INTERNSHIP REPORT

LabVIEW implementation of 3D tracking of particle trajectories

GUO RAN

JULY 31, 2021

Polytech Nice Sophia

Polytech'Lab

This project is initiated by Polytech'Lab.

Polytech'Lab is the Polytech Nice-Sophia school's research lab, funded in 2017 from fusing I-City and EpOC.

The main research activities of Polytech'Lab, focused on the socio-economic world and industrial transfer, are focused on the issue of energy, water and risk management around the concept of Smart building and smart Cities, with the use of ICT (Information and Communication Technologies) The strategy of Polytech'Lab is based on three pillars:

(i) an ambitious and disruptive scientific vision;
(ii) a privileged relationship with the socio-economic world, whether industrialists or local and regional authorities;
(iii) a very strong synergy between education, research and innovation.

The main team is composed of 13 teacher-researcher, 13 associates and 1 administrative clerk.

Gilles Jacquemod is the director of Polytech'Lab.

This document was written by *Ran GUO*, a trainee from the Electronical Engineer department of Polytech Nice-Sophia, under the supervision of *Eric Dekneuvel*, a teacher-researcher of Polytech'Lab.

Acknowledgements

I would like to specially thank professor Eric Dekneuvel, my supervisor, for his constant encouragement and guidance. He has walked me through all the stages of writing of this report. Without his consistent and illuminating instruction, this report could not have reached its present form.

I would also like to thank professor Gilles Jacquemod. I was warmly welcomed by his research team in Polytech'Lab where I stayed these months. He is the director of Polytech'Lab. I could not have gotten this far in this work without his help.

I am also deeply grateful professor Pascal Biwole of the University Clermont Auvergne. He helped me solve a lot of academic problems and friendly explained me the algorithm of 3D PTV.

Finally, I would like to express my thanks to all the other friends who help me a lot during this internship.

Abstract

In a building, knowing the trajectory and the speed of air or the speed of the particles vehiculated by the air becomes strategic today, both for energetical and for environment purposes. On the energetical side, the aim is to understand the nature and the dynamics of the aeraulic transfers to be able to optimize the air quality. On the environment side, the aim is to predict the displacement of aerosols or dangerous gas in a closed space to optimize the ventilation command or for improving evacuation procedures. Since about twenty years, several research teams try to get the properties of the fluid using Lagrangian measurement techniques rather than using a fixed point. The aim is to track neutral density particles flowing in the fluid. The more the number of immerged particles, the better we can understand the variations, the trajectories, and the structure of the fluid. The Lagrangian methods lead to the measurement of individual trajectories of the particles, and, more specifically, can use the tridimensional velocimetry approach by particle tracking (3D *particle tracking velocimetry or 3DPTV*).

Developing an instrument for particle tracking is not an easy task. To fully integrate the instrument in its environment, additional services are usually expected like calibration, configuration functions and the design of such a smart device requires an embedded computer system inside. Such an embedded product undergoes a wide range of processes — designing the architecture, developing the platform with programming language and tools, integrating processors, peripherals, and software and lastly, testing compliance and functioning. Today, model-based Design is transforming the way engineers and scientists work by moving design tasks at a higher level than the software and hardware generation to improve product quality and highly reduce development time.

In the rest of the paper, we address the problem of the rapid prototyping of complex smart instrument through the implementation of a high-level model of the PVT instrument using the LabVIEW language. First, we develop the formal description of the instrument using the generic formal model. In the next section, we talk about the transcription of this model using the LabVIEW language for functional validation of the instrument. Then, we talk about an internal model for performance evaluation and design optimization of the instrument. This internal logical model is mapped on a platform made of multiple processors (hardware and software) coupled by a PCI bus. Last, we talk about experimental we have got thanks to HLS and code generation technics on National instrument devices.

Acronym List

CFD	Computational Fluid Dynamics
MPTV	Multi Particle Tracking Velocimetry
PCI	Peripheral Component Interconnect Bus
PCIe	PCI Express
PIV	Particle Image Velocimetry
PTV	particle Tracking Velocimetry
USOM	USer Operating Modes

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Large-scale 3D PTV is a key asset when studying large-scale turbulent thermal convection in rooms. The local properties of velocity and temperature fields around heat sources are still poorly known, in spite of the widespread occurrence of the phenomenon. The hot-wire anemometers which are frequently used are intrusive and give a point-wise measurement with large errors for low ascendant airflows since the probes create their own convection. Stereoscopic particle image velocimetry yields instantaneous field-wise 3D velocity vectors only inside thin laser sheets. The experimental data retrieved from large-scale 3D PTV are crucial for air quality engineering when designing ventilation strategies or monitoring airborne pollutants dispersion in inhabited spaces as well as in livestock compounds. The measured 3D velocity vector fields help validate and improve CFD codes dedicated to indoor airflow simulation. Longer tracks help identify coherent structures in steady flows and provide statistical trajectories in unsteady flows.

Biwole et al (2008) used eight 1000 W compact fluorescent lamps to illuminate millimetric helium-filled soap bubbles in a 1 m3 volume. Their spatial matching was based on an algorithm comparing the 3D coordinates calculated by two cameras at a time. The speed of their cameras allowed the use of a normalized 2D cross-correlation scheme to perform tracking in object space. Tracking conflicts were resolved by minimizing the Euclidian distance from an extrapolated 2D estimated position created from three previous frames through linear regression. 3D reconstruction was performed after each time step using a least-squares method on the set of equations generated by the multiple views. Their method gave good results in low-density cases, but many trajectories were mixed in high-density cases where mean particle spacing to mean displacement ratios ξ were less than 3.75. Moreover, 3D trajectory tracking could not be initiated unless the particle was seen by at least three cameras, thus reducing the initial field of measurement.

The internship, offered by polytech lab, aims to develop the image processing chain to calculate the pixel coordinates of particles present in successive images from three cameras. This chain is mainly composed of the following operations : subtracting background images, filtering, calculation of the pixel coordinates and tracking. The LabVIEW software will be used to develop the application model and automate the implementation.

LabVIEW is a system-design platform and development environment for a visual programming language from National Instruments. This programming language is compiled and is well suited for dataflow behavior.

In the section 2, we develop the formal description of the instrument using the generic formal model. In the section 3, we talk about the transcription of this model using the LabVIEW language for functional validation of the instrument. Then in the section 4, we talk about an internal model for performance evaluation and design optimization of the instrument. This internal logical model is mapped on a platform made of multiple processors (hardware and software) coupled by a PCI bus. And in the section 5, we talk about experimental we have got thanks to HLS and code generation technics on National instrument devices.

2. Description

We used helium-filled bubbles as particles. In 3D PTV system, we use 3 cameras to capture photos, then we develop the image processing chain to calculate the pixel coordinates of particles present in successive images from these three cameras.

You can find below (Figure 1) MPTV system which has two 3D PTV systems of 3 cameras each. This system has 6 cameras to catch bubbles generated by Bubble generator. Then transfer the images to PTV algorithm processor to do the calculation.

In my internship project, we use 3 cameras to develop the image processing chain.

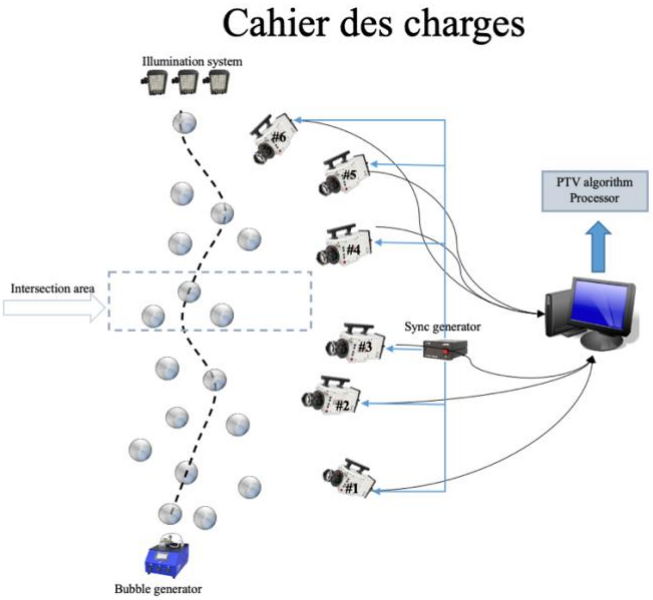


Figure 1. MPTV system

The algorithm of 3D PTV system is shown as Figure 2. It mainly composed 4 operations: *particle detection*, *temporal tracking*, *spatial matching* and *3D reconstruction*.

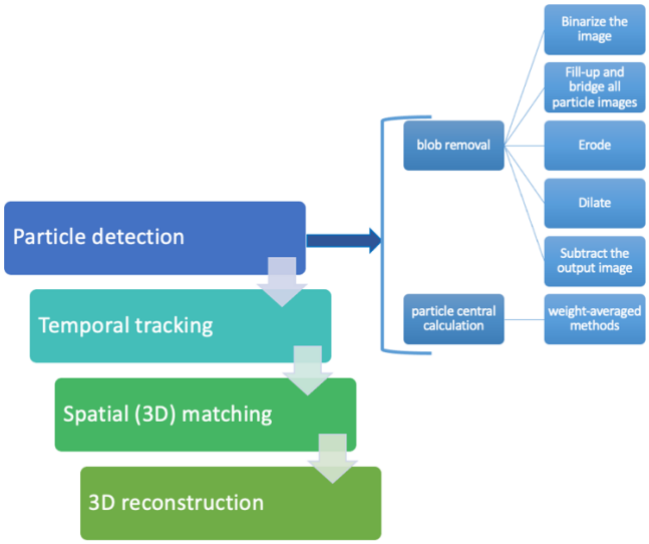


Figure 2. 3D PTV system algorithm

2.1 particle detection

In the first part, *particle detection*, it can be divided in to 2 sections, *blob removal* and *particle central calculation.*

The term *'blob'* refers to over-large bubble images. PTV algorithms for large volumes with small distance of the cameras from the measuring volume must include a step where oversized particles are removed from the images. Those blobs are created by images of bubbles getting close to the cameras, since particles are not constrained to remain inside a delimited small volume.

oversized helium-filled bubbles can be removed efficiently by the following procedure:

- **Binarize** the image.
- **Fill-up and bridge** all particle images. (To get homogeneous blobs)
- **Erode** the output image with a square structuring element. (To remove the correct sized ones)
- **Dilate** the resulting image with a square structuring element. (To get a good coverage during the subtraction step)
- **Subtract** the output image.

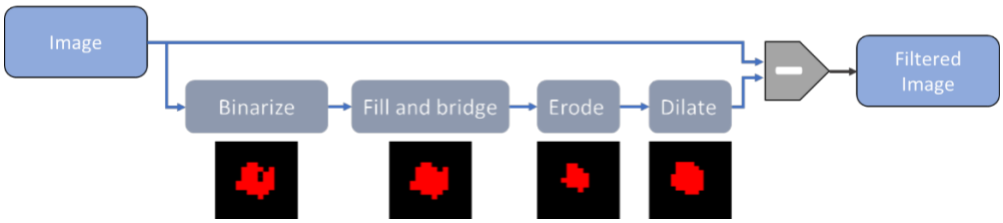


Figure 3. blob removal

The method used by particle central calculation is *weight-averaged methods*. The coordinates (x_c , y_c) of the center of mass are given by:

$$X_c = \frac{\sum x\, I(x,y)}{\sum I(x,y)}, \quad Y_c = \frac{\sum y\, I(x,y)}{\sum I(x,y)}$$

- (x, y) are the pixel coordinates of each pixel belonging to the particle
- $I(x, y)$ is the pixel luminance

2.2 Temporal tracking

Temporal tracking describes the research of particle trajectories on successive 2D particle images. There is no attempt to match particles in 3D (unless needed to resolve ambiguities) at this time of the process. Therefore, temporal tracking can be made separately, one camera at a time.

There are two methods of temporal tracking: *Modified fast normalized cross-correlation tracking* and *polynomial regression.*

2.2.1 polynomial regression

We divide the algorithm in three steps depending on the number of particle matched.

- First particle: initialize the first position.
- Second and third particle: match the next particle using the *nearest neighbor algorithm*.
- Starting from the fourth particle: match the next particle using a *second order polynomial fit*.

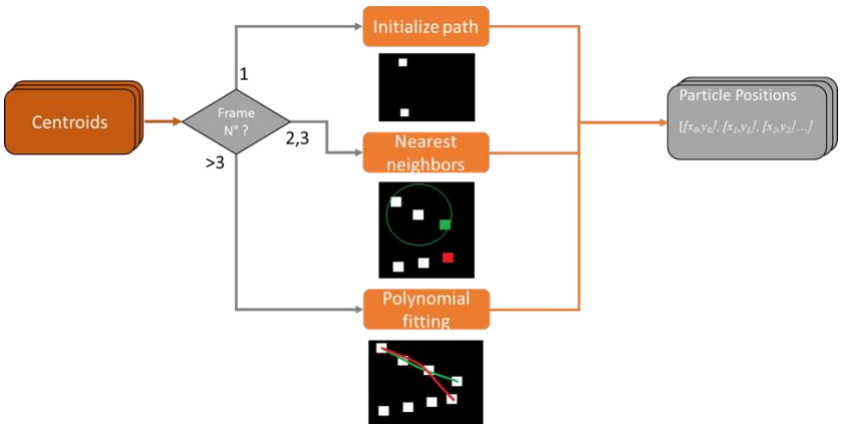


Figure 4. Polynomial regression

2.2.2 Modified fast normalized cross-correlation tracking

This temporal tracking method is partly inspired from traditional PIV cross-correlation methods. However, instead of looking at the next frame for a pattern composed of a group of particles, the pattern is a single particle. Besides, there is a procedure to solve ambiguities. The template matching criteria taken into account are *size*, *shape* and *luminance* of the particle.

