OLIVER DRESSLER

ACTIVE AND PASSIVE CONTROL IN HIGH-THROUGHPUT MICROFLUIDIC EXPERIMENTATION

DISS. ETH NO. 24961

ACTIVE AND PASSIVE CONTROL IN HIGH-THROUGHPUT MICROFLUIDIC EXPERIMENTATION

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

OLIVER DRESSLER
MSc, ETH Zurich

born on 26 April 1989
from Riehen, Basel-Stadt
citizen of Switzerland

accepted on the recommendation of

Prof. Dr. A. deMello, examiner
Prof. Dr. P. Arosio, co-examiner

2018

DOI: ?

ABSTRACT

Microfluidic high-throughput experimentation has attracted significant research effort in recent years. While a multitude of traditional chemical and biological assays have been transferred to microfluidic platforms, not many advanced control schemes have been implemented to date.

This thesis presents several key techniques, allowing for the control of both single-phase and two-phase flow in microfluidic channels by applying active and passive control schemes and advanced control algorithms.

Droplet synchronization is frequently required in multi-step droplet-based assays, as it enables controlled ordering of multiple types of droplets. We present a novel, passive microfluidic architecture, which allows for droplet ordering and synchronization with unprecedented precision. We have further applied the presented architecture to study osmotic transfer between droplets in motion and found that inter-droplet transfer rates are increased by larger flow velocities.

While many high-throughput techniques allow for the manipulation and screening of microfluidic droplets at high frequencies, individual droplets can typically not be identified. Therefore, we have developed an active strategy for the production of large droplet populations, labeled using fluorescent dyes. Due to the active approach the developed strategy is flexible in regard to the type of fluorophore, the type of detector and the number of labeled droplets. The presented droplet barcoding system was further used to produce more than two thousand unique, fluorescently-labeled droplets. We further present a long-term droplet storage system to aid droplet storage during high-throughput experimentation, allowing for low-loss storage and retrieval of microfluidic droplets.

Recent advances in neural network-based control algorithms have shown human-level performance in a variety of environments, such as games. Therefore, we have applied two state-of-the-art reinforcement learning algorithms to two microfluidic environments. We could show that superhuman performance is attainable in microfluidic environments, highlighting the utility of novel control algorithms for automatic high-throughput microfluidic experimentation.

Finally, we have employed a neural network-based classifier to perform real-time sorting of analytes flowing through microfluidic channels. Us-

ing the presented sorting system we have successfully separated cells and microbeads.

ZUSAMMENFASSUNG

Mikrofluidische Hochdurchsatz-Experimente haben in den letzten Jahren erhebliche Forschungsbemühungen ausgelöst. Während eine Vielzahl traditioneller chemischer und biologischer Assays auf mikrofluidische Plattformen übertragen wurden, sind nur wenige fortgeschrittene Kontrollsysteme implementiert worden.

Diese Arbeit stellt einige Schlüsseltechniken vor, die es ermöglichen, Flüsse in mikrofluidischen Kanälen durch Anwendung aktiver und passiver Kontrollschemata, sowie fortschrittlicher Kontrollalgorithmen zu steuern.

Die Synchronisation von mikrofluidischen Tröpfchen wird häufig benötigt in mehrstufigen Assays, da sie eine kontrollierte Anordnung mehrerer verschiedenen Tröpfchenarten ermöglicht. Wir präsentieren eine neuartige, passive mikrofluidische Architektur, welche die Tröpfchenanordnung und -synchronisation mit bisher unerreichter Präzision ermöglicht. Zudem haben wir die vorgestellte Architektur angewendet, um den osmotischen Transfer zwischen Tröpfchen in Bewegung zu untersuchen und fanden heraus, dass die Übertragungsraten zwischen den Tröpfchen durch größere Strömungsgeschwindigkeiten erhöht werden.

Während viele Hochdurchsatztechniken die Manipulation und das Screening mikrofluidischer Tröpfchen mit hohen Frequenzen ermöglichen, können einzelne Tröpfchen typischerweise nicht identifiziert werden. Daher entwickelten wir eine aktive Strategie für die Produktion von Tröpfchen, mittels Markierung durch Fluoreszenzfarbstoffen. Aufgrund des aktiven Ansatzes ist die entwickelte Strategie flexibel hinsichtlich der Art des Fluorophors, der Art des Detektors und der Anzahl der markierten Tröpfchen. Das vorgestellte System wurde zudem verwendet, um Tausende von einzigartigen fluoreszenzmarkierten Tröpfchen zu erzeugen. Darüber hinaus präsentieren wir ein Langzeit-Tröpfchenspeichersystem zur Unterstützung der Tröpfchenspeicherung während eines Hochdurchsatz-Experiments, das eine verlustarme Speicherung und Wiedergewinnung von mikrofluidischen Tröpfchen ermöglicht.

Jüngste Fortschritte mittels neuronalen netzwerkbasierten Steueralgorithmen haben gezeigt, dass Algorithmen Menschen übertreffen koennen in einer Vielzahl von Umgebungen, wie zum Beispiel Spielen. Daher haben

wir zwei Steuerungsalgorithmen auf zwei mikrofluidische Umgebungen angewendet. Wir konnten zeigen, dass die Algorithmen besser Resultate erzielen als Menschen in mikrofluidischen Umgebungen, was den Nutzen neuer Steuerungsalgorithmen für automatische Hochdurchsatz-Mikrofluidik-Experimente unterstreicht.

Schließlich haben wir einen Klassifikator basierend auf einem neuronalen Netzwerk verwendet, um eine Echtzeit-Sortierung von Analyten durchzuführen, die durch mikrofluidische Kanäle fliessen. Mit dem vorgestellten Sortiersystem haben wir erfolgreich Zellen und Mikropartikel getrennt.

ACKNOWLEDGEMENTS

No man is an island, entire of itself.

— John Donne

This journey has only been possible thanks to the support of my family, mentors, and friends.

I would like to thank my thesis advisors Andrew J. deMello and Robert C. R. Wootton for their valuable guidance, inspiring conversations and boundless patience, allowing me to discover and pursue exciting and novel science.

To my family, Luzius, Franziska and Angela, I want to extend my sincere gratitude for inspiring me since a young age, challenging me to explore novel areas, and continuous support throughout my whole life. I know you have always believed in me, thank you!

I also want to thank my Grandfather, Kurt, for opening the door to the world of science. I have fond memories of our visits to museums and the observatory!

To my friends, thank you for supporting me throughout my whole PhD, keeping me grounded and providing welcome distractions!

Many members of my research group have contributed to this thesis through direct collaboration, encouragement and interesting discussions. I would particularly like to thank Xavier for fruitful collaborations and always having an ear for my problems. Further, I would like to thank Gregor (Bender) for a great collaboration on cell sorting and a lot of fun moments.

Finally, I want to thank Alexandra for her boundless support. You are my rock!

Parts of this research have been funded by the National Research Foundation of Korea (Global Research Laboratory Programme Grant K20904000004-10A0500-00410).

CONTENTS

1

INTRODUCTION

The concept of miniaturization has proved to be a dominant paradigm in almost all areas of science and technology over the past half century. Most notably, the semiconductor revolution (catalyzed by Jack Kilby's demonstration of the integrated circuit in 1958) has in large parts been driven by a continual reduction in both the size and cost of electronic components (2, 3), which have in turn engendered enormous increases in computing power. Indeed, transistor sizes have decreased by a factor of 10^5 over the past 60 years, with contemporary integrated circuits containing more than 10^9 transistors per cm^2. Significantly, and in addition to the immediate improvements in computational power and cost, such reductions in component size and density also enable entirely new modes of operation and areas of application (3).

In a similar manner, the development of microfluidic technologies over the past 25 years has attempted to transfer related and additional benefits of miniaturization to the fields of chemical and biological experimentation. The rapid acceptance of microfluidic technologies has largely been driven by concomitant advances in the fields of genomics, nanotechnology, proteomics, drug discovery, single-cell analysis, high-throughput screening, and diagnostics, with an identified need to perform rapid measurements on small sample volumes (4). However, at a more fundamental level, microfluidic research has been accelerated by the fact that physical processes can be more easily controlled (in space and time) when instrumental dimensions are reduced to the micron scale. The relevance of such a technology set is significant and is characterized by various features that accompany system miniaturization. Such features include the ability to process small volumes of fluid, enhanced analytical performance, reduced instrumental footprints, high analytical throughput, facile integration of functional components within monolithic substrates and the capacity to exploit atypical fluid behavior to control chemical and biological entities (5, 6).

Unsurprisingly, the efficient manipulation of fluids on femtoliter to nanoliter scales is a non-trivial undertaking. As such, a diversity of dedicated methods has been developed to allow the performance of basic processes,

Parts of this chapter have been published in (1).

such as fluidic pumping, sample preparation, aliquoting, dilution, concentration, mixing, incubation, and isolation (*7*). Despite the substantial progress made in this direction, many inherent characteristics of microfluidic systems can be advantageous under certain circumstances but injurious in others. This is the case for continuous flow microfluidic platforms, which leverage the direct scale dependencies of heat and mass transfer while maintaining a high degree of operational and structural simplicity (*6*). Despite their obvious attractions, continuous-flow formats frequently become less attractive and/or impractical due to issues such as Taylor dispersion, solute surface interactions, cross-contamination, and the need for excessive reagent volumes or relatively long channels (*8*). Accordingly, alternative methods of fluid manipulation are required if the true potential of system miniaturization is to be fulfilled.

To this end, recent years have seen significant interest in the development and application of droplet-based (or segmented-flow) microfluidic systems for chemical and biological experimentation. In basic terms, droplet-based microfluidic systems generate, manipulate, and process discrete drop-lets contained within an immiscible carrier fluid. They leverage immiscibility to create discrete and isolated volumes that reside and move within a continuous flow. Significantly, these platforms allow for the production of monodisperse droplets at rates in excess of tens of kilohertz and provide for independent control over each droplet in terms of its size, location, and chemical makeup. Critically, the use of droplets in complex chemical and biological processing leverages the ability to perform a range of integrated unit operations in high throughput. Such operations include droplet generation, merging/fusion, sorting, splitting, dilution, storage, and sampling (*9*, *10*). Based on these compelling characteristics, it is unsurprising that droplet-based microfluidic systems are increasingly being used as environments in which to perform a range of biological and chemical assays in an efficient, integrated, and rapid manner.

1.1 FUNCTIONAL COMPONENTS AND UNIT OPERATIONS

1.1.1 *Droplet Generation*

Femtoliter- to nanoliter-volume droplets can be generated in a variety of ways within microfluidic systems, but it is important to note that passive strategies for droplet generation have proved to be remarkably straightforward in their implementation and effective in their ability to generate

large numbers of user-defined droplets (*11*). Put simply, passive strategies leverage geometric control of microfluidic intersections to transform arbitrary volumes of fluid into multiple, uniform, and sub-nanoliter droplets at megahertz rates. The most widely adopted microfluidic constructs for droplet production are the T-junction (or crossflow structure) (*12*), the flow-focusing geometry (*13*), and the co-flow structure (*14*) (see Figure 1.1 a–c), although refinements of these basic components have also been reported (*15*, *16*). As noted by Collins and co-workers (*11*), the basic mechanism governing the operation of each of these geometries involves the creation of an interface between two co-flowing and immiscible fluids and the self-segregation of one of the fluids into (discrete) droplets that are surrounded by the other (continuous) fluid; the identities of the discrete phase (that forms the droplets) and the continuous phase (that surrounds the droplets) being defined by the surface energies of the two fluids with respect to the channel surface.

The earliest reported use of a capillary-based system for the formation of monodisperse droplet populations was presented by Weitz and coworkers in 2000 (*14*). Specifically, a co-flow of two immiscible phases within a tapered capillary was used to generate droplets (when streamwise forces exceed interfacial tension), whose size is dependent on the capillary tip diameter, continuous phase velocity, extrusion rate, and the viscosities of the two phases. Soon after, Quake and colleagues (*12*) reported the rapid generation of droplets using a crossflow geometry. Here, immiscible fluid streams meet at an angle (normally 90°) to each other, with the fluid of one phase being sheared by a second immiscible phase flowing from the orthogonal channel. Again, droplet size could be controlled by altering the relative flow rates of the two immiscible phases. Finally, the flow-focusing geometry introduced by Anna et al. (*13*) in 2003 involves the acceleration of concentric but immiscible flows upstream of a small orifice. Such acceleration causes a narrowing (due to pressure and viscous stresses) of the inner fluid thread, which then breaks inside or downstream of the orifice. Significantly, flow-focusing formats provide for control of both droplet size and generation rate over extremely wide ranges.

Despite an overwhelming reliance on droplet generators based around crossflow structures, flow-focusing geometries, or the co-flow schemes, other passive methods for droplet generation are highly effective under certain circumstances. These include extrusion over a continuous phase-flooded terrace and step (*17*) and the use of channel height variations to subject immiscible interfaces to gradients of confinement (without the

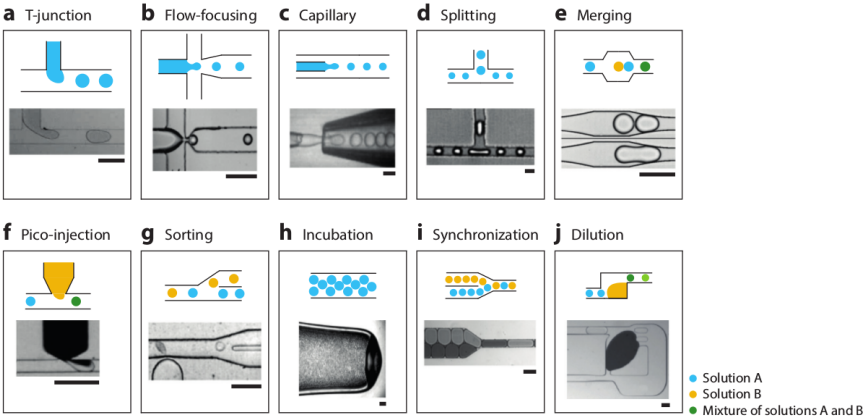

FIGURE 1.1: Droplet formation and manipulation (scale bar - 100 µm). The first three panels illustrate common geometries for the production of monodisperse droplets. The (a) cross-flow or T-junction geometry relies on the two fluids meeting perpendicularly, whereas (b) the flow-focusing geometry involves concentric acceleration of immiscible fluids through a small orifice. Finally, (c) a tapered capillary can be inserted into a larger channel, which yields a flow-focusing–like geometry. Droplet manipulation methods are necessary for the translation of complex assays to a microfluidic format. (d,e) Splitting and merging droplets can be achieved by controlling channel geometries. (f) Pico-injection can be used to actively inject a user-defined volume into a passing droplet and is typically triggered using an electrical field. (g) Droplets can be sorted by deflection through a variety of forces; here, dielectrophoretic sorting at high frequencies is shown. (h) Space-efficient incubation of droplets requires removal of the carrier fluid prior to storage, yielding a tightly packed droplet configuration. (i) Low-error passive synchronization of two types of droplets into an alternating configuration can be achieved by self-ordering droplets. (j) Serial dilutions of droplets can be achieved by repeatedly merging and resplitting buffer droplets with a stationary mother droplet.

need for flow of the external phase) (*18*). Both approaches are noteworthy, due to their simplicity of operation and ability to robustly produce large populations of monodisperse droplets. Finally, the ability to generate sub-picoliter-volume droplets at megahertz frequencies has significant implications for high-throughput single molecule experimentation. To this end, Shim and coworkers (*19*) recently presented the use of a constricted flow-focusing geometry to enhance flow velocities during droplet formation (without generating high internal pressures), and via a tip-streaming mechanism in dripping mode, they were able to generate <10 fL-volume droplets at rates well in excess of 10^6 per second. Droplet production rates of several megahertz were also attained by Mittal et al. (*20*), who used a highly parallelized passive step–emulsification method.

1.1.2 *Droplet Storage and Payload Retention*

One key feature of droplet-based microfluidic systems is the facility to generate extremely large populations of isolated droplets in very short times. Such parallelization and automation of fluid handling, combined with the ability to control the chemical/biological payload of individual droplets, make the platform appealing for many biotechnology and large-scale screening applications. As previously noted, droplets are produced in a manner that ensures envelopment by the carrier fluid and separation from both the microchannel surface and adjacent droplets. Accordingly, reactions or assays occurring within droplets can be monitored (through assessment of temporal concentration changes) by following droplets as they move downstream, for example using wide-field imaging methods (*21*). Such an approach requires that the reaction kinetics are sufficiently rapid to ensure that any reactions are essentially complete before the droplet under investigation has exited the microfluidic device. In many situations, however, the time scale of the assay or screen is excessively long, and thus droplets need to be stored in a static fashion over periods ranging from hours to weeks (*22*, *23*). Under these conditions, it is critical to ensure that droplet integrity is maintained (i.e. droplets should not merge or split) and that analytes within droplets remain encapsulated and are not transported to the continuous phase or adjacent droplets. Stabilization of droplets is typically achieved using surfactants, which preserve the emulsion and inhibit droplet coalescence by stabilizing the interface between the immiscible phases. Surfactant molecules are normally mixed into the continuous phase, and upon contact with the discrete phase self-arrange at the inter-

face. As discussed by Baret (*24*), surfactant choice is intimately linked to the chemical nature of the phases used and the surface of the containing microchannel.

Due to the need for emulsion stability in a wide range of chemical and biological systems, many surfactants have been developed and used in droplet-based assays. For example, aqueous droplets dispersed in hydrocarbon oils can be efficiently stabilized using commercially available surfactants such as Span80 (*12*) or Tween20 (*25*). Nevertheless, due to the widespread adoption of fluorinated oils as carrier fluids (due to their excellent biocompatibility, high gas permeability, and poor solvent capacity for organic compounds) in droplet-based microfluidics (*26*), fluorosurfactants (perfluoropolyethers containing hydrophilic head groups) offer exceptional long-term stabilization of droplets. In this respect, Weitz and coworkers (*23*) have reported a novel class of fluorosurfactants synthesized by coupling oligomeric-perfluorinated polyethers with polyethyleneglycol. Such block copolymer surfactants have been shown to be highly effective in stabilizing water-in-oil emulsions during droplet formation and incubation, and have been used for in vitro translation and single-cell analysis experiments. More recently, the same group reported the synthesis and characterization of polyglycerol-based triblock surfactants (with tailored side chains), which exhibited lower critical micellar concentrations and efficient operation in cell encapsulation and in vitro gene expression studies (*27*).

As previously noted, an equally important aspect of droplet stability is the capacity of droplets to effectively retain their contents over long periods of time. Molecular retention is to a lesser or greater extent compromised by mass transport effects through the continuous phase, which leads to cross talk between droplets and the equalization of their contents over time. This is especially problematic during storage, when droplets are in close proximity for long periods of time. Although surfactants are extremely effective in stabilizing droplets and indispensable in preventing coalescence, they play a fundamental role in driving cross-talk effects (through micelle formation and molecular interactions between their amphiphilic groups and the molecules contained in the droplets) (*28*). Crucially, it was recently shown that the surfactant concentration within the continuous phase positively correlates with the permeability of the continuous phase (*28*). Other parameters that are shown to correlate with droplet permeability include inter-droplet distance and flow of the continuous phase around stationary droplets (*29*). Both studies demonstrate that transport is limited by

surfactant-mediated transfer of components through the continuous phase, which occurs more rapidly for closely spaced droplets or under flow conditions (see Figure 1.2 a).

Solid particles can also be used to stabilize emulsions. Pickering emulsions (in which surfactants are replaced by nanoparticles such as colloidal silica) are interesting formulations because they are simple in their construction and yield droplets with a high resistance to coalescence. In recent years, such systems have been used to emulsify microfluidic droplets, minimizing leakage, whilst maintaining droplet stability (*30*). For example, fluorinated silica nanoparticles are highly efficient at generating Pickering emulsions when using fluorinated oil carrier fluids (*30*). Nevertheless, it must also be remembered that the inherent stability of Pickering emulsions can also lead to experimental challenges, as nanoparticles are strongly adsorbed at the liquid–liquid interface and thus impede droplet manipulations (such as splitting and fusion).

Finally, as previously noted, many droplet-based experiments require the incubation or storage of droplets over extended time periods. High droplet densities during storage are normally achieved by partial removal of the continuous phase, resulting in tightly packed droplet configurations (see Figure 1.2 b). Such an approach ensures that large droplet populations may be assayed, but adversely affects both emulsion stability and droplet cross talk (*29*). Droplets can be incubated in extended channels or tubing, but despite ensuring droplet integrity and minimal cross talk, such techniques do not scale well due to increasing hydrodynamic pressure (*31*).

Alternatively, droplets can be stored at the interface between the continuous phase and a third immiscible phase (see Figure 1.2 c). This prevents evaporation of droplets over time by shielding them from ambient air, but it does significantly hinder the retrieval of all droplets due to their strong adherence to this interface. To mitigate such shortcomings, mesofluidic storage chambers can be used for the efficient storage of large droplet populations. This approach, as exemplified by work described in this thesis, allows the facile storage and retrieval of droplets (see Figure 1.2 d), with minimal droplet merging or splitting during filling and reinjection. Specifically, a three-dimensional–printed glass hybrid device can be used to store the emulsion. During deposition, droplets rise to the top of a glass vial due to buoyancy, with retrieval being enabled by turning the device upside-down and applying pressure to the outlet. This simple protocol combined with the lack of a fluid interface enables retrieval of more than 99 % of

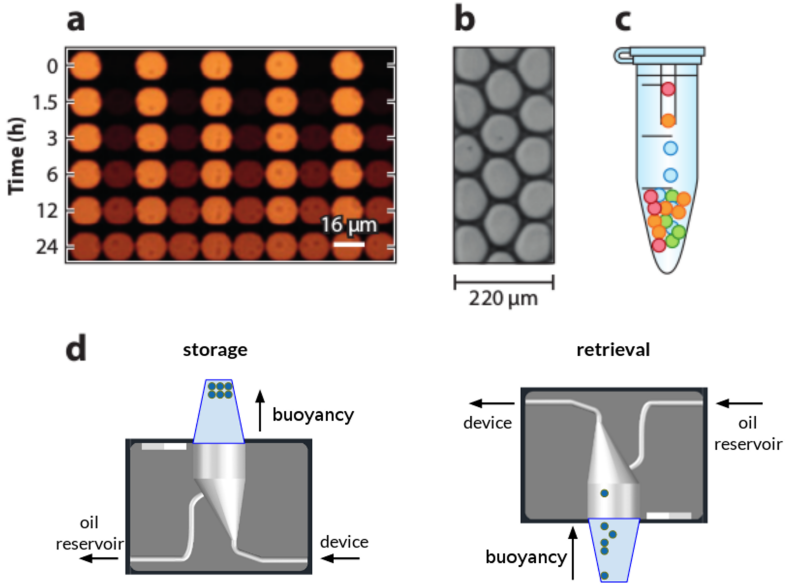

FIGURE 1.2: Droplet leakage and storage. (a) Gruner et al. (*29*) reported the transfer of fluorophores between stationary droplets ultimately resulting in homogenous droplet populations. (b) Space-efficient storage of droplets requires the removal of the carrier liquid. The resulting tightly packed emulsion exhibits enhanced contact between droplets, which in turn increases the likelihood of inter-droplet transfer of analytes. (c) Traditionally, droplets are stored in tubes and capped with a third immiscible fluid to prevent evaporation. Nevertheless, droplets adhering to this fluid interface are typically lost during retrieval. (d) A simple three-dimensional–printed glass hybrid droplet storage device, which allows for very high droplet retrieval efficiencies. During storage, droplets rise to the top of a silanized glass vial due to buoyancy. Retrieval simply involves turning the device upside-down and applying pressure to the outlet. This simple protocol ensures that droplet losses are <0.5 % and can be used for the storage of small and large droplet populations in a reliable manner.

collected droplets. It should be noted that a more detailed description of this work is provided in Chapter 3.

1.1.3 *Droplet Manipulations*

The efficient transferal of chemical and biological workflows to droplet-based microfluidic formats requires the development of a toolbox of unit operations that mirror their macroscale counterparts. Components should perform a given task in a (preferably) passive/automated manner and be readily integrated with other functions that suit the requirements of a specific biological experiment. Such functional components are likely to include modules for droplet generation, fusion, splitting, sorting, payload mixing, analysis, and dilution, incubation, storage and reinjection. Fortunately, over the past decade, numerous microfluidic components have been developed for such tasks and are now available to the experimentalist. A selection of these functional components is shown in Figure 1.1 d–j.

Aliquoting or droplet sampling is commonly performed by splitting a droplet under study into two or more daughter droplets. Controlling the size of daughter droplets can be achieved directly and passively by forming constricted (non-relaxed) droplets through variations in channel geometry (*32*). In this respect, the use of T-junctions or isolated obstacles within the flow path can be used to generate shear forces that overcome surface tension and lead to droplet breakup. Importantly, such geometry-mediated breakup allows the creation of segmented flows with controllable droplet volumes and volume fractions. Conversely, the fusion or merging of two droplets is a fundamental process, affording operations such as reaction initiation/termination, droplet dilution/concentration and reagent dosing. Numerous techniques for merging droplets within segmented flows have been reported. These are either active [involving the use of electric fields (*33*)] or passive [leveraging the surface properties or geometry of the flow channel (*34*)] in nature, but most commonly involve bringing droplets into close contact, with subsequent destabilization of the interface between them. Elegant examples of passive droplet merging include the use of channel expansions to destabilize droplet pairs (*34*) and the exploitation of differences in hydrodynamic resistance of the continuous phase and the surface tension of the discrete phase (through the use of pillar arrays) to controllably merge multiple droplets on millisecond time scales (*35*). It should also be noted that active approaches for droplet fusion have been realized using a range of external stimuli, including thermocapillary forces (*36*) and elec-

tric fields (*37*), and provide a degree of dynamic control not available to passive methods.

Efficient droplet merging typically requires the efficient synchronization of droplets of interest, which in turn ensures the correct alignment of droplets entering the merging structure. In the simplest scenario, two types of droplets should be arranged in an alternating configuration prior to merging. This can be achieved through active control (requiring the integration of control algorithms and architectures) or the use of (passive) geometrical variations. That said, previously reported passive ordering structures have exhibited limited synchronization capabilities when droplets enter the structure at significantly different frequencies (*38*). To address this limitation, we have thus developed a novel (and passive) synchronization structure that allows for robust and accurate synchronization across a wide range of droplet frequencies and flow rates (*39*). This study is presented in Chapter 2.

As previously noted, the dilution or concentration of droplet contents can also be initiated through controlled droplet merging. For example, Niu et al. (*40*) reported a simple and elegant method for the creation of droplet concentration gradients by leveraging water–oil hydrodynamic interactions. Specifically, through a series of droplet merging, mixing, and resplitting operations, a nanoliter-volume mother droplet may be combined with a series of smaller buffer droplets to generate a sequence of daughter droplets defining a digital concentration gradient. The addition of reagents to a droplet can be achieved either by droplet coalescence (using the aforementioned methods) or by injecting reagents directly into preformed droplets (a process commonly termed pico-injection) (*41*). Pico-injection works in a similar fashion to electrically mediated droplet fusion (*42*), whereby a pressurized microchannel is used to inject a user-defined volume of reagent into a moving droplet. Put simply, when a droplet passes the pico-injector, electrodes opposite the channel containing the reagent are energized. This destabilizes water–oil interfaces and allows reagent to enter the droplet (since the pico-injector is maintained at a high pressure). As the droplet moves downstream, it remains connected to the orifice by a narrow bridge of fluid, which eventually breaks and detaches the newly formed droplet. Critically, control of electrode switching allows selective injection of reagent into droplets at kilohertz rates.

Finally, the ability to sort and collect droplets based on their content is a key component of many biochemical assays. Sorting can be achieved passively based on parameters, such as droplet size (*43*) or deformability (*44*),

but normally requires the rapid assessment of the droplet phenotype followed by the active triggering of the sorting process (*45*, *46*). Sorting strategies typically force passively flowing droplets to switch streamlines using an external perturbation, such as a dielectrophoretic force (*45*), thermocapillarity (*36*), valves (*47*) or surface acoustic waves (*48*). By integrating a gapped channel divider, recent refinements in electrophoretic-based sorting strategies have demonstrated efficient droplet sorting at frequencies up to 30 kilohertz (*49*). Such an innovation is enormously significant, because at such high frequencies, sorting is no longer the rate-determining step in the analytical process. Accurate sorting at high frequencies, as presented in (*49*), forms an integral part of our barcoding strategies described in Chapter 3.

1.2 PROBING SMALL VOLUMES: DETECTION IN DROPLETS

Basic requirements of any chemical or biological experiment include the identification and quantitation of relevant species within the assay volume. Detection within the picoliter volumes that characterize microfluidic-based droplets is especially challenging due to the paucity of sample present, and the fact that under normal circumstances, large numbers of droplets are moving at appreciable velocities through the system. Moreover, the study of dynamic processes imposes additional constraints on the detection method, typically requiring multiple measurements of a given droplet over a period of time that may range from a few milliseconds to hundreds of hours. Unsurprisingly, and due to their nondestructive nature, high sensitivities, fast response times and simple integration with chip-based systems, optical methods are by far the most commonly used methods for detection in microfluidic formats.

1.2.1 *Fluorescence*

A cursory survey of the literature confirms that (laser-induced) fluorescence methods are, by a large margin, the most utilized optical detection techniques in droplet-based (and continuous flow) microfluidics. The exquisite sensitivity and selectivity of both time-integrated and time-resolved emission spectroscopy are ideally suited to the non-invasive probing of the picoliter volumes and low-analyte concentrations representative of such droplets. Moreover, the ability to perform fluorescence measurements on sub-millisecond time scales means that extremely large numbers

of droplets (produced at kilohertz frequencies and moving at linear velocities in excess of 1 m/s) can be probed in an efficient and sensitive manner (*50*). The wide diversity of commercially available fluorescent probes, biological assays based on emissive reporters and the ability to perform multiplexed (single point or imaged based) detection, make fluorescence an ideal detection technique in many applications. As noted, fluorescence detection is well suited for the extraction of kinetic data. This now well-accepted approach was exemplified by Bringer et al. (*51*), who used time-integrated fluorescence measurements in a single digital polymerase chain reaction (PCR) experiment to simultaneously determine 16 separate nucleic acid targets.

The parallel detection of multiple fluorophore populations can also be realized by aligning multiple optical fibers perpendicular to a microfluidic channel (*52*). Using such a strategy, Cole et al. (*52*) presented a scheme for multicolour detection of moving droplets by integrating spatially offset optical fibers and a single photodetector. As droplets transitted the excitation regions, fluorescent bursts, separated by the time taken to move between excitation regions, could be generated and used to perform multicolour detection without the need for spectral filtering of the emission. More complex assays involving fluorescence resonance energy transfer (FRET) (*53*) or fluorescence lifetime imaging (FLIM) (*54*) are now routinely used to screen rapid dynamic processes occurring within picoliter-volume droplets. Significantly, FRET allows the distance between sites on macromolecules to be estimated by utilizing dipole–dipole interactions between an excited electronic state of a donor fluorophore and the ground state of a proximate acceptor chromophore, with the rate of energy transfer being directly related to the donor–acceptor separation (*53*). In this respect, Srisa-Art and coworkers (*55*) successfully employed FRET to study the binding kinetics between streptavidin and biotin within microfluidic droplets on millisecond time scales. Additionally, FLIM allows the extraction of fluorescence lifetimes with high spatial resolution and provides an absolute measurement of molecular concentrations, which when compared to time-integrated signals (intensities), is far less susceptible to artifacts, such as non-uniform illumination, scattered light, and excitation intensity variations. Unsurprisingly, FLIM has been used to characterize mixing dynamics in microfluidic droplets on microsecond time scales (*56*) and is facilitated by the highly controlled and reproducible nature of droplet flows. Interestingly, a combination of FRET and FLIM has also been successfully used to quantify molecular binding in microfluidic droplets, leading to significant reduc-

tions in both sample volumes and assay times when compared to bulk measurements (*57*).

1.2.2 *Droplet Barcoding*

The ability to identify a single droplet within a large population is central to many of the applications described herein. For relatively small populations, this can be achieved by ordering droplets in a linear fashion (*58*), trapping droplets at specific positions (*59*) or by labeling droplets with identifiable probes (barcodes) (*60*). Sequential ordering can directly be achieved by storing droplets inside an extended channel segment or length of tubing. Such an arrangement allows droplets to be re-read and identified according to their position relative to the sequence termini (*58*). Sequential ordering, however, does not scale to large droplet populations, as associated backpressures increase rapidly with the flow path (*39*). In a similar manner, immobilization of droplets in stationary traps (within larger chambers) does permit the repeated analysis of individual droplets (*59*, *61*, *62*), but again, it is limited to a few thousand traps at most. Due to these limitations, several techniques that enable the labeling (or barcoding) of individual droplets have been reported in recent years. The most common and straightforward method uses the coencapsulation of multiple fluorophores (leveraging variations in the relative flow rates of the input streams) to spectrally encode droplets and generate uniquely identifiable signatures (*63*, *64*). However, most studies to date have only managed to produce limited numbers (< 100) of discrete barcodes and typically generate an unacceptably large number of duplicates (*63*, *64*). Conversely, several recent studies have employed nucleic acid barcodes to label up several million droplets (*60*). Nevertheless, it should be noted that these methods exclusively work with assays where the readout is based on nucleic acids sequence analysis. Furthermore, it is also noted that nucleic acid sequencing is a destructive process (*60*, *65*, *66*).

To address the limitations associated with barcoding large droplet populations, we recently developed a programmable system able to generate large populations of droplets, with each droplet containing a uniquely identifiable barcode. This is achieved by combining the generation of populations of labelled droplets (containing varying concentrations of fluorescent dyes) with a barcode reading/sorting technique that captures unique droplets as soon as they are produced. Proof-of-principle experiments have successfully demonstrated the generation of droplet populations contain-

ing over two thousand droplets with unique fluorescent barcodes using only two fluorophores. A detailed description of the approach and associated studies is presented in Chapter 3.

1.3 NEURAL NETWORK BASED CONTROL

Traditional microfluidic experiments often require extensive manual intervention to maintain operational stability over the extended timescales associated with large-scale experimentation. Accordingly, there is a need for purpose-built control algorithms that enable real-time processing of large amounts of experimental data. In this respect, recent advances in machine learning, and specifically the development of algorithms based on artificial neural networks (or ANNs) (*67*), represent general purpose tools, which can readily be applied to a variety of microfluidic control problems. Such algorithms should ideally allow the performance of high-throughput independent experimentation through the efficient control of the microfluidic environment, based on real-time observations.

ANNs are algorithms inspired by biological neural networks, where activation of series of interconnected neurons defines a recognizable linear pathway (*67*). Typically, ANNs are made up of a collection of connected units (artificial neurons), where connections between neurons are weighted. Thus, when an artificial neuron is activated (receives a non-zero, scalar signal) it will propagate the received signal to all connected neurons downstream. The weighted connections will amplify or dampen the corresponding signal during propagation. In a standard configuration, the artificial neurons are organized into layers, with data travelling from an input layer, through a series of hidden layers, to an output layer. The data read back from the output layer corresponds to the desired result and is mainly dependent on the weights of the ANN. Correct results are typically achieved by gradually adjusting the weights within the ANN using a dataset of known input and output signals. The weights of the connections between artificial neurons are then successively adapted to ensure the correct output signals, often via a process called *back-propagation* (*68*). Due to their generalized formulation, ANNs have been used for a diversity of data transformation tasks, including image pattern recognition (*69*), speech synthesis (*70*) and machine translation (*71*).

When first developed in the 1950s, ANNs consisted of a few hundred artificial neurons (*67*). Nowadays, modern ANNs contain millions of artificial neurons and neuronal connections, enabling them to be effectively used

in a range of complex scenarios (*69*). Such an increase in complexity has been possible through improved network topologies (connection schemes), network training strategies and utilization of the vastly increased computational power of modern computers. Indeed, most recently, the inherent parallel processing power of graphical processing units (or GPUs) has been harnessed by ANNs to excellent effect, due to the ability to perform extremely fast matrix operations (*72*). To this end, we have used supervised image classification techniques to achieve the rapid and on-line sorting of micron-sized objects within microfluidic devices. Specifically, this was achieved by initially training an ANN to classify various cell types and beads offline. The trained ANN was then used to perform online sorting of such objects at rates of up to 20 hertz. Critically, such an approach provides for greater flexibility and lower operational costs than traditional fluorescence activated cell sorting (FACS) techniques, due to the ability to interpret images rather than one-dimensional data. These studies are described in detail in Chapter 5.

In contrast to supervised learning methods, where an algorithm improves its prediction based on a known dataset, algorithms based on unsupervised learning (*73*) or reinforcement learning (RL) (*74*) have been developed to master situations, where there is no dataset of known input-output pairs. Instead, RL-based algorithms repeatedly interact with an environment and incrementally improve performance in the process. Thus, RL differs from supervised learning in that the "correct" answer to an input is not known initially, with learning being achieved by iteratively improving the correctness of the result based on a scalar reward signal obtained from the environment. For example, a RL-based algorithm for temperature control might receive positive reinforcement if the correct temperature is maintained, and will learn over time how to manipulate the system to maximize such positive rewards.

Recently, many-layered ANNs have been combined with reinforcement learning (for example Deep Q-network, DQN) to interpret high-dimensionality data (such as visual inputs) and deduce optimal actions to perform in the observed environment (*75*). Significantly, it was shown that DQN can surpass human performance in a variety of games, including Atari games (*76*) and Go (*77*) (see Figure 1.3).

Nevertheless, only few applications of reinforcement learning in non-simulated environments have been shown thus far, mainly due to difficulties obtaining input data and exerting tight control over the environment. Examples of reinforcement learning in non-simulated environments

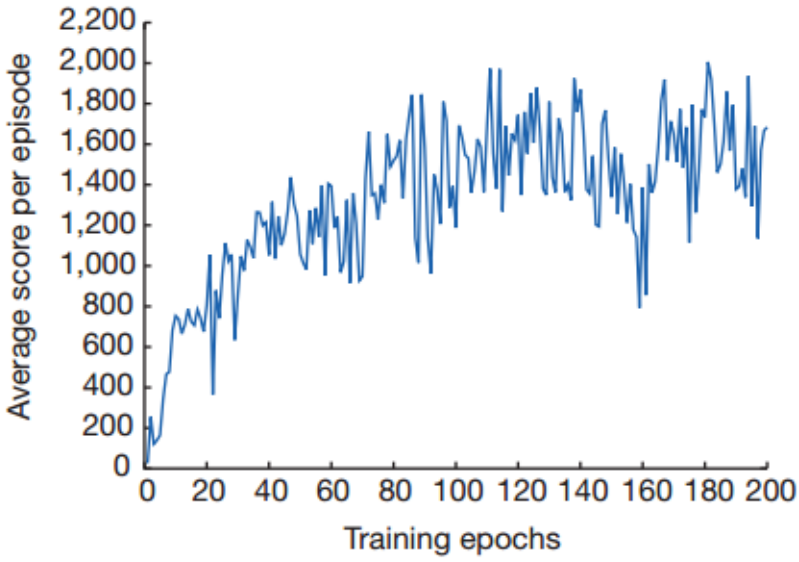

FIGURE 1.3: Training curves tracking DQN average score when playing Space Invaders, a game for the Atari 2600 video game platform (*75*). Each point is the average score achieved per episode. Human testers scored on average 1692 points. It can be seen that the agent increases performance during training and eventually reaches human-level performance.

include the control of robotic arms (*78*) as well as the control of building air conditioning systems (*79*). To this end, we have utilized reinforcement learning to achieve automated, on-line microfluidic experimentation. Our algorithm is provided with bright-field observations of a microfluidic device obtained through an optical microscope. The algorithm can control the flow rates of input flows with a view to influencing the microfluidic environment exclusively. By using such an approach, we demonstrate the optimization of various flow control problems, involving both segmented and laminar flows, and highlight the general-purpose applicability of the presented architecture. These studies, as well as an extended introduction to ANNs and machine learning in general are described in Chapter 4.

1.4 CONCLUSIONS

Since the first proof-of-concept experiments (*12–14*), droplet-based microfluidic technologies have become an increasingly important and powerful tool in chemical and biological experimentation. Significantly, a large variety of droplet-based platforms and tools are now being used in a routine manner by researchers from a wide range of scientific disciplines. Despite this growing popularity, droplet-based techniques have found a more limited application in the study of dynamic systems, primarily due to the lack of appropriate methodologies to label and identify large numbers of individual droplets and the difficulties associated with controlling microfluidic systems at sufficiently high frequencies to enable responsive high-throughput experimentation.

That said, the studies and innovations described herein clearly suggest that a diversity of methods are now available for the generation, manipulation, and detection of droplets. Moreover, these core technologies are being increasingly applied to the study of chemical and biological processes that were hitherto inaccessible, with some remarkable successes being reported in the fields of single-cell sequencing (*60*, *65*, *66*) and synthetic biology. Nonetheless, many challenges (and indeed opportunities) await as droplet-based microfluidic systems transition from proof-of-concept platforms to integrated and commercialized instruments.

2

PASSIVE SYNCHRONIZATION OF MICROFLUIDIC DROPLETS

2.1 INTRODUCTION

Enclosing droplets (that contain reagents for an assay or reaction) within an immiscible carrier fluid allows for precise control of the reactive process in both time and space. Under certain circumstances, each droplet may be considered to be an isolated reaction vessel (*12*) separated from the other droplets (and the walls of the microfluidic channel) by the carrier fluid, greatly reducing cross-contamination between samples (*62*). Typical microfluidic architectures for droplet generation produce droplets having volumes in the picoliter to nanoliter range and at generation frequencies well in excess of 1 kilohertz (*51*, *80*). Several chip-based architectures for the formation of highly monodisperse droplets have been demonstrated and refined over recent years. These include T-junctions (where droplet formation is induced by a pressure drop) (*12*, *81*), V-junctions (that support a wider combination of droplet sizes, spacings and generation frequencies) (*15*) and flow-focusing junctions (where droplet formation is induced by capillary number related instabilities) (*32*). All of these strategies exploit shear forces present at the interface between two immiscible phases to split a continuous fluid flow into separate droplets without the requirement for any external forces such as magnetic, ultrasound or electrical fields (*82*). Typically, droplet size, shape and formation rates can be controlled by factors such as input flow rates, interfacial surface tension and the viscosities of the two immiscible phases used. Other important factors include the channel geometry and channel surface properties. Finally, and of paramount importance, it is noted that surfactants are almost always used to stabilize droplets subsequent to formation (*23*, *83*).

In recent years there has been significant interest in performing complex biological assays using droplet-based microfluidic systems (*9*, *10*, *84*,

Tianjin Yang has contributed to synchronization architecture development. Michael Ehrenstein and Adrian Müller have aided in data collection for inter-droplet transfer. Parts of this chapter have been published in (*39*).

85). Such assays almost always require the sequential performance of defined unit operations (*84*). For example, droplets may be split (*86*, *87*), trapped (*88*), diluted (*41*) and incubated (*89*) over extended periods of time (*23*, *89*–*92*). Sorting of droplets based on content (*45*, *82*, *93*, *94*) or morphology (*95*) can be realized using a range of strategies, and droplet merging (an essential step in the implementation of almost any assay) can be achieved both passively and actively (*35*, *96*–*98*). Moreover, it is also important to note that effective methods for pairing droplets in a sequential (A-B) fashion are very useful when probing inter-droplet transfer (*99*), contacting droplets using shift registers (*100*) or initiating cell fusion (*101*).

The functional components described above have been used in a wide range of applications, including the assessment of ultra-fast reactions (*102*), large-scale single-cell genomics (*66*), long-term manipulation and screening of microbial populations (*103*), creation of a droplet populations with combinatorial chemical probes (*104*) as well as self-assembly of droplets into complex three-dimensional structures (*105*). In all these studies, the combination of individual toolbox functions to create a process or assay requires precise control over the spatial and temporal ordering of individual droplets to ensure the correct workflow and maintain experimental reproducibility. In this respect, droplet synchronization can be achieved using a range of active architectures, where each droplet is addressed and manipulated on an individual basis. For example, dosing reagents into droplets can be achieved using pico-injectors (*41*) or on-chip mechanical valving (*106*). Active dosing can be performed at high speeds but requires both complex control architecture and in-line process monitoring (*41*, *106*). Conversely, passive droplet synchronization methods provide the experimenter with control over droplet ordering whilst requiring minimal external control. The simplest mode of passive synchronization arranges two types of droplets into an alternating sequence prior to a droplet merging architecture (*106*–*109*). In such a system it is crucial to minimize synchronization errors so as to ensure a constant composition of the merged droplet stream.

Streams of alternating (A-B) droplets have previously been produced using coupled droplet formation schemes. For example, combined T-junctions can be used to form droplets in parallel (*110*–*113*). Interestingly, Frenz and co-workers utilized a similar approach to form A-B droplet streams with error rates as low as 1 in a million droplets (*114*). A related approach described by Hong et al. coupled two droplet forming junctions using a passive pressure oscillator to achieve a similar level of synchronization (*108*).

Finally, step emulsification can also be used to form streams of alternating droplets (*115*). In step emulsification the dispersed phase is injected through a small channel into a higher channel filled with the continuous phase. When two step-emulsification junctions are placed in close proximity and are fluidically close-coupled the interplay leads to the formation of A-B droplet streams (*18*). Although a number of approaches for forming A-B droplet streams have been reported, the operation of more than two droplet forming junctions in parallel usually leads to large disparities in both droplet size and formation frequency (*81*, *116*). Accordingly, previous approaches based on controlling droplet formation have been limited in terms of the maximal distance between the point of droplet formation and the point of merging, and thus do not allow subsequent merging events to be performed in a controllable manner.

Given that droplet merging is only one of a series of operations that may need to be performed within an assay, the inability to couple pre-formed droplets in a high-throughput manner is a significant drawback. Consequently, an optimal droplet synchronization architecture should be able to process pre-formed droplets, such that droplet formation is handled independently from downstream operations. Several studies have shown synchronization of pre-formed droplets or bubbles. For example Prakash and Gershenfeld (see Figure 2.1) demonstrated the removal of timing errors between two streams of gas bubbles using a fluidic ladder (*38*, *117*). In this case, a bubble traversing the ladder is slowed down by diversion of the oil flow through the alternate path. When two bubbles are present simultaneously, there is a net flow from the channel containing the leading bubble to the one containing the lagging bubble, creating a velocity gradient that disappears as the bubbles synchronize. This elegant concept has been adapted for use with droplets, but it was found that accurate synchronization requires precise control over both droplet size and formation rate (*110*).

If droplets are formed using a relatively low oil fraction, spontaneous pattern formation can be observed (*33*). Several studies have used variations of such self-ordering behavior to create specific droplet configurations. For example A-B patterns have been formed using a single T-junction or multiple T-junctions in parallel (*81*). Step emulsification of two droplet-forming phases into a single channel can also be used to create a densely-packed "zig-zag" arrangement (*115*). Further, oil removal from a stream of pre-formed droplets leading to a more densely packed droplet stream can be achieved using a simple pillar array (*35*). Finally, Surenjav and co-

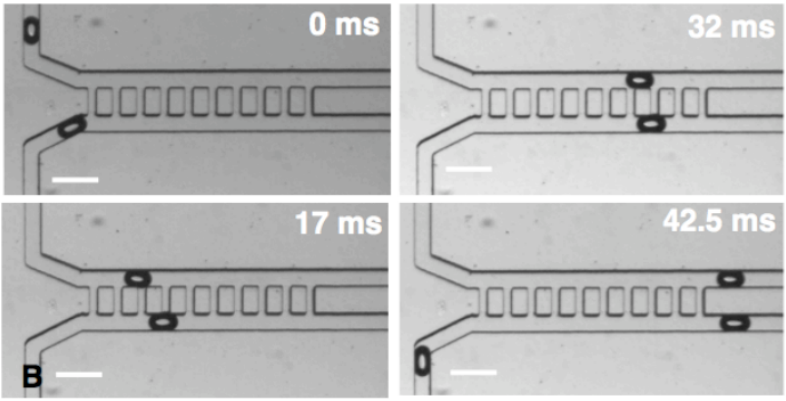

FIGURE 2.1: Bubble synchronization as shown by Prakash and coworkers (*38*) (scale bar - 100 µm). Initially, droplets enter the synchronization structure (from the left) with a significant timing error. During passage, the ladder-like synchronization structure diverts continuous phase from the channel containing the leading bubble to the channel with the lagging bubble due to an inbalance of hydrodynamic resistance (*31*). This allows the lagging bubble to catch up during passage, resulting in a synchronized bubble pair exiting the structure. Using this synchronization principle, timing errors of up to 10 ms could be restored over a span of 40 ms.

authors have shown that different patterns can be formed and manipulated by varying both the channel geometry and droplet size (*118*).

In practice the frequency of droplet formation will fluctuate over time due to pressure and flow rate instabilities. Such fluctuations increase over longer time scales, primarily due to droplets "catching up" with each other, which leads to groups of droplets traveling together rather than as a regular, dispersed stream. It is therefore essential that a low-error droplet synchronizer be able to compensate for variations in both droplet frequency and size. Indeed, without such a buffering operation, synchronization efficiency will be directly dependent on the formation frequency. Leveraging the fact that smaller droplets travel at higher speeds than larger ones, Mazutis and co-workers merged two streams of heterogeneous droplets using a zig-zag channel and locally lowered surfactant concentrations, which triggered the droplet merging process through interface destabilization (*90*). This approach could be used to synchronize droplets even if an excess of smaller droplets is present. Nevertheless, such an approach is limited to droplets of different size and only the smaller droplets can be present in excess.

