/Installation.html. You should then find the executable file of *Tesseract* (generally found at 'C:\Program Files\Tesseract-OCR\tesseract.exe') and copy its path to the variable 'pytesseract_exe_file_path' in the 'run_parameters.py' file.

Third-Party *python* librairies that you also need to install :

- numpy (all steps)
- matplotlib.pyplot (all steps)
- cv2 (step1)
- PIL (step1, step2)
- pytesseract (step2)

How to run them program

The file 'run_parameters.py' contains variables that you should change to match your needs and your installation of *pytesseract*, especially the *Tesseract* executable path, but also :

- 'info_level' : How much the user should be informed of the inner workings of the programs during runtime.
- 'save' : Should the coupled images be saved to a separate folder.

A more complete description of these values can be found directly as comments in the 'run_parameters.py' file.

The program can be used in two ways, depending on which file is the ran as the main file.

- Using 'main.py' to do all the steps at once (easier to use but fallible) ; you just need to follow the prompts of the program. This is meant to be almost entirely command-line (except choosing the cropping for the OCR).
- Using each step individually to control more the results. Using these individual steps, there is more dialogs and user interfaces. You can also go back easily when you are not happy with the results of one of the steps.

License

```
### VIDEO FRAME COUPLER

### running parameters (can be changed in the run_parameters.py file) :
### user info level = 3/3 ; save = True
### Tesseract EXE filepath C:\Program Files\Tesseract-OCR\tesseract.exe

### [c] how many videos to couple ? 2
### [c] between how many frames should we perform OCR ? (1 means every frame, 2
means every other frame, etc.) 27
### [c] select the filepath of the videos
### [c] input the absolute filepath of video 1 : c:/images_dic/c2/1.mp4
### [i] video 0 is c:/images_dic/c2/1.mp4
### [c] input the absolute filepath of video 2 : c:/images_dic/c2/2.mp4
### [i] video 1 is c:/images_dic/c2/2.mp4
### [c] where do you want to output the results (absolute path) ?
c:/images_dic/c2/res
### [i] results folder is c:/images_dic/c2/res

### STEP 1 : Conversion to frames

### [c] it seems like the videos were already converted to gray frames, is it
correct ? (Y/n)
### [i] skipping step 1

### STEP 2 : raw OCR

[step2][i] beginning OCR of frames from folder c:/images_dic/c2/1
[step2][i] choose a crop frame (see opened window)
[step2][i][0001/3876] found 00:00:03.036 in 0_0.jpg
[step2][i][0028/3876] found 00:00:03.253 in 900_0.jpg
(...)
[step2][i][3835/3876] found 00:00:34.984 in 127800_00000000001.jpg
[step2][i][3862/3876] found 00:00:35.201 in 128700_00000000001.jpg
[step2][w] file /__raw_OCR_results already exists, it will be removed and replaced
[step2] finished OCR on c:/images_dic/c2/1

[step2][i] beginning OCR of frames from folder c:/images_dic/c2/2
[step2][i] choose a crop frame (see opened window)
[step2][i][0001/7308] found 00:00:03.720 in 0_0.jpg
[step2][i][0028/7308] found 00:00:03.870 in 899_6444444444445.jpg
(...)
[step2][i][7264/7308] found | 00:00:34,008 in 242003_2.jpg
[step2][i][7291/7308] found | 00:00:34-1 47 in 242902_7777777778.jpg
[step2][w] file /__raw_OCR_results already exists, it will be removed and replaced
[step2] finished OCR on c:/images_dic/c2/2


### STEP 3 : clock fitting

[step3][i] raw OCR results loaded from c:/images_dic/c2/1
[step3][i][001/144] 00:00:03.036 becomes 3036 ms
```

```
[step3][i][002/144] 00:00:03.253 becomes 3253 ms
(...)
[step3][i][143/144] 00:00:34.984 becomes 34984 ms
[step3][i][144/144] 00:00:35.201 becomes 35201 ms
[step3][i] computed 144 values.
[step3][w] 13 values were unreadable
[step3][w] 1 values were obviously too high
[step3][w] there is 130 values left
[step3][w] 0 values were discarded on iter 1 with threshold 2000
[step3][w] there is 130 values left
[step3][w] 0 values were discarded on iter 2 with threshold 500
[step3][w] there is 130 values left
[step3][w] 0 values were discarded on iter 3 with threshold 100
[step3][w] there is 130 values left
[step3][r] average sampling frequency : 4.445386809763284 Hz with sigma
6.026098258984974e-14 Hz
[step3][i] 90.27777777777777 % of the frames with OCR results where deemed
appropriate for fitting
[step3][i] 3.353973168214654 % of the total of frames where deemed appropriate for
fitting
[step3][w] file /__clock_fitting already exists, it will be removed and replaced
[step3] finished clock fitting on c:/images_dic/c2/1

[step3][i] raw OCR results loaded from c:/images_dic/c2/2
[step3][i][001/271] 00:00:03.720 becomes 3720 ms
[step3][i][002/271] 00:00:03.870 becomes 3870 ms
(...)
[step3][i][270/271] | 00:00:34,008 becomes 34008 ms
[step3][i][271/271] | 00:00:34-1 47 becomes 34147 ms
[step3][i] computed 271 values.
[step3][w] 54 values were unreadable
[step3][w] 21 values were obviously too high
[step3][w] there is 196 values left
[step3][w] 72 values were discarded on iter 1 with threshold 2000
[step3][w] there is 124 values left
[step3][w] 4 values were discarded on iter 2 with threshold 500
[step3][w] there is 120 values left
[step3][w] 3 values were discarded on iter 3 with threshold 100
[step3][w] there is 117 values left
[step3][r] average sampling frequency : 8.892914867319062 Hz with sigma
0.0005640822479941742 Hz
[step3][i] 43.17343173431734 % of the frames with OCR results where deemed
appropriate for fitting
[step3][i] 1.6009852216748768 % of the total of frames where deemed appropriate
for fitting
[step3][w] file /__clock_fitting already exists, it will be removed and replaced
[step3] finished clock fitting on c:/images_dic/c2/2


### STEP 4 : frame coupling
```

```
[step4][i] clock fitting results loaded
[step4][i] there is 30432.053433018973 ms in common starting from
3765.755422954779 ms
[step4][i][1/100] with threshold 0.01, found 10 common frames
[step4][i][2/100] with threshold 0.011497569953977356, found 11 common frames
(...)
[step4][i][99/100] with threshold 8697.490026177835, found 10960 common frames
[step4][i][100/100] with threshold 10000.0, found 10960 common frames

[step4][c] choose a threshold for the frame coupling (ms) : 1

[step4][i] created saving folders
[step4][i][1/1496] saving frame at t = 3770.408171449431 ms
[step4][i][2/1496] saving frame at t = 3778.7410965641216 ms
(...)
[step4][i][1495/1496] saving frame at t = 34189.462444182485 ms
[step4][i][1496/1496] saving frame at t = 34197.79606370506 ms
[step4][i] with threshold 1.0, found 1496 common frames
[step4][i] this represents :
[step4][i] 38.59649122807018 % of the frames from video 1
[step4][i] 20.47071702244116 % of the frames from video 2
[step4][i] finished saving times

### DONE
```

Selfie stick trigger

Tristan Chevreau, tristan.chevreau@gmail.com, summer 2023

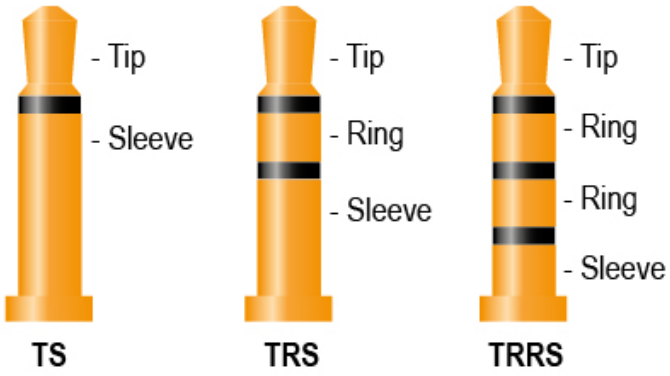
Note : a Fritzing setup file as well as the Fritzing files for the schemes can be found in this folder.

