`pykitPIV`: Rich and reproducible synthetic image generation for training machine learning algorithms in velocimetry

Kamila Zdybał*[a], Claudio Mucignat[a], Stefan Kunz[a] and Ivan Lunati[a]

[a]*Laboratory for Computational Engineering, Swiss Federal Laboratories for Materials Science and Technology, Empa, Dübendorf, Switzerland*

ARTICLE INFO

ABSTRACT

We describe `pykitPIV`, a Python library for generating synthetic images that mimic those obtained from particle image velocimetry (PIV) or background-oriented Schlieren (BOS) experimental techniques in fluid dynamics. The library can be readily integrated with machine learning (ML) algorithms, such as convolutional neural networks, variational approaches, or reinforcement learning. This can support the current goals in the velocimetry community for faster and more accurate real-time experimental inference, moving towards autonomous experimentation. Our image generation exploits the kinematic relationship between two PIV/BOS snapshots, advecting particles from one time frame to the next with a second-order accurate numerical scheme. This results in paired image intensities, (I_1, I_2), separated by Δt in time. The goal of this library is to give the user, or the ML agent, extensive flexibility in selecting various parameters that would normally be available in an experimental setting. This includes particle seeding density, thickness of the laser plane, camera exposure, particle loss due to out-of-plane movement, experimental noise, *etc*. The richness and reproducibility of image generation can help advance the growing development of ML applications in experimental fluid dynamics.

1. Motivation and significance

The last decade has seen advances in training convolutional neural networks (CNNs) for optical flow estimation, *i.e.*, predicting motion information from recorded image frames separated by a short time interval. To date, numerous network architectures have been developed that are highly specialized for this application. These include various implementations of FlowNets [1–3], the spatial pyramid network (SPyNet) [4], the pyramid, warping, and cost-volume network (PWC-Net) [5], and, more recently, the recurrent all-pairs field transforms (RAFT) [6]. In addition, the introduction of iterative residual refinement (IRR) [7] allowed for a significant reduction in the number of trainable parameters thanks to weight sharing at several levels of successively upscaled image resolution.

Experimental fluid dynamics can especially profit from those architectures. Specifically, particle image velocimetry (PIV) and background-oriented Schlieren (BOS) are experimental techniques used to visualize flow patterns with high precision. Their main goal is to predict flow targets, such as displacement fields, velocity components, or vorticity, either from paired snapshots of illuminated tracer particles injected into the flow (PIV) or from recorded deformations of a dotted background image. Recently, RAFT-PIV [8] and lightweight image-matching architecture (LIMA) [9] were proposed as versions of CNNs that are further optimized for inference from velocimetry experiments. Thanks to their targeted architecture and parameters, both RAFT-PIV and LIMA achieve high accuracy and per-pixel spatial resolution. LIMA is also significantly leaner in terms of the number of trainable parameters than its predecessors used for general-purpose optical flow estimation. The successes of

RAFT-PIV and LIMA have been demonstrated on a number of classic experimental fluid dynamics settings such as flow behind a cylinder, boundary layer flow, or convective flow of a hot air plume [10]. Currently, the main precedence that motivates the need for those networks is that they can be trained on GPUs within the matter of hours and then ported to laboratory hardware to make flow predictions in real-time, parallel to experimental measurements. In fact, our group already uses LIMA in real-time for 15Hz PIV, but further improvements to inference speed are needed for higher image acquisition frequencies. The advancements to these architectures are continually being made in the context of PIV [11, 12]. The appealing goal of the PIV community is that CNNs replace and outperform state-of-the-art PIV post-processing in the future.

To advance the accuracy of training machine learning (ML) algorithms for more complex experimental applications, a number of research questions will have to be addressed in the future:

1. How rich should the training dataset be for a given experimental setting?
2. How should we generate new training data samples to accomplish transfer learning, *i.e.*, to make a trained ML model applicable in the next experimental setting, moving us towards autonomous data acquisition and re-training?
3. Are there extreme settings (time separation between images, noise level, laser and camera properties) at which the current CNNs would fail?
4. How we generate support data to extend CNN's range of applicability?
5. As ML for PIV post-processing becomes widely used, how do we make sure that training-data generation is reproducible and can be easily shared between research groups?

