

**Raphaël GUEGAN<sup>1</sup>**

<sup>1</sup> FGE4, EPF Cachan, Cachan, 94230, France

Emails: [raphael.guegan@epfedu.fr](mailto:raphael.guegan@epfedu.fr) or [raphaelguegan.pro@gmail.com](mailto:raphaelguegan.pro@gmail.com)

Date: December 2024

Link to the RaspiCam folder:

<https://drive.google.com/drive/folders/1jNGNBqXZzGSTqxBBkZALONDus-nEtzvN?usp=sharing>

RaspiCam for MultiDIC

## Capturing Synchronized Stereo Images

# TABLE OF CONTENTS

<b>LIST OF FIGURES .....</b>	<b>3</b>
<b>LIST OF TABLES .....</b>	<b>4</b>
<b>1. ABSTRACT .....</b>	<b>5</b>
<b>2. INTRODUCTION .....</b>	<b>6</b>
2.1. State of the Art.....	6
2.2. Aims .....	6
2.3. Plan.....	7
<b>3. MATERIALS AND METHODS .....</b>	<b>8</b>
3.1. Hardware.....	8
3.1.1. List of components .....	8
3.1.2. Full system .....	11
3.2. Software .....	11
3.2.1. Raspberry Pi setup .....	11
3.2.2. Python scripts.....	14
3.2.3. Upload images on a computer .....	18
3.2.4. Instructions for use.....	18
3.2.5. Choice of solutions.....	20
3.3. Methods .....	26
3.4. Results .....	27
3.4.1. Synchronization Accuracy (Photo capture scripts).....	27
3.4.2. Method comparison (Test scripts) .....	29
3.4.3. Direct Timing Measurement (Test scripts).....	30
3.4.4. Impact of Prolonged System Use.....	32
3.4.5. Frame Rate Considerations.....	33
3.5. Discussion .....	35
<b>4. CONCLUSION .....</b>	<b>38</b>
<b>BIBLIOGRAPHY .....</b>	<b>40</b>

## LIST OF FIGURES

Figure 1: System deployment diagram.....	11
Figure 2: Sequence diagram of capture programs .....	15
Figure 3: Activity diagram of speed test programs .....	17
Figure 4: Graphic of the time difference between clients and the server over time (one recording every 16s). Blue line represents Camera 1. Red line represents Camera 2. Green line represents Camera 3.....	23
Figure 5: Picture with adjusted parameters .....	25
Figure 6: Time difference between cameras with different resolutions (time exposure: 10000 $\mu$ s). (a) Camera 1 and 2. (b) Camera 1 and 3. (c) Camera 2 and 3. Blue line represents Resolution: 4056x3040. Red line represents Resolution: 2028x3040. Green line represents Resolution: 1080x3040. Purple line represents Resolution:540x3040. ....	28
Figure 7: Time difference between cameras with different time exposure (resolution: 4056x3040). (a) Camera 1 and 2. (b) Camera 1 and 3. (c) Camera 2 and 3. Blue line represents Time exposure 10000 $\mu$ s. Red line represents Time exposure 5000 $\mu$ s. Green line represents Time exposure 1000 $\mu$ s. Purple line represents Time exposure 500 $\mu$ s. ....	29
Figure 8: Time difference between both methods (Resolution: 4056x3040, Time exposure: 10000 $\mu$ s). Blue line represents Camera 1. Red line represents Camera 2. Green line represents Camera 3. ....	30
Figure 9: Time difference between cameras with different resolutions (time exposure: 10000 $\mu$ s). (a) Camera 1 and 2. (b) Camera 1 and 3. (c) Camera 2 and 3. Blue line represents Resolution: 4056x3040. Red line represents Resolution: 2028x3040. Green line represents Resolution: 1080x3040. Purple line represents Resolution:540x3040. ....	31
Figure 10: Time difference between cameras with different time exposure (resolution: 4056x3040). (a) Camera 1 and 2. (b) Camera 1 and 3. (c) Camera 2 and 3. Blue line represents Time exposure 10000 $\mu$ s. Red line represents Time exposure 5000 $\mu$ s. Green line represents Time exposure 500 $\mu$ s.....	32
Figure 11: Relative error trend curve with different resolutions (time exposure: 10000 $\mu$ s). (a) Camera 1. (b) Camera 3. (c) Camera 3. Blue line represents Resolution: 4056x3040. Red line	

represents Resolution: 2028x3040. Green line represents Resolution: 1080x3040. Purple line represents Resolution:540x3040.....33

Figure 12: Photo capture time as a function of the number of pixels in the image height (Time exposure: 10000 $\mu$ s, Pixels in the image width: 4056). Blue line represents Camera 1. Red line represents Camera 2. Green line represents Camera 3. ....34

Figure 13: Photo capture time as a function of the number of pixels in the image width. (Time exposure: 10000 $\mu$ s, Pixels in the image height: 3040). Blue line represents Camera 1. Red line represents Camera 2. Green line represents Camera 3. ....35

Figure 14: Photo capture time as a function of the time exposure. (Resolution: 4056x3040). Blue line represents Camera 1. Red line represents Camera 2. Green line represents Camera 3. ....35

Figure 15: System picture .....39

## LIST OF TABLES

Tableau 1. List of components.....8

Table 2: User Guide .....18

## 1. ABSTRACT

This project addresses the challenges of creating a synchronized, high-resolution image capture system compatible with MultiDIC, a toolbox for multi-view 3D digital image correlation. Previous systems developed by Aaron Jaeger, a Senior Research Support Associate at the MIT Media Lab, were based on software and hardware components that have now become obsolete, making it difficult to implement and use the system today. Therefore, the goal was to modernize the system, enhancing both accuracy and ease of use while maintaining synchronization similar to that of the previous system.

To achieve this, the Raspberry Pi Zero W and Camera V2 modules were replaced with Raspberry Pi 4 and High-Quality Camera modules, resulting in improved image quality. Using the Picamera2 library, Python scripts were developed to capture images at a resolution of 4056x3040, with a synchronization accuracy of  $\pm 30$  ms between cameras. An NTP-based clock synchronization system was implemented to ensure this accuracy, allowing simultaneous image capture by multiple cameras.

Extensive tests were conducted to analyze the stability of capture time, resolution, and the effects of exposure time on synchronization accuracy. These tests confirmed that the updated system met the project's accuracy requirements, provided that the chosen resolution and exposure time parameters allowed capture time to be aligned on a step value. This alignment helps avoid oscillations that can occur with intermediate capture times between two steps levels.

This modernized capture system offers a user-friendly, scalable solution that meets the requirements established at the project's outset and enables the acquisition of sample images for the various tests necessary to analyse a system using the MultiDIC software.

**Keywords:** DIC, Raspberry Pi, MultiDIC, Synchronized Imaging, Time Exposure, Displacement and Strain measurement, Python

## 2. INTRODUCTION

---

### 2.1.STATE OF THE ART

In 2018, Aaron Jaeger, a Senior Research Support Associate at MIT Media Lab, created a Python script specifically designed to work with MultiDIC, an open-source toolbox for multi-view 3D Digital Image Correlation [1] [2]. The system includes a desktop computer, Raspberry Pi Zero-W equipped with Raspberry Pi Camera Module V2 (3280x2464 pixels resolution [3]), and an internet router, providing a cost-effective and scalable solution for multi-camera synchronization. There are two main programs: `raspiCam.py` and `ScannerMaster.py`. The `raspiCam.py` script manages the Raspberry Pi units, handling device IP discovery, camera configuration, status updates, image capture, local image storage, and communication of results with the master. The `ScannerMaster.py` script coordinates user inputs, monitors camera status updates, issues camera commands, and provides real-time status feedback to users. To enable concurrent image capture, setting up an NTP local time server on a separate Raspberry Pi to synchronize system clocks is crucial. Adequate lighting is essential to minimize exposure time, enabling cameras to capture synchronized images with a tolerance of  $\pm 30$  ms. This synchronization can be achieved by simultaneously capturing images of an LED array controlled by an Arduino [4], ensuring precise alignment across all devices. This system has facilitated the development of low-cost solutions for wearable technologies that interface with the skin, offering a flexible and scalable platform for capturing high-resolution images. However, compatibility issues caused by outdated software libraries and dependencies further complicate the installation process. It's important to note that the system is currently not being actively maintained, and certain libraries, such as "Picamera" (now replaced by Picamera2), are no longer supported, limiting the functionality of the original scripts.

---

### 2.2.AIMS

Considering the challenges posed by the current system, it is imperative to create an updated solution that overcomes the previously identified limitations. While functional, the existing Python scripts depend on outdated libraries like "Picamera" that are no longer supported and

are linked to hardware that would greatly benefit from substantial upgrades to enhance performance and user-friendliness. This project is focused on modernizing the system by leveraging newer libraries such as Libcamera or Picamera2, which provide better support for the latest Raspberry Pi models and offer more advanced control over camera settings. The aim is to maintain a similar synchronization accuracy of  $\pm 30$  ms by improving the power and resolution of the system by replacing the old components (Raspberry Pi Zero-W and V2 camera) with the Raspberry Pi 4 and the Raspberry Pi High-Quality camera to enable better image quality and therefore greater accuracy. Another essential goal is to streamline the overall setup process by minimizing the requirement for manual terminal commands and administrative privileges, thereby making the system more accessible to users with varying levels of technical expertise. This could be accomplished by developing an automated script or interface that manages the initialization and synchronization of the various components, allowing for a smoother and quicker deployment. Additionally, comprehensive documentation and user support will be created to facilitate the system's installation and usage, ensuring that researchers and professionals can benefit from the improved solution with minimal difficulty. It will also be necessary to implement several similar codes to obtain images with the correct naming convention (Stereo calibration images, Checkerboard images, Speckle images [5]) to be processed directly with MultiDIC.

---

### 2.3.PLAN




It is essential to understand the necessary hardware components for the system's functionality to develop this upgraded system. Subsequently, comprehensively assimilating all required software for the system must be accompanied by thorough documentation of the decision-making process underlying its development. The upcoming explanation will elaborate on the methodologies for evaluating the system's achievement of its objectives, concluding with a discussion of the obtained results and the system's operational status.

