Lab on a Chip

TECHNICAL INNOVATION

View Article Online
View Journal | View Issue

Cite this: *Lab Chip*, 2017, **17**, 256

Image processing and analysis system for development and use of free flow electrophoresis chips†

Sven Kochmann and Sergey N. Krylov*

Received 7th November 2016,
Accepted 2nd December 2016

DOI: 10.1039/c6lc01381c

www.rsc.org/loc

We present an image processing and analysis system to facilitate detailed performance analysis of free flow electrophoresis (FFE) chips. It consists of a cost-effective self-built imaging setup and a comprehensive customizable software suite. Both components were designed modularly to be accessible, adaptable, versatile, and automatable. The system provides tools for i) automated identification of chip features (*e.g.* separation zone and flow markers), ii) extraction and analysis of stream trajectories, and iii) evaluation of flow profiles and separation quality (*e.g.* determination of resolution). Equipped with these tools, the presented image processing and analysis system will enable faster development of FFE chips and applications. It will also serve as a robust detector for fluorescence-based analytical applications of FFE.

Introduction

Free-flow electrophoresis (FFE) is a technique for continuous electrophoretic separation of multiple species in a device (FFE chip) that combines a hydrodynamic flow and an electric field.[1] FFE is promising for applications in flow synthesis and bioanalysis.[2] Therefore, a number of research groups, including ourselves, have been developing FFE-based methods and FFE chips of a variety of designs, made with different fabrication methods of different materials.[3–14] FFE chips of new designs have to be tested to evaluate their performance (*e.g.* flow profile within the entire separation zone and separation power) in a simple, flexible, fast, quantitative, and reproducible fashion. These requirements can only be met with real-time optical imaging of the entire chip.

A commonly-used way of optical imaging in FFE is taking pictures with a digital camera.[15–17] While being simple, flexible, and fast, this approach cannot serve for quantitative and reproducible evaluation. In this approach, many parameters (*e.g.* illumination, camera angle, focus, and distance from chip) can significantly differ between measurements. This variability makes it impossible to perform quantitative analysis of a time series of images by software, such as *ImageJ*.[18,19] Without quantitative comparison of images, any conclusion drawn from such data is subjective, and, thus, the chip optimization process cannot be carried out in a technically sound fashion.

Some groups use commercially available, camera-equipped microscopes for imaging FFE separation.[20–22] As a result of a fixed geometry, this approach eliminates the issue of non-comparability between images. The groups of Nagl and Belder have employed the microscopy-based imaging approach in their recent developments.[23–29] This way of optical imaging is, however, not flexible. Scientific microscopes are relatively bulky instruments, which cannot easily be repositioned or adapted to an FFE platform under investigation. Indeed, when evaluated with a microscope, an FFE platform is more likely to be adapted to the geometry of the microscope, which immensely limits microscope imaging as tool for FFE development.

Besides the hardware, the software for collecting, storing, processing, evaluating, and presenting data acquired from experiments play an important role in FFE development. A suitable software method should be able to detect and process the separation zone as well as the analyte trajectories. The nature of FFE experiments requires such a method to be highly automated as well as readily adaptable. Naturally, the design and the features of FFE chips may be a subject to change during the development. Processes taking place in FFE might be in the range of a few minutes to several hours, thus, the evaluation of stream trajectories and velocities may require several tens of images to be processed. While examining the literature, we found that none of the published works discuss software methods – not even superficially. Mainly, this is due to the target audience of most FFE publications being presumed FFE users rather than developers.

In this contribution, we present a flexible and cost-effective image processing and analysis system for FFE chip development and use. The system consists of a self-built

Department of Chemistry and Centre for Research on Biomolecular Interactions, York University, Toronto, Ontario M3J 1P3, Canada. E-mail: skrylov@yorku.ca; Tel: +1 416 736 2100 ext 22345

† Electronic supplementary information (ESI) available: Experimental details, hardware setup, software, and other results. See DOI: 10.1039/c6lc01381c

portable imaging setup and a customizable software suite. The hardware component was designed to provide an intermediate approach between the highly flexible but non-robust imaging using compact cameras and the robust but non-flexible imaging using microscopes. The software suite assists the user in setting up and aligning the system and the chip as well as recording, storing, and processing images and extracting performance parameters from these images. The software is made available to the reader.

Materials and methods

Chemicals

All solutions were prepared using analytical grade reagents. 4-(2-Hydroxyethyl)piperazine-1-ethanesulfonic acid sodium salt (HEPES), rhodamine 6G hydrochloride (rhodamine), fluorescein sodium salt (fluorescein), 5(6)-carboxyfluorescein (carboxyfluorescein), sodium hydroxide, and immersion oil (UV-transparent, fluorescent-free) were purchased from Sigma Aldrich (Oakville, ON, Canada). Deionized water (18.2 MΩ cm^{-1}) was used for preparation of all solutions.

Chip fabrication

An FFE chip was made out of poly(methyl methacrylate) (PMMA) with a refractive index of $n_D^{20°C}$ = 1.4918.[30] FFE chip model was designed using Solid Edge (Siemens PLM, Plano, TX) software. An MDX-540 milling machine (Roland DGA, Irvine, CA) was used to fabricate the device. A detailed description of the general procedure can be found elsewhere.[31] The model files of the chip used in this study and a short overview of the fabrication process can be found in the ESI.† A scheme and photo of the chip can be found in Fig. 1. Immersion oil ($n_D^{20°C}$ = 1.517) was applied to the top of the chip (the side with the markings) to reduce scattering and refraction. The chip has 5 sample inlets and outlets. We did not optimize the chip or chip fabrication for this study.

COMSOL simulations

We used COMSOL Multiphysics software version 5.1 to simulate and compare the experimental flow profile and velocities with computational ones. The 3D results are illustrated in a 2D format for ease of visualizing the flow profiles. A steady state system was designed using the *Laminar Flow* module. Conditions and settings of the COMSOL simulations can be found in the ESI.†

Portable imaging setup

The image chamber was designed for photoluminescence imaging. It consists of a camera objective and electronics (extracted from a QuickCam® Orbit™ MP, Logitech, Newark, CA, USA), a micropipette tip box acting as a case, optical filters (see ESI† for spectra), and a 470 nm LED (M470L3, Thorlabs, Newton, NJ, USA) as a source of illumination. A standard laptop computer (TravelMate 6592G from Acer) running Windows XP was used for data acquisition. A sketch and a photo of the imaging setup are shown in Fig. 1 (see ESI† for more details).

Software suite

The software suite includes a function library and some ready-to-use programs for setting up and aligning the system and the chip as well as recording, storing, and processing images and extracting performance parameters from these images. The programs were written in *Python 2.7* according to *Eric Raymond's 17 Unix Rules* (in particular, *Rules of Simplicity and Modularity*).[32] Main modules used are *Numpy*, *Mathplotlib*, *Scipy*, and *OpenCV* (Open Source Computer Vision Library). The source code as well as full documentation can be found in the ESI.† The software suite is compatible with every camera supported by *OpenCV* including a variety of web-cameras (see documentation of *OpenCV*).[33]

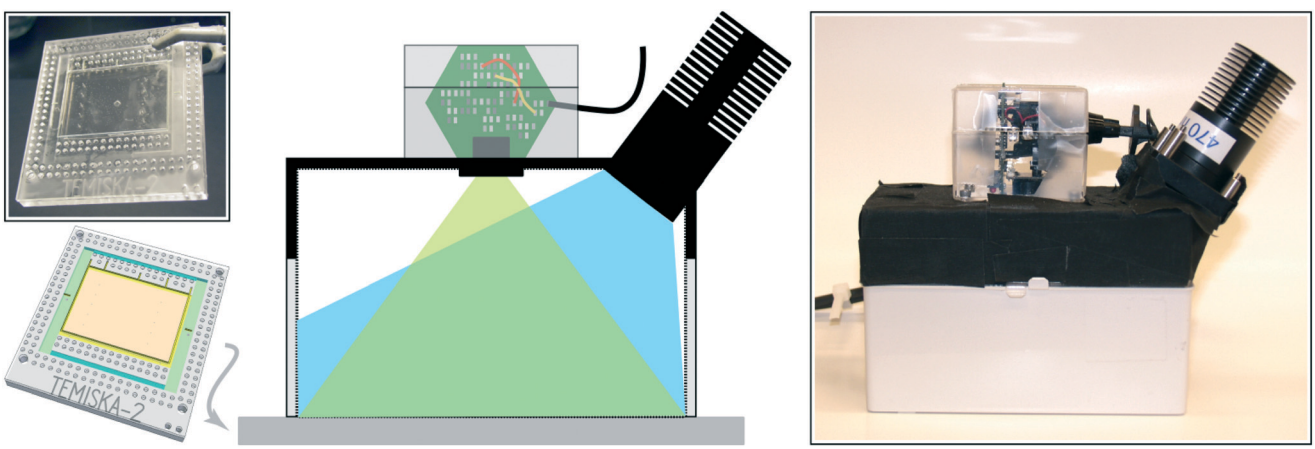

Fig. 1 Scheme and photo of the FFE chip and customized fluorescent imaging system.

FFE setup and experimental details

We used HEPES (10 mM, pH 7.6) as the background electrolyte (BGE) and solutions of rhodamine, fluorescein, and carboxyfluorescein (500 μM each) in HEPES as *tracer dyes*. Flow rates of the BGE and the sample solution were in the range of 3–4.5 mL min^{-1} and 2–5 μL min^{-1}, respectively. All sample inlets were closed except for the one used in the respective study. Sample outlets were opened only for flow-profile measurements. More details can be found in the ESI.†

Results and discussion

Process of FFE development

Briefly, the overall process of FFE development can be split into four, subsequent steps: i) chip design, ii) computational evaluation (*e.g.* with COMSOL), iii) chip fabrication, and iv) experimental evaluation. In the absence of appropriate tools, the experimental evaluation is usually neglected or limited to minimum thus making computational methods the main way of chip evaluation. However, computational methods cannot account for practical imperfections introduced in the fabrication step. Furthermore, there is no confirmation of agreement of a computational model and real-device performance without a respective experimental evaluation. Therefore, a strategy for experimental evaluation is required and presented in this study. Concisely, FFE developers can use fluorescent tracer dyes to 'interrogate' different zones of a chip under various conditions or to model separation of real-life analytes with similar electrophoretic mobilities. The available diversity of fluorescent dyes allows choosing tracers for a likewise diverse set of situations. The presented image processing and analysis system facilitates fluorescence measurements and their processing.

