Manual to Laser cooling simulation and pylcp connection

Fernando Flores Mendoza

August 2024

Contents

1 Introduction

To increase the signal for the NL-eEDM experiment it is necessary to laser-cool in the transverse direction the cryogenic beam of BaF(Barium Fluoride). A BaF target is located inside a cell that is filled with a buffer gas at a very low temperature. Then a laser ablates the BaF target generating the molecular beam. When the BaF molecules exit the cryogenic source they exit with a ~ 190 m/s velocity and a ~ 5 m/s transverse velocity.(1)
After the molecular beam is generated it goes through the hexapole lens and then through the 2D Laser cooling region. The following picture shows the experimental set up.

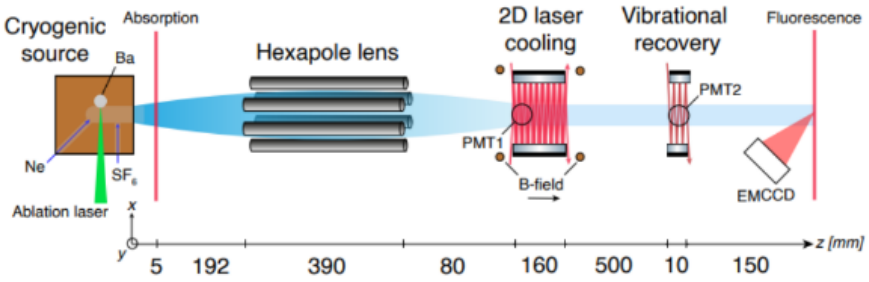


Figure 1: Experimental setup for laser cooling. The hexapole lens focuses the molecular beam and the laser cooling reduces the transverse velocities of said beam. Figure made by J.W.F. van Hofslot

The simulations have four main parameters. The hexapole voltage, the scattering rate, the saturation, and the detuning. The hexapole focuses the molecular beam using electric fields. The hexapole voltage parameter refers to the voltage supplied to the hexapole lens. For a more complete explanation of how the hexapole works revise (2)

On the other hand, the scattering rate, saturation, and detuning are all related to laser cooling. The scattering rate (also sometimes referred to as the fudge factor) is a scaling factor related to the rate at which a molecule absorbs and re-emits a photon.
The saturation parameter is defined as

$$s = \frac{I}{I_s} \tag{1}$$

where I is the intensity used to drive the transition between the ground state and the excited state and I_s is the saturation intensity, $I_s = \pi hc\Gamma/(3\lambda^3)$ where Γ is the natural line width of the molecule and λ is the wavelength of the corresponding transition. (3)

To drive a particle from a ground state k into an excited state j it is necessary that said particle absorbs a photon with a frequency ω_{jk}. Given that the particles (in our case molecules) are not in a stationary state it is also necessary to account for a Doppler shift, since the particles won't be in a stationary state and thus will have inherent velocities different than zero. The detuning is how we take into account this frequency shift the molecules experience when transitioning between the ground state and the excited state.

The goal of the simulations is to reproduce experimental results to have a better understanding of the experiment in itself. The simulations aim to replicate the xy-distributions generated by the experiment, as well as simulate the trajectories of the molecules throughout the whole setup.

In this manual, I will explain how all the functions I implemented in the laser cooling simulation work and how they can be edited. I will also include explanations of the work I did with the pylcp simulations and how they can work together with the laser cooling trajectory simulations.

2 Overview

I mainly worked on two things during my stay in Groningen. I worked on improving the laser cooling simulations written by Bethlem, H.L. This included simulating the xy distribution (same coordinate system as in Fig.1) of the molecules at different distances, adding the low field seekers to the simulations, changing the shape of the lasers to have a Gaussian distribution, and changing the

laser cooling region to have the shape of a laser that bounces between two mirrors instead of being uniform along the laser cooling region.
I also worked on adjusting a polynomial function to the data Max Vos generated using the pylcp package. This data is a different simulation of the laser cooling region. Its goal is to produce different acceleration curves by altering different simulation parameters, like the detuning, the saturation, and the magnetic field. The goal of the connection between the pylcp simulations and our simulations is to have a point of comparison between the parameters we use in our simulations and how these same parameters compare in the pylcp simulations.

3 Laser cooling simulation

To simplify the simulation I split the simulation into four python files.

- **simulate_trajectories_model.py**: This file contains all the physics-related functions in the simulations. It has the laser cooling function, the hexapole function, and the freeflight function, among others. (I'll dig deeper into this in the following section.)
- **plot_maker.py**: This file makes all the plots of interest to the laser cooling simulations.
- **execution.py**: This file defines the functions that set certain simulations of interest. For example, we might be interested in having six heatmaps representing LC on and off along with the hexapole. Instead of writing the code for this set of conditions every time, a function was written to have them ready.
- **simulate_trajectories.py**: This file imports the function written in execution.py, and it's just a matter of passing the parameters we are interested in.

The following diagram explains the hierarchy between these files.

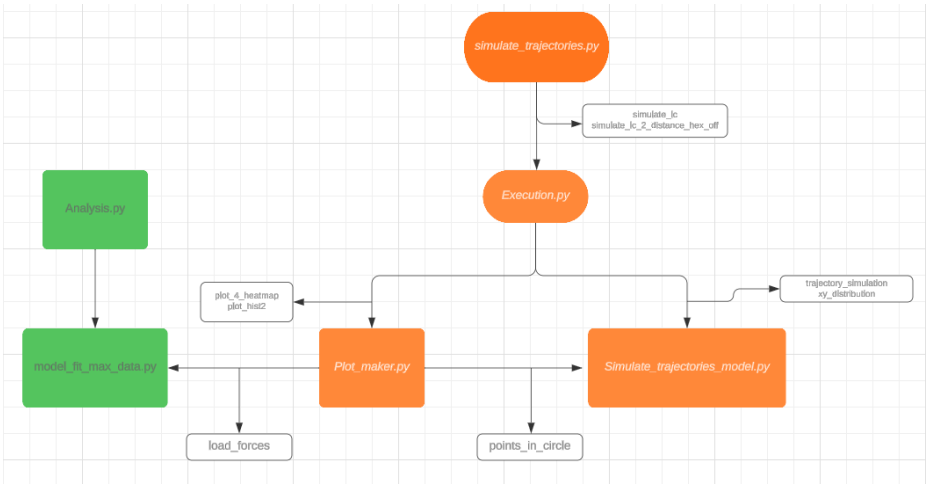


Figure 2: Diagram of hierarchy between files. Orange shapes represent the name of the file and white shapes represent the functions that the file above uses from the file below. So for example simulate_lc is defined in Execution.py but used in simulate_trajectories.py

As can be seen, simulate_trajectories.py is the file with the highest hierarchy. It only imports functions from execution.py like simulate_lc. execution.py imports function from plot_maker.py and simulate_trajectories_model.py.functions defined in execution.py start by simulating a given scenario using functions from simulate_trajectories_model.py and then with the given data plot_maker.py plots the desired plot. Plot_maker.py only imports the points_in_circle function from simulate_trajectories_model.py .

