University of California
Santa Barbara

# An Accessible Python Library for Magnetic Tweezer Bead Tracking

A thesis submitted in partial satisfaction
of the requirements for the degree

Bachelor of Science
in
Physics

by

Shiheng Li

June 2023

The Dissertation of Shiheng Li is approved.

Professor Omar A. Saleh

# Acknowledgements

I would like to express my sincere gratitude to Professor Omar Saleh, Frank Truong, Ian Morgan, and all the members of the Saleh Lab for the support of my research and study at UCSB. I am grateful for the opportunity to have worked with such a talented and dedicated group of people. The project was funded by the National Science Foundation (NSF).

Nobody has been more important to me than my family, especially my mom and my girlfriend Louise 🍊. I am so grateful for your love and companionship.

I would like to thank Handa, Haopu, Changyuan, Yuki, Tianyi, Yishi, and all my other friends for their talented minds and friendship. I have learned so much from all of you.

I would also like to thank my friends Tina, Cynthia, Lavender, Sashimi, Cmdblock, Vampire and all members of ITI and GoGaucho. Our collaboration brought skills for this project, meaningful contribution to the community, and most importantly, enjoyment in my leisure life.

Finally, I would like to thank all the people who have offered great help in the past. It is not possible for me to reach today without your help. I am grateful for the beautiful world in the present, and I am looking forward to the splendid future.

**Abstract**

An Accessible Python Library for Magnetic Tweezer Bead Tracking

by

Shiheng Li

Single-molecule manipulation experiments are broad and powerful tools for studying biomolecules and their interactions. A major issue is that single-molecule experiments are not widely accessible, as they either rely on expensive turn-key instruments or complex custom-built setups. This project introduces an accessible software distribution for one particularly simple single-molecule instrument, the magnetic tweezer. In a magnetic tweezer, force is applied to a biomolecule by attaching it to a magnetic micro-bead and applying a magnetic field; data is acquired by tracking the bead in 3-D with a video camera. Writing or obtaining tracking software is the biggest hurdle in using a magnetic tweezer. Here, we develop a tracking program in Python using the libraries NumPy and Taichi, for process acceleration on both CPU and GPU. The program can track 20 beads simultaneously at 60Hz with less than 10 nm error in three dimensions, comparing favorably to other implementations. We demonstrate the application of this software to biomolecular manipulation measurements.

# Contents

# Chapter 1

# Introduction

Single-molecule experiments are important techniques for our understanding of physics in biomolecules [1, 2, 3]. In comparison to macroscopic biological experiments, it is favorable for researchers to be able to manipulate and measure individual biomolecules in particular circumstances. However, limited by the size of biomolecules, it is relatively hard to probe a single molecule and run experiments on it. This chapter will introduce a simple but powerful single-molecule experimental instrument: the magnetic tweezer. It can apply force on polymers and measure their extension accurately, and therefore provide information about the underlying physical properties of the polymers [4, 5].

## 1.1 Magnetic Tweezer Experiment

A magnetic tweezer is a single-molecule experimental instrument, which can apply a pica-newton scale force on biopolymers [6, 7]. It can then measure the end-to-end extension of the polymer of interest and output the extension data in real time. This section will provide an overview of the magnetic tweezer and its applications.

A magnetic tweezer is built in the framework of an optical microscope (Figure 1.1).

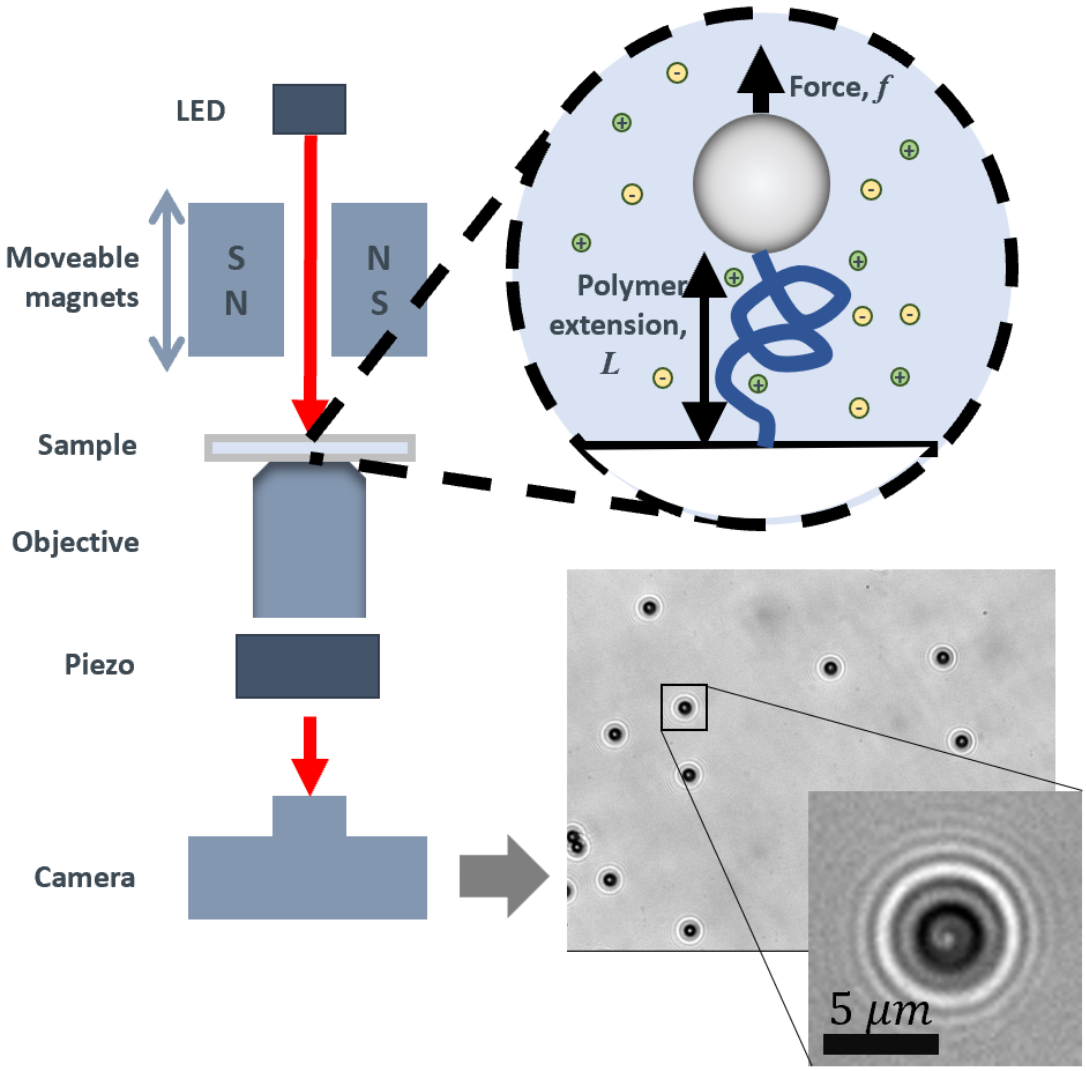


Figure 1.1: Left: a diagram of a magnetic tweezer. Top right: a diagram of a magnetic micro-bead attached to a polymer. Bottom right: an image acquired by the camera and a zoom-in around a bead. The diagrams are not to scale.

It uses an LED as a light source and a video camera as the end of the light path. In between, the light passes through several lenses before focusing on the sample and then goes through an objective lens that can be adjusted by a piezoelectric device (piezo). The sample is fixed on a platform. Without further construction, the real-time image acquired by the video camera is transferred to a computer, while the focal plane can be set by moving the objective lens under the control of the piezo.

To apply force on the polymers, permanent magnets are attached to a movable platform just above the sample (Figure 1.1), without blocking the light path. During the sample preparation, chemical reactions attach microscopic magnetic beads (about 1 $\mu$m in diameter) to one end of the polymer and attach the other end to the bottom glass of the flow cell. The magnetic fields produced by the permanent magnets can apply forces on the magnetic micro-bead and therefore on the polymer. The magnitude of the magnetic force can be precisely adjusted by moving the permanent magnets.

To obtain the end-to-end extension of the polymer of interest, the computer tracks the 3-dimensional position of the magnetic beads in real time by analyzing the image acquired by the camera. The position data can be processed by established algorithms on the power spectral density or Allan variance to extract the force and extension length [8]. To summarize, the whole procedure of data analysis from the image stream can be divided into two parts:

1. Bead tracking: input the real-time 2-dimensional image data, output the estimated 3-dimensional positions of micro-beads.

2. Force calibration: input the sequential position data for one micro-bead, output the estimated force with respect to time.

The force and extension length data can be useful in multiple experiments that explore the Physical properties and principles of biomolecules. In the following subsections, I will briefly introduce two typical applications of the magnetic tweezer in biomolecular experiments.

### 1.1.1 Folding Energy of DNA Hairpins

DNA hairpins are specific DNA structures in which one segment of single-strand DNA folded on itself with a small unpaired loop at one end (Figure 1.2 left). Under specific

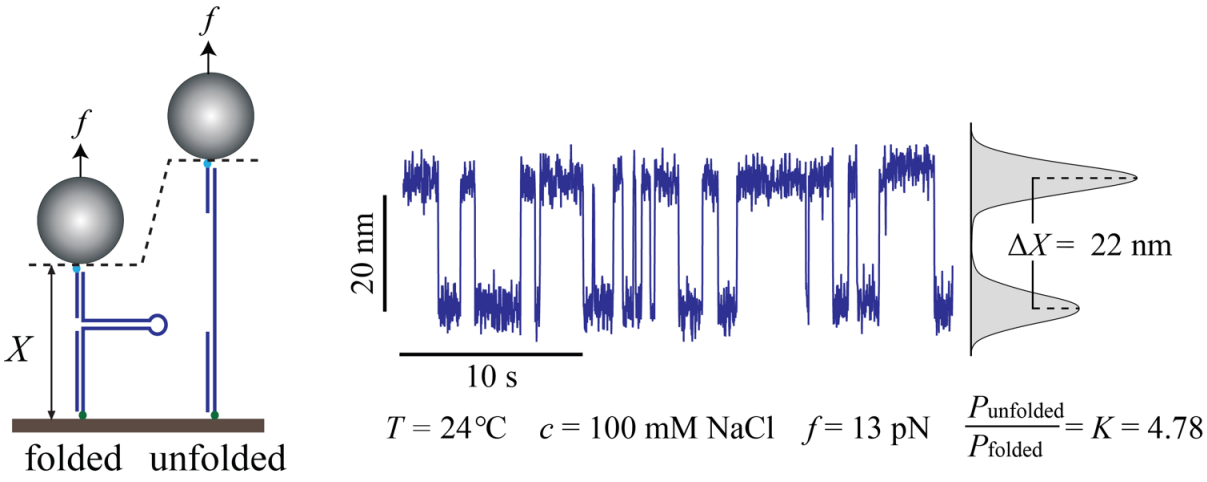


Figure 1.2: Left: a diagram of folded and unfolded DNA hairpins with magnetic beads attached. Right: a plot of extension versus time of one DNA hairpin. The diagram is not to scale [9].

forces, the DNA hairpins will fluctuate between folded and unfolded structures which can be recorded with the measurement of polymer extensions (Figure 1.2 right). With an estimation of force and the ratio between folded and unfolded time, the relevant energy scale for DNA hairpin folding can be calculated from this experiment.