## 3. MATERIALS AND METHODS





### 3.1. HARDWARE

#### 3.1.1. LIST OF COMPONENTS

TABLEAU 1. LIST OF COMPONENTS

Name	Image	Numbers	Use for
Raspberry Pi 4		x12 4 Go (Clients)  x1 8Go (Serveur)	12 Raspberry Pi 4 (4Go) units are used as clients to capture pictures.  1 Raspberry Pi 4 (8Go) is a server that sends instructions to the clients, uploads the images, and is the NTP server.
Raspberry Pi HQ Camera		12	Capture images
Raspberry Pi HQ Camera Lens		12	Capture images



<b>Aluminium Case for Raspberry Pi 4</b>		13	Protect the Raspberry Pi units
<b>Ethernet Cable</b>		14	<p>13 ethernet cables communicate between all Raspberry Pi units via a switch.</p> <p>One Ethernet cable is used to transmit images from the Raspberry Pi server to a computer, where the user can directly use MultiDic.</p>
<b>USB-C Power Supply</b>		13	To power the Raspberry Pi units
<b>PoE Switch</b>		1	Enables Ethernet networking to connect all cameras.
<b>Micro HDMI to HDMI Cable</b>		1	<p>Enable to connect Raspberry Pi units to a screen. It will be useful to set up each Raspberry Pi for the first time and to use the Raspberry Pi server as the main computer.</p>



---

**Computer  
Screen**



1

Enable screen feedback from the connected Raspberry Pi.

---

**Keyboard**



1

Enable to use the connected Raspberry Pi as a computer.

---

**Mouse**



1

Enable to use the connected Raspberry Pi as a computer.

---

**USB Key**



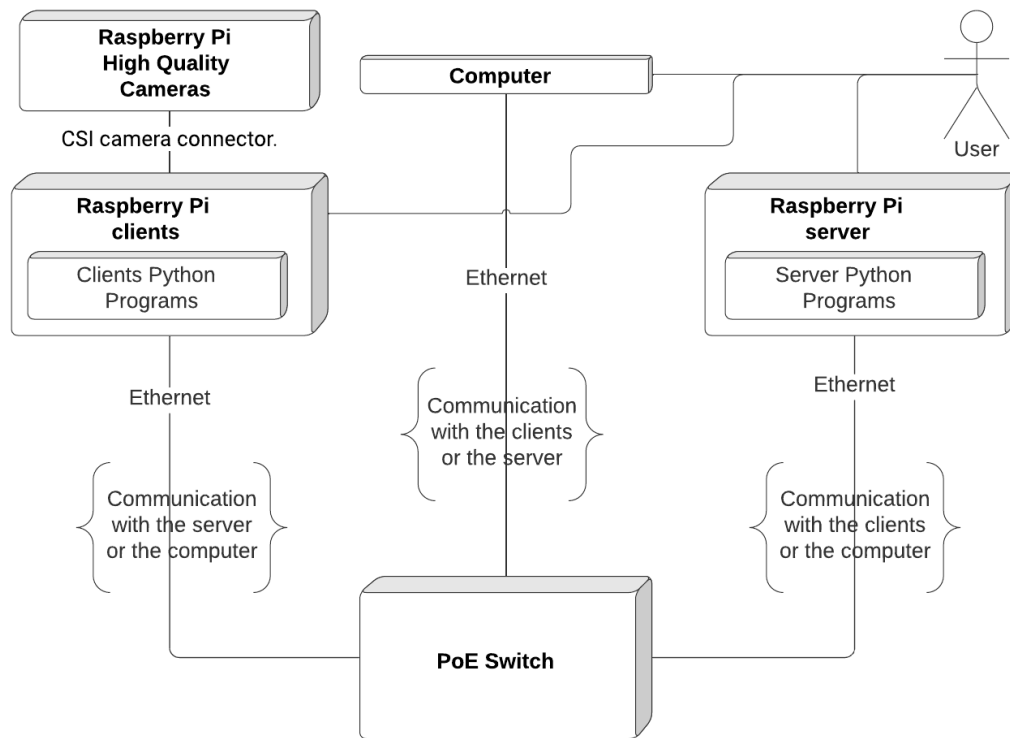
1

It will be used to upload the client or the server folder on Raspberry where all programs needed will be there.

---

### 3.1.2. FULL SYSTEM

All communications between the computer and Raspberry Pi are via Ethernet cables, all connected to a PoE switch. Figure 1 illustrates the various connections, which will be detailed throughout the project.



**FIGURE 1: SYSTEM DEPLOYMENT DIAGRAM**

---

## 3.2.SOFTWARE

---

### 3.2.1. RASPBERRY PI SETUP

---

#### 3.2.1.1. WRITE IMAGES OF THE SD CARDS

The simplest way to set up a Raspberry Pi is by transferring the image from an existing SD card to a newly formatted one. Locate the image file in the "CaptuRPI" folder that will be copied to the new SD card. An application such as Win32 Disk Imager needs to be used for this process. Open the application, select the SD card, and choose the image file that will be copied. Then, wait for the process to finish. Ensure that the new SD card is at least the same size as the

original. If this method proves unsuccessful, follow the alternative steps "3.2.1.2.", "3.2.1.3.", "3.2.1.4." and "3.2.1.5." which will take a bit more time but will yield the same results. The server will then need to be connected to the wifi network. To do this, follow the same instruction as in part 3.2.1.3.

#### 3.2.1.2. INSTALL THE OS ON THE SD CARD

---

To set up a Raspberry Pi, obtaining an operating system (OS) is needed, and using the Raspberry Pi Imager application to upload it onto an SD card. Once the SD card is inserted into the computer, it must be formatted before opening the Raspberry Pi Imager. Next, select the Raspberry Pi 4 model and choose the recommended Raspberry Pi OS as the operating system. Designate the SD card as the storage medium and initiate the writing process. A window will open with the option to change the settings. Click on it. A settings modification window will appear, allowing to assign unique hostnames (e.g., sim1 for the first one, sim2 for the second one, etc.), usernames (e.g., admin1, admin2, etc.), and passwords (e.g., Admin1, Admin2, etc.) for each Raspberry Pi client. For the Raspberry Pi server, it is advisable to use "sim" as the hostname, "admin" as the username, and "Admin" as the password. Configuring the Wi-Fi settings to connect to the preferred network is essential for acquiring various libraries. Additionally, configure the time zone according to the specific location. It is important to check that the SSH password connection is enabled in the services section. After saving the settings and confirming by clicking "Yes", wait for the process to complete before inserting the SD card into the Raspberry Pi.

#### 3.2.1.3. UPLOAD FOLDER FROM COMPUTER TO RASPBERRY PI

---

When the Raspberry Pi is connected to the display and other peripheral devices, verifying its Wi-Fi connectivity status is advisable. If the connection is not established, manual configuration may be necessary (It is possible to run `sudo raspi-config` in the terminal and select System Options then Wireless LAN). A wifi connection is required to download the various libraries and for the local NTP server. Upon successful connectivity, the next step involves inserting a USB storage device containing the "Client" and "Server" directories and subsequently transferring the requisite folder to the "Documents" directory on the Raspberry

Pi. The “Client” and “Server” folders are made up of a multitude of programs, which will be described in greater detail later.

#### 3.2.1.4. LIBRARIES

---

Installing the various required libraries is important for the server and clients to function correctly throughout this project. The Python scripts `Libraries_server.py` and `Libraries_client.py` are designed to automate these installations.

The `Libraries_server.py` script configures the server and installs essential tools such as NTP for time synchronization and Python libraries such as paramiko and SCP for file transfers and SSH connections.

The `Libraries_client.py` script configures the client by installing similar tools but with additional components such as `NTPdate` and `cpufrequtils`, needed to adjust the system clock and optimize performance.

#### 3.2.1.5. SETUP IP AND NTP

---

Two points are essential to this project: communication between the Raspberry Pi units and synchronization between them. To simplify communication, IP addresses need to be fixed. For synchronization, the NTP protocol had been used to set up the Raspberry server as a reference clock.

Three programs have been developed for this purpose, 2 for the clients (`Set_static_ip.py` and `Setup_ntp_client.py`) and one for the server (`Setup_server.py`). `Set_static_ip.py` formats each Raspberry's IP address in the format “192.168.1.x”, the “x” representing the number at the end of the hostname. For example, `sim1`'s IP address will be “192.168.1.1”. This program is also included in the first part of `Setup_server.py` to set the server IP as “192.168.1.253”. The second part of `Setup_server.py` tells the server to act as a local NTP server to enable client synchronization. So that the client can synchronize with the server's clock, the `Setup_ntp_client.py` program tells the client the IP of the server it needs to synchronize with. Before starting the client programs, the Raspberry server must already be set up. In addition,

the client programs had to be split in two, as it is necessary to reboot the Raspberry to set up the IP, and without an IP, communication with the server is not possible.

---

### 3.2.2. PYTHON SCRIPTS

#### 3.2.2.1. CAMERA SETUP

---

Camera\_setup.py Python program creates a live video stream from a Raspberry Pi camera and serves it over HTTP using the MJPEG format to enable the user to adjust the camera aperture and focus [6]. It captures frames from the camera, encodes them as JPEG images, and continuously streams them to a web page accessible from devices on the same network. The program sets up an HTTP server that can handle multiple clients at once. It provides a simple HTML page to display the live stream. The stream continues until the script is interrupted, ensuring efficient, real-time video transmission. To access the web page, the user needs to write “http://192.168.1.x:8000/” in a browser of a device connected via ethernet to the Raspberry client (the Raspberry server or a computer) with x depending on the number of the Raspberry Pi.

#### 3.2.2.2. NTP TEST

---

Before using any capture program, it may be necessary to check that the NTP synchronization between client and server is working correctly. The Csvntp.py program meets this requirement, checking the offset between client and server every 16 seconds. It then saves these values in a CSV file, which the user can later retrieve to verify correct synchronization.

#### 3.2.2.3. PHOTO CAPTURES

---

This project is designed to capture images simultaneously from 12 cameras, which will be analyzed by MultiDIC software. The software is organized into five components: 0. distortion calculation, 1. stereo calibration, 2. 2D-DIC, 3. 3D reconstruction, and 4. post-processing. To successfully carry out steps 3 and 4, it is essential to complete the first three steps first. Step 0 is only necessary if the user opts for distortion correction [5].

To simplify this process, three client programs (Checkerboard\_client.py, Speckle\_client.py, and Stereo\_client.py) and three server programs (Checkerboard\_server.py,

Speckle\_server.py, and Stereo\_server.py) have been developed. Each pair of programs enables simultaneous image capture to be adapted to the test being carried out.

Figure 2 gives an overview of how these three pairs of programs work. The main distinctions between them lie in the folder and image naming conventions, which are intended to facilitate differentiation. The only significant difference between these programs is with the Stereo program pair, distinguished by the fact that it requires only one image per camera, thus eliminating the need for a multiple-image capture loop.

Using these programs is straightforward: launch the server programs, which will start the respective client programs.