2.3 Spatial matching (3D matching)

The method we used to realise 3D matching is *Stereo pair matching.* The stereo pair matching or spatial matching is first done using a three-camera arrangement.

Two 2D trajectories are considered matched if six pairs of time-synchronous points, one in each trajectory, can be found verifying the equation below.

$$[x_1^t\ y_1^t\ 1]\ F12 \begin{bmatrix} x_2^t \\ y_2^t \\ 1 \end{bmatrix} < s$$

- $(x_1^t\ y_1^t)$ and $(x_2^t\ y_2^t)$ are normalized pixel coordinates from each trajectory.
- t is an instant time (t = 1, . . . , 6).
- s is a threshold value. (s is generally given the value 1).

You can find the algorithm of 3D matching of CAM1 & CAM2 in Figure 5.

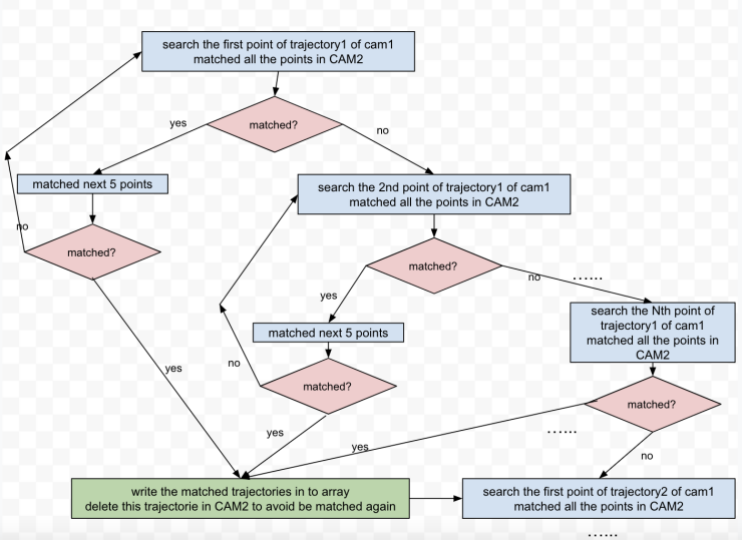


Figure 5. algorithm of 3D matching of CAM1 & CAM2

At first, each trajectory is matched using all three fundamental matrices. The remaining unmatched trajectories are then matched using only one fundamental matrix. Those trajectories come from particles whose displacement is seen by only two cameras. After these two processes, the remaining unmatched trajectories are discarded.

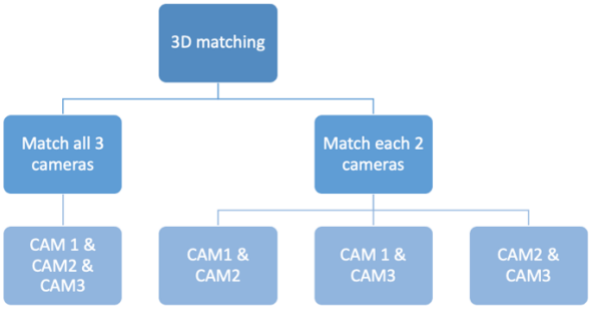


Figure 6. 3D matching

After 3D matching, we got the trajectories shown in each camera. The next step is 3D reconstruction.

2.4 3D reconstruction

This is the final step to finding 3D coordinates of a point from multiple 2D views.

We know the relationship between the reference frame of one camera and the normalized projection on the camera image plane, it can be written as below.

$$X_n = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} Xc/Zc \\ Yc/Zc \end{bmatrix}$$

- (*X*c, *Y*c, *Z*c) : The coordinates in the reference frame of one camera.
- **x**n = (*x*, *y*): The normalized projection of P on the camera image plane.

The relation between each camera 3D reference frame *XXc* (*Xc*, *Yc*, *Zc*) and the calibration target 3D reference frame *XX* (*X*, *Y*, *Z*) shown below:

$$XXc_i = R_i \cdot \mathrm{XX} + T_i$$

- T_i is respectively the 3 × 1 translation matrix.
- R_i is respectively the 3 × 3 rotation matrix.
- *XXc*i: camera *i* 3D reference frame
- XX: calibration target 3D reference frame

From these 2 equations above, it can be rewritten as:

$$\begin{cases} x_i = \dfrac{R_{11}^i X + R_{12}^i Y + R_{13}^i Z + T_1^i}{R_{31}^i X + R_{32}^i Y + R_{33}^i Z + T_3^i} \\ y_i = \dfrac{R_{21}^i X + R_{22}^i Y + R_{23}^i Z + T_2^i}{R_{31}^i X + R_{32}^i Y + R_{33}^i Z + T_3^i} \end{cases}$$

- (xi , yi) are the normalized pixel coordinates of the particle on camera i.
- X, Y, Z are the real-world 3D particle coordinates.

With *i* = [1 . . . *n*] (*n* cameras), this equation gives rise to an overdeterminated system of 2*n* equations for only three unknowns which is solved by a least-squares method.

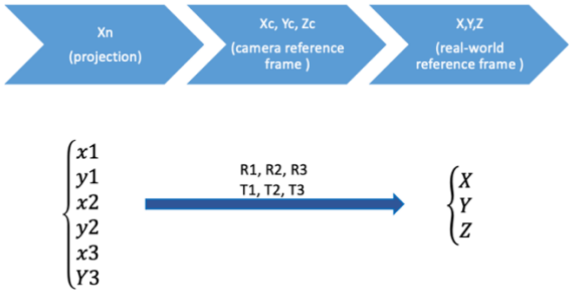


Figure 7. 3D reconstruction

If we have 3 cameras, we can get 4 or 6 equations (3D matching of 2 cameras or 3 cameras) to get 3 unknowns X, Y, Z.

3. LabVIEW implementation

LabVIEW is a system-design platform and development environment for a visual programming language from National Instruments. This programming language is compiled and is well suited for dataflow behavior.

3.1 Objective

The system needs to run in a real-time environment. The latency needs to be less than 30s and the processing speed needs to be 100 fps, with 2048x2048 pixels images.

The implementation solution is shown as Figure 8, we use 3 cameras to do the acquisition, then we use 3 FPGA cards to *detect the central of particles*. Next transfer the coordinates to PC windows to realize *temporal tracking*, *3D matching* and *3D reconstruction*. At last, we can get the real-world particle coordinates.

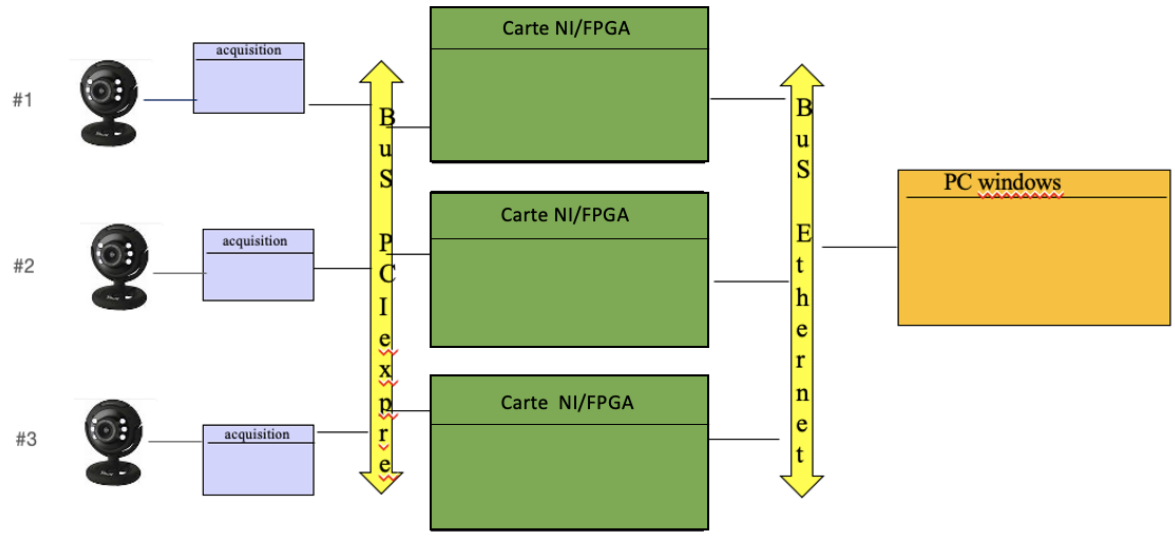


Figure 8. implementation solution

In my internship, the software I used is LabVIEW, the FPGA card is NI PCIe 1477. We divided this project to 2 parts:

- *Particle detection*: LabVIEW FPGA
- *temporal tracking*, *3D matching* and *3D reconstruction:* LabVIEW Windows

3.2 LabVIEW FPGA Implementation

3.2.1 FPGA card NI PCIe 1477

We use the PCIe 1477 to realise image acquisition and coordinate calculation, the PC realizes the *temporal tracking* and *3D matching.*

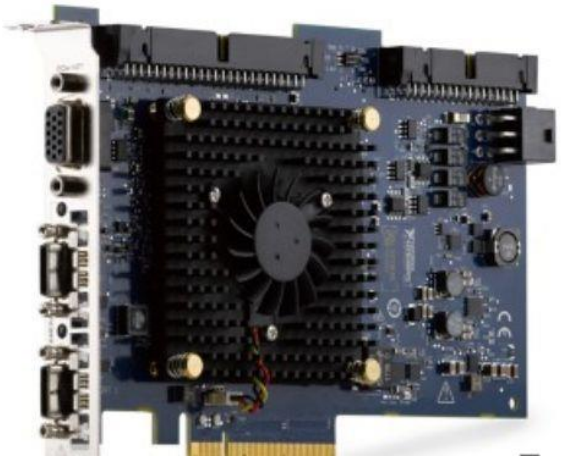


Figure 9. FPGA NI PCIe 1477

This table show us the Device utilization of FPGA card *NI PCIe 1477*.

Device Utilization	Total
Total Slices	50950
Slice Registers	407600
Slice LUTs	203800
Block RAMs	445
DSP48s	840

Table 1. Device utilization of FPGA card NI PCIe 1477

At first, we load the images to the board, then we process the images and get the coordinates of each particle on the FPGA, next, we send the coordinates to the PC by using FIFO. At last, We deal with the data and draw the tracking image on the computer.

3.2.2 Implementation transfer images

At first, we test with just image transfer. We send the image from the PC to the board, then we receive it from the board. The program is divided into 2 parts: FPGA VI and HOST VI. The image below is the FPGA VI which realizes the image transfer.

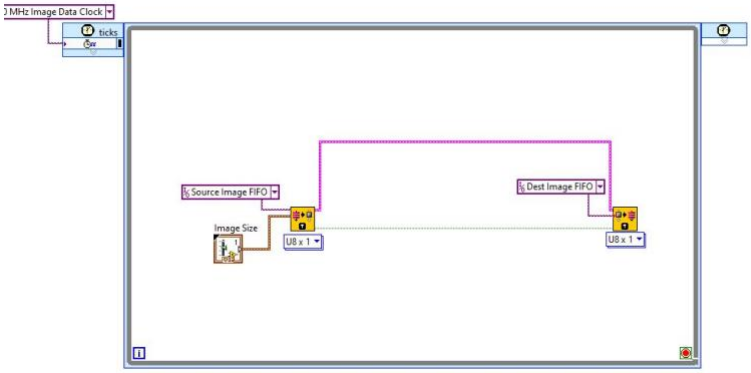


Figure 10. implementation - transfer image to FPGA

3.2.3 Implementation subtraction

The figure 11 is the FPGA vi of subtraction. This step we can remove the oversized particles by subtracting the background.

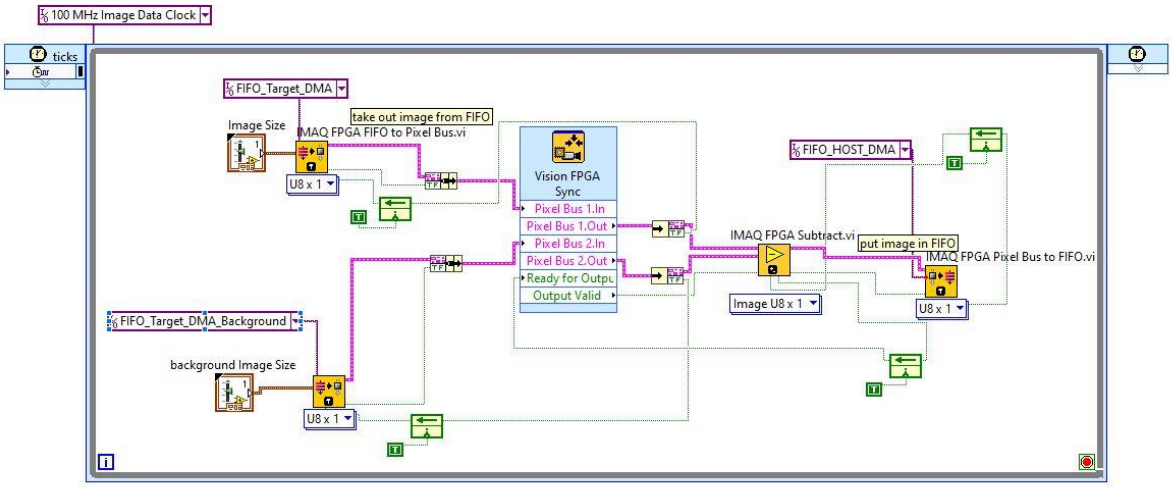


Figure 11. implementation - subtract the background

3.2.4 Implementation segmentation

The figure 12 is the FPGA vi of segmentation. The aim is filtering the noise from the image and only keep the particles.