WARNING 1: Analysis.py and model_fit_max_data.py weren't originally made to work alongside the main structure, but to be able to use the *acceleration_curve_pylcp_plot* function they had to be integrated. There is a separate section explaining their structure. Unless the reader is trying to use the pylcp curve I'd advise ignoring the green structure in this section.

WARNING 2: My recommendation is to run the code in Pycharm, rather than VS Code or any other IDE. This is because Pycharm adds the root directory of the files to the main path for any file being run, while other editors don't. It is possible to change the settings of Visual studio Code to add the path but it's a little too complicated to change the json file that contains the settings.

3.1 simulate_trajectories_model.py

- **W_stark(E,s), Phihex(x,y,phi0hex, r0hex), Ehex(x,y,phi0hex,r0hex), Whex(x,y,phi0hex,r0hex,s), axhex(x,y,phi0hex,r0hex,s),ayhex(x,y,phi0hex,r0hex,s)**: W_stark defines the stark shift generated by an electromagnetic field, Phi0hex is related to the forced felt by a particle due to an electric field, Ehex is the electric field and Whex is the shift generated by Ehex. Finally, axhex and ayhex are the accelerations generated by the previous forces. All these functions work together to make the hexapole stage of the simulation.

- **ascat(vx,ff,s0, det)**: *vx* is the velocity of the molecule, *ff* is the scattering rate, *s0* is the saturation intensity, *det* is the detuning of the laser. This function simulates the acceleration of a molecule induced by a laser. It fits the acceleration curve of a two-level system. It returns the scalar of the acceleration.

- **phasespaceellipse2D(xx0,dxx,lfs_percentage=0.666)**: *xx0* is a 7-element array representing with this shape, (t,x,y,z,vx,vy,vz), *dxx* is a 7-element array with each element representing the differential quantity of xx0. *lfs* is the low field seekers percentage. This function sets the initial trajectory for any molecule. It returns [xx,s, hit]. xx a 7-element array with the new initial coordinates of the molecule, s=0,1, and represents if a molecule is a high field seeker or low field seeker respectively, hit=True, False and it refers to whether the molecule is still in this stage or not.

- **freeflight(endpoint,xxs,hit)**: *endpoint* is the z-distance at which this stage should be stopped. *xxs* is a 7-element array representing (t,x,y,z,vx,vy,vz), these are the initial coordinates with which a molecule starts this stage. *hit* is a bool value representing whether the molecule is still in this stage. This function simulates the free flight of a molecule. It returns [xxp,hit], xxp is a 7-element array with the final coordinates of the molecule after this stage, hit=True, False.

- **hexapole(endpoint,phi0hex,r0hex,xxs,s,hit)**:*endpoint* is the z-distance at which this stage should be stopped. *phi0hex* is the voltage of the hexapole. *r0hex* is the radius of the rods of the hexapole. *xxs* is a 7-element array representing (t,x,y,z,vx,vy,vz). *s=0,1* represents whether a molecule is a high or low field seeker. *hit* is a bool value representing whether the molecule is still in this stage.This function simulates the trajectory of a molecule going through

the hexapole. It returns [xxp,hit], xxp is a 7-element array with the final coordinates of the molecule after this stage,hit=True, False.

- **lasercooling(endpoint,ff,s0,detun,xxs,hit,xcooling = True ,ycooling = True , n_reflections = 35)**: *endpoint* is the z-distance at which this stage should be stopped. *ff* is the scattering rate, *s0* is the saturation intensity, *det* is the detuning of the laser. *xxs* is a 7-element array representing (t,x,y,z,vx,vy,vz).*hit* is a bool value representing whether the molecule is still in this stage. *xcooling* is a bool representing whether the laser cooling in the x direction is turned on.*ycooling* is a bool representing whether the laser cooling in the y direction is turned on. *n_reflections* is the number of reflections of the laser inside the laser cooling stage. This function simulates the trajectory of a molecule going through the laser cooling region. It returns [xxp,hit], xxp is a 7-element array with the final coordinates of the molecule after this stage,hit=True, False.

- **trajectory_simulation(initial_pos ,nn ,nj ,ff ,s0 ,detun ,phi0hex , xcooling = True ,ycooling = True ,lc = True, hex = True, sub_doppler = False,par_sub_doppler =np.zeros(6))**: *initial_pos* is an nn-shaped array with the initial coordiantes of nn molecules. nn is the number of molecules to simulate, nj is the number of steps to be simulated inside the free-flight, hexapole and laser cooling region; if it's not of interest to see the entire trajectory and one just wants to plot the file distribution of the molecules then n=2 is sufficient. *ff* is the scattering rate, *s0* is the saturation intensity, *det* is the detuning of the laser.*phi0hex* is the voltage of the hexapole. *xcooling* is a bool representing whether the laser cooling in the x direction is turned on.*ycooling* is a bool representing whether the laser cooling in the y direction is turned on. *lc* is a bool representing whether the entire laser cooling stage should be on or not.*hex* is a bool representing whether the entire hexapole stage should be on or not (if one wants to turn off the hexapole is better to set phi0hex=1e-3 instead of turning of the hexapole). *sub_doppler* represents if the acceleration curve of a sub-doppler cooling should be taken into account. *par_sub_doppler* are the parameters to be passed if *sub_doppler=True*. This function simulates the entire trajectory of the molecules going through the experiment. The respective endpoint for each stage is defined inside the function itself. It returns the following array. [x_position,y_position,z_position,vx,vy,ax,ay]. Each element is a nn-size array with all the coordinates of the molecules, for example, x_position has nn elements, the n_k element of x_position is an array with all the x coordinates of the trajectory of the molecule n_k.

- **xy_distribution(xpoints,ypoints,zpoints,vx,vy,Ldetection)**: *xpoints* is and nn-shaped array with all the x-coordinates. *ypoints* is and nn-shaped array with all the y-coordinates.*zpoints* is an nn-shaped array with all the z-coordinates. *vx* is and nn-shaped array with all the vx-coordinates.*vy* is and nn-shaped array with all the vy-coordinates. *Ldetection* is the distance at which the xy-distribution should be measured. It returns [x,y,z,vx1,vy1]. x, y are all the coordinates of the molecules in the x and y direction. z are also all the z coordinates in the z direction but given that the distribution is measured at a given distance z=[Ldetection, ..., Ldetection]. vx1,vy1 are the velocities of the molecules at Ldetection.

- **points_in_circle(x,y,r)**: x is an array with all the x-coordinates of the distribution.y is an array with all the y-coordinates of the distribution. r is the radius of the beam we want to measure. It returns the number of molecules inside said beam

3.2 plot_maker.py

Unless stated directly on the function description, none of the following functions will have the bin size as a parameter for plotting, and thus to change the bin size it will be required to change it directly on the function itself.