As demonstrated in the data, the relevant force scale is about 10 pN, while the extension variation is within tens of nanometers [9]. Therefore, the magnetic tweezer instrument has to provide a precision of several nanometers to conduct this experiment, while exerting adjustable forces in the piconewton scale.

### 1.1.2 Persistence Length of Polymers

Another typical application of a magnetic tweezer is to measure the persistence length of biomolecules like DNA and proteins [5]. The persistence length is the characteristic length within which the polymer units have a strong correlation with each other. It is proportional to the theoretical equivalent of the random walk step size of a polymer (the Kuhn length), which is significant in understanding the physical properties of a polymer.

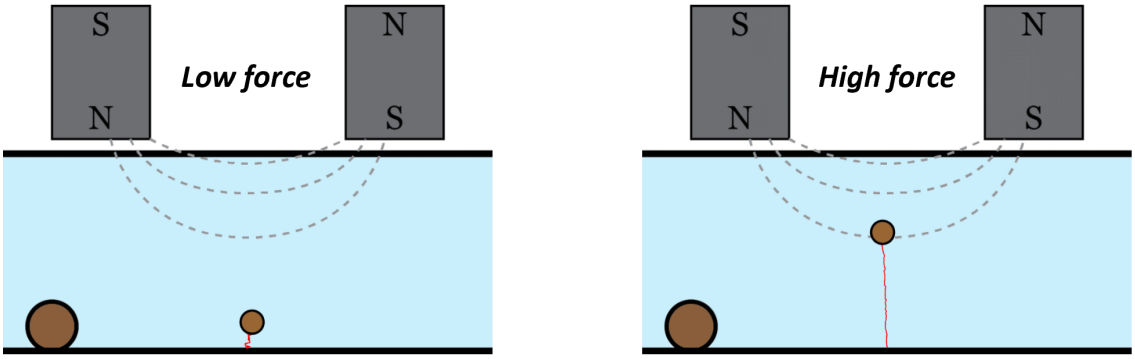


Figure 1.3: Diagrams of polymers under low (left) and high (right) forces. Magnetic beads are attached to polymer ends, and reference beads are sitting on the glass surface in both diagrams. The diagrams are not to scale.

Physically, the persistence length has a strong relation to the entropic elasticity of polymers. Statistically, there are more configurations available when the polymer extension is smaller. Thus, entropy favors smaller polymer extensions under thermodynamic laws. Thus, polymers behave like springs under external forces and tend to retract to their equilibrium extensions (Figure 1.3).

By measuring the polymer extensions under a range of forces, we can fit the force-extension data to established models of polymer physics [10]. In this way, we can estimate the persistence length from a magnetic tweezer experiment.

## 1.2 Software Implementations

Work has been devoted to the development of software for data analysis in magnetic tweezer experiments. As discussed in the previous section, the procedure of data analysis has two main parts: bead tracking and force calibration. The first part converts image data into bead trajectories (sequential beads positions), while the second part estimates the force from the position data. This section will briefly introduce some previous im-

plementations of these two parts and argue for the challenges involved in the first part, which is the main focus of my project.

In contrast to the bead tracking part, there have been many publicly accessible tools and programs developed for force calibrating. For example, the Tweezepy package by Ian Morgan is designed for force calibration from the sequential bead position data in Python [11]. It applies both the power spectral density method and the Allan variance method to determine the force from the bead traces. The error of this estimation is also calculated, which can serve as a good indication for different kinds of noise presented in the experimental system as well as the data analysis process.

However, the bead tracking part is typically highly dependent on the hardware of the experimental instruments and the computational resources and thus tends to be less organized. For example, there is an implementation of the tracking algorithms in the LabVIEW platform, which includes all the necessary hardware drivers, user interface, complicated mathematical computation, and GPU optimization [12]. This highly entangled combination of components makes it almost impossible to adjust the program at the code level or share it with other research groups.

The biggest barrier against the development of a well-organized bead tracking software is the high requirement for performance. For the biomolecular experiment, it is often demanded to have real-time feedback on the bead position and extension data. Moreover, it is not feasible to cache the image data in an experiment that runs for a long time, as the high-speed camera can produce much more data than the capacity of the data storage. Hence, it is better to be able to extract the bead position from the image in real time, and only output the 3-dimensional position of the beads. Therefore, the previous implementation tends to be lower level in programming language and architecture to maximize the performance. These low-level codes with complicated algorithms and concurrent computation are in nature difficult to maintain and distribute.

Along with the high requirement of performance, precision is also a big problem preventing us from using wide-used packages or libraries. Normally when problems like locating the center of a circle, algorithms like Hough transform can produce concise results. However, due to the small size of biomolecules and the limited resolution of the camera, magnetic tweezer experiments typically ask for a precision of 1 percent of the pixel, which is in the nanometer scale. To achieve such precision, more complex algorithms are implemented to make use of the whole diffraction pattern of the bead, which will be discussed in the next chapter. These mathematical methods involve a more intense computational burden on the hardware, which makes it even harder to reach the desired performance.

To summarize, the high requirements for performance and precision impose great challenges on the programs for bead tracking, forcing lots of the previous implementations to use low-level programming technologies and therefore deeply coupled to the specific hardware. This situation inspires my project. In this project, I tried to implement a bead-tracking program for magnetic tweezer experiments in the high-level programming language Python, while reaching the required effectiveness and efficiency by various optimizations in computational technologies as well as algorithms. With a very flexible design of interfaces, this Python package can be distributed easily and make the magnetic tweezer experiments more accessible in general.

# Chapter 2

# Methods

This chapter will explain the algorithmic methods and optimization techniques used in the Python library developed for magnetic tweezer bead tracking.

## 2.1 Setup of the Magnetic Tweezer

Before discussing of technical details of the tracking algorithm, this section will introduce the details of the magnetic tweezer setup and the functionalities of various components. During this project, a magnetic tweezer was assembled specifically for testing purposes. It has all the critical components of a fully functional magnetic tweezer.

### 2.1.1 Hardware Setup

The instrumental hardware setup of a magnetic tweezer includes two parts: the microscope and the computer.

The microscope part of the magnetic tweezer is basically a real-world implementation of the diagram in the previous chapter (Figure 1.1). For the development of the tracking program, a testing magnetic tweezer (test tweezer) was assembled (Figure 2.1).

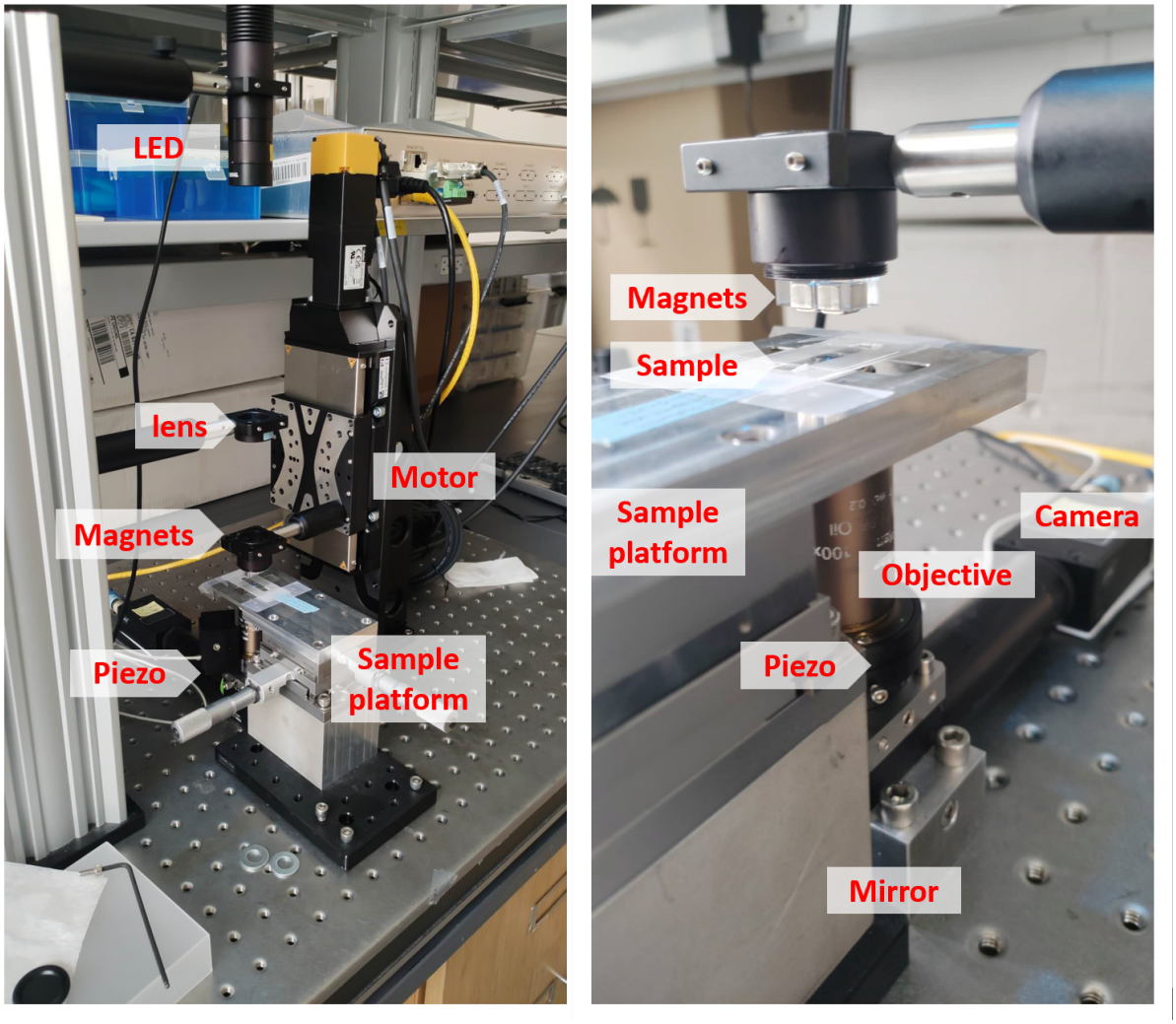


Figure 2.1: Photos of the whole testing magnetic tweezer (left) and zoomed-in near objective (right). The whole instrument is about 50 cm, and the magnets have a side length of 6.4 mm.

