THD2 Project

v1.4

Date: 2022/10/15

Authors:

Marcin Kovalevskij, David Picard

Versioning

Date	**Version**	**Description**
14 oct. 2022	v1.0	Creation of the document
21 oct. 2022	v1.1	System requirements definition
28 oct. 2022	v1.2	General document update
1 janv. 2023	v1.3	Final document update

Reference documents

Reference	Document	Description
R1	THD2_0700-WFS-001_LOWFS_upgrade_v1.	
R2	Mako_Ethernet_configuration_linux	Camera configuration.
R3	Data_conversion_memo	ADC or DAC memo
R4	T-T mirror documentation	ATP P-915K952
R5	Vimba Documentation	
R6	Vimba Programming Guide	

Acronym list

Acronym	**Description**
COG	Center of gravity
DAC	Digital-to-analog converter
ROI	Region of interest
PSF	Point Spread Function
T-T	Tip-Tilt mirror

RX/TX	Receive/Transmit
IPC	Inter Process Communication

About the project

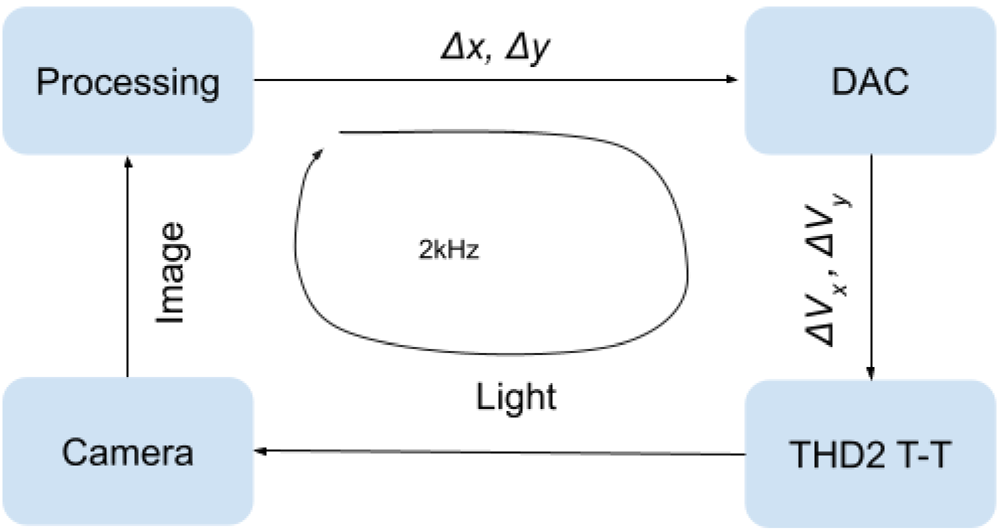


Image 1 : Closed loop THD2 T-T solution

The main goal of this project is to develop a pointing closed loop solution to correct vibration induced effects on the beam of light on the THD2 optical bench. In the exoplanet detection context, the THD2 bench is an R&D instrument designed to study and test high contrast imaging of space-based instruments. Among others, the THD2 optical bench is composed of a T-T mirror. The closed loop solution that we aim to develop should adjust the T-T mirror axes (x, y) to keep the PSF in a specific region of the camera. As shown on the image above, the closed loop consists of a computing device that calculates the PSF position error between expected and real PSF position, a DAC that converts the error into the corresponding voltage values, a T-T mirror that adjusts the position of the beam and a camera that acquires the PSF.

In this project, we particularly focus on the development of the image acquisition and processing subsystems. Mainly, one of the specifications is to develop a visualization and closed loop control interface that allows communication between the user and the closed loop.

System specifications

Loop specifications (LS)

Numbering	Requirement	Description
LS1	System should loop at 2 kHz.	
LS2	T-T correction of min. 7n rad deviation	
LS3	DAC : min. 14 bits precision	See DAC requirements

Visualization specifications (VS)

Numbering	Requirement	Description
VS1	Should display image	Adjustable frame size coming from the camera
VS2	Should display current loop rate	Value in kHz (goal is 2 kHz). It should count the images.
VS3	Should display average Δx, Δy	Computed from the COG
VS4	Should display average voltage sent to T-T	Related to Δx, Δy
VS5	Allow to enable start/stop loop	On-Off buttons to enable/disable the loop

System architecture

The system is composed of the hardware and software subsystems. The goal of the global system architecture is to integrate the hardware and the software interfaces together.

Hardware subsystem

As described in the About the project section, the hardware subsystem is composed of the processing unit, DAC, T-T mirror and a camera. In the following system hardware subsystem analysis our aim is to understand how system elements interact together. We also analyze what are the hardware subsystem requirements that satisfy the System specifications.

Camera

The image acquisition is performed by the Allied Vision Mako G-040B camera. Its full hardware and software documentation can be found in the accompanying project documentation folder. However, the main camera specifications are described in the table below.

NUMBERING	SPECIFICATION	VALUE
CS1	Interface	IEEE 802.3 1000BASE-T IEEE 802.3af (PoE)
CS2	Resolution	728 (H) × 544 (V)
CS3	Sensor	Sony IMX287
CS4	Sensor type	CMOS
CS5	Sensor size	Type 1/2.9
CS6	Pixel size	6.9 µm × 6.9 µm
CS7	Shutter mode	Global shutter
CS8	Lens mounts (available)	C-Mount, CS-Mount
CS9	Max. frame rate at full resolution	286 fps
CS10	ADC	12 Bit
CS11	Image buffer (RAM)	64 MByte
CS12	Exposure time	[min, max] ∈ [19 ; 85899300] µs

Given the total loop frequency specification of at least $2\,kHz$ it is required that the camera acquires images at a frequency greater than $2\,kHz$. Therefore, camera image acquisition speed shouldn't be the closed loop system bottle-neck. Camera acquisition speed (frame rate CS9) can be increased by decreasing camera resolution (CS2), and/or decreasing the camera exposure time (CS12) and by changing acquisition mode (see Image acquisition) . We further discuss system latency requirements in the Latency estimation section.

Processing

The system processing part should communicate with the camera to control the image acquisition, provide a control interface between the user and the loop system and send control information to the DAC. Processing unit being a central control hub of the system, we want the data to be received, processed and transmitted at high speeds, on the order of 100 µs. Processing unit should run on the Linux based operating system and have at least four or more CPU cores running at ~3GHz or higher speeds to handle multi-tasking. The processing unit doesn't require either a lot of RAM (4 Gb is sufficient) nor a separate graphics card.

DAC

The main DAC specifications that are of interest in this project are DAC precision and DAC sampling rate (described below), also the DAC should have two output channels (for the x and y directions correspondingly), it should support Linux based OS and have a price tag of less than $1000.

Minimum resolution

Considering the systems block diagram above, one of the functions of the computing card is to perform digital-to-analog conversion function. This is necessary as the TT mirror input is the analog voltage signal, in the range of $[-\ 5; 5]\ V$. The finest precision achievable by the TT mirror is $7\,nrad$ corresponding to $1mV$. From the finest precision $q\ =\ 1mV$, we can calculate the minimum number of bits needed to control the TT mirror with the maximum system achievable resolution, $q\ =\ V_{ref}/2^{N}\ \Rightarrow\ N\ =\ \lceil \ln_{2}(V_{ref}/q) \rceil$, which yields $N\ \geq\ 14\,bits$.

Sampling rate per second

From the maximum operating frequency (data rate) of the T-T mirror, $f_{d}\ =\ 1.5\,kHz$, we can determine the minimum sampling rate per second. $f_{d} = f_{s}/(M)$, where M is the decimation count (decimation count - number of skipped clock cycles until the next data point acquisition (as described in the R4 Data_conversion_memo), taking M = 2, we obtain $f_{s}\ =\ 3\,kHz$.

DAC solution propositions

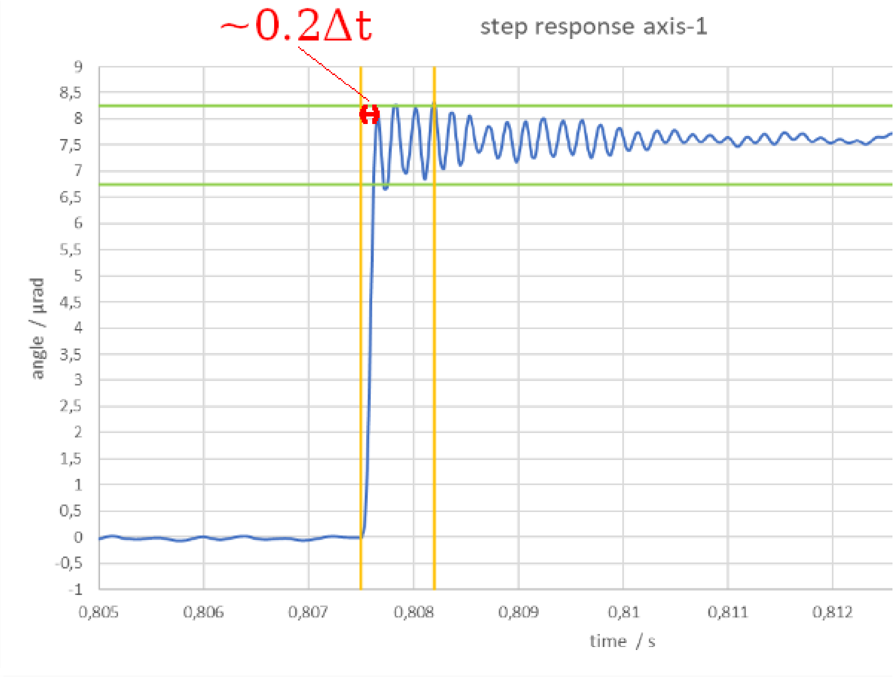
Given the DAC specification above we compile a list of DACs that correspond to some or all above mentioned specifications. Particalary, the DAC solution No3. seems to be the optimal choice.

N^0	Solution	Description	Price
1	LabJack-DAQ-supplier	Not expensive	N/A
2	Microdaq-DAQ-supplier	Windows OS only	N/A
3	USB-1608GX-2AO Datasheet	The best choice	$919.00
4	MCC USB-230 Datasheet	100 kS/s Windows OS only	$289.00 $489.00
5	Mcc-118-128-daq Datasheet	Raspberry Pi DAC shield.	$100
6	Keysight -34894-usb Datasheet	32 bit. No info if 2 Ch.	$669
7	USB-1208 Datasheet	16 bit, 2 ch output. Windows OS only.	$369
8	NI-DAQ Datasheet Datasheet User guide	14 bit 16 bit Windows OS only.	$265 $520
9	Advantech-USB 4716 Datasheet User manual Linux drivers	200 kS/s 2x16 bit analog/O Linux support	$734.95
10	ESP32 microcontroller ESP32 microcontroller-wiki	Could be a proof of the concept solution	N/A

T-T mirror

The fast open-loop tip-tilt mirror (P-915K952) is part of the THD2 bench. Mainly, the T-T mirror is part of the loop solution, the mirror main purpose is to correct vibration induced beam pointing into the camera errors.

The constraints of the T-T mirror are as follows. The minimum achievable mirror angular displacement is $7\,nrad$. The value of $7\,nrad$ means that it is not relevant to send a related voltage smaller than this value. As voltage command values smaller than this won't cause the mirror to budge. Another constraint comes from the mirror latency, according to the R4 document page 12 (see Reference documents) the mirror latency is of $0.7\,ms$. However, as shown on the image below, the first response seems to be reached after 20% of this value, in other words at $0.2\Delta t = 140\ \mu s$. Theoretically, this constraint allows to achieve frequencies up to 7 kHz.



