Contents

1 Theoretical Background

This section is inspired by [5] and treats the theoretical background needed to perform the experiment and analyse the data.

1.1 VMI-Spectroscopy

A Velocity-Map-Imaging-spectrometer is used to determine the velocity of ions/electrons originating from an ionization volume. The spatial distribution is of no direct interest using this method but the experiment should be build in a way that the initial spatial distribution does not influence the measurement, which is realized by a proper setting for the ion optics. A schematic setup of a VMI-spectrometer is shown in fig. 1: Between the electrodes atoms are ionized by a laser beam and accelerated towards the detector system. The electrodes forming the ion optics should be setup to compensate for any initial spatial-distribution. The setup used to carry out the experiment is described in section 2.

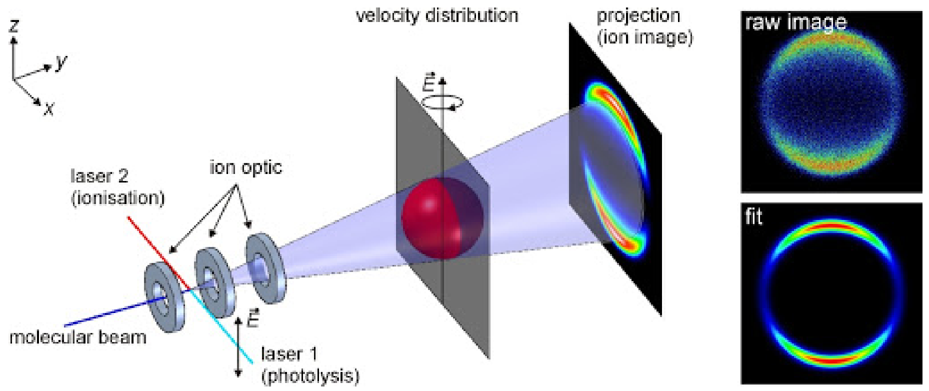


Figure 1: Schematic sketch of the setup of a VMI-spectrometer. Potassium is ionized by a laser and the ions are accelerated by the ion optics. Due to different velocities the beam expands on a sphere that is mapped on the detector what gives the raw image. With help of the Abel inversion the raw image can be the velocity map can be created. Image taken from [1]. Note however that the shown setup differs from the actual setup used in this experiment. The used setup crosses the laser beam and the particle beam in a plane parallel to the detector screen.

1.1.1 The Velocity Distribution

To determine the properties of the velocity-distribution we assume the particles are ionized in a single point such that the initial spatial distribution is not relevant. Each of the ionization products has an initial velocity $\mathbf{v}_i = (v_{xi}, v_{yi}, v_{zi})^\top$ that we want to measure. The particles are accelerated in $\mathbf{e}_z$-direction by the electric field between the electrodes (ion optics), such that the initial 3-dimensional distribution is mapped on a 2-dimensional detector surface. Using a linearly polarized laser for ionization a cylindrically symmetric velocity-distribution in direction of the polarisation is created. With help of the inverse Abel-transformation the initial distribution can be reconstructed out of the two-dimensional image, given the symmetry properties of the initial distribution.

Abel Transformation and Inversion For a cylindrically symmetric distribution $f(r, y)$ the observed signal can be calculated by

$$F(x, y) = \int_{-\infty}^{\infty} f(r, y)\, \mathrm{d}z = 2 \cdot \int_{0}^{\infty} f(r, y)\, \mathrm{d}z, \tag{1}$$

which can be rewritten by substituting $r^2 = x^2 + z^2$ and $\mathrm{d}z = \frac{r}{\sqrt{r^2 - x^2}}\, \mathrm{d}r$, which yields

$$F(x, y) = 2 \cdot \int_{|x|}^{\infty} \frac{f(r, y)\, r}{\sqrt{r^2 - x^2}}\, \mathrm{d}r. \tag{2}$$

Where $F(x, y)$ is the Abel-transformation, used to calculate the observable image of an initial distribution f.

In this experiment the Abel-inverse-transformation is needed to calculate the initial distribution with help of the image. The inverse transformation is given by

$$f(r, y) = -\frac{1}{\pi} \int_{|r|}^{\infty} \frac{\mathrm{d}F(x, y)}{\mathrm{d}x} \frac{1}{\sqrt{x^2 - r^2}}\, \mathrm{d}x. \tag{3}$$

To analyse experimental data this inverse transformation can not be used directly, since the image $F(x, y)$ will not be continuously differentiable due to the finite resolution of the detector signal. Hence, we need a numerical method to reconstruct the desired 3-dimensional velocity distribution.

BASEX-Method As the formal inversion eq. (3) can not be solved analytically for the experimental data, the BASEX-method is used as a numerical approach. The BASEX-method uses basis functions $\bar{f}_k$ in the space of f with known projection to the detector, calculated by eq. (2). The basis functions should be analytically integrable, such that the Abel-transformation can be calculated. Furthermore the distribution after the transformation should be able to show sensible small structures and be smooth on small distances. The basis

$$\bar{f}_k = \left(\frac{e}{k^2}\right)^{k^2} \left(\frac{r}{\sigma}\right)^{2k^2} \exp\left(-\left(\frac{r}{\sigma}\right)^2\right) \tag{4}$$

satisfies these conditions. The parameter σ determines the width and positions of the maxima and should be chosen close to the magnitude of the smallest structure the detector can resolve. For this basis the Abel-transformation is

$$\bar{F}_k(x) = 2\sigma\rho_k(x) \left(1 + \sum_{l=1}^{k^2} \left(\left(\frac{x}{\sigma}\right)^{-2l} \prod_{m=1}^{l} \frac{(k^2 + 1 - m)(m - \frac{1}{2})}{m}\right)\right) \tag{5}$$

Expanding the measured image F according to $\bar{F}_k$ gives the searched initial distributions f as a linear combination of the basis functions $\bar{f}_k$. In fig. 2 some basis functions for the BASEX-methods and the Abel-transformed images are shown.

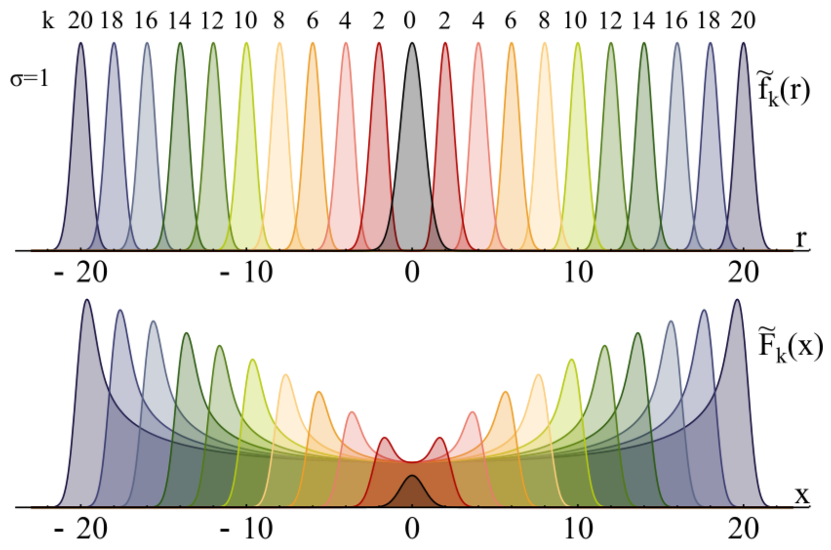


Figure 2: In this figure the basis functions for the fit of the raw data are shown (above) and the Abel inversed basis functions are shown (below) (source: [5]).

1.2 Properties of a Gaussian Beam

In the previous parts it was assumed that the ions originate a point source. In the experiment this is not realized and the ionization area depends on the appearance of the focused laser. To describe the width of the laser beam the model of Gaussian beams is used, the principle is sketched in fig. 3. In direction of the propagation the beam shows a Lorenz-profile and perpendicular to the axis the profile is Gaussian. The minimum width can be found in the focus of the beam. The intensity in dependence of the distance from the z-axis r, where z is the direction of propagation is given by

$$I(r,z) = I_0 \left(\frac{w_0}{w(z)}\right) \exp\left(-\frac{2\,r^2}{w(z)^2}\right). \tag{6}$$

In this equation the origin is set in the focus of the beam and

$$w(z) = w_0\sqrt{1+\left(\tfrac{z}{z_R}\right)^2} \tag{7}$$

gives the width of the laser perpendicular to the direction of propagation. The parameter w_0 is the width of the laser in the focus, hence the radius of the beam at $\frac{1}{e}$-amplitude. For a beam focused by lenses and assuming a parallel setup, the width of the beam can be calculated by $w_0 = \frac{\lambda f}{\pi w_l}$ with λ the wavelength of the laser, w_l the width of the beam at the position of the lens and f the focal length of the lens. The parameter $z_R = \frac{\pi w_0^2}{\lambda}$ is called Rayleigh length and the focal area of the laser is the interval $[-z_R, z_R]$ centered around the position of the focus. At the edges of this interval the width is given by $w(-z_R) = w(z_R) = \sqrt{2}w_0$.

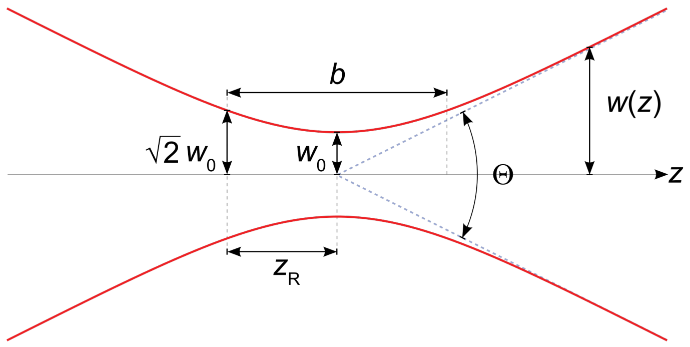


Figure 3: Sketch of the focal region of a Gaussian beam. Image from [2], accessed: 11.03.2021

As the VMI-spectroscopy should be independent of the initial spatial distribution, the finite ionization area created by the laser has to be compensated, since a point-like ionization area was assumed. This is done by arranging the ion-optics such that an inhomogeneous electric field is generated. In this experiment this is done following the Eppink-Parker-Design with three electrodes as sketched in fig. 4. The electrodes are the closed repeller- and the open extractor- and ground-electrodes. Depending on the distances of the electrodes as well as the radius of the holes in the open electrodes an ideal ratio of extractor- and repeller-voltage $0 \leq \frac{U_E}{U_R} < 1$ has to be chosen. The procedure to determine this ratio is described in section 4.

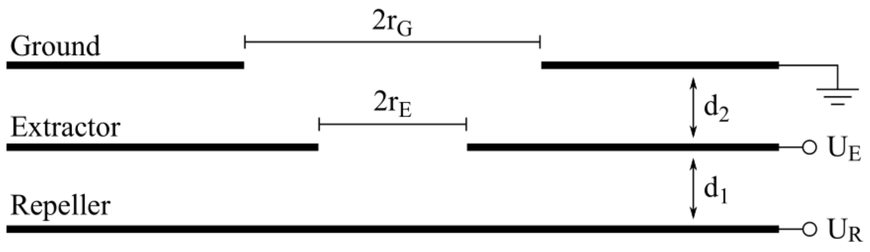


Figure 4: Eppink-Parker-Design with repeller-, extractor-, and ground-electrode. Figure taken from [5].

1.3 SMI-Spectroscopy

The same spectrometer with different settings in the ion optics can also be used to determine the spatial distribution of a beam. This setup is called Spatial-Map-Imaging-spectrometer and the ion optics have to be configured in a way that two atoms or electrons that originate the same place are mapped on the same place on the detector, regardless of their initial velocities. The switch between SMI- and VMI-spectroscopy can be done fast and it solely depends on the voltages applied on the electrodes in the ion-optics, so one setup can be used to determine both distributions.

1.4 Properties of Potassium

Potassium is an alkali metal. It has atomic number 19 and the term symbol K. The most common isotope is ^{39}K with an natural occurrence of 93.26 % [3]. Potassium is very reactive, hence in the setup of the experiment it hast to be handled carefully. Any contact with air humidity has to be avoided, therefore potassium has to be kept under vacuum conditions. The electron configuration is [Ar]4s^1 and the level scheme is shown in fig. 5.