The whole apparatus is about 50 cm. At the very top is a LED light source, which is a blue LED with a long coherence length. It is focused with several lenses to pass through the slit between permanent magnets before reaching the sample. Two cube-shaped permanent magnets are held with a machined plastic holder, with a 1 mm slit between them. They are mounted to the electric motor controlled by the computer. The sample is typically a flow cell, and it is fixed on the sample platform with tapes. Directly

below the sample is an oil-immersion objective lens (objective) with a 100x magnification. The objective sits on the piezoelectric device (piezo) which can adjust the vertical position of the objective on the nanometer scale. For the convenience of setup and saving of space, there is a reflective mirror placed under the piezo that reflects the light in the horizontal direction. Through an eyepiece lens, the light finally goes into a monochromatic camera, which transfers the image stream into the computer. As calibrated, 1 pixel takes the length of 166 nm in the focal plane.

The computer is a typical PC with an outdated GPU chip. It connects to the camera, piezoelectric driver, and motor driver through wires. The detailed specs of computer hardware are in Appendix A.

### 2.1.2 Software Setup

For simplicity and generality, the testing instrument runs *MicroManager 2.0* on the operator system Windows 7, and the Python code interacts with the MicroManager core with the official Python package *PycroManager*.

MicroManager is an experimental software platform developed to link various devices together [13, 14]. It has the advantage that it centralizes the control of all the devices used in a magnetic tweezer while maintaining a relatively easy-to-use user interface. With the programming interface officially provided for Python, the tracking program can call the Python package PycroManager to interact with hardware instruments [15]. The operation and data fetching from the hardware are then wrapped as several functions with standardized data format to enable flexible customization. The programming interface of hardware control and tracking algorithms will be discussed in detail in the next chapter.

In the test tweezer, one instrument that is not integrated with MicroManager is the electric motor that moves the magnets. It is connected directly through the official SDK

of ACS Motion Control. The Python code interacts with the motor by calling C functions directly with *ctypes* package.

## 2.2 Scattering Diffraction of Micro-beads

The most essential difficulty of the bead-tracking algorithm is that we have to restore the 3-dimensional position information from the 2-dimensional image. The reason for the possibility of doing so lies in the scattering diffraction of the micro-beads [16].

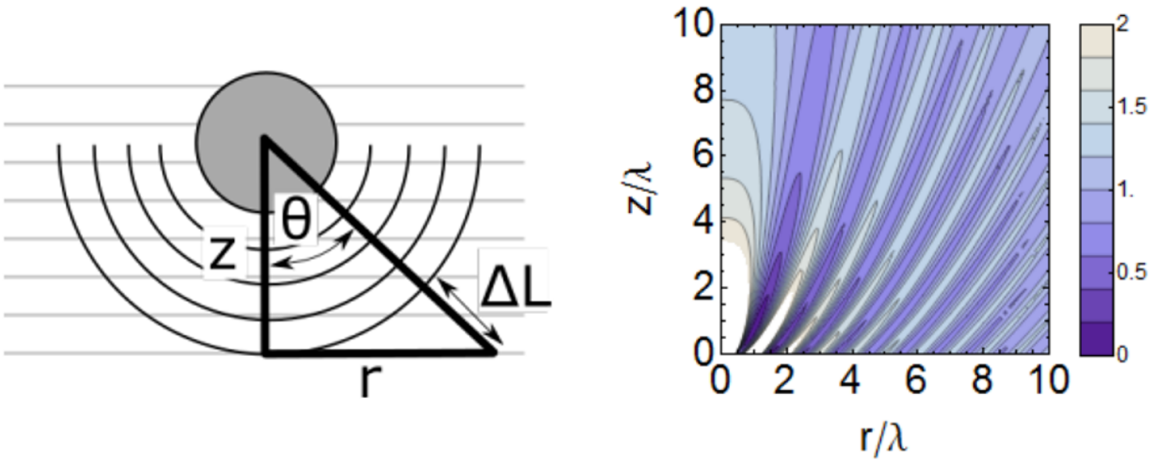


Figure 2.2: Left: a diagram of plane wave scattering of a bead. Right: a plot of theoretical interference intensity w.r.t. vertical distance $z$ and radial distance $r$ [16].

As shown in Figure 2.2 (left), when there is an incident between a plane wave of monochronic light and a micro-bead, the scattering diffraction of the light will produce an extra spherical wave. When it interfered with the original plane wave, oscillating patterns of intensity will emerge due to the difference $\Delta L$ in spatial lengths travelled by the interference waves. If the focal plane of the camera is placed below the bead with a distance $z$ and the radial distance from the projection of the center of the bead is $r$, the theoretical intensity of light detected is [16]

$$I = |E_R + E_S|^2 = \left| A_R e^{i(\omega t + kz)} + A_S e^{i(\omega t + k\sqrt{z^2+r^2})} \frac{\sqrt{2 + r^2/z^2}}{\sqrt{1 + r^2/z^2}} \frac{1}{\sqrt{r^2 + z^2}} \right|^2 \tag{2.1}$$

where the first term is reference(R) plane wave and the second term is scattered(S) spherical wave approximated with Rayleigh scatter. It is assumed that the phase is not changed during scattering, and there is no rotation in the polarization.

A plot of this intensity function versus $z$ and $r$ is plotted in Figure 2.2 (right). As indicated in the plot, the diffraction pattern varies for different heights. Therefore, by generating a lookup table for the bead, the bead position perpendicular to the focal plane can be obtained by the regression of the acquired diffraction pattern against this table.

In practice, to generate similar patterns, the light source has to be specifically chosen to be a LED with a long coherence length. If the coherence length of the light source is shorter than the difference of the light path in the interference, no meaningful interference will take place. However, if the light source is very strongly coherent as a laser, the interference from the optical instruments like lenses and mirrors will be significant and overwhelming.

The next two sections will describe the bead-tracking algorithm based on the theoretical and instrumental discussion above.

## 2.3 Bead Tracking in XY Plane

The XY plane is the 2-dimensional focal plane that is perpendicular to the light path in the magnetic tweezer. Locating the bead position in 2 dimensions is essentially locating the center of the circular diffraction pattern of the beads. However, the precision requirement asks for a limited error with about 1 percent of the pixel size, which makes

the circle-finding algorithms like Hough transforms only a rough estimation. Therefore, an advanced center-finding algorithm is applied to utilize the whole diffraction pattern of the micro-beads.

### 2.3.1 Estimation of Center Position

Before applying the fine center-finding algorithm, an estimated center point must be located within about 5 pixels of the true center. When running simulations, a Hough transform is used to find the circle center with the precision of 1 pixel. In the real experiment, beads are selected manually by users, and the first estimation of the center position is thus provided by the mouse click or number input. After the initial estimation, the tracking program will keep track of the bead and use the 2-dimensional output from the last time step as the estimation of the center position for the new time step.

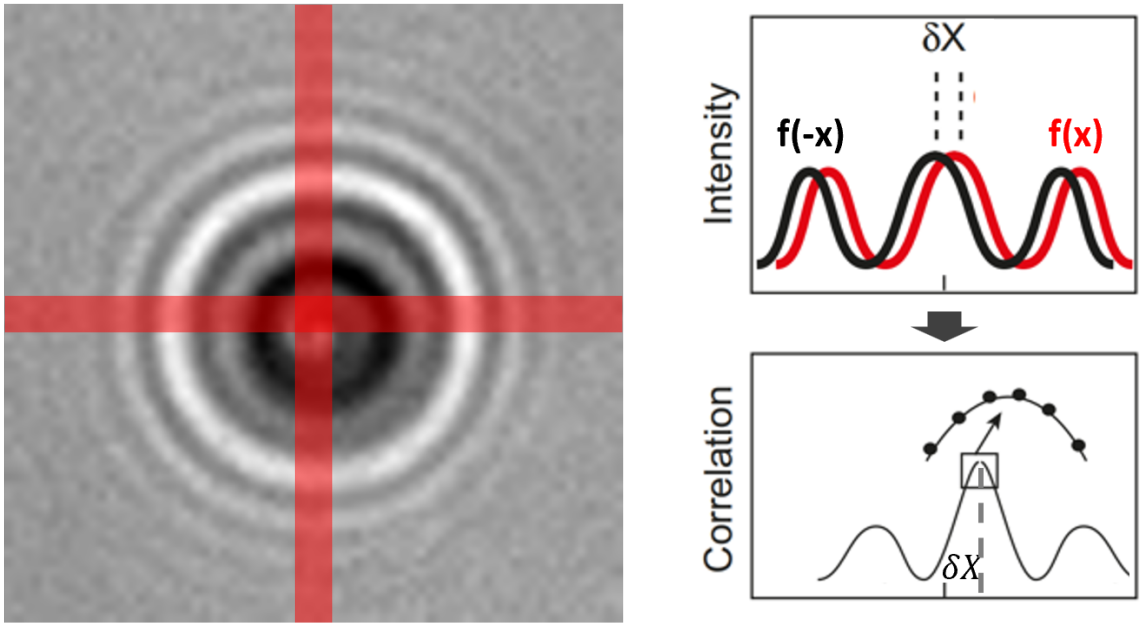


Figure 2.3: Left: a bead image cropped out from its estimated center, with vertical and horizontal slices marked in red. Right: a diagram of the cross-correlation method to calculate the symmetry center of the intensity function marked in red [17].

As shown in Figure 2.3 (left), the estimated center is not necessarily accurate enough to be treated as a good estimation of the 2-dimensional position of the bead. Therefore, further correction has to be applied.

With respect to the estimated center position, slices of pixels in both the horizontal and vertical axis are picked out (Figure 2.3 left). The other pixels are omitted in this step because they will consume unnecessary computational resources. This step has a slight difference between the CPU and GPU versions. In the CPU version, three rows or columns of pixels near the estimated center are averaged together to form a 1-dimensional intensity function (Figure 2.3 top-right in red). In the GPU version, the bilinear interpolation is used to sample a row/column of pixel points to get a smoother intensity function.

### 2.3.2 Bilinear Interpolation

Bilinear interpolation is an interpolation technique that estimates the signal value at any non-integer point within the domain of a 2-dimensional discrete signal [17]. For example, the bilinear interpolation can be used to calculate the interpolated intensity value of any point on an image, based on the intensity values of 4 nearest pixels. Given any point $(x, y)$ on the image, we pick out the 4 nearest pixels: $(\lfloor x \rfloor, \lfloor y \rfloor)$, $(\lfloor x \rfloor, \lceil y \rceil)$, $(\lceil x \rceil, \lfloor y \rfloor)$ and $(\lceil x \rceil, \lceil y \rceil)$. For simplicity of math, treat these 4-pixel points as a unit square, and their intensity function values are given as $f(0,0)$ , $f(0,1)$, $f(1,0)$ and $f(1,1)$, respectively. The bilinear interpolation of any point is then given by

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}, \quad \text{for} \quad x, y \in [0,1] \tag{2.2}$$

Note that this calculation is a matrix multiplication, and different points of $(x, y)$ can

be interpolated independently. Therefore, it is perfect for parallel computation and GPU acceleration.

