

# pykitPIV: Rich and reproducible virtual training of machine learning algorithms in velocimetry

Kamila Zdybał<sup>\*a</sup>, Claudio Mucignat<sup>a</sup>, Stefan Kunz<sup>a</sup> and Ivan Lunati<sup>a</sup>

<sup>a</sup>Laboratory for Computational Engineering, Swiss Federal Laboratories for Materials Science and Technology, Empa, Dübendorf, Switzerland

## ARTICLE INFO

### Keywords:

particle image velocimetry; flow estimation; convolutional neural networks; machine learning; reinforcement learning; Python

## ABSTRACT

In recent years, machine learning (ML) has become an increasingly integral part of experimental fluid dynamics. This integration calls for new tools that can emulate the complexity of real experiments while offering the control and reproducibility needed for ML development. Here, we present `pykitPIV`, a Python library that provides rich and reproducible virtual environments for training ML algorithms in optical velocimetry. The generated synthetic datasets and environments mimic those coming from particle image velocimetry (PIV) and background-oriented Schlieren (BOS) experimental techniques. The library integrates with various ML algorithms, such as convolutional neural networks, variational approaches, active learning, and reinforcement learning. This library gives the user, or the ML agent, flexibility in selecting various parameters that would normally be available in an experimental setting, such as seeding density, properties of the laser plane, camera exposure, particle loss, or experimental noise. We also provide an atlas of challenging synthetic velocity fields from analytic formulations where the effects of particle drift and diffusion in stationary isotropic turbulence can also be added using the simplified Langevin model. In that sense, `pykitPIV` acts as a “virtual wind tunnel”, providing a playground for training and testing ML models. With `pykitPIV`, ML agents have the freedom to interact with the virtual experiment, can assimilate data from real experiment, and can be trained to perform a variety of tasks using diverse sensory cues and rewards. Our goal is to support the current trends in the velocimetry community for faster and more accurate real-time experimental inference, moving the field towards autonomous experimentation and building robust models from experimental data.

## 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 optical 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 (in PIV) or from recorded deformations of the dotted background image (in BOS). Recently, RAFT-PIV [8] and lightweight image-matching architecture (LIMA) [9] were proposed as versions of CNNs that are optimized for inference from velocimetry experiments. 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]. The advancements to these architectures are continually being made in the context of PIV [11–15] with the goal to replace and outperform state-of-the-art PIV post-processing in the future. Currently, the main precedence that has motivated 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. Moreover, the advances made in optical flow architectures open up new avenues of research and allow for new machine learning (ML) use-cases to emerge that have not been possible before. In that sense, CNNs can become a gateway to various other ML algorithms such as variational approaches, active learning, or reinforcement learning (RL). This can allow the experimental community to develop new applications such as autonomous experimentation, data assimilation and digital twinning, building models from experimental data, and enhanced flow control.

To advance the development and performance of ML algorithms for complex experimental fluid dynamics applications, a number of research questions will have to be addressed in the future. Answering these questions will help improve the CNN architectures but also will open new research avenues that were not possible thus far:

1. How rich should the training dataset be for a given experimental setting?
2. At which challenging experimental settings do the current CNNs fail and why?

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

3. Can we generate new training data samples to accomplish transfer learning, *i.e.*, to make a trained ML model applicable in the next experimental setting?
4. Can we extend CNN's range of applicability within a single experimental setting with active learning?
5. Can assimilate data collected on-the-fly with the experiment, *e.g.*, using active learning procedure, so that its distribution belongs to the distribution of a given experimental setting?
6. As ML for PIV post-processing becomes widely used, how do we make sure that training settings are reproducible and can be easily shared between research groups?