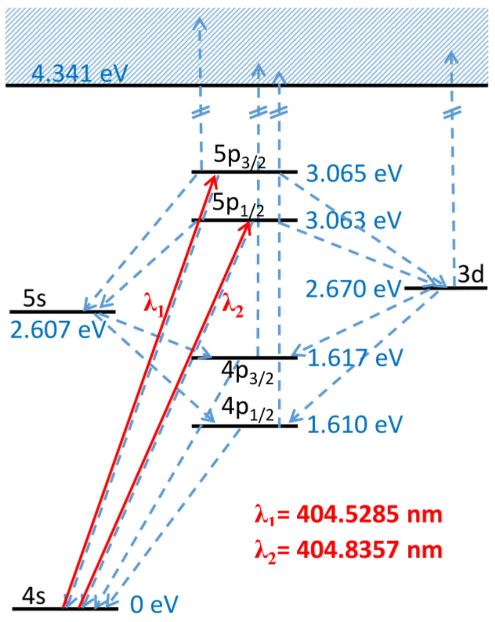


Figure 5: Level scheme of Potassium. The laser populates the 5p$_{3/2}$-state, from which the atom can be ionized (REMPI-path). The other levels get populated by spontaneous decay and can also be ionized by the used laser, allowed transitions are indicated by the dashed lines. Figure taken from [8]

1.5 Photoionization and the Anisotropy-Parameter

Potassium can not be directly ionized from the ground state with the used laser. The laser has a wavelength of about 404.5 nm and can be used to populate the $5p_{3/2}$ state from which it can be ionized, so it follows a two photon REMPI (Resonant Enhanced Multi Photon Ionization) process $4s_{1/2} \rightarrow 5p_{3/2} \rightarrow K^+$, which leads to a kinetic energy of circa 1.8 eV of the emitted electron. If the system relaxes before ionisation it can be ionized out of one of the lower states, which leads to a different kinetic energy. The different possible paths that can be taken are illustrated in fig. 5. Therefore the energies of the different excited states can be determined using

$$E_{\text{level}} = E_{\text{kin}} + E_{\text{ion}} - E_{\text{laser}}, \tag{8}$$

where E_{laser} is the energy of the used laser, E_{kin} the measured kinetic energy and E_{ion} the energy needed to ionize Potassium.

The K-atoms inside the ionisation volume are not polarised, therefore the directions of the orbital angular momenta are oriented arbitrarily. The laser used for ionisation is linear polarized so we will observe all possible projections of the orbital angular momenta on the direction of polarization. Therefore the measured distribution $J_{nl}(\Theta, \Phi)$ is the cumulation over all possible distributions $J_{nlm}(\Theta, \Phi)$ with respect to the quantum number m, so for the absorption of a linear polarised Photon we find

$$J_{nl}(\Theta) = 1 + \beta P_2(\cos\Theta). \tag{9}$$

where $P_2(x) = 1.5x^2 - 0.5$ is the second Legendre polynomial and $\beta \in [-1, 2]$ is the anisotropy parameter. As the REMPI process is a two photon ionisation two anisotropy factors will be encountered and the distribution is given by

$$J_{nl}(\Theta) = 1 + \beta_2 P_2(\cos\Theta) + \beta_4 P_4(\cos\Theta) \tag{10}$$

with the fourth Legendre polynomial $P_4(x) = 4.375x^4 - 3.75x^2 + 0.375$.

2 Experimental Setup

2.1 The Laser System

The used laser is a 'Toptica DL pro', a single mode tunable diode laser. The frequency of the laser can be changed by adjusting the laser current or changing the piezo voltage. The piezo element moves a grating which reflects light back into the laser diode and adjusting the grating will change the frequency which gets amplified by the laser diode. Reflections and scattering could cause disturbances in the laser so they are avoided by setting up an optical diode into the beam directly before the laser exits. Additionally the temperature of the diode impacts the band gap of the semiconductor and therefore can also change the frequency of the emitted light.

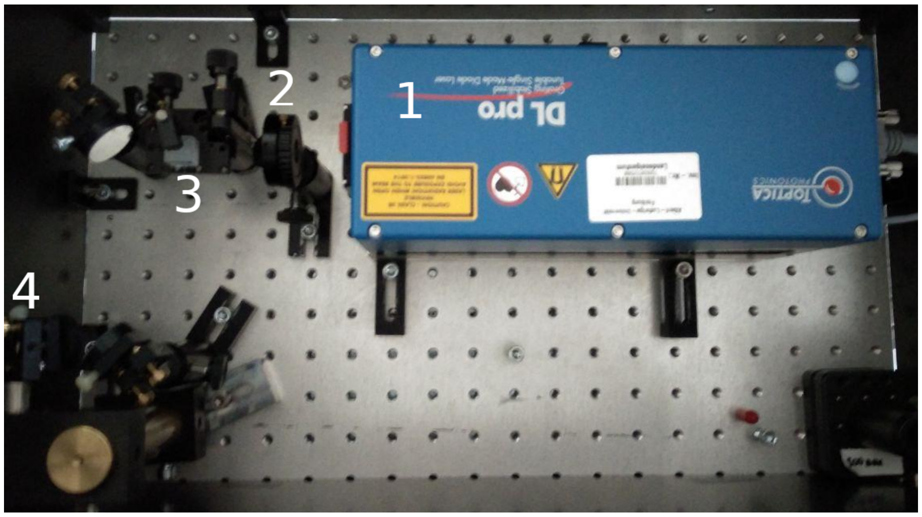


Figure 6: Setup of the beam path. 1: laser, 2: $\lambda/2$-plate, 3: PBS, 4: the beam leaves this setup here through a hole in the wall on the left side

Optical Elements To adjust the beam path lenses, mirrors, a polarizing beam splitter and a $\lambda/2$-plate are used. The setup is shown in fig. 6.

$\lambda/2$-plate: A $\lambda/2$-plate is made of an anisotropic material which has different refractive indices for the ordinary and the extraordinary beam. Going through the material both beams undergo a phase shift and by adjusting the length of the material the relative shift between the two beams can set to $\lambda/2$. Hence linearly polarised light with an angle of θ to the optical axis of the material will have a polarization of $-\theta$ after passing the $\lambda/2$-plate.

Polarizing Beam Splitter (PBS): A PBS splits the beam in two perpendicular polarized parts, so the initial beam could be described as the superposition of an orthogonal and a parallel polarized wave.

2.2 K-Oven

In an oven solid potassium is heated so K-atoms evaporate. The solid potassium is placed inside a small steel container which is wrapped by a copper clamp on which the heaters are mounted. As a well collimated beam is needed just a simple hole were the atoms could leave the oven would not be sufficient, but the metal around the hole is cooled. Hence around the hole atoms condensate at the walls and only the atoms with a well defined velocity vector can leave the container. In the ionisation region the potassium beam has a diameter of about 3.5 mm. The placement of the oven in the setup is shown in fig. 7.

2.3 LT-Detector

The Langmuir-Taylor-detector is a simple apparatus used to detect alkali atoms. In this setup it is used to check if the evaporation of K-atoms works as expected, therefore it is placed directly opposite of the oven. A Rhenium filament ionizes the K-atoms between two electrodes. The electric field accelerates the ions through a hole in one of the electrodes towards a Faraday cup, where a current is measured. With the measured current and the area of the ionising filament, which is given as 6 mm ([7]) the intensity of the beam can be calculated. The placement of the atom-detector is shown in fig. 7.

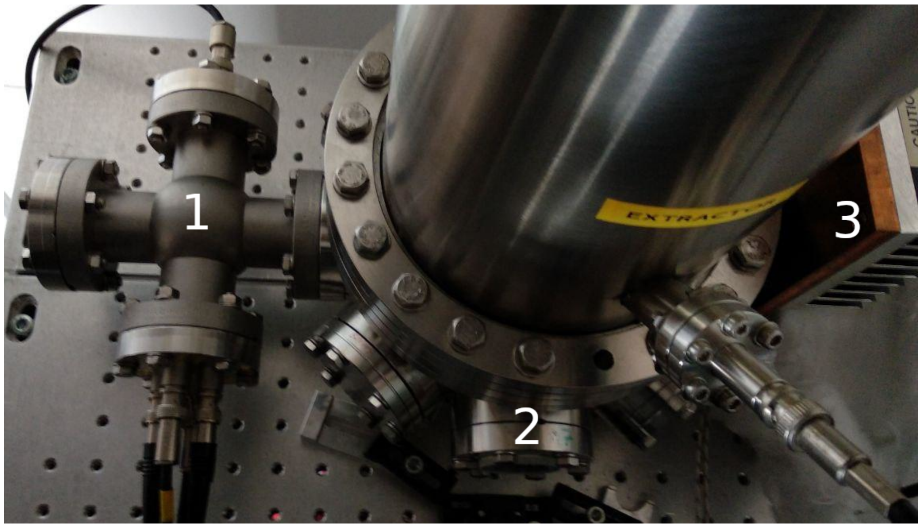


Figure 7: K-oven, LT-detector and ion-optics. 1: LT-detector, 2: The light that leaves the beam path shown in fig. 6 is focussed by a lens and enters the detector setup here, 3: K-oven

2.4 Surface Detector

The ionized K-atoms are accelerated towards the surface detector so the data for the 2-dimensional velocity map can be taken. The detector consists of three parts: Amplifying micro channel plates, a detecting phosphor screen and a light sensitive camera. The setup of the detector is shown in

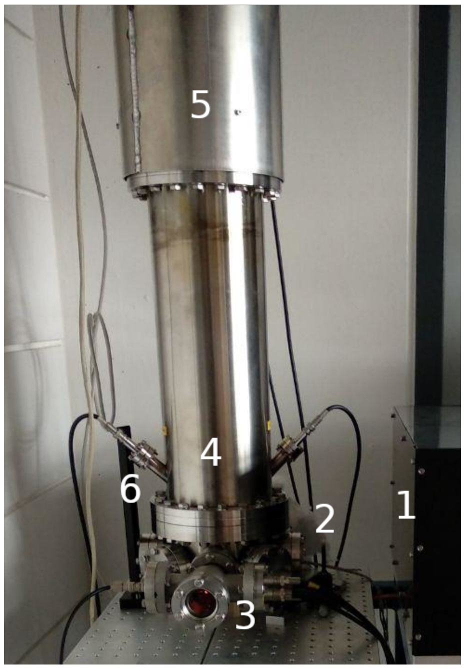


Figure 8: Outside of the detector. 1: the beam leaves the setup shown in fig. 6 here, 2: lens used to focus the laser (f = 150 mm, 3: glowing filament as part of the LT-detector, 4: at this high the ion-optics (Eppink-parker-Design with repeller-, extractor-, and ground-electrode) are placed, 5: from bottom to top the MCP, the Phosphor-screen and the camera are placed here, 6: the laser beam leaves the vacuum chamber here

Micro Channel Plates (MCP) To amplify the signal before it is detected by the screen, a set of micro channel plates is used. A MCP is made of highly resistive material and when a charged particle hits the front of the MCP electrons are generated which leave the MCP on the other side. The principle is further illustrated in fig. 9.

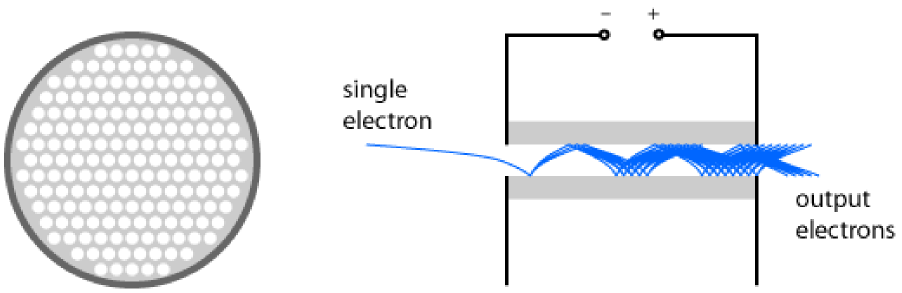


Figure 9: Illustration of a MCP. On the left the front view is shown. On the right side the amplification process is illustrated: When an electron (or any charged particle) hits the MCP the electron is accelerated due to the voltage between front and back of the MCP. While passing one of the channels, the electron hits the walls and generates more electrons, that leave the MCP on the other side. Image taken from [4].

Phosphor Screen The amplified signal, so the electrons that leave the MCP are accelerated towards a phosphor screen. Each time an electron hits the screen they cause a light flash which can be detected by a suitable camera. The used phosphor screen has a flash that takes circa 4 ms to decay [7].

Camera The flashes of light are captured by a charge coupled device camera and the pictures are transferred to a computer for further use.

2.5 Vacuum System

As potassium is very reactive, it needs to be kept under high vacuum conditions. Additionally the MCP should only be used in a vacuum, otherwise electrical discharges of possible contamination could occur. The vacuum is established with help of a membrane pump and a turbo-molecular pump.

3 Procedure

The first part of the experiment consisted of VMI- and SMI-spectroscopy with help of simulations. During this part the data shown in section 4 was generated and is also discussed there.

General Preparation Before starting the experiment, the optical path was checked. A piece of paper was used to check the path before and after each optical element and the diameter and the brightness did not change during the path, so it was already aligned. Furthermore it was checked if the laser entered and left the vacuum chamber at the same hight. Since this was also the case the beam path was not readjusted.

