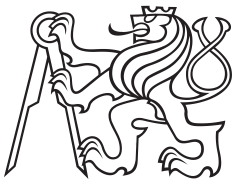
Bachelor's thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

GPU-accelerated computer vision system for feedback micro-manipulation

Viktor-Adam Koropecký

Supervisor: Ing. Martin Gurtner
Field of study: Cybernetics and Robotics
May 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Koropecký Viktor-Adam** Personal ID number: **465906**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

GPU-accelerated computer vision system for feedback micro-manipulation

Bachelor's thesis title in Czech:

Zpracování obrazu na GPU pro zpětnovazební mikro-manipulaci

Guidelines:

Develop a computer vision system measuring 3D positions of micro-objects based on digital holographic microscopy (DHM). The system will display the reconstructed DHM image on a monitor and output the measured position of micro-objects via TCP packets. For efficiency, the system will be implemented in CUDA and run on a dedicated computing device NVIDIA Jetson TX2.
1. Get familiar with image reconstruction algorithms used in DHM. Choose and describe one.
2. Implement the chosen image reconstruction algorithm in CUDA.
3. Opt for a suitable object tracking algorithm and implement it in CUDA.
4. Wrap the functionalities of image reconstruction and object tracking in a parameterizable service displaying the reconstructed image and sending the objects' position via TCP packets.

Bibliography / sources:

[1] M. Gurtner and J. Zemánek, "Twin-beam real-time position estimation of micro-objects in 3D," Meas. Sci. Technol., vol. 27, no. 12, p. 127003, 2016.
[2] A. Greenbaum et al., "Imaging without lenses: achievements and remaining challenges of wide-field on-chip microscopy," Nature methods, vol. 9, no. 9, pp. 889–895, 2012.
[3] W. Liu et al., "SSD: Single shot multibox detector," in European conference on computer vision, 2016, pp. 21–37.

Name and workplace of bachelor's thesis supervisor:

Ing. Martin Gurtner, Department of Control Engineering, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **18.02.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

Ing. Martin Gurtner Supervisor's signature	prof. Ing. Michael Šebek, DrSc. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Acknowledgements

I would love to take this moment to show my gratitude to my supervisor Martin Gurtner for all of his help and guidance, as well as for the opportunity to start working on this project as an intern during the last summer. My sincere thanks also goes to my family for their patience and support not only in the past couple of weeks. And, of course, I have to thank all of my friends and classmates, for being a great company while writing the following work.

Declaration

I confirm that I wrote the following work based on my individual efforts and I used to complete a list of the references that I mention in the list annexed to the thesis in accordance with Methodological guideline number 1/2009 on the observance of ethical principles in the preparation of university thesis.

Prague, May 24, 2019
Viktor-Adam Koropecký

Abstract

This bachelor's thesis focuses on designing, implementing and testing a system for real-time position estimation for micro-particles. The system uses *digital holographic microscopy* (DHM) for visualizing observed micro-particles and the twin-beams illumination method for estimating their 3D positions. The Rayleigh-Sommerfeld back-propagation method is used as the image reconstruction method. Linear filtering is then implemented for finding the micro-particles in the reconstructed images. Implementation of this system is in CUDA, to allow for direct calculation on the GPU. The implemented system is then deployed on an embedded platform NVIDIA Jetson TX2. This platform is then able to communicate and send the processed positions to a main manipulation computer. The utilized algorithms are optimized to allow for time-efficient and accurate estimates. The final implementation then offers substantial progress when compared to previously implemented systems.

Keywords: GPU, linear filtering, digital holographic microscopy, CUDA

Supervisor: Ing. Martin Gurtner
Fakulta elektrotechnická,
Karlovo náměstí 13,
Praha 2

Abstrakt

Tato bakalářská práce se zaobírá návrhem, implementací a testováním systému pro odhad pozic mikročástic v reálném čase. Tento systém využívá *digitální holografickou mikroskopii* (DHM) pro zobrazování pozorovaných mikročástic, a dále je v návrhu využita dvoupaprsková osvětlovací metoda pro odhad jejich 3D poloh. Rayleigh-Sommerfeldova zpětno-propagační metoda je použita jako metoda pro rekonstrukci obrazu. Lineární filtrování je dále implementováno pro nalezení poloh mikro-částic v rekonstruovaném obraze. Implementace systému je provedena v CUDA, pro přímou kalkulaci na GPU. Implementovaný systém je nasazen na platformu NVIDIA Jetson TX2. Tento systém je dále schopen komunikovat s hlavním manipulačním počítačem a zasílat mu zpracované pozice. Použité algoritmy jsou optimalizované pro časově efektivní a přesné odhady pozic. Finální implementace tohoto systému pak nabízí značný pokrok oproti předchozím implementovacím systému.

Klíčová slova: GPU, linearní filtrování, digitální holografická mikroskopie, CUDA

Překlad názvu: Zpracování obrazu na GPU pro zpětnovazební mikro-manipulaci

Contents

Chapter 1

Introduction

This thesis focuses on implementing and testing a system for real-time position estimation of micro-particles by the use of lensless *digital holography*. This system will then later be used for tracking micro-particles in a feedback micro-manipulation platform developed by research group AA4CC[1].

Other versions of such a position estimation system have already been implemented in the past. Our implementation then strives to serve as a clear improvement, by achieving much higher frame rate. This would make for a smoother micro-manipulation. In addition, achieving higher frame rate would allow us to implement better image sensors, and thus become more accurate as well.

In the implementation, we will be utilizing one of the image reconstruction methods used in *digital holographic microscopy* and a selected object detection algorithm. We will then implement these in CUDA, which will let us program these methods directly on a GPU.

The final system should then be deployed on an embedded computer and connected to the main manipulation computer via a TCP communication protocol. The main manipulation computer should be able to control aspects of the running system through this communication protocol. At the same time, the system should be sending calculated results of every frame to the main manipulation computer via this communication protocol.

1.1 Structure of thesis

In this thesis we will first take a look at existing methods of estimating 3D positions of micro-objects and on the original setup developed by the members of research group AA4CC in chapter 2. Then, we select appropriate methods for reconstructing the captured interference patterns in section 3.1. After that, we will select methods for finding and tracking the positions of micro-particles from the reconstructed images in section 3.2. Later, we propose an implementation of these methods on a GPU in chapter 4 and finally we experimentally set important values and evaluate the implemented algorithm in chapter 5.

[1] More information about the group at: http://aa4cc.dce.fel.cvut.cz/

Chapter 2

Related works

Regular methods of observing micro-objects use a selected type of optics to project an image of the micro-object itself. DHM, on the other hand, is a method of capturing interference patterns, i.e., holograms of observed micro-objects. The interference patterns form when the observed micro-objects are illuminated with a source of coherent light, and can be captured with a simple image sensor. Then these patterns are post-processed via computer to visualize the observed micro-objects. Therefore a major advantage of DHM is the lack of need for any lenses or other optics.

An interference pattern encodes the complete information about the distance and shape of the micro-object it originated from. That is encoded at every point in the phase and amplitude of the electromagnetic wave representing the interference pattern. Part of this information is lost however, since the image sensor only captures the intensity of light at each pixel. Thus algorithms that wish to estimate the 3D positions of micro-objects have to process the interference patterns just from the intensity image.

As with regular methods, estimating 2D position, or the lateral position, is mostly straight-forward. The center of an interference pattern or of a reconstructed image of the observed micro-particle corresponds to the lateral position of that micro-object.

When it comes to 3D position estimation in DHM, several methods already exist—their review can be found in [1]. Most of these then depend on either one of two different approaches to calculating axial distance, or the perpendicular distance from the image sensor. The first one tries to fit the captured interference pattern to a model describing the hologram's appearance in proportion to its axial distance [2]. This approach can be very precise—down to units of nanometers—but that also requires high resolution of the captured hologram. That is possible by using additional lenses, but that reduces observable area. This approach is also computationally heavy. The other approach uses a process called back-propagation to calculate the axial distance by comparing the reconstructed hologram in different heights to the objects themselves [3, 4]. Back-propagation itself is not a computationally intensive algorithm, but it usually has to be run multiple times for the method to work.