Versatility and adaptability

As described in the introduction, FFE features, design, and methods are subject to change during development. Since our image processing and analysis system targets FFE developers, it must be versatile and adaptable. We achieved this versatility and adaptability by modularization of hardware and software. The components of the hardware (camera, LED, filters, case) can be replaced individually and rearranged to adapt to new geometries or new analytes. Similarly, the core of the software suite consists of a function library providing the base functionalities such as *data-file-operations*, *separation-zone-detection*, or *trajectory-finding*. These can be modified, extended, and assembled to create new programs to suit new situations. Our software suite also includes complete user programs based on the library, which provide a user interface to the aforementioned functionalities. Naturally, these programs can also be modified and extended.

Setup and alignment of the system and chip

Setup of an acquisition system is a critical task since wrong settings can affect data's consistency and reproducibility. An acquisition system must aid the user in setting and aligning the camera, and ensure that correct settings are employed (*e.g.* brightness, contrast, saturation, gain, exposure). While the camera driver should save and preserve all camera settings by itself, the reliability of the driver is poor: we have noticed that settings are often saved on the basis of the universal serial bus (USB) port that the camera is connected to, instead of the camera device itself, *i.e.* each USB port holds its own instance of the camera device including its own settings. Moreover, sometimes the camera settings may be reset randomly, either by the driver or the firmware – when the camera is not in use. Also, it might be beneficial to save camera settings as a transferable file to enable their use on other machines or with other cameras. To address these issues we created a user program for configuring the camera (**setupcamera.py**), which utilizes a simple graphical user interface (GUI) to facilitate this step. Subsequently, camera settings are automatically loaded and invoked by the acquisition and alignment programs (using the corresponding library functions). This assures that the camera is set to the state desired before it is used (*e.g.* for recording) to ensure consistency and reproducibility.

Another important step in the setup process is the alignment of the chip's separation zone. The separation zone should match the horizontal and vertical directions of the imaging view to facilitate its detection during the evaluation process. The program (**alignchip.py**) aids the user in performing this task by providing direct visual feedback on the state of alignment while the user rotates the chip or the system. Briefly, the program detects the separation zone and encloses it within a rectangle. It also draws a reference rectangle which represents the target (optimal) orientation of the separation zone in the imaging field of view. The overlap between the areas of the two rectangles is calculated and presented. If the alignment is perfect, both rectangles are equal and the overlap is 100%. The rectangles are rendered on a black canvas with using red (detected separation zone of the chip) and green (target orientation) filling colour, respectively. In the overlapping area, the resulting colour is yellow. The overlap is calculated by counting the yellow pixels and dividing their number by the number of green pixels. Fig. 2 shows the scheme for alignment.

Finally, the user can set basic measurement and output parameters (*e.g.* output directory, number of frames per second to record, *etc.*) by another program (**setupmeasurement.py**). Likewise, these details are saved and used by the subsequent acquisition and evaluation programs.

Acquisition

The processes taking place in FFE are in the range of minutes to hours. Thus, the rate of image acquisition is only required to be with an interval precision of seconds. The low time precision reduces the requirements for (and cost of) the detection hardware, in particular the camera. A common, consumer grade webcam is sufficient to perform this task.

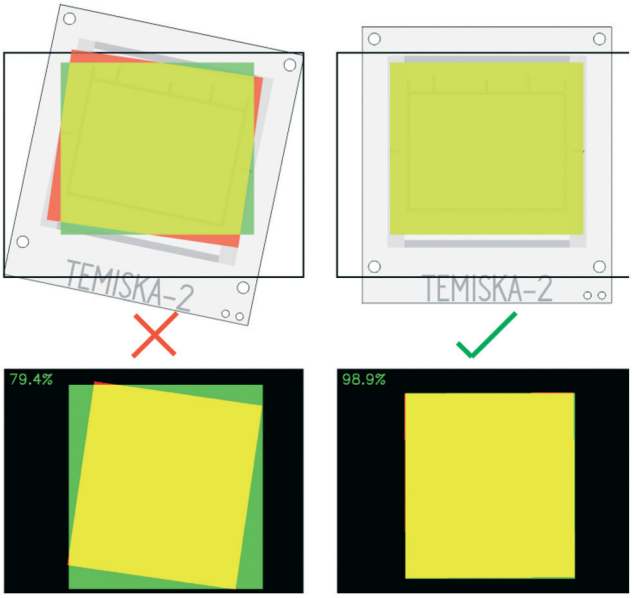

Fig. 2 Alignment of the FFE chip. The top schemes illustrate the alignment of the chip with regards to the camera view (black rectangle). The bottom pictures show the alignment view presented to the user by **alignchip.py**.

However, webcams might lack sensitivity as they are designed for use in moderately-to well-lit environments. As a result, they may have low signal to noise ratios in fast fluorescence measurements. In our application the signal to noise ratio can be improved by integrating images from the camera over a relatively long time of a few seconds.

We implemented the task of acquisition in a user program (**acquire.py**), which records time-dependent snapshots of the separation zone. Its user interface consists only of a live picture of the recording with an overlay displaying timestamp and some user information. Starting the program starts a new measurement while closing the program stops the measurement. We kept the user interface and functionality minimalistic to reduce errors (both from human and computer) during acquisition. For the same reason, no *online* evaluation (*e.g.* detection of the separation-zone) was implemented.

Storing information and data

A fundamental question of any acquisition system is *how to store information?* In general, information has to be encoded into a file format in order to be stored. Many approaches, in particular commercially available solutions, use complex and proprietary file formats. The disadvantage of such formats is that the files cannot be opened without proprietary software. Therefore, for our imaging system we decided to use common standardized formats.

Properly answering the question of *how to store information* requires understanding of what information to store and for whom to store. There are two types of recipients of information acquired by our imaging system: humans and machines (computer programs). File format(s) should thus be readable by both *humans* and *machines*. Our FFE imaging system produces a sequence of time-lapse images, which are saved both as a video file (mainly for *humans*) and single image files (mainly for *machines*).

Video file for human perception

A sequence of images can be saved as a video, which can be viewed by humans and, in theory, can also be processed by programs (*via* extraction of individual frames). Videos can be compressed or uncompressed. While we have attempted to save the imaging data as uncompressed videos, we quickly realized that the resultant file sizes were too large (up to 1 GB min^{-1}) for practical data storage and sharing. Therefore, videos have to be compressed. However, common video compression methods (implemented as *codecs*) are made for *humans* only. They reduce file size drastically (down to 1 MB min^{-1}) by removing information imperceptible to humans, at the expense of irretrievably losing information that is crucial for machine reading.[34] Since there is no common format that can balance needs of *humans* and *machines* we decided to use different formats for *humans* and *machines*.

The video format of choice for *humans* was a widely distributed standard, MPEG-4.[35] MPEG-4 files can be viewed natively on almost every platform including a variety of handheld devices. To further improve the convenience of this output format, the raw image frames received from the camera are automatically modified by **acquire.py** in two ways. First, the overlay showed to the user during acquisition (see *Acquisition*) is added to each video frame. Second, the playback speed of the resulting video file is increased, resulting in a quick-motion video. Thus, a process recorded over several minutes can be viewed in a few seconds. Note that these two modifications only apply to the output video file.

Single image files for machine evaluation

The best way to store individual images is by using an image file format such as *Portable Network Graphics* (PNG).[36,37] PNG files can be viewed natively on almost every platform. Furthermore, PNG images are stored using lossless compression (in contrast to *e.g.* JPEG files) efficiently utilizing disk space and facilitating portability, backup (long-term storage), exchange, and publication of data.

In addition to storing raw images, it is important to save setup parameters such as experiment name or colour channels recorded. Furthermore, the stored data has to be somehow linked to a corresponding image file. This includes, for instance, inlet positions, physical dimensions, timestamp, position of detected separation zone, and trajectories. This data can be stored as separate text file(s) complementing the corresponding image file. However, it is much more practical to embed acquisition, processing, and evaluation data directly as metadata into the PNG file; accordingly, we have done exactly that.

Adding metadata to PNG images

A PNG file itself consists of several chunks represented by a four-letter code (*e.g.* IHDR, IDAT, tEXt). The code labels are case sensitive. The case of the first letter, for instance, marks the corresponding chunk as *critical* or *ancillary* chunks; more information can be found elsewhere.[36,37] *Critical* chunks are mandatory and hold the information for rendering an image. *Ancillary* chunks can hold any additional non-critical information which can be ignored by the program when encountered but not understood. This allows the addition of metadata to a PNG file without breaking it.

In principle, an application can use one of the predefined, public chunks (*e.g.* tEXt) to store data in a PNG file as it is done for the Exchangeable Image File Format (EXIF; used for saving digital camera settings). Some PNG viewers and editors can directly display the content of such standard chunks, which eliminates the need for special software. The disadvantage of storing data in such universal chunks is that not only may editors and viewers misinterpret them, but they can also alter or overwrite this data without user involvement. This renders this approach to information storage unreliable.

Hence, we decided to use a twin-track approach in order to achieve both reliability and readability. First, a new, private chunk labelled **dfFe** (**d**ata of **f**ree **f**low **e**lectrophoresis) was introduced for reliably storing data. Second, the data stored in this chunk is mapped to the public tEXt-chunks for easy readability. Chunks are generated by serialization of a *Python* dictionary (an associative array), which contains information generated and put together during the various stages of experimentation and evaluation by the corresponding programs.