Step begin: t = 0.8075 s step end: t2 = 0.8082 s

Settling time Δt = t2 – t1 = 0.7 ms

Image 2 : Step response of the T-T mirror, ref: R4 document

Software subsystem description

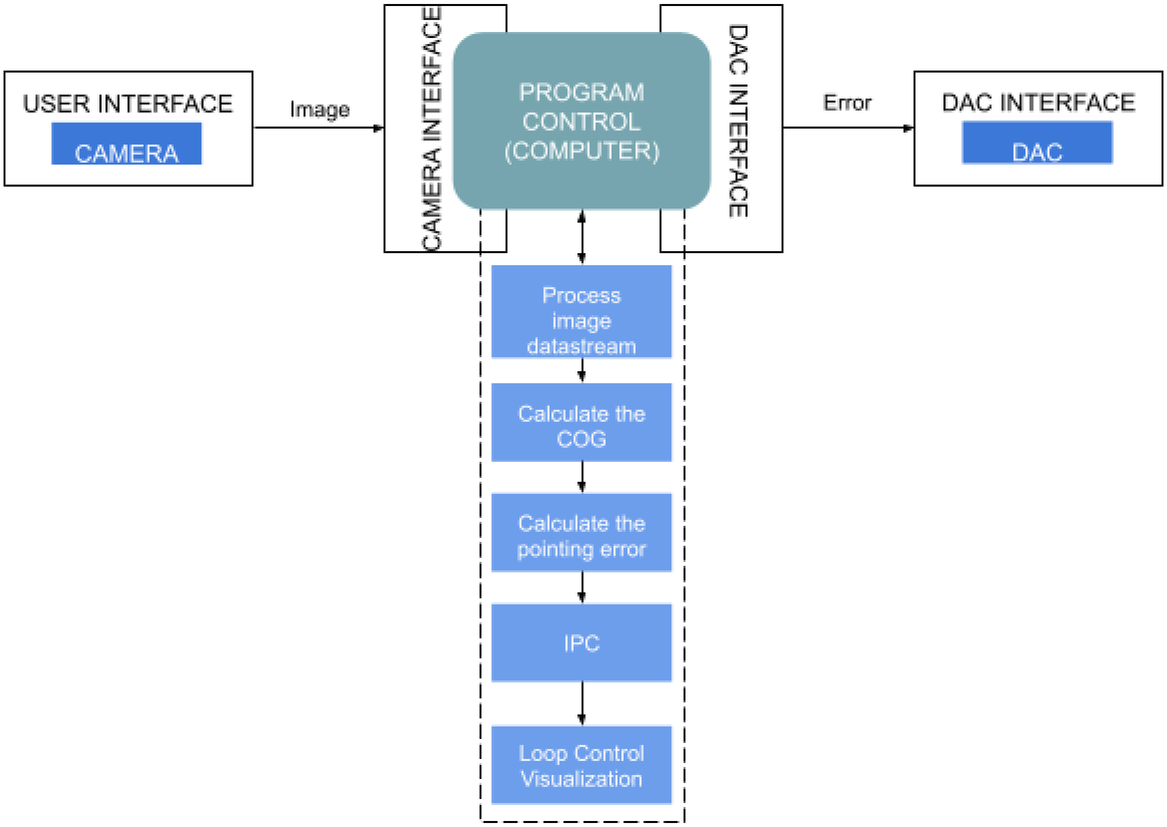


Image 3 : Software subsystem diagram

The software subsystem consists of three main parts :

1. Camera interface;
2. Program control;
3. DAC interface;

During this project we developed mainly the program control software. The camera interface software was already provided by the camera manufacturer. The DAC interface wasn't developed because the DAC wasn’t purchased due to the reasons explained in the System latency estimation section.

Camera Interface

The Vimba camera manufacturer provides extensive description on how to interface with the camera, for a complete description please see (R5) and (R6) of the Reference documents. In summary, the manufacturer provides examples regarding camera functionalities and interface in C, C++ and Python3 programming languages. The development language of our choice was C mainly because of the speed constraints.

Camera software interface

The lowest level external handle, that is made public to a developer to interface with the camera, in the C language is a structure `VmbFrame_t`,it is represented below. We use this structure, for example, to get the image datastream by accessing `VmbFrame_t void* buffer` field.

```
typedef struct
{
    //----- In -----
    void*               buffer;         //Image and ancillary data
    VmbUint32_t         bufferSize;     //Size of the data buffer
    void*               context[4];     //4 void pointers for user

    //----- Out -----
    VmbFrameStatus_t    receiveStatus; //Receiving op. state
    VmbFrameFlags_t     receiveFlags;  // Flags frame information

    VmbUint32_t         imageSize;     // Image data size
    VmbUint32_t         ancillarySize; // Ancillary data size

    VmbPixelFormat_t    pixelFormat    // Image pxl format

    VmbUint32_t         width;         // Width of an image
    VmbUint32_t         height;        // Height of an image
    VmbUint32_t         offsetX;       // Horizontal image offset
    VmbUint32_t         offsetY;       // Vertical image offset

    VmbUint64_t         frameID;       // Unique ID of a frame
    VmbUint64_t         timestamp;     // Frame timestamp

} VmbFrame_t;
```

Image acquisition

Image acquisition can be performed synchronously or asynchronously. Because of its simplicity, we chose a synchronous acquisition method. We briefly describe each method below.

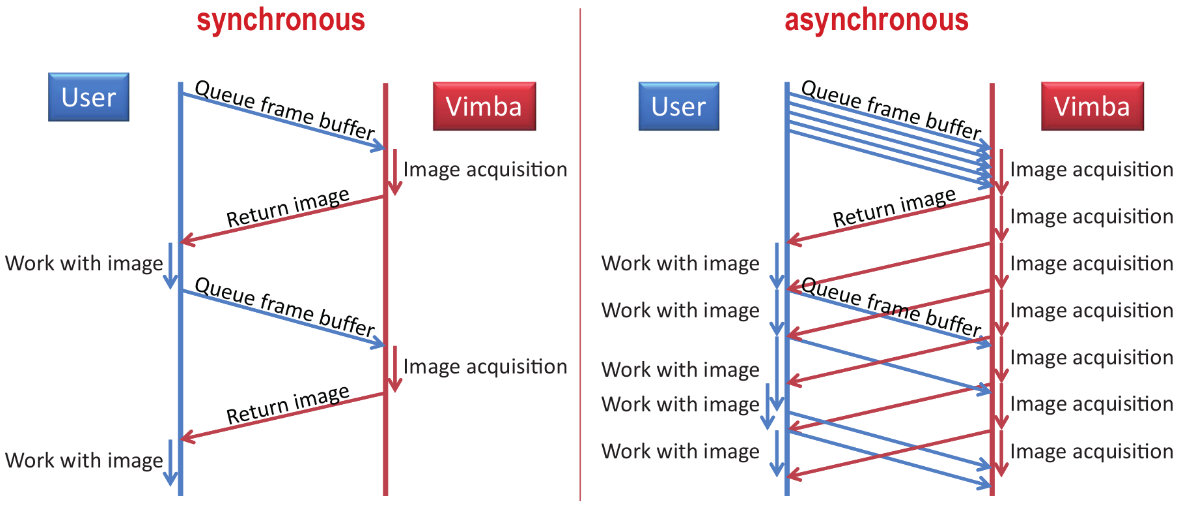


Image 4 : Synchronous vs Asynchronous acquisition, ref. Vimba Manual for Linux

Synchronous acquisition

Synchronous frame acquisition principle is quite simple. User requests for an image, then the user waits for the image to return. Because software isn't doing anything but waiting for the image to arrive after the request, this method is inefficient as it doesn't allow high frame rates. As the analogy one can think about the synchronous acquisition as juggling with one ball.

Asynchronous acquisition

Asynchronous frame acquisition principle is more complicated than the latter. A user requests for several images at once separated by a short time interval. It allows processing of images while other images are still acquired without having to wait for the return of an image. However, it requires handling several buffers at once, therefore this method is more complicated. As the analogy one can think about the asynchronous acquisition as juggling with several balls.

Program Control

The program control is composed of the : image acquisition, calculation of the COG and the corresponding voltage, inter process communication (in order to communicate between the acquisition program and the python visualization) and closed loop control/visualization parts.

Center of gravity (COG) computation

COG algorithm

Below we present the algorithm that we use to compute the COG. The algorithm below is extremely simple - it traverses a 2D image, then it calculates the respective COG with respect to x and y coordinates as the weighted average of the pixel intensity and pixel position.

```
i = 0;
while (i < height)
{
       j = 0;
       while (j < width)
       {
              intensity = get_pixel(frame,i,j);
              intensity_sum += intensity;
              cog_x += i * intensity;
              cog_y += j * intensity;

              j++;
       }
      i++;
}

cog_x /= intensity_sum;
cog_y /= intensity_sum;

delta_cog_x = cog_x_ref - cog_x;
delta_cog_y = cog_y_ref - cog_y;
```

Inter-process communication

One of the system requirements is to provide for the end-user a possibility to visualize the acquired PSF and also to control some loop parameters as described in Visualization specifications (VS). Therefore, the loop specific information should be sent from the main acquisition program written in C language to the visualization/control program written in Python3 language. In addition, we want to send the information from the visualization/control program to the main image acquisition program. Therefore, the specification for this part is to allow two-way process communication. Because direct memory sharing between C and Python3 is impossible we use inter-process communication methods to communicate the information between two programs.

In the Linux based OS environments there are three possible ways to perform inter-process communication, such as using the pipe system, using the shared memory or the message queues. We briefly describe and compare each of these methods below.

Pipes

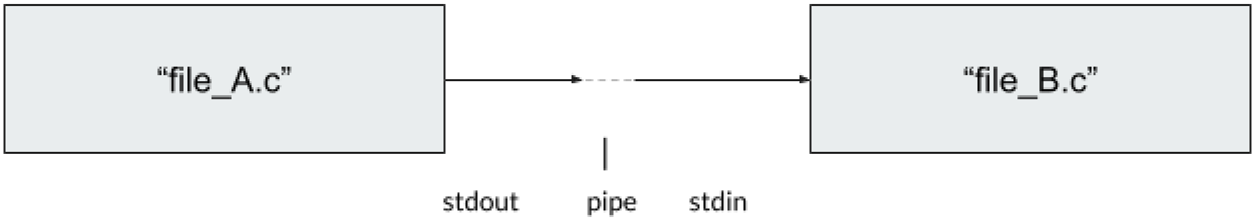


Image 5 : Linux OS pipe IPC concept representation

The idea of pipe is straightforward. Conceptually, the pipe allows connection between two different processes. It does so by connecting the standard output of the sending process to the standard input of the receiving process. In spite of the pipe method being fast and thread safe, its main disadvantage is that it's a one-way communication method.

