Article

Towards Open Access and Open Source Software for Image-Based Velocimetry Techniques

Hessel Winsemius [*] , Frank Ohene Annor , Rick Hagenaars , Nick Van de Giesen , Willem Luxemburg , Olivier Hoes

Posted Date: 11 August 2023

doi: 10.20944/preprints202308.0896.v1

Keywords: river monitoring; image-based; discharge estimation; open-source software

Article

Towards Open Access and Open Source Software for Image-Based Velocimetry Techniques

Hessel Winsemius [1,2,3,*], Frank Annor [1,4], Rick Hagenaars [1,4], Nick Van de Giesen [1,4], Willem Luxemburg [1] and Olivier Hoes [1]

[1] Delft, University of Technology, Water Resources Management, Delft, The Netherlands; h.c.winsemius@tudelft.nl
[2] Rainbow Sensing, Den Haag, The Netherlands
[3] Deltares, Delft, The Netherlands
[4] Trans-African Hydro-Meteorological Observatory, Nairobi, Kenya
* Correspondence: h.c.winsemius@tudelft.nl

Abstract: River monitoring has the potential to grow substantially with present-day available affordable and locally sourced hardware. Yet river monitoring networks are still under pressure due to lack of resources, difficulties with maintenance, and rapidly changing conditions which in part may be due to climate change. We advocate that to turn around this trend, monitoring stations must rely on local people, work with locally available devices, work without contact with water and operate through openly available knowledge. River observations with camera videos have this potential. IP cameras, drones or smartphones are widely available as observation platforms. The scientific methods are well established in literature. Yet current attempts to establish scalable open-source software solutions that can be operated anywhere are lacking. In fact, currently available software is either research-oriented, more aimed at incidental observations, is restricted to single-use licenses, entirely proprietary, or restricted to operations through a third-party Software as a Service. To overcome this obstacle, we present OpenRiverCam, a free and open-source software ecosystem for river observations that can fulfill a wide variety of use cases and business cases through a well-documented and simple to use Application Programming Interface, service workflows, cloud scalability and interoperability, and options to extend to several applications within the hydrological, hydrodynamic, environmental and geospatial domains. We demonstrate its current technical abilities through three different case studies that all originate from a user perspective. We discuss the future developments to meet further requirements, which include documentation, widely available training materials and embedding in curricula, and further hardware and software developments.

Keywords: river monitoring; image-based; discharge estimation; open-source software

1. Introduction

The United Nations Office for Disaster Risk Reduction (UNDRR) Guidelines for the reduction of flood losses states that "The operation of a flood warning and response system is the most effective method for reducing the risk of loss of life and economic losses". Scientific evidence also demonstrates the cost effectiveness of flood warnings [1]. In flow forecasting for flood early warning, real-time river observations play a key role, as these provide significant skill for short-term flood forecasting. Furthermore, river flow observations are a key requirement to understand, plan and manage water resources, design infrastructure cost-effectively and up to standards, and study changes in the world's water balance, and establish and calibrate scenario models (see e.g. Hirpa et al., 2018).

Yet hydrological monitoring station networks are declining [2]. In particular in the Global South, the Global Runoff Data Centre, the WMO body for distributing global hydrometric data, is observing a strong decline in the number of stations [3]. Continuous engagement by the authors with entities that are responsible for managing water resources have confirmed a wide variety of reasons why monitoring remains difficult for them. These include costs, risks of vandalism, lack of human resources with specific technical skills, and lack of spare parts. Moreover, a possible clear lack of connection to value adding use cases is one of the often-overlooked reasons for the declining networks. Finally, accessibility of hardware and software is limited, as monitoring equipment is highly specialized while software is often of a proprietary nature.

Non-contact observation methods that rely on relatively simple and widely available camera systems have enormous potential to close this gap. In essence, the principle is to track movements on the water surface, translate these into surface velocities, integrate these over a measured depth cross-section to obtain a river flow estimate. The first part was coined by Fujita [4] as Large-Scale Particle Image Velocimetry (LSPIV). It uses cross-correlation principles to estimate velocities. Several other methods were developed after this including Particle Tracking Velocimetry (PTV) [5], Space-Time Image Velocimetry [6] (STIV) and variations on the aforementioned. Ref [7] provides a more complete overview of these methods.

There are several existing software packages built around these techniques that facilitate a more user-oriented processing of image-based river flow estimates. Examples include FUDAA-LSPIV [8], which is a free user-interface for applying LSPIV, including stabilization, orthorectification, and discharge processing; KLT-IV [9], relying on Optical Flow methods, and including methods for image stabilization; DischargeKeeper [10], relying on an adapted form of LSPIV called Surface Structure Image Velocimetry (SSIV) and including (besides features mentioned for the other packages) optical water level estimates; and Hydro-STIV [11] which applies the mentioned STIV method.

These packages are already in use by many agencies but the uptake of image-based flow methods in operational use cases remains limited to date, in particular in the Global South. Here, image-based methods may be highly advantageous to use due to the semi-arid nature of many rivers, and the strong differences between high and low flow associated with this. This makes observation stations subject to difficult circumstances. There may be many reasons why the uptake is limited, including a limited exposure to the methods, limited application experience, calibration and validation with local institutes [7] and the limited availability, applicability of, and trust in software.

The mentioned software packages, although mostly presented with a user interface and usually having documentation to facilitate handling the software, have several challenges in their uptake. First and foremost, all mentioned packages are closed-source and with this, focusing on a limited set of use cases. Hence the software cannot be further developed for locally specific needs (e.g. use of local surveying practices, locally specific devices and applications for collecting and transmitting raw data, and use of local language). For instance, FUDAA-LSPIV is well capable of providing a discharge estimate in a stream with many tracers and on incidental observations, but with less tracers (i.e. requiring longer integration times) and with many observations on the same spot, the software may become too slow in use and processing time for efficient operations, and may require too much disk space to practically use it for operational needs. These were choices made in the design of the software, narrowing the possible use cases.

Another issue may be the chosen business case, tied to a software package. The DischargeKeeper system provides an excellent interface for operational use cases (e.g. operational discharge monitoring for flow and flood forecasting), through an operational dashboard that allows a user to interrogate multiple sites and very rapid processing. The offer for use of DischargeKeeper is a Service Level Agreement with agreed costs per site and time (SEBA, Photrack, personal communication). While such a business model is very much fit for operational agencies with high staff costs and little staff to spare, for cases where staff rates are much lower, such as in the Global South, it would be more advantageous to pursue a model where local people can fulfill many of the tasks of maintenance of sites and even server infrastructure.

Finally these software stacks are provided as is, without any access to its code base, database or other. This means that changes in the software, and ability to do research and development with it is only possible by the core developer. There is also no guarantee that the software remains available in the future. Given the closed data and software model, this may make the use for an operational entity risky as one has to rely on the assumption that the software and its data will remain in development. For researchers, only "as is" computations can be done.

All three issues mentioned advocate for the freedom of an open-source and modular software ecosystem in which different apps, serving different use cases may be developed. Use cases we collected from our experience may include incidental monitoring of floodplain velocities for monitoring effectiveness of groin fields, monitoring of debris and plastic flow, critical to understand the journey of plastic into the ocean, dike breach monitoring, and fixed site operational discharge monitoring, e.g. by water authorities or dam authorities. These use cases may require specific functionalities and user experiences that hence the requirement of modularity is important. The issue of local languages should also not be underestimated.

We wish to stress that the three issues mentioned may not be a problem for all potential users. It is the choice of the developers to follow a closed source model and for commercial cases this is very understandable given that the costs of development somehow need a return on investment. It may however limit the uptake due to use case obstructions, language obstructions, limits of applicability of the business case elsewhere, limits in ability to perform research, and limits in financial resources to acquire the software.