- **plot_all(xls, zls, vxls)**: *xls* is an n-shaped array (where n is the number of molecules simulated) that has all the x-coordinates of all the molecules.*zls* is an n-shaped array (where n is the number of molecules simulated) that has all the z-coordinates of all the molecules.*vxls* is an n-shaped array (where n is the number of molecules simulated) that has all the transverse velocities of all the molecules. It returns. It returns a plot of the x-axis vs z-axis and a plot of vx vs z-axis.

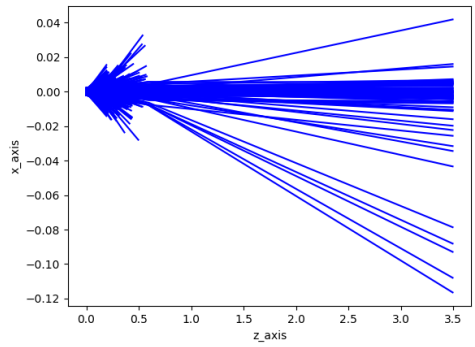


Figure 3: Trajectories of molecules. Transverse position as a function of longitudinal direction

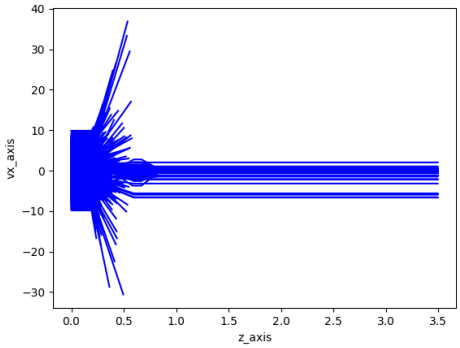


Figure 4: Velocity as a function of longitudinal position.

- **plot_side_by_side(x1, z1, x2, z2)**: *x1* is an n-shaped array (where n is the number of molecules simulated) that has all the x-coordinates of all the molecules.*z2* is an n-shaped array (where n is the number of molecules simulated) that has all the z-coordinates of all the molecules. *x2.z2* are the same type of arrays as *x1,z1* but are used for a second scenario. It plots the trajectories of the molecules as a function of the longitudinal direction.

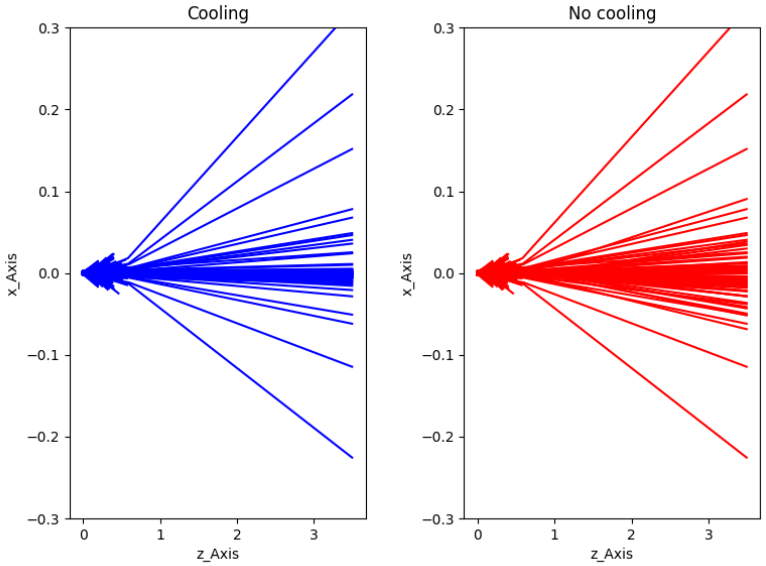


Figure 5: Comparison plot of the laser-cooled molecules trajectories vs no laser-cooled molecules trajectories

- **plot_heatmap(xcooling, ycooling, zcooling)**: *xcooling, ycooling, zcooling* are arrays returned by the function *xy_distribution*. It plots the xy-distribution at a given distance.

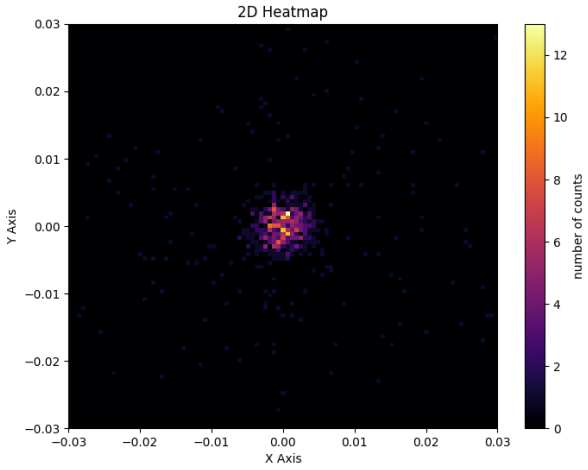


Figure 6: xy-distribution of molecules at a given distance

- **plot_2_heatmap(xcooling1, ycooling1, zcooling1, xcooling2, ycooling2, zcooling2, ff, s0, det, hexvolt, Ldetection)**:*xcooling1, ycooling1, zcooling1, xcooling2, ycooling2, zcooling2*

are the returned arrays of *xy_distribution. ff, s0, det. hexvolt, Ldetection* are the scaling factor, saturation, detuning, hexapole voltage, and detection distance respectively.
This function plots two xy-distributions to compare them.

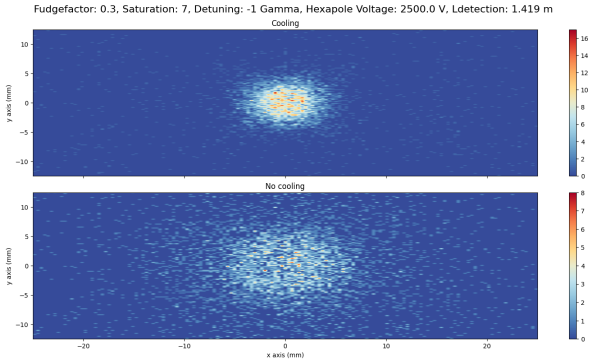


Figure 7: Comparison of laser-cooled molecules distribution vs no laser-cooled molecules distribution

- **plot_4_heatmap(x1, y1, x2, y2, x3, y3, x4, y4, ff, s0, det, hexvolt, Ldetection1, Ldetection2, legends, plot_circle = False, save_plot = False)**: *x1, y1, x2, y2, x3, y3, x4, y4* are the returned arrays of xy_distribution. *ff, s0, det, hexvolt, Ldetection1, Ldetection2* are the scattering rate, saturation, detuning, hexapole voltage, first detection distance and second detection distance respectively.*legends* are labels to be plotted in the figure, they usually refer to the scenario been plotted (like 2D LC on, hexapole off etc). *plot_circle* plots a circle of radius 5 mm. If *save_plot=False* the function just shows the plot, if *save_plot=True* then it saves it in the current directory.
This function plots four heatmaps to represent the xy-distribution of molecules for laser-cooled and non-laser-cooled molecules for two distances.