To address the aforementioned inadequacies we present herein, a structure for the continuous and passive synchronization of pre-formed droplets with very low error rates. Our approach involves densely packing two types of droplets and co-injecting them into a channel to form an ordered A-B alternate pattern. This pattern is then rearranged into a single stream of alternating droplets. The high error-tolerance results as a consequence of a buffer structure that allows the removal of excess droplets during the packing process.

2.2 EXPERIMENTAL METHODS

2.2.1 *Microfluidic Device Fabrication*

Microfluidic devices were fabricated using conventional soft lithographic methods in polydimethylsiloxane (PDMS) (*119*). Initially, microfluidic geometries were designed using AutoCAD 2014 (Autodesk GmbH, Munich, Germany) and subsequently printed onto high-resolution film masks (Micro Lithography Services Ltd, Chelmsford, UK). In a cleanroom environment, a silicon waver (Si-Mat, Kaufering, Germany) was spin-coated with a layer of SU-8 2050 photoresist (MicroChem, Westborough, USA) and subsequently exposed using a collimated UV source. After development

using SU-8 developer (MicroChem, Westborough, USA), the fabricated master mold was characterized using a confocal 3D laser scanning microscope (VK-X, Keyence, Neu-Isenburg, Germany). The Sylgard 184 PDMS base and curing agent (Dow Corning, Midland, USA) were mixed at a ratio of 10:1 *wt/wt*, degassed and decanted onto the master mold. The entire structure was subsequently cured in the oven at 70 °C for at least 8 hours and then separated by peeling. After punching inlet and outlet ports through the structured PDMS layer, the PDMS was bonded to another flat PDMS substrate using oxygen plasma and incubated on a hot plate at 95 °C for at least 2 hours to complete the bonding process. Finally, a hydrophobic surface treatment solution, 5 % *v/v* 1H-1H-2H-2H-perfluorooctyltrichlorosilane (PFOS; abcr GmbH, Karlsruhe, Germany) in isopropyl alcohol (Sigma-Aldrich, Buchs, Switzerland), was applied for one minute to ensure hydrophobicity of the channel surface.

2.2.2 *Synchronization Performance*

A mixture of FC-40 (3M, St. Paul, USA) and 4 % *wt/wt* EA-surfactant (RainDance Technologies, Billerica, USA) was used as the continuous phase for all experiments in this chapter. Ink (Waterman, Paris, France) diluted with deionized water (10 % *v/v*) was used to form the dispersed phase. neMESYS low pressure dosing modules (Cetoni GmbH, Korbussen, Germany) were used to pump fluids using 1 mL gastight syringes (Hamilton Bonaduz AG, Bonaduz, Switzerland).

A MotionPro Y5 Compact Digital Camera (IDT, Hitchin, United Kingdom) was used to image the passage of droplets through the microfluidic system at 200 frames per second (fps). A time series of the average pixel intensity in a defined region of interest was extracted from the image data using ImageJ (National Institutes of Health, Bethesda, USA) and further processed using custom-written Python scripts (Python Software Foundation, Beaverton, USA). Specifically, a threshold to differentiate between different colored droplets was selected manually and packet lengths calculated from the time between transits over this threshold value. Using a histogram of packet lengths, the number of droplets per packet was then mapped to packet length. Synchronization performance (defined as the percentage of error-free alternating droplets) was determined by iterating over all packets in a series and counting the number of packets where both the packet itself and the next packet consist of a single droplet.

2.2.3 *Inter-droplet Osmotic Transfer*

Inter-droplet transfer experiments were performed using a device adapted from the long-term device (see Figure 2.10). Specifically, we reduced the distance between separate sections of the meandering channel, to increase the number of droplets visible within one field of view. Osmosis experiments quantified transfer between droplets containing distilled water and droplets containing aqueous solutions with varying concentrations of NaCl (Sigma-Aldrich, Buchs, Switzerland). Proton transfer was quantified using aqueous droplets containing HCl (Sigma-Aldrich, Buchs, Switzerland) at pH 0.5 and droplets containing 100 µM Fluorescein (Sigma-Aldrich, Buchs, Switzerland) solution. Transfer rates were quantified from videos, captured during operation of the device using the aforementioned high-speed camera at 16 hertz. Finally, we used custom Python scripts to extract radii and fluorescence intensity from the raw data and subsequently visualizing the results.

2.3 SYNCHRONIZATION DEVICE DEVELOPMENT

The initial synchronization device design was based on the synchronization principles introduced by Prakash and co-workers, where ladder-like connections between two parallel channels are used to resolve timing errors between bubbles flowing along these channels (*38*). As previously noted, in such a structure the leading bubble (or droplet) is slowed down relative to a following bubble, since its presence in the microfluidic channel increases the hydrodynamic resistance within that channel (*31*). The increase in hydrodynamic resistance results in the diversion of a portion of the continuous phase into the neighboring channel, thereby reducing the velocity of the leading bubble. By design, diversion of the continuous phase is only possible while one bubble moves ahead of the other. As soon as both bubbles occupy the same channel segment, hydrodynamic resistance is equalized and both bubbles will move at the same velocity. Figure 2.2 a illustrates a schematic of the initial device design, with the image in Figure 2.2 b showing the device during operation, where two droplet streams are separated (and connected) by a ladder-like array. The geometry consists of two parallel channels of 50 µm diameter and connecting channels of 30 µm width and was operated using 50 µm diameter droplets. Channel dimensions were chosen to prevent droplets from transiting be-

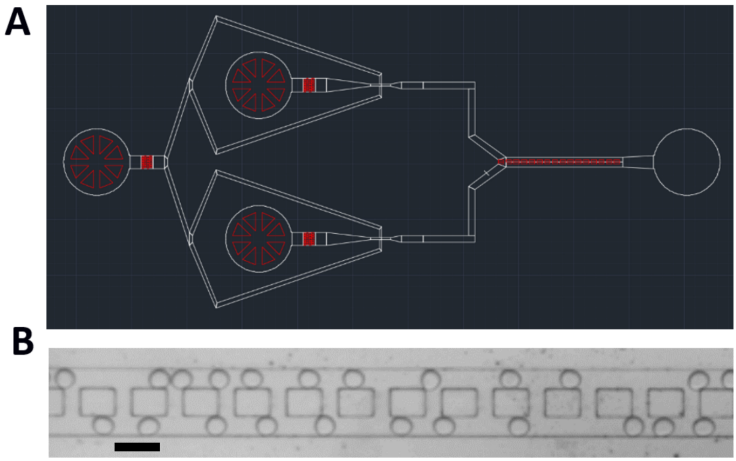

FIGURE 2.2: Synchronization architecture, based on the synchronization principle shown in (*117*) (scale bar - 100 µm). Two parallel main channels of 50 µm width are used to synchronized droplets entering the geometry. The smaller, ladder like connections (30 µm diameter) between the droplet-containing channels allow continuous phase to be diverted along the hydrodynamic pressure gradient between the two main channels. This allows lagging droplets to "catch-up" by slowing down the leading droplet, thereby reducing timing errors between droplet pairs. (A) Schematic of the synchronization device. (B) Image of the device during operation. It proofed challenging to achieve sufficiently large inter-droplet spacings without the introduction of additional continuous phase.

tween the two main channels while still allowing for sufficient transfer of continuous phase between the channels.

Unfortunately, we could not achieve proper droplet synchronization using this approach. If a large number of droplets are contained within the synchronization structure at the same time (see Figure 2.2 b) synchronization efficiency is significantly diminished. Specifically, we could not achieve a low enough droplet production frequency using standard flow-focussing junctions. In general terms, diversion of continuous phase (required for the synchronization of a droplet pair) negatively affects timing errors between adjacent droplet pairs. To mitigate this effect, droplets should enter the synchronization geometry in a sufficiently spaced manner. Therefore, to handle decreased inter-droplet distances larger synchroniza-

tion structures are required, which complicates microfluidic experimentation through increased hydrodynamic resistance and compounding fabrication errors in long channels. Moreover, we found that the synchronization efficiency was highly dependent on the frequency that droplets enter the synchronization structure, since the ladder rectifies timing errors but does not discard excess droplets. This is particularly problematic in cases when two different types of droplets are synchronized (the most likely situation in biological assays), and requires that both droplet types be produced or re-injected into the structure at precisely equal rates.

To enable synchronization of droplets entering the structure at unequal rates, we included a droplet overflow channel. Figure 2.3 highlights a subset of various buffering and overflow structures that were tested within the current work. Specifically, control over droplet flow is achieved by utilizing a filter (or pillar-array) that is able to retain the majority of droplets while allowing a large fraction of the continuous phase to pass through unhindered. Moreover, addition of a droplet buffer region allows efficient droplet trapping, filling up the chamber before the overflow is required. Accordingly, we were able to capture a large proportion of droplets using the buffer structure, whilst passively discarding excess droplets through the overflow channel in a controlled manner. This simple approach means that droplets enter the synchronization architecture at a constant rate. Critically, the combination of the buffer and the overflow channel allows the architecture to compensate for large differences in droplet production frequencies.

Figure 2.3 a highlights the initial droplet capture chamber design. We use a pillar array to retain droplets while letting the continuous phase pass through unhindered to achieve a packed droplet configuration. In the initial design, droplets entering the main chamber encounter a circular pillar, which retains droplets at the center of the chamber, increasing local droplet concentration. Crucially, the distance between the pillar array and the chamber wall was chosen to be one droplet diameter (100 µm) to achieve maximum capture efficiency. While the structure captured a sufficient share of droplets (exceeding 60 %) we could observe significant droplet splitting at the pillar array. Specifically at the end of the circular pillar array structure we could observe an accumulation of droplets due to its orientation perpendicular to the droplet flow direction. This resulted in droplets being pushed through the gaps of the pillar array and splitting in the process. To reduce splitting at the perpendicular pillar array we designed a pillar array with decreased curvature (see Figure 2.3 b). While

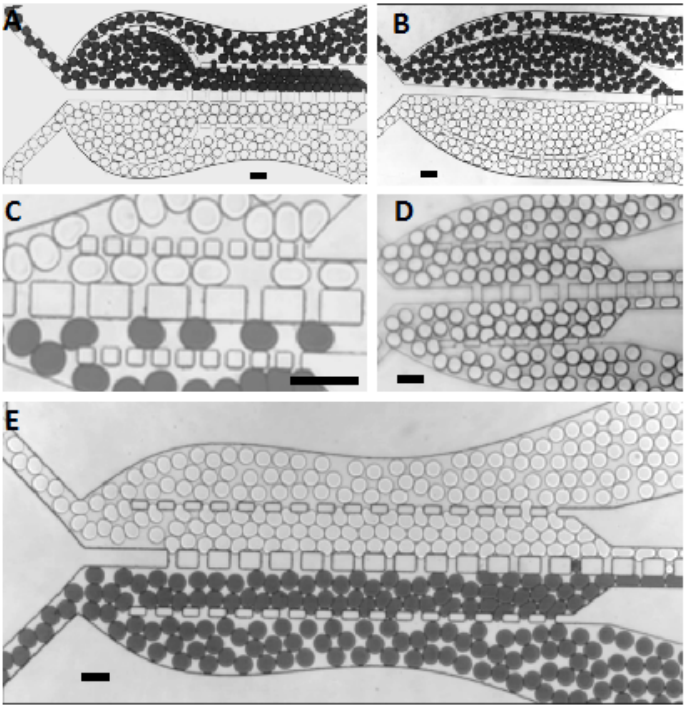
A
B
C
D
E

FIGURE 2.3 *(previous page)*: Droplet buffer and overflow structure design (scale bars - 200 µm). Droplets (flowing left to right) are captured from the incoming stream using a pillar array. This optimally results in a completely filled capture region before any droplets are discarded. Once the droplet buffer is filled, additional droplets are discarded at the top of the buffer region. (A) As droplets enter the main chamber they encounter a circular pillar array. The design was scaled based on an overflow channel (between the pillar array and the chamber wall) of exactly one droplet diameter (100 µm). It could be observed that the structure captures sufficient droplets. However, the end of the circular pillar array is almost perpendicular to the flow direction. This lead to droplet accumulation at the pillar array and frequent droplet splitting. (B) Capture chamber with a reduced pillar array curvature. While this design eliminated excess droplet splitting it also exhibited insufficient droplet capture efficiency.(C) Experimental design for a minimal footprint capture chamber using straight pillar arrays for capture. While this chamber design exhibited similarly low capture rates (below 50 %) we chose to improve on this design due to the elimination of droplet splitting. (D) Device design showing improved capture efficiency while preventing droplet splitting at the pillar array. (E) Selected capture geometry, allowing for densely packed droplet configurations at the chamber entrance while preventing any merging or splitting at the outer droplet capturing pillar arrays.

we could indeed observe reduced droplet splitting at the pillar array the design exhibited insufficient droplet capture efficiency (below 30 %).

To completely eliminate droplet splitting at the pillar array we successfully tested straight pillar arrays (see Figure 2.3 c). While this device exhibited no droplet splitting we could still not achieve sufficient droplet capture efficiency. However, using successive designs (see Figure 2.3 d,e) we found that capture efficiency could be significantly increased by enlarging the capture chamber size. The final design (Figure 2.3 e) was able to achieve packed droplet configurations continuously while showing no droplet splitting at the straight pillar array.

During testing of previous device designs it became evident that the gaps between capture chambers allow a significant number of droplets to pass between the two chambers, thereby limiting achievable synchronization efficiencies. Further, an experimental design with a reduced number of gaps (see Figure 2.3 b) indicated that a separation between both chambers did not have negative effects on the flow balance if reasonably similar flow rates are used for both droplet generators. Accordingly, we used subsequent designs to systematically reduce the number of gaps between the two capture chambers (3, 2, 1 and zero gaps for devices shown in Figure 2.4 a-d respectively). As a result, the design shown in Figure 2.4 d was adopted for further use, since elimination of all gaps did not significantly impact the ability to compensate for flow rate differences, whilst completely preventing transfer of droplets between the two capture chambers.

The device shown in Figure 2.4 d could be used to achieve tightly packed droplet configurations. However, the synchronization principle as reported by (*38*) requires large spacings between droplets to achieve efficient synchronization. Sufficient spacing in turn requires additional microfluidic channels and extensive flow balancing. Accordingly, we chose to eliminate the ladder-like structure altogether and utilize geometric self-ordering of droplets. The resulting device (see Figure 2.5) exhibited exceptional synchronization efficiency, and could not be further improved on in subsequent iterations.

2.4 LOW-ERROR SYNCHRONIZATION DEVICE

Figure 2.5 a shows a schematic of the final passive microfluidic droplet synchronization structure. The primary components include two flow-focusing droplet generators and the synchronization architecture (see Figure 2.5 b), which consists of a packing chamber, a co-injection channel and a constric-

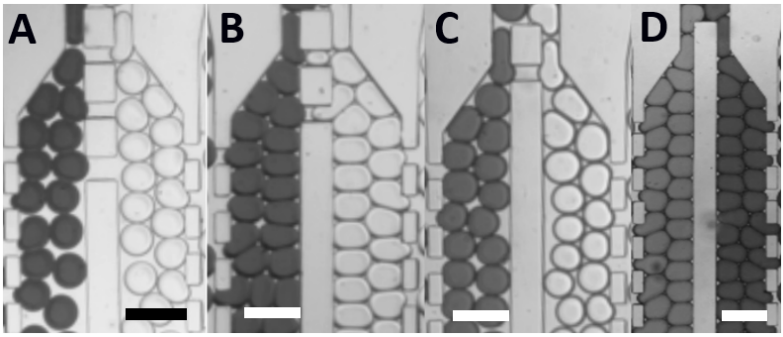

FIGURE 2.4: Variation of the number of gaps between the droplet capture chambers (scale bars - 200 μm). Gaps between the chambers were hypothesized to aid in balancing flow rates through the retention of droplets, whilst allowing the flow of the continuous phase. In reality, droplets were often observed to pass through or get stuck in these gaps (as seen in panel B). Accordingly, the chamber design without any gaps, shown in panel D, was adopted for future experiments.

tion that moves synchronized droplets into an alternating order. As discusse above, we also included a path for droplets to exit the packing chamber when the chamber becomes completely full (overflow mode). Such an overflow capability reduces the impact of any errors in the upstream droplet formation process and ensures stable droplet co-injection into the common channel. It should also be noted that the two "Out 2" channels were only recombined to aid device fabrication and we have successfully operated devices with distinct outlets, allowing for independent control of back-pressure. Figure 2.5 c presents an image of the synchronization architecture in operation.

Figure 2.6 shows a series of images that follows a set of four droplets as they pass through the synchronization architecture. After removal of most of the spacer oil (upstream of the field of view), each input channel is constricted so that droplets are forced to move in a single file. This results in the droplets fully occupying the main synchronization channel in a zigzag configuration. If the width of this channel is further reduced, the two rows of droplets collapse into a single file of alternating droplets. To investigate the performance of the synchronization architecture, we assessed the synchronization efficiency for a variety of droplet sizes and total flow rates.

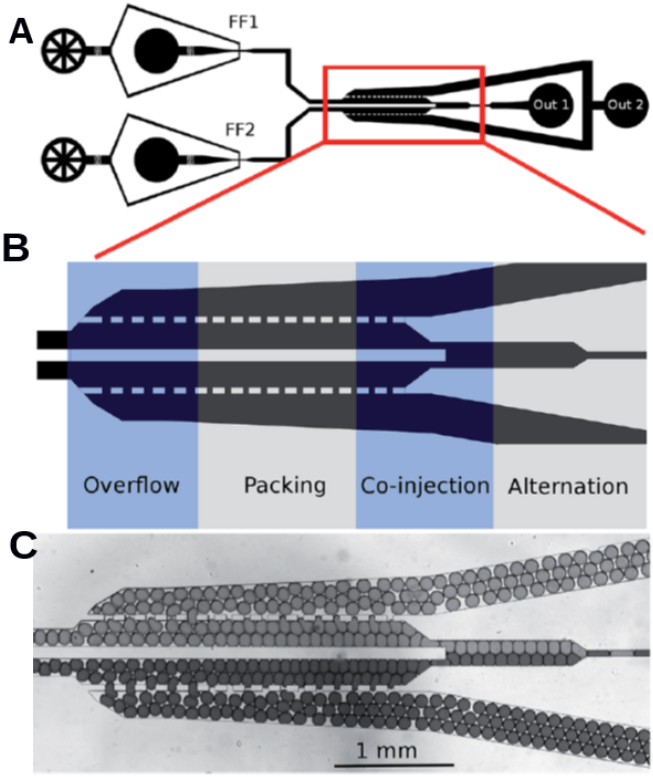

FIGURE 2.5: Architecture of the final synchronization device (scale bar - 1 mm). (A) The complete device includes two independent flow-focusing junctions for droplet formation and a droplet synchronization chamber. (B) Schematic of the droplet synchronization chamber. Droplets are able to exit readily through the 75 µm gaps between the pillars within the overflow section. Droplets are further compacted in the packing section, through drainage of the continuous phase. Droplets are then alternately injected into a common channel that narrows to push the droplets forward. (C) Image of the microfluidic device in operation.

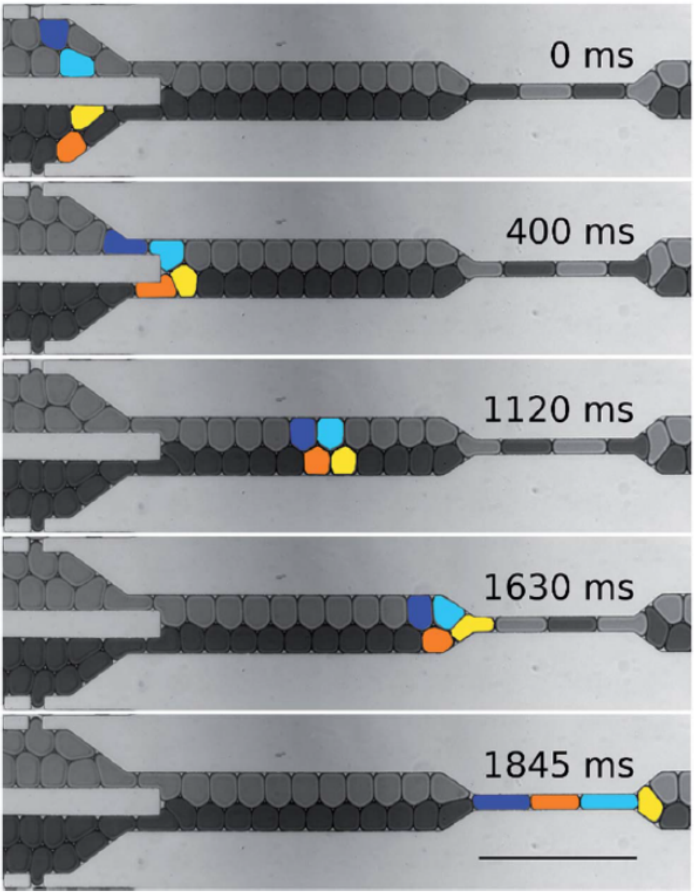

FIGURE 2.6: A series of images extracted from a high-speed video showing a set of droplets moving through the synchronization architecture (scale bar - 600 µm). Droplets are passively ordered into an A-B alternating pattern when they are injected into the common channel (after 400 ms). When this channel is constricted, so as to only allow a single file of droplets (after 1600 ms), the two rows of droplets are impelled into an A-B sequential order.

2.5 RESULTS AND DISCUSSION

2.5.1 *Synchronization Performance Characterization*

Figure 2.7 a reports the droplet synchronization efficiency of the final device for a range of droplet sizes, with Figure 2.7 b illustrating the variability in synchronization performance across all data points (N = 3, with an average of 2000 synchronized droplets per experiment). Inspection of Figure 2.7 a indicates that the current device performs optimally within a defined region of phase space, perfectly synchronizing two equally-sized droplet populations with a diameter of 126 µm. Since the synchronization channel has a width of 220 µm, optimal droplet diameters represent 57 % of this width, with two rows of droplets fully occupying the main channel in a side-by-side arrangement. Additionally, variability data provided in Figure 2.7 b confirm that quoted synchronization efficiencies could be obtained repeatedly and reliably. Finally, it can be seen that synchronization performance (as well as reliability) falls dramatically when both droplets have a diameter smaller than 107 µm (49 %).

The key concept behind the presented architecture is the co-injection of densely packed droplets into a common channel. Since droplets are deformed prior to entering the common channel, a significant increase in hydrodynamic resistance is generated through Laplace pressure build-up. This pressure increase leads to a delay of the following droplet, favouring the passage of droplets entering from the other channel. We hypothesize, that this results in droplets being injected into the common channel in an alternating fashion. After injection, these droplets rearrange into an A-B alternate pattern through minimization of surface area. Once a stable A-B alternate conformation is achieved, droplets can be collapsed into an alternating order through a reduction in the channel width.

Given the need to pack droplets in an A-B configuration, it is apparent that the performance of the synchronization architecture depends on both device geometry and droplet size. It is to be expected that the optimal diameter of synchronized droplets should be marginally larger than half the width of the common channel, due to the need to densely pack two rows of droplets into this channel. Data in Figure 2.7 confirms that optimal synchronization could indeed be achieved using equally sized droplets with a diameter of 126 µm, which is 57 % of the width of the 220 µm common channel). Additionally, reliable synchronization could also be achieved with droplets of different sizes (with up to a 15 % difference

A

107 µm	81.04					
110 µm	83.29	86.13				
118 µm	88.34	88.41	93.49			
126 µm	92.00	99.41	97.78	100.00		
137 µm	87.74	80.80	81.80	99.85	97.30	
150 µm	83.92	75.14	71.76	95.74	95.41	89.26
	107 µm	110 µm	118 µm	126 µm	137 µm	150 µm

B

107 µm	25.97					
110 µm	24.75	20.32				
118 µm	12.78	17.83	9.44			
126 µm	9.55	0.88	0.30	0.01		
137 µm	8.75	8.53	16.09	0.26	3.55	
150 µm	5.66	6.58	10.56	6.46	6.27	12.06
	107 µm	110 µm	118 µm	126 µm	137 µm	150 µm

FIGURE 2.7: (A) Dependence of synchronization performance (quantifying the percentage of correctly synchronized droplets) on droplet diameter. Each tabulated value represents the mean synchronization performance from three independent experiments. Darker shadings correspond to improved synchronization performance. Optimal synchronization was achieved when two droplet types of equal size (126 µm diameter) were synchronized. This corresponds to a droplet diameter of around 57 % of the total channel width. The reported value of 100.00 % synchronization performance is rounded to two decimal places from a measured value of 99.997 %. (B) Standard deviation of the measured synchronization performance. Darker shadings correspond to a lower standard deviation.

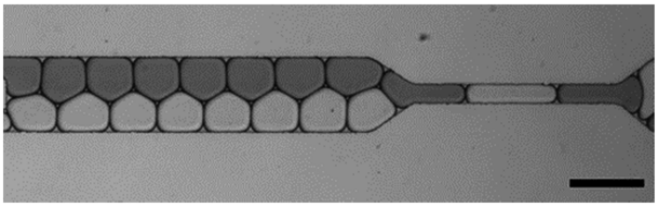

FIGURE 2.8: Synchronization of droplets with diameters of up to 150 µm (scale bar - 300 µm). Application of vacuum to the output allows for more accurate and faster synchronization. The device shown is a directly scaled up version of the previously described design.

from the optimal diameter) as long as the sum of both droplet diameters is slightly larger than the channel diameter. Significantly, rearrangement of the ordered droplets allows the synchronizing architecture to compensate for variations in droplet size. Although the presented synchronization architecture was optimized for a set range of droplet diameters, it is noted that larger droplets were more challenging to synchronize using the current architecture due to droplet splitting at the pillar array during the packing phase. Nevertheless, modifications to the pillar array dimensions have been shown to be successful in allowing the synchronization of droplets with diameters up to 150 µm (see Figure 2.8).

Another key factor in ensuring reliable droplet synchronization is the ability to allow for droplet overflow, which acts to compensate for potential variations in droplet generation frequency. It was observed that under optimal conditions the device synchronizes approximately 17 % of incoming droplets, with 83 % of droplets being discarded. However, these rates were determined after densely packing the synchronization chambers with droplets and initial capture rates (for empty) are significantly higher. Finally, it should be noted that overflow droplets exit the device unaltered and can simply be collected, reintroduced and synchronized at a later time.

The balance of back-pressures between the two outlets proved to be essential in achieving high accuracy synchronization. For example, an increase in the back-pressure in the synchronization channel ("Out 1") due to an additional droplet unit operation (such as merging or incubation) must be compensated either by applying negative pressure on "Out 1" or increasing the back-pressure on "Out 2". Analogously, the back-pressure within the two channels leading to "Out 2" should be kept identical. Accordingly, separation of these two channels to independent outlets allows

for independent control of back-pressure, which will improve the synchronization performance, especially when operating with a high disparity in the droplet formation frequencies.

The long-term stability of the synchronization architecture was studied under the identified optimal conditions (i.e. with both droplets having a diameter of 126 µm). Over a period of 45 minutes, 47871 alternating droplets were produced with an error rate of less than 0.02 %. Synchronization of equal-sized droplets entering the synchronization chamber at different flow rates was additionally evaluated to investigate the possibility of compensating for errors in droplet formation frequency. Synchronization was further characterized for conditions where the input droplet flow rates are markedly different. Figure 2.9 a shows that efficient synchronization is highly dependent on the maintenance of equal flow rates. As soon as the ratio of total flow rates between the two flow-focusing junctions deviates from unity, there is a sharp decline in synchronization performance. Figure 2.9 b shows the variation of the droplet synchronization frequency with the total flow rate through the synchronizer. As expected, the number of synchronized droplets increases with higher flow-rates. In the current study, we were able to achieve synchronization rates in excess of 31 hertz. In general, it was found that under optimal conditions, droplets could be synchronized at rates between 8.8 hertz and 31 hertz whilst maintaining a synchronization error below 1 %. It was also observed that synchronization rates could be increased if negative pressure was applied to the "Out 1" channel, although this unsurprisingly required the modification of the flow conditions. We hypothesize that higher synchronization rates could be achievable using higher flow rates than those investigated, although at the expense of increased error rates. In general, the dense packing of synchronized droplets required for an efficient synchronization operation depends on several factors including device geometry (i.e. the spacing and size of the pillars in the packing section), the back-pressure balance ("Out 1" vs. "Out 2") and the volumetric flow rate ratios during droplet formation. Spacing between densely packed, synchronized droplets could further be increased using an additional spacer oil inlet after synchronization, which produces a stream of AB droplets exhibiting uniform distances.

Although all results presented in the current study were obtained using a surfactant concentration of 4 %, concentrations as low as 1 % are sufficient to prevent spontaneous droplet fusion within the device. This reduction further facilitates down-stream merging of droplets, since most passive fusion methods require low surfactant concentrations. Neverthe-

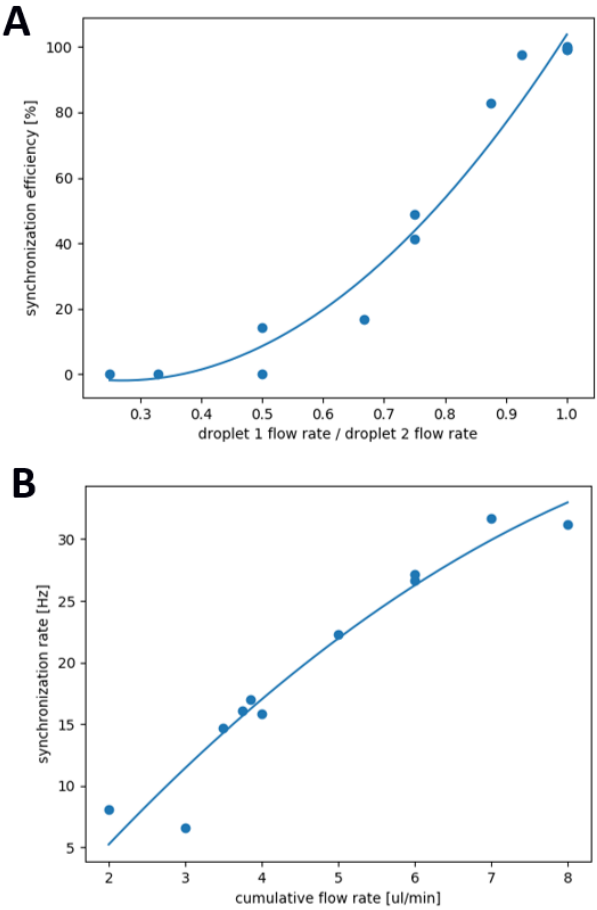

FIGURE 2.9: (A) Dependence of synchronization performance on droplet size difference (N > 500 for each datapoint). A synchronization efficiency above 95 % is only achieved if the two synchronized droplet types enter the chamber at equal flow rates. (B) Dependence of the synchronization rate on total flow rate, showing that throughput increases as a function of flow rate.

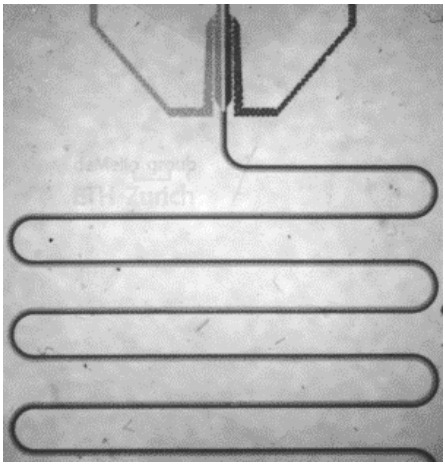

FIGURE 2.10: Long-term stability of synchronized droplet configuration. Application of a vacuum to the outlet channel allows droplets to stay in an alternate configuration for extended time periods, even when in motion. The image shows stable contact between droplets over 200 seconds along a 19 centimeter microfluidic channel.

less, many passive merging strategies do not provide the sufficiently low error rates to be useful in combination with the presented method (*90*). Accordingly, it is suggested that electrowetting would act as an ideal method for initiating droplet fusion through the continuous operation of electrode potentials. We further found that synchronized droplets can be kept in an alternate configuration over extended distances if vacuum is applied to the elongated output 1 channel, lowering the back-pressure and thereby stabilizing droplet synchronization (see Figure 2.10). This feature offers the possibility of studying inter-droplet transfer within flowing environments.

To showcase the modularity of the developed structure, we combined the synchronization architecture with a randomization geometry, which allowed the production of a droplet populations of highly controlled composition (1:1), while still producing droplets in a semi-random order (see Figure 2.11). This was achieved successfully by including delay chambers with various shapes which add a random delay to each droplet due to varying path lengths through the randomization chamber.

Finally, streams of synchronized droplets could further be re-split into the constituent droplet streams (see Figure 2.12), a process previously reported by Surenjav and co-workers (*118*). This operation allows the pro-

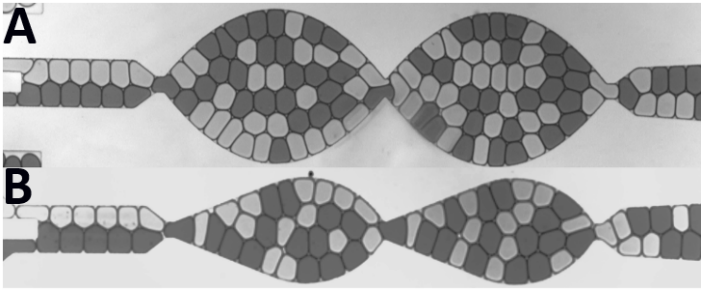

FIGURE 2.11: Randomizing the order of synchronized droplets. The randomization chambers apply a random delay to the passing droplets through the provision of several possible paths (of varying length) within the chamber. This results in a randomly ordered droplet population with a known distribution of droplet types (1:1). (A) Symmetrical, oval randomization chambers. (B) Drop-shaped randomization chambers.

duction of two streams containing an equal number of droplets, thereby discretizing the number of droplets within a channel and ensuring equal droplet frequencies.

We believe the presented microfluidic structure constitutes an interesting and high-efficiency tool for synchronizing droplets because of its inherently low error rate and ability to synchronize pre-formed droplets. We have demonstrated that the presented architecture is able to compensate for variations between the two synchronized streams in both droplet size and frequency. This architecture is therefore exceptionally useful for droplet merging and timing restoration, which are key operations in many biological and chemical assays.

2.5.2 *Quantification of Inter-droplet Osmotic Transfer*

To increase the storage density of droplets and decrease back-pressure in microfluidic channels, most of the continuous phase is removed from an emulsion, resulting in tightly packed droplet configurations. It has previously been shown that cross-talk will occur between such stationary droplets, leading to a homogenization of droplet contents over time (*29*). Furthermore, even though two immiscible phases are used for droplet formation, extensive interaction between the continuous and the dispersed phase has been reported (*28*). For example, extensive transfer of fluores-

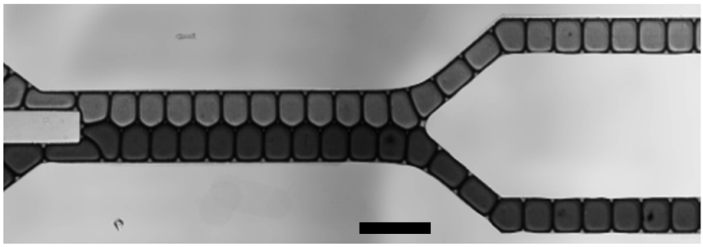

FIGURE 2.12: Splitting of a stream of droplets in an A-B alternate configuration into its constituent droplet populations (scale bar - 220 µm). Direction of flow is from left to right. The split generates a uniform flow of metered droplets through the individual channels, each side-channel containing an equal amount of droplets.

cein molecules between droplets through the continuous phase has been shown for aqueous droplets in fluorinated oils (*28*, *29*). Gruner and co-workers have further found that transfer speeds increase with increased surfactant concentrations, indicating an extensive involvement of the surfactant in the transfer process (*29*).

Water mass transfer due to osmosis between two droplets is dependent on a number of factors, including the concentration gradient, the contact area, the contact time, the type of surfactant used, as well as the permeability of the interface between two droplets (*29*, *120*). Osmosis can be used to alter droplet volumes (resulting in the concentration or dilution of the contained analyte) without coalescence or injection, and has previously been used to study crystallization in nL-volume droplets (*120*), label-free sorting of droplets containing cells (*121*) and in the determination of droplet interface characteristics (*110*). Using the presented synchronization architecture, we are able to produce tightly packed droplet populations, with a highly controlled and reproducible interface area between all adjacent droplets (see Figure 2.13). This allows for the stable monitoring of gradual changes in volume and fluorescence intensity via high-speed microscopy.

To prove principle, we quantified the concentration gradient dependence of osmotic transfer between water droplets in a fluorinated oil, stabilized using EA-surfactant (*23*). Figure 2.13 a shows such an experiment, where water is gradually transfered along the osmotic gradient, resulting in a significant difference in diameter after prolonged contact. Our microfluidic architecture allows for the continuous monitoring of droplets in contact over periods up to 180 seconds, only limited by the available field of view.

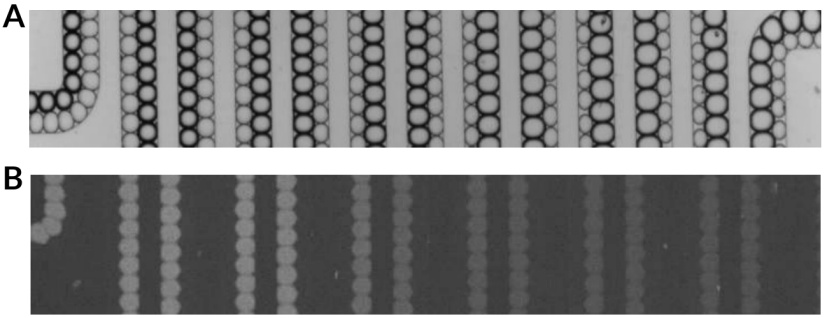

FIGURE 2.13: Studying inter-droplet transfer in flow. Ordered droplets are flowing from left to right through 220 µm diameter channels. (A) Osmotic transfer of water from the diluted droplet into the osmotically active droplet (darker periphery, 4 M NaCl), resulting in a significant size imbalance. (B) pH-based fluorescein quenching. H_3O^+ ions are transfered into a droplet containing a fluorescein solution, which quenches fluorescence activity (*122*).

Using automatic image processing we were able to quantify osmotic transfer between up to 5 droplet pairs per second. We were able to continuously increase the volume of osmotically active droplets by a factor of up to 1.9 within 130 seconds (see Figure 2.14). We could not observe a decrease in the volume change rate which would indicate the equilibration of the concentration gradient. We hypothesize that equilibrium could not be reached due to the relatively large osmotic gradient employed (distilled water and up to 4M NaCl concentration respectively). To our knowledge there is no previous study exploring osmotic transfer between droplets in flow. However,compared to previous measurements of osmotic transfer through static droplet interface bilayers (DIBs) we could observe an increased rate of volume change (*123*).

The fluorescence intensity of fluorescein is highly dependent on the pH of the fluorescein solution (*122*). Fluorescence emission can thus be reduced by lowering the pH of the solution. We could therefore use the developed droplet synchronization architecture to determine the rate of H_3O^+ transfer between droplets through quantification of fluorescein quenching (see Figure 2.13 b). Results in Figure 2.15 show a decrease in fluorescence emission of droplets containing fluorescein (100 µM) upon prolonged contact with an acidic droplet (at pH 0.5).

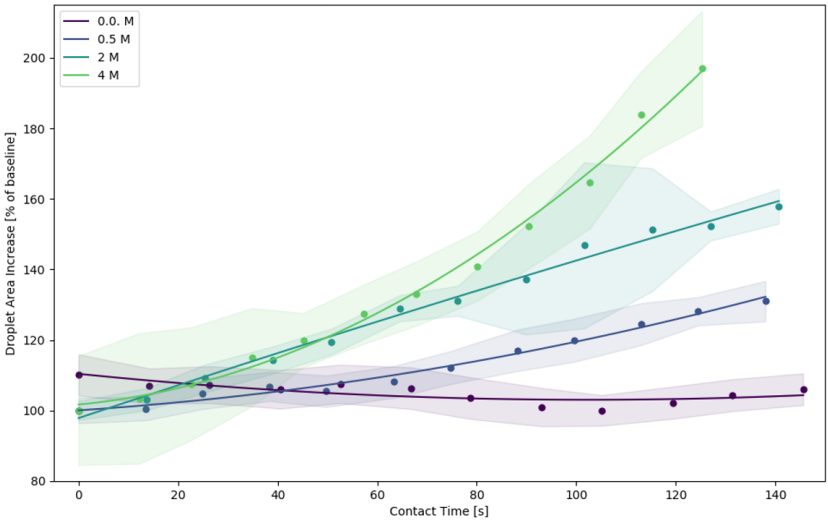

FIGURE 2.14: Droplet area variations due to osmosis. Osmosis is achieved by arranging two separate types of droplets in a "zig-zag" formation (see Figure 2.13 a) and monitoring droplet size change. Data shows mean droplet diameter and one standard deviation. At least 500 droplets were measured per data point. High salinity droplets in contact with droplets containing deionized water resulted in transfer of water in the direction of the osmotic gradient. This in turn lead to an increase in the observable droplet area over time. It is evident that an increased sodium chloride concentration in the osmotically active droplet results in an increased rate of osmosis. The plot shows the mean droplet diameter, experimental variation, as well as a second-order polynomial fit of the droplet area as a function of time.

Since previous studies have shown that surfactants play a key role in mediating inter-droplet transfer (*29*), we hypothesized that droplet movement can increase osmotic transfer by enhancing surfactant exchange between adjacent droplets. We also expected the increased mixing within moving droplets (*124*) to further support osmotic transfer by reducing depletion zones along the surface of the droplets. Therefore we measured fluorescein quenching across a wide range of flow velocities (see Figure 2.15). We found that droplets emitted identical fluorescence intensities after a defined contact distance, regardless of flow velocity. Our results thus indicate that inter-droplet transfer rates are directly proportional to flow velocities and larger flow rates lead to increased H_3O^+ transfer in densely packed droplet configurations.

Therefore, using the presented system we could show that inter-droplet transfer can be used to control the pH of a droplet in flow (allowing for passive quenching of pH-sensitive reactions in droplets), as well as adjusting droplet size, without the necessity of droplet merging. We could also show that the inter-droplet droplet barrier is highly permeable to both charged and neutral species. We believe the results highlight the importance of investigating inter-droplet transfer as it has a significant impact on a large variety of droplet microfluidic assays. Additionally, the observed increased transfer rates at higher flow velocities suggest that low-perturbation droplet storage systems are an essential for long-term droplet incubation (see Chapter 3).

2.6 CONCLUSION

Herein, we have demonstrated a passive microfluidic architecture able to perform high efficiency droplet synchronization using pre-formed droplets. Using this architecture, we are able to synchronize in excess of 45000 droplets in 45 minutes with an error rate below 0.02 %. We have further evaluated the synchronization performance of the device when synchronizing droplets of variable size and have established that excellent synchronization (with efficiencies in excess of 90 %) can be achieved over a broad droplet size range. We have also shown that droplet synchronization rates of up to 33 hertz are possible, whilst maintaining a synchronization performance in excess of 99 %. Although synchronization efficiencies could be further improved through optimization of device geometry, we believe that the achieved error rate is sufficiently low for the successful incorporation in a range of multi-step assays.

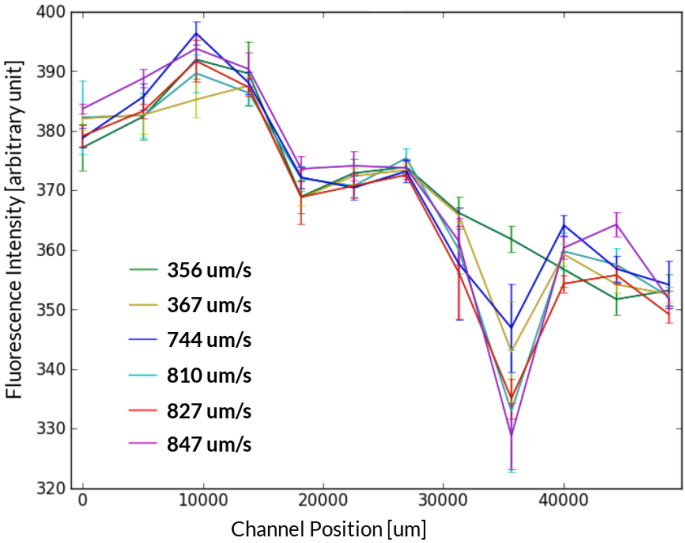

FIGURE 2.15: Transfer of H_3O^+ ions into fluorescein-containing droplets from low pH droplets (see Figure 2.13 b) results in the quenching of fluorescence signal intensity. Fluorescein quenching through H_3O^+ transfer between droplets at various flow velocities was measured (at least 500 droplets per flow velocity and channel position). We could observe that droplets exhibited similar fluorescence activity at a fixed channel position, regardless of flow velocity. We thus find that larger flow velocities lead to increased transfer rates. Therefore, we suggest that inter-droplet osmotic transfer of H_3O^+ ions in densely packed droplet configurations is directly proportional to the flow velocity.

To highlight the applicability of the presented architecture we have successfully used the presented method to quantify osmotic inter-droplet transfer in flow. We could observe that inter-droplet transfer between tightly packed droplets is highly dependent on the flow velocity of the emulsion. Thus our results highlight that low-perturbation droplet storage, requiring minimal motion during the storage process, is paramount to successful long-term incubation (see Chapter 3).

Whilst many novel microfluidic techniques employ active control schemes, we believe that the proposed system highlights the power and simplicity of passive microfluidic control. That said, as with many other passive droplet unit operations, the presented synchronization architecture is highly engineered to perform only a specific task, whereas microfluidic systems using active control are often more flexible. Accordingly, we believe that the informed microfluidic experimenter should carefully weigh the benefits of active and passive control schemes before choosing a mode of experimentation.

3

LARGE-SCALE ACTIVE DROPLET BARCODING

3.1 INTRODUCTION

In recent years numerous biological and chemical assays have been transferred to droplet-based microfluidic formats due to a range of key advantages, including reduced reagent and sample consumption, improved control over thermal and mass transport and high analytical throughput (*1*). Due to the simplicity in compartmentalizing reagents within droplets surrounded by an immiscible carrier fluid, droplet-based assay platforms can are able to produce large number of experimental repeats in a few milliseconds. This in turn, enhances statistical robustness (and precision) and reduces experimental error when compared to bulk measurements (*1*). That said, although most droplet-based microfluidic workflows are effective in assaying and interrogating individual droplets, they cannot reliably track large numbers of individual droplets as a function of time and are therefore limited to quantifying population averages instead of individual time courses. A number of literature studies have shown that the ability to measure and probe individual analytes (such as cells) over time is often crucial, with many chemical and biological systems exhibiting fundamentally different dynamics when analyzed on the single-cell level as opposed to the population or ensemble average level (*125*). For example, Elowitz and coworkers proposed a system to differentiate intrinsic noise from extrinsic noise in the process of gene expression. Here, the ability to perform measurements on a single-cell level was critical in establishing a quantitative foundation for modeling noise in genetic networks, with population averaged measurements being insufficient (*125*).

3.1.1 *Droplet Barcoding*

In droplet-based microfluidic platforms, the precise identification of individual droplets within a large population allows the experimenter to

Michael Lütolf has contributed to droplet storage chamber development.

repeatedly assay droplets. Importantly, accurate droplet identification enables the subsequent correlation of separate measurements (of the same droplet) obtained at different time points, and thus allows reconstruction of the temporal signal for each droplet in the population. Put simply, such droplet identification enables high-throughput microfluidic experimentation, where droplets may be repeatedly assayed and processed within complex chemical and biological workflows. Examples of high-throughput and high-efficiency assays employing droplet barcoding include expression profiling of individual cells (*65*) and the analysis of enzyme kinetics (*126*). However, it should be noted that the precise identification and manipulation of individual droplets within a large population remains enormously challenging, as positioning strategies (such as sequential ordering and use of trap arrays) become unrealistic when dealing with very large numbers of droplets (above 1000 droplets) due to unacceptably high back-pressures within the microfluidic device (*31*). Accordingly, several techniques for "labelling" individual droplets have been reported in recent years, all relying on encapsulation of a labelling agent within individual droplets. The subsequent readout of such labelling agents provides a direct method of identifying a droplet regardless of its spatial location. Unsurprisingly, the unique signal of the encapsulated labelling agent is often referred to as a "barcode". Several droplet barcoding strategies have been reported in the literature and rely on the use of particles (*127–130*), nucleic acids (*60*, *66*) or fluorescent dyes (*63*, *64*, *131*, *132*) as labelling agents.

Particle-based barcoding methods involve the synthesis of unique particles, which can then be encapsulated into droplets (see Figure 3.1 a,b). For example, single-color particles that rely on bright-field observation of particle shape for droplet identification have been proposed (*127*). Typically, such particles are synthesized using microscale lithographic techniques, such as stop-flow lithography (*133*). Although, adept at the construction of complex particle geometries, stop-flow lithographic methods are difficult to scale up, since new photomask templates are normally required to fabricate novel particle shapes. In contrast, fluorescent particles can be manufactured using combinatorial synthesis principles, by forming many combinations from a small number of fluorescent dyes, enabling the fabrication of multiple individual barcodes in a facile process. For example, Bong and co-workers have recently reported the synthesis of fluorescent particles that incorporate up to four separate fluorescent dyes within a single particle using stop-flow lithography (*128*). Further, a recent study by Nguyen and co-workers has reported a system allowing for the pro-

duction of over one thousand uniquely barcoded fluorescent microparticles made from five different Lanthanide precursor solutions (*130*). Additionally, unique optical patterns can be transferred into fluorescently-dyed particles using spatially selective photobleaching (*129*). However, besides their complex synthesis, many particle-based barcoding strategies require image-based read-out (via bright-field or fluorescence measurements) and extensive image post-processing to determine particle identity, which can be challenging to implement in real-time assays. Moreover, as highlighted in the presented studies, it has been challenging to scale particle-based barcoding strategies beyond one thousand unique particles.