For convenience, we developed a command-line tool **pngdata.py**, which can read and modify the **dfFe**-chunk as well as display all chunks present in any PNG file. To distinguish regular PNG files from the ones containing FFE data we use a double extension ".FFE.PNG" (instead of ".PNG") for the latter.

Detection of separation zone

The imaging system is designed for photoluminescence imaging. Hence, the imaging chamber is dark and only LED-illuminated luminophores can be detected. Obviously, the chip material (PMMA) does not emit any luminescence. Thus, the imaging system does not 'see' the chip and, therefore, cannot resolve spatial coordinates of signals on the chip. Luminescent features, which mark the separation zone, were added to the chip to overcome this problem. With such luminescent markers, it is possible to visualize the separation zone and determine the positions of luminophores relative to zone boundaries. There are limitless ways of designing such markers. For instance, one could use markers for the edges of the separation zone, similar to those used in alignment of 2D bar codes (QR codes).[38] While these patterns have proven to be useful for *machines*, they cannot be read by *humans*. However, it is essential for *humans* to know where important features of the chip are for proper evaluation and troubleshooting. Therefore, markers are needed, which can be evaluated and recognized by both machines and humans.

We came up with a very simple and descriptive design consisting of a rectangle that surrounds the separation zone and contains small features on the top representing the start, middle, and end of the zone as well as the x-positions of inlets and outlet (Fig. 3). Due to these features, the whole marking scheme becomes asymmetric and allows simple determination of chip orientation. In practice, we carved our marking scheme (1 mm deep channels) in the top of the chip during fabrication with our milling machine and coloured the valleys afterwards with a common yellow highlighter (0.7 mm wide). Common text markers utilize luminophores (*e.g.* fluorescein), which are visible during imaging. Conveniently, the milling machine allows the perfect alignment and positioning of the markings on the chip. Alternatively, one can think of printing a marking pattern using luminescent ink on an adhesive foil, which is subsequently deposited on the chip. While this seems to be convenient at first glance, exact alignment and positioning of such a foil needs more effort than milling.

In order to identify a feature, such as the markings for the separation zone, on an image, one has to consider all of the properties and relationships of such a feature. In our case, the area of the separation zone i) is the largest feature on the image and takes up a certain fraction of the whole image area, and ii) consist of two nested rectangles (inner and outer boundary), which have a certain ratio of areas (0.80–0.90). For finding the separation zone, all boundaries have to be detected (done with functions provided by *OpenCV*) and evaluated with regards to the properties and relationships described in i) and ii). When found, the four corners of each of the inner and outer boundaries are saved for further evaluation (*e.g.* extraction of the separation zone for trajectory finding). Subsequent evaluation programs just use these coordinates to identify the separation zone.

Besides marking the separation zone, it may be useful for certain applications, such as non-orthogonal FFE,[39] to mark the BGE flow orientation. Therefore, we added two small bars to the left and right sides of the separation zone. Regardless of where they are actually located, these bars are always of the same size and mirroring each other relative to the centre point of the separation zone. These features of the flow markers are sufficient for finding them and saving their coordinates for further use. Note that flow markers are not needed for standard, orthogonal FFE, in which BGE is assumed to flow from left to right on the x-axis of the separation zone. If the flow are markers are present, however, the extracted separation zone is rotated to match its x-axis with the direction of the hydrodynamic flow. An example image with all detected patterns is depicted in Fig. 3.

The whole process of feature detection is performed by **findfeatures.py**, a console program evaluating a single image. It provides various parameters to control the above-described

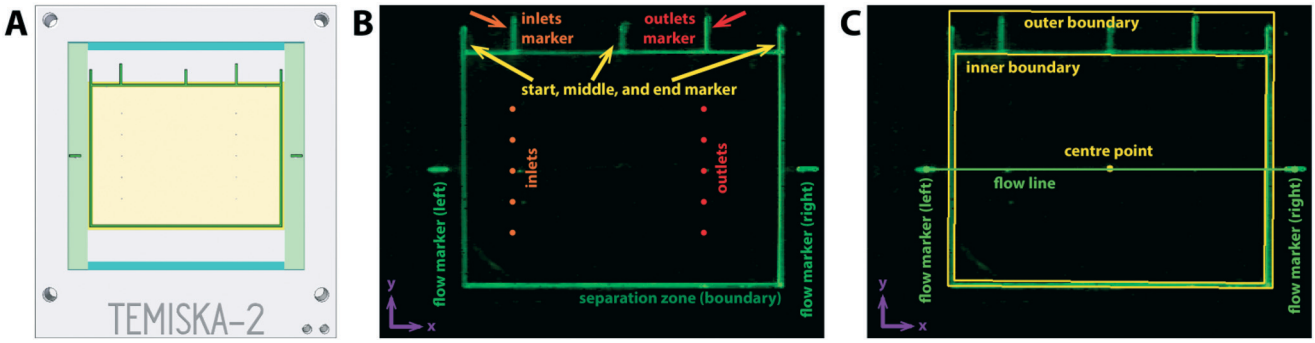

Fig. 3 Detection of the separation zone and flow markers with the help of luminescent markers. A) Scheme of the markings on the model of the chip. B) Real image taken from the imaging system showing the luminescent markings and its features. C) Image of B) fully evaluated by **findfeatures.py**.

feature recognition methods and can be included in batch processes for further and consistent automated evaluation of a time series of images. The determined feature information is saved into data chunks of the *FFE.PNG* file. It should be emphasized that during image processing and evaluation, the actual image is never altered. Thus, both the *raw data* and the *evaluation data* is always present in the same file.

Finding trajectories

The main features to be detected and evaluated in an FFE image are the trajectories of analyte streams inside the separation zone. Stream trajectories provide critical information on the separation process as well as the general behaviour of the chip (*e.g.* the flow profile). Trajectory detection is not trivial, and a suitable detection strategy has to be established.

First, a suitable starting point is required from which a given trajectory can be traced. Such a starting point must be exclusively within a respective trajectory, *i.e.* not a part of another trajectory. Furthermore, finding and identifying a starting point should be relatively fast, reliable, and consistent. The only characteristic points of a trajectory are the tail points. Each trajectory has two tail points: the point from which it starts (the inlet position) and the end point it flows to. The points at and near the inlet are shared by every trajectory, since all the analytes are introduced through a single inlet. This leaves the end points as the only suitable starting points for trajectory tracing. Theoretically, these points can be anywhere within the separation zone, making their detection and identification difficult. For convention, the hydrodynamic flow in the chips is always directed from left to right (see the previous section) and the end points of trajectories must be on the right side of the image. For finding them algorithmically, we use a technique we refer to as *light casting*. On each line (y-axis) of the image, we cast 'light' from the right side to the left side (x-axis). This 'light' passes all zero-pixels but stops at the first non-zero pixel. For each line of the image we record the position of this first 'obstacle' and plot the positions as a function of y. The minima of the resulting curve are the end points of the trajectories (see Fig. 4A for a vivid example).

To exclude false positive minima arising from noise or artefacts, we introduce two exclusion criteria. First, we define a maximum penetration depth for the 'light' to exclude all points to the left of the inlet. Secondly, we define a minimum width for the stream at the end point to exclude single pixel noise (see example in Fig. 4A). Note that *light casting* is a very simple and flexible method, which does not require that all trajectories end at the same x-plane.

Starting from the end points found, the rest of the trajectories can be traced. We implemented two different tracing algorithms, namely Dijkstra's algorithm and our own gradient-based method.

In general, a trajectory is a network of connected pixels, which can be described as a mathematical graph (graph theory). Two nodes of this graph are already known, namely the origin point (inlet position) and the end point (determined in the previous paragraph). All one has to do is to find a path between these two nodes using a suitable algorithm. We decided to implement and evaluate Dijkstra's algorithm, which has proven to be robust in a number of applications such as route planning.[40–42] This algorithm is designed to find the most cost-effective path between two nodes. For route planning, the costs are usually calculated as a function of the route distance or time required to travel a route. For trajectory finding, we define the cost *via* pixel intensity density: the higher the intensities of a pixel and its neighbours are, the less expensive it is to travel to this pixel. The cost for zero-intensity pixels is set to be infinite, *i.e.* these pixels are not passable. This cost-definition ensures that a path, *most likely*, will go along the spine of a trajectory instead of its outer border (see Fig. 4B).

Dijkstra's algorithm is robust, *i.e.* small changes in input data produce similar trajectories, as long as both nodes, *i.e.* origin and end point, are connected by an uninterrupted set of non-zero pixels. In practice, however, disconnections in a path might occur due to noise, a bubble, or turbulence. In such cases, Dijkstra's algorithm will fail to find a path and, subsequently, no trajectory will be recognized. Furthermore, the algorithm is time-consuming as it requires evaluation of practically every pixel in a trajectory. For example, finding a path in the image in Fig. 4B took over 1 min.