Shared memory

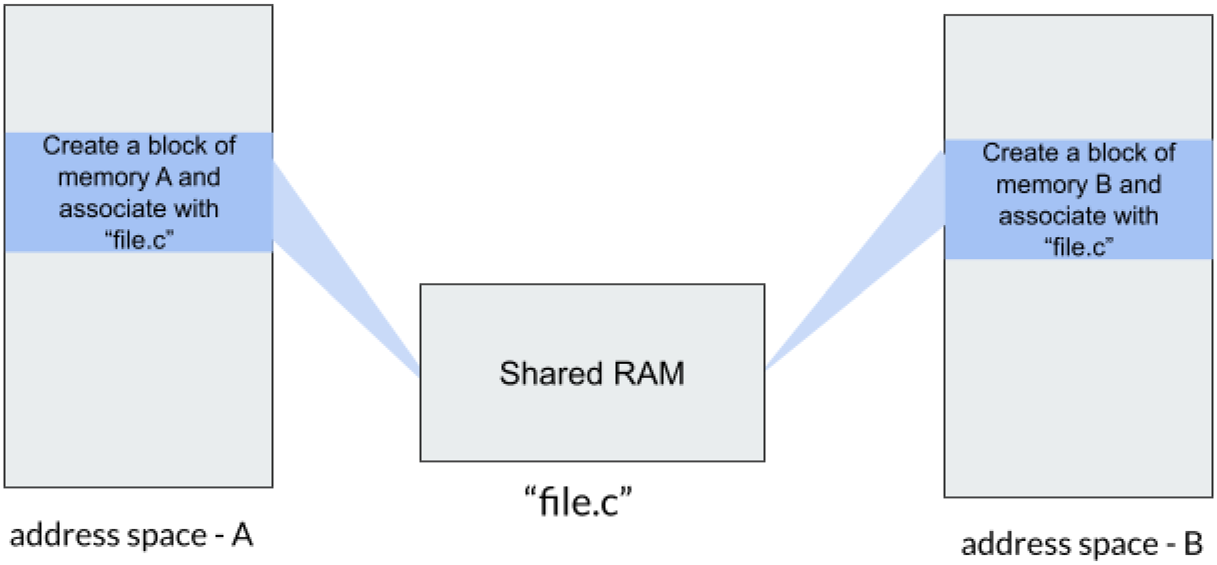

Create a block of memory A and associate with "file.c"
Shared RAM
Create a block of memory B and associate with "file.c"
address space - A
"file.c"
address space - B

Image 6 : Linux OS shared memory IPC concept representation

Shared memory establishes communication between two processes by associating a block of RAM memory with a file name, for instance "file.c". Then each of the process willing to communicate with another process (e.g. process A communicates with a process B) connects to the shared memory space. The IPC using shared memory is a fast solution as it doesn't have a memory limit and the communication can be in both ways. However, the synchronization is not easy to set up and requires the use of semaphores or mutexes, a clean solution would be difficult to implement given the constraints of the project.

Message queue

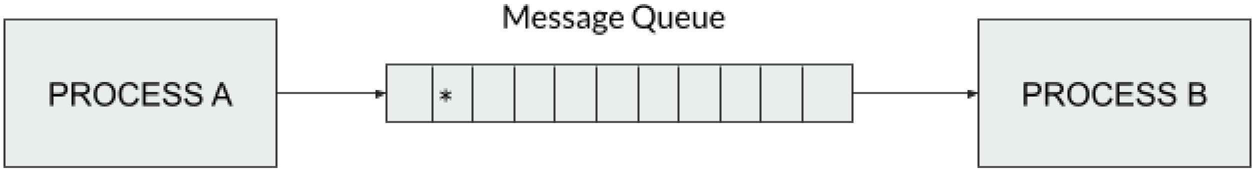


Image 7 : Linux OS Message queue IPC concept representation

The message queue method allows to establish a connection between two processes. The message queue, as the name implies, places a message into a queue and then passes the message from the transmitting to the receiving process. By choosing, for instance, a queue size of 1 the IPC is very fast, also another advantage of this method is that it is thread safe (no possible memory conflicts between receiving and emitting processes). In addition, by creating two queues it is possible to establish a two-way communication system. The main disadvantage of the message queue that it has a message size limit of $2^{16} = 65,536\,bytes$. However, because the data structures passed between two programs occupy on the order of $2^{11} = 2048\,bytes$ memory is not a limiting factor in our application. As a consequence, as an IPC solution we choose the message queue method.

Method	Advantages	Disadvantages
Pipes	Easy to implement	One directional communication
Shared memory	Very fast No memory size limits Bidirectional communication	No synchronization Not memory safe Difficult to implement
Message queues	Fast Synchronization Easy to implement Bidirectional communication	Message size limit : 2^{16} bytes

Table : Table comparing advantages and disadvantages of all three IPC solutions

Linux OS IPC commands

On the Linux based OS one can visualize IPC resources by typing to the terminal the command:

```
ipcs
```

Which yields the output :

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x62062963 0          marcinmust 666        1664         1

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 6          marcinmust 600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

One can also remove message queues and semaphores by typing to the terminal the command:

```
ipcrm -a
```

The above commands can be particularly useful for debugging purposes.

Shared data structures

The two processes in our application share the `ACQ_msg` structure to transfer the data from the acquisition program to the visualization program and `GUI_msg` to transfer data from the visualization to the acquisition program. We represent the corresponding structures below :

```c
typedef struct
{

    long mtype;
    struct GUI_data
    {
         CAM_set Cam_set;
         ACQ_set Acq_set;
         COG_rx  Cog_rx;

    }GUI_data;

}GUI_msg;
```

```c
typedef struct
{
    long mtype;
    struct ACQ_data
    {
         unsigned char buffer[BUFFER_SIZE];

         CAM_set Cam_set;
         ACQ_set Acq_set;
         COG_tx  Cog_tx;
         DAC_tx  Dac_tx;

    }ACQ_data;

}ACQ_msg;
```

It is important to note that the message queue in Linux OS is defined as a C type structure as represented above. Where the first `long mtype` field is required by default and the second field is the actual message to send, which can be of any data type, in our case we chose it to be another structure `struct GUI_data` and `struct ACQ_data`.

The data fields in the `struct ACQ_data` and `struct GUI_data` are defined in the corresponding project folders as defined in the Project software structure section.

Visualization/loop control

The visualization tool displays images detected by the camera. Also, it allows for the user to interact with the closed loop.

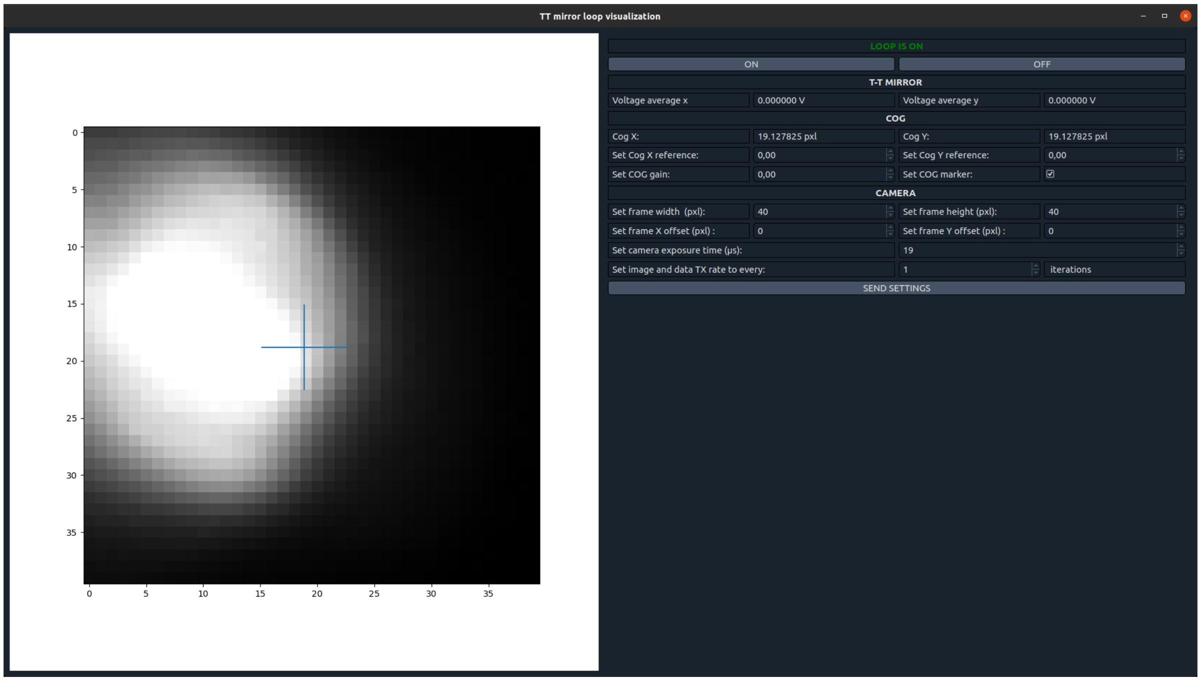
There is no need for the GUI program to be fast, it just needs to offer a GUI interface for the user. That's why we decided to develop it with Python3 and we display images in the matplotlib framework, the GUI layouts are handled using the Qt framework. Nevertheless, the GUI program features two additional threads, one is to handle message receiving and other is to refresh the image. Without these two threads, it wouldn't be possible to use the GUI as it would freeze.

From the Visualization specifications (VS), we define below technical solutions of the image visualization program:

Reference	Description
TSV-1	Retrieve the datastream in Python for display purposes. The datastream recovery should be done at ~20 images per second (If the loop rate is 2kHz, it represents only 1 image displayed per one hundred acquired images).
TSV-2	Setup the main layout using the Qt framework to organize the image frame, buttons, labels and other widgets.
TSV-3	Using matplotlib display the 2D image from the retrieved datastream. Display the image in the real time.
TSV-4	Suggest to enable or disable the loop using ON / OFF buttons.
TSV-5	Display an average of the voltages sent to the tip-tilt mirror.
TSV-6	Allow to change the COG reference with +/- buttons on X and Y axes. The COG reference is not necessarily at the center of the image, we should be able to move it.
TSV-7	Allow to modify the COG gain.
TSV-7	Allow to modify the exposure time.
TSV-8	Allow to modify the frame width and height.
TSV-8	Allow to modify the frame X and Y offsets.

The extensive documentation of the C acquisition code and Python visualization/loop control code and modular programming approach allows easy integration of new functionalities into the loop.

Visualization/loop control solution

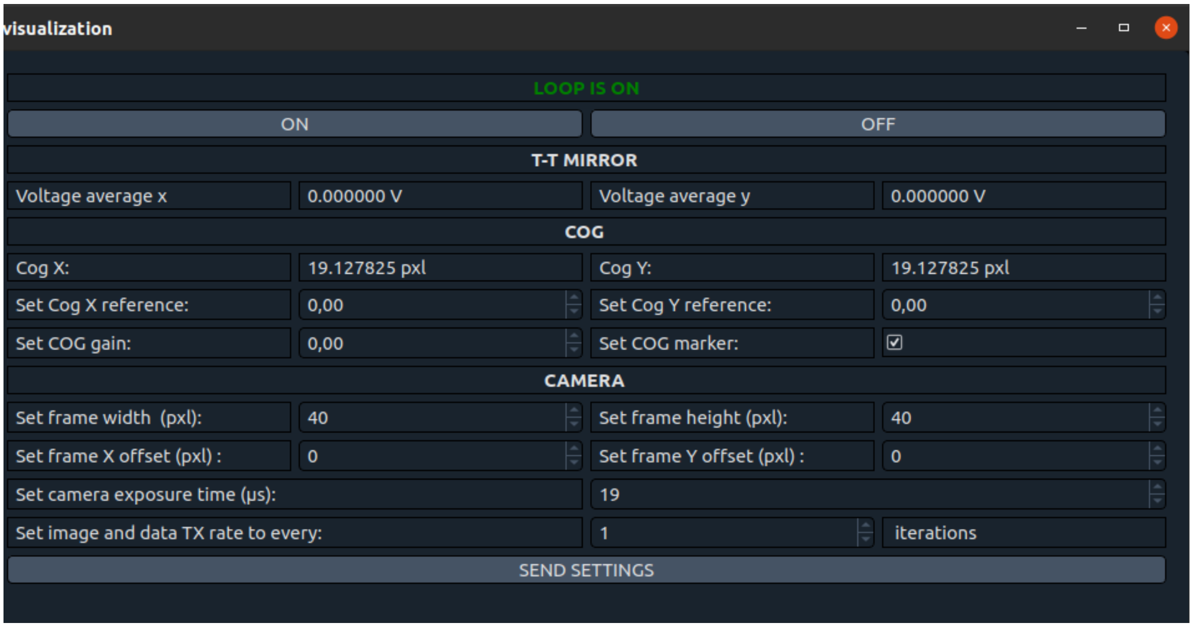


LOOP IS ON

ON | OFF

T-T MIRROR			
Voltage average x	0.000000 V	Voltage average y	0.000000 V
COG			
Cog X:	19.127825 pxl	Cog Y:	19.127825 pxl
Set Cog X reference:	0,00	Set Cog Y reference:	0,00
Set COG gain:	0,00	Set COG marker:	☑
CAMERA			
Set frame width (pxl):	40	Set frame height (pxl):	40
Set frame X offset (pxl) :	0	Set frame Y offset (pxl) :	0
Set camera exposure time (µs):		19	
Set image and data TX rate to every:		1	iterations

SEND SETTINGS


visualization

LOOP IS ON

ON | OFF

T-T MIRROR

Voltage average x: 0.000000 V | Voltage average y: 0.000000 V

COG

Cog X: 19.127825 pxl | Cog Y: 19.127825 pxl

Set Cog X reference: 0,00 | Set Cog Y reference: 0,00

Set COG gain: 0,00 | Set COG marker: ☑

CAMERA

Set frame width (pxl): 40 | Set frame height (pxl): 40

Set frame X offset (pxl) : 0 | Set frame Y offset (pxl) : 0

Set camera exposure time (µs): 19

Set image and data TX rate to every: 1 iterations

SEND SETTINGS

Image 8 : Implemented Visualization/loop control solution

Above we represent the implemented GUI visualization/loop control solution.