A yet another approach then uses multiple sources of coherent lights to

illuminate the micro-particles [5]. Here the projected holograms are shifted with respect to each other. The magnitude of this shift corresponds to the axial distance of the observed micro-particle. If only two sources of coherent light are used, this method is called the twin-beams illumination method [6].

Most of these methods are not, however, intended for real-time use, but rather for off-line post-processing.

The members of the AA4CC research group have developed a feedback micro-manipulation platform. Because they were limited by the lack of real-time 3D position estimation method, the feedback micro-manipulation ran on estimating only 2D positions of observed micro-particles. Later, a real-time method of 3D position estimation method was developed in the group [7]. This method serves as the basis for this thesis. A short introduction of this method is included in the following section.

2.1 Twin-beam real-time 3D position estimation of micro-objects

The objects of focus of the feedback micro-manipulation are polystyrene micro-particles of diameter 50 µm. Those are suspended in water contained in a 2 mm deep pool. Under this pool is then located an electrode array, which serves as the manipulation unit. The manipulation process uses the phenomenon called dielectrophoresis—by changing the potentials on on the electrodes, force is generated on the micro-particles.

For feedback manipulation it is however necessary to utilize a tracking mechanism, that would feed the information about the current positions of tracked particles back to the loop. Therefore, the setup utilizes a twin-beam method, in which two sources of coherent light—one set directly above the pool and one in an angle of approximately 30°—illuminate the pool and an image sensor situated below the electrode array [7, 8]. The translucent micro-particles in the pools then partially absorb, reflect, diffract the incoming light, as well as letting some of it pass. The resulting diffraction patterns and scattered light then form interference patterns on the image sensor below.

The light sources utilized here are red (625 nm) and green (525 nm) LEDs butt-coupled with plastic optical fibers. Because the LEDs emit in a narrow band of frequencies, we can consider their light to be partially temporally coherent. Spacial coherency is the ensured by the filtering through the fibers, which have a diameter of 500 µm. The complete original setup is visualized in figure 2.1.

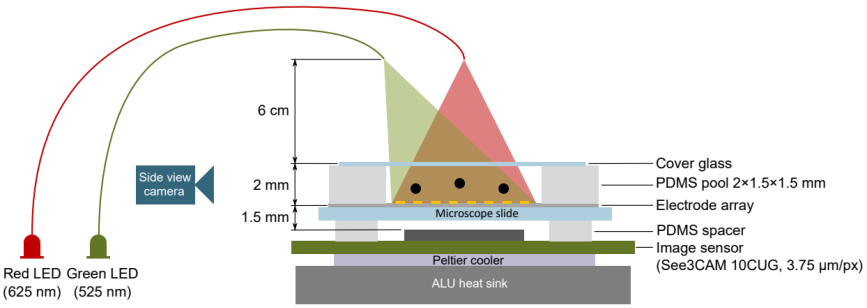


Figure 2.1: Visualization of the hardware setup utilized for previous implementations. The image is reprinted from [8].

Interference patterns caught on the image sensors form red and green pairs. These pairs are shifted from each other in proportion to the axial distance of the micro-particle from the image sensor. From this the axial distance can be easily numerically calculated.

The lateral position is estimated from the positions of the interference patterns in the green channel.

Originally, to enhance the estimation, the interference patterns were back-propagated to a distance where they focus to a single point. This allows us not only to more easily predict the positions of micro-particles but also The back-propagation is explained more in depth in 3.1. After that the center of mass of predicted regions is calculated to give the exact position of the micro-particle. The region is selected either as the close proximity of the micro-particle's last position or manually during the initialization process.

2.1.1 Issues

While the developed algorithm has yielded favorable results, it could still be improved in some fields. Mainly, the algorithm itself was programmed on Simulink Matlab and runs on the CPU. This results in a significant slow-down during the image processing. Programming on the GPU would result in much faster computation times in these fields. Faster computation times would then allow us to utilize more precise image sensors and thus achieve, not only higher calculation speeds, but also higher accuracy when predicting positions of observed micro-particle.

In addition particles are currently only detected, if their position is selected manually during the initialization process. Subsequently, the positions of these particles are then estimated by calculating the center of mass around the last known position. This could be improved by implementing an algorithm, which can find all micro-particles in the observed environment without the need for manual input.

Chapter 3

Selected methods

This chapter focuses on selecting an appropriate method of image reconstruction in DHM. That will not only be used for the object detection algorithm but also to visualize current state of the observed environment. In later sections, this chapter will discuss possible object detection algorithms, that could be used in the given problem. The algorithms selected in this chapter have to be suitable for real-time use and thus should be highly parallelizable.

The methods will be later implemented on a similar platform as is described in [8]. Thus, these methods should be able to sufficiently process interference patterns obtained via the twin-beams illumination method.

3.1 Back-Propagation

As it is possible to see in 3.2, from interference patterns it is difficult to deduce any information about the displayed objects. If interference patterns overlap, this gets even more difficult. Object detection algorithms would have trouble distinguishing between the pictured interference patterns.

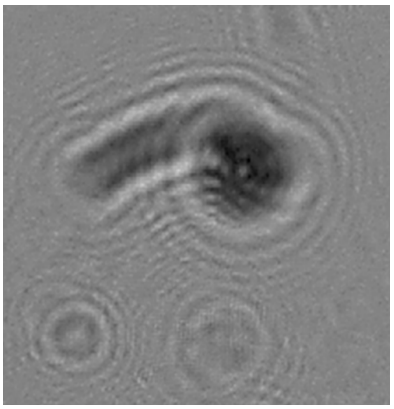


Figure 3.1: Captured interference patterns of a dust particle next to a polystyrene micro-particle.

Thus we need to introduce an algorithm that would transform these patterns into a more accurate representation of the original micro-objects. One method we could use is the Rayleigh-Sommerfeld back-propagation method. This

method actually simulates back-propagating the interference patterns to a set axial distance. If this axial distance is set to the actual distance of the object from the image sensor, we will get a clear image of the micro-particle itself. That, however, is only true when observing the micro-particle through a homogeneous medium. A back-propagated projection of a micro-particle is visualized in figure

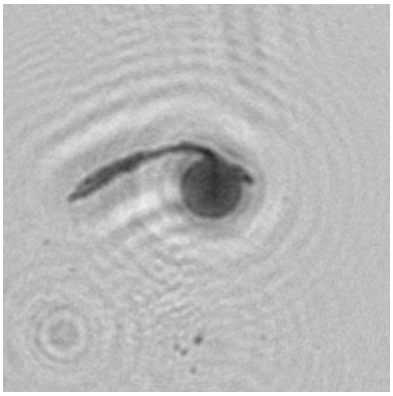


Figure 3.2: Reconstructed image of a dust particle next to a polystyrene micro-particle.

This method was already used in the original method proposed by the AA4CC research group.

3.1.1 Calculation

This method is in our case calculated by convolving the taken image with the Rayleigh-Sommerfeld propagator. This is also a numerical solution to the Rayleigh-Sommerfeld diffraction integral. That can be seen in the expression

$$I_z(x,y) = \mathcal{F}^{-1}\{H_{-z}(f_x,f_y)\mathcal{F}\{I(x,y)\}\}, \tag{3.1}$$

where (x,y) are the image coordinates, (f_x,f_y) are the spatial frequencies, I is the original image, I_z is the image back-propagated to a distance z a $\mathscr{F}$ and $\mathscr{F}^{-1}$ are Fourier and inverse Fourier transformations respectively. The Fourier transform of the propagator is then given by the expression

$$H_z(f_x,f_y) = \begin{cases} exp(i2\pi z\frac{n}{\lambda}\sqrt{1-(\frac{\lambda f_x}{n})^2-(\frac{\lambda f_y}{n})^2}), & \sqrt{f_x^2+f_y^2} \leq \frac{n}{\lambda}, \\ 0, & \text{otherwise}, \end{cases} \tag{3.2}$$

where λ is the wave length of the illumination and n is refractive index of the surrounding medium. The distance to which the image is propagated to is then given by z.

From this we can see that the method has a clear advantage in being highly parallelizable and thus suited for calculation on the GPU.

3.1.2 Issues

The 3.2 equation expects that the medium between the micro-particle and the image sensor is homogeneous. This however is clearly not through, as the light has to pass through several layers of different mediums, such as the protective glass or the electrode array. This means that the distance z put to the calculation of the expression 3.2 does not actually represent the real axial distance—it is however proportional to it. That does not actually pose a problem in our situation. We only use the back-propagated images to allow for more comprehensible representation of the observed micro-particles for both a human user and the object detection algorithm.