Intro

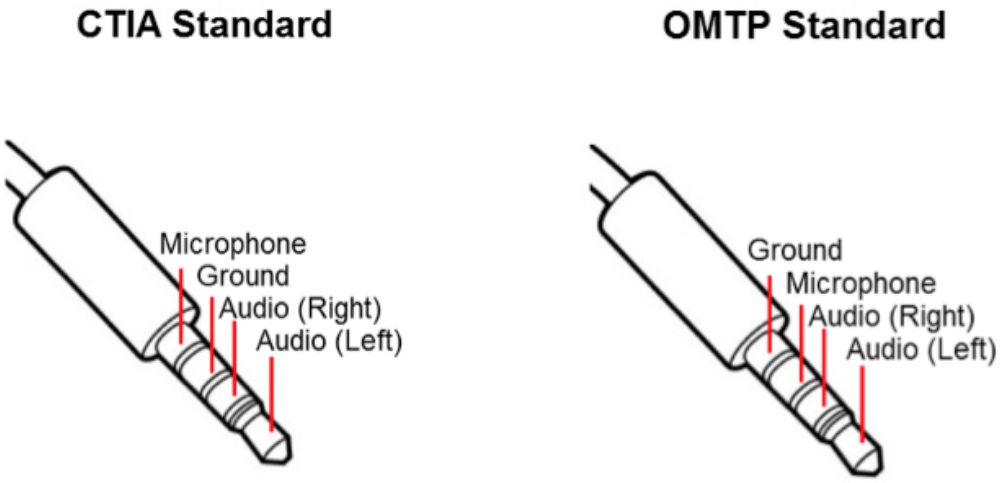
This document describes a way to trigger simultaneously some functions on *Android* smart phones (volume up/down, play/pause, media) using the 3.5 mm jack plug and some circuitry, by simulating a headset or a selfie stick. This can be useful to trigger simultaneously videos recordings, photos or any other app detecting these functions.

Required connectors

To make this, you should have 3.5 mm Jack TRRS connectors. These are also called 4-pole jacks. See the image below to understand the difference.

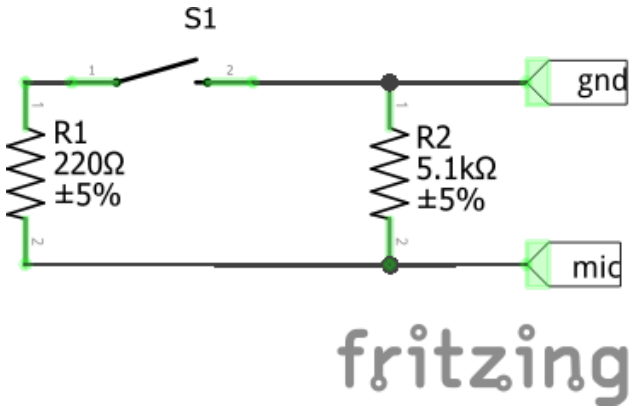


There are two standards for the cabling : CTIA (L,R,Gnd,Mic) and OMTP (L,R,Mic,Gnd). CTIA is more frequent in the USA. If it doesn't work, try the OMTP cabling. If you use Chinese TRRS-to-nude-cables adapters the labels will probably be inverted, as the Chinese standard is supposed to be OMTP. Confusion between OMTP or CTIA cabling should not destroy your phone nor your circuitry but it may explain why it doesn't work at first.



The basic scheme for a selfie stick (1 phone and switch only)

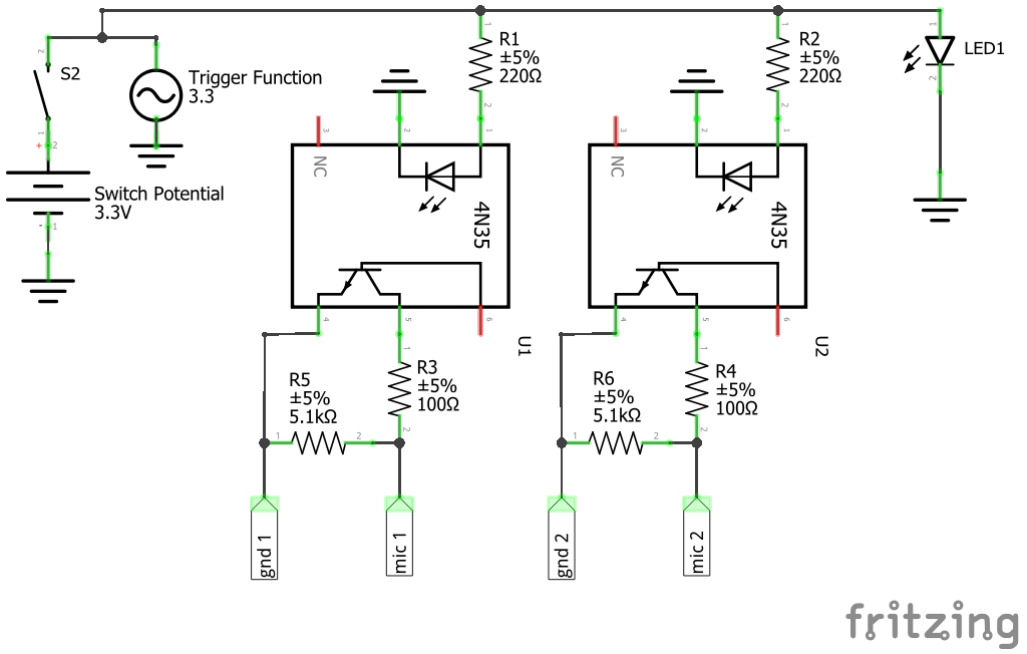
It is very easy to do a selfie stick. For one phone and no function-generator trigger, we obtain this :



The phone detects resistance changes between "ground" and "mic", and this is what triggers specific functions. Different resistances mean different functions. See "Functions and equivalent resistors" paragraph to adjust R1 to trigger the desired function. This is a basis for the design of the next scheme.

The scheme (2 phones or more)

This scheme can only work for *Android* phones, as *Apple* phones require a dedicated solution to trigger functions with the 3.5 mm jack plug.