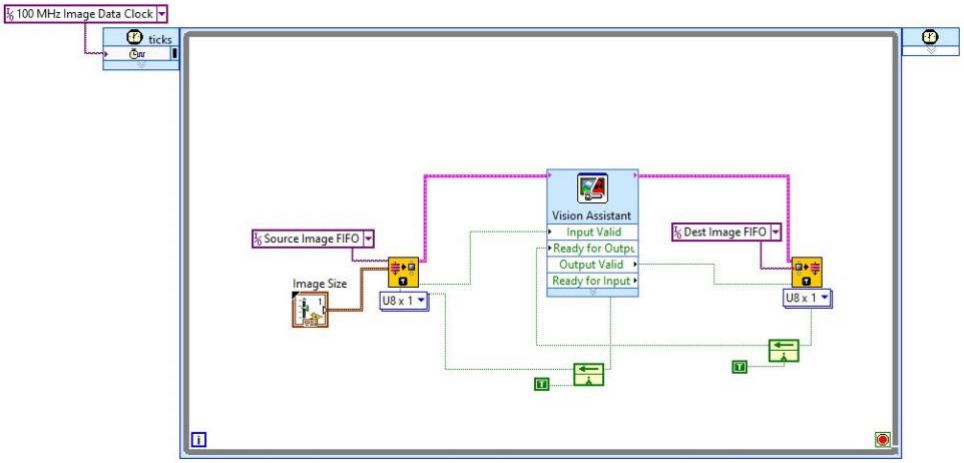


Figure 12. implementation – Segmentation

3.2.5 Implementation centroid

The figure below is the FPGA vi of centroid. Thanks to the operator *IMAQ FPGA ParticleAnalysisReport* which helps us find the centroid coordinates of particles. Then we used 2 FIFOs to transfer coordinates X and Y to PC. You can find the coordinates on Windows shown as the Figure 13.

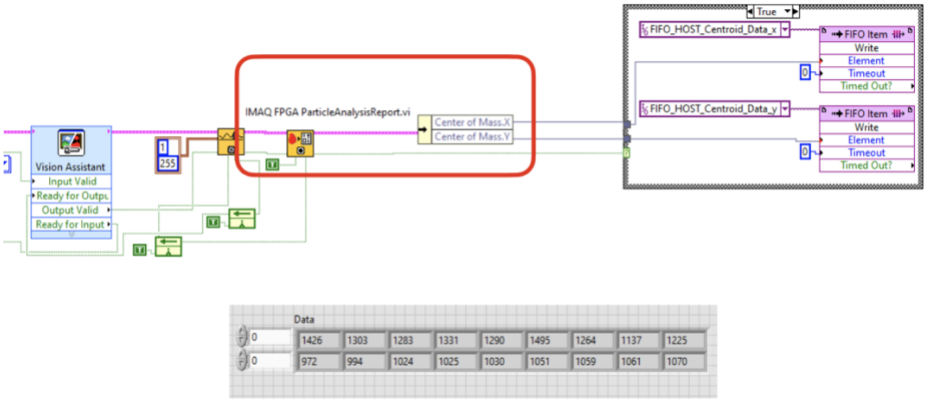


Figure 13. implementation – Centroid

3.2.6 Implementation particle detection (subtraction – segmentation – centroid)

To detect particle central coordinates, we execute all these steps: *subtraction, segmentation* and *centroid*. We put together the previous programs, shown as Figure below. This program runs on our FPGA card: NI PCIe 1477.

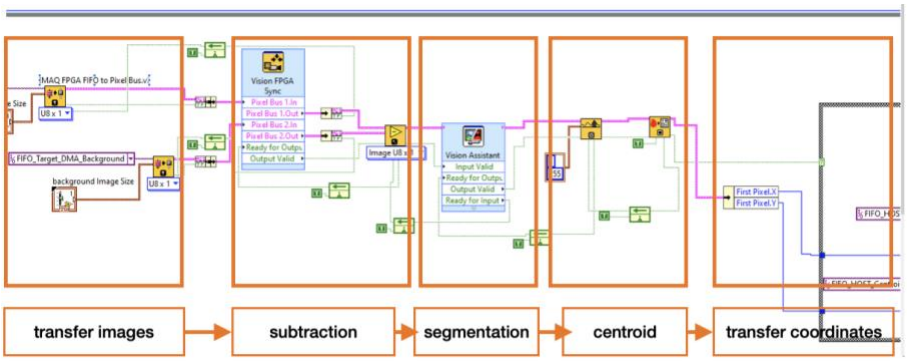


Figure 14. implementation – subtraction, segmentation, centroid

With the clock 100 MHz, we get the report surface and timing. You can find more details at the table below.

Device Utilization	Used	Total	Percent
Total Slices	24671	50950	48,4
Slice Registers	65647	407600	16,1
Slice LUTs	67242	203800	33,0
Block RAMs	155	445	34,8
DSP48s	12	840	1,4
Timing	42ms		

Table 2. report surface and timing of particle detection

3.3 LabVIEW Windows Implementation

3.3.1 Particle detection with multi-images

We add *for loop* on FPGA Windows to run multi times *FPGA particle detection module*. So that we can get particle coordinates from multi-images. We add an input *number of images* to control the loop time. The implementation is shown below.

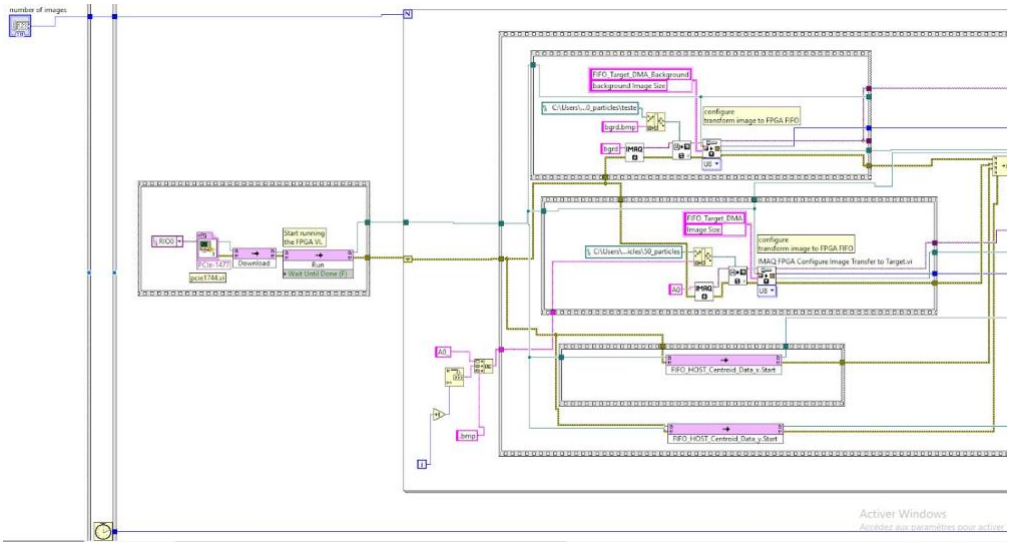


Figure 15. implementation - particle detection with multi-images

The coordinates we got is shown below, which is a table of 3 dimensions: *image*, *x* and *y*. The execution time of each image is about 42 ms.

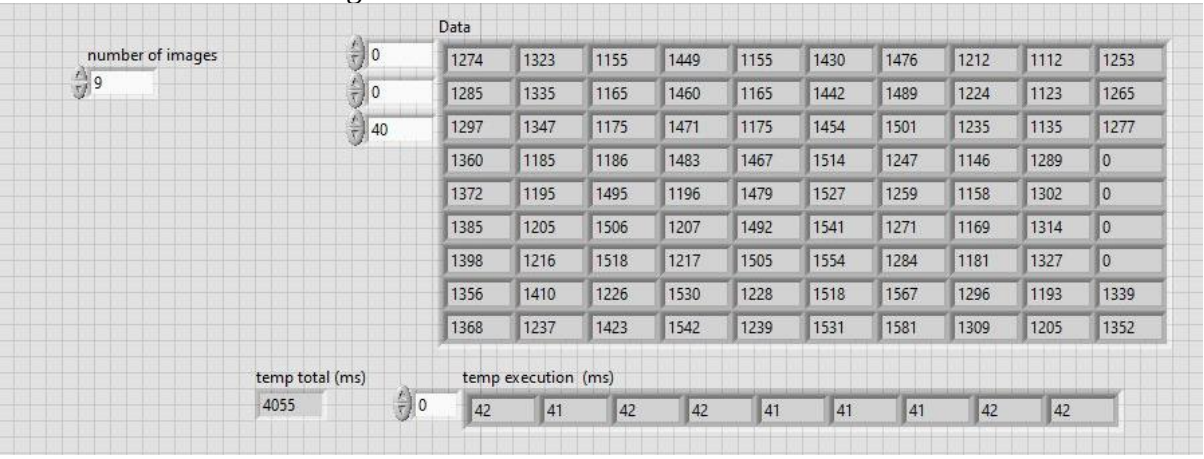


Figure 16. result - particle detection with multi-images

3.3.2 image generation

To generate images with particle, we implemented a project to get line particle images, sine particle images and helix particle images.

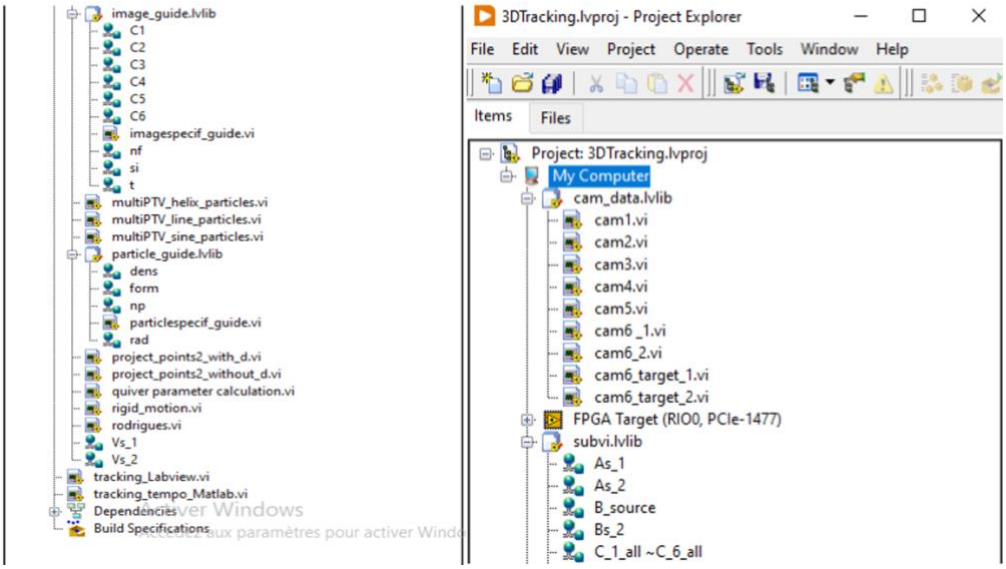


Figure 17. LabVIEW project image generation

We have the library cam_data.lvlib to store the internal parameters of each camera, and the particle_guide.lvlib, image_guide.lvlib to configure the parameters of particles and the quantity of frames.

We use the global variables to store the datas of these 2 guides: *particle_guide.lvlib*, *image_guide.lvlib*. So we should do the configuration by running *particle_guide.vi*, *image_guide.vi* before we do the 3D tracking, if not, it will use the default datas which the quantity of particle is 50 and we have 120 frames.

It mainly composed 4 libraries:

- *Cam_data.lvlib* : store parameters of 3 cameras, such as Rc, Tc.
- *Image_guide.lvlib* : run *imagespecif_guide.vi* to configure the number of frames, the size of images, image names and type.
- Particle_guide.lvlib : configure the number of particles, particle form, particle pixel radius and 3D positions range.
- Subvi.lvlib
    - *multiPTV_helix_particles.vi :* generate helix particles images.
    - *multiPTV_line_particles.vi :* generate line particles images.
    - *multiPTV_sine_particles.vi :* generate sine particles images.

3.3.2.1 multiPTV_line_particles.vi: Line particles images generation

The example of *multiPTV_line_particles.vi* : when we run the *multiPTV_line_particles.vi* which will randomly generate 50 particle coordinates, and in these 120 frames of images, make the particles move along the Y axis. We can set the moving distance. In the example shown in the figure below, the distance the particles move every two frames is 4 mm.

