Thermal Motion

James Li (1007974248)
Claire Zhang (1008001822)

February 8, 2023

I Introduction

Brownian motion, as coined in 1828 by Robert Brown, is the random movement of a particle in fluid as a result of surrounding viscous forces interfering with its velocity, position, and thermal energy. Without observing the motion of a Brownian particle, Einstein postulated that microscopically visible particles suspended in a fluid will undergo a similar random motion that can be observed as per the molecular-kinetic theory of heat. According to Einstein's 1905 paper regarding the molecular-kinetic theory of heat, this molecular motion is a result of heat generated in the fluid. In addition, he postulated that the motion of the particles of following the molecular-kinetic theory of heat were identical to the motion of a Brownian particle, leading to the motivation behind the experiment.

When a microscopically visible particle is placed in a fluid, fluid molecules erratically collide with the particle, resulting in a force that evidently changes the velocity of the particle. However, this velocity is immediately dampened by the viscous forces of the fluid acting on the particle. By nature, the direction and magnitude in which the collisions occur is random − in a sense that the prior collision with the particle has no effect on the following collision. Thus, the particle will undergo a "random walk", formally denoted as Brownian motion.

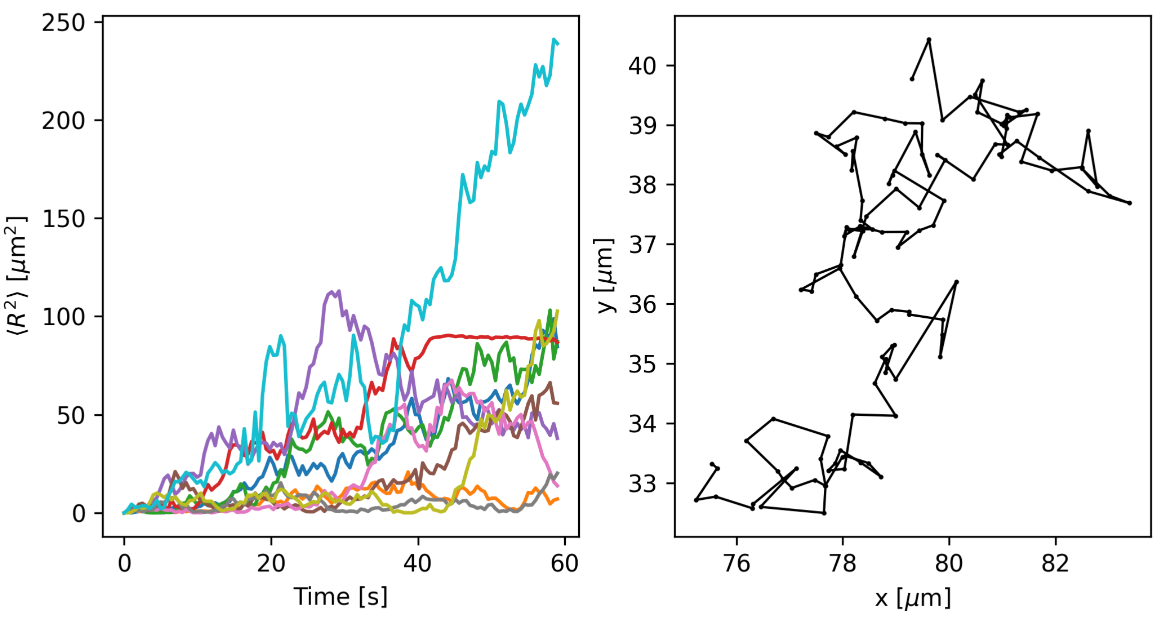


Figure 1: The left figure tracks the mean squared displacement, $\langle R^2 \rangle$, of all 10 of the Brownian particles. The right figure tracks the position of a Brownian particle in Cartesian coordinates over the elapsed time.

Following this discovery, Einstein also theoretically proved that the mean squared displacement of the Brownian particle increased linearly with time, denoted by:

$$\langle x^2 \rangle = 2Dt \tag{1}$$

where D is the diffusion coefficient and t is the time elapsed.

Newton's Second Law of motion for the microscopically visible particle relates the net force with the viscous drag force and a postulated thermal force.

$$m\frac{d^2x}{dt^2} = -\gamma\frac{dx}{dt} + X \tag{2}$$

Solving Equation 2 will yield Einstein's relation, relating the diffusion coefficient with Boltzmann's constant, k, temperature, T, and Stokes' drag:

$$\frac{d(x)^2}{dt} = \frac{2kT}{\gamma}t \qquad \longrightarrow \qquad \langle x^2\rangle = \frac{2kT}{\gamma}$$

Setting Equation 1 equivalent to the expression above:

$$2Dt = \frac{2kT}{\gamma}$$

$$D = \frac{kT}{\gamma} \tag{3}$$

$$\gamma = 6\pi\eta r \tag{4}$$

The experimental objective of this lab is to study the Brownian motion of polystyrene beads under a fluorescence-illuminated microscope by tracking the change in position of particle for a period of time. As such, Boltzmann's constant and Avogadro's number can be experimentally obtained using two methods – a linear fit (1D "random walk") and a Rayleigh distribution curve fitting.

In addition, when fitting the motion of the bead to a Rayleigh distribution in Method 2, it is important to note that the probability distribution function of determining the distance of a particle travelled over a time interval t can be represented as a 2D Gaussian distribution.

$$P(x, y; t) = \frac{1}{4\pi Dt}e^{-\frac{x^2+y^2}{4Dt}}$$

After simplifying with polar coordinates where $r^2 = x^2 + y^2$, the Gaussian Distribution can be expressed as:

$$P(r; t) = \frac{1}{4\pi Dt}e^{-\frac{r^2}{4Dt}} \tag{5}$$

By applying a Maximum Likelihood Estimation, the value of D can be calculated using Python code, and as such, the Boltzmann's Constant can be determined.

In short, the purpose of this report details the experimental calculation of the Boltzmann's constant (k) by finding the diffusion coefficient (D) using 2 methods: a linear distribution and Rayleigh distribution. Then, the calculated results are validated with Avogadro's number to verify its accuracy.

II Procedure

Materials List

Microscope Slide and Cover	Pipette	Double-Sided Tape	Petroleum Jelly
Water-Diffused Polystyrene Bead Solution	Microscope	Fluorescence Illumination (X-Cite Box)	LabView and Image Object Tracker

Table 1: List of Materials used in Thermal Motion experiment.

- Pixel Resolution of LabView (and its error — see Figure 7): 1 pixel = 0.12048 μm ± 0.003 μm

- Reading Error of Image Object Tracker when choosing the center of bead tracked: The uncertainty of the radius of the bead can postulated as half the radius since when choosing the center of the bead manually, the location is likely to be within a half-radius distance away from the center at all times, thus $\boldsymbol{r_b = 0.95 \pm 0.48 \mu m}$.

II.I Data Collection Method

See Table 1 for materials mentioned in the procedure below.