Preparing the Oven and the Voltages To heat up the oven, the heater voltage was set to 33 V and it was taken care, that the oven temperature does not get higher than 160 °C. For measurements in SMI- and in VMI-mode the voltages of the MCP, the phosphor screen and ion-optics had to be set. The desired polarity (positive for ions and negative for electrons) was chosen and afterwards the input voltage of the voltage divider was set to 3 kV. Afterwards MCP and phosphor screen were slowly set to the right voltages in steps of about 50 V to 100 V. This process has to be slow and alternating, so the difference between U_{MCP} and U_{Ph} does not get higher than 3 kV, and stays as low as possible, and in a way that any dust or remaining charge in the vacuum chamber does not destroy the MCP$\dot{\mathrm{T}}$he used voltages were about $U_{\mathrm{MCP}} = 1600\,\mathrm{V}$ and $U_{\mathrm{Ph}} = 3.4\,\mathrm{kV}$.

Setting the Laser Wavelength As the K-Cell-setup originally used to set the laser to the correct wavelength was broken, a spectrometer was used to roughly set up the laser. The spectrometer was placed behind the vacuum chamber (see position 6 in fig. 8) and connected to a computer. The placement of the spectrometer had to be chosen in a way that the laser does not directly hit the spectrometer, but only a small part is measured, otherwise the measured intensity would be too high. With help of the output of the spectrometer (see fig. 29) it was tried to set the maximum of the peak to about 405.5 nm by changing the laser current and the piezo voltage. As the signal of the spectrometer varied a lot, even though the parameters on the laser were not changed this method does not seem to be a good approach to determine the right laser settings. It has to be noted that the optimal settings for the laser using the spectrometer where not used for later measurements since no signal was measured using them. Altough the wavelength was set right the intensity was probably not enough. Therefore it was decided to continue with setting up the laser with help of the signal of ions in SMI-mode.

With the voltages set as explained before, the voltage ratio $U_{\mathrm{E}}/U_{\mathrm{R}} = 0.9$ was set and the program `FlyCapture` was used to see the signal of the camera. For the set voltages and the chosen voltage ratio a signal is expected, so the parameters of the laser were changed until signal was found. The first signal was found for a laser current of about 50.625 mA and a piezo voltage of 1.575 V. As these laser settings were not really stable the values were adjusted a lot during the following measurements, but were mostly kept close to the first obtained values.

Spatial Map Imaging with Ions Having set up the laser, the first measurements were performed. To later determine the image ratio, the lens used to focus the laser before entering the vacuum chamber (see fig. 8) was moved to different positions and for each position the obtained signal of the camera (in SMI-mode a small horizontal line) was saved and the corresponding position of the lens was noted.

As the optimal voltage ratio $U_{\mathrm{E}}/U_{\mathrm{R}}$ found in the simulations is not necessarily the perfect ratio for the experimental setup, the signal of the camera was taken for different voltage ratios. This data can later be used to determine the ratio with the best, thus the sharpest signal.

Velocity Map Imaging with Electrons Same as for SMI-mode the simulated voltage ratio is most probably not perfect. Therefore the signal for VMI with electrons (we expect three circles, but for most of the cases only two are easy to see) has been taken for different voltage ratios. Again this data can be used to find the ratio with the sharpest signal. For the best voltage ratio the signal has been optimized again by tuning the laser and changing MCP- and Phosphor-screen voltage a little, so the signal is bright and sharp and several pictures have been taken.

Furthermore a background signal has been taken, means the voltages were kept but the laser was turned off.

To find a relation between the signal and the repeller-voltage, the VMI-signal was taken for different repeller voltages while keeping the ratio constant.

Abel-Inverse with `pBASEX` To determine the energy of the K-states and the anisotropy parameters the program `pBASEX` has been used to process the taken pictures. The raw data can be loaded in the program and the Abel-inverse can be calculated. Different options can be used to achieve a better picture and also the center of the image has to be set by hand carefully. Moreover the background can be loaded into the program and subtracted. The data of the whole Abel-inverse picture, the photo electron spectra in dependence on the radius and estimated anisotropy-parameters in dependence on the radius can be saved.

Oven-Measurement To check the relation between the atomic flux and the temperature of the K-oven, the flux was measured for different temperatures. First of all the oven was heated to a maximum. Afterwards the current through the Re-filament was switched on and checked whether it started glowing. To measure the flux in dependence on the temperature, the repeller and extractor voltages were changed so a maximum current was measured in the femtoamperemeter. This maximum current was found for a ratio of $U_{\mathrm{E}}/U_{\mathrm{R}} \approx 0.54$. For this fixed ratio the heater-voltage was switched off and pairs of measured current and corresponding temperature were collected.

4 Simulations

In the first part of the experiment we did simulations using the program `SimIon`[6]. This program solves the Poisson equation for a given charge distribution numerically. Therefore the trajectory of a massive charged particle with given initial conditions can be simulated, which is also done by `SimIon`. An implementation of the charge distribution was given and loaded into the program. Since there was no information given on the precision of the simulation, no direct errors on the simulated data are taken into account. Also for reasons discussed later in the protocol, it has to be assumed that the loaded charge distribution differed from the one used in the experiment.

4.1 Optimal Voltage Ratio in VMI Mode

In the theoretical description of the experiment it is assumed that the potassium atoms are ionized in a single point, which of course cannot be realized in experiment. It is therefore necessary to compensate for the size of the ionization volume. Hence, we adjust the ion optics (repeller and extractor) in a way that two velocity vectors, which source at different points within the ionization volume, are mapped on the same spot on the detector screen.

At first two electrons where placed in the ionization volume with identical velocity vectors but different spatial starting points. The first electron was set $z = 0.5\,\mathrm{mm}$ above the center of the ionization volume and electron 2 was set to $z = -0.5\,\mathrm{mm}$ below the center. The repeller voltage was set to $U_\mathrm{R} = -3\,\mathrm{kV}$. To find the optimal ratio of the extractor voltage U_E and the repeller voltage U_R, we scanned the range $U_\mathrm{E} = 2\,\mathrm{kV}, \ldots, 3\,\mathrm{kV}$ in steps of $0.1\,\mathrm{kV}$. The optimal ratio is given when the distance on the detector screen is minimal, since ideally the two electrons should be mapped on the same point because they started with the same initial velocity. We did a quick analysis after the first scan to check for a smaller interval within which the minimum should be found. This strategy was iterated a few times. The results of the simulations are shown in fig. 10. The position of the minimum was estimated to be 0.862 ± 0.002. This results in an optimal extractor voltage of $U_\mathrm{E} = (2586 \pm 6)\,\mathrm{V}$, given a repeller voltage of $U_\mathrm{R} = -3\,\mathrm{kV}$. Since there is no known theoretical function which would be sensible to fit to the data the estimation of the minimum was performed by direct examination of the experimental data. The error on the estimated ratio was estimated likewise.

Variation of U_R keeping U_E/U_R constant

For the next task we checked the behavior of the simulated signal, for changing repeller voltages U_R while keeping the optimal voltage ratio, hence changing the extractor voltage U_E according to the ratio. As expected the distance on the screen was still a minimum, since when we chose U_E different from the value dictated by the optimal ratio the distance on the screen increased. It was noticeable that the minimum distance at this ratio was changing when we changed the repeller voltage U_R. The minimal distances are plotted against the repeller voltage in fig. 11 and we see, that the distance gets smaller for higher voltages.

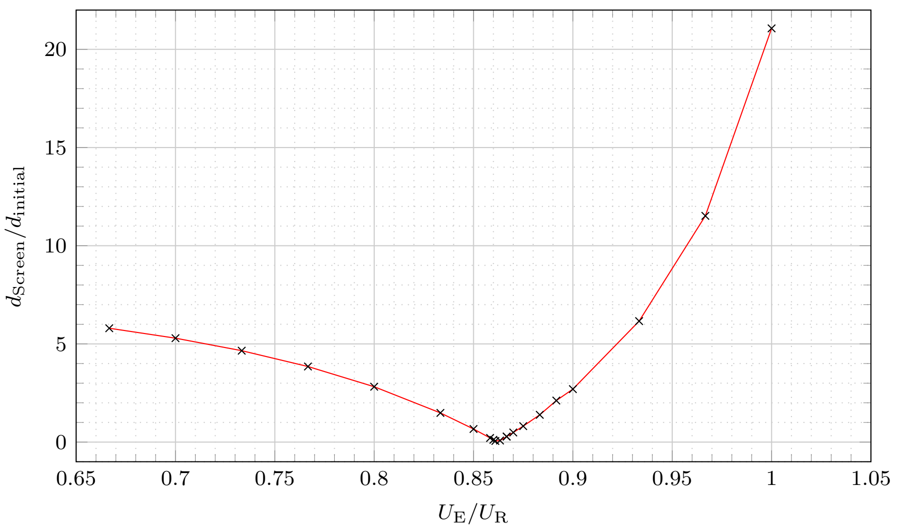


Figure 10: In this figure the ratio of the distances is plotted against the voltage ratio. The position of the minimum was estimated to be 0.862 ± 0.002. Since there is no theoretical function which could be fitted to the data the estimation was performed by examination of the data with help of the red line.

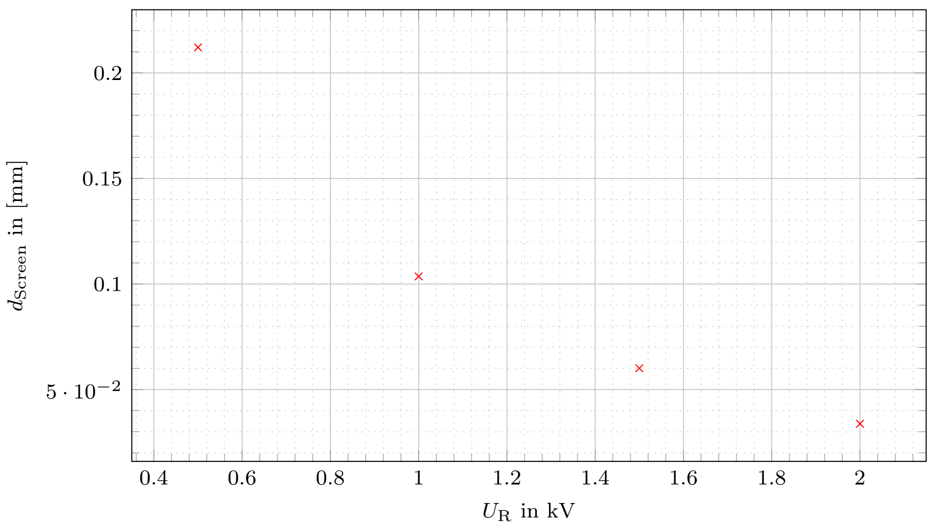


Figure 11: In this figure the minimal distance on the screen is plotted against the chosen repeller voltage U_R.

4.2 Simulation of many Particles in VMI Mode

Now 500 particles where set to the same initial position in the center of the ionization volume. The azimuthal and polar direction of the velocity vectors where distributed by a uniform distribution, whereas the absolute of the velocity vector was set by choosing a particular kinetic energy for all the particles particle. We have chosen the kinetic energies $E_{\text{kin.}} \in \{0.1\,\text{eV}, 0.2\,\text{eV}, 0.3\,\text{eV}\}$. The measured signal for the three kinetic energies is shown in fig. 12. It is seen that the three kinetic energies correspond to signals with three different radii, just as expected.

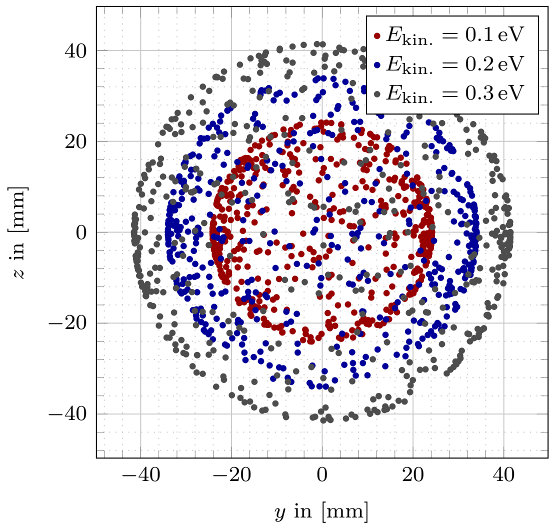


Figure 12: In this figure the simulated detector signal is shown.

Estimation of the Radii (Optional)

To estimate the radii we used the data displayed in fig. 12. For every point we calculated the distance to the middle of the detector, and then sorted the data by distance. To estimate the radius for a given kinetic energy the ten greatest radii of the corresponding sample where used to calculate the mean value. For good measure we also calculated the standard deviation, although it has to be emphasized that a sample size of 10 radii wont allow an extremely precise estimation of the mean value and the standard deviation. For a kinetic energy of $E_{\text{kin.}} = 0.1\,\text{eV}$ we got a radius of $r_1 = (24.1661 \pm 0.0005)\,\text{mm}$. For a kinetic energy of $E_{\text{kin.}} = 0.2\,\text{eV}$ we got a radius of $r_2 = (33.9873 \pm 0.0006)\,\text{mm}$. For a kinetic energy of $E_{\text{kin.}} = 0.3\,\text{eV}$ we got a radius of $r_3 = (41.4535 \pm 0.0005)\,\text{mm}$.