✉ kamila.zdybal@gmail.com (K. Zdybał*)
ORCID(s):

To help researchers answer those questions, in this paper, we describe pykitPIV (**Py**thon **ki**nematic **t**raining for **PIV**), a Python library for synthetic PIV image generation that allows to create rich and challenging experimental scenarios. The library generates paired image intensities, I_1 and I_2, separated by Δt in time, and the corresponding displacement fields, $d\mathbf{s} = [d\mathbf{x}, d\mathbf{y}]$ that have per-pixel resolution by construction. pykitPIV exploits the kinematic relationship between two consecutive PIV image frames [13]. Given any velocity field, tracer particles are advected from one time frame to the next using a second-order accurate numerical scheme. pykitPIV thus provides PIV-like images of illuminated tracer particles and the associated PIV postprocessing targets (*e.g.*, $d\mathbf{s}$) which establish the ground truth for ML algorithms. This is in contrast to raw experimental data which lacks the ground truth. In fact, synthetic dataset are scarce and often do not exploit challenging flow scenarios. pykitPIV addresses this gap and allows for rich experimental conditions to be generated.

To date, we have been able to identify four openly-available synthetic image generation (SIG) packages: one written in the ANSI C language coming from the EUROPIV project [14], two written in MATLAB [15, 16], and third package, implemented in both MATLAB and Python, with a limited scope in order to specifically track defocusing and tackle astigmatic PIV. These existing SIG implementations make integrating with Python interfaces more difficult. With the emerging ML applications, a package that readily ports to libraries such as PyTorch [17, 18], TensorFlow [], or Keras [] is the ideal solution. Our library allows for easier porting with ML algorithms. This not only allows to generate training data for ML in a single Python workflow, but also allows the ML algorithm to interact with the image generation process. Specifically, the high flexibility in the types of created images can port very well with variational approaches (VA) or with reinforcement learning (RL) algorithms, where an agent may learn to augment the training dataset in real-time to account for changes in experimental settings. Within the tutorials provided with the software, we delineate interesting examples for each of these ML applications.

2. Software description

2.1. Software architecture

All functionalities of pykitPIV are organized in five classes: Particle, FlowField, Motion, Image, and Postprocess, each achieving its own role in generating synthetic image pairs and the corresponding flow targets. Fig. 1 illustrates the hierarchy of using pykitPIV classes and briefly describes what can be achieved with each class. The user selects the number of image pairs to generate (batch size) and their dimensions (height and width). At each stage of image generation, the user can fix random seeds to assure that data generation is reproducible.

Kamila: Mention that image properties are Monte Carlo-generated! We can show the span of conditions generated.

2.2. Software functionalities
2.2.1. Class: *Particle*
The Particle class seeds the two-dimensional flow domain with tracer particles. The user can steer the range of particle diameters and their standard deviation, the seeding density, or the average distances between particles. The initial particle positions are those appearing on snapshots I_1.

2.2.2. Class: *FlowField*
The FlowField class allows to generate the velocity field to be applied on the two-dimensional domain. We implemented several methods to generate velocity fields, such as random smooth field, checkered field, Chebyshev polynomial field, or spherical harmonics field. Those are illustratively visualized in Fig. 2a-d. This variety of velocity fields span cases with smooth and sharp velocity gradients and can help put machine learning algorithms to test.

The user also has the option of uploading an external velocity field, *e.g.*, coming from a numerical simulation of Navier-Stokes equations (*cf.* Fig. 2e), or coming from a synthetic turbulence generator (*cf.* Fig. 2f) [19, 20].

2.2.3. Class: *Motion*
The Motion class applies the flow field to the particles. It uses a forward Euler or the Runge-Kutta 4th order numeric scheme to advect particles by a user-specified time separation, Δt. Velocity components in-between the grid points are interpolated with a regular grid interpolation. The main output of this class are particle positions that will appear on snapshots I_2, each paired with a respective snapshot I_1.

2.2.4. Class: *Image*
The Image class generates image intensities. It adds the reflected laser light to the generated PIV image pairs. The core functionality is to add a Gaussian intensity to each particle [21, 22]. The user has a lot of flexibility in setting up the laser plane and camera properties. The user can also steer the amount of particles lost between frame I_1 and I_2 due to out-of-plane movement.