To prepare the microscope slide for inspection, first, firmly lay two parallel 4 cm-long pieces of double sided tape 2 cm apart, no creases or air bubbles. Then, perpendicular to the tape, place 2 lines of petroleum jelly to create a 4 cm by 2 cm rectangular barrier. Next, using the pipette, put 50 µl of the bead solution into the sealed slide space and place a cover glass over the space; make sure to avoid fingerprints on the cover.

To collect data on the microscope (see Figure 2), switch on fluorescence illumination (X-Cite box), increase the illumination intensity, initialize the phase ring to (Ph1) and place the sample on the microscope stage. Using LabView phase microscopy on the computer, digitally focus the microscope on a bright bead point on the dark green background by adjusting "Gain Value" and "Brightness Values". Then, start the Microscope Camera Controller on Multiple Image Capture mode with 120 images per 2 second frames; for undisturbed data, avoid choosing beads in close proximity to others to avoid path interference. Next, using the Image Object Tracker computer application, select the center of the bead and record the x/y pixel values of the moving bead at each frame. These are the same (x,y) values recorded as raw data in Appendix and Data (Section A).

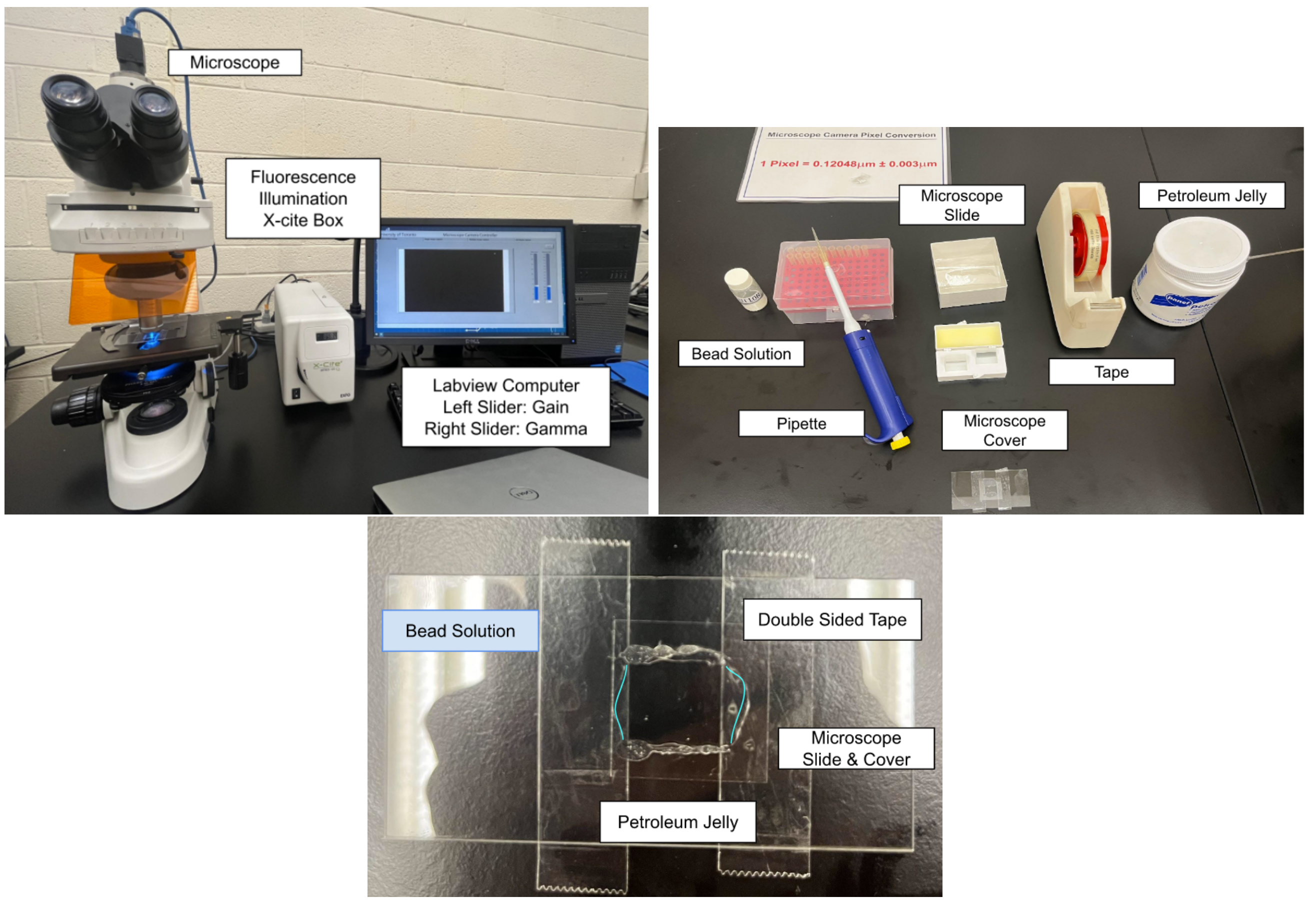

Microscope
Fluorescence Illumination X-cite Box
Labview Computer Left Slider: Gain Right Slider: Gamma
Microscope Camera Pixel Conversion
1 Pixel = 0.12048μm ± 0.003μm
Microscope Slide
Petroleum Jelly
Bead Solution
Tape
Pipette
Microscope Cover
Bead Solution
Double Sided Tape
Microscope Slide & Cover
Petroleum Jelly

Figure 2: Materials used in procedure.

III Analysis

III.I Data Analysis Methods

After repeating steps 6-8, collecting data for 10 different beads. There are two methods of solving for Boltzmann's constant, k — 1.380649 $\times 10^{-23} \frac{\mathrm{J}}{\mathrm{K}}$.

Method 1: 1D Random Walk

One way to interpret Brownian motion is to picture particles on a "random walk" where at each time step τ, the particles move a distance δ either left or right with equal probability. To determine the position of the i-th particle within the sample space after n steps can be expressed as:

$$x_i(n) = x_i(n-1) \pm \delta$$

Now, if one were to determine the average position of the sample space of particles, one can express it as a summation of the above equation divided by the total number of particles, N.

$$\langle x(n) \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i(n-1) \pm \delta \tag{6}$$

The $\pm\delta$ term can be neglected in this expression as it is a constant with equal probability of being positive and negative. Thus, taking the average of that term would result in a zero sum and evaluating the sum would eventually result in an expression as such:

$$\frac{1}{N}\sum_{i=1}^{N} x_i(n-1) \pm \delta = \frac{1}{N}\sum_{i=1}^{N} x_i(n-1) = \langle x(n-1) \rangle$$

Using Equation 6, the average squared displacement can be determined by simply squaring the expression $\langle x(n) \rangle$, where, once again, the term following the $\pm$ will result in a zero sum due to the equal probability of its result being positive or negative:

$$\langle x^2(n) \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i^2(n-1) \pm 2\delta x_i(n-1) + \delta^2 = \langle x^2(n-1) \rangle + \delta^2 = n\delta^2 \tag{7}$$