Acquisition and loop control software interface

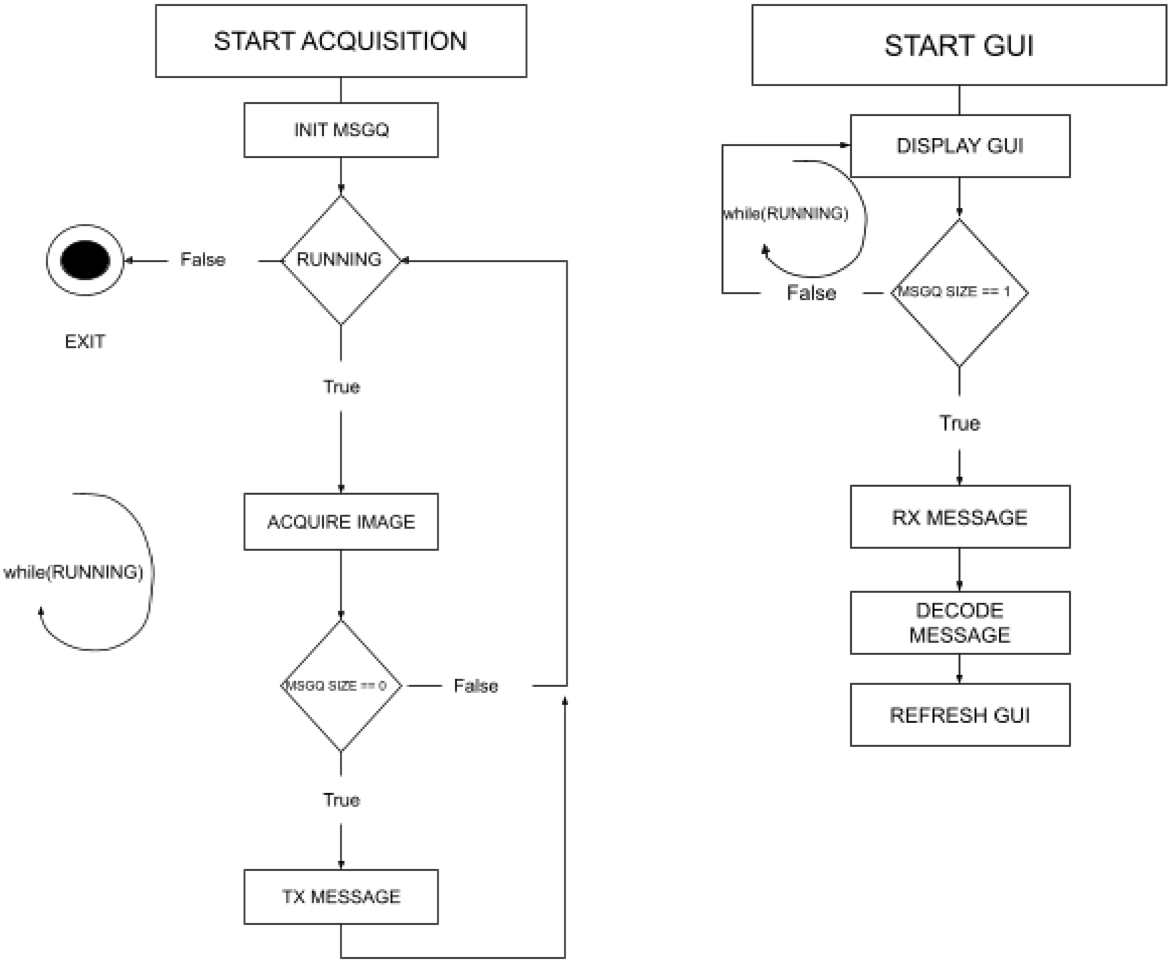


Image 9 : Software control diagrams, left acquisition program, right visualization program

In the image above we represent a simplified diagram of how the message transmission from the acquisition program to the visualization/loop control program algorithm works. Mainly, upon launching the program executable as described in the Program execution section first starts the image acquisition program. Upon starting the image acquisition program and sending the first images, the GUI program launches and waits for the incoming messages. The image acquisition program initializes, among others, the message queue then enters the infinite loop of image acquisition. After acquiring an image, if the message queue size equals 0 message is sent (which means that no message yet has been received by the visualization program). If the message queue size equals 1 the message isn’t sent, it means that message has been sent but not yet opened from the visualization/loop control program side. Once the message is read by the GUI program, message queue size is reset to 0. The message transmission algorithm from the GUI program to the acquisition program is somewhat simpler as the message is sent only when the user emits a signal by pressing the send settings button.

The diagram below allows to understand how files, which handle the interprocess communication between the acquisition and the visualization programs, are organized.

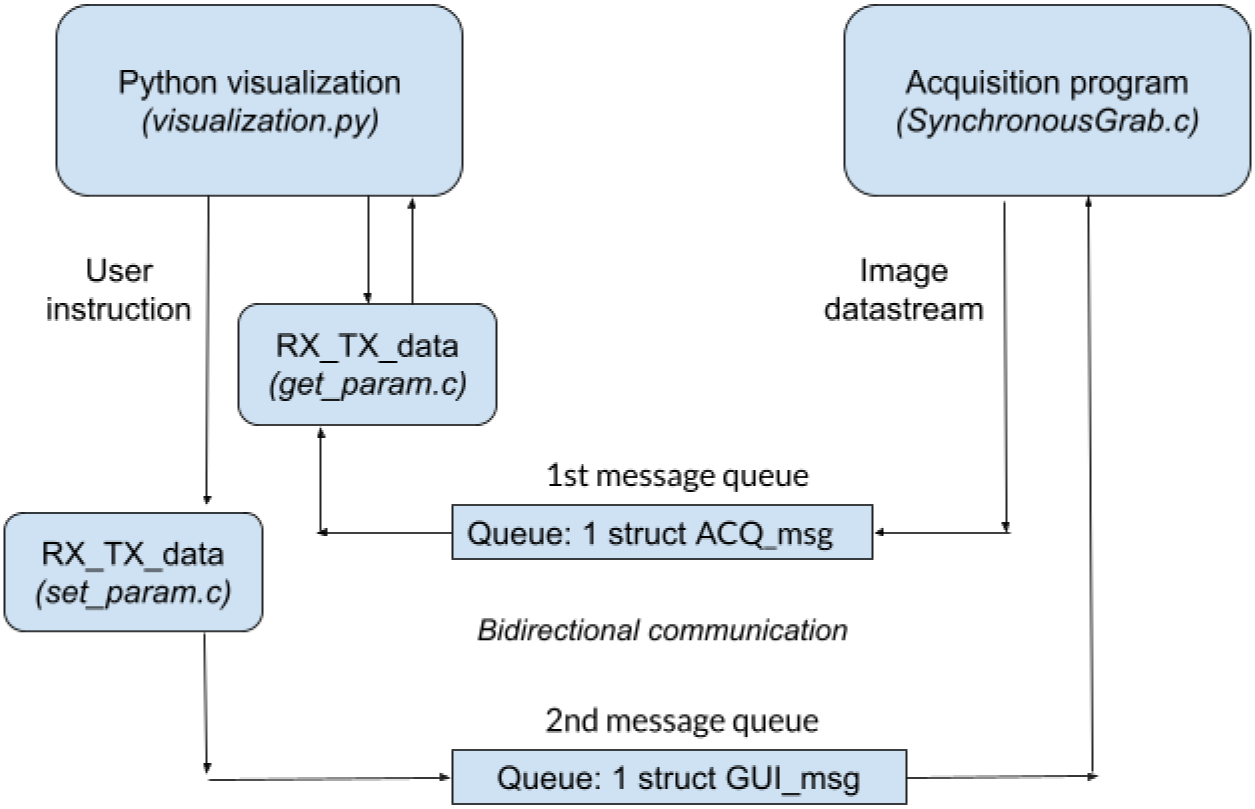
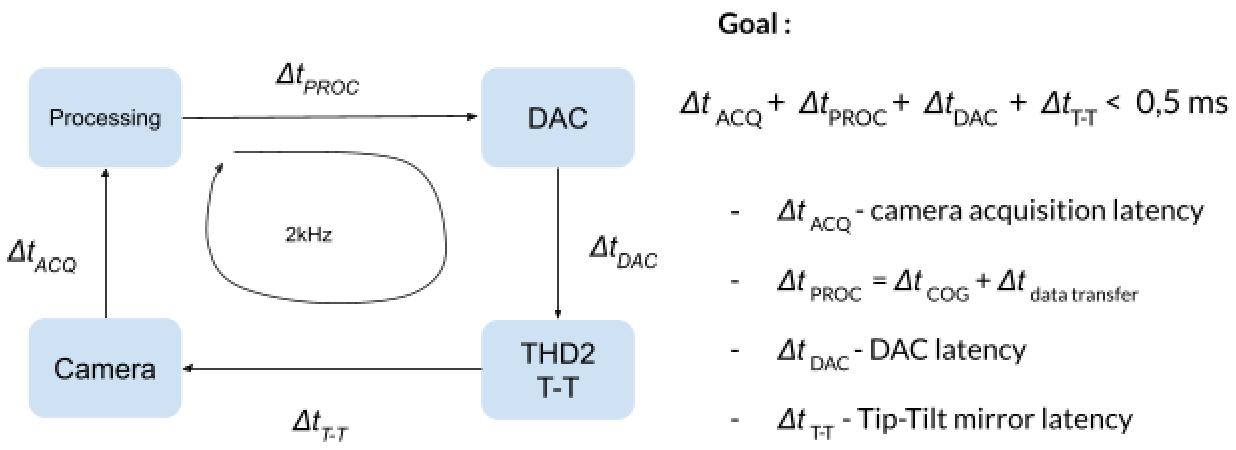


Image 10 : Files organization for the interprocess communication

System latency estimation



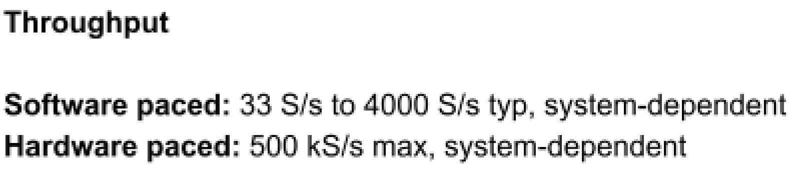
Goal :

$$\Delta t_{ACQ} + \Delta t_{PROC} + \Delta t_{DAC} + \Delta t_{T\text{-}T} < 0{,}5\ \text{ms}$$

- Δt_{ACQ} - camera acquisition latency
- $\Delta t_{PROC} = \Delta t_{COG} + \Delta t_{\text{data transfer}}$
- Δt_{DAC} - DAC latency
- $\Delta t_{T\text{-}T}$ - Tip-Tilt mirror latency

Image 11 : System latency source decomposition

The idea to determine the latency of each system part has arised after understanding that the DAC sampling rate depends on the overall system latency. For instance, considering the documentation of the DAC No. 3 as described in the DAC solution propositions we find that the DAC analog output throughput is system dependent :



Throughput

Software paced: 33 S/s to 4000 S/s typ, system-dependent
Hardware paced: 500 kS/s max, system-dependent

Image 12 : System latency DAC No 3. example

As indicated on the image above (same information can be found in the documentation of the DAC No3), software paced system throughput theoretically could achieve sampling rates twice exceeding the sampling rate specified in the system specifications. However, the throughput rate depends on the total system latency. It means that the system lag time should be between [1/4000 ; 1/33], otherwise DAC won't achieve the expected performance. Indeed, except buying the DAC and testing the DAC on the real system, we can measure each system component latency time independently, then add corresponding latency times and compare with the latency time provided in the DAC datasheet.