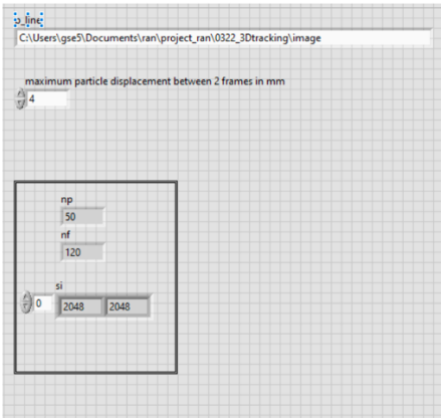


Figure 18. input of multiPTV_line_particles.vi

The output in Figure 19 shows us:

- *C_1_all ~ C_6_all* : all the coordinates of the 50 particles of 120 frames with 3 cameras by using B_source and B_transformed_all which did the 2D projection.
- *Vs*: the velocity of particle motion.
- *As*: the acceleration of particle motion.

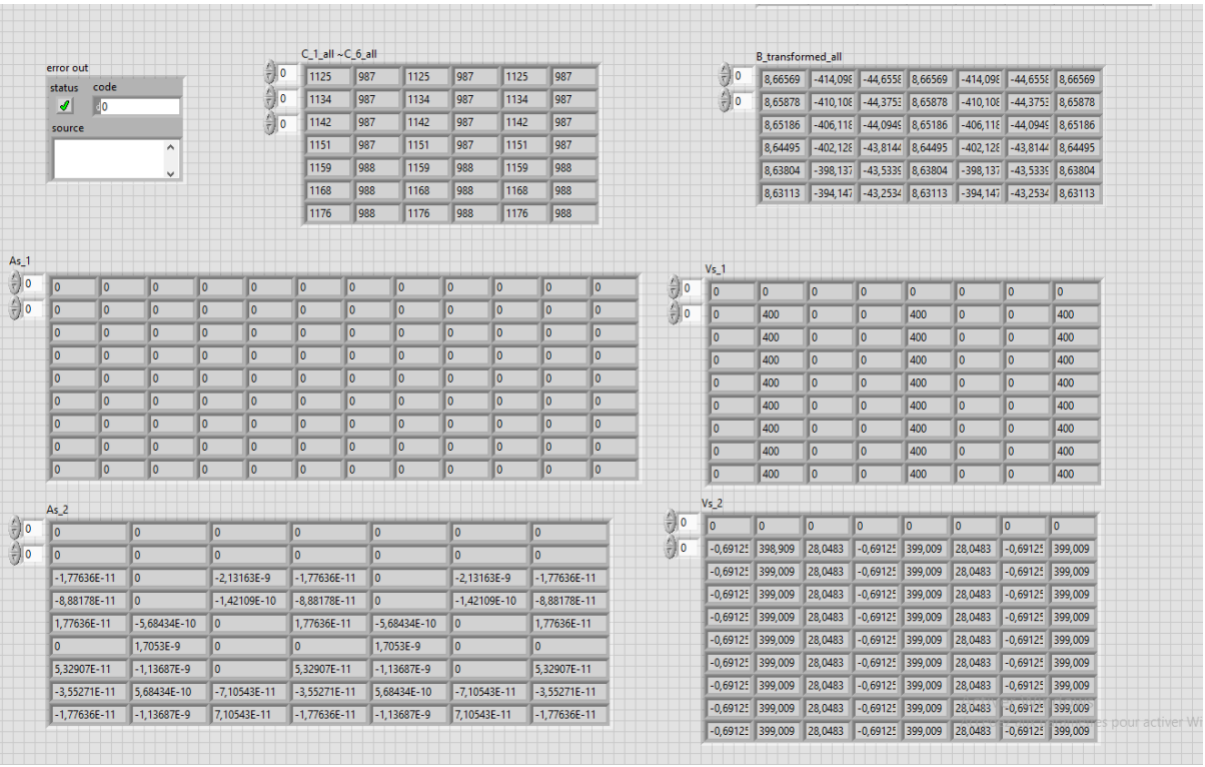


Figure 19. output of multiPTV_line_particles.vi

3.3.2.2 Tracking_Labview.vi: tracking of particle images in LabVIEW.

In this vi, we can get

- path 3D (mm) from work frame point of view with source B and B_2
- velocity 3D (mm/s) from work frame point of view with source B and B_2
- 2D representation on images planes reference frames with source B and B_2

  B is the original 3D coordinates generated randomly.
  B_2 is the transformed 3D real coordinates of the target system.

We can see the results below, an example of *sine particles images*.

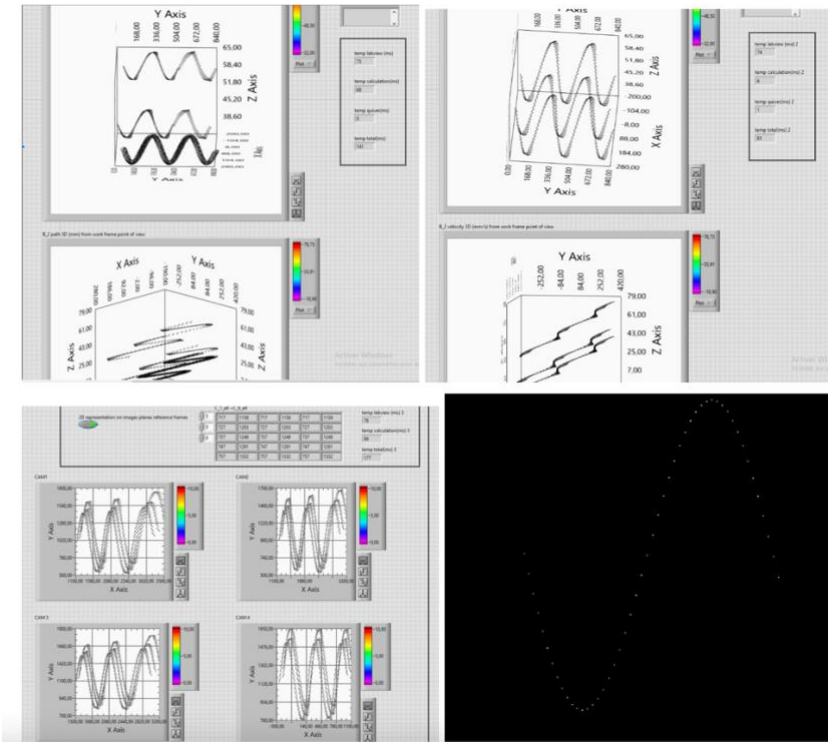


Figure 20. The results of Sine particles of tracking_Labview.vi

3.3.2.3 performance of Tracking_Labview.vi

We use the global variables of sine particles to measure the execution time.

1st running			
	time of read global variables(ms)	time calculation (ms)	total time (ms)
Path	46	966	1012
Velocity	46	9	56
2D representation	51	1446	1497
2nd running			
	time of read global variables(ms)	time calculation (ms)	total time (ms)
Path	52	470	522
Velocity	52	9	61
2D representation	57	991	1048
3rd running			
	time of read global variables(ms)	time calculation (ms)	total time (ms)
Path	50	460	511
Velocity	50	7	57
2D representation	54	954	1008

Table 3. execution time of tracking sine particles by Tracking_Labview.vi

3.3.3 Temporal tracking

Temporal tracking describes the research of particle trajectories on successive 2D particle images. The Figure below shows our project temporal tracking.

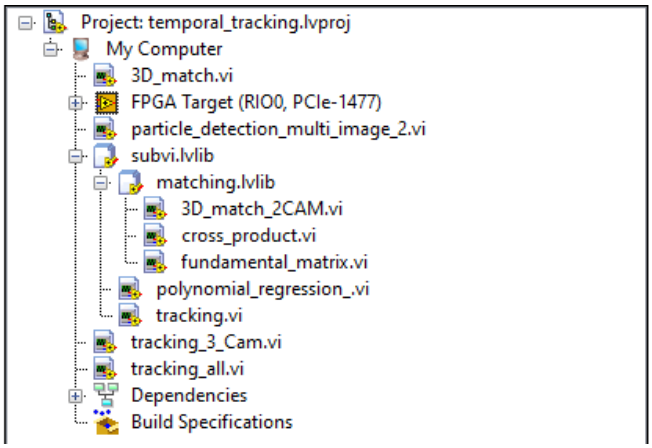


Figure 21. LabVIEW project temporal tracking

3.3.3.1 Temporal_tracking of one CAM

Tracking_all.vi
IN:

- particle speed : default 30
- number of images : default 9
- base path : image path
- image name : default 'C1_'

OUT:

- coordinates : particle coordinates detected by PCIe1477 card, size [2*nf*np]
- out : tracking image display
- Array out : reshape coordinates to size [np*nf*2]
- matched centroid : trajectories matrix after using nearest neighbor and polynomial regression method, size [np*nf*2]
- error out
- exec time per image(ms)
- execution time of particle detection (ms)
- total time of particle detection (ms)

In this vi, we can see from Figure 22, the program can be separate to 2 parts: *particle detection* and *temporal tracking.*

Particle detect.vi is using what we did before, detecting the particles by using FPGA card PCIe1477. We transferred the image to the card and did the subtraction, segmentation and centroid, then transferred the coordinates to the host PC (Figure 15).

Tracking vi called *polynomial_regression.vi* to analyse the coordinates, the drawed the point and the trajectories (Figure 23).

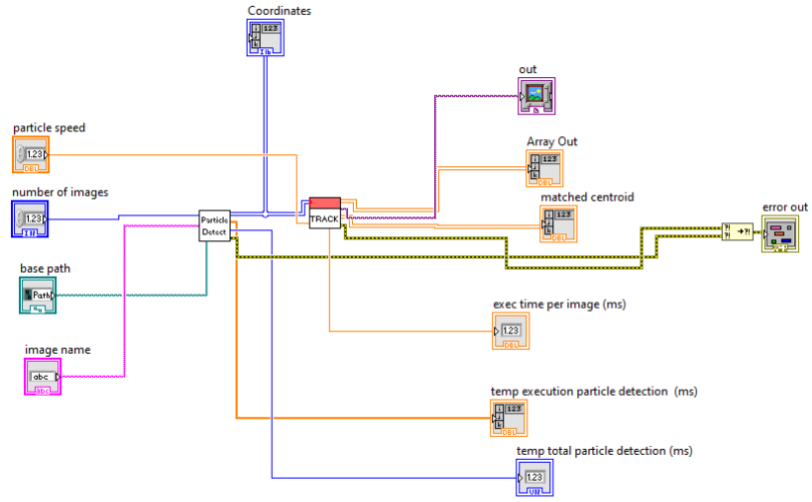


Figure 22. Program of temporal tracking of one CAM

Tracking vi:

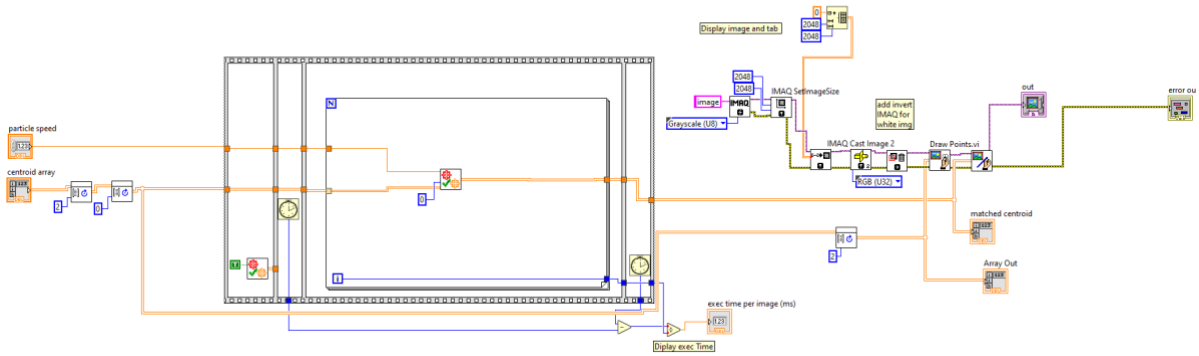


Figure 23. implementation - Tracking.vi

The image below shows the trajectories of 9 images with line particles by running *Tracking_all.vi*.

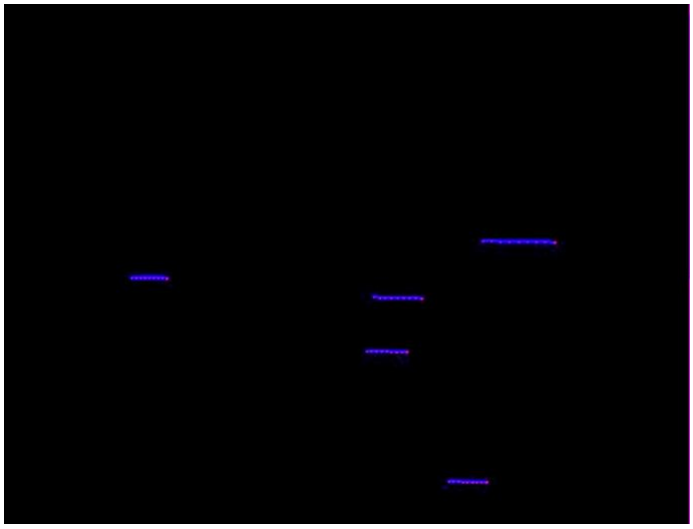


Figure 24. trajectories - line particles

3.3.3.2 Temporal_tracking of 3 CAMs

The implementation of *temporal_tracking.vi* shown as Figure 25. It calls *tracking_all.vi* (Figure 22) to get the trajectories of each camera.

