Chapter IV

Baby Videos: High-speed Optical Microscopy Observes Early Growth of Bubbles

> You can observe a lot just by watching.
>
> Yogi Berra

> The question is not what you look at, but what you see.
>
> Henry David Thoreau, August 5, 1851

Data analysis was made significantly more streamlined and efficient by Isaac Swanlund. Much thanks to Larry Vladic and Michael Estela from Elite Motion Systems, LLC, for assistance in setting up the high-speed camera.

An apparatus is useless without detection. A powerful mode of detection for humans is visual observation. Our eyes train our whole lives to see patterns: they are just missing patterns to look at. While X-ray or light scattering would have provided earlier detection of bubble nuclei, optical microscopy provided a powerful platform for our intuition. Just by watching, we learned a tremendous amount about this system. Unfortunately, our brains and eyes are slow: just watching all the recorded video generated for the present thesis would take weeks, let alone analyzing it. Here, we briefly describe the image-processing algorithms that watched and analyzed these videos. From these videos, the algorithms detect, track, and measure each bubble, and extract the size, speed, position, time, and other important measured properties into a reduced dataset. This dataset provides the input for fitting the bubble growth model presented in Chapter V that ultimately predicts the conditions of nucleation discussed in Chapters VI and VII.

IV.1 Image Processing Detects, Tracks, and Measures Bubbles

Amidst the many phenomena occurring in the flow-focusing channel, the image-processing algorithm detected, tracked, and measured bubbles observed under high-speed microscopy as they flowed down the flow-focusing channel. While it could not identify bubble nucleation due to the limited resolution of the microscope, the algorithms could measure the early growth precisely enough to fit a model that could predict the nucleation (Chapter V). The key components of this algorithm were background subtraction, image segmentation, and object tracking. These algorithms were primarily based on the OpenCV computer vision library [1]. The completed `CvVidProc` algorithm is available on github [2], as is its implementation for the present work `bubbletracking_koe` [3].

Background Subtraction

Among the most important steps of successful image processing is the distinction between objects of interest and background. Images are rich with detail, but often only a few of those details matters to the analysis. In this case, few frames in each video contained a bubble, and the bubble occupied a small fraction of the field of view. A simple technique for distinguishing the objects of interest is to generate an estimate for the parts of the image that are not interesting (the "background") and subtract it from each frame. In more complex measurements, this background may change over time. Here, we fixed the position of the observation capillary within the field of view of the microscopy during each recording, so the background remained static, excepting an instability in the inner stream (see Section VIII.4).

We considered three methods for background subtraction, which are compared in Figure IV.1: selecting a frame without any observable bubbles (usually the first frame), taking the mean of several frames, and taking the median of several frames. The median is typically recommended [4], but can be computationally more expensive due to higher memory requirements without a multithreaded algorithm like `CvVidProc` [2]. Indeed, taking the median provided the cleanest image of the background despite the presence of bubbles in some of the frames (*e.g.*, the first frame). The median algorithm fails, however, if there are pixels that are obstructed by objects in more than half of the frames sampled, which could occur in videos taken near the outlet of the observation capillary where a continuous foam (see Figure III.3) or fluid instability (see Figure VIII.11) had formed.