With this paper, we focus on what is needed for the uptake of the OpenRiverCam methods world-wide (including the Global South) with limited software development. We propose a framework for software development for image-based surface velocity and discharge measurements, that collects methods in an open-source environment, and works in the form of building blocks, to allow a user to select a level of entry most fit for local use. The remainder of this paper is organized as follows: In Section 2, we first describe the methods we followed in designing the software ecosystem, the modularity and choices made in licensing. We apply several of the Principles for Digital Development in this approach, to identify requirements for a successful development of software with intended goals in mind. Furthermore, we describe methods by which we scientifically confirm that the software can be used in three different application cases. In Section 3, we describe the results of the software development and the current state of the OpenRiverCam ecosystem. We also describe the results of the scientific validation for the three cases. In Section 4, we discuss required future developments in the ecosystem, based on the current state and recent discussions with potential end users.

2. Materials and Methods

2.1. Design of the software

Over the past decade, several donor organizations led by USAID, have collectively discussed challenges related to digital development. Based on the discussions, several principles for digital development were developed and endorsed by a large group of international organizations, currently referred to as "Digital Principles". Several of these principles are already applicable to the development of the OpenRiverCam ecosystem, several may become applicable once the components are used by a larger number of stakeholders, and several may become of interest once also integration of the software components in hardware will be established. In this paper we limit ourselves to OpenRiverCam as a software ecosystem, with the potential to eventually be uptaken in edge computing hardware. Table 1 defines the five Digital Principles that we are currently working with, and how we apply these principles in the design and development of the OpenRiverCam ecosystem.

Table 1. The digital principles used throughout the design and development and how these are applied.

Principle	Specific implementation in OpenRiverCam design and development
Design with the user	Through interaction on social media, in particular LinkedIn, and discussion with interested stakeholders, virtually and through in-person meetings, we gather user feedback and potential use cases, and make choices for the design that lead to fulfillment of an as wide as possible set of use cases. We encapsulated the interactions into so-

	called "user stories" that led to the design of the OpenRiverCam ecosystem.
Design for scale	OpenRiverCam is designed in an entirely modular fashion. Several building blocks are developed that can be used as stand-alone or in combinations so that uses at many different levels are assured, such as research, stand-alone application or scaled application through compute nodes, edge processing, and dashboard interfaces. In particular the compute nodes and edge processing abilities will make OpenRiverCam scalable.
Build for sustainability	We use widely accepted and well-maintained libraries only and provide software entirely with a copyleft license so that anyone can contribute to the software, or further develop the software.
Use Open Standards, Open Data, Open Source and Open Innovation	We use the Python language, as this is very broadly known and can therefore be used by many users. We use the NetCDF-CF convention and OGC standards as open standard for data, allowing for easy integration with many other platforms.
Reuse and improve	In order to guarantee that a small group of people can maintain the software indefinitely, we adopt well-maintained data models such as those embedded in the well-known *xarray* library. REST API development is planned in the well-known and maintained *Django* framework.
Be collaborative	We are in the process of defining a forum for user and designer/developer interaction and a code of conduct. We use the GitHub framework with continuous integration for unit testing and releases to assure quality of the code, striving for a minimum of 80% unit test coverage, and accepting issues of users on the platform. We already accept issues from users on Github.

[1] Tables may have a footer.

2.2. Validation of application cases

We selected three different application case studies in which we technically check if the software provides the intended result. This is done by using reference observations as a means to verify the outputs of the software. Table 2 summarizes these case studies and provides an overview of the data used and the methods for validation.

Table 2. Short description case, data + fieldwork used, functionality ORC used, validation means.

Case study	Short description	Data Acquisition	Processing with OpenRiverCam	Validation approach
Alpine River (Perks et al., 2020)	Incidental observation with UAS, with partly artificial partly natural tracers	DJI Phantom FC40 at nadir orientation, with GoPro Hero3+ 4K camera, stabilized, resampled to 12.5Hz, orthorectified to 0.021 m / pixel	Background noise reduction with subtraction of time-averaged frames. PIV, masking of spurious velocities with correlation and outlier filter	Comparison against in-situ propeller observations in m s^{-1}

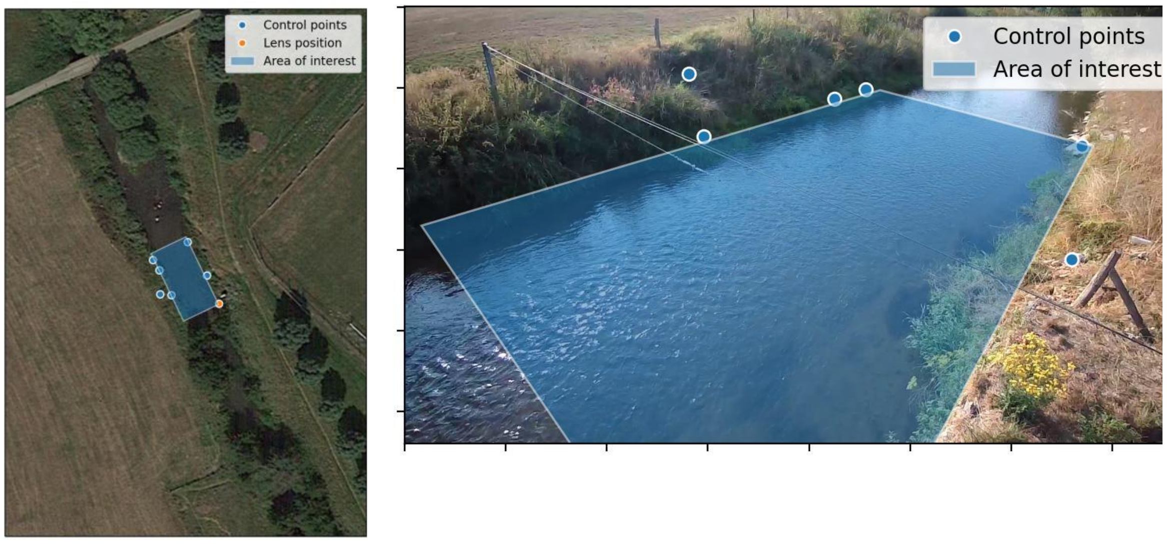
Tidal channel at "De Waterdunen" Zeeland - The Netherlands	Several videos from drone perspectives and a mobile camera on nearby bridge, at different moment in time during incoming tide	DJI Phantom 4, at 4096x2160 resolution	Stabilization, background noise reduction with time differencing and thresholding (minimum intensity >= 5), orthorectification (0.03 m), PIV (15 pixels window size), masking of spurious velocities using correlation, outlier filtering, and minimum valid velocity counting, cross section derivation and integration to discharge	Surface velocities: comparison with ADCP surface velocities; Discharge: comparison with ADCP discharge estimates.
Geul stream at "Hommerich", Limburg - The Netherlands	5 months collected videos with 15-minute frequency	FOSCAM FI9901EP IP camera with power cycling scheme and FTP server	Automated processing. background noise reduction with time differencing and thresholding negative values to zero, orthorectification (0.02 m), PIV (20 pixels window size), masking of spurious velocities using correlation, outlier filtering, and minimum valid velocity counting, cross section derivation and integration to discharge.	Comparison of water level discharge pairs collected through hand measurements by the Waterboard Limburg.

The three case studies, the datasets we used, and analysis performed are more comprehensively described below.

Case Alpine River: Incidental observations of surface velocities in a stream with widely varying conditions: This case is typically needed for monitoring of environmental conditions e.g. for habitat studies, surveys for suitability for small hydropower structures, infrastructural surveys, and monitoring of erosion potential. As a dataset, we used one of the videos of a dataset used for benchmarks of optical methods for velocity and discharge estimation, collected in the Harmony COST action project, reported by [12]. The video chosen is labeled "Alpine River" and its precise location is unknown. The video has a 4K resolution (4320x2160) and is taken by a drone. The video is already stabilized and taken at nadir (90 degree angle with the surface) and therefore does not require any orthorectification. Within the stream and at known pixel locations, velocities were measured in-situ using classical propeller-based velocity measurements. These were used to validate the surface velocities estimated through OpenRiverCam.

Case Tidal Channel: Incidental observations of discharge through mobile or drone platforms: For this, we collected three sets of videos taken by a DJI Phantom 4 Pro drone, flying at different altitudes and with different camera orientation, and one set using a GoPro Hero 10 camera, positioned on a bridge. We recorded videos using the different platforms and positions and orientations at about 30-minute intervals in a tidal channel. At the same time, we recorded cross-sectional velocities and discharge using an ADCP instrument.