We therefore proceed, in the sections below, to estimate the latency of each sub-system.

Processing latency

As indicated on the (image 7), the processing latency is the sum of the image processing latency and data transfer from the computer to the DAC latency : $\Delta t_{PROC} = \Delta t_{COG} + \Delta t_{data\ transfer}$. However, because we didn’t had an opportunity to measure the image acquisition latency by the camera Δt_{ACQ} from the hardware side independently, yet we were able to measure Δt_{ACQ} from the software side, we incorporate Δt_{ACQ} into the Δt_{PROC}. Therefore we obtain that : $\Delta t_{PROC} = \Delta t_{ACQ} + \Delta t_{COG} + \Delta t_{data\ transfer}$.

As the processing latency word implies, the processing latency somewhat depends on the processing platform. The main system unit is the computer that coordinates sub-system actions, such as : request image acquisition from the camera, compute the COG and other. Below is the table summarizing system on which latency benchmarking is done.

Parameter	**Value**
Memory (RAM)	8 GB
Processor	Intel® Core™ i7-7500U CPU @ 2.70GHz × 4
OS name	Ubuntu 20.04.4 LTS
OS type	64-bit

Image acquisition latency

We measure the image acquisition latency by measuring the time the CPU requires to synchronously acquire 10000 images. We use the *sys/time.h* library which yields the computation duration in microseconds. Below we represent the method.

```
clock_t begin = clock();

/* time-consuming job */

clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC / N_ACQUISITIONS;
```

For the image acquisition latency, we find $\Delta t_{ACQ} = 7\,ms$. We observe that the synchronous image acquisition is a potential system latency bottle-neck.

COG latency

To retrieve the computing time of the COG program execution, as in the section above, we use the *sys/time.h* library. We print the clock time right before and after the COG calculation calling the function gettimeofday() which returns the time of a day. To make sure the calls of the *gettimeofday()* function do not require additional time, we execute this code twice in a row. It shows a result of 0 microseconds. As a result, it guarantees that the time shown is fully due to the code between these two function calls.

We obtain the latency of the center of gravity calculation $\Delta t_{COG} = 9\,\mu s$ for 50x50 images. Only according to this, the frequency would be $1/(9.10\wedge - 6) = 111\,kHz$. As a result, the loop rate (the goal is 2 kHz) should not be affected by the COG calculation for these kinds of images.

However, we can keep in mind that the result depends on the image dimensions. According to this result, we compute that a 100x100 image requires $9 \times 4 = 36\,\mu s\ (i.e.\ 27\,kHz)$.

As a result, for a $n \times n$ image we can extrapolate:

$$\Delta t_{COG}(\mu s) = 36 \times (n/100)^2$$

Using the formula above, we calculate 9 µs and 36 µs latencies for 50x50 and 100x100 images respectively.

In order to obtain computing frequency of 2 kHz (i.e. under $\Delta t_{COG} = 500\ \mu s$), **not taking into account other latencies**, we solve for the maximum image size, **n,** given the frequency constraint, we thus obtain the biggest possible image of : 373x373 pixels

$$n = 100\sqrt{\Delta t_{COG}(\mu s) / 36}$$

$$n = 100\sqrt{500/36}$$

$$n = 373$$

We therefore conclude that for bigger images the COG algorithm should be optimized.

By taking the inverse of Δt_{COG} we can also express and plot the frequency in Hz $f(n)$, where n is the image size in pixels.

$$f(n) = 1 / (36 \times (n/100)^2 \times 10^\wedge - 6)$$

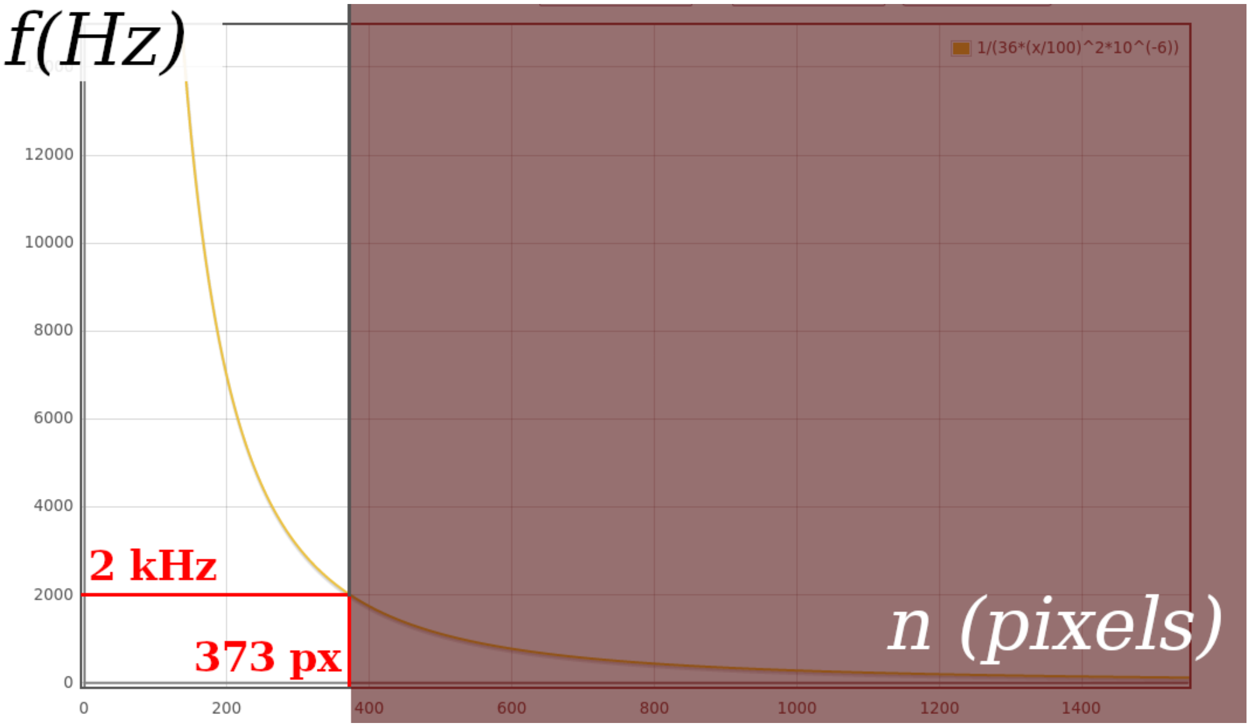


Image 13 : Plot of the possible COG frequency according to the image size (side n)

As we see on the image above, we should absolutely avoid the interval on the right side in transparent red, it means that the frequency should be lower than 2 kHz. Because $f(n)$ depends on the computer resources we indicate the computer on which we performed the calculations, mainly *Dell Latitude 7490 (CPU Intel i5 8th gen)*. It is possible to adapt the above formula to another computer, it can be done by replacing the factor 36 by the required time in µs that the COG program would take to execute a 100x100 image once.

Data transfer latency

Because we communicate between camera and DAC interface from the USB port, we perform USB latency measurements by using FT232 chip. The FT232 chip allows sending and receiving data directly to/from an USB port.

Practically, we do such measurement as follows :

1. We connect the FT232 chip module to the PC USB as indicated on the diagram below;
2. We send a high byte to the USB, e.g. with a signal generator;
3. We read the high byte with an oscilloscope;
4. We time the Δt (latency) between the send command and receive command.