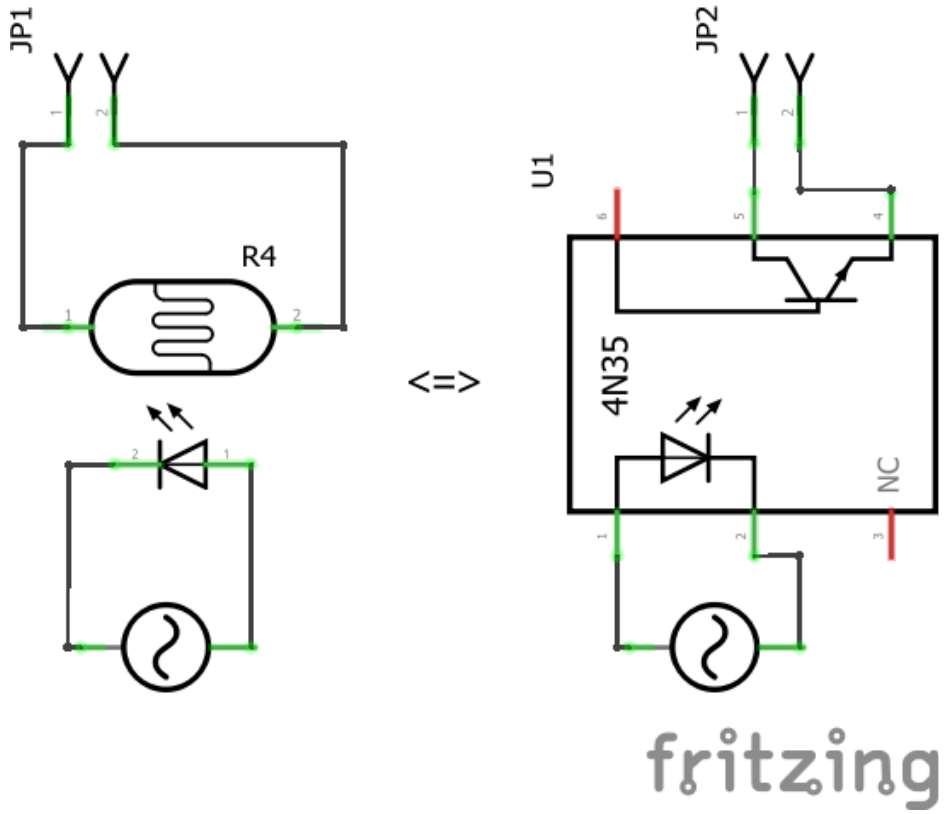


- The triggering can be done in two ways :
  - Using a function generator sending 3.3 V impulses (0 V at rest) ("trigger function")
  - Using a switch and a 3.3 V voltage source ("switch potential")
  - On the scheme, both of them are represented for convenience. **In reality, you should only implement one of the two solutions.** If both solutions are implemented, when clicking on the switch, the voltage source ("switch potential") will try to force a 3.3 V on the node of R1, R2 and LED1. This will conflict with the function generator ("trigger function") that will try to force 0 V on this node. This may damage both equipments. If you want to have both solutions available, you can leave the S2 switch but turn off or disconnect the voltage source when using the function generator.
- The ground of the scheme is here the ground of the function generator ("trigger function") or the voltage source ("switch potential") of the button S2. The input voltage **should not exceed 3.3 V**.
- The 4N35 is a way to link two circuits without having any electrical contacts. This is especially useful as we can't be sure that the ground of both phones as well as the ground of the trigger voltage is exactly the same potential. **Do not connect the grounds of the two phones together ; Do not connect the ground of the function generator or the voltage source of the S2 switch to the ground of any phone. Doing so WILL damage your phone batteries, and creates a fire hazard.** If you don't have any 4N35 component, it can also be done by putting a photo-resistor close to a LED light, see next paragraph. Using such methods is a way to ensure that the phone won't be damaged by the experiment as it won't be connected to a power source.
- LED1 is just a visual indicator to know if the button was clicked, or if the trigger was on. It is not necessary to add this LED light for the system to work properly. If a high current may pass through the LED, you should add a current-limiting resistor.
- R1 and R2 are a protection for the 4N35 opto-couplers.

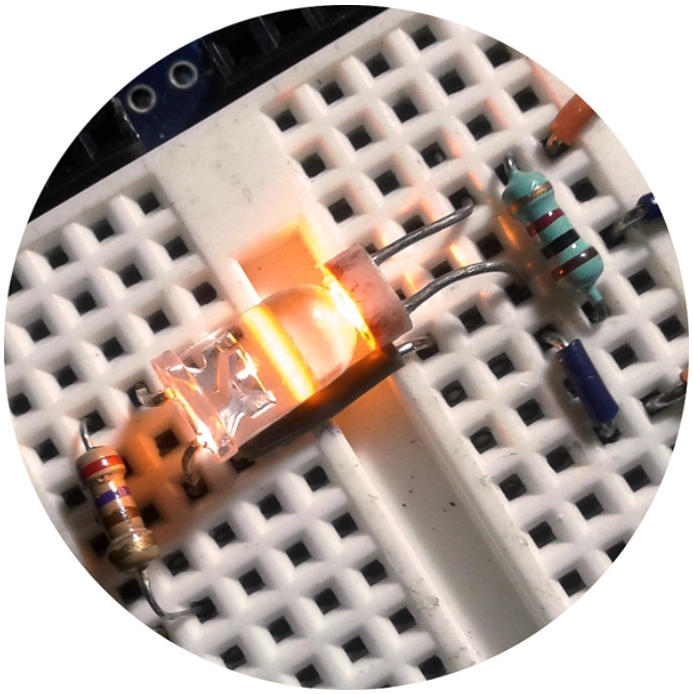
- R3 and R4 can be changed to another resistor to match other functions (volume up, volume down, media button, play/pause), see the *Android* specifications in the sources below. When changing R3 and R4 to match another function, do not forget that the 4N35 also acts as a series resistance. You can measure this resistance using a multimeter while the jack plug is not connected to any phone.
- R5 and R6 are a way to simulate the presence of a microphone, thus enabling triggering functions on the phone. Without these resistors, the phone will believe that it is only a 3 pole Jack and not a 4 pole Jack. **The phone decides whether the jack is 3 or 4 pole just after it is connected, so if there was a problem in your connection or if you changed the circuitry, you should plug the jack connector out and in once again.** On some phones, a "headset with a microphone" icon is displayed on the top right corner of the screen when it is detected as a 4 pole (and "headset only" for 3 pole). See the image below.

- The mic1, mic2 and gnd1, gnd2 ouputs should be connected to the last ring and sleeve of the TRRS connectors, as explained on the previous paragraph. If this doesn't work, **try inverting gndX and micX, because of CTIA vs OMTP cabling** (see previous paragraph).
- If you want to use this for more than one phone, just duplicate the 4N35 circuit.