Using the median as the background, we subtracted the background from

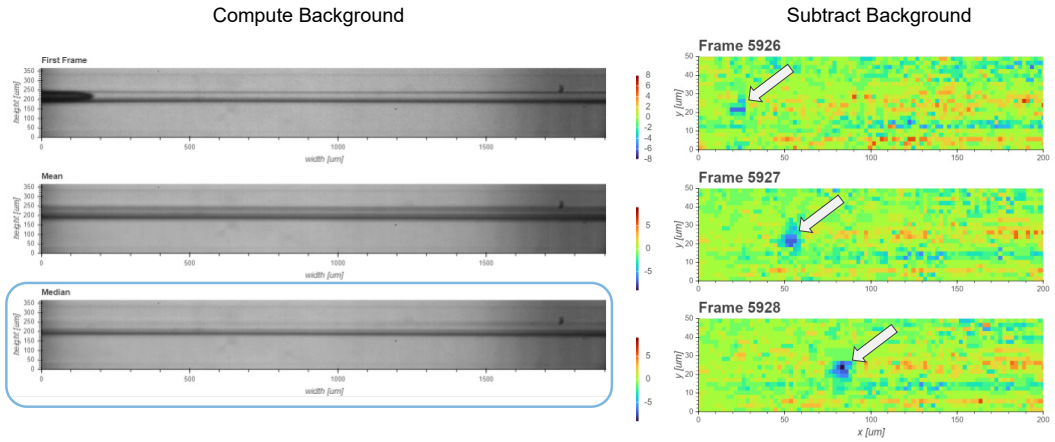

Figure IV.1: Left) Three background-subtraction methods are compared: selecting the first frame, taking the mean of the pixels in several frames, and taking the median of the pixels in several frames. Selecting an individual frame can sometimes include an anomaly or even an object, such as the bubble emerging from the left in the example shown. The mean is sensitive to objects in the foreground like bubbles, resulting in the darker pigment of the inner stream. The median provides the most accurate estimate of the background (circled in blue). Right) A sequence of three frames with the median-calculated background subtracted. The sign of the difference is kept and given a false coloring, in which red represents positive values (brighter than the background), blue represents negative values (darker than the background), and green represents values near 0. A bubble can be discerned as a blue spot growing in size from frame to frame.

each image in a high-speed video of the flow-focusing channel. Because bubbles are always darker than the background due to their strong scattering of light, we kept track of the sign of the image upon subtracting the background, which is not commonly implemented in background-subtraction algorithms due to the inconvenience and added memory of changing from an unsigned to a signed datatype. When detecting bubbles, we could ignore any positive differences from the background (brighter regions) and apply image segmentation only to the darker regions. On the right of Figure IV.1 are three example frames after applying background subtraction. False-coloring allows for the visualization of positive (red) and negative (blue) differences from the background. These frames show the first optical detection of a bubble (frame 5926) followed by its subsequent growth (bubble is blue spot indicated by white). Due to the partial volume effect resulting from the large pixel size relative to the initial bubble size, the first signal of the bubble is fainter than in later frames when the bubble has grown. Nevertheless, the signal from even the smallest bubble ($\sim 1\ \mu$m) is easily distinguishable from the noise in the background, indicating that bubbles could be segmented almost at pixel scale.

Image Segmentation

Having removed the background and enhanced the signal from the bubbles, we perform image segmentation, in which pixels are classified as belonging to different objects or to the background. Image segmentation provides the basis for measurement because it reveals the spatial extent of each object. While the background has been subtracted, it has not yet been identified, so the first step of most image segmentation is the distinction between background and foreground. This distinction can quickly be made by applying a threshold to the image: pixels with values (after subtracting) beyond the threshold are classified as objects (foreground) while the rest are classified as background. The edges of bubbles tend to be fainter than the core, however. If applying a uniform threshold, a high threshold will exclude these dimmer edges while a low threshold will risk the inclusion of noise in the foreground; a useful compromise is not always feasible. Instead, we apply a hysteresis threshold, whose operation according to the `scikit-image` package [5] is depicted schematically in Figure IV.2a. After applying a high uniform threshold, a hysteresis threshold will apply a lower threshold to pixels contiguously connected to those pixels that exceeded the high threshold. The result is more accurate detection of edges with less detection of noise, and the resulting segmented shape more accurately represents the object (see Figure IV.2b).

In the present work, both a uniform and hysteresis threshold are combined. The thresholds are determined by performing the analysis at a sequence of threshold values for a few videos and identifying a value for which the number of true bubbles detected varies minimally under perturbations to the threshold value.