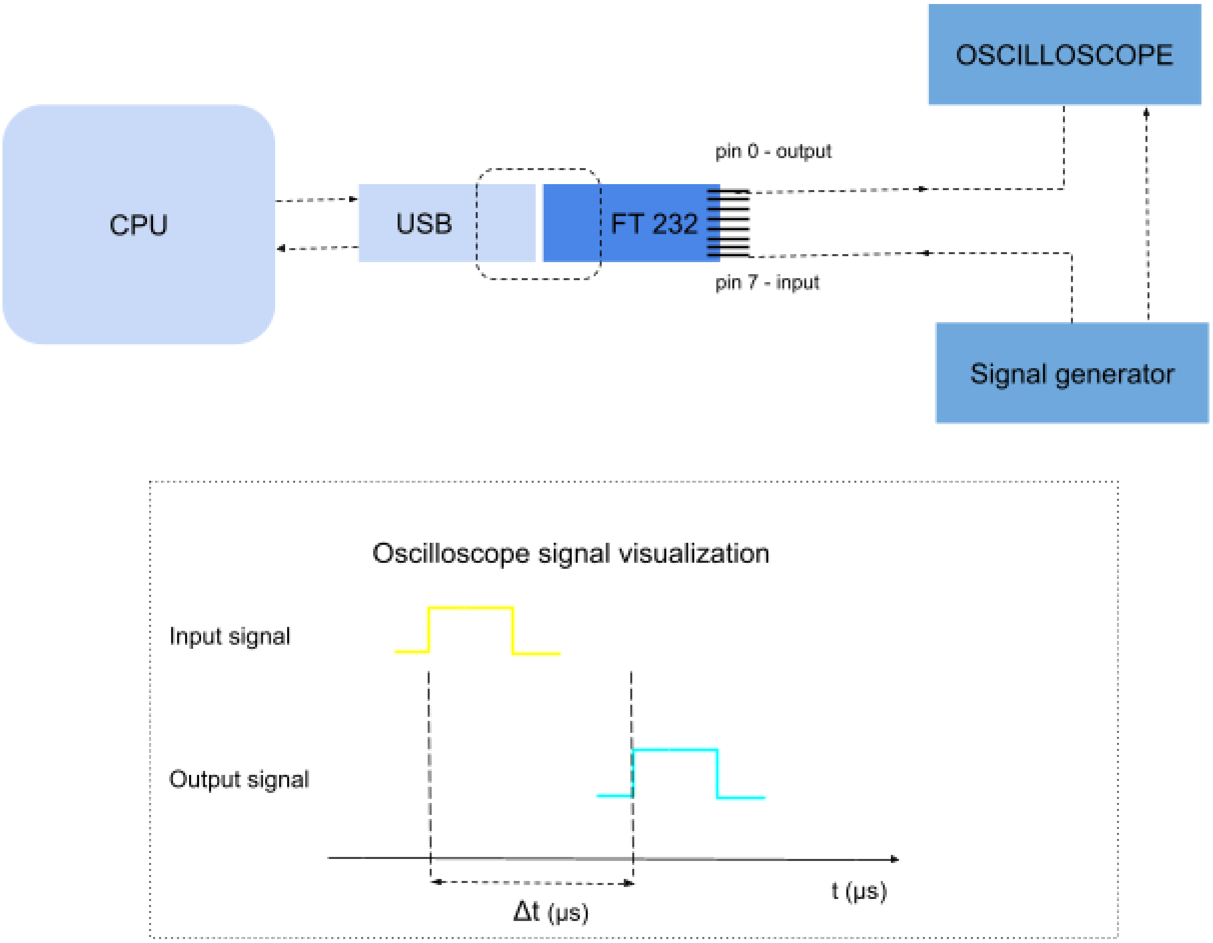
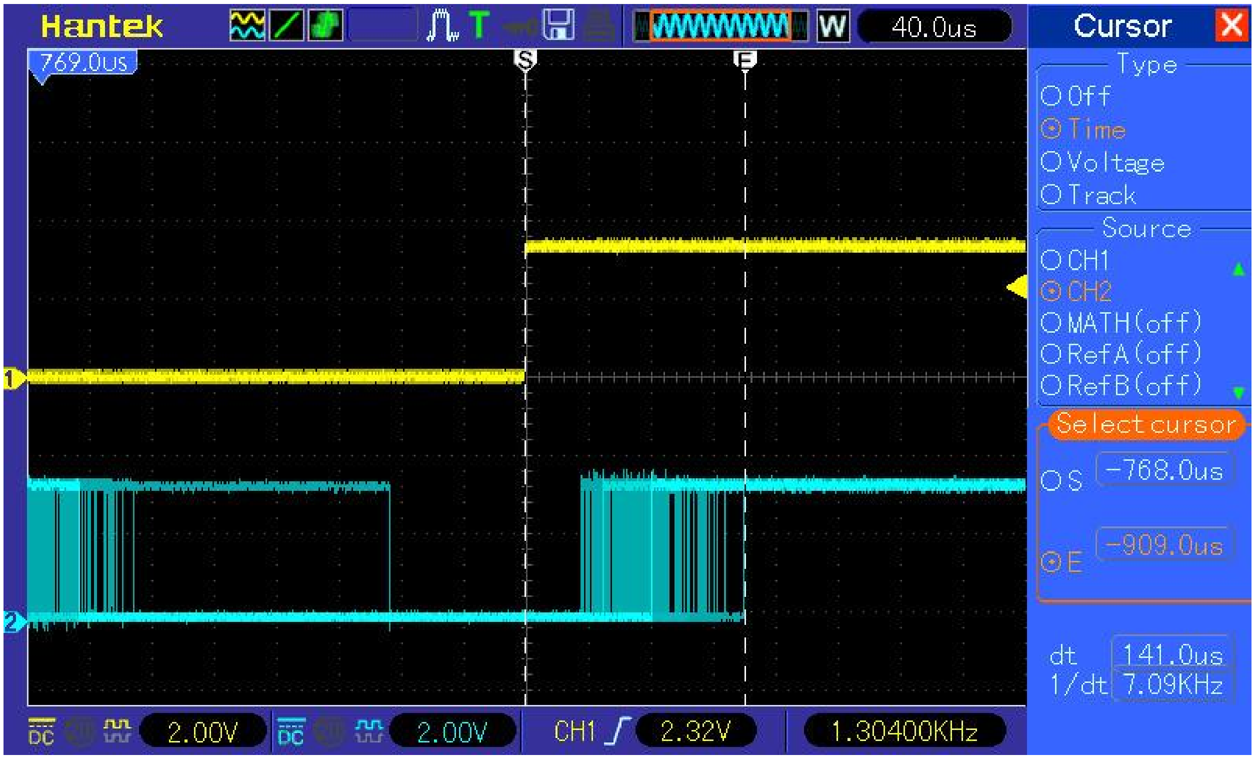


Image 14 : Data transfer latency diagram

In the image below we can see USB data transfer latency measurements. In yellow is the input signal at 2 kHz in cyan is the output signal. We observe that the worst delay time between input and output signal is $dt = 141\ \mu s$. Important to note that the dt is dependant on the OS load, for instance if an interrupt occurs during the program execution then dt increases and can reach up to $dt = 500\ \mu s$. The varying dt (consequence of the interrupts) can be seen on the CH2 output signal which was set to infinite persistency.



Type: Time

Source: CH2

S -768.0us

E -909.0us

dt 141.0us

1/dt 7.09KHz

2.00V 2.00V CH1 2.32V 1.30400KHz

Image 15 : USB data transfer latency measurement

We therefore conclude by taking an average transmission time, not the worst case time, and estimate $\Delta t_{data\ transfer} \approx 200\ \mu s$.

Total system latency

We summarize and comment on the total measured system latency in the table below.

Process	Latency time	Comment
Acquire image	7 ms (143 Hz)	This value could be improved, for example by testing an asynchronous acquisition algorithm.
Compute COG	9 µs (111 kHz)	It is not an overhead for the 2 kHz goal.
Compute pointing error	1 µs (1 MHz)	It is not an overhead for the 2 kHz goal.
USB response	200 µs (5 kHz)	USB response time depends on the system interruptions. The value could potentially reach up to 500 µs or higher.
Mirror TT movement latency	140 µs (7 kHz)	
DAC latency	N/A	The DAC has not been bought yet.
Mirror TT latency	N/A	The mirror has not been tested yet. There is a latency between the voltage reading and the mirror response.

Tests

Testing the COG

Manual tests

We begin by mentioning that the COG program and acquisition programs were developed on different computers. As a result, we use a few simulated images to test the correctness of the COG program. From several examples of light sources (presented below), a Python program *image_to_stream.py* (located in the COG_automatic_tests folder) allows to convert “png” images to simulated datastream. The *image_to_stream.py* program simulates the fact that these data came from the camera. The images are then injected into the COG program to check manually the expected results. They are the distances Δx and Δy between the COG of the light source and the reference position, which is taken here as the center of an image.

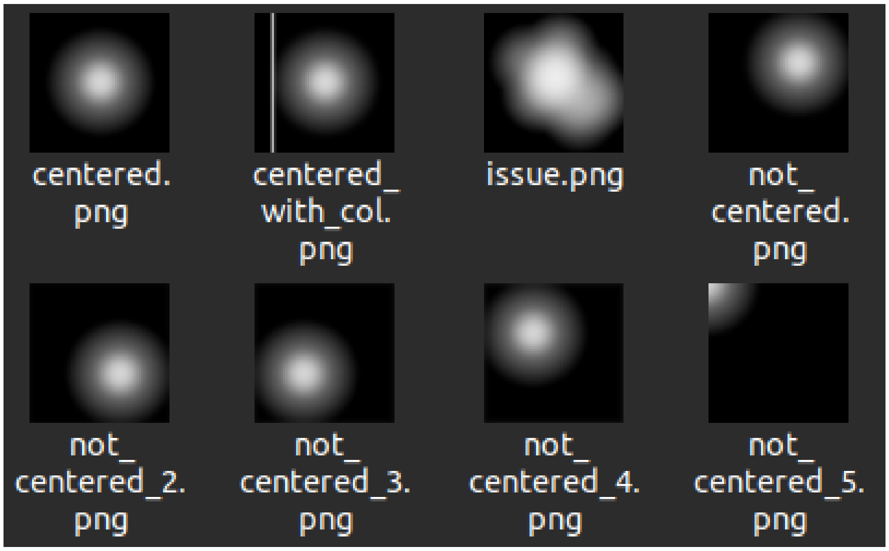
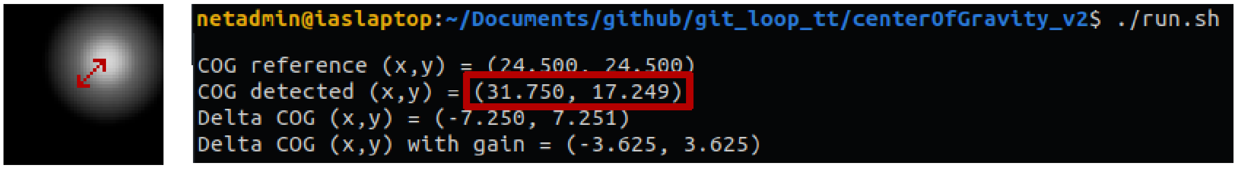


Image 16 : Simulated images to test manually the COG program

We were analyzing what would happen if the source is not a unique clear PSF. As you can see on the image above, if the PSF contains a column of luminosity (centered_with_col.png), or other issues for some reason (for example, someone removed something on the bench), we still need to define the required behavior of the algorithm. Perhaps we could ignore the flawed iteration of the loop or maybe we could turn the loop off. In both cases, we don’t want to send an absurd voltage order to the T-T mirror.

Results of manual tests:

- The *centered.png* image was drawn freehand, this is why the PSF is not perfectly centered, as expected the COG of the PSF is not zero but $(\Delta x = 0.304,\ \Delta y =- 0.304)$. We add a gain G to prevent the COG program finding too high values after a lot of iterations of the loop. Adding no gain could lead to erroneous values after several iterations which could be sent to the T-T mirror. The value of G will have to be adapted once the loop is ready to be tested. We apply a gain $G = 0.5$, the new results therefore become $(\Delta x = 0.152,\ \Delta y =- 0.152)$.



```
netadmin@iaslaptop:~/Documents/github/git_loop_tt/centerOfGravity_v2$ ./run.sh
COG reference (x,y) = (24.500, 24.500)
COG detected (x,y) = (31.750, 17.249)
Delta COG (x,y) = (-7.250, 7.251)
Delta COG (x,y) with gain = (-3.625, 3.625)
```

Image 17 : Terminal which is running the COG on not_centered.png

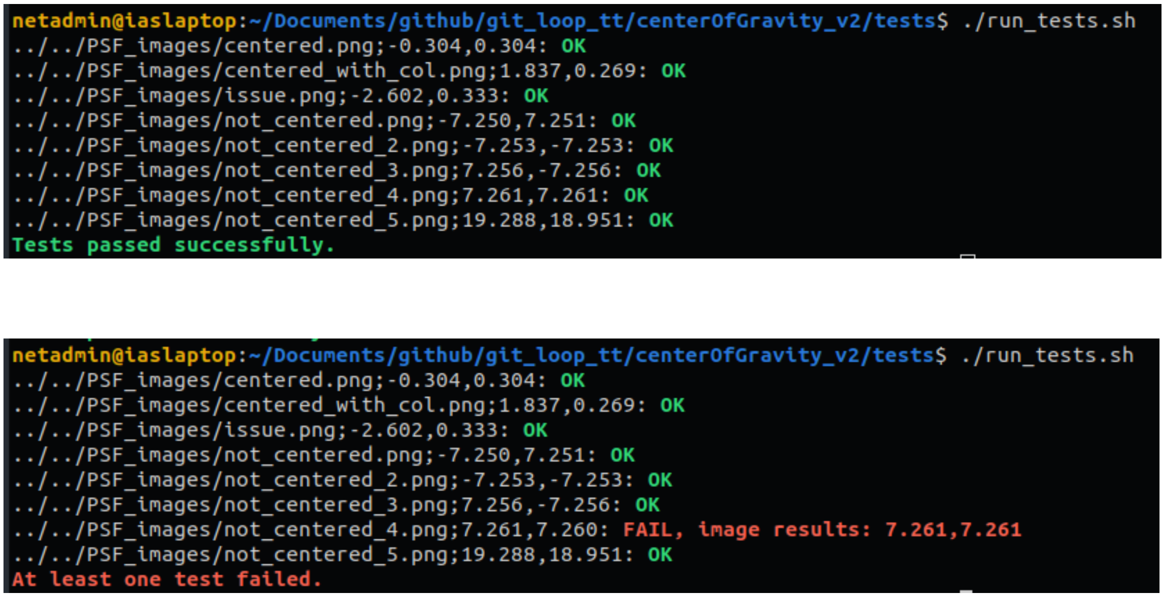
- We now inject the *not_centered.png* image to the program, it is clear that the COG of this PSF has a too high value of the COG in x direction and too low in y direction (the reference position here is $x = 24.5$, $y = 24.5$), the detected COG position is correct ($x > 24.5$ and $y < 24.5$)

Automatic tests

We also provide a possibility to test the PSFs in an automated manner. The main goal of these tests is to assure that the COG program provides relevant values in several cases. For example, after modifying the COG algorithm it could be useful to test the validity of the modification.

