Journal of
open research software

oflibnumpy & oflibpytorch: Optical Flow Handling and Manipulation in Python

SOFTWARE METAPAPER

]u[ubiquity press

CLAUDIO S. RAVASIO
LYNDON DA CRUZ
CHRISTOS BERGELES
Author affiliations can be found in the back matter of this article

ABSTRACT

We present oflibnumpy and oflibpytorch, an optical flow library based on NumPy arrays and PyTorch tensors, respectively. It provides a structured approach to the representation of optical flow, i.e 2D vector fields, as a custom class offering a number of methods to apply, manipulate, analyse, and visualise the flow. The library takes into account the two possible frames of reference in optical flow calculation, namely the source (first frame) and target (second frame). The collection of methods and their rigorous mathematical underpinning makes the library broadly applicable to any project that uses flow fields. It is implemented as a Python 3 package whose source can be found on GitHub, and which can be installed either from the git repository or the Python Package Index (PyPI).

CORRESPONDING AUTHOR:
Claudio S. Ravasio
Research Assistant, King's College London / PhD student, University College London, GB
claudio.s.ravasio@gmail.com

KEYWORDS:
Optical flow; Flow field; Flow vector; Flow composition; Python; NumPy; PyTorch

TO CITE THIS ARTICLE:
Ravasio CS, Da Cruz L, Bergeles C 2021 oflibnumpy & oflibpytorch: Optical Flow Handling and Manipulation in Python. *Journal of Open Research Software,* 9: 31. DOI: *https://doi.org/10.5334/jors.380*

(1) OVERVIEW

INTRODUCTION

Optical flow calculation constitutes one of the fundamental tasks of computer vision. It aims to reconstruct the motion between two consecutive images by finding pixel mappings between them, expressed as a 2D vector field. Seminal works by Horn and Schunck [6] and Lucas and Kanade [9] are now over 40 years old, and underpin the progress of the field.

Interest in optical flow algorithms remains high, as an analysis of the publication date of papers evaluated on SINTEL shows [3]. The explosion of the use of deep learning methods for optical flow prediction coupled with new potential applications such as autonomous driving has led to a renewed push for improvements on the state of the art on ever more complex benchmarks, such as SINTEL [2], KITTI [10], and DAVIS [14].

Several methods to estimate optical flow from image sequences are available in Python, ranging from the OpenCV function `cv2.calcOpticalFlowFarneback` [1] based on Farnebäck's algorithm [4] to implementations of methods aiming to better merge different scales, such as the Python wrapper by Pathak et al. [13] for Liu et al. [8]. Virtually all recent deep learning based methods were developed in Python, with many important works, e.g. FlowNet2 [7], PWC-Net [19], or RAFT [20] either making use of the PyTorch machine learning framework [12] from the onset, or making available a Pytorch implementation later on.

Despite the extensive research to date, to the best of our knowledge there exists no off-the-shelf software that allows for easy handling and manipulation of optical flow fields. We examined existing code, and what was retrieved was either algorithm dependent, deals mainly with visualisation (`flowvid` [16], `flow-vis` [17]), or adds only basic flow warping with a focus on very specific tasks such as reading/writing flows to the display capabilities (`flowpy` [18]).

In contrast, we provide a structured approach to flow fields and their manipulation. We take both possible reference frames for flow vectors into account (see *Figure 1*), and offer a wide range of operations on the flow fields themselves. This rigorous method ensures mathematically correct handling, thereby helping users avoid pitfalls such as simply negating flow vectors to obtain the inverse of a flow field. Beyond that, we especially highlight the flow composition functions that have not previously been implemented in this form. As an example, composition functions allow users to calculate the flow field which combined sequentially with another (known) flow is equivalent to a third flow. These operations were derived from first principles, and proved to be paramount for the construction of the optical flow ground truth in the synthetic datasets used in our work presented in Ravasio et al. [15].