After separating foreground from background, accurate image segmentation relies on the application of processing steps that utilize the unique properties of the objects of interest. For example, because bubbles are generally round, we can apply erosion and dilation steps to smooth out the edges of objects and arrive at less noisy segmentations of bubbles. We also fill holes in our segmentation because the center of some larger bubbles may appear translucent and thus be counted as part of the background. Because we know that these regions are always surrounded by pixels classified as foreground, they can be added to the foreground by filling holes. Finally, because we cannot always distinguish the detection of a bubble the size of a single pixel and salt-and-pepper noise in the background, we require the foreground to include only objects larger than a minimum number of pixels (usually 4). Once segmented, area, dimensions, orientation, centroid, position, and other properties

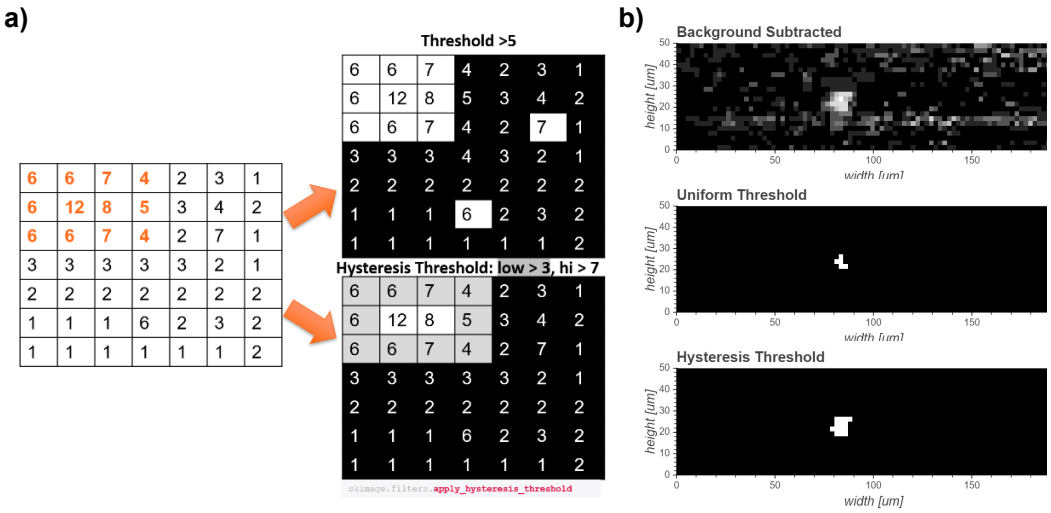

Figure IV.2: Schematic of hysteresis thresholding, as implemented by `scikit-image` [5]. a) The large array of pixels represents the pixel values following background subtraction. The values highlighted in orange are those representing the object (a rectangle), while the values in black text resulted from noise. Using a uniform threshold of 5 incorrectly segments the object and detects noise (follow orange arrow up and to the right). Perfect segmentation is achieved by using a hysteresis threshold, which first applies a high threshold of 7 followed by a threshold of 3 on pixels contiguously connected to those that passed the high threshold (follow orange arrow down and to the right). b) Top image shows a background-subtracted frame containing a small bubble (gray patch in the center). Applying a uniform threshold that does not pick up any noise poorly segments the object and the shape is unrecognizable (middle image). Applying a hysteresis threshold yields a better segmentation (bottom image).

of objects in the foreground can be measured. An example result of this image-segmentation algorithm and subsequent measurement is shown in Figure IV.3, in which an especially complex image is parsed into reasonable representations of bubbles. Further filtering by shape and dimensions can distinguish bubbles (round edges and less slender) from particles (jagged edges and often more slender).