Knowing that t = nτ, Equation 7 can be rearranged to prove Einstein's theory of linear time dependence:

$$\langle x^2(n) \rangle = \frac{\delta^2}{\tau} t = 2Dt$$

This method verifies Einstein's theory of Brownian Motion by plotting the motion of a bead. The mean squared displacement is the average of all the squared change in position the particle travels each frame (Equation 7). By plotting the mean squared displacement against time, the graph has a linear fit that follows Equation 1, where D represents the slope of the linear fit. To determine the diffusion constant, D, through the fit, divide the slope of the line of best fit by 2. Then substitute D, η, r, and T into Equations 3 and 4 and rearrange to solve for k, assuming Stokes drag, $\gamma = 6\pi\eta r$. The diameter of the bead is given in the lab manual [1] as 1.9μm, then the radius r can be calculated to be 0.95 $\pm$ 0.1μm. The viscosity, η, is given as 1.00 $\pm$ 0.05 cP at 20°C but decreases 2% with each degree increase in temperature (η = 1.00 - 0.02(T - 293.15)). Therefore, at a temperature of 296.5 $\pm$ 0.5 K (provided in the lab manual [1]), the viscosity used in the final calculations is 0.933 $\pm$ 0.05 cP. The final calculated Boltzmann's constant, k, value is $\mathbf{1.5956 \times 10^{-23} \pm 8.026 \times 10^{-24} \frac{J}{K}}$ [1] using Equation 8.

$$k = \frac{6D\pi\eta r_b}{T} \tag{8}$$

This value and its uncertainty is consistent and within the theoretical value of k: $1.380649 \times 10^{-23} \frac{J}{K}$. The percent error from the actual value and this lab value is 15.5%, calculated with the equation written below. Visually and computationally, the data follows a linear fit very well and correctly aligns with Einstein's theory of Brownian motion (Figure 2).

[1] This uncertainty was obtained through error propagation calculations with the center of the radius uncertainty defined in Section II. The Python script written does not take into account the center of the radius uncertainty and results in an uncertainty of 7.24 $\times 10^{-24}$, which does not fall in the range of the theoretical Boltzmann's constant value and is much smaller than the radius-propagated uncertainty.

$$\% \text{ error} = \left| \frac{\text{measured value} - \text{theoretical value}}{\text{theoretical value}} \right|$$

The reduced chi-squared value for the residuals shown in Figure 3 is calculated to be 1.64 $\boldsymbol{(\chi_\nu^2 = 1.64)}$ using the standard Least Squares Method equation:

$$\chi_\nu^2 = \frac{1}{\nu} \sum_{i=1}^{N} \frac{[y_i - f(x_i)]^2}{\sigma_{y_i}^2} \tag{9}$$

This reduced chi-squared value reveals that the data collected is a very accurate fit to the linear fit applied. This conclusion drawn can also be explained by the residual plot, Figure 4, where there is little deviation in the positive and negative directions for the whole data set.

See below for the linear fit (Figure 3) and its residuals plot (Figure 4).

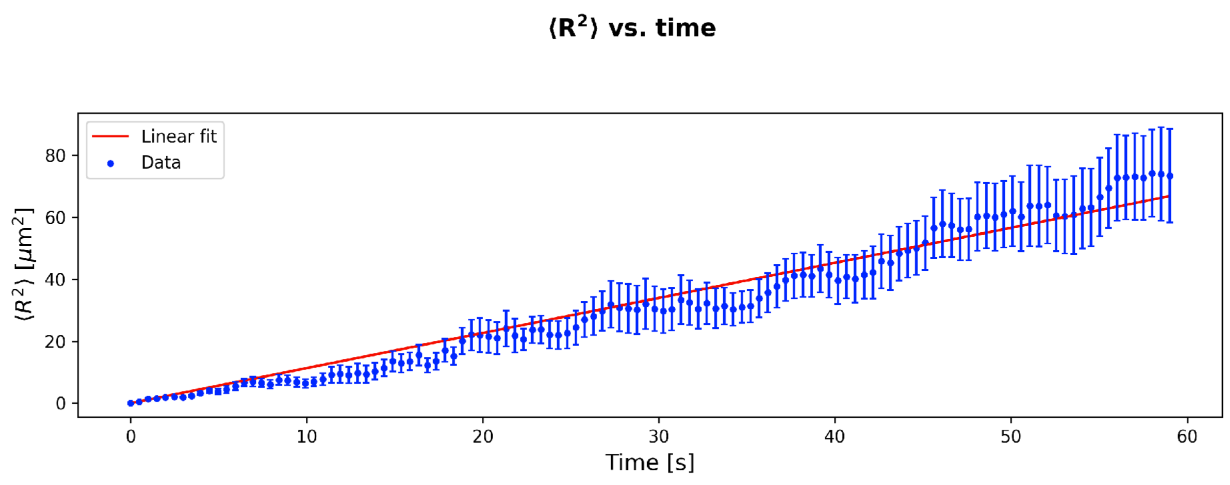


Figure 3: The mean squared distance $\langle R^2 \rangle$ plotted in relation to time. The slope of the linear fit (in red) represents the value of the diffusion constant, $\boldsymbol{D = 0.2832 \pm 0.0058 \frac{\mu m^2}{s}}$.

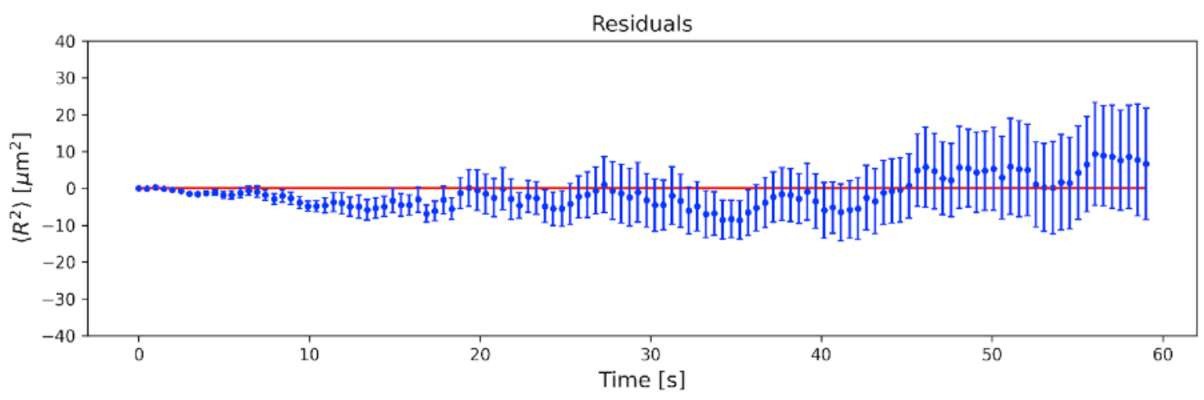


Figure 4: The accuracy of data to a linear fit.

Method 2: Distribution of Steps in a Constant Interval

This method calculates the diffusion constant by fitting the probability density function to a histogram under a Rayleigh distribution. A Rayleigh distribution is chosen as the most accurate model to represent the x-y data sets for the 2D Brownian motion because 2 independent variables are compared simultaneously. The derivation of the Rayleigh distribution is from integrating the 2D Gaussian distribution (Equation 5) into its polar form since computing the definite integral of a probability density function provides the probability.