Electron groups for different U_E, E_R (Optional)

In accordance with the previous simulations we put 500 electrons on the same position inside the ionization volume. Again the polar and azimuthal angles are distributed uniformly. The same three energies as in the task before were used. In fig. 13 the simulated data is shown. As expected the radii of the signal decreases with increasing the repeller voltage. This is caused by the decrease of the time of flight, since the acceleration towards the detector screen is higher for higher voltages.

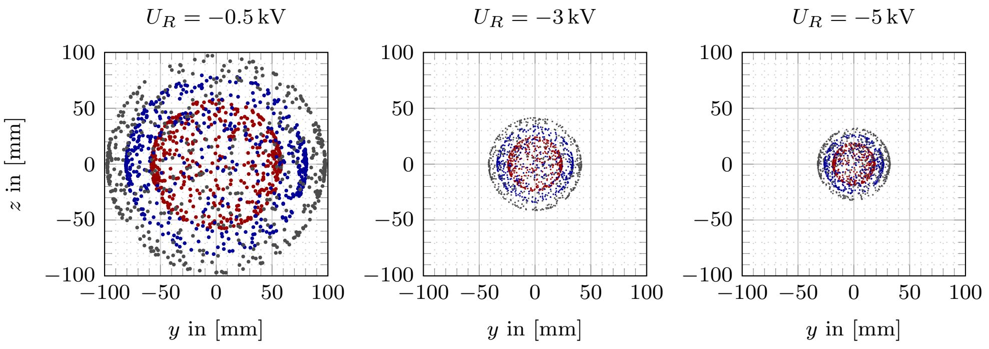


Figure 13: In this figure the simulated detector signal is shown for different choices of the repeller voltage U_R. Again the signal for three different particle energies are shown. The energies displayed are $E_{\text{kin.}} = 0.1\,\text{eV}$ (red), $E_{\text{kin.}} = 0.2\,\text{eV}$ (blue), $E_{\text{kin.}} = 0.3\,\text{eV}$ (grey). As expected the radii of the signals decrease for higher repeller voltages, which is due to a reduced time of fly caused by the higher acceleration toward the detector screen.

4.3 Optimal Voltage Ratio in SMI Mode

To determine the optimal voltage ratio for the SMI mode, two potassium ions where positioned on the same point within the ionization volume. The velocity vectors where set in a way that they point in opposite directions. The chosen orientation is that one velocity vector points up and one points down parallel to the detector surface. The absolute of the velocities where set to the same kinetic energy of $E_{\text{kin.}} = 0.1\,\text{eV}$. Now the same range of voltages was scanned as in the VMI mode. The data is shown in fig. 14. The position of the minimum was estimated to be 0.915 ± 0.002. Since there is no theoretical function which could be fitted to the data the estimation was performed by visual examination of the data. The error was estimated the same way.

4.4 Simulation of many Particles in SMI Mode

Now a bunch of 500 potassium ions where put inside the ionization volume. Now we calculated the parameters which control the size of the ionization volume. These parameters are explained in fig. 3. Using the relations

$$\begin{aligned} z_R &= \frac{\pi w_0^2}{\lambda} \\ w_0 &= \frac{\lambda f}{\pi w_l} \end{aligned} \tag{11}$$

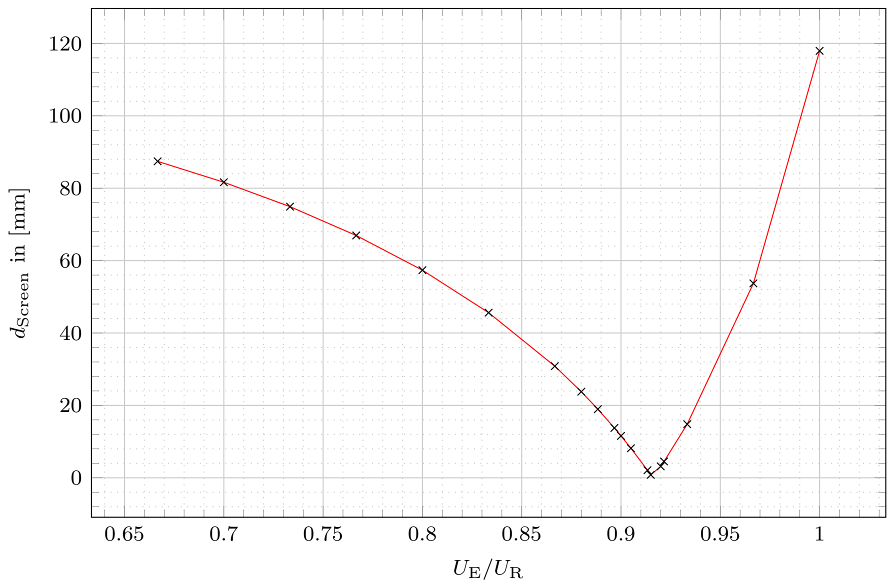


Figure 14: In this figure the distances of the point where the electrons hit the screen is plotted against the voltage ratio. The position of the minimum was estimated to be 0.915 ± 0.002. Since there is no theoretical function which could be fitted to the data the estimation was performed by examination of the data with help of the red line.

and the optimal laser wavelength $\lambda = 404.528\,47\,\mathrm{nm}$, the focal length of the used lens $f = 150\,\mathrm{mm}$ and the diameter of the laser beam before collimation $w_l = 1\,\mathrm{mm}$ we get

$$\begin{aligned} w_0 &= 19.315\,\mu\mathrm{m} \\ z_R &= 2.897\,\mathrm{mm}. \end{aligned} \tag{12}$$

For the positions of the potassium ions a cylindrical distribution was chosen, where the radius of the cylinder was set to $r = \sqrt{2}w_0$ and the height of the cylinder was set to $h = 2z_R$. The velocity vectors where again defined by a normal distributed polar and azimuthal angle and a constant magnitude. The optimal voltage ratio for the SMI mode was used, hence we expected that the different velocities should not affect the signal at all. This expectation, although reasonable, was not met to the desired extend. The signal, which is displayed in fig. 15, does change slightly for the different magnitudes of the velocity vectors.

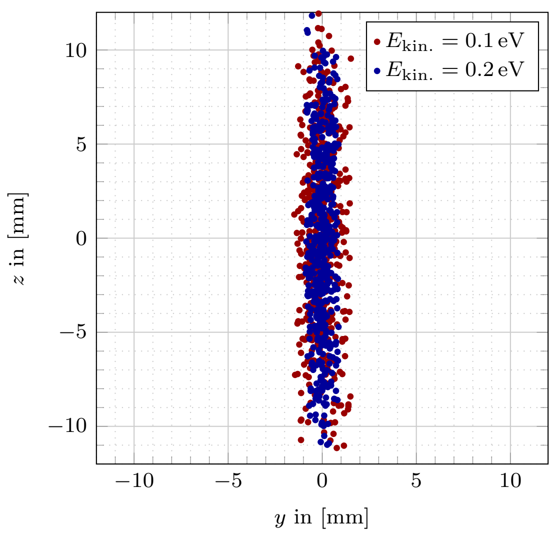


Figure 15: In this figure the signal of the SMI simulation is presented. It is seen that the signals for the different velocity magnitudes differ. Note however, that the difference in both spatial directions is of the same magnitude. This might be an indicator for an error caused by the numerical simulation of the trajectories. Additionally the signal shows the expected rectangle/line shape although the velocities are distributed spherically. Hence a numerical error of the simulation might be the cause for the unexpected broadening of the spatial signal.

Potassium groups for different U_E, E_R (Optional)

Since the signal in fig. 15 was not that satisfactory, caused by the deviation between the signals with different magnitudes of velocity, we varied the ratio of the voltages again. Instead of just checking for the distance on the screen of two ions we now examined the signal for 500 ions, to check whether or not the different energies will be mapped on the same area for a different ratio of voltages. Therefore we kept the repeller voltage constant and varied the extractor voltage by a little such that the ratio is varied. The results of this simulation are shown in fig. 16. It is seen however that the signal with the optimal ratio remains to be the optimal signal.

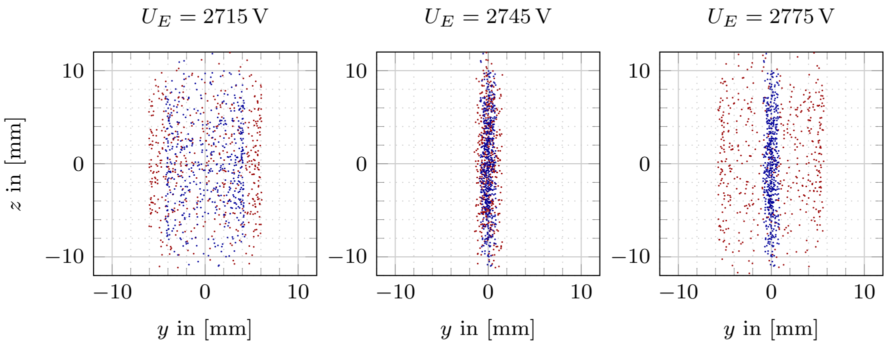


Figure 16: In this figure the simulated detector signal is shown for different choices of the extractor voltage U_E in SMI mode, while keeping the repeller voltage constant, hence the ratio is varied. Again the signal for two different particle energies are shown. The energies displayed are $E_{\text{kin.}} = 0.1\,\mathrm{eV}$ (blue), $E_{\text{kin.}} = 0.2\,\mathrm{eV}$ (red). It is seen that although the two energies are not mapped perfectly to the same area for the optimal ratio, the ratio should be still the optimal one, since when varying the ratio the, signal gets worse.

5 Data Analysis

5.1 Atom Beam Detector

To check the dependence of the atomic flux on the oven temperature, the intensity was measured for different temperatures. This was done by measuring a current in a Faraday-cup created by ionized atoms. The ratio between repeller and extractor voltage in the ion optics with the maximal flux was $U_{\mathrm{E}}/U_{\mathrm{R}} \approx 0.54$, but as the flux itself varied a lot it was difficult to determine a perfect ratio. Thus the atomic flux J can be calculated by

$$J = \frac{I}{eA}, \tag{13}$$

where I is the measured current, e the electric charge and A is the area of the ionising filament.

The result is shown in fig. 17 and it shows the expected exponential behaviour. The uncertainties are $0.2\,^{\circ}\mathrm{C}$ on the temperature and between $1.4 \cdot 10^{7}$ atoms/mm^{2}s and $1.4 \cdot 10^{6}$ atoms/mm^{2}s for the flux. The error on the flux varies, because the change was much faster and the values were fluctuating a lot for higher temperatures.

The later measurements were performed between $150\,^{\circ}\mathrm{C}$ and $153\,^{\circ}\mathrm{C}$ which gives a flux that is high enough for the measurements but the slope of the exponential is not too steep, so the error is smaller than for higher temperatures.

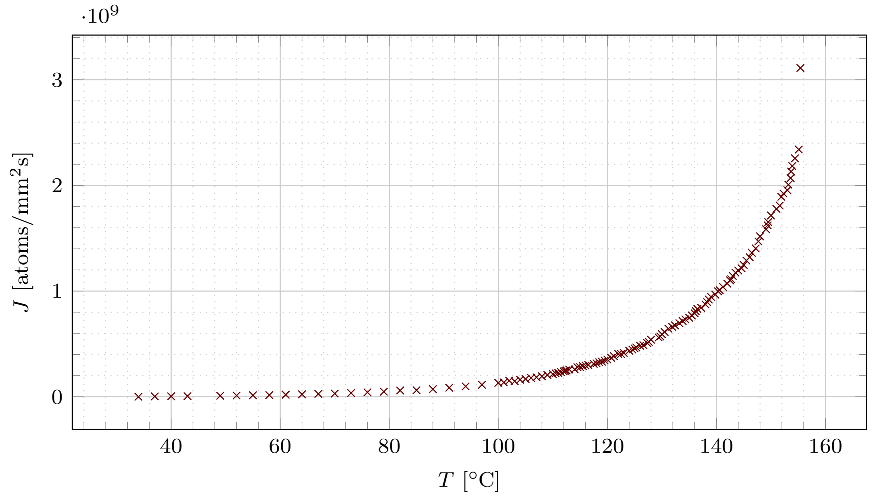


Figure 17: Measurement of the intensity of the atom beam in dependence on the oven temperature, an offset in the measured current has been subtracted. The errors on the temperature and on the flux are too small to be shown. The intensity of the atom beam shows the expected exponential behaviour.

5.2 Spatial Map Imaging with Ions

Image ratio To determine the image ratio for spatial map imaging with ions the position of the signal on the screen was determined for different positions of the focussing lens, the taken data is shown in fig. 30. To determine the position of each signal in pixel, each signal was summed up along the x-axis to achieve a signal with one spatial dimension without having to choose by eye (or arbitrary calculations) which x-position is the most representing. Afterwards a Gauss-fit was applied to find the position and an error for the position, the fits are shown in fig. 18 and the relevant fit parameters can be found in table 3.