The PIV image pair tensor has shape $(N, 2, H, W)$, where N is the batch size, H is image height and W is image width. The second dimension can be thought of as the number of channels and those correspond to I_1 and I_2, respectively. This is compatible with tensor shape accepted by convolutional layers implemented in PyTorch. Note that the whole batch of N images is generated all at once. The Image class contains convenient functions for saving images to .h5 files and for plotting or animating image pairs. pykitPIV uses sequential colormaps by Crameri et al. [23].

The format for generated image tensors is consistent with the input required by the classic convolutional layers available in PyTorch (torch.nn.Conv2d).

2.2.5. Class: *Postprocess*
The Postprocess class contains functions that apply transformations to generated images. It can be especially

pykitPIV

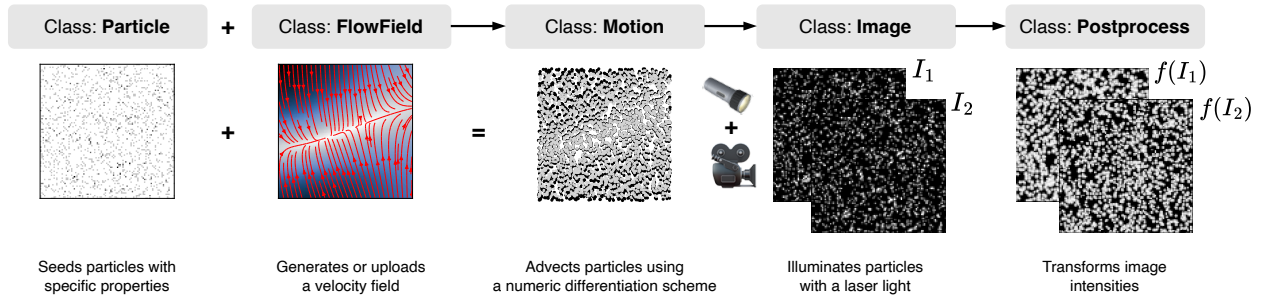| Class: **Particle** | + | Class: **FlowField** | → | Class: **Motion** | → | Class: **Image** | → | Class: **Postprocess** |

Figure 1: Order of using `pykitPIV` classes. At each stage of synthetic image generation, the user has freedom in selecting various parameters that would normally be available in an experimental setting such as particle seeding density, thickness of the laser plane, camera exposure, particle loss due to out-of-plane movement, or time separation between images.
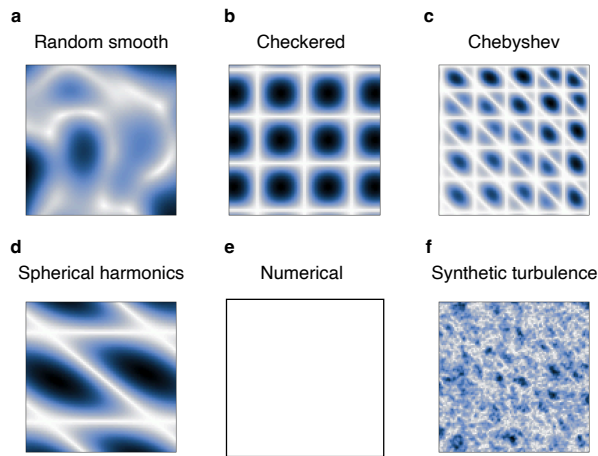

Figure 2: a-d: Types of two-dimensional velocity fields that can be generated with the `FlowField` class. **e-f**: The user also has the option to upload an external velocity field.
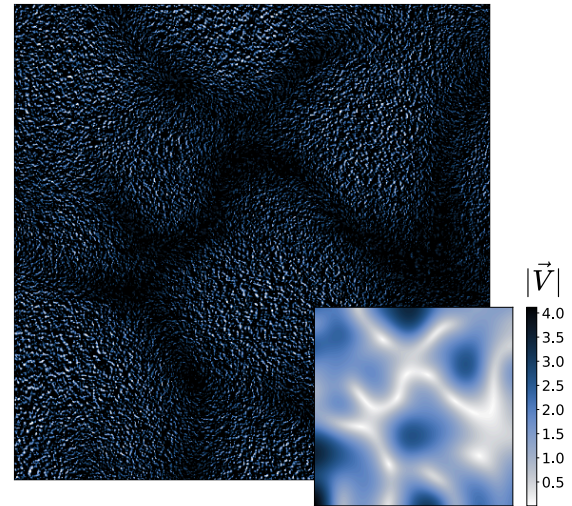