Case Geul stream: Continuous observations for real-time monitoring and Q-h rating curve maintenance: for this purpose, we established a fixed camera setup at the Geul at Partij, in close collaboration with the Waterboard of Limburg, The Netherlands. An IP camera of type FOSCAM FI9901EP was installed at approximately 2 meters above the natural levee. A modem of type Teltonika RUT955 was used to arrange power cycling, temporary storage of videos, and transmission of videos to a central FTP server. The setup as well as a map view of the site with measured control points and area of interest is shown in Figure 1. We collected videos of 10 seconds at 15-minute intervals over a 5-month period. The videos were processed with pyOpenRiverCam, using operational water levels monitored by the Waterboard of Limburg on a server using a (at the time of writing) Proof-of-Concept continuously running service for automated processing. The resulting discharge values were used to reconstruct a Q-h rating relationship which was compared against several in-situ measurements, performed and made available by the Waterboard of Limburg over the period January 2022 and April 2023.



(**a**) (**b**)

Figure 1. Situation overview of camera objective at the Geul River. (**a**) overview of site and area of interest. (**b**) area of interest in image frame.

3. Results

In this section, we first describe the results of the software design, followed by the results of the three case studies.

3.1. Design of the software

Based on the discussions with end users (see Section 2), we developed user stories that led to the design of the ecosystem. The most important user stories defined at the time of writing are listed in Table 3. The user stories are kept anonymous.

Table 3. User stories for OpenRiverCam.

As a…	I want…	So that…	Core requirements
remotely operating user	to process videos immediately in the field	I know for sure I have the right results before going back from the field	Processing from a thin client
drone surveyor	to make videos of streams in the field		Dashboard, operated from a smartphone

drone surveyor	and send them through smartphone to not have to cross a river to receive accurate velocity results	I can safely collect data from the banks, with only a drone as material	Get a result with only drone-based data collection, ideally no control points
hydrologist	to combine velocities with bathymetry sections	I can measure river flows	Integrate velocity estimates with cross-section surveys (x,y,z and s,z)
hydrologist	to combine several videos of velocity and discharge of single sites	I can reconstruct rating curves	single site processing with multiple videos
hydrologist	to combine videos from a smart phone that are more or less from the same position, without redoing control points all the time	I can quickly make videos during an event without a fixed camera	image co-registration and corrections on known points, or let user select these
environmental expert/hydrologist/other	to show velocities in a GIS environment	I can combine my data with other data such as land use changes, bathymetric charts, and so on	exports to known GIS raster or mesh formats
environmental expert	to show habitat suitability of certain species based on velocity results	I can use this for monitoring habitats	allow for a visualization of classes, and raster export of habitat suitability (API + dashboard)
engineer	to see how structures in the water influence the velocity patterns	I can show if structures do what they need to do, or show locations prone to erosion	combined visualization of velocity and CAD drawings of structures
engineer	to understand over large multi-km stretches where flow velocities are suitable for hydrokinetics	I can provide guidance where a local hydrokinetic system can be installed	multi-video processing with enough geospatial accuracy to enable seamless combination of results
engineer	to see differences between pre and post construction velocities for infrastructure or river restoration projects	I can provide monitoring as a product to my clients (engineering firms, environmental agencies, etc.)	comparability between videos, accurate co-registration, GIS interoperability

The many user stories shown above, and the specific requirements listed in the left column, led to the general decision to make OpenRiverCam highly modular so that different applications can be built at different levels, serving different users. For instance, the "researcher" may want to directly access a library API, while an environmental engineer, having several user stories, wants to simply drag and drop videos into a dashboard, visualize results to clients, and not worry about maintenance of software at all. The library API may then sit in the application for that user but is never directly approached by that user. Based on the user stories, we have developed a roadmap. Figure 2 shows this roadmap, consisting of the currently available, currently in development, and future planned components of OpenRiverCam as a software ecosystem. The roadmap was inspired in part by the highly successful photogrammetry software ecosystem OpenDroneMap [13], which has a similar modular structure, and currently has thousands of users.

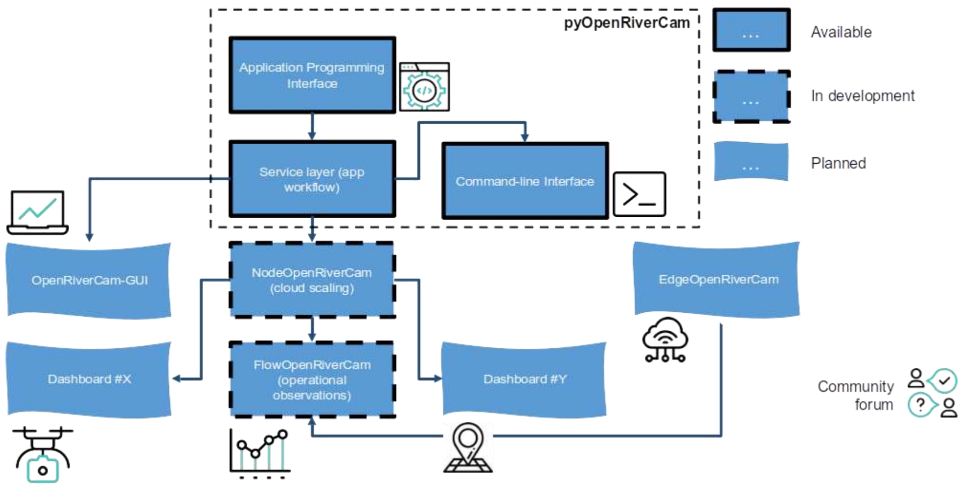


Figure 2. Roadmap of the OpenRiverCam ecosystem, with its current status.

3.1.1. Subsubsection

We made decisions for the modality of sharing, and modular setup of the ecosystem following the Digital Principles as outlined in Section 2. For collaboration purpose, and to allow any user or developer to make their own changes or developments on top of existing code, three decisions were made:

- Have the entire framework available in Github, with issue management.
- Establish a user forum with a code of conduct (not yet established at the time of writing, but part of the roadmap).
- Choose a license that guarantees on the one hand that anyone can further develop the software, and build applications, but also ensures that those developments then become available to other users of the code. We have chosen the Affero General Public License version 3 for this purpose.

We list the key components of the roadmap below:

pyOpenRivercam: this component provides a library with API (in Python) with all the methods available in openly coded classes. The most prominent classes are:

- *CameraConfig*: defines the camera extrinsic and intrinsic characterization, using several methods that make it very easy and intuitive for the user to provide the right information. For instance an area of interest can be defined in the camera objective, by clicking 4 points. A geographically perfectly rectangular bounding box will be fitted through them.
- *Video*: defines the video file to read, region to use for stabilization (if needed), the start and end frame and the *CameraConfig* object required to perform orthorectification on frames.
- *Frames*: derived from the *Video* object. A user can perform several preprocessing filters and thresholding on the frames and orthorectify the frames to the chosen resolution and area of interest.
- *Velocimetry*: derived from the *Frames* object, after performing orthorectification, provides raw velocities, and several masking functions to remove spurious velocities. The PIV methods are using the *OpenPIV* library as a basis.
- *Transect*: derived from *Velocimetry* object by combination with a depth cross-section. Provides velocities over a cross-section by sampling from the 2D surface velocities using (optionally) x and

y-directional window sizes, infilling of missing values with several methods, and interpretation of depth averaged velocities and river flow through integration.

- *Plot*: holds several plotting functions on top of the *Frames*, *Velocimetry* and *Transect* classes that can be combined to create plots with a raw or orthoprojected frame, 2D velocity information, and transect information in one graph.