Replacing the 4N35 with a LED light and a photo-resistor or LDR (Light-Dependent Resistor)

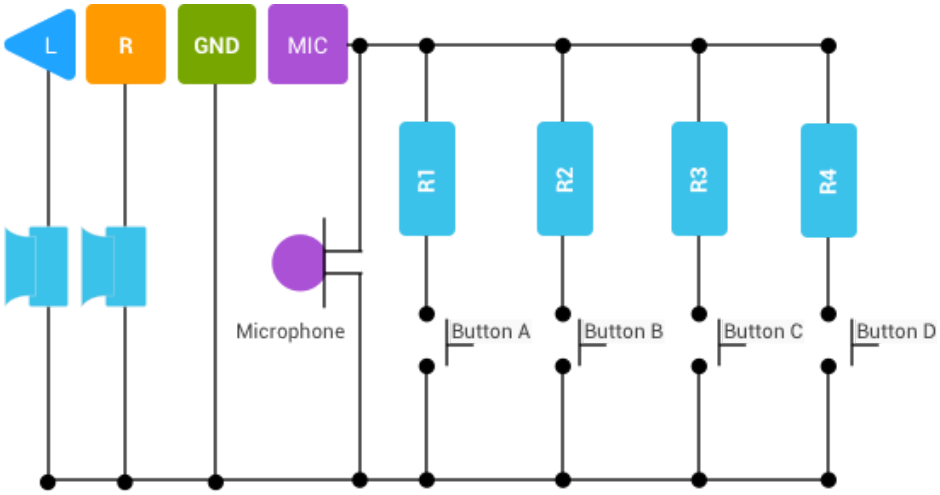


Here is how you can replace the 4N35 with a photo-resistor (these are also called LDR). The output resistance will be different (depending on your photo-resistor) ; on the previous scheme, you may have to adjust R3 and R4, just use a multimeter to test it. This kind of circuit or method is called a *vactrol* or resistive opto-isolator. See a real-life example below :



Functions and equivalent resistors

To adjust R3 and R4 plus their series equivalent resistor of the 4N35 opto-couplers, you can use these values of reference :



Rmic = 5000 ohms R1 = 0 ohms R2 = 252 ohms R3 = 519 ohms R4 = 139 ohms

You can see that the resistor values are very precise. In reality, there is some kind of tolerance, of the order of $\pm 50\Omega$.

This scheme shows the equivalent resistor corresponding to the functions (see tabular of functions below)

Control Function	Accessory Support	Description
Function A	Required	Play/pause/hook (Short Press), Trigger Assist (Long Press), Next (Double Press)
Function B	Optional	Vol+
Function C	Optional	Vol-
Function D	Optional	Reserved (Pixel devices use this to launch voice commands)

Sources

- https://source.android.com/docs/core/interaction/accessories/headset/plug-headset-spec?hl=en contains the specifications for which equivalent resistor does which function
- https://www.vishay.com/docs/81181/4n35.pdf datasheet for a typical 4N35 opto-coupler
- https://en.wikipedia.org/wiki/Resistive_opto-isolator
- https://en.wikipedia.org/wiki/Opto-isolator

Measurement and analysis of the delay between photos taken with a smartphone using an external trigger

Note : All of the code is commented and available in this folder.

Setting of the experiment

Two phones are triggered externally with a 'selfie stick' trigger as explained in another file. The input of the 'selfie stick' circuitry is a function generator doing a square wave with an amplitude of 3.320 V, an offset of 1.65 V, a duty of 3% and a frequency of 0.5 Hz.

A photo is then triggered every 2 seconds : this is the good compromise between speed of the experiment and being sure that each photo is not impacted by the processing of the previous photo.

The phones take a photo of a computer displaying a 60 fps chronometer. The chronometer has a maximum increment of $16.6667ms$, meaning each measure has a $\pm 8.3334ms$ uncertainty.

The phones are both *Huawei Mate s*. To take the photo, we used the default photo application in 'pro' mode to control the settings. The phones were very recently factory reset. Most options were deactivated (wi-fi, bluetooth, ...), and all programs were shut off. This is to ensure that nothing would slow down the phones.

We varied the type and resolution of the photo taken, and the ISO setting (sensitivity to light). The shutter speed was always the same : 1/4000 s, which is the smallest (fastest) possible.

Experiment N	Type of the photo	ISO
1	4:3 13M 4160x3120	400
2	4:3 13M 4160x3120	1600
3	16:9 6M 3264x1840	400

Theory

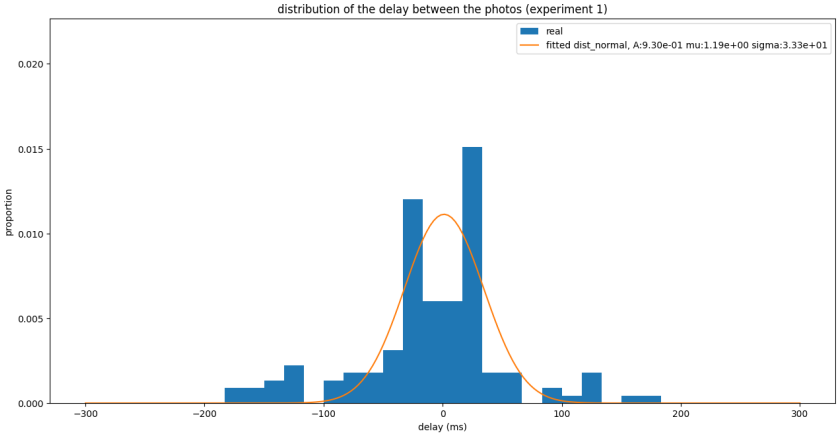
We write T_1 and T_2 the random variable associated to the time displayed on the photos of the phone, and if we suppose that $T_i = t + X_i$ where t is the time at which the photo was triggered (same for both) and X_i follows a normal distribution of mean μ_i and variance ${\sigma_i}^2$. We suppose that both X_i are independent. The delay between the two photos,
$D = T_1 - T_2 = t + X_1 - t - X_2 = X_1 - X_2$, should be following a normal distribution of mean $\mu_D = \mu_1 - \mu_2$ and variance of ${\sigma_D}^2 = {\sigma_1}^2 + {\sigma_2}^2$. In our experiment, we can't measure directly the output of X_i. We measure the output of D to estimate the parameters of X_i.

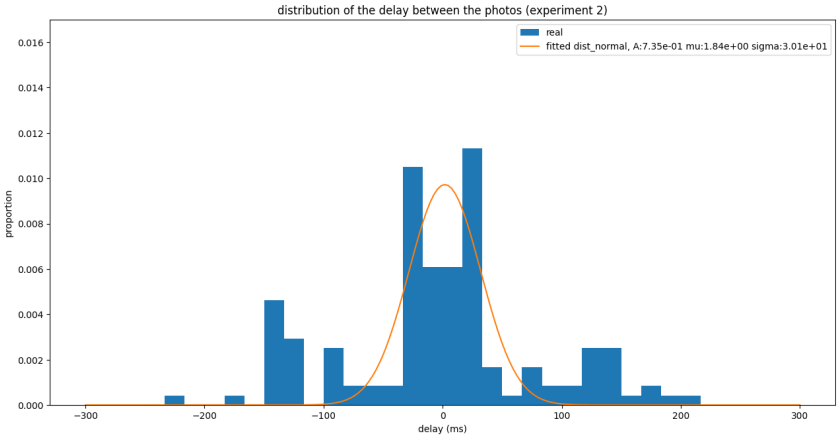
Post-processing

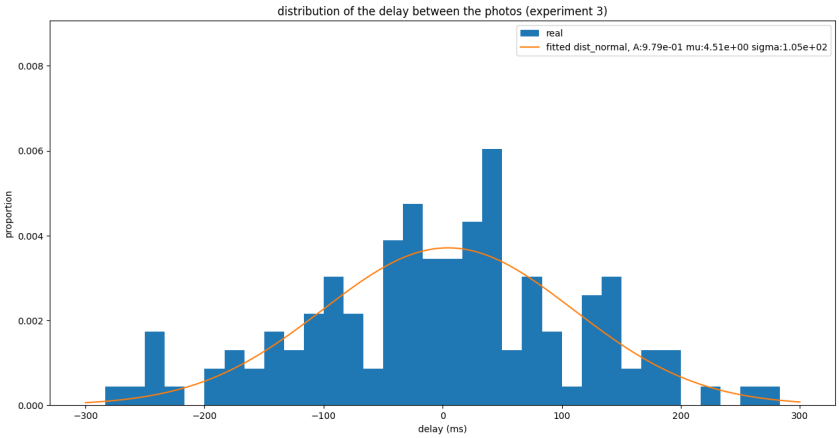
Using Optical Character Recognition (OCR) with *pytesseract* (a *Python* API for *Google*'s *Tesseract* OCR engine), we extracted the displayed time on each photo. A small percentage of the photos were not usable (ambiguous display on the screen for example). We then calculated the delay between the two photos using the time we extracted with the OCR.

