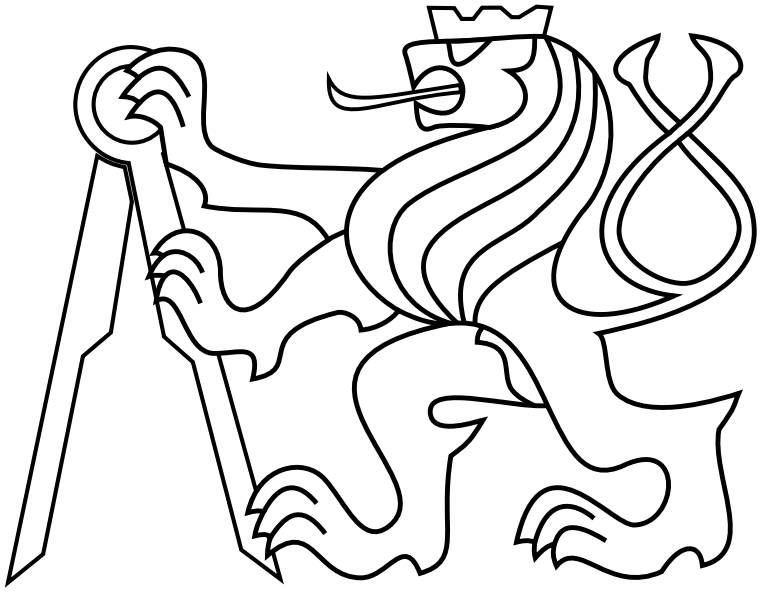
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Dan Šuster

Visual tracking and trajectory optimization for dielectrophoretic manipulation

Department of Control Engineering

Thesis supervisor: **Ing. Martin Gurtner**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne .....

.....
Podpis autora práce

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Šuster Dan** Personal ID number: **457158**

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

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

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Visual tracking and trajectory optimization for dielectrophoretic manipulation

Master's thesis title in Czech:

Vizuální sledování a optimalizace trajektorií pro dielektroforetickou manipulaci

Guidelines:

1) Develop a visual tracker based on convolutional neural networks for dielectrophoretic manipulation platform in the twin-beam setup. The tracker must be capable of processing at least 20 frames per second and tracking at least two arbitrary objects when running on NVIDIA Jetson AGX Xavier development kit.
2) Specifically for spherical objects, implement an algorithm making more accurate the coarse positions found by the tracker.
3) Use the more accurate positions to estimate 3D positions of the objects.
4) Formulate and solve the optimization problem of getting an object from a point A to a point B in the manipulation area as quickly as possible while taking into account the position-dependent maximum achievable dielectrophoretic force.

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] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in European conference on computer vision, 2016, pp. 850–865.
[3] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in European Conference on Computer Vision, 2016, pp. 749–765.
[4] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," SIAM Review, vol. 59, no. 4, pp. 849–904, 2017.

Name and workplace of master's thesis supervisor:

Ing. Martin Gurtner, Department of Control Engineering, FEE

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

Date of master's thesis assignment: **13.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Martin Gurtner	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

Acknowledgements

I would like to thank my supervisor Ing. Martin Gurtner for his help and guidance throughout the thesis. I have very much enjoyed working together because he is a brilliant and kind person. Furthermore, I would like to thank my family for their love and continuous support throughout my studies.

Abstract

This thesis deals with two crucial problems in the task of feedback dielectrophoretic (DEP) micromanipulation: real-time visual tracking and trajectory optimization. Real-time visual tracking of manipulated objects is of pivotal importance as the feedback control system must know the positions of manipulated objects. Two basic image processing methods with a simple tracking algorithm are proposed and tested. These turn out to be insufficient, and thus a state-of-the-art tracker based on convolutional neural networks is employed and experimentally verified. The tracker is modified so that it can track up to 4 objects in 30 frames per second. Trajectory optimization for DEP manipulation is specific in having position-dependent limits for the maximum achievable DEP forces, which makes the trajectory optimization particularly tricky. Based on the dynamic model of a spherical particle, a simplified solution consisting of a pre-computed map of maximum achievable forces and an optimal path-finding algorithm is proposed and verified by conducted experiments.

Keywords: visual tracking, convolutional neural networks, trajectory optimization, dielectrophoresis, micromanipulation

Abstrakt

Tato práce se zabývá dvěma stěžejními tématy související s mikromanipulací pomocí dielektroforézy (DEP): vizuálním sledování objektů v reálném čase a časovou optimalizací trajektorií. Vizuální sledování objektů je kritickou součástí systému manipulace, jelikož potřebujeme znát polohu manipulovaného objektu pro zpětnovazební řízení. Pro tuto úlohu byly použity dvě známé metody z oboru zpracování obrazu v kombinaci s jednoduchým sledovacím algoritmem. Tyto metody se ukázaly jako nedostačující, a proto byl využit sledovací algoritmus založený na konvolučních neuronových sítích. Původní sledovací algoritmus byl upraven na základně zjednodušujících podmínek tak, aby dokázal sledovat až čtyři objekty ve 30 snímcích za sekundu. Úloha optimalizace trajektorií je v našem případě specifická tím, že maximální dosažitelná DEP síla pro manipulaci je závislá na poloze, což děla tuto úlohu těžce řešitelnou. Na základě dynamického modelu pro kulové částice je navrženo zjednodušené řešení, které spočívá v předpočítání maximálních dosažitelných sil v manipulačním poli. Na tomto poli lze poté vygenerovat optimální trajektorii pomocí algoritmu pro hledání optimálních cest v grafu. Všechny metody v této práci byly experimentálně ověřeny na reálném zařízení pro mikromanipulaci.

Contents

List of Figures

Part I

Introduction

This thesis deals with visual tracking and trajectory optimization in the task of *dielectrophoretic* (DEP) micromanipulation. DEP phenomenon is described as the movement of a dielectric particle when it is subjected to a non-uniform electric field. It is mostly exploited for contact-less micromanipulation of particles, with applications in nanotechnology and biology [1, 2, 3]. This work continues the long-term research in DEP contact-less micromanipulation conducted by the group *Advanced Algorithms for Control and Communications* (AA4CC), Department of Control Engineering, Faculty of Electrical Engineering at Czech Technical University in Prague.

The members of AA4CC group have already developed a working hardware setup for DEP manipulation, which is illustrated in Fig. 1. We give a detailed description of the setup later, but for now, let us briefly describe the very principle of the manipulation. The manipulated particles are manipulated by the application of varying potentials on the electrodes, which are computed by a phase-shifting feedback controller [4, 5]. Since there is no physical way of measuring the exact position of the particles, a position estimation method based on processing images from a camera was proposed in [5]. One of the aims of this thesis is to propose a method for visual object tracking in a video, which could provide—in combination with the mentioned position estimation method—a position estimate for the feedback controller, and thus enable precise manipulation of the particles. Such tracker should be able to track objects of general shape, and it should run in real-time.

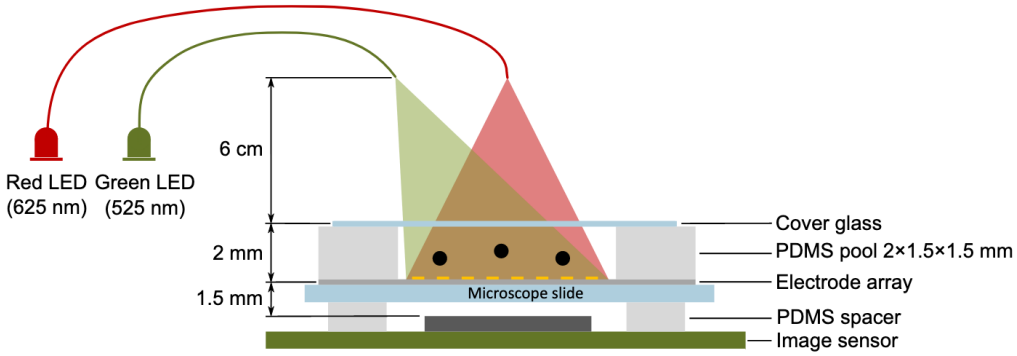


Figure 1: Illustration of the hardware setup (*reprinted from [6]*).

Another aim of the thesis is to generate time-optimal trajectories when moving the objects from a point A to a point B while taking into account the position-dependent maximum achievable DEP force, for which we do not have an analytical solution.

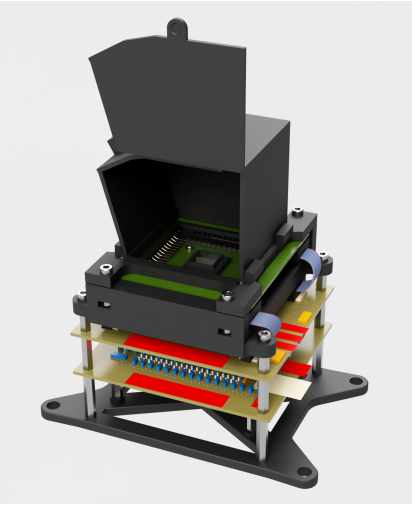
This thesis is divided into three parts. First, we describe our context for DEP manipulation and the hardware setup in Section 1. The second part deals with visual object tracking. We describe two basic methods in image processing and combine them with a simple tracker in Section 2, and later, we explore a state-of-the-art tracker based on convolutional neural networks (CNNs) in Section 3. Finally, we deal with trajectory optimization for DEP manipulation (Section 4), while taking into account the position-dependent maximum achievable forces. In addition, based on our results, we propose improvements to the control algorithm in Section 5.

1 Preliminaries

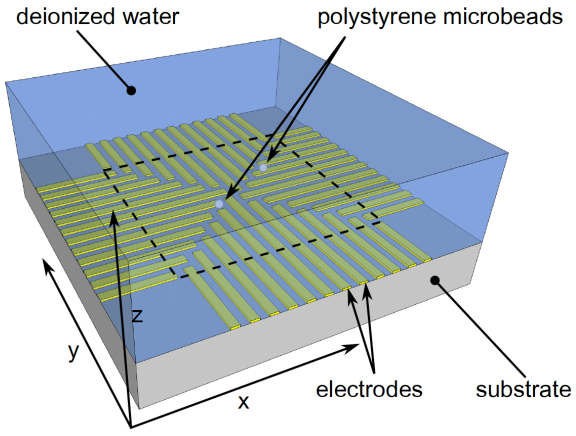
In this section, the hardware setup used throughout the thesis is described, along with definitions of the coordinate system and the current state of the DEP controller, which is employed for manipulation. Furthermore, we summarize twin-beam method for 3D position estimation in Section 1.3. Finally, *backpropagation* method for image reconstruction is briefly presented in Section 1.5. All of the topics discussed within this section are based on previous works of the members of AA4CC group, and we give a very simplified description to get a basic idea of the topics relevant to this thesis.

1.1 Hardware setup and coordinate system

The hardware setup for DEP manipulation is illustrated in Fig. 1 and a 3D rendered model is displayed in Fig. 2a. The device for manipulation consists of 5 main parts. The first one is an electrode array formed by 56 electrodes with a manipulation area approximately $1.5 \times 1.5\,\text{mm}$. The electrodes are $50\,\mu\text{m}$ wide, with the inner gap between them also $50\,\mu\text{m}$. Above the array, there is a $2\,\text{mm}$ high reservoir filled with de-ionized water, in which the particles are being manipulated. Beneath the array, there is a lens-less image sensor. Above the reservoir, there are 2 light-emitting diodes (LEDs), which are used for 2D or 3D position estimation based on processing images captured by the camera chip. Finally, the whole device is closed in a box, shielding it from the surrounding light.



(a) 3D model of the manipulation device.



(b) Electrode array layout (*reprinted from [4]*).

Figure 2: The hardware setup used throughout this thesis.

A novel electrode layout for DEP manipulation was proposed in [4] and it is visualized in Fig. 2b. Throughout this thesis, we use the identical coordinate system orientation, but with the origin of the system located in the center of the array. Therefore, our manipulation area lies in the range of $[-750, 750]$ µm in both x and y axes. Likewise, images from the camera are oriented such that the top and right directions are positive in y and x, respectively. The transformation between the image coordinates (pixels) and 2D system coordinates is given by a simple augmented transformation matrix $\mathbf{H}_{r/g} \in \mathbb{R}^{3\times3}$, which is obtained by the calibration process, described in [5].

1.2 Phase-shift control

In DEP manipulation, movement of a dielectric particle is caused by a non-uniform electric field. The electric field is shaped by the application of varying potentials on the electrodes in the array. The method proposed in [4] uses constant amplitude and frequency, while shifting the phases of the signals which are applied to the electrodes. The issue is that only the forward model is known, meaning that it is possible to compute the resulting force when the phases are known, but not the other way around. As a result of the previous works [4, 5, 7], we are able to specify the desired force as an input to the system and solve the inverse problem numerically in real-time. The optimization problem is non-convex, and it is stated as

$$\underset{\theta_1,\theta_2,\ldots,\theta_n}{\text{minimize}} \quad ||\mathbf{F}_{\text{dep}}(\theta) - \mathbf{F}_{\text{des}}||\,, \tag{1}$$

where $\mathbf{F}_{\text{des}}$ is desired force and θ_i are the phases of the voltage signals, which are applied to the electrodes. The DEP model is given as

$$\mathbf{F}_{\text{dep}}(\theta) = \begin{bmatrix} \mathbf{c}^{\text{T}}\boldsymbol{\Psi}_x\mathbf{c} + \mathbf{s}^{\text{T}}\boldsymbol{\Psi}_x\mathbf{s} + \mathbf{c}^{\text{T}}\boldsymbol{\Omega}_x\mathbf{c} \\ \mathbf{c}^{\text{T}}\boldsymbol{\Psi}_y\mathbf{c} + \mathbf{s}^{\text{T}}\boldsymbol{\Psi}_y\mathbf{s} + \mathbf{c}^{\text{T}}\boldsymbol{\Omega}_y\mathbf{c} \\ \mathbf{c}^{\text{T}}\boldsymbol{\Psi}_z\mathbf{c} + \mathbf{s}^{\text{T}}\boldsymbol{\Psi}_z\mathbf{s} + \mathbf{c}^{\text{T}}\boldsymbol{\Omega}_z\mathbf{c} \end{bmatrix}, \tag{2}$$

where $\mathbf{s}$ and $\mathbf{c}$ represent sines and cosines of the phases, and $\boldsymbol{\Psi}$ and $\boldsymbol{\Omega}$ are matrices that depend on the position of the object. Mentioned model is derived especially for spherical particles, and here it is stated only in a simple form for illustration. The desired force is currently given by a simple proportional regulator as

$$\mathbf{F}_{\text{des}} = k\,(\mathbf{p}_{\text{des}} - \mathbf{p}), \tag{3}$$

where $\mathbf{p}_{\text{des}}$ is the desired position, $\mathbf{p}$ is the actual position, and k is the proportional constant.