To help researchers answer those questions and move the field towards novel research directions, in this paper, we present `pykitPIV` (**P**ython **k**inematic **t**raining for **P**IV), a Python library that provides virtual velocimetry environments and ready-to-use ML integrations. In that sense, our library can serve as a “virtual wind tunnel”. At its core, `pykitPIV` exploits the kinematic relationship between two consecutive image frames [16]. Given any velocity field, tracer particles are advected from one time frame to the next using a second-order accurate numerical scheme. Synthetic dataset used to train CNNs are scarce and often do not exploit challenging flow scenarios. `pykitPIV` addresses this gap and allows for rich experimental conditions to be generated and presented to ML algorithms as batches of training data. The associated post-processing targets (*e.g.*, displacement fields) establish the ground truth for ML algorithms. This is in contrast to raw experimental data which lacks the ground truth. Finally, `pykitPIV` contains a standalone ML module which integrates with the virtual wind tunnel and gives the ML agents flexibility in interacting with the experimental settings. We also provide a variety of sensory cues and rewards for training RL agents. This paves the way towards development of autonomous experimentation in the future.

## 2. Software description

### 2.1. Software architecture

The functionalities of `pykitPIV` that provide the “virtual wind tunnel” are organized in five classes: `Particle`, `FlowField`, `Motion`, `Image`, and `Postprocess`, each achieving its role in generating synthetic image pairs and the corresponding flow targets. Beyond the five main classes, we provide a dedicated ML module, `pykitPIV.ml`, which contains wrappers and integrations to various ML algorithms. Fig. 1 illustrates the hierarchy of using `pykitPIV` classes and briefly describes what can be achieved with each class.

Experimental settings can be Monte-Carlo-generated and individual images within a training batch can span a range of conditions. At each stage of image generation, the user can fix random seeds to assure that data generation is reproducible.

### 2.2. Software functionalities

Synthetic PIV/BOS image generation can only be a crude approximation of the real experimental flow conditions. For this reason, our goal in creating `pykitPIV` is to capture various complexities of the real-world experimental settings. For example, the effect of particle drift and diffusion in stationary isotropic turbulence can be added atop any synthetic velocity field. This procedure is modeled with the simplified Langevin model (see §12.6 in [18] for more information).

The user selects the number of image pairs to generate (batch size) and their dimensions (height and width).

The `pykitPIV` library generates paired image intensities,  $I_1$  and  $I_2$ , separated by  $\Delta t$  in time, and the corresponding displacement fields,  $ds = [dx, dy]$  that have per-pixel resolution by construction.

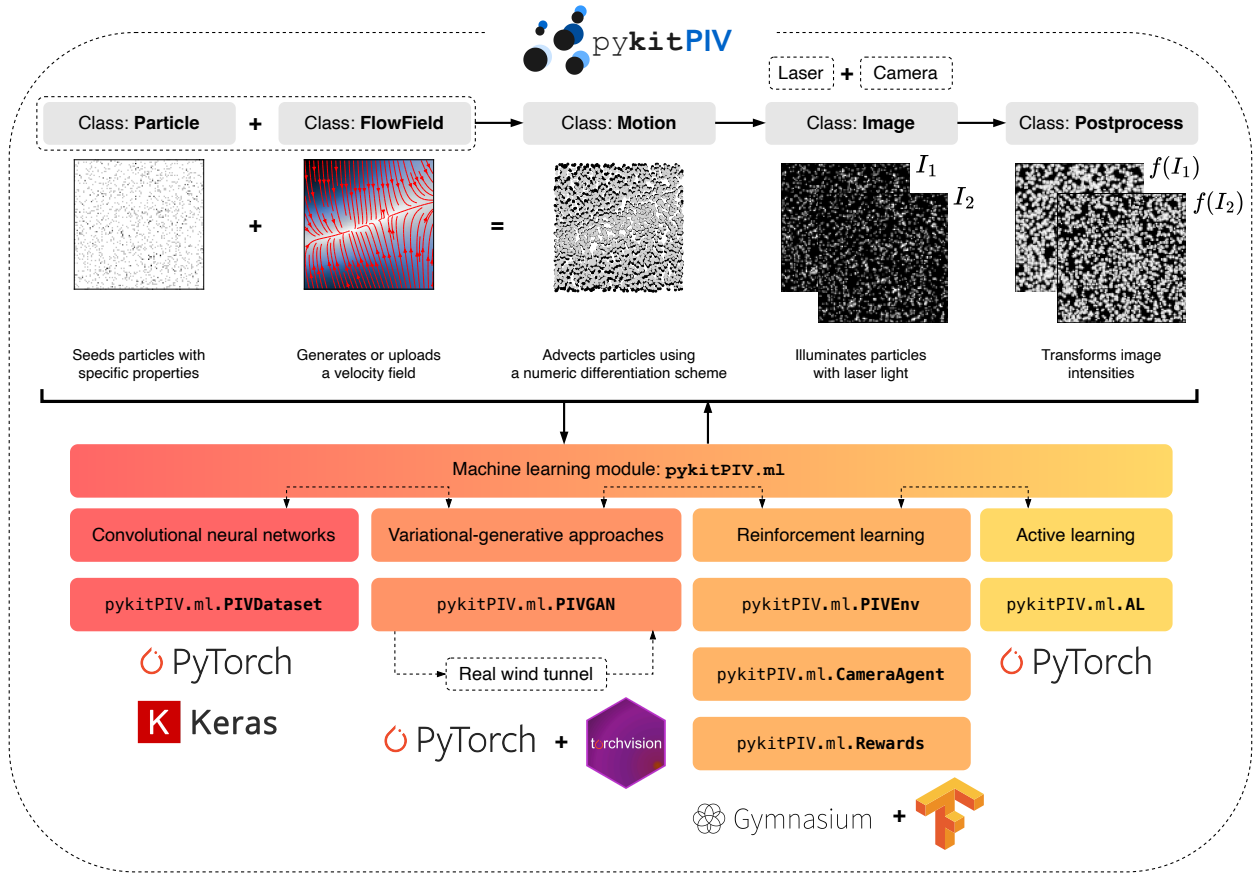
The `Particle` class seeds the two-dimensional flow domain with tracer particles. The user can steer the range of particle diameters and their standard deviation, the seeding density, or the average distances between particles. The initial particle positions are those appearing on snapshots  $I_1$ .