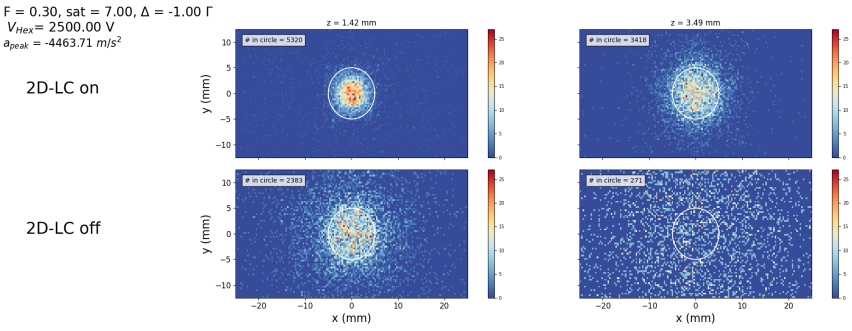


Figure 8: xy-distribution of laser-cooled, no laser-cooled for two distances.

- **plot_hist2(velr_1, velr_2, ff, s0, det, hexvolt, Ldetection1, bins = 60,save_plot = False)**: *velr_1,velr_2* are radial velocities corresponding to the the velocities returned by *xy_distribution* i.e $velr\_1[i] = \sqrt{vx[i]^2 + vyc[i]^2}$. *ff, so, det, hexvolt, Ldetection1* are the scattering rate, saturation, detuning, hexapole voltage, and detection distance respectively.

bins is the bin size. If *save_plot=True* the plot will be saved in the current directory, otherwise the plot will just be shown.
This function plots two histograms of two sets of velocities. It was made to compare the velocity distribution with laser cooling and without laser cooling.

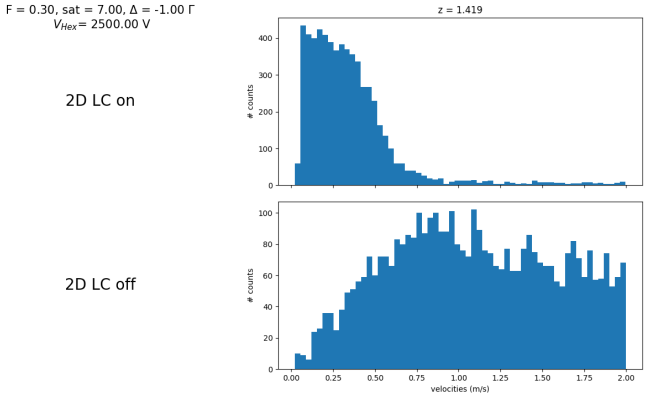


Figure 9: It plots the histogram of the velocity distribution for laser-cooled molecules and for non-laser-cooled molecules.

- **plot_6_heatmap(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, x5, y5, z5, x6, y6, z6, ff, s0, det, hexvolt, Ldetection1, Ldetection2, plot_circle = False, plot_safe = True, extra_title = '', sub_doppler = False, par_sub_doppler = np.zeros(6))**:

  x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, x5, y5, z5, x6, y6, z6 are arrays returned by *xy_distribution*. *ff, s0, det, hexvolt, Ldetection1, Ldetection2* are the scattering rate, saturation, detuning, hexapole voltage, first detection distance, and second detection distance respectively. If *plot_circle=True* a circle of radius 5mm and with the center in the origin will be plotted to represent the desired beam size. *extra_title* is an additional title that can be plotted in case there's more information that needs to be presented.*sub_doppler* is a parameter passed to let the function know it needs to adjust the scales to be able to fit the parameters of the sub_doppler curve in the plot.*par_sub_doppler = np.zeros(6))* are the parameters used in the sub_doppler cooling.
  This function plots six heatmaps to show the distribution of the molecules in three different scenarios, for two distances.

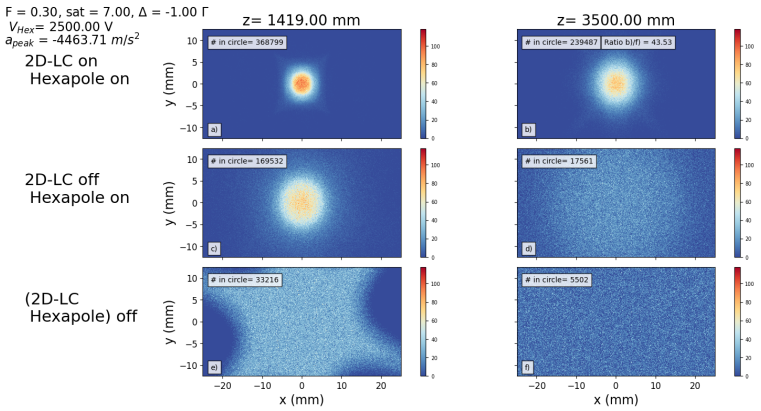


Figure 10: xy-distribution of laser-cooled, non-laser-cooled and hexapole free molecules.

- **plot_8_heatmap(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6,x7, y7, x8, y8, ff, s0, det, hexvolt, Ldetection1, Ldetection2, plot_circle = False, save_plot = True, extra_title = ")**: *x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, x5, y5, z5, x6, y6, z6* are arrays returned by *xy_distribution*. *ff, s0, det, hexvolt, Ldetection1, Ldetection2* are the scattering rate, saturation, detuning, hexapole voltage, first detection distance, and second detection distance respectively. If *plot_circle=True* a circle of radius 5mm and with the center in the origin will be plotted to represent the desired beam size. *extra_title* is an additional title that can be plotted in case there's more information that needs to be presented.
  This function is made to plot eight heatmaps representing four scenarios, laser-cooled molecules, non-laser-cooled molecules, non-laser-cooled molecules without hexapole, and heated-molecules with hexapole on. To simulate the laser heating the detuning needs to change sign.