3.2 Object detection and tracking

After reconstructing the images to a more representative state, we are faced with a problem of actually finding the now visualized objects and interpolating their spatial position. We are still pressed by speed demands of a real-time estimation. Therefore the selected algorithm needs to be highly time-effective.

One such algorithm offers itself in this situation. Linear filtering [9, 10] is a very simple feature detection algorithm based on 2D convolution, that will be further discussed in the next section. Major advantages in choosing such an algorithm is in the shape of tracked micro-particles and their surroundings. Firstly the shape of these micro-particles varies only very slightly between one another and the micro-particles are circularly symmetric. Secondly, the background is largely homogeneous. The non-tracked micro-objects, such as random dust particles, found are substantially different to the desired objects.

3.2.1 Linear filtering

Linear filtering works on a very similarly to 2D convolution. It works by sliding a 2D filter, otherwise called a kernel, across an image. At each step, the overlaying elements are multiplied and the total of these products form a new 2D image of extracted features.

Since the environment in which we track the micro-particles is monotonous and few similar looking invasive particles are present, we can utilize this for objection detection by making a kernel of the size of one tracked micro-particle. By experimentally setting its values, we can accurately find the two-dimensional positions of desired objects. Linear filtering of an image f of size $M \times N$ with a filter h of size $m \times n$ is then formally defined by the expression

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s,t)h(x+s,y+t), \tag{3.3}$$

where $a = (m-1)/2$ and $b = (n-1)/2$. $g(x,y)$ is then a new filtered pixel at position (x,y). To get complete filtered image, this expression must be applied for all x going from 0 to M-1 and y going from 0 to N-1 [10]. From

this we can see that to convolute an image f with a kernel h, $(M \times N)(m \times n)$ multiplications and $(M \times N)(m \times n - 1)$ additions need to be calculated. Thus the time complexity of this algorithm is $O(M, N, m, n) = MNmn$ [9].

The problem here is that the complexity is proportional to the product of the filter's dimensions. If we choose a very large kernel, which is actually advantageous in the given problem, the filtering calculation might take way too much time to be worth using. Luckily, there are ways to lower the proportion of time complexity to the size of used filter by a large margin.

Separable filters

The most obvious way is to use a separable filter. A two-dimensional filter is separable, if there are two one-dimensional filters, one horizontal and one vertical, which when multiplied return the original two-dimensional filter. The filtering is then carried by first filtering the original image with either one of the one-dimensional filters and then filtering the result with the second filter. The time complexity with a separable filter is then given as $O(M, N, m, n) = MN(m + n)$ [9].

The obvious problem here is that not all two-dimensional filters are separable, and as such we would be very limited when choosing appropriate filters. Also, while the speed-up is definitely significant, we are still limited by the size of the filter.

Linear filtering in frequency domain

As was mentioned before, linear filtering is very closely based on 2D convolution [10]. We can show that, if we take a look at the expression which defines 2D convolution of discrete functions $f(x, y)$ and $h(x, y)$ of the size $M \times N$:

$$f(x, y) * h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n)h(x - m, y - n) \tag{3.4}$$

Here we can see that the linear filtering in (3.3) only differs in two ways:

- Function h is mirrored by the origin, which is emphasized by the minus signs.
- Linear filtering lacks the leading constant of $\frac{1}{MN}$.

Thus, if the filter is mirrored by the origin, we can use 2D convolution for the same effect. Then, if we define $F(u, v)$ and $H(u, v)$ as the Fourier transforms of $f(x, y)$ and $h(x, y)$, respectively, from the properties of the Fourier transformation we know that

$$\mathcal{F}\{f(x, y) * h(x, y)\} = F(u, v)H(u, v). \tag{3.5}$$

This leads to possibly the best version of the convolution algorithm for the current problem. By getting the Fourier transform of both the filter and the

original image, we can drastically lower impact of the filter's size on the time complexity. Actually, the time complexity now only depends on the size of the original image [10, 9].

Issues

In the project behind this thesis, we utilized linear filtering algorithm in the frequency domain as our micro-particle detection algorithm. Linear filtering does however have a couple disadvantages disregarding its speed.

- As was already mentioned, linear filtering, or in our case 2D convolution, requires monotonous environment. Under the coherent lighting in the used setup, this requirement is well satisfied.
- There should be no invasive particles with similar geometry as the tracked particles. It's not really possible to ensure this, however other algorithms struggle with this problem as well.
- If tracked micro-particles get too close together and overlap, convolution might have a problem detecting individual particles. This is not possible to easily solve with linear filtering.

Other options

Obvious choice when doing any object detection would be the use of *convolutional neural networks* (CNN). In this case, their use might actually prove to be less effective than the earlier proposed method. The only issue, which poses a real problem for regular linear filtering would also pose a problem for CNNs. This is of course the issue of overlapping particles. However, this changes if we were to track multiple types of particles. A CNN based on the *Single shot multibox detector* (SSD) [11] might prove to be more effective in such situations. Similarly, micro-particles that are not circularly symmetric would be way harder to track with linear filtering.

Another methods suggest that instead of immediately using the filter to find tracked micro-particles, different smaller filters could be used beforehand to pre-process the reconstructed image and highlight some features. This could render the final filtering more effective.

3.2.2 Particle tracking

After finding the individual micro-particles it is also important to be able to follow each micro-particle through the manipulation process. User should be able to both pick individual subject micro-particles to be tracked in the initialization process or not select any so that the algorithm can keep track of them all.

Luckily, in our case we can assume that the micro-particles will not change their position significantly between frames. Thus, in each frame we can assume that the new position is the closest position to the one in the frame before.

Issues

Issues with this method arise again from the problem of overlapping particles. If two or more micro-particles get too close together, this algorithm might estimate that the particles switched and classify them wrong. Such a problem will probably occur only if the observed micro-particles overlap completely.

Possible improvements

Any problems stemming from using the previously proposed tracking algorithm would be fixed by implementing *linear quadratic estimation*, otherwise known as *Kalman filtering*. This algorithm is a widely used recursive prediction-update based state estimator algorithm. Its goal is to estimate a state of a process (in our case the positions of individual micro-particles), while minimizing the squared error [12, 13].

Chapter 4

Implementation

After finding the appropriate methods it is time to propose how to connect them to make a working 3D position estimation system and then implement[1]. it on our setup.

In this chapter, we will take a look at the hardware setup utilized for our implementation. Later we will mention the APIs used. After that we will propose the basic structure of the implemented process. Then, finally, we will try to propose the fastest implementation for the methods selected in the previous chapters.

In the following sections, we will often refer to the *green channel* and *red channel*. That is because the coherent light sources are of red and green colors and their effects can therefore be separated from each other by converting the image to the RGB format. Thus the red channel corresponds to an image, where the effects of the red light source are observable and the green channel corresponds to an image, where the effects of the green light source are observable.

4.1 Hardware

For the implementation of the image processing pipeline we used two pieces of commercial hardware. First, we needed a powerful, yet cost-effective, computing platform, which would serve as the image processing unit. Therefore, we are looking for a platform with a powerful GPU. For this we selected the Jetson TX2 embedded computer developed by NVIDIA. We used the LI-TX1-CB by Leopard Imaging to dock this computer.

Second, we required an image sensor that can capture and feed frames to the pipeline at appropriate speeds of at least 30 FPS. Another requirement for the camera is pixel size of at most $2 \times 2\mu m$

Apart from these two, we utilize the experimental feedback micro-manipulation platform. That is then built around the selected image sensor.

[1] All the code implementation of the algorithms described in this chapter can be found in an online repository: *https://github.com/aa4cc/twinbeam-setup*

4.1.1 Jetson TX2

The Jetson TX2 is an embedded computer designed by NVIDIA with AI and image processing in mind. Compared to regular GPUs for personal computer uses, it has two major advantages. Firstly, regular GPUs have their own dedicated RAM units separated from the RAM designated for the CPU. In the Jetson TX2, however, the 8 GBs of dynamic RAM is shared between all the CPU cores and the GPU. As such transfer of data between GPU and CPU processes is almost much faster than on a regular computer. Secondly, the architecture is designed to continuously receive images from up to 6 4K cameras through the *camera serial interface* (CSI) at 30 FPS. Both of these points will increase the speed of our pipeline in the proposed implementation.