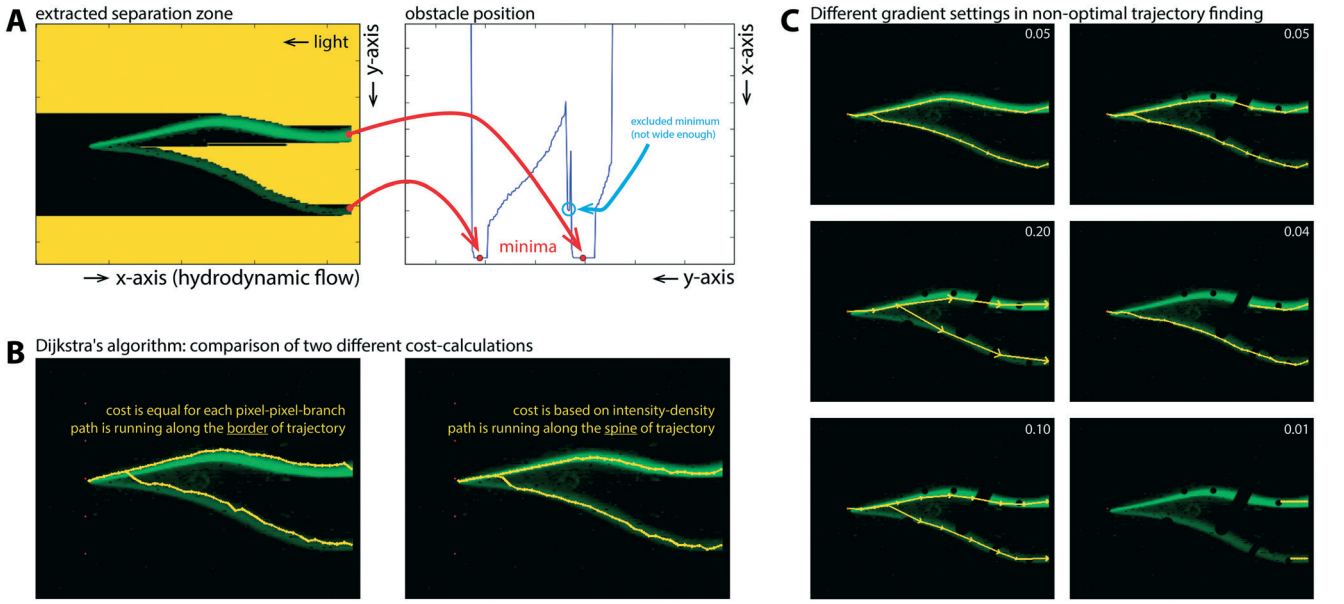

Fig. 4 Strategies for finding trajectories on an image. A) Light casting: light is casted from right to left on the image (left picture). The obstacle position on each line is plotted as function of *y* (right picture). The minima of the resulting graph are the end points of the trajectories. B) Dijkstra's algorithm: these two pictures show the effect of different cost-calculation strategies on the shape of the resulting path. Equal-cost function results in a path running along the border of trajectory (left picture) while a cost function based on intensity-density results in a path running along the spine of trajectory (right picture). C) Gradient-based algorithm: the pictures show the effect of different step-sizes (here represented by a gradient-factor) on the resulting path on the original image (top-left picture, same input image as in B) and images with holes and stains (manually modified). The ability of the algorithm to cross holes in a trajectory heavily depends on a sufficient step size. However, too large step sizes result in less fitted trajectories.

In order to address these two disadvantages, we decided to design and implement our own, gradient-based algorithm. Instead of evaluating every single pixel, this algorithm *estimates* the course of the trajectory to find a path from the end point towards the inlet. Naturally, the quality of estimation defines how well the path will resemble the actual trajectory. Beginning with the end point, the algorithm takes small steps into the direction the trajectory most likely comes from. The inverse of the hydrodynamic flow is taken as the initial direction. For every subsequent step, a new direction vector is derived as the one of the previous step adjusted using the following two rules. First, the spine of the trajectory possesses the highest pixel intensity density, *i.e.* high density attracts the direction vector more than low density. Second, the trajectory originates from the inlet and the closer it is to the inlet the more likely the trajectory will directly flow towards it. Thus, the closer the estimated path comes to the inlet, the more the direction vector is attracted by it. Note that this does not mean nor require that the trajectory is continuously connected to the inlet. On the contrary, with this algorithm it is possible to backtrack trajectories even if they are discontinuous (not connected to the inlet, consist of disconnected parts, *etc.*). Moreover, the algorithm is about 1000 faster than Dijkstra's algorithm, since it only evaluates a few pixels along each step. It took us less than 0.1 s to find the trajectory in Fig. 4C (top-left picture).

Apart from the direction vector, the step size (gradient) is another critical factor in the gradient algorithm. If the gradient step is too large, the trace might easily end up outside of a flow stream, unable to return and continue. Also, larger gradients result in paths which less resemble the evaluated trajectory. If the gradient is too small, the algorithm may not be able to pass interruptions or stains in the trajectory. Examples of different gradient settings are shown in Fig. 4C. Hence, while this algorithm is fast and can deal with poor trajectory data, it is clearly not as robust as Dijkstra's algorithm. Furthermore, the resulting paths do not resemble the spine of the trajectory as much as Dijkstra's algorithm.

An important question, which rises at this point, is *which algorithm is preferable?* Our conclusion is that both algorithms have their advantages and limitations. These give them benefits over each other in different evaluation scenarios. Thus, we leave for the users to choose the best tool for every situation.

Determining the widths of trajectories

A stream is not only defined by a trajectory path but also by its width. The width is not constant along its path due to dispersion and other effects; therefore, it should be determined at each trajectory point. For this, a line orthogonal to the stream-flow is constructed at each point of the trajectory, and the intensity profile along this line is evaluated (see example in Fig. 5A). Analytically, the most useful description of stream width is the full-width-at-half-maximum (FWHM). In order calculate FWHM, the stream profile curve is first fitted by a spline function (smooth, piecewise polynomial) to account

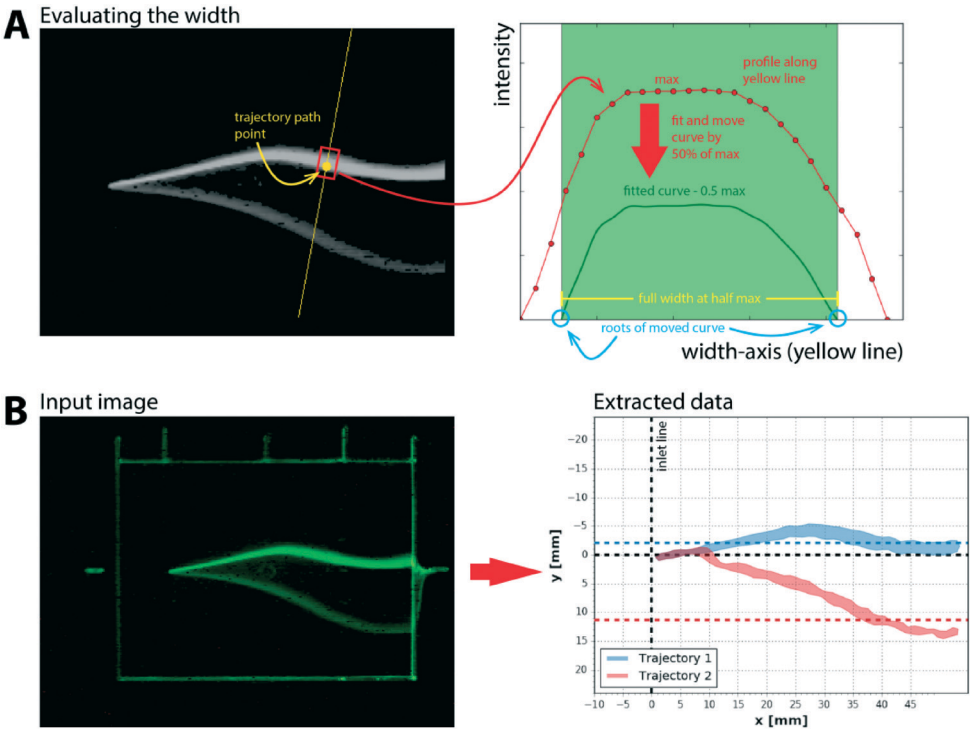

Fig. 5 A) Example of width evaluation. The profile along the yellow line in the left image is extracted and evaluated to find the FWHM at a trajectory path point. B) Comparison of the original image (input) and the extracted data.

for the fact that the intensity profile can have a variety of shapes.[43] Algorithmically, it is simple to find the FWHM of a peak or function, if one subtracts half the peak maximum value from this function, *i.e.* moves the function downward (negative intensity direction). Consequently, the distance between the roots of this moved function (on the previously constructed line) is exactly the FWHM (see Fig. 5A). After evaluating the widths, the trajectory of a stream can be represented fully without any additional information from the image (see Fig. 5B). This trajectory data subsequently can be used to evaluate separation quality (*e.g.* resolution) chip properties (*e.g.* flow profile).

Resolution

Resolution is a central parameter for evaluating the separation power of any separation technique. For common techniques, such as chromatography and capillary electrophoresis, resolution is calculated as a function of peak positions and the average of their FWHMs. For FFE, resolution is commonly calculated as function of position of the stream centre (y) and the corresponding stream width (w) at this position. Naturally, the stream centres and widths can be evaluated anywhere along the hydrodynamic flow axis (x) resulting in R being a function of x:

$$R(x) = \frac{|y_n(x) - y_m(x)|}{\frac{1}{2}(w_n(x) + w_m(x))}$$

where the indices n and m refer to the respective stream trajectory numbers. As described in the previous sections, trajectories are extracted from images with both the stream positions and the corresponding FWHMs. Due to this, the resolution between any two trajectories by design falls into one's lap. Since the trajectory path itself is discontinuous, the path itself and its widths are approximated by a cubic spline function. An example for evaluating the resolution can be found in Fig. 6. The resolution function is very unstable; it is sensitive to small differences in widths and positions which can have a huge impact on the value of R. For instance, trajectory 3 in Fig. 6 could not be fully resolved in terms of widths due to the low signal intensity of this trajectory in the input image. Thus, the resulting trajectory path has inconsistent width, which leads to an oscillating resolution function for this trajectory with any other trajectory. In contrast, the resolution function is very smooth for the resolution between trajectories 1 and 2. Of course, this can be improved by smoothing the resultant trajectory paths or resolutions, or by improving the signal intensity through adjustment of the acquisition settings. However, these results also suggest that the conventional definition of resolution might not be the best measure for (automatic) evaluation of separation in FFE. Better measures of FFE separation quality need to be developed in future works. Until they are available, resolution can provide a reasonably good, first-approximation measure of FFE separation power.