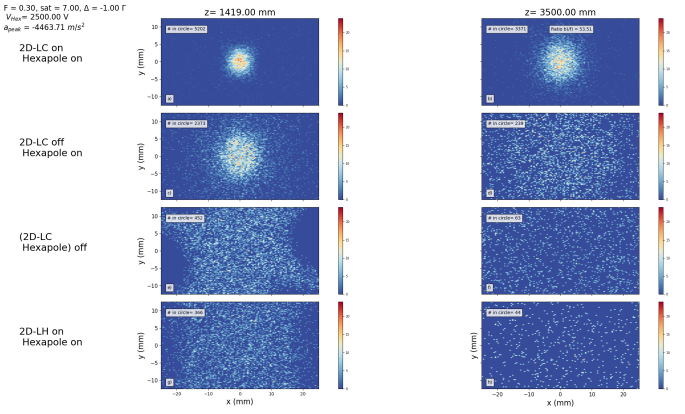


Figure 11: xy-distribution for laser-cooled, non-laser-cooled, hexapole free and laser-heated molecules

- **acceleration_curve_pylcp_plot(det_pylcp,sat_pylcp,ff,s0,det, save_plot=False)**: *det_pylcp,sat_pylcp* represent a detuning and saturation used in one of the pylcp simulations, i.e there must exist a file with the name 'obe_det_pylcp_sat_pylcp.txt'.Given that the previous pylcp file had been

fitted to the appropriate curve *ff,so,det* are the scattering rate, saturation and detuning that correspond to the fit of said data.
It loads the velocities and accelerations corresponding to det_pylcp and sat_pylcp and plots the data and the fitted curve.

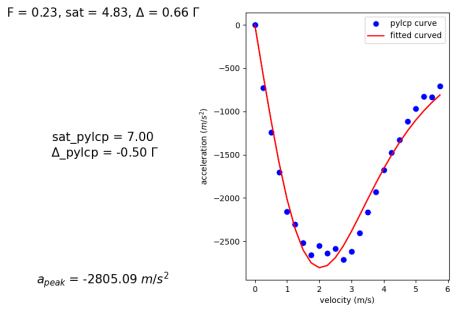


Figure 12: Acceleration curve of the laser cooling force. The dots represent the curve generated with the pylcp simulations, this simulations take into account the transitions between the electronic and vibrational levels of the molecules. The solid red curve is the fit of said data to a two-level system

- **plot_n_heatmaps(xs, ys, FFS, s0s, dets, hexvolts,hfs_bool,plot_titles, plot_circle)**:*xs, ys* are arrays of x and y coordinates, they have a n dimension. *FFS,s0s,dets,hexvolts* are the scattering rates, saturations, detuning and hexapole voltages used in the n simulations.*hfs_bool* is a list of boolean values that represent whether or not the high field seekers are simulated. *plot_titles* are the labels assigned to each one of the distributions.
  This function plots n-heatmaps representing xy-distributions. I recommend saving the coordinates in a file and using the PlotManager.py script written by Anno Tauwen to plot the desired plots.
- **plot_mol_vs_voltage(Voltagehex, num_molecules,title = '', r=[5,2.5],make_title = False, save_plot = False,normalize=True)**: *Voltagehex* is an array of voltages used in the simulation. Given the xy-distribution formed by the molecules at a given distance, *num_molecules* is the number of molecules inside a circle of radius r. *title* is the title to be plotted in the figure, *r* is a list of the radius used. *make_title* is a boolean value referring to whether or not plot the title. *save_plot* is a boolean value that saves or not the plot. *normalize* normalizes the number of counts, if true, the maximum value is 1.
  It plots the number of counts of the xy-distribution in a given area (a circle) as a function of voltage.

3.3 execution.py

In all the following functions nn= number of molecules, nj= number of steps during the hexapole and laser cooling stages,fudgefeactor = scattering rate, s00 = saturation, detuning=detuning, Voltagehex = Hexapole voltage, Ldet1= detection distance one and Ldet2 = detection distance 2, unless stated otherwise.

- **simulate_lc(nn,nj ,fudgefactor,s00, detuning,Voltagehex,Ldet1)**: This function simulates the trajectories of laser-cooled molecules and non-laser-cooled molecules and plots the distribution side by side. Its output is Fig 7

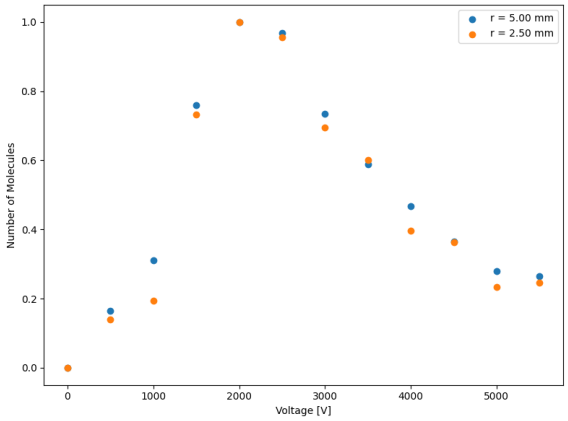


Figure 13: Number of molecules inside a certain section as a function of Voltage

- **simulate_lc_2_distance(nn,nj,fudgefactor,s00,detuning,Voltagehex ,Ldet1 ,Ldet2 , plot_circle = False)**:

  Simulates the trajectories of laser-cooled molecules and non-laser-cooled molecules. It plots the distributions for two distances. Its output is fig. 8

- **simulate_x_vx_dis(nn,nj ,fudgefactor,s00 , detuning ,Voltagehex,Ldet1)**: Simulates the trajectories for laser-cooled molecules and plots the transverse velocities against the transverse position.

- **simulate_vr_hist(nn,nj,fudgefactor,s00,detuning,Voltagehex,Ldet1,save_plot=False)**: Simulates the trajectories of laser-cooled molecules and non-laser-cooled molecules. It plots the histogram of the radial velocities at a given distance. Its output is Fig. 9

- **simulate_lc_2_distance_hex_off(nn, nj, fudgefactor, s00, detuning, Voltagehex, Ldet1, Ldet2,lfs_bool = True, plot_circle = False, plot_safe = False, extra_title = '', sub_doppler = False,par_sub_doppler = np.zeros(6)))**: Simulates the trajectories of laser-cooled molecules, non-laser-cooled molecules and non-laser-cooled molecules and hexapole free molecules. Its output is Fig 10

- **simulate_lc_2_distance_hex_off_lH(nn, nj, fudgefactor, s00, detuning, Voltagehex, Ldet1, Ldet2,lfs = 0.666, plot_circle = False, save_plot = False, extra_title = '', sub_doppler = False,par_sub_doppler = np.zeros(6))**: Simulates the trajectories of laser-cooled molecules, non-laser-cooled molecules and non-laser-cooled molecules,hexapole free molecules and laser-heated molecules. Its output is Fig 11

- **simulate_number_mol_vs_hexvoltage(nn, nj, fudgefactor, s00, detuning, Voltagehex, Ldet1, lc_bool, lfs=0.666)**:**fudgefactor, s00, detuning** are arrays of size len(Voltagehex) with repeating values i.e s00[i]==s00[j],ff[i]==ff[j], det[i]==det[j] $\forall i, j$. *Voltagehex* is an array with the different voltages to be used in the simulation. *lc_bool* is an array of size len(Voltahex). It refers to whether or not laser cooling should be simulated.
  It simulates the trajectories of molecules going through the hexapole and counts the number of molecules inside a determined area (a circle) as a function of voltage. Its output is Fig 13

- **simulate_and_save_data(nn,nj,fudgefactor, s00, detuning, Voltagehex, Ldet1,lc=True ,title='', directory='')**: Given a *fudgefactor, s0, detuning* it simulates the trajectories of the molecules and saves the x and y coordiantes at a given distance.