Object Tracking

To track objects between frames, we adopted the "tracking-by-detection" paradigm, in which object detection and tracking are separate tasks, with tracking relying on the features and position of the objects detected (for a deeper discussion of this paradigm, refer to Chapter 2 of the thesis by Murray [6]). After segmenting the objects in two consecutive frames, we compared the distances between the positions of the objects. We used a custom distance metric to take into account that bubbles

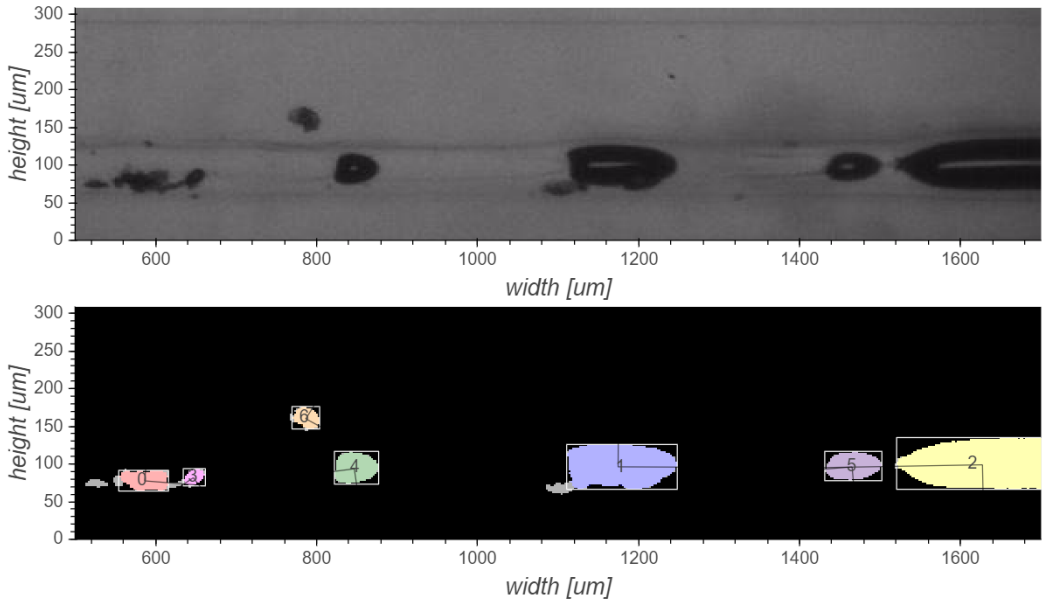

Figure IV.3: Example of the result of applying the image segmentation algorithm described in the text. Given a frame from a video recording of a foaming experiment (top image), the image segmentation algorithm identifies which pixels belong to which object, labels the objects (label drawn at centroid), and computes their orientation and dimensions, among other properties not shown (bottom image). Gray regions are those that passed the threshold but were excluded from the segmented object by the segmentation algorithm. Note that non-bubble features like the contaminant particle at the lower left of object 1 and the particles surrounding object 0 are removed by the algorithm.

tend to travel at a consistent speed along the flow direction. Distances off the flow axis were penalized more highly and distance along the flow axis was measured from the predicted position based on the estimated speed. An object that appeared upstream of an object in a subsequent frame was considered infinitely far away based on the assumption that bubbles only travel downstream. Using this distance metric, we applied the classic Hungarian algorithm to associate objects of the same identity in consecutive frames [7], as depicted schematically in Figure IV.4.

After evaluating the distance metric d between each pair of objects between the two consecutive frames, the resulting distance matrix is searched for the smallest value, which is 2 in this example (orange circle). Object 3 in the previous frame is then identified as the same object as object c in the new frame, and all entries in the corresponding row and column are removed from the matrix (indicated by orange lines). The algorithm repeats the process, looking for the smallest distance metric in the remaining matrix, which is 4 (blue circle). Note that, although the distance