We then fitted the distribution with a normal distribution ('gauss'), as we expect a normal distribution. The fitting was done using *scipy*'s *curve_fit*.

Results







Experiment N	**Number of samples**	σ_D **(from the fitting of the normal distribution)**	σ_i **(supposing** $\sigma_1 = \sigma_2$**)**	μ_D
1 (13M ISO400)	135	33.3 ms	16.7 ms	1.2 ms
2 (13M ISO1600)	144	30.1 ms	15.1 ms	1.8 ms
3 (6M ISO400)	143	105 ms	52.5 ms	4.5 ms

Analysis

General observation

- Using a normal distribution as a model of the distribution of the delay seems reasonable.
- It seems like the ISO does not impact the quality of the simultaneous trigger.

- However, the type of the photo has a significant impact, with the data from the third experiment being more than three times more spread out than the first and the second ones. The results could be described as counter-intuitive because using a two times smaller photo (6 Mega-pixels instead of 13) is not better.
- μ_D is fairly small ($< 5ms$), meaning that $\mu_1 \approx \mu_2$.
- the discrete nature of the chronometer does not significantly impact the shape of the distribution.

Application to dynamic DIC

Let's try to see if this method for coupling photos could be used for DIC, by calculating the maximum frequency the phenomenon can be:

- In a best-case scenario (experiment 2)
- We model the delay as a random variable following a normal distribution of variance ${\sigma_D}^2 = 30.1^2 ms^2$ and mean 0.
- We accept that a certain percentage of pairs of images is too delayed, and these are discarded.
- The delay between the two pictures should be of less than 10% of the period of the observed phenomenon to be deemed as "correct".

we obtain the following table for the expected maximum frequency of the phenomenon using $0.1 \times T_{max} = max\_delay$, where max_delay is the size of the confidence interval. This confidence interval is sized for a given expected percentage of correct pairs of images:

Approximate expected percent of correct pairs of images	**Corresponding Interval**	**Max delay (Interval size) in ms**	**Maximum frequency of the phenomenon in HZ**
68 %	$[-\sigma_D, \sigma_D]$	60.2	1.7
90 %	$[-1.65 \times \sigma_D, 1.65 \times \sigma_D]$	99.3	1.0
95 %	$[-2 \times \sigma_D, 2 \times \sigma_D]$	120.4	0.8
99 %	$[-2.58 \times \sigma_D, 2.58 \times \sigma_D]$	155.4	0.6

We can clearly see that the phenomenon needs to be really slow ($< 1Hz$) for this method to work correctly.

Conclusion

We cannot use this method to perform dynamic DIC as the studied phenomenon must be very slow if we want a significant proportion of usable image pairs. This is due to the fact that the delay between photos that should be simultaneous cannot be neglected.

Sources

- https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables
- https://financetrain.com/confidence-intervals-normal-distribution
- https://en.wikipedia.org/wiki/Film_speed

A Script for Visualizing and Analyzing DICe results

Principle

The tests were made on *DICe v2.0 beta 16* (https://github.com/dicengine/dice/releases/tag/v2.0). DICe stands for Digital Image Correlation Engine.

This folder contains scripts to analyse and visualize the results from a DICe analysis. The behavior is different depending on the analysis mode of DICe, that you chose on the top right corner of the software : "tracking" or "subset-based full-field"

Work flow

Each step described here has it's own *python* file in the folder. For step2, there is two choices depending of your analysis mode.

step1: Convert the video to grayscales and a csv containing the timestamps

For both analysis, the first step is to convert the video to gray scales. For that, the *python* script of this folder uses the *cv2* library. It also saves the timestamps of each frame as a CSV file (which is used for step2)

DICe analysis

Use DICe in 'tracking' or 'subset-based' mode depending on your application to obtain the results for the next step. The results are then saved in a particular folder by DICe.

step2a: Work flow for results from a DICe analysis in 'Tracking' mode

Specify your DICe results folder, the CSV file containing the timestamps, and other parameters (see the PARAMETERS paragraph in the code)

Then the program will extract the Phase, Amplitude and Frequency using NHFA of the signal and save it to a CSV results file.

step2b: Work flow for results from a DICe analysis in 'Subset-Based' mode

Specify your DICe results folder, the CSV file containing the timestamps, and other parameters (see the PARAMETERS paragraph in the code)

Then the program will extract the Phase, Amplitude and Frequency using NHFA of the signal for each subset and save it to a CSV results file.

Then the program will create a MPEG-4 video showing the results.

Parameters Description

This paragraph describes what the parameters of the steps 2a and 2b are meant to do.

Common parameters to step2a and step2b

Variable Name	Type	Description	Example
TIMESTAMPS_FILE_PATH	String	path to the file "timestamps.csv" created by step1	'examples/1/timestamps.csv'
DATA_CONTAINS_OFFSET	Boolean	Does the analyzed data contain a mean value	True
NHFA_F_MIN	Float	the lowest frequency to search for with NHFA (Hz)	10
NHFA_F_MAX	Float	the highest frequency to search for with NHFA (Hz)	50
NHFA_N_HARMONICS	Integer	the number of harmonics to search for with NHFA	1
NHFA_FREQ_SEARCH	Integer	the quantization size of the frequency spectrum with NHFA	1000
F_BUILD	Integer	the sampling frequency to build back the signal extracted from NHFA (Hz)	1000

Parameters specific to step2a

Variable Name	Type	Description	Example
DICE_RESULTS_FILE_PATH	String	The path of the result file of DICe from a 'tracking' analysis run	'examples/1/results_tracking/DICe_solution_0.txt'
RESULTS_FILE	String	Where does the NHFA results needs to be saved (CSV file)	'examples/1_Tracking/NHFA.csv'
FIELD_TO_ANALYZE	String	The name of the field produced by DICe to analyze.	DISPLACEMENT_Y

Parameters specific to step2b

Variable Name	Type	Description	Example
DO_NHFA	Boolean	Should the NHFA be done on all the subsets	True
DICE_RESULTS_FOLDER_PATH	String	The path of the result folder of DICe from a 'subset' analysis run	'examples/1/results_subsets/'
RESULTS_FOLDER	String	Where does the NHFA results needs to be saved (CSV file)	'examples/1_Subsets/NHFA_results'
N_DIGITS	Integer	number of digits of the DICe results saving number	3
RENDER_VIDEO_FILE	Boolean	Should a video of the results be rendered	True
DPI	Integer	Dots Per Inch (resolution) of the video	600
FPS	Integer	Frames Per Second of the video	4
VIDEO_METADATA	Dict	Metadata of the video (explorer display)	dict(title='DICe results')
VIDEO_FILE_NAME	String	Where the rendered video should be saved	DICe_results_video" (will render a video named "DICe_results_video.mp4")

Examples

A folder named 'examples' contains two examples for the use of these scripts and results : In reality, the beam vibrates at around 29 Hz for example 1 and 27 Hz for example 2. Both videos have a framerate that do not respect Shannon's criterion to analyze these beams vibration frequency, thus justifying the use of a NHFA algorithm.