In Cartesian coordinates, we postulate that the probability of locating a particle within a rectangular box of dimensions x = a, y = b, can be expressed as:

$$P(x,y;t) = \int_0^a \int_0^b \frac{1}{4\pi Dt} e^{-\frac{x^2+y^2}{4Dt}} dxdy$$

Converting the above equation from Cartesian coordinates to polar coordinates yields the following equation, assuming the particle can travel up to a radius R:

$$P(r;t) = \int_0^{2\pi} \int_0^R \frac{r}{4\pi Dt} e^{-\frac{r^2}{4Dt}} drd\theta$$

Computing the definite integration above with respect to θ yields the Rayleigh distribution, which represents the probability density function for a 2D diffusion process.

$$p(r;t) = \frac{r}{2Dt} e^{-\frac{r^2}{4Dt}} \tag{10}$$

Measuring 119 distances between 120 frames for 10 particles, a total of 1190 step sizes were recorded. Using the square root rule [2], the number of bins in the histogram to best fit the Rayleigh distribution is 35. By substituting the Maximum Likelihood Estimation for 2Dt, the histogram data can be fit to the probability density function (Equation 10).

The Maximum Likelihood Estimation formula is derived from fitting a likelihood function to the Rayleigh distribution. As we seek to maximize the probability of observing a particle, the parameter 2Dt must be minimized as it appears in the denominator of the expression. In statistics, a likelihood function can be generally expressed as:

$$\mathbf{L}(x_1,...,x_n;\mu,\sigma) = n(x_1;\mu,\sigma)...n(x_n;\mu,\sigma) = \prod_{k=1}^{n} n(x_k;\mu,\sigma)$$

This function can be applied to the Rayleigh distribution:

$$\mathbf{L}(r;2Dt) = \prod_{k=1}^{N} p(r_k;t) = \prod_{k=1}^{N} \frac{r_k}{2Dt} e^{-\frac{-r_k^2}{4Dt}} \tag{11}$$

Optimizing the likelihood function would require differentiating the function and equating it to zero. Since working with and visualizing the product operator is foreign and complex, the logarithm of both sides must be taken since the natural logarithm of the product operator is simply the summation operator.

$$\ln \mathbf{L}(r;2Dt) = \ln\Big[\prod_{k=1}^{N} \frac{r_k}{2Dt} e^{-\frac{r_k^2}{4Dt}}\Big] = \sum_{k=1}^{N}\Big[\ln(\frac{r_k}{2Dt} e^{-\frac{r_k^2}{4Dt}})\Big] = \sum_{k=1}^{N}\Big[\ln(\frac{r_k}{2Dt}) - \frac{r_k^2}{4Dt}\Big] \tag{12}$$

Applying properties of logarithms, the expression can be simplified to:

$$\sum_{k=1}^{N}\Big[\ln(\frac{r_k}{2Dt} e^{-\frac{r_k^2}{4Dt}})\Big] = \sum_{k=1}^{N}\Big[\ln(\frac{r_k}{2Dt}) - \frac{r_k^2}{4Dt}\Big] = \sum_{k=1}^{N} \ln r_k - N\ln(2Dt) - \sum_{k=1}^{N} \frac{r_k^2}{4Dt}$$

Finally, applying its derivative and equating it to zero will yield the following expression:

$$\frac{\partial(\ln \mathbf{L})}{\partial(2Dt)} = -\frac{N}{2Dt} + \frac{1}{2}\sum_{k=1}^{N}\frac{r_k^2}{(2Dt)^2} = 0 \tag{13}$$

$$N = \frac{1}{4Dt}\sum_{k=1}^{N} r_k^2 \tag{14}$$

$$(2Dt)_{\text{MLE}} = \frac{1}{2N}\sum_{k=1}^{N} r_k^2 \tag{15}$$

which is the maximum likelihood estimate of the 2Dt term.

When using Python's curve$_{\text{fit}}$() function in the SciPy library, the number of bins used will alter the calculated D value, even after using the square root rule; for a more accurate result, the Maximum Likelihood Estimation is used (Equation 15) to calculate D. N is the number of bins used to plot the relative frequency histogram, and the radial distance, r, is $\sqrt{x^2+y^2}$. The final calculated k value is $\mathbf{1.00025 \times 10^{-23} \pm 5.03 \times 10^{-24} \frac{J}{K}}$ [2] with a percent error of 28% from the theoretical value.

The reduced chi-squared value for this Rayleigh distribution can be calculated using Equation 9; it comes out to be 3.814 $(\boldsymbol{\chi^2_\nu = 3.814})$. This reduced chi-squared value reveals that the data used to analyse does not fit perfectly with the curve. This can be explained by the first bin of the histogram in Figure 5 being highly concentrated.

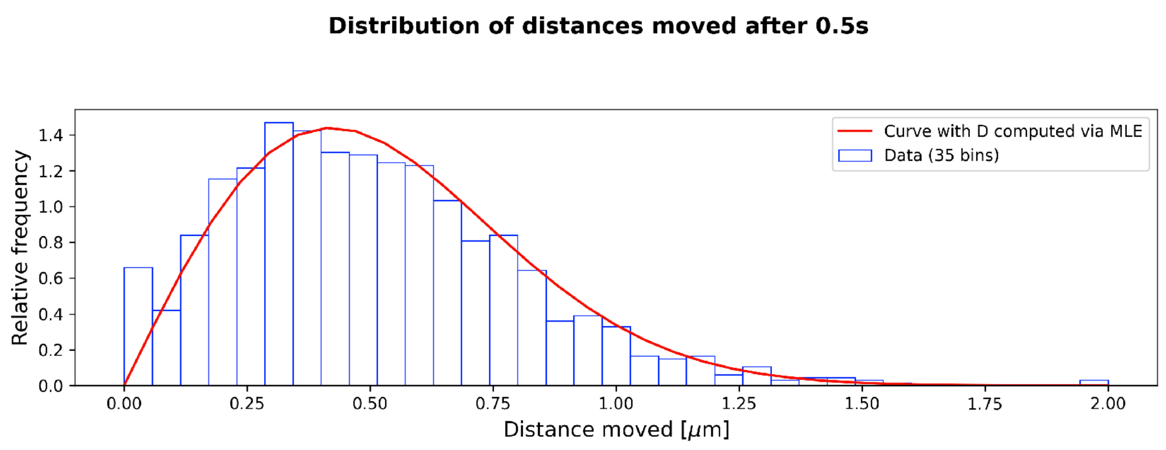


Figure 5: The probability distribution of the 2D diffusion process fitted to a Rayleigh distribution (plotted in red), plotted alongside a relative frequency histogram (plotted in blue). From the fitted curve, the Boltzmann's constant value obtained is k = 1.00025 $\times 10^{-23} \pm 5.03 \times 10^{-24}\frac{\text{J}}{\text{K}}$.