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].

The `Motion` class applies the flow field to the particles. It uses the forward Euler or the Runge-Kutta 4th order numeric scheme to advect particles by a user-specified time separation,  $\Delta t$ . 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  $I_1$ .

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 (`torch.nn.Conv2d`). Note that the whole batch of  $N$  images can be generated all at once or can be optimized for memory and generated in batches. The `Image` class contains convenient functions for saving images to .h5 files and for plotting or animat-



**Figure 1:** Order of using the five main `pykitPIV` classes that act as a virtual experimental setup with which the ML module, `pykitPIV.ml`, interacts. At each stage of synthetic virtual experimentation, the user, or the ML agent, has freedom in selecting various parameters that would normally be available in an experimental setting such as the seeding density, properties of the laser plane, camera exposure, particle loss, or experimental noise. The various ML components can also communicate and enrich each other.

ing image pairs. `pykitPIV` uses sequential colormaps by Crameri et al. [23].

The `Postprocess` class contains functions that apply transformations to generated images. It can be especially useful for data augmentation, where the training dataset is extended with images with various levels of noise or illumination levels.

### 3. Illustrative examples

#### 3.1. Porting with convolutional neural networks

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

Thanks to its targeted architecture and parameters, LIMA achieves high accuracy and per-pixel spatial resolution. Our group already uses LIMA in real-time for 15Hz PIV, but further improvements to inference speed are needed for higher image acquisition frequencies.

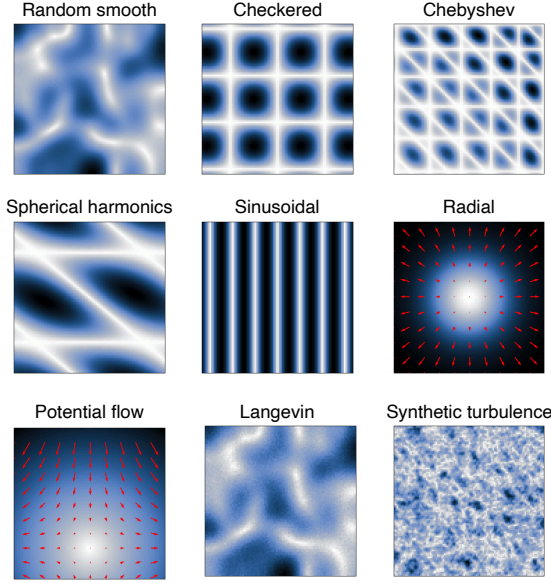
#### 3.2. Reinforcement learning environments