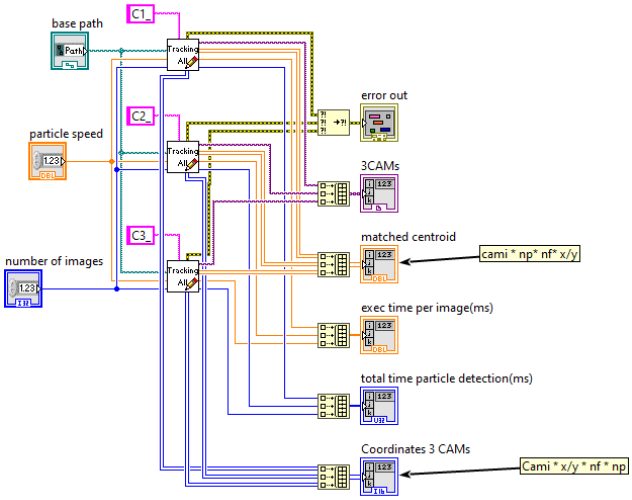


Figure 25. Program of temporal tracking of 3 CAMs

3.3.4 Spatial matching

The method we used to realise 3D matching is *Stereo pair matching*. The aim of this step is to match the trajectories of 3 cameras, to get the real-world 3D coordinates.

The project is shown below.

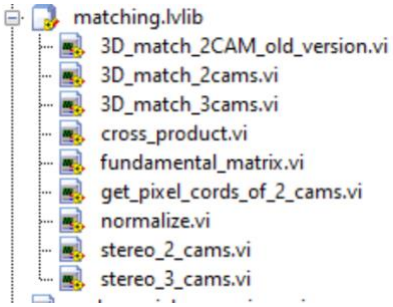


Figure 26. project spatial matching

3.3.4.1 3D matching of 3 cameras

At first, each trajectory is matched using all three fundamental matrices. So we begin our implementation from spatial matching of 3 cameras.

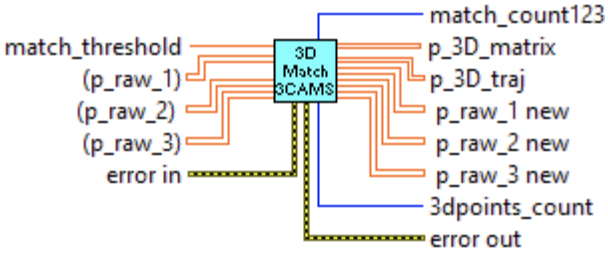


Figure 27. 3D matching of 3 cameras

Input:

- match_threshold: The right part of stereo matching *s*. Default is 1.
- p_raw_1: Coordinates of trajectories of Cam1.
- p_raw_2: Coordinates of trajectories of Cam2.
- p_raw_3: Coordinates of trajectories of Cam3.
- error in

Output:

- match_count123 match count of Cam1&Cam2&Cam3
- p_3D_traj 3D coordinates after matching.
- p_raw_1 new Coordinates of trajectories not matched of Cam1.
- p_raw_2 new Coordinates of trajectories not matched of Cam2.
- p_raw_3 new Coordinates of trajectories not matched of Cam3
- 3dpoints_count The count of matched 3D point.
- error out

The implementation in Labview:

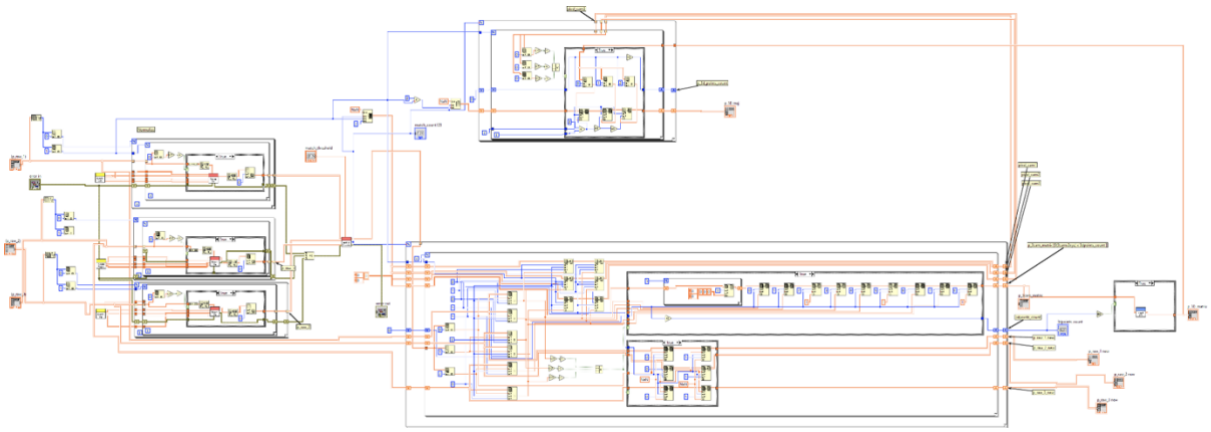


Figure 28. implementation 3D matching of 3 cameras

It calls the other vi *stereo_3_cams* which shows the main method of stereo matching.

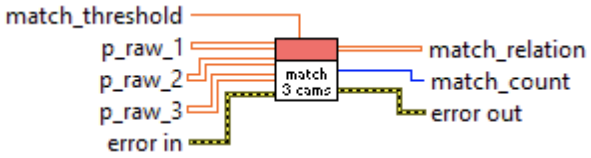


Figure 29. stereo_3_cams

The implementation is shown below, it begins with matching of cam1&cam2, if matched, it will match cam1&cam3, then cam2&cam3.

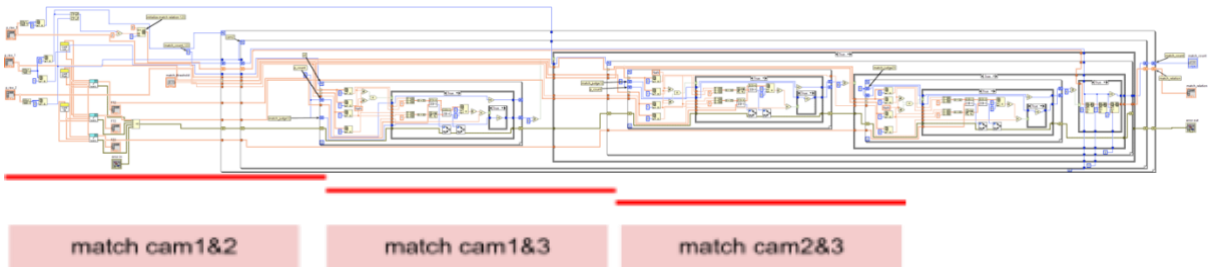


Figure 30. implementation stereo_3_cams

This vi gets the trajectories which are matched by 3 cameras. we will get a new matrix with the coordinates matched.

3.3.4.2 3D matching of 2 cameras

After spatial matching of 3 cams, what we should realise is 3D matching of 2 cameras. Those trajectories come from particles whose displacement is seen by only two cameras.

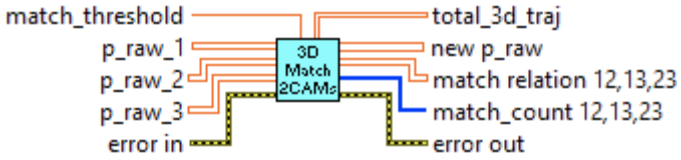


Figure 31. 3D matching of 2 cameras

Input :

- match_threshold: The right part of stereo matching *s*. Default is 1.
- p_raw_1: coordinates of trajectories of Cam1.
- p_raw_2: coordinates of trajectories of Cam2.
- p_raw_3: coordinates of trajectories of Cam3.
- error in

Output:

- total_3d_traj: 3D coordinates after matching.
- new_p_raw: coordinates of trajectories not matched of Cam1, Cam2 & Cam3
- match_relation 12,13, 23: (3D Array) match relation of 3 cameras.
- match_count 12,13, 23: (1D Array) match count of cam1&cam2, cam1&cam3, cam2&cam3
- error out

The implementation in Labview:

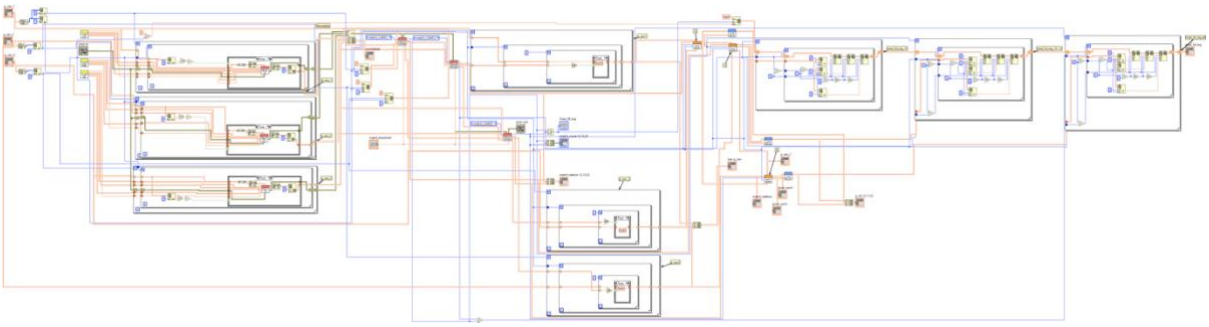


Figure 32. implementation 3D matching of 3 cameras

It calls the other vi *stereo_2_cams*

which shows the main method of stereo matching.

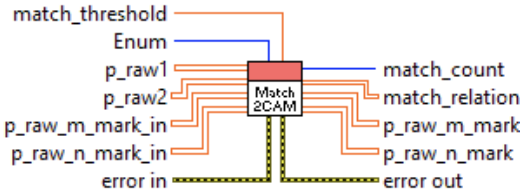


Figure 33. stereo_2_cams

The implementation of *stereo_2_cams.vi* is shown below. We have an enum input to choose spatial matching of which 2 cameras.

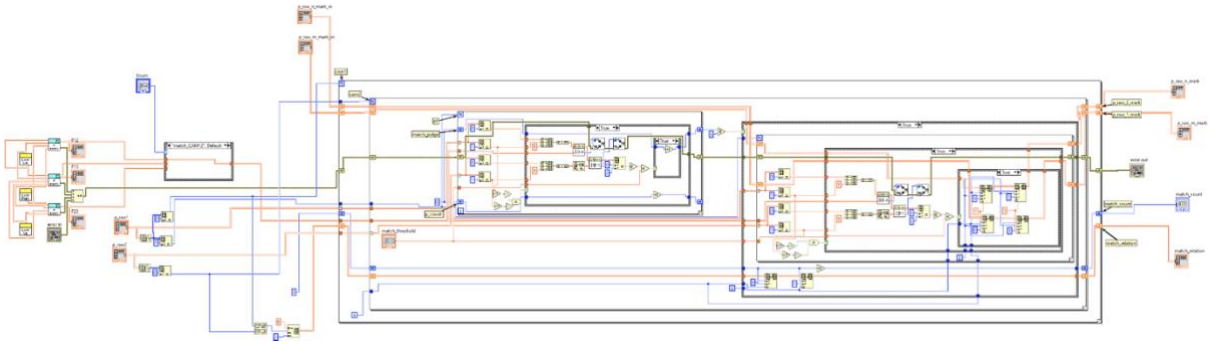


Figure 34. implementation stereo_2_cams

3.3.4.3 Test 3D matching of 2 cams

We use the matlab script to read the matlab data file, and then we continue by normalized coordinates and matching of each 2 cameras. We get the result below. What we care about is matched by 2 cams, if one trajectory is not matched by the others, we will delete it by replacing all coordinates with -1.

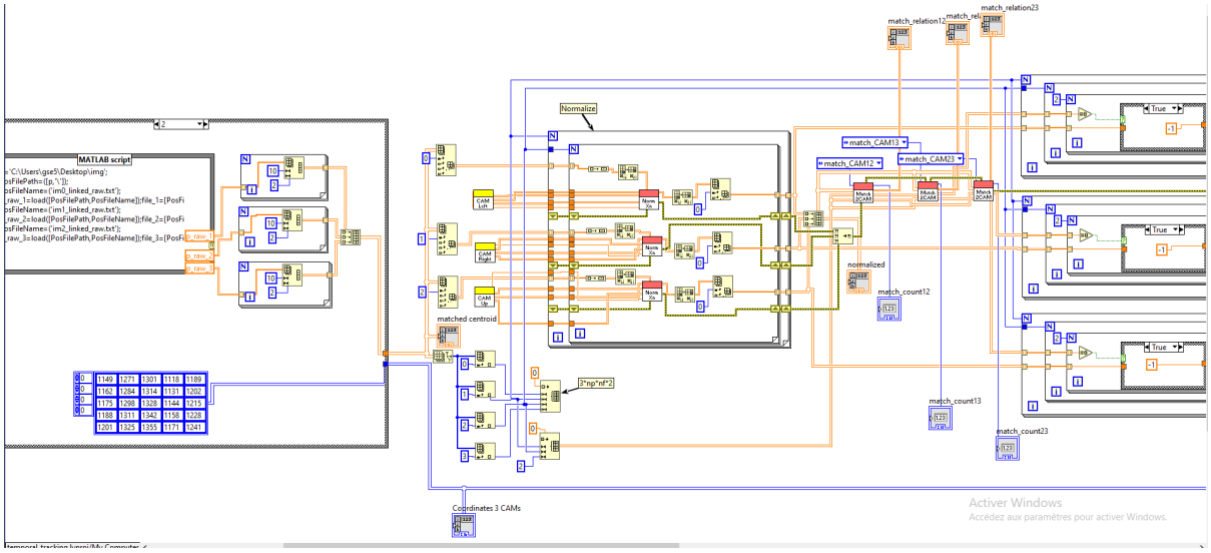


Figure 35. implementation test 3D matching of 2 cams

After the 3D matching, we will get all coordinates matched and calculate the real 3D coordinates. As the figure below, *match_relation 12/ 13/ 23* shows the match relation between cam1 & cam2, cam1 & cam3 and cam2 & cam3. The *match_count 12/ 13/ 23* count the matched number of each 2 cameras.