For both videos, we have (video X.mp4):

- 'X/': this folder contains all of the frames extracted from the MPEG-4 video named 'X.mp4', as well as the parameters, results and workspace from DICe.
- 'X_Tracking/': contains the results and plots from a run of 'step2a', on the DICe results file from a tracking run that can be found at 'X/results_tracking/DICe_solution_0.txt'

- 'X_Subsets/': contains the results and plots from a run of 'step2b', on the DICe results folder from a subset run that can be found at 'X/results_subsets/results_subsets/'

Using SFTP to transfer files between a computer and a phone

Reasons to use SFTP

SFTP (SSH File Transfer Protocol) should be more universally compatible and reliable than bluetooth file transfer, which is one of the most common alternatives to file transfer. SFTP is wireless, which is useful as a lot of phones' USB ports are broken. Other alternatives include sending mails or cloud links, but this means sending the files to an outside server, and downloading them: this creates useless copies of the file and is longer.

How to use SFTP

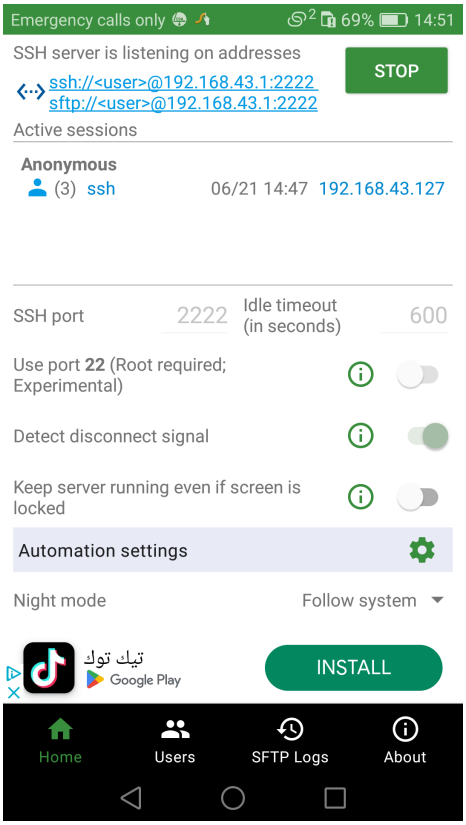
On a computer, you can use *FileZilla*, which is free (https://filezilla-project.org/)

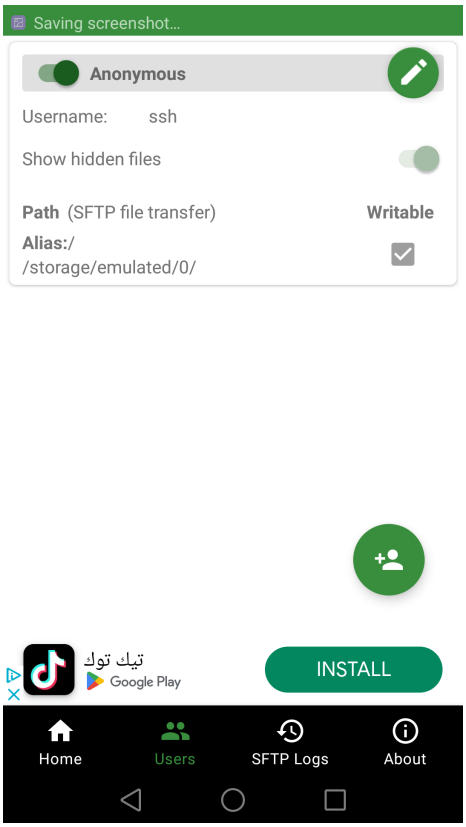
On a phone, you can use *SSH/SFTP Server - Terminal* (https://play.google.com/store/apps/details?id=net.xnano.android.sshserver&hl=en_US)

The computer and the phone should be connected to the same Wi-Fi (or ethernet) network. You can use the 'Wi-Fi Hotspot' mode of your phone, this will be faster.

Phone-side

These parameters should be fine :



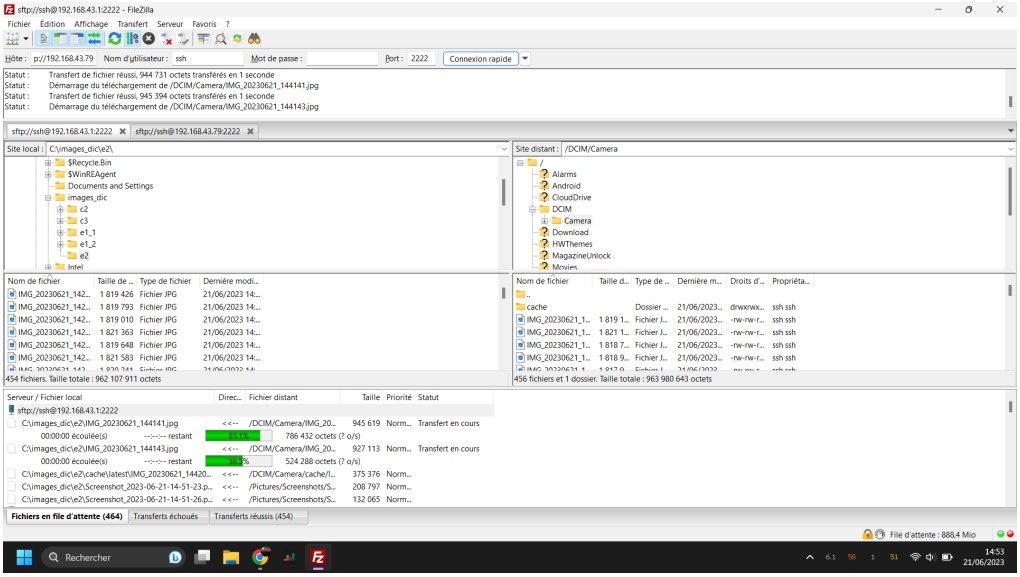


Computer-side

With the parameters described beforehand, you should connect *FileZilla* to the SFTP server of your phone. The parameters are as follows:

- Host ('Hôte'): for example `sftp://192.168.xxx.xxx`, see the top of your the screen
- User Name ('Nom d'utilisateur'): ssh
- Password ('Mot de passe'): *leave empty*
- Port ('Port'): 2222

Your computer will then display the folders inside your phone (and inside your computer). You can then download and manage files on your phone. You can connect to two phones at once, if your PC and both phones are connected to the same network.