Figure 3: Example visualization of particle motion resulting from superimposing I_1 and $-I_2$ on one image and given the velocity magnitude, $|\vec{V}|$.

useful for data augmentation, where the training dataset is extended with images with various levels of noise or illumination levels.

3. Illustrative examples

Below, we present the most straightforward workflow for generating a batch of $N = 100$ PIV image pairs and their associated targets and we save the resulting tensors for later use in PyTorch. Each image is 256px × 256px. We create a 10px boundary buffer during image generation to allow for new particles to enter the image area and hence to prevent spurious disappearance of particles near image boundaries.

We import `pykitPIV`'s classes and specify the global parameters:

```
from pykitPIV import Particle, FlowField, Motion, Image

n_images = 100
image_size = (256, 256)
size_buffer = 10
random_seed = 100
```

We instantiate an object of the `Particle` class:

```
particles = Particle(n_images,
                     size=image_size,
                     size_buffer=size_buffer,
                     diameters=(4,4.1),
                     distances=(1,2),
                     densities=(0.05,0.1),
                     diameter_std=0.2,
                     seeding_mode='random',
                     random_seed=random_seed)
```

We instantiate an object of the FlowField class and we generate random smooth velocity fields for each of the $N = 100$ image pairs:

```
flowfield = FlowField(n_images,
                      size=image_size,
                      size_buffer=size_buffer,
                      random_seed=random_seed)

flowfield.generate_random_velocity_field(
        gaussian_filters=(10,11),
        n_gaussian_filter_iter=10,
        displacement=(2,10))
```

We instantiate an object of the `Motion` class and we run a forward Euler numerical scheme to advect the particles:

```
1  motion = Motion(particles,
2                  flowfield,
3                  time_separation=0.1)
4
5  motion.forward_euler(n_steps=10)
```

Finally, we instantiate an object of the `Image` class, add particles, flow field, and motion objects and add reflected light with the specific laser plane properties:

```
1   image = Image(random_seed=random_seed)
2
3   # Add particles to images:
4   image.add_particles(particles)
5
6   # Add flow field to images:
7   image.add_flowfield(flowfield)
8
9   # Add motion to images:
10  image.add_motion(motion)
11
12  # Add reflected light to images:
13  image.add_reflected_light(exposures=(0.7,0.8),
14                            maximum_intensity=2**16-1,
15                            laser_beam_thickness=1,
16                            laser_over_exposure=1,
17                            laser_beam_shape=0.95,
18                            alpha=1/10)
```

Once all images are created, we can remove buffers from the images, convert image pairs and the flow targets to tensors, and save tensors to `.h5` files for future use:

```
1   # Remove image buffers:
2   image.remove_buffers()
3
4   # Prepare image tensors:
5   image_pairs = image.image_pairs_to_tensor()
6   targets = image.targets_to_tensor()
7
8   # Save images to .h5:
9   image.save_to_h5({'I': image_pairs,
10                    'targets': targets},
11                   filename='PIV-dataset.h5')
```

4. Impact

The kinematic training methodology [13], which is the core concept behind generating PIV image pairs with `pykitPIV`, has been used to train CNNs in optical flow estimation [9, 10, 13]. The approach proved successful, which suggests that for small Δt, learning the kinematic relationship between two consecutive PIV snapshots, as opposed to knowing the full dynamic relationship, is sufficient to train CNNs.

Kamila: It would be great if we could extend image generation to synthetic event-based camera datasets. This would make the software truly novel.

Kamila: Perhaps a nice novelty would be to allow the user to add solid boundaries into the image?

4.1. Porting with convolutional neural networks

Kamila: Here we can describe what can be achieved in terms of training a CNN.