THEORY

We can define a flow field as mapping the coordinates **x** of features 1 to i at time t_1 to coordinates at time t_2:

$$\mathcal{F}_{1\to 2} := \mathrm{X}_{t_1} \to \mathrm{X}_{t_2}; \mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_i\} \qquad (1)$$

In the context of this work, the features are image pixels, corresponding to a discretised regular grid in the theoretically continuous image. However, there are two possible frames of reference, illustrated in *Figure 1*:

- **Source, "s":** The pixel features whose motion is tracked by the flow vectors are those in what we term the source domain, or the image at time t_1. Thus, the flow field $\mathcal{F}_{s,1\to 2}$ indicates the motion of

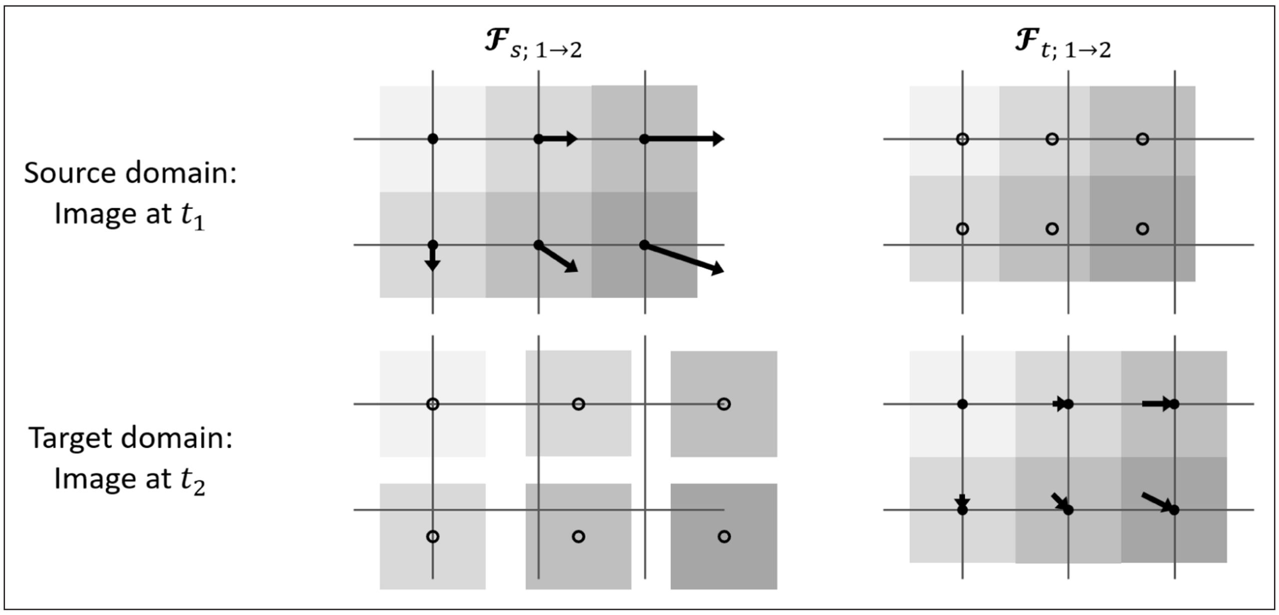
$\mathcal{F}_{s;\,1\to 2}$
$\mathcal{F}_{t;\,1\to 2}$
Source domain: Image at t_1
Target domain: Image at t_2

Figure 1 Flow fields in the two possible frames of reference: "source" means all pixels at time t_1 are mapped to a new location at time t_2, while "target" means all pixels at time t_2 are matched with a different previous location at the time t_1.

every pixel present in the image at time t_1 to their respective end position at time t_2.

- **Target, "t":** The pixel features whose motion is tracked by the flow vectors are those in what we term the target domain, or the image at time t_2. The flow field $\mathcal{F}_{t;1\to 2}$ matches every pixel present in the image at time t_2 to their origin at time t_1.

