remote sensing

MDPI

Article

Enabling Image-Based Streamflow Monitoring at the Edge

Fabio Tosi [1], Matteo Rocca [1], Filippo Aleotti [1], Matteo Poggi [1], Stefano Mattoccia [1], Flavia Tauro [2], Elena Toth [3] and Salvatore Grimaldi [2,4,*]

1 Department of Computer Science and Engineering, University of Bologna, 40136 Bologna, Italy; fabio.tosi5@unibo.it (F.T.); matteo.rocca5@studio.unibo.it (M.R.); filippo.aleotti2@unibo.it (F.A.); m.poggi@unibo.it (M.P.); stefano.mattoccia@unibo.it (S.M.)

2 Department for Innovation in Biological, Agro-food and Forest Systems, University of Tuscia, 01100 Viterbo, Italy; flavia.tauro@unitus.it

3 Department of Civil, Chemical, Environmental, and Materials Engineering, University of Bologna, 40136 Bologna, Italy; elena.toth@unibo.it

4 Department of Mechanical and Aerospace Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 10003, USA

* Correspondence: salvatore.grimaldi@unitus.it; Tel.: +39-0761-357326

Received: 28 April 2020; Accepted: 15 June 2020; Published: 25 June 2020

Abstract: Monitoring streamflow velocity is of paramount importance for water resources management and in engineering practice. To this aim, image-based approaches have proved to be reliable systems to non-intrusively monitor water bodies in remote places at variable flow regimes. Nonetheless, to tackle their computational and energy requirements, offload processing and high-speed internet connections in the monitored environments, which are often difficult to access, is mandatory hence limiting the effective deployment of such techniques in several relevant circumstances. In this paper, we advance and simplify streamflow velocity monitoring by directly processing the image stream in situ with a low-power embedded system. By leveraging its standard parallel processing capability and exploiting functional simplifications, we achieve an accuracy comparable to state-of-the-art algorithms that typically require expensive computing devices and infrastructures. The advantage of monitoring streamflow velocity in situ with a lightweight and cost-effective embedded processing device is threefold. First, it circumvents the need for wideband internet connections, which are expensive and impractical in remote environments. Second, it massively reduces the overall energy consumption, bandwidth and deployment cost. Third, when monitoring more than one river section, processing "at the very edge" of the system efficiency improves scalability by a large margin, compared to offload solutions based on remote or cloud processing. Therefore, enabling streamflow velocity monitoring in situ with low-cost embedded devices would foster the widespread diffusion of gauge cameras even in developing countries where appropriate infrastructure might be not available or too expensive.

Keywords: streamflow velocity; optical tracking velocimetry; computer vision; embedded system

1. Introduction

Although river discharge observations are essential for hydrological studies and related practical applications, accurate measurements in diverse flow regimes and in difficult-to-access locations are still unfeasible with standard instrumentation [1,2].

In recent years, several contributions have demonstrated that image-based analysis approaches for estimating the flow surface velocity are promising alternatives to traditional measurement [3]. Since the

pioneering work by [4], optical techniques previously developed in fluid dynamics laboratories have been successfully implemented to the observation of the surface flow velocity field of natural rivers. Among many diverse applications, image-based methodologies have been instrumental to estimate flow discharge through fixed and mobile setups [5–7], to survey floods from crowdsourced data [8–10], and to investigate complex flow structures in natural streams [11]. In addition, embedded system solutions for continuously monitoring river flows have been explored by some hydrometric agencies and environmental monitoring companies.

While it was proved that this innovative image-based approach is reliable and affordable, state-of-the-art algorithms recently evaluated in [12], have some severe constraints that still hamper their widespread diffusion. In fact, they are rather computational demanding, and this fact dictates the use of power-hungry architectures, thus preventing in most cases their practical deployment in situ due to energy and cost requirements. Therefore, to tackle this issue, such approaches typically rely on offload processing leveraging wideband internet connections to transfer the whole image stream to a remote facility for cloud processing. Such a strategy is however quite costly and requires a wideband internet connection that might be limited or not available at all. The outlined difference between power-hungry offload and power-efficient approaches is summarized in Figure 1, showing the acquisition of the images by the camera, the identification of the trajectories (either by the local low-power embedded system or through offload cloud processing) and the resulting information (velocity estimates). On the left, the in situ processing strategy developed in this paper that leverages a possibly solar powered embedded system with limited connectivity is displayed and, on the right, the current solution based on cloud processing and a high-speed data connection is depicted.

An additional significant limitation of the cloud-based solution consists of its poor scalability, that is, the difficulty in controlling a cluster of multiple monitoring devices in different river sections, see Figure 2. In situ river flow monitoring with a low-power system would mitigate most of the above-mentioned criticalities and encourage the diffusion of image-based methodologies in diverse environments . To this end, in this paper, we propose a cost-effective solution based on a lightweight embedded system that yields an accuracy equivalent to state-of-the-art solutions with a fraction of their power budget. Moreover, this strategy is almost agnostic to the communication infrastructure available in remote environments: a simple SMS text message, or other available low-bandwidth communication solutions would suffice to transfer the processed velocity estimates to a central server.

Therefore, our system would enable capillary streamflow surface monitoring without cumbersome and costly infrastructures seldom available in remote sites. This would also afford additional analysis through a massive cluster of low-cost monitoring devices as sketched in Figure 2. Finally, such a low-budget solution would also foster its deployment in developing countries often even more deeply affected by the limitations of power-hungry and expensive systems.

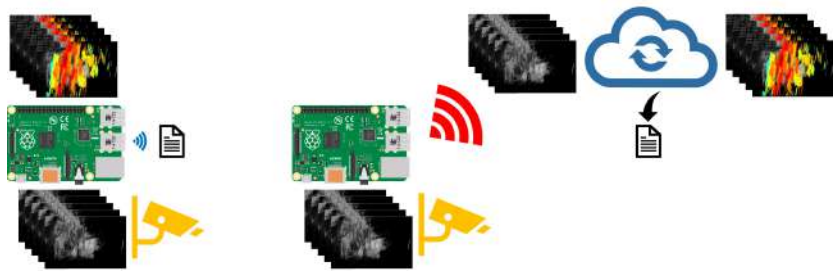

Figure 1. (**Left**) In situ processing with an embedded system as proposed in this work, (**Right**) offload cloud processing strategy deployed by conventional solutions. The latter requires a high-bandwidth data connection, not always available in remote monitored places, to transfer the image stream gathered in the field to a remote location for deferred processing.

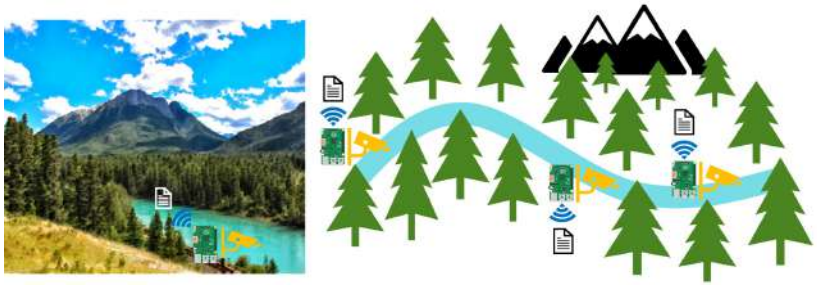

Figure 2. Sketch of a cluster of monitoring devices installed along a river located in a remote place with constrained/limited data connectivity.

2. Related Work

In this section we review the literature concerning image-based techniques and systems for image-based techniques for river flow velocity estimates. Flow visualization and quantitative characterization dates back to laboratory-based particle image velocimetry (PIV), a fluid dynamics technique originally developed by [13–15]. PIV is a correlation-based algorithm that enables the reconstruction of the flow velocity field from the similarity of consecutive images captured through a high-speed camera. The flow is densely seeded with neutrally buoyant tracers whose highly reflective surface is clearly illuminated with a laser system. The displacement vectors of tracer patterns of pixels are computed in image sequences, and the camera acquisition frequency is utilized to compute instantaneous velocity vectors. A similar approach was then applied in [4] to the analysis of the surface flow velocity field of rivers. In fact, large scale particle image velocimetry (LSPIV) implements a high-speed cross-correlation scheme between interrogation areas and larger search regions. Similar to traditional PIV, each image is divided into a grid and the location of the maximum value of the cross-correlation coefficient in consecutive frames yields displacement vectors. However, the LSPIV setup is much simpler than laboratory PIV and may include a standard RGB camera and, in some cases, filters and polarizers to mitigate the effect of solar reflections. Buoyant particle tracers of several centimeters are typically deployed onto the stream surface to facilitate flow pattern recognition. In some implementations, simple laser-based systems have been introduced to remotely assign metric dimensions to image pixels [16].