The automatic tests convert test images (test image are found in the test folder see Project software structure) to simulated datastream, then the COG program calculates ΔX and ΔY displacements from the reference point. The test will pass if the results are equal the expected ΔX and ΔY.

On the image below we see test outputs for the image datasets (that are found in the test folder). As per output seen on the first image all tests are passed. While changing the reference COG for the second tests the *not_centered_4.png image* hasn't passed the test.



```
netadmin@iaslaptop:~/Documents/github/git_loop_tt/centerOfGravity_v2/tests$ ./run_tests.sh
../../PSF_images/centered.png;-0.304,0.304: OK
../../PSF_images/centered_with_col.png;1.837,0.269: OK
../../PSF_images/issue.png;-2.602,0.333: OK
../../PSF_images/not_centered.png;-7.250,7.251: OK
../../PSF_images/not_centered_2.png;-7.253,-7.253: OK
../../PSF_images/not_centered_3.png;7.256,-7.256: OK
../../PSF_images/not_centered_4.png;7.261,7.261: OK
../../PSF_images/not_centered_5.png;19.288,18.951: OK
Tests passed successfully.
```

```
netadmin@iaslaptop:~/Documents/github/git_loop_tt/centerOfGravity_v2/tests$ ./run_tests.sh
../../PSF_images/centered.png;-0.304,0.304: OK
../../PSF_images/centered_with_col.png;1.837,0.269: OK
../../PSF_images/issue.png;-2.602,0.333: OK
../../PSF_images/not_centered.png;-7.250,7.251: OK
../../PSF_images/not_centered_2.png;-7.253,-7.253: OK
../../PSF_images/not_centered_3.png;7.256,-7.256: OK
../../PSF_images/not_centered_4.png;7.261,7.260: FAIL, image results: 7.261,7.261
../../PSF_images/not_centered_5.png;19.288,18.951: OK
At least one test failed.
```

Image 18 : COG tests

Anticipating other results, it is also possible to easily add images to the automatic tests. To do this, you should follow the steps :

- Create the required image
- Add a line, in the tests_list.txt file, with the syntax: image_path;expected Δx,expected Δy (note that there is a semicolon and a comma)

See the Program installation and execution section to know how to run the tests.

Project software structure

```
THD2_loop
├── Common - to look into if further developing program
│   ├── acquisition.h - contains acquisition structs
│   ├── camera.h      - contains camera related structs
│   ├── config.h      - contains typical enhanced data types structs
│   ├── dac.h         - contains DAC related structs
│   ├── msg_queue.c   - contains message queue init/rx/tx functions
│   ├── msg_queue.h   - contains messages : GUI_msg and ACQ_msg data structures
│   ├── cog.c         - contains algorithm to compute cog and other data processing
│   └── cog.h         - contains cog related structs
├── Other - contains automatic COG tests scripts
│   ├── COG_automatic_tests - contains automatic COG tests scripts
├── README.md - contains general software description
├── run.sh - IMPORTANT - compiles and runs acquisition and visualization programs
├── VimbaC - contains camera interface software
│   ├── Camera_settings - contains camera settings
│   ├── DynamicLib - not important, contains VimbaC software dependencies
│   └── Examples
│      └── SynchronousGrab - contains synchronous acquisition software
│          ├── Build
│          │   └── Make
│          │       └── Makefile - make if software is changed
│          └── Source  - contains acquisition software
│              ├── program.c
│              ├── SynchronousGrab.c - to look into if further developing program
│              └── SynchronousGrab.h
└── Visualization - contains visualization/loop control software
    ├── Install
    ├── libs
    ├── Othe
    ├── RX_TX_data - to look into if further developing program
    └── visualization.py - to look into if further developing program
```

Program installation and execution

To install and run the program in the terminal type the following:

```
cd/THD2_loop

chmod +x setup_vimba.sh
./setup_vimba.sh

chmod +x setup_visualization.sh
./setup_visualization.sh

chmod+x run.sh
./run.sh
```

This will install all dependencies if needed then it will launch the acquisition and the visualization programs.

Automated tests

To run the automatic tests:

1. Go to *Other/COG_automatic_tests*
2. Read README.md
3. Run *./run_tests.sh*

Planning

Task	Assigned to	Deadline	Status
Add the Gantt chart in the related section	davidpa.p…	Oct 27, …	Approved
Add a diagram of the loop in the “About project” section	davidpa.p…	Oct 27, …	Approved
Define the visualization requirements	davidpa.p…	Oct 27, …	Approved
Software sub-system block diag.	Marcin K…	Oct 28, …	Approved
Retrieve the datastream : begin	Marcin K…	Oct 28, …	Approved
Compute COG and gain : begin	davidpa.p…	Oct 28, …	Approved
Mako_Ethernet_configuration_linux	Marcin K…	Oct 28, …	Approved
Create a git environment to share code	davidpa.p…	Oct 31, …	Approved
Retrieve the datastream : end	Marcin K…	Nov 3, 2…	Approved
To pass the datastream to the COG : begin	Marcin K…	Nov 3, 2…	Approved
To pass the datastream to the COG : end	Marcin K…	Nov 3, 2…	Approved
Define Image visualization (STB)	davidpa.p…	Nov 10, …	Approved
Implement Image visualization prototype	davidpa.p…	Nov 25, …	Approved
Latency estimation: USB data transfer	Marcin K…	Dec 8, 2…	Approved
Define the coding style Google Coding style	davidpa.p… Marcin K…	Dec 16, …	Approved
System block diagram	Marcin K…	Oct 27, …	Approved
Refactor image acquisition code in C	Marcin K…	Dec 23, …	Approved
Refactor all code into a single codebase	Marcin K…	Dec 23, …	Approved
Update GIT : remove the clutter	Marcin K…	Dec 28, …	Approved
Update “About **the** project” section	davidpa.p…	Nov 3, 2…	Approved

Task	Assigned to	Deadline	Status
Compute COG, Δx, Δy and gain : end	davidpa.p…	Nov 3, 2…	Approved ▾
Unit test datastream	Marcin K…	Nov 6, 2…	Approved ▾
Unit test COG	davidpa.p…	Nov 6, 2…	Approved ▾
Develop automatic tests for COG	davidpa.p…	Nov 25, …	Approved ▾
IPC MSGQ : acquisition	Marcin K…	Dec 6, 2…	Approved ▾
IPC MSGQ : fix N image acquisition	Marcin K…	Dec 22, …	Approved ▾
Image visualization solution	Marcin K…	Dec 23, …	Approved ▾
IPC MSGQ : visualization Python side	Marcin K…	Dec 25, …	Approved ▾
Threads : implement threads Python	Marcin K…	Dec 28, …	Approved ▾
Tests COG with real images	Marcin K…	Dec 28, …	Approved ▾
Report redaction : Finish	Marcin K… davidpa.p…	Jan 2, 2…	Approved ▾
Visualization and program integration	Marcin K…	Dec 15, …	Approved ▾
Rewrite the code according to the Google Coding style	Marcin K…	Dec 28, …	Approved ▾
IPC MSGQ : acquisition C side	Marcin K…	Dec 23, …	Approved ▾
Integration test Datastream & COG	Marcin K…	Nov 8, 2…	Approved ▾
Propose DAQ solution	Marcin K… davidpa.p…	Oct 28, …	Discontinued ▾
Validate DAQ solution	Marcin K… davidpa.p…	Oct 30, …	Discontinued ▾
Order DAQ solution	Marcin K… davidpa.p… pierrebau…	Oct 31, …	Discontinued ▾
DAC integration 1	Marcin K… davidpa.p…	Nov 25, …	Discontinued ▾
Latency estimation: Image acquisition	Marcin K…	Dec 9, 2…	Discontinued ▾

Task	Assigned to	Deadline	Status
Total latency DAC proposal	Marcin K…	Dec 9, 2…	Discontinued
DAC integration 2	Marcin K… davidpa.p…	Dec 2, 2…	Discontinued
Integration test Program Control & DAC	Marcin K… davidpa.p…	Dec 9, 2…	Discontinued
Describe synchronous vs asynchronous acquisition	Marcin K…	Jan 2, 2…	Optional
Develop partial P[ID] controller	davidpa.p…	Dec 30, …	Optional
Write camera integration documentation	Marcin K…	Dec 28, …	Under review
Write code documentation : acquisition	Marcin K…	Dec 28, …	Under review
Write code documentation : visualization	Marcin K…	Dec 28, …	Under review
Fix image acquisition bug : offset	Marcin K…	Dec 23, …	Under review
Acquisition loop optimization : rewrite	Marcin K…	Dec 28, …	Under review
Threads : implement threads C	Marcin K…	Dec 28, …	Under review

Deliverables

Deliverable	Delivery date	Status
Report THD2-Project	Jan 2, 2023	Approved
Reference documents	Jan 6, 2023	Under review
Source code	Jan 6, 2023	Under review
Source code documentation, including: - User guide	Jan 6, 2023	Under review

Other

Meetings

With : Denis Perret (electronician), on Nov 4, 2022 11h00, online
Topic : DAC choice specifications

Meeting summary

The main meeting idea was to discuss the DAC Card requirements as given per Loop requirements (LR). According to Denis, when choosing the DAC the most relevant parameters are :

- Sampling frequency
  - See DAC_memo
- System latency, which is due to the:
  - FIFO and buffers of the DAC (the bigger the FIFO buffers the greater is the latency)
  - Processing (computer) latency
  - Data transmission latency
- System jitter
- Slew rate of the DAC

Among the parameters above, the system latency is a parameter of the highest importance. The response speed of a DAC depends on the general system latency (also known as throughput), i.e. latency of the transmission chain (our case: camera → computer → DAC).

During the meeting we discussed technical specifications (provided in the data sheets [Card solution proposition]) of DAC N^{0}3 and N^{0}8. Generally speaking, both of the DAC's meet the Loop requirements (LR). However, contrary to the DAC N^{0}8, DAC N^{0}3 specifies Analog Output throughput (the parameter that is of the most interest to us). On the bottom of page 4 of the DAC N^{0}3 datasheet we see that Analog Output throughput Software paced is : 33 S/s to 4000 S/s typ, system-dependent. It means that the DAC manufacturer doesn't give specific details regarding the specific system latency. Precisely because the throughput is system-dependent. In other words, it is impossible to know the real system latency without buying the DAC and then testing its speed on the overall system (camera → computer → DAC). Therefore, the system may or may not run @2kHz. Indeed, before committing purchase of a DAC we want to make sure that the system throughput lies in between the range 33 S/s to 4000 S/s. To do so, we want to measure the system throughput time. One way to do it is to connect to the computer USB a device (instead of the DAC) that could “see” the transmitted data. Then we would like to measure the time it takes for the system to :

> capture image (camera) →acquire image (camera) → process image (camera) → transmit image to the computer → process image data (computer) → send the data to the USB port (computer).

Such measurements can be done with devices that Denis has at his laboratory at the OBSPM. Mainly, it can be either of the following devices FT232 FTDI Mini USB to TTL Serial UART or Analog Discovery 2. With the Analog Discovery 2 it’s possible to measure the throughput time or frequency and also to use its internal 14 bit DAC.