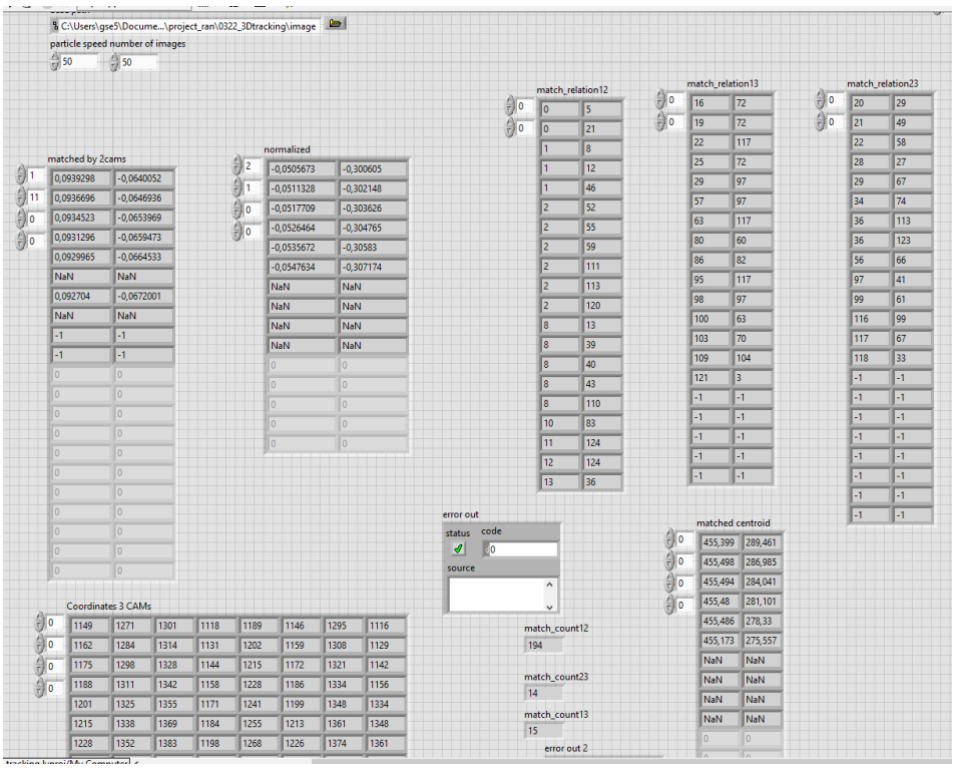


Figure 36. result of test 3D matching of 2 cams

3.3.5 3D reconstruction

Our trajectories can be matched by 2 or 3 cameras, so that we can get 4 or 6 equations, and we have 3 unknowns (real world coordinates X, Y, Z). This vi cord3D is to calculate the real world coordinates of 3 cameras.

The cord3D.vi is used after spatial matching of 3 cameras.

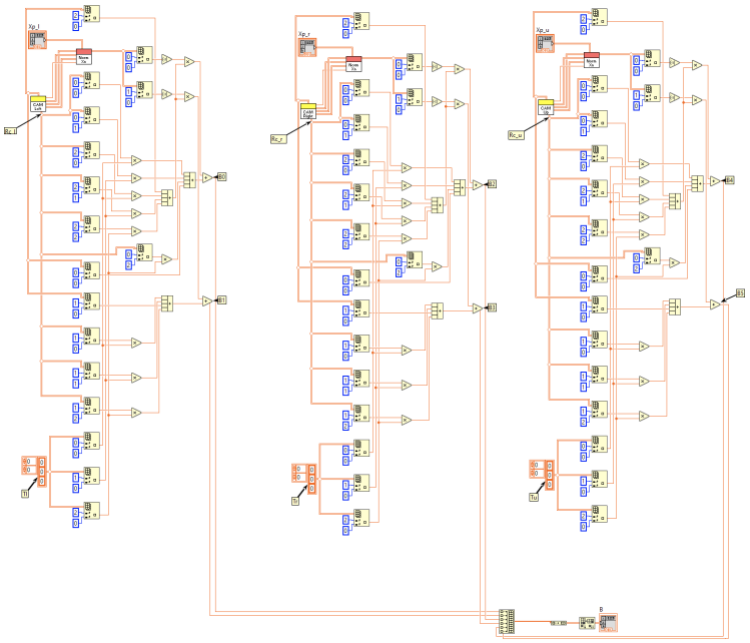
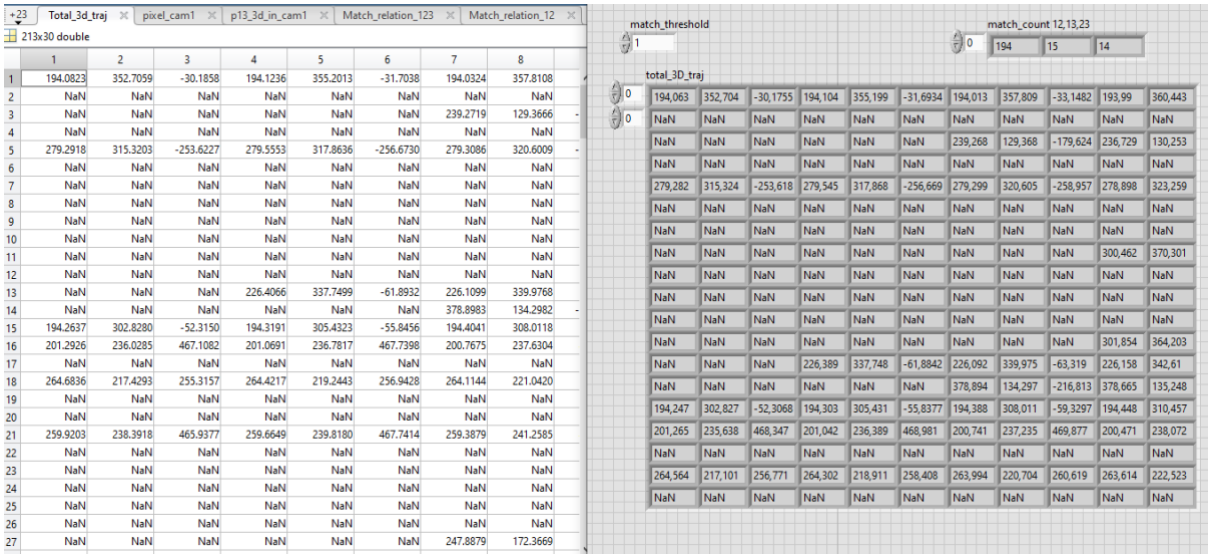


Figure 37. implementation 3D reconstruction

At last, we get the real-world 3D coordinates from LabVIEW shown below. It's similar to the Matlab result. (The figure left is the matlab result, and on the right is the LabVIEW result.)



Total_3d_traj | pixel_cam1 | p13_3d_in_cam1 | Match_relation_123 | Match_relation_12

213x30 double

	1	2	3	4	5	6	7	8
1	194.0823	352.7059	-30.1858	194.1236	355.2013	-31.7038	194.0324	357.8108
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	239.2719	129.3666
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	279.2918	315.3203	-253.6227	279.5553	317.8636	-256.6730	279.3086	320.6009
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	226.4066	337.7499	-61.8932	226.1099	339.9768
14	NaN	NaN	NaN	NaN	NaN	NaN	378.8983	134.2982
15	194.2637	302.8280	-52.3150	194.3191	305.4323	-55.8456	194.4041	308.0118
16	201.2926	236.0285	467.1082	201.0691	236.7817	467.7398	200.7675	237.6304
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	264.6836	217.4293	255.3157	264.4217	219.2443	256.9428	264.1144	221.0420
19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
21	259.9203	238.3918	465.9377	259.6649	239.8180	467.7414	259.3879	241.2585
22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
27	NaN	NaN	NaN	NaN	NaN	NaN	247.8879	172.3669

match_threshold: 1

match_count 12,13,23: 0 | 194 | 15 | 14

total_3D_traj

0	194,063	352,704	-30,1755	194,104	355,199	-31,6934	194,013	357,809	-33,1482	193,99	360,443
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	239,268	129,368	-179,624	236,729	130,253
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	279,282	315,324	-253,618	279,545	317,868	-256,669	279,299	320,605	-258,957	278,898	323,259
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	300,462	370,301
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	301,854	364,203
	NaN	NaN	NaN	226,389	337,748	-61,8842	226,092	339,975	-63,319	226,158	342,61
	NaN	NaN	NaN	NaN	NaN	NaN	378,894	134,297	-216,813	378,665	135,248
	194,247	302,827	-52,3068	194,303	305,431	-55,8377	194,388	308,011	-59,3297	194,448	310,457
	201,265	235,638	468,347	201,042	236,389	468,981	200,741	237,235	469,877	200,471	238,072
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	264,564	217,101	256,771	264,302	218,911	258,408	263,994	220,704	260,619	263,614	222,523
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 38. results 3D coordinates (MATLAB and LabVIEW)

The match count we obtained from LabVIEW of CAM1&CAM2, CAM1&CAM3, CAM2&CAM3 is similar to the result of Matlab

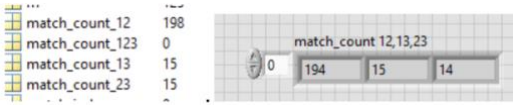


Figure 39. results match count (MATLAB and LabVIEW)

4. Performance optimization

4.1 Particle detection optimization

4.1.1 Timing optimisation

At first, we optimised the program by increasing frequency. We used the frequency 130MHz Clock, which is the maximal frequency of this implementation, there will be compilation errors over this frequency.

We can see from the Resource report total slices utilisation is 51.8%, the execution time is about 32 ms, which frame rate is 31.25 fps.

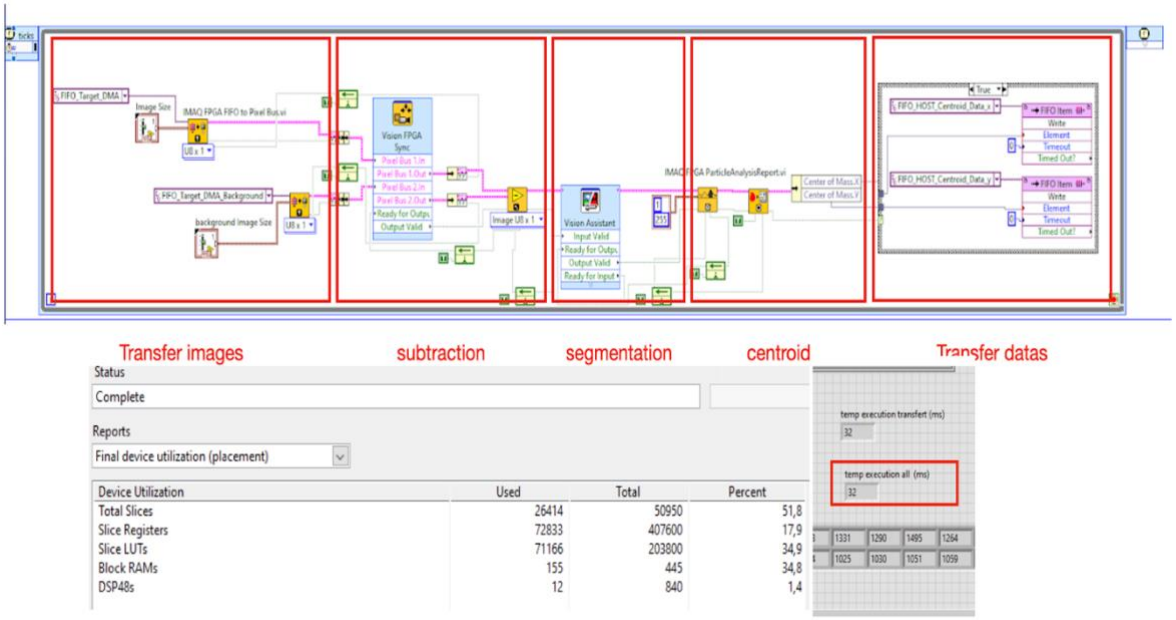


Figure 40. optimization FPGA - increase frequency

4.1.2 Optimisation: Pipeline

Then we optimised the program by pipeline it. To realise the pipeline in LabVIEW, we have 2 methods, one is using operator feedback, the other is using registers. We used registers to realise pipeline. Because of the pipeline, all five parts are running at the same time.

We can find the maximal frequency of the implementation is increasing, which is 133MHz now. We can see from the Resource report total slices utilisation is 48.5%, and the execution time is about 31 ms, which frame rate is 32.258 fps.