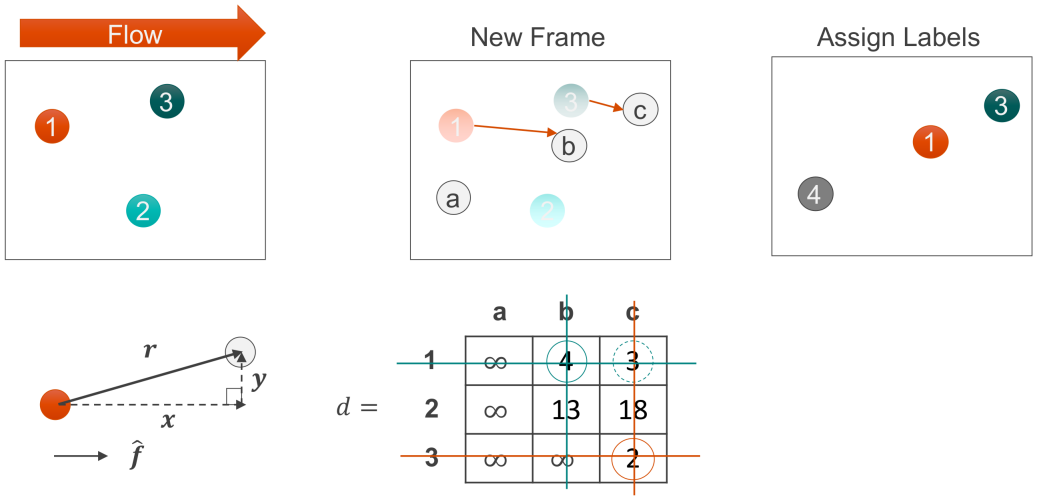

Figure IV.4: Schematic depicting the implementation of the Hungarian tracking algorithm [7] with a custom distance metric. Given a frame in which three objects have been segmented (1, 2, and 3 in the upper left) with flow from left to right, and given a second "New Frame" in which three other objects have been segmented (a, b, and c in the upper center), the distance metric is computed between each pair of objects between the frames by weighing the on- and off-flow-axis distances differently (depicted by triangle in lower left), where any bubble that is upstream of another bubble is treated as infinitely far away. The resulting distances are given in the matrix d (lower center). The algorithm successively identifies the smallest distance in the matrix, identifies the objects corresponding to its row and column, and removes that row and column from the matrix. Specifically, it first matches objects 3 and c (orange circle and omission of row and column crossed out by orange lines) and then matches objects 1 and b (same but in blue). Objects 2 and a are not matched because they are treated as infinitely far away, so object 2 is removed from the list and object a is registered as a new object, object 4. The new labels are assigned and can be used for tracking in the next pair of consecutive frames (upper right).

between object 1 and object c is smaller (3, blue dashed circle), because object c has already been assigned to object 3, this value is omitted from the search. Object 1 in the previous frame is then identified as the same object as object b in the new frame, and all entries in the corresponding row and column are removed from the matrix (indicated by blue lines). Because the only remaining distance is infinite (between object 2 and object a), we declare those two objects as distinct. Object 2 is then removed from our list of objects and object a is added as object 4 for the next frame. This process repeats until the end of the video is reached. In a more sophisticated implementation of this algorithm, "memory" can be implemented, in which case objects are not removed from the list of objects in the previous frame until

they have been absent for multiple consecutive frames. In this implementation, the object's position is extrapolated from its previous locations. While this strategy may improve the robustness of tracking some fainter bubbles through inhomogeneities in the background, it would also incorrectly assign a speck of noise that was registered in one frame to a true object a few frames later, so it was omitted from the final analysis.