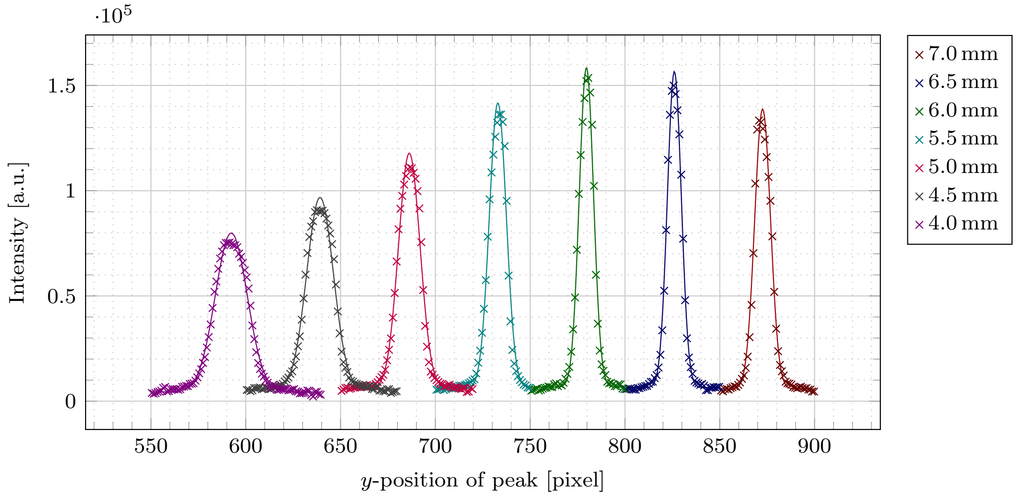


Figure 18: Gauss-fits to determine the position of the peaks. The corresponding position of the lens is indicated by color.

The image ratio $\mathfrak{I}$ can be found as the ratio between the position of the peak on the screen and the position of the lens. Thus a linear fit (fig. 19) of this data has been made and the slope yields the searched ratio:

$$\mathfrak{I} = (93.33 \pm 0.11)\,\text{pixel/mm}. \tag{14}$$

Optimal ratio U_E/U_R for spatial map imaging To find the optimal voltage ratio for the experimental setup the signal has been taken for 9 different ratios and the signal with the minimal width has been searched. In accordance to section 5.2 each signal was summed up over the x-axis. To determine a sensitive parameter for the sharpness of the signal Gauss-fits were applied and the variance of each peak was determined. The calculated parameters are given in table 4. To determine the ratio with the best signal, e.g. the ratio with the minimal variance, the variance was plotted against the voltage ratio fig. 20. As the button to change the ratio was not precise at all there was not data taken in smaller steps around the minimum, furthermore the data is not really symmetric, so it does not seem to make sense to fit a quadratic function.

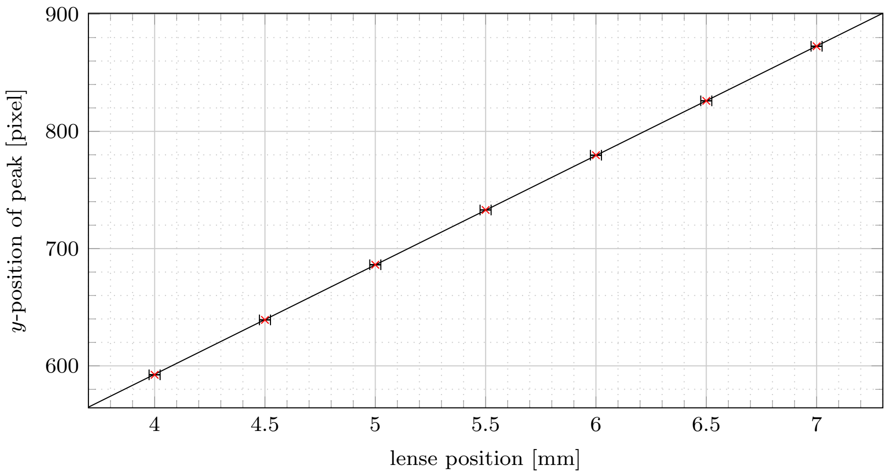


Figure 19: Position of the peaks as determined by the Gauss-fit plotted against the position of the lens. A linear fit of the form $f(x) = mx + b$ has been calculated and gives the parameters $m = (93.33 \pm 0.11)$ pixel/mm and $b = (219.4 \pm 0.7)$ pixel.

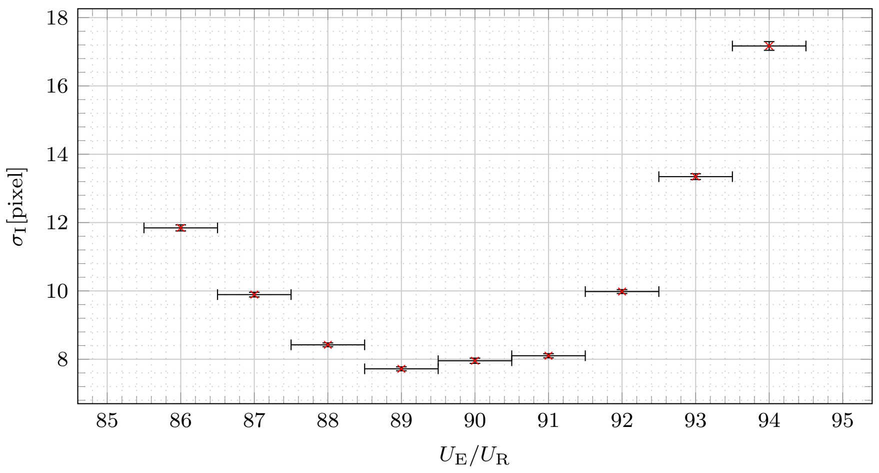


Figure 20: Variance of the peaks for different voltage ratios.

Therefore the minimum has been estimated by eye and a rather high error, taking the experimental setup and the estimation into account, has been applied. The optimal voltage ratio for SMI-spectroscopy with this setup is

$$\left(\frac{U_{\mathrm{E}}}{U_{\mathrm{R}}}\right)_{\text{SMI, exp.}} = 90.0 \pm 0.5. \tag{15}$$

The ratio determined in the simulations was $(U_{\mathrm{E}}/U_{\mathrm{R}})_{\text{SMI, sim}} = 91.5 \pm 0.2$, the simulated ratio and the experimentally determined one are of same magnitude. Taking a look at the curve one sees, that even though the minimum is comparable, the lines differ and the simulated curve looks even less as a quadratic function. This difference could be due to the differences between the real and the simulated setup, both geometrical differences, as well as just slightly different voltages. Another possible error is the unknown error of the simulation that could lead to different results.

Dimension of the Focal Area With the image ratio, the beam waist and the Rayleigh length can be computed by

$$\omega_0 = \frac{2\sigma_{\text{SMI, exp}}}{\Im}, \quad z_\text{R} = \frac{\pi\omega_0^2}{\lambda} \quad \text{and} \quad \omega_l = \frac{\lambda f}{\pi\omega_0}, \tag{16}$$

so the relevant factor from the measurements is the variance. In the previous parts the signal was summed over one axis so the error of choosing the wrong cut is avoided. This leads to rather high values for the variance which is not a problem when searching for the minimum. For the calculation of the focal area the variance for a not-summed signal is needed. Therefore the signal for the determined optimal ratio was taken and the x-position with the highest signal, $x = 705$ pixel was chosen to cut to obtain the intensities in dependence on y. For this cut the position of the peak and the variance was determined with a Gauss-fit, the parameters are shown in table 4. The variance determined and used in the calculation of the focal area is $\sigma_{\text{SMI, exp.}} = (3.31 \pm 0.04)$ pixel. Thus the characteristics of the focal area are

$$\omega_0 = (70.9 \pm 0.4)\,\mu\text{m}, \tag{17}$$

$$z_\text{R} = (39.0 \pm 0.4)\,\text{mm} \quad \text{and} \tag{18}$$

$$\omega_l = (272.5 \pm 0.6)\,\mu\text{m}, \tag{19}$$

with $\lambda = (404.5 \pm 0.5)$ nm as the wavelength of the laser. The errors were calculated by Gaussian error computation. The experimentally determined values are not compatible with the values we determined in the simulation.

5.3 Velocity Map Imaging with Electrons

In this section the data measured in VMI mode is analysed. We start by experimentally determining the optimal voltage ratio for the ion optics.

Optimal ratio U_E/U_R for velocity map imaging To determine the optimal voltage ratio we performed measurements at various voltage ratios. To the obtained data we fitted a Gauss model to the peak corresponding to the lowest energy. We chose this peak due to its behavior. While the peak for the highest energy did not vary visibly, the peak corresponding to the second highest energy is very low in intensity and vanished fast for small variations around the optimal ratio. So the optimal candidate for this analysis is the peak corresponding to the smallest kinetic energy. The raw data and the fits are shown in figs. 31 to 33. The FWHM is plotted against the voltage ratio in fig. 21, so for the optimal voltage ratio this plot should show a minimum. The visual estimation of the optimal ratio yields $U_E/U_R = (71.5 \pm 0.5)\,\%$. Since during the experiment the visual signal on the screen seemed to be a bit clearer for a ratio of $U_E/U_R = 71\,\%$ we took a lot of data for this ratio. Since we fortunately performed some measurements at $U_E/U_R = 71.5\,\%$, we decided to analyse both datasets. We begin with the analysis of the dataset measured with a ratio of $U_E/U_R = 71\,\%$.

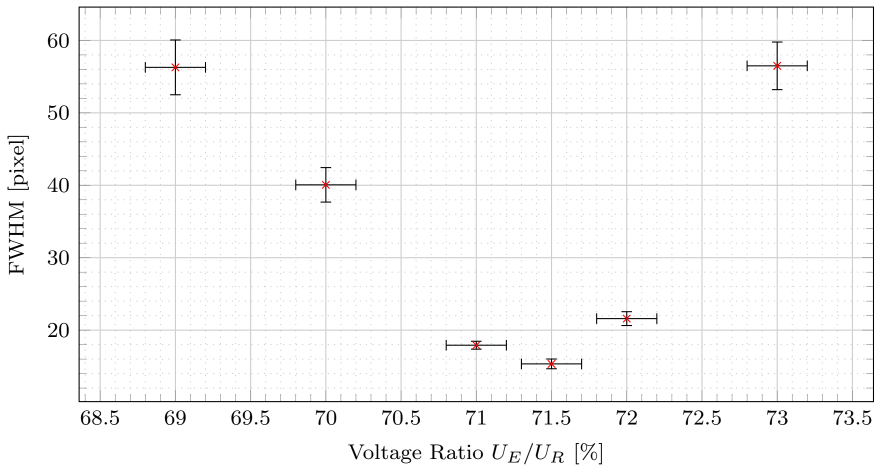


Figure 21: The FWHM is plotted against the chosen ratio of the measurement. This is used to experimentally determine the optimal voltage ratio.

Calibration of the Energy Axis for $U_E/U_R = 0.71$ To calibrate the energy axis it is used that the peak showing the highest intensity should correspond to the REMPI process shown in fig. 5. Hence the peak position of the REMPI peak is estimated using a Gauss model, which is fitted to the data using a least-squares method. The data and the fitted Gauss model is shown in fig. 22. The used data is actually averaged over seven measurement, each of the measurements is shown in figs. 34 to 37. The kinetic energy corresponding to the REMPI process is calculated using

$$E_{\text{kin.}}^{\text{REMPI}} = E_{\text{Laser}} + E_{\text{Level}}^{\text{REMPI}} - E_{\text{Ion.}} = 1.789\,15\,\text{eV}, \tag{20}$$

where $E_{\text{Ion.}} = 4.340\,663\,54\,\text{eV}$ is the ionization energy given (see e. g. [8]). The axis is then calibrated by assigning an energy to every radius via

$$E_{\text{kin.}} = E_{\text{kin.}}^{\text{REMPI}} \frac{r_{\text{pixel}}^2}{\mu_{\text{REMPI}}^2}, \tag{21}$$

as shown in the literature (see e. g. [5]). The parameter μ_{REMPI} is the mean of the Gauss model fitted to the REMPI peak. The fit parameter was estimated to be

$$\mu_{\text{REMPI}} = (455.67 \pm 0.19)\,\text{pixel}, \tag{22}$$

which results in a relative error of 0.4 %. We resume determining the energy levels of potassium using the data measured with a voltage ratio of $U_E/U_R = 0.71$.