### 2.3.3 Symmetry Center of the Intensity Function

From either averaging rows/columns or bilinear interpolation, a 1-dimensional slice of pixels is picked out with respect to the estimated center of the bead image. To determine a more accurate center position, the task is to find the symmetry center of these horizontal and vertical slices of pixels. It can be calculated from the cross-correlation method [17].

The cross-correlation function between two functions $f, g$ is a representation of their similarities, defined for both continuous and finite discrete functions as

$$(f \star g)(\tau) = \int_{-\infty}^{\infty} f(t)^* g(t+\tau) dt \quad \text{or} \quad (f \star g)[n] = \sum_{m=0}^{N-1} f[m]^* g[(m+n)_{\text{mod } N}] \tag{2.3}$$

By definition, if a function is a perfect symmetry with respect to the origin, then its mirror image w.r.t. origin will perfectly overlap with itself. If the symmetry center of the function is at $x = s$, then there will be a $\delta X = 2s$ shift between its mirror image $f(-x)$ and itself $f(x)$ (Figure 2.3 top-right). Hence, the cross-correlation function between the function and its mirror image will have a peak at $\delta X = 2s$ that indicates the maximized overlapping (Figure 2.3 bottom-right).

In practice, the cross-correlation function between the pixel slice and its mirror image is calculated. Then, 5 points near the maximum value are fit to a parabola to find the peak of the cross-correlation function. The displacement of the symmetry center from the origin is approximated by half of the peak displacement from the origin in the cross-correlation function.

### 2.3.4 Optimization and Fitting Error

The mathematical method described above can offer a reasonable approximation of the symmetry center of a 1-dimensional signal. However, for better precision and performance, several other techniques have been applied in the 2-dimensional tracking.

In the CPU version, the cross-correlation function can be calculated faster with the Fast Fourier Transform (FFT), which runs recursively and reduce the time complexity of the computation. Mathematically, for the signal function $f(x)$ and its Fourier transform $\mathcal{F}[f(x)] = F(s)$, definition Eq. 2.3 directly implies

$$\mathcal{F}[f(-x) \star f(x)] = \mathcal{F}[f(-x)]^* \mathcal{F}[f(x)] = F(s)F(s) = F^2(s) \tag{2.4}$$

Therefore, by squaring the Fourier transform of the signal and then applying the inverse Fourier transform, the cross-correlation function can be calculated with FFT, which is a great optimization in performance. But FFT has a recursive structure and can hardly be accelerated by GPU. Since the signal here has a relatively small size (100 px per slice), it is even faster to compute the cross-correlation without FFT. Thus, the cross-correlation function is directly computed by definition in parallel in the GPU version.

Besides performance optimization, there is a precision problem to be solved. When fitting the 5 points to a parabola to find the peak position in the cross-correlation function, the linear regression fitting method tends to output the peak position closer to an integer data point, causing a bias when the true position is not aligned with an integer point.

Since this fitting error is minimized with the peak perfectly aligned with an integer point, the simplest solution to this problem is to shift the peak of the signal to an integer point iteratively. In the first iteration, a symmetry center is calculated with the algorithm

discussed above. According to this result, the signal is shifted back with this amount, and its symmetry center is calculated again. In the new iteration, the peak will be better aligned with an integer point and the discrete fitting error will be smaller. Therefore, higher order corrections of the symmetry center can be computed with the iteration of signal shift and symmetry center finding.

This shift of signal can be accomplished in two ways, and they are implemented in CPU and GPU versions of the tracking program. In the CPU version, this space shift $\Delta$ is applied in the Fourier space by multiplying the Fourier image by a phase shift,

$$F(s) \to F(s)e^{2\pi i s\Delta} \iff f(x) \to f(x+\Delta) \tag{2.5}$$

In the GPU version, however, the Fourier transform is not an intermediate step, so the signal shift is directly computed by resampling the slices of the bead image with all the sample points shifted from the last iteration. Although this naturally leads to more computational work, it can provide more precise results since more information is fetched from the image in the resampling process for each iteration. Accelerated by the parallel processing in GPU, such amounts of interpolation can be computed with sufficient efficiency and therefore are favored. In practice, the iteration described here is only run twice for reasonably good results in bead tracking in the XY plane.

To summarize, the fine calculation of the center position in the XY plane is reduced to the calculation of the symmetry center of horizontal and vertical slices w.r.t. an estimated center position, which is performed by the cross-correlation method with various optimization and iteration. For data analysis in real-world experiments, reference beads will be mounted on the sample flow cell bottom. By subtracting the position of the reference bead, noises from instrumental instability can be reduced.

## 2.4 Bead Tracking in Z Direction

The Z direction is the direction perpendicular to the focal plane. Namely, it is the height of the bead in the sample. As discussed previously, the information on the Z position is inferred from the diffraction pattern of the bead. In comparison to the 2-dimensional tracking in the XY plane, the tracking in the Z direction is more complicated.

The tracking algorithm for the Z position is dependent on the XY position of the bead, which is described in the previous section. Therefore, this section will treat the 2-dimensional position $(x, y)$ as the given information. The Z direction bead tracking involves calibration that is done before the actual experiment, which will also be included in detail in this section.

### 2.4.1 Radial Intensity Profile

Before any calibration or fitting, a method has to be developed to extract the information in the diffraction pattern of a bead image, just like the slices in the XY tracking.

For better precision, this representation of the diffraction pattern should include most information in the bead image. Previously this is done by accumulating all the pixels within a fixed radial range from the bead center. However, bilinear interpolation is better at producing a smooth function while the computation is accelerated by GPU.

The tracking algorithm proposed here uses bilinear interpolation to resample the bead image. Sample points form a grid in polar coordinates with the origin fixed at the 2-dimensional center of the bead computed in the XY tracking [17]. The XY position of the sample points relative to the center of the bead is:

$$x = r\left(\frac{R}{N_r}\right)\cos\left[\theta\left(\frac{2\pi}{N_\theta}\right)\right] \qquad r \in \{0, 1, 2, \cdots, N_r - 1\} \tag{2.6}$$

$$y = r\left(\frac{R}{N_r}\right)\sin\left[\theta\left(\frac{2\pi}{N_\theta}\right)\right] \qquad \theta \in \{0, 1, 2, \cdots, N_\theta - 1\} \tag{2.7}$$

where $R$ is the radial size of the diffraction pattern in pixel, $N_r$ and $N_\theta$ are the sampling number in radial and angular directions.

These sample points are an evenly distributed grid in polar coordinates, with the step in radial direction $R/N_r$ and the step in angular direction $2\pi/N_\theta$. In practice, $R = 40$ and $N_r = N_\theta = 80$. In other words, the useful bead radius is about 40 pixels, while 80 points are sampled on each of the total 80 radius segments. In the radial direction, two sample points are equivalent to a pixel distance, which can offer better extraction of information in the diffraction pattern. The parameters here can be adjusted for specific optimization in effectiveness or performance.

After the bilinear interpolation of these sample points, values from each radius with different angular directions are averaged together. Therefore, the final result is a 1-dimensional discrete function of intensity profile $I[r]$, where $r$ is the index of the radial sample point, and the spatial distance is equal to 0.5px.

However, this intensity profile is not good enough to represent the diffraction pattern, as there are noises caused by lighting conditions. Therefore, the intensity profile is filtered in the Fourier space. The intensity profile will mirror itself to make a symmetry signal. The tracking algorithms will then apply a Hanning window over the major positive frequency domain, for example, $[3, 40]$ out of $[-80, 80]$, and remove all the other frequency amplitude. After the inverse Fourier transforms, the result becomes a complex function $\tilde{I}[r]$, and it is then cut off from below, for example, $r > R_f = 10$, where $R_f$ is the forget radius. This step is supposed to filter out all the noisy frequencies while ignoring

the complicated and irregular diffraction pattern at the center of the bead image. Note that since the Hanning window is only applied to positive frequencies, the signal is not symmetric in Fourier space and thus the filtered intensity profile $\tilde{I}[r]$ is a complex-value function.

This completes the computation of the filtered intensity profile. For each bead image, a filtered intensity profile $\tilde{I}[r]$ is computed for the following usage. This step is the most time-consuming part due to the large amount of bilinear interpolation, and thus it is the most important part of optimization discussed later.

### 2.4.2 Calibration

Although a filtered intensity profile is computed from the bead image, it does not provide any direct information on the Z position of the bead. To get that information, it has to be compared with a lookup table generated before the real experiment. This lookup table is called calibration, which is a stack of filtered intensity profiles with different Z positions of the bead.

To generate a lookup table, the bead is fixed first. Commonly this is done by applying a large force on the bead attached to a polymer by bringing the magnets close to the sample. The bead will have little Brownian motion and it will not affect the diffraction pattern. The objective is then moved to the desired position, which adjusts the Z position of the focal plane. In this way, the relative height of the bead relative to the focal plane is changed. Since the objective is moved by piezo which has a nanometer precision, the relative Z position of the bead can be tuned in a nanometer scale.

For each position of the objective, 5 images are taken. The images are taken after the instrument rests from the objective movement that lasts for about 0.02 s. The intensity profiles are calculated respectively for these 5 images and they are averaged together to

output 1 filtered intensity profile. This average can further remove the noises caused by optical fluctuation.

In one example, the objective moves 100 steps in the Z direction with each step size of 100 nm. In total, it calibrates for a 10 $\mu$m range in the Z direction and produces a stack of 100 calibration data, which is plotted in Figure 2.4. It is an approximation to the theoretical prediction plotted in Figure 2.2 (right).

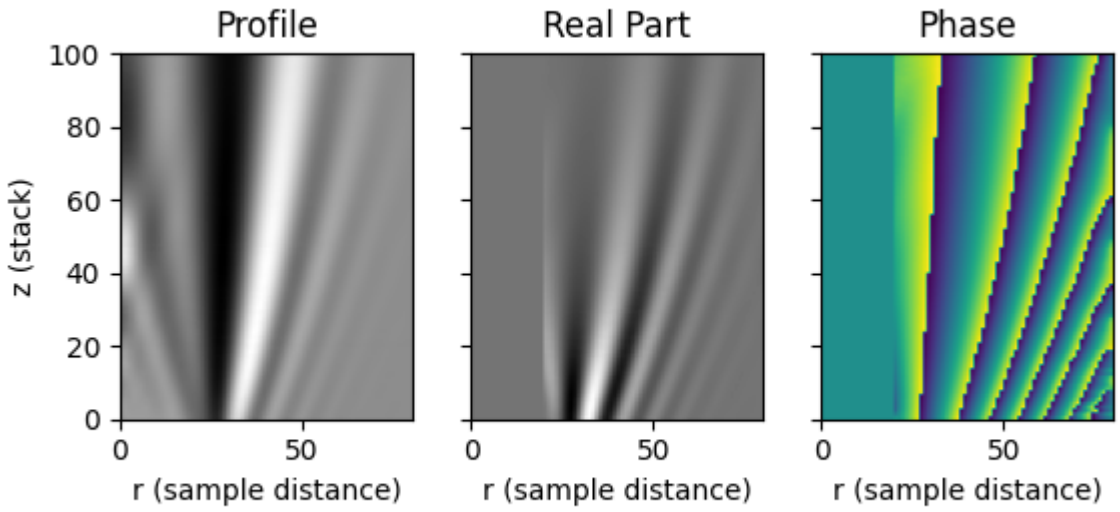


Figure 2.4: Calibration data for one reference bead (d = 2.8 $\mu$m). The vertical axis is 100 stacks in the Z direction (step size = 100 nm), and the horizontal axis is measured in the radial distance between sample points (0.5 px). Left: original intensity profile. Middle: real part of the filtered intensity profile. Right: phase (mod $2\pi$) of the filtered intensity profile.