In case of sparsely seeded surfaces, several techniques can be considered as valid alternatives to LSPIV. The space-time image velocimetry (STIV) method exploits the brightness distribution on the stream surface to measure streamwise flow velocity [17]. Different from STIV, particle tracking velocimetry (PTV) leads to the reconstruction of the trajectories of actual objects transiting in the field of view. PTV encompasses particle identification and tracking, and can be executed through an array of diverse algorithms, with cross-correlation the most commonly implemented [18–20]. When combined with trajectory-based filtering, PTV has led to accurate surface flow velocity estimations [21]. Traditional PTV works well with unsteady and poorly seeded flows, which is the usual condition in real-world experimental settings, with no artificial seeding. However, the algorithm efficacy relies on the presence of particles of a priori known shape in highly-defined images. In [22], a nearest-neighbor PTV approach enables the identification and tracking of features of any shape, thus partially alleviating the need for tracer deployment.

Stream surface flow velocity in the absence of artificial tracers can be effectively achieved through optical flow approaches [23]. In environmental monitoring, the Lucas-Kanade algorithm has been adopted to investigate flows above a stepped chute [24] and a flash flood event [10]. In [25], a novel optical flow approach, entailing automated feature detection, named optical tracking velocimetry (OTV) was proposed, based on tracking through the differential sparse Lucas-Kanade algorithm, and a posteriori trajectory-based filtering. OTV proved successful in an "offline" approach analyzing videos recorded in natural streams where it was applied to reconstruct the surface flow velocity field in case of low flow conditions and high seeding density in a natural river as reported in [12].

3. Optical Tracking Velocimetry Algorithm

The Optical Tracking Velocimetry (OTV) algorithm [25] estimates surface flow velocity by tracking sparse features leveraging across two consecutive image frames I_t and I_{t-1} captured at time t and $t-1$, respectively as depicted in Figure 3. Although substantially agnostic to the image feature extracted from the input cue, OTV relies on the FAST feature detector [26] due to its good trade-off between accuracy and computational complexity as reported for surface flow velocity estimation in [25]. The middle row of Figure 3 reports the FAST features, superimposed to the original image, extracted from the two consecutive frames shown in the top row. Once the sparse features are extracted from the input images, OTV tracks them deploying the Lucas-Kanade algorithm [27], as outlined in Figure 4. Such algorithm is a widely used differential methodology that computes optical flow for a sparse set of features assuming that the displacement of the image content between two frames is approximately constant and small within a neighborhood of the feature taken into account. The peculiar pyramidal data structure employed by this algorithm, keeping fixed the search area at each image resolution starting from the coarse one, enables to manage large displacements across consecutive frames at a reduced computational cost as outlined in Figure 4.

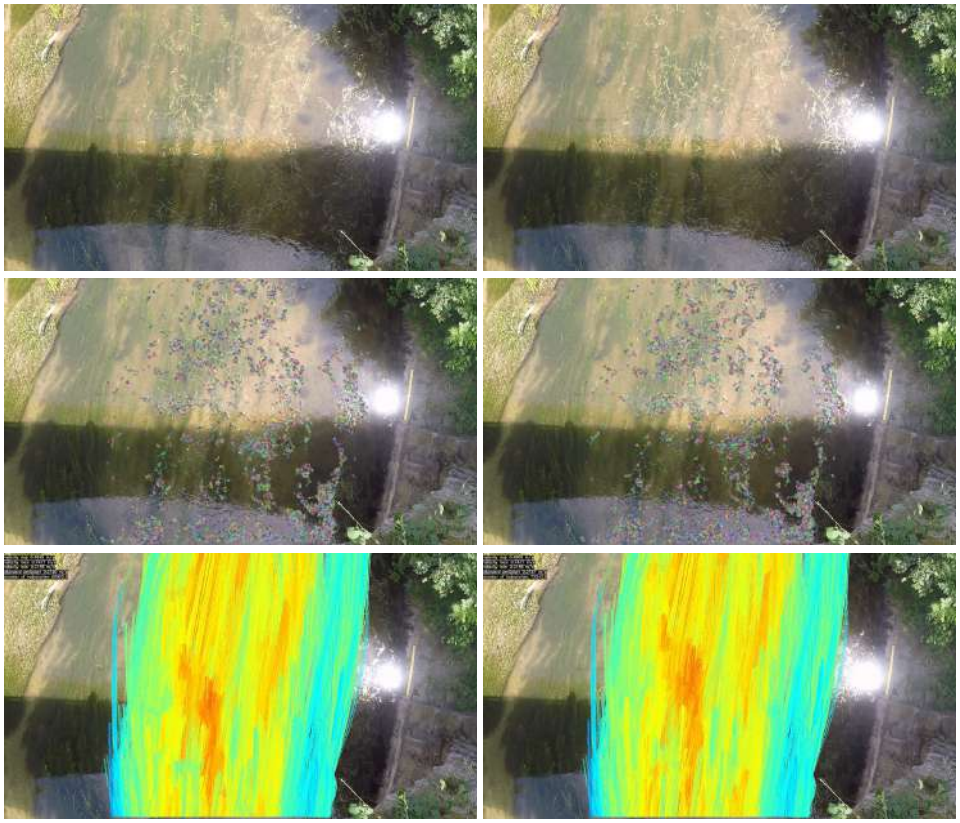

Figure 3. The outcome of OTV on two frames, I_{t-1} (**left**) and I_t (**right**), of a sequence captured in the Brenta river, in Italy. From top to bottom: input frame, feature points detected by the FAST algorithm highlighted with coloured circles, and estimated trajectories (each colour encodes a different velocity—The warmer, the faster).

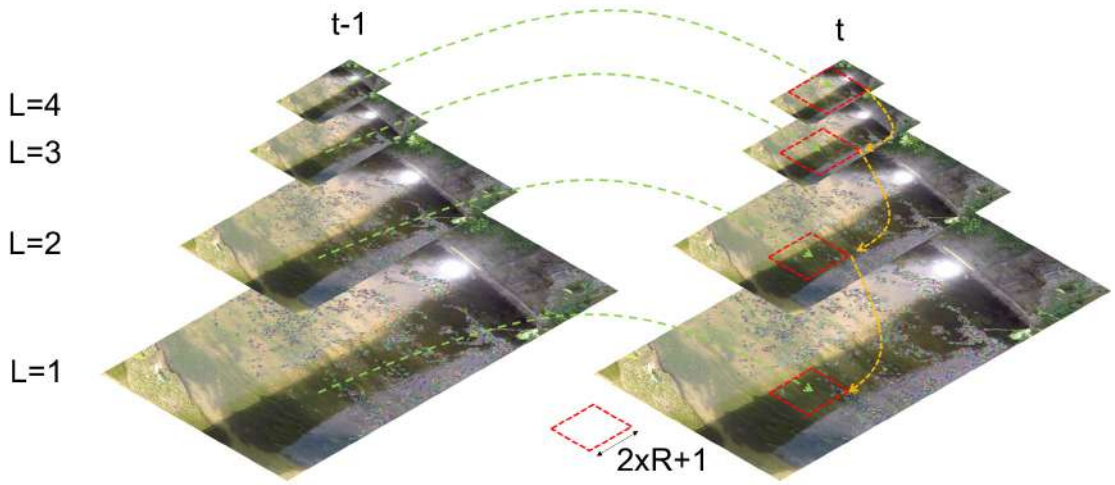

Figure 4. Overview of the tracking strategy used by the OTV algorithm. The FAST features extracted from the input images are used to identify corresponding points across two adjacent frames I_{t-1} and I_t, employing the Lucas-Kanade algorithm. At each pyramid level L, the corresponding point is searched within an area of radius R.