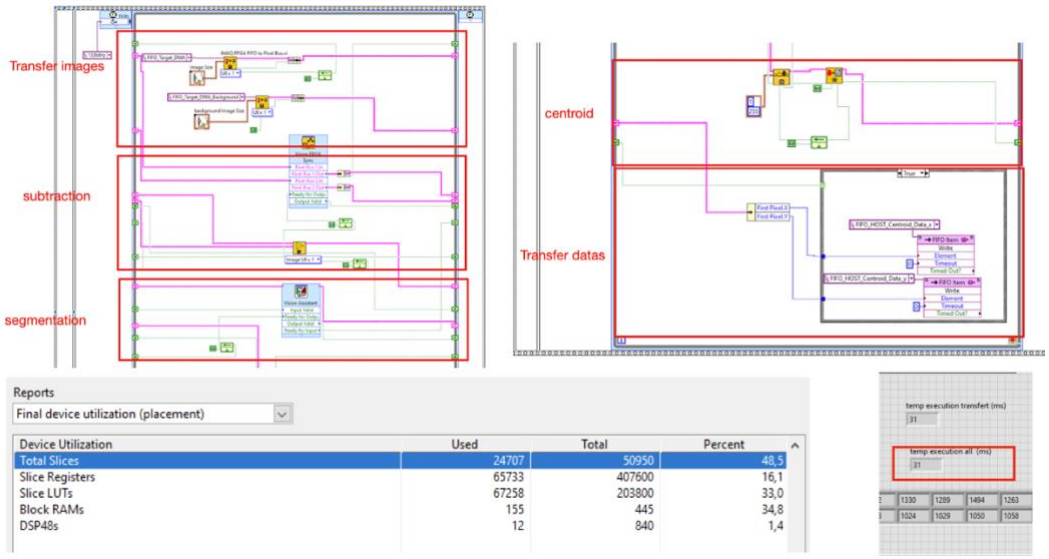


Figure 41. optimization FPGA – pipeline

4.1.3 Optimisation: Reduction the number of FIFOs

Then we optimised the program by reducing the number of FIFO, we used only one FIFO instead of 2 FIFOs.

After this optimisation, we can find the maximal frequency of the implementation is increasing, which is 135MHz now. We can see from the Resource report total slices utilisation is 47.7%, and the execution time is about 30 ms, which frame rate is 33.33 fps.

Finally, this is the best performance now, with the maximal frame rate.

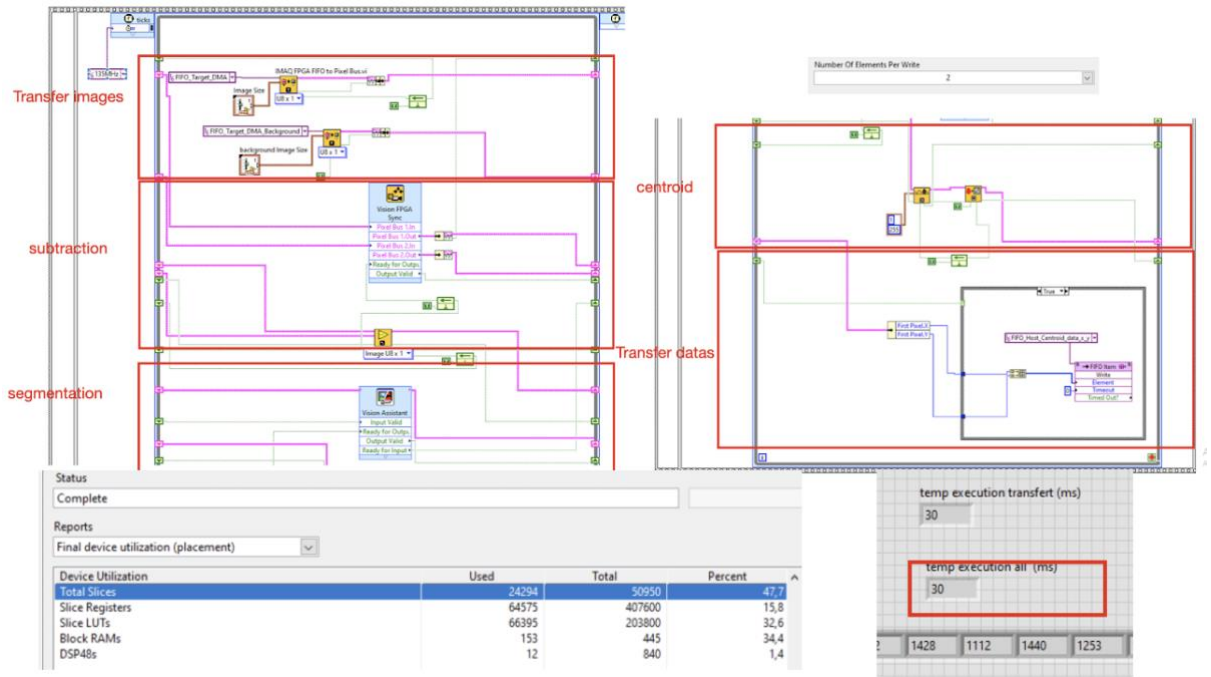


Figure 42. optimization FPGA – reduction number of FIFO

4.1.4 conclusion

Here is the conclusion of all our optimisation for the image processing, calculation the coordinates of particle center by FPGA.

To optimise the performance, the methods we used are increase frequency of time loop, pipeline and reduction the number of FIFOs.

The best performance is 135MHz, total slices utilisation is 47.7%, execution time is 30ms and frame rate is 33.33 fps.

Methods	timing/frequency	total slices utilisation	execution time	frame rate
original	100MHz	48.4%	42ms	23 fps
Timing optimization	130MHz	51.8%	32ms	31.25 fps
pipeline	133MHz	48.5%	31ms	32.258 fps
reduction the number of FIFOs	135MHz	47.7%	30ms	33.33 fps

Table 4. summary FPGA optimization

4.2 3D matching optimization

4.2.1 Algorithm and implementation

To realise spatial matching of 2 cameras, we have to match the trajectories three times with each 2 cameras. The connection of these three parts is *p_raw_mark*, which marks whether the three camera tracks are matched.

- particles with a mark of 0 means that it has not been matched successfully. We are not interested in this and will delete them later
- particles with a mark of 0 means that it has been matched successfully.

At first our algorithm of implementation is shown as the Figure 43, which is sequential, they are connected by *p_raw_mark_1, p_raw_mark_2, p_raw_mark_3.*

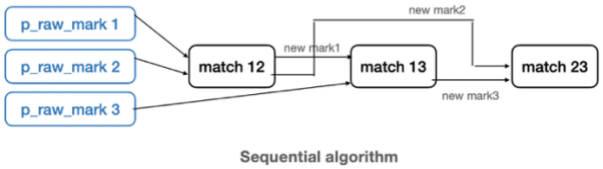


Figure 43. Sequential algorithm of 3D matching of 2 cameras

The implementation shown as Figure 44.

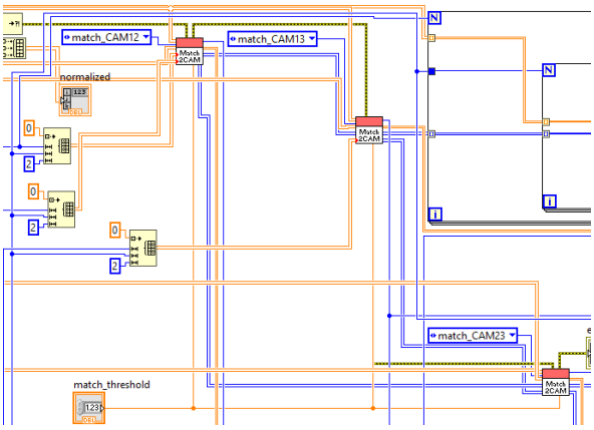


Figure 44.implementation - sequential 3D matching of 2 cameras

The Figure 45 shows a new algorithm of parallel. The spatial matchings of cam1&cam2, cam1&cam3, cam2&cam3 execute at same time, then we do the or operation of each *p_raw_mark* to get the final data.

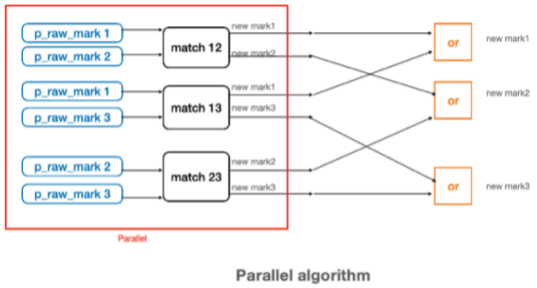


Figure 45. Parallel algorithm of 3D matching of 2 cameras

The implementation shown as Figure 46.

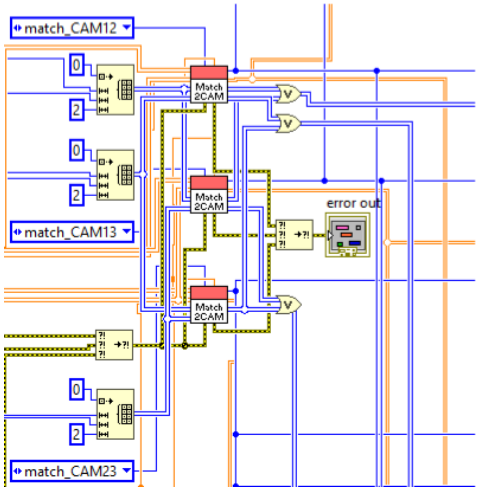


Figure 46. implementation - parallel 3D matching of 2 cameras

4.2.2 Performance compare

The datas show the execution time of spatial matching of 2 cameras, sequential or parallel, with 10 images, 50 images and 100 images where 50 particles each image.

Execution time (ms)	parallel	sequential	ratio
10 images	302	398	0.7587939
50 images	952	1142	0.8336252
100 images	6607	8918	0.7408611

Table 5. Performance compare - optimization of 3D matching of 2 cameras

We can find that it's optimised by using the parallel method.

4.3 Comparison timing by using MatlabScript and G-language

I compared execution time of MatlabScript and G-language. I used 2 simple examples to compare the execution time of MatlabScript and G-language, with matrix and without matrix.

4.3.1 Ratio timing MatlabScript/LabVIEW without matrice

At first, I compared the execution time of MatlabScript and G-language without matrix. The program is shown below.

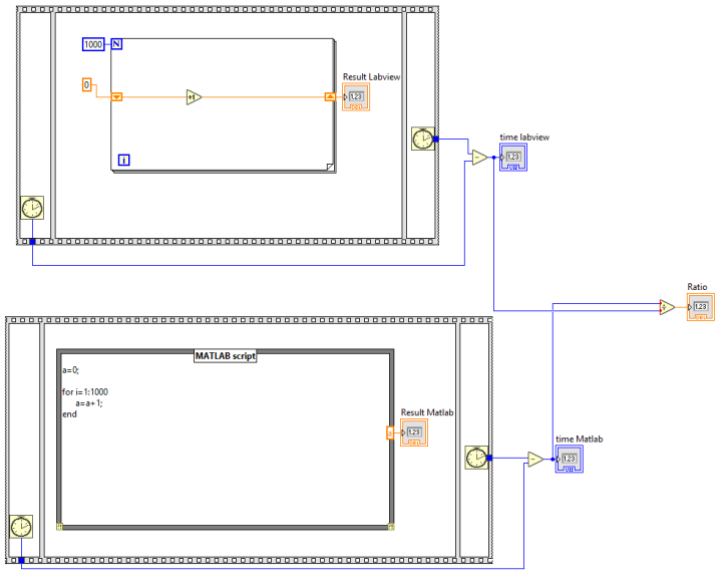


Figure 47. program - MatlabScript and G-language without matrix

We can find the results below. We can see the running time of the LabVIEW is less than 1 ms, and the running time of MATLAB gradually decreases with the running times. As the figures above, the first time of execution, timing of MATLAB is 13 ms, the second time is 9 ms and the last time, 2 ms.

	time MatlabScript(ms)	time G-language(ms)	Ratio
1st running	13	0	Inf
2nd running	9	0	Inf
3th running	2	0	inf

Table 6.compare timing MatlabScript/LabVIEW without matrix

4.3.2 Ratio timing MatlabScript/LabVIEW with matrice

Then, I compared the execution time of MatlabScript and G-language with matrix. The program is shown below.