Service layer (part of pyOpenRiverCam): On top of the API, a service layer is established that collects a number of typical methods into a workflow that is then executed. Such workflows typically can be called from applications. Currently we have two workflows in the service layer that are both accessible from command-line:

- the camera configuration workflow (see Figure 3): this workflow creates a configuration file that defines the camera lens characteristics, perspective, region of interest, water level datums, and stabilization parameters (for e. g. drone videos). Within the command-line interface, the workflow can collect parts of the required information interactively from the user through several interactive displays of frames.
- the velocimetry workflow (see Figure 4): this workflow processes a video, with a camera configuration, and a "recipe" into end products such as velocity fields, cross section velocities, discharge, and plots. This is done through several steps, being frame stabilization (only if needed), preprocessing, orthorectification, velocimetry estimation and estimation of flow through measured cross sections. The "recipe" contains different steps, configured in a yaml formatted text file. The entire processing chain contains frame extraction, stabilization, preprocessing, orthorectification, PIV analysis, post-processing, transect extraction, discharge estimation and plotting all of which can be configured by the user through well-documented methods. The recipe is entirely serializable so that in future developments of platforms around pyOpenRiverCam, such as user interfaces, thin client server-side operated web dashboards or edge applications, can easily be configured through a serialized recipe sent from a user interface to the underlying API, wherever this API runs. The recipe can be re-used for many videos of the same objective, as long as the characteristics of the video and the orientation of the camera remain the same.

pyOpenRiverCam's command-line interface allows the user to use these two workflows through the command-line, but they are naturally also available to other future applications such as a graphical user interface or other interfaces. The number of workflows within the service layer naturally can also be extended. For instance, if a future development leads to methods that lead to abilities to estimate a water level with optical methods, then this can easily be added as a workflow and disclosed to the user via the existing command-line interface or other future interfacing.

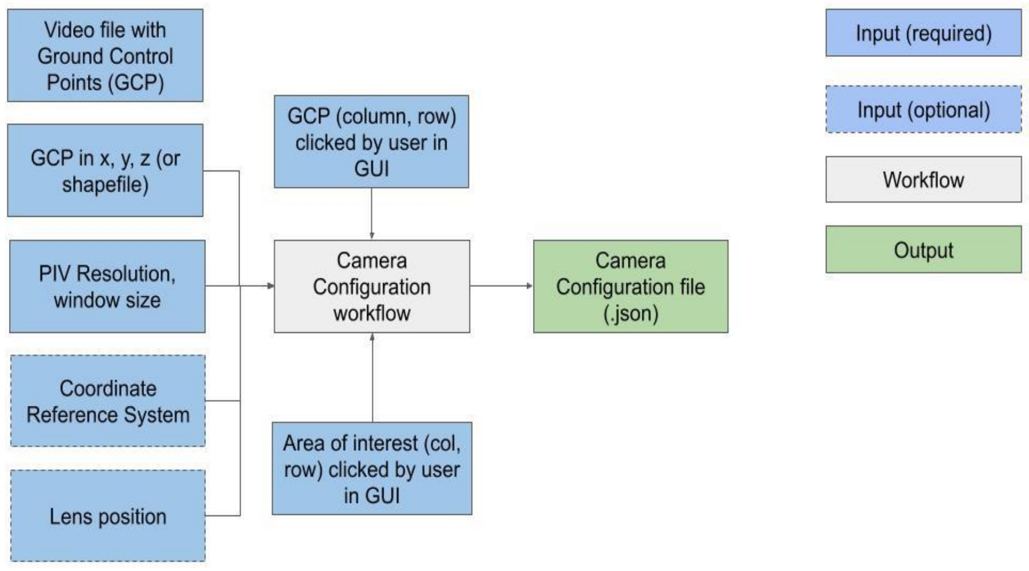

Video file with Ground Control Points (GCP)
GCP in x, y, z (or shapefile)
PIV Resolution, window size
Coordinate Reference System
Lens position
GCP (column, row) clicked by user in GUI
Camera Configuration workflow
Area of interest (col, row) clicked by user in GUI
Camera Configuration file (.json)
Input (required)
Input (optional)
Workflow
Output

Figure 3. Camera configuration workflow.

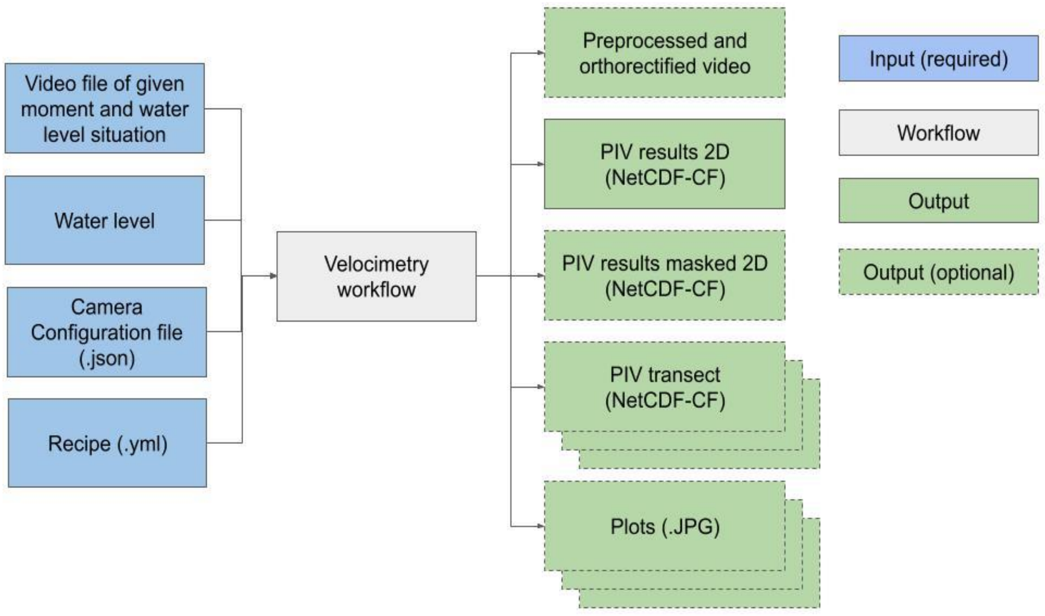

Video file of given moment and water level situation
Water level
Camera Configuration file (.json)
Recipe (.yml)
Velocimetry workflow
Preprocessed and orthorectified video
PIV results 2D (NetCDF-CF)
PIV results masked 2D (NetCDF-CF)
PIV transect (NetCDF-CF)
Plots (.JPG)
Input (required)
Workflow
Output
Output (optional)

Figure 4. Velocimetry workflow.

nodeOpenRiverCam and edgeOpenRiverCam: The Digital Principles also emphasize the importance of scalability. For this purpose, we will establish nodeOpenRiverCam (in development during writing) and edgeOpenRiverCam (planned). nodeOpenRiverCam can be deployed on a cloud, e. g. as a Kubernetes cluster. Through a queueing manager, a node deployed with nodeOpenRiverCam, can pick up jobs for processing videos, write files to a chosen cloud-based file system, and send callbacks to a platform on an instructed url. This will allow nodeOpenRiverCam to be connected to any web API or dashboard that may be established for future applications. edgeOpenRiverCam will only send callbacks to urls whilst processing data in-situ instead of in the cloud. This will be needed in cases where connections to mobile networks and transmission of large amounts of data, such as videos, is not feasible.

Dashboards: The different possible dashboards that may be developed on top of nodeOpenRiverCam and edgeOpenRiverCam are not yet defined and there could potentially be many. The modularity, however, provides freedom to developers with a use case and/or business in mind, to develop their own application. A dashboard that our team will work on, is already more explicitly defined in the roadmap, and has a working name “flowOpenRiverCam”. This dashboard will be made for operational observations. Within several projects of our team, we will further define what flowOpenRiverCam should look like, so that it works within the context of that user, following the “design with the user” principle mentioned in Section 2. For instance, first discussions with end users in Ghana revealed that data collection through a smartphone may be a highly sustainable and manageable approach. To accommodate that, we are considering connecting user-configured sites to an OpenDataKit server that collects survey forms from a surveyor that uses a smartphone on a rig that ensures the objective remains stable. This will allow for data collection in a way that works in the local context.