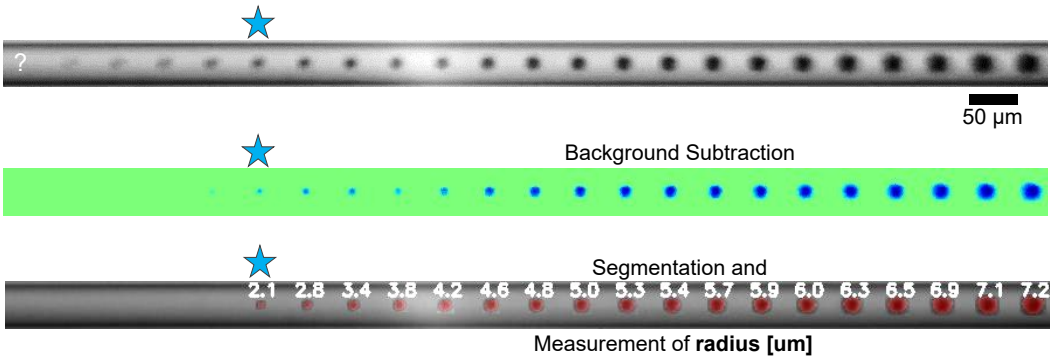

Figure IV.5: The result of image segmentation and tracking is applied to a bubble detected in an inner stream of PPG 2700 g/mol saturated with CO_2 at 7.0 MPa and 22 °C (same conditions as plotted in Figure V.7). Top) Superimposed images of bubble as it flows and grows along inner stream (outer stream outside field of view). Middle) Top frame after background subtraction of the median and application of a hysteresis threshold. Some images of the bubble are removed because they were too faint to pass the threshold. Bottom) Result of image segmentation highlighted in red with the estimated radius in μm listed above each observation of the bubble. While the first detection of the bubble by the algorithm is at the blue star, the bubble can be distinguished in earlier frames, but the algorithm can nevertheless not detect nucleation directly. Recorded with the 10x objective listed in Table III.1.

The result of the background subtraction, image segmentation, and object-tracking algorithms described above is the measurement of the size, shape, position, speed, etc. of a bubble over several frames. Given a bubble whose observations are superimposed in the frame shown in the top of Figure IV.5, background subtraction and thresholding can highlight the bubbles as shown in the middle panel, and segmentation and tracking identify the bubble (red highlights) and estimate properties like its radius (recorded in μm above each observation of the bubble). While false positives from fluctuations in the inner stream and other sources of noise must be filtered out based on position, speed, orientation, shape, and growth of the object, we can then estimate the number of true bubbles observed in an experiment. As shown in the Figure, even a blurry image can be processed to track a reasonable bubble size down to a radius of about 2 μm. Given that the human eye can still

detect the bubble four frames (corresponding to about 70 μs at 60,000 fps) before the first detection by the image processing (marked by a blue star in the Figure), the image processing could be fine-tuned further to capture the bubble at a smaller size. Nevertheless, image processing will never detect bubble nucleation, which occurs on the scale of less than 10 nm (see critical bubble volume predicted by the string method in Figure VI.4).

Instead of fine-tuning the image processing, we used a theoretical model to "see" smaller. In Chapter V, we discuss how we fit a model of bubble growth to the bubble radius measured by the image-processing techniques discussed in this Chapter. By extrapolating its predicted growth dynamics back to the critical radius, we estimated the point along the observation capillary at which the bubble nucleated.

A Comment on the Importance of Efficient Algorithms

The algorithm to performing the image-processing tasks described in this Chapter was originally developed in Python for its simplicity. While steps like object tracking were computationally cheap enough to continue running in Python, the calculation of the median to estimate the background and the loading and processing of images was prohibitively slow. Analyzing a single 6 GB video, of which over 100 were collected in a typical experiment, would take several minutes to an hour. The largest bottleneck was loading an entire image (an array of 10^4–10^6 pixel values) into memory and processing it. Relieving this burden required distributing the image across multiple threads so each would only need to load a fraction of the full image. Additionally, because Python is an interpreted programming language, every thing is compiled at run-time, slowing down computations.

