


Regular / Image Processing

Reproduction of ODP-PIV from the 1998 “Particle Image Velocimetry with Optical Flow” paper

Georges Quénot¹, ¹Univ. Grenoble Alpes, CNRS, Grenoble-INP, LIG, F-38000 Grenoble France

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
–

Published
–

DOI
–

1 Introduction

In this paper, I report the reproduction of the experiments with the ODP-PIV method described in the 1998 article entitled “Particle Image Velocimetry with Optical Flow” that I published in *Experiments in Fluids*¹ (200+ citations) with my colleagues Jaroslav Pakleza from LIMSI and Tomasz Kowalewski from IPPT-PAN, now both retired. This was the first time that a dense Optical Flow (OF) technique was applied for Particle Image Velocimetry (PIV) which was so far mostly processed only using block correlation, producing velocity vectors at the block resolution (typically 32×32). This was an interdisciplinary work between my colleagues who were involved in velocity measurements in fluids flows and myself who was involved at the time in image and video processing. This was eased by the fact that the LIMSI laboratory hosted two departments, one dedicated to physics (mechanics) and another dedicated to computer science (human-machine communication). The reproduction was successful and the corresponding code and data are available from <https://github.com/quenot/opflow/tree/master/jEIF98>

2 Particle Image Velocimetry (PIV)

Particle Image Velocimetry is a technique for measuring velocities in fluid flows by seeding a fluid with small light-reflecting particles and recording two or more digital images while a plane in the fluid is illuminated by a light sheet produced by a pulsed laser. Tracking the transported particle patterns across an image pair or a longer sequence provides an estimation of the velocities in the fluid. Figure 1 shows typical particle image, a superimposed sequence of the same particle images, and a visualization of the flow field extracted with the proposed method from a sequence of four such images¹. Before the use of optical flow popularized by this work, the standard approach was to use FFT-based block correlation with significant drawbacks in terms of spatial resolution, velocity accuracy and robustness.

3 Orthogonal Dynamic Programming (ODP) for image matching and flow estimation

The optical flow technique used in this work was not a mainstream one but one that I developed previously for image matching and for flow estimation in videos. This

Copyright © 2020 G. Quénot, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to Georges Quénot (Georges.Quenot@imag.fr)
The authors have declared that no competing interests exists.
Code is available at <https://github.com/quenot/opflow/tree/master/jEIF98>.

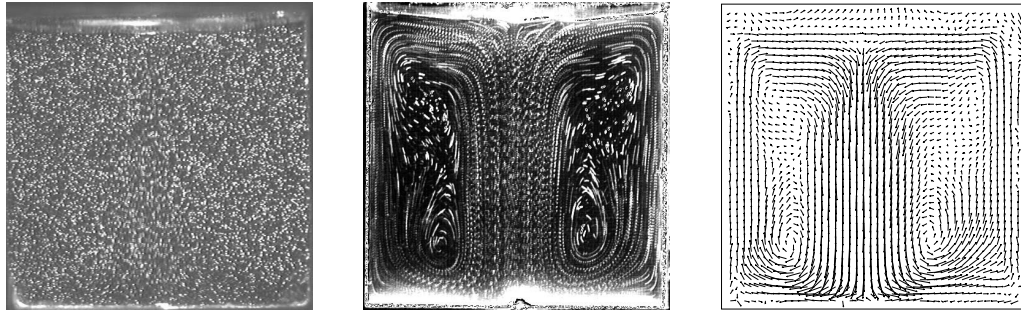


Figure 1. Principle of PIV. Left: typical particle image, middle: a superimposed sequence of the same particle images, right: a visualization of the flow field extracted with the ODP-PIV method from a sequence of four such images (non-linear velocity scale: small velocities are enlarged compared to large ones).

method was based on the use of Dynamic Programming (DP) and was inspired by the Dynamic Time Warping (DTW) method that was used in speech recognition. DTW is able to perform optimal alignments in one direction only, typically between sonograms (time \times frequency decomposition of an audio signal) along the time axis. The extension that I proposed of it made it possible to use it for (quasi-)optimal alignments in two directions via a carefully selected sequence of alternated horizontal and vertical alignments^{2,3}. In reference to the use of alternate iterations in orthogonal directions, the method was called "Orthogonal Dynamic Programming" (ODP). Though this optical flow detection method was not a mainstream one, it was on par with such methods at the time of its first application to PIV.