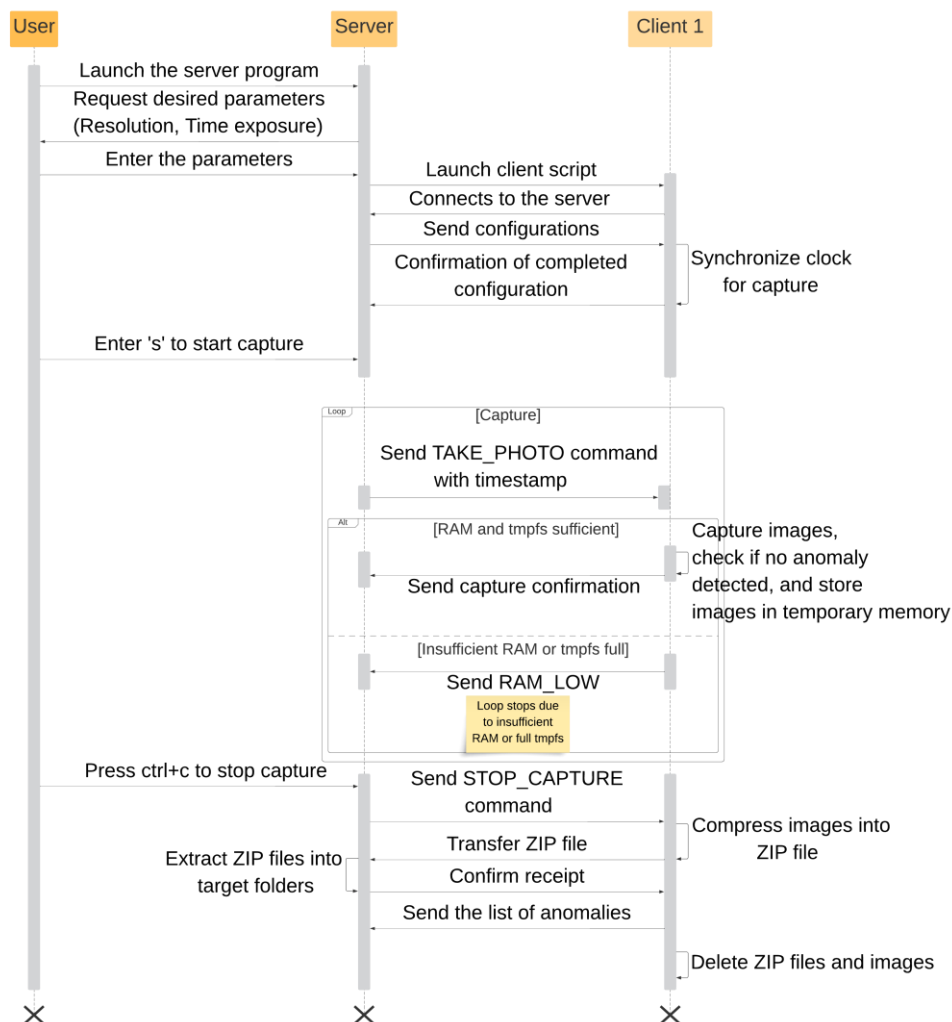


FIGURE 2: SEQUENCE DIAGRAM OF CAPTURE PROGRAMS

#### 3.2.2.4. FUNCTION TEST

---

In addition to checking that the NTP synchronization is working properly, as described above, it may be necessary to test all the cameras. It's possible that, at some point, the use of certain raspberries could lead to instability in the capture time, resulting in a time lag between cameras. Hence the need for the `Test_client.py` and `Test_server.py` program pairs, which will operate in a very similar way to the various image capture programs seen above. The difference between these test programs and capture programs is that they will record the capture times of each image in a CSV file. A relative error calculation will also be performed and saved in the CSV file, using as reference value the capture time of the 3rd photo, which is the first value converging towards the capture time constant associated with the parameter used (Resolution, Exposure time). At the end of the recording, all client CSV files will be sent to the server, which will merge them and, from these values, plot a capture time curve and a relative error curve.

However, unlike the previous capture program, it will be necessary for the user to start the client programs on the various Raspberry Pi units.

#### 3.2.2.5. SPEED TEST

---

During experimentation, it may be useful to know the speed of image capture to know the number of images per second, for example, to estimate the number of images over a given time range. To achieve this, 4 different programs have been written, all following the same logic as the Figure 3: `TimeCapture.py` (variation of resolution height and width), `TimeCaptureHeights.py` (variation of resolution height), `TimeCaptureWidths.py` (variation of resolution height) and `TimeCaptureTimeExposure.py` (variation of exposure time). In this way, a CSV file is obtained with the capture time as a function of the chosen parameter.



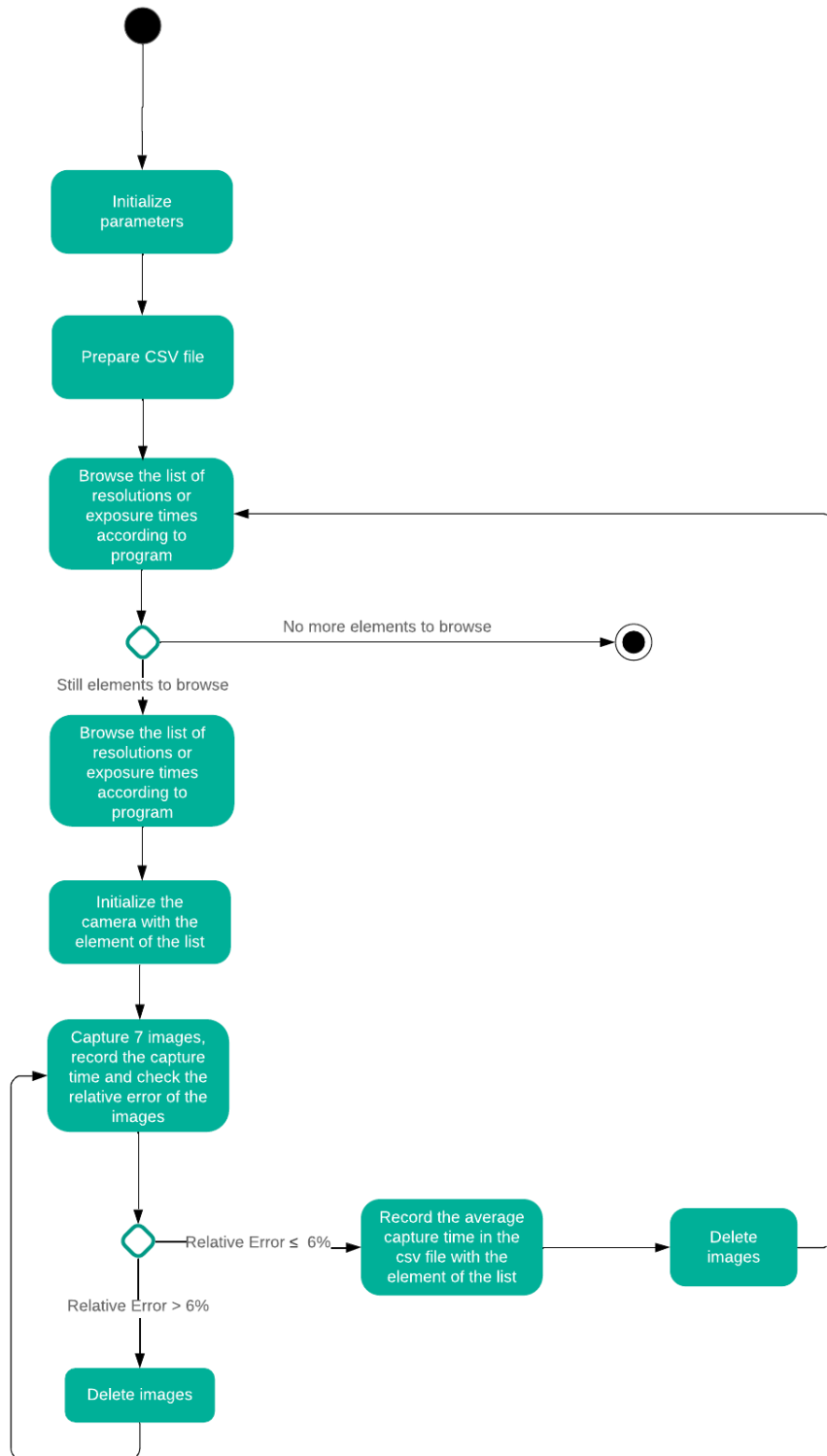


FIGURE 3: ACTIVITY DIAGRAM OF SPEED TEST PROGRAMS

---

### 3.2.3. UPLOAD IMAGES ON A COMPUTER

After completing the image capture process on the Raspberry Pi, two options for transferring the images to another computer exist. The first method involves using a USB key to copy the images from the Raspberry Pi and then transferring them to the target computer. The second method entails using the FileZilla app. To do this, it will need to launch the FileZilla app and enter the IP address of the server (192.168.1.253), the username (admin), the password (Admin), and the port (22). Once connected, users will have direct access to transfer the necessary files and folders from the Raspberry Pi to their computer.

---

### 3.2.4. INSTRUCTIONS FOR USE

*TABLE 2: USER GUIDE*

	Goal	Where to run it	How to run it	How to stop it	Action required
1.	Write images of the sd cards	App as Win32 Disk Imager	Follow 3.2.1.1	Stops automatically	Download the CaptuRPI folder
2.	Install the OS on the SD card	App as Raspberry Pi Imager	Follow 3.2.1.2	Stops automatically	
3.	Upload folder from computer to Raspberry Pi	On Raspberry wanted	Follow 3.2.1.3	Stops automatically	2.
4.	Upload server libraries	On the Raspberry server terminal	cd /home/admin/Documents/Server #Need to be in the right folder sudo python3 Libraries_server.py	Stops automatically	3.
5.	Upload client libraries	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi sudo python3 Libraries_client.py	Stops automatically	3.

6.	Server ip and ntp configuration	On the Raspberry server terminal	cd /home/admin/Documents/Server #Need to be in the right folder sudo python3 Setup_server.py	Stops automatically	4.
7.	Client ip configuration	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi sudo python3 Set_static_ip.py	Stops automatically	5.
8.	Client ntp configuration	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi sudo python3 Setup_ntp_client.py	Stops automatically	7.
9.	Connecting to a raspberry remotely	On a computer terminal connected via ethernet to the desired Raspberry	For clients: ssh admin"x"@192.168.1."x" And enter the code Admin"x" # "x" is the number of the Raspberry pi For server: ssh <a href="#">admin@192.168.1.253</a> And enter the code Admin (It is normal for the code not to appear in the terminal)	Entre "exit" in the terminal	6. (server) 7.(client)
10.	Setting camera aperture and focus	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi sudo python3 Camera_setup.py And follow 3.2.2.1	Do ctrl+c	7.
11.	Verification of speeds as a function of camera parameters	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi For resolution: sudo python3 TimeCapture.py For height resolution only: sudo python3 TimeCaptureHeights.py For width resolution only: sudo python3 TimeCaptureWidths.py	Stops automatically	7.