4.1.2 Image sensor

For the image sensor, the LI-IMX477-MIPI camera board by Leopard Imaging was chosen. The pixel size of the sensor is 1.55 µm, which is well in the requested specification. It can also be connected to the Jetson TX2 via the *camera serial interface* (CSI), which has a bandwidth of 2.5 Gb/s. Such bandwidth is enough for streaming 4K (4056 × 3040 pixels) in 30 FPS.

4.1.3 Experimental platform

The experimental platform utilized in our implementation is, at the time of finishing this thesis, being developed by members of the AA4CC research group. The author of this thesis does not partake on this development.

The platform itself is built on top of the IMX477 image sensor and does not vary much from the original setup mentioned in section 2.1. The cross section of this setup with description of relevant parts is visible on the figure 4.1. The inside look to the platform is then photographed on figure 4.2

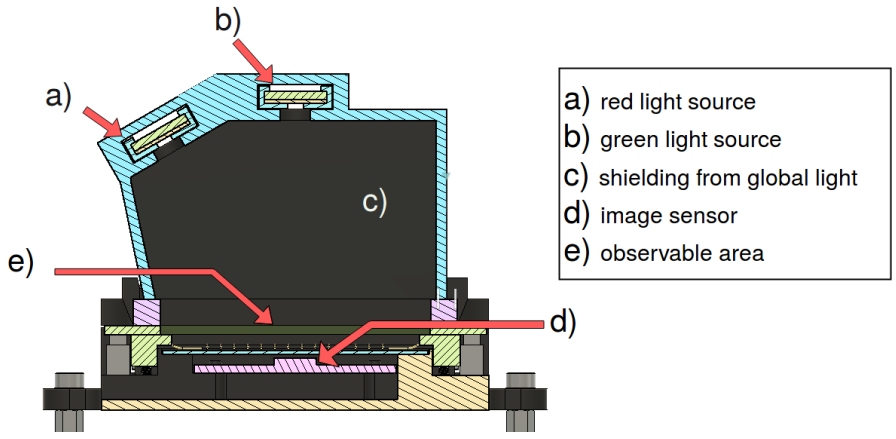


Figure 4.1: The experimental setup for feedback micro-manipulation of micro-particles.

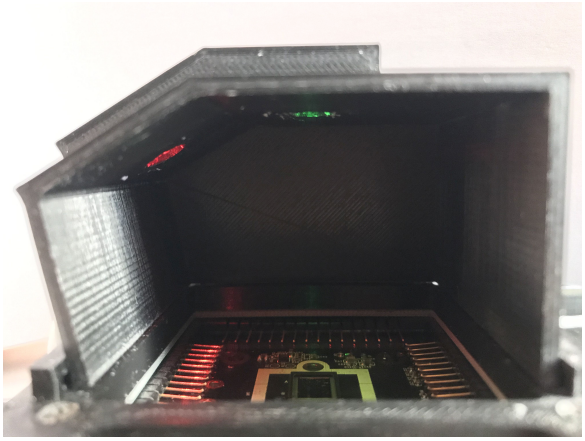


Figure 4.2: A photo of the experimental setup for feedback micro-manipulation of micro-particles.

The light sources in this setup are again formed by a green (525 nm) LED and a red (625 nm) LED. Unlike in the original setup however they are not butt-coupled to plastic optical fibers. Instead, a thin layer of aluminum is attached to the LED. A circular hole of 50 µm is pierced through it. This hole then ensures better partial spatial coherency of light than the plastic optical fiber utilized before. The light source is clearly visible on figure 4.3.

Figure 4.3: The green light source. There is a slightly visible hole in the aluminum layer. Behind is situated a green LED.

The setup used in this implementation lacks the electrode array required for feedback micro-manipulation. In addition the pool of water in which the micro-particles would normally be contained is missing as well. The micro-particles are thus only dropped on a layer of protective glass. These parts are missing due to manufacturing issues, but they will be added in the soon future.

4.2 APIs

Several APIs were used for the implementation of the methods selected in previous chapters. In this section we will only mention the ones crucial to successfully finish the implementation. Those are CUDA, EGL and LibArgus.

4.2.1 CUDA

CUDA itself consists of a driver, a runtime, an API and other mathematical libraries. Its main purpose is to exploit the computing capabilities of a GPU via the use of kernels - sections of code that are designated to run on a number of GPU threads. Therefore, it allows us to fully utilize the parallelization capabilities of the used GPU.

The APIs also include several mathematical functions that will become necessary in our implementation. For example, the fully parallelized implementation of *fast fourier transform* (FFT) will be used whenever there is a need to calculate either the direct of inverse Fourier transform.

4.2.2 libargus

libargus is a low-level API for acquiring images and other meta-data from cameras. Here it is utilized to transfer captured frames from the image sensor and transfer it directly to GPU memory for CUDA usage. It follows a fairly strict initialization process. Through this API we can also set parameters for image capture such as the requested resolution, exposure time or analog gain of the camera.

It also allows us to manage direct requests for the camera. That is done by initializing an output stream for the camera. Then by repeatedly calling the request, we can pass the captured frames to an EGLStream entity. EGLStream is then capable of passing the frame directly to the GPU's texture memory for post-processing by CUDA.

4.2.3 EGL

We use the EGL library, because it allows us to connect the libargus pipeline to CUDA through the use of the entity EGLStream.

4.3 Process structure

The proposed pipeline is built in four main CPU threads plus a debug thread for easier control during implementation. Each of these controls a separate part of the process.

Main processing loop thread – This thread controls both the initialization of libargus pipeline and the main loop for receiving and processing the captured frames.

Displaying thread – This thread displays the current back-propagated image on an external display every third cycle of the main processing loop.

Input thread – This thread takes care of processing the TCP communication with Matlab running on the main manipulation computer. The whole process can be started and put to sleep from this thread.

Output thread – Every frame the calculated positions of the micro-particles is first processed to an array of positions and later sent from this thread to the main manipulation computer. Per request this thread can also send the current processed image.

Keyboard input thread – During the implementation it was useful to have the opportunity to control the process from the keyboard. It has the same functionality as the *input thread* and serves only for debugging. That is because in the final setup, the Jetson will not be connected to a keyboard. This thread does not require further description.

In the following sections we will go through the functionalities of each of these individual threads.

4.3.1 Main processing loop

The main processing loop should takes care of a number of repeating steps that proceed in the following order.

1. Initializing the camera driver through the libargus API.
2. Allocating the necessary GPU memory.
3. Setting the parameters of the libargus output stream and requests.

   Here the exposition time, maximum FPS and capture size are be set.
4. Creating an EGLStream entity and attaching it to a CUDA consumer.
5. Requesting a new frame from the camera and passing it to texture memory.

   We can access the texture memory, by binding CUDA texture references to appropriate pointers. Texture memory is cached directly on the GPU chip and thus using it can achieve higher bandwidths than using regular RAM. However, the texture memory is limited in size and harder to operate and so is not used further.
6. Decoding the color format to the RGB format and passing red and green channels to RAM.

   The libargus can only fetch frames in the YUV-420 format. This means that to separate the red and green channels we first need to convert the image to the appropriate RGB format. The blue channel is useless in our process. The conversion algorithm is further explained in section 4.4.1.

7. Back-propagating both channels.

   Here the back-propagation method proposed in section 3.1 is implemented. However, in this step the method does not end, as the inverse Fourier transformation is run after convolving both channels in the next step. The inverse Fourier transformation is then only run on the green channel, as the red channel is not needed for visualization. FFT implemented in CUDA library CUFFT is utilized in this step. The implementation is further explained in section 4.4.2.

8. Convolving both channels with appropriate filters.

   Here the frequency domain convolution proposed in section 3.2.1 is implemented. The direct Fourier transformation does not have to be run on the individual channels, as we can use the transformed channels from the previous step. After this step finishes, the convolved channels are transformed with the inverse Fourier transformation. The implementation of this step is further explained in section 4.4.3.

9. Finding the local maxims in the filtered channels.

   In this step we find the local maxims in both of the filtered channels. This should highlight only the centers of each particle and nothing else. Later the processed channels are passed to the output thread for ordering.