4 Application to Particle Image Velocimetry (ODP-PIV)

The ODP method has been applied to different problems since it was developed including 3D reconstruction and video segmentation for instance but here we are concerned by (the reproduction of) its first application to Particle Image Velocimetry (ODP-PIV). Our 1998 article compared the ODP-PIV method with a state-of-the-art block-correlation-based one. Both methods were applied to synthetic image sequences with known velocities for quantitative error evaluation, and on real image sequences for a visual estimation of the robustness and density of the obtained vector fields. This article reports only the reproduction of the results obtained using the dense optical flow estimation technique. The results for the classical DPIV method were obtained by my colleague Tomasz Kowalewski for which I never had the code.

5 Reproduction of the results

The ODP-PIV software was fully written in ANSI C without the use of any external library or other tool. This is probably why it did compile without any problem and produced the same results with only sometimes some rounding differences (more on this below).

5.1 Quantitative results on synthetic images

Synthetic particle 800×400 4-image sequences were generated in calibrated conditions. The average displacement between consecutive images is of 7.58 pixels. Conditions include perfect synthesis (neither noise nor particle appearance or disappearance), additive noise at various levels, appearance and disappearance of particles at various rates (in order to simulate out of plane motion effects) and combinations of both. Table 1

shows the velocity error in pixel/frame for two variants of the classical correlation-based PIV method, with block sizes of 32×32 and 48×48 and three variants for the ODP-PIV method, one using only two images and two using the four available images. These results are commented in detail in the original article but ODP-PIV was the clear winner.

	DPIV32	DPIV48	ODP2	ODP4S	ODP4M
Perfect	0.55 ± 0.94	0.87 ± 1.46	0.13 ± 0.10	0.13 ± 0.54	0.07 ± 0.07
Noise 5%	0.61 ± 1.18	0.86 ± 1.49	0.21 ± 0.46	0.10 ± 0.13	0.08 ± 0.08
Noise 10%	0.77 ± 1.57	0.91 ± 1.59	0.53 ± 1.44	0.17 ± 0.53	0.11 ± 0.09
Noise 20%	3.11 ± 4.14	2.06 ± 2.88	0.88 ± 1.58	0.30 ± 0.68	0.20 ± 0.14
Add/rm 5%	0.55 ± 0.90	0.86 ± 1.45	0.14 ± 0.11	0.08 ± 0.11	0.07 ± 0.08
Add/rm 10%	0.55 ± 0.93	0.87 ± 1.47	0.34 ± 1.28	0.14 ± 0.56	0.08 ± 0.09
Add/rm 20%	0.56 ± 0.99	0.88 ± 1.52	0.16 ± 0.12	0.18 ± 0.69	0.10 ± 0.10
Mixed 5%	0.60 ± 1.12	0.86 ± 1.51	0.20 ± 0.13	0.15 ± 0.53	0.09 ± 0.08
Mixed 10%	0.91 ± 1.89	0.93 ± 1.66	0.57 ± 1.71	0.20 ± 0.59	0.13 ± 0.11
Mixed 20%	3.73 ± 4.39	2.49 ± 3.19	0.74 ± 0.52	0.43 ± 1.08	0.27 ± 0.22

Table 1. Original results (table 2 in the original article¹): absolute displacement error with a mean displacement module of 7.58 pixels/frame. Errors are displayed as mean \pm standard deviation.