Therefore, we can extend the previous definition as follows:

$$\begin{aligned}\mathcal{F}_{s;1\to 2} &:= \mathrm{G}_{t_1} \to \mathrm{X}_{t_2}\\ \mathcal{F}_{t;1\to 2} &:= \mathrm{X}_{t_1} \to \mathrm{G}_{t_2}\end{aligned} \tag{2}$$

where **G** = **H** × **W**, a 2*D* space with a regular grid defined by **H** = {0, 1, ..., *H*–2, *H*–1} and **W** = {0, 1, ..., *W*–2, *W*–1}.

Given this, we set up the following equation:

$$\mathcal{F}_{1\to 2} \oplus \mathcal{F}_{2\to 3} = \mathcal{F}_{1\to 3} \tag{3}$$

where $\mathcal{F}_{1\to 2}$ is the flow from time t_1 to time t_2, and ⊕ is the non-commutative operation corresponding to the sequential application of two flow fields. To calculate the actual flow vector values of $\mathcal{F}_{1\to 3}$, the naive approach would be to simply add the vectors:

$$\mathcal{F}_{1\to 2} + \mathcal{F}_{2\to 3} = \begin{cases}(\mathrm{G}_{t_1} \to \mathrm{X}_{t_2}) + (\mathrm{G}_{t_2} \to \mathrm{X}_{t_3}) & \text{if source reference}\\ (\mathrm{X}_{t_1} \to \mathrm{G}_{t_2}) + (\mathrm{X}_{t_2} \to \mathrm{G}_{t_3}) & \text{if target reference}\end{cases} \tag{4}$$

However, a closer inspection reveals this to be incorrect: a different set of features is tracked from time t_1 to t_2 than from time t_2 to t_3 (see also ***Figure 2***). It is therefore necessary to add an intermediate step which refers the features in $\mathbf{G}_{t2}$ back to $\mathbf{G}_{t1}$, or the features in $\mathbf{G}_{t2}$ forward to $\mathbf{G}_{t3}$, for the source and the target reference frame respectively. The correct operation in the source reference frame is then:

$$\begin{aligned}\mathcal{F}_{s;1\to 3} = \mathcal{F}_{s;1\to 2} \oplus \mathcal{F}_{s;2\to 3} &= \mathcal{F}_{s;1\to 2} + \mathcal{F}_{2\to 1}\{\mathcal{F}_{s;2\to 3}\}\\ &= \mathcal{F}_{s;1\to 2} + \mathcal{F}^{-1}_{s;1\to 2}\{\mathcal{F}_{s;2\to 3}\}\end{aligned} \tag{5}$$

where $\mathcal{F}_{s;1\to 2}$ {**G**} means applying the flow $\mathcal{F}_{s;1\to 2}$ to the grid $\mathbf{G}_{t1}$ to obtain the new feature coordinates $\mathbf{X}_{t2}$, followed by an interpolation operation to obtain new grid values $\mathbf{G}_{t2}$. The grid can either contain the pixel values of an image or, as in the previous equation, the vector values of another flow field. The inverse of a flow field can be calculated with the formula $\mathcal{F}^{-1}_{s;1\to 2} = \mathcal{F}_{s;1\to 2}\{-\mathcal{F}_{s;1\to 2}\}$, i.e. reversing the flow vectors and then applying the flow field to the result in order to obtain an interpolation of the result in a new regular grid, as above.