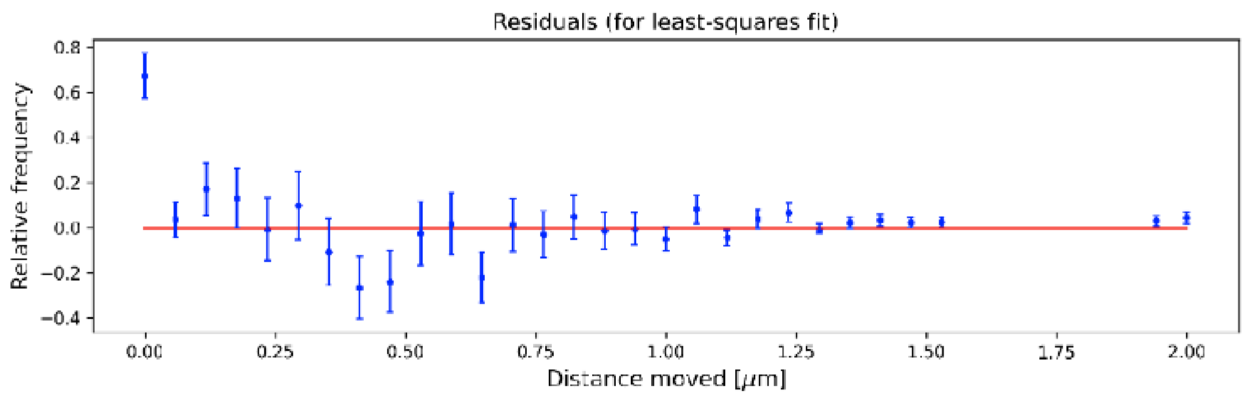


Figure 6: Residuals for the 35 bins of the relative frequency histogram, including their error bars.

[2]This uncertainty was obtained through error propagation calculations with the center of the radius uncertainty defined in Section II. The Python script written does not take into account the center of the radius uncertainty and results in an uncertainty of 1 $\times 10^{-24}$, which does not fall in the range of the theoretical Boltzmann's constant value and is much smaller than the radius-propagated uncertainty.

Verifying Boltzmann's Constant with Avogadro's Number

To verify the accuracy of the experimentally obtained Boltzmann's constant (k), use Avogadro's number (N_A) and the gas constant (R) — $N_A = 6.022 \times 10^{23}$, $R = 8.3145 \frac{J}{mol \cdot K}$ — with the relation:

$$N_A = \frac{R}{k} \tag{16}$$

Using the Boltzmann's Constant obtained during Method 1 of $k = 1.5956 \times 10^{-23} \pm 7.28 \times 10^{-24} \frac{J}{K}$, the calculated Avogadro's number is $\mathbf{5.2108 \times 10^{23} \pm 2.62 \times 10^{23}}$ [3], which is not experimentally equivalent to the theoretical value. Using the Boltzmann's Constant obtained during Method 2 of $k = 1.00025 \times 10^{-23} \frac{J}{K}$, the calculated Avogadro's number is equal to $\mathbf{8.3123 \times 10^{23} \pm 4.18 \times 10^{23}}$ [4], which is also not experimentally equivalent to 6.022×10^{23}.

These experimental inaccuracies can be explained by a variety of sources of error including systematic error, digital error, and calculation error, which will be developed further in the following section.

IV Sources of Error

IV.I Systematic Error

All major sources of systematic error are due to the physical setup of the sample slide (glass cover, tape, petroleum jelly) and the polystyrene bead solution.

First, restriction of flow or excessive flow will drastically change the measurement of Brownian motion. If there is no flow within the sample of beads observed, there is no feasible way to measure motion. In addition, it is difficult to differentiate the random motion of beads from water solution having extra flow causing the beads to move.

If the glass cover is pressed down with excessive force, it will cause the beads to get stuck to the top cover or bottom slide to prohibit motion. If the tape and the jelly are not fully connected or not enough jelly is used or too much solution is added, there will be leakages, causing extra flow from the beads wanting to escape the jelly/tape enclosure. To best avoid excess or lack of motion, by visual inspection, faulty slides were eliminated and all beads were taken from the same slide to avoid discrepancies. As per the manual instructions, the sample was left still for a couple of minutes on the microscope stage to reach equilibrium of particles against flow, created by moving the slide around (ie. adjusting/focusing to find beads, moving the slide to the stage). Dirty slide covers (thumb-printed) were also avoided.

IV.II Digital Error

Digital error in this experiment is a result of manually choosing the center of the bead on the tracking software. Rather than the application determining the center of the bead in a consistent manner, the initial x-y coordinate to calculate the difference in motion between all time frames was visually chosen. An accurate central point of the bead contributes to error when calculating the slope of the mean squared distance in Method 1 and r^2 for the diffusion constant in Method 2 (Equation 12). Additionally, "Gain Value" and "Brightness Value" were visually adjusted differently for each bead to best display a bright and round shape to identify the middle in the Image Object Tracker application. While this was helpful for data collection, it may have caused discrepancies between each bead's data.

[3] This value is the uncertainty of Avogadro's number using the radius propagated uncertainty for Boltzmann's constant, σ_k, which is within the range of the theoretical value. The uncertainty that is propagated using the Python code is 7.28×10^{22}, which does not fall within the range of the theoretical value.

[4] Similarly to the above footnote, the radius propagated uncertainty agrees with the theoretical value, whereas the Python calculated uncertainty of 8.31×10^{22} does not.

IV.III Calculation Error

Another contribution to error is the temperature and viscosity used to calculate the diffusion coefficient (Equation 3). Viscosity is sensitive to temperature; it changes by 2% for every degree increased. For Boltzmann's constant calculations using the diffusion constant, the temperature was taken as the room temperature at the start of the data collection, a constant. For future improved data, temperature should be continuously monitored and the dependent variables should be adjusted accordingly. In addition, the data collected only accounts for 2D Brownian motion (x-y data) as the bead samples constitute extremely thin layers of liquid; random motion in the z-direction was neglected in our calculations. For a fuller understanding of the motion of the beads during the experiment, a third perspective of motion should be included in calculations and data collection since, in reality, the beads do fluctuate in the z-direction as well.