User forum: to underwrite the Digital Principle “be collaborative”, we see it as essential to connect users. As the team of writers of this article is too small to handle all queries, users will need to inform each other. We are still in the definition process of this user forum. The library pyOpenRiverCam is already at a far developed stage. For its full documentation, we refer to the documentation pages of OpenRiverCam [13].

3.2. Validation of the OpenRiverCam software

The validation cases have been undertaken with the command-line interface of pyOpenRiverCam for processing all videos. The exact manner of processing depended on the case and is described further below.

3.2.1. Case 1: Alpine river

The first case only refers to one video, which is already orthorectified and stabilized. Hence the only part of OpenRiverCam that is validated is its ability to reproduce velocities using the underlying OpenPIV library. No further rescaling was performed and so the original resolution was kept at 0.02 m per pixel. We used two preprocessing filters that in turn apply the difference between subsequent frames, and threshold the differenced intensities to a minimum value of 5. This is meant to reduce background noise. We used a default interrogation window size of 20 pixels (i.e. 20 x 0.02 = 0.4 meter) to perform cross-correlation of patterns, assuming that in this window size enough patterns should be visible to trace over time. We show the resulting velocity validation in Figure 5. The spatial view shows the first frame of the video with the color representing the difference in velocity as measured in-situ and as measured using OpenRiverCam (negative means that OpenRiverCam estimates a lower velocity than in-situ). The scatter plot shows all measurements together. The plots demonstrate that the combination of preprocessing and OpenPIV provides estimates close to the in-situ observations. Some quite high velocities are slightly underestimated. This may be due to the fact that the velocities were measured at the location in the stream where velocity was the highest. As OpenPIV requires an interrogation window, this can lead to a slightly lower velocity as this velocity is representative for a larger interrogation window size than the in-situ point measurement.:

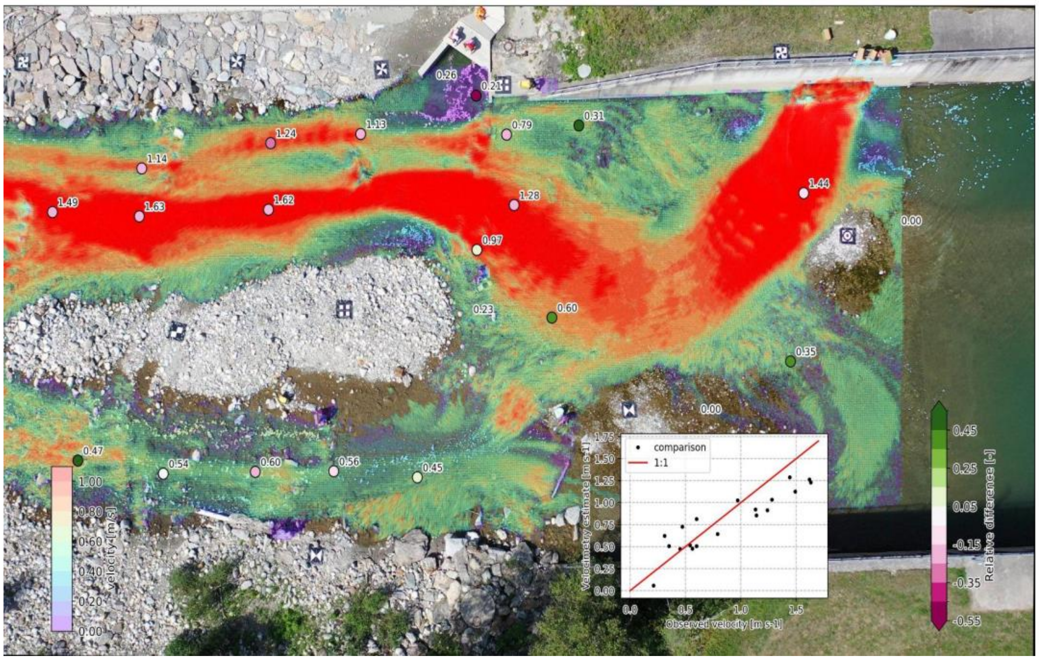


Figure 5. Validation with in-situ point measurements. The numbers with each point display the in-situ measured velocity in m/s. The relative difference is computed as r = (v_{piv} - v_{insitu}) / v_{insitu} where r is the relative difference, v_{piv} is the velocity [m/s] by pyOpenRiverCam, and v_{insitu} is the insitu measured velocity [m/s].

3.2.2. Case 2: tidal channel at Waterdunen

In the second case, many videos were taken by drone on 5 April 2023 of a tidal channel during different moments of incoming tide during the day. In addition videos were taken from a nearby bridge using a GoPro camera. Because mobile platforms were used, and the processing started from the raw videos, more processing steps were required. In brief, we established a camera configuration (see Section 3.1) and selected a stabilization region for each individual video. This is necessary because mobile platforms never have exactly the same objective, and therefore the location of control points in the objective as a result of a different perspective will always be slightly different. The camera configuration was performed using 8 Ground Control Points, selected in the video's objective interactively through the graphical interfacing presented to the user in the command-line process (also described in Section 3.1, example shown in Figure 6), the lens calibration and orthorectification is performed within pyOpenRiverCam automatically in a combined fashion by optimizing the error in perspective transform of all control points, through optimization of the focal length and (barrel) distortion parameters. The "recipe" (see Section 3.1) was kept entirely constant for all videos and used to process each video on command-line in a batch process. In this use case, each video required a separate camera configuration, because the drone videos never have exactly the same objective, but the recipe was kept constant for each video.

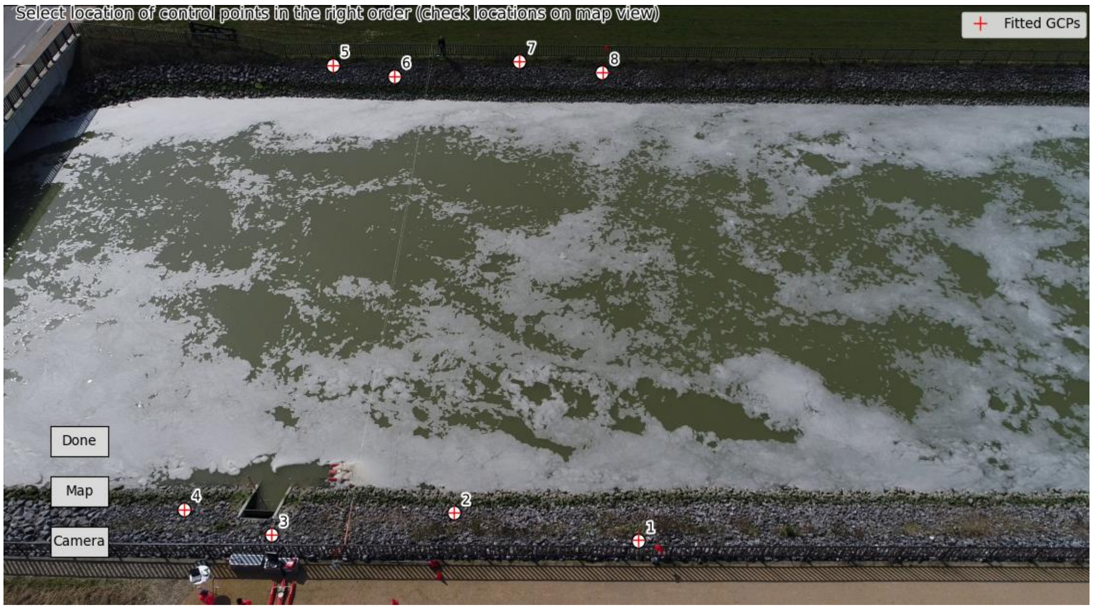

Select location of control points in the right order (check locations on map view)
Fitted GCPs
5
6
7
8
Done
Map
Camera
4
3
2
1

Figure 6. Interface for selecting ground control points.