Once feature correspondence is achieved, OTV adopts a filtering strategy aimed at minimising the occurrence of unrealistic trajectories according to the method proposed in [21]. Indeed, considering that extracted features are supposed to travel along the main direction of the river flow (in our setup orthogonal to the river cross-section), several assumptions regarding spatial motion can be made. Among them, a set of filtering constraints based on length and slope allows retaining reliable trajectories only. More specifically, such a strategy imposes that two feature vectors $\vec{I}_t$ and $\vec{I}_{t-1}$ of each pair of consecutive frames should follow the rule:

$$\arctan\left[\frac{(\vec{I}_t - \vec{I}_{t-1}) \cdot \mathbf{j}}{(\vec{I}_t - \vec{I}_{t-1}) \cdot \mathbf{i}}\right] > \alpha \tag{1}$$

where **j** and **i** represent unit vectors orthogonal and parallel to the river cross-section, while α represents the validity angle. Moreover, unreliable trajectories are further discarded depending on their total slant and length. More specifically, we select only those that following conditions:

$$\arctan\left[\frac{(\vec{I}_{end} - \vec{I}_{start}) \cdot \mathbf{j}}{(\vec{I}_{end} - \vec{I}_{start}) \cdot \mathbf{i}}\right] > \beta, \qquad ||\vec{I}_{end} - \vec{I}_{start}|| > \gamma \tag{2}$$

whereby β and γ indicate the desired slant angle and the minimum trajectory length, respectively. Once highly confident trajectories are kept, the associated streamflow velocity is then computed as:

$$v = \frac{||\vec{I}_{end} - \vec{I}_{start}||}{\Delta t} \tag{3}$$

where Δt represents the interval time between $\vec{I}_{end}$ and $\vec{I}_{start}$. It is worth to highlight that each parameter α, β and γ are appropriately selected based on the scene of interest. Typical values are: $\alpha = 4/9\pi$, $\beta = \pi/4$ and $\gamma = ImageHeight/5$.

The bottom row of Figure 3 shows the trajectories estimated by the OTV algorithm.

Indeed, according to recent evaluations [12,25], the accuracy of OTV compares favourably to state-of-the-art methodologies, even if a much lower computational complexity characterises it. Nonetheless, in its original form, despite its reduced execution time compared to other approaches, it requires standard PCs or cloud processing to estimate streamflow velocity in a reasonable amount of time (minutes for a 20-s video sequence like the one mentioned in Figure 3). This fact still prevents its deployment on computing devices compatible with a remote site as the one envisioned

in Figure 2, typically constrained by size, cost and power budget. Therefore, an approach allowing image processing through OTV on a low-power embedded system, like the one deployed in our experiments and discussed next, is paramount to enable streamflow velocity estimation in the field.

4. Embedded Computing Architecture

Mostly driven by the smartphone industry, a broad range of embedded systems, characterized by low power consumption and cost, is nowadays available at competitive costs. Among them, the Raspberry PI family is a well-known cost-effective solution deployed in countless applications. It is exceptionally cheap; for instance, the 3B model (3B) used in our experiment costs less than 40$. It is also highly energy-efficient; for the same model, the overall power consumption is about 5 W using all its cores, enabling its deployment in low-power setups, including those relying on solar panels. Moreover, it has a large community, and it is continuously improved. Similar considerations apply to the more recent Raspberry PI 3B+ and 4 models. As notable improvements compared to the 3B model, the former has a slightly higher clock frequency and also features the possibility to be powered through power-over-ethernet. Model 4 sports further major improvements compared to the Raspberry 3 lines, with the most notable one a much more efficient ARM architecture (Cortex A72). However, compared to the Raspberry 3, both models are slightly more power-hungry, especially the 4 model, although significantly more powerful. Moreover, in addition to consuming more power, the Raspberry Pi 4 runs considerably hotter than the previous PI models, possibly leading to failure in embedded, solar-powered systems.

Although other embedded platforms with comparable features like Beagleboard, Intel Odyssey or Odroid exist, we decided to use the PI series since they have the largest community of users and developers. Nonetheless, porting our code to other systems like those previously highlighted would be straightforward. For our experiments, we selected the Raspberry PI 3B model. It is built around a Broadcom BCM2837B0 System on Chip (SoC), containing a four-core Cortex-A53 (ARMv8) 64-bit CPU clocked at 1.4 GHz and a lightweight Graphic Processing Unit (GPU) Broadcom Videocore-IV. The overall memory available is 1 GB LPDDR2 SDRAM. The device also sports standard network connectivity (Gigabit ethernet, Wi-fi 802.11ac (2.4 GHz and 5 GHz), Bluetooth 4.2), USB ports and 40 Input-Output connectors. We conducted experiments with the Linux operating system using the Raspbian distribution targeting this particular device. The overall size of the device is 85 mm $\times$ 56 mm $\times$ 17 mm. This very same setup device is currently installed on the Secchia river, in Italy. It also includes an AXI camera installed on a bridge. Unfortunately, in the current setup, the inability of the system to process images in situ needs an additional SSD drive connected to the Raspberry PI 3B for storing image sequences. For the same reason, since the internet connection available is not fast enough to transfer large image sequences, a human operator goes to the monitored river about once a month to retrieve them from an external and capacious SSD for offload processing with a standard PC in a laboratory. This procedure is not only time-consuming and cumbersome but it does not allow real-time monitoring, and cannot be used to trigger alarms when anomalous flow situations occur. In this work, we aim at overcoming all such limitations, including the need for the external SSD, by processing image sequences in the monitored remote place with a solar or even with a battery-powered embedded device.

5. Functional Optimization to the Baseline OTV Algorithm

The 1 GB RAM available in the Raspberry PI 3B was barely sufficient for running the OTV algorithm in its original implementation and the algorithm also performed many redundant operations including saving video results. Therefore, at first, we cleaned up the code (The source code is available upon request) to have a baseline version, yet equivalent to the original version in terms of accuracy, that could be deployable on our target platform. After this preliminary phase, the overall memory footprint was about 15 MB compared to the 680 MB of the original code. Despite these massive optimizations, in the original form, the overall execution time for a 20-s sequence was 44 s on a

high-end PC with a power consumption of more than 100 W and more than two hours on the Raspberry PI 3B. The former solution would be impractical due to the massive energy consumption with the typical constraint found at the very edge. Moreover, the cost of such a solution would be very high since a PC-class computing architecture is needed. Despite the Raspberry PI 3B solution is compatible with energy requirements and costs, its execution time is so high that it would yield only a measurement every more than two hours. Although this rate could be acceptable for a limited number of applications, on the other hand, it also means that the Raspberry would continuously drain, for more than two hours, a significant amount of energy to provide such a single measurement. In contrast, a faster approach characterized by similar or slightly higher power consumption would allow for significant energy saving since the Raspberry has a meagre power consumption when idle. Of course, a faster strategy would also allow for a higher monitoring rate if required by the specific application context.

Arguing these facts, starting from the baseline version referred to as OTV, we will describe the functional simplifications and computational optimizations carried out to achieve streamflow monitoring with a much lightweight approach. Specifically, we aim at its deployment in the field with the constrained computing architecture selected, enabling streamflow monitoring *at the very edge*. In the most experimentally challenging condition, we envision such setup made up of a Raspberry PI 3B, a camera, a battery coupled with a small solar panel and an unconstrained data communication channel. Specifically, for the latter purpose, a simple SIM enabling SMS messaging would suffice for transmitting the processed velocity estimates. All the components of such setup are commercially available as off-the-shelf devices at low cost.

In our analysis, we will consider the parameters of the OTV algorithm discussed next in order to find the best trade-off between accuracy and execution time.

5.1. Search Area

OTV tracks the FAST features across two consecutive video frames captured at time $t-1$ and t using the well-known Lucas-Kanade [27] algorithm through a coarse-to-fine approach to speed-up the computation. In detail, it scales the input images at different resolutions L according to a pyramidal data structure as the one depicted in Figure 4. In the OTV algorithm [25], parameter L is set to 3.

5.2. Pyramid Levels

Then, starting from the coarser level, the scheme looks for corresponding features between two adjacent frames within a patch of radius R. The size of the search area is kept constant across different pyramid levels to increase the search area at the lower resolutions. Figure 4 outlines the role of the constant search area across different levels of the pyramid data structure. In the OTV algorithm [25], R is set to 7. Herein, we assess the accuracy of OTV varying both L and R and measuring their impact on the execution time.

5.3. Number of Tracked Features

According to [25], the FAST [26] feature detector represents the best trade-off between accuracy and computational complexity. Figure 4 shows, superimposed to the input image at different scales, the output FAST features. Although the maximum number of points processed in each frame was originally set to 20,000 in the OTV algorithm [25], we also assessed how this parameter impacts the estimation of surface flow velocity. Purposely, in the experimental results section, we also provide detailed analysis regarding this relevant parameter to find the value enabling acceptable accuracy and the lowest execution time.