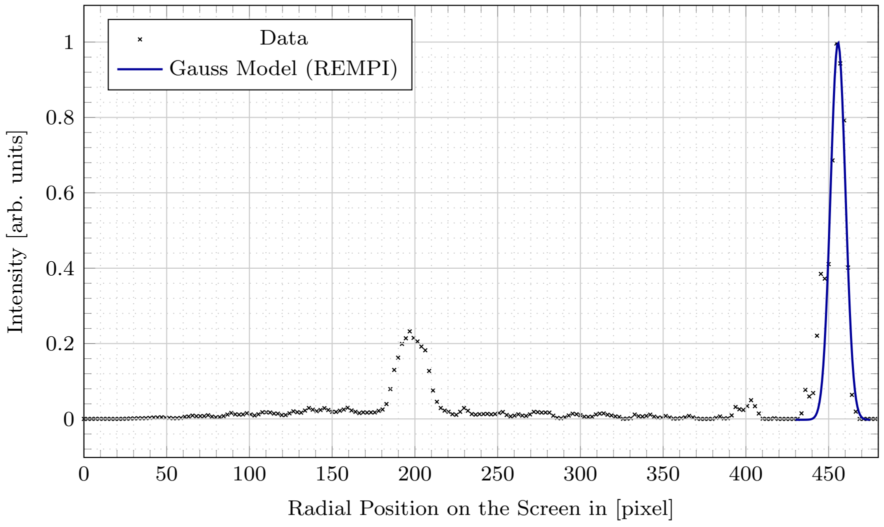


Figure 22: In this figure the radial signal of the spectrometer is shown. The peak with the highest Intensity does correspond to the REMPI process shown in fig. 5. The energy corresponding to the REMPI process is known and therefore used to calibrate the energy axis (x-axis). The data shown is actually averaged over seven measurements using the same voltage ratio of $U_E/U_R = 0.71$.

Energy Levels of Potassium for $U_E/U_R = 0.71$ Using the now calibrated energy axis the level energies the of the remaining peaks where determined by additional Gauss fits shown in fig. 23. These fits result in the following kinetic energies

$$E_{\text{kin.},2} = (1.380 \pm 0.005)\,\text{eV}$$
$$E_{\text{kin.},3} = (0.339 \pm 0.001)\,\text{eV}.$$

These kinetic energies result in level energies of

$$E_{\text{Level},2} = (2.656 \pm 0.006)\,\text{eV}$$
$$E_{\text{Level},3} = (1.615 \pm 0.004)\,\text{eV}.$$

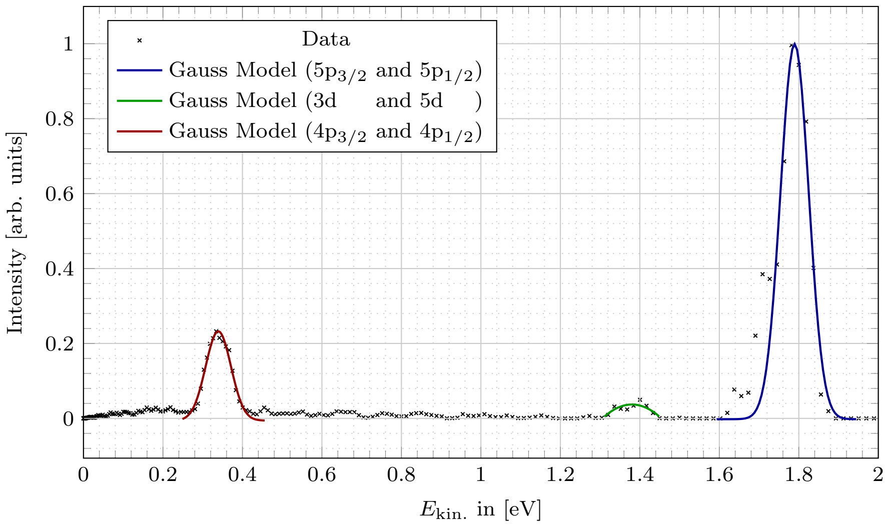


Figure 23: The radial signal of the spectrometer with calibrated x-axis. There are three gauss models which are fitted to the data using a least squares approach. The legend shows, to which ionization channel the peaks correspond. The data shown is actually averaged over seven measurements using the same voltage ratio of $U_E/U_R = 0.71$.

The errors on the energy level where calculated by Gaussian error propagation using the error on the fit parameter and the error of the laser energy which was computed to be $\Delta E_{\text{Laser}} = 0.004\,\text{eV}$, presupposing an error on the wavelength of $\Delta\lambda = 0.5\,\text{nm}$.

Calibration of the Energy Axis for $U_E/U_R = 0.715$ The calibration for the second ratio is performed in direct analogy to the first one. The mean of the REMPI peak was again determined by using a Gauss model, which is fitted to the data using a least-squares method. The fit and the data used for calibration are shown in fig. 45. The computation yields

$$\mu_{\text{REMPI}} = (449.89 \pm 0.06)\,\text{pixel}, \tag{23}$$

which results in a relative error of 0.01 %. The relative error on the peak position is significantly smaller then the one for the other voltage ratio. Hence, the error on the calibration should propagate less and therefore better results should be possible. The used data is again the average over all measured angular intensity distributions shown in figs. 38 to 40.

Energy Levels of Potassium for $U_E/U_R = 0.715$ Again we used the data with the now calibrated energy axis to fit the resuming peaks. The data and the fits are shown in fig. 24. The fits resulted in the kinetic energies

$$E_{\text{kin.},2} = (1.380 \pm 0.002)\,\text{eV}$$
$$E_{\text{kin.},3} = (0.340 \pm 0.002)\,\text{eV}.$$

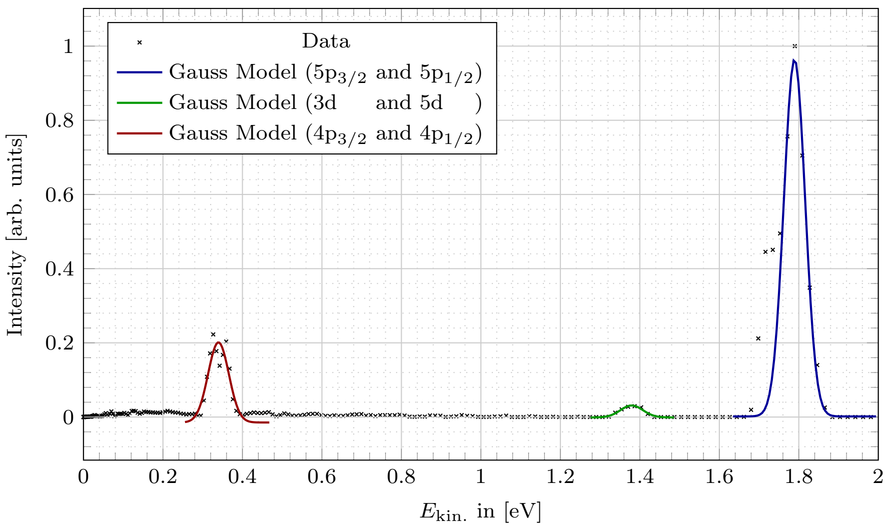


Figure 24: The radial signal of the spectrometer with calibrated x-axis. There are three gauss models which are fitted to the data using a least squares approach. The legend shows, to which ionization channel the peaks correspond. The data shown is actually averaged over five measurements using the same voltage ratio of $U_E/U_R = 0.715$.

These kinetic energies result in level energies of

$$E_{\text{Level},2} = (2.655 \pm 0.004)\,\text{eV}$$
$$E_{\text{Level},3} = (1.615 \pm 0.005)\,\text{eV}.$$

Energy Resolution of the Spectrometer Finally the energy resolution δE was estimated using

$$\delta E = \frac{2\sqrt{2\ln 2}\,\sigma}{\mu}, \tag{24}$$

where σ, μ are the fit parameters of the corresponding peaks. For the energy resolution we got

$$\begin{aligned} \delta E_1 &= (3.52 \pm 0.03)\,\% \\ \delta E_2 &= (4.8 \pm 0.2)\,\% \\ \delta E_3 &= (18 \pm 1)\,\% \end{aligned} \tag{25}$$

The error was estimated by gaussian error propagation of the errors on the fit parameters determined by the least-squares routine.

Changing the Repeller Voltage at Constant Voltage Ratio Finally we also performed a measurement to check how the radius of the signal behaves, when the repeller voltage is raised, while the voltage ratio of the ion optics remained the same. As already seen in the simulations in fig. 11 it is assumed, that the radius of the signal decreases for higher repeller voltages. This assumption is perfectly sensible, since the electron gets accelerated more towards the screen, which results in a smaller time of flight. If the time of flight is reduced, the electron has less time to propagate parallel to the detector surface, which results in a smaller displacement within the plane parallel to the detector plane. Experimentally we set the repeller voltage to different values and measured the radial intensity distribution on the screen. To quantise the change of the radius we fitted a Gauss model to the most inner peak of the radial intensity distribution for every repeller voltage used. The measured data for the different repeller voltages, including the Gauss fit, are shown in figs. 41 to 43. The resulting dependence of the radius of the signal on the repeller voltage is shown in fig. 25.

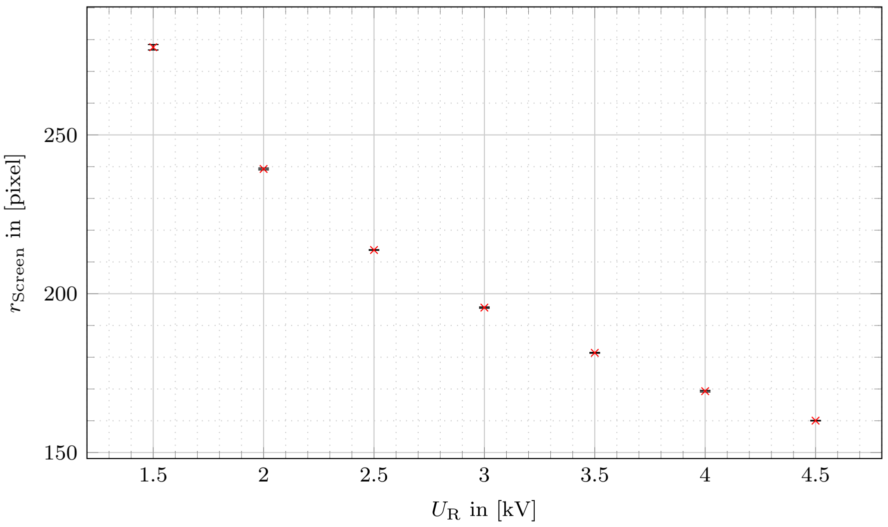


Figure 25: In this figure the dependence of the radius of the signal in VMI mode on the repeller voltage is shown.

5.4 Anisotropy Parameter

The taken data also gives chance to estimate the anisotropy parameters β_2 and β_4 for the different transitions in potassium. To determine the anisotropy parameters the angular data from the program `pBasex` is taken and weighed by the normalised radial signal. This was done for each dataset with the optimal voltage ratio $U_\mathrm{E}/U_\mathrm{R} = 71.5$ and the result is shown in fig. 26. The anisotropy parameter for each transition can now be determined by averaging over all values that lie in the FWHM of the corresponding peak in fig. 45. In fig. 26 this FWHM is indicated in grey. The calcu-

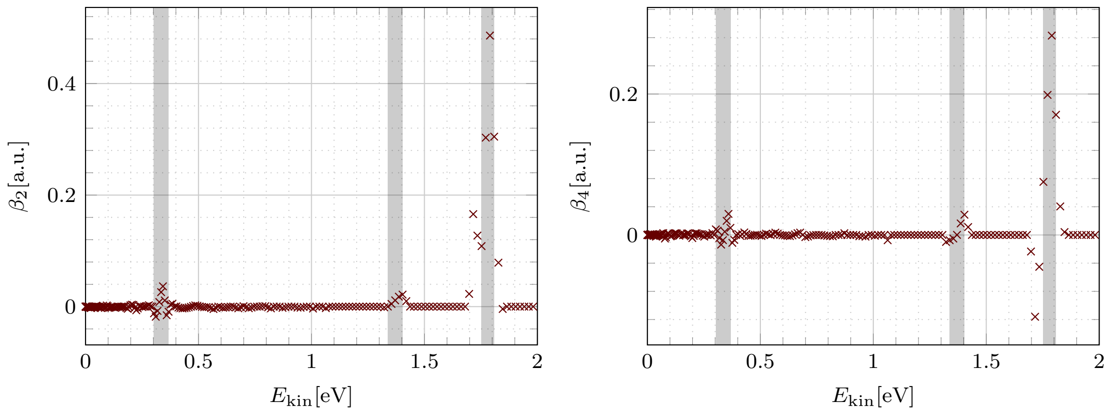


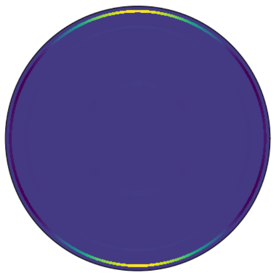
Figure 26: Estimation of the anisotropy parameters β_2 on the left and β_4 on the right. The grey areas represent the FWHM taken from the peaks shown in fig. 45. The values in the FWHM were averaged to determine the anisotropy parameters.

lated values are shown in table 1 and can be compared to the values that were calculated in [8]. The errors that are given for our values correspond to the standard deviation of the averaged values.

transition	β_2	β_2 (ref.)	β_4	β_4 (ref.)
4_p	0.002 ± 0.028	0.17 ± 0.03	0.006 ± 0.015	0
3_d	0.0097 ± 0.0092	0.86 ± 0.09	0.002 ± 0.012	0
5_p	0.4 ± 0.2	1.07 ± 0.04	0.3 ± 0.2	0.52 ± 0.08

Table 1: Calculated anisotropy parameters for the different transitions in potassium. The benchmark values are taken from [8].