3.4 Simulate_trajectories.py

Simulate_trajectories.py doesn't have any functions defined but instead it's used to set the parameters for the simulations and run them in a file with less than ten lines of code.

I present the following example.
The following code is the entire script of **simulate_trajectories.py**

```
from execution import *
import datetime
a = datetime.datetime.now()
print(a)
nn = int(1e6) #number of molecules
nj = 2 #steps in laser cooling stage, freeflight and hexapole
#laser cooling detection distance, defined in simulate_trajectories_model.py
Ldet1 = L[0]+L[1]+L[2]+L[3]+L[4]
#spin-precession distance, defined in simulate_trajectories_model.py
Ldet2 = L[0]+L[1]+L[2]+L[3]+L[4]+L[5]
fudgefactor=0.3 #scattering rate
s00=7 #saturation
detuning=-1 #detuning
Hexvoltage = 2.5e3 #Hexapole voltage
#imported from execution
simulate_lc_2_distance_hex_off(nn,nj,fudgefactor,s00,detuning,Hexvoltage,Ldet1,Ldet2)
b = datetime.datetime.now()
print(b-a)
```

As it can be seen, L is defined from **simulate_trajectories_model.py**.
Simulate_lc_2_hex_off is defined as follows. It is defined in **execution.py**

```
    def simulate_lc_2_distance_hex_off(nn, nj, fudgefactor, s00, detuning, Voltagehex,
        Ldet1, Ldet2,lfs= 0.666,plot_circle = False, save_plot = False, extra_title =
        '',sub_doppler = False,par_sub_doppler = np.zeros(6)):
    initial = [phasespaceellipse2D(xx0, dxx,lfs_percentage=lfs) for i in range(nn)]
    xlist, ylist, zlist, vx, vy, ax, ay = trajectory_simulation(initial, nn, nj, ff =
        fudgefactor, s0 = s00, detun = detuning, phi0hex = Voltagehex, lc = True,
        sub_doppler=sub_doppler,par_sub_doppler=par_sub_doppler)

    xlist2, ylist2, zlist2, vx2, vy2, ax2, ay2 = trajectory_simulation(initial, nn, nj, ff
        = fudgefactor, s0 = s00, detun = detuning, phi0hex = Voltagehex, lc = False,
        sub_doppler=sub_doppler,par_sub_doppler=par_sub_doppler)

    xlist3, ylist3, zlist3, vx3, vy3, ax3, ay3 = trajectory_simulation(initial, nn, nj, ff
        = fudgefactor, s0 = s00,detun = detuning, phi0hex = 1.e-3, lc = False)

    xc1, yc1, zc1, vxc1, vyc1 = xy_distribution(xlist, ylist, zlist, vx, vy, Ldet1)
    xc2, yc2, zc2, vxc2, vyc2 = xy_distribution(xlist2, ylist2, zlist2, vx2, vy2, Ldet1)
```

```
xc3, yc3, zc3, vxc3, vyc3 = xy_distribution(xlist, ylist, zlist, vx, vy, Ldet2)
xc4, yc4, zc4, vxc4, vyc4 = xy_distribution(xlist2, ylist2, zlist2, vx2, vy2, Ldet2)
xc5, yc5, zc5, vxc5, vyc5 = xy_distribution(xlist3, ylist3, zlist3, vx3, vy3, Ldet1)
xc6, yc6, zc6, vxc6, vyc6 = xy_distribution(xlist3, ylist3, zlist3, vx3, vy3, Ldet2)
plot_6_heatmap(xc1, yc1, xc2, yc2, xc3, yc3, xc4, yc4, xc5, yc5, xc6,
    yc6,fudgefactor,s00, detuning, hexvolt = Voltagehex, Ldetection1 = Ldet1,
    Ldetection2 = Ldet2,plot_circle = plot_circle,save_plot = save_plot,extra_title =
    extra_title)
```

Finally the output of the **simulate_trajectories.py** file is

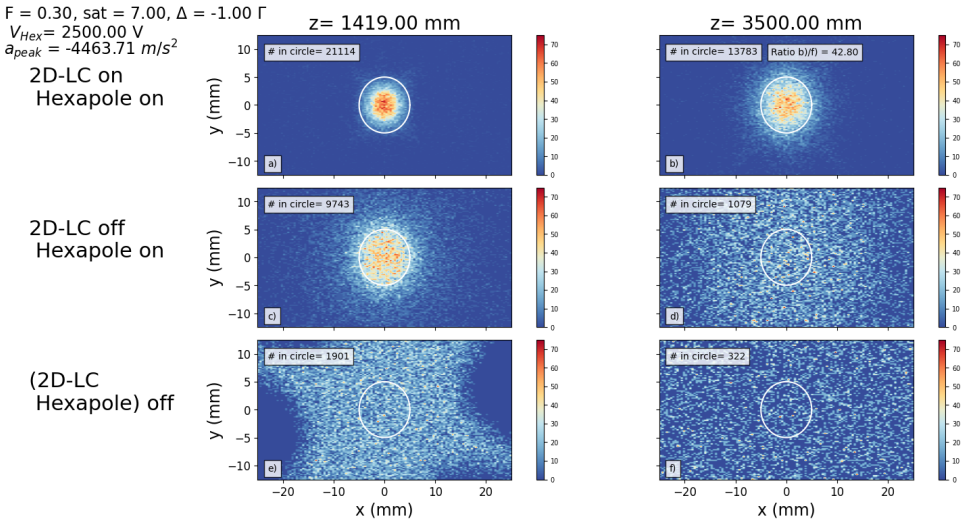


Figure 14: x-y distribution for different simulations

4 Modifications to DataManager.py and PlotManager.py

To be able to make a better comparison between the experimental data and the simulated data it was necessary to be able to plot the simulated data with the software that plots the experimental data. This software is comprised of two scripts, **DataManager.py** and **PlotManager.py**

4.1 DataManager.py

The DataManager.py script handles all the data collected by the experiment. It handles the files generated by all the instrumentation related to the laser cooling experiment. It was necessary to add a function to this script to be able to read data generated by the simulations.

The only modification made to DataManager.py was the addition of a method to the DataManager class.

- **read_simulated_data(self, filename, _directory="")**: *fillename* is the file name of the data we want to read, *_directory* is the directory where the data is located. It reads the data saved by the function *simulate_and_save_data*.

4.2 PlotManager.py

The PlotManager.py script works by plotting all the plots of interest to the laser-cooling experiment. It plots the xy-distributions of the molecules after they exit the experiment, the intensity of the beam as a function of voltage, among many other things. To make the simulated xy-distributions it was necessary to add a few functions.