5.4. Frame Rate

Another simplification taken into account to speed-up computation consists of reducing the initial frame rate according to a parameter J. Setting $J = 1$ means the same input frame rate while $J = t$, with $t > 1$, means to process 1 out of t frames considered as adjacent. Specifically, we processed 1 out of n frames from the raw output video at the original frame rate provided by the camera. In our experiments, we processed video stream from an AXIS PI1365-E MK II network camera or a GoPro 341 Hero 4 Black edition cam. An accurate analysis, concerning the impact of J on accuracy and execution time, is reported in the experimental results section. Nonetheless, it is worth noting that the actual frame rate to be set for field experiments is constrained by multiple factors and thus it should be adjusted accordingly to achieve the best trade-off between computational requirements and accuracy. Some of the latter factors are: streamflow velocity, position of the camera, camera lens and imaging devices, and image resolution.

5.5. Video Resolution

A final simplification to speed up the OTV algorithm consists of using images at reduced resolution. Precisely, given the input video frame, we evaluate in each experiment the accuracy at Full, Half and Quarter resolution and the impact of this parameter on the overall execution time.

In all our experiments, we report results obtained only with the JPEG image encoding. Compared to other methods such as PNG, BMP and TIFF, JPEG reduces the memory footprint at least by a factor 20 inducing a notable reduction in the overall processing time of OTV. Moreover, JPEG encoding is also the fastest method among those considered enabling even a shorter conversion time from the input raw video sequence acquired by the camera.

6. Parallel Optimization Strategies

Modern CPUs, including low-power embedded systems such as our target device, have standard yet powerful computing capability to exploit data-level as well as thread-level parallelism to significantly improve the performance of data crunching algorithms. The two strategies are not mutually exclusive and, since the OTV algorithmic structure allows it, we jointly deploy both optimizations to achieve the best performance. Additionally, we also consider a further optimization strategy exploiting the low-power GPU integrated in SoC of the Raspberry PI 3B. For all our experiments, we used standard open-source libraries, and the widespread diffusion of the Raspberry PI facilitates this task. In particular, we used the highly optimized OpenCV library [28] for computer vision and image processing operations in its two versions 2.4.13 and 4.1.

Individually, we evaluate the three following optimizations of the original OTC algorithms to make it suited for in situ processing with the Raspberry PI 3B.

6.1. Improving Parallelism through Single Instruction Multiple Data (SIMD) Instructions

Data-level parallelism acts inside each CPU by performing the same operation on multiple data with a single instruction, referred to as Single Instruction Multiple Data (SIMD), thus enabling parallel computational operations. The advantage of this specific optimization compared to the standard single processing strategy can be perceived in the first row of Figure 5. A single instruction, can process a group of data as shown in the right of the figure. The maximum (theoretical) speed-up achievable, by each processor, with SIMD instructions, is not easily predictable and depends on many factors. They are: the number of operations of the overall code mapped on such instructions, the type of data deployed for processing, the size of the SIMD registers and others, but the number of data per instruction certainly bounds computational time. The Cortex-A53 (ARMv8) sports the NEON advanced SIMD instruction set [29], with registers of size 128-bit deployable as sixteen 8-bit data, eight 16-bit data, four 32-bit data or two 64-bit data. The lower the data type, the higher is the speed-up potentially

achievable. We increased the parallelism on every single CPU core leveraging the NEON-SIMD instructions available in the ARM Cortex-A53 core.

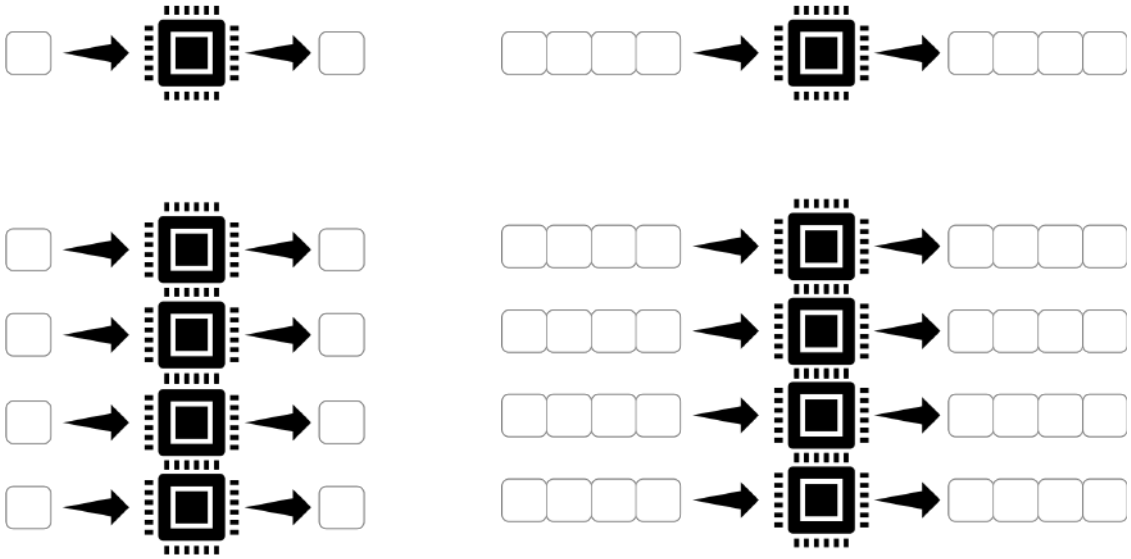

Figure 5. Parallel computing paradigms exploitable in multi-core CPUs such as the one inside the SoC of the Raspberry PI 3B. (**Top-Left**) No optimization, (**Top-Right**) single instruction multiple data (SIMD), (**Bottom-Left**) Multi-threading, (**Bottom-Right**) SIMD and multi-threading.

6.2. Improving Parallelism through Multiple CPU

In contrast to the previous SIMD approach, thread-level parallelism aims at improving performance exploiting the multiple cores available in modern CPUs to split computation across multiple processing units. Compared to the basic single core single-instruction processing (Top-Left of Figure 5), distributing computations across multiple cores enables to speed-up data computation as can be perceived in the Bottom-Left sketch of Figure 5. Specifically, we exploited the four Cortex-A53 cores available in the SoC of the Raspberry PI 3B to distribute the processing load of the OTV algorithm across the four cores to reduce the overall computation. With this parallel computation paradigm, the maximum speed-up achievable is upper bounded by the number of available cores, the nature of the algorithm and its data structure. In our specific case, since the two latter constraints are compatible with this processing paradigm, the maximum speed-up achievable is 4 since there are four physical cores in the Raspberry SoC. Indeed, this is a very efficient strategy to reduce the execution time on multi-core CPUs, nowadays available in almost any processing system, including low-power embedded systems. Concerning implementation details, there are two distinct solutions to achieve this parallelism strategy, both supported by OpenCV, one based on OpenMP [30] and the other one on Thread Building Blocks (TBB) library [31]. We have considered both. It is worth noting that multi-threading, can be seamlessly coupled with SIMD, as sketched in the Bottom-Right scheme of Figure 5.

6.3. Improving Parallelism through the Graphic Processing Unit

Finally, the SoC of the Raspberry PI also features a low-power GPU mostly designed to accelerate visualization operations on external monitors. Such a device is not comparable to high-end and power-hungry GPUs, available in desktop PCs, designed for video gamers and researchers. Nonetheless, the low-power GPU of the Raspberry PI could potentially allow achieving a further speed-up since it is capable of parallel operations and purposely, we investigated this opportunity too. Although different, the computational paradigm at the core of this approach is somehow similar to

the combination of SIMD and multi-threading outlined in the Bottom-Right scheme of Figure 5 but carried out on multiple computing units inside the GPU.

With this kind of parallel processing paradigm, the speed-up is bounded by the number of computing units available in the GPU, the computational structure of the algorithm and its data. Although the two latter constraints fit well with our algorithm, predicting the achievable speed-up is not trivial since there are significant issues concerning data movement back and forth the GPU that can significantly impact the overall performance. OpenCV 4.1 allows us to take advantage of the parallel computing paradigm enabled by the GPU and we investigate its impact on the performance in our experiments.