1.3 Twin-beam setup

A novel method for 3D position estimation was proposed in [5], which is based on triangulation. An intuitive sketch is shown in Fig. 3. The particles are illuminated by two light sources under two different angles. When a particle is illuminated by two sources, it also casts two shadows on the image sensor. As it is stated in the mentioned work, the height (axial distance) of the particle corresponds to the lateral shifts between the two shadows. The height is derived as

$$h = l \tan \beta, \tag{4}$$

where l is the lateral shift distance in electrode coordinate system, and β is the angle of the oblique light source in the water stated as

$$\beta = \sin^{-1}\left(\sin \frac{\mathrm{n_{air}}}{\mathrm{n_{water}}} \alpha\right), \tag{5}$$

where α is the angle under which the oblique source emits (in our case 30°), and $\mathrm{n_{air}}$ and $\mathrm{n_{water}}$ are the refractive indices of air and water, respectively.

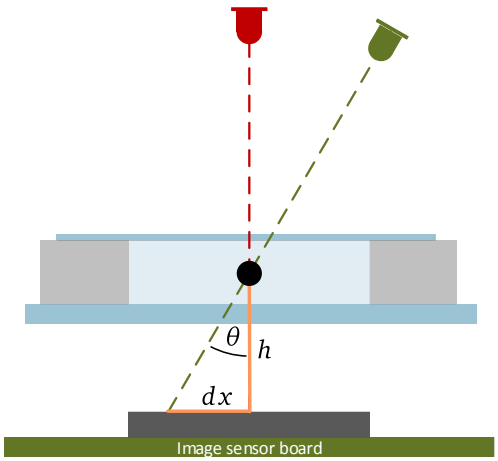


Figure 3: An illustration of the basic principle of the twin-beams method. (*reprinted from [5]*).

1.4 NVIDIA Jetson AGX Xavier

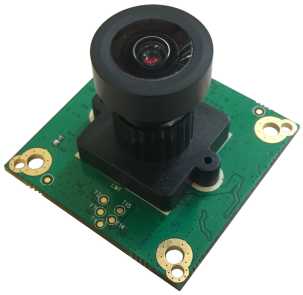
The whole manipulation device is connected to NVIDIA Jetson AGX Xavier embedded computer. Xavier is a small – yet powerful – device designed for deep learning and image processing. An illustrative photo is shown in Fig. 4. Hardware specification is as follows: octal-core NVIDIA ARM CPU, 16GB RAM shared with GPU, 512-core Volta GPU with 64 Tensor Cores, and *camera serial interface* for connecting up to 6 4K cameras. The system runs on modified Ubuntu by NVIDIA with packages for video processing and deep

learning – such as CUDA Toolkit, CUDNN, and PyTorch – provided by NVIDIA JetPack. All of the experiments in this thesis run on Xavier.

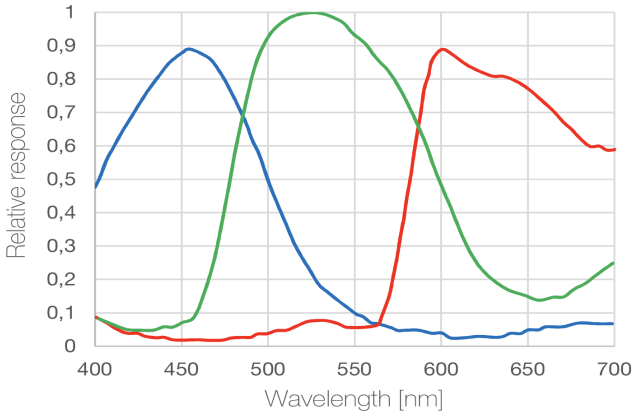
Figure 4: Illustration photo of the NVIDIA Jetson AGX Xavier[1].

1.4.1 Camera

The camera used within the DEP manipulation setup is LI-IMX477-MIPI-M12. It is a 4K camera with 1.55 µm pixel size. The camera is used without lenses, and since our manipulation area is 1.5 × 1.5 mm, we work with images of size 1024 × 1024 pixels, which approximately correspond to the size of manipulation area. Fig. 5 shows an illustrative photo of the camera and the sensitivity profiles. LEDs employed in the twin-beam method were selected based on these profiles, and their wavelengths are 525nm and 625nm for the green and the red channel, respectively. More details about the setup and reading of the images from the camera are described in [8].



(a) Illustrative photo of the IMX477 4k camera module. Our setup does not use the lenses.



(b) Color sensitivity profiles of the camera.

Figure 5: Camera model LI-IMX477.[2].

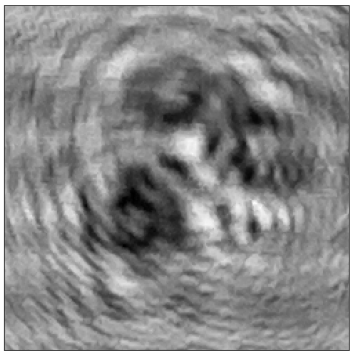
[1]Source: `https://elinux.org/Jetson_AGX_Xavier`

1.5 Image backpropagation

Images from the camera do not capture the objects *per se*, but rather interference patters, formed by illumination from a coherent light source. A selected part of the image with interference patterns is shown in Fig. 6a. Backpropagation is a technique for image reconstruction from the interference patterns, and it comes from the field called *digital holographic microscopy*.

In principle, the method simulates backpropagation of the captured light field. And in practice, this method is computed as a convolution between the image and the Rayleigh-Sommerfeld propagator. The convolution is implemented by taking the Fourier transform of the image, multiplying it with the Fourier image of the propagator, and finally taking the inverse Fourier transform to obtain the reconstructed image. Computation of the Fourier transform is possible in real-time since efficient GPU implementations are available. You can see that Fig. 6b—in contrast to Fig. 6a—shows clearly a spherical microparticle with a piece of dust. This method is very useful since it enables us to see microparticles clearly without the need of a microscope.

We have given – again – a very simplified description to get the basic idea of the backpropagation technique, which we exploit in the visual tracking part of this thesis. For further details, this method is thoroughly described in [5, 8].



(a) Original image.

(b) Backpropagated image.

Figure 6: Reconstruction of the image by applying backpropagation (*reprinted from [5]*).

[2]Source: `https://www.leopardimaging.com/uploads/LI-IMX477-MIPI-M12_datasheet.pdf`

Part II

Visual tracking

Visual object tracking is a very active research area with a lot of attention over the last decades. Especially with the rise of convolutional neural networks (CNNs) in the image recognition and object detection fields [9, 10, 11], a lot of researchers have been focusing on exploiting CNNs for visual tracking. In image recognition, the primary focus is on learning the visual representation of given classes, but in the general object tracking, we are interested in learning the similarity between objects within consecutive frames. It is usually a very challenging task to apply a general tracker for multiple domains, because of varying light conditions and backgrounds.

Online tracking algorithms based on CNNs usually tackle this issue by utilizing well-known networks such as Alexnet [11] or VGG-16 [9] for feature extraction and training additional layers to adapt to a specific target object [12, 13]. For example, MDNet [14] is pre-trained on a domain-independent training set and trains the last 3 fully connected (FC) layers during the tracking, with 2 of the 3 FC layers starting pre-trained and the last one being replaced with random initialization on every start.

Offline tracking algorithms work by learning general similarities between features extracted by the CNNs, and even though they still have an accuracy gap compared to the online ones, they are gaining attention because of their efficiency in terms of speed [15]. They do not - compared to the online trackers - require stochastic gradient descent updates, which are computationally demanding.

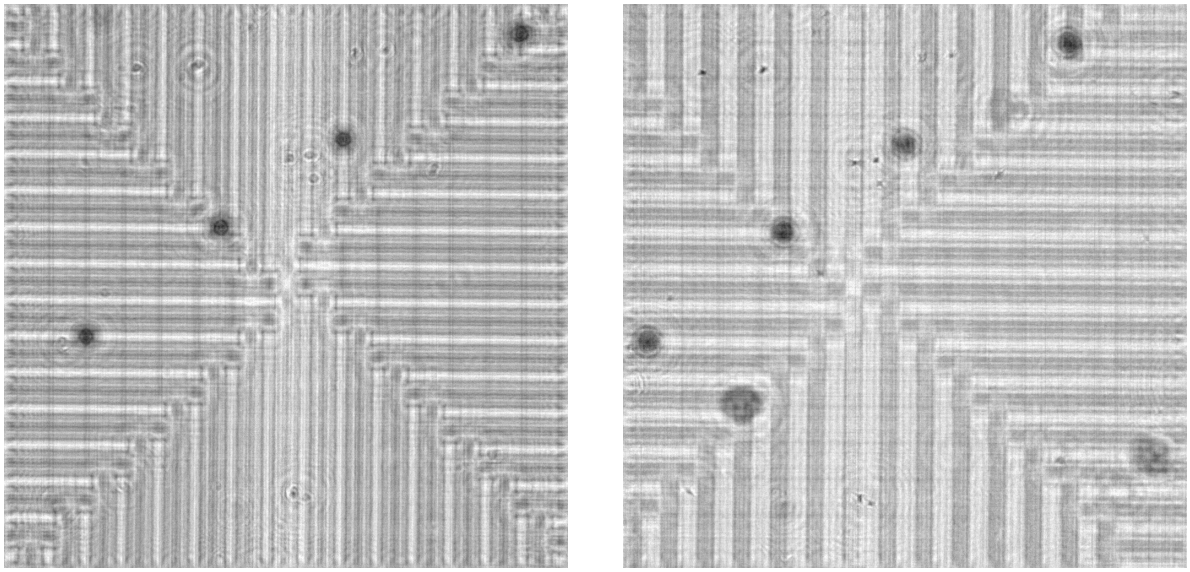
Lately, there has been done a lot of interesting research in object detection visual tracking exploiting Siamese networks. Siamese networks are neural networks that share the same weights while processing different input vectors and output a comparable result. With CNNs utilized as feature extractors, deep similarity learning methods with Siamese networks were on the rise. A well-known example of such networks is Deepface [16], which is used for face recognition. Siamese CNNs in visual tracking are usually used to generate feature volumes of a target template and a search region, and then apply cross-correlation to obtain a similarity map [17, 15].

Visual tracking has a vast area of applications, such as surrounding object detection for cars [18], target tracking and relative localization in autonomous robotics [19, 20], simultaneous localization and mapping for augmented reality [21], or in sports for ball tracking in soccer [22].

One of the aims of this thesis is to develop a tracking algorithm that is able to track two objects simultaneously in at least 20 frames per second (FPS) and thus provide position feedback for DEP force controller. We are given a video stream of green and red channels

from the camera, and we should ideally be able to track objects of general shapes, but with main focus on spherical objects. We can choose between the raw or the *backpropagated* (Sec 1.5) images. An example of our typical setup for both green and red channels is shown in Fig. 7.

We chose to approach our problem by firstly utilizing two very well known image processing methods in combination with custom, yet simple, tracking algorithm (Section 2). And later, in Section 3, we explore a more robust solution based on Siamese networks. Bertinetto *et al.* [15] proposed SiamFC tracker that achieves competitive results in video object tracking benchmarks even with offline-trained networks. Furthermore, SiamFC is fast and well-suited for real-time applications. These are exactly the reasons why we decided to use this particular tracker.



(a) Green channel. **(b)** Red channel.

Figure 7: Examples of green and red channel images of our environment for tacking. Both of the raw images from the camera were filtered using *backpropagation*, mentioned in Section 1.5. The dark spherical objects represent our targets.

2 Basic methods

In this section, we divide the general visual tracking task into two separate parts - object detection and object tracking. In the first part, we present two of the basic methods from image processing with which we use for object detection. The first method consists of image segmentation using thresholding and morphology operations (Section 2.1) and the second method lies in finding regional minima (Section 2.2). In the second part, we present a simple tracking algorithm to work with object detections in Section 2.3.

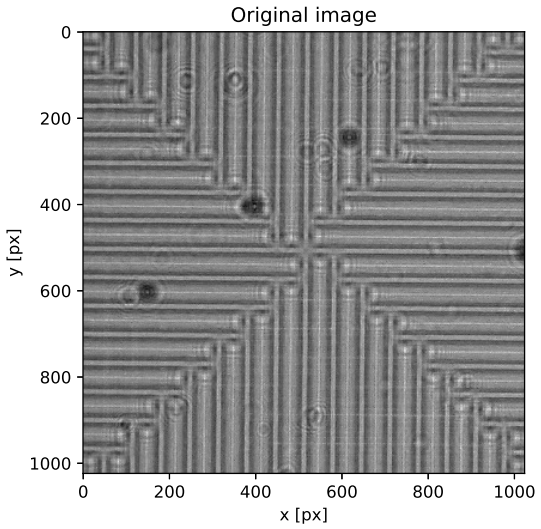
2.1 Image segmentation

In the case of image segmentation, our goal is to divide the objects (low intensity) from the background (higher intensity), but there are also the electrodes which have a similar intensity as the objects. Fortunately, we approximately know the shape and size of target objects; therefore, we can assume that the objects are bigger than the low-intensity parts of the electrodes.