IV.IV Error Propagation

The majority of the uncertainties are listed in the lab manual, such as the temperature uncertainty, viscosity, etc. However, the Boltzmann's constant, diffusion coefficient, and Avogadro's number uncertainty must be determined through error propagation calculations. The error propagation formula is as follows, for a function $z = f(x, y)$:

$$\sigma_z^2 = \frac{\partial f}{\partial x}(\sigma_x)^2 + \frac{\partial f}{\partial y}(\sigma_y)^2$$

Rearranging Equation 15 to isolate for D, the error propagation calculation is as follows and is calculated using Python code as it is a fairly large summation expression:

$$\begin{aligned}\sigma_D^2 &= \frac{\partial D}{\partial r}(\sigma_r)^2 + \frac{\partial D}{\partial t}(\sigma_t)^2 \\ \sigma_D &= \sqrt{\frac{1}{4Nt}\sum_{k=1}^{N} 2r_k(\sigma_r)^2 - \frac{1}{4Nt^2}\sum_{k=1}^{N} r_k^2(\sigma_t)^2}\end{aligned}$$

Given a multiplication expression $z = xy$, the uncertainty can be calculated by:

$$\sigma_z = z\sqrt{\left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2}$$

To calculate the uncertainty of Boltzmann's constant, the uncertainty of Stokes' drag must be calculated first. As such, the uncertainty for Stokes' drag is:

$$\sigma_\gamma = \gamma\sqrt{\left(\frac{\sigma_\eta}{\eta}\right)^2 + \left(\frac{\sigma_{r_b}}{r_b}\right)^2} \tag{17}$$

Next, the uncertainty of the Boltzmann's constant can be calculated as:

$$\sigma_k = k\sqrt{\left(\frac{\sigma_\gamma}{\gamma}\right)^2 + \left(\frac{\sigma_D}{D}\right)^2 + \left(\frac{\sigma_T}{T}\right)^2} \tag{18}$$

Finally, the uncertainty of Avogadro's number can be calculated as:

$$\sigma_{N_A} = N_A\frac{\sigma_k}{k} \tag{19}$$

V Conclusion

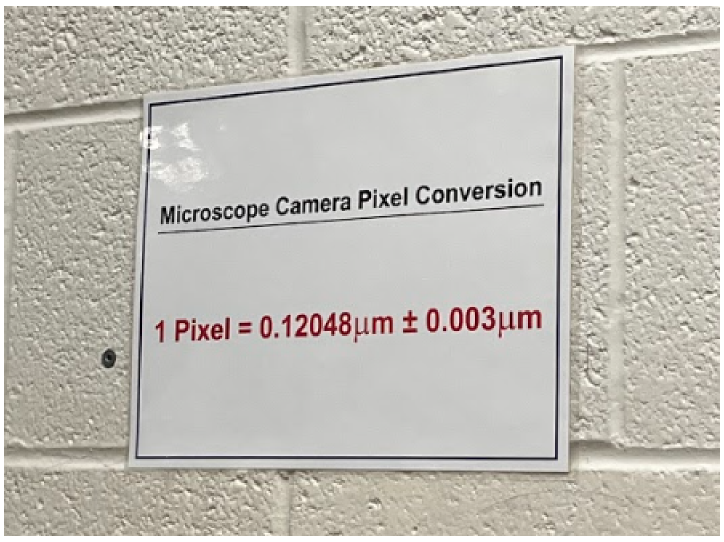
In conclusion, Einstein's theory of Brownian Motion was used to calculate the Boltzmann's constant of bead particles in liquid solution with 2 different methods. First, the 2D "random walk" of 10 beads was analyzed; the individual mean squared distance of the x-y coordinates of each individual dataset was fitted linearly. Ultimately, after leveraging Stoke's Drag and the slope of the linear fit (diffusion coefficient), the final Boltzmann's constant calculated with Method 1 is $\mathbf{1.5956 \times 10^{-23} \pm 8.026 \times 10^{-24} \frac{J}{K}}$. Second, the Maximum Likelihood Estimate was used to fit a histogram containing bins organizing the datasets to a Rayleigh distribution. Python's curvefit() function in the SciPy library was used to calculate a diffusion coefficient. Similarly using Stoke's drag, the Boltzmann's constant for Method 2 was numerically determined to be $\mathbf{1.00025 \times 10^{-23} \pm 5.03 \times 10^{-24} \frac{J}{K}}$. Though both values of k from Method 1 and 2 are relatively close to the actual value (15.5% and 28% percent error respectively), Avogadro's number was used to verify the accuracy. Method 1 obtained a N_A of $\mathbf{5.2108 \times 10^{23} \pm 2.62 \times 10^{24}}$ and Method 2's N_A is $\mathbf{8.3123 \times 10^{23} \pm 4.18 \times 10^{23}}$, both of which were experimentally equivalent to Avogadro's number given their uncertainty values. The inconsistent values of Avogadro's number and Boltzmann's can be credited to systemic, digital, and calculation error.

References

[1] Serbanescu, R.M., Ryu, W., Ladan, J.; "Thermal Motion", 2017.

[2] “Adjusting the number of bins in a histogram,” Datacamp. [Online]. Available: `https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/graphical-exploratory-data-analysis?ex=7#:~:text=The%20%22square%20root%20rule%22%20is,for%20the%20number%20of%20bins.`

A Additional Figures and Pictures



Microscope Camera Pixel Conversion

1 Pixel = 0.12048μm ± 0.003μm

Figure 7: Pixel conversion with its uncertainty.

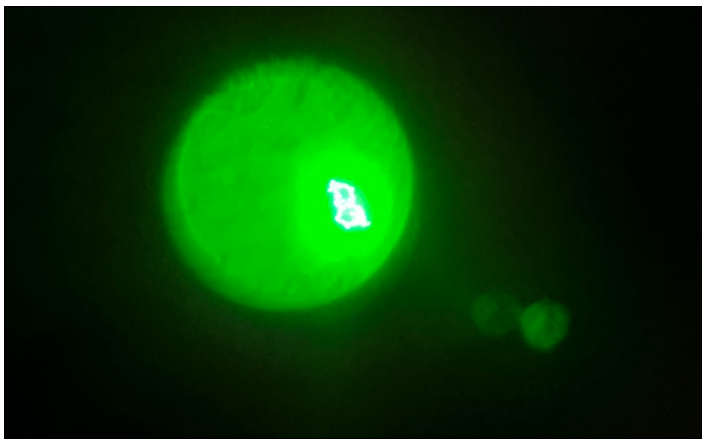


Figure 8: Bead clump (in white) through the microscope.