To speed up this algorithm, Isaac Swanlund rewrote the computation of the background and image segmentation steps as a parallel-computed, multithreaded algorithm in C++ using optimized image-processing algorithms from the OpenCV computer vision library [1]. This backend was embedded in a Python frontend so it can be `pip` installed as a Python package (`CvVidProc` [2]; currently only available for Linux and Mac). By distributing computations among multiple threads (usually 8–12) and performing computations with optimized algorithms in C++, a compiled language, the analysis time was reduced by a factor of up to 100x. A 6 GB can now be analyzed in under a minute, and typically in under 5 seconds. The analysis performed to generate the plots in Chapters VI and VII would have taken at least hundreds of additional hours without this speed-up.

IV.2 Recommendations for Further Improvements

The image-processing pipeline used in the present work, while sufficient, can be improved by implementing a few additional steps. First, while images are recorded with 12-bit depth (2^{12} = 4096 pixel brightness values), the image processing was performed on an 8-bit compressed image (2^8 = 256 pixel brightness values) for simplicity and efficiency. This compression reduces both the sensitivity and the contrast in the images, limiting the smallest detectable bubble size and resulting in less precise boundaries during segmentation. Data storage and analysis of 12-bit images will be more memory-intensive, however. Second, threshold values are currently estimated manually by identifying the value for which the number of true bubbles detected is least sensitive to perturbations in the threshold value. This algorithm could be automated and made more quantitative to speed up the analysis and make it more robust. Third, true bubbles are currently distinguished from other detected features (contaminant particles, fluctuations of the inner stream, bubbles and particles in the outer stream, etc.) based on a heuristic set of requirements, including nearness to the center line, growth in size over time, range of velocities along the flow axis, maximum width, minimum number of frames observed, disappears from view upon reaching downstream side of field of view, and maximum aspect ratio. A machine-learning algorithm could identify more salient features and more precise parameter ranges for this classification if someone manually classified a few hundred objects as bubbles or not. Fourth, given the bubbles tracked with the current algorithm, the range of bubble growth could be extended by using that information to estimate the location of the bubble in frames prior to the first detection. In just the pixels near these locations, a more sensitive threshold and less erosive image segmentation could be applied to have a higher sensitivity without picking up noise. Finally, the object-tracking algorithm currently fails to detect merging and splitting events of bubbles. More complex algorithms like MHT-X [8] could be implemented to track objects properly through merging and splitting events, which would expand our ability to make quantitative measurements of ripening beyond our currently qualitative estimates (Section VIII.3).

References

1. Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools for the Professional Programmer* **25,** 120–123 (2000).
2. Swanlund, I. *CvVidProc: OpenCV Video Processing for Object-tracking: v1.0.0* 2022. https://github.com/UkoeHB/CvVidProc (2022).

3. Ylitalo, A. S. *bubbletracking_koe: v0.0.1* 2022. https://github.com/andylitalo/bubbletracking_koe (2022).

4. Liu, W., Cai, Y., Zhang, M., Li, H. & Gu, H. *Scene background estimation based on temporal median filter with Gaussian filtering* in *23rd International Conference on Pattern Recognition (ICPR)* (IEEE, Cancun, Mexico, 2016), 132–136. ISBN: 9781509048472.

5. Van der Walt, S. *et al.* scikit-image: image processing in Python. *PeerJ* **2,** e453. ISSN: 2167-8359. https://peerj.com/articles/453 (June 2014).

6. Murray, S. *Real-Time Multiple Object Tracking A Study on the Importance of Speed* PhD thesis (National Institute of Informatics, 2017). arXiv: 1709.03572. https://arxiv.org/pdf/1709.03572.pdf.

7. Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2,** 83–97. ISSN: 00281441. https://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109 (Mar. 1955).

8. Zvejnieks, P. *et al.* MHT-X: offline multiple hypothesis tracking with algorithm X. *Experiments in Fluids* **63,** 1–18. ISSN: 14321114. arXiv: 2101.05202. https://doi.org/10.1007/s00348-022-03399-5 (2022).