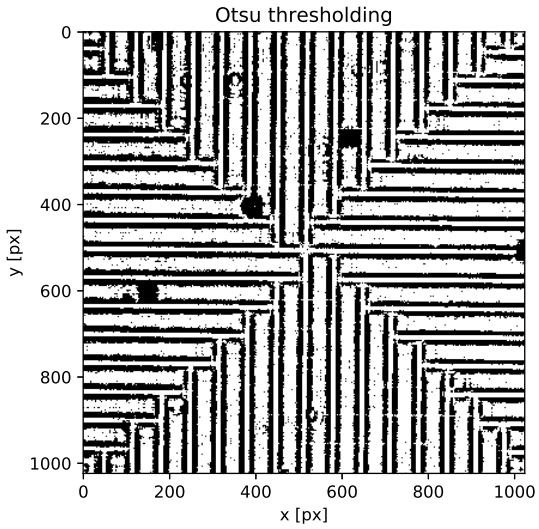
The segmentation process example is shown in Fig. 8. First, we threshold the image using Otsu's method, which minimizes the variance between classes (in our case binary). In the binary image, there are the target objects and the thin electrode lines merged into one class (8b). Since we assume that the objects are bigger than the electrode lines, we apply morphology opening, which consists of morphology erosion and then dilatation. Note that the white color represents the background and black the foreground. This operation closes the inner holes of the objects and then removes some of the electrodes left. After that, another dilatation is applied to clear the rest of the electrodes (8c). Finally, we find the contours and filter the objects based on their area. If we know the number of objects being detected, we can process that and select only the ones with the largest area. Finally, the detected objects are shown in Fig. 8d.

We use raw images since it is then possible to use automatic thresholding like Otsu. If the image is *backpropagated*, the variance between classes is too low, and we obtain very different segmented binary image within consecutive frames, which could cause the following operations to fail. Red channel images also suffer from this low contrast/variance problem, and therefore, this method is reliable only for green channel images, which can still be used if 2D position estimation suffices.

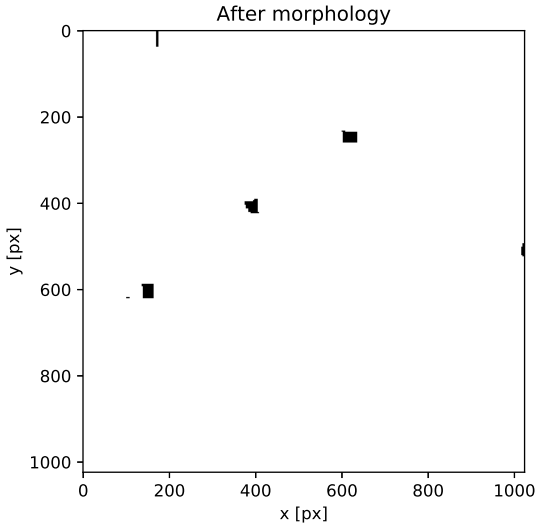
Tunable parameters are the kernel sizes for morphology operations and the number of iterations for final dilatation. Even though the objects were successfully detected in the example, this method is prone to errors caused by changes in light conditions and must be calibrated every time for the actual setup.



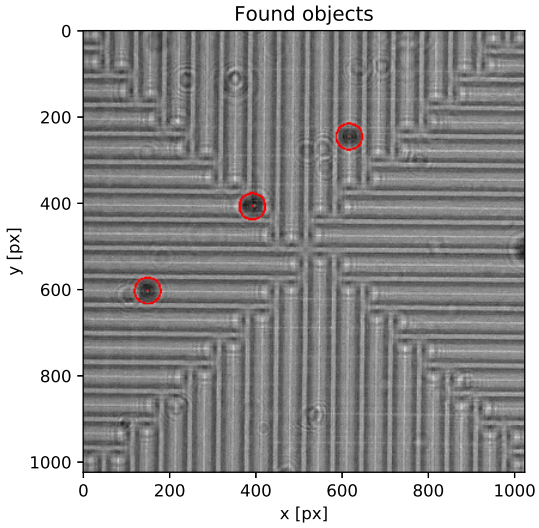
(a) Input image with 3 visible objects.



(b) Image after Otsu thresholding applied.



(c) Image after morphology opening and dilatation.



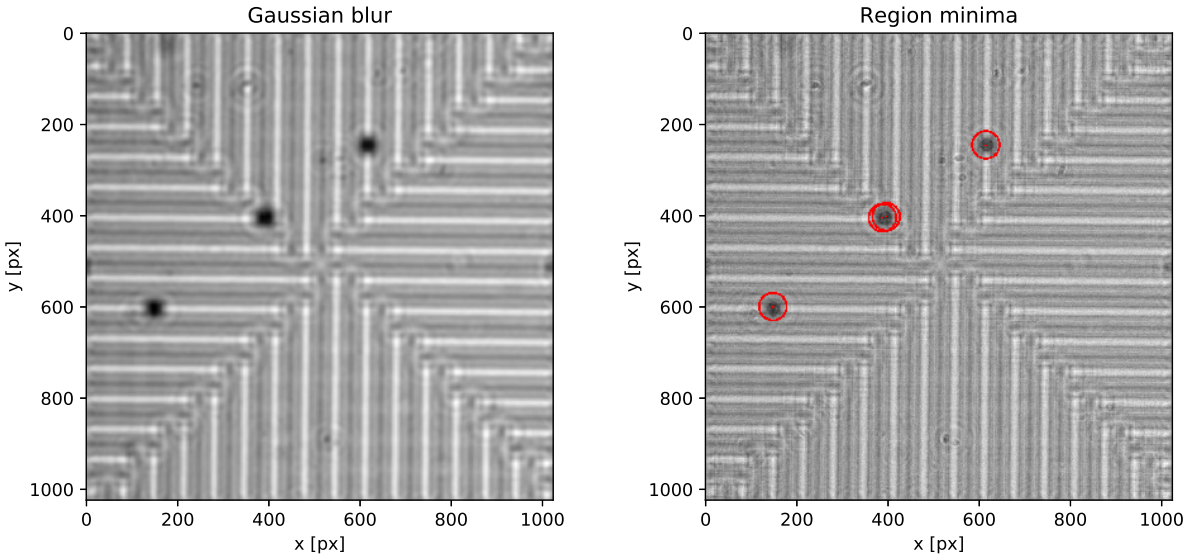
(d) Final image with found objects drawn into it.

Figure 8: Example of image segmentation. This method consists of 4 main steps. First, Otsu thresholding is applied to an input image. It is followed by 2 morphology operations, opening and dilatation. Next, we find contours of the segmented objects. Finally, the candidate objects are filtered by their area, and center coordinates are computed. This method relies heavily on parameter tuning. During testing on a video, consecutive frames often had different results (some objects were missing, or an error detection occurred). Therefore it is not suited for deployment in the final application.

2.2 Regional minima

This method assumes that the target objects have the lowest intensity in the image. It is, therefore, possible to find the objects as the darkest pixels in the image. Since the input image is very noisy - which would result in many false-positive detections - we apply a Gaussian filter to smooth the image. Afterwards, a *minimum filter* (also known as *regional minima* in Matlab) is applied to find the centers of the dark spots (target objects). Furthermore, we filter out those centers that have higher intensity values than selected threshold so that we separate possible target objects from the dark spots in the background.

Similarly to the previous method, some parameters have to be tuned for actual conditions. We primarily adjust the kernel size of the Gaussian blur, which has to be changed for different shapes and sizes of the objects, and the intensity threshold for selecting desired minima. An example is shown in Fig. 9, where the objects were detected as expected, but there was a double detection on the middle object. During the tracking, we have to assume that this can happen.



(a) Gaussian blur applied to the original image. **(b)** Found objects with one double-detection.

Figure 9: An example of the regional minima method, with the kernel size of the Gaussian blur set as 29 x 29. The input image is the same as in Fig. 8. Even though this method found desired objects, the middle object has 2 minima very close to each other. This could be resolved by increasing the kernel size of Gaussian blur or by increasing the region size when detecting minima. Nevertheless, this method is less prone to errors due to environmental changes than previous image segmentation, but results are still not consistent enough, and the parameters require tuning for a specific setup.

2.3 Simple tracking algorithm

With object detection algorithms ready to use, we need to design an algorithm to keep track of the positions of the objects. We chose the approach of manual initialization; otherwise we could start with false detections. We start by selecting the number of objects for tracking and manually finding their position in the first image. Then every time an input image is changed, object detection is called, and new positions candidates are obtained. For every tracked object, we find the closest position from the candidate list, and if it is within distance threshold, we update the position. By this, we assume that the positions do not vary much within consecutive frames. The algorithm is summarized in Alg. 1.

Algorithm 1: Tracking algorithm built on top of object detection

```
Input: numObjects, distThreshold
Data: image, positions[numObjects], keepRunning
image = readImage();
positions = initializePositionsManual(image, numObjects);
while keepRunning do
    image = readImage();
    detections = objectDetection(image);
    for i=0; i<numObjects; i++ do
        newPos = findClosestDetection(positions[i], detections);
        if norm(positions[i] - newPos) < distThreshold then
            positions[i] = newPos;
```

2.4 Conclusion

Even though the proposed tracking algorithm based on object detection works most of the time, it is not robust since we have to adjust all the parameters every time we run it. Furthermore, the image segmentation method has problems thresholding red channel images, since they have low between-class variance, and thus it is reliably applicable only on the green channel images. We need a stable solution that works without manual interventions. Therefore, a more robust tracking algorithm based on convolutional neural networks is discussed within the next section.

3 Tracking with Siamese networks

In the previous section, we have concluded that the presented basic methods are not sufficient for our application, and therefore we employ SiamFC [15] tracker, which belongs to the group of the state-of-the-art offline trackers. Furthermore, it is a well-suited solution for real-time applications. In this section, we start by describing the CNN architecture, which is used within the tracker (Section 3.1). Afterwards, we dive into to the tracking process of the SiamFC tracker in Section 3.2. Since computational time is one of our main concerns, we present a performance benchmark for various setups in Section 3.3. We follow by proposing an improved method for position estimation in Section 3.4, and finally, the experiments with the results are presented in Section 3.5.

3.1 CNN Architecture

The architecture of the convolutional neural network used within SiamFC tracker is similar to the one proposed by Alex Krizhevsky *et al.* [11], also known as AlexNet, which won the ImageNet 2012 challenge in image classification [23]. Original AlexNet follows the convolutional layers with two fully-connected and one softmax layers. Those layers are not used for tracking since they are present only to learn how to connect the features from convolution and output confidence scores for classes. We are interested solely in the feature embedding process; therefore, only the convolutional part of the network is exploited.

The convolutional part consists of 5 layers. The first one is a 2D-conv layer with kernel size 11×11, stride 2, and 96 filters. It is followed by a max-pooling layer with window size 3×3 and stride 2. The second 2D-conv layer has a kernel size of 5×5 with stride 1 and 256 filters. It is followed by max-pooling with the same parameters as before. The third and the fourth 2D-conv layers have the same parameters with kernel size 3×3, stride 1, and 384 filters, but they are not followed by max-pooling layers. The parameters of the last 2D-conv layer differ only in the number of filters, which is 32. Every convolutional layer, apart from the last one, is followed by RELU activation layer. The network is visualized in Fig. 10 with example input image of size 255×255.

This architecture yields many advantages. It is fully-convolutional with respect to search image x [15]. This property enables us to use any size of the input image without re-training of the network. Furthermore, the network does not consist of many or large layers, which is convenient for real-time applications when exploiting the parallel power of GPUs.

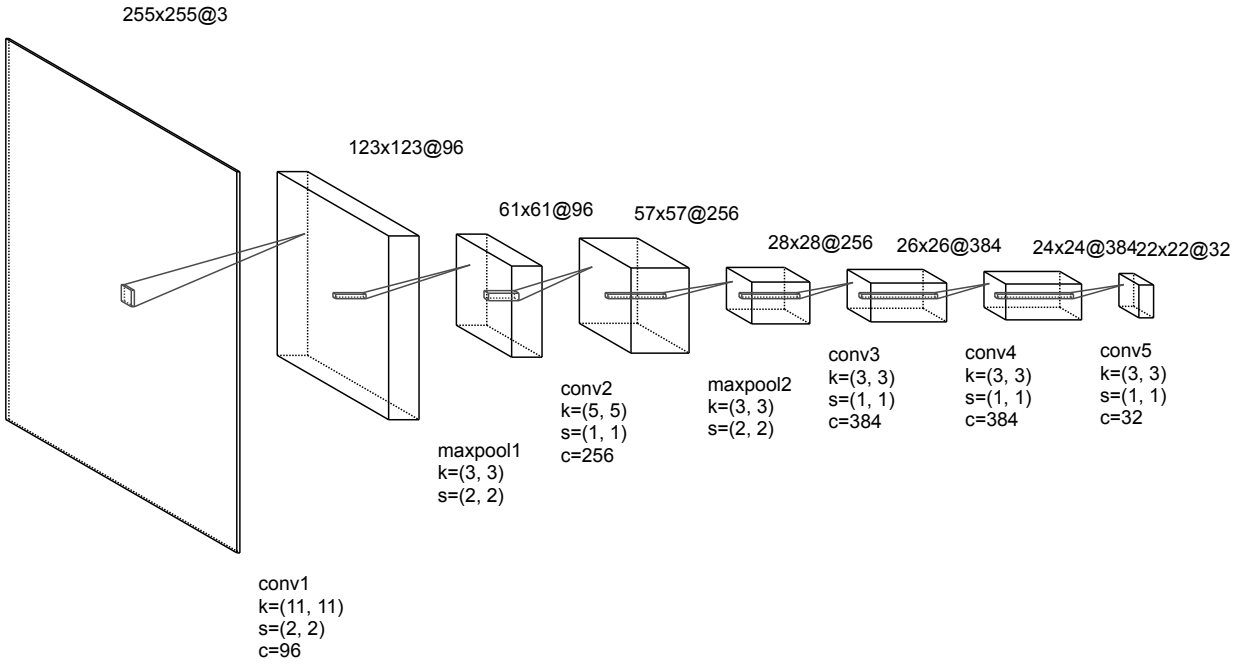


Figure 10: Architecture of the CNN used within SiamFC tracker with example input image size of 255×255. The network consists of 5 convolutional and 2 max-pooling layers. Every convolution layer is followed by RELU layer, except for the last one. Symbols represent: k stands for kernel size, s for stride size, c represents number of channels (filters).