The calibration process is done for each bead of interest. The calibration data is stored in the program memory for lookup in the experiment.

### 2.4.3 Regression for Z Position

In the real-time experiment, a new filtered intensity profile will be computed for each bead in each frame. This filtered intensity profile is compared to the calibration data to determine the best place it can fit. This regression is done by the following algorithm.

First, a close neighborhood is determined by the real part of the filtered intensity profile $\mathrm{Re}\,\tilde{I}[r] = R[r]$. A $\chi^2$ value is calculated between $R[r]$ with each Z stack in the calibration data $R_z[r]$, as

$$\chi_z^2 = \sum_{r=R_f}^{N_r} (R[r] - R_z[r])^2 \tag{2.8}$$

where $R_f$ is the forget radius of the filtering and $N_r$ is the sample number in the radial direction.

The $z_0$ value with the minimum $\chi_{z_0}^2$ is considered to be an estimation of the Z position and the neighborhood is defined as $[z_0 - 2, z_0 + 2]$. Fine regression will be applied in this neighborhood to compute the precise Z position of the experimental filtered intensity profile.

Regarding variation of the lighting amplitude, the magnitude and the real part of the complex function $\tilde{I}[r]$ might change during the experiment, making them unstable in regression. Therefore, phase data (Figure 2.4) is focused in this step. Within the neighborhood for each z, $\Delta\phi_z$ is calculated as a weighted average of the difference in phase between the experimental $\tilde{I}[r]$ and the calibration $\tilde{I}_z[r]$. The averaging weight is the product of amplitude, so that phase difference with larger amplitude is emphasized.

$$\Delta\phi_z = \frac{\sum_{r=R_f}^{N_r} A[r]A_z[r](\theta[r] - \theta_z[r])}{\sum_{r=R_f}^{N_r} A[r]A_z[r]} \tag{2.9}$$

where

$$\tilde{I}[r] = A[r]e^{i\phi[r]} \quad \text{and} \quad \tilde{I}_z[r] = A_z[r]e^{i\phi_z[r]} \tag{2.10}$$

Within the neighborhood $[z_0 - 2, z_0 + 2]$, the values of $\Delta\phi_z$ are plugged into a linear regression for these 5 points to find the expected $\tilde{z}$ that makes $\Delta\phi_{\tilde{z}} = 0$. This value is finally reported as the Z position of the bead in this frame of the experiment.

To summarize, the Z position tracking is based on the filtered intensity profile which characterizes the diffraction pattern of the bead image. It is fitted linearly against predetermined calibration data by the weighted phase difference to get the expected Z position of the bead. This concludes the discussion of the algorithm for bead tracking.

## 2.5 Parallel Computation

As discussed above, the bead tracking algorithm is complicated with many steps and various computations. It is therefore not easy to optimize the computation with GPU acceleration. Traditional Python packages including NumPy and just-in-time GPU compilers for NumPy like Numba are not flexible enough to implement complex algorithms without jumping back and forth between GPU and CPU. The GPU overhead caused by the initiation of a GPU thread is also overwhelming when starting a GPU function for each step in the algorithm.

The desired architecture of the implementation is to pass the image data into GPU-accelerated code in the beginning and retrieve the result of the bead positions in the end. Furthermore, for the accessibility of the program, it is better to be compatible with various GPU chips. One satisfactory solution is to use the *Taichi* package.

### 2.5.1 Taichi Language

*Taichi*[18] is in fact a language that shares almost the same syntax as Python. It is widely used for graphic computation and Physical simulation, as it has advantages in performance and simplicity, and it offers high-level performance optimization utilities. After installing the taichi package in Python, developers can write Python-like code in Python code files while they are compiled into low-level C++ programs and accelerated by GPU automatically when it runs for the first time. This is known as the just-in-time compilation [19]. Another advantage is that Taichi will automatically determine the available GPU chips without specifying. The only thing to do is to import the taichi package while marking the Python functions that are sent to be compiled, as

```
import taichi as ti
ti.init(arch=ti.gpu) # running on GPU

@ti.kernel # Mark the function
def func():
    # The function will run on GPU
```

Therefore, the Taichi language is suitable for this complicated algorithm of bead tracking, while preserving the simplicity of code compared to other low-level programming.

### 2.5.2 Image Data Transfer and Batch Processing

For the compilation of Taichi language for GPU acceleration, structural data storage like arrays and fields must be defined statically. And the image data acquired from the camera has to be copied into this taichi `field` so that it can be accessed by the Taichi code running on GPU.

Naturally, there are two ways to transfer the image data into the taichi `field`. An obvious solution is to transfer the whole image and a list of estimated bead centers into the taichi `field`. In this way, the taichi code has access to the whole image and it will locate the estimated bead centers by itself. Although this provides simplicity in code structure, it costs unnecessary storage for the empty and useless pixels in the big image acquired by the camera. Often only a small fraction of the image is used during the computation.

A better solution is to crop the image around the estimated bead center and only transfer the bead images (Figure 2.5 (a)) into the taichi `field`, which is now a 3-dimensional array to store a list of bead images. Despite the extra computational effort put into the cropping, fewer data are copied into the taichi `field` and therefore it takes less memory space and less time for data transfer.

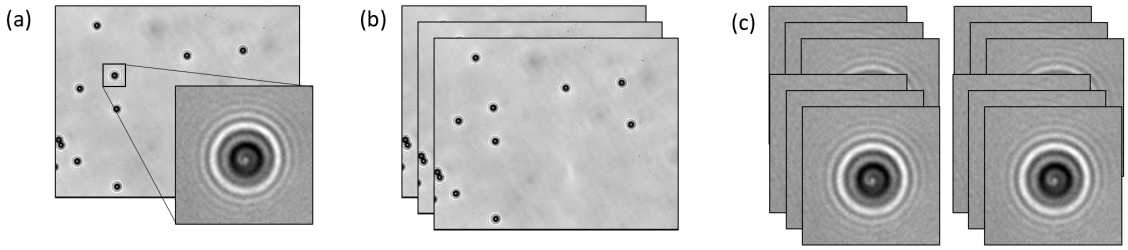


Figure 2.5: (a) A whole camera image and one bead image cropped according to the estimated center. (b) Several whole images can be grouped together for batch computation. (c) Instead of passing the whole image into the tracking algorithm, only groups of bead images are transferred to the GPU program.

Another problem with GPU acceleration is the GPU overhead. Various codes are coordinated to initiate a GPU-accelerated function, and this initiation costs some time before any meaningful information is processed. Therefore, it is better for GPU to process data in larger amounts instead of dividing them into small packages. Therefore, to

achieve maximized performance, several frames of images can be cached and passed into the GPU code together (Figure 2.5 (b)) [16]. Considering the batch processing, cropping bead images has a big advantage, since all the bead images from various frames can be combined together to run concurrently (Figure 2.5 (c)), making the program even more efficient and well-organized in the batch processing situation.

### 2.5.3 Computational Staging

Technically, the parallel computation in the Taichi language is achieved through `for` loops since the outer-most `for` loop will be compiled as parallel computing automatically by the Taichi compiler. In most cases, the outer-most loop is looping around all the bead images, which makes them computed in parallel automatically. The whole tracking algorithm is naturally divided into many steps. For the optimization of efficiency, it is better for the code to access only part of the memory during each single step. Therefore, the tracking program running on the GPU is specifically designed to perform tasks one by one, but simultaneously complete each task for all the bead images. In other words, instead of each bead image going through the whole algorithm by itself in parallel, all beads are coordinated to run the same step in the algorithm collectively. The steps are like the stages in a rocket launch that are placed in strictly sequential order.

For example, in the XY tracking algorithm, the XY position is calculated as the symmetry center of the horizontal and vertical slices. The GPU program will first run the bilinear interpolation to generate the slices for all bead images. The slices are stored in a pre-defined taichi `field` for temporary storage. It is only when slices are generated for all bead images, that the program will start to compute the symmetry center of each slice in parallel. Although this might sound inefficient as the program might wait for a slow computational thread among fast ones, the delay is in fact negligible due to the

extremely strong capability of parallel computation in GPU chips. The advantage of the locality of memory access and more organized logic outweighs this slight delay.

Even more importantly, another advantage of computational staging is that it provides extra flexibility in the concurrent design of each stage. At some stage, the parallel number is significantly larger than the number of bead images. For example, to extract the radial intensity profile of each bead image, 6400 points have to be resampled with bilinear interpolation. In this stage, all the points in all bead images are independent of each other, which makes them suitable for a super large parallel computation. Indeed, all the bilinear interpolation in all beads images is executed concurrently, storing the intensity profile data in another pre-defined taichi `field`.

To summarize, the parallel computation is implemented with Taichi language embedded in Python, and it is optimized with various techniques like batch processing and staging. With these methods, the GPU-accelerated code offers great performance with a Python interface, while preserving the code's simplicity at the same time.

This concludes the discussion about the algorithmic and technical details of the bead-tracking program. It is a combination of complex mathematical theories of optical diffraction, delicate algorithms, and cutting-edge computational optimization technologies. The next chapter will provide some demonstration of the effectiveness of the tracking program built into this project.

# Chapter 3

# Results

This chapter will provide detailed demonstrations of the effectiveness of the tracking program developed with the methods described in the previous chapter. The effectiveness includes both tracking precision and tracking performance based on experimental data, and the accessibility of the program, illustrated by the description of programming and user interfaces.

## 3.1 Tracking Precision

The beads used in the testing experiment are reference beads that are melted and mounted on the surface of the bottom glass in a water-filled flow cell. The magnetic bead attached to the polymer is the MyOne C1 bead with a diameter of about 1 $\mu$m, and the reference bead has a diameter of 2.8 $\mu$m. With calibration, the pixel size of the acquired images is about 166 nm.