Table 2 shows the reproduced results using the GNU compiler (gcc) and the Intel compiler (icc). As can be seen, the results are exactly the same with the GNU compiler even though the architecture (R4400), the operating system (SGI) and the compiler (acc) were different. After investigation, it turned out that the differences using the Intel compiler were due to the use of the single-instruction multiply-add operation with intermediate result (between the multiplication and the addition) being performed using more significant bits. When this default behavior is turned off, the results become identical to those obtained with the GNU compiler. In principle, the default behavior of the Intel compiler leads to a better accuracy in the computations. As can be seen in the table, the results are sometimes better (in green), sometimes worse (in red) and often unchanged. The differences are sometimes quite important. I think that this is due to the principle of dynamic programming that chooses the alignment path with the smallest cost. Some tiny ("microscopic") differences in path costs may turn into significant ("macroscopic") differences in paths. The overall algorithm involves a lot of such path selections and it suffices that one very small instability be sampled once to produce a "macroscopic" change. In practice, such instabilities sometimes increase and sometimes decrease the performance with no clear global trend. This could be a hint about how to improve the method, were it not now obsolete.

GNU compiler			Intel compiler		
ODP2	ODP4S	ODP4M	ODP2	ODP4S	ODP4M
0.13 ± 0.10	0.13 ± 0.54	0.07 ± 0.07	0.13 ± 0.10	0.13 ± 0.55	0.07 ± 0.07
0.21 ± 0.46	0.10 ± 0.13	0.08 ± 0.08	0.21 ± 0.46	0.10 ± 0.11	0.09 ± 0.08
0.53 ± 1.44	0.17 ± 0.53	0.11 ± 0.09	0.28 ± 0.18	0.17 ± 0.51	0.11 ± 0.09
0.88 ± 1.58	0.30 ± 0.68	0.20 ± 0.14	0.83 ± 1.36	0.29 ± 0.65	0.21 ± 0.23
0.14 ± 0.11	0.08 ± 0.11	0.07 ± 0.08	0.14 ± 0.11	0.08 ± 0.10	0.07 ± 0.08
0.34 ± 1.28	0.14 ± 0.56	0.08 ± 0.09	0.34 ± 1.29	0.14 ± 0.56	0.08 ± 0.09
0.16 ± 0.12	0.18 ± 0.69	0.10 ± 0.10	0.16 ± 0.12	0.18 ± 0.70	0.09 ± 0.10
0.20 ± 0.13	0.15 ± 0.53	0.09 ± 0.08	0.20 ± 0.13	0.15 ± 0.56	0.09 ± 0.08
0.57 ± 1.71	0.20 ± 0.59	0.13 ± 0.11	0.59 ± 1.67	0.20 ± 0.59	0.13 ± 0.11
0.74 ± 0.52	0.43 ± 1.08	0.27 ± 0.22	0.92 ± 1.36	0.40 ± 0.94	0.27 ± 0.18

Table 2. Reproduced results using the GNU (left) and Intel (right) compilers

The other quantitative results (typically with higher velocities) from the original paper were also successfully reproduced.

5.2 Visual results on synthetic images

We also reproduced the visual results with one difficulty which was related to the visualization software which was the only external component that we used in the paper. This tool had two problems, one was that it did not manage the big-endian and little-endian differences across processor architectures (unlike the main ODP-PIV software that relies on `htonl()` and `ntohl()` for that) and a bounding box problem in the postscript file generation. This was manually fixed either in the code or in the generated flow fields and it was possible to generate flow fields that are visually indistinguishable from those displayed in the original paper.

5.3 Results on real images

Only visual results were reproduced regarding the real images as the "true" velocity field is unknown. In the same conditions as with the synthetic images, it was possible to generate flow fields that are visually indistinguishable from those displayed in the original paper. One of them is shown in figure 1 (right).

6 Speed-up

One interesting point in this reproduction work is to evaluate what speed-up the progress in hardware and possibly in compilers allowed for in the last 22 years on this task. We evaluated the speed-up on the velocity estimation on real sequences of $4\,496 \times 496$ images. The original paper reported 20 minutes, 210 minutes and 200 minutes for ODP2, ODP4S and ODP4M respectively, running on a R4440 @ 250MHz, and using the SGI® acc compiler. Similar execution times were obtained with a 200MHz Pentium with gcc, indicating a slightly higher throughput per MHz for the Intel processor. We ran the reproduction on a machine hosting two Intel® Xeon® CPU E5-2643 @ 3.30GHz, each with 4 cores and 8 threads. The original code is not parallel anyway. As we had access to an Intel icc compiler, we tried it along with the default Gnu gcc one. Regarding the Intel compiler, we evaluated both the default (with multiply-add) and the compatible (without multiply-add) modes.