3.1.1 Pre-trained model

Authors of the SiamFC tracker [15] publicly released pre-trained networks on the ILSVRC 2015 (ImageNet) dataset [23]. The dataset consists of almost 5000 videos with over than 1 million labelled images in total. Dataset as vast as ImageNet is convenient for tracking since the videos were shot in many varying domains. This enables safer training without over-fitting the deep net model and thus learning better the general similarities. Fig. 11 shows examples of 3 videos from this dataset. Images on the top represent exemplar images, and the bottom images represent instance images. If the cropped image overextends the bounds of the original image, it is padded with the average color of the whole image.

It is not worth noting that the pre-trained models are trained on 3 channel (color) input images, but we have one channel output for tracking. Therefore, we use stacked grayscale images as the input to match the network structure.

Figure 11: Training data example. The network was trained on almost 5000 videos from the ILSCVRC 2015 dataset[23], which consists of videos from various domains with different objects, which helped the network to learn how to extract general similarities and thus it is not required to perform online training for any part of the network for specific domain (*reprinted* from [15]).

3.2 SiamFC tracking process

The original SiamFC tracker is designed to track arbitrary objects with shapes changing in time without updating the target or online adapting to a specific domain. We are able to simplify some parts of the original process since our environment does not change much during the experiment, our camera is static, and tracked objects do not change in size.

We describe the simplified algorithm and then specify the modifications we made with respect to the original method proposed in [15]. See Figures 12 and 13 for visual examples. The tracking process starts by selecting a bounding box around the target object. The exemplar image z is formed by cropping and padding from the frame image (13b). Since we track offline and without updating the target, we can save the $6 \times 6 \times 32$ feature volume of z, which is obtained from the CNN forward pass, and reuse it during the tracking, and thus save computational time. Every iteration starts by cropping and padding instance image x. It is of size 255×255, which is approximately 4 times the area of exemplar image (13d). From the CNN forward pass, we obtain a $22 \times 22 \times 32$ volume. In the next step, a cross-correlation between the exemplar and instance volumes is computed. This results in a 17×17 score map. The score map is now resized to 272×272 using bicubic interpolation (13f) since the authors of the SiamFC tracker stated that the position estimate was more precise when interpolating the score map [15], and afterwards, the score map is multiplied

with cosine window to penalize for larger displacements. Finally, we find the highest score in the map and compute the position displacement. Target position is updated, and the process is repeated when a new image is available. This process is also described using pseudocode in Alg. 2.

The original method processes instance images in 3 or 5 different scales and processes them as a mini-batch. First, the best scale is found and only after that the displacement is computed. But since we assumed that the objects do not change in scale and we have a static camera, we omit the scaling part and process only one instance image. This has the benefit of cutting the computation time approximately in half.

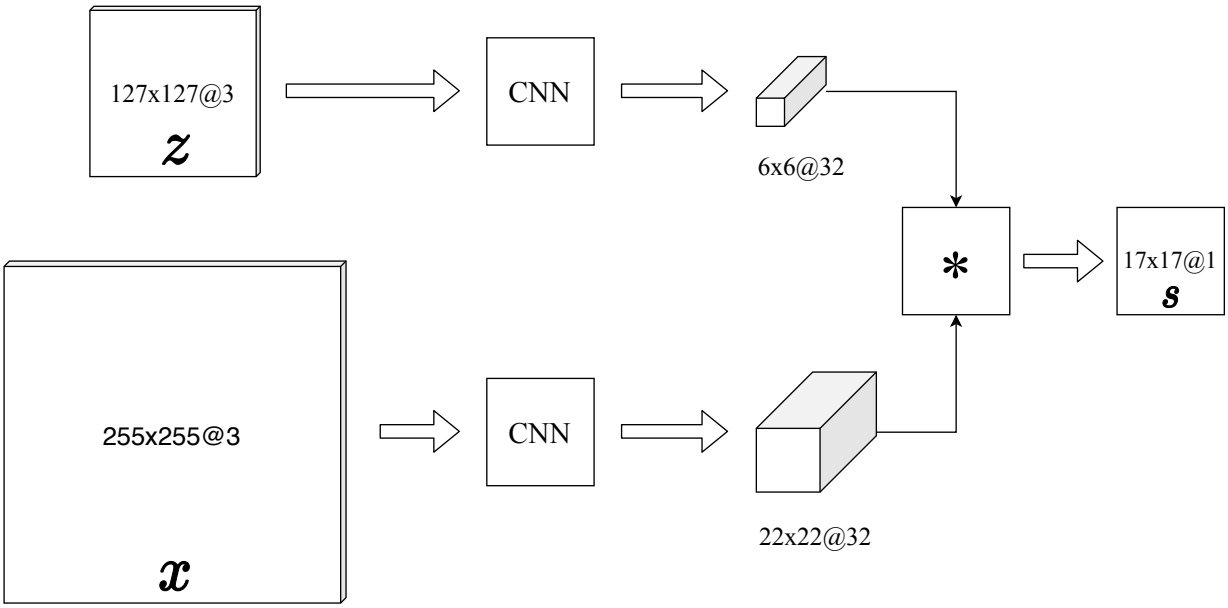
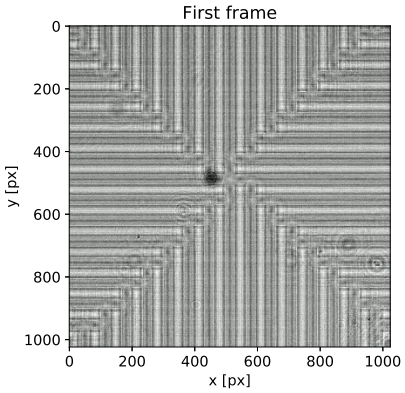


Figure 12: Instance and exemplar images are forwarded through the CNN to obtain feature volumes and then a score map is formed from cross-correlation between these volumes.

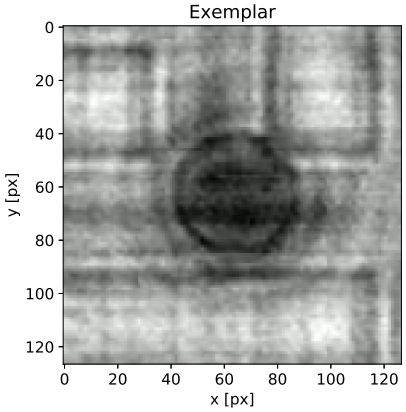
The exemplar and instance images are cropped and scaled so that that the bounding box with some additional context margin has a fixed area. We use the context margin size as half of the mean dimension of the bounding box. The scale ratio s is given as

$$s(w + 2p) + s(h + 2p) = A, \tag{6}$$

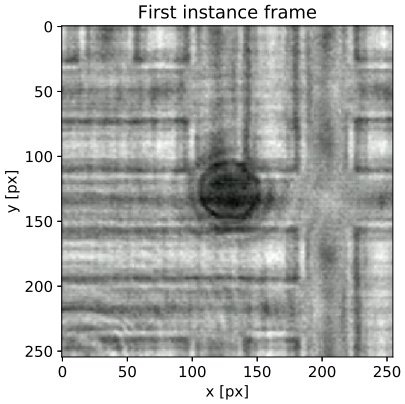
where w and h are width and height of the bounding box, p is context margin, and A is area. Exemplar image has area 127×127, and instance image has area 255×255. The images are not deformed and keep the aspect ratio. Cropped images are resized using bilinear interpolation to match the structure. The displacement found from the score map is scaled accordingly.



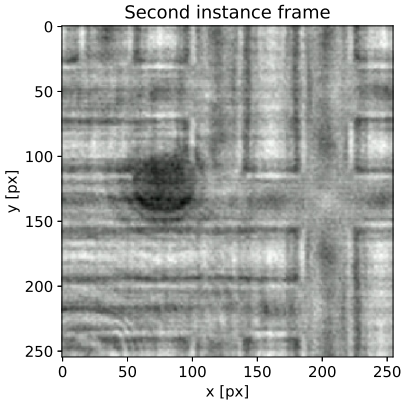
(a) First frame of size 1024×1024.



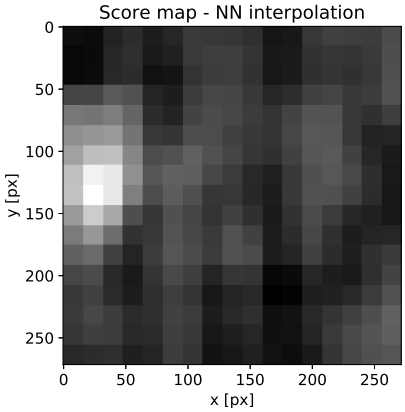
(b) Exemplar image of the object we track.



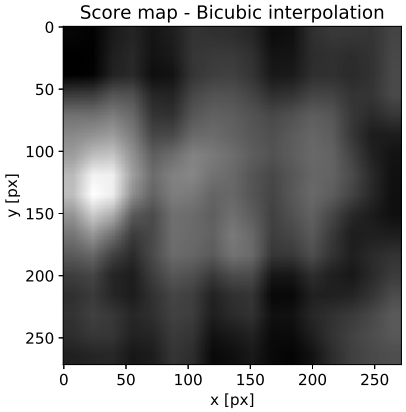
(c) First frame - cropped instance image.



(d) Second frame - cropped instance image.



(e) Score map of original size 17×17 resized to 272×272 using nearest-neighbour interpolation.



(f) Score map of original size 17×17 resized to 272×272 using bicubic interpolation. Displacement is computed from this map.

Figure 13: An example of the tracking process is shown on two consecutive frames. First, we initialize the tracker by selecting a bounding box in the first frame. This gives us the exemplar image (13b). Second instance frame (13d) is then used to update the tracker. The result is interpolated (bicubic) score map (13f), from which the displacement is computed. Note that the displacement from the score map has to be scaled accordingly by our given setup (bounding box size, image sizes, upscaling ratios).

Algorithm 2: SiamFC tracking algorithm

```
Input: exemplarSize, instanceSize, upscaleSize
Data: image, bbox, exemplar, instance, keepRunning
image = readImage();
bbox = initializeBoundingBox(image);
exemplar = CNN(cropAndPad(image, exemplarSize, bbox));
while keepRunning do
    image = readImage();
    instance = CNN(cropAndPad(image, instanceSize, bbox));
    score = crossCorrelation(instance, exemplar);
    scoreUp = resizeBicubic(score, upscaleSize) * cosineWindow(upscaleSize);
    displacement = computeDisplacement(scoreUp, exemplarSize, instanceSize);
    bbox += displacement;
```

3.3 Performance benchmark

The performance benchmark is divided into two parts. In the first part, the focus is on measuring individual parts of one tracking iteration, and in the second part, we compare performance when multiple trackers are running simultaneously (our goal is to track at least two objects in 20 FPS). This benchmark was performed on Jetson AGX Xavier computer (Section 1.4) with maximum performance setting, running on approx. 6000 frames long video with frame size 1024×1024 and moving target objects. The tracker is implemented in PyTorch[3], which is a well-known framework for deep learning with neural networks.

The tracking iteration is divided into six parts. *Crop and pad* part consists of cropping the instance image from the current frame based on last known position and padding with mean of the image if the frame bounds are exceeded. *Transfer to GPU* part is the time it takes to transfer instance image to the GPU. It is followed by *CNN* part, where we measure forward pass of the network (with cross-correlation at the end). The next part is the time it takes to transfer the score map back to CPU. The fifth part consists of resizing the score map using bicubic interpolation. And the final part, *Position displacement*, consists of normalizing the score map, multiplication with cosine window, finding indices of the maximum score, and updating the position.

[3] `https://pytorch.org`

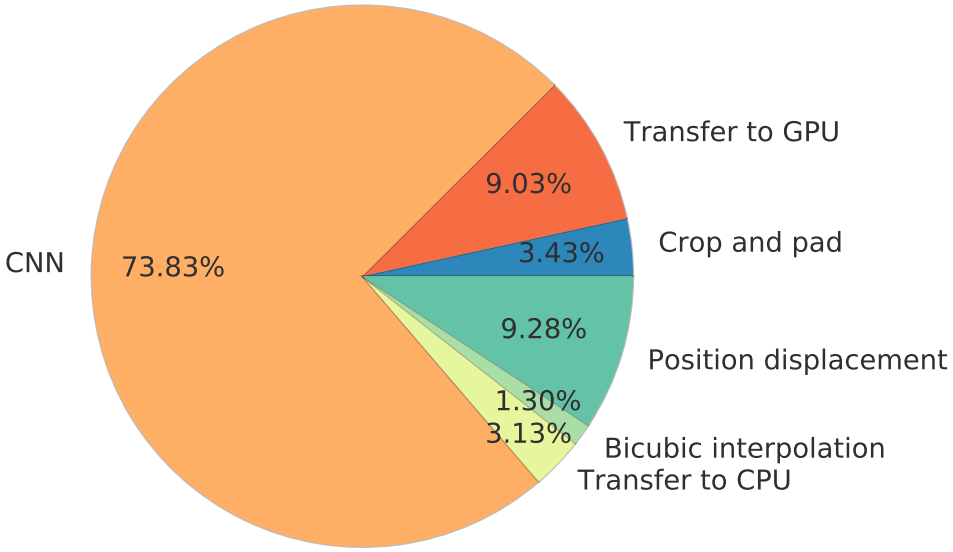


Figure 14: This chart shows the per-part time portions of an average iteration. The average iteration time was approx. 20ms during this benchmark. The most computationally demanding part is the forward pass of the CNN.

Fig. 14 shows per-part percentage of the average iteration time when one object was tracked, taken from Table 1. It is clear that the CNN forward pass takes most of the computational time. The average iteration time is approx. 20ms, which corresponds to 50 FPS. If speed requirements were higher than ours, the primary focus would be on simplifying the network architecture.

Value in [ms]	C&P	GPU	CNN	CPU	INTERP	POS	TOTAL
$\bar{t}$ 1 object	0.67	1.76	14.39	0.61	0.25	1.81	**19.49**
std 1 object	0.09	0.37	3.58	0.13	0.06	0.4	4.02
$\bar{t}$ 2 objects	0.65	2.1	11.71	0.6	0.22	1.37	**16.66**
std 2 objects	0.06	1.4	3.58	0.28	0.03	0.12	3.78
$\bar{t}$ 4 objects	0.72	3.83	19.79	1.34	0.24	1.57	**27.5**
std 4 objects	0.27	2.41	4.88	1.15	0.15	0.4	5.92

Table 1: This table compares the average iteration times ($\bar{t}$) and the standard deviations (*std*) for 6 parts of the tracking process for various setups. The average iteration time of 2 tracker instances was 60 FPS and 4 trackers were able to run at approx. 35 FPS.