Recently, nucleic acids have been extensively used for the identification of droplets (*60*, *66*). Such techniques have allowed the creation of very high numbers of distinct barcodes, due to the relative ease of synthesizing large numbers of unique nucleic acid sequences via randomized processes. Typically, unique nucleic acid barcode sequences are introduced into a droplet along with a biological analyte. Variants of this method incorporate specific carriers for barcode sequences, such as hydrogel spheres (see Figure 3.1 c) (*66*), or directly dissolve the barcode sequence within the aqueous phase (see Figure 3.1 d) (*60*). Before the barcode can be read back, the biological sample (cell) is typically lysed, and target sequences are conjugated with the barcode sequence in the droplet. Finally, the emulsion is broken, and all nucleic acids are sequenced. Subsequently, target sequences are correlated with their droplet of origin through the conjugated (and specific) barcode sequence. For example, Lan and co-workers have barcoded a population of 10^7 droplets using this technique (*60*). Despite its high-throughput nature, nucleic acid-based barcoding methods do not allow time-course measurements, since droplets can only be identified at a single time, due to the destructive nature of the readout process (i.e. breakage of the droplet emulsion for subsequent nucleic acid sequencing). Further, the technique is currently limited to the detection and quantification of nucleic acids exclusively.

Several studies have used co-encapsulation of multiple fluorescent dyes for barcoding, where variations in the relative concentrations of several dyes allow the formation of a fluorescent signature, which can be used as a barcode (*63*, *64*, *131*, *132*) (see Figure 3.2). Due to the accessibility of rapid, sensitive and non-destructive fluorescence detection methods, fluorescent barcodes, encapsulated within microfluidic droplets can be measured repeatedly and on a sub-millisecond timescale. For example, in an early study, Neils and co-workers employed passive on-chip dilution and

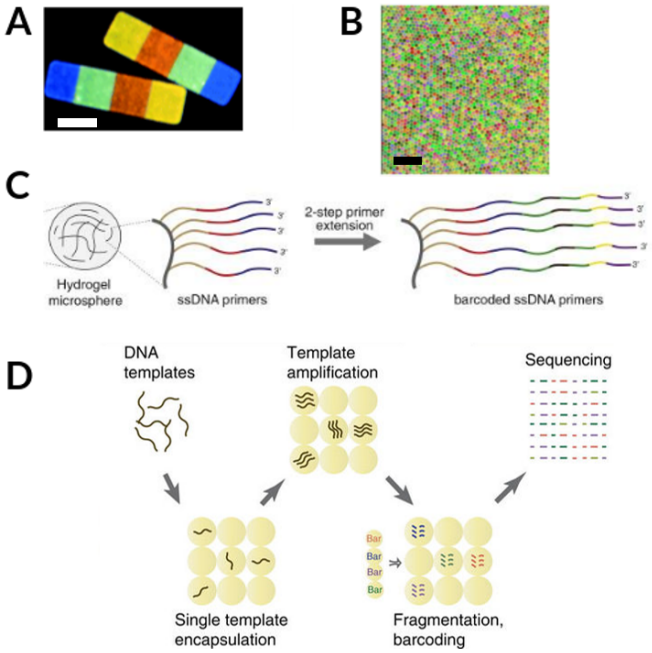

FIGURE 3.1: Particle and nucleic-acid based barcoding methods. (A) Particle-based barcodes presented by Bong and co-workers (scale bar - 50 µm) (*128*). Barcodes are fabricated using stop-flow lithography from precursor solutions containing various fluorescent dyes (*133*). Variation in the order and type of dye within the fabricated particle allows for image-based identification. (B) Microbeads formed by polymerizing microfluidic droplets containing varying concentrations of Lanthanides (scale bar - 500 µm) (*130*). (C) DNA-based droplet barcoding as presented by Klein and co-workers (*66*). Hydrogel spheres are barcoded using a random DNA library (a hydrogel barcoding process is shown) and subsequently co-encapsulated with a single cell in a droplet. After lysis, the cellular DNA is bound to the barcode DNA and all droplets are sequenced. This technique can yield single-cell genomic information for large cell populations, but the barcode readout process is destructive. (D) A similar approach to (C) uses DNA templates directly dissolved within a droplet as opposed to fixed inside a hydrogel sphere (*60*). This technique allowed Lan and coworkers to generate several million unique nucleic acid-based barcodes.

mixing to create 16 distinct combinations from two fluorescent dyes in continuous flow (*131*) (see Figure 3.2 b). Although this study did not involve the generation of barcoded droplets, the use of laminar flows to create unique optical signals is facile. That said, scaling up this strategy to generate larger numbers of color combinations dramatically increases device size and complexity due to the passive mixing scheme employed. In a more recent study, Lin and colleagues produced 10 different barcoded droplets using a parallel emulsification approach (*64*) (see Figure 3.2 a).

Double emulsions are "emulsions of emulsions", where droplets contain one or more types of smaller dispersed droplets, and thus require the use of three separate immiscible phases (*134*). Zhao et al. have reported the generation of barcoded double emulsions (*132*), where five distinct photonic crystal oil solutions are used to form the inner phase. Varying numbers of photonic crystal droplets are then injected into an aqueous droplet, which in turn is encapsulated within a hexadecane solution. Despite the fact that this strategy theoretically allows the production of more than of 10^4 unique barcodes, fabrication remains challenging and indeed the study only reported the creation of five distinguishable barcodes. Accordingly, it is fair to say that most studies have only managed to produce limited numbers of individual barcodes (e.g. 10 in (*63*)) and typically generate an unacceptably large number of duplicate barcodes (e.g. 1500 in (*63*)).

Based on an analysis of literature barcoding methods, it can be concluded that an optimal droplet barcoding strategy enables the production of large quantities of distinguishable barcodes. Furthermore, each produced barcode should be unique, as duplicate barcodes limit down-stream applications, by prohibiting definite identification of individual barcoded droplets. Finally, the barcoding strategy should allow for non-destructive read-out, enabling multiple measurements in complex workflows.

3.1.2 *Droplet Storage*

In addition to efficient droplet barcoding, high-efficiency droplet storage and retrieval forms an integral part of any successful barcoding strategy, particularly when performing experiments over extended time periods. In recent years, two common methods have emerged for facile long-term storage of microfluidic droplets. In the first, basic laboratory tubing is used to store droplets (see Figure 3.3 a). The main advantages of this approach are the lack a complex chip-to-world interface, and the ability to surround droplets by large amounts of carrier fluid during storage. Due to the wide

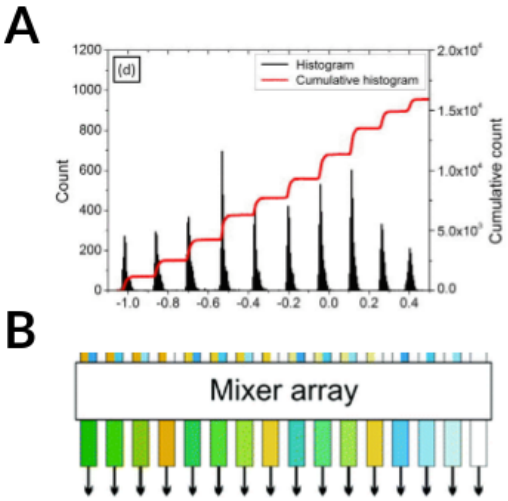

FIGURE 3.2: Dye-based barcoding methods. (A) Dye-based fluorescent barcodes, produced using on-chip dilution (*63*). The histogram shows 10 clearly separated droplet populations containing varying concentrations of a single fluorescent dye. Similar to other on-chip dilution barcoding strategies, this approach produces a relatively small number of unique barcodes (10) and a high number of repeat structures (>500). (B) Schematic of an on-chip mixing array producing 16 unique fluorescent signatures from two fluorescent dyes (*131*). The study highlights the utility of combinatorial mixing but only produces 16 different color combinations.

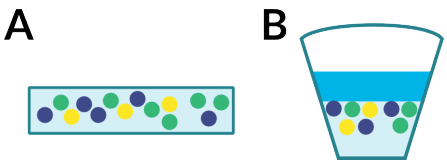

FIGURE 3.3: Traditional droplet storage methods. (A) Illustration of droplets stored in tubing. Such a storage method allows for storage of an appreciable but not excessive number of droplets. Droplet retrieval can be challenging due to droplet buoyancy and tubing length. (B) Droplet storage within a container sealed by an immiscible fluid. This is a popular method for droplet storage due to its simplicity. However, due to buoyancy droplets may adhere to the interface between the continuous phase and the sealant, greatly complicating retrieval and leading to the loss of a large number of droplets.

range of tubing materials available, the user can choose the optimal material for the investigated droplet system, with inert materials such as PTFE being popular choices.

However, there are a series of practical disadvantages to storing droplets in tubing. Retrieving droplets stored in tubing requires pumping of the tubing contents and use of excess amounts of continuous phase. Furthermore, in a typical two-phase droplet system, the densities of the phases are not matched. As a result, simply pumping continuous phase into the storage tubing may not be sufficient to retrieve droplets, as the buoyancy retains droplets inside the tubing. This can be mitigated by rearranging the tubing such that the outlet is at the highest point but may also require altering the microfluidic design to enable side-access to the device. Higher expulsion flow rates or smaller tubing diameters can also be used, but both options increase stress on the droplets, which in the worst case will result in droplet merging or splitting. Additionally, storing large droplet populations in tubing requires long tubing sections, which further dilutes the emulsion during retrieval.

Microfluidic droplets can also be stored in a container by simply covering the emulsion with a third immiscible phase (see Figure 3.3 b). This prevents droplet evaporation, since the third phase effectively seals the continuous phase from contact with the atmosphere. The primary advantages of this storage method are ease of setup and scalability, since droplets can be stored in an open container with the simple addition of a sealing fluid. However, since the most stable aqueous droplet systems frequently

use fluorinated oils, droplets will normally rise to the surface due to buoyancy and are therefore captured at the interface between the continuous phase and the sealing liquid. During droplet retrieval, droplets trapped at this interface are commonly discarded, as they cannot be retrieved without polluting the emulsion with the sealant phase. The resulting loss of droplets must therefore be factored in during experimental planning and is only feasible if the droplet population contains large numbers of duplicate droplets.

To mitigate the above challenges, we herein develop a droplet storage system which combines the advantages of both traditional methods. Our storage strategy relies on the use of a glass/polymer hybrid device, made using additive manufacturing methods. The device geometry allows for two different modes of operation, namely droplet storage and droplet retrieval. Switching between modes is performed by physically flipping the device and reversing the flow direction. The presented storage system allows for easy handling with minimal human interaction. As previously shown in Chapter 2, such low-perturbation droplet storage systems are crucial as inter-droplet transfer is increased at higher flow velocities. Furthermore, the developed droplet storage system enables the long-term storage of large droplet populations whilst preventing droplet merging or splitting during incubation. Finally, and most crucially for droplet barcoding, the developed storage method enables the near perfect retrieval of all droplets.

3.1.3 *Active Sorting Barcoding*

Herein, we present a novel barcoding strategy (see Figure 3.4), which incorporates the automatic and scalable formation of barcoded microfluidic droplets and is based on the controlled co-encapsulation of multiple fluorescent dyes. Droplets of uniform size (and containing a specific barcode) are formed by controlling the relative flow rates of separate fluorophore solutions prior to droplet formation. Using a field-programmable gate array-based (FPGA) data processing algorithm and a dielectrophoresis-based sorting scheme (*49*), we demonstrate the automatic detection of droplets containing a specific barcode and subsequently use active sorting to collect and isolate the corresponding droplet. Barcoded droplets are subsequently stored in the droplet storage unit to prevent merging and loss of droplets. The sorting algorithm identifies regions of missing barcodes and automatically adjusts the flow rates of the fluorescent dyes to produce novel barcodes. Significantly, this ensures rapid and maximal coverage of the full dy-

namic range of the detector. The presented system allows the production of more than 2000 unique barcodes from only two fluorophores. Accordingly, the proposed droplet barcoding system is highly flexible regarding the number of barcodes produced, fluorophore types, fluorophore concentrations and can be tuned to incorporate various detectors with differing resolutions.

3.2 BARCODING SYSTEM DEVELOPMENT

The proposed strategy aims to generate a large number of droplets, each containing a unique fluorescent barcode (see Figure 3.4). In contrast to previously published barcoding methods that use combinations of fluorescent dyes, we aim to ensure that each barcode exists only once within the droplet population. Since we are using water-in-oil emulsions, we are able to produce a unique fluorescence signature by varying the concentrations of water-soluble fluorescent dyes incorporated into each droplet. Fluorescent dye concentration variations are produced by combining multiple aqueous pure-dye precursor solutions within a microfluidic channel (see Figure 3.4 a). Subsequently, droplets are formed from this multi-dye solution using a standard flow focusing geometry (*13*) (see Figure 3.4 b). Such an approach allows the straightforward variation of the chemical payload by changing the flow rates of the constituent solutions. Accordingly, large numbers of unique (fluorescent) barcodes can be produced from a relatively small number of precursor solutions by applying user-defined flow profiles, thereby varying the fluorophore concentration in the formed droplets.

However, due to the rapid droplet formation rates, typical to microfluidic droplet formation, and the comparatively slow response time of the syringe pumps used to control flows, the observable difference in fluorescence intensity between adjacent droplets is typically below the limit of detection of a PMT. Furthermore, small-scale fluctuations in the aqueous flow rate arise from a variety of fluidic phenomena. For example, syringe pumps introduce flow rate oscillations due to step-wise rather than continuous plunger movement. Additionally, the elasticity of the tubing, pressure wave propagation in the tubing and competitive flow prior to droplet formation will produce further fluctuations. When combined, all of these effects can introduce sufficient flow rate noise for adjacent droplets to contain identical fluorophore concentrations. Therefore, fluctuating fluorescence gradients during droplet formation are likely to preclude iden-

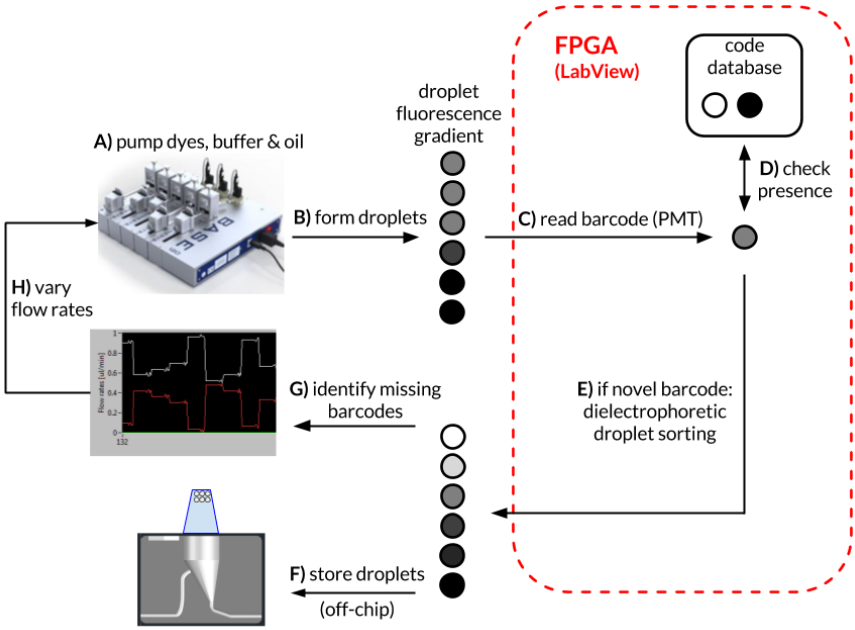

FIGURE 3.4: Droplet barcoding strategy. (A) Syringe pumps are used to pump multiple dye solutions, buffer and oil (continuous phase) into a microfluidic device (see Figure 3.6 a for a schematic of the device used in current experiments). (B) Dye and buffer solutions are mixed on-chip and droplets are formed at a flow focusing geometry. Changing the relative flow rates of the aqueous phases results in a population of droplets containing varying amounts of fluorescent dyes, but also leads to the formation of multiple duplicate droplets. (C) Time-integrated fluorescence is measured as each droplet passes through a laser light sheet using multiple photomultiplier tubes (PMTs). Subsequently, dye concentrations within each droplet are quantified by integrating over the fluorescence intensity signal obtained. (D) The measured fluorescence signature is compared with a database of previously obtained barcodes. (E) If a novel barcode is identified, the corresponding droplet is removed from the main stream using high-speed dielectrophoretic droplet sorting. (F) Sorted droplets containing unique barcodes are stored in the droplet storage chamber. (G) Regions in the barcode phase space with numerous missing droplet barcodes are identified and the corresponding pump flow rates are estimated. (H) The barcoding algorithm attempts to produce novel droplet barcodes by adjusting the flow rates of the constituent dyes and buffer.

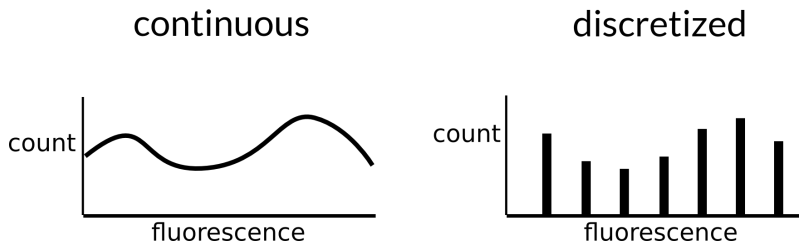

FIGURE 3.5: Droplet discretization step. The production of identifiable droplet barcodes requires discretizing the droplet fluorescence signals. Co-encapsulation of various fluorescent dyes typically results in a continuous distribution of fluorescence intensities. Such continuous droplet populations must be discretized prior to experimentation, such that a detector is able to discern differences between individual droplets. The resulting discretized droplet population is characterized by a non-continuous histogram.

tification of unique droplets based on their "theoretical" barcode due to a large number of duplicate barcodes in the resulting droplet population. This effect is further exacerbated during long-term storage by droplet leakage (Chapter 2) and photo-bleaching effects. Accordingly, successful barcoding strategies based on fluorescent dyes typically require a high-confidence discretization step (see Figure 3.5). This step ideally separates the fluorescence signatures of the droplet population into sufficiently different intensities, enabling the definite assignment of each droplet without duplicate assignments. Several passive approaches have been developed to discretize barcodes, including the generation of pre-defined mixtures through on-chip dilution (*131*) or the direct production of larger but distinct droplets at slower rates (*15*). However, these methods yield relatively few specific barcodes and cannot guarantee the absence of duplicate fluorescence signatures within the final droplet population.

To address the aforementioned limitations, we propose a method that achieves discretization by active sorting. Initially we produce droplets containing color gradients by varying the volumetric flow rates of the delivery pumps. Subsequently, the fluorescence signature of each droplet is measured using one PMT per fluorophore (see Figure 3.4 c). By default, all formed droplets are discarded. However, if the detected barcode is novel and unique, it is sorted into a separate collection outlet (see Figure 3.4 d,e). We can therefore ensure that each sorted droplet is unique and sufficiently

different from all "similar" barcodes given a specific detector resolution. This greatly simplifies subsequent error-free barcode identification.

Several additional operational benefits arise when using a sorting-based barcoding strategy. First, disparately-sized droplets, such as droplets formed during the flow equilibration phase, can be easily discarded. Furthermore, the user knows which barcodes have been collected, and is thus is able to selectively generate novel barcodes (see Figure 3.4 g,h). In the following sections we highlight key aspects of our barcoding strategy, including flow profile generation, barcode readout, sorting logic and low-loss storage of droplets in 3D-printed storage chambers.

3.2.1 *FPGA-based Barcode Generation*

In the described system, the fluorescence intensity of a droplet is measured using a PMT at a sampling rate of 120 kilohertz. After each sample or measurement, signal processing is used to determine the signature of the passing droplet. A critical aspect of the strategy, is that a sorting signal is triggered as soon as a desired droplet is detected. Due to the inherently strict time requirements, a FPGA-based (*135*) detection and sorting algorithm was developed using LabView.

When compared to traditional computer programming, algorithms implemented using FPGAs are not executed using a general-purpose processor, e.g. the central processing unit of a computer. Instead, FPGA-based algorithms are directly implemented on a FPGA chip by routing various types of logical gates together. Accordingly, an FPGA program should be compiled into a logic circuit diagram, which can then be realized on the FPGA chip. This provides significant advantage for low-latency applications, since input data is directly fed into the circuit and output signals are directly retrieved from the chip. In contrast, transferring real-time data into a CPU typically requires many steps (and thus significantly increases runtime) since there are several abstraction layers in between, for example a USB port and the CPU. Furthermore, the CPU also handles a number of other processes simultaneously, such as running the operating system.

Additionally, and due to its low-level implementation, an FPGA can typically guarantee a fixed execution time for each program, which eliminates unexpected delays (or lags) during algorithm execution (*135*). However, FPGAs do possess some undesirable features, even when programmed in a high-level language such as LabView. First, since algorithms must be compiled into logic gate circuits, many convenient standard tools avail-

able in general programming languages, such as variable-sized array data structures, are inaccessible when programming an FPGA. This typically increases program complexity, which increases development time. Furthermore, as all FPGA-based algorithms are constructed by connecting logical components on an FPGA chip, the resources available for implementation are inherently limited. For example, the current application requires tracking all previously collected barcodes, which is immensely challenging with the limited memory available (216 kilobyte block RAM in our case). Despite such drawbacks, an FPGA-based approach is well-suited to our barcoding strategy, as we require low-latency and real-time signal processing with only limited computational complexity.

In a first step, the proposed FPGA-based algorithm measures PMT voltages, corresponding to fluorescence intensities (see Figure 3.4 c). Analysis of these signals over time allows the algorithm to determine whether a droplet is currently passing through the detection volume, with the majority of signal processing step taking place after a droplet has completed passage through the detector. Next, each possible fluorescent signature is assigned a fixed number (for example 50 % intensity of fluorophore 1 and 30 % of fluorophore 2 corresponds to barcode number 17). Crucially, such a conversion into continuous decimal indices allows a maximal number of barcodes to be stored in the limited memory of the FPGA device, as each barcode is stored as a simple Boolean (True/False) in a large database. To check whether a barcode has previously been sorted, the algorithm simply checks if the value at the specific index in the database is set to true. Accordingly, using the recorded fluorescence signature, the algorithm can subsequently decide whether the fluorescence footprint represents a novel barcode or whether it is too similar to a previously detected fluorescence signature (see Figure 3.4 d). Finally, if a novel barcode is confirmed, a sorting signal is triggered (see Figure 3.4 e), the droplet is sorted to the collection outlet (see Figure 3.4 f), and the corresponding entry in the barcode database is updated.

Our microfluidic device (see Figure 3.6 a) uses the basic sorting principle presented by Sciambi and co-workers (*49*). This, in principle, enables precise electrophoretic droplet sorting at rates exceeding 30 kilohertz (see Figure 3.6 b highlighting the sorting principle reported in (*49*)). For sorting, we algorithmically generate a square wave signal (at 1000 hertz), which is amplified to 1000 V using a high voltage amplifier. Channels filled with a concentrated salt solution (2M NaCl) serve as electrodes relaying the sorting signal from the amplifier to the sorting junction. Crucially, the sorting

architecture contains a gapped divider along the center line of the channel. This divider separates the two possible sorting outlets but is only half the height of the microfluidic channel. Due to this partial separation, droplets require only minimal deviation from the default stream line to be sorted into the collection outlet. Moreover, the gap in the divider allows for exchange of continuous phase between sorting channels, further stabilizing the sorting operation.

Using the developed FPGA software, we were able to achieve exquisite control over the final barcoded droplet population. Several parameters were introduced to allow flexible configuration of the barcoding algorithm. The experimenter can adjust the number of fluorescent dyes used, as well as the number of barcodes requested for each fluorescent dye. Further, the developed software allows for an adjustable precision of the requested barcodes. This is crucial, since the PMTs output a continuous fluorescence intensity signal, and a flexible precision allows the experimenter to tune the allowed deviation of the measured fluorescence intensity from the predetermined barcode fluorescence intensity. However, too high precision results in an excessive amount of discarded barcodes, extending experiment runtimes.

Once the allowed barcode regions have been properly defined, a barcoding experiment consists of sorting exactly one droplet per barcode region. After sorting the first droplet, exhibiting the desired characteristic of a region, the region is locked and no further droplets with similar fluorescence intensities are collected, thus resulting in a droplet population containing only unique fluorescence signatures.

3.2.2 *Flow Rate Feedback Algorithm*

Using the described barcoding algorithm, droplet production is completely independent from droplet sorting, thus allowing for separate optimization of both processes. Initially, we considered producing droplets in a separate step and sorting droplets that had been reinjected into the microfluidic device. Ultimately, we chose to incorporate both processes on a single microfluidic device, since this enables both automatic and integrated flow rate control. During the droplet barcoding process, the primary goal is to produce a maximum number of droplets of the same size and having unique fluorescence signatures. Accordingly, the volumetric flow rate of the continuous as well as the (total) volumetric flow rate of the dispersed phase are

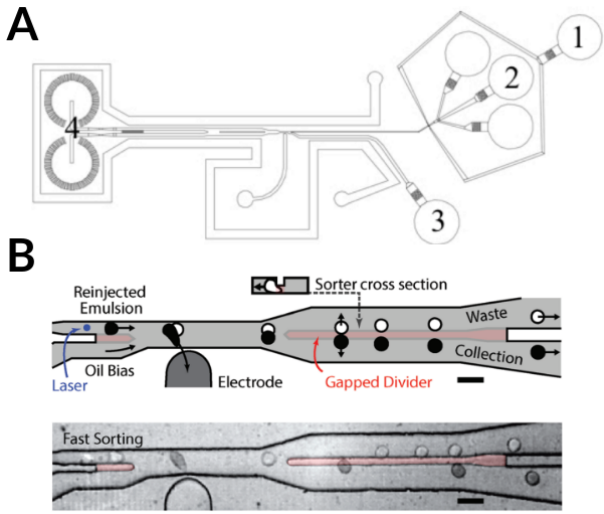

FIGURE 3.6: Microfluidic device design and high-speed droplet sorting. (A) Schematic of the microfluidic device used for droplet barcoding. Droplets are formed from the continuous phase (inlet 1) and a mixture of various aqueous phases (inlets 2) using a flow focusing geometry. Prior to sorting, droplets are further spaced by introduction of additional continuous phase (inlet 3). The droplet barcode is measured, and droplets sorted prior to exiting the device through two separate outlets (4). (B) Droplet sorting principle shown in (*49*). The fluorescence originating from droplets entering from the left-hand side are detected using a laser sheet. A high-frequency and high-voltage signal is used to move desired droplets into a separate collection channel. A gapped divider between the outlet channels (which spans half the channel height) means that only a very small droplet deflection is required to sort a droplet.

kept constant, with variations in the composition of the dispersed phase controlling the fluorescence signature.

When producing single-color barcodes, a dye solution is combined with an aqueous buffer solution. Similarly, multi-color barcodes are formed by altering the flow rate of multiple dye solutions as well as adapting the flow rate of the buffer solution. Three pre-programmed flow profile strategies were tested for their ability to produce all possible combinations of dye concentrations.

Figure 3.7 shows examples of the pre-programmed flow profiles tested in this study. A ramping strategy involved systematically producing all possible combinations of flow rates by varying the flow rates of constituent fluorescent dye solutions (see Figure 3.7 a). In contrast, Figure 3.7 b shows a "random" flow profile, where the volumetric flow rates of both dyes are randomly increased or decreased at each time step. Finally, Figure 3.7 c highlights a "semi-random" strategy where combinations of colors were selected from all permutations of desired dye concentrations without replacement, in a sense representing a combination of the strategies highlighted in Figure 3.7 a and b. Whilst these flow profiles should (theoretically) produce all possible color combinations, we found that each pre-programmed flow profile missed a significant number of possible barcodes (Figure 3.8). We hypothesize, that the pre-determined flow profiles do not directly correspond to the fluorescence intensity measured due to a variety of fluidic complications, such as the flexibility of the PDMS chip and tubing used, as well as the pinch flow regime present at the mixing junction. Thus, we chose to implement a direct feedback mechanism connecting the barcode sorting mechanism and the droplet-forming pumps, to actively increase the probability of generating novel color combinations.

Figure 3.8 shows droplet barcodes produced using the flow profile highlighted in Figure 3.7 a. It can be observed that a large number of possible color combinations were not obtainable. Accordingly, we chose to implement an advanced feedback algorithm, which directly adjusts the flow rates of the droplet producing pumps based on all previously identified droplets.

Initially, the real-time flow rate feedback algorithm determines the lowest occupancy region in the phase space diagram using a two-dimensional binning process (identifying empty regions in Figure 3.8). This effectively identifies dye concentrations with the highest chance of generating a novel (yet to be fabricated) barcode. Next, the flow rates are estimated so that they produce the desired dye concentration based on previous flow rates.

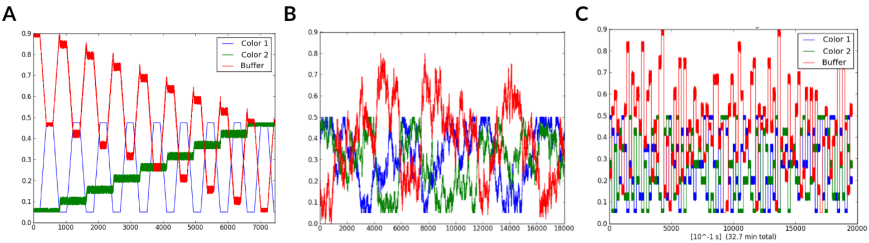

FIGURE 3.7: Droplet formation flow profiles. All flow profiles involve the variation of the flow rates of two fluorescent dye solutions and the control of buffer flow rates (red) to maintain a constant total flow rate. (A) A step-wise flow profile. While one dye solution is ramped upwards in a step-wise fashion (green) the other color fluctuates quickly (blue). (B) Both colors are varied using a random walk strategy, resulting in a random flow profile. (C) A semi-random flow profile, obtained by randomly choosing flow rates from a fixed set without replacement. This ensures that all possible combinations will be produced eventually, while the paths between set points are random due to randomly selecting the order of set points.

If the desired droplet is not found (due to an incorrect flow rate estimate), the flow rates are gradually adjusted with a view to producing droplets exhibiting the desired fluorescence intensity. In a next step, artificial noise is added to the flow rates, which enables production of droplets with similar barcodes. Using such an algorithm, droplets with novel fluorescence signatures can be selectively produced. Finally, these steps are repeated ad infinitum or until stopped by the experimenter, increasing phase space coverage to well above 95 % within 20 minutes (Figure 3.14).

3.2.3 *Droplet Storage Chamber Design*

Since the developed droplet barcoding strategy produces droplets with unique fluorescence signatures, each droplet lost means the loss of a barcode. To prevent excessive droplet loss, it was therefore crucial to develop a high-performance droplet storage strategy, that allows stable and long-term storage of barcoded droplets and low-loss droplet retrieval. Importantly, the developed device should allow for complete retrieval of stored droplets whilst preventing droplet merging or splitting. Finally, the en-

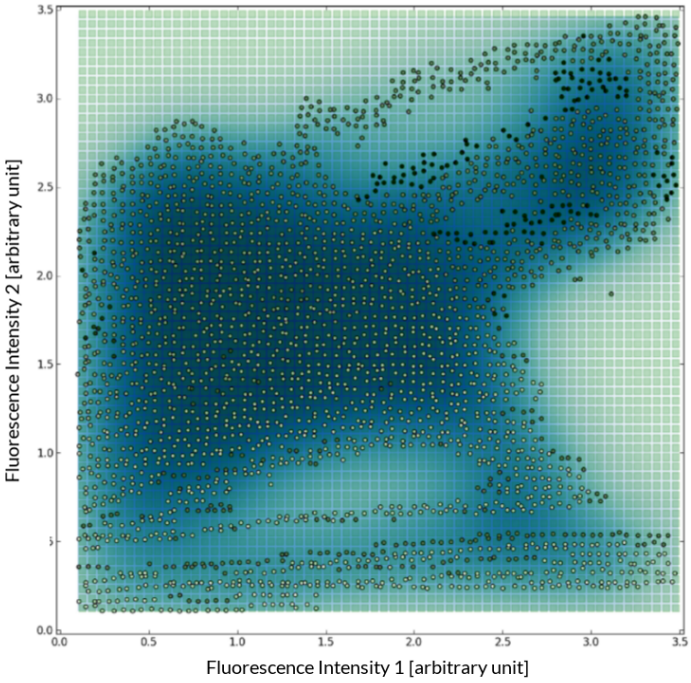

FIGURE 3.8: Droplet barcode collection using pre-defined flow profiles. The scatter plot shows the measured color (fluorescent intensity) of each unique droplet collected using a step-wise flow profile (see Figure 3.7 a). Each possible barcode is represented with a green area, and the barcode density (blue overlay) is calculated using gaussian kernels. Furthermore, the shading of data points highlights the order in which they were collected (with darker barcodes being collected later). The employed flow-profile strategy produces only a small share of all attainable barcodes.

tire system should require minimal operator interaction and no additional fluid to seal the droplets from ambient air.

The initial storage system was designed to store droplets in inverted plastic vials due to their availability and excellent chemical compatibility. However, reliably interfacing microfluidic devices with polymer vials proved challenging. Therefore, using additive manufacturing (3D-printing), we fabricated a connector that facilitates interfacing the storage vial with a microfluidic device (Figure 3.9 a). Additive manufacturing allows for facile and rapid prototyping, and the use of a multi-jet modeling 3D-printer (ProJet 3510 HD, 3D Systems, Rockhill, USA) provided for a feature resolution below 100 µm. The tubing which delivers the droplets from the microfluidic device into the storage system is directly connected to the 3D-printed connector (Figure 3.9 a). A plastic vial is then placed upside-down over the hollow pillar in the center of the square reservoir. This vial is held in place and sealed by firmly pressing it onto the 3D-printed collar at the bottom of the pillar. This fixation method was adopted since no additional adhesive is required, and replacement of the storage vial between separate storage cycles is facile.

Prior to operation, the assembled droplet storage system is primed by injecting continuous phase through the connected tubing, and filling up the storage vial and main reservoir. The emulsion is then pumped into the connector block. This emulsion descends through the printed channel in the connector block and enters the hollow pillar from the bottom. Once droplets reach the top of the pillar, they enter the inverted plastic vial. Due to buoyancy, droplets remain at the top of the vial, with the displaced continuous phase being continuously released into the main reservoir through the channels at the bottom of the vial. Droplets may be retrieved by reversing the flow of the continuous phase through the same tubing, and since droplets are located near the entrance of the hollow channel (at the top of the pillar) they are readily aspirated and rapidly returned to the microfluidic device.

To minimize use of excessive amounts of continuous phase in the storage reservoir, we produced the improved design shown in Figure 3.9 b, where the reservoir is replaced by a long section of tubing. This design has the additional benefit that negative pressure is now not needed to retrieve droplets, with retrieval being possible by application of positive pressure from the oil outlet. However, due to frequent fracturing of the interface between the 3D-printed connector block and the plastic vial, we chose to test a completely 3D-printed storage chamber (see Figure 3.9 c). This device in-

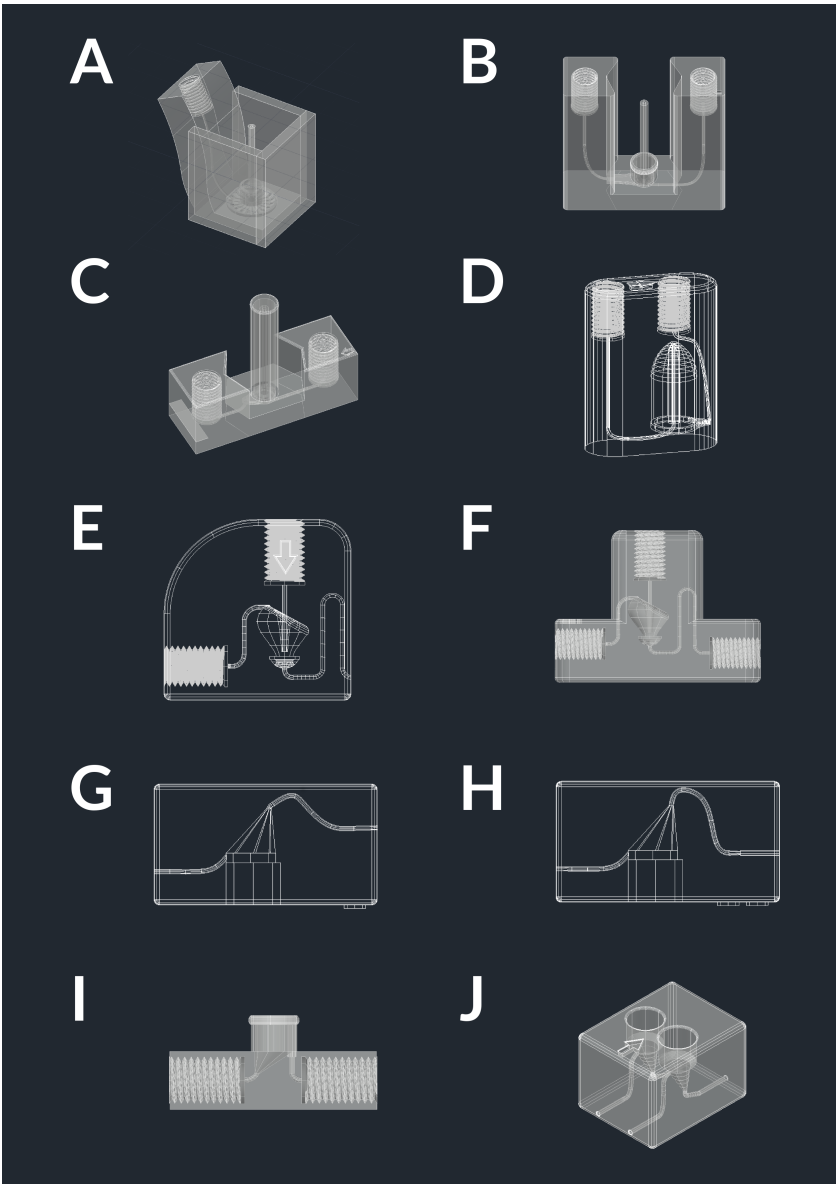
A
B
C
D
E
F
G
H
I
J

FIGURE 3.9 *(previous page)*: Droplet storage chamber designs. (A) An inverted plastic vial is placed over a pillar in the center of the chamber and the complete reservoir is filled with continuous phase. The storage chamber is connected to the microfluidic device using a screw connector. Droplets can be deposited by introducing them into the primed storage chamber, and moving them up through the hollow pillar to the top of the vial. Retrieval is performed by reversing the flow (through the application of negative pressure). (B) Storage chamber design that eliminates the need for excessive volumes of continuous phase and an open container. Here, excess continuous phase is released through a secondary access port. (C) A fully 3D-printed storage chamber. This device does not require any assembly but the removal of excess support material in the manufacturing process remains challenging. (D) A bullet-shaped storage chamber that increases storage size and retrieval efficiency due to droplets rising to the top of the chamber. (E,F) Three-way storage chamber designs, that allow separation of excess continuous phase through a third access port. Droplets are retained in the storage chamber due to buoyancy. (G-H) Chamber designs using device inversion for droplet retrieval. A glass vial is glued into the round slot at the bottom of the device (Figure 3.10 illustrates an assembled device). Furthermore, screw connectors were replaced by metal connectors to limit vibrations during setup. (I) A variant of the inversion device, involving minimal internal volume and screw connectors. (J) An array of the final droplet storage device, used for parallel experimentation.

corporates a printed storage chamber that closely resembles the plastic vial used during initial experiments. Devices containing a bullet-shaped storage chamber were also assessed to increase the amount of emulsion stored (see Figure 3.9 d). However, it was found that droplet retrieval required excessive continuous phase flows and thus yielded sparse (or highly diluted) emulsions. To increase droplet concentrations, a three-way design based on a buoyancy filter (which simultaneously stores the droplets and discards excess continuous phase for recycling) was also assessed (see Figure 3.9 e-f). Such a design typically retains droplets through buoyancy, discarding excess continuous phase, and allows for retrieval of concentrated emulsions.

The additive manufacturing process employed in the current experiments involves filling hollow spaces with a paraffin-based resin (of proprietary composition), which must be melted and removed after completion of the printing process. Removal of the filler resin becomes increasingly challenging as the size of access ports reduces, with increased amounts of resin remaining trapped inside the printed structure. Thus, after extensive experimentation, we reverted to the use of hybrid devices instead of fully 3D-printed chambers, with a glass vial being used instead of a plastic vial due to its chemical inertness.

Additionally, it was found that storage systems using a pillar often do not allow retrieval of all droplets if flow rates are too low, since the distance between the top of the pillar and the storage vial could not be reduced sufficiently. Accordingly, we removed this pillar from subsequent designs and used an adhesive to attach the glass vial to the 3D-printed connector piece. Crucially, we also introduced the use of an inverted storage vial, where droplets are stored in an upright chamber and rise to the top of the storage vial through buoyancy. Droplet retrieval is subsequently performed by inverting the storage structure and applying a small positive pressure on the outlet (see Figure 3.9 g-i). Additionally, metal pins (glued to the 3D-printed connector piece) were used instead of screw connectors, since vibrations caused by the connection process induced unacceptably large amounts of droplet merging. Due to excellent droplet storage and retrieval performance, we fine-tuned the internal structure of the connector block (Figure 3.9 g-i) and began producing arrays containing multiple but separate storage modules within a single device (Figure 3.9 j).

Figure 3.10 presents the working principle of the developed droplet storage chamber. Initially, the upright device is used to store an emulsion (Figure 3.10 a). When droplets enter the glass vial through the right inlet, they

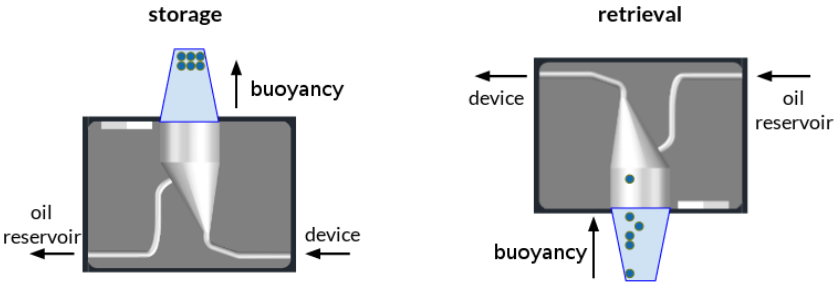

FIGURE 3.10: Droplet storage via device inversion. (A) During storage, an emulsion is introduced into the device from the right inlet. Due to buoyancy, droplets rise to the top of the storage vial, while excess oil drains through the oil outlet. (B) Droplets can easily be retrieved and pushed back into the microfluidic device by inverting the storage chamber and reversing the flow of the oil outlet.

float upwards due to buoyancy, with excess oil draining through the outlet. Crucially, displacement due to lateral flow of the continuous phase is smaller than vertical displacement due to droplet buoyancy, allowing complete capture of droplets in the storage vial. To retrieve droplets, the storage system is simply flipped upside-down. At this point, droplets enter the connector block again, due to buoyancy. Once a small flow of continuous phase is introduced to the outlet, droplets are pushed back into the microfluidic device through the same tubing through which they were delivered, requiring no additional fluidic connections.

3.3 MATERIALS AND METHODS

3.3.1 *Experimental Setup*

Microfluidic devices were manufactured according to standard soft lithographic techniques, which are described in more detail in Chapter 2 (*119*). Microfluidic droplet sorting devices were modelled after the device shown in (*49*) (see Figure 3.6 a). Channel height was 50 µm. We used multiple inlets to deliver several fluorophore and buffer solutions and employed on-chip mixing to produce the desired droplet composition prior to flow-focusing.

Droplets contents were excited using both a 488-nm laser and a 640-nm laser (Laserage-4; Omicron, Rodgau, Germany). A combined laser beam was shaped into a light sheet, approximately 5 µm thick, using a cylindrical lens (LJ1558RM f = 300 mm; Thorlabs, Newton NJ, USA) and aligned perpendicular to the microfluidic channel. An inverted microscope (Nikon Ti-E; Nikon AG, Egg, Switzerland) with a 20X objective (Nikon GmbH, Egg, Switzerland) was used for all experiments, as both excitation and detection can be performed from the underside. This configuration provides easy access to the microfluidic device from the top, facilitating connections to the syringe pumps and the signal amplifier. Fluorescence emission was collected and separated from the excitation light using a dichroic mirror (AT DC 505; AHF, Tübingen, Germany) and quantified using two photomultiplier tubes (H10722-20; Hamamatsu, Solothurn, Switzerland). We additionally employed a bright-field high-speed camera (Phantom Miro M310, VRI, Wayne, USA) to observe the microfluidic flows during experimentation.

Droplets were formed at a flow-focusing junction by co-flowing an aqueous dispersed phase with a fluorinated continuous phase (HFE 7500; 3M, Rüschlikon, Schweiz) as the continuous phase. Additionally, the continuous phase contained 0.5 % *wt/wt* of a PEG-block co-polymer (*23*) (008-Fluorosurfactant; RAN Biotech, Beverly, MA) to stabilize the resulting emulsion. In initial experiments, we used a 100 µM solution of fluorescein (Sigma-Aldrich, Buchs, Switzerland) to monitor droplets. Subsequently, PBS buffer (Sigma-Aldrich, Buchs, Switzerland) solutions containing 100 µM CF647 or CF488 (Sigma-Aldrich, Buchs, Switzerland) were used as precursor dye solutions for barcoding experiments. All fluids were pumped using Nemesys low-pressure syringe pumps (Cetoni GMBH, Korbussen, Germany) and 1 ml glass syringes (Gastight 1001, Hamilton, Bonaduz, Switzerland). Pumps were connected to the microfluidic device using PTFE tubing (Adtech Polymer Engineering Ltd., Stroud, UK).

3.3.2 *Software*

The PMT voltage (an analog signal corresponding to the fluorescence signal) was recorded using a multifunction FPGA module (PCIe-7842R, National Instruments, Ennetbaden, Switzerland). The FPGA module was programmed using LabView 2014 (National Instruments, Ennetbaden, Switzerland). The connected workstation ran a separate control and monitoring application, also programmed in LabView. All signal processing and sort-

ing decisions are made by the FPGA module independently, with only observations and results being reported to the monitoring application. As the experimenter exclusively interacts with the monitoring application it also relays parameter changes (e.g. during calibration) to the FPGA device, albeit not in real-time (but with a 1-3 seconds delay). The square sorting signal is generated by the FPGA module and amplified 1000 times using a high-frequency and high-voltage amplifier (623B; Trek, Lockport, USA).

3.3.3 *Barcoding Experiments*

Before barcoding, the microfluidic device, as well as the droplet storage chamber were flushed with continuous phase (HFE 7500; 3M, Rüschlikon, Schweiz). Subsequently the dead-end electrode channels were filled with a 2 M aqueous NaCl (Sigma-Aldrich, Buchs, Switzerland) solution by applying pressure to the solution. Due to the applied pressure, air initially occupying the electrode channel diffuses through the thin PDMS membrane separating the electrode channel from the main channel. After the electrode channels are filled with the electrode solution, the amplifier is connected to the syringe containing the electrode solution using clamp connectors. All dispersed phases were connected and pumped into the device until stable droplet formation was achieved. The electrophoretic sorting signal consists of a square wave with the following adjustable parameters: square wave amplitude (500 V - 2000 V), square wave frequency (5 - 200 kilohertz), number of square pulses (1 - 50) and delay of the sorting pulses (10 µs – 5000 µs). Whilst most sorting parameters can be fixed between experiments, the sorting delay must be calibrated separately and prior (approximately 1 millisecond) to each experiment. For delay calibration, a flip-flop sorting algorithm, which detects all droplets and sorts out every second droplet it encounters, is used. High-speed observations, are used to subsequently adjust the delay, such that the sorting signal is correctly timed to exclusively sort single droplets, while not influencing the path of preceding or following droplets.

The detected barcoding signal is directly correlated to the laser power and PMT amplification voltage and thus must be calibrated prior to each experiment. To this end, we produced droplets containing the maximum dye concentrations of both dyes, and subsequently tuned the maximum signal intensity to cover the full range of the detectors. Next, barcodes were generated and sorted using our automatic algorithm. Generated barcodes were stored in the developed droplet storage device. Finally, we employed

custom Python scripts to post-process and visualize data produced during barcoding, allowing for confirmation of the number of unique barcodes produced. We could further produce kernel-density estimates using Gaussian kernels to identify regions in the barcode phase space which were poorly occupied (as observed in Figure 3.8). During barcode generation, the developed feedback algorithm was used to selectively produce novel fluorescent signatures.

3.3.4 *3D-printed Storage Chamber*

3D-printed devices were manufactured using a multi-jet modeling (MJM) 3D-printer (ProJet 3510 HD; 3D Systems, Rockhill, USA) from a proprietary substrate (Visijet M3 Black; 3D Systems, Rockhill, USA) with excellent strength and flexibility properties. During the printing process, hollow spaces are filled using a paraffin-like proprietary substance (3D Systems, Rockhill, USA), to allow for the construction of roof-like structures over hollow spaces.