7. Experimental Evaluation

Our experimental evaluation consists of two distinct parts using, at first, an image sequence of 20 s, acquired at 25 FPS on the Brenta river in Italy at three resolutions: Full (1430 $\times$ 1080), Half (715 $\times$ 540) and Quarter (357 $\times$ 270) pixels. The original sequence, encoded as AVI file, is of the size of about 100 MB and before processing, all the 500 video frames are converted to JPEG images. Each image has approximately the size of 205 KB, and the whole conversion requires about 30 s, substantially equivalent at each resolution. Nonetheless, it is worth noting that such a small overhead would vanish deploying a camera capable of saving directly images rather than videos.

We first assess the impact of the functional simplifications in order to determine the best trade-off between accuracy and execution time using as reference the baseline OTV algorithm. Then, starting from the best configuration found in the previous phase, we compare its performance to the baseline OTV algorithm leveraging the parallel computing capabilities available in the Raspberry PI 3B. Finally, we report additional experimental results with image sequences gathered in other rivers in Italy with the best configuration found in the first part of our experimental analysis. In our experiments, for a given video sequence, we report both the average velocity and standard deviation computed by averaging values obtained from Equation (3) with the baseline OTV algorithm [25] and its optimized counterpart proposed in this paper.

7.1. Assessment of Functional Simplifications

In this section, we first evaluate the accuracy of the OTV algorithm varying its functional parameters to find the best trade-off between accuracy and execution time. Then, given the best cost-effective setting referred to as OTV-Opt, we further reduce its execution time exploiting the parallel computing capability of the Raspberry PI 3B.

In a trial-and-error process, we have tested different values of the following three parameters of the Lucas-Kanade approach at the core of the OTV algorithm to find corresponding points across two adjacent frames: the radius of the search window $(2, ..., 10)$, the number of pyramid levels $(1, 2, 3)$, and the maximum number of extracted features to track $(5000, 10,000, 15,000, 25,000, 50,000)$ at a given time. Moreover, we also evaluate a further functional optimization consisting in reducing the frame rate. Overall, evaluating more than 1500 different configurations, we found out that the best trade-off between computation accuracy and complexity consists of the following parameters: 4 as the radius of the search area, 3 pyramid levels and a maximum of 15000 FAST trajectories to track for each pair of frames. We also discovered that halving the frame rate yields a significant computational saving without substantially compromising the accuracy. Moreover, we noticed higher execution times increasing the number of pyramid levels to 4.

Figure 6 reports the detailed outcome of our evaluation. Therein, we compare the performance of the original OTV algorithm with its optimized counterpart obtained at the end of the outlined functional analysis. Specifically, at the three resolution Full (F), Half (H) and Quarter (Q), we compare the number of trajectories, the mean value and the standard deviation of the inferred streamflow velocity and the execution time on the Raspberry PI 3B. As previously pointed out, the processing

frame rate should be adjusted to achieve computational efficiency as well as to fit with the specific constraints of the monitoring site and setup.

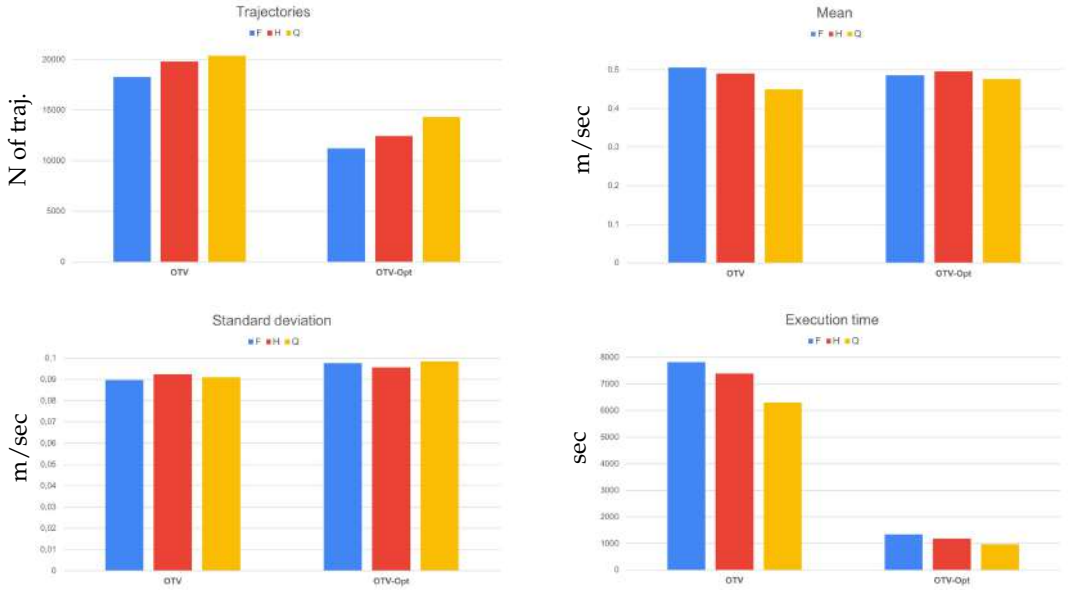

Figure 6. Comparison between the original OTV algorithm and its optimized counterpart obtained by the functional analysis at three resolutions on a sequence acquired on the Brenta river. (**Top-left**) Estimated trajectories, (**Top-Right**) average velocity, (**Bottom-Left**) standard deviation of velocity, (**Bottom-Right**) overall execution time on the Raspberry PI 3B without additional optimizations.

In Figure 6, we can notice that the number of trajectories estimated by the optimized version is substantially reduced. Moreover, for both versions of the OTV algorithm, the number of trajectories increases reducing the image resolution. Looking at the mean and standard deviation of the estimated velocity, the difference between the two methods is marginal. On the other hand, the impact on the execution time of the functionally optimized OTV-Opt algorithm is dramatic, yielding, at each resolution, a speed-up always around 6. This outcome proves that OTV-Opt is a much faster, yet substantially equally performing strategy to estimate streamflow velocity. Despite this remarkable improvement, which enables to reduce the execution time from more than 2 h to about 22 min at full-resolution and half resolution and from 105 min to about 16 min at quarter resolution, respectively, the running time in each configuration is still too high for most practical applications.

In the top left-hand panel of Figure 6, we can notice that the number of trajectories estimated by the optimized version is substantially reduced. Moreover, for both versions of the OTV algorithm, the number of trajectories increases reducing the image resolution. Figure 7 provides a comparison between the original OTV algorithm and its functionally optimized counterpart OTC-Opt on the three frames 100, 200 and 300 of the sequence acquired on the Brenta river. From the images, we can notice a reduced number of trajectories but a similar outcome in the same regions.

In order to further validate the settings of the OTV-Opt configuration, we compare its performance to OTV on another sequence, of 33 s and 1920 $\times$ 1080 resolution, acquired on the Tevere river in Italy. Figure 8 reports the outcome of this analysis. Therein, we can notice that the same set of parameters selected in the previous experiments yields a very similar trend concerning the number of estimated trajectories. Nonetheless, in this case, the highest value is achieved at the half resolution for both methods. Concerning the estimated streamflow velocity, we can notice minimal differences between OTV and OTV-Opt with fluctuations, encoded by the standard deviation, even smaller than OTV at F and H resolution, while at Q OTV-Opt has higher variability. The execution time, measured on a standard PC equipped with an Intel 7700K at 4.2 GHz, shows a considerable saving by OTV-Opt compared to the original OTV algorithm. Moreover, the same figure shows also the results

obtained when fine-tuning the OTV-Opt parameters to achieve the optimal trade-off between accuracy and execution time on this very video sequence referred to as OTV-Opt-Tuned. Such tuned parameters allow on the same platform a further notable reduction of the execution times. In addition, the tailoring of the parameters for the Tevere case study allows to get, also for the Q resolution, values of the standard deviation of the velocity that are comparable with those of the original OTV algorithm.