- **add_simulation_image(self, _filename, _colorbar_orientation=None, _subplot = [0,0], bins=129,x_edges=[-25,25],y_edges=[-12.5,12-5],**kwargs)**: *_filename* is the name of the file to be read. *_colorbar* is the orientation of the color bar. *bins* is the number of bins to be used for the plotting. *x_edges* are edges of the image in the x axis,*y_edges* are edges of the image in the y axis.
  It adds a heatmap to an already-created figure. It works in the same way as *add_camera_image*

- **add_simulation_2D_gauss_contour(self, _filename,x_edges=[-25,25],y_edges=[-12.5,12.5] ,bins=129, **kwargs)**: *_filename* is the name of the file to be used.*bins* is the number of bins to be used for the plotting. *x_edges* are edges of the image in the x axis,*y_edges* are edges of the image in the y axis. It returns popt, pcov which are the result of the 2D-gaussian fitting to the xy-distribution. *add_camera_2D_gauss_contour* returns the same parameters in addition to also returning contour_2D_gauss_fit and unfitted_noise.

5 Pylcp integration.

Given a detuning, a saturation and magnetic field the pylcp simulations return the acceleration curve produced by the transition $X^2\Sigma^+ \rightarrow A^2\Pi_{1/2}$ on a BaF molecule. This acceleration curve is a result of the scattering force. For a two-level system, the scattering force looks like.

$$F_{scatt} = \hbar k \frac{\Gamma}{2} \frac{I/I_{sat}}{1 + I/I_{sat} + 4\delta^2/\Gamma^2} \tag{2}$$

This equation doesn't consider the transition of the molecule and just works with the natural line width of the molecule, the saturation, and the detuning of the laser used for the laser cooling.

Given a saturation s and a detuning δ, the acceleration curve the pylcp simulations generate with these parameters is not the same acceleration curve that the fitting of eq. 2 would produce. It is necessary to find a saturation s' and detuning δ' to compare these two curves (the one generated with pylcp and the one fitted to eq. [2]). See Fig. 12.
If a continuous function of the scattering rate ff, saturation s' and detuning δ' as a function of the saturation s was fitted (for a given δ), then it could be possible to find the corresponding parameters to fit the acceleration curve of a two-level system to a pylcp simulation that hasn't been made.

The code that does this polynomial fitting is written in **model_fit_max_data.py** and its run in **analysis.py**.

5.1 model_fit_max_data.py

- **ascat(vx,fudgefactor,s0,detuning)**: *vx* is the velocity of the molecule. *fudgefactor* is the scattering rate. *s0* is the saturation, *detuning* is the detuning.
  This function fits the acceleration curve to the two-level system equation. eq. 2.

- **get_parameters_from_force_curves(det,saturation)**: *det* is a list of detunings corresponding to a pylcp simulation, *saturation* is a list of saturations corresponding to a pylcp simulation. It returns the parameters that fit the respective acceleration curve best. Its output is *(ff,sat',det'*, the scattering rate, saturation, and detuning.

- **pol_to_rule_them_all(x,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11)**: It's an 11th-grade polynomial that fits the curve that a given parameter would make as a function of the saturation s. This is not the fit for any physical parameter per se but instead is the fitting of the curve that a given fitted parameter makes. (It is the fitting of a fitting).

- **fit_polynomial(sat,y,sat_pylcp)**:*sat* is a list of saturation fitted values. y is an array of fitted values (it can be either the scattering rate, the saturation or the detuning). *sat_pylcp* is an array of saturations used in the pylcp simulations.
  It fits the corresponding curve of y as a a function of *sat_pylcp*

- **get_ff_sat_det(detun_pylcp,sat_pylcp)**:*det_pylcp* is a detuning corresponding to a pylcp simulation, *sat_pylcp* is a a saturation corresponding to a pylcp simulation.
  It returns the corresponding values that would fit the same acceleration curve made by a two-level system, *fudgefactor, saturation, detuning*

- **connect_dots(x,y)**: Given a set of data points it parametrizes a line between each of those dots returning a continuous function of lines.

- **plot_fitted_parameter(sat2,fit, parameter)**: *sat2* is an array of pylcp saturations,*fit* is an array with the same length len(*sat2*), *parameter* is a string with the name if the fitted parameter.

The following is an example of the **analysis.py** script

```
from model_fit_max_data import *
import numpy as np
detun = np.array([-0.5,-1,-1.5,-2]) #pylcp detunings
saturation = np.arange(0.5, 15.5, 0.5) #pylcp saturations
ss=[]
dd=[]
ff=[]
ss2=[]
for det in detun:
    ff_fit,sat_fit,det_fit,sat2,R2=get_parameters_from_force_curves(det,saturation)
    ss.append(sat_fit)
    dd.append(det_fit)
    ff.append(ff_fit)
    ss2.append(sat2)
plot_fitted_parameter(ss2,dd,'detuning')
```

We define detun and saturation as a list of pylcp detunings and saturations respectively. Then we define four empty lists that will store the given parameters of the respective detunings and saturations. It's redundant but ss2=saturation. This code returns the following figure.

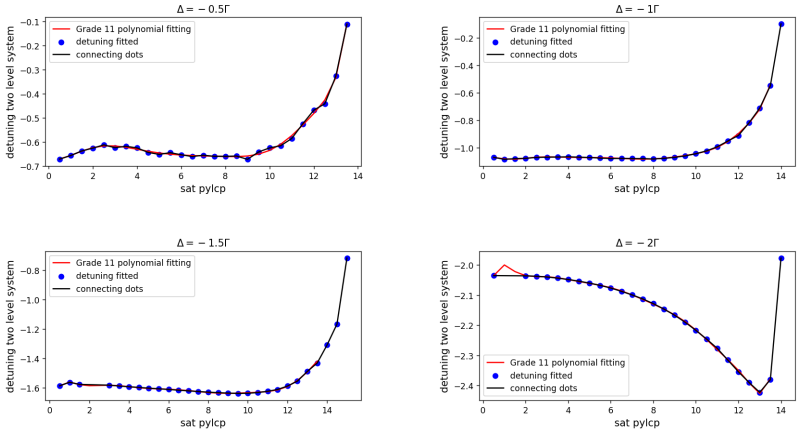


Figure 15: detuning fitting as a function pylcp saturation

6 General remarks

Up to this point, I have explained how the code works and what are the results one can obtain. In this section, I will venture more into the significance of some plots and some general conclusions.

The goal of the simulations is to have a method of comparison with the experiment to have a better understanding of the measurements that are made and, moreover to have a prediction of some quantities we can't measure. That is the case of the FWHM of the beam at the spin precession distance. As one can see in figure 10, one of the goals was to compare the top right image to the top left image. The parameters that maximize the number of counts inside the beam at $z = 3500mm$ are *ff =0.3*, *saturation = 7*, *detuning = -1* Γ, and *Hexapole voltage = 2.5 kV*.