To understand the performance of the API's methods, we compared surface velocities measured by the ADCP device against the surface velocities measured as a result of the processing chain of pyOpenRiverCam. To account for differences between the first layer of measurement of the ADCP (0.3 meters below the surface) and the velocities measured by pyOpenRiverCam, we transformed the velocities of pyOpenRiverCam to 0.3 meter depth, using the assumption that velocities follow a logarithmic profile with depth, as described in Appendix A. In Figure 7, we show a 2x2 grid of visualizations of the velocities measured at the surface for one time slot, with the different altitudes and camera orientation for the drone in the first 3 subplots, and the GoPro bridge setup in the last subplot. The time slot chosen was immediately after the turn of the tide. At this moment, the available patterns for tracing advective transport were still very limited. The plots show that pyOpenRiverCam is remarkably well capable of monitoring the velocities where tracers are sufficiently visible, but also that it excludes areas where clearly not enough tracers are visible to establish a robust velocity estimate. The GoPro camera imagery was of poor quality due to smoothing, which causes velocities far away in the objective to be noisy and underestimated. This is likely the cause of the poorer correlation between velocities measured by ADCP and by pyOpenRiverCam.

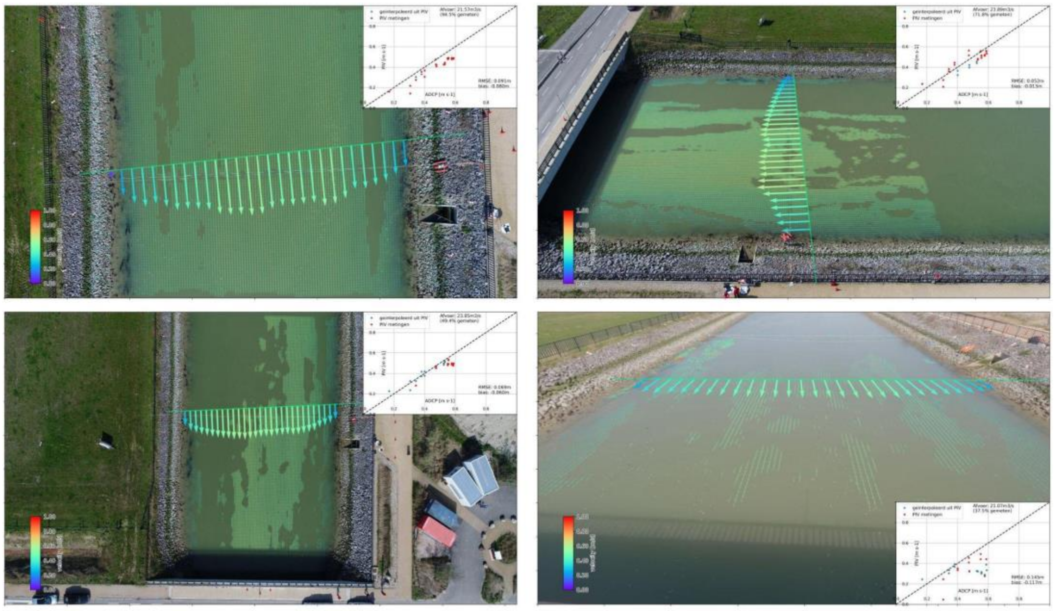


Figure 7. Velocity estimates for 5 April 2023, 12.55 PM local time. Each subplot shows the 2D velocity fields in small quivers, the transect sampled velocities perpendicular to the cross-section in larger quivers, and a scatter plot with cross section velocities compared against ADCP velocities. Top-left: drone at 30m altitude nadir; Top-right: drone at 25m altitude under 45 degrees from the side; Bottom-left: drone at 70m altitude nadir; Bottom-right: GoPro camera mounted on bridge.

Figure 8 shows another 2x2 panel, with a time slot close to the moment with the largest tidal difference. Here the surface velocities are much higher, and many more tracers could be observed. The resulting velocimetry estimates are very strongly correlated to the ADCP velocities, and many more velocity estimates are resolved throughout the area of interest.

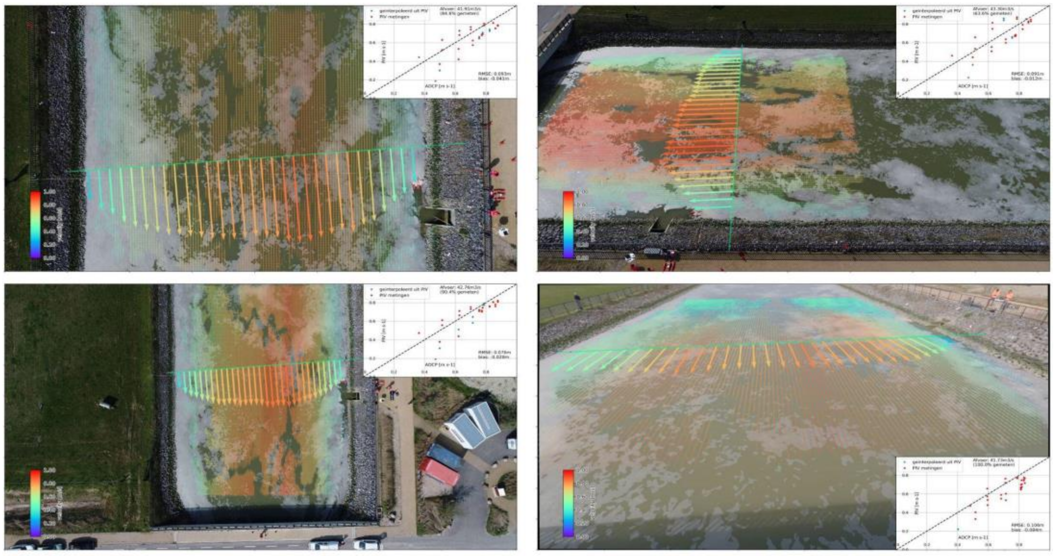


Figure 7. Same as Figure 6, but for 5 April 2023, 3PM local time.

Finally in Figure 9 we show correlation scatter plots of the integrated discharge, using a simple average depth velocity to surface velocity ratio of 0.85. The transparency of each dot represents the fraction of the discharge estimate that was resolved by optical visible velocities (where 1 minus this fraction represents the part of the discharge resolved by interpolation using the principle described in Appendix A to fill in missing values). The scatter plots reveal a remarkable resemblance in discharge estimates over the entire range of observations made during the day.