6.1 Original version

Table 3 shows the execution time in seconds and the relative speed-up compared to the 1998 version. It is striking that the speed-up is much higher than the ratio between the processor clock frequency, even though the execution is strictly single-threaded. The Intel compiler is significantly faster than the GNU one on this task while the Intel compiler with disabled multiply-add is in-between. Using the Intel compiler in its default mode, the speed-up is over 100. Last but not least, the Intel processor-based server that we used for the reproduction costed less than one third of the cost of the R4400-base SGI server originally used. Regarding the 100+ times overall speed-up related to the 11-16 times increase in processor clock frequency, possible explanations include newly available vectored instructions, aggressive compilation techniques and highly efficient use of the processor caches.

6.2 Threaded version

The ODP-PIV method is now obsolete so it was not worth spending time in improving it. Yet, I still tried to produce a multi-threaded version of the code with a minimal effort.

	Exec. time (s)			Speed-up		
	gcc	icc-	icc	gcc	icc-	icc
ODP2	22.7	14.8	10.3	52	81	116
ODP4S	167.3	123.6	119.7	75	101	105
ODP4M	162.4	126.2	111.8	73	95	107

Table 3. Execution time and speed-up relative in the reproduction, original version. icc- is icc in the compatible mode (with disabled multiply-add).

This involved inserting the following two lines (at different places):

```
#include <omp.h>
#pragma omp parallel for num_threads(16) private(dd,gg,bb,...)
```

The latter was applied to the main computing loop (there was one higher level loop). It was also necessary to move three memory allocation and freeing calls inside the loop so that threads get their private working buffers. A procedure which keeps track of the amount of allocated memory also had to be made thread-safe, which basically involved inserting four

```
#pragma omp atomic
```

declarations at appropriate places. Nothing else needed to be modified and the results are shown in table 4. The additional speed-up was obtained only for the ODP4X variants, which intrinsically contain more parallelism than the ODP2 one. The additional speed-up was "only" slightly above a factor of 3, which is not that much considering the 8 available cores but quite reasonable considering the very low investment in software engineering required. Multi-threading did not change at all the result; only the reported maximum allocated memory in the output log (to stdout) is slightly increased due to the need of having some private memory buffers for the threads.

	Exec. time (s)			Speed-up		
	gcc	icc-	icc	gcc	icc-	icc
ODP2	25.8	12.5	10.7	46	96	112
ODP4S	50.7	35.3	33.2	248	356	379
ODP4M	50.0	34.5	31.4	240	347	382

Table 4. Execution time and speed-up relative in the reproduction, threaded version

7 Conclusion

I successfully reproduced the results obtained for the ODP-PIV method presented in our 1998 paper in Experiments in Fluids. Not much difficulty was encountered in the reproduction. This was greatly due to the fact that the C language was not discontinued since then and that it was kept highly backward-compatible. This was also due to the fact that the ODP-PIV was a standalone software without any dependency on any other tool or library. That was 22 years ago though and I don't think that something significant can nowadays be achieved using such a completely autonomous approach. That reproduction also gave a milestone regarding the progress in brute computing performance on this task, indicating a 100+ speedup just thanks to progress in architecture and compilers, a 360+ speed-up exploiting the new multi-threaded capabilities with an extremely small engineering investment and finally a 1000+ factor in brute computing power per euro. Overall, this reproduction, including additional experiments and the writing of this paper, required about two full-time weeks.

References

1. G. Quénot, J. Pakleza, and T. Kowalewski. "Particle Image Velocimetry with Optical Flow." In: **Experiments in Fluids** 25.3 (1998), pp. 177–189.
2. G. M. Quénot. "The 'orthogonal algorithm' for optical flow detection using dynamic programming." In: **[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing**. Vol. 3. 1992, 249–252 vol.3.
3. G. Quénot. "Computation of Optical Flow Using Dynamic Programming." In: **Proceedings of IAPR Workshop on Machine Vision Applications, MVA 1996, November 12-14, 1996, Tokyo, Japan**. 1996, pp. 249–252.