If $\mathcal{F}_{2\to 3}$ in Equation (3) is unknown, given known inputs $\mathcal{F}_{1\to 2}$ and $\mathcal{F}_{1\to 3}$, a similar consideration applies. In this case, the flow vectors can be subtracted directly, but then need to be mapped onto the grid at time t_2:

$$\mathcal{F}_{s;2\to 3} = \mathcal{F}_{s;1\to 2}\{\mathcal{F}_{s;1\to 3} - \mathcal{F}_{s;1\to 2}\} \tag{6}$$

Finally, if $\mathcal{F}_{1\to 2}$ is unknown, we can subtracts known input $\mathcal{F}_{2\to 3}$ from the second known input $\mathcal{F}_{1\to 3}$, after mapping the former from t_2 to t_1 via t_3. The following equation applies:

$$\begin{aligned}\mathcal{F}_{s;1\to 2} &= \mathcal{F}_{s;1\to 3} - \mathcal{F}_{2\to 1}\{\mathcal{F}_{s;2\to 3}\}\\ &= \mathcal{F}_{s;1\to 3} - (\mathcal{F}_{2\to 3} \oplus \mathcal{F}_{3\to 1})\{\mathcal{F}_{s;2\to 3}\}\\ &= \mathcal{F}_{s;1\to 3} - (\mathcal{F}_{s;2\to 3} \oplus \mathcal{F}^{-1}_{s;1\to 3})\{\mathcal{F}_{s;2\to 3}\}\\ &= \mathcal{F}_{s;1\to 3} - (\mathcal{F}_{s;2\to 3} + \mathcal{F}^{-1}_{s;2\to 3}\{\mathcal{F}^{-1}_{s;1\to 3}\})\{\mathcal{F}_{s;2\to 3}\}\end{aligned} \tag{7}$$

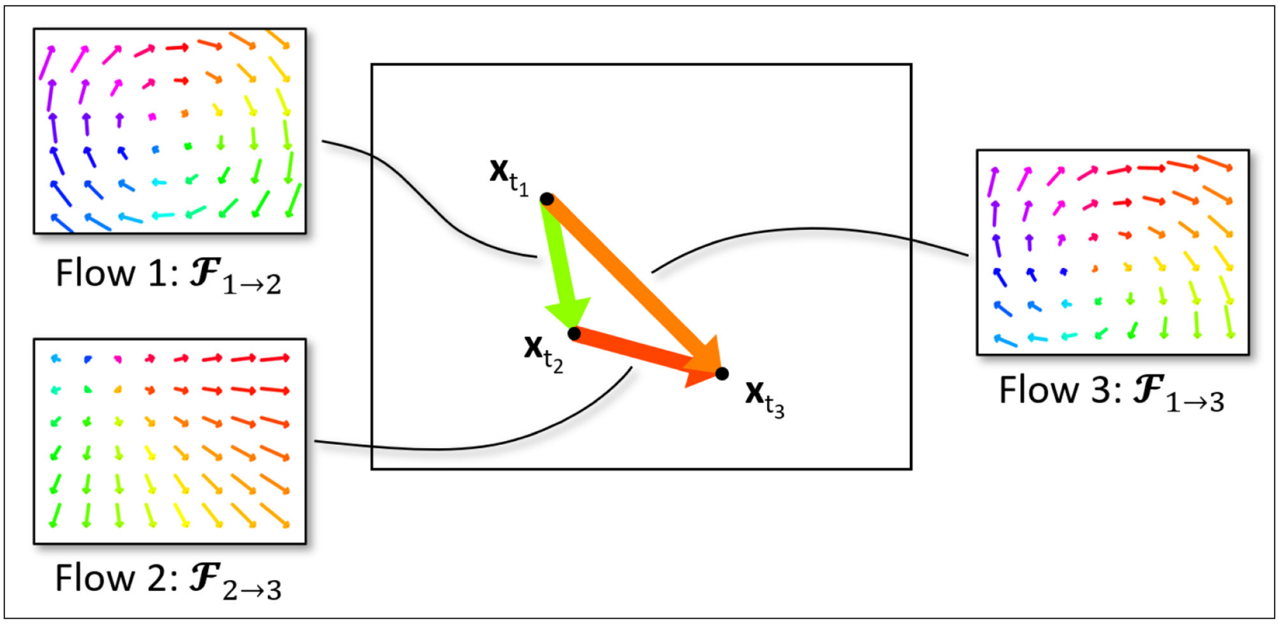
Flow 1: $\mathcal{F}_{1\to 2}$
Flow 2: $\mathcal{F}_{2\to 3}$
Flow 3: $\mathcal{F}_{1\to 3}$
$\mathbf{x}_{t_1}$
$\mathbf{x}_{t_2}$
$\mathbf{x}_{t_3}$