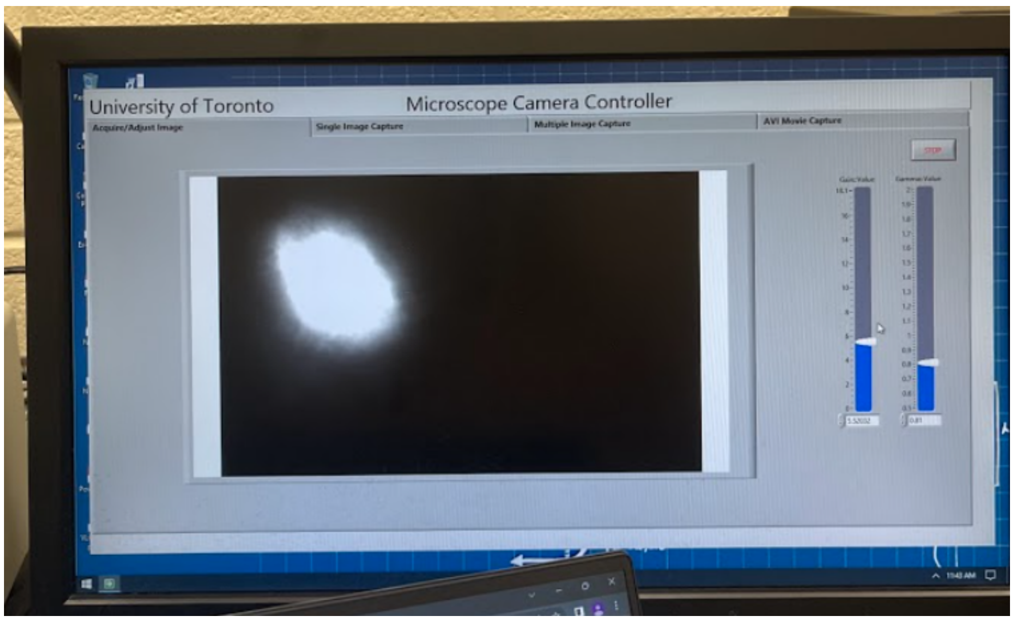
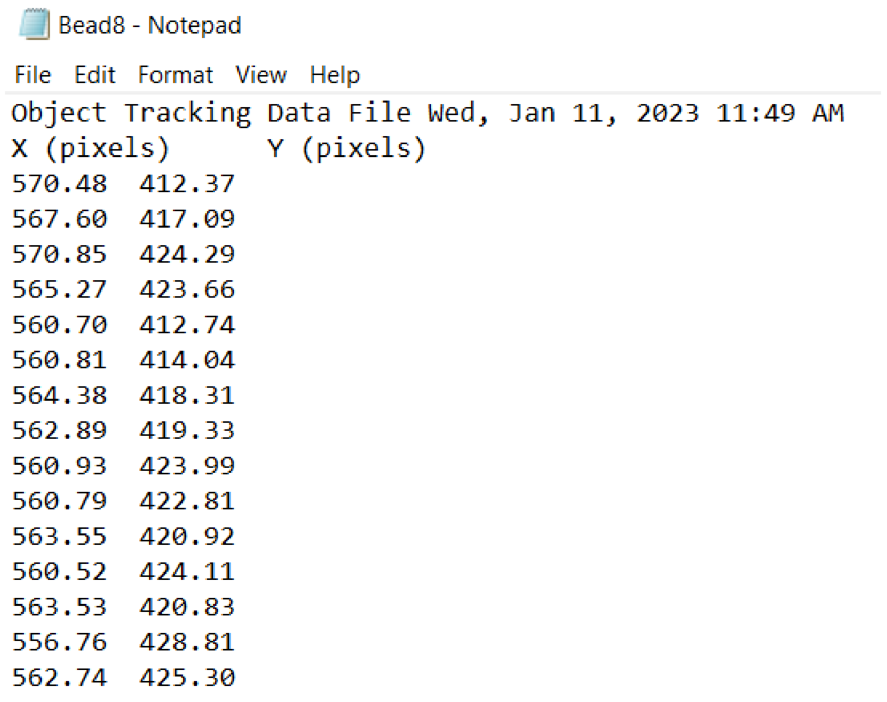


Figure 9: Bead clump through computer LabVIEW program.



```
Object Tracking Data File Wed, Jan 11, 2023 11:49 AM
X (pixels)      Y (pixels)
570.48  412.37
567.60  417.09
570.85  424.29
565.27  423.66
560.70  412.74
560.81  414.04
564.38  418.31
562.89  419.33
560.93  423.99
560.79  422.81
563.55  420.92
560.52  424.11
563.53  420.83
556.76  428.81
562.74  425.30
```

Figure 10: Sample of data collected during the lab session.

B Additional Calculations

For Method 2, the uncertainty of Boltzmann's constant was obtained using the Python code and by hand-calculation. The following shows how it was calculated by hand:

We use Equations 3 and 4 to obtain the values of k, D and γ. Then, we proceed as follows with our error propagation equations 17, 18, 19.

$$\sigma_\gamma = \gamma\sqrt{\left(\frac{\sigma_\eta}{\eta}\right)^2 + \left(\frac{\sigma_{r_b}}{r_b}\right)^2} = (16.707)\sqrt{\left(\frac{0.05}{0.933}\right)^2 + \left(\frac{0.475}{0.95}\right)^2} = 8.40 \tag{20}$$

$$\sigma_k = k\sqrt{\left(\frac{\sigma_\gamma}{\gamma}\right)^2 + \left(\frac{\sigma_D}{D}\right)^2 + \left(\frac{\sigma_T}{T}\right)^2} = (1E-23)\sqrt{\left(\frac{8.401}{16.707}\right)^2 + \left(\frac{0.0021}{0.1775}\right)^2 + \left(\frac{0.5}{296.5}\right)^2} = 5.03E-24 \tag{21}$$

$$\sigma_{N_A} = N_A\frac{\sigma_k}{k} = (8.3123\times 10^{23})\cdot\frac{5.03E-24}{1.00025E-23} = 4.18\times 10^{23} \tag{22}$$

C Python Code

C.I Linear Fit Graph

```
import numpy as nump
import pandas as panda
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

def error(mydata, r_sq): #using error propogration formulas
  #delta = (N * sum(x^2)) - sum(x)^2
  #variance = 1/(N-2) * sum((y - (b +mx))^2)
  #slope^2 = N * variance^2/delta : substitute delta, variance into equation.

  N = len(x_val)
  delta = N * nump.sum(x_val**2) - (nump.sum(x_val))**2
  var = (1 / (N - 1)) * sum((y_val - (r_sq[0] * x_val))**2)
  error_slope = nump.sqrt(N * var / delta)
  return [error_slope]
```

```
def k_var(D): #Equation 3 and 4 in Lab Report: 6*pi*Viscosity*r*D/T
  return (((D / 4) * 6 * nump.pi * 0.933 * 1.9 / 2) / 296.5) * 10**(-21)

def make_into_matrix(list):
  mylist = []
  for i in range(0, len(list)):
    mylist.append([list[i]])
  return nump.array(mylist)

if __name__ == "__main__":
  #importing file
  mydata = panda.read_csv("r_squared.csv")
  x_val = mydata.ind
  y_val = mydata.dep
  x_error = mydata.deltaInd
  y_error = mydata.deltaDep

  #r^2 = x^2 + y^2
  mymatrix = make_into_matrix(x_val)
  myvector = y_val.transpose()
  a = nump.linalg.inv(nump.matmul(mymatrix.T, mymatrix))
  b = nump.matmul(mymatrix.T, myvector)
  r_sq = nump.matmul(a, b)

  #residuals
  err = error(mydata, r_sq)
  result = []
  for i in range(0, len(x_val)):
    result.append(y_val[i] - (x_val[i] * r_sq[0]))


  #graphing linear fit
  gs = gridspec.GridSpec(2, 1)

  y_error = y_error / nump.sqrt(10)
  x = nump.linspace(min(x_val), max(x_val))
  y = nump.array((x * r_sq[0]))

  fig = plt.figure()
  ax1 = fig.add_subplot(gs[0, 0])
  fig.suptitle('r^2 vs Time')
  ax1.set_ylabel('r^2 [$\mu$m$^2$]')
  ax1.set_xlabel('Time [s]')

  ax1.plot(x, y, 'red')
  ax1.scatter(x_val, y_val, marker='.', color='blue')
  ax1.errorbar(x_val, y_val, xerr=x_error, yerr=y_error, color='blue', fmt='o',markersize=0.5)

  #graphing residuals
  ax2 = fig.add_subplot(gs[1, 0])
  ax2.set_ylabel('r^2 [$\mu$m$^2$]')
  ax2.set_xlabel('Time [s]')
  ax2.set_title('Residuals')

  ax2.plot(x, 0 * y, color='red')
  ax2.scatter(x_val, result, marker='.', color='blue')
  ax2.errorbar(x_val, result,xerr=x_error,yerr=y_error, color='blue', fmt='o', markersize=0.5)

  print('4D: ', r_sq[0], '+/-', err[0])
  print("k: ", k_var(r_sq[0]))

  plt.savefig('linear', dpi=300)
```