The interesting part is the multi-object tracking, since that is our use-case. As it was mentioned before, we run multiple instances of the tracker in parallel for multi-object tracking (our objects are of the same time). Table 1 presents measured data of per-part averages and the average total iteration times for different setups of 1, 2, and 4 simultaneous trackers. There is an irregularity that 2 running instances are faster than just a single one (60 vs 50 FPS). It was measured multiple times, but the results were always similar. Even though this behaviour seems strange, we did not examine this further, since it was not causing any problem. When 4 instances were running, we measured approx. 35 FPS on average with higher deviation, which is significantly slower, but nonetheless, it is still a good result that satisfies our requirements.

3.4 Improved position estimation for spherical objects

During the testing of the tracker, we noticed that the estimated position of the target object was occasionally behind the real position, especially when there was a fast movement between consecutive frames. Note that we have no way of measuring the precise position. Therefore our assumptions are based only on observations from the video stream. Even though the estimated position from the tracker is sufficient for the position feedback controller to move the object, we propose an improved method for position estimation of spherical objects, which reduces small oscillations in position estimates during movement.

Similarly to the method of finding regional minima discussed in Section 2.2, we assume that the target objects have the lowest intensity in the image. This method consists of adding some context around the bounding box computed by the tracker, then applying Gaussian blur with kernel size selected based on the approximate size of our target object, and finally, finding the minimum intensity value, which corresponds to the center position of our spherical object.

An example with fast movement between the frames is shown in Fig. 15. In the first frame, the estimated positions from the tracker and our method are almost identical, but when there was a large translation between the first and the second frame, our method produces a visually better estimate. The estimated position from the tracker would also converge to the same position, but it would take at least 2 frames, which could result in small oscillations produced by the feedback controller.

We could also improve the original method for larger displacements by increasing the search region of the tracker or enlarging the context area. This would work well if only one spherical object were on the electrode array. If two spherical objects were closer together, and we had a broader search region, the tracker could detect the other object and change its focus.

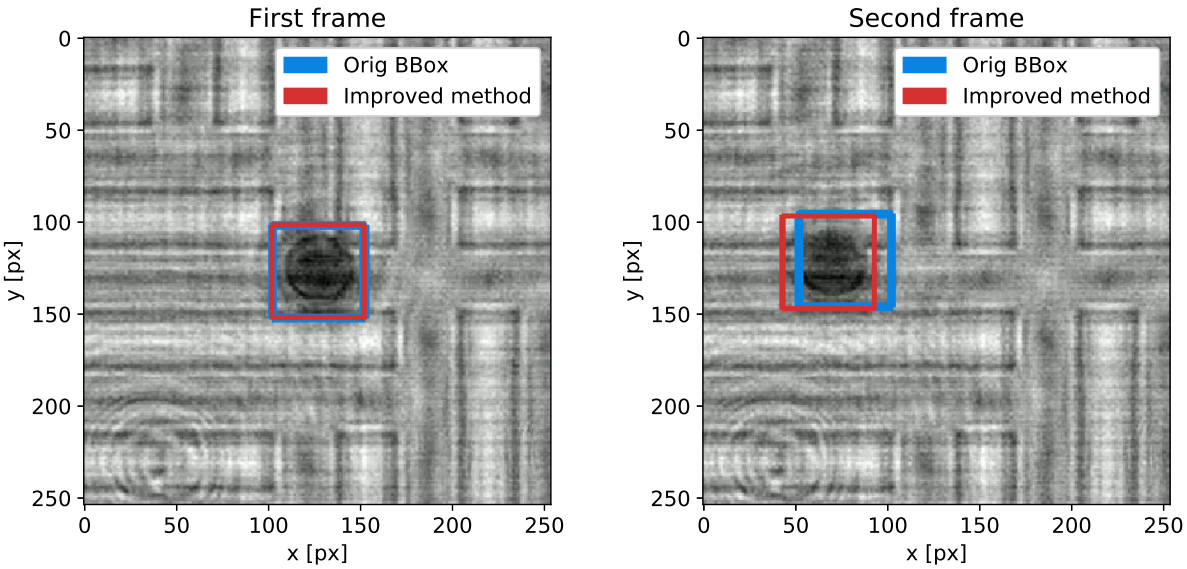


Figure 15: Example of our proposed method for position estimation demonstrated on two consecutive frames with large object displacement between them. Our method improves the estimation of the center of the tracked spherical object.

3.5 Tracking experiments

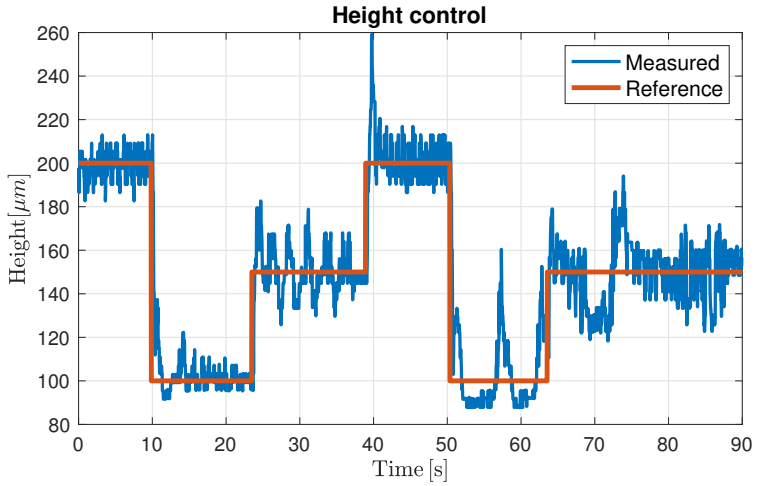


Figure 16: In this experiment, a PI controller was employed to follow a height reference signal. Tracking in both green and red channels was necessary for height estimation.

Two experiments were conducted to verify the described tracking algorithm. In the first experiment, the object was following a circular trajectory. Our aim was to test if the tracker can keep focus on the target and thus provide feedback to the controller. Recorded

trajectories are shown in Fig. 17. The tracker was able to follow the target object (in both channels) despite the fact that there were dust particles placed along the trajectory.

In the second experiment, which is shown in Fig. 16, we employed two trackers (for tracking in greed and red channels) to estimate height (4) of the target object. Afterwards, we tried to manipulate the height of the object, and a PI controller was used to follow the reference signal. Note that our aim is not to follow the reference perfectly, but to prove that the tracker with improved position estimation can be used for height estimation in the twin-beam setup (Sec. 1.3). Furthermore, the PI controller for height is hard to tune. Therefore, the oscillations were expected. Nevertheless, the height levels were approximately kept. Videos from the experiments are at `https://github.com/aa4cc/twinbeam-tracker`.

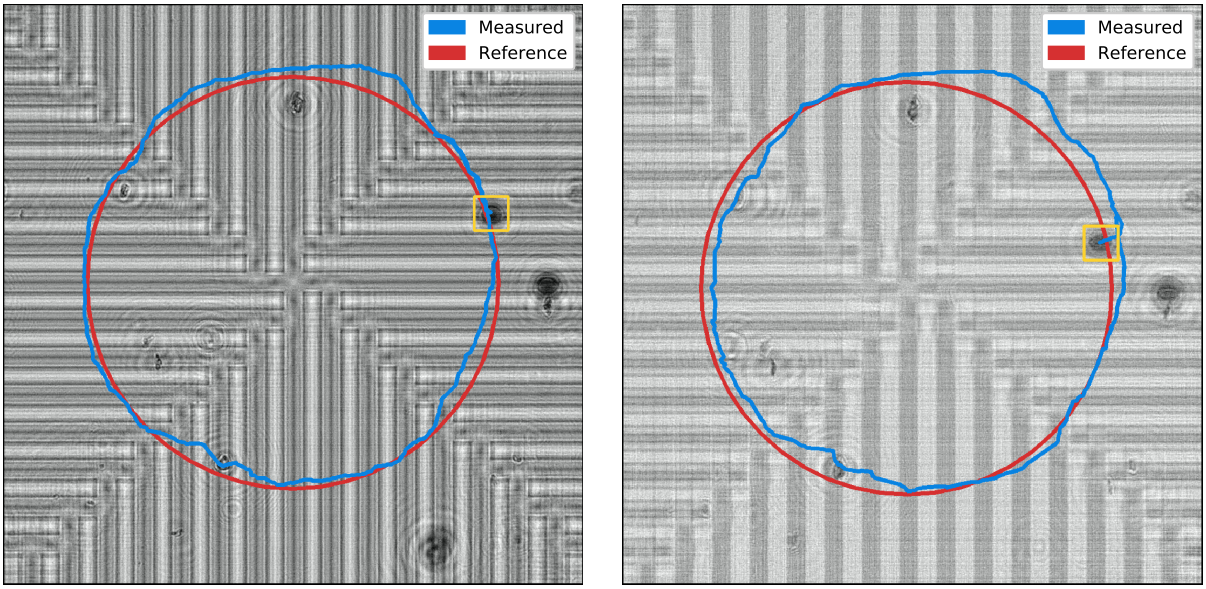


Figure 17: Tracking experiment with the modified SiamFC tracker. The yellow rectangle marks the bounding box around the target object. The tracker was able to track the object during the whole experiment in both green and red channels.

3.6 Conclusion

In this section, we described the architecture and the tracking process of SiamFC tracker. The tracking process was simplified since we assumed that the shape of the object do not change over time. Furthermore, we compared time benchmarks for several instances of the tracker and verified that it meets our real-time requirements. Finally, we tested the tracker on a circular trajectory a it performed well. Therefore, this particular tracker is used for further experiments in this thesis.

Part III

Trajectory optimization and control

One of the aims of this thesis is enabling us to move a microparticle from point A to point B in the shortest possible time. We have a dynamic system, where the input is a position-dependent force, for which we do not have an analytical solution. This problem belongs to the field of time-optimal trajectory optimization.

Trajectory optimization is a broad and developed field, with many applications, such as minimum-lap-time trajectory optimization for Formula One cars [24]. Especially in the field of time-optimal trajectory optimization, the task usually consists of moving something along a path as fast as possible, while taking into account dynamical and geometry constraints. The optimization problem quickly becomes very complicated and hard to solve. The problem is usually simplified or reformulated to deal with these issues. For example, in robotic manipulation, a system reformulation method in terms of an independent path parameter instead of time was proposed in [25]. A similar reformulation method was utilized to obtain optimal speed trajectory for a train in [26]. Nevertheless, all these methods share one major drawback: in the end, the optimal trajectory is obtained by solving a non-convex optimization problem. As a result, one never has a guarantee that the obtained trajectory is the globally optimal one. We show that our particular instance of a time-optimal trajectory optimization problem can be solved very quickly by a graph search algorithm to global optimality.

In this part, we deal with time-optimal trajectory optimization for a microparticle in the task of DEP manipulation in Section 4. We simplify the problem by restricting the motion to 8 directions and pre-computing a map of maximum achievable forces based on position. That enables us to employ an optimal path-finding algorithm to obtain a time-optimal trajectory directly. We present simulation results and optimal trajectory examples in Section 4.2. Afterwards, we compare an optimal trajectory to a naive one in Section 4.3. Finally, we propose changes to the current control algorithm in Section 5.

4 Trajectory optimization

The dynamic model of a spherical particle in a liquid can be written as

$$\begin{aligned} \dot{x} &= k_F F_x^{\text{dep}}, \\ \dot{y} &= k_F F_y^{\text{dep}}, \\ \dot{z} &= k_F \left(F_z^{\text{dep}} - F^{\text{sed}} \right), \end{aligned} \tag{7}$$

where F^{dep} is the *dielectrophoretic* (DEP) force, F^{sed} is the sedimentation force caused by gravity, and the force coefficient is defined as $k_F = \frac{1}{6\pi\mu r}$, where r is radius of the particle, and μ is dynamic viscosity of the fluid [5]. Furthermore, this model assumes that the inertia of a microparticle particle is negligible [27]. Note that the model is valid only for a spherical microparticle, therefore it is not extendable to other objects. Throughout this theses, we use polystyrene particles with 50 µm diameter.

Based on the dynamic model of the particle, it is possible to formulate the task of getting the particle from the initial state $\mathbf{x}_i$ to the final state $\mathbf{x}_f$ in the shortest time as follows:

$$\begin{aligned} \underset{\substack{\theta_k \in \mathbb{R}^{56} \\ \Delta T}}{\text{minimize}} \quad & \Delta T \\ \text{subject to} \quad & \mathbf{x}_{k+1} = \mathbf{x}_k + \Delta T(\mathbf{A}\,\mathbf{x}_k + \mathbf{B}\,\mathbf{f}(\theta_k, \mathbf{x}_k)), \\ & \mathbf{x}_0 = \mathbf{x}_i, \qquad k = 1, ..., N, \\ & \mathbf{x}_N = \mathbf{x}_f, \end{aligned} \tag{8}$$

where ΔT is the time step, $\mathbf{x}$ represents 3D position, $\mathbf{A}$ and $\mathbf{B}$ are the system matrices, $\mathbf{f}$ is the force input (vector notation of the forces in (7)), $\theta \in \mathbb{R}^{N\times 56}$ are the phases of the signals applied to the electrodes, and N is the number of discretization steps. Note that the Euler forward integration method in (8) is applied only for illustration. This formulation minimizes the time between steps, which results in minimization of the total time. Unfortunately, we are not able to solve such an optimization problem since the expression for the input force based on the phases (2) is too complex. Even a simpler problem (1) that finds phases resulting in a desired force is non-convex and channeling to solve.

Therefore, we propose a simpler solution. From the dynamics model, we exploit that the velocity is proportional the applied DEP force, and since we seek a time-optimal trajectory, we want to maximize the velocity directly, and thus minimize the total trajectory time. To maximize the velocity, a maximum achievable force based on position and direction have to be known. For that, we can exploit the previously solved problem (1) of finding phases for a desired force. To find a maximum achievable force, we iterate over a set of candidates for $\mathbf{F}_{\text{des}}$ from (1), and try to solve the non-convex optimization problem numerically. After it is solved for a candidate, the error between the actual DEP force and the candidate for $\mathbf{F}_{\text{des}}$ is computed. If the error lies within defined bounds, the force is

enlarged, and the process is repeated until the solver fails to provide us with a feasible solution.