Figure 2 Schematic of the flow field composition operation. To obtain the correct coordinate mapping $\mathbf{x}_{t1} \to \mathbf{x}_{t3}$ from Flows 1 and 2, it is necessary to select the flow vector from $\mathcal{F}_{2\to 3}$ which starts at the end position of the tracked feature at time t_2, or $\mathbf{x}_{t2}$. Therefore, it is not sufficient to simply add the vectors of the two flow fields: this would be equivalent to selecting the flow vector from $\mathcal{F}_{2\to 3}$ which starts at $\mathbf{x}_{t1}$ (assuming the "source" frame of reference is used).

Equations (5) to (7) allow us to achieve all three important modes of composition of flow fields in the "source" reference frame using just two fundamental operations: vector addition, and applying a flow field to an input. An analogous approach yields corresponding formulae for flow fields in the "target" frame of reference.

IMPLEMENTATION AND ARCHITECTURE

Oflibnumpy and oflibpytorch are implemented as a custom flow class based on either NumPy arrays or PyTorch tensors, respectively, with detailed documentation and usage guides available on *oflibnumpy.rtfd.io* and *oflibpytorch.rtfd.io*. Using tensors instead of arrays means oflibpytorch can partially run on GPU instead of being limited to CPU operations, and therefore integrate better into PyTorch-based deep learning algorithms that use optical flow fields. This comes at the cost of having to control the tensor devices of inputs and outputs.

The flow class has three main attributes:

- Vectors `vecs`: The flow vectors themselves. They are expressed as an array of shape (*H, W,* 2) (oflibnumpy) or a tensor of shape (2*, H, W*), following the PyTorch channel-first convention (oflibpytorch). The dimension of size 2 corresponds to the vector components (*x, y*), with x defined positive towards the right, and y defined positive downwards. This follows the OpenCV convention for flow vectors, e.g. as output by the `calcOpticalFlowFarneback` function.
- Reference `ref`: The flow reference determines which frame of reference the flow vectors are in. The options are "source" or "target": either the flow vectors originate from a regular grid in the flow source, or they point to a regular grid in the flow target (see ***Figure 1*** and Equation (2)).
- Mask `mask`: A boolean array or tensor of shape (*H, W*) which indicates which flow vectors are valid. This is not relevant for simple flow operations such as resizing, or tracking points, but becomes very important when several flow fields are combined. In that case, often only parts of the area $H \times W$ of the resulting flow field will contain useful vector values. This is not a limitation of the algorithm, but a characteristic that arises from the nature of the operation.

The implemented class methods can be grouped as follows:

- Constructors: to create flow fields from a given transformation matrix, a list of transforms, or filled with zero-magnitude vectors
- Manipulation: inverting, resizing, and padding flows, or switching their reference frame
- Application: warping an input, or tracking specific points
- Evaluation: finding the valid source or target area, necessary padding of inputs, or fitting a transformation matrix to the flow field
- Visualisation: either using the classic hue-based method, or showing arrows

Finally, as a key component building on the entire rest of the flow library, the method `combine_flows` allows for the composition of two input flow fields into one output flow field. Three modes are available: the output of the function corresponds to either $\mathcal{F}_{1\to2}$, $\mathcal{F}_{2\to3}$ or $\mathcal{F}_{1\to3}$ in Equation (3), the other two being the given input flows. The equations used for this function were derived from first principles as shown in Equations (5) to (7), and rely exclusively on the implemented flow class methods. These also ensure the valid area is tracked correctly through each operation, and returned as the mask of the resulting flow object.