For time exposure: sudo python3 TimeCaptureTimeExposure.py					
12.	Check ntp synchronization	On the desired Raspberry client terminal	cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi  sudo python3 Csvntp.py	Do ctrl+c	8.
13.	Checking camera operation	1.On the Raspberry server terminal  2.On each Raspberry client terminal	1.cd /home/admin/Documents/Server #Need to be in the right folder python Test_server.py  2.cd /home/admin"x"/Documents/Client #Need to be in the right folder. "x" is the number of the Raspberry pi  Sudo python3 Test_client.py	Do ctrl+c on the Raspberry server terminal	8. (server) 13.1 (client)
14.	Performing the distortion calculation test	On the Raspberry server terminal	cd /home/admin/Documents/Server #Need to be in the right folder python Checkerboard_server.py	Do ctrl+c	8.
15.	Performing cameras calibration test	On the Raspberry server terminal	cd /home/admin/Documents/Server #Need to be in the right folder python Stereo_server.py	Stops automatically	8.
16.	Performing experimental test	On the Raspberry server terminal	cd /home/admin/Documents/Server #Need to be in the right folder python Speckle_server.py	Do ctrl+c	8.

---

### 3.2.5. CHOICE OF SOLUTIONS

#### 3.2.5.1. PROGRAMMING LANGUAGE

The two most extensively utilized programming languages for using Raspberry Pi cameras are Python and C++. Python is valued for its user-friendly nature and comprehensive documentation, while C++ is esteemed for its optimized performance and enhanced control

over hardware, such as the camera. Python has been favored in this project for its ability to simplify usage and enhance comprehensibility. Nevertheless, it is possible to use the C++ language to enhance performance.

#### 3.2.5.2. PHOTOS OR VIDEO RECORDINGS

---

The system of simultaneous imaging presents two potential approaches: capturing images individually or recording a video and subsequently extracting individual frames. Recording offers the advantage of a faster process, achieving speeds of up to 30 frames per second, and eliminates the need to send a timestamp for each image. However, the resolution of recorded images is comparatively lower than that of individually captured images (2028x1080 for recording, 4056x3040 for individual captures using the Raspberry Pi Camera HQ [5]). Given the paramount objective of precision within this project and that the experiments planned with such a system do not require very high speeds, the decision was made to opt for the individual capture method.

#### 3.2.5.3. CAPTURING LIBRARIES

---

In the context of this project, two primary options for camera control on the Raspberry Pi were evaluated: libcamera, a powerful low-level library that offers highly customizable camera settings, and Picamera2, a Python interface specifically designed for more straightforward and efficient image handling. While libcamera offers advanced control over image parameters, Picamera2 was ultimately selected for its accessibility and efficiency, which align more closely with the project's requirements. Picamera2 allows for the configuration of camera parameters at initialization, facilitating a higher frame rate as the settings do not need to be adjusted continuously [7]. This method supports capturing a greater number of images per second, essential for achieving the desired performance without compromising ease of use.

#### 3.2.5.4. ETHERNET

---

For this project, both Ethernet and Wi-Fi were considered for data transfer. While Wi-Fi reduces the need for cables, it has higher latency and struggles with high data rates, making it less reliable for large data transfers. Ethernet was chosen for its superior speed, stability, and efficiency in handling large amounts of data without delays or interruptions.

#### 3.2.5.5. FILE TRANSFER

---

The file transfer protocol in this project employs SCP (Secure Copy Protocol). An alternative option considered was the utilization of SFTP (Secure File Transfer Protocol); however, the drawback with SFTP lies in its slightly slower transfer speed due to the overhead of additional encryption layers and session handling [8]. Consequently, SCP was chosen for its simplicity and faster performance in the context of transferring large image files, even though SFTP offers more features such as better control over file transfer and directory listing capabilities.

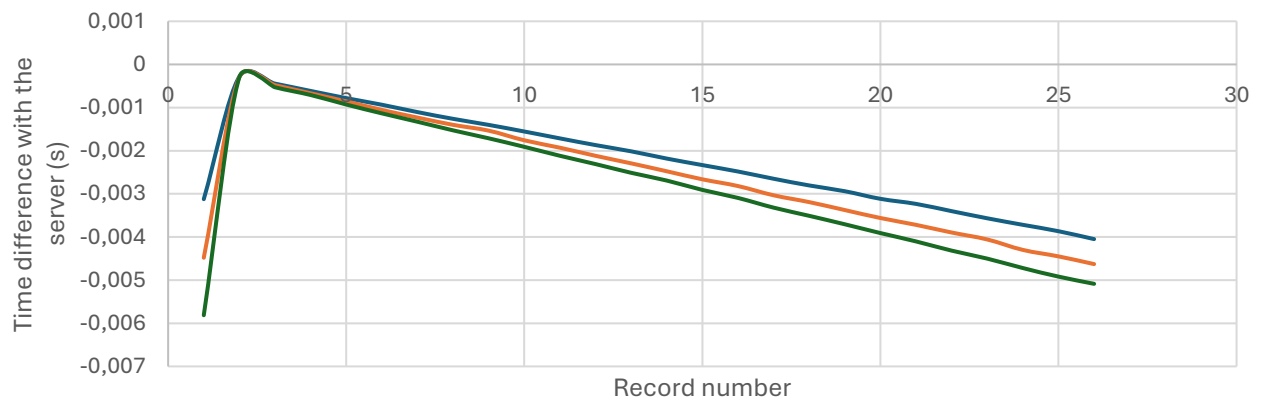
#### 3.2.5.6. LOCAL NTP SERVER

---

In a project like this, it is crucial to synchronize the acquisition of photos with a common reference time. In the context of the MultiDIC framework, any temporal inconsistencies can have significant implications for image analysis. While one way to ensure uniform timing across Raspberry Pi units is to synchronize them directly with internet time servers, this approach is vulnerable to latency and network congestion. Therefore, deploying a local Network Time Protocol (NTP) server is advisable. By eliminating the impact of network congestion and minimizing latency, this method, supported by Ethernet connectivity, guarantees the necessary precision and reliability in temporal synchronization across all participating devices during simultaneous image capture.

Comparatively, while the Precision Time Protocol (PTP) is another viable option for high-precision time synchronization, NTP was selected for this project. Although PTP offers higher precision and more frequent synchronization, it also imposes higher demands on CPU resources, which could introduce instability and potential degradation in time precision on the Raspberry Pi units. NTP, on the other hand, does not require constant updates, which results in a slight drift in synchronization relative to the server's clock (*Figure 4*). However, this drift is approximately the same for all cameras (slope between -0.00016 and -0.0002), and the maximum deviation reached after 6 minutes is limited to about 1 ms, which demonstrates that this offset will not have a major impact on the accuracy of the entire system. This controlled, minimal resource usage approach with NTP offers an effective balance between

precision and system stability, making it a practical choice for maintaining synchronized image capture without overtaxing the devices.



*FIGURE 4: GRAPHIC OF THE TIME DIFFERENCE BETWEEN CLIENTS AND THE SERVER OVER TIME (ONE RECORDING EVERY 16S). BLUE LINE REPRESENTS CAMERA 1. RED LINE REPRESENTS CAMERA 2. GREEN LINE REPRESENTS CAMERA 3.*

#### 3.2.5.7. ZIP FILE

---

Using SCP to transfer files in a ZIP format offers several key benefits. First, ZIP files reduce data size, leading to faster transfers and less bandwidth consumption, which is particularly useful for large data volumes. Combining multiple files into a single archive simplifies the process, allowing SCP to manage one file instead of several, enhancing speed and reliability.

Additionally, ZIP files improve efficiency by minimizing the number of connections needed, reducing strain on system and network resources.

ZIP files help maintain data integrity as well. If a transfer is interrupted, it's easier to resume with a single compressed file, reducing the chance of corruption compared to multiple uncompressed files. In summary, using ZIP files with SCP speeds up transfers, enhances efficiency, and simplifies management.

#### 3.2.5.8. RAM STOCKAGE

---

In this system, the decision was made to first store images in RAM before sending them to the server, rather than saving them directly on the Raspberry Pi's SD card. This approach offers several advantages.

Firstly, writing images to RAM is significantly faster, which not only reduces capture times but also minimizes discrepancies in capture times across different Raspberry Pi devices. Secondly, it helps reduce wear and tear on the Raspberry Pi's SD cards, allowing the system to remain operational for a longer period without the need to replace the SD card.

While this method limits storage capacity to a fixed 2GB compared to what the SD card can offer, it ultimately enhances the system's long-term performance

#### 3.2.5.9. PROGRAM OPTIMIZATION

---

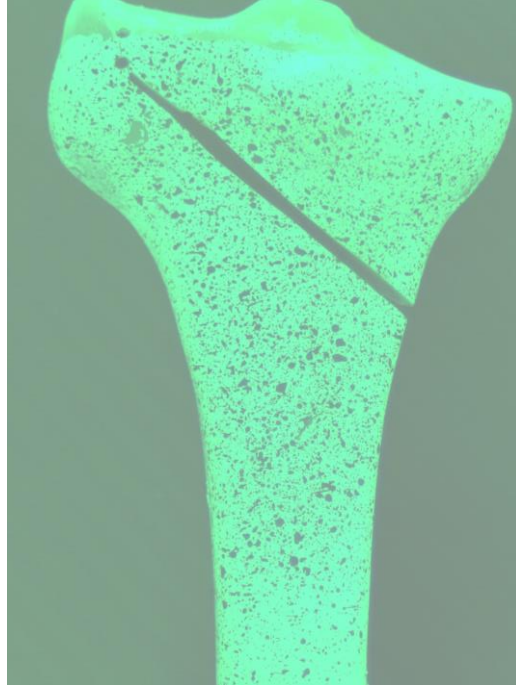
Despite the various decisions made throughout the project, the system sometimes exhibited unstable behavior. To solve this problem, several modifications were made to the code.

Firstly, specific image parameters were adjusted, including white balance, brightness and image contrast, as these factors could vary from photo to photo, resulting in inconsistent capture times for the same camera. It's important to point out that these parameter modifications lead to a change in image hue, which remains perfectly legible, but if the user prefers a more natural rendering, the values set in the code need to be adjusted (*Figure 5*). In all cases, the images will be converted to black and white by MultiDIC. However, this single adjustment proved insufficient. Further analysis of the CPU's behavior revealed fluctuations in its operating frequency. As a result, the “cpufreq” library was used to increase the CPU frequency, thereby reducing capture times.