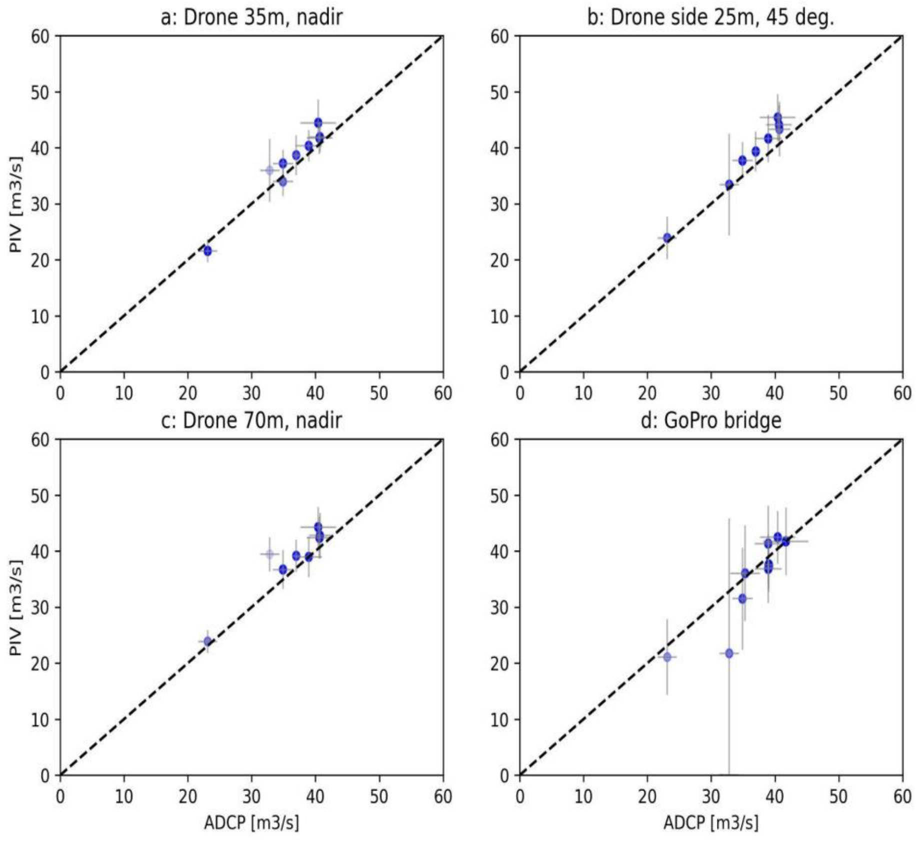
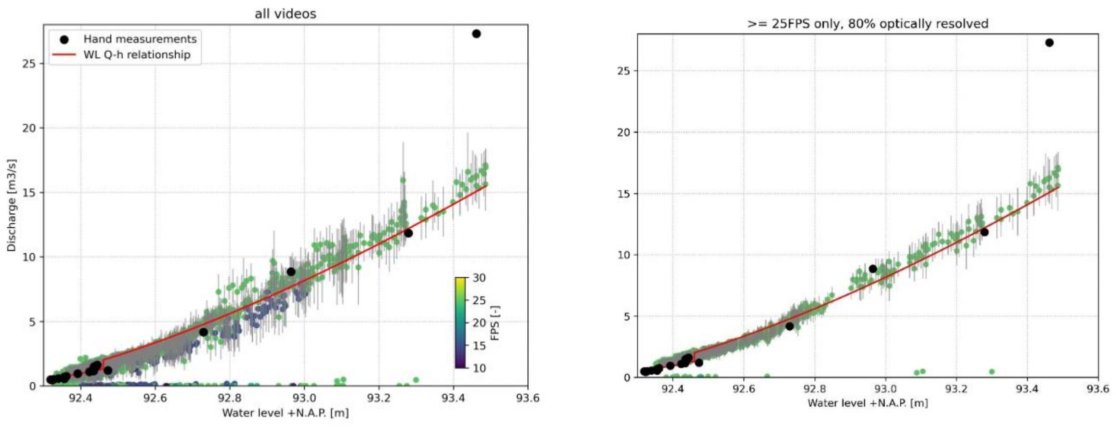


Figure 8. Correlation of discharge estimates with pyOpenRiverCam and ADCP for different perspectives. ADCP on x-axis, PIV median of OpenRiverCam on y-axis. The level of transparency indicates the percentage of the discharge that was resolved from PIV. For instance 70% would be plotted with a 0.7 level transparency and means that 70% of the flow in m3/s was estimated from PIV and the remaining 30% comes from filled in missing values. Error bars are highly conservatively estimated from the variability in velocity on frame-by-frame basis. (a) Drone videos at 35 meter nadir. (b) drone videos at 25 meters from the side under 45 degree angle. (c) Drone videos at 70 meters at nadir. (d) GoPro videos from the bridge deck.

3.2.3. Case 3: Geul River at Hommerich

Establishing a “Q-h” (Discharge – Water level) rating relationship requires continuous observations of the same cross-section over a significantly long period of time. Hence, we monitored over an approximately 5-month period and analyzed all videos using a fixed camera configuration, based on 6 control points measured with Real Time Kinematics (RTK) Global Navigation Satellite Systems (GNSS). A similar recipe as in case 2 was used, but here a resolution of 0.02 (instead of 0.03 m) was used to accommodate the smaller stream width and possibly smaller surface tracers. Figure 9 displays 5 months of (daytime) data, plotted into a relationship between water level (x-axis) and discharge (y-axis). The results show that sometimes, the FOSCAM camera gives a lower framerate than the settings of the camera suggest. Furthermore, nighttime videos (where IR video is used with an IR light) performed poorly, to a large extent because the IR light was pointed at the overhanging cable used for hand measurements, and partly because the IR light can only illuminate a part of the cross-section, requiring a lot of infilling.



(**a**) (**b**)

Figure 9. scatter plots of water level (x-axis) and discharge (y-axis), each point representing the analysis of one video. **(a)** all videos (including nighttime). **(b)** only videos that have at least 25 frames per second and resolve at least 80% of the flow through optical velocimetry.

Therefore, the videos were filtered to only videos that reported a framerate of at least 25 Frames Per Second (FPS) and with at least 80% of the cross-section resolved by optical velocimetry values. Some videos give zero flow due to a limitation in the infilling method, causing flow to be classified as 100% observed, while only a few cross-section points had optically resolved velocities. This issue has been solved in a recent release of pyOpenRiverCam (v0.5.0).

The filtered results clearly show a reliable rating curve without any additional calibration or hand measurements. The resemblance of the video analyses with hand measurements is remarkable save for one point measured on 9 April 2023. On this date a relatively high flow was measured by hand, but the extreme offset compared to the video analysis could not be explained by, for instance, a jump in the channel's geometry or other obvious reasons. Hence it is likely that the hand measurement contains errors. The variability bars on the filtered videos are also a lot smaller than the bars on the non-filtered results. After further inspection of the videos with little optical information, it was concluded that this was partly caused by obscurement of the lens due to droplets and partly because of poor light conditions. With poor light conditions, combined with the low maximum bitrate of the camera (4Mbps) very strong compression occurs on the water surface, removing any small details on the water that may have been picked up by the PIV cross-correlation. This is a known limitation of low-cost IP cameras, and hence it is recommended to use a slightly more expensive IP camera with options for higher bitrates. Nonetheless the case study results demonstrate that the pyOpenRiverCam methods provide good results under varying flow circumstances and that poor results can easily be filtered out.

4. Discussion

4.1. Software development

In this paper we demonstrated how, by listening to end users, and by acknowledging requirements from the Digital Principles, software can be designed in such a way that it fits use cases, user context, and available user resources. Based on the last use case, we consider our software to be at Technological Readiness Level (TRL) 7 as the software has been operationally tested in a real-world environment with an end user. However, despite the fact that important developments have been made, only a small number of the components of our roadmap is currently in place. In order to reach a strong user base, and guarantee long term sustainability of the software, we see the following as essential to become successful:

- Start implementing use cases as soon as possible, even during low TRL phases. This enables short feedback loops from users within the use cases and ensures we develop what truly fits to their needs.

- Develop training materials, ideally multi-lingual. Currently we ensure that documentation remains up to date as a first step, but training materials will be essential to get started. This can be in the form of DIY online, videos, instructables or on-site training materials for dedicated training on-site, ideally with a real-world site visit and data collection.
- Develop and demonstrate data collection methods. Even though this paper focused strongly on the design and first applications of our software framework, its use will strongly depend on the ability of people to perform local data collection. Providing versatility in how this is done is of great importance in order to facilitate as many users as possible. E.g. an operational IP-camera with modem, power and internet facilities may in many cases prove to be complicated to maintain, and subject to vandalism.
- Develop a community of practice. Our GitHub issues pages are a good starting point, but a community forum would create much more interaction.
- Stay on top of the latest science and continuously improve and implement new methods. This is essential to stay relevant in this field. As new methods and approaches are developed, we will seek to implement these to ensure the latest science is available. This may include new methods to trace velocities e.g. [14], scientific developments in data assimilation and combination with hydraulic models [15,16], inclusion of satellite proxies or videos, and machine learning for data infilling, segmentation (e.g. for habitat studies) and more. Authors should discuss the results and how they can be interpreted from the perspective of previous studies and of the working hypotheses. The findings and their implications should be discussed in the broadest context possible. Future research directions may also be highlighted.

4.2. Application development

To ensure that (besides the currently available command-line interface) more user-oriented applications become available, it is important to start focusing on scalable web applications. A prerequisite for this is the development of nodeOpenRiverCam, which will lead to abilities to link one or more dashboard environments to the methods. This will greatly enhance abilities to run many videos, possibly from many different users, within one platform and will offer local entrepreneurs the opportunity to develop a dedicated environment on top of our software, for dedicated use cases and business cases. For operational cases in remote areas, edgeOpenRiverCam will become very important and should communicate in the same manner with platforms so that edgeOpenRiverCam and nodeOpenRiverCam can work together on one dashboard platform if needed.