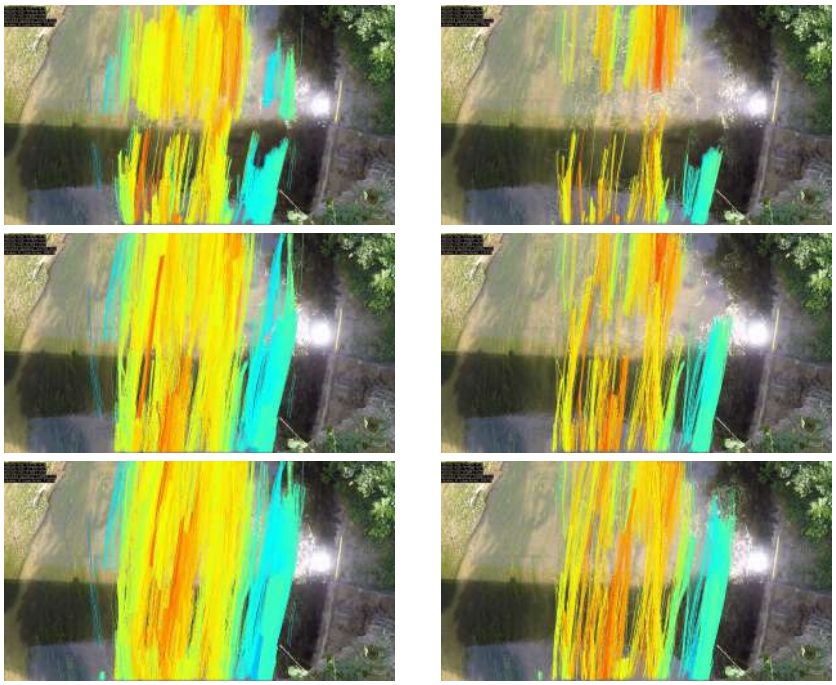

Figure 7. Comparison between the original OTV algorithm, on the left, and its optimized counterpart obtained by the functional analysis, on the right, on three frames (100, 200 and 300) of the 20-s sequence acquired on the Brenta river in Italy.

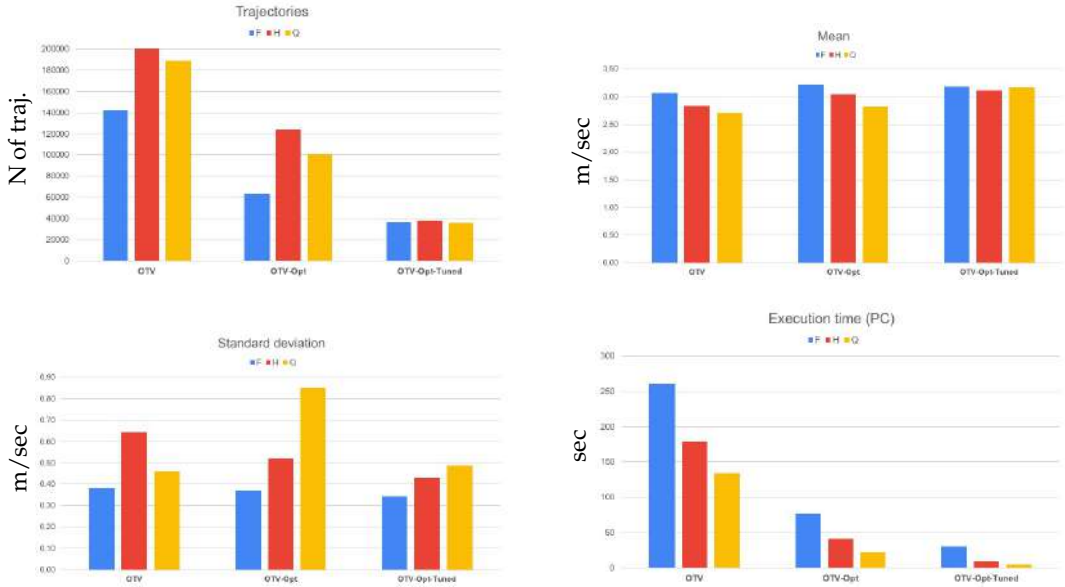

Figure 8. Comparison between the original OTV algorithm and its optimized counterpart obtained by the functional analysis at three resolutions on a sequence acquired on the Tevere river. For the optimized version, we report the outcome using the same optimal parameters found for the Brenta video (OTV-Opt) and optimal parameters found for the Tevere river (OTV-Opt-Tuned). (**Top-left**) Estimated trajectories, (**Top-Right**) average velocity, (**Bottom-Left**) standard deviation of velocity, (**Bottom-Right**) overall execution time on a standard PC (Intel 7700K at 4.2 GHz) without additional optimizations.

In Figure 9, the output trajectories are shown on three frames (100, 200 and 300) of the sequence acquired on the Tevere river. Similar to results for images of the Brenta river, even if a smaller number of trajectories is estimated with OTV-Opt, the algorithms exhibit comparable accuracy in streamflow velocity estimation.

The preliminary evaluation on a standard PC was valuable for setting up the optimization on the Raspberry, to be discussed in the next section. Starting from the evidence that OTV-Opt compares similarly to OTV at a reduced execution time, in the next section, we will analyze the impact of computational optimizations feasible on the power-efficient Raspberry PI 3B.

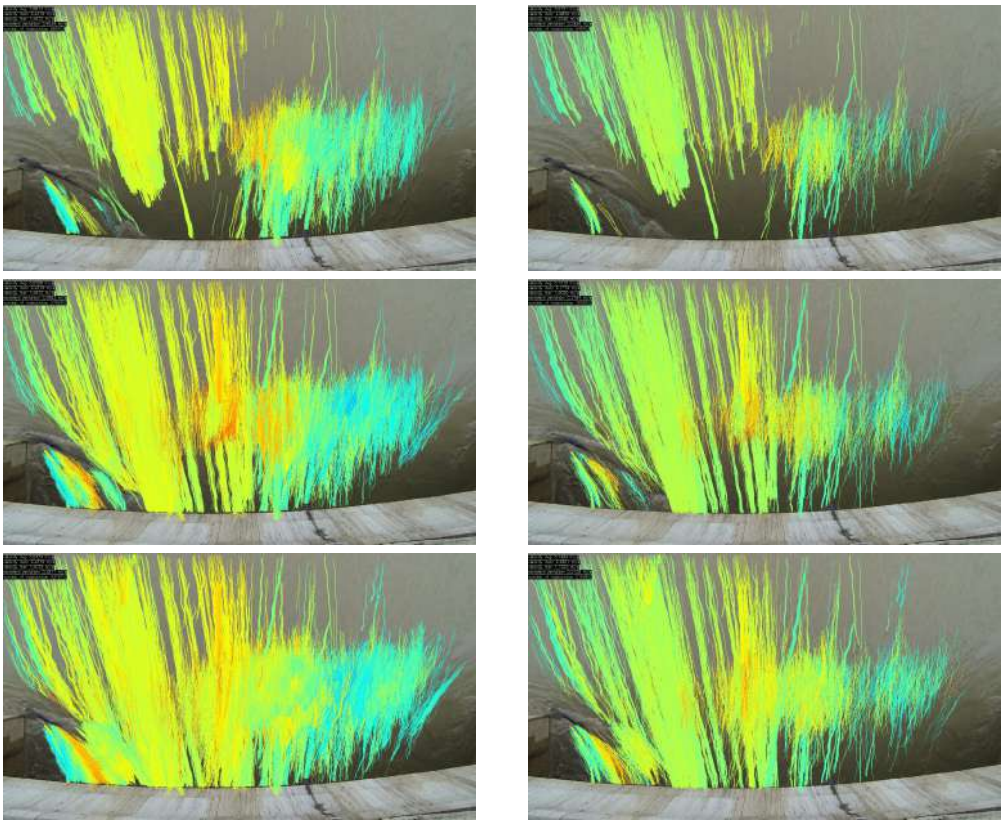

Figure 9. Comparison between the original OTV algorithm, on the left, and its optimized counterpart obtained by the functional analysis, on the right, on three frames (100, 200 and 300) of the 33-s sequence acquired on the Tevere river in Italy.

7.2. Assessment of Parallel Computation Strategies

In this section, we report the impact of the different optimization schemes previously outlined (with a Raspberry PI 3B) using as baseline the OTV-Opt version outlined before. Moreover, as for the previous functional evaluation, we use the same image sequence of 20 s, acquired at 25 FPS on the Brenta river in Italy, consisting of 500 images at 1430×1080 resolution.

From the panel i of Figure 10, despite the functional optimization previously described, we can notice that the baseline OTV-Opt approach requires more than 22 min on the Raspberry PI 3B to process the whole sequence at full resolution F and more than 16 min at the smallest resolution Q. Such execution times are not acceptable in most practical applications and would drain energy for a long time to infer a single velocity estimation.