Meeting outcome:

This meeting has led to the agreement of the follow-up meeting on the Nov 18, 2022 at OBSPM at 11h00 AM. The goal of the next meeting is to connect the Analog Discovery 2 board to the system and to measure the throughput time.

With : Denis Perret (electronician), on Nov 18, 2022 11h00, in-person
Topic : USB latency specifications

Meeting summary

The main objective of the meeting was to specify how to perform system USB latency measurements. In particular, we chose to use the FT232 chip. The FT232 chip allows sending/receiving data directly to/from an USB port. In addition FT232 chip is open-source and costs only 2$. On how to measure USB latency (see Latency estimation: USB data transfer).

Conclusion

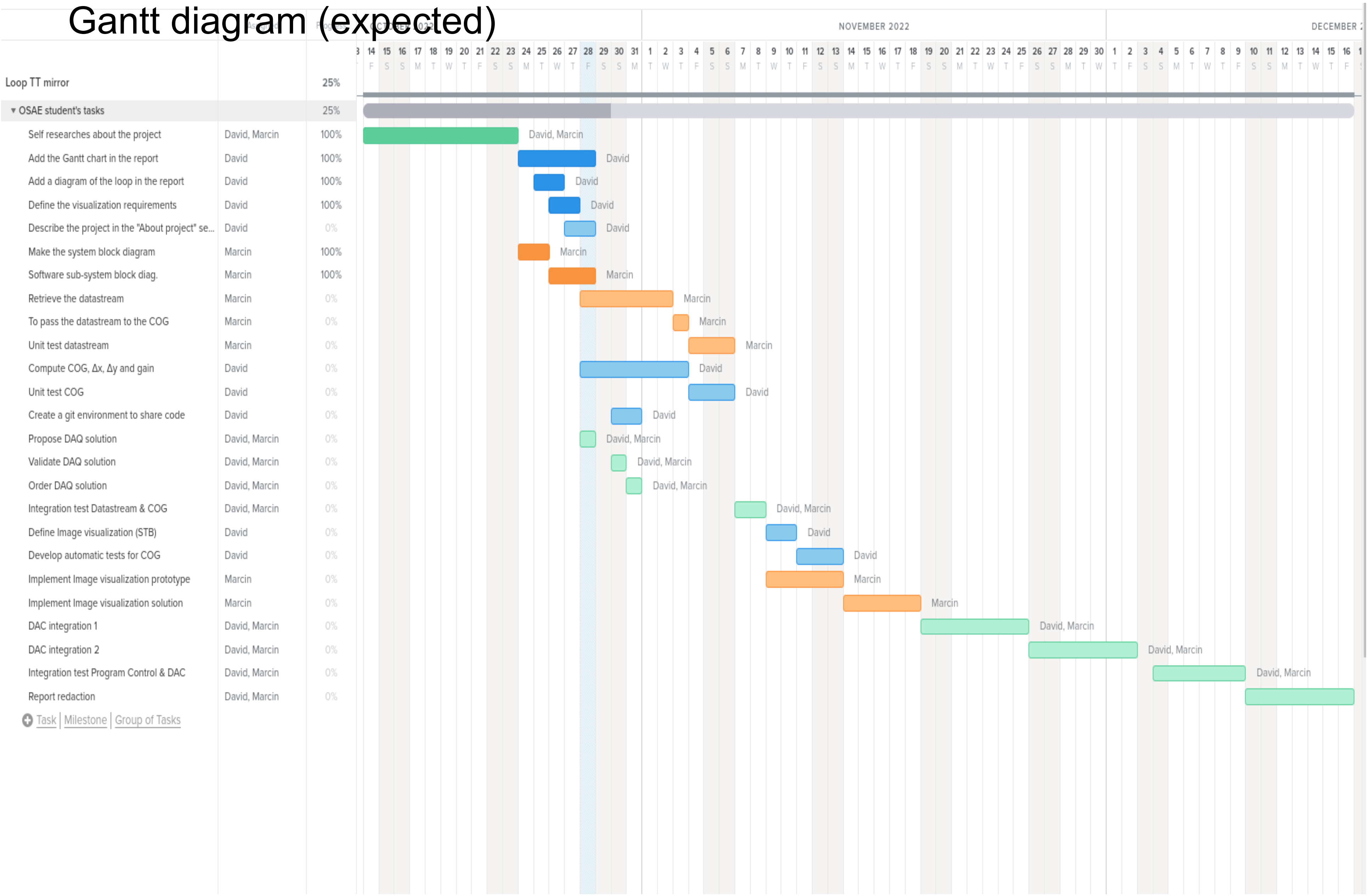
The main project idea was to implement a closed loop tip-tilt mirror solution to correct the vibration induced PSF pointing error. In the beginning the project ambitions were big, however as the project advanced and some issues occurred we had to re-think the project planning and change the course of the action accordingly. Regarding the hardware side, we haven't ordered the DAC solution because we had to perform latency measurements. Especially the USB latency measurements that took us at least two weeks. However, regarding the software side we consider accomplishing a great job. Mainly, we develop a functioning program that is able to interface with the camera, acquire the image and to send the acquisition information back to the user via the visualization/loop control GUI program.

In the summary, we would like to thank Mr. Pierre Baudoz and Mr. Denis Perret for their ample support regarding technical project issues.

References

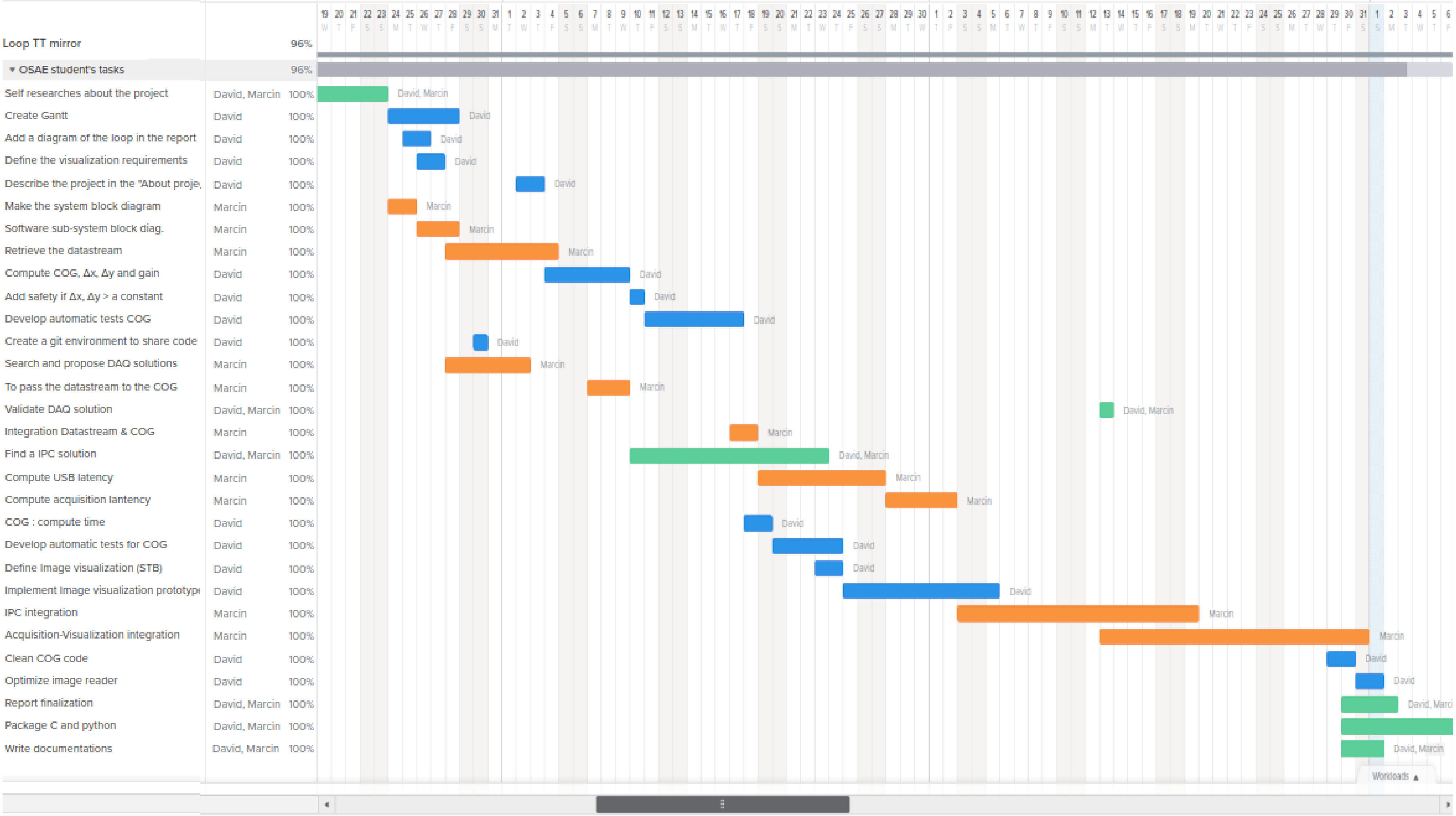
N0	Reference
R1	Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Ninth Edition, Willey, 2013, USA, ISBN:9781118063330

Gantt diagram (expected)



Loop TT mirror		25%
▾ OSAE student's tasks		25%
Self researches about the project	David, Marcin	100%
Add the Gantt chart in the report	David	100%
Add a diagram of the loop in the report	David	100%
Define the visualization requirements	David	100%
Describe the project in the "About project" se...	David	0%
Make the system block diagram	Marcin	100%
Software sub-system block diag.	Marcin	100%
Retrieve the datastream	Marcin	0%
To pass the datastream to the COG	Marcin	0%
Unit test datastream	Marcin	0%
Compute COG, Δx, Δy and gain	David	0%
Unit test COG	David	0%
Create a git environment to share code	David	0%
Propose DAQ solution	David, Marcin	0%
Validate DAQ solution	David, Marcin	0%
Order DAQ solution	David, Marcin	0%
Integration test Datastream & COG	David, Marcin	0%
Define Image visualization (STB)	David	0%
Develop automatic tests for COG	David	0%
Implement Image visualization prototype	Marcin	0%
Implement Image visualization solution	Marcin	0%
DAC integration 1	David, Marcin	0%
DAC integration 2	David, Marcin	0%
Integration test Program Control & DAC	David, Marcin	0%
Report redaction	David, Marcin	0%

Gantt diagram (final)



Loop TT mirror		96%
▾ OSAE student's tasks		96%
Self researches about the project	David, Marcin	100%
Create Gantt	David	100%
Add a diagram of the loop in the report	David	100%
Define the visualization requirements	David	100%
Describe the project in the "About proje	David	100%
Make the system block diagram	Marcin	100%
Software sub-system block diag.	Marcin	100%
Retrieve the datastream	Marcin	100%
Compute COG, Δx, Δy and gain	David	100%
Add safety if Δx, Δy > a constant	David	100%
Develop automatic tests COG	David	100%
Create a git environment to share code	David	100%
Search and propose DAQ solutions	Marcin	100%
To pass the datastream to the COG	Marcin	100%
Validate DAQ solution	David, Marcin	100%
Integration Datastream & COG	Marcin	100%
Find a IPC solution	David, Marcin	100%
Compute USB latency	Marcin	100%
Compute acquisition lantency	Marcin	100%
COG : compute time	David	100%
Develop automatic tests for COG	David	100%
Define Image visualization (STB)	David	100%
Implement Image visualization prototyp	David	100%
IPC integration	Marcin	100%
Acquisition-Visualization integration	Marcin	100%
Clean COG code	David	100%
Optimize image reader	David	100%
Report finalization	David, Marcin	100%
Package C and python	David, Marcin	100%
Write documentations	David, Marcin	100%