On the other hand, the polynomial fitting of the different parameters for the two-level system acceleration curve suggests that for saturations above $s = 10$, the data starts to skew a lot, so I would recommend to not using saturations greater than 10.

Again revising Fig 10 one can see that the shape of the beam curves on its sides, this is due to the shape of the beam, being of a Gaussian shape. The original plots returned a more squared beam.

The original code had a uniform force throughout the entire laser cooling region. I did the implementation of a laser being reflected inside the laser cooling region to make the simulations more realistic. This is the shape of the intensity of the laser light after this implementation.

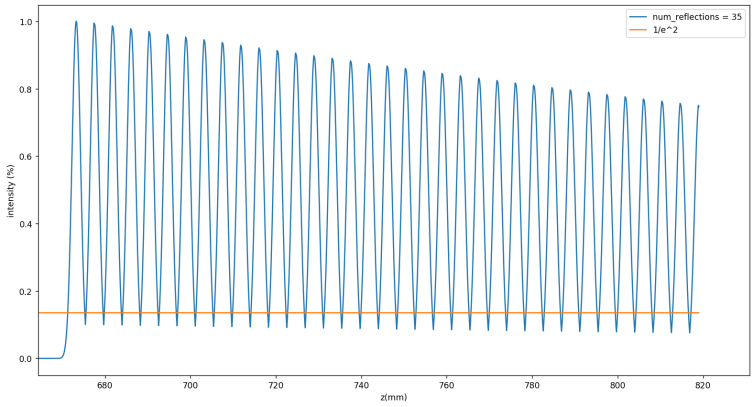
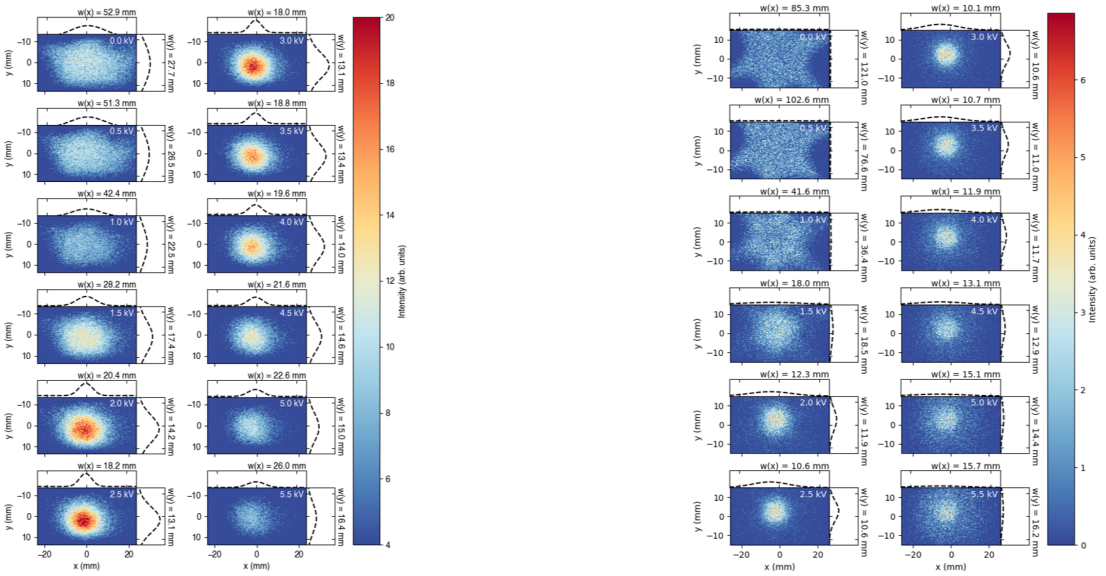


Figure 16: Intensity of laser light as a function of longitudinal distance

Finally, the implementations made in **PlotManager.py** and **DataManager.py** allow to made the following comparison.



(a) Experimental data

(b) Simulated data

Figure 17: This images contain the xy-distribution for molecules going through the hexapole for different voltages (no laser cooling). On the left we have the experimental data and on the right we have simulated data

The FWHMs on the y-xis of the experimental data (right image) has a similar behavior as the same value in the experimental data for most voltages. For $v = 2.5kV$, $w_{exp}(y) = 13.1mm$ and $w_{sim}(y) = 10.6mm$ which represent a difference of $2.7mm$. Most $w_exp(y)$ values don't diverge to

much from their simulated counterparts.
The problem arises when trying to compare the same value in the x-axis. Most $w_{exp}(x)$ are at least $7.4mm$ away from their simulated counterparts, that's more than twice the difference as in the y-axis. I think this difference is due to an alignment in the hexapole lenses. Nonetheless, I didn't work in the experimental setup so I can't make any sound suggestions as to the actual cause of this difference between $w_{exp}(x)$ and $w_{exp}(y)$.

For the previous figure the laser cooling stage was off. Simulating the laser cooling stage we get the following figure.

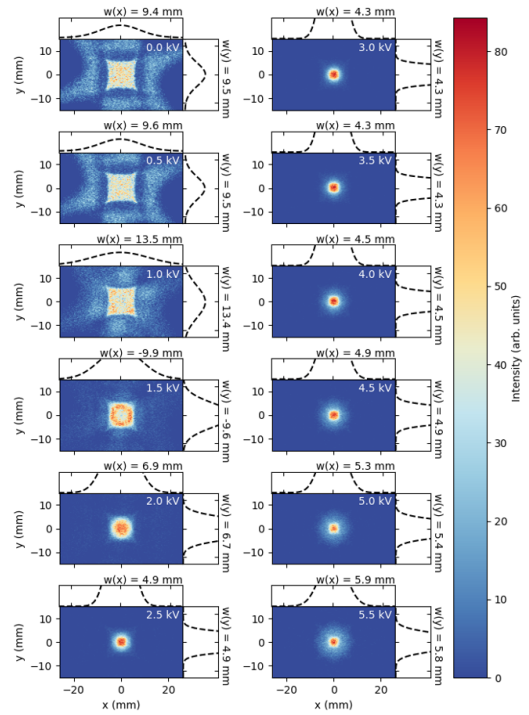


Figure 18: xy-distribution for different voltages, *ff=0.3,s=7, det=-1*

As it can be seen, the width of the laser cooling beam will decrease by almost half for most voltages. This is a direct consequence of laser cooling.

References

[1] M. Vos, "Laser cooling of barium monofluoride: Detailed simulation studies and reintroduction of vibrationally excited molecules," Master's thesis, University of Groningen, 2024.

[2] S. Y. T. van de Meerakker, H. L. Bethlem, N. Vanhaecke, and G. Meijer, "Manipulation and control of molecular beams," *Chemical Reviews*, vol. 112, no. 9, pp. 4828–4878, 2012. PMID: 22449067.

[3] N. Fitch and M. Tarbutt, "Chapter three - laser-cooled molecules," vol. 70, pp. 157–262, 2021.