Warping inputs with a flow field in the "source" frame of reference is a slow operation, as it requires interpolation from an unstructured to a regular grid. To optimise performance, we therefore adapt some of the flow composition equations to avoid this operation wherever possible. For example, we make use of a fundamental relationship between the two frames of reference derived from ***Figure 1***: we can invert a flow field and switch its frame of reference at the same time by simply inverting the flow vectors.

$$\begin{aligned}\mathcal{F}^{-1}_{s;1\to2} &:= \mathrm{G}_{t_1} \leftarrow \mathrm{X}_{t_2}\\ &= \mathrm{X}_{t_2} \to \mathrm{G}_{t_1}\\ &= \mathcal{F}_{t;2\to1}\end{aligned} \tag{8}$$

Applied to Equation (5), this allows us to replace two slow operations, namely inverting the flow field $\mathcal{F}_s$ and applying it to an input, with two fast operations: getting the inverse in the "target" frame of reference, and applying it to an input.

$$\begin{aligned}\mathcal{F}_{s;1\to3} &= \mathcal{F}_{s;1\to2} + \mathcal{F}^{-1}_{s;1\to2}\{\mathcal{F}_{s;2\to3}\}\\ &= \mathcal{F}_{s;1\to2} + (\mathcal{F}_{s;1\to2})^{-1;s\to t}\{\mathcal{F}_{s;2\to3}\}\\ &= \mathcal{F}_{s;1\to2} + \mathcal{F}_{t;2\to1}\{\mathcal{F}_{s;2\to3}\}\end{aligned} \tag{9}$$

QUALITY CONTROL

We implemented tests based on the `unittest` package for all relevant functions and class methods. They verify the mathematical validity of the flow composition operations, compare the output of functions such as `Flow.resize` with expected results, and ensure unexpected inputs throw the required error. They are available from the `test` folder on GitHub.

These tests were written and continuously updated during development. The `coverage` package (see *coverage.rtfd.io*) reports an overall test coverage of 99% for both oflibnumpy and oflibpytorch.

Official documentation is available on ReadTheDocs (RTD; see *oflibnumpy.rtfd.io* and *oflibpytorch.rtfd.io*). The introduction page provides simple code examples that use the core functionality, along with the expected output. Users can find this code along with further sample usages in a file called `examples.py` in the source code on GitHub. Comparing the outputs will serve to confirm the installation is working as intended. A more complete demonstration and explanation of the capabilities of oflibnumpy and oflibpytorch is available on the "Usage" page of the RTD documentation, including visualisations of the outputs.

(2) AVAILABILITY

OPERATING SYSTEM

As both oflibnumpy and oflibpytorch are pure Python packages, they are compatible with any operating system that can provide a Python 3 environment. Development took place on a Windows 10 system.

PROGRAMMING LANGUAGE

Oflibnumpy and oflibpytorch require Python *3*, and have been specifically tested for Python *3.7* and *3.9*.

ADDITIONAL SYSTEM REQUIREMENTS

There are no additional system requirements.

DEPENDENCIES

The following packages are required and will be installed as dependencies when using the commands `pip install oflibnumpy` or `pip install oblibpytorch`, or when installing the code from source via `setup.py`:

- NumPy ≥ 1.15 [5]
- SciPy ≥ 1.4 [21]
- OpenCV ≥ 3.4 [1]

Oflibpytorch additionally requires PyTorch ≥ 1.4 [12], and a compatible version of the CUDA toolkit [11] if operations on GPU are required. We recommend an installation in a virtual Conda environment, using the install command suggested on *PyTorch.org*.

LIST OF CONTRIBUTORS

The contributors to this work are Claudio S. Ravasio, Christos Bergeles, and Lyndon Da Cruz.