The precision of XY tracking is tested by tracking 2 reference beads for 1000 frames with a frame rate of 50 Hz. To eliminate the noise caused by the instrumental oscillations, the relative positions of the two beads are calculated and plotted in Figure 3.1. As

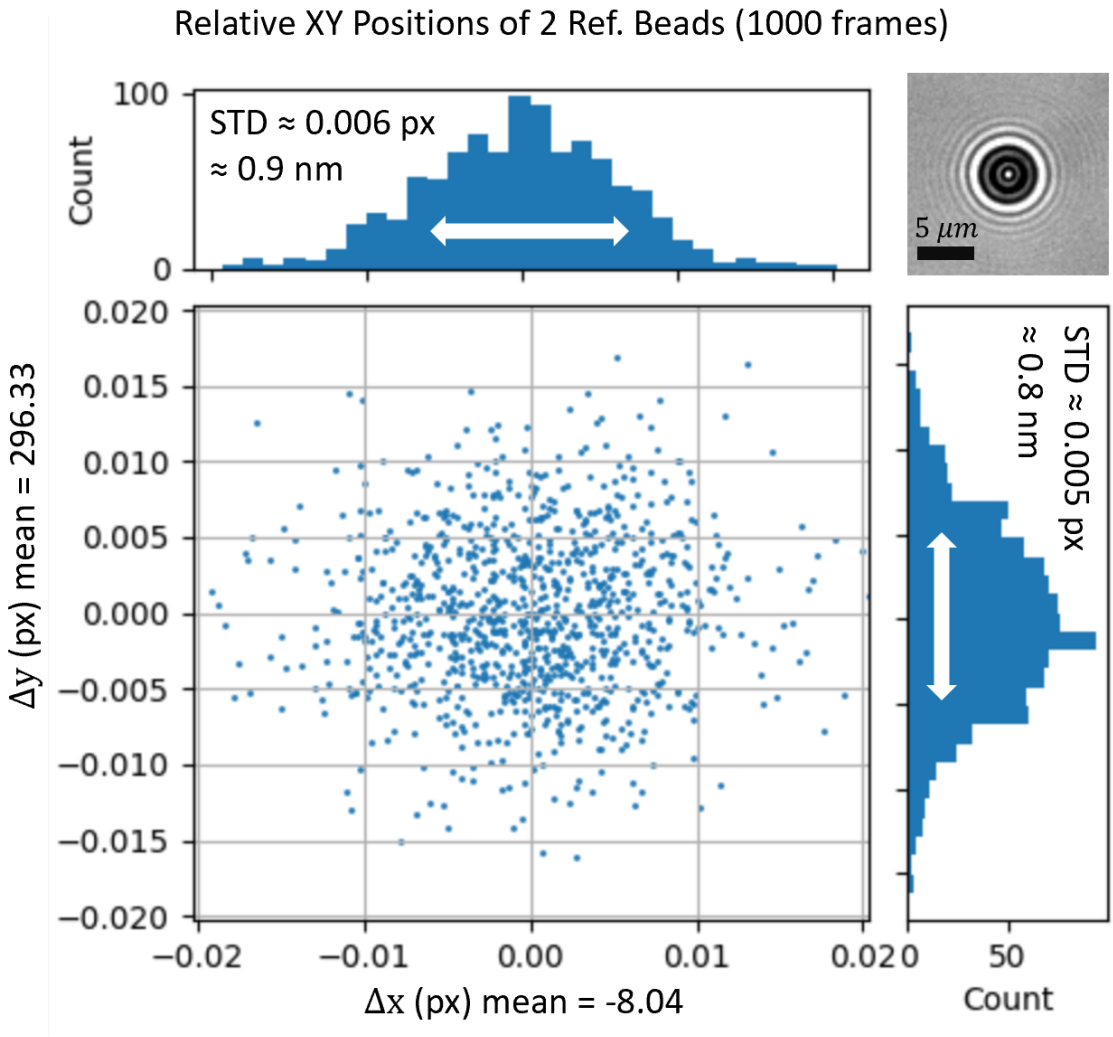


Figure 3.1: A plot of relative XY positions of 2 reference beads, tracking for 1000 frames (about 10 s). The differences in the X and Y positions between the two beads are plotted in the scatter plot. The distribution of the relative X and Y positions are plotted in the histograms. The standard deviation in both X and Y positions is less than 1 nm. One example of bead images is displayed in the top-right corner.

demonstrated in the plot, both X and Y positions have a standard deviation within 1 nm. The reference bead image in this test (Figure 3.1 top-right) is extraordinarily well-focused with the significant bright diffraction ring and central bright spot. In real experiments, the precision of XY tracking may be slightly worse due to varying lighting conditions and the moving focal plane. The precision tested with magnetic beads under

various lighting conditions is around 2 nm.

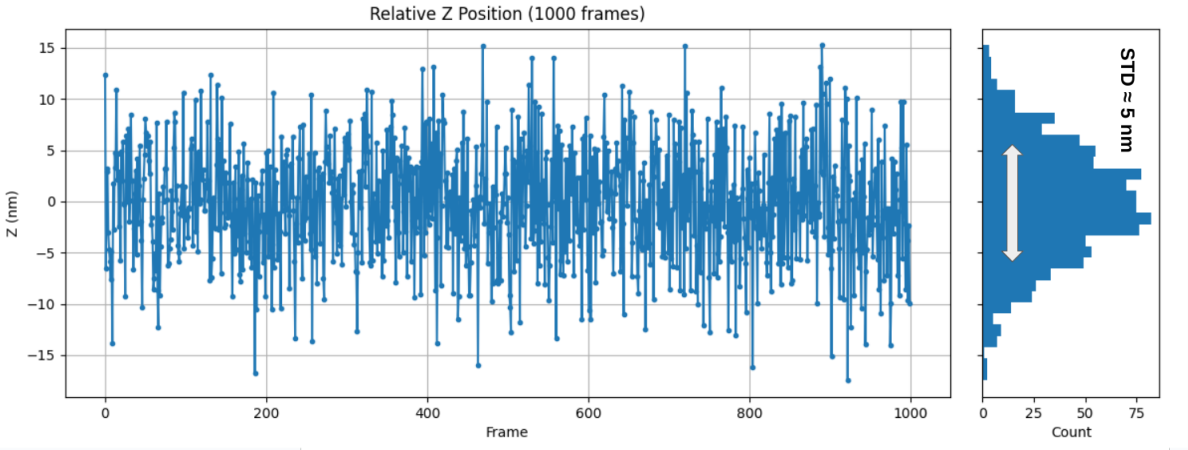


Figure 3.2: A plot of relative Z positions of 2 reference beads, tracking for 1000 frames. The distribution of the relative Z positions is plotted in the histograms. The standard deviation in Z positions is roughly 5 nm.

A similar test has been run for the Z position tracking. The calibration adapted in this experiment includes 100 stacks in the Z direction with a step size of 100 nm. The calibration data is very similar to Figure 2.4. Two reference beads mounted on the bottom of the flow cell are calibrated and then tracked. When running the tracking, the piezo is moved to a height of 2000 nm (around the 20th stack in the calibration data) to simulate a real experiment within the calibration range. The differences in the Z position between the two reference beads are plotted in Figure 3.2. Since they are mounted on the flow cell, they are supposed to have the same Z position and the difference should be zero. As demonstrated in the plot, the tracking program indeed gives a result with a mean value very close to zero and a standard deviation around 5 nm.

The precision in Z tracking demonstrated here is sufficient for the single-molecule experiments discussed in the introduction. For example, in the DNA hairpin experiment shown in Figure 1.2, a 5-nm precision should be good enough to distinguish a DNA hairpin molecule's folded and unfolded states with $\Delta X = 22$ nm.

Since the Z position is computed from the regression of calibration data, we can

examine the bias of the Z tracking at different Z positions by real-world experiments. In this experiment, only one fixed reference bead is calibrated. After calibration, the piezo is moved to a particular Z position, and the bead is tracked for 100 frames. The trajectory data of the bead is then compared to the Z position of the piezo acquired from the device, and the differences between the tracking results and the piezo readings are plotted as the bias at this particular piezo position. Then the piezo is moved to another Z position and this tracking and comparing process is repeated.

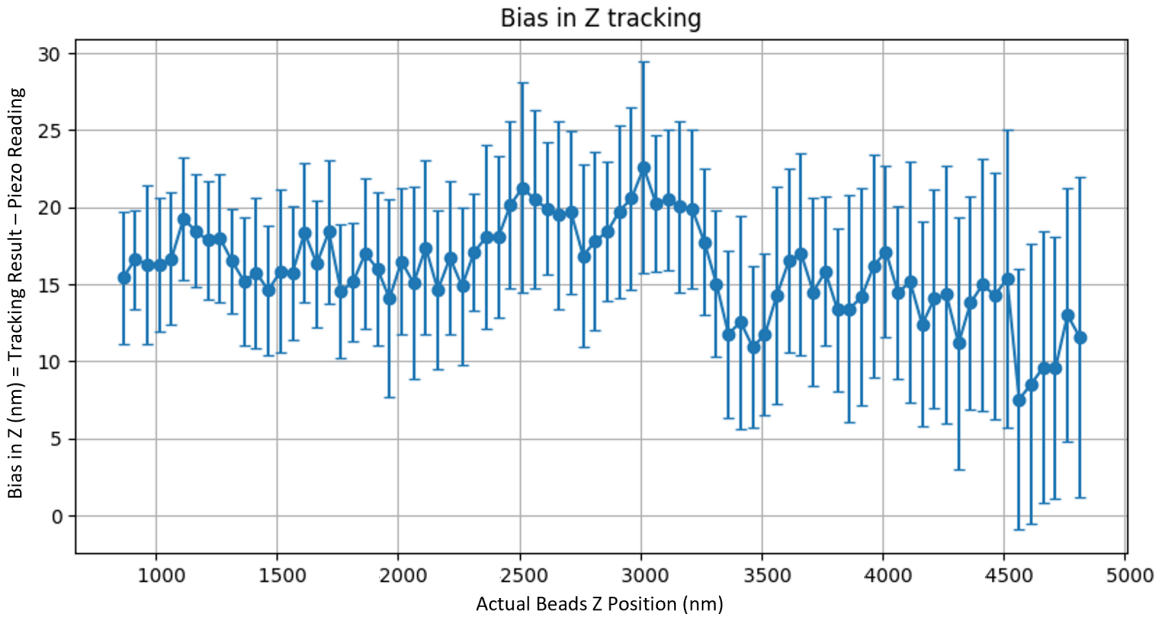


Figure 3.3: A plot of bias in Z positions. The horizontal axis is the Z position of the piezo, and the calibration range is from 0 to 5000 nm. The plotted points are the mean values of the bias, and the error bars represent the standard deviation in the tracking results.