We employed a multi-step protocol to remove the paraffin support structure after 3D-printing. Initially 3D-printed devices were placed in an oven at 70 °C overnight. This leads to the melting and evacuation of significant amounts of paraffin wax. Subsequently, channels were cleaned using high pressure steam, which allowed for removal of most of the paraffin trapped in the siphon-like channels. Next, interior surfaces of the printed devices were cleaned by sonication (FB15053; Thermo Fisher Scientific, Reinach, Switzerland) in fresh aqueous 2 % *wt/wt* sodium dodecyl sulfate (Sigma-Aldrich, Buchs, Switzerland) solution at 50 °C for 40 minutes and subsequently drying using pressurized air. The remaining surfactant solution was removed by sonicating devices twice for 15 minutes using distilled water. Finally, a glass vial (200 µL conical glass insert; BGB Analytik, Böckten, Germany) and steel connector pins (OD: 0.25 mm, ID: 0.13 mm; New England Small Tube Corporation, Litchfield, USA) were attached to the clean structure using a two-component epoxy adhesive (Araldite Standard; Huntsman Advanced Materials, Basel, Switzerland).

Storage performance of the 3D-printed droplet storage chambers was determined both quantitatively and qualitatively. The resolution of the 3D-printed materials was assessed by observing printed devices using a stereoscope (SMZ1500; Nikon GmbH, Egg, Switzerland). Storage performance was assessed by observing droplets as they exit the storage chamber under the stereoscope, after a 2-hour incubation period. This allowed for the de-

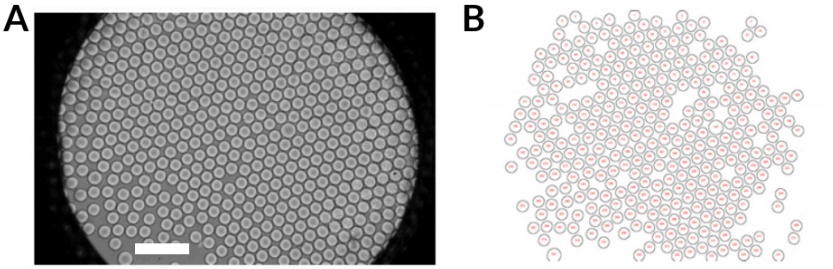

FIGURE 3.11: Determination of droplet diameters using ImageJ (scale bar - 200 µm). (A) Droplets exiting the droplet storage chamber were deposited on a glass slide and imaged using a stereoscope. (B) Raw images were processed using ImageJ to quantify droplet diameters. Retrieval rates of stored droplets were quantified by counting a precise number of droplets into the droplet storage chamber using the droplet sorting architecture. After incubation for 2 hours, droplets were re-introduced into the device and counted.

tection of droplet merging or splitting events, through the observation of differently sized droplets. Image processing was performed using ImageJ (Figure 3.11) (*136*).

3.4 RESULTS AND DISCUSSION

3.4.1 *3D-printed Storage Chamber*

During initial assessment of the additive manufacturing resolution, we concluded that channel diameters down to 200 µm and channel lengths of up to 5 centimeters could be manufactured in a reproducible fashion. Figure 3.12 presents magnified images of droplet storage chambers made using the multi-jet modeling printer, captured using a stereoscope. We further quantified droplet merging and splitting after 2 hours of storage, by measuring droplet radii from images captured using a stereoscope (as shown in Figure 3.11). Using such an approach, it was found that on average 0.5 % of droplets showed a different diameter (resulting from droplet merging or splitting) upon retrieval.

Droplet retrieval rates were quantified using separate storage and retrieval sequences with intermediary incubation periods of two hours. Using this strategy, we could demonstrate that the droplet storage chamber

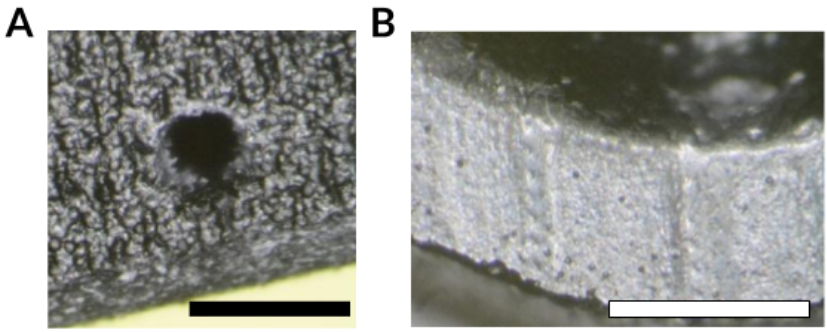

FIGURE 3.12: Assessment of additive manufacturing resolution. Magnified images of the finished 3D-printed devices were obtained using a stereoscope and served as exemplary resolution checks of the employed additive manufacturing process. Results indicate that channels of less than 200 µm diameter can be manufactured in a reproducible fashion. All scale bars are 1 mm in length.

allows for droplet retrieval rates exceeding 99 %, with experiments indicating that only 0.44 % ± 0.69 % (N=4) of all stored droplets cannot be retrieved. Such excellent retrieval rates highlight the potential of 3D-printing as a manufacturing tool in microfluidics. Indeed, additive manufacturing methods enable the rapid prototyping of complex structures in an automated fashion.

That said, 3D-printing technologies still possess limitations, and thus should be applied with caution. The resolution achieved by most commercially available 3D-printers is insufficient to fabricate microfluidic channels, which typically have cross-sectional dimension below 100 µm. Moreover, additive manufacturing methods can only be applied to a select number of substrate materials. Whilst a variety of polymers and metals have been used for 3D-printing purposes, there remains a need for more specialized materials, such as transparent or elastomeric materials. Furthermore, many 3D-printable materials are proprietary mixtures, with exact chemical compositions and material properties being undisclosed. Nevertheless, the current study confirms that 3D-printing allows for the rapid prototyping of complex structures (such as siphons and chambers) on sub-millimeter scales, unrivalled by traditional fabrication techniques. Accordingly, we recommend the integration of 3D-printing methods into the traditional microfluidic workflow, particularly when used to interface microfluidic devices with peripheral devices (the chip-to-world interface).

3.4.2 *Single-color Barcodes*

Initial droplet barcoding experiments were performed using a single-dye system. Droplets with different fluorescence intensities were produced by mixing a single fluorescent dye with a PBS buffer solution on-chip, prior to droplet formation. Figure 3.13 shows data from two barcoding experiments, where each measurement (dot) in the scatter plot corresponds to the fluorescence intensity of a single droplet passing through the detection volume. Regions in the phase space, corresponding to the unique barcodes, are marked as green areas, with identified and sorted droplets being shown as red dots. The average fluorescence intensity of droplets shows periodic oscillation due to varying dye concentrations. The one-dimensional barcoding system was also used to develop standard procedures for the calibration of the excitation source power, signal amplification in the detector and droplet sorting parameters (see Experimental Methods).

Figure 3.13 a highlights an experiment, where only two separate barcoded droplets are requested by the user. Accordingly, the two desired phase space regions in the attainable fluorescence intensity range are maximally separated. Since only two different fluorescent signatures are required, these regions can be made relatively wide (approximately 0.4 V width), whilst still allowing for proper identification of droplets. Due to the rapid droplet generation frequency relative to the change in color, we were able to observe approximately 200 separate droplets during one pass of a region and around 500 droplets between separate regions. The first droplet entering a region is sorted into the collection outlet (marked in red), with all subsequent droplets being ignored until the next region is reached. We thus conclude that the developed barcoding system can reliably identify and sort novel fluorescent signatures from a stream of droplets in real-time.

The proposed barcoding system easily scales to multiple barcodes per fluorescent dye. Indeed, Figure 3.13 b shows the results for a barcoding experiment identifying up to 34 individual barcodes. Due to the high number of barcodes compared with the experiment shown in Figure 3.13 a, the width of the barcode regions is reduced significantly. Due to this reduced width and the steep flow rate gradient, a single barcode is "missed" during the first pass (approximately droplet number 5100). However, a droplet with the required fluorescent signature is found and sorted during the second pass (approximately droplet number 9000). Therefore, at a droplet generation frequency of approximately 500 Hz all requested barcodes could

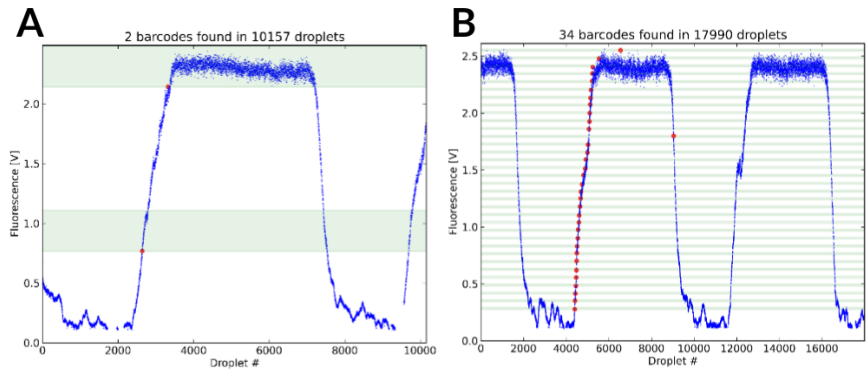

FIGURE 3.13: Single-color barcoded droplet populations. (A) A barcoding experiment that sorts two separate barcoded droplets. Droplets containing varying concentrations of a fluorescent dye are produced using gradient flow profiles. The scatter plot shows fluorescence intensities of subsequent droplets (small blue dots). Green areas highlight the desired fluorescence intensities of all requested barcodes. The first droplet exhibiting the desired fluorescence properties is sorted from the droplet stream and diverted to a separate outlet (red circles highlight successful sorting events). As can be seen the developed barcoding algorithm correctly sorts the first droplet arising in each area and ignores subsequent identical barcodes. (B) Larger scale single-color barcoding experiment identifying 34 separate barcodes. Interestingly, due to the resolution of the colored droplet population not all barcodes could be identified during the first passage, with the missing barcode being sorted during a subsequent pass.

be identified and sorted in less than 20 seconds. Such performance confirms the high-throughput capabilities of the developed droplet barcoding system, especially when compared to typical experiment setup times of approximately 30 minutes.

Since the widths of desired phase space regions are independently adjustable, it is apparent that such widths should be minimized. This results in more precise droplet barcodes at the cost of only marginally increased experiment times. However, the maximal number of barcodes per color will be limited by detector resolution. Indeed, in our experiments we found that scaling beyond 50 barcodes does not allow for precise barcode identification due to diminishing intensity differences between adjacent barcodes. In addition, it should be noted that the recorded fluorescence intensity gradient does not directly correspond to the linear flow profile of the pumps. Particularly, the plateaus at the top and bottom of the fluorescence gradient and the rapid change in between are in stark contrast to the linear flow profile run on the syringe pumps. We believe that this discrepancy arises due to the inherent flexibility of the PDMS device and tubing used and the pinched flow regime present at the mixing junction. However, in our system such fluctuations do not influence barcode quality as the production of barcodes and the subsequent droplet-sorting logic are completely independent. Nevertheless, a tighter coupling between flow profiles and measured fluorescence intensity could be beneficial in improving barcoding speed and allowing for faster production of barcoded droplet populations.

3.4.3 *Two-color Barcodes*

As the number of possible barcodes identifiable by a single detector is limited, scale up of the number of unique barcodes using a single fluorescent dye is challenging. However, the use of combinations of multiple dyes encapsulated in droplets exponentially increases the number of accessible barcodes. Since it is comparatively easy to add another fluorescent dye and another detector into our system, this approach is well suited for the generation of large numbers of barcodes. At a general level, the number of possible barcodes is given by the number of variations with repetition, i.e.

$$n_{barcodes} = \prod_{i=1}^{n_{colors}} codes_i$$

where codes_i corresponds to the number of barcodes that are attainable using color_i given the corresponding point detector resolution.

We found that the use of more than one fluorescent dye required the implementation of a more complex flow profile strategy. Accordingly, we employed a flow rate feedback algorithm that allows for dynamic adjustments to pump flow rates (see Flow Profile Strategies section). Figure 3.14 highlights a large-scale experiment designed to collect the maximum number of unique barcodes using two fluorescent dyes. Each axis corresponds to the fluorescence intensity of a single detector and each data point corresponds to a collected barcode. The color of the data points represents the time, when the barcode was identified and sorted (darker points were collected later during the experiment). We further used Gaussian kernel density estimates to highlight regions with high barcode occupancy (blue overlay) and low barcode occupancy (white overlay). To simplify visualization, the plot omits non-sorted droplets and only shows sorted droplets. Compared to single-color experiments (Figure 3.14, allowed regions in the two-dimensional barcoding experiments are now constrained to two dimensions (shown as green areas). Due to the required separation, these allowed phase space regions are arranged on a regular grid.

Similar to single-color barcodes (Figure 3.13), one can observe that the sorted barcodes typically occur along the border of the respective region, as only the first droplet "entering" the region is sorted. We aimed at collecting 47 and 48 different levels of fluorescence intensity per detector respectively, resulting in a theoretical maximum of 2256 possible barcodes and managed to collect 2205 (97.7 %) out of all possible barcodes in less than 25 minutes. Given the presented data, it is evident that the largest region of missing barcodes theoretically contains high concentrations of both dyes. We expect, that these missing barcodes could be produced and collected by tuning the buffer flow and lowering the maximum fluorescence at which barcodes are collected. Moreover, we believe that this effect is likely to stem from the slight overlap of the emission spectra of the dyes, and the number of accessible barcodes could potentially be increased by using fluorophores with more clearly separated spectra (*137*). We also found, that peripheral barcodes were identified later (identified by darker data points), when compared to central barcodes within the phase space diagram. This strongly suggests that the developed feedback algorithm works well, correctly identifying peripheral regions with missing barcodes and successfully producing the required flow rates at later times during barcode generation.

To highlight the robustness of the developed barcoding system, we repeated the experiment shown in Figure 3.14 three times, plotting the frac-

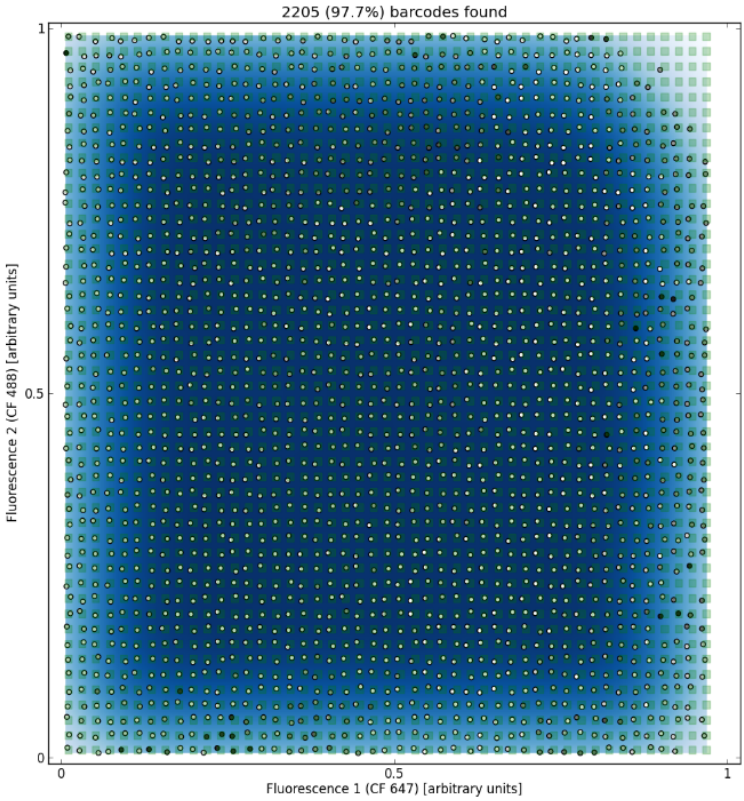

FIGURE 3.14: Large-scale droplet barcoding. Barcodes are produced using two fluorescent dyes and the presented barcoding strategy. Each possible fluorescent signature is represented with a green square, and is well separated from adjacent barcodes. Data points indicate successfully identified and sorted barcodes. The color of the data points indicates when the barcode was found (darker droplets were found later), highlighting initially missing regions and allowing us to judge the performance of the flow rate feedback algorithm. A kernel density estimate (blue overlay) is used to identify high density regions. By employing the developed flow rate feedback algorithm, we could increase the share of barcodes produced to 97.7 % of all possible barcodes (2205 out of 2256 barcodes collected during the highlighted experiment). Missing barcodes are predominantly associated with high concentrations of both fluorescent dyes. We hypothesize that this effect could be mitigated by further fine-tuning of detection intensities and barcoding bins, as well as using fluorescent dyes with more clearly separated emission spectra (*137*).

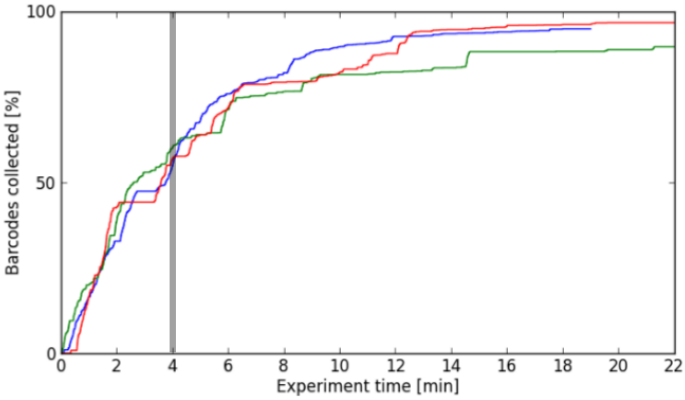

FIGURE 3.15: Droplet barcoding reproducibility. Variation of the number of barcodes collected as a function of time for three separate experimental repeats of the experiment shown in Figure 3.14. It is observed that barcode collection is an asymptotic process with more than 50 % of possible barcodes being collected within the first 4 minutes of experiment time (highlighted in Figure). Furthermore, all experiments identify at least 90 % of obtainable barcodes within a period of 20 minutes.

tion of barcodes collected as a function of time (Figure 3.15). For each experiment, the number of barcodes collected over time shows the expected asymptotic profile of a semi-random collection process, with collection frequency decreasing proportionally to the number of remaining barcodes. It is further observed that 50 % of barcodes are typically collected within the first 4 minutes of an experiment (highlighted in Figure 3.15). We hypothesize that step wise increases in the number of barcodes stem from the employed flow rate feedback algorithm, which successfully identifies regions with missing barcodes, resulting in a temporary surge in the production of novel barcodes.

3.5 CONCLUSION

The studies presented in this chapter have successfully described the design and implementation of a droplet barcoding strategy, able produce, code and store more than 2200 droplets, while maintaining flexibility with respect to fluorophore type and barcode number. To our knowledge, the formed droplet library defines the largest number of fluorescently barcoded droplets yet reported. Based on these initial experiments, we hypothesize that increasing the number of distinct dyes to four will enable the production of more than one million unique droplet barcodes (32^4 = 1048576). This in turn, will permit repeated readout of large droplet populations and facilitate key experiments, such as single-cell tracking.

Furthermore, the 3D-printed droplet storage device developed in this study was shown to enable long-term droplet storage, whilst ensuring near loss-free droplet retrieval. We believe the ability to incubate droplet population in a reproducible fashion, will greatly facilitates the implementation of biological assays based on droplet-based microfluidic technology. Since the long incubation periods required for the cultivation of many biological samples leads to lower experiment throughput, the low-loss nature of the developed droplet storage device will directly enable the increased collection of experimental results per cultivation effort spent.

While microfluidic geometries should still be fabricated using conventional and soft lithographic techniques, we found that additive manufacturing is well-suited to the fabrication of larger structures on the millifluidic or mesofluidic scale. 3D-printed devices have great potential, particularly at the chip-lab interface as connectors, gaskets or incubation chambers. Indeed, since currently there exists no standard format for microfluidic devices, additive manufacturing also allows the experimenter to adjust de-

signs and produce interfaces that are customized to a specific application. Accordingly, we propose that even though improvements in resolution and material choice are needed, additive manufacturing shows great potential as a component tool in microfluidic research.

Whilst the proposed barcoding system relies on existing sorting architectures, we were only able to achieve the presented results by incorporating improved decision-making processes. Even though active microfluidic devices typically increase experiment complexity, this tradeoff is often justified, as passive devices require a large number of design iterations due to a general lack of simulation tools. Therefore, we assume that despite their complexity, other active microfluidic techniques, such as droplet screening or long-term reaction monitoring, could greatly profit from improved control algorithms.

In conclusion, the developed methodology has the potential to greatly increase the number of biological and chemical assays that can be performed using droplet-based microfluidic strategies, by enabling the repeated scanning of large droplet populations with single droplet resolution. When scaled-up, this general approach will allow the use of droplet-based microfluidics in chemical/biological assays that are currently only feasible using inferior microtiter plate technologies. Finally, we believe that this study highlights the benefits of using "smarter" control algorithms in conjunction with existing experimental formats.

4

REINFORCEMENT LEARNING FOR MICROFLUIDIC CONTROL

4.1 INTRODUCTION

In recent years high-throughput screening (HTS) or more generally high-throughput experimentation (HTE) has attracted significant interest in the chemical and biological sciences due to the increased availability of large compound libraries (*138*). HTE allows for rapid screening and evaluation of such molecular libraries. Indeed, HTS technologies are the driving force behind many novel medical diagnostic assays, such as single-cell sequencing, which depends on a high-speed screening capability to analyze large numbers of cells within a reasonable time frame.

Unsurprisingly, microfluidic technologies have emerged as a powerful tool in a range of high-throughput screening assays (*139*, *140*) because the miniaturization of functional operations and analytical processes is almost always accompanied by a number of inherent advantages when compared to the corresponding macroscale process. Such advantages include significant reductions in sample and reagent consumption, decreased assay costs and high analytical throughput (*1*). Additionally, the high surface area-to-volume ratios typical in microfluidic environments ensure that both heat and mass transfer rates are enhanced, allowing for unrivalled control over the chemical or biological environment.

Although a range of high-throughput chemical and biological screens have been realized using microfluidic techniques, workflows that incorporate ad hoc decision making at high speed and over extended time periods have proved to be immensely challenging to implement. Due to the speed with which results are evaluated during HTS experiments, real-time processing capabilities are often limited to simple thresholding operations (such as sorting droplets based on fluorescence intensity) (*141*). For example, in Chapter 3 we show a droplet barcoding strategy that is based on sorting droplets at around 500 hertz but requires implementation using FPGA modules to achieve sufficient processing speeds. More complex decision processes on microfluidic devices, involving image-based cell sorting for example, have thus not been shown to date.

The proportional–integral–derivative (PID) controller (*142*) and its variants have been extensively used to control reaction conditions within microfluidic devices. For example, PID controllers have been used to good effect in controlling temperature within nL-volume chambers during on-chip PCR (*143*) or improving the process of droplet generation within electrowetting devices (*144*). That said, PIDs are quite limited in their general applicability. In simple terms, a PID controller looks at a "set-point" (defined by the user) and compares it with the actual value of the process variable. This is effective in maintaining "homeostasis", but less useful when exploring complex environments (or parameter spaces). Further, due to their one-dimensional nature it is a non-trivial process to extend their use to multi-dimensional input and output data, without exploiting multiple separate PID controllers in a parallel fashion. Finally, PIDs assume a symmetric environment, which is frequently not the case, for example a temperature controller may heat the system but might not have the ability to actively force the system to cool down other than waiting. Such unsymmetrical environments therefore require additional controller tuning (such as over-dampening) to ensure consistent performance (*142*). In contrast, the use of an adaptable control algorithm allows the experimenter to perform a variety of assays using the same equipment and algorithm. In this respect, machine learning methods constitute a promising approach to developing flexible control algorithms, as they "explore" algorithms able to adapt to novel challenges and make predictions purely based on experimental data (*75*).

4.1.1 *Machine Learning*

In simple terms, machine learning describes a class of algorithms that enable computers to improve their ability to perform a given task (i.e. learn) without being explicitly programmed (*145*). Although, originally conceptualized in the late 1950s (*67*), it is only in the last few years that the field of machine learning has received enormous research attention. This is primarily due to improved access to and availability of computational resources, which has in turn generated practical results (*146*). Nevertheless, the current generation of algorithms largely focus on achieving high performance of specific tasks while general artificial intelligence (high performance across a wide variety of problems) remains elusive (*147*). Accordingly, state-of-the-art results typically only manage to exceed human-level performance in a narrow problem domain, often related to pattern match-

ing (*148*, *149*). Indeed, machine learning algorithms have for example been shown to excel at recognizing objects in images (*148*) or translating text between languages (*149*).

Machine learning algorithms can be broadly sub-divided in two separate categories, namely, those that incorporate supervised learning or unsupervised learning strategies (*150*). The vast majority of recent studies in the field have been obtained using supervised learning approaches (or variants thereof). In simple terms, a supervised learning algorithm attempts to understand a problem domain using a dataset of known inputs and associated outputs. Through this dataset, the algorithm learns to map input data to expected output data. Importantly, if the training dataset represents a sufficiently large proportion of the problem space, the algorithm will be able to make "educated guesses" regarding the output of novel inputs within the same problem space. Additionally, several variants of supervised learning have been developed for tasks where only partial input data are available (*151*, *152*). Semi-supervised learning algorithms aim to produce an estimate, when some of the output data is missing but complete input data is available (*151*). On the other hand, in reinforcement learning (*152*), training data only becomes available after interaction with an environment. Therefore, the environment produces the target data online (during the training phase).

In contrast, unsupervised learning algorithms aim to find underlying structure in input data without any target output data being available. As such, an unsupervised algorithm can for example cluster similar images without any previous knowledge of the sorting criteria, by attempting to find the maximal separation between images. If a meaningful underlying pattern is found for a given input dataset, the pattern can subsequently be used to simplify novel input data. For example, an image might be simplified as a linear combination of a small set of component images using unsupervised learning (*153*).

An important distinction should also be made between the processes of altering and retraining an algorithm. Whereas a machine learning algorithm is typically designed to perform well within a specific problem domain, it can be further specialized in its scope based by training (using a specific training dataset). Accordingly, a given algorithm can be re-trained and adapted to a different problem using a novel training dataset, assuming that the problems are sufficiently similar in nature. As an example, image classification algorithms that are able to recognize cells within images, could be "retrained" to recognize animals without requiring any algorith-

mic changes. However, the algorithm will most likely be poorly suited to the analysis of videos of cells flowing through a microfluidic channel, because it will not capture the temporal relationship between video frames. Therefore, a novel (altered) algorithm is only required if the problem domain is sufficiently different.

4.1.2 *Artificial Neural Networks*

First developed in the 1940s (*154*), artificial neural networks (ANNs) are a class of algorithms inspired by (biological) neural networks found in animal brains. Such systems are made up of a collection of connected units (termed artificial neurons), where connections between units are weighted. When signals (scalar values) are propagated between neurons through these weighted connections, the signal value is either amplified or dampened based on the weight of the connection. Typically, neurons in ANNs are organized into multiple layers, with data travelling linearly from an input layer through a series of hidden layers and being read back from an output layer.

Figure 4.1 a shows a simple example of an ANN; in this case a multilayer perceptron containing a single hidden layer (*67*). Each neuron in the input layer receives a scalar input; for example, the gray-scale value of a single pixel within an image. During evaluation, the input values are propagated through several weighted connections to the neurons in the hidden layer. Each neuron in the hidden layer aggregates the signal from all incoming connections: In a first step, the total activation of the neuron (total input signal) is calculated using the sum of all weighted input connections. Next, an optional transformation function (activation function) is applied to the total value (see Figure 4.1 b). The activation function typically suppresses signal noise and in turn allows the ANN to converge to a more stable solution. Popular activation functions include simple thresholding using a binary step function (see Figure 4.1 b), sigmoidal functions or more complex functions such as the rectified linear unit (ReLU) (*155*). The final activation value of the artificial neuron is then propagated to the next layer of neurons. This process is repeated for each neuron in each layer until a scalar value for the neurons in the output layer is generated. Thus, given constant weights, an ANN produces a deterministic output vector for each possible input vector.

During training and evaluation of an ANN, the network topology and other hyper-parameters (such as the type of activation function) are kept

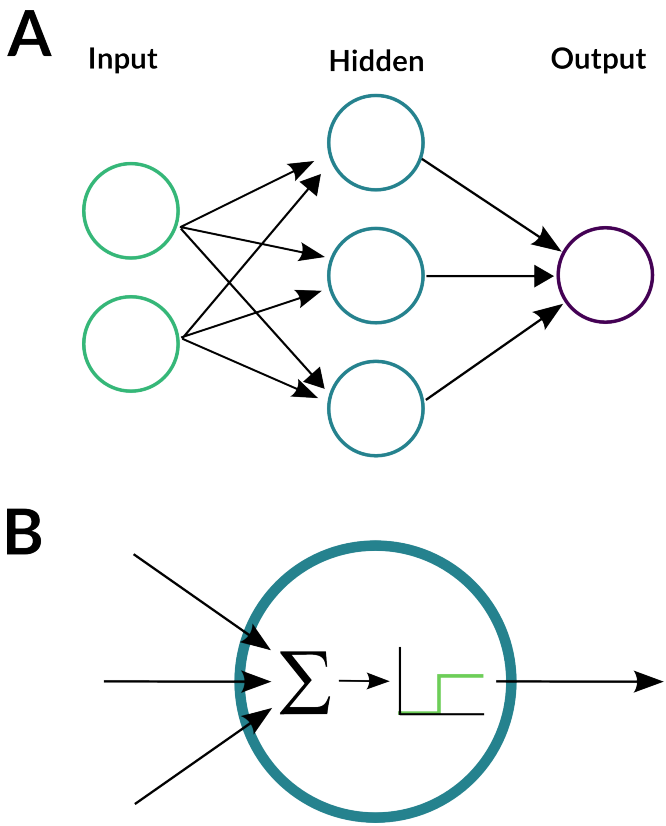
A
Input
Hidden
Output
B
Σ

FIGURE 4.1 *(previous page)*: Artificial neural network. (A) Topology of a simple artificial neural network (ANN). Circles represent artificial neurons, with arrows representing the weighted connections between such neurons. ANNs, such as the one shown, can be used to transform an input vector into an output vector based on the connections between neurons. Each neuron in the input layer receives a scalar input value, such as a single pixel of a grayscale image. These values are then propagated to the next layer through the weighted connections (arrows). In the shown example, each input neuron is connected to each neuron in the hidden layer. Neurons of the hidden layer subsequently aggregate all input values and further process and propagate the signal to the neurons in the output layer. The final output for a given input can then be obtained from the neuron in the output layer. An example output could be the probability that an image contains a certain object. It should be noted that the example shown represents one of the simplest possible artificial neural networks, with typical ANNs containing many layers with millions of neurons and more complex connection topologies. (B) An example operation of an artificial neuron. Initially the neuron sums all weighted input signals from connected upstream neurons. Often an activation function is applied to this aggregate signal, such as a simple thresholding operation. Finally, the computed scalar signal is propagated down-stream to all connected neurons.

constant. Accordingly, an ANN can conceptually be modeled as a black box function, which maps input data to output data on the basis of the weights of the connections between neurons. To learn correct outputs, supervised ANNs typically use a dataset containing known input-output data pairs. The weights of the connections between artificial neurons are then successively adapted to achieve the best output signal using an optimization algorithm, such as backpropagation (*156*). During backpropagation, an output vector is first calculated based on a given input vector and the current set of weights. Next a loss function is applied. This yields a scalar that quantifies the divergence between the predicted output value and the known, true, output value. The gradient of this loss function is then calculated for each connection in the network, essentially quantifying how much each connection contributes to the final divergence. Finally, each weight is adjusted along its gradient, using gradient descent optimization. Thus, each connection within the ANN attempts to find a local minimum for its weight, such that it contributes as little as possible to the final prediction error. To achieve the highest accuracy output, it is crucial to alter the involved weights using small and incremental steps. Accordingly, an ANN often requires many training passes (typically millions of example frames) before it is fully optimized (see Figure 1.3 from (*75*)). During the training process, the structure contained in the training dataset is gradually transferred into the weights of the ANN. It should therefore be noted that the trained ANN is truly a black box, since the final weights are not interpretable anymore. For example, it is rarely possible to observe the value of a single neuron in a hidden layer and interpret its meaning or significance in the complete decision process.

Dramatic increases in computational power as well as insights into how many-layered ("deep") artificial neural networks can be trained, has led to a recent surge in the use of neural-networks in machine learning applications (*147*). Whilst a complete survey of recent advances in ANNs is beyond the scope of this chapter, we would like to highlight two developments that have proved critical in the interpretation of visual data, namely graphics processor unit (GPU)-based training (*146*) and convolutional neural networks (*157*).

Although the theoretical basis of many ANN-based models was introduced and developed in the 1980s and early 1990s, computational limitations prevented the implementation and training of large ANNs, made up of millions of artificial neurons and connections (for example 6.8 million optimizable parameters in (*158*)). To circumvent such limitations, software

developments in the early 2000s began to allow the use of GPUs in training and running artificial networks (*146*). GPUs were originally designed to rapidly manipulate memory to accelerate the creation of images in a frame buffer intended for output to a display. This operation involves the use of an algorithm that performs a relatively simple calculation for each pixel on a screen, in parallel, and at high frequencies (typically above 60 hertz for modern computers). GPUs were therefore optimized to perform many simple calculations in parallel, in contrast to the central processing unit (CPU), which can perform single but complex operations in series. Since most operations during ANN training and inference are easily parallelizable, GPUs proved an ideal computational medium, allowing for speed increases of several orders of magnitude (*146*). Unsurprisingly, to this day most ANNs are trained and evaluated using either a GPU or custom hardware (*159*).

The second important milestone in the interpretation of visual data using ANNs relates to the development of convolutional neural networks or CNNs (*157*). CNNs employ a specific connection scheme between artificial neuron layers, mimicking the network topology found in the animal visual cortex. Simple perceptrons (see Figure 4.1) connect each artificial neuron with each neuron in the following layer (to yield fully connected layers). The core assumption for CNNs is that pixels in close proximity are more related with respect to the output than pixels at larger separations. This is almost always true for visual data, since pixels close to each other are more likely to belong to the same object, whereas distant pixels will more likely be part of two separate objects.

Figure 4.2 shows a schematic of two layers in an ANN, connected via a convolutional connection scheme. An example neuron in the hidden layer is connected to a set of neurons in the input layer (the so-called receptive field). The neuron to the left of the example neuron typically employs an identical receptive field shifted one position to the left, with receptive fields of two neighboring neurons typically overlapping. Additionally, multiple neurons are used to learn separate representation of the same receptive field (visualized as a stack of neurons in Figure 4.2). A CNN typically consists of multiple convolutional layers in series, allowing the ANN to interpret the image using an increasing field of view. For example, if each neuron in the first convolutional layer interprets a 3x3 receptive field in the input image and each neuron in the second convolutional layer interprets a 3x3 receptive field of the first convolutional layer, then each second layer neuron can indirectly observe a 5x5 (due to overlap) receptive field

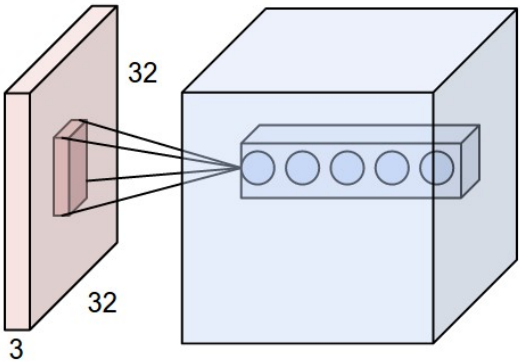

FIGURE 4.2: Schematic of the operation of a single convolutional neural layer. In contrast to the fully connected ANN shown in Figure 4.1 (layer 1), a convolutional neural network architecture only connects local neighboring neurons. Numbers indicate the size of the exemplary input matrix. Each neuron in the hidden blue layer is connected to a set of input neurons in the red layer. Each neuron in the blue layer thus has a specific field of view, illustrated in the schematic as a cone. The neuron thus receives input values from all neurons within the cone (shown as a selection of the neurons in the input layer). Multiple neurons typically share the same field of view (represented as a stack of neurons in the blue layer). Neighboring neurons (in the blue layer) will also have an equally sized receptive field but all positions in the input layer are shifted by one neuron. A convolutional ANN architecture generally works well if two adjacent input values are more closely related than two distant input values, which holds true for most visual observations.

in the original input layer. CNNs have several distinct advantages over fully connected layers, most notably they are shift-invariant and are thus able to give the same interpretation for images which are rotated or translated (*160*). Overall, CNNs have achieved state-of-the-art results in many applications, particularly for computer vision, where CNNs have been successfully used to categorize images into 1000 separate categories (such as "paintbrush" or "zebra") with very low error rates (5 %) (*148*).

4.1.3 *Reinforcement Learning*

Reinforcement learning (RL) describes a class of supervised machine learning algorithm inspired by behavioral psychology and is concerned with how a control algorithm (frequently called an agent) repeatedly interacts with an environment and iteratively maximizes a reward signal obtained from the environment (*152*). In simple terms, the agent observes the environment and performs an action in the environment based on its observation. The environment is then updated based on the action, and a scalar reward signal ("score"), representing in some way the quality of the action is returned (see Figure 4.3 for an example of a microfluidic RL environment). The general formulation of the problem allows application to a variety of environments, including robot control (*161*), visual navigation (*162*), network routing (*163*) and playing computer games (*75*).

As an example, consider a self-driving car that is controlled using reinforcement learning. The test environment for the self-driving car is a race track, where the car has to complete laps without deviating from the track. During each "step" the car will observe the environment using sensors such as cameras and radar. Based on these observations the RL algorithm in the car decides which action to take, such as "steer straight", "move to the left" or "move to the right". The chosen action is executed, which results in the car progressing along the test track in some way. Based on the driving performance of the car in the environment (for example, whether it has successfully stayed on the street), a reward for the calculated action is determined. In the example, the reward could represent whether the car has crashed (negative reward) or whether it has completed a full lap (positive reward). Over time, the RL algorithm learns to choose actions that increase the future reward, such that after training, the car stays on the test track more often, and thus receives higher rewards. Accordingly, at the core of an RL algorithm is a function which determines the optimal next action based on a set of observations. Improving this function directly correlates to performance improvements within the problem domain.

It should also be noted that the reward calculation within RL is entirely defined by the experimenter. This enables alterations in the objective of the algorithm because novel tasks can often be represented as a novel rule set with associated rewards. In our example, if we would like to re-train an off-road, self-driving car we could simply give a negative reward for staying on the road. The primary disadvantage of such flexibility relates to the required algorithmic fine-tuning, since there is no fixed set of rules

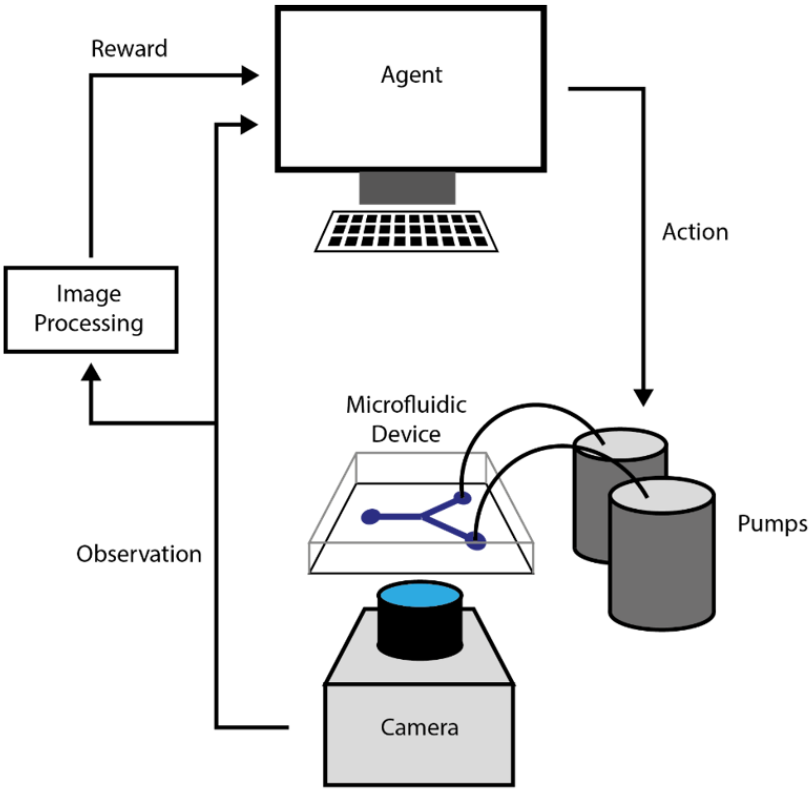

FIGURE 4.3: Framing of a microfluidic reinforcement learning experiment. Reinforcement learning is a variant of machine learning, which allows a controller to interact with an environment and continuously improve its "performance". Initially, an observation of the environment is made. In the studies described herein, we observe a microfluidic environment using a camera connected to a microscope. A reward is then calculated from the observation using classical image processing. Normally, a higher reward tells the agent that the previous action has been a good choice given the current environment. Based on the camera observation, the agent then selects an optimal action to perform next. In our microfluidic environment, the actions are alterations of the flow rates of constituent flow streams. Crucially, the agent improves its performance by choosing better actions for a certain observation, which results in an overall higher reward signal.

for reward design. This is especially problematic because empirically small changes in the reward calculation can result in big performance changes.

Another key challenge for RL is the fact that goal situations in challenging environments are often non-trivial to achieve as they require complex sequences of actions, with a positive reward signal only being obtained at the end of such a sequence of actions (reward sparsity). For example, in a chess game the ultimate reward is obtained by winning the game, but many preceding actions are required to win a particular game. Thus, a successful RL strategy often judges the quality of a specific action based on a long-term future reward and not simply immediate benefits.

Q-learning is a RL technique, which maintains an estimator function (Q-function) for the future reward in an environment based on the current state of the environment (*164*). This Q-function allows the agent to predict the expected reward if a given action is performed in the current state. The agent typically chooses the action with the highest expected future reward. The Q-function is continuously updated and improved after each action, when the real reward is received. As the quality of the estimator function increases, so does the performance of the agent in the environment. Additionally, the choice of Q-function is independent from the algorithm, which gives this method great flexibility (*75*, *165*).

To date, only a small number of applications of RL in non-simulated environments have been shown, since these environments introduce additional challenges, such as obtaining systematic input data and exerting tight control over the environment. However, previous examples of RL in non-simulated environments include robotic arms grasping objects (*78*), where a RL algorithm observes the spatial relationship between the gripper and objects in the scene, and learns hand-eye coordination, resulting in successful grasps. Another study has shown advanced building climate control (*79*), achieving reduced energy costs for room temperature control through the use of RL control algorithms. While simulated environments exhibit a clear correlation between executed actions and observed results, non-simulated environments are often less predictable. Therefore, a delay has to be applied after an action is executed, artificially limiting the interaction frequency of the algorithm, to increase the correlation between cause and effect. Furthermore, resetting the environment to a precise state is often non-trivial in non-simulated environments, and prohibits the use of several recent advances in RL algorithms, such as multiple parallel environments (*166*).

4.1.4 *Deep Q-learning*

One driver for the recent surge in RL-based activities has been the successful combination of RL-based learning with deep artificial neural networks. One such implementation, Deep Q-network (DQN, see Figure 4.4) (*75*), uses Q-learning in combination with a CNN to estimate future expected reward (Q-function) directly from image data. The current visual observation of the environment is fed into the neural network, which transforms the data and outputs an expected reward for every possible action. Subsequently, the action with the highest expected reward is performed and the real reward is obtained. Finally, the real reward is used to update the weights within the neural network, thus improving the prediction in future rounds. DQN and its variants (*76*, *166*) have recently and notably achieved human-like performance in several game environments, including ATARI computer games (see Figure 1.3) (*75*) or the Chinese strategy game, Go (*77*).

However, a major drawback of ANN-based RL methods are the extended training times (due to slow updates to the weights of the underlying neural network) required to prevent model over-fitting. This means, that DQN methods need to revisit a successful situation many times during training to fully incorporate the knowledge in the underlying model. Additionally, a prolonged exploration phase is required, during which the algorithm mixes random actions with predicted actions to properly explore the entire environment and prevent being caught in local optima. One strategy to accelerate the learning process involves the simultaneous use of multiple asynchronous agents operating separate environments (*166*). However, this process is challenging in non-simulated environments, as multiple independent experimental setups must be operated in parallel.

4.1.5 *Model-free Episodic Controller*

Recently, a more data-efficient RL algorithm has been proposed, namely the Model-Free Episodic Controller (MFEC) (*165*). Analogous in many ways to hippocampal learning (*167*), the algorithm stores a table of observations with their associated reward values. The optimal action for a novel observation is then deduced by estimating the reward from previous but closely-related observations. To find such closely related observations, nearest neighbor (*168*) or similar techniques are most commonly used. As its name suggests, the MFEC does not attempt to build a model of the environment, unlike the ANN in DQN. Instead, the MFEC simply chooses

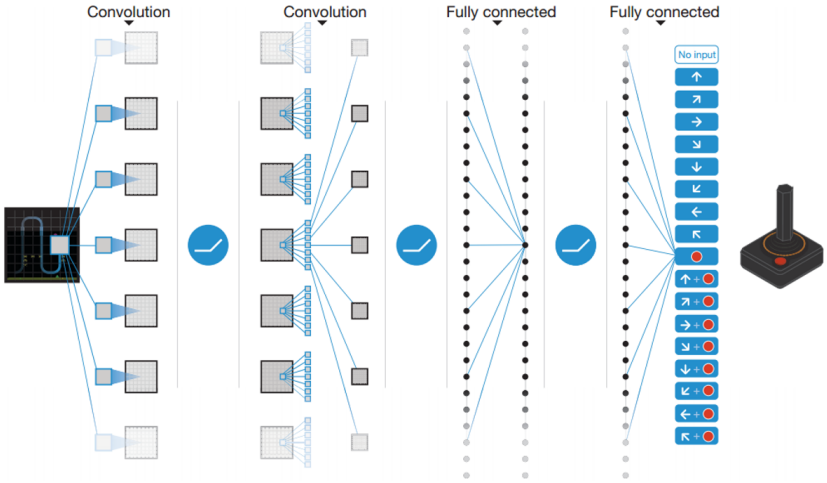

FIGURE 4.4: ANN used for RL in Atari 2800 games [see (75)]. The schematic highlights the neural network architecture used for DQN. This neural network represents the core part of the agent (see Figure 4.3) and attempts to choose an optimal action based on the current observation. Initially, a visual observation (84 x 84 pixels, because it is small enough for efficient neural network computation) is fed into three consecutive convolutional layers, which interpret the information contained in the input frame. Next, a fully connected layer is used to consolidate the extracted information. Finally, an output layer containing a single neuron for each possible action is connected to the last hidden layer. Thus, the ANN judges the future reward for each action given an input frame and typically the single action showing the maximum reward is picked and executed in the next step.

the action which has historically seen the largest reward. Since picking an action is essentially a table-lookup, a single and high-reward situation can readily be exploited, leading to greatly reduced training times when compared to DQN. MFEC can thus repeat high-reward sequences of actions, even if a sequence has only been visited a single time. However, due to the simple choice of the previous maximum reward, MFEC methods are likely to show decreased performance in highly stochastic environments. Furthermore, as the experiment proceeds the model-free controller requires a large amount of memory to store all previous observations. In general, training times for the MFEC are reduced but at the cost of suboptimal peak performance when compared to DQN.

4.1.6 *Reinforcement Learning in Simulated Environments*

We used VizDoom, a port of the popular 1994 video game DOOM, to test algorithm performance during initial development (*169*). DOOM is a 3D first person shooter, where the player must navigate a maze-like world and defeat enemies using a variety of weapons. The VizDoom suite is itself based on ZDoom (*170*), an open source DOOM clone adapted for easy use in visual reinforcement learning experiments. Playing the original game would prove extremely challenging for an algorithm due to sparse rewards. Accordingly, VizDoom offers the possibility of creating custom scenarios specifically adapted to reinforcement learning. Each scenario aims to provide a consistent 3D environment combined with a set of challenges to be learned by a particular reinforcement learning-based algorithm. Crucially, VizDoom scenarios include pre-defined rewards based on the algorithm's performance in the scenario. Figure 4.5 presents an example view of the training scenario used during algorithm development (see Section "Materials and Methods: Algorithm Tests using VizDoom") for a more extensive description).

Testing the algorithm in a simulated environment has several key advantages over immediate use in a non-simulated environment. A simulated environment can be setup and reset using simple commands, allowing for an infinite number of unsupervised test runs. In contrast, a fully automatic setup procedure is simply impossible for the microfluidic environments to be investigated. Additionally, a simulated environment will run at the maximal speed of the available computational hardware, unhindered by interactions with the physical world. Using the VizDoom environment, we were able to test algorithm performance at up to 500 frames per second,

FIGURE 4.5: VizDoom simulated test environment. This image shows the simulated test environment used during algorithm development. The agent spawns in the center of a room and can perform one of only three possible actions: move left, move right or shoot. A stationary enemy is spawned at a random position along the back wall of the room. The challenge in this simple scenario involves aligning the player with the enemy and then shooting to eliminate the enemy. A positive reward is given for hitting the enemy, which also resets the episode. Negative rewards are given for wasting ammunition and time. We chose this scenario to evaluate initial algorithm performance, since it allowed algorithm testing at rates exceeding 500 frames per second.

which represents as rate two orders of magnitude larger than accessible within the microfluidic environment. This decreased iteration time proved invaluable, especially during initial software development. Rapid testing procedures further allowed for a relatively quick assessment of the long-term behaviour of the learning algorithms.