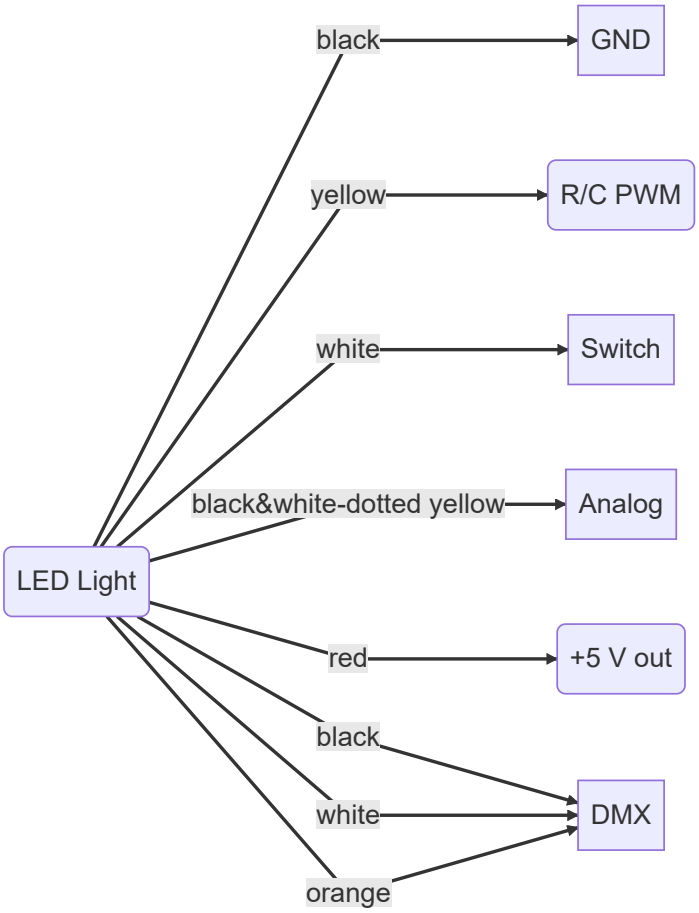

sftp://ssh@192.168.43.1:2222 - FileZilla
Fichier Édition Affichage Transfert Serveur Favoris ?
Hôte : p://192.168.43.79 Nom d'utilisateur : ssh Mot de passe : Port : 2222 Connexion rapide
Statut : Transfert de fichier réussi, 944 731 octets transférés en 1 seconde
Statut : Démarrage du téléchargement de /DCIM/Camera/IMG_20230621_144141.jpg
Statut : Transfert de fichier réussi, 945 394 octets transférés en 1 seconde
Statut : Démarrage du téléchargement de /DCIM/Camera/IMG_20230621_144143.jpg
Site local : C:\images_dic\e2\
Site distant : /DCIM/Camera
454 fichiers. Taille totale : 962 107 911 octets
456 fichiers et 1 dossier. Taille totale : 963 980 643 octets
Fichiers en file d'attente (464)
Transferts échoués
Transferts réussis (454)
File d'attente : 888,4 Mio
Rechercher

External Control of the Stratus Lights 200 W LED Module

Tristan Chevreau, tristan.chevreau@gmail.com, summer 2023

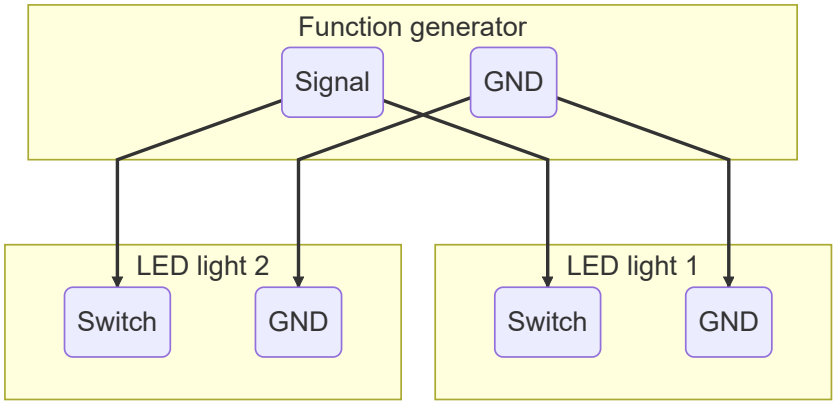
Meaning of the wire colors

The following diagram explains, for both lights, what does the color of the wire mean (when available)

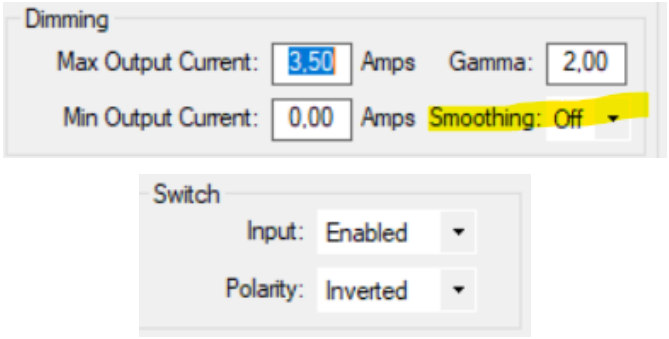


*NB : The +5 V out is supposed to be used to power an Arduino for example. It is **not** an input for power. It has nothing to do with the +5 V of the USB port.*

Use as a stroboscope



The light can be controlled with the "Switch" port, using TTL values (0, +5 V). It is set to be "ON" when the value is 0 V and "OFF" when the value is +5. This means, with no input, the LED light is ON (see configurator in "inverted" position). To have the fastest strobo possible, you need to configure 'smoothing' to 'Off'.



Between Switch and GND, a 50 Ohms impedance is good enough. This value is typical of all-purpose function generators. But if there is an "high impedance" mode on the signal generator, it is probably better.

The LED light and the function generator should be electrically isolated, or the grounds should have the exact same potential to avoid having current flowing through the devices (earth loop). This can be measured with a multimeter. If the difference between voltage potentials of the ground of the LED light and the ground of the function generator is greater than 0.5 V, there is probably a problem in the electrical installation and/or the grounding of the electrical devices. Using this method may destroy the LED controller card.

The relation of the pulse width in relation to the switch input is described in another document.

Analysis of the Pulse Length of the Stratus LED 200 W Module in Stroboscope Mode with Respect to the Input Signal Characteristics

Tristan Chevreau, tristan.chevreau@gmail.com, summer 2023

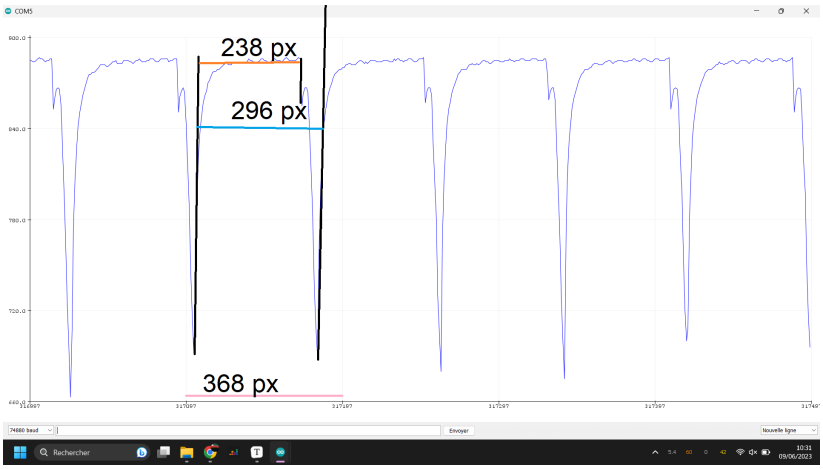
Experimental conditions

- Function generator: *Iso-Tech AFG 21 005* in pulse mode (0, 5 V : TTL) with varying duty cycles at 8, 10 and 15 Hz.
- *Stratus* LED 200 W module is controlled with the Switch pin. The default state (switch = 0 V) is ON, and the HIGH state (switch = 5 V) is OFF. Smoothing is set to zero in the configurator. All other parameters are the default ones (for the 200 W module).
- *Arduino* analog reading of a photo-resistor, using the Serial Tracer. Reading with a 1 ms delay at a 74880 baud-rate.
- The photo-resistor manufacturer reference is unknown. It is read using a voltage divider (photo-resistor, 240 Ohms resistor).