10. Check for interrupt signal from the input thread. If the signal is detected proceed to next step. Otherwise, repeat steps 5-10.

11. Deallocate GPU memory and reset the output stream.

12. Wait for the starting signal from the input thread and then repeat steps 2-12.

4.3.2 Displaying thread

Every three cycles of the main processing loop we want to visualize the current state of the observed environment. That is done simply by copying the current back-propagated image in the green channel to temporary GPU memory and using the openCV API to visualize this array on an external display.

To prevent any memory conflicts when copying the current image, we lock either the main processing loop thread or the displaying thread whenever the other is using the afflicted memory. For this reason we copy the memory to another GPU array, instead of directly to CPU, which is, even on the Jetson TX2, marginally slower.

4.3.3 Input thread

When the Jetson TX2 is started, this process will automatically start and takes care of parsing packets from the main manipulation computer. According to the provided guidelines, the protocol was designed using TCP packets and, apart from connecting to the Jetson TX2, has the following functionalities:

Initialize

When the process is woken up from the sleep state by sending the starting signal, it captures and runs one loop of the main processing loop. After that the Jetson TX2 will send an image with a set of found micro-particles. On the main manipulation computer, user can manually set, which of these particles are to be tracked throughout the process. If no particles are selected, the process tracks all particles in the observable area.

Change settings

Almost all settings need to be adjustable from the main manipulation computer. Thus, when in the sleeping mode, a change settings signal can be sent to change the following parameters:

- *Size of the observable area* – By default this is given by $(w, h) = (1024, 1024)$, where w is the width and h is the height of the observable area. This corresponds to approximately 1.5×1.5 mm^2.
- *Offset* – This sets the position of the top left pixel of the observable area. By default it is given by $(w_o, h_o) = (1500, 1000)$, where w_o is the horizontal coordinate and h_o is the vertical coordinate. This places the default observable area roughly to the center of the 4K frame fetched by the camera. For the settings to be accepted these conditions must be true: $0 \geq w + w_o \leq 4056$ and $0 \geq h + h_o \leq 2578$. This is because the observable area must fit into the captured frame.
- *Back-propagation distance* – With different samples it might become advantageous to reconstruct the captured frames in different distances from the sensor. As such two values can be pass for the back-propagation distance for both the red and green channels. By default it is given by $(z_g, z_r) = (0.0025, 0.0025)$. These values correspond to the distance of 2.5 mm.
- *Exposition time* – This is mostly an experimental parameter, since it is unlikely to be changed for any reason during the implementation.

The settings can be changed via a dialog window that can be seen in figure 4.4.

Picture request

User can send a request signal, after which Jetson TX2 will send the current back-propagated image.

Put to sleep

When the user wishes to end the tracking process, he can send an exit signal. After that an interrupt signal is sent to all threads to end all loops and wait for starting signal.

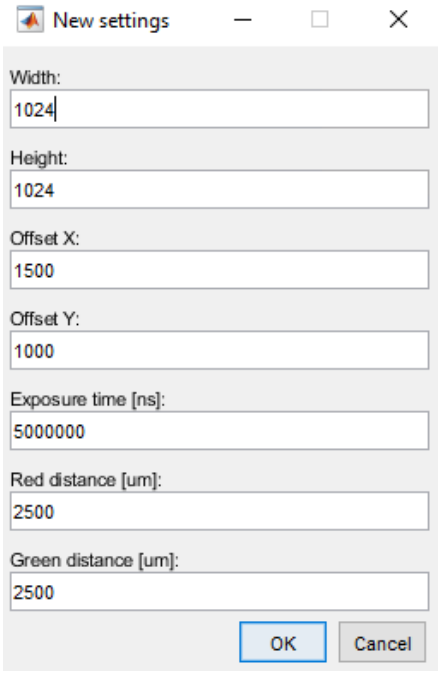


Figure 4.4: Dialog window for changing the image processing settings.

Handshake

If the user wants to check the connection status, he can send a handshake signal, to which the Jetson TX2 should respond with a corresponding message.

4.3.4 Output thread

The output thread takes care of sending the found locations to the main manipulation computer. The positions of found micro-particles in both channels are sent every frame after being extracted from the maps of local extremes. The position estimation then runs on the main manipulation computer.

If requested or during the starting sequence, this thread can also send the current back-propagated picture. This picture is down-sampled. To prevent any conflicts during accessing the memory area, where back-propagated images are saved, this thread locks all other threads trying to access the same bit of memory.

4.4 Implementation of proposed methods

In this section, we will describe the implementation of methods selected in 3. In addition we will describe the converting process from the YUV-420 to RGB color format.

4.4.1 Format conversion

The libargus API can only pass frames in the YUV-420 format. Therefore, if we want to separate the red and green channels, we first need to convert this format to the RGB format. During this process, the captured frame is bound to parts in the texture memory. Converted pixels then move to RAM.

YUV-420

This format consists of three components. Y represents the luminance, or the brightness of pixel, and we get one value per every pixel. The UV components represent the chrominance, or the color of the pixel, and we get one U value and one V value per every square of 4 pixels. This is visualized in the figure 4.5.

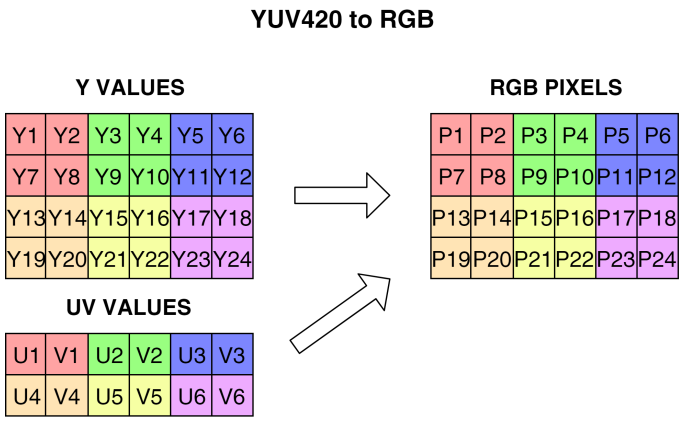


Figure 4.5: Visualization of conversion from YUV-420 to RGB color format. Each Y value corresponds to one RGB pixel. Each pair of UV values then corresponds to 4 RGB pixels.

Conversion

For conversion we only care about the red and green channels and so we can completely omit conversion to the blue channel. Then the conversion between individual components is given by the following expressions:

$$\begin{aligned} R &\doteq 1.164383(Y-16) &&+ 1.596027(V-128), \\ G &\doteq 1.164383(Y-16) - 0.391762(U-128) - 0.812968(V-128), \end{aligned} \tag{4.1}$$

where R represents a red pixel and G represents a green pixel.

4.4.2 Back-propagation

Back-propagation is implemented in four steps.

1. Calculate the Fourier transforms of the Rayleigh-Sommerfeld's propagators for both channels. The equation for these is given in 3.2. These

Fourier transforms do not theoretically need to be calculated each cycle, however, the time it takes to calculate them is negligible. The resulting arrays need to have the same amount of elements as the appropriate channels.

2. Calculate the Fourier transform of the appropriate channel by the use of CUFFT.

3. Multiply each of the elements in the Fourier transform of a channel with the corresponding elements from the appropriate propagator transform.

4. Calculate the inverse Fourier transform of the product.

The final step is only calculated for the green channel as the result of the inverse Fourier transform is in our implementation only utilized for visualization. The image detection algorithm then uses the back-propagated images still in the frequency domain.

4.4.3 Linear filtering

The method is is implemented to highlight the centers of individual micro-particles. We are doing that by filtering the back-propagated channel with a filter of similar size to a micro-particle. On the back-propagated image, the micro-particles are much darker than their surrounding area. Therefore, we can set the center area of the filter to be negative and a small ring around it to be positive. This way the bright environment surrounding the particles will be filtered to negative values, while the dark particles will be filtered to positive values. Such a filter is visualized in figure 4.6.