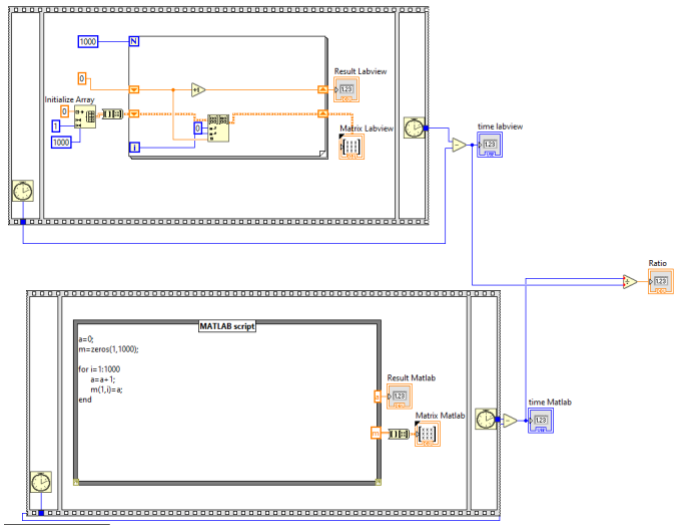


Figure 48. program - MatlabScript and G-language with matrix

We can see the running time of the LabVIEW is also less than 1 ms, and the running time of MATLAB gradually decreases with the running times. As the figures above, the first time of execution, timing of MATLAB is 16 ms, the second time is 14 ms and the last time, 6 ms.

	time MatlabScript(ms)	time G-language(ms)	Ratio
1st running	16	0	Inf
2nd running	14	0	Inf
3th running	6	1	6

Table 7. compare timing MatlabScript/LabVIEW with matrix

5 Experiment

To execute our project, we should follow these steps:

- generate images with particles
- particle detection
- temporal tracking
- 3D matching
- 3D reconstruction

Figure 49. execution steps

We execute the whole project, and the execution time is shown below.

We have a Boolean button called "*parallel?",* true means 3D matching with optimized method parallel, false means sequential. The figure below shows the execution time of the whole project with method sequential, by using 10 images with 50 particles each frame.

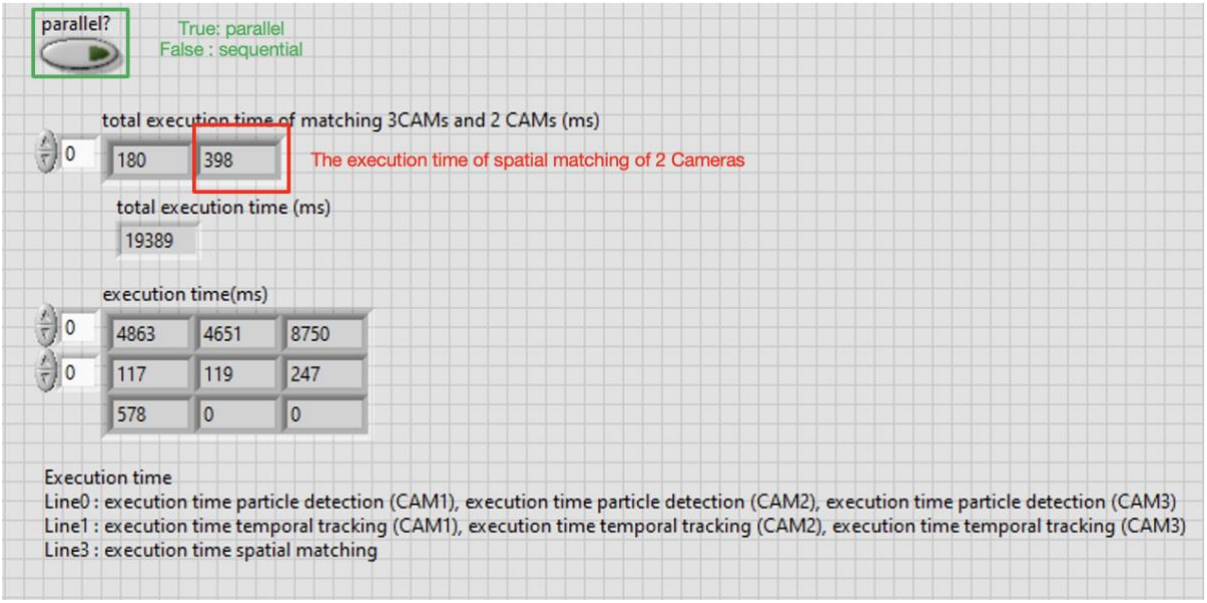


Figure 50. performance - 10 images with 50 particles (sequential)

Performance - 10 images with 50 particles (parallel)

We are more interested to the optimized method parallel, the table below shows the execution time of the whole project with method parallel, by using 10 images with 50 particles each frame.

particle detection(CAM1)	particle detection(CAM2)	particle detection(CAM3)
4266 ms	4296 ms	4288 ms
temporal tracking(CAM1)	temporal tracking(CAM2)	temporal tracking(CAM3)
115 ms	116 ms	126 ms
spatial matching (3 CAMS)	spatial matching (2 CAMS)	spatial matching (All)
247 ms	302 ms	549 ms
total execution time		
13786 ms		

Table 8. performance - 10 images with 50 particles (parallel)

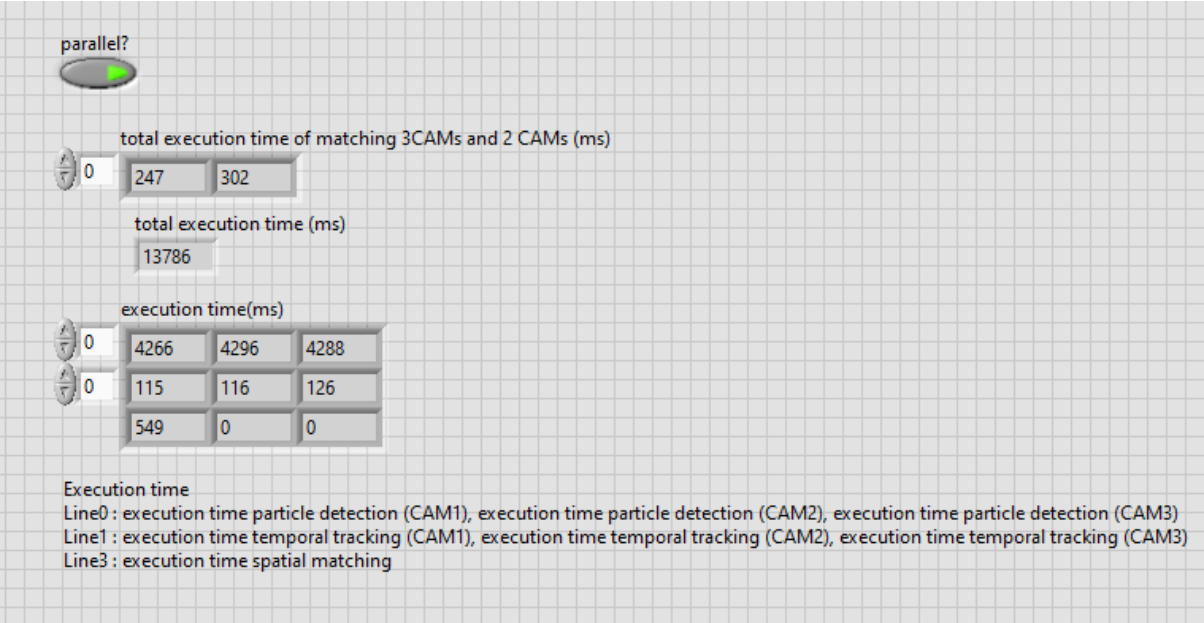


Figure 51. performance - 10 images with 50 particles (parallel)

Performance - 50 images with 50 particles (parallel)

The table below shows the execution time of the whole project with method parallel, by using 50 images with 50 particles each frame.

particle detection(CAM1)	particle detection(CAM2)	particle detection(CAM3)
7115 ms	7059 ms	7109 ms
temporal tracking(CAM1)	temporal tracking(CAM2)	temporal tracking(CAM3)
153 ms	159 ms	210 ms
spatial matching (3 CAMS)	spatial matching (2 CAMS)	spatial matching (All)
605 ms	952 ms	1557 ms
total execution time		
23400 ms		

Table 9. performance - 50 images with 50 particles (parallel)

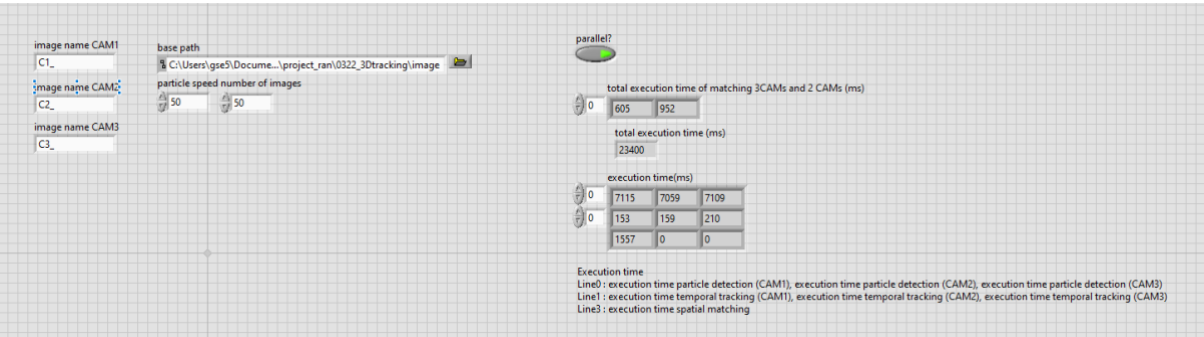


Figure 52. performance - 50 images with 50 particles (parallel)

Performance - 100 images with 50 particles (parallel)

The table below shows the execution time of the whole project with method parallel, by using 100 images with 50 particles each frame.

particle detection(CAM1)	particle detection(CAM2)	particle detection(CAM3)
9723 ms	9715 ms	9737 ms
temporal tracking(CAM1)	temporal tracking(CAM2)	temporal tracking(CAM3)
657 ms	548 ms	601 ms
spatial matching (3 CAMS)	spatial matching (2 CAMS)	spatial matching (All)
4621 ms	6607 ms	11228 ms
total execution time		
42267 ms		

Table 10. performance - 100 images with 50 particles (parallel)

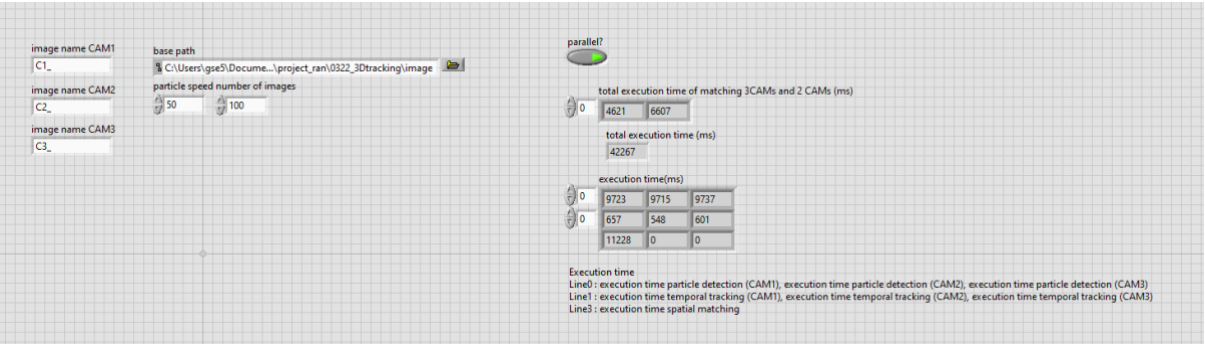


Figure 53. performance - 100 images with 50 particles (parallel)

At first, we use 10 images, and 50 particles in each image. The total execution time is about 10 seconds, most of the time spends on particle detection of 3 cameras.

Then, we use 50 images, and 50 particles in each image. The total execution time is about 24 second, most of the time also spends on particle detection of 3 cameras. And we can find the time of spatial matching is increased.

At last, we use 100 images, and 50 particles in each image. The total execution time is about 47 second, we can find the time of spatial matching is dramatic increased, it's normal because it should traverse all the trajectories to get matched, there are lots of **loop** will be done.

6 Conclusion

During this internship, we developed the image processing chain to calculate the pixel coordinates of particles present in successive images from three cameras. We analyze the 2048x2048 pixels images with 50 particles from 3 cameras, calculating particle center coordinates, tracking particle trajectories, spatial matching trajectories of 3 dimensions, and do the 3D reconstruction to get the real-world 3D coordinates.

We use the FPGA card NI PCIe 1477 to detect particle center coordinates, with the help of LabVIEW, a system-design platform and development environment for a visual programming language from National Instruments. The card NI PCIe 1477 is big enough to run the whole system on the board. After the optimization, the best performance is 135MHz, total slices utilisation is 47.7%, execution time is 30ms and frame rate is 33.33 fps, which do not meet the requirements of 100fps. We can see from the performance of the whole system that particle detection of 3 cameras eats too much time, which needs further optimization.

For the temporal tracking, we use the method now is *polynomial regression*, the other method *Modified fast normalized cross-correlation tracking* has not been implemented yet, we will work on that in the future work. And we will focus on transfer the LabVIEW implementation to VHDL, find another optimization method to approach our objective 100 fps.

Even if the constraint of 100 fps is not yet reached, the implementation on labview allows me to have a first fast prototype of the PVT and that this has allowed me to understand the algorithms and the problems of algorithm-architecture adequacy that they pose and that I will bring solutions in the PhD that comes by a deeper exploration of architecture.

7 Reference

1) Pascal Henry Biwole1, Wei Yan2, Yanhui Zhang2 and Jean-Jacques Roux1/2009/**A complete 3D particle tracking algorithm and its applications to the indoor airflow study**

2) Pascal Henry BIWOLE /2009/**Large Scale Particle Tracking Velocimetry for 3-Dimensional Indoor Airflow Study**

3) Nasser Kehtarnavaz and Namjin Kim/**Digital Signal Processing System-Level Design Using LabVIEW**

4) optimization FPGA vi with parallel, pipeline

5) FPGA FIFO types and configuration