Error factors and assumptions

- The photo-resistor probably has a rise time (dark to light) of the order of 5 ms and a fall time (light to dark) of the order of 100 ms. These are typical characteristics for a photo-resistor. But on the serial tracer, we can see that the signal 'falls' in two steps. We'll assume the first step is the real falling date, and the second one is just the fall time of the photo-resistor. We neglect the rise time.
- We measure only one period for each input duty cycle because I am lazy and this is just a quick test.
- The width of the line done with paint may be a bit too wide.
- The Serial Tracer horizontal value is not in milliseconds; we'll call them time units, and we'll assume it is consistent throughout the experiments. We cannot use pixel measures as there is an auto-scale. This is not really a problem since we are using percentages of cycles.

Method



- the image above is the serial tracer of the analog input read (voltage) through time units
- Using *Microsoft Paint* as the measuring tool
- The values in pixels on the image represent only the horizontal pixel length (not the segment length)
- Colors of the lines
  - Orange: flash length
  - Blue: period
  - Pink: used for time units/pixel

Results

Note : the '=' sign in front of a value means it is calculated from other values rather than measured. % Duty is the value on the display of the signal generator.

Results (10 Hz)

% Duty (Input Signal)	Time Unit/Pixel	= Time Unit/Pixel	Flash Length (Pixels)	= Flash Length (Time Unit)	Period (Pixels)	= Period (Time Units)	= % Flash (Real)
5	100 TU / 368 px	0.272	161	43.79	293	79.70	55
15	125 TU / 560 px	0.223	189	42.15	292	65.12	64
25	83 TU / 307 px	0.270	209	56.43	287	77.49	73
30	100 TU / 368 px	0.272	219	59.57	296	79.42	74
35	100 TU / 368 px	0.272	238	64.74	296	80.51	80

Results (15 Hz)

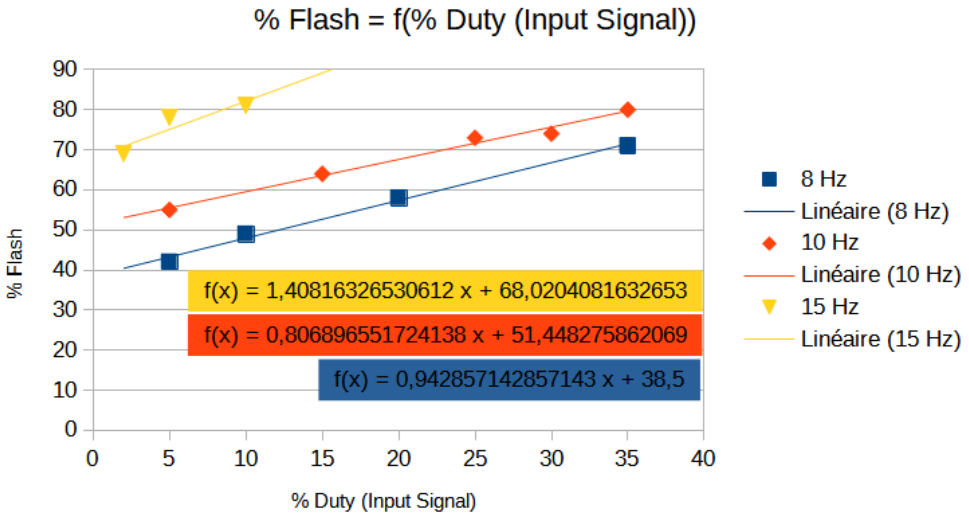
% Duty (Input Signal)	Time Unit/Pixel	= Time Unit/Pixel	Flash Length (Pixels)	= Flash Length (Time Unit)	Period (Pixels)	= Period (Time Units)	= % Flash (Real)
2	100 TU / 370 px	0,270	137	36.99	198	53.46	69
5	100 TU / 370 px	0,270	155	41.85	199	53.73	78
10	100 TU / 370 px	0,270	157	42.39	195	52.65	81

Results (8 Hz)

% Duty (Input Signal)	Time Unit/Pixel	= Time Unit/Pixel	Flash Length (Pixels)	= Flash Length (Time Unit)	Period (Pixels)	= Period (Time Units)	= % Flash (Real)
5	100 TU / 370 px	0,270	155	41.85	365	98.55	42
10	100 TU / 370 px	0,270	177	47.79	363	98.01	49
20	100 TU / 370 px	0,270	213	57.51	367	99.09	58
35	125 TU / 460 px	0,272	264	71.808	371	100.912	71

Fitting

'Linéaire' means Linear in French.



Analysis

- For 10 Hz, the period is approximately 80 time units. Given that we are operating at 10 Hz, we can approximate 1 ms as 0.8 time units. This seems coherent:
  - for 8 Hz, we have a period of 100 time units, which represents 125 ms (expected 125 ms)
  - for 15 Hz, 52 time units, 62.5 ms (expected 66 ms)
- The flash cannot be "ON" less than 40 % of the time, and this value is often higher than 50 %, which could pose a challenge for creating a good strobe effect at the correct strobe frequency.
- A more or less linear relationship can be observed between the input duty cycle value and the output flash duty value for each frequency. However, for the 15 Hz series, it seems like there is a threshold effect and the linear fitting is not very satisfying.
- Outside of the bounds of the series, the behavior of the LED module is often chaotic.
- It seems like lesser strobing frequency values allow for a smaller proportion of "ON" time. It does not mean that the flashes are shorter, as the period increases when the frequency lowers.

Conclusion

For the 200 W stratus LED module used in stroboscope mode using the 'switch' input, the duty cycle of the input signal from the function generator is related to, but not equivalent to, the duty cycle of the flash for a pulse function. The strobing frequency is also a variable of the flash duty cycle.

The irregularities and weird behavior of the light for the 15 Hz input frequency is probably due to an internal refresh rate or clock of the LED controller. In this case, the behavior observed at 8 and 10 Hz should probably be the same for lower frequencies, which means the duty cycle of the input signal and the duty cycle of the flash follow a linear relation (but with different values for the slope and initial offset).

Using the multitude of approximations mentioned earlier and the fittings, we can obtain the following table for using this stroboscope mode at 8 and 10 Hz (for other frequency values, it is probably better to measure it once again):

% Duty (Input Signal)	Approx. flash length (ms) at 8 Hz	Approx. flash length (ms) at 10 Hz
5	55,5	54,0
10	59,5	59,9
15	63,6	65,8
20	67,6	71,7
25	71,6	77,6
30	75,7	83,5
35	79,7	89,4