4.2. Porting with reinforcement learning

Kamila: Here we can describe what can be achieved in terms of training an RL agent, e.g. in the context of autonomous experimentation. Maybe the agent will learn to augment the dataset in real time to account for changing experimental settings.

5. Conclusions

We plan a continued development of this library.

Future application can also include the use variational approaches to inform training data collection, or train a RL agent to construct necessary support data in new environments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author contributions

Acknowledgments

References

[1] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, T. Brox, FlowNet: Learning optical flow with convolutional networks, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 2758–2766.

[2] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox, FlowNet 2.0: Evolution of optical flow estimation with deep networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2462–2470.

[3] T.-W. Hui, X. Tang, C. C. Loy, Liteflownet: A lightweight convolutional neural network for optical flow estimation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8981–8989.

[4] A. Ranjan, M. J. Black, Optical flow estimation using a spatial pyramid network, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4161–4170.

[5] D. Sun, X. Yang, M.-Y. Liu, J. Kautz, 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, pp. 8934–8943.

[6] Z. Teed, J. Deng, Raft: Recurrent all-pairs field transforms for optical flow, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, Springer, 2020, pp. 402–419.

[7] J. Hur, S. Roth, Iterative residual refinement for joint optical flow and occlusion estimation, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 5754–5763.

[8] C. Lagemann, K. Lagemann, S. Mukherjee, W. Schröder, Deep recurrent optical flow learning for particle image velocimetry data, Nature Machine Intelligence 3 (7) (2021) 641–651.

[9] L. Manickathan, C. Mucignat, I. Lunati, A lightweight neural network designed for fluid velocimetry, Experiments in Fluids 64 (10) (2023) 161.

[10] C. Mucignat, L. Manickathan, J. Shah, T. Rösgen, I. Lunati, A lightweight convolutional neural network to reconstruct deformation in bos recordings, Experiments in Fluids 64 (4) (2023) 72.

[11] L. Shan, L. Xiaoying, J. XIONG, B. Hong, J. Jian, M. Kong, A lightweight optical flow model for particle image velocimetry, Flow Measurement and Instrumentation (2024) 102762.

[12] M. Elrefaie, S. Hüttig, M. Gladkova, T. Gericke, D. Cremers, C. Breitsamter, On-site aerodynamics using stereoscopic piv and deep optical flow learning, Experiments in Fluids 65 (12) (2024) 1–20.

[13] L. Manickathan, C. Mucignat, I. Lunati, Kinematic training of convolutional neural networks for particle image velocimetry, Measurement Science and Technology 33 (12) (2022) 124006.

[14] B. Lecordier, J. Westerweel, The europiv synthetic image generator (sig), in: Particle Image Velocimetry: Recent Improvements: Proceedings of the EUROPIV 2 Workshop held in Zaragoza, Spain, March 31–April 1, 2003, Springer, 2004, pp. 145–161.

[15] H. Ben-Gida, R. Gurka, A. Liberzon, Openpiv-matlab—an open-

source software for particle image velocimetry; test case: Birds' aerodynamics, SoftwareX 12 (2020) 100585.

[16] L. Mendes, A. Bernardino, R. M. Ferreira, piv-image-generator: An image generating software package for planar piv and optical flow benchmarking, SoftwareX 12 (2020) 100537.

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch (2017).

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., PyTorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).

[19] T. Saad, D. Cline, R. Stoll, J. C. Sutherland, Scalable tools for generating synthetic isotropic turbulence with arbitrary spectra, AIAA journal 55 (1) (2017) 327–331.

[20] A. Richards, T. Saad, J. C. Sutherland, A fast turbulence generator using graphics processing units, in: 2018 Fluid Dynamics Conference, 2018, p. 3559.

[21] M. Olsen, R. Adrian, Out-of-focus effects on particle image visibility and correlation in microscopic particle image velocimetry, Experiments in fluids 29 (Suppl 1) (2000) S166–S174.

[22] J. Rabault, J. Kolaas, A. Jensen, Performing particle image velocimetry using artificial neural networks: A proof-of-concept, Measurement Science and Technology 28 (12) (2017) 125301.

[23] F. Crameri, G. E. Shephard, P. J. Heron, The misuse of colour in science communication, Nature communications 11 (1) (2020) 5444.