4.3. Potential outcomes

Given the free and open-source model chosen, the technology becomes favorable for use in less resource-rich environments. For instance, data collection with a smartphone and sending the data off through an OpenDataKit form, is a very low-key yet powerful solution requiring relatively little investment on the side of the hosting agency. Such a solution would also have almost no problems with theft or other vandalism. Another possible setup could be a fixed setup using openly available hardware such as raspberry pi running edgeOpenRiverCam. For use cases, more driven by incidental observations, e.g. during unforeseen floods, or in complex areas with drones, a more dedicated dashboard may be built. Finally, such flexible and simple systems will not stand in the way of setting up more technologically advanced hardware on-site where needed, with the added benefit that, potentially, the software platform can run locally, creating local entrepreneurship within the local economy, therefore also making it more locally affordable.

To make this happen and to make this software development endeavor sustainable, we will need a strong increase in the number of users, more interaction with developers, and more feature requests. The foreseen user forum may become a catalyst for this growth of the user base. The OpenRiverCam ecosystem sets the stage for a wide range of potential use cases in environmental monitoring, small-scale hydropower, river flow observations, humanitarian community-based monitoring, and early flood warnings. These use cases will play the central role in choices for further development in the forthcoming decade. We highly welcome contributions from the community.

Author Contributions: Conceptualization, all; methodology, H.W., W.L., N.G, and F.A.; software, H.W. and R.H.; validation, H.W. and O.H.; investigation, all; resources, F.A. and W.L.; writing—original draft preparation, H.W.; writing—review and editing, all; visualization, H.W..; All authors have read and agreed to the published version of the manuscript.

Funding: This work has been developed partly in scope of the TEMBO Africa project. TEMBO Africa has received funding from the EU's Horizon Europe research and innovation programme under Grant Agreement no. 101086209. This result only reflects the authors' view and the EU is not responsible for any use that may be made of the information it contains.

Data Availability Statement: The data used to produce "Case 1" are not part of this study. Data used to produce results of "Case 2" are property of Rijkswaterstaat and can therefore only be requested with Rijkswaterstaat. Data for "Case 3" can be requested at the main author. For inquiries about data of both Case 2 and Case 3, requests should be sent to info@rainbowsensing.com.

Acknowledgments: Case 2: tidal channel at de Waterdunen, has been established with support from Rijkswaterstaat through project funding and in-kind contributions of Rijkswaterstaat. These contributions have also assisted in expanding the pyOpenRiverCam library. We greatly acknowledge these contributions to this work. Case 3: Geul River at Hommerich has been established with support from the Waterboard of Limburg in the form of in-kind contributions and collaboration in field work and feedback. We also greatly appreciate these contributions.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

To fill in missing values of surface velocity, and to translate surface velocities to a given depth (where the ADCP provides its first observation) we assume that velocities have a logarithmic relationship with depth following:

The velocity as a function of depth can be described using the following relationship:

$$u(z) = u^* \ln(z / z_0), \qquad \text{(A1)}$$

where z_0 [m] is an assumed depth above the bottom where the velocity becomes zero, z is the distance above the bottom where the velocity u occurs, and u* [m/s] is a velocity constant that describes the magnitude of the velocity. Based on the type of the bottom, a value for z0 may be assumed. With the measured velocity at a given depth, we can then derive u* and consequently estimate velocities at any depth. Assuming a sandy bottom, the value has been set to 0.05 m.

Using this method, we estimated values for u* in known measured areas and interpolated this to unknown areas in locations over the transect, where PIV did not lead to satisfactory results. We also used the found values for u* to estimate the velocity at the depth, closest to the surface, where the ADCP was providing observations. This was at 0.3 meters below the surface.

References

1. Pappenberger, F.; Cloke, H.L.; Parker, D.J.; Wetterhall, F.; Richardson, D.S.; Thielen, J. The Monetary Benefit of Early Flood Warnings in Europe. *Environ. Sci. Policy* **2015**, *51*, 278–291. https://doi.org/10.1016/j.envsci.2015.04.016.
2. Vorosmarty, C.; Askew, A.; Grabs, W.; Barry, R.G.; Birkett, C.; Doll, P.; Goodison, B.; Hall, A.; Jenne, R.; Kitaev, L.; et al. Global Water Data: A Newly Endangered Species. *Eos Trans. Am. Geophys. Union* **2001**, *82*, 54–54. https://doi.org/10.1029/01EO00031.
3. GRDC Data Portal Available online: https://portal.grdc.bafg.de/ (accessed on 23 June 2023).
4. Fujita, I.; Muste, M.; Kruger, A. Large-Scale Particle Image Velocimetry for Flow Analysis in Hydraulic Engineering Applications. *J. Hydraul. Res.* **1998**, *36*, 397–414. https://doi.org/10.1080/00221689809498626.
5. Tauro, F.; Piscopia, R.; Grimaldi, S. PTV-Stream: A Simplified Particle Tracking Velocimetry Framework for Stream Surface Flow Monitoring. *CATENA* **2019**, *172*, 378–386. https://doi.org/10.1016/j.catena.2018.09.009.

6. 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. https://doi.org/10.1080/15715124.2007.9635310.
7. Tauro, F.; Petroselli, A.; Grimaldi, S. Optical Sensing for Stream Flow Observations: A Review. *J. Agric. Eng.* **2018**, *49*, 199–206. https://doi.org/10.4081/jae.2018.836.
8. Jodeau, M.; Bel, C.; Antoine, G.; Bodart, G.; Coz, J.L.; Faure, J.-B.; Hauet, A.; Leclercq, F.; Haddad, H.; Legout, C.; et al. New Developments of Fudaa-LSPIV, a User-Friendly Software to Perform River Velocity Measurements in Various Flow Conditions. In *River Flow 2020*; CRC Press, 2020 ISBN 978-1-00-311095-8.
9. Perks, M.T. KLT-IV v1.0: Image Velocimetry Software for Use with Fixed and Mobile Platforms. *Geosci. Model Dev.* **2020**, *13*, 6111–6130. https://doi.org/10.5194/gmd-13-6111-2020.
10. Peña-Haro, S.; Carrel, M.; Lüthi, B.; Hansen, I.; Lukes, R. Robust Image-Based Streamflow Measurements for Real-Time Continuous Monitoring. *Front. Water* **2021**, *3*.
11. Hydro-STIV | Numerical Analysis Specialized Company, Hydro Technology Institute. Available online: https://hydrosoken.co.jp/en/service/hydrostiv.php (accessed on 23 June 2023).
12. Perks, M.T.; Dal Sasso, S.F.; Hauet, A.; Jamieson, E.; Le Coz, J.; Pearce, S.; Peña-Haro, S.; Pizarro, A.; Strelnikova, D.; Tauro, F.; et al. Towards Harmonisation of Image Velocimetry Techniques for River Surface Velocity Observations. *Earth Syst. Sci. Data* **2020**, *12*, 1545–1559. https://doi.org/10.5194/essd-12-1545-2020.
13. OpenDroneMap Available online: https://www.opendronemap.org/ (accessed on 26 June 2023).
14. Dolcetti, G.; Hortobágyi, B.; Perks, M.; Tait, S.J.; Dervilis, N. Using Noncontact Measurement of Water Surface Dynamics to Estimate River Discharge. *Water Resour. Res.* **2022**, *58*, e2022WR032829. https://doi.org/10.1029/2022WR032829.
15. Mansanarez, V.; Westerberg, I.K.; Lam, N.; Lyon, S.W. Rapid Stage-Discharge Rating Curve Assessment Using Hydraulic Modeling in an Uncertainty Framework. *Water Resour. Res.* **2019**, *55*, 9765–9787. https://doi.org/10.1029/2018WR024176.
16. Samboko, H.T.; Schurer, S.; Savenije, H.H.G.; Makurira, H.; Banda, K.; Winsemius, H. Evaluating Low-Cost Topographic Surveys for Computations of Conveyance. *Geosci. Instrum. Methods Data Syst.* **2022**, *11*, 1–23. https://doi.org/10.5194/gi-11-1-2022.