As with most artificial neural network systems, the algorithms used in the current studies depend on a large parameter set, including network topology, learning speed and interaction with the environment. The combination of automatic setup and fast runtimes allowed the testing of many hyper-parameter sets and ensured optimal algorithmic performance before any non-simulated experiments were run. VizDoom also constitutes a deterministic environment, where identical actions lead to an identical game state. We used this property to test whether algorithm implementation is deterministic. Finally, it is important to note that running the same algorithm in simulated and physical environments serves to proof its general applicability.

4.1.7 *Microfluidic Reinforcement Learning*

The application of RL to high-throughput experimentation in microfluidic environments can help mitigate several traditional drawbacks. First, inherent variations in the fabrication process often lead to significant performance differences between "identical" microfluidic devices. The use of RL can in principle ensure consistent operation (between different devices) despite fabrication defects, reducing the need for manual intervention and ensuring experiment consistency. Furthermore, temporal variations in the microfluidic environment (such as surface fouling or substrate swelling) often prohibit consistent long-term microfluidic experimentation. This is particularly problematic when using polydimethysiloxane (PDMS) as the substrate material, due to the adsorption of hydrophobic molecules onto the substrate (*119*). In such situations, application of RL can be used to maintain stable flow conditions over extended time periods.

Even though many detection methods have been adapted to and integrated with microfluidic environments (*1*), optical bright field microscopy remains one of the simplest and most commonly available detection methods. That said, the rapid interpretation of raw visual data and subsequent interaction with a microfluidic environment is traditionally only feasible using purpose-built, and thus inflexible algorithms. Accordingly, RL has

great potential for inclusion into complex microfluidic assays, since it allows for automated but general decision-making based on visual data.

In the current study we decided to use reinforcement learning to navigate fluidic control problems. The two challenges involved controlling the size of a water-in-oil droplet formed at a flow focusing geometry and positioning a laminar interface between two miscible flows within a microchannel. In both cases the algorithm is used to control the volumetric flow rates of piston-based pumps that deliver fluids into a microfluidic device. Significantly, all decisions are solely based on visual observations of the microfluidic device using a standard microscope. In both experiments, the control algorithm maximizes a scalar reward, calculated independently for each frame using classical image processing. Even though the reward signal is calculated from the same frame used by the controller, this is not strictly necessary, and the reward could be calculated independently (e.g. from observations obtained at a different position along the microfluidic channel).

Limitations on training speed are less problematic if an environment can be queried at high speed. This is typically the case for simulated environments, but microfluidic environments cannot be controlled at arbitrary rates, due to a range of physical limitations, including the delay between altering a flow rate and the flow rate within the microchannel actually changing and available camera frame rates. Given that the obtained rewards are arbitrarily scaled, we chose to compare the control algorithms to a human tester and a random agent performing the same task.

To the best of our knowledge this study represents the first example of reinforcement learning in a microfluidic environment. We believe the ability for intelligent control in dynamic environments, such as those within microfluidic devices, will enable more reproducible and long-term microfluidic experimentation. Since the examined situations represent two basic, but fundamentally different fluidic challenges, we note that this study serves as proof-of-concept for the wider applicability of machine learning to microfluidic research and high-throughput screening in the chemical and biological sciences.

4.2 MATERIALS AND METHODS

4.2.1 *Investigate Fluidic Environments*

We are considering two fundamentally different fluidic challenges for the RL algorithms to solve. In a single-phase laminar flow environment the control algorithms are adjusting the flow rates of two differently colored aqueous solutions and thereby influence the position of the laminar flow interface between the two phases in a microfluidic channel, attempting to position this interface at 30 % of the channel width.

In a two-phase dispersed flow environment the algorithms control the flow rates of two phases, a continuous phase of fluorinated oil (HFE7500; 3M, Rüschlikon, Switzerland) and an aqueous dispersed phase. We use flow-focusing (*13*) to produce regularly sized microfluidic droplets on chip from these two immiscible phases. Altering the flow rates results in the production of differently-sized droplets and the control algorithms aim to produce droplets of 54 µm diameter.

Both environments limit the attainable flow rates between 0.5 µL/min and 10 µL/min in 0.5 µL/min steps. Interaction frequency is limited to 1.5 hertz and the environments are reset to random flow rates after a fixed number of interaction (250 interactions), thereby splitting the challenge into separate episodes. Due to extensive training times we have ended separate experiments at different points, after a performance plateau was reached.

4.2.2 *Experimental Setup*

Microfluidic devices were fabricated using conventional soft lithographic methods in polydimethylsiloxane (*119*). A detailed description of the fabrication process is provided in Chapter 2, and it should be noted that for the current experiments all microfluidic channels were 50 microns deep. Deionized water and deionized water containing 1 % *v/v* ink were used as the two phases for the laminar flow experiments. For droplet-based experiments the same ink solution was used as the dispersed phase and HFE7500 (3M, Rüschlikon, Switzerland) containing 0.1 % *wt/wt* EA-surfactant (PicoSurf 1; Sphere Fluidics, Cambridge, UK) was used as the continuous phase. Two piston-based pumps (milliGAT; GlobalFIA, Fox Island, USA) were used to deliver fluids and control volumetric flow rates, and high-speed fluorescence camera (pco.edge 5.5; PCO AG, Kelheim, Germany) was used

to observe fluids through an inverted microscope (Ti-E; Nikon GmbH, Egg, Switzerland) equipped with a 4X objective (Nikon GmbH, Egg, Switzerland).

4.2.3 *Data Pre-Processing*

Observations from the high-speed camera are minimally pre-processed before being fed as an input into the controller. Initially, the raw camera frame is converted to a floating-point representation, where a pixel value of 0.0 corresponds to a black and 1.0 corresponds to a white pixel respectively. Finally, the frame is resized to a size of 84 x 84 pixels, according to the original publication (*75*).

4.2.4 *Reward Calculation*

The reward estimator for the laminar flow environment evaluates the position of the laminar flow interface across the microfluidic channel by performing a thresholding operation on the raw frame. The dye-containing solution yields black pixels, whereas the clear solution produces white pixels. The interface position is then estimated using the average intensity of pixels across the complete image. Finally, the reward is calculated as an error between the current position and the desired position. It should be noted that the desired position was chosen to be one third of the channel width to prevent the "simple" solution of using the maximum flow rate on both pumps. The reward in the droplet-based experiments was calculated by detecting the radii of droplets in the observed frame. Initially, both Gaussian blur (5x5 kernel) and Otsu thresholding (*171*) operations were applied to achieve proper separation of the black, dye-containing droplets from the background. A dilation operation (with a 3x3 kernel) was then used to additionally discriminate the droplets from the channel walls. Subsequently, circles were detected in each processed frame using a Hough circle transform (*172*) and the radii of all detected droplets extracted. The final reward is calculated from the mean error between the droplet radii and a desired radius of 27 pixels (corresponding to 54 µm). All reward calculations were performed using classical image processing employing the OpenCV Python module (*173*). It should also be noted that both reward calculation methods produce noisy estimates, providing an additional challenge to the RL agent.

4.2.5 *Environment Characterization*

Due to the limited complexity of our model environments, we chose to perform a full characterization of the reward space. Using an automated scheme, observations for every possible flow rate combination were obtained and post-processed offline, using the respective reward estimators. However, it should be noted that the obtained reward surface is specific to a single microfluidic device, since variations in the manufacturing and treatment process of identical devices result in an altered reward surface.

4.2.6 *DQN Algorithm*

Our DQN architecture is similar to the dueling network architecture reported by Wang and co-workers (*76*). Specifically, we used raw camera frames as inputs to the neural network-based Q-function. However, due to reduced update rates, we used an initial random phase of 10'000 frames and an annealing phase of 135,000 frames (number of frames to change from 100 % random actions to 0.05 % random actions). Furthermore, we updated the target network parameters every 5000 frames, storing and learning from only the most recent 50,000 frames. We used a custom DQN version, implemented in Python 2.7 using Keras (*174*) and the Theano (*175*) backend running on Windows 7 (Microsoft Corporation, Redmond, USA). For training and inference of the involved ANN we used a GPU (Quadro K2000; Nvidia, Santa Clara, USA). Finally, we used custom Python scripts to post-process and visualize results.

4.2.7 *Model-free Episodic Control Algorithm*

We used a custom version of MFEC, implemented using Python 2.7 according to the architecture outline described by Blundell and co-workers (*165*). We use an approximate nearest neighbor search to determine related observations [Locality Sensitive Hashing forest (*176*), LSHForest, implementation provided by the sklearn Python module (*177*)] with 10 estimators. This method was chosen since it allows for a partial fit (addition) of new data, without the need to recalculate the entire tree for each new observation. Such a complete re-balancing of the tree is only performed in 10 % (randomly sampled) of data additions. Observations are pre-processed using the same pre-processing pipeline as DQN. Subsequently, input frames are encoded using a random projection into a vector with 64 components.

We chose random encoding because it shows similar performance, when compared to a more complex encoding scheme using a variational autoencoder (*165*). The MFEC algorithm requires the environment interaction to be split up into episodes (regular intervals at which the complete environment is reset, and performance evaluated).

4.2.8 *Algorithm Tests using VizDoom*

We used a basic scenario included with VizDoom (*169*) for algorithm testing (see Figure 4.5). In this scenario, the player starts in the center of a small room, with a static enemy being spawned at a random position on the far end of the room. The goal is to simply eliminate the enemy using a series of the following actions: walk left, walk right or shoot. Accordingly, the algorithm must line up the player with the enemy and then shoot. A large reward is given for hitting the enemy, at which point the scenario is restarted. A missed shot is punished with a negative reward and the algorithm additionally obtains a small negative reward after each action, to punish indirect paths to the solution.

4.2.9 *Benchmarking Learning Performance*

To benchmark controller performance in the fluidic environments, we used scores obtained by a human tester and a random agent. Random performance benchmarks were obtained by choosing a random action from the available action set every frame and recording the obtained rewards. The random agent represented a lower bound on performance and served to check initial DQN performance, as it is expected to be random. Human-level performance results were obtained by having two separate, trained human agents solve an identical task (observation at the identical position, with identical resolution and an identical action set) for approximately 30 minutes while recording the rewards. Prior to benchmarking, each human tester was given an explanation of the underlying physics and allowed to practice the task for at least 30 minutes. All benchmarks shown represent the mean reward obtained as well as a 95 % confidence interval.

4.3 RESULTS AND DISCUSSION

4.3.1 *Laminar Flow Control*

Low Reynolds numbers (Re) are typical for fluids flowing through microfluidic channels, with fluid flow being dominated by viscous forces rather than inertial (turbulent) forces (*1*). Such low Reynolds numbers are characterized by the existence of laminar flows, where fluid flows in parallel layers, with no disruption between the layers. The ability to control and align the interface between two co-flowing streams within a microfluidic channel is critical in many applications (see Figure 4.6 a). In the current experiments, which involve the confluence of two aqueous streams under low Re, this interface is made visible by the addition of ink to one of the input solutions (see Figure 4.6 b).

We use the presented environment (see Figure 4.6 a) to investigate automatic control over a laminar flow environment. The controller repeatedly alters the flow rates of the constituent fluid phases resulting in various laminar flow interface positions. After a fixed number of interactions (250, corresponding to one episode) the environment is reset to random flow rates and the controller restarts its task.

While we use this system as a simplified proof-of-concept model, there are several applications which require the establishment and control of a stable linear interface between two co-flowing phases, such as the controlled synthesis of vesicles (*178*) or droplet trapping and transport systems (*179*).

The volumetric flow rates of each flow stream are limited to values between 0.5 µL/min and 10 µL/min (resulting in total flow rates between 1 µL/min and 20 µL/min), representing typical flow rates used in microfluidic experimentation over extended time periods. As previously indicated, flow rates are set to random values within this acceptable range at the start of every episode. The challenge then involves adjustment of the flow rates such that the fluid interface moves to an (arbitrary) optimal position (30 % of the channel width) within one episode (corresponding to 250 interactions). The scalar reward for the previous action is defined as the proximity of the laminar flow interface to the optimal position, which is extracted from the captured frame via classical image processing methods. The control algorithm adjusts the volumetric flow rates by performing one out of five discreet actions, namely, increasing or decreasing the flow rate of the continuous phase, increasing or decreasing the flow rate of the dis-

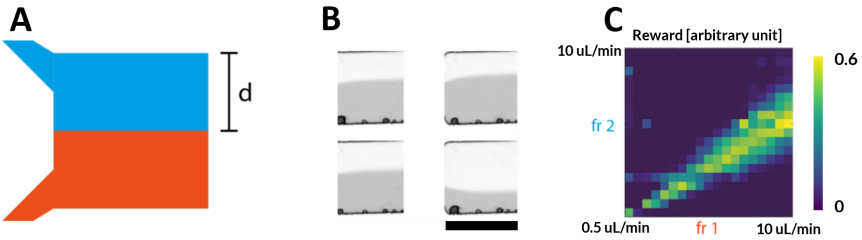

FIGURE 4.6: Laminar flow control. (A) Schematic of a standard laminar flow environment, established in a microfluidic device. Two aqueous solutions are introduced into a common microfluidic channel. One solution contains a dye to allow direct visualization of the interface between the two streams. The control algorithm is used to alter the volumetric flow rate of both input streams, thereby influencing the position of the interface in the channel cross-section (being quantified by d). (B) Example image frames captured during the training phase (scale bar - 150 µm). Depending on the volumetric flow rates of the input streams, the laminar flow interface will occupy a different position across the channel width. It should be noted that a small number of trapped bubbles can be seen along the lower channel wall. These bubbles originate due to fluidic defects (aspiration of air in the piston-based pumps) and pose an additional challenge to the control algorithm, by increasing the amount of noise in both the reward calculation and the observed frame. (C) Results of a complete environmental characterization of a single microfluidic device. Rewards are shown for various flow rates (fr1, fr2, from 0.5 µL/min to 10 µL/min). An optimal flow rate regime can be observed, which produces a desired value (d in panel A) of 30 % of the channel width. However, due to variations in the fabrication and setup process, reward surfaces will be different for different experiments, even when using "identical" microfluidic devices.

persed phase, or maintaining the flow rates unchanged. An optimal fixed step size of 0.5 µL/min was determined empirically to limit any strain on the pumps and ensure that an optimum is found within one episode. Additionally, control algorithms were limited to 1.5 hertz interaction frequency to prevent equipment damage and enhance the coupling between the performance of an action and the observation of the resulting conditions within the microfluidic systems.

4.3.1.1 *Environment Characterization*

Figure 4.6 c shows a complete characterization of a reward surface for the laminar flow challenge. Intuitively, it is expected that the position of the laminar flow interface should be correlated with the ratio between the flow rates of the two fluid phases. Indeed, the reward surface shown clearly identifies an optimal region, where the flow rates produce the desired interface position and thus achieve high rewards. However, as previously noted, the data graphically shown in Figure 4.6 c is valid for a specific microfluidic device, with replicate devices (having the same putative dimensions) exhibiting significantly different behavior due to variations intrinsic to the fabrication process.

4.3.1.2 *Laminar Flow Control using DQN*

Figure 4.7 shows algorithmic performance in the laminar flow environment. Inspection of Figure 4.7 a indicates that DQN performance during the first 5500 frames (approximately one hour in experimental time) is comparable to the performance of the random agent due to the initial exploration phase of the DQN, where the share of predicted actions is slowly increased from 100 % random actions to 95 % controller-based actions (see Figure 4.7 a, where the exploration phase ends after 27 hours). Over the course of the next 36 hours of training (which equates to approximately 195,000 image frames) the algorithm manages enhance performance to a level that is comparable to a human tester, at times even surpassing human-level performance (e.g 27 hours to 37 hours during the "blue" experiment). It is also observed that although separate experiments indicate the same general trends in performance, short term performance variations differ markedly between experiments. We hypothesize that performance should be improved further through the use of longer training phases, noting that typical benchmarks for ATARI environments involve training for up to 200 million frames (*180*). Such an approach was impractical in the current

study, since the testing of 200 million frames would correspond to over 4 years of training time at the investigated frame rates.

During the initial exploration phase, as the share of random actions is slowly reduced and DQN improves its accuracy, a gradual increase in performance is expected. Even though such a trend is apparent, some experimental runs required longer than the initial exploration phase to realize peak performance. We hypothesize that the control algorithm is captured within the vicinity of a local minimum during poorer performing experiments. That said, such effects could be mitigated through the use of multiple asynchronous experimental setups, such as A3C (*166*), which allows the controller to interact with multiple similar environments at the same time, greatly reducing the chances of being captured in such a local optimum. However, while using multiple environment is trivial when using simulated environment, it is unpractical in real-world scenarios. It is also noted, that all experiment repeats surpassed the performance of human testers eventually. Generally, performance fluctuated around human-level performance after 48 hours (approximately 260,000 frames) of training. However, we could observe that longer run times did not significantly improve performance.

In the current study, DQN is retrained from scratch for each new experiment. Accordingly, in future experiments, algorithm training from pooled experimental data (collected using multiple devices over multiple experiments) could improve the stability of the control algorithm across a wider variety of situations.

On a practical level, deposition of debris within microfluidic channels often leads to blockage, with gas bubble accumulation leading to flow significant instabilities. Accordingly, it is remarkable that the presented control algorithm is successful in maintaining performance and adjusting to changing conditions over extended periods of time. Indeed, the presence of a gas bubble appears to have only short-term effects (see inset highlighting the performance dip shown in Figure 4.7 a), with the algorithm recovering quickly after the bubble dissipates. However, it should also be noted that we did not uncover evidence of the algorithm learning how to get rid of the bubbles actively within the observed time frame. That said, such a feat would constitute a non-trivial task even for human operators! Consequently, we conclude that DQN is able to achieve human-level performance for the laminar flow challenge, albeit requiring considerable training time to achieve peak performance. We find that DQN is well-suited to the automated handling of real-world complications that arise due to the extended

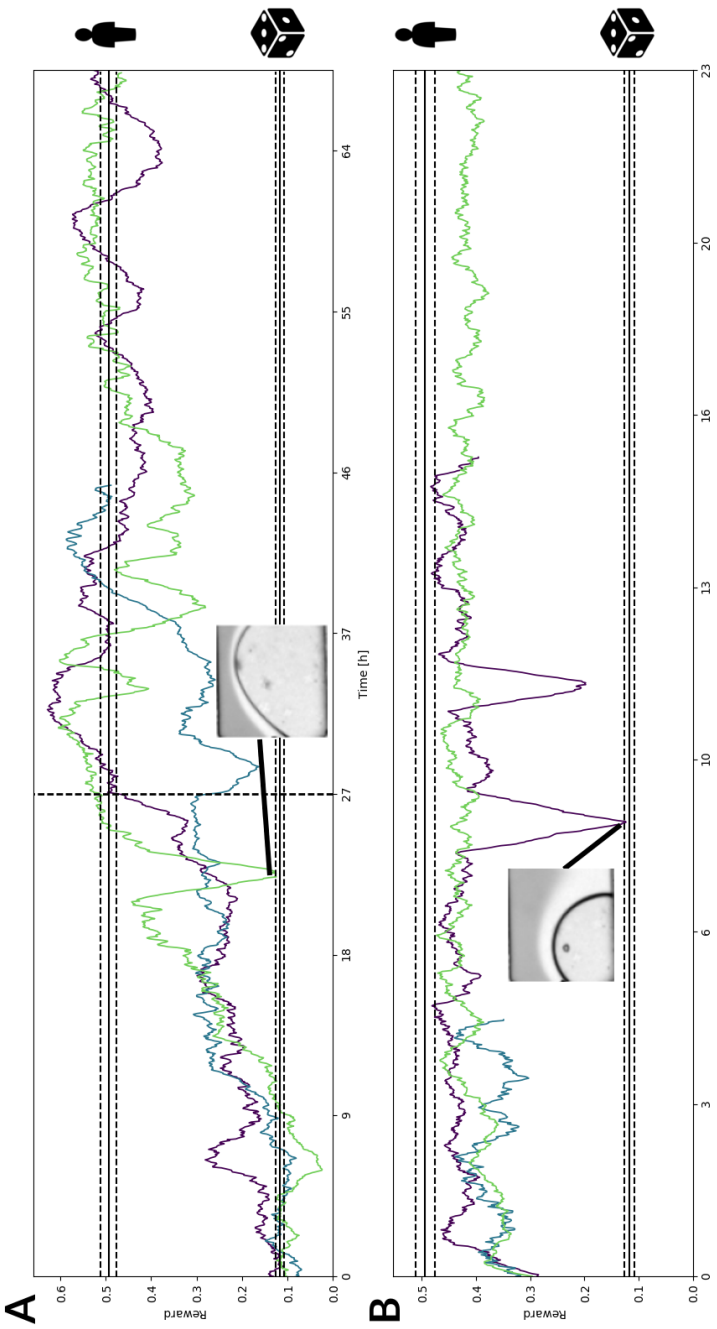
A
B
Reward
Time [h]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
9
18
27
37
46
55
64
3
6
10
13
16
20
23

FIGURE 4.7 *(previous page)*: Controller performance during laminar flow control (inset width - 150 μm). Benchmark performance (random and human-level, highlighted using icons) are indicated using mean performance and 95 % confidence intervals. (A) Results of the DQN controller in the laminar flow environment. Three experimental runs are shown, highlighting that DQN can attain human level performance for the laminar flow challenge. It can be seen that each experiment initially exhibits a random performance level, as is expected for an untrained controller. Furthermore, it is observed that controllers exhibit variable performance improvements over time, with all experiments eventually surpassing human performance. In this respect, it should be noted that some of the sharp drops in performance (for example the "green" experiment run at around 22 hours, see inset) are due to experimental complications. For example, as shown in the inset, a gas bubble trapped within the field of view of the camera disturbs both reward calculation and controller performance. (B) MFEC performance in the laminar flow challenge. The MFEC exhibits a rapid learning capability and shows peak performance within two hours of training. However, absolute performance is lower than DQN, and for this task remains below human-level performance. In a similar fashion to DQN experiments, sharp drops in performance are observed when bubbles become trapped within the microchannel (see inset), but interestingly MFEC performance recovers more rapidly than for DQN controller, which is expected because the MFEC does not require visiting a particular situation multiple times to incorporate it into the control scheme.

experimental time-frames, enabling automation of a variety of long-term experiments. Therefore, our results highlight for the first time, the capabilities of DQN for maintaining complex control situations in microfluidic devices based on visual inputs over extended time periods.

4.3.1.3 *Laminar Flow Control using MFEC*

The MFEC is able to achieve peak performance within the first 11,000 frames (approximately 2 hours of experimental time). This compares favorably to the 130,000 frames (or 24 hours of experimental time) needed by DQN (see Figure 4.7 b). Such a situation is to be expected, since every single rewarding situation can be exploited by the algorithm. However, the maximum performance achieved by the model-free controller did not consistently reach human-level performance (unlike DQN), albeit showing only marginal reduction in performance (typically 90 % of human-level performance in terms of achieved scores). In a typical experiment this might pose an acceptable trade-off, given the significant reductions in initial training time.

Similar to the disturbances observed during DQN experiments, sharp performance drops were detected when a bubble enters the microfluidic channel (see inset highlighting the performance dip shown in Figure 4.7 b). However, the model-free controller exhibits a substantially faster recovery, once the bubble dislodges and the environment reverts to the default state, when compared to the DQN controller. We hypothesize that such behaviour is due to the model-free nature of the MFEC algorithm, which does not update an internal model when encountering flawed observations caused by short-term fluctuations. Therefore, the MFEC can quickly recover performance as soon as the bubble is dislodged, and normal observations are obtained again. Furthermore, empirically the model-free controller shows less performance fluctuations than DQN, especially over long time-frames. Indeed, due to its consistent performance, the MFEC is well-suited to the control of relatively simple experimental environments, where slightly reductions in peak performance are acceptable. In practical terms, the short training time requirements heavily favor MFEC over DQN, since training a controller in a few minutes is simply not feasible using DQN.

4.3.2 *Droplet Size Challenge*

Under certain circumstances, co-flowing two immiscible fluids through a narrow orifice (a flow-focusing geometry) within a microfluidic channel results in the formation of monodisperse droplets of one of the fluids within the other (*1*). Importantly, these droplets represent separate reaction containers and can be produced at rates exceeding 10,000 droplets per second. Unsurprisingly, such segmented-flow formats have attracted enormous attention from the biological research community and are now an essential part of high-throughput experimental platforms for single-cell genomic sequencing (*66*), early stage kinetic studies (*181*) or high-throughput screening (*21*).

The goal of the droplet size challenge was to adjust the flow rates of the two droplet forming phases to produce droplets of a predetermined size (see Figure 4.8 a). In a similar manner to the laminar flow challenge, volumetric flow rates are limited to values between 0.5 µL/min and 10 µL/min, with the step size being fixed to 0.5 µL/min and the interaction frequency limited to 1.5 hertz. Furthermore, the control algorithms interact with the environment using the same set of actions used in the laminar flow challenge, i.e. increasing or decreasing the flow rates of the two droplet-forming phases, as well keeping the flow rates constant.

4.3.2.1 *Environment Characterization*

Figure 4.8 c shows an exemplar reward surface for the droplet size challenge. In a similar manner to the laminar flow reward surface (see Figure 4.6 c), results indicate optimal flow rate ratios, which frequently produce droplets of the correct size (e.g. continuous phase (fr1) = 5 µL/min and dispersed phase (fr2) = 3.2 µL/min resulting in a diameter of 54 µm). However, the boundaries of this optimal region are much less well defined than those observed in the laminar flow challenge. Furthermore, we found larger variations between reward surfaces originating from separate microfluidic devices. We believe this increased uncertainty stems from the sensitivity of the droplet formation process to surface wetting effects (*182*), as well as the circle Hough transform used in the reward calculation, which in turn results in a noisier reward signal. On the basis of the direct comparison between reward surfaces, we expect that the droplet size environment will require a more sophisticated control solution, providing additional challenges for the control algorithms applied.

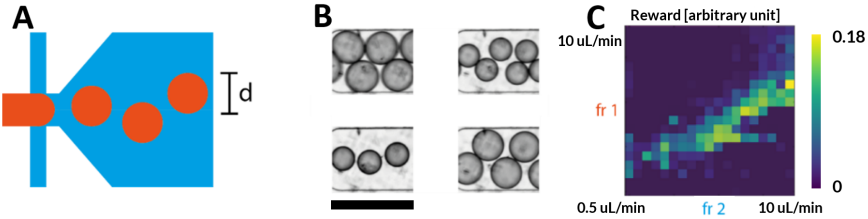

FIGURE 4.8: Droplet size challenge. (A) Schematic illustration of the droplet size challenge. Droplets are produced at a flow-focusing junction by co-flowing an aqueous solution (containing a dye) with an immiscible oil phase. The control algorithm attempts to optimize the flow rates of the constituent solutions such that droplets of the desired size (optimal droplet diameter d = 30 pixels = 54 µm) are produced. (B) Example frames captured during an experimental run (scale bar - 150 µm). Droplets of varying diameters are produced depending on the magnitude of the input flow rates. (C) An example reward surface for a complete scan of the environment for various flow rates of the dispersed phase (fr1) and the continuous phase (fr2) both within a range of 0.5 µL/min to 10 µL/min. An extended region exhibiting high rewards exists but the boundaries are less clearly defined than the reward surface shown in Figure 4.6 c.

4.3.2.2 *Droplet Size Control using DQN*

Starting from random performance, the DQN controller typically managed to surpass human-level performance prior to the end of the exploration phase (see Figure 4.9 a). As in the case of the laminar flow challenge, superhuman-level performance was achieved in all experiments, despite the fact that short-term performance variations and absolute performance variations can be observed between experiments. Again, this is unsurprising, given that separate experiments utilized different microfluidic devices, different reagent solutions and were performed at different times. Given that similar differences in maximal performance were observed when using the MFEC, it is likely that such differences originate partially from differences in the fabrication, surface treatment, and during initial setup of the microfluidic platform (connecting the microfluidic device to pumps and aligning the optical setup). However, since RL in non-simulated environments constitutes a stochastic process, performance variations stemming from the algorithm (due to capture in local optima) are also expected, especially given the limited training times involved.

In a similar manner to the laminar flow challenge, large-scale performance fluctuations over extended time periods were observed. This could be explained by increased sensitivity of droplet formation to surface wetting effects, when compared to the single-phase system examined in the laminar flow challenge. For example, Xu and co-workers have shown that altering the wetting properties by changing the surfactant concentration results in different co-flow regimes, varying between laminar flow and droplet flow (*182*) due to surface aging effects of the PDMS microfluidic device. Therefore, reaction conditions in the flow focusing geometry are expected to vary greatly, as surface conditions change over long time-frames. Further, long-term experiment drift can also be caused by small-scale fluid leakage, as previously faulty fluidic connectors loosen more over time, increasing fluid leakage further. However, despite these phenomena DQN performance was observed to remain close to or exceed human-level performance. Such results clearly indicate that DQN is a viable option for maintenance of reaction conditions during long-term microfluidic experiments, even in complex environments.

4.3.2.3 *Droplet Size Control using MFEC*

The performance of the MFEC in the current task was outstanding and on par with DQN performance (see Figure 4.9 b). Typically, the model-

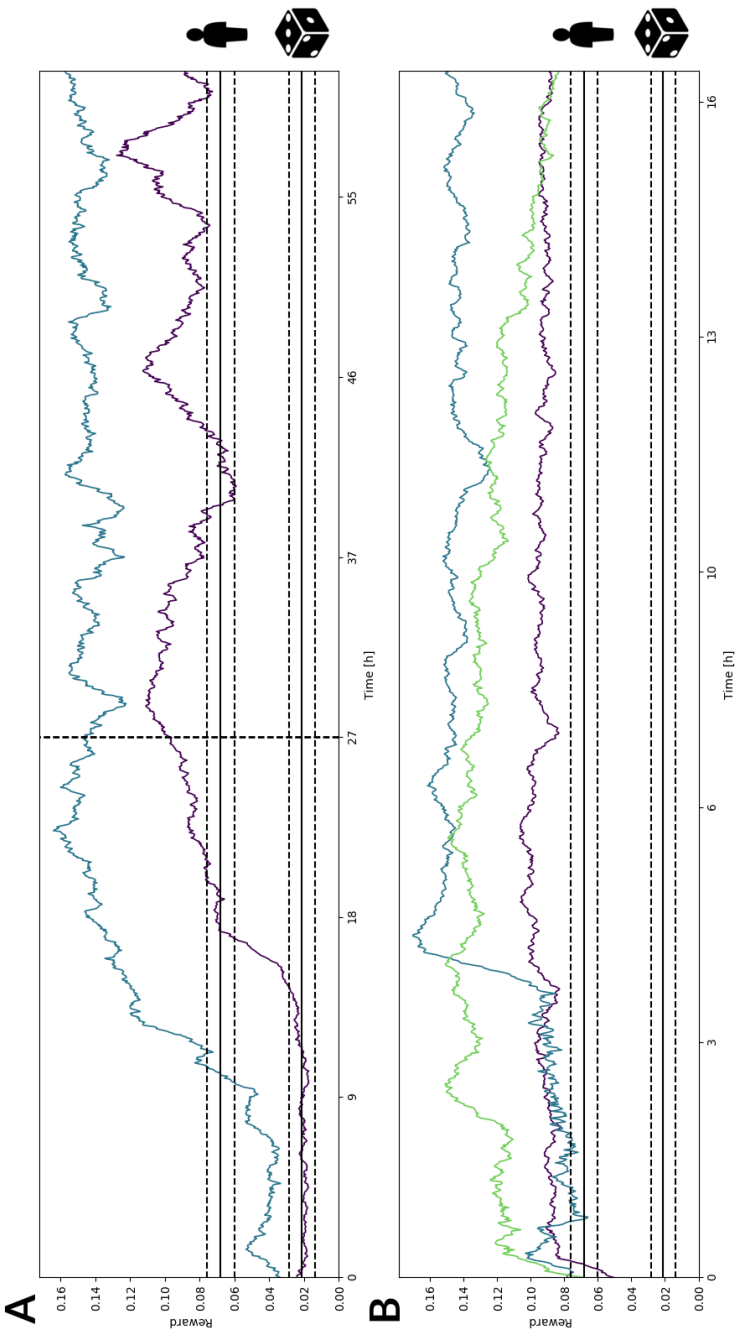
A
B
Reward
Time [h]
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
9
18
27
37
46
55
3
6
10
13
16

FIGURE 4.9 *(previous page)*: Controller performance in the droplet size challenge. Benchmark performance values (random and human-level, highlighted using icons) are indicated using mean performance and 95 % confidence intervals. (A) DQN is able to surpass human-level performance in the droplet size challenge. However, the experimental runs exhibit different absolute performance levels. While this difference, in part, can be attributed to variations in the fabrication and setup process it most likely stems from the stochastic learning process underlying RL. (B) MFEC excels in the droplet size challenge, with the controller quickly surpassing human-level performance and maintaining high performance levels throughout the whole experiment.

free controller achieves human-level performance very soon after the start of the experiment and rapidly surpasses it for most of the time. Interestingly, after quickly surpassing human-level performance, one experiment ("green" data in Figure 4.9 b) showed a slow but steady decline towards human-level performance over the next 20 hours. Since we did not observe a similar decline in any of the following experimental repeats, we believe that this effect was specific to the device and that the gradual decline was caused by an aged surface treatment solution, only resulting in temporary adherence of surface coating silanes to the channel surface (*183*).

In general, the MFEC is very well-suited to the droplet size challenge. The absolute performance of the MFEC is comparable to DQN and almost always superior to human-level performance. Even though significant attention was focused on ensuring a level playing field for the human testers (see Experimental Methods: Benchmarking Learning Performance), we believe that the super-human performance observed in this task is partly due to the rapid decision making available to the algorithm. Indeed, the human and control algorithm are adjusting droplet formation happening at approximately 1000 hertz. Even though human testers are not limited by a fixed interaction frequency, we could observe that interactions at rates in excess of 1.5 hertz (interaction frequency of the algorithm) were challenging to our testers.

4.4 CONCLUSION

Given the rapid decision making inherent to modern computational systems, there are numerous microfluidic tasks that can be performed in a previously unachievable manner using machine learning methods. This is especially true for operations which are currently performed using fixed or manually tuned parameters.

Herein, we have demonstrated for the first time that state-of-the-art machine learning algorithms can surpass human-level performance in microfluidic-based experiments, solely based on visual observations. Moreover, we have confirmed such a conclusion through the use of two different reinforcement learning algorithms, based on neural networks (DQN) and episodic memory (MFEC) respectively. We hypothesize that a combination of these algorithms could provide a solution that leverages the advantages of each method. For example, MFEC could provide initial guesses, via rapid policy improvement, that could then be used to improve DQN training. This would almost certainly decrease the overall time required for DQN to reach peak performance (which as shown herein is super-human in all studied environments).

Bright-field microscopy is one of the most commonly used experimental techniques in chemical and biological analysis due to its simplicity and high information content. Since visual observations are exclusively used in the current experiments, the proposed control algorithms are easily integrated into existing experimental setups. Moreover, we found that the computational requirements for learning were much lower than anticipated, presumably due to the fact that the rate limiting step was typically the interaction with the physical environment and not controller evaluation. This further highlights the applicability of reinforcement learning to various microfluidic environments.

At a more general level, this study purposely used proof-of-concept level challenges. Therefore, simpler control algorithms, such as PID controllers, could be applied to such environments. That said, it should be noted that due to their general-purpose nature the investigated algorithms are likely to perform well in a large variety of visual tasks. Indeed, a novel environment is simply established by defining a reward function and then re-training the same algorithm. Accordingly, further research will extend the presented findings by investigating more complex environments using the same algorithms. Finally, we believe that this study highlights the benefits of combining experimental platforms with “smart” decision mak-

ing algorithms. To date, there have been few applications of reinforcement learning in non-simulated environments. Nevertheless, we expect that a large variety of microfluidic-based experiments could be used to generate state-of-the-art results through the use of advanced interpretation or control algorithms. Examples of such experiments include the manipulation of organisms on chip, cell sorting or reaction monitoring. To conclude, and based on the results presented herein, we think that reinforcement learning and machine learning in general has the potential to disrupt and innovate not only microfluidic research but many related experimental challenges in the biological and life sciences.

5

DEEP LEARNING ENABLED REAL TIME CELL SORTING

5.1 INTRODUCTION

5.1.1 *Cytometry*

Cytometry, the measurement of the characteristics of cells, is a key technique in cell biology (*184*) and medical diagnostics (*185*). Cytometric methods allow for the quantification of a large variety of cell properties including cell number, morphology (cell size, shape and internal structure) and aggregation. When paired with molecular staining methods, cell cytometry additionally allows for the detection and analysis of various intracellular biomolecules, such as proteins (intracellular and cell membrane bound) or nucleic acids (*184*, *185*).

Image cytometry, where static cells are observed under a microscope, is the oldest form of cytometry, initially developed in the late 19th century by Louis Charles Malassez amongst others (*186*). Here, cells under observation are stained to improve contrast and subsequently imaged through a microscope. Modern imaging cytometry systems are often partially automated, nevertheless their static nature inherently limits the throughput of such systems (*187*).

In contrast to the imaging of static cells in conventional cytometry, flow cytometry analyzes cells that are rapidly moving through a detection volume (*188*). Typically, the cellular population under investigation is hydrodynamically focused using an additional sheath flow, which lines up the cells in single file before they pass through the detector. In the vast majority of embodiments, flow cytometers do not image the transiting cells, but instead measure various optical properties such as forward-scattered light (which is proportional to cell-surface area or size), side-scattered light (which is proportional to cell granularity) and fluorescence emission (*188*). The ability to move cells through the detection volume at high linear ve-

This work has been developed in collaboration with Gregor Holzner, who provided hardware and the C++ control software.

locities means that modern flow cytometers can assay cells at throughputs between 10^4 and 10^6 cells per second (*188*). Accordingly, the multiparametric data obtained using flow cytometry has significant utility in medical diagnosis and the characterization of large and heterogeneous cell populations (*184*, *185*, *187*).

Fluorescence-activated cell sorting (FACS) is a direct extension of flow cytometry and enables the sorting and isolation of specific cells based on the observed characteristics (*189*). FACS allows cellular populations containing multiple cell types to be partitioned into separate sub-populations based on their optical properties. This is achieved by splitting the cell suspension into charged droplets, each containing no more than one cell, using a vibrating nozzle. These charged droplets are subsequently passed through an electric field where they can be selectively directed towards one of several containers via electrostatic deflection. Selective enrichment through purification has proven to be extremely powerful in enabling novel methods in protein engineering (*134*) and single cell analysis (*190*).

Despite the fact that point-based flow cytometers and FACS systems allow the high-throughput and high-sensitivity analysis of cellular populations, the information content, on which the sorting decision is based is inherently limited due to the use of one-dimensional point detectors. To this end, much effort has recently focused on the development and application of imaging flow cytometry (*191*). Imaging (or image-based) flow cytometry combines the high-throughput characteristics of standard flow cytometry with the imaging capabilities of optical microscopy to allow for high-resolution imaging of single cells within flowing environments (*191*, *192*). Over the last decade, a number of novel benchtop systems have been developed, allowing for multispectral imaging of cells in flow (*193–195*). Such high-information content data has been used to localize fluorophore labeling within intracellular environments, successfully identify sub-cellular structures (*196*) and also probe the stages of apoptosis with high-temporal resolution (*192*).

5.1.2 *Microfluidic image-based flow cytometry*

Traditionally, visual data obtained using a microscope has been analyzed manually. This is a time-consuming process often yielding highly biased results due to inconsistent analytical practices and human interpretation. Importantly, software tools such as ImageJ (*136*) and CellProfiler (*197*) have been developed to automate aspects of the data processing pipeline. By

enabling the experimenter to perform identical processing steps on all image data, automated tools greatly enhance the reproducibility of the data processing pipeline. However, the analysis of image data is more time-consuming than one-dimensional data, even with the aid of modern tools. Thus, the implementation of real-time data processing platforms required for cell sorting has proven particularly challenging, often necessitating advanced computational techniques such as multi-threading (*198*), or specialized computational hardware such as field-programmable gate arrays (FPGA) or graphics processing units (GPU) (*187*). Nevertheless, recent studies have demonstrated the feasibility of image-based sorting of particles or cells based on fluorescence (*199*) or cell deformability (*200*). That said, it is fair to say that novel image-based sorting algorithms typically involve hand-crafting sorting criteria and non-trivial programming of a separate data processing pipeline (*200*). Optimally, a classification algorithm is easy to develop and adapt to novel problems. It should further minimize hand-engineering of classification rules.

Recent advances in machine learning have yielded state-of-the-art performance levels in a range of vision-based tasks, including image classification (*69*). Many of these results have been achieved by using algorithms based on artificial neural networks (ANN). These networks imitate biological computation as performed in the animal brain (*201*). Specifically, data are processed via artificial neurons organized into layers. Numerous weighted connections exist between neurons of different layers and information is stored in the weights of these connections (resulting in amplification or dampening of transferred signals). Typically, an ANN consists of an input layer, several hidden layers and an output layer. In the classical supervised case, an ANN is trained using a dataset of known input-output pairs. Therefore, while requiring large datasets for training, a neural network-based classifier can be automatically trained to perform novel classification experiments, rendering ANNs a highly promising candidate for flexible image-based cell sorting systems. The interested reader is directed to Chapter 4 for a more detailed introduction to ANNs and machine learning in general.

In the current study we employ a neural network-based image classification pipeline at the core of a flexible real-time cell sorting and flow cytometry architecture. We additionally use an elasto-inertial focusing strategy recently developed by Holzner and co-workers to align micron-sized species (cells or beads) at the center of the channel prior to detection (*202*). Training data are obtained by recording images from pure cell populations

in separate experiments prior to classification. Subsequently, the neural network-based image classification algorithm is trained offline using the collected dataset. Finally, a mixed analyte population is sorted and purified in real-time using the trained neural network. To our knowledge, these studies represent the first example of using neural network-based image classification in combination with microfluidic sorting technology to achieve real-time cell sorting.

5.2 MATERIALS AND METHODS

5.2.1 *Experimental Setup*

PDMS microfluidic devices were manufactured using standard soft-lithographic technique that are described in more detail in Chapter 2 (*119*). Specifically, the microfluidic device consists of a elasto-inertial focusing channel (square cross section 50 µm x 50 µm) and 4.15 mm length. The sorting junction (see Figure 5.2 b) consists of a 4-way microfluidic junction, located after the focusing channel and integrated push-down valves (*203*), located in a separate PDMS layer above the fluid layer.

Figure 5.1 provides a schematic of the entire optical setup used to perform the real-time analysis of cells in flow. The system allows for both bright-field and dark-field illumination of the sample, albeit we only used bright-field illumination during the current experiments. The optical system is self-contained, providing integrated illumination, magnification and sample detection. Observations during bright-field observation were magnified using a 20X objective (Nikon GmbH, Egg, Switzerland). After passing through a 50/50 mirror (only required for dark-field observation), the beam was cleaned using an iris (CP205; Thor Labs, Newton, USA) and detected using a high-speed camera (UI 3060 CP; IDS, Obersulm, Germany).

Sample is pumped into the microfluidic device using a piston-based pump (milliGAT, Globalfia, Fox Island, USA). Preliminary experiments involved the use of 12 µm (standard deviation < 0.2 µm) diameter polystyrene beads (Sigma-Aldrich, Buchs, Switzerland). Subsequent experiments, involved the use of Jurkat (Sigma-Aldrich, Buchs, Switzerland) and H562 (Sigma-Aldrich, Buchs, Switzerland) cell lines. Cell and bead suspensions used in cytometric experiments are prepared to an average concentration of 3 million beads and 3 million cells per milliliter in Dulbecco's Phosphate-Buffered Saline (DPBS; Sigma-Aldrich, Buchs, Switzerland) respectively.

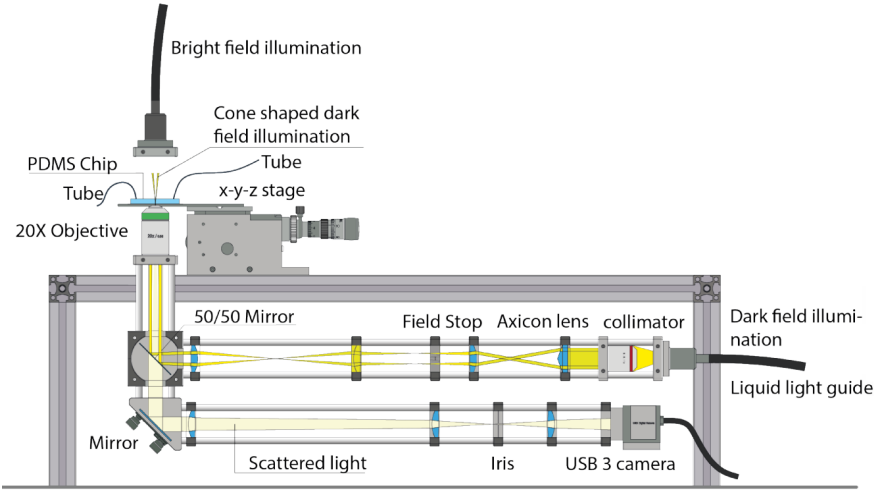

FIGURE 5.1: Optical detection system for imaging flow cytometry. For bright-field illumination, a light source is mounted above the microfluidic device. Light is captured from the bottom using a 20X objective, and further cleaned using an iris before detection using a high-speed camera.

Figure 5.2 illustrates the microfluidic geometry used for screening and sorting experiments. In brief, analytes are focused to the channel center during passage along the focusing channel (*202*). Visco-elastic focusing occurs as a result of the small height of the focusing channel and the high flow velocity during passage. Subsequently, cells or particles are observed within a narrow detection region (see the yellow region of interest (ROI) in Figure 5.2 b). Based on this measurement, the analyte is classified, and a set of solenoid valves (197020; Festo AG, Lupfig, Switzerland) is activated. This allows analyte sorting through the pressurization of control channels an activation of on-chip push-down valves (*203*).

Custom-written software, developed in C++ (*204*), was used to perform image classification and control all operational devices, including camera, pumps, light source and solenoid valves. Figure 5.3 shows the user interface of this software suite. Custom Python software was used to train the classifier offline. Finally, custom-written Python scripts were used to post-process and visualize the obtained results.

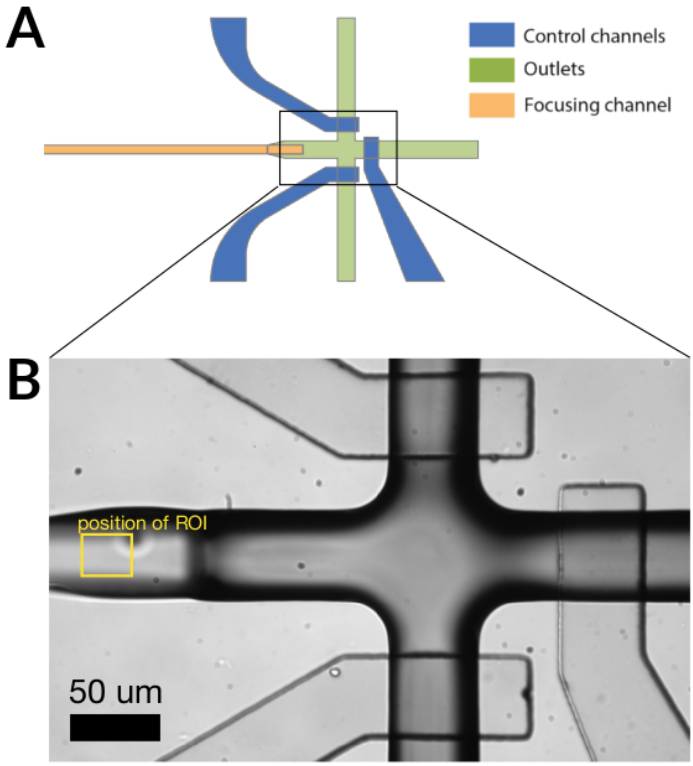

FIGURE 5.2: Push-down valve-based particle sorting architecture. Particles are focused using a visco-elastic focusing strategy before detection and sorting. (A) Schematic of the employed sorting architecture. Particles enter from the left and exit the sorting junction through one of the outlets. Push-down valves (*203*) filled with water allow the selective closure of outlet channels through the application of pressure to the respective control channel. (B) Microscope image of the sorting junction, highlighting the position of the region of interest (ROI) used for analyte classification, prior to sorting.

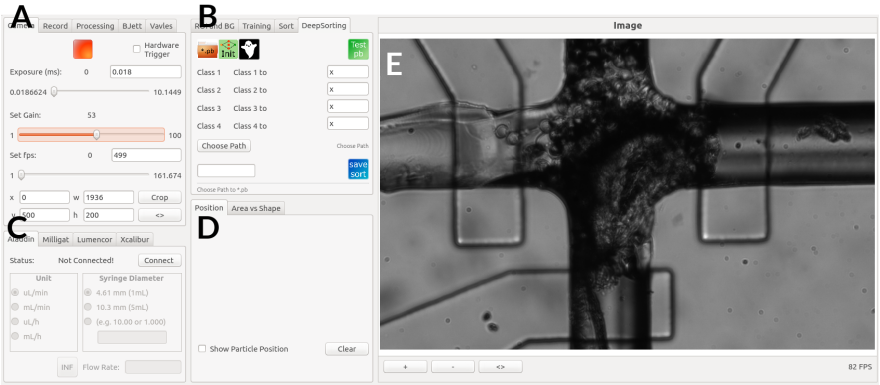

FIGURE 5.3: Control software interface. A custom software was developed in C++, allowing interactive control of the camera, region of interest (ROI) and image pre-processing (A), neural-network used for classification (B), peripheral devices, such as pumps and illumination sources (C), real-time results, and real-time observations (E). Such an integrated software solution greatly facilitates experimental work by combining all necessary control within a single interface.