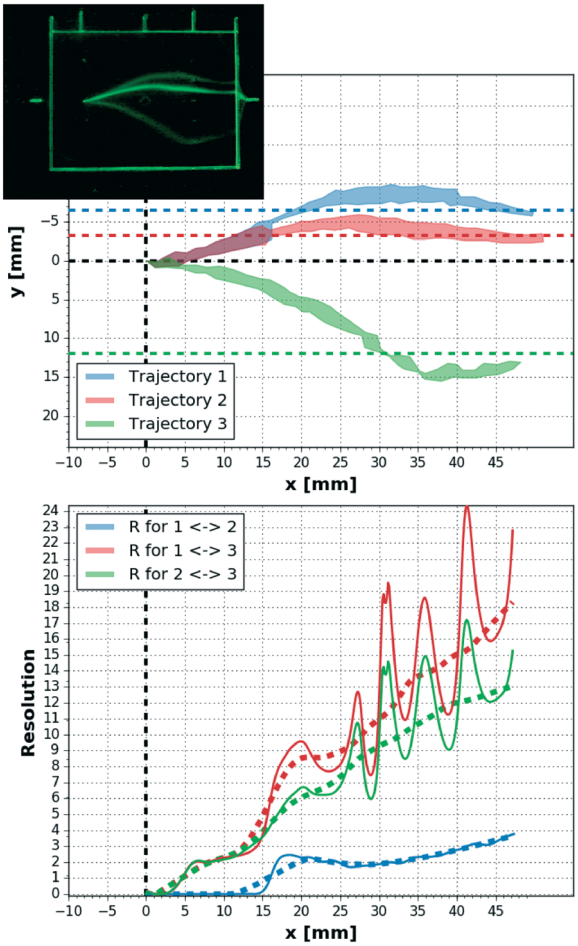

Fig. 6 Evaluating the resolution of separation (upper plot; inset shows input image) of carboxyfluorescein (trajectory 1), fluorescein (trajectory 2), and rhodamine (trajectory 3). The resolution for every combination of above trajectories is shown in the lower plot. The dashed lines are smoothed resolution functions.

Flow profile

Apart from straightforward measures such as resolution, more complex evaluations are also possible. For instance, one is able to characterize the flow profile of an FFE chip by combining the evaluation of multiple trajectories from different inlets. For this, a fluorescent tracer compound is introduced through inlets with different *y*-positions into the chip to observe the flow from the inlet to the outlets or the end of the separation zone. No electric field is applied in such measurements, as electrophoretic movement of analytes will convolute the flow profile information. Subsequently, the resulting images are evaluated to find the fronts of the trajectories. Since every image has a time stamp, it is possible to evaluate the front position as function of time. The derivation of such a function gives the velocity vector at the corresponding point. Plotting and interpolating these velocity vectors result in a flow profile, which can be used to draw important conclusions about the hydrodynamic flow in an FFE chip. Knowledge of real hydrodynamic flow profile is pivotal to the developer as this profile largely defines analyte trajectories.

An experimentally measured flow profile in our chip can be seen in Fig. 7A. In order to compare it to an idealized chip, we created a COMSOL simulation using the laminar flow module (Fig. 7B). Visual comparison of the experimental and simulated profiles leads to important conclusions. First, the velocity range and the overall profile in both plots are comparable, suggesting that the model and the physical chip are in good agreement with each other. This is even more evident when comparing the complete, *i.e.* inter/extrapolated data, of both plots. The average velocities in the *x*-direction are 0.29 ± 0.14 and 0.26 ± 0.13 mm s^{-1} for the experimental and theoretical data, respectively. The velocities in *y*-direction are likewise similar, *i.e.* 0.008 ± 0.064 and 0.002 ± 0.088 mm s^{-1}, respectively. In turn, this means that the chosen COMSOL model is valid as an FFE model. Second, the flow profile of the physical chip shows some zero-flow areas and shifted flow vectors (*e.g.* in the lower half of the chip). This is probably due to a non-homogenous cross-section in the separation zone arising, most likely, from a non-optimal assembly of the chip (as mentioned before we did not optimize the chip or its assembly for this study). Thirdly, the measurements show that the outlets do not work as intended. In the model chip, they act as sinks and attract the flow around them. However, in the real chip the flow just passes them. This is in agreement with our experimental observation that no flow was going through the outlets (probably due to air pressure from outside). These three straightforward conclusions show that our chip is non-optimal and requires improvement. They also strongly suggest that the flow profile is an important characteristic in FFE chip development. Since the presented image processing and analysis system provides a handy way for measuring these flow profiles, it will be indispensable for development of high-quality FFE chips.

Automation

Every evaluation program in our software suit is a console-program that can be controlled through command line arguments. This allows seamless integration of these programs into batch processes for evaluation of basic parameters such as trajectories as well as more advanced and complex ones such as the flow profiles of FFE chips. For instance, the evaluation of the previously discussed flow profile is done by a total of six batch programs (see ESI†). Five of them evaluate the corresponding set of flows from the inlets (**evalch1.bat**–**evalch5.bat**). These call the respective programs to find the separation zone and trajectories as well as to render these trajectories on every single image file. Another batch file (**evalflowprofile.bat**) calls these five batch files and, subsequently, the programs for extracting the position-velocity information, as well as for compiling this information into a flow profile plot. Every program in the batch process logs its actions for further review. The advantages of this batch

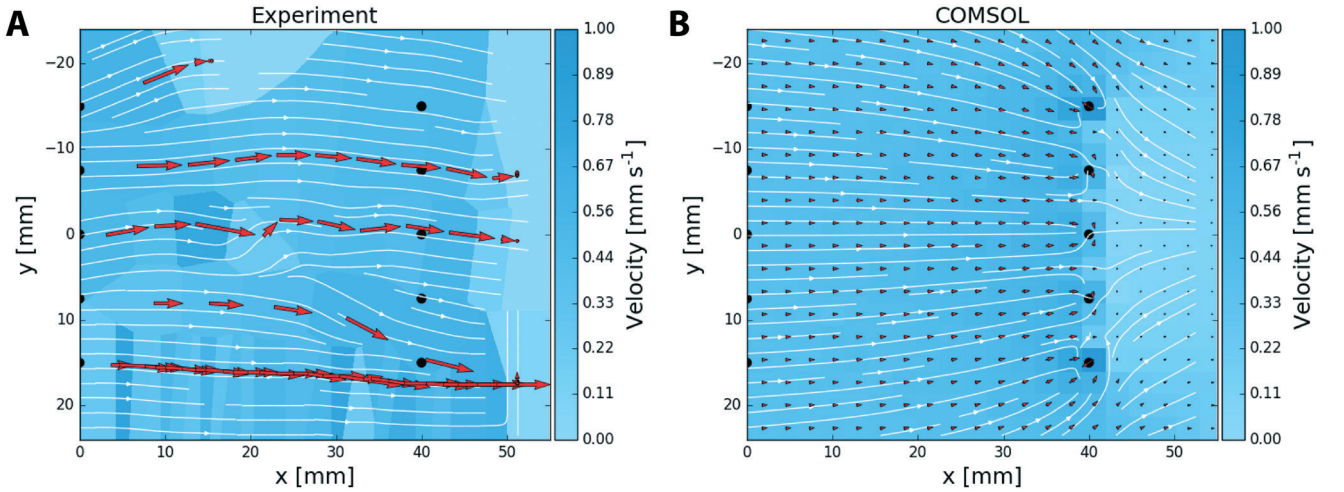

Fig. 7 Flow profiles of our FFE chip according to experiment (A) and COMSOL calculations (B). The arrows (red) depict the input data for interpolating the streamlines (white). The black half-circles at the left depict the inlets, while the black circles at 40 mm in the *x*-direction depict the outlets.

process are obvious: simple automation and customization, full documentation, and good reproducibility.

Portability

Although the presented system is aimed at FFE, the described approach may be directly applicable to some 2D techniques (*e.g.* gel electrophoresis and chip electrophoresis) and 1D techniques requiring whole-column imaging-detection[44] on an array of columns. The extent and type of the necessary modifications (*e.g.* new marking scheme, different chamber geometry, *etc.*) will depend on specifics of a target application. Such modifications are facilitated by the versatility and adaptability (see corresponding section) of our system.

Conclusions

We have developed an image processing and analysis system for comprehensive evaluation of FFE chips. It consists of a cost-effective self-built imaging setup and a software suite. Both components have demonstrated to be extremely versatile and powerful in determining important characteristics of FFE chips such as separation power and flow profiles. Furthermore, the modular design allows simple automation while remaining very flexible and adaptable. The presented modules – software and hardware – can be simply substituted and extended. For instance, future versions of the presented library and programs might implement advanced and better algorithms for trajectory finding or the like. Conclusively, we foresee numerous scenarios and applications where the presented image processing and analysis system can and will facilitate FFE development.

Acknowledgements

This work was supported by a Strategic Projects Grant 463455 from NSERC Canada. The authors thank Mirzo Kanoatov and Victor Galievsky for reading the manuscript and providing valuable suggestions. The first author thanks his former math teacher, Michael Holzapfel, for introducing him to graph theory and path-finding algorithms.

Notes and references

1 D. E. Raymond, A. Manz and H. M. Widmer, *Anal. Chem.*, 1994, **66**, 2858–2865.
2 S. Jezierski, V. Tehsmer, S. Nagl and D. Belder, *Chem. Commun.*, 2013, **49**, 11644–11646.
3 P. Hoffmann, M. A. Olayioye, R. L. Moritz, G. J. Lindeman, J. E. Visvader, R. J. Simpson and B. E. Kemp, *Electrophoresis*, 2005, **26**, 1029–1037.
4 Y. Liu, D. Zhang, S. Pang, Y. Liu and Y. Shang, *J. Sep. Sci.*, 2015, **38**, 157–163.
5 M. C. Larson, *Blood Transfus.*, 2015, **13**, 342–344.
6 L. Eichacker, G. Weber, U. Sukop-Köppel and R. Wildgruber, in *Proteomic Profiling*, ed. A. Posch, Springer, New York, 2015, ch. 29, vol. 1295, pp. 415–425.
7 E. R. Castro and A. Manz, *J. Chromatogr. A*, 2015, **1382**, 66–85.
8 C. Benz, M. Boomhoff, J. Appun, C. Schneider and D. Belder, *Angew. Chem., Int. Ed.*, 2015, **54**, 2766–2770.
9 F. J. Agostino and S. N. Krylov, *TrAC, Trends Anal. Chem.*, 2015, **72**, 68–79.
10 R. Wildgruber, G. Weber, P. Wise, D. Grimm and J. Bauer, *Proteomics*, 2014, **14**, 629–636.
11 M. Geiger, N. W. Frost and M. T. Bowser, *Anal. Chem.*, 2014, **86**, 5136–5142.
12 K. Hannig, *Electrophoresis*, 1982, **3**, 235–243.
13 P. Hoffmann, H. Ji, R. L. Moritz, L. M. Connolly, D. F. Frecklington, M. J. Layton, J. S. Eddes and R. J. Simpson, *Proteomics*, 2001, **1**, 807–818.
14 P. Hoffmann, H. Wagner, G. Weber, M. Lanz, J. Caslavska and W. Thormann, *Anal. Chem.*, 1999, **71**, 1840–1850.
15 F. J. Agostino, L. T. Cherney, V. Galievsky and S. N. Krylov, *Angew. Chem., Int. Ed.*, 2013, **52**, 7256–7260.