The bias data is plotted in Figure 3.3. As demonstrated in a 5000 nm calibration range, the bias has a relatively stable mean value around 15 nm which is due to the drift of the instrument, and the standard deviation is between 5 and 10 nm at all testing Z positions. The bias varies more than the precision shown in Figure 3.2, because the fluctuations of the instrument are directly reflected in the bias data. When doing real

biomolecule experiments, this bias caused by instrumental noise can be reduced greatly by measuring the reference beads fixed on the flow cell. Overall, the tracking program offers a result of less than 10 nm tracking precision and accuracy for the Z position of bead tracking.

## 3.2 Tracking Performance

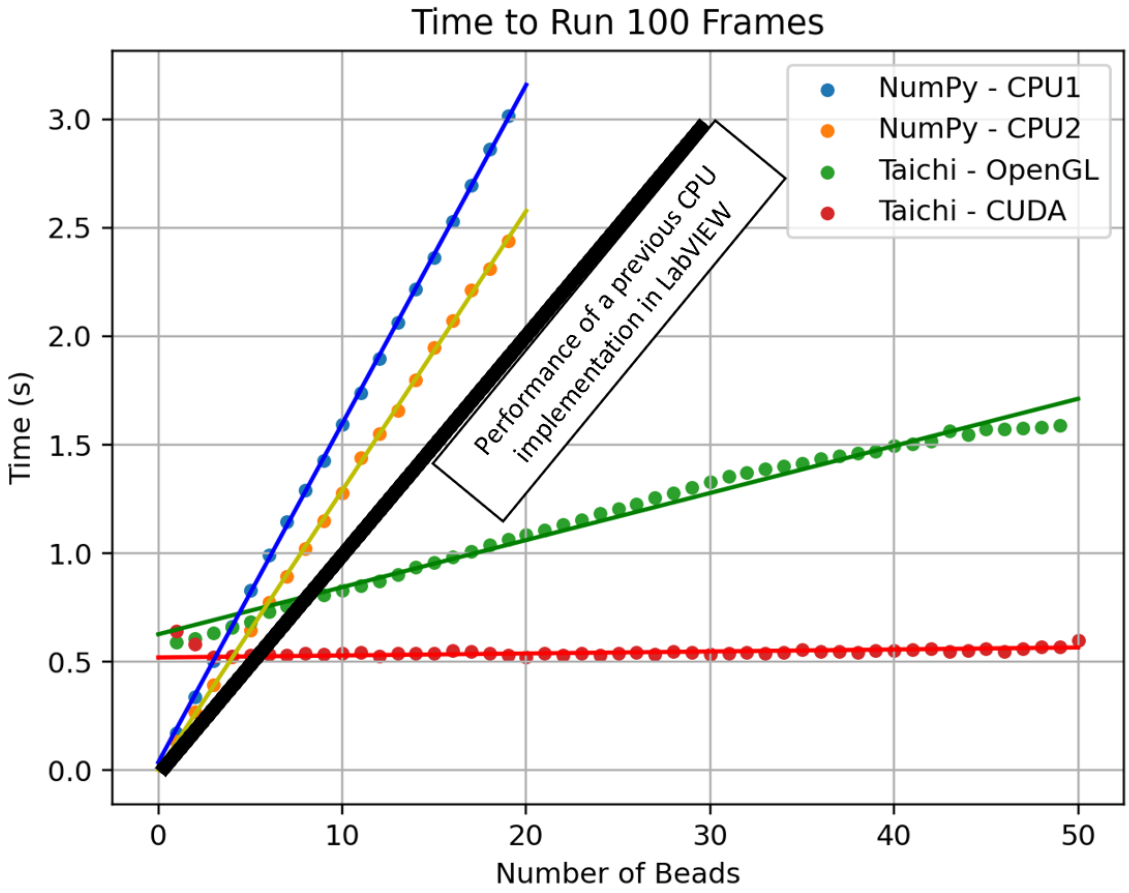


Figure 3.4: A performance measurement by simulation: total running time for 100 frames is plotted and fitted versus the number of beads. The same simulation is performed on 4 different CPUs and GPUs with NumPy and Taichi implementations respectively. The benchmark in black is the performance of the previous implementation in LabVIEW [16].

As described in the introduction, making the tracking program run fast and efficiently is critical. The tracking program's CPU and GPU versions are tested on simulated images on two computer sets. The testing is performed as a measurement of the total time to run 100 frames without any batch computation. The data obtained are plotted in Figure 3.4. The detailed processing chip information is listed below:

1. NumPy - CPU1: CPU version running on Intel Core i7-8650U CPU @ 1.90GHz
2. NumPy - CPU2: CPU version running on Intel Core i7-2600K CPU @ 3.40GHz
3. Taichi - OpenGL: GPU version running on NVIDIA Quadro 4000
4. Taichi - CUDA: GPU version running on NVIDIA GeForce GTX 1050

Also, the black line indicates a benchmark of the previous implementation in LabVIEW. As clearly demonstrated in the plot, the CPU version Python program optimized with NumPy is slightly less efficient than the low-level compilation code. The GPU version optimized with Taichi has an overhead shown as the interception on the time axis. However, despite the overhead, the GPU version has a better performance than the previous implementation.

With a better GPU chip (NVIDIA) and the CuDA platform, the marginal cost in computational time for each bead is almost negligible within a reasonable range of bead numbers, as almost all the data are processed in parallel. It is even hard to test the limit of the tracking program. It is clear that the tracking program can run up to 100 frames per second for 15 beads even on the old computer used in the testing magnetic tweezer. Obviously, if the batch computation strategy is applied to save the unnecessary GPU overhead time, a higher frame rate can be achieved with the GPU-accelerated tracking program.

## 3.3 Programming Interface

Besides sufficient effectiveness in precision and performance, the accessibility of the tracking program is also critical in this project. Therefore, a very flexible design of the programming interface is implemented, enabling the tracking program to interact with customized device drivers and provide fine-tuning for the parameters used in the tracking algorithm.

To enable the interaction between the tracking program and the instrumental hardware, several device driver functions have to be prepared in Python:

1. `GetImg()`: return a 2-dimensional array of the image acquired by the camera.
2. `GetZ()`: return the Z position of the piezo in nanometers.
3. `SetZ(z)`: set the piezo Z position.
4. `SetMotor(pos)`: (optional) set the position of magnets.

The architecture of the tracking program is illustrated in Figure 3.5. As a user, you can use either Python code or the provided UI to bridge the gap between data acquired by the device drivers and the tracking program. If the instruments and programs are well established and the performance has to be optimized, then you can execute each step of the experiment with your own code by calling the functions provided by the tracking package. The core tracking program will not operate devices directly, your experimental code will control everything and pass data from the hardware to the tracking program.

The programming interface is presented in the core tracking package `T`. The beads can be represented by objects generated from constructor `T.Bead(X, Y)`. For example, you can create an object for one bead by calling `T.Bead(123, 456)`, specifying the estimated XY position of the bead at $(123, 456)$. The parameters used in the tracking algorithm,

like the forget radius $R_f$ and the filter hanning window span, can also be set on the bead objects. All the tracking steps can be done as operations on a list of beads. For example, `T.XY(beads, images)` tracks the list of beads in the list of images. This will not only return the XY positions of the beads in the images but also update the internal state of the beads to record the estimated XY positions. The detailed reference of functions is in the README of the GitHub repository provided in the next chapter.

Besides the core tracking program, some auxiliary modules are also written in the repository. For example, the `mm` module is a quick demo of the device driver functions when using MicroManager 2.0 and PycroManager as the interface between Python and devices. The `utils` module provides several functions that are useful in plotting data like the bead image and calibration data.

However, if you want to observe the real-time video stream of the camera or select beads by mouse-clicking, a user interface module is also provided, which is demonstrated in the next section.

## 3.4 User Interface

The provided user interface (UI) is very simple and is built by the Taichi GUI package. If initiated, the UI functions will take over specific experimental steps like bead selection and it will automatically call the device driver functions. Therefore, after importing the `UI` module, experimental steps can be performed with minimal codes.

One tiny example is provided in Figure 3.6. In this example, the executed Python script has only 4 lines in total (not counting import statements). The first line initiated a bead-selection UI dialog which is displayed in Figure 3.6 as well. In this dialog, users can select beads by left-clicking the mouse and cancel the selection by right-clicking the mouse. The XY positions of the selected beads will be automatically tracked in

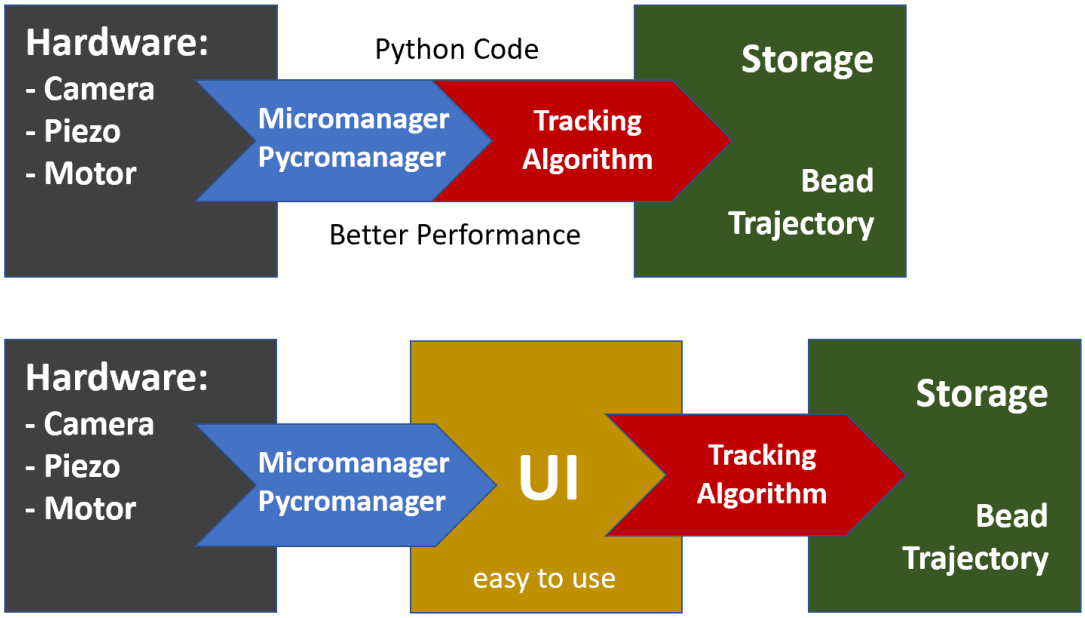


Figure 3.5: A diagram illustrating the architecture of the tracking program. The tracking program can interact with customized device drivers with an optional user interface. Tracking results are output as bead trajectory data directly.

real time. When completed, just close the dialog, and the list of bead objects will be automatically returned and assigned to the variable `beads` in the code. The second line of code does a similar job by opening a dialog that displays the live video stream through calibration. The third line finalizes the calibration by calling the core tracking function. The fourth line, therefore, starts a 3-dimensional real-time tracking, with the bead trajectory returned after closing.