5.2.2 *Classification Algorithm*

Raw observations, captured by the high-speed camera (see Figure 5.1) are retrieved from the camera buffer using the presented control software. Subsequently, the camera frame is resized and pre-processed to remove background noise.

Figure 5.4 shows the neural network-based classification algorithm used in the current study. Briefly, we employ a simple convolutional neural network (CNN) architecture for image classification, implemented using the Tensorflow framework (*205*). Our algorithm uses three convolutional layers with a max pooling operation between each layer and a ReLU (*155*) activation function after each pooling layer. Following the third convolutional layer, we reduce dimensionality by flattening the resulting layer into a single dimensional vector. Finally, we use two fully connected layers with a 20 % dropout layer in between (*206*). This results in an output layer with an artificial neuron for each possible analyte class (i.e. cells or beads). Accordingly, when used for classification, the trained neural network is fed a pre-processed camera frame, and produces an estimated probability for each of the possible classes in the output layer. Therefore, the final class of

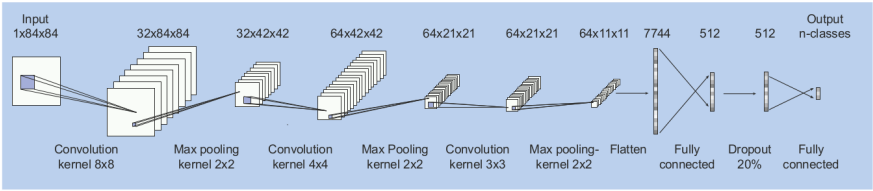

FIGURE 5.4: CNN-based classification algorithm. A custom neural network architecture is used for image classification, and is based on a relatively simple convolutional neural network. Specifically, each input frame (84 x 84 pixels) is fed into a series of three, successively smaller, convolutional layers, with intermittent max pooling layers, employing ReLU activation. Finally, two fully connected (dense) layers with a dropout layer between them are used for dimensionality reduction of the output to match the number of analyte classes. A scalar probability for each possible class (for example empty, bead or cell) is generated and classified according to the highest probability.

the analyte is determined according to the maximum estimated probability, read from the output vector. Figure 5.4 provides more details regarding the neural network architecture.

5.2.3 *Offline Algorithm Training*

We collected a labeled training dataset by recording observations from pure sample populations (e.g. pure beads). Each frame was subsequently processed to determine the presence of an analyte (either a cell or a microbead). This generated a dataset containing labeled observations for each of the following classes: empty frames, frames containing beads and frames containing cells (see Figure 5.5). Due to large time requirements for training, the CNN-based classification algorithm was typically trained offline using collected data. Due to the simplicity of our neural network and GPU-based (GeForce GTX980 Ti; NVIDIA, Santa Clara, USA) training, we were normally able to achieve sufficient accuracy within 10 epochs, corresponding to 10 minutes of training time. Training datasets normally consisted of at least 1000 frames for each class investigated. Additionally, an independent and randomly selected dataset (consisting of 15 % of the complete dataset) was used to test classification accuracy (see Figure 5.5). It should also be noted that due to the deterministic nature of neural networks, the network topology in conjunction with the connection weights contains all

the information stored in the neural network. This enables the use of a pre-trained neural network in subsequent experiments by initializing the classification network using the weights obtained during training. Accordingly, we used two different programs for training and evaluation of the classifier. Initially, training frames are collected using the aforementioned software suite (see Figure 5.3). Next, training is performed from the collected dataset using a custom program developed in Python, which stores the resulting network topology and weights upon completion. Finally, this trained neural network is loaded into the control software, for use during classification and sorting. Such an approach allowed the rapid testing of various neural network architectures in Python, independent of the control software, whilst maintaining the execution speed inherent to C++ during real-time sorting experiments.

5.2.4 *Real-time Particle Sorting*

To perform real-time cellular sorting, we use a pre-trained network (obtained using offline training). A mixture of analytes was then introduced into the microfluidic device and sorting performed on the basis of the classification output provided by the neural network. Observations were collected from sorting events to assess the correctness of the sorting decision and the resulting analyte movement (see Figure 5.7). Correct execution of the sorting signal is then verified via manual analysis of the visual data.

5.3 RESULTS AND DISCUSSION

5.3.1 *Classification algorithm*

To test the accuracy of the proposed neural network-based classifier, various neural networks were trained using a dataset containing at least 1000 images each of all analytes, namely two cell types (i.e. Jurkat and H562), microbeads, and empty camera frames. A simpler neural network topology was generally favored to increase image processing throughput and reduce training times. After comparing multiple classification algorithms, it was found that the relatively simple neural network architecture used in this study is sufficient for the sorting task investigated. Current state-of-the-art classification algorithms, such as squeezenet (*207*) or inception v-4 (*69*) did not provide justifiable improvements, when considering their increased training time requirements. However, we hypothesize that, sim-

ilar to other visual classification tasks (*69*, *207*), employing novel classification algorithms has the potential to provide significant improvements in classification accuracy.

We observed that the classifier was able to successfully classify empty frames, frames containing beads, and frames containing cells in 83.9 % +- 3.9 % (N=5) of the frames within the test dataset (see Figure 5.5 provides exemplar classifications). While cells are normally classified correctly (separate from empty frames or microbeads), we found a significant portion of misclassified frames were due to confusion of the two cell types investigated (e.g. the bottom left panel of Figure 5.5). We believe that this stems from the large size-heterogeneity of the investigated cell populations, when compared to the commercial bead populations. For example, while Jurkat cells are frequently smaller and more compact than H562 cells (see Figure 5.5 bottom right panel and top left panel respectively), there exist a significant number of deviating Jurkat cells, and we believe that this morphological diversity is the main cause of mis-classification.

To confirm that the main classification difficulty stems from analyzing two different cell types, we have trained the classifier to identify empty frames, microbeads and Jurkat cells exclusively. Indeed, we could achieve near perfect classification accuracy of 99.87 % +- 0.05 % (N=3) for this classification task. This further confirms, that the classification algorithm can successfully identify Jurkat cells despite a high heterogeneity in cell size, cell shape and internal structure.

Compared to classical image processing, the use of custom-built classification algorithms provides several key advantages. Even though an ANN requires extra computation, we found that classification speeds actually rival classical approaches. Due to the GPU-based ANN inference in the proposed system, classification speeds exceeding 500 frames per second can be reached. Moreover, the classification algorithm shows excellent classification of partial images. For example, Figure 5.6 shows images where the algorithm correctly identifies only partially visible analytes. Interestingly, the algorithmic behavior resulting in this increased accuracy has never been specified manually (for example as a definite rule set) and is achieved automatically, because the training dataset contains countless partial images and the classification algorithm automatically identifies the underlying pattern. We believe that it would be non-trivial to construct an equally flexible classification system using classical image processing, as an extensive rule set that accounts for all possible observations, must be engineered manually.

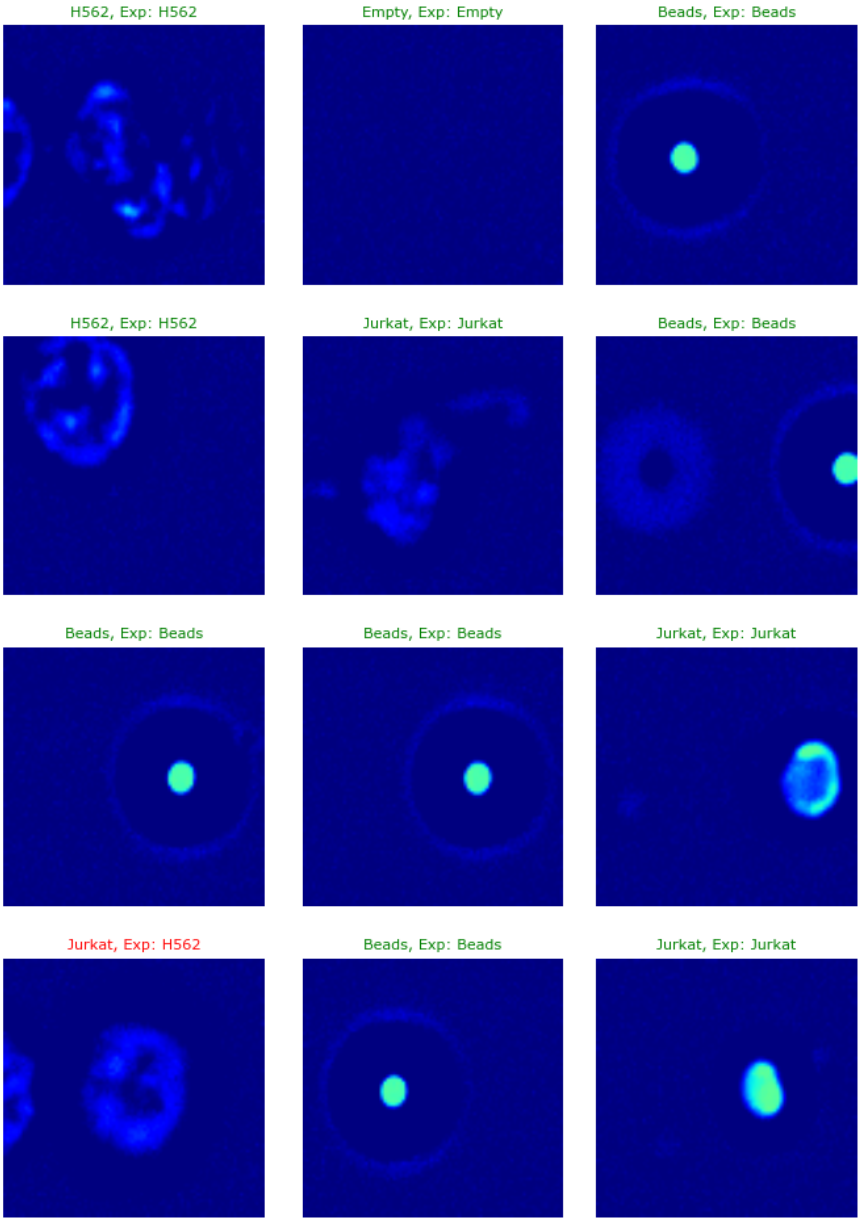
H562, Exp: H562
Empty, Exp: Empty
Beads, Exp: Beads
H562, Exp: H562
Jurkat, Exp: Jurkat
Beads, Exp: Beads
Beads, Exp: Beads
Beads, Exp: Beads
Jurkat, Exp: Jurkat
Jurkat, Exp: H562
Beads, Exp: Beads
Jurkat, Exp: Jurkat

FIGURE 5.5 *(previous page)*: Neural network for real time classification of H562 cells, Jurkat cells, polystyrene beads and empty frames. Identified class and expected (Exp) ground-truth is noted above each panel. Green titles indicate successful identification, whilst red titles indicate mis-classification. The ANN-based classifier correctly identifies 89 % observations in a large test dataset (>500 frames per analyte class), even if the analyte is only partially visible (see Figure 5.6). Incorrectly classified frames (e.g. bottom row, left panel) are often due to confusion of the two cell types. Nevertheless, we can confirm that the classifier can successfully identify analytes even if there are additional objects in the field of view (e.g. cell debris in the right frame of the center row). We hypothesize, that classification accuracy can be further improved by using a larger training dataset and more advanced neural network architectures.

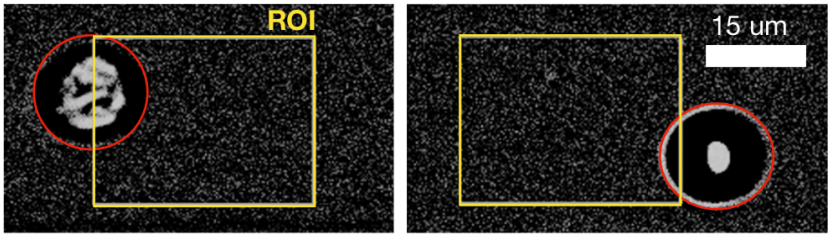

FIGURE 5.6: Identification of Partially Visible Analytes. Only the highlighted region of interest (ROI) is used for classification. The use of neural networks for image classification allows for the successful identification of partially visible analytes. The classifier has automatically incorporated this knowledge during training as the known dataset also contains partial images. In contrast, using classical image processing for classification would require the engineering of an extensive rule set to capture all special or unusual cases.

Given the initial results, it is evident that the proposed classifier can successfully be used to classify complex mixtures containing various cell types and beads. This allows us to combine the trained classifier with the microfluidic sorting architecture, for real-time perform neural network-based classification.

5.3.2 *Neural network-based sorting*

We performed a range of preliminary experiments, to serve as a proof-of-concept for the proposed sorting architecture. Based on previous results from the training process, a mixture of one cell type (Jurkat) and microbeads was sorted. This mixture was chosen, since the classifier showed near perfect (>99 %) classification accuracy for this mixture over multiple (N=3) datasets. Combination of the trained classifier with the microfluidic sorting architecture allowed real-time sorting of microbeads and Jurkat cells at rates up to 20 hertz. Significantly, it should be noted that sorting rates were inherently limited by the solenoid valves used, and adoption of an alternate sorting strategy, should easily be able to increase sorting rates by more than one order of magnitude, remembering that we could achieve classification rates in excess of 500 hertz.

To confirm correct sorting, automatic snapshots were captured during sorting events (see Figure 5.7). Manual analysis of these data suggests that our automatic particle sorting system correctly classifies passing analytes before sorting them into the correct outlet. However, it must be noted that further research is required to fully characterize the sorting strategy and its application to state-of-the-art biological assays.

5.4 CONCLUSION

This proof-of-principle study presents, for the very first time, an automatic cell sorting system based on bright-field observations and neural network-based image classification. We show that high-accuracy, real-time analyte classification within a microfluidic system is feasible using such a classification algorithm. That said, we find that while classification is possible at 500 hertz, PDMS valve-based sorting limits sorting speeds to 20 hertz. Future systems will incorporate improved sorting mechanisms, such as electrophoretic sorting (*49*) or bubble-jet sorting (*208*) to significantly enhance sorting rates.

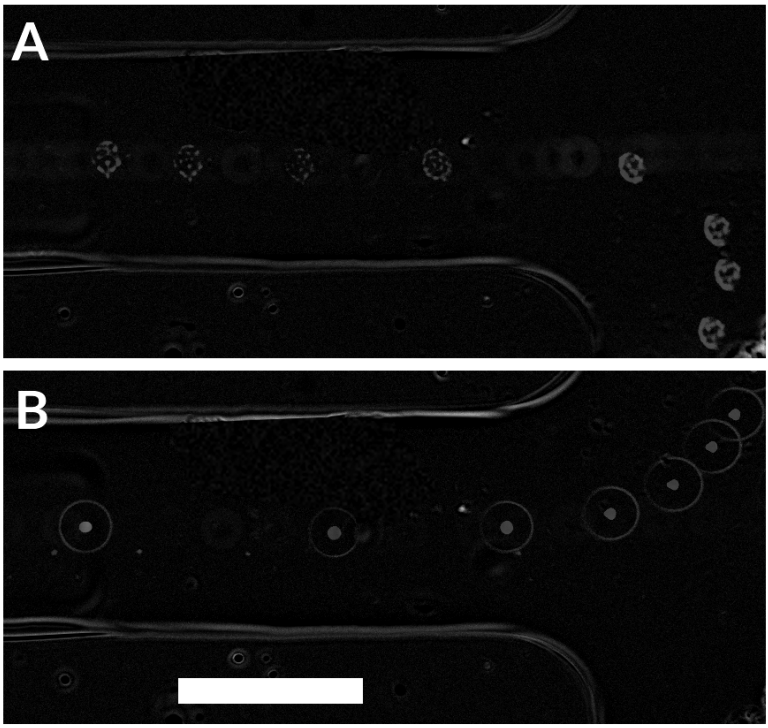

FIGURE 5.7: Use of the presented architecture for successful real-time particle sorting (scale bar - 50 µm). The two panels show an overlay of multiple images taken whilst analytes are being sorted into separate collection outlets. (A) A Jurkat cell is sorted into the bottom outlet after correct classification, (B) A polystyrene bead is sorted into the top outlet after correct classification.

Besides classification accuracy, the main appeal of ANN-based classifiers lies in their general applicability and easy adaptation to novel classification tasks. Indeed, re-training an algorithm typically only requires the use of a novel training dataset. We found that the controlled fluidic environment within a microfluidic system greatly facilitates the collection of large, labeled datasets. In our case, adaptation of the proposed setup to novel sorting problems simply requires the collection of observations of pure cell solutions and subsequently re-training using this dataset. Adjustment of the presented setup is thus feasible with less than one day of experiments, greatly reducing experimental iteration times. Accordingly, we believe that the versatile nature of the proposed system more than offsets any increases in experimental complexity. Since our classifier is based on visual observations (bright-field or dark-field), we can evaluate high-dimensionality data when compared to point-detectors normally used in flow cytometry experiments. Furthermore, the proposed classifier can recognize internal cell morphological features without the use of fluorescent labels, which lies in stark contrast to traditional FACS-based techniques. This enables a novel set of label-free sorting criteria and experiments, currently unobtainable using FACS, such as sorting cells based on cell cycle phase.

The supervised classification approach shown in this study relies on collecting a training dataset prior to classification. However, we propose that future studies could use unsupervised classification approaches to achieve automatic classification without any prior training. Typically, unsupervised classification, such as clustering, is effective without requiring labeled data, with the classification being performed based on internal criteria (often related to the maximal separation between separate groups of data). Accordingly, by using unsupervised classification the experimenter can rely on the algorithm to identify the relevant criteria without the need for external input. We hypothesize, that an unsupervised approach to cell sorting will also reduce bias in the final results at the cost of a reduced classification accuracy.

Finally, and given the presented results, we conclude that the proposed particle sorting system enables real-time, high-accuracy detection and sorting exclusively based on visual observations. Furthermore, the neural network-based algorithm is easily adapted to novel experiments and enables classification based on previously unattainable criteria. We believe this study highlights the benefits of combining microfluidics with novel control algorithms and paves the way for the large-scale application of machine learning technologies to microfluidic systems.

6

SUMMARY AND OUTLOOK

The studies shown in this thesis highlight the importance of active and passive control schemes for microfluidic high-throughput experimentation.

In Chapter 2 we have presented a high-accuracy droplet synchronization architecture. Like other droplet unit operations, this technology serves as a basic building block for multi-step biological or chemical assays. Droplet synchronization is achieved by producing a dense droplet configuration and subsequent geometric self-ordering of droplets. The presented architecture exhibits a low synchronization error rate across a wide variety of flow rates and droplet sizes investigated. Besides an extensive characterization of the synchronization effect, we have also used the device to study inter-droplet transfer between microfluidic droplets in motion, as the geometry produces highly reproducible droplet configurations. This has allowed us to investigate osmotic transfer of water and H_3O^+ ions between droplets in motion.

Future work includes applying the proposed passive synchronization architecture in a multi-step droplet assay, which could greatly increase the reliability of subsequent droplet unit operations, such as droplet merging. Due to our passive control approach, the synchronization architecture could also be included in point-of-care medical diagnosis. Nevertheless, passive control is limited by fluidic complexity and often requires extensive design iterations to optimize fluidic properties. Active control, on the other hand, can often achieve similar results at the cost of increased experimental setup complexity.

In Chapter 3 we therefore show a flexible droplet labeling system relying on active droplet sorting combined with extensive signal processing and automatic control. The proposed system can produce a large number of droplets with unique fluorescent signatures (barcodes). While other dye-based barcoding approaches rely on precise control over droplet formation (for example on-chip dye dilution), we instead decoupled droplet formation from the droplet selection process. Therefore, we can initially produce droplets exhibiting random fluorescent signatures and subsequently select and sort out droplets showing unique barcodes, resulting in unprecedented control over the final barcoded droplet population. We have

further developed a droplet storage system using additive manufacturing (3D-printing). The low-loss nature of the developed storage method highlights the utility of 3D-printing as an interface technology for microfluidics on the millifluidic or mesofluidic scale.

The proposed barcoding system could be used to allow for repeated scanning of droplet populations allowing time-course measurements with single droplets resolution. Especially applications combining our barcoding approach with other barcoding methods, such as nucleic acid-based barcoding, could yield interesting results by first allowing for cultivation and time-course measurements of cell populations using fluorescent barcodes and subsequent single-cell sequencing to investigate the underlying molecular dynamics.

We believe this study has shown that existing droplet platforms (such as droplet sorting) can be greatly augmented by employing "smart" control algorithms. Still, our barcoding system requires extensive human intervention, specially during the initial calibration phase, as all processing steps are highly parametrized. However, novel self-improving control algorithms have shown promising results in other domains, highlighting that human intervention is not necessary and might in fact be detrimental to experiment outcome (*75*).

In Chapter 4 we therefore investigate the possibility of autonomous control over fluidic conditions on a microfluidic device solely based on visual input. Fluidic control was achieved using two separate state-of-the-art, self-improving algorithms based on reinforcement learning, one using artificial neural networks (ANNs) and the other basing decisions on a large database of previous observations. Using these control algorithms, we investigated two fundamentally different fluidic regimes: laminar flow and two-phase droplet flow. By comparing algorithmic performance to human testers, we could show that super-human performance is attainable in both fluidic regimes using both algorithms. This study thus highlights the general applicability and flexibility of reinforcement learning to microfluidic control.

To our knowledge this is the first study using reinforcement learning to control non-simulated fluidic environments. Due to the optimistic results obtained we hypothesize that reinforcement learning can provide benefits to a large variety of microfluidic experiments, specially ones that require frequent human interaction. However, excellent recent results using machine learning have not been limited to control algorithms but extend to a wide variety of domains, particularly involving visual interpretation (*69*).

Chapter 5 investigates the use of ANN-based classification to sort various analytes in real-time. We have shown that artificial neural networks can be used to classify images of cells and particles flowing through a microfluidic channel. Based on this classification we could achieve real-time sorting of cells and microbeads employing exclusively visual input. Crucially, due to the use of self-learning algorithms no prior definition of sorting criteria were required to achieve excellent classification accuracy. Further, the proposed microfluidic architecture allows for facile collection of training data, in stark contrast to traditional ANN-based classification tasks, which require extensive work to assemble labeled training datasets. In summary, we have shown a fast and flexible system for real-time image-based cell sorting. This enables studying exiting and novel questions, as cell populations can now be sorted label-free and based on visual criteria. For example, we hypothesize that the proposed system could be used to separate cells based on their cell cycle phase, which was previously only possible using extensive fluorescent labeling.

In conclusion, using the studies presented in this thesis, we have shown that advanced control over the fluidic environment in microfluidic devices enables a host of novel applications in microfluidic research. We hypothesize, that the study of biological systems, such as on-chip cell culture or long-term monitoring of organisms, can profit vastly from the insights gained, as such studies often require interaction with an unpredictable organism. However, applications of smart control algorithms extend beyond biological applications, including chemical synthesis monitoring or high-throughput reactant screening. We are therefore convinced that combining microfluidic applications with advanced control schemes will enable many insights, both in microfluidics and a variety of other scientific disciplines.

A

APPENDIX

OSMOTIC INTER-DROPLET TRANSFER (CHAPTER 2)

process_osmosis.py

```
from SimpleCV import Image, VirtualCamera, Display
from matplotlib import pyplot as plt
import cv2
import numpy as np
import time
import pickle
from math import sqrt, pi


conc= '00'

filepath = 'video_data/{}.avi'.format(conc)
mode = 3
num_frames = 10000
skip = 5
show = False

# Max size deviation from average (larger is considered a coalesence event and removed)
max_deviation = 1.4

# fps = iter([150, 100, 100])

video = VirtualCamera(filepath, 'video')

# Do some of the calibration (croping channel positions, ...)
first = video.getImage()
s = first.size()
if s[0] > 1850:
  print 'Video too wide. Calibration will not work properly. Automatically cropping {} px on
                                                    the right...'.format(s[0]-1850)
  first = first.crop((0,0), (1850, s[1]))

first = first.getNumpyCv2()

edges, max_dist, channels = [], [], []

# mouse callback function
def clickedy(event,x,y,flags,param):
  if event == cv2.EVENT_LBUTTONDOWN:
    if len(edges) < 4:
      color = (255,0,0)
      edges.append((x, y))
    elif len(max_dist) < 2:
      color = (0,255,0)
      max_dist.append((x, y))
```

```
    else:
      color = (0, 0, 255)
      channels.append(x)

    cv2.circle(first,(x,y),3,color,-1)

# Create a black image, a window and bind the function to window
cv2.namedWindow('Calibration')
cv2.setMouseCallback('Calibration', clickedy)

while True:
  cv2.imshow('Calibration', first)
  if cv2.waitKey(20) & 0xFF == 13 and len(max_dist) == 2:
    break

cv2.destroyAllWindows()


# Calculate distance between green points entered
channel_size = sqrt((max_dist[0][0] - max_dist[1][0])**2 + (max_dist[0][1] - max_dist[1][1])
                                                    **2)
channel_tolerance = channel_size/4.

max_area = 2 * pi * channel_tolerance**2

print 'Processing the data for {} M ...'.format(conc)

# The actual processing
bins = {c: [] for c in channels}
before = time.time()

lowest = [s[1]] * 4

for i in range(num_frames):

  # Skip a few frames
  for j in range(skip):
    video.getImage()

  # Get image
  img = video.getImage()

  # No img: end the processing
  if img.size() == (0,0):
    break

  # Crop image
  img = img.crop((edges[2][0],edges[0][1]),(edges[3][0],edges[1][1]))

  # Pre-processing (thresholding)
  if mode == 1:
    img = img.morphGradient().binarize()
  elif mode == 2:
    img = img.erode().morphGradient().binarize()
  elif mode == 3:
    img = img.binarize().invert()

  blobs = img.findBlobs(minsize=500, maxsize=5000, threshblocksize=0, threshconstant=5,
                                                  appx_level=3)
```

```
  if blobs:
    # Remove blobs on edge
    blobs = blobs.notOnImageEdge(tolerance = 3)

    # Put blobs in bins
    for b in blobs:
      # IMPLEMENT: Check if droplet has the right color..
      # Further IMPLEMENT: Check ordering and also discard neighbouring droplets


      for bin in bins.keys():
        if abs(b.coordinates()[0] - bin + edges[2][0]) < channel_tolerance:
          bins[bin].append(b.area())

          '''
          # Record y to estimate flow speed
          if bin == channels[0] and i < 4:
            y = int(b.coordinates()[1])

            if i:
              if lowest[i-1] < y < lowest[i]:
                lowest[i] = y
            else:
              if y < lowest[i]:
                lowest[i] = y
          '''

    if show:
      blobs.show()


# Processing Speed
print 'Finished processing {} frames @'.format(i), round(i / (time.time() - before),1), '
                                                frames/s'

# Data postprocessing for plotting
ordered = sorted(bins.keys())
bins = {b:np.array(d) for b,d in bins.items()}


# Check for merged droplet and remove
for k,v in bins.items():
  avg = v.mean()
  deviation = np.absolute(v - avg)
  bins[k] = v[deviation < max_deviation*avg]


y = [bins[b].mean() for b in ordered]
yerr = [bins[b].std() for b in ordered]


# Simple x coords
x = range(len(ordered))
xerr = [0] * len(ordered)


# Normalize to minimal value and convert to %
y_min = min(y)
```

```
y = [f/y_min * 100 for f in y]
yerr = [f/y_min * 100 for f in yerr]


import csv
# with open('')
print x, y, xerr, yerr
```

MICROFLUIDIC REINFORCEMENT LEARNING (CHAPTER 4)

agents.py

```
agent12 = {
    # Various
    'scale_reward': 1.0,
    #'game_var_scale': [0.1,0.1,0.01], # Flow rates (2 pumps) and desired radius (px)

    'memory_size': 4,
    'batch_size': 32,
    'max_memory': 100000,
    'final_epsilon': 0.05,
    'n_observe': 10000, # 5000
    'n_anneal': 135000, # 120000

    # DDQN learning scheme
    'learn_delay': 1,
    'n_learn_per_q': 2,
    'target_update_delay': 5000,

    # Various environment params
    'size_x': 84, # The cropped size of the input frame (square)
    'pump_step': 0.5, # ul/min
    'total_fps': 1.5,
    'action_delay': 1./10, # Delay after performing action (before reward cala)
    'n_frame_averaged': 5, # Reward is calc from radii of past frames
    'episode_length': 250 # Number of frames in one episode
}

# Network params
params['network'] = {
    'n_filters': [16,32],
    's_filters': [8,4],
    's_pool': [0,0],
    'stride': [4,2],
    'dueling_units': [256,256], # N hidden units in value and advantage network
    'gamma': 0.99,
    'learning_rate': 0.0001, # Very sensitive?
    'gradient_clip': 10.0,   # 10 in Dueling networks [Wang 2016]
    'network_path': 'networks/test2'
}
```

doom.py

```
from random import randrange

import tensorflow as tf # Import this first!
```

```
import numpy as np
from scipy.misc import imresize
import matplotlib.pylab as plt

from vizdoom import DoomGame
from vizdoom import Mode
# from vizdoom import ScreenFormat
# from vizdoom import ScreenResolution


def histeq(im,nbr_bins=256):
    '''
        http://www.janeriksolem.net/2009/06/histogram-equalization-with-python-and.html
    '''
    #get image histogram
    imhist,bins = np.histogram(im.flatten(),nbr_bins,normed=True)
    cdf = imhist.cumsum() #cumulative distribution function
    cdf = 255 * cdf / cdf[-1] #normalize

    #use linear interpolation of cdf to find new pixel values
    im2 = np.interp(im.flatten(),bins[:-1],cdf)

    return im2.reshape(im.shape)


class CustomDoom(DoomGame):
    def __init__(self, config_path = 'config/basic.cfg', size_x = 120, competitive = False,
                                                equalize_histogram = False, *args, **
                                                kwargs):
        super(CustomDoom, self).__init__(*args, **kwargs)
        self.load_config(config_path)
        self.equalize_histogram = equalize_histogram

        # Set Screen format manually (R,G,B,DEPTH_BUFFER)
        # self.set_screen_format(ScreenFormat.CRCGCBDB)
        # self.set_screen_resolution(ScreenResolution.RES_640X480)

        self.competitive = competitive

        if self.competitive:
            # Start multiplayer game only with your AI (with options that will be used in
                                                    the competition, details in
                                                    cig_host example).
            self.add_game_args("-host 1 -deathmatch +timelimit 100000.0 "
                       "+sv_forcerespawn 1 +sv_noautoaim 1 +sv_respawnprotect 1 +
                                                            sv_spawnfarthest 1")

            # Name your agent and select color
            # colors: 0 - green, 1 - gray, 2 - brown, 3 - red, 4 - light gray, 5 - light
                                                    brown, 6 - light red, 7 - light
                                                    blue
            self.add_game_args("+name AIII +colorset 4")

            # Multiplayer requires the use of asynchronous modes, but when playing only with
                                                     bots, synchronous modes can also
                                                    be used.
            self.set_mode(Mode.PLAYER)
            #self.set_mode(Mode.ASYNC_PLAYER)
```

```
        self.init()
        self._size_x = float(size_x)
        # self._game_var_cutoff = -(1+2*NUM_WEAPONS)

        # First episode and update
        # self.new_episode()
        self.start_new_episode(force_restart = True) # For setting a correct seed and adding
                                                     bots
        self._update_state()


    def start_new_episode(self, seed = False, force_restart = False):
        if force_restart or self.is_episode_finished():
            if not seed:
                seed = randrange(1E6,1E7-1)
            self.set_seed(seed)
            super(CustomDoom,self).new_episode()

        if self.competitive:
            # Respawn player
            if self.is_player_dead():
                self.respawn_player()

            # Add bots
            self.send_game_command("removebots")
            for i in range(9):
                self.send_game_command("addbot")


    def _update_state(self):
        '''
            Despite the name this does not update the game state but
            merely updates the internal state variables.

            Currently only perform_action updates the state.
            This function should only really be called during init or
            straight after such an update.
        '''
        self._current_state = super(CustomDoom,self).get_state()

        # Game variables
        self._cur_var = np.float32(self._current_state.game_variables)

        # Image data
        if self._current_state.image_buffer is None:
            return False

        img = self._current_state.image_buffer[0]

        # Image data (RGB+DEPTH) doesnt work (no depth)
        # img = current_state.image_buffer[:3]
        # img = img.swapaxes(0,2)
        # img = np.dot(img[...,:3], [0.299, 0.587, 0.114])

        factor = self._size_x/img.shape[1]

        if not self.equalize_histogram:
            self._cur_img = (imresize(img, factor)/255.).astype('float32')
```

```
        else:
            img = imresize(img, factor)
            self._cur_img = histeq(img).astype('float32')/255.

        # self._cur_depth = (imresize(current_state.image_buffer[3], factor)/255.).astype('
                                                    float32')

        # plt.imshow(self._cur_img, cmap = plt.get_cmap('Greys'), interpolation= 'None')
        # plt.show()
        return True

    def perform_action(self,action,frames=1, auto_loop = True):
        '''
            Wraps make_action, calls update_state
            returns: state image and reward
        '''
        frames = 5
        reward = self.make_action(action,frames)
        terminal = self.is_player_dead() or self.is_episode_finished()

        if terminal:
            self.start_new_episode()

        # Update the state (the only time this should happen in the mainloop)
        # started_new = False
        self._update_state()

        return self._cur_img, np.float32(reward), self._cur_var, terminal

    def get_game_variables(self):
        return self._cur_var

    def get_state_image(self):
        return self._cur_img

    def close(self):
        pass
```

doom_test.py

```
import numpy as np
import tensorflow as tf

from doom import CustomDoom
from qagent import QAgent

import matplotlib.pylab as plt

params = {
    # Various
    'scale_reward': 0.01,
    #'game_var_scale': [0.1,0.1,0.01], # Flow rates (2 pumps) and desired radius (px)

    'memory_size': 4,
    'batch_size': 32,
    'max_memory': 50000,
    'final_epsilon': 0.05,
    'n_observe': 5000, # 5000
```

```
    'n_anneal': 120000, # 120000

    # DDQN learning scheme
    'learn_delay': 1,
    'n_learn_per_q': 1,
    'target_update_delay': 5000,

    # Various
    'size_x': 84, # the region should be square...

    # Transition sampling method
    # 'sample_type': 'uniform',
}

# Network params
params['network'] = {
    'n_filters': [16,32],
    's_filters': [8,4],
    's_pool': [0,0],
    'stride': [4,2],
    'dueling_units': [256,256],     # N units in advantage and value network
    'gamma': 0.99,
    'learning_rate': 0.0001,
    'gradient_clip': 10.0,          # 10 in Dueling networks [Wang 2016]
    'network_path': 'networks/test6'
}


# The simple doom scenario
environment = CustomDoom(config_path = '../VisualAI/config/basic.cfg', size_x = params['
                                                size_x'],
    competitive = False, equalize_histogram = False)

# The Q-learning agent
agent = QAgent(params, environment)


# Setup status plots
fig = plt.figure()
q_max = fig.add_subplot(211)
reward = fig.add_subplot(212)

q_max.set_ylim([-1,1])
q_max.set_title('Max Q')

reward.set_ylim([-1,1])
reward.set_title('Reward')

# some X and Y data
x = np.arange(2500)

q_max_val = np.zeros(2500)
reward_val = np.zeros(2500)

d_qmax, = q_max.plot(x, q_max_val)
d_reward, = reward.plot(x, reward_val)

# draw and show it
fig.canvas.draw()
```

```
plt.show(block=False)


plt_update_delay = 10
summary = []
for i in xrange(100000000):
    # Plotting updates
    if len(summary) == plt_update_delay:
        q_max_val[:-plt_update_delay] = q_max_val[plt_update_delay:]
        q_max_val[-plt_update_delay:] = [s[0] for s in summary]

        reward_val[:-plt_update_delay] = reward_val[plt_update_delay:]
        reward_val[-plt_update_delay:] = [s[1] for s in summary]

        summary = []

        d_qmax.set_ydata(q_max_val)
        d_reward.set_ydata(reward_val)
        fig.canvas.draw()

    if not agent._steps%1000:
        q_max.set_ylim([np.min(q_max_val),np.max(q_max_val)])
        reward.set_ylim([np.min(reward_val),np.max(reward_val)])

    # Advance one step
    summary.append(agent.make_step(learning = True))

    if i and not i%12500:
        agent.save_network(params['network']['network_path'])
```

environment.py

```
import time
import random
from os import listdir
from collections import Counter

import cv2
import numpy as np
import matplotlib.pyplot as plt


empty_array = np.array([])

class DropletEnvironment:
  '''
    This environment controls 3 pumps and a high-speed camera.

    Pumps: Oil, Buffer, Particles
  '''

  def __init__(self, params):
    # The cropped size of the input frame (square)
    self._size_x = params.get('size_x', 84)
    # Amount of change in the flow rates of the pumps
    self._pump_step = params.get('pump_step', 0.25) # ul/min
    # Delays: Total fps puts a hard limit on achievable fps (save the pumps...)
    self._total_fps = params.get('total_fps', 6.5)
```

```
    # Time to wait between changing the flow rate and getting the next frame (larger =
                                            slower but better correlation)
    self._action_delay = params.get('action_delay', 1./20) # Has to be much shorter than
                                            total_fps (bc total includes image processing
                                            and learning)
    # Number of frames over which reward is decided (smoothing the inherent error of hough
                                            transform)
    self._n_frames_averaged = params.get('n_frame_averaged', 5)
    # Number of frames in one episode
    self._episode_length = params.get('episode_length', 1000)

    # The camera and the pump controller
    # Importing here because simulation environment does not need it
    from pco import Edge
    from pump2 import MilliGAT, open_pump_connection

    # Counts the updates and terminates an episode if it lasts too long
    self.counter = 0

    # Action flow rate conversion for all pumps (2)
    ps = self._pump_step
    self._actions = [[0.0,0.0], [ps,0.0], [-ps,0.0], [0.0,ps], [0.0,-ps]]

    # Set up camera
    self._camera = Edge(pco_edge_type = '5.5')
    self._camera.apply_settings(
      region_of_interest=(961, 1000, 1120, 1100),
      exposure_time_microseconds=500,
      verbose = False)

    self._camera.arm(num_buffers=3)
    self._camera._prepare_to_record_to_memory()

    # Set up pumps
    self._pump_connection = open_pump_connection()
    self._pumps = [MilliGAT(self._pump_connection, adr) for adr in ['E', 'F']]

    # Set up counter for radii (average over multiple frames)
    self.radius_storage = [np.array([]) for i in range(self._n_frames_averaged)]

    self._last_action_time = time.time()
    self._cur_reward = 0.0

    self._frames_in_episode = 0
    self._total_reward = 0.0

    self.update_state()


  def get_num_actions(self):
    '''
      TODO: Less static:)
    '''
    return 5


  def start_new_episode(self):
    # Calc average reward during the previous episode
    if self._frames_in_episode > 0:
```

```
    avg_reward = float(self._total_reward)/self._frames_in_episode
  else:
    avg_reward = 0

  self._frames_in_episode = 0
  self._total_reward = 0.0

  # Set to a random flow rate within the allowed step size
  max_pump_step = int(10.0 / self._pump_step)
  for p in self._pumps:
    flow_rate = self._pump_step * random.randrange(1,max_pump_step)
    p.slew(flow_rate)

  return avg_reward


def get_state_image(self):
  return self._raw_img


def update_state(self, simulate = False):
  if simulate:
    print 'SIMULATING FRAME!'
    frame = self.simulate_frame()
  else:
    frame = self.get_camera_frame()

  self._cur_img = self.preprocess_frame(frame)

  # The raw image is now scaled (only for neural net not for reward calc)
  self._raw_img = cv2.resize(frame, (self._size_x, self._size_x), interpolation = cv2.
                                          INTER_LINEAR).astype('float32')
  self._raw_img /= 255.


def perform_action(self, action, frame = False, plot = False):
  '''
    Perform an action (change flow rates),
    get new frame from the camera,
    process frame to get reward,

    Also assures that delay between frames and delay between changing flow rate
    and taking a frame is more or less constant.

    Returns: frame and reward
  '''

  delay_per_frame = 1./self._total_fps # Run at x fps overall
  delay_after_action = self._action_delay # Wait a bit between changing and taking a frame

  diff = delay_per_frame - (time.time() - self._last_action_time)
  if diff > 0:
    time.sleep(diff-0.001)

  self._last_action_time = time.time()

  # Change the pump flow rates according to the action
  flow_change = self._actions[action]
  for i, change in enumerate(flow_change):
```

```
      self._pumps[i].change_flow_rate(change)

    # Wait a bit (might help to increase correlation between action and reward)
    if delay_after_action > 0:
      time.sleep(delay_after_action)

    # Get a new frame from the camera and preprocess it
    self.update_state()

    # Calculate reward for frame
    self._cur_reward = self.calc_reward(self._cur_img, plot_frame = plot)

    terminal = False
    # End episode after a set time period
    if self.counter > self._episode_length:
      self.counter = 0
      terminal = True
    else:
      self.counter += 1

    self._frames_in_episode += 1 # TODO: unify with counter (it's the same)
    self._total_reward += self._cur_reward

    # Use raw image to learn
    return self._raw_img, self._cur_reward, terminal


  def preprocess_frame(self, frame):
    # Convert to grayscale
    if frame.ndim == 3:
      frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    # Simple thresholding
    __, frame = cv2.threshold(frame, 100, 200, cv2.THRESH_TOZERO)

    # Adaptive thresholding
    # (bc we want this to be as automated as possible)
    # opts = {'maxValue': 1,
    #     'adaptiveMethod':cv2.ADAPTIVE_THRESH_GAUSSIAN_C,# ADAPTIVE_THRESH_MEAN_C
    #     'thresholdType': cv2.THRESH_BINARY,
    #     'blockSize': 51,
    #     'C': 2}

    # frame = cv2.adaptiveThreshold(frame, **opts)

    # # Now erode to segment the droplet from the walls and remove single pixel noise
    kernel = np.ones((3,3), dtype='int')
    frame = cv2.dilate(frame, kernel)

    # plt.imshow(frame)
    # plt.show()
    return frame


  def calc_reward(self, frame, plot_frame=False):
    '''
      Calculate reward based on droplet size, size uniformity and number of particles

      Only use plotting for debugging; also do not use both ploting options at the same time
```

```
'''

# Find droplets (large circles)
opts_droplets = {
  'method': cv2.cv.CV_HOUGH_GRADIENT,
  'dp': 1, 'minDist': 37,
  'minRadius': 20,
  'maxRadius': 34,
  'param1': 5.0, # Canny edge detector higher threshold
  'param2': 29, # Circlyness (smaller = more false positives)
}

# Hough circle transform
circles = cv2.HoughCircles(frame, **opts_droplets)

# Plot
if plot_frame:
  plt.imshow(frame)
  fig = plt.gcf()
  gca = fig.gca()

reward = 0.0

if circles is not None:
  circles = np.round(circles[0, :]).astype("int")

  # n_particles = []
  if plot_frame:
    for c in circles:
      # plot Droplets
      circ = plt.Circle(c[:2], c[2], color='g', alpha = 0.35)
      gca.add_artist(circ)
      #plt.text(c[0]-10,c[1]-40, str(n_particles[-1]), fontsize = 30, color = 'b')

  # Add the radii to the storage
  self.radius_storage.pop()
  self.radius_storage.insert(0, circles[:,2])

# No circles found
else:
  self.radius_storage.pop()
  self.radius_storage.insert(0, empty_array)

if plot_frame:
  # Maximize window, show plot
  mng = plt.get_current_fig_manager()

  # Qt backend (anaconda)
  # mng.window.showMaximized()

  # WX backend (canopy)
  # mng.frame.Maximize()

  # Tk backend (standard)
  mng.window.state('zoomed')
  mng.resize(*mng.window.maxsize())

  plt.show()
```

```
    # Calc reward (mean squared error over past few frames)
    self.desired_radius = 27
    error = np.concatenate(self.radius_storage) - self.desired_radius

    # Absolute error (linear increase from edge of circle detection to max)
    if error.shape[0] > 0:
      m = np.mean(np.absolute(error)) # Abs mean error
      if m > 7:
        reward = 0
      elif m < 0.5:
        reward = 1.0
      else:
        reward = 1.-(m/8.)
    else:
      reward = 0.0

    return reward


  def simulate_frame(self, size = (100,100)):
    # A random frame
    # frame = np.random.random(size) * 256
    # frame = frame.astype('int')

    # Grab a random frame from the folder
    folder = 'test_frames/'
    file = folder + random.choice(listdir(folder))
    frame = cv2.imread(file)#, cv2.CV_LOAD_IMAGE_GRAYSCALE)

    return frame


  def get_camera_frame(self):
    frame = self._camera.record_to_memory(num_images=1, verbose=False)[0]

    # Scale image across 0-255
    print 'INVESTIGATE ME! NEED NORMALIZATION?'
    frame -= np.min(frame)
    frame = 255. * frame.astype('float') / np.max(frame)

    # Converts to uint8 (needed for opencv stuff)
    frame = cv2.convertScaleAbs(frame)
    return frame


  def close(self):
    for p in self._pumps:
      p.close() # Mainly stops pumps

    self._pump_connection.close() # Serial connection for pumps
    self._camera.close()


class SimulationEnvironment(DropletEnvironment):
  def __init__(self):
    # Counts the updates and terminates an episode if it lasts too long
    self.counter = 0

    # Amount of change in the flow rates of the pumps
```

```
    self._pump_step = 0.25 # ul/min

    # Pump min and max flow rates
    # self._pump_range = (0.01, 12.) # ul/min

    # Action flow rate conversion for all pumps (3)
    ps = self._pump_step
    self._actions = [[0.0,0.0], [ps,0.0], [-ps,0.0], [0.0,ps], [0.0,-ps]]

    # Set up counter for radii (average over multiple frames)
    self.n_frames_averaged = 3
    self.radius_storage = [np.array([]) for i in range(self.n_frames_averaged)]

    self.update_state(True)

  def new_episode(self):
    pass

  def perform_action(self, action = False, frame = False, frame_delay = 0.0, plot = False):
    '''
      Perform an action (change flow rates),
      then wait a bit (frame_delay),
      get new frame from the camera,
      process frame to get reward,

      Returns: frame and reward
    '''

    # Change the pump flow rates according to the action
    # flow_change = self._actions[action]
    # for i, change in enumerate(flow_change):
    #   self._pumps[i].change_flow_rate(change)

    # Wait a bit (might help to increase correlation between action and reward)
    if frame_delay > 0.0:
      time.sleep(frame_delay)

    # Get a new frame from the camera and preprocess it
    self.update_state(simulate = True)

    # Calculate reward for frame
    reward = self.calc_reward(self._cur_img, plot_frame = plot)

    terminal = False
    # End episode after a set time period
    if self.counter > 500:
      self.counter = 0
      terminal = True
    else:
      self.counter += 1

    return self._raw_img, reward, terminal


  def simulate_frame(self, size = (100,100)):
    # A random frame
    # frame = np.random.random(size) * 256
    # frame = frame.astype('int')
```

```
    # Grab a random frame from the folder
    folder = 'test_videos_ink/'
    file = folder + random.choice(listdir(folder))
    frame = cv2.imread(file)#, cv2.CV_LOAD_IMAGE_GRAYSCALE)

    return frame

  def close(self):
    pass



if __name__ == '__main__':
  env = SimulationEnvironment()

  for i in range(10):
    env.perform_action(plot = True)
```

env_tests.py

```
import unittest
import random
import time

import numpy as np
from numpy.testing import assert_array_equal

from environment import DropletEnvironment


class TestEnvironment(unittest.TestCase):

  def test_init(self):
    env = DropletEnvironment()

    exp = [env._pump_step, -env._pump_step, env._pump_step, -env._pump_step, env._pump_step,
                                            -env._pump_step]
    assert_array_equal(env._action_conv, np.array(exp))


  def test_make_action(self):
    env = DropletEnvironment()

    for i in range(0):
      action = np.zeros(6, dtype='int')
      action[random.randrange(6)] = 1
      frame, reward = env.make_action(action, frame_delay = 0.0, plot = True)


  def test_profile_make_action(self):
    env = DropletEnvironment()
    example_frames = [env.simulate_frame() for i in range(10)]

    before = time.time()
    for i in xrange(100):
      action = np.zeros(6, dtype='int')
      action[random.randrange(6)] = 1
      frame, reward = env.make_action(action, frame = random.choice(example_frames),
                                              frame_delay = 0.0, plot = False)
```

```
    diff = time.time() - before
    per = round((diff/100.) * 1000.,4)
    print 'Preprocessing & Reward Calculation: {} ms per action ({} fps)'.format(per, round(
                                            1000./per,2))
    time.sleep(0.01) # to display correctly



if __name__ == '__main__':
  unittest.main()
```