16 Y. Liu, D. Zhang, S. Pang, Y. Liu and Y. Shang, *J. Sep. Sci.*, 2015, **38**, 157–163.
17 Y. Yang, F. Z. Kong, J. Liu, J. M. Li, X. P. Liu, G. Q. Li, J. F. Wang, H. Xiao, L. Y. Fan, C. X. Cao and S. Li, *Electrophoresis*, 2016, **37**, 1992–1997.
18 C. A. Schneider, W. S. Rasband and K. W. Eliceiri, *Nat. Methods*, 2012, **9**, 671–675.
19 J. Schindelin, C. T. Rueden, M. C. Hiner and K. W. Eliceiri, *Mol. Reprod. Dev.*, 2015, **82**, 518–529.
20 H. Ding, X. Li, X. Lv, J. Xu, X. Sun, Z. Zhang, H. Wang and Y. Deng, *Analyst*, 2012, **137**, 4482–4489.
21 M. Geiger and M. T. Bowser, *Anal. Chem.*, 2016, **88**, 2177–2187.
22 A. Takanori, K. Ryosuke, K. Masashi and I. Takanori, *Jpn. J. Appl. Phys.*, 2015, **54**, 06FN05.
23 E. Poehler, C. Herzog, M. Suendermann, S. A. Pfeiffer and S. Nagl, *Eng. Life Sci.*, 2015, **15**, 276–285.
24 E. Poehler, C. Herzog, C. Lotter, S. A. Pfeiffer, D. Aigner, T. Mayr and S. Nagl, *Analyst*, 2015, **140**, 7496–7502.
25 C. Hackl, R. Beyreiss, D. Geissler, S. Jezierski and D. Belder, *Anal. Chem.*, 2014, **86**, 3773–3779.
26 S. Jezierski, A. S. Klein, C. Benz, M. Schaefer, S. Nagl and D. Belder, *Anal. Bioanal. Chem.*, 2013, **405**, 5381–5386.
27 S. Jezierski, D. Belder and S. Nagl, *Chem. Commun.*, 2013, **49**, 904–906.
28 R. Beyreiss, D. Geißler, S. Ohla, S. Nagl, T. N. Posch and D. Belder, *Anal. Chem.*, 2013, **85**, 8150–8157.
29 S. Kohler, S. Nagl, S. Fritzsche and D. Belder, *Lab Chip*, 2012, **12**, 458–463.
30 M. J. Weber, *Handbook of Optical Materials*, Taylor & Francis, 2002.
31 F. J. Agostino, C. J. Evenhuis and S. N. Krylov, *J. Sep. Sci.*, 2011, **34**, 556–564.
32 E. S. Raymond, *The Art of UNIX Programming*, Pearson Education, 2003.
33 OpenCV Dev Team, OpenCV 2.4.13 Reference manual, http://docs.opencv.org/2.4.13/modules/refman.html.
34 W. Fischer, in *Digital Video and Audio Broadcasting Technology: A Practical Engineering Guide*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, DOI: 10.1007/978-3-642-11612-4_7, pp. 111–146.
35 D. Marpe, T. Wiegand and G. J. Sullivan, *IEEE Communications Magazine*, 2006, **44**, 134–143.
36 T. Boutell, *RFC 2083*, 1997, DOI: 10.17487/rfc2083.
37 ISO, *Computer graphics and image processing - Portable Network Graphics (PNG)*, 2004, 15948.
38 Y. Liu, J. Yang and M. J. Liu, *Recognition of QR Code with Mobile Phones*, IEEE, New York, 2008.
39 V. Okhonin, C. J. Evenhuis and S. N. Krylov, *Anal. Chem.*, 2010, **82**, 1183–1185.
40 R. K. Ahuja, K. Mehlhorn, J. Orlin and R. E. Tarjan, *J. ACM*, 1990, **37**, 213–223.
41 D. Delling, P. Sanders, D. Schultes and D. Wagner, in *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*, ed. J. Lerner, D. Wagner and K. A. Zweig, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, DOI: 10.1007/978-3-642-02094-0_7, pp. 117–139.
42 E. W. Dijkstra, *Numer. Math*, 1959, **1**, 269–271.
43 L. Schumaker, *Spline Functions: Basic Theory*, Cambridge University Press, 2007.
44 X.-Z. Wu, T. Huang, Z. Liu and J. Pawliszyn, *TrAC, Trends Anal. Chem.*, 2005, **24**, 369–382.

Lab on a Chip

TECHNICAL INNOVATION

Image processing and analysis system for the development and use of free flow electrophoresis chips – Supporting information

Sven Kochmann and Sergey N. Krylov

Table of contents

Additional files included

- **Chip model files (Solid Edge)**
- **COMSOL simulation file (Version 5.1)**
- **Python FFE-library and programs including full API documentation**

Chip fabrication

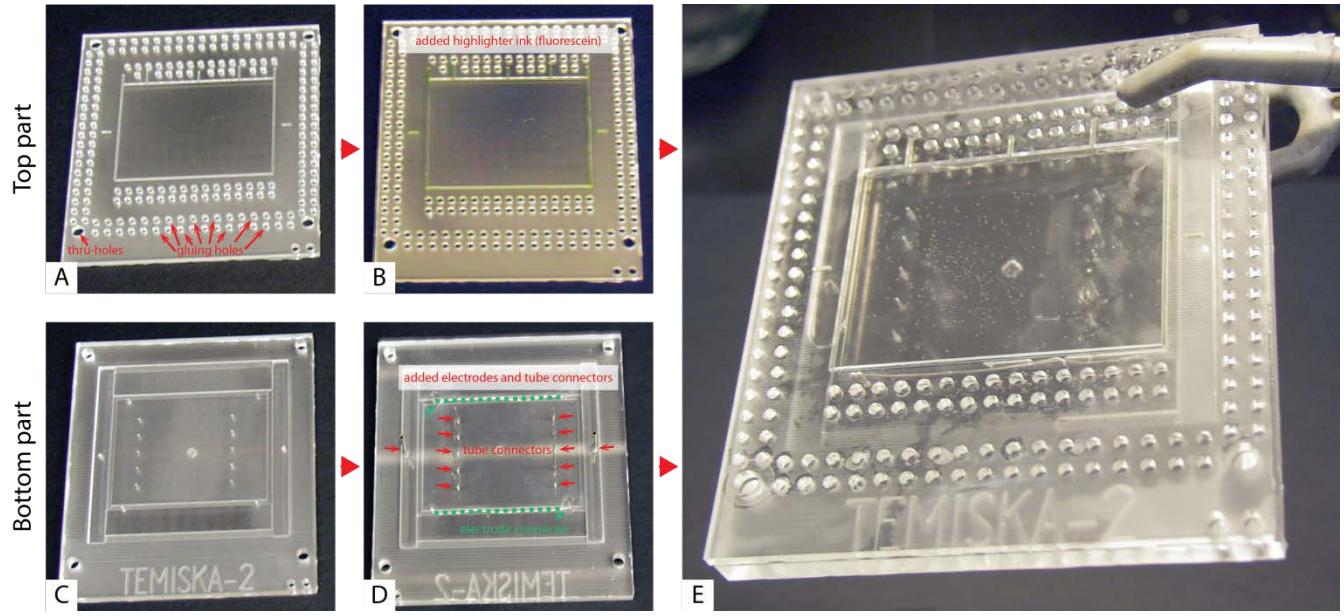

Figure S1. Assembly of the chip. Images show the top part of the chip without (A) and with (B) highlighter ink, the bottom part without (C) and with (D) added electrodes (green dashed lines) and tube connectors (red arrows) at inlets and outlets, and the final assembled chip (E). Please note that we increased the contrast of all pictures and the saturation of (B) to increase visibility.

The FFE chip was milled out of poly(methyl methacrylate) (PMMA, refractive index $n_D^{20°C} = 1.4918$ [1]) using an MDX-540 milling machine (Roland DGA, Irvine, CA). A detailed description of the general milling procedure can be found elsewhere.[2]

The chip consists of a top and bottom part (see **Figure S1**). A marking scheme (1 mm deep channels) was carved in the top of the chip (**Figure S1A and B**) and coloured by a common yellow highlighter (0.7 mm wide). Common text markers consist of luminophores (in this case fluorescein), which are visible during imaging. Metal electrodes (platinum wire, 0.5 mm diameter) and metal tube connectors (Luer stub adapters, Gauge 16 and 25) were glued to the bottom part using common two-component epoxy glue. Both parts are aligned and temporarily fixed on top of each other by using bolts in the thru-holes (larger holes in the corner of both the top and bottom part). Subsequently, both parts are bonded together by introducing dichloromethane into the small gluing holes (only present on top part) to provide a strong and irreversible seal. After the bonding is complete (10–30 minutes), the bolts are removed.

The chip name (*Temiska-2*) was derived from *Temiskaming Shores*, a small town in the north of Ontario, Canada. The *2* indicates the serial number. FFE chip model was designed using Solid Edge (Siemens PLM, Plano, TX) software. Immersion oil (refractive index $n_D^{20°C} = 1.517$) was applied to the top of the chip (the side with the markings) to improve the visibility and reduce scattering and refraction. Please note that we did not optimize the chip or the chip manufacture for this study. The chip possesses five sample inlets and five sample outlets.