SOFTWARE LOCATION: OFLIBNUMPY

Archive 1

Name: Zenodo
Persistent identifier: *10.5281/zenodo.4916270*
License: MIT License
Publisher: Claudio S. Ravasio
Version published: 1.0.0
Date published: 09/06/21

Archive 2

Name: Python Package Index (PyPI)
Persistent identifier: *pypi.org/project/oflibnumpy*
License: MIT License
Publisher: Claudio S. Ravasio
Version published: 1.0.0
Date published: 09/06/21

Code repository

Name: GitHub
Persistent identifier: *github.com/RViMLab/oflibnumpy*
License: MIT License
Date published: 09/06/21

SOFTWARE LOCATION: OFLIBPYTORCH

Archive 1

Name: Zenodo
Persistent identifier: *10.5281/zenodo.4916367*
License: MIT License
Publisher: Claudio S. Ravasio
Version published: 1.0.0
Date published: 09/06/21

Archive 2

Name: Python Package Index (PyPI)
Persistent identifier: *pypi.org/project/oflibpytorch*
License: MIT License
Publisher: Claudio S. Ravasio
Version published: 1.0.0
Date published: 09/06/21

Code repository

Name: GitHub
Persistent identifier: *github.com/RViMLab/oflibpytorch*
License: MIT License
Date published: 09/06/21

LANGUAGE

English

(3) REUSE POTENTIAL

An early version of the library described in this paper was used in Ravasio et al. [15]. We especially made use of the ability to find a flow field which, when combined sequentially with a first known flow, results in a known third flow (see Equation (6)). This `combine_flow` function, which in turn relies on the implementation of the flow class and its methods, is key to the creation of the complex synthetic optical flow datasets used in our work and continues to be used in ongoing research on the topic. We therefore see this software as both of great value for specialised work with flow fields as well as broadly applicable to any optical flow task that requires common operations such as warping an input, resizing a flow field, or inverting it. We intend it to be an off-the-shelf tool that is easy to install, easy to use, and well documented for researchers from any field. As an example, Sintel [2] as well as KITTI [10] provide ground truth flow fields along with data on invalid pixels. Once loaded into NumPy, both can be used to construct a single oflib flow object, making use of the `mask` attribute, which will then automatically keep track of any changes to invalid pixels effected by operations carried out on the flow field. More experienced programmers with very specific needs can also modify the source code, or simply make use of the existing structure by extending the flow class as required.

The main support channel for oflibnumpy and oflibpytorch are their respective GitHub issue pages. Users are also welcome to contact the first author of this paper via email to ask for support or report issues.

ACKNOWLEDGEMENTS

We thank Theodoros Pissas and Ross Henry for useful discussions during development.

FUNDING STATEMENT

This work was supported by the National Institute for Health Research NIHR (Invention for Innovation, i4i; II-LB-0716-20002). The views expressed are those of the authors and not necessarily those of the NHS, the NIHR, or the Department of Health and Social Care. The project was also supported by core funding from the Wellcome/EPSRC Centre for Medical Engineering, Wellcome Trust [WT203148/Z/16/Z].

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR CONTRIBUTIONS

Lyndon Da Cruz and Christos Bergeles have contributed equally.

AUTHOR AFFILIATIONS

Claudio S. Ravasio *orcid.org/0000-0002-6453-5376*
Research Assistant, King's College London/PhD student, University College London, GB

Lyndon Da Cruz *orcid.org/0000-0002-7695-6354*
Consultant Ophthalmic Surgeon, Moorfields Eye Hospital, London, GB

Christos Bergeles *orcid.org/0000-0002-9152-3194*
Associate Professor, King's College London, GB