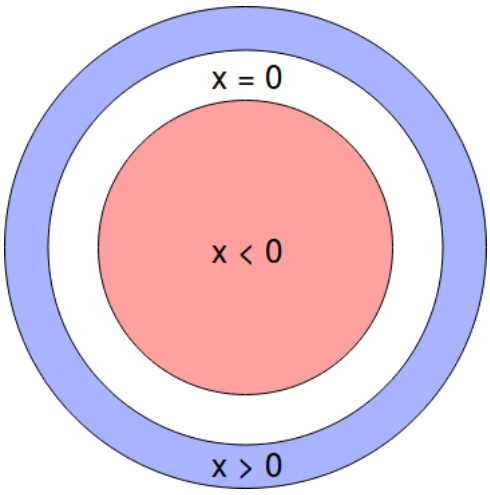


Figure 4.6: Visualization of the proposed filter. Values of x in the picture represent the values of pixels set in the individual rings. The outer ring has positive values of pixels, the middle ring has pixels set to zero and the inner circle has negative values of pixels.

For the filtering itself we are utilizing 2D convolution in the frequency domain as discussed in section 3.2.1. That can be reasoned by two points.

1. Time complexity does not depend on the size of the utilized filter. This is particularly important since we are using large filters.

2. We do not have to run another FFT operations as we can simply utilize the intermediate results of the back-propagation.

Before the filtering itself, the filter has to be expanded to the size of the back-propagated channel. That is done by shifting the original filter cyclically, so that the central element of the kernel is in the top left corner of the new array [14]. That is visualized in 4.7.

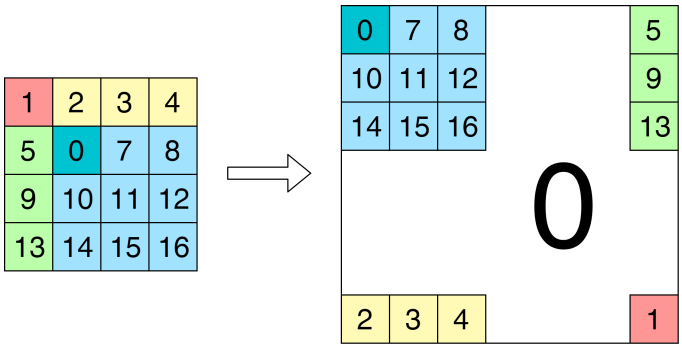


Figure 4.7: Visualization of cyclical shifting of a filter used for 2D convolution in the frequency domain.

The filtering itself is then implemented from the expression 3.5. It is run after the third step of the back-propagation algorithm to skip redundant FFT calculations.

4.4.4 Position estimation and tracking

Position estimation implemented here is based on the simple prediction method proposed in 3.2.2. Since it is reasonable to expect at most a couple tens of observable particles, this part of the process was moved to the main manipulation computer. That is enforced by the fact that the process of tracking individual particles is not easily parallelizable.

The micro-particle tracking runs in 3 steps.

1. Clean the received positions.

   The object detection algorithm sometimes sends two very close possible results for one micro-particle. That can be caused by a slightly different shape of the micro-particle, or its close environment. However, this can be fixed by taking the average between these two values and taking it for the actual position.

2. Find the corresponding particle in the red channel.

   The red channel is shifted in respect to the green channel, so that the corresponding images of the same particle have on average the same position in both channels. Thus, finding which micro-particle in the red channel corresponds to which in the green channel can be figured out very easily. The positions closest to each other between channels form the corresponding pair. This method will work even when the particles will be able to change their axial distance as at most the difference in shift will be in a couple tens of pixels.

3. Number the individual particles.

   When the first set of positions is received, it is numbered and stored for later set. When the next set arrives and is processed by the first two steps, we compare the new set of positions to the last one. For each numbered micro-particle we look for its new position by looking for the closest position in the new set in a small radius around the last position. If such a position is found, it updates the last position and takes its number. If not, the position remains unchanged. If a new position is not found for a number of frames, the corresponding micro-particle is considered to be lost.

Chapter 5

Experiments

In this chapter, we will focus on experimentally finding values of several variables used in the implementation: back-propagation distances of both channels and shapes of the filters used in linear filtering. After that, we will test the implemented method on an environment of unmoving micro-particles and discuss the results. Finally, we will propose experiments that should take place in the future.

5.1 Back-propagation distances

Back-propagation itself is a process, in which we can simulate actual back-propagation of interference patterns to a plane in a set distance z from the image sensor. That distance is then what we call the back-propagation distance. Setting this distance to a specific value could then allow us to see a picture of the micro-particle itself. In a homogeneous environment, this value would be equal to the actual distance of the particle from the sensor.

In the final experimental setup, micro-particles can move freely in a pool of water. That means their axial distance will not be constant as it is with the setup used for the current implementation. Thus, we are looking for such a value of z, that would return comprehensible results for micro-particles at all accessible axial distances.

Setting z to such a value would take care of two things beneficial in our implementation:

- Create an image comprehensible for a human observer.
- Define shapes of the observed micro-particles to allow for easier object detection.

5.1.1 Green channel

Non back-propagated picture of micro-particle in the green channel is pictured on figure 5.1. From this picture it is possible to deduce only very little about the shape of pictured objects, let alone about their exact position.

Starting from an original value of back-propagation distance in the green channel $z_g = 2100\,\mu\text{m}$ we increased the value by the same increment until we

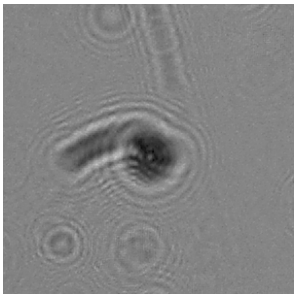
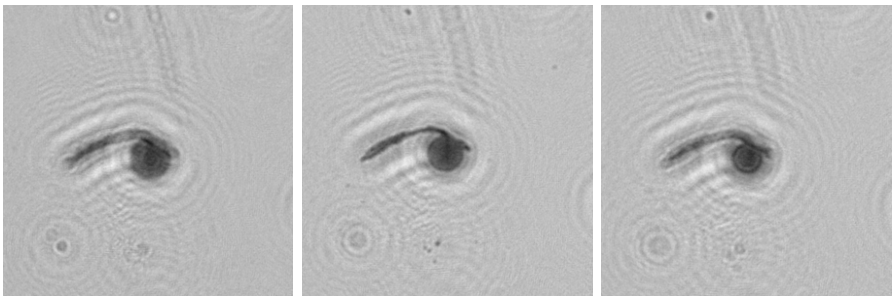


Figure 5.1: Non back-propagated picture in the green channel of a polystyrene micro-particle next to a dust particle.

found the desired distance of 2750 µm. The same micro-particle from figure 5.1 is visualized after back-propagating to different distances in figure 5.2. In that figure we can see that the selected distance of 2750 µm results in the most well defined pictures of the observed micro-particle.

Yet another comparison between different values of z_g can be seen in the case of a picture of 5 grouped micro-particles. Comparison of that situation in three different back-propagation distances is visualized in figure 5.3. The image back-propagated with $z_g = 2750\,\mu\text{m}$ has the most well defined shapes of the observed micro-particles.



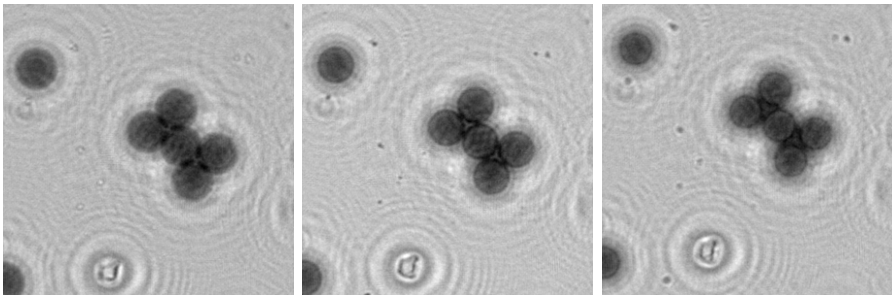
(a) : $z = 2.1\,\text{mm}$ **(b) :** $z = 2.75\,\text{mm}$ **(c) :** $z = 3.1\,\text{mm}$

Figure 5.2: Micro-particle next to a dust particle reconstructed in different back-propagation distances. All pictures taken from the green channel.