Imaging chamber assembly

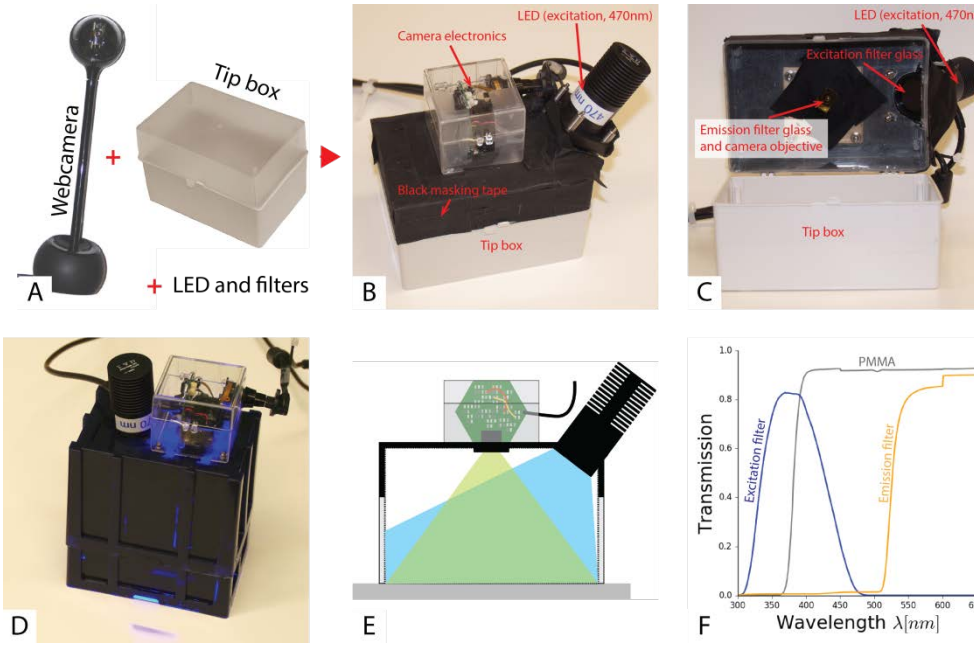

Figure S2. Assembly and components of image chamber. Images show the main components (A), the outside (B) and inside (C) of the imaging chamber, an example of another version of the imaging chamber (D), and a sketch of the imaging box (E). The plot in (F) shows the transmission spectra of PMMA, the excitation filter, and the emission filter. Filter and PMMA transmission spectra were recorded on a Beckman DU 520 Spectrophotometer (SCIEX, Vaughan, ON, Canada).

The image chamber was designed for photoluminescence imaging and consists of a camera objective, a micropipette tip box acting as a case, glass filters, and an LED for excitation (see **Figure S2**). A standard laptop (TravelMate 6592G from Acer) running Windows XP was used as the acquisition platform.

An empty micropipette tip box (127 × 85 × 82 mm) is acting as a case for the imaging chamber. It was adjusted by cutting in holes for fixing the camera electronics and the excitation light. Also, an imaging window/hole was cut into the bottom (100 × 70 mm). (Semi)-Transparent parts of the box were covered with black masking tape (see **Figure S2B**). Since the setup is very modular, it is simple to exchange parts to adapt to new situations. An example of another setup with a different case can be seen in **Figure S2D**.

The camera objective and electronics were extracted from a QuickCam® Orbit™ MP from Logitech (Newark, CA, USA). They were placed in a small plastic box prepared with holes for fixing and the USB cable. The box was fixed on top of the actual imaging chamber using small screws. The focus of the camera was adjusted with the focus ring on the objective. In front of the objective, an orange emission filter (see **Figure S2F** for spectrum) was placed.

A 470 nm LED (M470L3) from Thorlabs (Newton, NJ, USA) was used for illumination. An excitation filter was placed (see **Figure S2F** for spectrum) in front of the LED.

Signal distribution

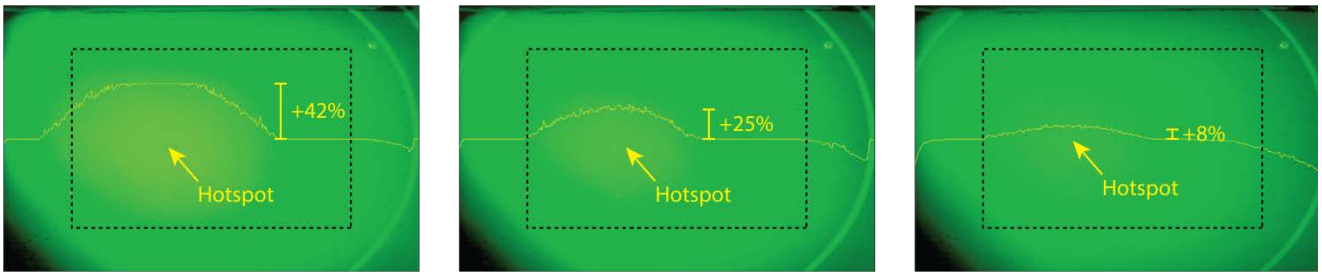

Figure S3. Three example pictures showing the signal distribution in the imaging chamber for different excitation intensities (arbitrary, decreasing from left to right). The yellow lines depict the light intensity in the centre of the frame (cross section; same scale for every picture). The black dashed rectangle shows the size of the separation zone of the FFE chip used in this study.

To check the signal distribution in the measurement area, we filled a large Petri dish with a fluorescein solution (500 µM) and placed the imaging system on top to take pictures at different excitation intensities. The three example pictures in **Figure S3** show that there is a hotspot on the left side of the image.

However, this hotspot can be tuned by adjusting the excitation light intensity. Here, reducing the intensity (from left to right picture) leads to an almost homogeneous signal distribution in the centre of the chamber with a large enough area to cover the separation zone of the FFE chip used in this study. Therefore, we did not use any additional lenses to shape the excitation or emission light for the presented study.

Experimental details

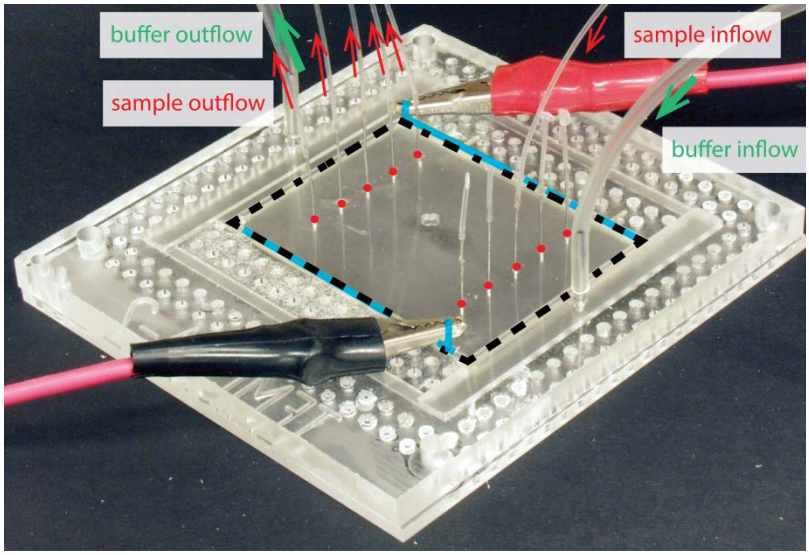

Figure S4. Bottom view of the chip showing the tube and electrode connections. Large tubing (3 mm inner diameter; polyethylene) is used for buffer in and outflow (green arrows), while small tubing (1/16" inner diameter; polyethylene) is used for sample in and outflow (red arrows and red circles). The black dashed lines mark the separation zone. The blue lines mark the electrode and the connectors at which the high voltage is applied (the electrodes themselves run over the whole side of the separation zone; see also **Figure S1D**).

Chemicals

All solutions were prepared using analytical grade reagents. 4-(2-hydroxyethyl)piperazine-1-ethanesulfonic acid sodium salt (HEPES), rhodamine 6G hydrochloride (rhodamine), fluorescein sodium salt (fluorescein), 5(6)-Carboxyfluorescein (carboxyfluorescein), sodium hydroxide, and immersion oil (UV-transparent, fluorescent-free) were purchased from Sigma Aldrich (Oakville, ON, Canada). Deionized water (18.2 MΩ cm^{-1}) was used for preparation of all solutions.

General experimental details

We used HEPES (10 mM, pH 7.6) as the BGE and solutions of rhodamine, fluorescein, and carboxyfluorescein (500 µM each) as *tracer dyes*. Flow rates of the BGE and the sample solution were in the range of 3–4.5 mL min^{-1} and 2–5 µL min^{-1}, respectively. All sample inlets were closed except for the one used in the respective study. Sample outlets were only opened for the flow-profile-measurement but were closed otherwise (see **Figure S4**).

Measurement of a mixture of all three tracers (directory: evalmix/mix1)

Buffer flow rate and sample flow rate were 4.5 mL min^{-1} and 2 µL min^{-1}, respectively. 37 s after start of measurement the voltage was turned on (275 V, *i.e.* 56.1 V cm^{-1}; top: anode; bottom: cathode). Current measured was between 8 and 9 mA. 205 s after start of measurement the voltage was turned off. Only the centre inlet was open and all outlets were closed.

The corresponding video file reveals that the separation was successful but not stable (can be seen at about 200s). The reason was the huge bubble formation at both electrodes. Also, the stream does not go back to its neutral position but rather stays shifted even after the voltage is turned off (from 300s).