nn.py

```
## ConvQ network for Q-learning agent, Oliver Dressler 2016
import numpy as np

from keras.layers import Convolution2D, MaxPooling2D, Input, Dense, Flatten, Lambda, Merge
from keras.models import Model
from keras.optimizers import RMSprop
from keras import backend as K


class ConvQ:
    def __init__(self, **kwargs):
        self._gamma = 0.99

    def setup_network(self, input_shape, num_actions, add_len,
        n_filters = [32,64,64], s_filters = [8,4,3],
        s_pool = [2,0,0], stride = [4,2,1], dueling_units = [512,512],
        gamma = 0.99, learning_rate = 0.00025, gradient_clip = 10, setup_target = False,
        network_path = ''):

        self.network_path = network_path
        self._num_actions = num_actions

        # Input
        self._in_layer = Input(shape = input_shape)

        # Convolution and pooling
        cur_layer = self._in_layer
        for i, n_filt in enumerate(n_filters):
            cur_layer = Convolution2D(n_filt, s_filters[i], s_filters[i],
                init = 'glorot_normal', activation = 'relu', dim_ordering = 'tf',
                subsample = (stride[i], stride[i])) (cur_layer)

            if s_pool[i]:
                cur_layer = MaxPooling2D((s_pool[i], s_pool[i]), dim_ordering = 'tf') (
                                                            cur_layer)

            print cur_layer._keras_shape

        cur_layer = Flatten()(cur_layer)

        # Normal (non dueling architecture)
        # layer = Dense(dueling_units[0], init = 'he_normal', activation = 'relu')(cur_layer
                                                    )
        # self._out_layer = Dense(num_actions, init = 'he_normal')(layer)
```

```
        # Dueling network architecture
        dueling = []
        for i_net, n_out in enumerate([1, num_actions]):
            layer = Dense(dueling_units[i_net], init = 'glorot_normal', activation = 'relu')(
                                                        cur_layer)
            dueling.append(Dense(n_out, init = 'glorot_normal')(layer))

        # out = value + adv - mean(adv)
        cur_layer = Merge(mode = 'concat')(dueling)
        self._out_layer = Lambda(lambda a: K.expand_dims(a[:,0], dim=-1) + a[:,1:] - K.mean(
                                                    a[:, 1:], keepdims=True), output_shape
                                                    =(num_actions,))(cur_layer)

        # And the final model (only needs to be compiled in on-line network)
        # target network is not optimized
        self._model = Model(input = self._in_layer, output = self._out_layer)

        # Turns out it is easier to set this up for both networks...
        # inp = Input(shape = input_shape)
        if setup_target:
            opt = RMSprop(learning_rate, clipvalue = gradient_clip)
            self._model.compile(optimizer = opt, loss = 'mse')
        else:
            # We actually never need to optimize the target network (cant hurt to compile
                                                            though?)
            self._model.compile(optimizer = 'rmsprop',
                loss = 'mean_squared_error',
                metrics = ['accuracy'])

        # Target Q network (nested within parent network)
        if setup_target:
            self._target_network = ConvQ()
            self._target_network.setup_network(input_shape, num_actions, add_len,
                    n_filters = n_filters, s_filters = s_filters,
                    s_pool = s_pool, stride = stride, dueling_units = dueling_units,
                    gamma = gamma, learning_rate = learning_rate, gradient_clip =
                                                                gradient_clip,
                    network_path = network_path,
                    setup_target = False)

            # Try to load previous weights
            self.load(self.network_path)

            # And update the target network
            self.update_target_network()


    def update_target_network(self):
        tn = self._target_network._model.layers

        for i, l in enumerate(self._model.layers):
            weights = l.get_weights()
            tn[i].set_weights(weights)

        print 'Updated target network'


    def learn(self, img_cur, img_next, act_cur, reward_cur, terminal):
        '''
```

```
            DDQN [van Hasselt 2015] using the provided minibatch,
            performs one gradient step using specified optimizer
        '''
        batch_size = reward_cur.shape[0]

        # Make some predictions (get Q)
        tn = self._target_network
        y_batch = self._model.predict(img_cur, batch_size = batch_size) # Current state
        q_next = self._model.predict(img_next, batch_size = batch_size) # Next state
        q_next_target = tn._model.predict(img_next, batch_size = batch_size) # Next state as
                                                    predicted by target network (rarely
                                                   updated to increase stability)

        for i in xrange(batch_size):
            # Only this value gets corrected (do not learn/correct non-deep decisions)
            chosen_action = act_cur[i]

            # Terminal = only reward
            if terminal[i]:
                y_batch[i, chosen_action] = reward_cur[i]
            else:
                # Online DQN (no target network)
                # y_t = reward_cur[i] + self._gamma * q_next[i, m_online]

                # DDQN (van Hasselt 2015)
                m_online = np.argmax(q_next[i])
                Q = q_next_target[i, m_online]
                y_t = reward_cur[i] + self._gamma * Q

                # Clipping update magnitude (methods Mnih 2015)
                diff = y_t - y_batch[i, m_online]
                if diff > 1:
                    y_t = y_batch[i, m_online] + 1.0
                elif diff < -1:
                    y_t = y_batch[i, m_online] - 1.0

                y_batch[i, chosen_action] = y_t

        # perform gradient step
        self._model.train_on_batch(img_cur, y_batch)


    def get_best_action(self, state):
        state = state.reshape([1] + list(state.shape))
        out_act = self._model.predict(state, batch_size = 1)[0]
        max_act = np.argmax(out_act)
        return max_act, out_act[max_act]


    def dump_memory(self, path):
        pass


    def get_best_action_observe(self, state):
        return self.get_best_action(state)


    def save(self, path = ''):
        if not path:
```

```
            path = self.network_path
        self._model.save_weights(path, overwrite = True)


    def load(self, path = ''):
        if not path:
            path = self.network_path

        try:
            self._model.load_weights(path)
            print 'Successfully loaded weights from:', path
        except Exception, e:
            print 'Could not load previous weights:', e


    def close(self):
        pass


if __name__ == '__main__':
    cq = ConvQ()
    cq.setup_network((162,160,4), 5, 0, setup_target = True)

    # A little test for update_target_network
    inp = np.random.random((162,160,4))
    prev = cq.get_best_action(inp)
    #assert  prev != cq._target_network.get_best_action(inp)
    cq.update_target_network()
    assert cq.get_best_action(inp) == cq._target_network.get_best_action(inp) == prev
    #cq.save()

    print 'Everything seems to be ok...'
```

prime_pumps.py

```
from pump2 import open_pump_connection, MilliGAT

import random, time
connection = open_pump_connection()

pump1 = MilliGAT(connection, 'E')
pump2 = MilliGAT(connection, 'F')

pump1.slew(1)
pump2.slew(1)
time.sleep(10)

pump1.close()
pump2.close()

connection.close()
```

pump2.py

```
import _winreg as winreg
import itertools
from time import sleep
import re
```

```
import serial
from serial.serialutil import SerialException

version_regex = re.compile('(\d\d*)(\D)')
# Hardware details
pump_details = {
    206140022: {'type': 'M50', 'backlash': 14.607, 'vol_rev': 629.680},
    206140020: {'type': 'M50', 'backlash': 13.224, 'vol_rev': 627.080}
    }
# Allowed flow rates [ul/min]
type_details = {'M50': {'flow_rates': (1., 25000), 'gear_ratio': 9.88}}


def enumerate_serial_ports():
    """
        From: http://stackoverflow.com/questions/1205383/listing-serial-com-ports-on-windows
    """
    path = 'HARDWARE\\DEVICEMAP\\SERIALCOMM'
    try:
        key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, path)
    except WindowsError:
        raise IterationError

    for i in itertools.count():
        try:
            val = winreg.EnumValue(key, i)
            yield (str(val[1]), str(val[0]))
        except EnvironmentError:
            break


def open_pump_connection(port = 'COM2'):
    '''
        Opens a serial connection to the pumps
    '''
    # ports = enumerate_serial_ports()

    connection = serial.Serial(port=port, timeout=0.25)
    return connection


class MilliGAT:

    '''
    This class controls a milliGAT low flow pump,
    which includes a simple three port valve.
    '''

    def __init__(self, ser, adress):
        '''
        adress is the specific adress of the pump (daisy chained).
        num_pos defines the number of positions on the valve
        '''
        self.ser = ser
        self.adress = adress
        self.current_rate = 0


    def read_response(self):
```

```
        '''
        Wait 0.05s for response from device
        '''
        resp = ''
        sleep(0.05)
        while self.ser.inWaiting() > 0:
            resp += self.ser.read(1)
        return resp


    def change_flow_rate(self, delta_r):
        '''
            Change slew using a definded delta flow rate
        '''
        new_rate = self.current_rate + delta_r
        if not 0.25 <= new_rate <= 10:
            # print 'Flow rate too extreme!!!'
            pass
        else:
            self.slew(self.current_rate + delta_r)


    def slew(self, pump_rate):
        '''
        Start pumping at constant rate [ul/min]
        '''

        if pump_rate == 0:
            pass
        elif not 0.1 <= pump_rate <= 500:
            print '{} ul/min not allowed: 0.1 - 500'.format(pump_rate)
            return

        self.current_rate = pump_rate

        # convert from ul/min to microsteps/
        rate = int(round(pump_rate * 2432/60))

        # set pump rate
        command = self.adress + 'SL=' + str(rate) + '\r\n'
        self.ser.write(command)

        # self.read_response()


    def stop(self):
        '''
        Stop the pump
        '''
        self.slew(0)


    def moving(self):
        '''
        Returns True if the pump is moving.
        '''
        self.read_response()

        # Get current rate from device
```

```
        command = self.adress + 'PR MV\r\n'
        self.ser.write(command)

        resp = self.read_response()

        if resp:
            pattern = self.adress + 'PR MV[^\d]*(\d)'
            res = re.compile(pattern).search(resp, 0)
            if res.groups()[0] == '1':
                return True
            elif res.groups()[0] == '0':
                return False
        else:
            return False


    def close(self):
        self.stop()



class MilliGATOld:
    '''
        Controls a single MilliGAT pump.
        Since these pumps are so simple we only really need one method
        .slew() # pumps at a constant rate.
    '''
    def __init__(self, port):

         # Open serial connection
        self.connection = serial.Serial(port=port[0], timeout=0.25)

        if not self.connection.isOpen():
            self.connection.close()
            raise SerialException

        # Check if device response makes sense (get serial number)
        self.connection.write('PRINT SER\n\r')
        response = self.connection.read(1000).replace(' ', '')
        version = version_regex.search(response)

        if version:
            version = version.groups()
            try:
                # Addr is (adress, serial_nr)
                self.serial_nr = int(version[0])
            except:
                self.connection.close()
                raise SerialException
        else:
            self.connection.close()
            raise RuntimeError('Could not connect to pump in port {}!'.format(port[0]))

        self.current_rate = 0
        self.p_details = pump_details[self.serial_nr]
        self.t_details = type_details[self.p_details['type']]

        # Set the correct backlash value
        self.connection.write('BLSH={}\r\n'.format(self.p_details['backlash']))
```

```
        d = self.connection.read(1000)

        # Set maximum flow rate
        max_rate = self.t_details['flow_rates'][1]/60.
        self.connection.write('VM={}\r\n'.format(max_rate))

        # Set the correct conversion factor on the device
        fact = (256. * (360./1.8) * self.t_details['gear_ratio'])/self.p_details['vol_rev']
        fact = round(fact, 3)

        self.connection.write('MUNIT={}\r\n'.format(fact))

        # Save to non-volatile memory just because
        self.connection.write('SAVE\r\n')
        sleep(0.05)
        self.connection.flushInput()

        self.connection.write('PRINT MUNIT\r\n')
        sleep(0.05)
        data = self.connection.read(1000)

        munit = data.split('PRINT MUNIT\r\n')[1].split('\r\n')[0]
        munit = float(munit)

        if munit != fact:
            perc = round(100. * (1 - min(munit,fact)/max(munit,fact)),2)
            print 'Pump on {}: incorrect calibration (off by {} %).'.format(port[0],perc)
        else:
            print 'Calibrated pump {} and saved values to non-volatile memory.'.format(self.
                                                        serial_nr)


    def change_flow_rate(self, delta_r):
        '''
            Change slew using a definded delta flow rate
        '''
        self.slew(self.current_rate + delta_r)


    def slew(self, rate):
        '''
            Pumps fluid at a constant rate [ul/min]
        '''
        assert type(rate) in (float,int)

        if rate == self.current_rate:
            return
        elif rate == 0:
            self.stop_pump()
        else:
            # Check if rate within range
            mi, ma = self.t_details['flow_rates']
            if not mi <= abs(rate) <= ma:
                print 'Rate should be within [{}, {}] ul/min'.format(mi, ma)
                return

            rate_sec = rate / 60.  # Also convert to ul/s

            self.current_rate = rate
```

```
            # Send the command to the pump
            self.connection.write('SLEW={}'.format(rate_sec))

    def stop_pump(self):
        self.connection.write('SSTP 0')

    def close_pump(self):
        if self.current_rate != 0:
            self.stop_pump()
        self.connection.close()


if __name__ == '__main__':
    import random, time
    connection = open_pump_connection()

    pump1 = MilliGAT(connection, 'E')
    pump2 = MilliGAT(connection, 'F')

    pump1.slew(3)
    pump2.slew(1)
    time.sleep(30)

    for i in xrange(0):

        p1 = round(random.random() * 19.9 + 0.1,1)
        p2 = round(random.random() * 19.9 + 0.1,1)

        print 'Flow rates:', p1, p2

        pump1.slew(p1)
        pump2.slew(p2)

        assert pump1.moving()
        assert pump2.moving()
        time.sleep(5)


    pump1.stop()
    pump2.stop()

    assert not pump1.moving()
    assert not pump1.moving()

    pump1.close()
    pump2.close()

    connection.close()
```

qagent.py

```
## Q-learning Agent, Oliver Dressler 2016
import time
from random import randrange, sample, random, choice
import itertools as it

import numpy as np
from scipy.misc import imresize
```

```
import matplotlib.pylab as plt

from nn import ConvQ
from environment import DropletEnvironment


def simple_actions_generator(n_actions):
    a = np.zeros((n_actions,n_actions))
    np.fill_diagonal(a, 1)
    return a


def permutation_actions_generator(n_actions):
    return np.array(list(it.product(range(2), repeat=n_actions)))


class QAgent:
    def __init__(self, params, environment):
        # All the required agent parameters
        self._memory_size = params['memory_size']
        self._batch_size = params['batch_size']
        self._max_memory = params['max_memory']
        self._final_epsilon = params['final_epsilon']
        self._n_observe = params['n_observe']
        self._n_anneal = params['n_anneal']
        self._learn_delay = params['learn_delay']
        self._n_learn_per_q = params['n_learn_per_q']
        self._target_update_delay = params['target_update_delay']
        self._scale_reward = params['scale_reward']
        # self._game_var_scale = params['game_var_scale']

        # The passed environment
        self._environment = environment

        self.rendered_episodes = 0
        self._num_actions = self._environment.get_num_actions() # len(self._actions)

        # Get a first state (img and game vars) and assemble current state (stack 4 times)
        img = self._environment.get_state_image()
        self._img_shape = img.shape
        self._current_state = np.stack([img]*self._memory_size, axis = 2)

        # Setup Q network
        self._nnet = ConvQ()
        self._nnet.setup_network(self._current_state.shape,
            self._num_actions, 0,#v.shape[0],
            # Sets up an identical nested target network (which will receive setup_target =
                                                        False to stop the turtles)
            setup_target = True,
            **params['network'])

        # Setup epsilon function
        def epsilon():
            epsilon_grad = (1.0 - self._final_epsilon) / self._n_anneal
            if self._steps < self._n_observe:
                return 1.0
            elif self._steps < (self._n_observe + self._n_anneal):
                s = self._steps - self._n_observe
```

```
            return 1.0 - s * epsilon_grad
        else:
            return self._final_epsilon

    self._epsilon = epsilon

    # Global steps and action statistics
    self._steps = 4
    self._action_occ = np.zeros(self._num_actions)

    # Initialize the memory (using fized size numpy arrays)
    maxm = params['max_memory']
    self._memory_img = np.zeros([maxm] + list(self._img_shape) + [4], dtype = 'float32')
                                                    # Image state
    self._memory_act = np.zeros((maxm), dtype = 'uint8') # Actions (index of action
                                                    chosen)
    self._memory_r = np.zeros((maxm), dtype = 'float32') # Rewards
    self._memory_t = np.zeros((maxm), dtype = 'bool') # Terminal

    self.new_epoch()
    print 'Set up environment, neural network and Q-agent...'


def get_flow_rates(self):
    rates = [self._environment._pumps[0].current_rate,
        self._environment._pumps[1].current_rate]
    return rates


def new_epoch(self):
    self.new_episode()

    # Reset episode and action counters
    self.rendered_episodes = 0
    self._action_occ.fill(0.0)

    # mainly to advance self._step
    self.make_step(learning = True)


def new_episode(self):
    self.rendered_episodes += 1
    self._current_state.fill(0.0)

    # Resets the pumps to a random level
    # returns the average reward of the last episode
    avg_reward = self._environment.start_new_episode()

    print 'Episode {}, Reward: {}'.format(self.rendered_episodes, avg_reward)
    time.sleep(0.25)


def make_step(self, learning = True):
    if learning:
        action_index, max_q = self._nnet.get_best_action(self._current_state)
        # choose an action epsilon greedily (if not observing)
        if random() <= self._epsilon():
            action_index = randrange(self._num_actions)
```

```
        else:
            action_index, max_q = self._nnet.get_best_action(self._current_state)
            # print action_index

        # action = self._actions[action_index,:]

        # Statistics
        self._action_occ[action_index] += 1

        # run the selected action and observe next state and reward
        # print 'CHANGE THIS IF NOT USING DOOM TEST: qagent.py: 147'
        # act = [0 for i in range(self._num_actions)]
        # act[action_index] = 1
        # print act
        # x_t, r_t, __, terminal = self._environment.perform_action(act)

        x_t,r_t,terminal = self._environment.perform_action(action_index)

        # scale reward & game variables
        r_t *= self._scale_reward

        # game_vars = np.multiply(self._environment.get_game_variables(), self.
                                                    _game_var_scale)

        # store the transition in memory
        #action_store = np.zeros((self._num_actions))
        #action_store[action_index] = 1

        # Create the next state for both image and game variables
        if not terminal:
            x_t = np.reshape(x_t, (self._img_shape[0], self._img_shape[1], 1))
            next_state = np.append(x_t, self._current_state, axis = 2)
            next_state = next_state[:,:,:-1]
        else:
            x_t = np.reshape(x_t, (self._img_shape[0], self._img_shape[1], 1))
            next_state = np.append(x_t, np.zeros_like(self._current_state), axis = 2)
            next_state = next_state[:,:,:-1]

        # var_t = np.reshape(game_vars, (len(self._game_var_scale), 1))
        # next_game_var = np.append(var_t, self._current_game_var, axis = 1)
        # next_game_var = next_game_var[:,:-1]

        # Save the transition
        s = self._steps%self._max_memory
        self._memory_img[s] = self._current_state
        self._memory_act[s] = action_index #action_store
        self._memory_r[s] = r_t
        self._memory_t[s] = terminal
        # self._memory_y[s] = 1.0 # 1 makes sure that new transitions are favoured (in
                                                    prioritized sampling)

        # New episode was started
        if terminal:
            self.new_episode()

        else:
            self._current_state = next_state
            # self._current_game_var = next_game_var
```

```
        # Target update
        if learning and not self._steps % self._target_update_delay:
            self._nnet.update_target_network()

        # Q learning
        if learning:
            if self._steps > self._n_observe:
                if not self._steps % self._learn_delay:
                    for i in range(self._n_learn_per_q):
                        self._qlearn()

        self._steps += 1

        # Some status info (not for the first round...)
        if not self._steps % 5000:
            n_a = sum(self._action_occ)
            fillers = [self._steps, self._epsilon(), np.round(100. * self._action_occ / n_a,
                                                               1)]
            print 'T: {}, Epsilon: {}, Actions: {}'.format(*fillers)
            self._action_occ.fill(0.0)

        # Return some stats to the trainer (for rt visualization)
        return max_q, r_t


    def _qlearn(self):
        '''
            Assemble the minibatch then update the nn using batch.
        '''
        if self._steps > self._max_memory:
            all_ind = range(self._max_memory)
        else:
            all_ind = range(self._steps)

        # uniform random minibatch
        selected = np.array(sample(all_ind, self._batch_size))
        sel_next = (selected + 1)%self._max_memory

        # Learn batch
        self._nnet.learn(self._memory_img[selected], self._memory_img[sel_next],
            self._memory_act[selected], self._memory_r[selected],
            self._memory_t[selected])


    def save_network(self, path):
        self._nnet.save(path)


    def close(self):
        self._environment.close()


if __name__ == '__main__':
    env = DropletEnvironment()

    params = {
        # Various
        'scale_reward': 0.01,
        'game_var_scale': [0.1,0.1,0.01], # Flow rates (2 pumps) and desired radius (px)
```

```
        'memory_size': 4,
        'batch_size': 32,
        'max_memory': 150000,
        'final_epsilon': 0.05,
        'n_observe': 50, # 5000
        'n_anneal': 100000, # 120000

        # DDQN learning scheme
        'learn_delay': 4,
        'n_learn_per_q': 1,
        'target_update_delay': 5000,

        # Various
        'size_x': 120, #@ 16:10 128x80, 160x100, 120x75
                       # @ 4:3 120x90, 100x75

        # Transition sampling method
        'sample_type': 'uniform',
    }

    # Network params
    params['network'] = {
        'n_filters': [32,64,64],
        's_filters': [8,4,3],
        's_pool': [0,0,0],
        'stride': [4,2,1],
        'dueling_units': [512,512],     # N units in advantage and value network
        'gamma': 0.99,
        'learning_rate': 0.0001,
        'gradient_clip': 10.0,          # 10 in Dueling networks [Wang 2016]
    }

    agent = QAgent(params, env)

    before = time.time()
    for i in xrange(10000):
        agent.make_step(learning = True)

        if not i%250 and i:
            fps = round(250./(time.time()-before),2)
            print 'Step {}, {} fps'.format(i, fps)
            before = time.time()
            agent.save_network('networks/test_net')

    agent.close()
```

trainer.py

```
import time

import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import numpy as np

from environment import DropletEnvironment
from qagent import QAgent
```

```
from agents import *

params = agent2

plot_it = True
plot_frame = True
learning = True

# All we need...
env = DropletEnvironment(params)
agent = QAgent(params, env)

if plot_it:
    # Setup status plots
    fig = plt.figure()
    q_max = fig.add_subplot(411)
    reward = fig.add_subplot(412)
    flow_rates = fig.add_subplot(413)

    if plot_frame:
        frame = fig.add_subplot(414)
        frame.axis('off')

    q_max.set_ylim([-1,1])
    q_max.set_title('Max Q')

    reward.set_ylim([0,1])
    reward.set_title('Reward')

    flow_rates.set_ylim([0,10.0])
    flow_rates.set_title('Flow Rates')

    # some X and Y data
    x = np.arange(2500)
    q_max_val = np.zeros(2500)
    reward_val = np.zeros(2500)
    flow1_val = np.zeros(2500)
    flow2_val = np.zeros(2500)

    d_qmax, = q_max.plot(x, q_max_val)
    d_reward, = reward.plot(x, reward_val)
    d_flow1, = flow_rates.plot(x, flow1_val, 'r')
    d_flow2, = flow_rates.plot(x, flow2_val, 'b')
    # show_frame = frame.imshow(np.zeros([params['size_x']]*2))

    # draw and show it
    plt.ion()
    plt.show()

    # # Training loop
    plt_update_delay = 25
    summary = []


before = time.time()
for i in xrange(10000000):
    if plot_it:
        # Plotting updates
        if len(summary) == plt_update_delay:
```

```
            q_max_val[:-plt_update_delay] = q_max_val[plt_update_delay:]
            q_max_val[-plt_update_delay:] = [s[0] for s in summary]

            reward_val[:-plt_update_delay] = reward_val[plt_update_delay:]
            reward_val[-plt_update_delay:] = [s[1] for s in summary]

            flow1_val[:-plt_update_delay] = flow1_val[plt_update_delay:]
            flow1_val[-plt_update_delay:] = [s[2] for s in summary]

            flow2_val[:-plt_update_delay] = flow2_val[plt_update_delay:]
            flow2_val[-plt_update_delay:] = [s[3] for s in summary]

            summary = []

            d_qmax.set_ydata(q_max_val)
            d_reward.set_ydata(reward_val)
            d_flow1.set_ydata(flow1_val)
            d_flow2.set_ydata(flow2_val)

            if plot_frame:
                frame.imshow(agent._environment.get_state_image()) # The frame used for
                                                                   training
                frame.set_title('Reward: {}'.format(round(agent._environment._cur_reward,2))
                                                                   )

            # Weird: have to use this instead of .draw()...
            plt.pause(0.0000001)

        # Rescale plot
        if not i%500 and i:
            q_max.set_ylim([np.min(q_max_val),np.max(q_max_val)])
            reward.set_ylim([np.min(reward_val),np.max(reward_val)])

        # Advance one step
        data = list(agent.make_step(learning = learning)) + agent.get_flow_rates()
        summary.append(data)
    else:
        agent.make_step(learning = learning)

    # Show some stats (mainly speed)
    if not i%500 and i:
        fps = round(500./(time.time()-before),2)
        print 'Step {}, {} fps'.format(i, fps)
        before = time.time()

    # Save sometimes
    if i and not i%15000 and learning:
        agent.save_network(params['network']['network_path'])
        print 'Saved network after {} steps'.format(agent._steps)



agent.close()
```

extract_images.py

```
import numpy as np
import matplotlib.pyplot as plt
```

```
f = np.load('scans/laminar7_before.npz')
data = f['arr_0']
f.close()

# Plot some random frames
print 'Loaded all frames'

for j in range(4):
  ind = np.random.randint(0,19,(3))
  frame = data[ind[0], ind[1], ind[2], :, :]
  plt.subplot(2,2,j+1)
  plt.imshow(-1*frame, cmap = plt.get_cmap('Greys'))
  plt.axis('off')

plt.savefig('laminar_examples.png')
```

plot.py

```
import matplotlib.pyplot as plt
import csv
import numpy as np
import scipy.stats as st

viridis = plt.get_cmap('viridis').colors


# Calc random and human benchmark
benchmark_folder  = 'benchmark/laminar/'

with open(benchmark_folder+'human.txt', 'r') as f:
  human = np.array([float(i) for i in f.readlines()])
  human_mean = np.mean(human)
  human_confidence = st.t.interval(0.95, len(human)-1, loc=human_mean, scale=st.sem(human))


with open(benchmark_folder+'random.txt', 'r') as f:
  random = np.array([float(i) for i in f.readlines()])
  random_mean = np.mean(random)
  random_confidence = st.t.interval(0.95, len(random)-1, loc=random_mean, scale=st.sem(
                                          random))


# Result plot
fig = plt.figure()


# DQN plot
fps = 1.5 # Fluctuates, more or less precise
n_anneal = 135000# 75000
frames_per_episode = 250# 350 #250



ax = fig.add_subplot(211)

f_names = ['laminar7_260916.txt', 'laminar8_071016.txt', 'laminar9_101016.txt']
# f_names = ['droplet10_300816.txt', 'droplet12_060916.txt']

# f_names = ['model_free_logs/laminar1_281016.txt', 'model_free_logs/laminar2_281016.txt']
```

```
max_x, max_y = 0, 0
for i, f_name in enumerate(f_names):

  data = []
  with open(f_name, 'rb') as file:
    reader = csv.reader(file)
    for row in reader:
      data.append(float(row[-1]))

  #ax.plot(data)

  # Running Average plot
  N = 35
  avg = np.convolve(data, np.ones((N,))/N, mode='valid')
  ax.plot(avg,lw = 1.25, c=viridis[i*100])

  max_x = max(max_x, len(data))
  max_y = max(max_y, max(avg))

max_y *= 1.05

# Change the tick labels
time_per_episode = frames_per_episode / (fps * 3600) # [h]
final_time = max_x * time_per_episode
x = range(0,max_x, 200)
labels = [int(xx*time_per_episode) for xx in x]
plt.xticks(x, labels)

# Plot end of epsilon decay
ep_anneal = float(n_anneal) / frames_per_episode
plt.plot((ep_anneal, ep_anneal), (0, max_y), 'k', lw= 2, ls = '--')


# Title and axis labels
ax.set_title('Laminar Flow (DQN)')

ax.set_xlabel('Time [h]')
ax.set_ylabel('Reward')
ax.set_ylim([0.0, max_y])
ax.set_xlim([0.0, 1500])

# Benchmark
plt.plot((0, 1500), (human_mean, human_mean), 'k-')
plt.plot((0, 1500), (human_confidence[0], human_confidence[0]), 'k--')
plt.plot((0, 1500), (human_confidence[1], human_confidence[1]), 'k--')

plt.plot((0, 1500), (random_mean, random_mean), 'k-')
plt.plot((0, 1500), (random_confidence[0], random_confidence[0]), 'k--')
plt.plot((0, 1500), (random_confidence[1], random_confidence[1]), 'k--')


# Model Free plot

fps = 1.5 # Fluctuates, more or less precise
n_anneal = 135000# 75000
frames_per_episode = 250# 350 #250

ax = fig.add_subplot(212)
```

```
# f_names = ['laminar7_260916.txt', 'laminar8_071016.txt', 'laminar9_101016.txt']
# f_names = ['droplet10_300816.txt', 'droplet12_060916.txt']

f_names = ['model_free_logs/laminar1_281016.txt', 'model_free_logs/laminar2_281016.txt', '
                                            model_free_logs/laminar3_311016.txt']

max_x, max_y = 0, 0
for i, f_name in enumerate(f_names):
  data = []
  with open(f_name, 'rb') as file:
    reader = csv.reader(file)
    for row in reader:
      data.append(float(row[-1]))

  #ax.plot(data)

  # Running Average plot
  N = 35
  avg = np.convolve(data, np.ones((N,))/N, mode='valid')
  ax.plot(avg,lw = 1.25, c=viridis[i*100])

  max_x = max(max_x, len(data))
  max_y = max(max_y, max(avg))

max_y *= 1.05

# Change the tick labels
time_per_episode = frames_per_episode / (fps * 3600) # [h]
final_time = max_x * time_per_episode
x = range(0,max_x, 200)
labels = [int(xx*time_per_episode) for xx in x]
plt.xticks(x, labels)

# Plot end of epsilon decay
#ep_anneal = float(n_anneal) / frames_per_episode
#plt.plot((ep_anneal, ep_anneal), (0, max_y), 'k', lw= 2, ls = '--')

ax.set_title('Laminar Flow (MFEC)')

ax.set_xlabel('Time [h]')
ax.set_ylabel('Reward')
ax.set_ylim([0.0, max_y])
ax.set_xlim([0.0, 1400])

# Benchmark
plt.plot((0, 1500), (human_mean, human_mean), 'k-')
plt.plot((0, 1500), (human_confidence[0], human_confidence[0]), 'k--')
plt.plot((0, 1500), (human_confidence[1], human_confidence[1]), 'k--')

plt.plot((0, 1500), (random_mean, random_mean), 'k-')
plt.plot((0, 1500), (random_confidence[0], random_confidence[0]), 'k--')
plt.plot((0, 1500), (random_confidence[1], random_confidence[1]), 'k--')

plt.savefig('laminar_benchmark.png', figsize=(20,20), dpi=300)
plt.show()#
```

TRAINING OF NEURAL NETWORK CLASSIFIER (CHAPTER 5)

data_manager.py

```
import numpy as np
from scipy.misc import imread, imresize
from os import listdir, walk, path
from random import shuffle, randrange
from collections import Counter

import matplotlib.pyplot as plt

class DataManager(object):
  def __init__(self, folder):
    self.folder = folder

    if not self.load_from_npz():
      print 'Npz file not found. Loading from folder.'
      self.parse_files()

      # Now save to npz so we don't have to do it again
      print 'Images loaded, saving to npz file'
      self.save_to_npz()
    else:
      print 'Loaded dataset from npz file'

    self.n_train = self.x_train.shape[0]
    self.n_test = self.x_test.shape[0]
    self.n_categories = len(self.categories)
    self.in_shape = self.x_train.shape[1:]

    print '{} train images, {} test images.'.format(self.n_train, self.n_test)

    # Distributions of train and test set
    train_dist = {self.categories[i]: round(100. * n/self.n_train,1) for i, n in Counter(
                                          self.y_train).items()}
    train_dist = ['{}: {}%'.format(n, p) for n,p in train_dist.items()]
    print 'Distribution train: ' + ', '.join(train_dist)

    test_dist = {self.categories[i]: round(100. * n/self.n_test,1) for i, n in Counter(self.
                                          y_test).items()}
    test_dist = ['{}: {}%'.format(n, p) for n,p in test_dist.items()]
    print 'Distribution test: ' + ', '.join(test_dist)


  def parse_files(self):
    # Top level folders => categories
    self.categories = listdir(self.folder)

    # Each category
    x_train, y_train, x_test, y_test = [], [], [], []
    for cat_i, cat in enumerate(self.categories):
      cat_path = path.join(self.folder, cat)

      # Smallest folder = test set, others = training
      f_size = []
      for sub in listdir(cat_path):
        sub_path = path.join(cat_path, sub)
```

```
      f_size.append((len(listdir(sub_path)), sub))

    f_size = sorted(f_size)

    # Load all the images into the correct category
    for sub in listdir(cat_path):
      sub_path = path.join(cat_path, sub)
      for p in [path.join(sub_path, f) for f in listdir(sub_path) if f.endswith('.png')]:
        if sub == f_size[0][1]:
          x_test.append(self._load_img(p))
          y_test.append(cat_i)
        else:
          x_train.append(self._load_img(p))
          y_train.append(cat_i)

  self.x_train = np.array(x_train)
  self.y_train = np.array(y_train)
  self.x_test = np.array(x_test)
  self.y_test = np.array(y_test)


def _load_img(self, path):
  '''
    Squeezes image into square shape 84x84
  '''
  img = imresize(imread(path), (84,84))
  img = img.reshape((84, 84, 1))
  img = img.astype(np.float32) / 255. # Convert to float32!
  return img


def train_stream(self, nb_epoch = 1, batch_size = 32, limit = None):
  if limit is not None:
    print '!!! Train limit is {} per epoch ({}) !!!'.format(limit, nb_epoch)

  # Yield batches
  for epoch in range(nb_epoch):
    inds = range(self.n_train)
    shuffle(inds)

    if limit is not None:
      inds = inds[:limit]

    print 'Epoch {}: {} batches'.format(epoch, len(inds)//batch_size)

    e=True
    for i in range(0, len(inds), batch_size):
      l = inds[i:i + batch_size]
      yield self.x_train[l,:,:], self.y_train[l], e
      e = False


def test_stream(self, nb_epoch = 1, batch_size = 32, limit = None):
  if limit is not None:
    print '!!! Test limit is {} per epoch ({}) !!!'.format(limit, nb_epoch)

  # Yield batches
  for epoch in range(nb_epoch):
    inds = range(self.n_test)
```

```
    shuffle(inds)

    if limit is not None:
      inds = inds[:limit]

    print 'Epoch {}: {} batches'.format(epoch, len(inds)//batch_size)

    e=True
    for i in range(0, len(inds), batch_size):
      l = inds[i:i + batch_size]
      yield self.x_test[l,:,:], self.y_test[l], e
      e = False


def get_random_train(self):
  ind = randrange(self.n_train)
  return self.x_train[ind:ind+1, :, :], self.y_train[ind]


def get_random_test(self):
  ind = randrange(self.n_test)
  return self.x_test[ind:ind+1, :, :], self.y_test[ind]


def save_to_npz(self):
  np.savez(path.join(self.folder, 'dataset.npz'),
    categories = self.categories,
    x_train = self.x_train,
    y_train = self.y_train,
    x_test = self.x_test,
    y_test = self.y_test)


def load_from_npz(self):
  try:
    data = np.load(path.join(self.folder, 'dataset.npz'))
  except IOError:
    return False

  self.categories = list(data['categories'])
  self.x_train = data['x_train']
  self.y_train = data['y_train']
  self.x_test = data['x_test']
  self.y_test = data['y_test']
  return True


def visualize_dataset(self):
  '''
    Show some of the images and their labels
  '''
  stream = self.train_stream(batch_size = 36)
  imgs, labels, __ = stream.next()

  # Some simple statistics
  print 'Min: {}, Max: {}, Mean: {} +- {}'.format(imgs.min(), imgs.max(), imgs.mean(),
                                                  imgs.std())

  f, axarr = plt.subplots(6, 6)
```

```
    f.suptitle('Dataset: ' + self.folder, fontsize=10)

    for i in xrange(36):
      # Display
      axarr[i // 6, i % 6].imshow(imgs[i,:,:,0], vmin=0, vmax=1)
      axarr[i // 6, i % 6].set_title(self.categories[labels[i]], fontsize=8)
      axarr[i // 6, i % 6].axis('off')

    plt.show()


if __name__ == "__main__":
  manager = DataManager('/home/o/Desktop/TestDataset/')

  manager.visualize_dataset()
```

neural_tf.py

```
import tensorflow as tf
import numpy as np
from random import shuffle, randrange
import os

# Save protobuf needs a whole lot of stuff..
from tensorflow.python.tools import freeze_graph
from tensorflow.python.training import saver
from tensorflow.core.protobuf import saver_pb2


# Helpers: variable initialization
def bias_variable(shape):
    init = tf.constant(0.1, shape=shape)
    return tf.Variable(init)


def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


# Helpers: Simple Layers
def conv_layer(in_layer, kernel_size=4, kernel_size_y = None, n_channels=32, stride=1,
                                            activation='', padding = 'SAME'):
    in_size = in_layer.get_shape().as_list()[-1]

    # Weights and bias in correct shape
    if kernel_size_y is None:
        weights = weight_variable([kernel_size, kernel_size, in_size, n_channels])
    else:
        weights = weight_variable([kernel_size, kernel_size_y, in_size, n_channels])

    layer = tf.nn.conv2d(in_layer, weights, strides=[1, stride, stride, 1], padding=padding)
    layer = tf.nn.bias_add(layer, bias_variable(shape=[n_channels]))

    if activation == 'relu':
        layer = tf.nn.relu(layer)

    return layer
```

```
def max_pooling(in_layer, kernel_size = 2, stride = 2, padding = 'SAME'):
    return tf.nn.max_pool(in_layer,
        ksize = [1, kernel_size, kernel_size, 1],
        strides = [1, stride, stride, 1],
        padding = padding)


def flatten(in_layer, name = None):
    s = in_layer.get_shape().as_list()
    return tf.reshape(in_layer, [-1, s[1]*s[2]*s[3]], name = name)


def dense_layer(in_layer, n_neurons=512, activation='', name = ''):
    n_in = in_layer.get_shape().as_list()[1]
    weights = weight_variable([n_in, n_neurons])
    layer = tf.matmul(in_layer, weights)
    if name:
        layer = tf.nn.bias_add(layer, bias_variable(shape=[n_neurons]), name = name)
    else:
        layer = tf.nn.bias_add(layer, bias_variable(shape=[n_neurons]))

    if activation == 'relu':
        if name:
            layer = tf.nn.relu(layer, name = name)
        else:
            layer = tf.nn.relu(layer)

    return layer


class TFModel(object):
    '''
        This class handles general stuff like trainining and prediction.
        Initialize your own network by extending the init function (run super().__init__
                                                        first) .
        Required:   - Use self.in_layer (and potentially self.labels)
                    - Create self.out_layer (name='out_layer') which produces class
                                                                probabilities
    '''

    def __init__(self, input_shape, clip_value = 0.05, learning_rate = 1e-4, learning_decay
                                                = 1.0, n_classes = None, dropout = 0.2):

        # Make sure we have fresh variable names
        tf.reset_default_graph()

        # Setup network
        self.in_layer = tf.placeholder(tf.float32, [None] + list(input_shape), name='
                                                        in_layer')
        self.labels = tf.placeholder(tf.int32, [None])

        # Feed this tensor the keep probability during training
        # (during evaluation this should be 1.0)
        self.keep_prob = tf.placeholder_with_default(1.0, [])
        self.default_keep_prob = 1. - dropout

        self.learning_rate = tf.placeholder(tf.float32, [])
        self.current_learning_rate = learning_rate
```

```
        self.learning_decay = learning_decay

        # Do some image preprocessing
        # Cut of low level noise
        l = tf.clip_by_value(self.in_layer, clip_value, 1.0, name=None)

        # Apply whitening to each image
        self.start_layer = tf.map_fn(tf.image.per_image_standardization, l, back_prop=False)


    def setup_training(self):
        # A prediction layer (argmax on out_layer)
        self.prediction_layer = tf.cast(tf.argmax(self.out_layer, 1), tf.int32, name='
                                                    prediction_layer')

        # Setup training
        losses = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=self.out_layer,
                                                    labels=self.labels)
        self.loss = tf.reduce_mean(losses)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate).minimize(
                                                    self.loss)

        correct_prediction = tf.equal(self.prediction_layer, self.labels)
        self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        # Checkpoint saver
        self.saver = tf.train.Saver(write_version=saver_pb2.SaverDef.V2)

        # And initialize all variables
        self.session = tf.Session()
        self.session.run(tf.global_variables_initializer())


    def fit(self, stream, verbose=1, path = None):
        epoch = 0
        last_path = None
        for i in xrange(int(1e8)):
            try:
                x, y, new_epoch = stream.next()
            except StopIteration:
                if path is not None:
                    last_path = self.save(path + 'model.ckpt', epoch = epoch)
                    print 'Saved model:', last_path
                    return last_path
                else:
                    return None

            # decay learning rate every new epoch
            if new_epoch and i:
                self.current_learning_rate *= self.learning_decay
                epoch += 1

                # Save checkpoint
                if path is not None:
                    last_path = self.save(path + 'model.ckpt', epoch = epoch)
                    print 'Saved model:', last_path

            # Training with 20% dropout if model includes dropout
```

```
            self.session.run(self.optimizer, feed_dict={self.in_layer: x, self.labels: y,
                                                        self.keep_prob: self.
                                                        default_keep_prob,
                self.learning_rate: self.current_learning_rate})

            if not i % 50 and verbose == 1:
                loss, acc = self.session.run([self.loss, self.accuracy],
                                             feed_dict={self.in_layer: x, self.labels: y})
                print 'Batch {}; Loss: {}, Accuracy: {}'.format(i, loss, acc)

        return last_path


    def evaluate(self, stream):
        losses, accuracies = [], []
        for i in range(int(1e8)):
            try:
                x,y, new_epoch = stream.next()
            except StopIteration:
                n = float(len(losses))
                return sum(losses)/n, sum(accuracies)/n

            l, a = self.session.run([self.loss, self.accuracy], feed_dict={self.in_layer: x,
                                                            self.labels: y})
            losses.append(l)
            accuracies.append(a)


    def predict(self, x):
        pred = self.session.run(self.prediction_layer, feed_dict={self.in_layer: x})
        return pred[0]


    def get_output(self, x):
        pred = self.session.run(self.out_layer, feed_dict={self.in_layer: x})
        return pred[0]


    def save(self, path, epoch = 0):
        return self.saver.save(self.session, path, global_step = epoch)


    def load(self, path):
        self.saver.restore(self.session, path)


    def save_protobuf(self, path):
        '''
            Inspired by: freeze_graph_test.py
        '''
        rand_check = str(randrange(1e10,1e11))
        rand_graph = str(randrange(1e10,1e11))

        saver_inst = saver.Saver(write_version=saver_pb2.SaverDef.V2)
        checkpoint_path = saver_inst.save(
              self.session,
              '/tmp/' + rand_check,
              global_step=0,
              latest_filename='checkpoint_state')
```

```
        # checkpoint_path = self.save('/tmp/'+rand_check)
        tf.train.write_graph(self.session.graph, '/tmp', rand_graph + '.pb')

        # Now freeze the graph including the weights
        freeze_graph.freeze_graph(input_graph = '/tmp/{}.pb'.format(rand_graph),
                                  input_saver = '',
                                  input_binary = False,
                                  input_checkpoint = checkpoint_path,
                                  output_node_names = 'prediction_layer',
                                  restore_op_name = "save/restore_all",
                                  filename_tensor_name = "save/Const:0",
                                  output_graph = path,
                                  clear_devices = False,
                                  initializer_nodes = '')

        print 'Saved protobuf:', path


###
#
# Our custom classifiers
#
###

class SimpleConv(TFModel):
    '''
        A simple convolutional classifier:
        3 times convolution then 512 fully connected.
    '''
    def __init__(self, **kwargs):
        super(SimpleConv, self).__init__(**kwargs)

        cur_layer = self.start_layer
        s_conv = [8, 4, 3]
        d_conv = [32, 64, 64]
        strides = [4, 2, 1]
        for i in range(3):
            cur_layer = conv_layer(cur_layer, kernel_size=s_conv[i],
                                   n_channels=d_conv[i], stride=strides[i], activation='relu
                                                                                        ')

        cur_layer = flatten(cur_layer)
        cur_layer = dense_layer(cur_layer, n_neurons=512, activation='relu')
        self.out_layer = dense_layer(cur_layer, n_neurons=kwargs['n_classes'], name ='
                                                        out_layer')

        # Setup training in super
        self.setup_training()


class AdvancedConv(TFModel):
    '''
        Similar to SimpleConv but with maxpooling instead of strides
        & 15 % dropout in the end.
    '''
    def __init__(self, **kwargs):
        super(AdvancedConv, self).__init__(**kwargs)
```

```
        cur_layer = self.start_layer
        s_conv = [8, 4, 3]
        d_conv = [32, 64, 64]
        for i in range(3):
            cur_layer = conv_layer(cur_layer, kernel_size=s_conv[i],
                                   n_channels=d_conv[i], stride=1, activation='relu')
            cur_layer = max_pooling(cur_layer, kernel_size = 2, stride = 2)

        cur_layer = flatten(cur_layer)
        cur_layer = dense_layer(cur_layer, n_neurons=512, activation='relu')

        # Dropout only during training
        cur_layer = tf.nn.dropout(cur_layer, self.keep_prob)

        self.out_layer = dense_layer(cur_layer, n_neurons=kwargs['n_classes'], name ='
                                                        out_layer')

        # Setup training in super
        self.setup_training()


# Squeezenet fire module
def fire_module(in_layer, s1x1, e1x1, e3x3, residual=False):
    fire = {}
    shape = in_layer.get_shape()

    # Squeeze
    fire['s1'] = conv_layer(in_layer, kernel_size = 1, n_channels = s1x1, activation='relu')

    # Expand
    fire['e1'] = conv_layer(fire['s1'], kernel_size = 1, n_channels = e1x1)
    fire['e3'] = conv_layer(fire['s1'], kernel_size = 3, n_channels = e3x3)

    # tf.concat changed order....\
    fire['concat'] = tf.concat([fire['e1'], fire['e3']], 3)

    if residual:
        cur_layer = tf.add(fire['concat'], in_layer)
    else:
        cur_layer = fire['concat']

    return tf.nn.relu(cur_layer)


class SqueezeNet(TFModel):
    '''
        SqueezeNet aims to reduce the number of parameters in a neural network.
        We use the squeezenet 1.1 according to:
        https://github.com/DeepScale/SqueezeNet
    '''
    def __init__(self, **kwargs):
        super(SqueezeNet, self).__init__(**kwargs)

        cur_layer = self.start_layer

        # Conv 1
        cur_layer = conv_layer(cur_layer, kernel_size = 3, n_channels = 64, stride = 1) #
                                                        stride = 2 in original but larger
                                                        images
```

```
        # Pool 1
        cur_layer = tf.nn.max_pool(cur_layer, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                                                    padding='VALID')

        # Fire 2
        cur_layer = fire_module(cur_layer, 16, 64, 64)

        # Fire 3
        cur_layer = fire_module(cur_layer, 16, 64, 64, True)

        # Pool 3
        cur_layer = tf.nn.max_pool(cur_layer, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                                                    padding='VALID')

        # Fire 4
        cur_layer = fire_module(cur_layer, 32, 128, 128)

        # Fire 5
        cur_layer = fire_module(cur_layer, 32, 128, 128, True)

        # Pool 5
        cur_layer = tf.nn.max_pool(cur_layer, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                                                    padding='VALID')

        # Fire 6
        cur_layer = fire_module(cur_layer, 48, 192, 192)

        # Fire 7
        cur_layer = fire_module(cur_layer, 48, 192, 192, True)

        # Fire 8
        cur_layer = fire_module(cur_layer, 64, 256, 256)

        # Fire 9
        cur_layer = fire_module(cur_layer, 64, 256, 256, True)

        # Dropout
        cur_layer = tf.nn.dropout(cur_layer, self.keep_prob) # 50% in original! 20% here

        # Conv 10
        cur_layer = conv_layer(cur_layer, kernel_size = 1, n_channels = kwargs['n_classes'],
                                                     activation = 'relu')

        # Pool 10 (Average pooling)
        cur_layer = tf.nn.avg_pool(cur_layer, ksize=[1, 9, 9, 1], strides=[1, 1, 1, 1],
                                                    padding='VALID')

        # Flatten (our version)
        self.out_layer = flatten(cur_layer, name = 'out_layer')

        # Setup training in super
        self.setup_training()



# The modules for inception resnet v2
def inception_a(cur_layer):
    # branches
```

```
    b1 = conv_layer(cur_layer, kernel_size = 1, n_channels = 32)

    b2 = conv_layer(cur_layer, kernel_size = 1, n_channels = 32)
    b2 = conv_layer(b2, kernel_size = 3, n_channels = 32)

    b3 = conv_layer(cur_layer, kernel_size = 1, n_channels = 32)
    b3 = conv_layer(b3, kernel_size = 3, n_channels = 48)
    b3 = conv_layer(b3, kernel_size = 3, n_channels = 64)

    combo = tf.concat([b1,b2,b3], 3)

    combo = conv_layer(combo, kernel_size = 1, n_channels = cur_layer.get_shape().as_list()[
                                                3])

    # The magic scaling to stabilize training, supposed to be around 0.1
    cur_layer += combo * 0.15

    # And an activation function
    return tf.nn.relu(cur_layer)


def reduction_a(cur_layer):
    b1 = max_pooling(cur_layer, kernel_size = 3, stride = 2, padding = 'VALID')
    b2 = conv_layer(cur_layer, kernel_size = 3, n_channels = 384, stride = 2, padding = '
                                                VALID')

    b3 = conv_layer(cur_layer, kernel_size = 1, n_channels = 256)
    b3 = conv_layer(cur_layer, kernel_size = 3, n_channels = 256)
    b3 = conv_layer(cur_layer, kernel_size = 3, n_channels = 384, stride = 2, padding = '
                                                VALID')

    combo = tf.concat([b1,b2,b3], 3)

    return combo


def inception_b(cur_layer):
    b1 = conv_layer(cur_layer, kernel_size = 1, n_channels = 192)

    b2 = conv_layer(cur_layer, kernel_size = 1, n_channels = 128)
    b2 = conv_layer(b2, kernel_size = 1, kernel_size_y = 7, n_channels = 160)
    b2 = conv_layer(b2, kernel_size = 7, kernel_size_y = 1, n_channels = 192)

    combo = tf.concat([b1,b2], 3)

    combo = conv_layer(combo, kernel_size = 1, n_channels = cur_layer.get_shape().as_list()[
                                                3])

    # Another magic scaling to stabilize training, supposed to be around 0.1
    cur_layer += combo * 0.15

    return tf.nn.relu(cur_layer)


def reduction_b(cur_layer):
    b1 = max_pooling(cur_layer, kernel_size = 3, stride = 2, padding = 'VALID')

    b2 = conv_layer(cur_layer, kernel_size = 1, n_channels = 256)
    b2 = conv_layer(b2, kernel_size = 3, n_channels = 384, stride = 2, padding = 'VALID')
```

```
    b3 = conv_layer(cur_layer, kernel_size = 1, n_channels = 256)
    b3 = conv_layer(b3, kernel_size = 3, n_channels = 288, stride = 2, padding = 'VALID')

    b4 = conv_layer(cur_layer, kernel_size = 1, n_channels = 256)
    b4 = conv_layer(b4, kernel_size = 3, n_channels = 288)
    b4 = conv_layer(b4, kernel_size = 3, n_channels = 320, stride=2, padding = 'VALID')

    combo = tf.concat([b1,b2,b3,b4], 3)

    return combo


def inception_c(cur_layer):
    b1 = conv_layer(cur_layer, kernel_size = 1, n_channels = 192)

    b2 = conv_layer(cur_layer, kernel_size = 1, n_channels = 192)
    b2 = conv_layer(b2, kernel_size = 1, kernel_size_y = 3, n_channels = 224)
    b2 = conv_layer(b2, kernel_size = 3, kernel_size_y = 1, n_channels = 256)

    combo = tf.concat([b1,b2], 3)

    combo = conv_layer(combo, kernel_size = 1, n_channels = cur_layer.get_shape().as_list()[
                                                3])

    cur_layer += combo * 0.15

    return tf.nn.relu(cur_layer)


class InceptionResNet2(TFModel):
    '''
        Inception ResNet-v2, adapted from:
        https://arxiv.org/abs/1602.07261

        We use a variant of the stem from Inception-ResNet-v1 bc we have different input
                                                    dimensions
        and it is easier to adapt. Actually our stem is quite custom but linear like v1.

        General structure:

        Stem
        10 x Inception A
        Reduction A
        20 x Inception B
        Reduction B
        10 x Inception C

        Bottom: Avg, Dropout, Softmax
    '''
    def __init__(self, **kwargs):
        super(InceptionResNet2, self).__init__(**kwargs)

        cur_layer = self.start_layer

        # Stem
        cur_layer = conv_layer(cur_layer, kernel_size = 5, n_channels = 32, padding = 'VALID
                                                    ')
```

```
        cur_layer = conv_layer(cur_layer, kernel_size = 3, n_channels = 64, padding = 'VALID
                                                    ')
        cur_layer = conv_layer(cur_layer, kernel_size = 3, n_channels = 80, padding = 'VALID
                                                    ')
        cur_layer = conv_layer(cur_layer, stride = 2, kernel_size = 3, n_channels = 192,
                                                    padding = 'VALID')
        cur_layer = conv_layer(cur_layer, kernel_size = 3, n_channels = 256, padding = '
                                                    VALID')

        # Inception A
        for i in range(1):
            cur_layer = inception_a(cur_layer)

        # Reduction A
        cur_layer = reduction_a(cur_layer)

        # Inception B
        for i in range(1):
            cur_layer = inception_b(cur_layer)

        # Reduction B
        cur_layer = reduction_b(cur_layer)

        # Inception C
        for i in range(1):
            cur_layer = inception_c(cur_layer)

        # Bottom
        cur_layer = tf.nn.avg_pool(cur_layer, ksize=[1, 8, 8, 1], strides=[1, 1, 1, 1],
                                                    padding='VALID')
        cur_layer = flatten(cur_layer)

        # Dropout only during training
        cur_layer = tf.nn.dropout(cur_layer, self.keep_prob)

        self.out_layer = dense_layer(cur_layer, n_neurons=kwargs['n_classes'], name ='
                                                    out_layer')

        self.setup_training()

if __name__ == "__main__":
    m = SqueezeNet(input_shape = (84,84,1),  n_classes = 3)
    m = InceptionResNet2(input_shape = (84,84,1),  n_classes = 3)
```