To mitigate potential problems arising from high CPU frequency, such as overheating or excessive usage, CPU temperature and usage controls are instituted at the start of each program. This approach guarantees more consistent performance between different Raspberry Pi units but does cause a longer start-up time, in case the script takes too much time to be ready it may be useful to let the Raspberry Pi units rest for a while. In addition, as a precautionary measure, the program is configured to run with the highest priority,



minimizing the risk of background operations causing disruptions. To this end, all programs are launched using the “sudo” command, which grants extended privileges to the Raspberry Pi, thus facilitating these modifications.



*FIGURE 5: PICTURE WITH ADJUSTED PARAMETERS*

---

### 3.3.METHODS

The primary goal of this project is to achieve synchronization accuracy within  $\pm 30$  ms across cameras in a multi-camera system. This level of precision is essential for capturing events in perfect synchrony, particularly in time-sensitive applications.

A systematic approach was implemented to assess synchronization accuracy. A script displays the current time, updated every 10 ms, on a high-refresh-rate screen. Cameras are strategically positioned to capture this display, recording the visible timestamp with each image taken. Timestamps extracted from these images are then analyzed to calculate the temporal variance between cameras, providing a quantitative measure of synchronization accuracy.

Three cameras instead of two are used for a more robust evaluation. This configuration allows for multiple pairwise comparisons and offers a more comprehensive analysis of potential synchronization discrepancies. The data collected from these comparisons is then used to identify and correct synchronization issues, ensuring reliable performance throughout the system.

However, this method is limited by the screen's 165 Hz refresh rate (approximately 6 ms per frame). Due to this limitation, analyzing timing differences smaller than 10 ms is challenging, as the screen does not refresh quickly enough to provide the necessary granularity.

To address this limitation, an alternative method is proposed, involving direct measurement of image capture timing at the code level. By comparing these measurements with timestamps obtained through the first method, the consistency between the two approaches can be evaluated. If a strong correlation is observed, this would validate the second method and could potentially achieve synchronization accuracy within 1 ms. Additionally, this approach enables the collection of a larger dataset, allowing for a more detailed analysis of the system's accuracy throughout the capture process. One point that does not change between these two methods is that all tests will be spaced with the next one by a time equivalent to its total capture time to avoid the different tests influencing each other.

In practical applications, understanding the system's capture speed is crucial for users to select the settings that best suit their needs. However, since this system does not permit direct control of capture speed, it will be necessary to adjust various parameters to observe their effects on speed. To accomplish this, several programs will be used to vary parameters such as resolution and exposure time, allowing to determine capture times based on these adjustments.

---

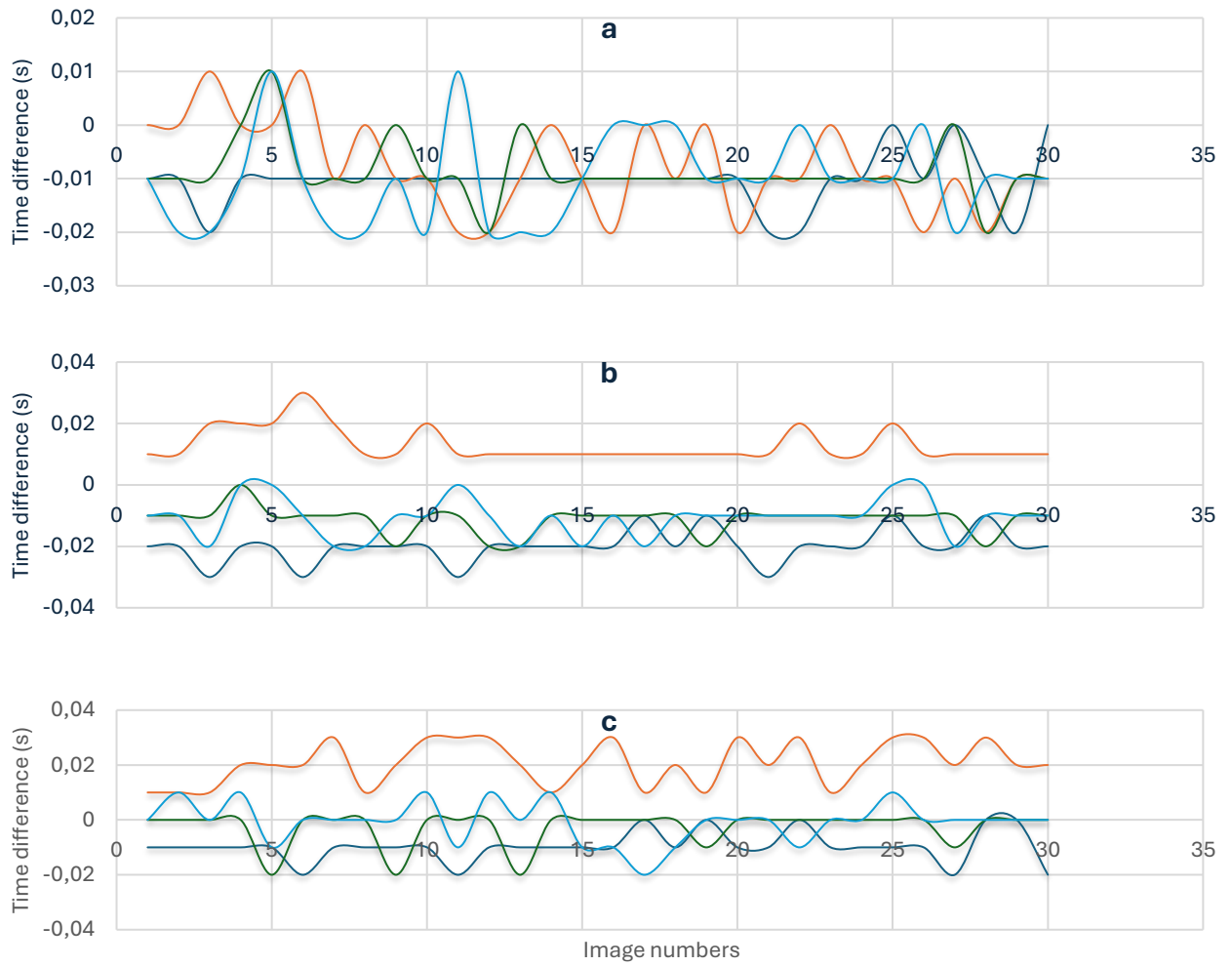
### 3.4.RESULTS

---

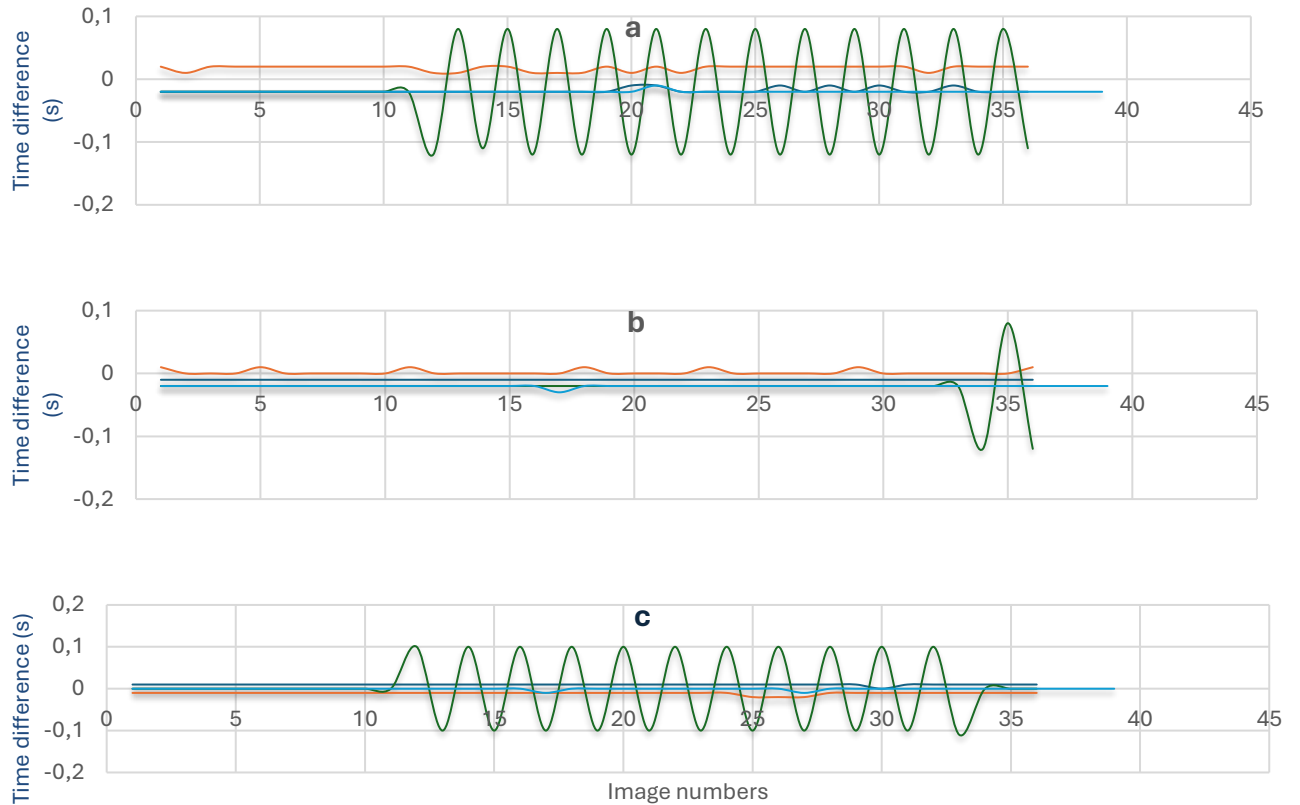
#### 3.4.1. SYNCHRONIZATION ACCURACY (PHOTO CAPTURE SCRIPTS)

To assess the accuracy of synchronization between the three cameras, two main parameters were studied: image resolution and exposure time. Time differences were calculated from the time stamps extracted from the images, in accordance with the first method described above.

Initial tests were carried out by modifying the width of the images, with resolutions of 4056, 2028, 1080 and 540 pixels. The results obtained, illustrated in Figure 6, highlight the time differences between the three cameras for a constant exposure time of 10,000  $\mu$ s. The time differences measured between the cameras for each resolution remained within a range close to  $\pm 30$  ms. This is not necessarily the case in the results, presented in Figure 7, which was carried out by varying the exposure time at 500, 1,000, 5,000 and 10,000  $\mu$ s, while maintaining a constant resolution of 4056x3040 pixels. In the case of an exposure time of 1,000  $\mu$ s, larger deviations were observed, reaching values between -120 ms and +80 ms.