Measurement of a mixture of fluorescein and rhodamine (directory: evalmix/mix2)

Buffer flow rate and sample flow rate were 4.5 mL min^{-1} and 5 µL min^{-1}, respectively. Initially, there was no sample flow. 31 s after start of measurement the sample flow was started. At 631 s the voltage was turned on (150 V, *i.e.* 30.6 V cm^{-1}; top: anode; bottom: cathode). Current measured was stable at 4.3 mA. 826 s after start of measurement the voltage was turned off. During the measurement, the excitation light was slightly reduced at 768 s. Only the centre inlet was open and all outlets were closed.

The corresponding video file reveals that the sample inflow started at about 360 s. Also, the electric field was lower than in the previous experiment, there was still no stable separation (again due to bubble formation at both electrodes).

Measurement of flow profile with fluorescein only (directory: evalprofile)

Buffer flow rate and sample flow rate were 3.0 mL min^{-1} and 2 µL min^{-1}, respectively. The sample was introduced into one inlet (channel 1–5, numbered from bottom to top). All outlets were open. Initially, there was no sample flow. Sample flow was started shortly after each measurement started (at 71, 51, 21, 20, and 10 s for channel 1, 2, 3, 4, and 5, respectively) and stopped after the stream reached the right border of the separation zone (at 300, 327, 241, 353, and 142 s for channel 1, 2, 3, 4, and 5, respectively). No electric field was applied at any of these experiments.

Data format

A PNG file itself consists of several chunks represented by a four-letter code (*e.g.* IHDR, IDAT, tEXt). The code labels are case sensitive. The case of the first letter, for instance, marks the corresponding chunk as *critical* or *ancillary* chunks (for more information see the standard[3, 4]). *Critical* chunks are mandatory and hold the information for rendering an image. *Ancillary* chunks can hold any additional information, which is not critical for rendering the image − if a program encounters an ancillary chunk it does not understand it can safely ignore it. This allows the addition of metadata to a PNG file without breaking it.

We decided to use a twin-track approach in order to achieve both reliability and readability. First, a new, private chunk labelled **dfFe** (**d**ata of **f**ree **f**low **e**lectrophoresis) was introduced for reliably storing data. Secondly, the data stored in this chunk is mapped to the public tEXt-chunks for easy readability. Chunks are generated by serialization of a *Python* dictionary (an associative array), which contains information generated and put together during the various stages of experimentation and evaluation by the corresponding programs.

Table S1 gives a short overview of data the dfFe-chunk can hold and by which program this data is generated. The format is open, so it can easily be extended. For convenience, we developed a command-line tool **pngdata.py**, which can read and modify the **dfFe**-chunk as well as display all chunks present in any PNG file. This is used by the provided batch files to add the information about chip features into the data files. In order to better distinguish regular PNG files from ones with FFE data in them we use a double extension *.FFE.PNG* (instead of pure *.PNG*) for the latter. This ensures the compatibility with viewers (e.g. the preview function of Windows Explorer) and editors.

Table S1. Overview of properties in a dfFe-chunk. Please see the respective programs for information on the specific format of each property.

Property	Description	generated by program
Channels recorded	Channels recorded during acquisition	**acquire.py** (set in **setupcamera.py**)
Dataline 1	Data line 1 provided by user	**acquire.py** (set in **setupmeasurement.py**)
Dataline 2	Data line 2 provided by user	**acquire.py** (set in **setupmeasurement.py**)
Experiment name	Name of experiment provided by user	**acquire.py** (set in **setupmeasurement.py**)
Image counter	Number of image in set	**acquire.py**
Inlets	List of inlet coordinates and widths (in mm)	**manual (pngdata.py)**
Inner separation zone	Four coordinates in pixels of the separation zone (inner box of markings)	**findfeatures.py**
Outer separation zone	Four coordinates in pixels of the outer box around the markings	**findfeatures.py**
Outlets	List of outlet coordinates and widths (in mm)	**manual (pngdata.py)**
Physical zone dimensions	Physical zone dimensions (in mm)	**manual (pngdata.py)**
Snapshot time	Time in seconds elapsed since start of measurement	**acquire.py**
Timestamp	Timestamp of image acquisition	**acquire.py**
Total frame counter	Number of frames recorded and integrated	**acquire.py**
Trajectories	List of lists containing the trajectory points and widths (in pixels)	**findtrajectories.py**

Short description of Python programs

General

The software suite includes a function library (**ffe.py**) and some ready-to-use programs for setup, acquisition, processing, and evaluation of FFE experiments. The programs were written in *Python 2.7* and according to *Eric Raymond's 17 Unix Rules* (in particular following the *Rules of Simplicity and Modularity*).[5] Main modules used are *Numpy*, *Mathplotlib*, *Scipy*, and *OpenCV* (Open Source Computer Vision Library). The software suite is compatible with every camera supported by *OpenCV* itself including a huge variety of web-cameras (see documentation of *OpenCV*).

Most of the programs are console-based programs that can be controlled through command line arguments. For a list of arguments call the respective program with `--help`. All console-based programs were designed to allow the seamless integration into batch processes. Examples of these batch processes are included in the software package, *e.g.* for evaluating the flow profile of the FFE chip. The programs for setup the camera and measurement provide a *Graphical User Interface* (GUI).

Please note that the general descriptions below are meant for providing a rough overview. Please see the source code and the library documentation (**ffe.m.html**) for more details. The step numbers given below correspond to the numbers given in comments in the respective programs.

Acquire.py (console)

This programs records snapshots and a video using the camera (**setupcamera.py**) and measurement settings (**setupmeasurement.py**). It is automated and only provides a live view with almost no user interaction. Briefly, the program

0. setups the logging
1. opens and setup the camera device
2. reads and setup measurement parameters (including creating a directory for saving the data)
3. records and saves a background image
4. setups the video recording device
5. starts recording (integrating frames, creating overlay for live view and video, saving frames to images and video)
6. cleans up

Alignchip.py (GUI)

This program aids the user in performing the alignment of the chip (see also main text). Direct, visual feedback on the state of alignment is provided. Briefly, the program detects the separation zone and bounds it within a rectangle. It also draws a reference rectangle which represents the target (optimal) orientation of the separation zone in the imaging field of view. The overlap between the areas of the two rectangles is calculated, and presented as a percentage. If the alignment is perfect, both rectangles are equal and the overlap is 100%. The rectangles are rendered on a black canvas with using red (detected separation zone of the chip) and green (target orientation) filling colour, respectively. In the overlapping area the resulting colour is yellow. The overlap percentage is calculated by counting the yellow pixels and dividing it by the pixels of one of the rectangles. **Figure 2** in the main text shows a scheme for alignment.

Combineandrenderflow.py (console)

This program is a part of a quick&dirty example on developing advanced evaluation methods (here: flow profile) based on the data provided by trajectory-finding and separation-zone-detection. The program itself depends on the output of **timeposition.py**, which extracts the time-dependent position of the stream-fronts for each inlet-channel (1–5, see experimental details). The script reads the data from **flowch1-5.csv** (experimental input) as well as from **calculatedvelocityfield.txt** (COMSOL input) calculates the velocities, and plots flow profiles for both input sets (see **Figure 7** in main text).

Ffe.py (library)

This module is the core and heart of the presented software suite. It defines commonly used functions and algorithms for the image processing and analysis system and its programs. Please see the module documentation (**ffe.m.html**, which was generated by **pdoc**) and the source code for details.

Findfeatures.py (console)

This program opens a FFE.png file and tries to detect the separation zone markings as well as the flow markers. It will then save this information to the same PNG file. Briefly, the program

0. parses command-line arguments
1. setups the logging
2. extracts all contours from the image
3. tries to find the separation zone (see main text for a description)
4. tries to find the flow markers (see main text for a description)
5. saves the position of the separation zone and flow markers in the data file
6. cleans up

Findtrajectories.py (console)

This program opens a FFE.png file and tries to detect all trajectories. It will then save this information to the same PNG file. Briefly, the program

0. parses command-line arguments
1. setups the logging
2. reads input image and data (*e.g.* separation zone coordinates, physical resolution, etc.)
3. determines the inverse flow direction (based on markers, if present)
4. calculates the resolution (in mm per pixel)
5. creates a list of starting points from the list of inlets
6. tries to find trajectories for each channel separately
7. saves trajectories positions and widths (pixel based) in the data file
8. cleans up

Pngdata.py (console)

This program allows viewing and editing the dfFe chunk in a PNG file on the command line. It can be used to add information about features to the file for evaluation such as it is done for the example batches. Please see help page (`--help`) and source code for more details on this program.

Rendertrajectories.py (console)

This program opens a FFE.png file and renders the trajectories in this file to a plot. If no output file is given, the plot is presented directly to the user. The origin is set to the inlet. Trajectories are rendered according to the output style provided.

Resolution.py (console)

This program opens a FFE.png file, and extracts and renders the resolution in this file to a plot. If no output file is given, the plot is presented directly to the user. It will consider and render every possible combination of trajectories present in the data chunk of the file.

Setup.py (console)

This small program registers the **ffe.py**-module with the local Python installation (using **disutils**). See library documentation for more details.

Setupcamera.py (GUI)

This program helps the user to set up the camera for the measurement. Settings are saved to a file, which is used by the recording and alignment programs.

Setupmeasurement.py (GUI)

This program helps the user to set up the various parameters for the measurement. Settings are saved to a file, which is used by the recording program.

Timeposition.py (console)

This program is a part of a quick&dirty example on developing advanced evaluation methods (here: flow profile) based on the data provided by trajectory-finding and separation-zone-detection. This program extracts the time-dependent position of the stream front for each measured inlet-channel (1–5, see experimental details). The data is subsequently used by **combineandrenderflow.py** to render the flow profile of the chip.

References

1. M. J. Weber, *Handbook of Optical Materials*, Taylor & Francis, 2002.
2. F. J. Agostino, C. J. Evenhuis and S. N. Krylov, *J. Sep. Sci.*, 2011, **34**, 556−564.
3. T. Boutell, *RFC 2083*, 1997, DOI: 10.17487/rfc2083.
4. ISO, *Computer graphics and image processing - Portable Network Graphics (PNG)*, 2004, **15948**.
5. E. S. Raymond, *The Art of UNIX Programming*, Pearson Education, 2003.