5.1.2 Red channel

The value of z_r chosen for the red channel differs from the one for the green channel. That is because the red light falls down at the image sensor at an angle of 30° from the the vertical. This causes the interference patterns to be skewed. The used back-propagation method therefore cannot correctly visualize the observed micro-particles. That is not a large problem, as we do not have the need to display the red channel anywhere. On the other hand it might make it harder to implicitly design a filter for linear filtering of the red channel.

An example of a skewed interference pattern in the red channel is captured in



(a) : $z = 2.5\,\mathrm{mm}$ **(b) :** $z = 2.75\,\mathrm{mm}$ **(c) :** $z = 2.9\,\mathrm{mm}$

Figure 5.3: An image of 5 micro-particles grouped together in 3 different back-propagation distances.

figure 5.4. The non back-propagated image is actually more incomprehensible in the red channel than it was in the green channel.

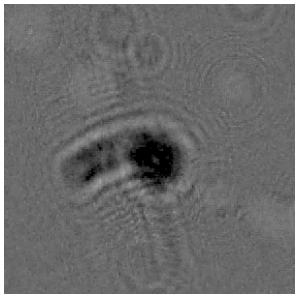


Figure 5.4: Non back-propagated picture in the red channel of a polystyrene particle next to a dust particle.

This time we found the desired value of $z_r = 3.1\,\mathrm{mm}$. The process of acquiring this value is the same as it was with the green channel. As is visible in figure 5.5, back-propagating the interference patterns to the selected distance does not result in well defined reconstructions. However, these reconstructions are the most fitted for linear filtering with a filter of the shape proposed in section 4.4.3.

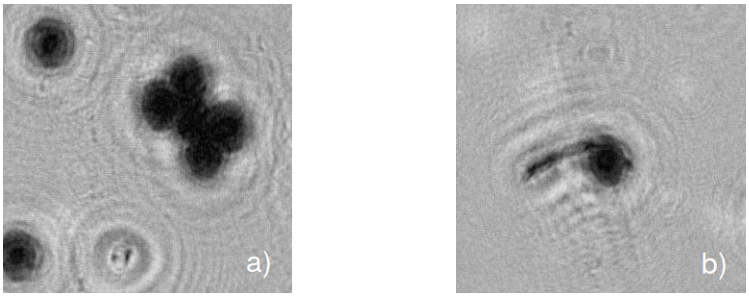


Figure 5.5: Two images in the red channel back-propagated to the selected distance of $z = 3.1\,\mu\mathrm{m}$. a) shows a picture of 5 grouped micro-particles. b) shows a picture of a micro-particle next to a dust particle.

5.1.3 Issues

The obvious problem with the selected values of z_r and z_g comes from the fact that the used experimental setup is bound to change after adding both the electrode array and the pool of water. When that happens, these values have to be selected again for the new environment.

5.2 Object detection

After setting the used back-propagation distances it is time to find the appropriate filters. The selected shape of the filters used in our implementation was proposed in section 4.4.3. These filters will then be tested on the observable environment and the processed results will then be discussed.

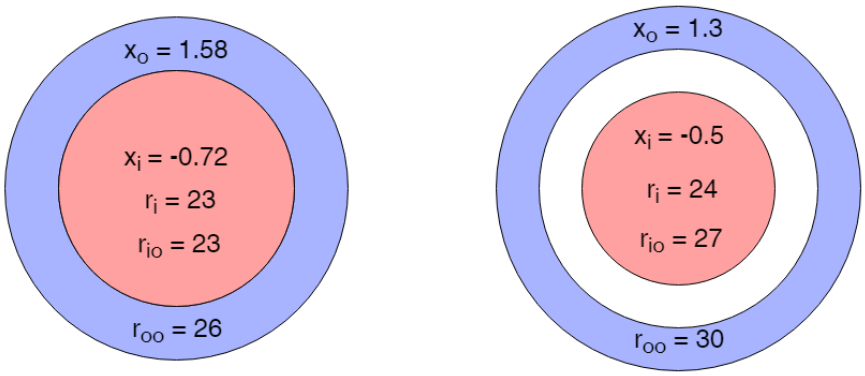
5.2.1 Selecting filters

Finding the individual filters is based on a simple method. First we set the display thread in our implementation to display the back-propagated image of a selected channel. At the same time this image is filtered by the tested filter and then in the resulting image we find local maximums. The resulting map is then projected on top of the currently displayed image so that we see the positions predicted by the chosen filter.

In the filter of the proposed shape we can change 5 different values to affect the results of the filtering. These values are defined in figure 5.6

- Radius r_i of the negative inner circle.
- Inner and outer radii of the outer positive ring. These are represented by r_{io} and r_{oo} respectively.
- Value x_i set to pixels in the inner negative circle.
- Value x_o set to pixels in the outer positive ring.

The radii are then set so that the inner circle is just barely larger than the projection of the observed micro-particle. The other values are then tinkered with until we see favorable results. This way we found working filters for both the green and red channel. Their visualization along with all set values is on figure 5.6.



(a) : Green filter **(b) :** Red filter

Figure 5.6: Visualization of filters used for linear filtering in both channels.

5.2.2 Testing

Now that we have the filters set up it is time to see how they fare on the observed environment. In the observable environment, we have found a couple of sample situations that we need the filters to be able to process correctly.

- Micro-particles are by themselves in an empty space.
- Micro-particles are grouped closely together.
- Dust particle is covering part of the micro-particle.
- Presence of dust particle that has a slightly similar shape to that of an observer micro-particle.

Green channel

When run through the sample situations, the green filter gives results visible in figure 5.7. As it is possible to see, in almost all of the expected samples it is successful. That is with the exception of the sample situation with grouped micro-particles. This issue was already predicted in section 3.2.2 and might not be possible to solve with linear filtering.

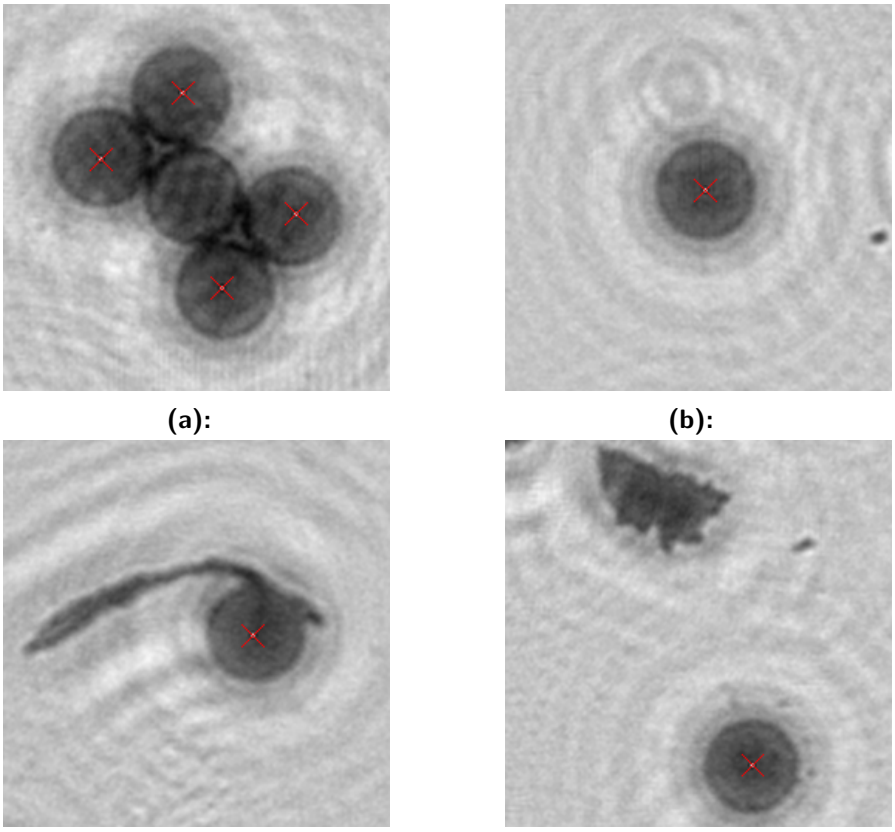


Figure 5.7: Pictures of sample situations in the green channel with found positions projected on top in the form of red crosses.