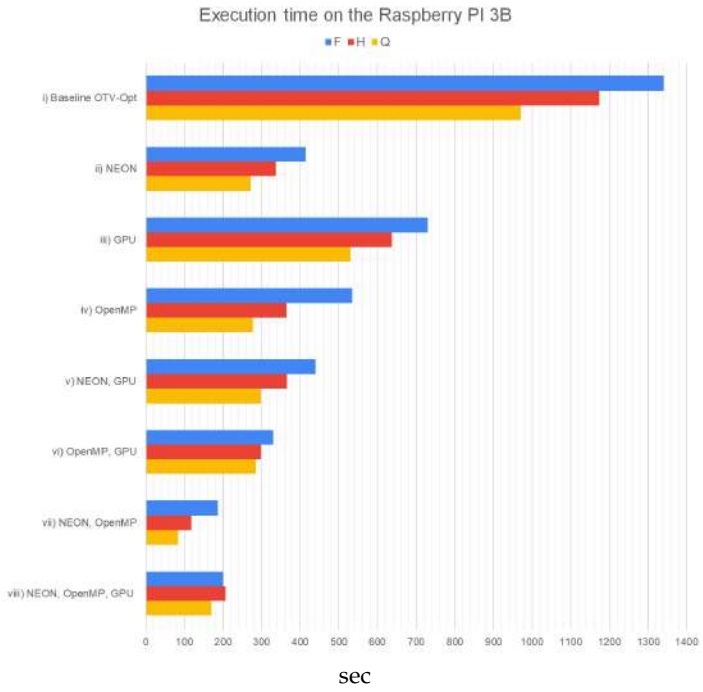

Figure 10. Execution time in seconds of the OTV-Opt algorithm on the Raspberry PI 3B at the three resolutions F, H and Q with different optimization strategies: (i) the vanilla baseline OTV-Opt with a single core without further optimizations, (ii) single core with NEON instructions, (iii) single core with the integrated GPU, (iv) using all the four cores with OpenMP, (v) single core with the integrated GPU exploiting NEON instructions, (vi) four cores and the GPU, (vii) using all the four cores with OpenMP exploiting NEON instructions, and finally (viii) combining all optimizations (GPU, NEON and OpenMP).

Observing the impact of NEON instructions, even with a single core as reported in panel ii, we can notice a remarkable speed-up at each resolution: the execution time of OTV-Opt drops at less than 7 min and 4.5 min at F and Q resolution, respectively. On the other hand, the impact of the integrated GPU of the Broadcom Soc (panel iii) on the execution time is not as good as with NEON instructions. The degradation in performance is likely due to data/instruction transfer slowdowns between the SOC and the GPU. In contrast, the impact of multithreading enabled by OpenMP (panel iv) is much better than using GPU, yet smaller as magnitude than that achieved with NEON instructions using a single core.

Panel v shows that the addition of the GPU to the NEON instructions is detrimental at all resolutions, whereas its use along with OpenMP (panel vi) decreases the execution times at full and half resolution, but not for the Q case (see comparison with panel iv).

The most effective optimization consists of deploying both strategies (panel vii: NEON, OpenMP), thus processing the whole 20-s sequence on the Raspberry in 3.11 min at F, less than 2 min at H and 1.24 min at Q resolution. This is a notable speed-up compared to the baseline OTV-Opt of about 7, 10 and 11 at F, H and Q, respectively. Moreover, compared to the original OTV algorithm, the execution time drops from 1340 s (more than 2 h) to 187 s (slightly more than 3 min) at full-resolution with a speed-up more significant than 40. Similar considerations, with different magnitude, apply for the other two resolutions H and Q. As already pointed out, this improvement comes at a small increase in the power consumption and, since the whole processing lasts only a fraction of the time, the overall energy drained for a single measurement is lower. Of course, this would also enable a higher measurement rate if required.

As reported in panel viii of Figure 10, the integrated GPU also deteriorates the performance of the combined NEON+OpenMP solution.

According to our analysis, using all the four cores with NEON instructions, as outlined at the bottom-right of Figure 5, yields the best overall performance at each resolution. Specifically, on the target device, OTV-Opt allows us to achieve almost 20 measurements per hour at the highest resolution and more than 40 at the lowest one with an accuracy comparable to the original OTV algorithm.

Finally, we report for OTV-Opt the execution time measured processing the Tevere sequence with the Raspberry PI 3B. Using the same optimal parameters found for the Brenta river (OTV-Opt), we measured 620.4, 476.6 and 234.3 s, respectively, at F, H and Q resolutions. Fine-tuning the parameters of OTV-Opt for the Tevere sequence, we measured (OTV-Opt-Tuned), for the same configurations on the same device and with comparable accuracy: 332.8, 112.7, 42.84 s. Thus, the execution time can be further reduced by fine-tuning OTV-Opt for the specific target environment/setup.

8. Experimental Evaluation Summary

In this section, we briefly summarize the outcome of our experimental evaluation aimed at making streamflow monitoring at the very edge with an Optical Tracking Velocimetry (OTV) approach applicable on a low-cost embedded device. First and foremost, we have shown how, through the functional optimization, we can achieve with OTV-Opt substantially the same degree of accuracy of the original OTV algorithm at a small fraction of its execution time. The impact of this strategy can be perceived comparing the charts at the left and in the middle of Figure 11. Then, coupling the functional optimization with the exploitation of data-level and multi-threading parallelism allows OTV-Opt to achieve the same goal at a further reduced execution time. The speed-up ranges from more than 7, at the full resolution, to almost 12, at the quarter resolution, as in the charts in the middle and the right in Figure 11. Nonetheless, it is worth noting that at the lowest resolution, the additional speed-up is sometimes, especially without any fine-tuning, obtained at the cost of slightly higher variance in mean velocity estimation as reported in Figure 8. Compared to the original OTV algorithm, the measured speed-up enabled by the two optimizations proposed in this paper ranges from more than 41 at full resolution to almost 75 at quarter resolution. Such a massive reduction in the execution time not only enables much higher measurement rates but also notably affects the overall power requirements. Indeed, the strategy proposed in this paper enables streamflow velocity monitoring on embedded devices like the Raspberry PI 3B and, thus, fosters its practical deployment for fully remote image acquisition and processing. To this aim, the OTV and OTV-Opt code are available upon request.

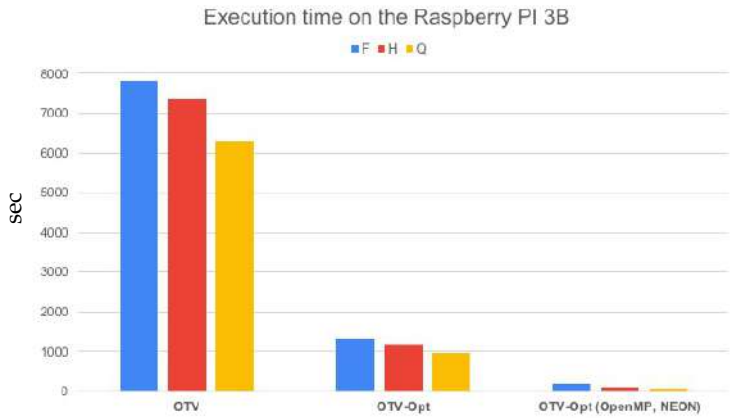

Figure 11. Execution time in seconds, at the three resolutions (F, H, Q), on the Raspberry Pi 3B. The original OTV algorithm [25] (**Left**), the optimized counterpart OTV-Opt obtained through the functional optimization (Middle), OTV-Opt exploiting the parallel computing capabilities of the target device (**Right**).

9. Conclusions

Nonintrusive monitoring of streamflow velocity is of paramount importance in several applications, and image-based approaches have proved successful alternatives to traditional measurement approaches. However, state-of-the-art systems have severe constraints regarding computational and energy requirements, and mandate offload processing and high-speed internet connections in the monitored environments, often located in remote places. We propose to achieve fully remote in situ streamflow velocity measurement by deploying a low-power and low-cost embedded OTV-based system, leveraging its parallel processing capability. A thorough evaluation confirms that our optimization strategies provide significant reduction of execution time and, most importantly, negligible differences with the original not-optimized algorithms in terms of average and standard deviation velocity. These results suggest that gauge-cam can directly process the image stream in situ paving the way its capillary diffusion. In turn, it will make available a large amount of case studies fostering further gauge-cam validation and uncertainty analysis in a variety of hydrological settings. We acknowledge that the results are dependent on the specific case studies and that the noise affecting real-world recordings may influence the performances and also the optimised parameters. The comparison of the two case studies in fact highlights that different sets of parameters provide the best results for the Brenta and for the Tevere rivers. However, gauge-cam are expected to be permanent installations in river cross-sections so a training period would allow to calibrate such parameters. Future research will allow to verify the parameter variability on more case studies and to design, if necessary, appropriate training procedures.