REFERENCES

1. **Bradski G.** The OpenCV Library. *Dr Dobb's Journal of Software Tools*, 2000; 25: 120–125.
2. **Butler DJ, Wulff J, Stanley GB, Black MJ.** A naturalistic open source movie for optical flow evaluation. In: Fitzgibbon A, et al (Eds.), *European Conf. on Computer Vision (ECCV)*, Springer-Verlag, 2012; 611–625. DOI: *https://doi.org/10.1007/978-3-642-33783-3_44*
3. **Butler DJ, Wulff J, Stanley GB, Black MJ.** 2021. URL *http://sintel.is.tue.mpg.de/results*, accessed: 11/06/21.
4. **Farnebäck G.** Two-frame motion estimation based on polynomial expansion. In: *Scandinavian conference on Image analysis*, Springer, 2003; 363–370. DOI: *https://doi.org/10.1007/3-540-45103-X_50*
5. **Harris CR, Millman KJ, van derWalt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ,** et al. Array programming with numpy. *Nature*, 2020; 585(7825): 357–362. DOI: *https://doi.org/10.1038/s41586-020-2649-2*
6. **Horn BK, Schunck BG.** Determining optical flow. *Artificial intelligence*, 1981; 17(1–3): 185–203. DOI: *https://doi.org/10.1016/0004-3702(81)90024-2*
7. **Ilg E, Mayer N, Saikia T, Keuper M, Dosovitskiy A, Brox T.** Flownet 2.0: Evolution of optical flow estimation with deep networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017; 2462–2470. DOI: *https://doi.org/10.1109/CVPR.2017.179*
8. **Liu C,** et al. Beyond pixels: exploring new representations and applications for motion analysis. PhD thesis, Massachusetts Institute of Technology; 2009.
9. **Lucas BD, Kanade T.** An iterative image registration technique with an application to stereo vision. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981; 674–679.
10. **Menze M, Heipke C, Geiger A.** Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2018; 140: 60–76. DOI: *https://doi.org/10.1016/j.isprsjprs.2017.09.013*
11. **NVIDIA.** 2020. URL *https://docs.nvidia.com/cuda/*, accessed: 11/06/21.
12. **Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S.** Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019; 32: 8024–8035.
13. **Pathak D, Girshick R, Dollár P, Darrell T, Hariharan B.** Learning features by watching objects move. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017; 2701–2710. DOI: *https://doi.org/10.1109/CVPR.2017.638*

14. **Pont-Tuset J, Perazzi F, Caelles S, Arbeláez P, Sorkine-Hornung A, Van Gool L.** The 2017 DAVIS challenge on video object segmentation; 2017. arXiv preprint arXiv:170400675.
15. **Ravasio CS, Pissas T, Bloch E, Flores B, Jalali S, Stoyanov D, Cardoso JM, Da Cruz L, Bergeles C.** Learned optical flow for intra-operative tracking of the retinal fundus. *International journal of computer assisted radiology and surgery*, 2020; 15(5): 827–836. DOI: *https://doi.org/10.1007/s11548-020-02160-9*
16. **Royo D.** flowvid: Optical flow video tools; 2021. URL *https://pypi.org/project/flowvid/*, accessed: 11/06/21.
17. **Runia T.** flow-vis: Easy optical flow visualisation in python; 2021. URL *https://pypi.org/project/flow-vis/*, accessed: 11/06/21.
18. **Seznec M.** flowpy: Tools for working with optical flow; 2021. URL *https://pypi.org/project/flowpy/*, accessed: 11/06/21.
19. **Sun D, Yang X, Liu MY, Kautz J.** PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018; 8934–8943. DOI: *https://doi.org/10.1109/CVPR.2018.00931*
20. **Teed Z, Deng J.** RAFT: Recurrent all-pairs field transforms for optical flow. In: *European Conference on Computer Vision*, Springer, 2020; 402–419. DOI: *https://doi.org/10.1007/978-3-030-58536-5_24*
21. **Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J,** et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 2020; 17(3): 261–272. DOI: *https://doi.org/10.1038/s41592-019-0686-2*

TO CITE THIS ARTICLE:
Ravasio CS, Da Cruz L, Bergeles C 2021 oflibnumpy & oflibpytorch: Optical Flow Handling and Manipulation in Python. *Journal of Open Research Software,* 9: 31. DOI: *https://doi.org/10.5334/jors.380*

Submitted: 21 June 2021 Accepted: 10 November 2021 Published: 26 November 2021