As demonstrated in these 4 lines of code, all necessary device driver functions are passed into `UI` module and they are called by the UI functions automatically, offering an easy-to-use user interface. Since the UI functions are all independent of each other, you can adapt UI during some steps and use Python code in others. For example, you can call `UI.SelectBeads` first and perform all the experiments without UI. This flexible design of programming and user interfaces maximizes the accessibility of this tracking

```
1  beads = UI.SelectBeads(T, mm.Get)
2  UI.Calibrate(beads, T, mm.Get, mm.GetZ, mm.SetZ)
3  T.ComputeCalibration(beads)
4  trace = UI.Track(beads, T, mm.Get)
```

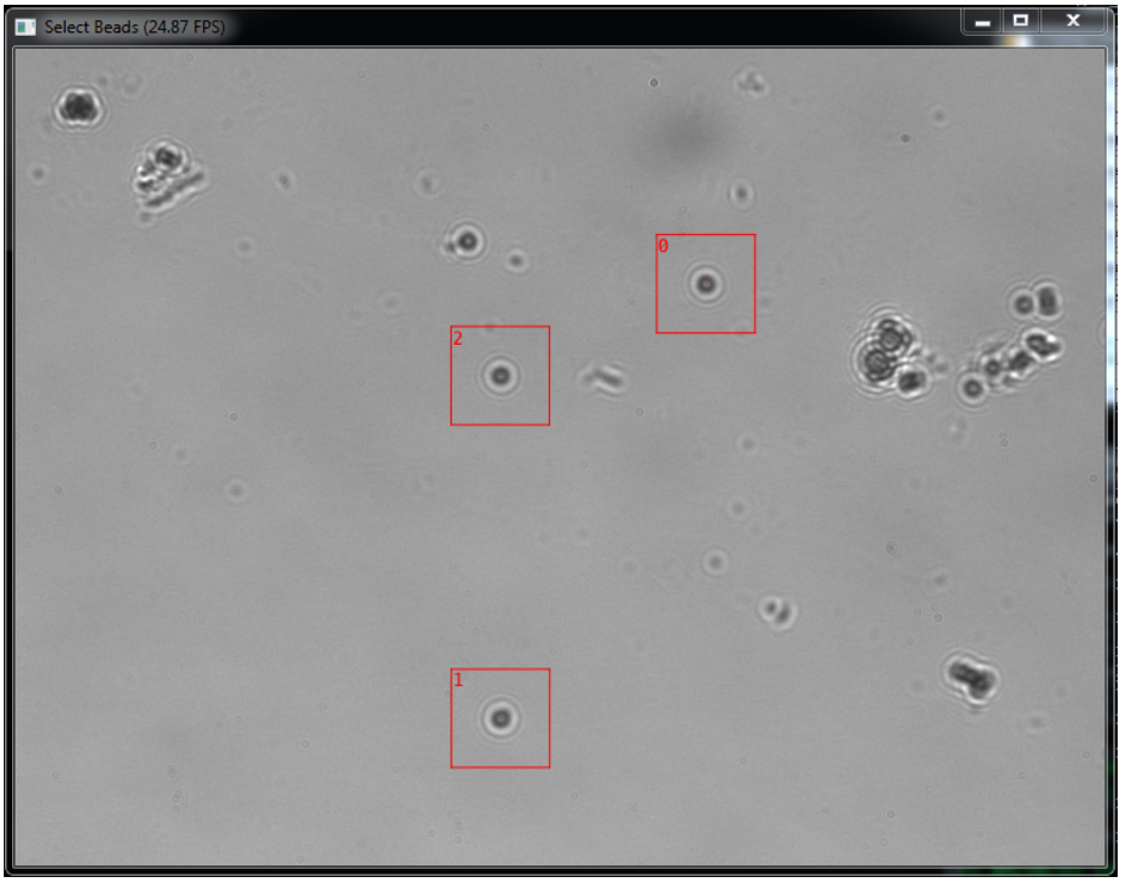


Figure 3.6: A screenshot demo of the user interface: just 4 lines of code (top) can initiate a fully functional tracking process. The Select Beads window initiated with the first line is displayed (bottom), on which the user can select beads by mouse clicking.

program in various circumstances.

# Chapter 4

# Conclusion

With complicated algorithms, cutting-edge optimization technologies, and a flexible design of software architecture, this project creates a Python program that provides high-precision and high-performance bead-tracking in magnetic tweezer experiments.

To truly realize the accessibility of this tracking program, it is fully open source and under further testing and development. We hope that the tracking program created in this project can be useful in more single-molecule research.

## 4.1 Open Source Code

The source code of the tracking program is published on GitHub. It can be directly accessed by the URL `https://github.com/phantomlsh/magnetic-tweezer`.

The code is published with MIT open-source license. All kinds of open-source contributions including but not limited to debugging, improvement, issue, and discussion are welcome.

The simplest way to use the code is to download or copy the Python modules from the code repository, which are intentionally made compact and convenient to use.

## 4.2 Test with Biomolecular Experiments

Further work might be done to perform biomolecular experiments with this tracking program. For example, it can be used in the measurement of the folding energy of DNA hairpins and the persistence length of double-strand DNA molecules. These experiments have verified results and good theoretical models, which can be used to verify the real-world application of the tracking program created in this project.

## 4.3 Machine Learning Algorithms

Although the algorithm described in this project is sufficiently effective, there may also be alternative solutions that are faster and simpler. One potential solution is to train a specific machine learning model since the whole bead-tracking problem is indeed a pattern recognition problem.

One barrier to the machine learning solution is that it might not be able to compute the results of bead position in high precision with a scale of 1 percent of the pixel size. Typical recognition task like face recognition only requires the pixel-scale location of a specific pattern. This problem may be solved by creating particular high-precision training data from theoretical generation.

Another problem is that it may be hard to generalize the trained model to different instrumental conditions. Even if one specific model is trained for one instrument, the lighting conditions will still vary from experiment to experiment. Therefore, maybe fine-tuned training of the machine learning model has to be done in the calibration step, or a simulated variation of lighting conditions can be used in the training data.

However, if these problems are solved, machine learning solutions would be favored as they are structurally simpler and the neural network, especially the convolutional

neural network (CNN), can be better accelerated on GPU and even specific AI chips. And relevant coding packages and environments are mature in the artificial intelligence industry. It is highly possible that further development in machine learning algorithms will create an opportunity that makes bead tracking much easier.

To conclude, the bead-tracking algorithm and program in this project can serve the current demand in the magnetic tweezer experiments, and it is built to be accessible and flexible for easier distribution and further development. It can also help to verify alternative solutions like machine learning. We hope that this project can be helpful for other researchers to push the limit of our understanding of biomolecules.

# Appendix A

# Hardware Specs

The hardware instruments used in the test magnetic tweezer:

The objective lens: 100x magnification.

The piezoelectric device: MCL NanoDrive Z Stage (MCLS01841) with MCL driver.

The computer connected to the test magnetic tweezer runs on the CPU chip Intel Core i7-8650U CPU @ 1.90GHz, the GPU chip NVIDIA Quadro 4000, an 8-GB RAM, and a 256-GB hard-drive disk.

# Bibliography


[1] M. C. Leake, *The physics of life: one molecule at a time*, 2013.

[2] D. Riveline, *'single molecule': theory and experiments, an introduction*, *Journal of Nanobiotechnology* **11** (2013), no. 1 1–10.

[3] M. F. Juette, D. S. Terry, M. R. Wasserman, Z. Zhou, R. B. Altman, Q. Zheng, and S. C. Blanchard, *The bright future of single-molecule fluorescence imaging*, *Current opinion in chemical biology* **20** (2014) 103–111.

[4] R. Sarkar and V. V. Rybenkov, *A guide to magnetic tweezers and their applications*, *Frontiers in Physics* **4** (2016) 48.

[5] T. R. Strick, J.-F. Allemand, D. Bensimon, A. Bensimon, and V. Croquette, *The elasticity of a single supercoiled dna molecule*, *Science* **271** (1996), no. 5257 1835–1837.

[6] K. Kim and O. A. Saleh, *A high-resolution magnetic tweezer for single-molecule measurements*, *Nucleic acids research* **37** (2009), no. 20 e136–e136.

[7] C. Gosse and V. Croquette, *Magnetic tweezers: micromanipulation and force measurement at the molecular level*, *Biophysical journal* **82** (2002), no. 6 3314–3329.

[8] B. M. Lansdorp and O. A. Saleh, *Power spectrum and allan variance methods for calibrating single-molecule video-tracking instruments*, *Review of Scientific Instruments* **83** (2012), no. 2 025115.

[9] A. Dittmore, J. Landy, A. A. Molzon, and O. A. Saleh, *Single-molecule methods for ligand counting: linking ion uptake to dna hairpin folding*, *Journal of the American Chemical Society* **136** (2014), no. 16 5974–5980.

[10] O. A. Saleh, *Perspective: Single polymer mechanics across the force regimes*, *The Journal of chemical physics* **142** (2015), no. 19 194902.

[11] I. L. Morgan and O. A. Saleh, *Tweezepy: A python package for calibrating forces in single-molecule video-tracking experiments*, *Plos one* **16** (2021), no. 12 e0262028.

[12] R. Bitter, T. Mohiuddin, and M. Nawrocki, *LabVIEW: Advanced programming techniques*. Crc Press, 2006.

[13] A. Edelstein, N. Amodaj, K. Hoover, R. Vale, and N. Stuurman, *Computer control of microscopes using μmanager*, *Current protocols in molecular biology* **92** (2010), no. 1 14–20.

[14] A. D. Edelstein, M. A. Tsuchida, N. Amodaj, H. Pinkard, R. D. Vale, and N. Stuurman, *Advanced methods of microscope control using μmanager software*, *Journal of biological methods* **1** (2014), no. 2.

[15] H. Pinkard, N. Stuurman, I. E. Ivanov, N. M. Anthony, W. Ouyang, B. Li, B. Yang, M. A. Tsuchida, B. Chhun, G. Zhang, *et. al.*, *Pycro-manager: open-source software for customized and reproducible microscope control*, *Nature methods* **18** (2021), no. 3 226–228.

[16] B. Lansdorp, *Pushing the envelope of magnetic tweezer resolution*. University of California, Santa Barbara, 2015.

[17] M. T. van Loenhout, J. W. Kerssemakers, I. De Vlaminck, and C. Dekker, *Non-bias-limited tracking of spherical particles, enabling nanometer resolution at low magnification*, *Biophysical journal* **102** (2012), no. 10 2362–2371.

[18] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, *Taichi: a language for high-performance computation on spatially sparse data structures*, *ACM Transactions on Graphics (TOG)* **38** (2019), no. 6 1–16.

[19] "Taichi lang: High-performance parallel programming in python." `https://www.taichi-lang.org/`, 2022. Accessed: 2023-06-07.