Author Contributions: Conceptualization, S.M., F.T., E.T. and S.G.; Data curation, Flavia Tauro; Investigation, S.M., Flavia Tauro, E.T. and S.G.; Methodology, F.T., M.R., F.A., M.P. and S.M.; Software, F.T., Matteo Rocca, F.A. and Matteo Poggi; Supervision, S.M. and S.G.; Writing—original draft F.T., M.R., F.A. and M.P.; Writing—review & editing, S.M., Flavia Tauro, E.T. and S.G. All authors have read and agreed to the published version of the manuscript.

Funding: Flavia Tauro acknowledges support by the "Departments of Excellence-2018" Program (Dipartimenti di Eccellenza) of the Italian Ministry of Education, University and Research, DIBAF-Department of University of Tuscia, Project "Landscape 4.0—Food, wellbeing and environment".

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tazioli, A. Experimental methods for river discharge measurements: Comparison among tracers and current meter. *Hydrol. Sci. J.* **2011**, *56*, 1314–1324. [CrossRef]
2. Tauro, F.; Selker, J.; vandeGiesen, N.; Abrate, T.; Uijlenhoet, R.; Porfiri, M.; Manfreda, S.; Caylor, K.; Moramarco, T.; Benveniste, J.; et al. Measurements and Observations in the XXI century (MOXXI): Innovation and multidisciplinarity to sense the hydrological cycle. *Hydrol. Sci. J.* **2018**, *63*, 169–196. [CrossRef]
3. Tauro, F. Particle tracers and image analysis for surface flow observations. *WIREs Water* **2015**, *3*, 25–39. [CrossRef]
4. Fujita, I.; Muste, M.; Kruger, A. Large-scale particle image velocimetry for flow analysis in hydraulic engineering applications. *J. Hydraul. Res.* **1997**, *36*, 397–414. [CrossRef]
5. Hauet, A.; Kruger, A.; Krajewski, W.; Bradley, A.; Muste, M.; Creutin, J.; Wilson, M. Experimental system for real-time discharge estimation using an image-based method. *J. Hydrol. Eng.* **2008**, *13*, 105–110. [CrossRef]
6. Kim, Y.; Muste, M.; Hauet, A.; Krajewski, W.F.; Kruger, A.; Bradley, A. Stream discharge using mobile large-scale particle image velocimetry: A proof of concept. *Water Resour. Res.* **2008**, *44*, W09502. [CrossRef]
7. Tauro, F.; Petroselli, A.; Porfiri, M.; Giandomenico, L.; Bernardi, G.; Mele, F.; Spina, D.; Grimaldi, S. A novel permanent gauge-cam station for surface-flow observations on the Tiber River. *Geosci. Instrum. Methods Data Syst.* **2016**, *5*, 241–251. [CrossRef]

8. LeBoursicaud, R.; Pénard, L.; Hauet, A.; Thollet, F.; LeCoz, J. Gauging extreme floods on YouTube: Application of LSPIV to home movies for the post-event determination of stream discharges. *Hydrol. Processes* **2015**, *30*, 90–105. [CrossRef]
9. LeCoz, J.; Patalano, A.; Collins, D.; Guillén, N.F.; García, C.M.; Smart, G.M.; Bind, J.; Chiaverini, A.; LeBoursicaud, R.; Dramais, G.; et al. Crowdsourced data for flood hydrology: Feedback from recent citizen science projects in Argentina, France, and New Zealand. *J. Hydrol.* **2016**, *541*, 766–777. [CrossRef]
10. Perks, M.T.; Russell, A.J.; Large, A.R.G. Technical Note: Advances in flash flood monitoring using unmanned aerial vehicles (UAVs). *Hydrol. Earth Syst. Sci.* **2016**, *20*, 4005–4015. [CrossRef]
11. Lewis, Q.W.; Rhoads, B.L. Resolving two-dimensional flow structure in rivers using large-scale particle image velocimetry: An example from a stream confluence. *Water Resour. Res.* **2015**, *51*, 7977–7994. [CrossRef]
12. Pearce, S.; Ljubičić, R.; Peña-Haro, S.; Perks, M.; Tauro, F.; Pizarro, A.; Dal Sasso, S.F.; Strelnikova, D.; Grimaldi, S.; Maddock, I.; et al. An Evaluation of Image Velocimetry Techniques under Low Flow Conditions and High Seeding Densities Using Unmanned Aerial Systems. *Remote Sens.* **2020**, *12*, 232. [CrossRef]
13. Adrian, R.J. Particle-imaging techniques for experimental fluid-mechanics. *Annu. Rev. Fluid Mech.* **1991**, *23*, 261–304. [CrossRef]
14. Adrian, R.J. Twenty years of particle image velocimetry. *Exp. Fluids* **2005**, *39*, 159 –169. [CrossRef]
15. Raffel, M.; Willert, C.E.; Wereley, S.T.; Kompenhans, J. *Particle Image Velocimetry. A Practical Guide*; Springer: New York, NY, USA, 2007.
16. Tauro, F.; Porfiri, M.; Grimaldi, S. Orienting the camera and firing lasers to enhance large scale particle image velocimetry for streamflow monitoring. *Water Resour. Res.* **2014**, *50*, 7470–7483. [CrossRef]
17. Fujita, I.; Watanabe, H.; Tsubaki, R. Development of a non-intrusive and efficient flow monitoring technique: The space-time image velocimetry (STIV). *Int. J. River Basin Manag.* **2007**, *5*, 105–114. [CrossRef]
18. Lloyd, P.M.; Stansby, P.K.; Ball, D.J. Unsteady surface-velocity field measurement using particle tracking velocimetry. *J. Hydraul. Res.* **1995**, *33*, 519–534. [CrossRef]
19. Brevis, W.; Niño, Y.; Jirka, G.H. Integrating cross-correlation and relaxation algorithms for particle tracking velocimetry. *Exp. Fluids* **2011**, *50*, 135–147. [CrossRef]
20. DalSasso, S.F.; Pizarro, A.; Samela, C.; Mita, L.; Manfreda, S. Exploring the optical experimental setup for surface flow velocity measurements using PTV. *Environ. Monitor. Assess.* **2018**, *190*, 460. [CrossRef]
21. Tauro, F.; Piscopia, R.; Grimaldi, S. Streamflow observations from cameras: Large-scale particle image velocimetry or particle tracking velocimetry? *Water Resour. Res.* **2017**, *53*, 10374–10394. [CrossRef]
22. Tauro, F.; Piscopia, R.; Grimaldi, S. PTV-Stream: A simplified particle tracking velocimetry framework for stream surface flow monitoring. *Catena* **2019**, *172*, 378–386. [CrossRef]
23. Horn, B.K.P.; Schunck, B.G. Determining optical flow. *Artif. Intell.* **1981**, *17*, 185–203. [CrossRef]
24. Zhang, G.; Chanson, H. Application of local opticl flow methods to high-velocity free-surface flows: Validation and application to stepped chutes. *Exp. Therm. Fluid Sci.* **2018**, *90*, 186–199. [CrossRef]
25. Tauro, F.; Tosi, F.; Mattoccia, S.; Toth, E.; Piscopia, R.; Grimaldi, S. Optical Tracking Velocimetry (OTV): Leveraging Optical Flow and Trajectory-Based Filtering for Surface Streamflow Observations. *Remote Sens.* **2018**, *10*, doi:10.3390/rs10122010. [CrossRef]
26. Rosten, E.; Porter, R.B.; Drummond, T. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 105–119. [CrossRef] [PubMed]
27. Lucas, B.D.; Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, 24–28 August 1981; pp. 674–679.
28. Bradski, G. The OpenCV Library. *Dr. Dobb's J.* **2000**, 25, 120–126
29. Mitra, G.; Johnston, B.; Rendell, A.P.; McCreath, E.; Zhou, J. Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms. In Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, Cambridge, MA, USA, 20–24 May 2013; IEEE Computer Society:Cambridge, MA, USA, 2013; IPDPSW '13, pp. 1107–1116. doi:10.1109/IPDPSW.2013.207. [CrossRef]

30. Dagum, L.; Menon, R. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* **1998**, *5*, 46–55. doi:10.1109/99.660313. [CrossRef]
31. Pheatt, C. Intel® Threading Building Blocks. *J. Comput. Sci. Coll.* **2008**, *23*, 298.