This process would still be computationally demanding and intractable in real time. Therefore, we have decided to restrict the motion of the particle to 8 directions and pre-compute a grid of maximum achievable forces. The fact that the force is proportional to the velocity enables us to employ an optimal path-finding algorithm, such as **A***, to find a time-optimal trajectory. We further restrict this problem only to 2D space, since we are mostly interested in moving the particle from point A to B in a plane fixed at a given height above the electrode array. Nonetheless, this method could be easily extended to 3D space, with the only cost being a larger grid.

4.1 Maximum force simulation

Let us briefly recapitulate. The goal is to pre-compute a map of maximum achievable forces in 8 directions (the angle resolution is, therefore, 45°) for every position in the grid, which is formed by a uniform discretization of the manipulation area. It is not possible to find maximum force directly; therefore, a set of forces have to be tested for feasibility by numerically solving the problem in (1).

To create such a map, three parameters have to be considered. First, what is the optimal size and resolution of the map, balancing between sufficient granularity and feasible path/trajectory computation time. Second, what is the valid range of forces that should be considered during simulation. Finally, since the force optimization problem stated in (1) is non-convex, how many times should the solver be restarted with new initial conditions before the candidate force is considered unachievable.

As we have mentioned before, our manipulation area is in range of $[-750, 750]$ µm. Therefore, the bounds of the simulation grid are identical. After testing and manual tuning of the parameters, we ended up with the following settings. The solver can be restarted 100 times with new conditions before considering the force unachievable, and the set of force candidates ranges from 10 pN to 1 nN with an incremental step of 1 pN. During the simulation, we keep the height of the particle constant at 130 µm, which is a typical height for manipulation. The force in the z axis is set to 33.4 pN, which correspond to the sedimentation force caused by gravity.

Figure 18 compares 4 grid resolutions for *left* direction. In our coordinates system, *left* means negative in the x axis. We can see that the highest values are located above the electrodes perpendicular to the force direction for all of the resolutions. The simulation seems to be sufficiently detailed for both 25 µm and 50 µm step sizes, while having reasonable map sizes. We have decided to continue and perform a full simulation with a grid step size of 25 µm to make the following experiments based on more detailed data.

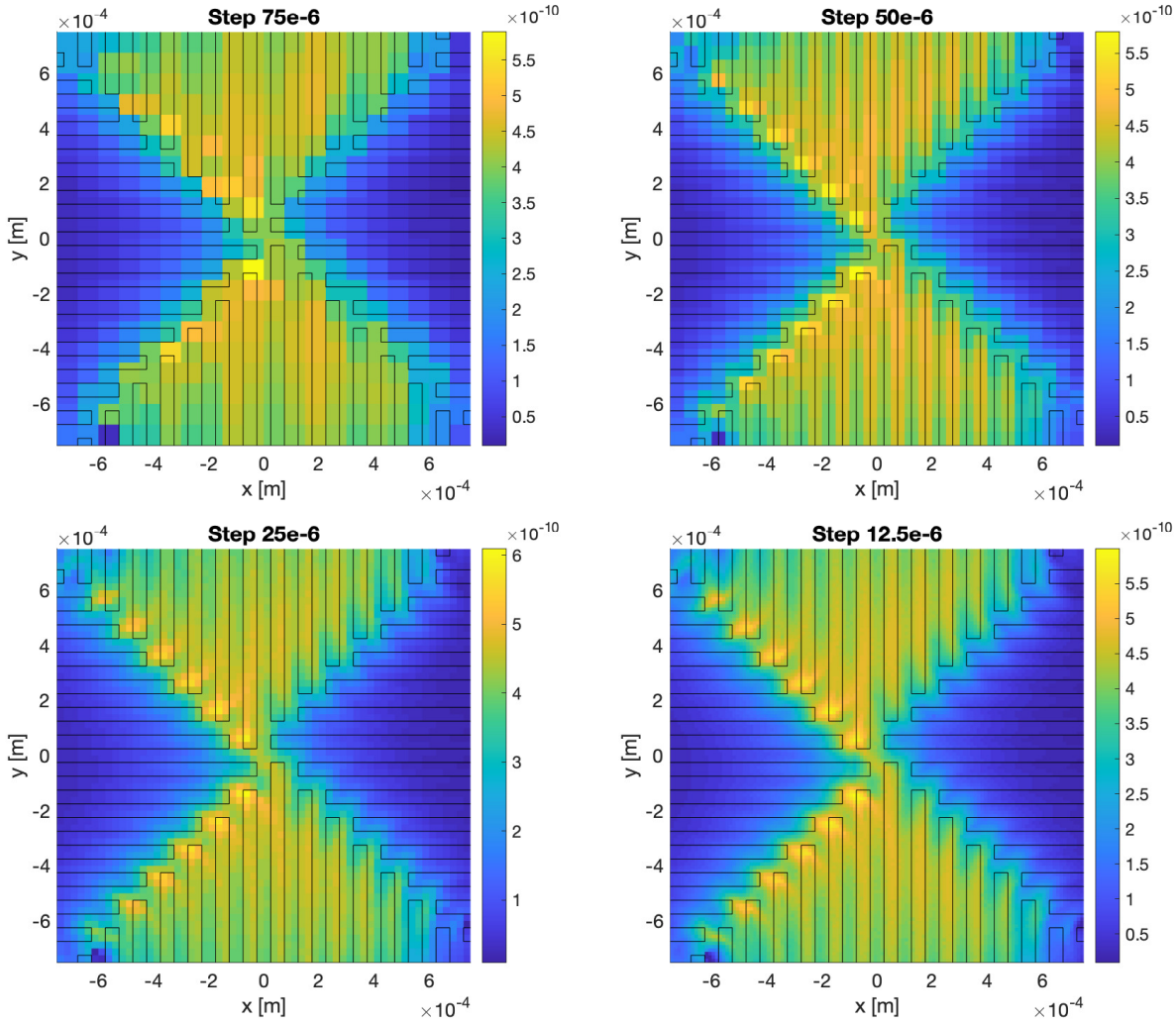


Figure 18: Multiple resolutions of the maximum forces simulation. A balance between the granularity and the path computation time has to be considered. Throughout this theses, a resolution of 25 µm is used.

Figure 19 visualizes results from a full— 8 direction—simulation for a 50 µm polystyrene particle with a 25 µm grid step. In the case of vertical and horizontal forces, the maximum forces are located on the electrodes that are perpendicular to the direction of the force. And in the case of diagonal forces, the maximum values are located at the boundaries of the electrode array segments. The data from this simulation are used in the following experiments.

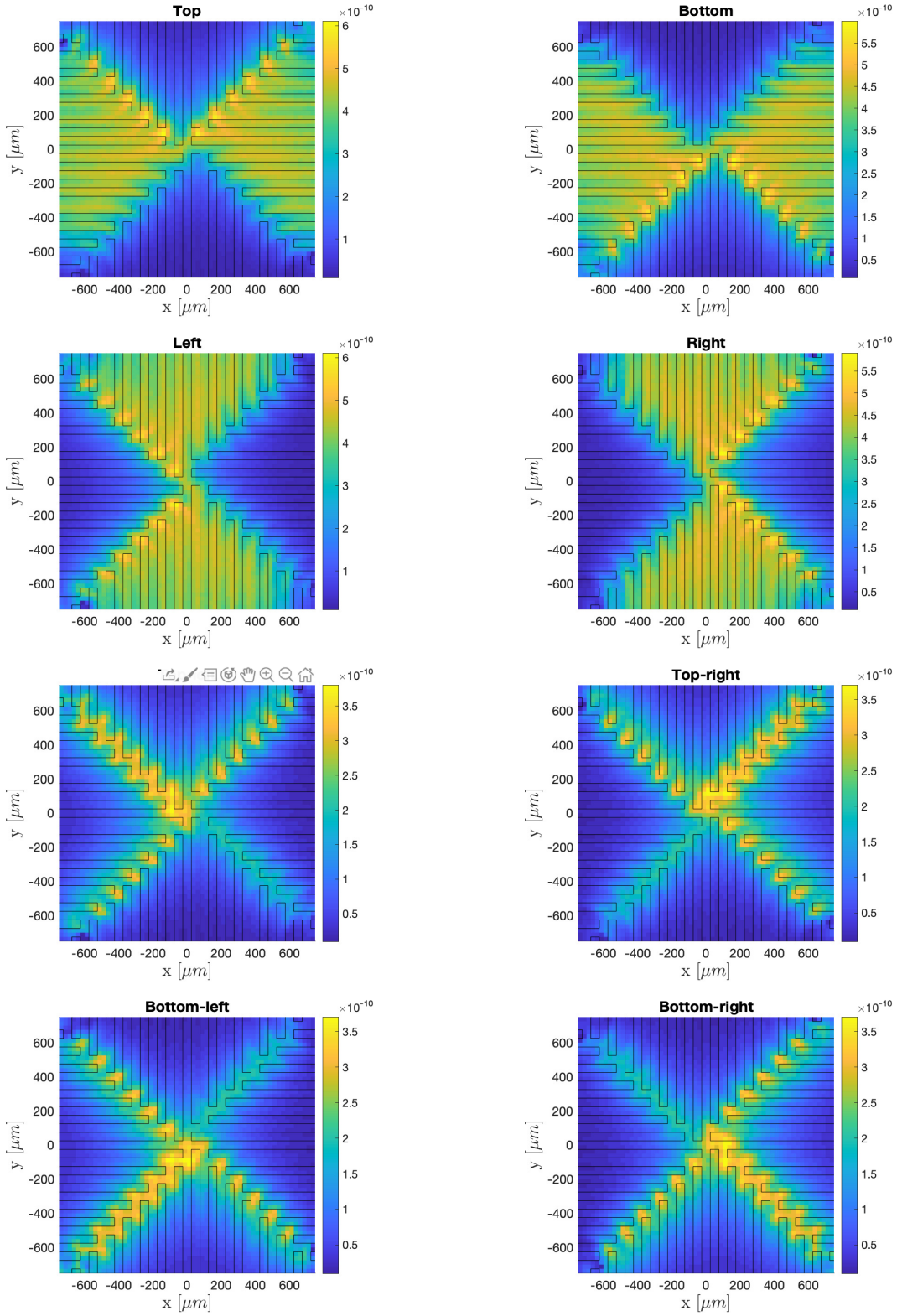
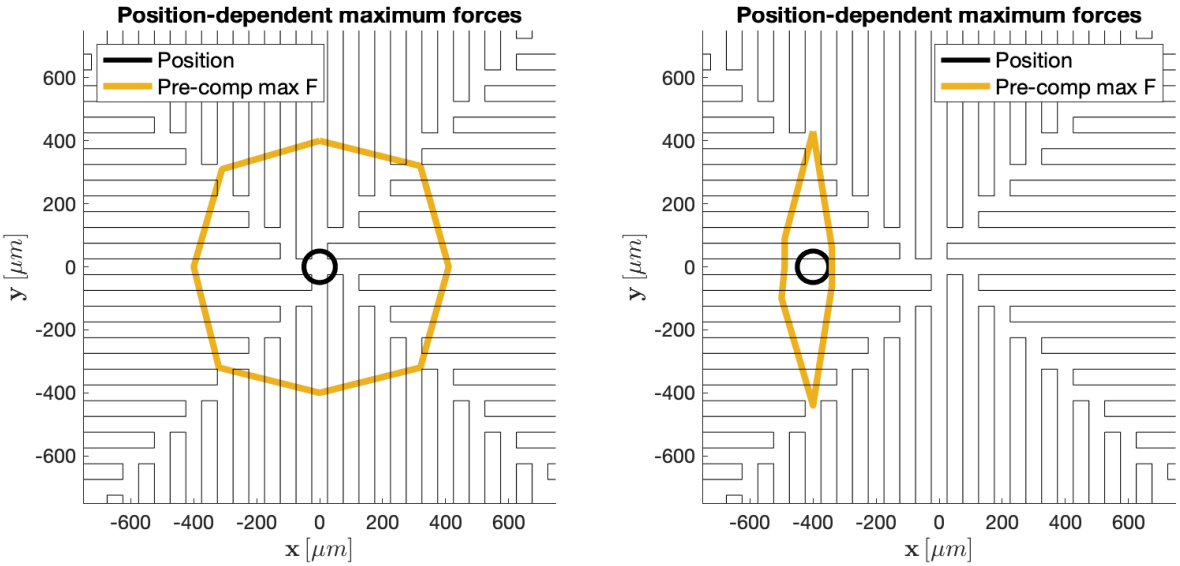


Figure 19: Simulation of maximum achievable forces across the electrode array in 8 directions for a 50 µm polystyrene particle with a 25 µm grid step.

4.1.1 Visualization in the electrode array

For an intuitive visualization in the following experiments, we draw the maximum achievable forces based on position linearly interpolated between the force vectors. This forms a polygon around the position, as shown in Fig. 20. Note that the force is scaled such that it is visible in the electrode array. Therefore, it is not drawn to present absolute values, but rather ratios between the forces in 8 directions. The scaling factor is identical for all of the figures in this thesis. Fig. 20b shows that the particle can move easily in the top and bottom direction, but it is hard to move it to the left and right.



(a) Position [0, 0] µm.

(b) Position [-400, 0] µm.

Figure 20: Visualization of the position-dependent maximum forces.

4.2 Trajectory generation

Now that we have a pre-computed map of maximum achievable forces in 8 directions, it is possible to employ an optimal path-finding algorithm (**A*** in our case) to find a time-optimal trajectory. This is possible since the velocity is proportional to the applied force (7). Our cost function for expanding a node is the time it takes to move by one grid step in a given direction. The heuristic function is the Euclidean norm between the position of an expanded node and the position of the goal node. Since the size of the grid and the maximum achievable force are known, we checked for the admissibility of the heuristic function.

In the real-world experiments following this section, we use paths with a path-following algorithm instead of time-varying reference position based on trajectory. This is a common practice in navigation since the behaviour in a real environment is usually different than in simulation. Our case is not an exception. During the experiments, the position feedback based on visual tracking brings delay to the system. Therefore, it is not possible to reasonably follow the time-varying reference position along the optimal trajectory.