C.II Rayleigh Distribution Graph

```
from scipy.optimize
import curve_fit
import os
import numpy as nump
import seaborn as sbrn
import pandas as panda
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt


def dist(x_1, x_2, y_1, y_2):
  return (nump.sqrt((y_2-y_1)**2 + (x_2-x_1)**2))

def write_dist_df(fname):
  mypath = ("thermalmotion/data/" + fname)

  point = panda.read_csv(mypath, sep = "\t", header = 1)
  point.columns.values[0] = 'x'
  point.columns.values[1] = 'y'

 #convert pixels to um with given conversion constant
  point.x = point.x * 0.12048
  point.y = point.y * 0.12048

  dist_list = []
  t = nump.linspace(1, 60, 119)
  #120 frames per minute (60s) recorded, 119 stepsizes used to calculate

  for p in range (0, 119):
    cur = dist(point.x[p], point.x[p+1], point.y[p], point.y[p+1])
    dist_list.append(cur)

  d = {'time_passed' : t, 'dist_moved': dist_list}
  return panda.DataFrame(data = d)

#Equation 9 in Lab Report: x^2 = sum((your value - fit value)^2/uncertainty^2))
def chi_sq(histogram_data, D):
  chi_squared = ryl_distro(histogram_data.full_domain, D)
  chi_squared = (histogram_data.data_in_bins - chi_squared)**2
  chi_squared = sum (chi_squared/(histogram_data.full_errors)**2 )
  chi_squared = chi_squared/35 #35 bins from histogram square root rule
  return chi_squared

def ryl_distro(r, D): #Equation  10 in Lab Report: p(r,t) = (r/2Dt) * e^(-r^2/4Dt)
  return (r/(2*D*0.5))*(nump.exp(-(r**2)/(4*D*0.5)))
  #t = 0.5, because for 2 frames per second, one frame/timestep takes 0.5 seconds

#Equation 15 in Lab Report: 2Dt = 1/2N * sum(r^2). Applying error propogration formulas
#in 4.4 Lab Report: error_D = sqrt(1/4Nt* sum(2r* error_r^2))
def MLE(r, err):
  N = len(r)
  two_Dt = (sum(r ** 2))/ (2 * N)
  err_D = nump.sqrt(sum(2*r*(err**2)/(4 * N * 0.5)))
  return two_Dt, err_D

def k_var(D): #Equation 3 and 4 in Lab Report: 6*pi*Viscosity*r*D/T
  return(((D * 6 * nump.pi * 0.933 * 1.9/2)/296.5) * 10**(-21))

if __name__ == "__main__":
  file_list = os.listdir(r"thermalmotion/data")
  i = 0

  for file in file_list:
    if i == 0:
      full_distro = write_dist_df(file)
      i = 1
    else:
      full_distro = panda.concat([full_distro, write_dist_df(file)], ignore_index=True)

  N = len(full_distro.dist_moved) #N = 1190 sets of data, 120 x 10 -  (120 frames per dataset, 19
  diff in distances)
  B = 35 #using square root, Q = number of bins

  del full_distro['time_passed']

#### Making histogram ####
  data_in_bins = nump.zeros(B)

  #making histogram domain
  full_domain = nump.linspace(0, 2, B)

  #making histogram indices
  full_indices= nump.digitize(full_distro.dist_moved, bins = nump.linspace(0, 2, B))
  for index in full_indices:
    data_in_bins[index - 1] += 1

  #making histogram error
  full_errors = []
  for i in range(0, len(data_in_bins)):
    full_errors.append(nump.sqrt(data_in_bins[i]))

  #initializing histogram
  histogram_data= {'full_errors': full_errors, 'full_domain': full_domain, 'data_in_bins':
  data_in_bins}
  histogram_data = panda.DataFrame(histogram_data)

  #removing 0s from data
  zeros = histogram_data[histogram_data['data_in_bins'] == 0 ].index
  histogram_data.drop(zeros, inplace = True)

  #normalize data
  histogram_data.full_errors = histogram_data.full_errors/(nump.sum(histogram_data.data_in_bins)
  * 2/B)
  histogram_data.data_in_bins = histogram_data.data_in_bins/(nump.sum(histogram_data.data_in_bins)
  * 2/B)
  popt, pcov = curve_fit(ryl_distro, histogram_data.full_domain, histogram_data.data_in_bins,
  sigma = histogram_data.full_errors)

  #MLE
  norm_distro = full_distro.dist_moved
  MLE_distro, err = MLE(norm_distro, 0.003)


  print("D: ", MLE_distro, "+/-", err) #um^2 s^-1
  print ("k: ", k_var(MLE_distro), '+/-', 1e-24)
  print ("Chi squared: ", chi_sq(histogram_data, MLE_distro))

##### PLOTTING ####

 #Histogram/Rayleigh Distro
  x = nump.linspace(0, 2, B)
  y = nump.linspace(0, 2, B)
  gs = gridspec.GridSpec(2, 1, hspace = 0.5)
  finalfig = plt.figure()

  finalfig.suptitle('Distribution of distance per timeframe (0.5s)')
  ax_1 = finalfig.add_subplot(gs[0, 0])
  sbrn.histplot(data = full_distro, x ="dist_moved", binrange = (0, 2), color = 'blue',
  stat = 'density')
  ax_1.plot(x, ryl_distro(x, MLE_distro), '-', color = 'red', label = 'D using MLE')
  ax_1.set_ylabel('Count in Bins')
  ax_1.set_xlabel('Distance [$\mu$m]')
  ax_1.legend()

  #Residuals
  residuals = []
  for index in histogram_data.index:
    residuals.append(histogram_data.data_in_bins[index] -
        ryl_distro(histogram_data.full_domain[index], popt[0]))

  #plotting residuals
  ax_2 = finalfig.add_subplot(gs[1, 0])
  ax_2.plot(x, 0 * y , color = 'red')
  ax_2.scatter(histogram_data.full_domain, residuals, color = 'blue', marker = '.')
  ax_2.errorbar(histogram_data.full_domain, residuals, yerr = histogram_data.full_errors, color =
  'blue', fmt = 'o', markersize = 0.5)
  ax_2.set_ylabel('Count in Bins')
  ax_2.set_xlabel('Distance [$\mu$m]')
  ax_2.set_title('Residuals')

plt.savefig('Rayleigh', dpi = 300)
```