**FIGURE 6: TIME DIFFERENCE BETWEEN CAMERAS WITH DIFFERENT RESOLUTIONS (TIME EXPOSURE: 10000MS). (A) CAMERA 1 AND 2. (B) CAMERA 1 AND 3. (C) CAMERA 2 AND 3. BLUE LINE REPRESENTS RESOLUTION: 4056X3040. RED LINE REPRESENTS RESOLUTION: 2028X3040. GREEN LINE REPRESENTS RESOLUTION: 1080X3040. PURPLE LINE REPRESENTS RESOLUTION:540X3040.**

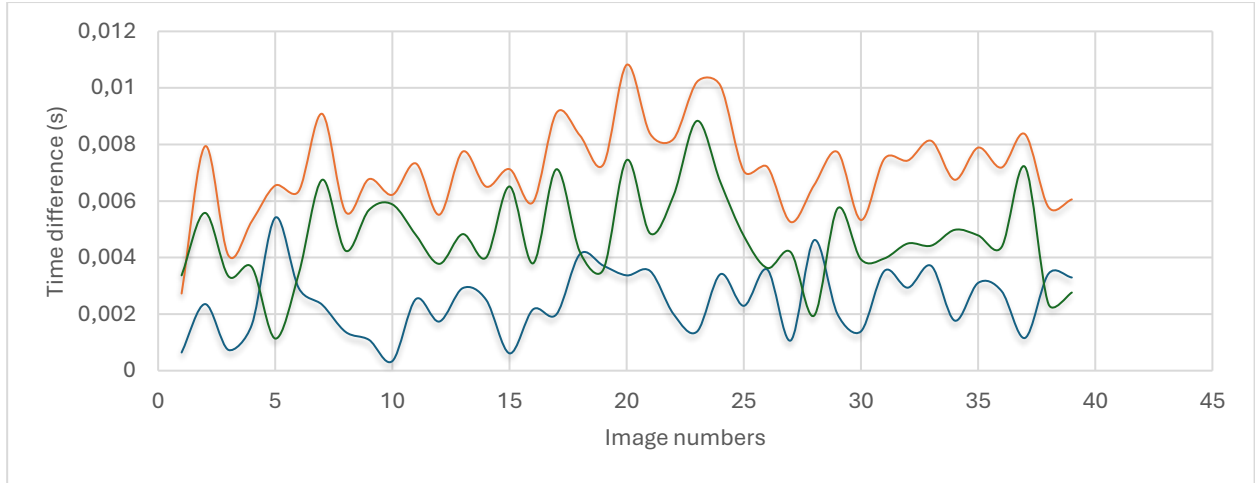


**FIGURE 7: TIME DIFFERENCE BETWEEN CAMERAS WITH DIFFERENT TIME EXPOSURE (RESOLUTION: 4056X3040). (A) CAMERA 1 AND 2. (B) CAMERA 1 AND 3. (C) CAMERA 2 AND 3. BLUE LINE REPRESENTS TIME EXPOSURE 10000MS. RED LINE REPRESENTS TIME EXPOSURE 5000MS. GREEN LINE REPRESENTS TIME EXPOSURE 1000MS. PURPLE LINE REPRESENTS TIME EXPOSURE 500MS.**

#### 3.4.2. METHOD COMPARISON (TEST SCRIPTS)

To facilitate a more complete analysis of the system, it is essential to compare the time difference results obtained by the method used previously and the second method, which calculates the time difference between the cameras directly by the program. This comparison is crucial to validating the effectiveness of the second method.

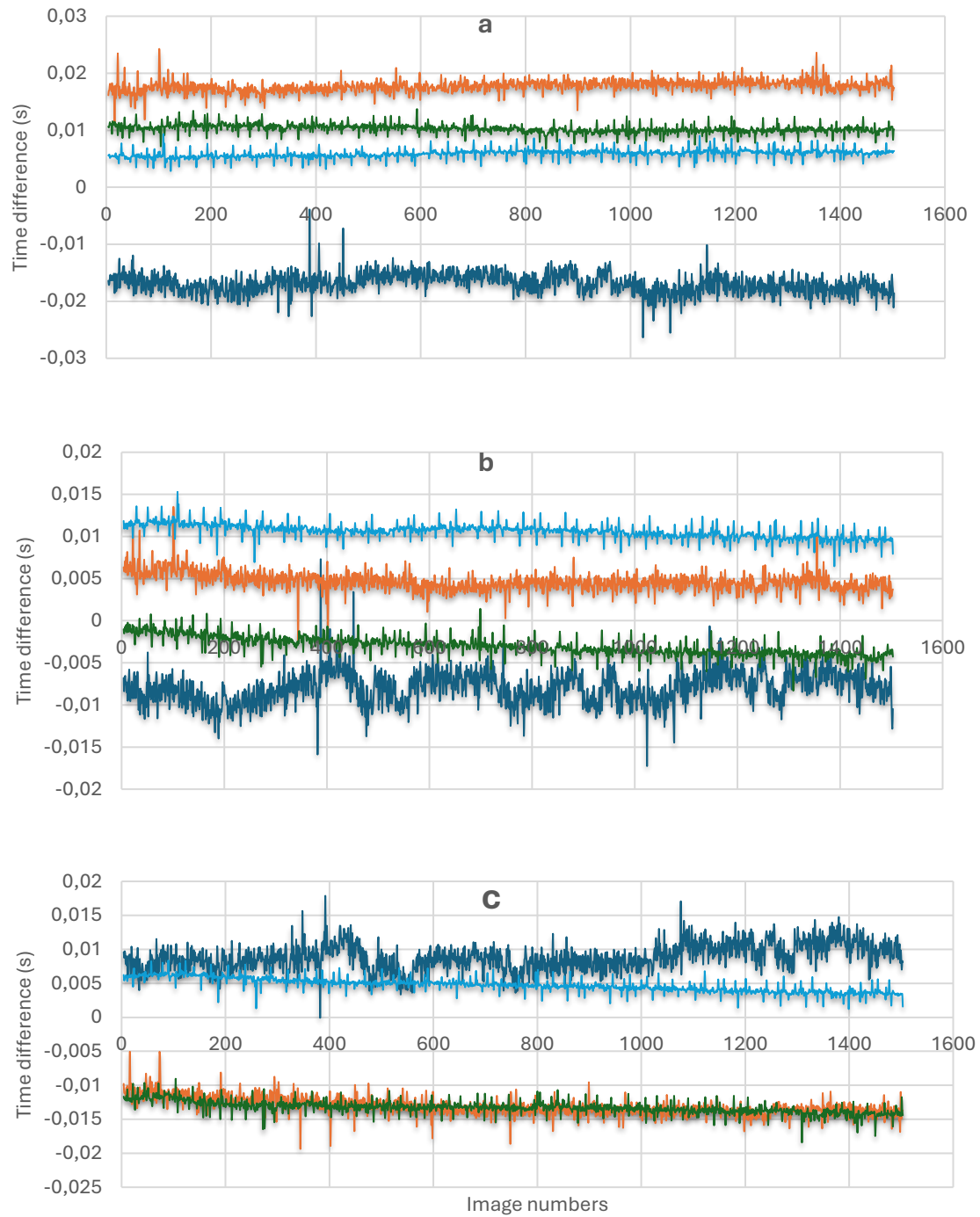
When both methods were employed simultaneously, the recorded time differences, as illustrated in Figure 8, exhibited slight variations between the three cameras. However, these differences consistently remained under 11 milliseconds across all tests.



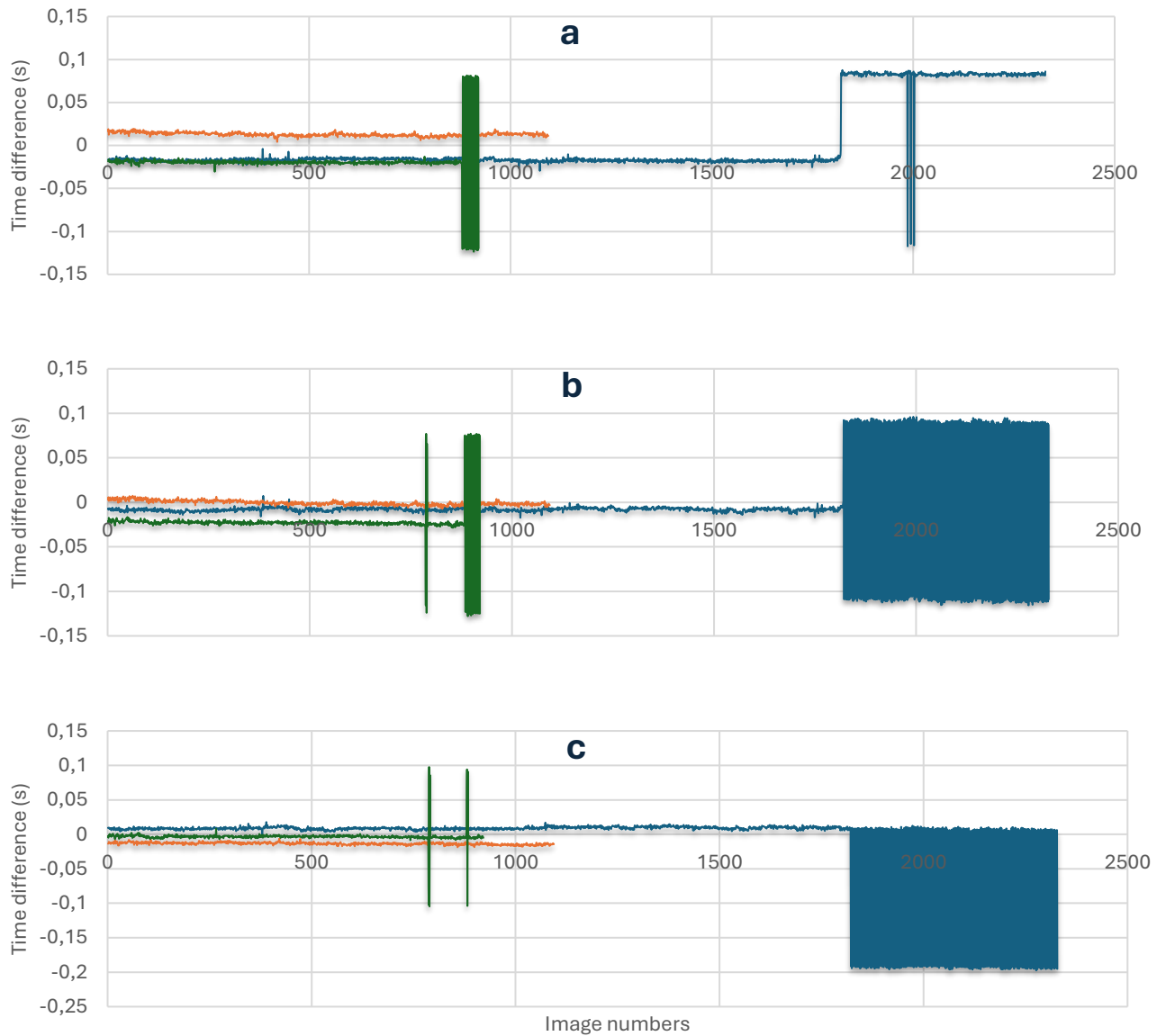
*FIGURE 8: TIME DIFFERENCE BETWEEN BOTH METHODS (RESOLUTION: 4056X3040, TIME EXPOSURE: 10000MS). BLUE LINE REPRESENTS CAMERA 1. RED LINE REPRESENTS CAMERA 2. GREEN LINE REPRESENTS CAMERA 3.*