A simple path-following algorithm is employed. Every time our particle reaches the current target position (with a radius threshold), the target position is updated to the next reference point of the path. The radius threshold is set to 50 µm in the experiments. Furthermore, we use the P regulator mentioned in (3) to move the particle along the path.

The path consists of many points since it is computed on the grid. To simplify the path, we use the *Douglas–Peucker* curve decimation algorithm [28]. An example is shown in Fig. 21. We can see that the path consists of many line segments, which are simplified by the algorithm. We use this method further to simplify all the paths. Finally, examples of the optimal paths/trajectories are shown in Fig. 22.

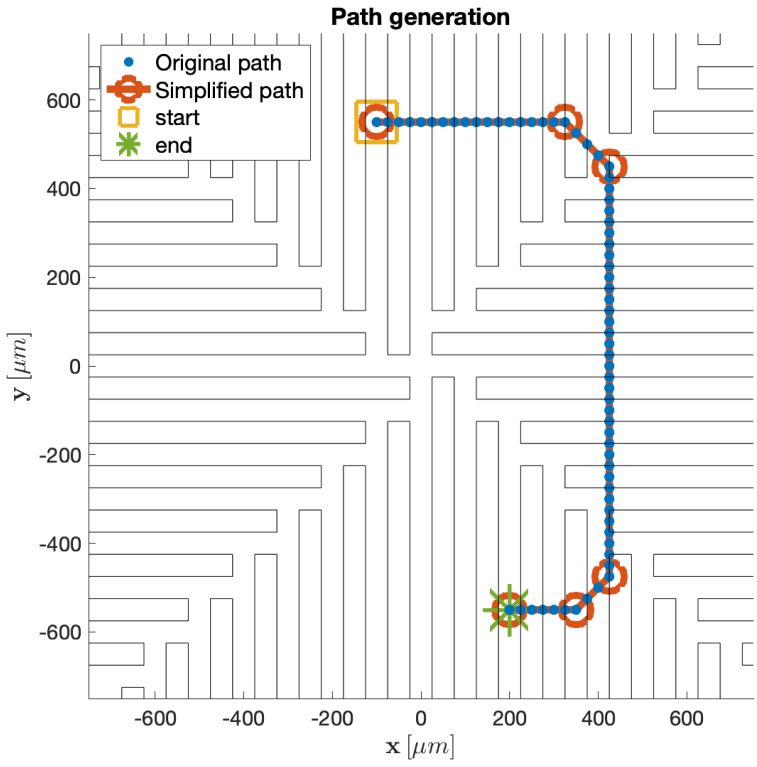
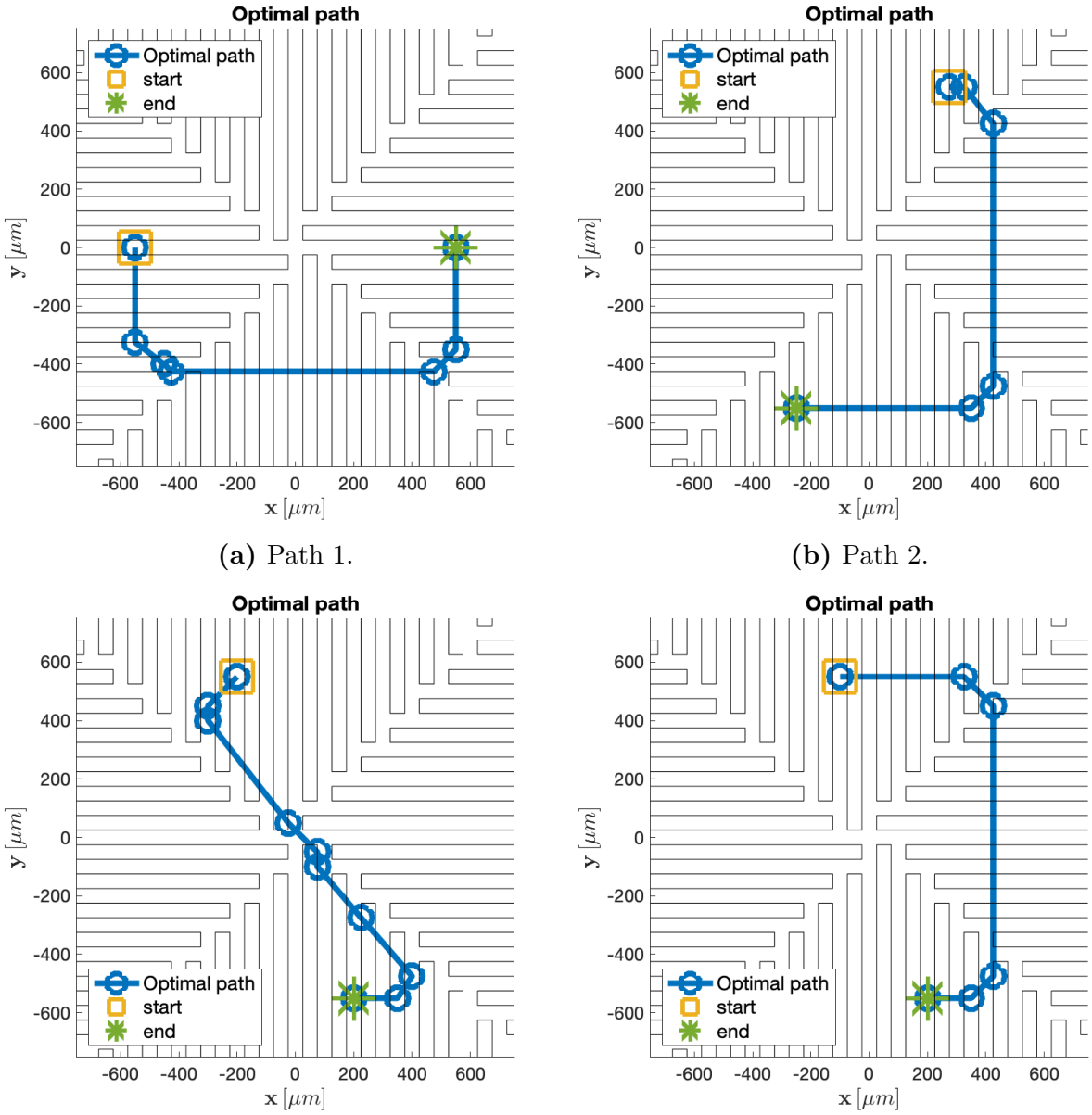


Figure 21: The original path is simplified using *Douglas–Peucker* algorithm. The key points are highlighted by the red circle.



(a) Path 1.

(b) Path 2.

(c) Path 3.

(d) Path 4

Figure 22: Optimal path examples, found by the A^* algorithm.

4.3 Optimal trajectory experiment

In this experiment, the time required to move the particle from point A to B is compared for the time-optimal and direct (shortest) paths. The path for the experiment is taken from Fig. 22a. This path gives a clear visual understanding that the optimal path

is much longer than the direct one. This may come as a surprise, but the path longer in distance is truly shorter in time need to follow it. The travel times of particle following the direct and optimal paths was 11.16 s and 8.87 s, respectively. This translates to a 20 % decrease in travel time for the optimal path. For both of the paths, the P constant of the regulator was identical. From this result, we conclude that our simplified formulation of the optimization task—with the motion restricted to 8 directions—is a valid approximation of the optimization problem stated in (8).

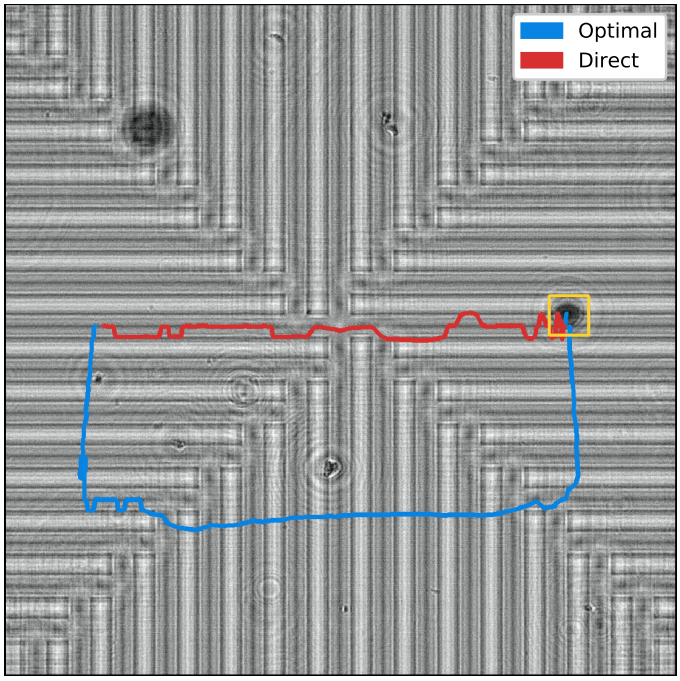


Figure 23: In this experiment, the time-optimal path was compared to the shortest path (in distance). The path is shown in Fig. 22a. The particle moved from the left to the right. The optimal path (blue) was indeed faster than the shortest one (red).

4.4 Conclusion

We dealt with the problem of getting a polystyrene microparticle from point A to B while taking into account that the input force in the dynamic model is position-dependent and complex to evaluate. We restricted the motion to 8 directions and pre-computed a grid of maximum achievable force. We exploited the fact that the velocity is proportional to the force applied to the particle, and employed an optimal path-finding algorithm to obtain a time-optimal path. The time-optimal path was compared to the shortest path and proved our approach to be a valid approximation to the optimization problem stated in (8).

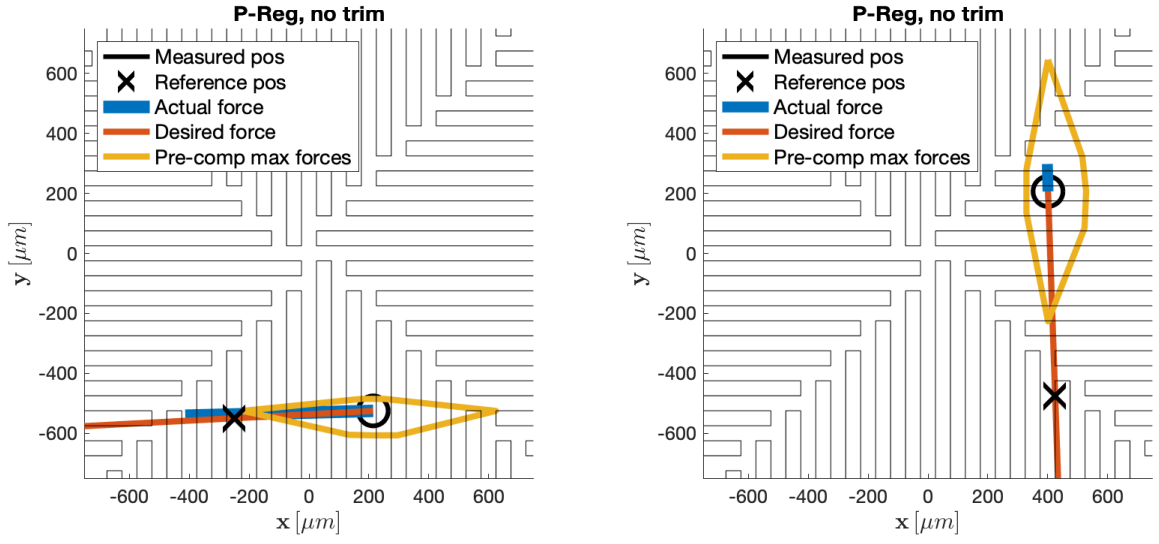
5 Control algorithm

In this section, we propose a minor modification to the current control algorithm, based on experimental observations. As was mentioned in the previous section, the path-following algorithm uses P-regulator (3) to move the particle between individual path nodes. The regulator outputs the desired force $\mathbf{F}_{des}$. For this specific force, the optimization problem (1) is solved numerically, and the phases of the voltage signals, which generate a very similar force when applied to the electrodes, are obtained. However, the solver may fail to find such phases, and a force with a different direction and magnitude is applied to the particle. A relative error of the solution from (1) is computed as

$$e_r = \frac{||\mathbf{F}_{\text{des}} - \mathbf{F}_{\text{dep}}||}{||\mathbf{F}_{\text{des}}||}, \tag{9}$$

where $\mathbf{F}_{\text{des}}$ is the desired force and $\mathbf{F}_{\text{dep}}$ is the actual applicable force. In the real-time application, we apply the force even when the error exceeds reasonable bounds. A large error causes the manipulated object to move in a different direction than it was required by the position controller. Surprisingly, this does not affect the control performance as much as one might think. This is due to the fact that large errors occur rather rarely and the frequency of the control loop is relatively high. Nonetheless, ignoring the error is not ideal.

5.1 P-Regulator without trim



(a) Slight error in the direction.

(b) Applied force is in the opposite direction.

Figure 24: P-Regulator in the current form. The solutions for desired force are often obtained with a high error.

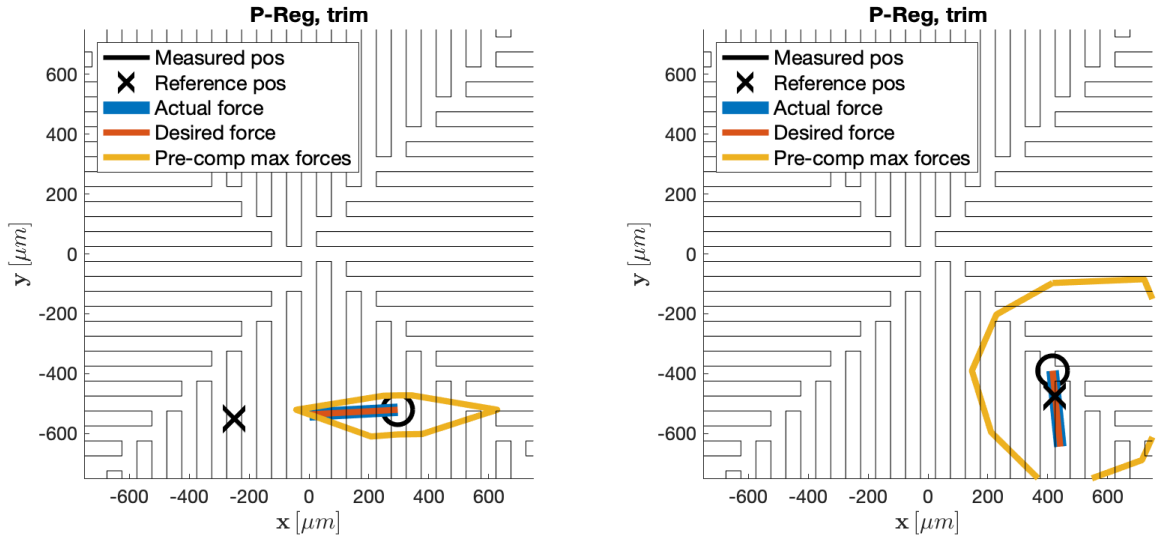
The P-regulator without trim is the one described at the beginning of this section. It computes the desired force directly from the difference between the target and actual positions and tries to find the corresponding phases of the voltage signals.