Even though the estimated parameters do not confirm the data found in the reference the parameters were used to calculate the image one would see for these values. This was done by calculating eq. (10) with the corresponding anisotropy parameters for each transition and showing them at the position of the peak with a with of the FWHM in the plot. The calculated images for our values and the reference values, as well as an Abel-inversed image from our data are shown in fig. 27 and fig. 28. The smallest circle corresponds to the $4_{\mathrm{p}_{3/2}}$- and the $4_{\mathrm{p}_{1/2}}$-state, the middle circle to 3_d and 5_d and the outer circle to the $5_{\mathrm{p}_{3/2}}$- and the $5_{\mathrm{p}_{1/2}}$-state. The distribution shows the expected higher intensities for the upper and lower parts of the circle, but as the estimated anisotropy parameters are not close to the ones found in the reference, the middle and innermost circle can hardly be seen.


(a) Calculated image with measured anisotropy parameters.

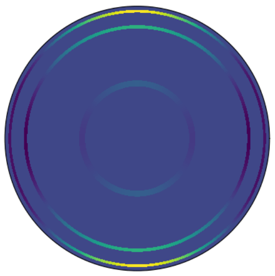

(b) Calculated image with anisotropy parameters taken from [8].

Figure 27: Calculated signal of Abel inversed images. Brighter green to yellow corresponds to higher intensities and purple indicates lower intensities.

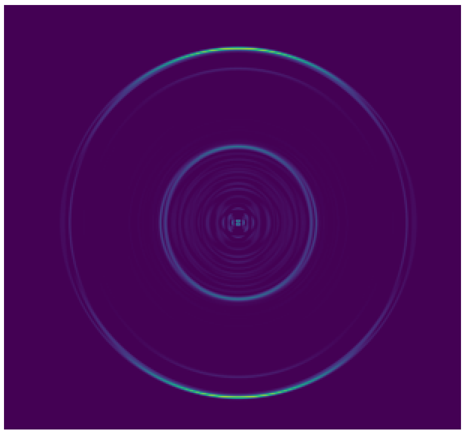


Figure 28: Abel inversed image for the voltage ratio $U_{\mathrm{E}}/U_{\mathrm{R}} = 71.5\,\%$. Brighter green to yellow corresponds to higher intensities and purple indicates lower intensities.

6 Summary and Discussion

Atom Beam Detector To check the dependence of the atomic flux on the oven temperature a LT-detector was used and the flux was measured for decreasing temperatures. The relation shows the expected exponential behaviour and temperatures at around 150 °C will lead to a sufficient flux.

Spatial Map Imaging The image ratio for spatial map imaging with ions was determined by analysing the dependence of the signal on the screen on the position of the focusing lens. A linear fit over peak positions plotted against the lens position gave

$$\mathfrak{I} = (93.33 \pm 0.11)\,\text{pixel/mm}.$$

Furthermore the optimal voltage ratio for SMI-spectroscopy was estimated by searching for the signal with the smallest variance. The determined ratio is

$$\left(\frac{U_\text{E}}{U_\text{R}}\right)_\text{SMI, exp.} = 90.0 \pm 0.5.$$

This optimal voltage ratio was also determined with help of simulated data what gave

$$\left(\frac{U_\text{E}}{U_\text{R}}\right)_\text{SMI, sim} = 92.5 \pm 0.2.$$

The experimental and the simulated ratio are of the same magnitude, but not compatible. This difference is most probably caused by differences between the geometry used for the simulation and the actual geometry of the setup. We suspect that the geometry used for the simulations had a longer tube in which the ions/electrons are propagating.

Additionally the dimensions of the focal area were determined with help of the experimental data. This yields

$$\begin{aligned} \omega_0 &= (70.9 \pm 0.4)\,\mu\text{m}, \\ z_\text{R} &= (39.0 \pm 0.4)\,\text{mm} \quad \text{and} \\ \omega_l &= (272.5 \pm 0.6)\,\mu\text{m}. \end{aligned}$$

Again these values are not compatible with the ones calculated with help of the simulated data:

$$\begin{aligned} w_0 &= 19.315\,\mu\text{m} \\ z_R &= 2.897\,\text{mm}. \end{aligned}$$

Velocity Map Imaging First of all the optimal voltage ratio for VMI-spectroscopy was determined, both with experimental and with simulated data. The determine values are

$$\begin{aligned} \left(\frac{U_\text{E}}{U_\text{R}}\right)_\text{VMI, exp.} &= 71.5 \pm 0.5 \\ \left(\frac{U_\text{E}}{U_\text{R}}\right)_\text{VMI, sim.} &= 86.2 \pm 0.2. \end{aligned}$$

Again the most probable reason for the difference in the values is the possible difference in the experimental and the simulated setup.

In simulations the dependence of the radius of the signal on the kinetic energy was checked and the data confirmed the assumption, that greater kinetic energy leads to a wider signal. This makes sense, taking the direction of the set velocity vectors into account.

In simulations, as well as in the experiment the dependence of the signal to different repeller voltages, but with a constant voltage ratio was checked and for increasing absolute value of the repeller voltage the signal will get smaller. This behaviour was expected, as a larger repeller voltage will shorten the time of flight so the velocity orthogonal to the direction of motion will be less dominant.

Energy Levels of Potassium The signal of VMI with electrons for the optimal experimental voltage ratio was used to determine the energy levels of potassium. The signal was Abel-inversed with the program `pBasex` and the radial signal was used to determine the position of the tree peaks. The peak of the REMPI process was used to calibrate the energy axis and the other two energies were calculated table 2. In addition to that the energy resolution of the spectrometer was determined and is also shown in table 2.

	measured [eV]	literature [eV]	resolution
$5p_{3/2}$	–	3.065	$(3.52 \pm 0.03)\,\%$
3d	2.655 ± 0.004	2.670	$(4.8 \pm 0.2)\,\%$
4p	1.615 ± 0.005	1.617	$(18 \pm 1)\,\%$

Table 2: Measured energy levels of potassium and the energy resolution for the voltage ratio $U_E/U_R = 0.715$. Literature values taken from [8]. The literature value for 4p corresponds to $4p_{3/2}$, the difference between $4p_{3/2}$ and $4p_{1/2}$ as well as for $5p_{3/2}$ and $5p_{1/2}$ is too small for the experimental resolution we had, so there is just one value for both levels.

Anisotropy Parameters Last the anisotropy parameters were estimated with help of the angular data of the Abel-inversed signal. The estimated values are shown in table 1 and none of the calculated values lies close to the values found in [8]. The main problem in this part is, that the analysis was not performed on the raw Abel-inversed data, but on a already analysed output. The program `pBasex` already calculates the anisotropy parameter for the different radii and returns these rather than the angular distribution. This leads to difficulties in error estimation as the program does not give any errors for its fits and the error we see in the end is hardly traceable. This problem could be solved by taking the two dimensional data the program returns and calculating the angular distribution out of this data, but as the data is in Cartesian coordinates one would need to rebin to polar coordinates which also leads to high errors. Nevertheless it can be assumed that calculation of the anisotropy parameters might be possible with the experimental data we achieved, as the distribution in the signal looks as expected.

A Procedure

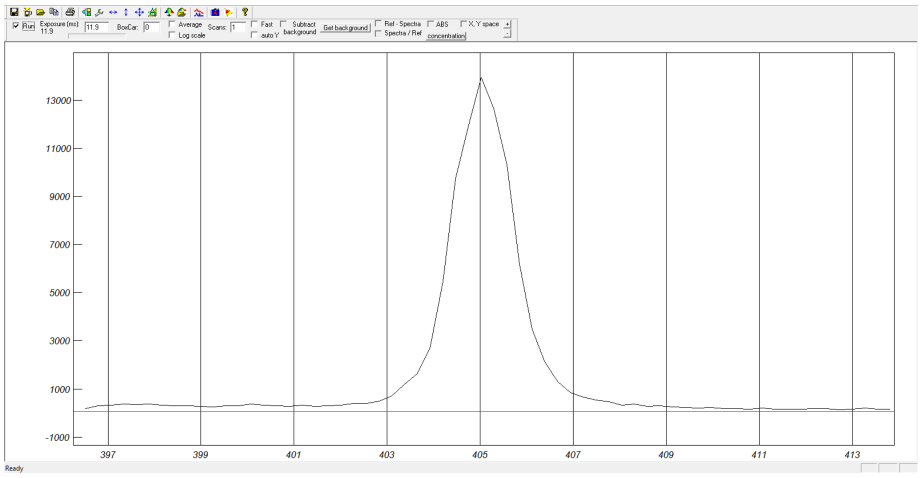

Exposure (ms): 11.9
BoxCar: 0
Scans: 1

Figure 29: Output of the spectrometer used to set the right wavelength.

B Analysis

B.1 Spatial Map Imaging with Ions

lens position [mm]	peak position [pixel]	peak error [pixel]
7.0	872.59	0.09
6.5	826.04	0.05
6.0	779.67	0.07
5.5	732.96	0.11
5.0	686.24	0.10
4.5	639.18	0.10
4.0	592.36	0.11

Table 3: Position of the peak as calculated with a Gauss fit for different positions of the lens.

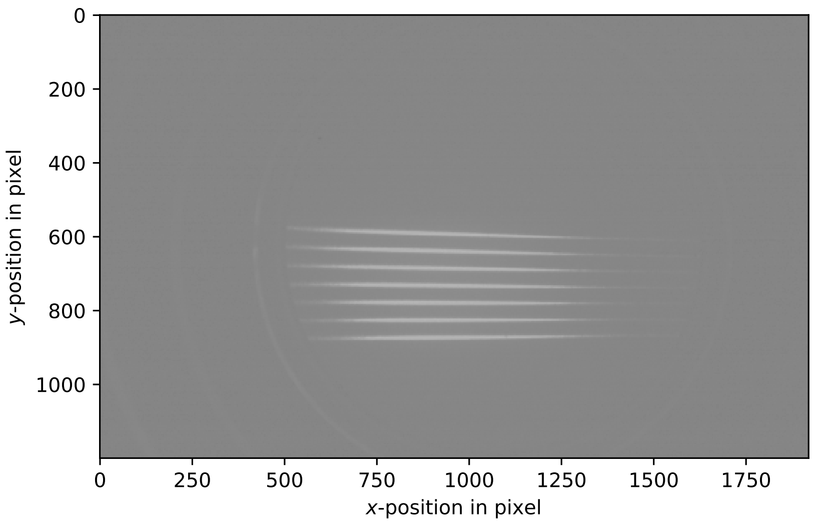


Figure 30: Signal for ions in SMI-mode for different positions of the lens. The background was subtracted and the different signals are shown summed up to show the dependence on the position of the lens. The highest line at $y \approx 550$ pixel corresponds to a position of the lens of 4 mm and the lowest line at $y \approx 850$ pixel corresponds to 7 mm. In between the lens was moved in equidistant steps of 0.5 mm

U_E/U_R	σ [pixel]	s_σ [pixel]
86.0	11.85	0.09
87.0	9.89	0.07
88.0	8.42	0.06
89.0	7.72	0.06
90.0	7.96	0.08
91.0	8.10	0.06
92.0	9.98	0.06
93.0	13.34	0.08
94.0	17.17	0.13
without sum		
90.0	3.31	0.04

Table 4: Variance of the Intensity as determined by the Gauss-fits for different voltage ratios. A larger variance corresponds to a blurred signal, hence a ratio with minimal variance is searched. For calculating the focal area the variance of a signal without summation over one spatial dimension is needed.