prepare_data.py

```
import numpy as np
from os import walk
from scipy.misc import imread, imresize

# Get images
train_data_dir = '/home/g/Desktop/TestSet05.04.17/'
categories = ['H562', 'Jurkat', 'Empty', 'Beads']

frames = {c: [] for c in categories}
for root, __, files in walk(train_data_dir):
    cat = root.split('/')[-2]
    for f in files:
      if f.endswith('.png') and cat in categories:
```

```
            img = imread(root + '/' + f) # 160x200
            img = imresize(img[20:-20,40:-40], (84,84))
            img = img.reshape((84, 84, 1))
            img = img.astype(np.float32) / 255. # Convert to float32!
            frames[cat].append(img)

x = np.concatenate([np.array(frames[c]) for c in categories])

y = np.zeros(len(x), dtype = np.int32)
n = 0
for i, c in enumerate(categories):
  y[n:n+len(frames[c])] = i
  n += len(frames[c])

# Save the dataset
name = 'datasets/complete.npz'
np.savez(name, x=x, y=y)

print 'Saved dataset at:', name
```

train_offline.py

```
import time
import os
from random import randrange
import matplotlib.pyplot as plt
import numpy as np

from data_manager import DataManager
from neural_tf import SimpleConv, AdvancedConv, SqueezeNet, InceptionResNet2


class MultiTrainer(object):
    def __init__(self, data_path, model_path, limit=None):
        self.data = DataManager(data_path)
        self.model_path = model_path
        self.limit = limit

        # Test frame is all 1.0
        self.test_frame = np.ones([1] + list(self.data.in_shape), dtype=np.float32)

        # Initiate all models
        self.models = {
            'simple_conv': SimpleConv(input_shape = self.data.in_shape,
              n_classes = self.data.n_categories,
              clip_value = 0.05,
              learning_rate = 1e-4,
              learning_decay = 1.0),
            'advanced_conv': AdvancedConv(input_shape=self.data.in_shape, n_classes=self.
                                                    data.n_categories),
            # 'squeezenet': SqueezeNet(input_shape = self.data.in_shape, n_classes = self.
                                                    data.n_categories,
            # learning_rate = 0.04, learning_decay = 0.94),
            # 'inception_resnet_v2': InceptionResNet2(input_shape = self.data.in_shape,
                                                    n_classes = self.data.n_categories
                                                    ,
            #   learning_rate = 0.045, learning_decay = 0.94)
        }
```

```
        # Save some model statistics (at least final loss and accuracy)
        self.model_stats = {}

    def train_all_models(self, nb_epoch=1, batch_size=32, verbose=1):
        # We are keeping a training log, some of this is used in cpp later
        legend = ['cat {}'.format(i) for i in range(self.data.n_categories)]
        legend += ['test {}'.format(i) for i in range(self.data.n_categories)]
        legend += ['n_classes', 'name', 'n_train', 'n_test', 'speed_train', 'loss', '
                                                    accuracy', 'path', 'in_shape_x',
                'in_shape_y']

        for name, m in self.models.items():
            # Create model folder if not exists
            model_path = self.model_path + name + '/'
            if not os.path.exists(model_path):
                os.makedirs(model_path)

            print '\nTraining', name
            before = time.time()

            # Train
            train_stream = self.data.train_stream(limit=self.limit, nb_epoch=nb_epoch,
                                                  batch_size=batch_size)
            last_path = m.fit(train_stream, verbose=verbose, path=model_path)
            fps = round(float(self.data.n_train * nb_epoch) / (time.time() - before), 1)

            # Test on complete test set
            print 'Testing', name
            test_stream = self.data.test_stream(limit=self.limit, nb_epoch=1, batch_size=
                                                  batch_size)
            loss, acc = m.evaluate(test_stream)

            self.model_stats[name] = (loss, acc)
            test_out = list(m.get_output(self.test_frame))

            # Assemble the csv
            all_data = self.data.categories + test_out
            all_data += [self.data.n_categories, name, self.data.n_train, self.data.n_test,
                                                  fps, loss, acc, last_path]
            all_data += [self.data.in_shape[0], self.data.in_shape[1]]
            all_data = [str(i) for i in all_data]

            # We use the csv format (first row is legend)
            log = ','.join(legend) + '\n' + ','.join(all_data)
            with open(model_path + 'log.csv', 'w') as f:
                f.write(log)

            print '{} fps (train), loss {}, accuracy {},'.format(fps, loss, acc)

            # Example output values
            print 'Test frame values:', test_out

            # print 'Saving {} as protobuf'.format(name)
            # m.save_protobuf(model_path + '{}.pb'.format(name))

    def plot_best_result(self):
        # Get largest accuracy
        losses = [(i[1], name) for name, i in self.model_stats.items()]
        best_model = max(losses)[1]
```

```
        f, axarr = plt.subplots(6, 6)
        f.suptitle('{}: {}% accuracy'.format(best_model, round(100. * self.model_stats[
                                                    best_model][1], 1)),
                   fontsize=14)

        cat = self.data.categories

        for i in range(36):
            frame, exp = self.data.get_random_test()

            # Predict
            pred = cat[self.models[best_model].predict(frame)]
            exp = cat[exp]

            color = 'g'
            if exp != pred:
                color = 'r'

            # Display
            axarr[i // 6, i % 6].imshow(frame.reshape(frame.shape[1], frame.shape[2]), vmin=
                                                    0, vmax=1)
            axarr[i // 6, i % 6].set_title('{}, Exp: {}'.format(pred, exp), color=color,
                                                    fontsize=8)
            axarr[i // 6, i % 6].axis('off')

        plt.show()


if __name__ == "__main__":
    trainer = MultiTrainer(data_path='/home/o/Desktop/TestDataset/', model_path='models/')
    trainer.train_all_models(nb_epoch=1)
    trainer.plot_best_result()

    # '/home/g/Desktop/Sorting/HL6015umBeads/'
```

convert_model.py

```
from neural_tf import SimpleConv, AdvancedConv, SqueezeNet

folder = 'models/simple_conv/'
# folder = 'models/squeezenet/'

# Open log file
with open(folder + 'log.csv', 'r') as f:
  legend, vals = f.read().split('\n')
  legend = legend.split(',')
  vals = vals.split(',')

  # Get the necessary params from the logfile
  savefile = vals[-3]
  name = vals[legend.index('name')]
  n_classes = len([n for n in legend if n.startswith('cat')])
  in_shape = [vals[legend.index('in_shape_x')], vals[legend.index('in_shape_y')], 1]

in_shape = [int(i) for i in in_shape]

# Create model
if name == 'simple_conv':
```

```
  model = SimpleConv(input_shape = in_shape, n_classes = n_classes)
elif name == 'advanced_conv':
  model = AdvancedConv(input_shape = in_shape, n_classes = n_classes)
elif name == 'squeezenet':
  model = SqueezeNet(input_shape = in_shape, n_classes = n_classes)
else:
  raise NotImplementedError('Can not convert network...')

# Load from checkpoint and save as protobuf
model.load(savefile)
model.save_protobuf(folder + name + '.pb')
```

BIBLIOGRAPHY

1. O. Dressler, X. Casadevall i Solvas, A. J. deMello, Chemical and Biological Dynamics Using Droplet-Based Microfluidics. *Annual Review of Analytical Chemistry* (2017).
2. J. S. Kilby, USP3, 138,743 (1964).
3. P. S. Peercy, The drive to miniaturization. *Nature* **406**, 1023–1026 (2000).
4. E. K. Sackmann, A. L. Fulton, D. J. Beebe, The present and future role of microfluidics in biomedical research. *Nature* **507**, 181–189 (2014).
5. T. M. Squires, S. R. Quake, Microfluidics: Fluid physics at the nanoliter scale. *Reviews of modern physics* **77**, 977 (2005).
6. G. M. Whitesides, The origins and the future of microfluidics. *Nature* **442**, 368–373 (2006).
7. A. J. deMello, Control and detection of chemical reactions in microfluidic systems. *Nature* **442**, 394–402 (2006).
8. S. Krishnadasan, R. Brown, Intelligent routes to the controlled synthesis of nanoparticles. *Lab on a Chip* **7**, 1434–1441 (2007).
9. B. Kintses, L. D. van Vliet, S. R. Devenish, F. Hollfelder, Microfluidic droplets: new integrated workflows for biological experiments. *Current opinion in chemical biology* **14**, 548–555 (2010).
10. M. T. Guo, A. Rotem, J. A. Heyman, D. A. Weitz, Droplet microfluidics for high-throughput biological assays. *Lab on a Chip* **12**, 2146–2155 (2012).
11. D. J. Collins, A. Neild, A.-Q. Liu, Y. Ai, The Poisson distribution and beyond: methods for microfluidic droplet production and single cell encapsulation. *Lab on a Chip* **15**, 3439–3459 (2015).
12. T. Thorsen, R. W. Roberts, F. H. Arnold, S. R. Quake, Dynamic pattern formation in a vesicle-generating microfluidic device. *Physical review letters* **86**, 4163 (2001).
13. S. L. Anna, N. Bontoux, H. A. Stone, Formation of dispersions using "flow focusing" in microchannels. *Applied physics letters* **82**, 364–366 (2003).

14. P. Umbanhowar, V. Prasad, D. Weitz, Monodisperse emulsion generation via drop break off in a coflowing stream. *Langmuir* **16**, 347–351 (2000).

15. Y. Ding, X. C. i Solvas, A. J. deMello, "V-junction": a novel structure for high-speed generation of bespoke droplet flows. *Analyst* **140**, 414–421 (2015).

16. J. Hong, M. Choi, J. B. Edel, A. J. deMello, Passive self-synchronized two-droplet generation. *Lab on a Chip* **10**, 2702–2709 (2010).

17. S. Sugiura, M. Nakajima, S. Iwamoto, M. Seki, Interfacial tension driven monodispersed droplet formation from microfabricated channel array. *Langmuir* **17**, 5562–5566 (2001).

18. R. Dangla, S. C. Kayi, C. N. Baroud, Droplet microfluidics driven by gradients of confinement. *Proceedings of the National Academy of Sciences* **110**, 853–858 (2013).

19. J.-u. Shim, R. T. Ranasinghe, C. A. Smith, S. M. Ibrahim, F. Hollfelder, W. T. Huck, D. Klenerman, C. Abell, Ultrarapid generation of femtoliter microfluidic droplets for single-molecule-counting immunoassays. *Acs Nano* **7**, 5955–5964 (2013).

20. N. Mittal, C. Cohen, J. Bibette, N. Bremond, Dynamics of step-emulsification: From a single to a collection of emulsion droplet generators. *Physics of Fluids* **26**, 082109 (2014).

21. D. Hess, A. Rane, A. J. deMello, S. Stavrakis, High-throughput, quantitative enzyme kinetic analysis in microdroplets using stroboscopic epifluorescence imaging. *Analytical chemistry* **87**, 4965–4972 (2015).

22. B. Zheng, R. F. Ismagilov, A microfluidic approach for screening submicroliter volumes against multiple reagents by using preformed arrays of nanoliter plugs in a three-phase liquid/liquid/gas flow. *Angewandte Chemie International Edition* **44**, 2520–2523 (2005).

23. C. Holtze, A. Rowat, J. Agresti, J. Hutchison, F. Angile, C. Schmitz, S. Köster, H. Duan, K. Humphry, R. Scanga, *et al.*, Biocompatible surfactants for water-in-fluorocarbon emulsions. *Lab on a Chip* **8**, 1632–1639 (2008).

24. J.-C. Baret, Surfactants in droplet-based microfluidics. *Lab on a Chip* **12**, 422–433 (2012).

25. L. Shui, A. van den Berg, J. C. Eijkel, Interfacial tension controlled W/O and O/W 2-phase flows in microchannel. *Lab on a Chip* **9**, 795–801 (2009).

26. I. Giaever, C. R. Keese, Behavior of cells at fluid interfaces. *Proceedings of the National Academy of Sciences* **80**, 219–222 (1983).

27. O. Wagner, J. Thiele, M. Weinhart, L. Mazutis, D. A. Weitz, W. T. Huck, R. Haag, Biocompatible fluorinated polyglycerols for droplet microfluidics as an alternative to PEG-based copolymer surfactants. *Lab on a Chip* **16**, 65–69 (2016).

28. P. Gruner, B. Riechers, L. A. C. Orellana, Q. Brosseau, F. Maes, T. Beneyton, D. Pekin, J.-C. Baret, Stabilisers for water-in-fluorinated-oil dispersions: Key properties for microfluidic applications. *Current Opinion in Colloid & Interface Science* **20**, 183–191 (2015).

29. P. Gruner, B. Riechers, B. Semin, J. Lim, A. Johnston, K. Short, J.-C. Baret, Controlling molecular transport in minimal emulsions. *Nature communications* **7**, 10392 (2016).

30. M. Pan, L. Rosenfeld, M. Kim, M. Xu, E. Lin, R. Derda, S. K. Tang, Fluorinated pickering emulsions impede interfacial transport and form rigid interface for the growth of anchorage-dependent cells. *ACS applied materials & interfaces* **6**, 21446–21453 (2014).

31. V. Labrot, M. Schindler, P. Guillot, A. Colin, M. Joanicot, Extracting the hydrodynamic resistance of droplets from their behavior in microchannel networks. *Biomicrofluidics* **3**, 012804 (2009).

32. D. Link, S. L. Anna, D. Weitz, H. Stone, Geometrically mediated breakup of drops in microfluidic devices. *Physical review letters* **92**, 054503 (2004).

33. C. Priest, S. Herminghaus, R. Seemann, Controlled electrocoalescence in microfluidics: Targeting a single lamella. *Applied Physics Letters* **89**, 134101 (2006).

34. N. Bremond, A. R. Thiam, J. Bibette, Decompressing emulsion droplets favors coalescence. *Physical review letters* **100**, 024501 (2008).

35. X. Niu, S. Gulati, J. B. Edel, A. J. deMello, Pillar-induced droplet merging in microfluidic circuits. *Lab on a chip* **8**, 1837–1841 (2008).

36. C. N. Baroud, M. R. de Saint Vincent, J.-P. Delville, An optical toolbox for total control of droplet microfluidics. *Lab on a Chip* **7**, 1029–1033 (2007).

37. K. Ahn, J. Agresti, H. Chong, M. Marquez, D. Weitz, Electrocoalescence of drops synchronized by size-dependent flow in microfluidic channels. *Applied Physics Letters* **88**, 264105 (2006).

38. M. Prakash, N. Gershenfeld, Microfluidic bubble logic. *Science* **315**, 832–835 (2007).

39. O. Dressler, T. Yang, S.-I. Chang, J. Choo, R. C. Wootton, A. J. deMello, Continuous and low error-rate passive synchronization of pre-formed droplets. *Rsc Advances* **5**, 48399–48405 (2015).

40. X. Niu, F. Gielen, J. B. Edel, A. J. deMello, A microdroplet dilutor for high-throughput screening. *Nature chemistry* **3**, 437–442 (2011).

41. A. R. Abate, T. Hung, P. Mary, J. J. Agresti, D. A. Weitz, High-throughput injection with microfluidics using picoinjectors. *Proceedings of the National Academy of Sciences* **107**, 19163–19166 (2010).

42. X. Niu, F. Gielen, A. J. deMello, J. B. Edel, Electro-coalescence of digitally controlled droplets. *Analytical chemistry* **81**, 7321–7325 (2009).

43. Y.-C. Tan, Y. L. Ho, A. P. Lee, Microfluidic sorting of droplets by size. *Microfluidics and Nanofluidics* **4**, 343 (2008).

44. A. C. Hatch, A. Patel, N. R. Beer, A. P. Lee, Passive droplet sorting using viscoelastic flow focusing. *Lab on a Chip* **13**, 1308–1315 (2013).

45. J.-C. Baret, O. J. Miller, V. Taly, M. Ryckelynck, A. El-Harrak, L. Frenz, C. Rick, M. L. Samuels, J. B. Hutchison, J. J. Agresti, *et al.*, Fluorescence-activated droplet sorting (FADS): efficient microfluidic cell sorting based on enzymatic activity. *Lab on a Chip* **9**, 1850–1858 (2009).

46. E. Zang, S. Brandes, M. Tovar, K. Martin, F. Mech, P. Horbert, T. Henkel, M. T. Figge, M. Roth, Real-time image processing for label-free enrichment of Actinobacteria cultivated in picolitre droplets. *Lab on a Chip* **13**, 3707–3713 (2013).

47. A. R. Abate, J. J. Agresti, D. A. Weitz, Microfluidic sorting with high-speed single-layer membrane valves. *Applied Physics Letters* **96**, 203509 (2010).

48. L. Schmid, D. A. Weitz, T. Franke, Sorting drops and cells with acoustics: acoustic microfluidic fluorescence-activated cell sorter. *Lab on a Chip* **14**, 3710–3718 (2014).

49. A. Sciambi, A. R. Abate, Accurate microfluidic sorting of droplets at 30 kHz. *Lab on a Chip* **15**, 47–51 (2015).

50. X. Casadevall i Solvas, X. Niu, K. Leeper, S. Cho, S.-I. Chang, J. B. Edel, A. J. deMello, Fluorescence detection methods for microfluidic droplet platforms. *Journal of visualized experiments: JoVE* (2011).

51. M. R. Bringer, C. J. Gerdts, H. Song, J. D. Tice, R. F. Ismagilov, Microfluidic systems for chemical kinetics that rely on chaotic mixing in droplets. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **362**, 1087–1104 (2004).

52. R. H. Cole, N. de Lange, Z. J. Gartner, A. R. Abate, Compact and modular multicolour fluorescence detector for droplet microfluidics. *Lab on a Chip* **15**, 2754–2758 (2015).

53. H. Edelhoch, L. Brand, M. Wilchek, Fluorescence studies with tryptophyl peptides. *Biochemistry* **6**, 547–559 (1967).

54. I. Bugiel, K. König, H. Wabnitz, Investigation of cells by fluorescence laser scanning microscopy with subnanosecond time resolution. *Lasers Life Sci* **3**, 47–53 (1989).

55. M. Srisa-Art, E. C. Dyson, A. J. deMello, J. B. Edel, Monitoring of real-time streptavidin- biotin binding kinetics using droplet microfluidics. *Analytical chemistry* **80**, 7063–7067 (2008).

56. X. Casadevall i Solvas, M. Srisa-Art, A. J. deMello, J. B. Edel, Mapping of fluidic mixing in microdroplets with 1 μs time resolution using fluorescence lifetime imaging. *Analytical chemistry* **82**, 3950–3956 (2010).

57. C. Benz, H. Retzbach, S. Nagl, D. Belder, Protein–protein interaction analysis in single microfluidic droplets using FRET and fluorescence lifetime detection. *Lab on a Chip* **13**, 2808–2814 (2013).

58. R. R. Pompano, W. Liu, W. Du, R. F. Ismagilov, Microfluidics using spatially defined arrays of droplets in one, two, and three dimensions. *Annual Review of Analytical Chemistry* **4**, 59–81 (2011).

59. W. Shi, J. Qin, N. Ye, B. Lin, Droplet-based microfluidic system for individual Caenorhabditis elegans assay. *Lab on a Chip* **8**, 1432–1435 (2008).

60. F. Lan, J. R. Haliburton, A. Yuan, A. R. Abate, Droplet barcoding for massively parallel single-molecule deep sequencing. *Nature communications* **7** (2016).

61. A. Huebner, D. Bratton, G. Whyte, M. Yang, C. Abell, F. Hollfelder, *et al.*, Static microdroplet arrays: a microfluidic device for droplet trapping, incubation and release for enzymatic and cell-based assays. *Lab on a Chip* **9**, 692–698 (2009).

62. W. Wang, C. Yang, C. M. Li, On-demand microfluidic droplet trapping and fusion for on-chip static droplet assays. *Lab on a Chip* **9**, 1504–1506 (2009).

63. R. E. Gerver, R. Gómez-Sjöberg, B. C. Baxter, K. S. Thorn, P. M. Fordyce, C. A. Diaz-Botia, B. A. Helms, J. L. DeRisi, Programmable microfluidic synthesis of spectrally encoded microspheres. *Lab on a Chip* **12**, 4716–4723 (2012).

64. J. Lim, O. Caen, J. Vrignon, M. Konrad, V. Taly, J.-C. Baret, Parallelized ultra-high throughput microfluidic emulsifier for multiplex kinetic assays. *Biomicrofluidics* **9**, 034101 (2015).

65. E. Z. Macosko, A. Basu, R. Satija, J. Nemesh, K. Shekhar, M. Goldman, I. Tirosh, A. R. Bialas, N. Kamitaki, E. M. Martersteck, *et al.*, Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell* **161**, 1202–1214 (2015).

66. A. M. Klein, L. Mazutis, I. Akartuna, N. Tallapragada, A. Veres, V. Li, L. Peshkin, D. A. Weitz, M. W. Kirschner, Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell* **161**, 1187–1201 (2015).

67. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review* **65**, 386 (1958).

68. D. E. Rumelhart, Learning internal representations by back-propagating errors. *Parallel distributed processing: Explorations in the microstructure of cognition*, 318–362 (1986).

69. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, presented at the IEEE Conference on Computer Vision and Pattern Recognition (2016), 2818.

70. J. Sotelo, S. Mehri, K. Kumar, J. F. Santos, K. Kastner, A. Courville, Y. Bengio, Char2Wav: End-to-end speech synthesis. *self-published* (2017).

71. I. Sutskever, O. Vinyals, Q. V. Le, presented at the Advances in neural information processing systems (2014), 3104.

72. K.-S. Oh, K. Jung, GPU implementation of neural networks. *Pattern Recognition* **37**, 1311–1314 (2004).

73. T. D. Sanger, Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks* **2**, 459–473 (1989).

74. R. S. Sutton, A. G. Barto, *Reinforcement learning: An introduction* (MIT press Cambridge, 1998), vol. 1.

75. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).

76. Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

77. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

78. S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 0278364917710318 (2016).

79. T. Wei, Y. Wang, Q. Zhu, presented at the 54th Design Automation Conference (2017), 1.

80. P. Garstecki, I. Gitlin, W. DiLuzio, G. M. Whitesides, E. Kumacheva, H. A. Stone, Formation of monodisperse bubbles in a microfluidic flow-focusing device. *Applied Physics Letters* **85**, 2649–2651 (2004).

81. M. Hashimoto, P. Garstecki, H. A. Stone, G. M. Whitesides, Interfacial instabilities in a microfluidic Hele-Shaw cell. *Soft Matter* **4**, 1403–1413 (2008).

82. Y.-C. Tan, J. S. Fisher, A. I. Lee, V. Cristini, A. P. Lee, Design of microfluidic channel geometries for the control of droplet volume, chemical concentration, and sorting. *Lab on a Chip* **4**, 292–298 (2004).

83. A. B. Theberge, F. Courtois, Y. Schaerli, M. Fischlechner, C. Abell, F. Hollfelder, W. T. Huck, Microdroplets in microfluidics: an evolving platform for discoveries in chemistry and biology. *Angewandte Chemie International Edition* **49**, 5846–5868 (2010).

84. X. Casadevall i Solvas, A. J. deMello, Droplet microfluidics: recent developments and future applications. *Chemical Communications* **47**, 1936–1942 (2011).

85. R. C. Wootton, A. J. deMello, Microfluidics: analog-to-digital drug screening. *Nature* **483**, 43–44 (2012).

86. D. N. Adamson, D. Mustafi, J. X. Zhang, B. Zheng, R. F. Ismagilov, Production of arrays of chemically distinct nanolitre plugs via repeated splitting in microfluidic devices. *Lab on a Chip* **6**, 1178–1186 (2006).

87. G. Christopher, J. Bergstein, N. End, M. Poon, C. Nguyen, S. L. Anna, Coalescence and splitting of confined droplets at microfluidic junctions. *Lab on a Chip* **9**, 1102–1109 (2009).

88. P. M. Korczyk, L. Derzsi, S. Jakieła, P. Garstecki, Microfluidic traps for hard-wired operations on droplets. *Lab on a Chip* **13**, 4096–4102 (2013).

89. L. Frenz, K. Blank, E. Brouzes, A. D. Griffiths, Reliable microfluidic on-chip incubation of droplets in delay-lines. *Lab on a Chip* **9**, 1344–1348 (2009).

90. L. Mazutis, J.-C. Baret, A. D. Griffiths, A fast and efficient microfluidic system for highly selective one-to-one droplet fusion. *Lab on a Chip* **9**, 2665–2672 (2009).

91. V. Trivedi, A. Doshi, G. Kurup, E. Ereifej, P. Vandevord, A. S. Basu, A modular approach for the generation, storage, mixing, and detection of droplet libraries for high throughput screening. *Lab on a Chip* **10**, 2433–2442 (2010).

92. J. Clausell-Tormos, D. Lieber, J.-C. Baret, A. El-Harrak, O. J. Miller, L. Frenz, J. Blouwolff, K. J. Humphry, S. Köster, H. Duan, *et al.*, Droplet-based microfluidic platforms for the encapsulation and screening of mammalian cells and multicellular organisms. *Chemistry & biology* **15**, 427–437 (2008).

93. C. Lee, J. Lee, H. H. Kim, S.-Y. Teh, A. Lee, I.-Y. Chung, J. Y. Park, K. K. Shung, Microfluidic droplet sorting with a high frequency ultrasound beam. *Lab on a Chip* **12**, 2736–2742 (2012).

94. Z. Cao, F. Chen, N. Bao, H. He, P. Xu, S. Jana, S. Jung, H. Lian, C. Lu, Droplet sorting based on the number of encapsulated particles using a solenoid valve. *Lab on a Chip* **13**, 171–178 (2013).

95. X. Mao, T. J. Huang, Exploiting mechanical biomarkers in microfluidics. *Lab on a Chip* **12**, 4006–4009 (2012).

96. M. Zagnoni, G. Le Lain, J. M. Cooper, Electrocoalescence mechanisms of microdroplets using localized electric fields in microfluidic channels. *Langmuir* **26**, 14443–14449 (2010).

97. E. Um, J.-K. Park, A microfluidic abacus channel for controlling the addition of droplets. *Lab on a Chip* **9**, 207–212 (2009).

98. L. M. Fidalgo, C. Abell, W. T. Huck, Surface-induced droplet fusion in microfluidic devices. *Lab on a Chip* **7**, 984–986 (2007).

99. T. Nisisako, S. A. Portonovo, J. J. Schmidt, Microfluidic passive permeability assay using nanoliter droplet interface lipid bilayers. *Analyst* **138**, 6793–6800 (2013).

100. M. Zagnoni, J. M. Cooper, A microdroplet-based shift register. *Lab on a chip* **10**, 3069–3073 (2010).

101. R. M. Schoeman, E. W. Kemna, F. Wolbers, A. Berg, High-throughput deterministic single-cell encapsulation and droplet pairing, fusion, and shrinkage in a single microfluidic device. *Electrophoresis* **35**, 385–392 (2014).

102. T. Rob, D. J. Wilson, A versatile microfluidic chip for millisecond time-scale kinetic studies by electrospray mass spectrometry. *Journal of the American Society for Mass Spectrometry* **20**, 124–130 (2009).

103. L. Baraban, F. Bertholle, M. L. Salverda, N. Bremond, P. Panizza, J. Baudry, J. A. G. de Visser, J. Bibette, Millifluidic droplet analyser for microbiology. *Lab on a Chip* **11**, 4057–4062 (2011).

104. H. Zec, T. D. Rane, T.-H. Wang, Microfluidic platform for on-demand generation of spatially indexed combinatorial droplets. *Lab on a Chip* **12**, 3055–3062 (2012).

105. A. C. Hatch, J. S. Fisher, A. R. Tovar, A. T. Hsieh, R. Lin, S. L. Pentoney, D. L. Yang, A. P. Lee, 1-Million droplet array with wide-field fluorescence imaging for digital PCR. *Lab on a chip* **11**, 3838–3845 (2011).

106. K. Churski, T. S. Kaminski, S. Jakiela, W. Kamysz, W. Baranska-Rybak, D. B. Weibel, P. Garstecki, Rapid screening of antibiotic toxicity in an automated microdroplet system. *Lab on a Chip* **12**, 1629–1637 (2012).

107. K. Ahn, J. Agresti, H. Chong, M. Marquez, D. Weitz, Electrocoalescence of drops synchronized by size-dependent flow in microfluidic channels. *Applied Physics Letters* **88**, 264105 (2006).

108. J. Hong, M. Choi, J. B. Edel, A. J. deMello, Passive self-synchronized two-droplet generation. *Lab on a Chip* **10**, 2702–2709 (2010).

109. L. Xu, H. Lee, R. Panchapakesan, K. W. Oh, Fusion and sorting of two parallel trains of droplets using a railroad-like channel network and guiding tracks. *Lab on a Chip* **12**, 3936–3942 (2012).

110. B. Ahn, K. Lee, H. Lee, R. Panchapakesan, K. W. Oh, Parallel synchronization of two trains of droplets using a railroad-like channel network. *Lab on a Chip* **11**, 3956–3962 (2011).

111. B. Zheng, J. D. Tice, R. F. Ismagilov, Formation of droplets of alternating composition in microfluidic channels and applications to indexing of concentrations in droplet-based assays. *Analytical chemistry* **76**, 4977–4982 (2004).

112. L.-H. Hung, K. M. Choi, W.-Y. Tseng, Y.-C. Tan, K. J. Shea, A. P. Lee, Alternating droplet generation and controlled dynamic droplet fusion in microfluidic device for CdS nanoparticle synthesis. *Lab on a Chip* **6**, 174–178 (2006).

113. E. Um, M. E. Rogers, H. A. Stone, Combinatorial generation of droplets by controlled assembly and coalescence. *Lab on a Chip* **13**, 4674–4680 (2013).

114. L. Frenz, J. Blouwolff, A. D. Griffiths, J.-C. Baret, Microfluidic production of droplet pairs. *Langmuir* **24**, 12073–12076 (2008).

115. V. Chokkalingam, S. Herminghaus, R. Seemann, Self-synchronizing pairwise production of monodisperse droplets by microfluidic step emulsification. *Applied Physics Letters* **93**, 254101 (2008).

116. R. M. Lorenz, G. S. Fiorini, G. D. Jeffries, D. S. Lim, M. He, D. T. Chiu, Simultaneous generation of multiple aqueous droplets in a microfluidic device. *Analytica chimica acta* **630**, 124–130 (2008).

117. M. Prakash, N. Gershenfeld, presented at the 11th International Conference on Miniaturized Systems for Chemistry and Life Sciences (2007).

118. E. Surenjav, S. Herminghaus, C. Priest, R. Seemann, Discrete microfluidics: Reorganizing droplet arrays at a bend. *Applied Physics Letters* **95**, 154104 (2009).

119. D. C. Duffy, J. C. McDonald, O. J. Schueller, G. M. Whitesides, Rapid prototyping of microfluidic systems in poly (dimethylsiloxane). *Analytical chemistry* **70**, 4974–4984 (1998).

120. Z. Michalak, D. Fartash, N. Haque, S. Lee, Tunable crystallization via osmosis-driven transport across a droplet interface bilayer. *CrystEngComm* **14**, 7865–7868 (2012).

121. H. N. Joensson, M. Uhlén, H. A. Svahn, Droplet size based separation by deterministic lateral displacement—separating droplets by cell-induced shrinking. *Lab on a Chip* **11**, 1305–1310 (2011).

122. M. J. Doughty, pH dependent spectral properties of sodium fluorescein ophthalmic solutions revisited. *Ophthalmic and Physiological Optics* **30**, 167–174 (2010).

123. E. Walsh, A. Feuerborn, P. R. Cook, Formation of droplet interface bilayers in a Teflon tube. *Scientific reports* **6** (2016).

124. J. D. Tice, H. Song, A. D. Lyon, R. F. Ismagilov, Formation of droplets and mixing in multiphase microfluidics at low values of the Reynolds and the capillary numbers. *Langmuir* **19**, 9127–9133 (2003).

125. M. B. Elowitz, A. J. Levine, E. D. Siggia, P. S. Swain, Stochastic gene expression in a single cell. *Science* **297**, 1183–1186 (2002).

126. S. L. Sjostrom, H. N. Joensson, H. A. Svahn, Multiplex analysis of enzyme kinetics and inhibition by droplet microfluidics using picoinjectors. *Lab on a Chip* **13**, 1754–1761 (2013).

127. D. C. Pregibon, M. Toner, P. S. Doyle, Multifunctional encoded particles for high-throughput biomolecule analysis. *Science* **315**, 1393–1396 (2007).

128. K. W. Bong, J. Lee, P. S. Doyle, Stop flow lithography in perfluoropolyether (PFPE) microfluidic channels. *Lab on a Chip* **14**, 4680–4687 (2014).

129. K. Braeckmans, S. C. De Smedt, C. Roelant, M. Leblans, R. Pauwels, J. Demeester, Encoding microcarriers by spatial selective photobleaching. *Nature materials* **2**, 169 (2003).

130. H. Q. Nguyen, B. C. Baxter, K. Brower, C. A. Diaz-Botia, J. L. DeRisi, P. M. Fordyce, K. S. Thorn, Programmable microfluidic synthesis of over one thousand uniquely identifiable spectral codes. *Advanced optical materials* **5** (2017).

131. C. Neils, Z. Tyree, B. Finlayson, A. Folch, Combinatorial mixing of microfluidic streams. *Lab on a Chip* **4**, 342–350 (2004).

132. Y. Zhao, Z. Xie, H. Gu, L. Jin, X. Zhao, B. Wang, Z. Gu, Multifunctional photonic crystal barcodes from microfluidics. *NPG Asia Materials* **4**, e25 (2012).

133. D. Dendukuri, S. S. Gu, D. C. Pregibon, T. A. Hatton, P. S. Doyle, Stop-flow lithography in a microfluidic device. *Lab on a Chip* **7**, 818–828 (2007).

134. A. Zinchenko, S. R. Devenish, B. Kintses, P.-Y. Colin, M. Fischlechner, F. Hollfelder, One in a million: flow cytometric sorting of single cell-lysate assays in monodisperse picolitre double emulsion droplets for directed evolution. *Analytical chemistry* **86**, 2526–2533 (2014).

135. A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, A. Mohsen, An architecture for electrically configurable gate arrays. *IEEE Journal of Solid-State Circuits* **24**, 394–398 (1989).

136. C. A. Schneider, W. S. Rasband, K. W. Eliceiri, NIH Image to ImageJ: 25 years of image analysis. *Nature methods* **9**, 671 (2012).

137. I. Lignos, L. Protesescu, S. Stavrakis, L. Piveteau, M. J. Speirs, M. A. Loi, M. V. Kovalenko, A. J. deMello, Facile droplet-based microfluidic synthesis of monodisperse IV–VI semiconductor nanocrystals with coupled in-line NIR fluorescence detection. *Chemistry of Materials* **26**, 2975–2982 (2014).

138. M. J. Wildey, A. Haunso, M. Tudor, M. Webb, J. H. Connick, High-Throughput Screening. *Annual Reports in Medicinal Chemistry*, 149–195 (2017).

139. J. R. Haliburton, W. Shao, A. Deutschbauer, A. Arkin, A. R. Abate, Genetic interaction mapping with microfluidic-based single cell sequencing. *PLOS ONE* **12**, ed. by Q. Zou, e0171302 (2017).

140. G. Du, Q. Fang, J. M. J. den Toonder, Microfluidics for cell-based high throughput screening platforms—A review. *Analytica Chimica Acta* **903**, 36–50 (2016).

141. L. Mazutis, J. Gilbert, W. L. Ung, D. A. Weitz, A. D. Griffiths, J. A. Heyman, Single-cell analysis and sorting using droplet-based microfluidics. *Nature Protocols* **8**, 870–891 (2013).

142. N. Minorsky, Directional stability of automatically steered bodies. *Naval Engineers Journal* **32** (1922).

143. E. T. Lagally, P. C. Simpson, R. A. Mathies, Monolithic integrated microfluidic DNA amplification and capillary electrophoresis analysis system. *Sensors and Actuators B: Chemical* **63**, 138–146 (2000).

144. C.-J. Kim, J. Gong, US Patent 9,266,076 (2016).

145. D. E. Goldberg, J. H. Holland, Genetic algorithms and machine learning. *Machine learning* **3**, 95–99 (1988).

146. D. Steinkraus, I. Buck, P. Simard, presented at the Eighth International Conference on Document Analysis and Recognition (2005), 1115.

147. I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, *Data Mining: Practical machine learning tools and techniques* (Morgan Kaufmann, 2016).

148. C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, presented at the AAAI (2017), 4278.

149. Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

150. M. I. Jordan, T. M. Mitchell, Machine learning: Trends, perspectives, and prospects. *Science* **349**, 255–260 (2015).

151. X. Zhu, A. B. Goldberg, Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning* **3**, 1–130 (2009).

152. L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285 (1996).

153. H. Lee, A. Battle, R. Raina, A. Y. Ng, presented at the Advances in neural information processing systems (2007), 801.

154. W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**, 115–133 (1943).

155. V. Nair, G. E. Hinton, presented at the 27th international conference on machine learning (2010), 807.

156. P. J. Werbos, Doctoral Dissertation, Applied Mathematics, Harvard University, MA, 1974.

157. D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, J. Schmidhuber, presented at the IJCAI Proceedings-International Joint Conference on Artificial Intelligence (2011), vol. 22, 1237.

158. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, presented at the IEEE conference on computer vision and pattern recognition (2015), 1.

159. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, presented at the 44th Annual International Symposium on Computer Architecture (2017), 1.

160. Y. LeCun, Y. Bengio, *et al.*, Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* **3361** (1995).

161. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

162. Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, presented at the IEEE International Conference on Robotics and Automation (2017), 3357.

163. M. Littman, J. Boyan, presented at the International Workshop on Applications of Neural Networks to Telecommunications (2013).

164. C. J. Watkins, P. Dayan, Q-learning. *Machine learning* **8**, 279–292 (1992).

165. C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, D. Hassabis, Model-free episodic control. *arXiv preprint arXiv:1606.04460* (2016).

166. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, presented at the International Conference on Machine Learning (2016), 1928.

167. R. J. Sutherland, J. W. Rudy, Configural association theory: The role of the hippocampal formation in learning, memory, and amnesia. *Psychobiology* **17**, 129–144 (1989).

168. P. J. Clark, F. C. Evans, Distance to nearest neighbor as a measure of spatial relationships in populations. *Ecology* **35**, 445–453 (1954).

169. M. Kempka, M. Wydmuch, G. Runc, J. Toczek, W. Jaśkowski, presented at the Computational Intelligence and Games (2016), 1.

170. ZDOOM (2004; `http://zdoom.org`).

171. N. Otsu, A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics* **9**, 62–66 (1979).

172. R. O. Duda, P. E. Hart, Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM* **15**, 11–15 (1972).

173. G. Bradski, The OpenCV Library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer* **25**, 120–123 (2000).

174. F. Chollet, *Keras*, self-published (2015).

175. J. Bergstra, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, I. Goodfellow, A. Bergeron, Y. Bengio, P. Kaelbling, Theano: Deep learning on gpus with python. *self-published* (2011).

176. M. Bawa, T. Condie, P. Ganesan, presented at the 14th international conference on World Wide Web (2005), 651.

177. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

178. S. Matosevic, B. M. Paegel, Stepwise synthesis of giant unilamellar vesicles on a microfluidic assembly line. *Journal of the American Chemical Society* **133**, 2798–2800 (2011).

179. M. P. Carreras, S. Wang, A multifunctional microfluidic platform for generation, trapping and release of droplets in a double laminar flow. *Journal of Biotechnology* **251**, 106–111 (2017).

180. H. Van Hasselt, A. Guez, D. Silver, presented at the AAAI (2016), 2094.

181. I. Lignos, S. Stavrakis, G. Nedelcu, L. Protesescu, A. J. deMello, M. V. Kovalenko, Synthesis of cesium lead halide perovskite nanocrystals in a droplet-based microfluidic platform: fast parametric space mapping. *Nano letters* **16**, 1869–1877 (2016).

182. J. Xu, G. Luo, S. Li, G. Chen, Shear force induced monodisperse droplet formation in a microfluidic device by controlling wetting properties. *Lab on a Chip* **6**, 131–136 (2006).

183. J. Zhou, A. V. Ellis, N. H. Voelcker, Recent developments in PDMS surface modification for microfluidic devices. *Electrophoresis* **31**, 2–16 (2010).

184. A. Adan, G. Alizada, Y. Kiraz, Y. Baran, A. Nalbant, Flow cytometry: basic principles and applications. *Critical reviews in biotechnology* **37**, 163–176 (2017).

185. O. D. Laerum, T. Farsund, Clinical application of flow cytometry: a review. *Cytometry Part A* **2**, 1–13 (1981).

186. M. Verso, The evolution of blood-counting techniques. *Medical history* **8**, 149 (1964).

187. Y. Han, Y. Gu, A. C. Zhang, Y.-H. Lo, imaging technologies for flow cytometry. *Lab on a Chip* **16**, 4639–4647 (2016).

188. M. A. Van Dilla, T. Truiullo, P. F. Mullaney, J. Coultex, Cell microfluorometry: a method for rapid fluorescence measurement. *Science* **163**, 1213–1214 (1969).

189. J. Steinkamp, M. Fulwyler, J. Coulter, R. Hiebert, J. Horney, P. Mullaney, A new multiparameter separator for microscopic particles and biological cells. *Review of Scientific Instruments* **44**, 1301–1310 (1973).

190. F. Porichis, M. G. Hart, M. Griesbeck, H. L. Everett, M. Hassan, A. E. Baxter, M. Lindqvist, S. M. Miller, D. Z. Soghoian, D. G. Kavanagh, *et al.*, High-throughput detection of miRNAs and gene-specific mRNA at the single-cell level by flow cytometry. *Nature communications* **5**, 5641 (2014).

191. A. S. Rane, J. Rutkauskaite, S. Stavrakis, A. J. deMello, High-Throughput Multi-parametric Imaging Flow Cytometry. *Chem* **3**, 588–602 (2017).

192. D. A. Basiji, W. E. Ortyn, L. Liang, V. Venkatachalam, P. Morrissey, Cellular image analysis and imaging by flow cytometry. *Clinics in laboratory medicine* **27**, 653–670 (2007).

193. T. C. George, D. A. Basiji, B. E. Hall, D. H. Lynch, W. E. Ortyn, D. J. Perry, M. J. Seo, C. A. Zimmerman, P. J. Morrissey, Distinguishing modes of cell death using the ImageStream® multispectral imaging flow cytometer. *Cytometry Part A* **59**, 237–245 (2004).

194. M. E. Skindersoe, S. Kjaerulff, Comparison of three thiol probes for determination of apoptosis-related changes in cellular redox status. *Cytometry Part A* **85**, 179–187 (2014).

195. A. Filby, J. P. Houston, Imaging cytometry: Automated morphology and feature extraction. *Cytometry Part A* **91**, 851–853 (2017).

196. A. D. Posey, O. U. Kawalekar, C. H. June, Measurement of intracellular ions by flow cytometry. *Current protocols in cytometry*, 9–8 (2015).

197. A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, *et al.*, CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology* **7**, R100 (2006).

198. O. Otto, P. Rosendahl, A. Mietke, S. Golfier, C. Herold, D. Klaue, S. Girardo, S. Pagliara, A. Ekpenyong, A. Jacobi, *et al.*, Real-time deformability cytometry: on-the-fly cell mechanical phenotyping. *Nature methods* **12**, 199 (2015).

199. J. Kovac, J. Voldman, Intuitive, image-based cell sorting using optofluidic cell sorting. *Analytical chemistry* **79**, 9321–9330 (2007).

200. Q. Gu, T. Aoyama, T. Takaki, I. Ishii, Simultaneous vision-based shape and motion analysis of cells fast-flowing in a microchannel. *IEEE Transactions on Automation Science and Engineering* **12**, 204–215 (2015).

201. D. H. Hubel, T. N. Wiesel, *Brain and visual perception: the story of a 25-year collaboration* (Oxford University Press, 2004).

202. G. Holzner, S. Stavrakis, A. DeMello, Elasto-inertial focusing of mammalian cells and bacteria using low molecular, low viscosity PEO solutions. *Analytical chemistry* **89**, 11653–11663 (2017).

203. M. A. Unger, H.-P. Chou, T. Thorsen, A. Scherer, S. R. Quake, Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science* **288**, 113–116 (2000).

204. S. Bjarne, *The C++ programming language*, 1997.

205. M. Abadi, A. A. B. P. TensorFlow, presented at the 12th USENIX Symposium on Operating Systems Design and Implementation (2016), 265.

206. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* **15**, 1929–1958 (2014).

207. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

208. R. B. Maxwell, A. L. Gerhardt, M. Toner, M. L. Gray, M. A. Schmidt, A microbubble-powered bioparticle actuator. *Journal of microelectromechanical systems* **12**, 630–640 (2003).