Here, we present how `pykitPIV`'s functionalities can be used to train a RL agent to locate regions of high di-

vergence in a radial flow by moving a virtual camera. The virtual wind tunnel is modeled as a subclass of the Gymnasium environment [24]. A registered Gymnasium environment can be ported to RL libraries, such as TF-Agents [25]. The basic structure of the PIV environment class is presented below:

```
1 import gymnasium as gym
2
3 class PIVEnv(gym.Env):
4
5     def reset():
6
7         ...
8
9     def step():
10
11         ...
12
13     def render():
14
15         ...
16
17     def record_particles():
18
19         ...
20
21     def make_inference():
22
23         ...
```

The functions `reset()`, `step()`, `render()` are commonly implemented in all reinforcement learning environ-



**Figure 2:** The types of two-dimensional velocity fields that can be generated with the FlowField class. The user also has the option to upload an external velocity field, e.g., coming from synthetic turbulence or turbulence databases.

ments and allow to reset the environment to an initial state, make a step in the environment, and render the current state of the environment, respectively.

The function `record_particles()` performs a “virtual PIV” at each step in the environment. The function `make_inference()` allows to use an inference model (such as a CNN model) to predict flow targets from PIV recordings. PIV parameters can be passed to the PIVEnv class through constructors: `ParticleSpecs`, `FlowFieldSpecs`, `MotionSpecs`, and `ImageSpecs`. An example constructor for particle properties can be:

```
1 particle_spec = {'diameters': (1, 4),
2                 'densities': (0.05, 0.2),
3                 'diameter_std': (0.1, 0.5),
4                 'min_diameter': 0.1,
5                 'seeding_mode': 'random',
6                 'random_seed': 100}
```

The sensory cues that the agent has available at each training step are 32 sampled displacement vectors at each interrogation window. The reward is constructed as per the following equation:

$$R = \max(|\nabla \cdot \vec{d}\vec{s}|) \quad (1)$$

The basic abstraction to use this reward in pykitPIV is to use the Rewards class to access one of the ready reward functions. Note that the rewards functions accept a custom transformation function that gives the user extra flexibility in how the final reward value is computed. The example for creating a reward as in Eq. (1) is presented below.

```
1 from pykitPIV import Rewards
2 import numpy as np
3
```

```
4 # Once we have the velocity field specified:
5 velocity_field = ...
6
7 # Instantiate an object of the Rewards class:
8 rewards = Rewards(verbose=True,
9                  random_seed=None)
10
11 # Design a custom transformation that looks for regions
12 # of high divergence (either positive or negative)
13 # and computes the maximum absolute value of divergence
14 # in that region:
15 def reward_transformation(div):
16     return np.max(np.abs(div))
17
18 reward = rewards.divergence(velocity_field=
19                             velocity_field,
20                             transformation=
21                             transformation)
```

Table ?? presents the available rewards and sensory cues functions.

**Kamila:** Here we can describe what can be achieved in terms of training an RL agent, e.g. in the context of autonomous experimentation.

We import pykitPIV’s classes and specify the global parameters:

```
1 from pykitPIV import Particle, FlowField, Motion, Image
2
3 n_images = 100
4 image_size = (256, 256)
5 size_buffer = 10
6 random_seed = 100
```

We instantiate an object of the Particle class:

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

## 4. Impact

The kinematic training methodology [16], which is the core concept behind generating PIV image pairs with pykitPIV, has been used to train CNNs in optical flow estimation in the earlier works coming from our group [9, 10, 16]. Parts of pykitPIV are translated from the earlier MATLAB code used in our group. The approach proved successful, which suggests that for small enough  $\Delta t$ , learning the kinematic relationship between two consecutive PIV snapshots, as opposed to knowing the full dynamic relationship, is sufficient to train CNNs.