Red channel

When run through the samples, the red channel gives results visible in figure 5.8. Here we can see that the results are very far from perfect. That is however not a problem in the red channel since as long as centers of the particles are highlighted, the position estimation algorithm should correctly pair the positions found on the green channel with the ones on the red channel. In addition, the extra positions found in the group sample have much lower values in the map of maximums than the actual centers of observed micro-particles. As such it should be fairly simple to filter out these extra values. Even when taking in mind the changing axial distance, the projections in the red channel will move at most by a couple tens of pixels. This should still allow us to pair the centers quite accurately.

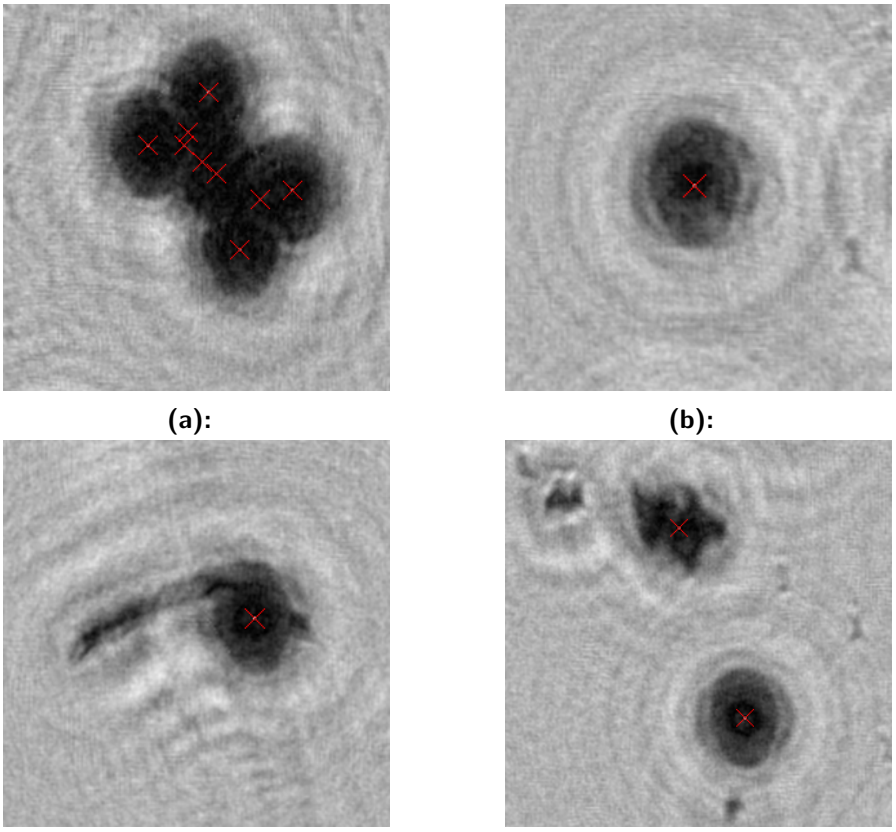


Figure 5.8: Pictures of sample situations in the red channel with found positions projected on top in the form of red crosses.

5.3 Time efficiency

One of the main goals of our implementation was to achieve higher calculation speeds while being more accurate than the original algorithm. The original algorithm ran at approximately 10 Hz, while processing frames the size of only 400×400 pixels [8].

In our implementation the default size of a processed frame is 1024×1024 pixels. The values for time measurements used in this part are taken by timing a large number of cycles of the main processing loop thread and averaging the result. The measurements were taken with the displaying thread turned both on and off as it will probably have the highest impact on the time efficiency. That is due to copying the whole processed frame from GPU memory to CPU memory.

From this we have found out that with the displaying thread turned off, a single cycle of the main processing thread takes on average 38.6 ms. This corresponds to roughly 26 FPS. Such a result is definitely an improvement.

On the other hand, a single cycle of the main processing thread with the displaying thread turned on takes on average 44.2 ms. This corresponds to

roughly 23 FPS. While still an improvement, the impact of memory copying is significant.

5.4 Future experiments

During the implementation, the experimental lacked—due to manufacturing problems—the electrode array which would allow us to test our object tracking algorithm on actually moving particles. Unfortunately, because of this problem we did not conduct any relevant experiments.

In the near future, the experimental platform will be completed and, thus, we will be able to track actually moving particles.

In addition, all micro-particles currently are all stuck to the same axial distance and thus any estimation of it is pointless. However, when the electrode array is added to the setup, we will, suddenly, be able to change the axial distances of individual particles. Then we will be able to experimentally find the proportion of the shift between the positions in the red and green channel to the axial distance of a particle.

Chapter 6

Conclusion

In this thesis we implemented a system for estimating 3D positions of micro-particles by the use of twin-beams illumination method and DHM.

First, we split the color channels produced by the twin-beams method by converting to the RGB format and operated on both of the channels separately. Then we had to find a suitable algorithm for reconstructing captured interference patterns. For this we chose the Rayleigh-Sommerfeld back-propagation method. Later, we had to find an algorithm for detecting the positions of micro-particles in the reconstructed images. For this we selected linear filtering.

After filtering and thus finding the center points of all projections, we were only one step away from finding the actual 3D positions of the observed micro-particles. If the particles could change their axial distance in the current experimental setup, we could find the proportion between the shift of positions of each micro-particle in both color channels. In the current version of the experimental setup that however was not possible.

For object tracking we chose a simple method of checking for the detected position, the closest to the old position.

After choosing the methods we went on to implement said methods in CUDA and deployed them on an embedded computer, Jetson TX2 developed by NVIDIA. The implemented process is then able to communicate with a main manipulation computer which runs Matlab.

Finally, we experimentally set the values for both the filters utilized in linear filtering and the back-propagation distances for the image reconstruction algorithm. Then we discussed the results. After that we measured the speed of the implemented process and compared it to the original method. From this we found out that the implemented method runs at 2.5 times the FPS of the original method, while also processing 6.5 times as many pixels.

Appendix A

Bibliography

[1] Xiao Yu, Jisoo Hong, Changgeng Liu, and Myung K. Kim. Review of digital holographic microscopy for three-dimensional profiling and tracking. *Optical Engineering*, 53(11):112306, 2014.

[2] J A Guerrero-Viramontes, D Moreno-Hernández, F Mendoza-Santoyo, and M Funes-Gallanzi. 3d particle positioning from CCD images using the generalized Lorenz–Mie and Huygens–Fresnel theories. *Measurement Science and Technology*, 17(8):2328–2334, August 2006.

[3] Sang-Hyuk Lee and David G. Grier. Holographic microscopy of holographically trapped three-dimensional structures. *Optics Express*, 15(4):1505, February 2007.

[4] Frank Dubois, Cédric Schockaert, Natcaha Callens, and Catherine Yourassowsky. Focus plane detection criteria in digital holography microscopy by amplitude analysis. *Optics Express*, 14(13):5895, 2006.

[5] Ting-Wei Su, Serhan O. Isikman, Waheb Bishara, Derek Tseng, Anthony Erlinger, and Aydogan Ozcan. Multi-angle lensless digital holography for depth resolved imaging on a chip. *Optics Express*, 18(9):9690, April 2010.

[6] Pasquale Memmolo, Andrea Finizio, Melania Paturzo, Lisa Miccio, and Pietro Ferraro. Twin-beams digital holography for 3d tracking and quantitative phase-contrast microscopy in microfluidics. *Optics Express*, 19(25):25833, December 2011.

[7] Martin Gurtner. *Real-time Optimization-based Control and Estimation for Dielectrophoretic Micromanipulation.* Diploma thesis, Czech Technical University in Prague, Prague, January 2016.

[8] Martin Gurtner and Jiří Zemánek. Twin-beam real-time position estimation of micro-objects in 3d. *Measurement Science and Technology*, 27:127003, 12 2016.

[9] Steven W. Smith. *The scientist and engineer's guide to digital signal processing.* California Technical Pub, San Diego, Calif, 1st ed edition, 1997.

[10] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J, 2nd ed edition, 2002.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[12] Greg Welch and Gary Bishop. An introduction to the kalman filter. *Proc. Siggraph Course*, 8, 01 2006.

[13] Mounika Gudipati. *Application of Kalman filter to estimate position of a mobile node in indoor environments*. Diploma thesis, University of Akron, May 2017.

[14] Victor Podlozhnyuk. Fft-based 2d convolution. *NVIDIA WhitePaper*, 07 2007.