#### 3.4.3. DIRECT TIMING MEASUREMENT (TEST SCRIPTS)

The second method enables testing a larger sample of photos compared to the first one, allowing for variations in resolution and exposure time to be analyzed. Figures 9 and 10 now present the differences in capture time between each camera, rather than the timestamps used in the first method. The data in Figure 9 represent the results obtained by varying the resolution (4056x3040, 2028x3040, 1080x3040, 540x3040) on a sample of 1,500 photos. Regardless of the camera or resolution, the differences remain within a range of approximately  $\pm 30$  ms. Figure 10 illustrates the time differences between cameras when the exposure time is varied (10,000, 5,000, 500) across a sample of over 2,000 measurements, with the resolution fixed at 4056x3040. Several important points can be noted. Firstly, the 5000 $\mu$ s case is limited to just over 1000 frames by the RAM storage of the Raspberry Pi units. Secondly, much larger differences appear from around 800 images for the 500 $\mu$ s exposure time and from around 1800 images for the 10000 $\mu$ s exposure time for all the cameras.



**FIGURE 9: TIME DIFFERENCE BETWEEN CAMERAS WITH DIFFERENT RESOLUTIONS (TIME EXPOSURE: 10000MS). (A) CAMERA 1 AND 2. (B) CAMERA 1 AND 3. (C) CAMERA 2 AND 3. BLUE LINE REPRESENTS RESOLUTION: 4056X3040. RED LINE REPRESENTS RESOLUTION: 2028X3040. GREEN LINE REPRESENTS RESOLUTION: 1080X3040. PURPLE LINE REPRESENTS RESOLUTION:540X3040.**



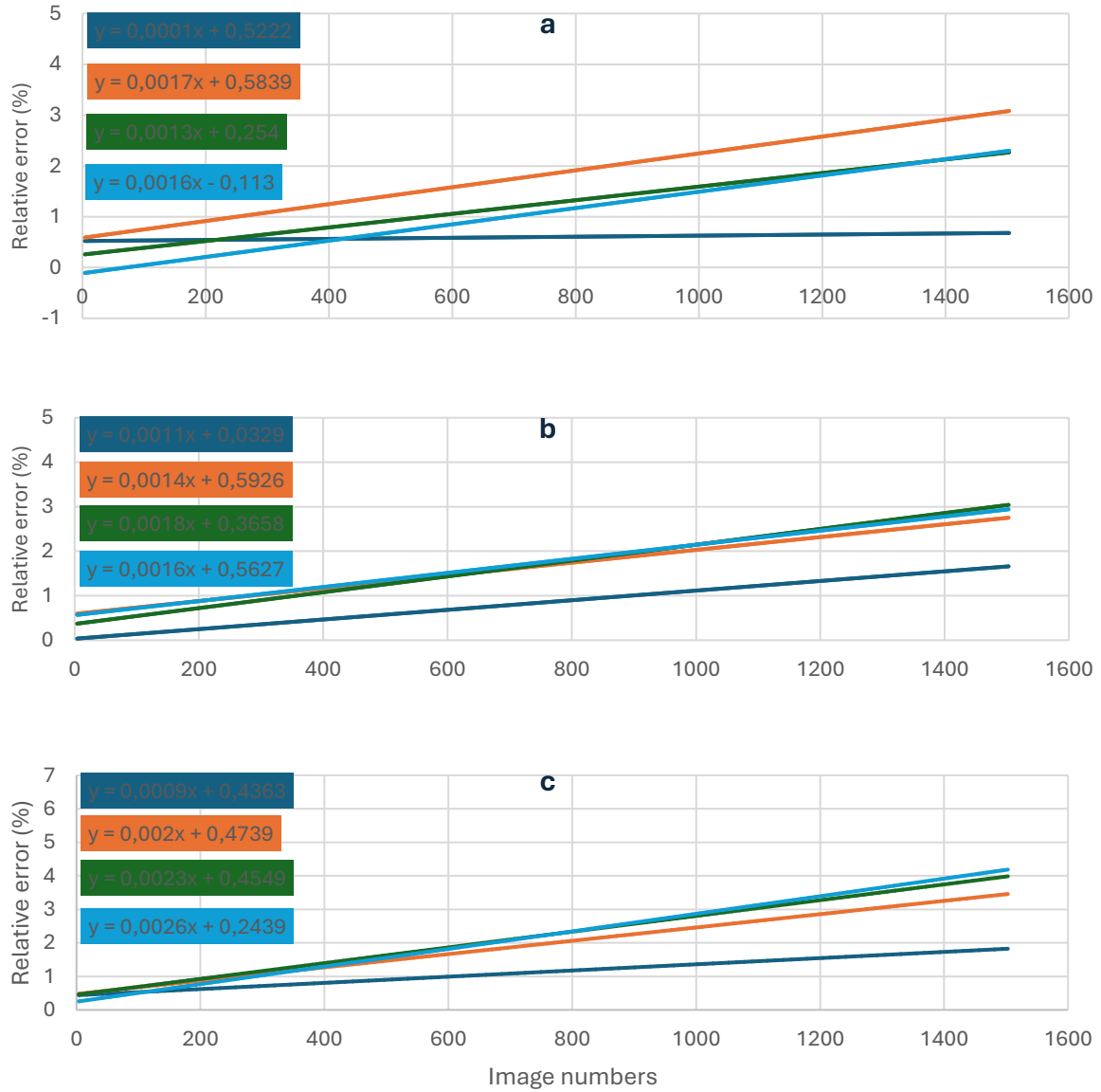
**FIGURE 10: TIME DIFFERENCE BETWEEN CAMERAS WITH DIFFERENT TIME EXPOSURE (RESOLUTION: 4056X3040). (A) CAMERA 1 AND 2. (B) CAMERA 1 AND 3. (C) CAMERA 2 AND 3. BLUE LINE REPRESENTS TIME EXPOSURE 10000MS. RED LINE REPRESENTS TIME EXPOSURE 5000MS. GREEN LINE REPRESENTS TIME EXPOSURE 500MS**

#### 3.4.4. IMPACT OF PROLONGED SYSTEM USE

In the previous tests, in addition to calculating the differences between cameras, the relative errors for each camera are also recorded. Figure 11 presents the affine trend lines for each camera as the resolution is varied. For each camera, the direct coefficient is observed to differ



across resolutions, ranging from 0.0013 to 0.0017 for camera 1, from 0.0014 to 0.0018 for camera 2, and from 0.002 to 0.0026 for camera 3.



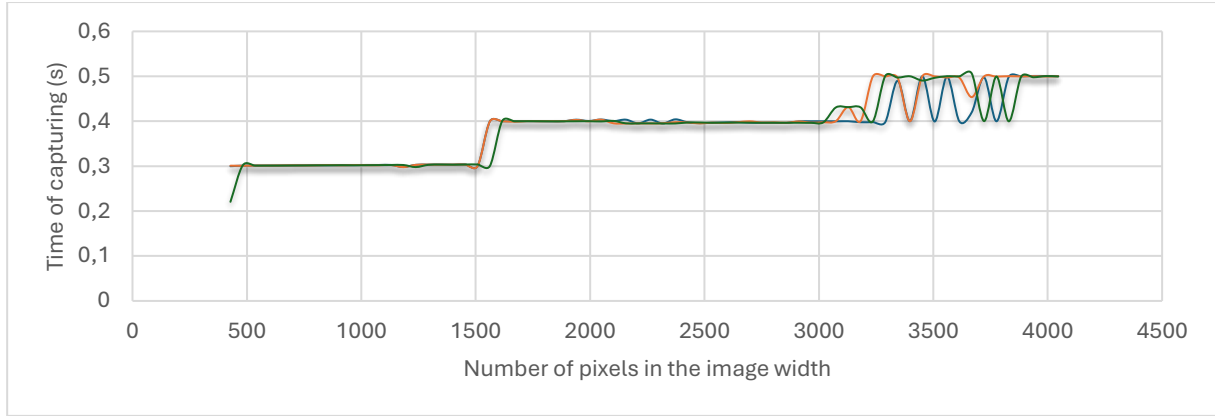
**FIGURE 11: RELATIVE ERROR TREND CURVE WITH DIFFERENT RESOLUTIONS (TIME EXPOSURE: 10000MS). (A) CAMERA 1. (B) CAMERA 3. (C) CAMERA 3. BLUE LINE REPRESENTS RESOLUTION: 4056X3040. RED LINE REPRESENTS RESOLUTION: 2028X3040. GREEN LINE REPRESENTS RESOLUTION: 1080X3040. PURPLE LINE REPRESENTS RESOLUTION: 540X3040.**

### 3.4.5. FRAME RATE CONSIDERATIONS

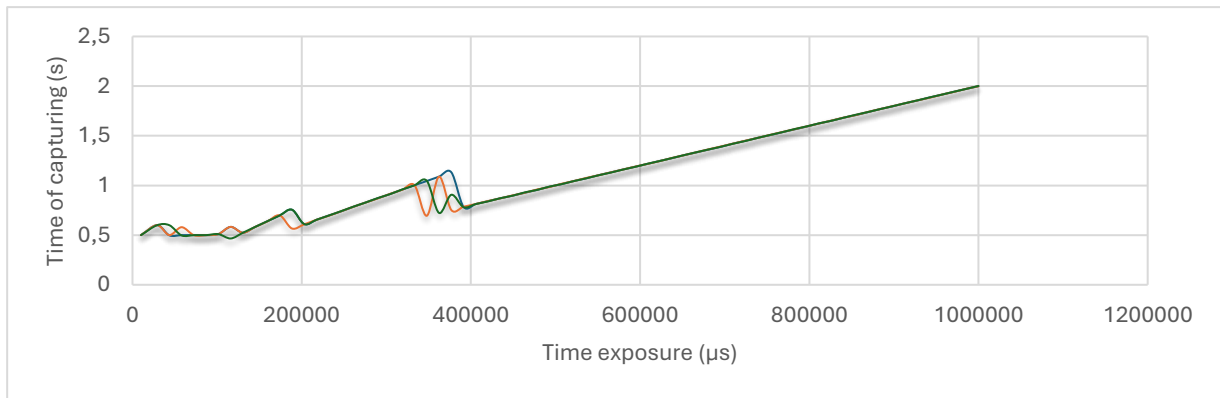
The frame rate is an important element of a capturing system. In the case of the studied system, controlling this value is not direct, so it is necessary to adjust the resolution or the exposure time. Figure 12 highlights the capture time (the inverse of the frame rate) of the different cameras as a function of the number of pixels in height. For Figure 13, the data is based on the number of pixels in width, while Figure 14 focuses on exposure time. The results illustrated in Figures 12 and 13 reveal step values (0.3, 0.4, 0.5), with the cameras producing similar outcomes in most cases. However, certain differences are observed, particularly for image heights between 2300 and 2700 pixels or widths between 3200 and 3800 pixels. In Figure 14, the capture time demonstrates step-like behavior for exposure times below 150,000  $\mu\text{s}$ . Beyond this range, the system transitions to linear behavior, with one trend occurring between 200,000  $\mu\text{s}$  and approximately 350,000  $\mu\text{s}$ , and another observed above 400,000  $\mu\text{s}$ .