Figure 24 presents two typical *frames* from a path-following process. In Fig. 24a, we see that the desired force is way out of the pre-computed maximum force bounds. As we expected, the solver failed at finding a solution within reasonable error bounds. Nevertheless, the actual force has only a small error in the direction. An interesting observation is that the actual force is larger than our pre-computed maximum force bounds. This may happen since the pre-computed grid is based on the restriction of the movement in 8 directions.

In the second case (Fig. 24b), the solver also fails to provide a feasible solution for the desired force. But this time, the actual force aims in the opposite direction of the desired force. This error is usually suppressed since the procedure is repeated in the next step with a new position estimate, for which the solver finds a different solution.

5.2 P-Regulator with trim

To stabilize the solutions of the solver, we propose to trim the desired force at the bounding polygon. The bounding polygon is formed by linear interpolation between the pre-computed maximum forces in 8 directions. This method is based on an idea, that if the solver was able to find a solution during the grid simulation process, it should be able to find a similar solution even during the real-time experiment.



(a) The desired force is trimmed to the bounding polygon.

(b) The desired force is not modified since it is within the bounding polygon.

Figure 25: Modified P-Regulator with trimming of the desired force.

Figure 25 shows two examples. In the first case, the desired force is trimmed to the bounding box, as we proposed. And the second example presents the case when no trimming is necessary since the desired force is within the polygon. In the next section, we compare these methods in an experiment.

5.3 Control experiments

In this experiment, we compare the two mentioned control methods in a path-following scenario. The path selected for the experiment corresponds to the time-optimal path shown in Fig. 22b.

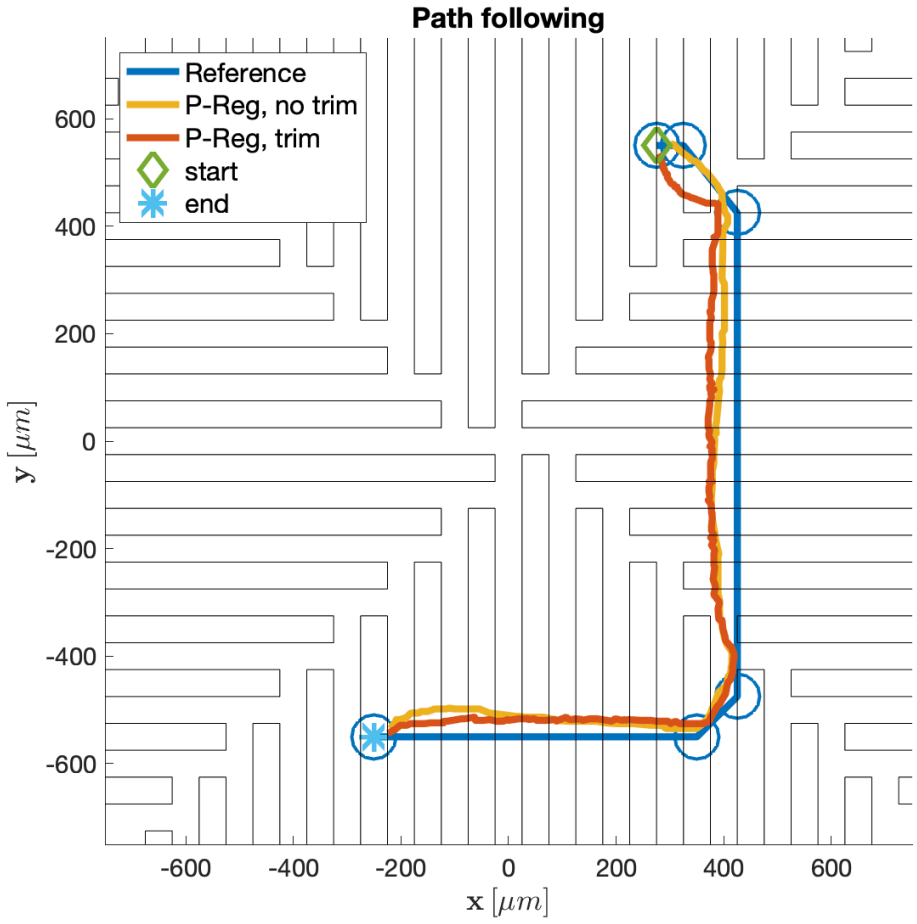


Figure 26: Comparison of the two control methods in a path-following scenario.

The trajectories of the particle for both of the methods are visualized in Fig. 26. We expected that the trajectory with trimmed regulator would be better at following the path since the desired forces should be within reasonable bounds, and thus the solver should find more stable solutions. However, we can see that both of the methods have similar trajectories. Furthermore, the total time of trajectory was longer for trimming the method since the desired forces are restricted.

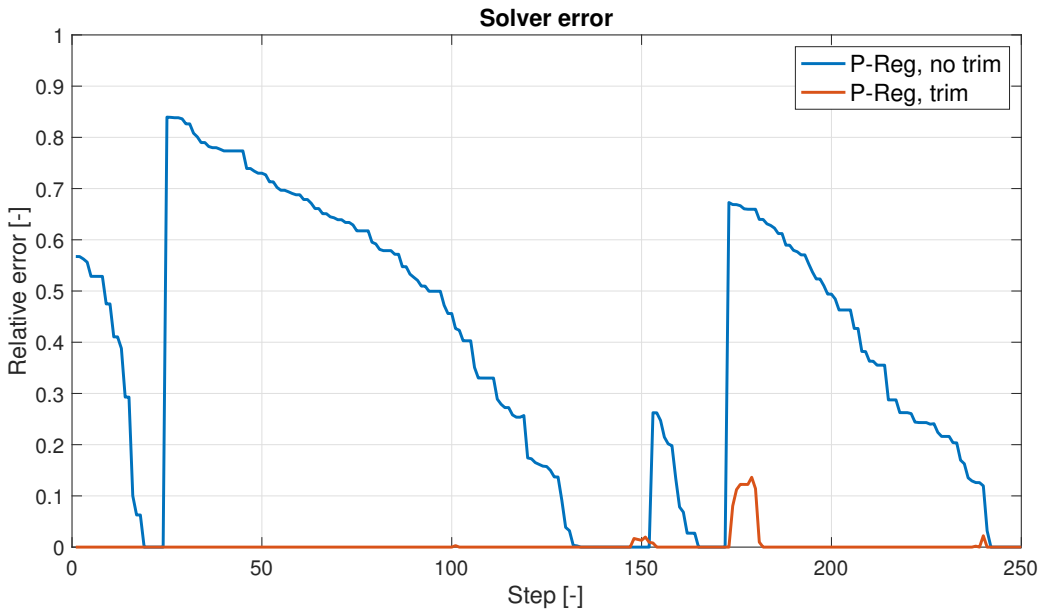


Figure 27: Comparison of the relative error between the desired force and the actual solution found by the solver.

However, when we look at the relative error of the solutions from the experiment in Fig. 27, we see that the solver found more stable solutions for the method with the trimmed desired force. Furthermore, the solver was able to find the solutions in fewer iterations. Therefore, if a more precise bounding polygon of the maximum achievable force were available, it would be possible to move the particle faster, while helping the solver to find fast and stable solutions.

Videos from the experiments are at `https://github.com/aa4cc/twinbeam-tracker`.

6 Conclusion

This thesis dealt with two problems in the task of feedback *dielectrophoretic* (DEP) micromanipulation: real-time visual tracking and time-optimal trajectory optimization. Furthermore, we had four main goals. The first goal was to develop a visual tracker that runs at least 20 frames per second (FPS) and can track at least two objects. The second goal was to implement, especially for spherical objects, a more accurate position estimation method. The third goal was to use the estimated positions from green and red channel images and estimate the height of the object. The last goal was to solve a time-optimal trajectory optimization problem for a microparticle, where the input to the system is position-dependent. All of the goals were fully achieved.

We approached the visual tracking problem with two possible solutions. First, we divided the tracking task into object detection and object tracking. For object detection, we explored two basic methods from image processing. Afterwards, we proposed a simple tracker that assumes that there is only a minimal movement between consecutive frames. This solution worked, but it required heavy parameter tuning. Therefore, we employed a more robust tracker (SiamFC) based on Siamese convolutional neural networks, which achieves results comparable to the state-of-the-art trackers. The tracker was experimentally verified. Furthermore, the time benchmark confirmed that we are able to run four trackers at around 30 FPS.

In the time-optimal trajectory optimization part, we dealt with the problem of getting a polystyrene microparticle from point A to B, with the input constraints in the dynamic model being position-dependent and complex to evaluate. We chose to restrict the motion to 8 directions and pre-compute a grid of maximum achievable force. We exploited the fact that the velocity is proportional to the force applied to the particle, and employed an optimal path-finding algorithm to obtain a time-optimal path. The time-optimal path was compared to the shortest path in an experiment, and the travel time of the optimal path was indeed shorter. As opposed to traditional time-optimal trajectory optimization methods (which suffer from being non-convex), our method guarantees to yield a globally optimal trajectory.

Finally, we proposed a modification to the current control algorithm, which lies in trimming the desired force based on the pre-computed maximum achievable force. This results in better stability of the numerical solutions but causes the motion of the particle to be slower.

References

[1] C Zhang, Khashayar Khoshmanesh, Arnan Mitchell, and Kourosh Kalantar-zadeh. Dielectrophoresis for manipulation of micro/nano particles in microfluidic systems. *Analytical and bioanalytical chemistry*, 396, 10 2009.

[2] Michael Hughes. Ac electrokinetics: Applications for nanotechnology. volume 11, 06 2000.

[3] Sina Mahabadi, Fatima Labeed, and Michael Hughes. Effects of cell detachment methods on the dielectric properties of adherent and suspension cells. *Electrophoresis*, 36, 04 2015.

[4] Jiří Zemánek, Tomas Michalek, and Zdeněk Hurák. Phase-shift feedback control for dielectrophoretic micromanipulation. *Lab on a Chip*, 18, 05 2018.

[5] Martin Gurtner. Real-time optimization-based control and estimation for dielectrophoretic micromanipulation. Master's thesis at CTU in Prague, 2016.

[6] Martin Gurtner and Jiří Zemánek. Twin-beam real-time position estimation of micro-objects in 3d. *Measurement Science and Technology*, 27, 11 2016.

[7] Jiří Zemánek. Distributed manipulation by controlling force fields through arrays of actuators. Dissertation at CTU in Prague, 2018.

[8] Viktor-Adam Koropenský. Gpu-accelerated computer vision systemfor feedback micro-manipulation. Bachelor's thesis at CTU in Prague, 2019.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[12] Mengyao Zhai, Mehrsan Javan Roshtkhari, and Greg Mori. Deep learning of appearance models for online object tracking. 07 2016.

[13] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3119–3127, 2015.

[14] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. *Arxiv*, pages 4293–4302, 10 2015.

[15] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. *arXiv preprint arXiv:1606.09549*, 2016.

[16] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[17] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. pages 4277–4286, 06 2019.

[18] Kuan-Hui Lee and Jeng-Neng Hwang. On-road pedestrian tracking across multiple driving recorders. *IEEE Transactions on Multimedia*, 17:1–1, 09 2015.

[19] T. Báča, P. Štěpán, V. Spurný, D. Heŕt, R. Pěnička, M. Saska, J. Thomas, G. Loianno, and V. Kumar. Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics*, 36(5):874–891, 2019.

[20] J. Chudoba, M. Saska, T. Baca, and L. Preucil. Localization and stabilization of micro aerial vehicles based on visual features tracking. In *2014 IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, volume 1, pages 611–616, Danvers, 2014. IEEE Computer society.

[21] Raul Mur-Artal and Juan Tardos. Orb-slam: Tracking and mapping recognizable features. 07 2014.

[22] Paresh Kamble, Avinash Keskar, and K. Bhurchandi. A deep learning ball tracking system in soccer videos. *Opto-Electronics Review*, 27:58–69, 03 2019.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[24] Giacomo Perantoni and David JN Limebeer. Optimal control for a formula one car with variable parameters. *Vehicle System Dynamics*, 52(5):653–678, 2014.

[25] Robin Verschueren, Niels van Duijkeren, Jan Swevers, and Moritz Diehl. Time-optimal motion planning for n-dof robot manipulators using a path-parametric system reformulation. In *2016 American Control Conference (ACC)*, pages 2092–2097. IEEE, 2016.

[26] Shaofeng Lu, Stuart Hillmansen, Tin Kin Ho, and Clive Roberts. Single-train trajectory optimization. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):743–750, 2013.

[27] Mohamed Kharboutly, Michael Gauthier, and Nicolas Chaillet. Modeling the trajectory of a microparticle in a dielectrophoresis device. *Journal of Applied Physics*, 106:114312 – 114312, 05 2010.

[28] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. *5th Intl Symp on Spatial Data Handling*, 11 2000.

Appendix A List of abbreviations

In Table 2 are listed abbreviations used in this thesis.

Abbreviation	Meaning
DEP	dielectrophoresis
CNN	convolutional neural network
FPS	frames per second
LED	light-emitting diode
GPU	graphical processing unit
CPU	central processing unit
RAM	random access memory

Table 2: Lists of abbreviations