Thus far, ML algorithms for PIV are often trained using synthetic images generated with open-source realistic flowfields, e.g., taken from the John Hopkins Turbulence Database (JHTD) [17]. While the JHTD database is openly available, the synthetic image generation is often performed by different groups using in-house codes. We have hopes that providing pykitPIV as an open-source Python library, researchers can easily share their image generation workflows, making the training of ML algorithms fully reproducible.

[26]

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 [27], two written in MATLAB [28, 29], 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 [30, 31], TensorFlow [32], or Keras [33] is the ideal solution. Our library allows for 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.

Finally, the powerful novelty of this package, as compared to previous MATLAB-based packages, is the introduction of new adjustable parameters that mimic various aspects of the experimental settings.

Another interesting application of CNNs is for learning location of particles for particle tracking velocimetry [34]

or exploring unsupervised training of CNNs that does not require flow targets [35].

Notably, our library can integrate with the post-processing Python software developed in [36].

## 5. Conclusions

The richness of experimental conditions and reproducibility of training data generation can help advance the growing development of ML applications in experimental fluid dynamics.

We plan a continued development of this library. The future functionalities will include: extension from 2D to 3D environments with the 3D PIV images generated using camera projections coming from the OpenCV 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. Rösger, I. Lunati, A lightweight convolutional neural network to reconstruct deformation in bos recordings, *Experiments in Fluids* 64 (4) (2023) 72.
- [11] C. Yu, X. Bi, Y. Fan, Y. Han, Y. Kuai, LightPIVNet: An effective convolutional neural network for particle image velocimetry, *IEEE Transactions on Instrumentation and Measurement* 70 (2021) 1–15.
- [12] Y. Fan, C. Guo, Y. Han, W. Qiao, P. Xu, Y. Kuai, Deep-learning-based image preprocessing for particle image velocimetry, *Applied Ocean Research* 130 (2023) 103406.
- [13] J. S. Choi, E. S. Kim, J. H. Seong, Deep learning-based spatial refinement method for robust high-resolution piv analysis, *Experiments in Fluids* 64 (3) (2023) 45.
- [14] 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.
- [15] 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.
- [16] L. Manickathan, C. Mucignat, I. Lunati, Kinematic training of convolutional neural networks for particle image velocimetry, *Measurement Science and Technology* 33 (12) (2022) 124006.
- [17] E. Perlman, R. Burns, Y. Li, C. Meneveau, Data exploration of turbulence simulations using a database cluster, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, 2007, pp. 1–11.
- [18] S. B. Pope, Turbulent flows, *Measurement Science and Technology* 12 (11) (2001) 2020–2021.
- [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.

- [24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, arXiv preprint arXiv:1606.01540 (2016).
- [25] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, E. Brevdo, TF-Agents: A library for reinforcement learning in tensorflow, <https://github.com/tensorflow/agents>, [Online; accessed 25-June-2019] (2018).  
URL <https://github.com/tensorflow/agents>
- [26] C. Mucignat, B. Roth, I. Lunati, A respiratory simulator for the study of pathogen transmission in indoor environments, *Indoor Air* 2024 (1) (2024) 8368202.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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).
- [31] 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).
- [32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).  
URL <https://www.tensorflow.org/>
- [33] F. Chollet, et al., Keras, <https://keras.io> (2015).
- [34] P. Godbersen, D. Schanz, A. Schröder, Peak-CNN: improved particle image localization using single-stage CNNs, *Experiments in Fluids* 65 (10) (2024) 153.
- [35] C. Lagemann, K. Lagemann, S. Mukherjee, W. Schröder, Challenges of deep unsupervised optical flow estimation for particle-image velocimetry data, *Experiments in Fluids* 65 (3) (2024) 30.
- [36] J. Aguilar-Cabello, L. Parras, C. del Pino, DPIVSoft-OpenCL: A multicore CPU–GPU accelerated open-source code for 2D Particle Image Velocimetry, *SoftwareX* 20 (2022) 101256.