**FIGURE 12: PHOTO CAPTURE TIME AS A FUNCTION OF THE NUMBER OF PIXELS IN THE IMAGE HEIGHT (TIME EXPOSURE: 10000MS, PIXELS IN THE IMAGE WIDTH: 4056). BLUE LINE REPRESENTS CAMERA 1. RED LINE REPRESENTS CAMERA 2. GREEN LINE REPRESENTS CAMERA 3.**



**FIGURE 13: PHOTO CAPTURE TIME AS A FUNCTION OF THE NUMBER OF PIXELS IN THE IMAGE WIDTH. (TIME EXPOSURE: 10000MS, PIXELS IN THE IMAGE HEIGHT: 3040). BLUE LINE REPRESENTS CAMERA 1. RED LINE REPRESENTS CAMERA 2. GREEN LINE REPRESENTS CAMERA 3.**



**FIGURE 14: PHOTO CAPTURE TIME AS A FUNCTION OF THE TIME EXPOSURE. (RESOLUTION: 4056X3040). BLUE LINE REPRESENTS CAMERA 1. RED LINE REPRESENTS CAMERA 2. GREEN LINE REPRESENTS CAMERA 3.**

### 3.5.DISCUSSION

Analysis of the results revealed several key factors influencing camera synchronization and system stability over time. The comparison of the two methods used to measure time differences between cameras revealed a difference of 11 ms between them. This difference can be explained by the inherent limitations of the first method, which introduces two types of uncertainty. The first uncertainty of 5 ms, is due to the stopwatch refresh delay. The second

of 10 ms, is linked to the 10,000  $\mu\text{s}$  exposure time of the cameras during testing, which affects the accuracy of temporal measurements.

Combining these two uncertainties using the quadratic formula (1) results in an overall uncertainty of 11.2 ms. The fact that the deviations between these two values are less than this uncertainty confirms that the second method is indeed operational for analyzing the system.

$$U_G = \sqrt{U_S^2 + U_T^2} \quad (1)$$

Using both methods, it was determined that image resolution had no significant impact on the accuracy of synchronization between cameras for a sample of fewer than 1,500 photos with an exposure time of 10,000  $\mu\text{s}$ . On the other hand, in tests with an exposure time of 1,000  $\mu\text{s}$  at a resolution of 4056x3040, significant oscillations appeared in the time differences between the cameras, even with low values. This oscillation phenomenon also appeared in other cases, notably when the number of photos increased.

An analysis of capture speeds based on resolution and exposure time revealed a stepped behavior. This indicates that when exposure times are less than 150,000  $\mu\text{s}$ , the system can only capture images at specific speeds that are approximately 10 ms apart. This stepping behavior could pose challenges, particularly when resolution and exposure time combinations result in intermediate capture times between two steps, potentially causing oscillations between these values.

Another critical observation is that the relative error tends to increase gradually over time for each camera. Although this increase is linear and similar across all three cameras, it indicates a gradual increase in capture time. This could result in capture values becoming trapped between two steps, leading to delayed oscillations, which have been observed in several instances.

Ensuring the stability of the selected torque throughout the operation is essential before using the various capture programs. To verify this stability, it is recommended to evaluate both the

capture time and the relative error by using the test programs `Test_client.py` and `Test_server.py`. Running these tests on the Raspberry Pi at least once can help confirm the desired stability. During experiments, tests should consistently be conducted for a duration equal to or longer than the total capture time of the previous test. Failing to meet this criterion may lead to system instability, particularly due to potential overuse.

These observations highlight the critical importance of carefully selecting the appropriate combination of resolution and exposure time while ensuring operational stability. Both elements are directly linked to the system's accuracy, regardless of the size of the photo sample being processed.

## 4. CONCLUSION

This project aimed to develop a synchronized capture system compatible with the MultiDIC software, addressing the limitations of the previous system. The objective was to create an easy-to-use solution capable of capturing images with a temporal accuracy of  $\pm 30$  ms between cameras while enhancing image resolution compared to the prior version.

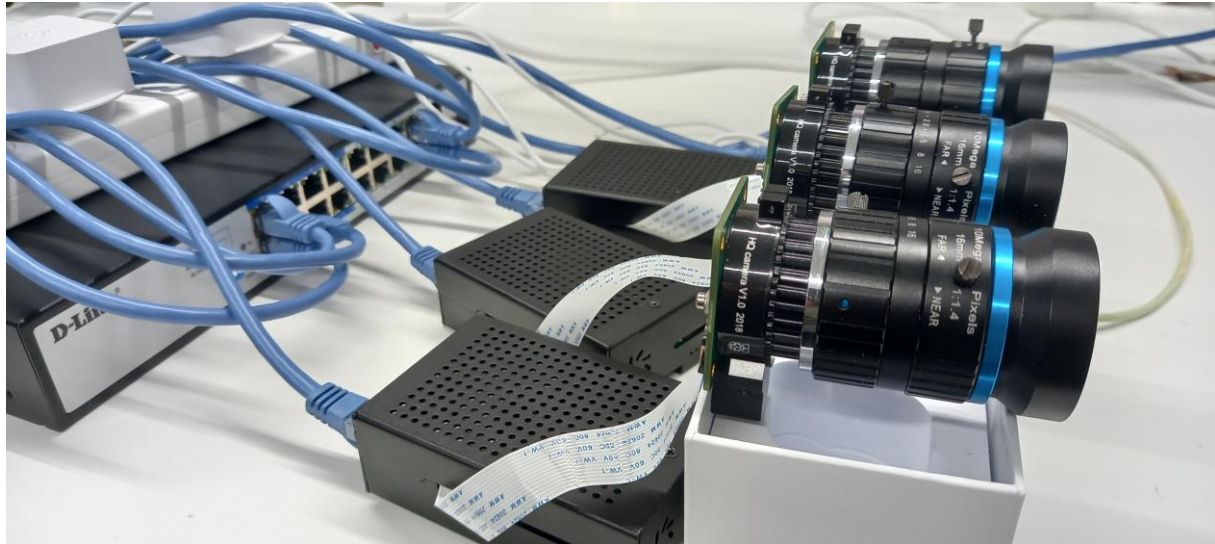
To accomplish this, the hardware was modernized by replacing the outdated components (Raspberry Pi Zero-W and V2 camera) with Raspberry Pi 4s and Raspberry Pi HQ cameras. A suite of user-friendly programs was designed to facilitate various operations, including image capture and system feature testing. These programs come with an instructional manual to help new users familiarize themselves with the system with ease.

To ensure that the various scripts met the project's expectations, the Picamera2 library were employed along with Python, allowing the system to operate efficiently while capturing images at a resolution of 4056x3040. Additionally, an NTP synchronization system was implemented to achieve the necessary synchronization for this project.

To validate the whole system and the scripts, a multitude of tests were carried out to analyze the temporal gaps between the cameras as a function of their resolution and exposure time, the evolution of the relative error of the capture time over time, and the capture speed of the cameras as a function of resolution and exposure time. All these tests revealed that synchronization accuracy met expectations when the chosen resolution/exposure pair allowed the capture time to align with a step value, avoiding the oscillations that occur when capture times fall between two steps.

The development of this system and these program combinations (Checkerboard, Stereo, Speckle) thus meets the expectations formulated at the start of the project, providing simultaneous captures from 12 cameras that can be used directly with MultiDIC software for precise analysis of future systems.

To finalize this project fully, tests will be necessary to obtain the number of images required to use the software and check that the analyses obtained meet the expectations of precision by measuring displacements of around 0.01mm and strains of around 0.001.



*FIGURE 15: SYSTEM PICTURE*

## BIBLIOGRAPHY

[1] Aaron Jaeger, MultiDIC/CaptuRPI, GitHub, 2018.

Link: <https://github.com/MultiDIC/CaptuRPI>

[2] Dana Solav, Kevin M. Moerman, Aaron M. Jaeger, Katia Genovese, Hugh M. Herr, MultiDIC: An Open-Source Toolbox for Multi-View 3D Digital Image Correlation, IEEE Access, June 26, 2018

[3] Raspberry Pi Documentation Accessories Camera, Raspberry Pi.

Link: <https://www.raspberrypi.com/documentation/accessories/camera.html>

[4] Dana Solav, Kevin M. Moerman, Aaron M. Jaeger, Hugh M. Herr, A Framework for Measuring the Time-Varying Shape and Full-Field Deformation of Residual Limbs Using 3-D Digital Image Correlation, IEEE Transactions on Biomedical Engineering, VOL. 66, NO. 10, October 2019

[5] Dana Solav, MultiDIC, InstructionManual, Version 1.1.0, Github, December 20, 2021.

Link:

[https://github.com/MultiDIC/MultiDIC/blob/master/docs/pdf/MultiDIC v 1 1 0 instruction manual.pdf](https://github.com/MultiDIC/MultiDIC/blob/master/docs/pdf/MultiDIC_v_1_1_0_instruction_manual.pdf)

[6] Raspberrypi/picamera2, mjpeg\_server.py, GitHub, 2022.

Link: [https://github.com/raspberrypi/picamera2/blob/main/examples/mjpeg\\_server.py](https://github.com/raspberrypi/picamera2/blob/main/examples/mjpeg_server.py)

[7] Raspberry Pi Ltd, The Picamera2 Library, August 2024.

Link: <https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf>

[8] John Carl Villanueva, SCP vs SFTP – 5 Key Comparisons, Jscape, 2024.

Link: <https://www.jscape.com/blog/scp-vs-sftp>