B.2 Velocity Map Imaging with Electrons

B.2.1 Optimal Voltage Ratio

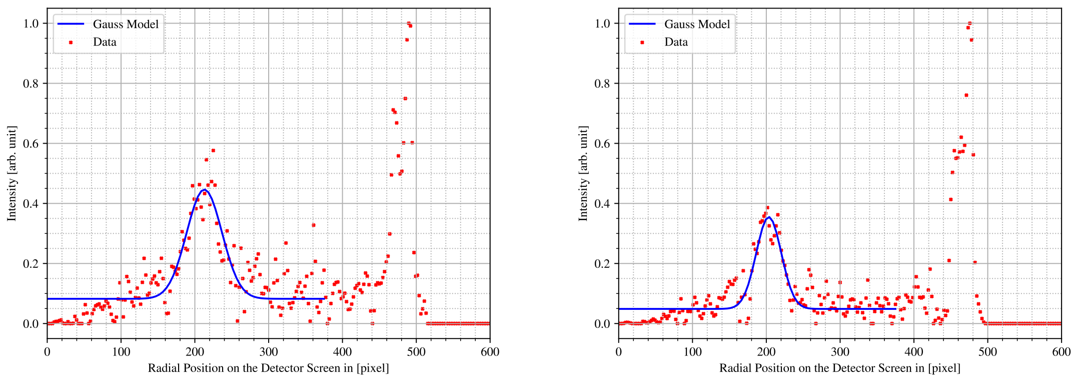


Figure 31: In this figure the data for the determination of the optimal voltage ratio in VMI mode is shown. Shown ratios: 69 %(left) 70 %(right)

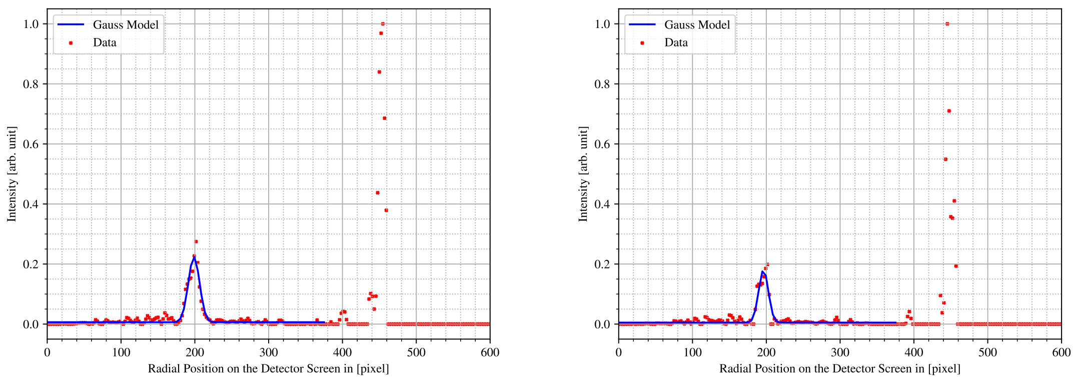


Figure 32: In this figure the data for the determination of the optimal voltage ratio in VMI mode is shown. Shown ratios: 71 %(left) 71.5 %(right)

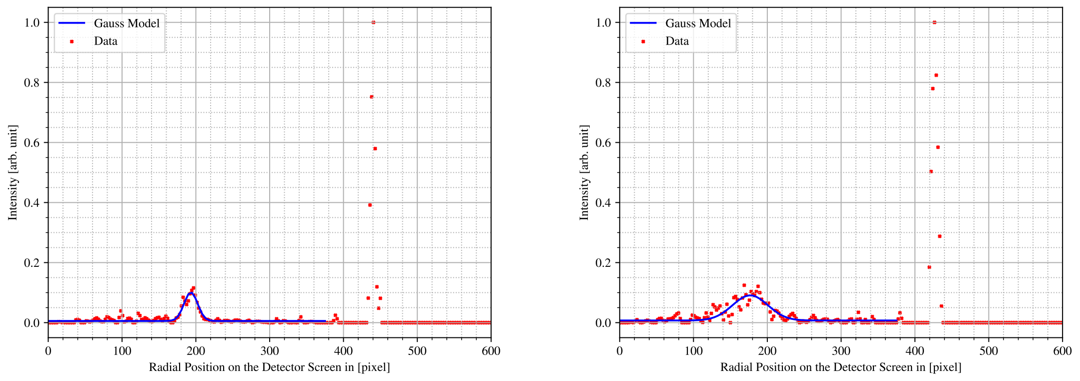


Figure 33: In this figure the data for the determination of the optimal voltage ratio in VMI mode is shown. Shown ratios: 72 % (left) 73 % (right)

B.2.2 Energy Analysis

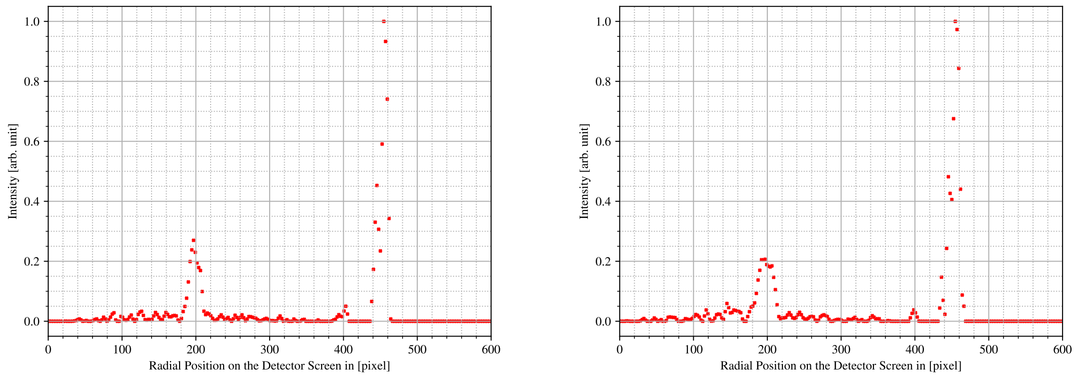


Figure 34: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.71$

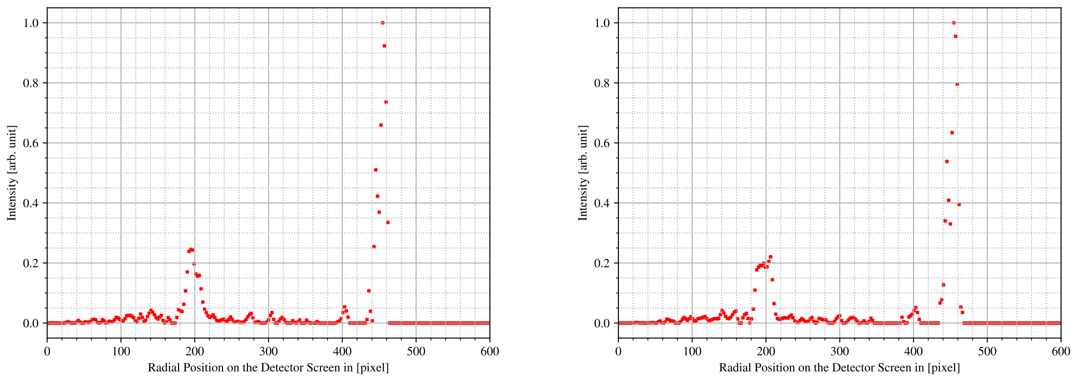


Figure 35: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.71$

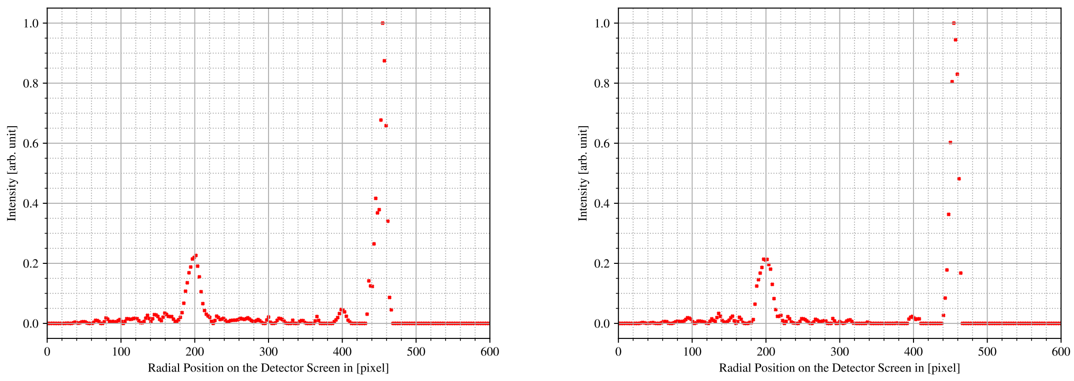


Figure 36: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.71$

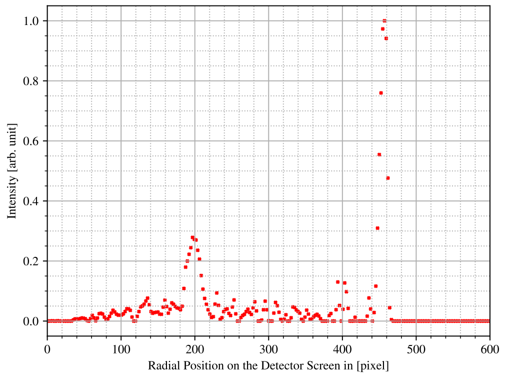
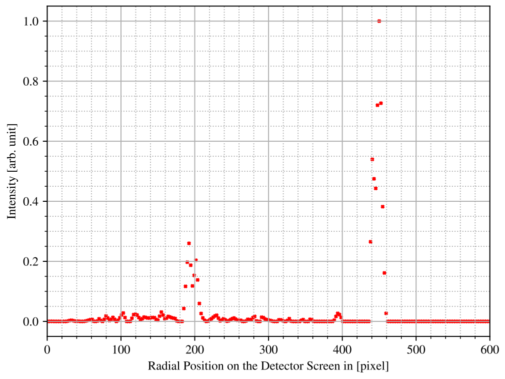


Figure 37: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.71$



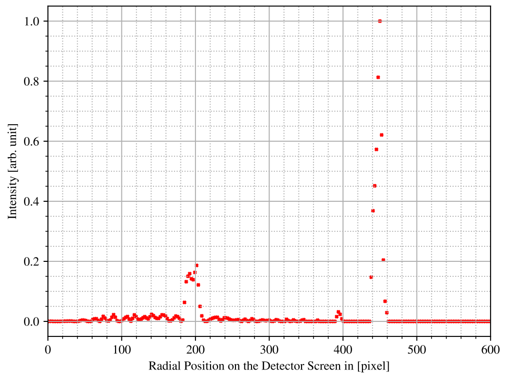


Figure 38: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.715$

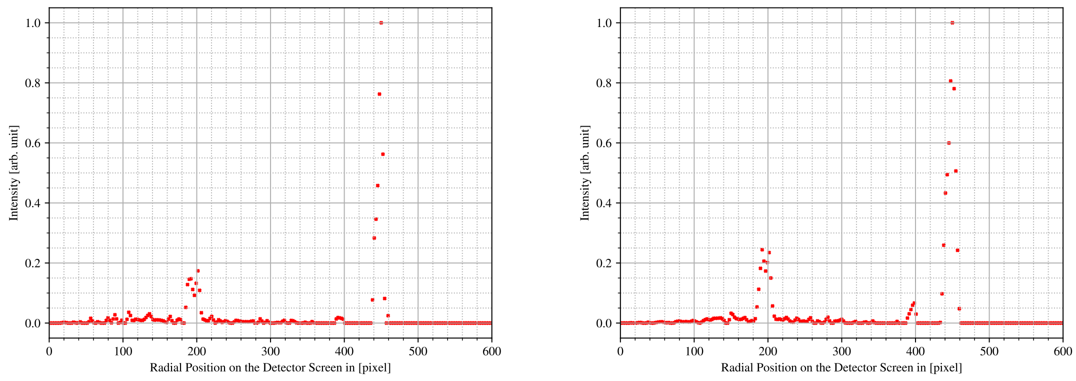


Figure 39: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.715$

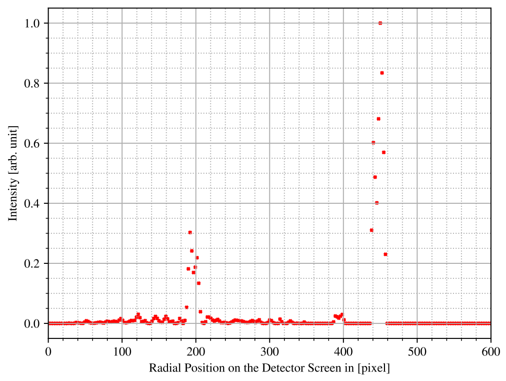


Figure 40: In this figure the raw data for radial intensity distribution in VMI mode is shown. Voltage ratio: $U_E/U_R = 0.715$

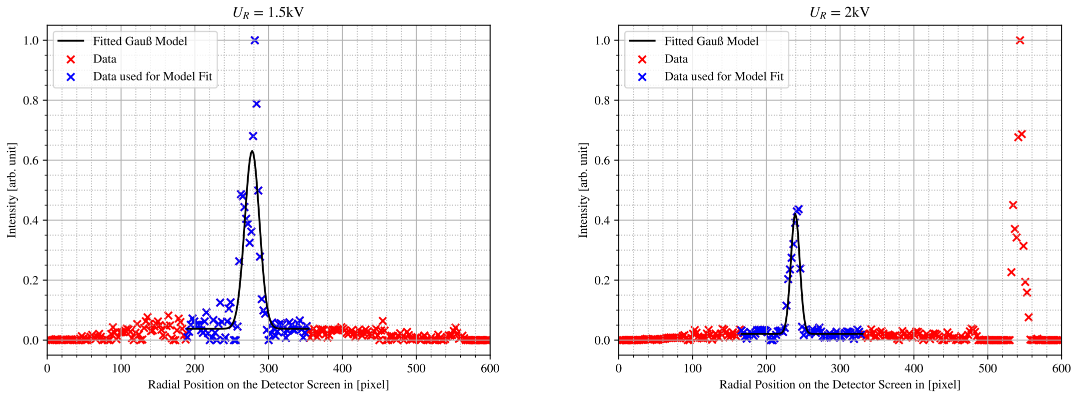


Figure 41: In this figure the data and fits are displayed, which are used to check the dependence of the signals radius in VMI mode on the repeller voltage.

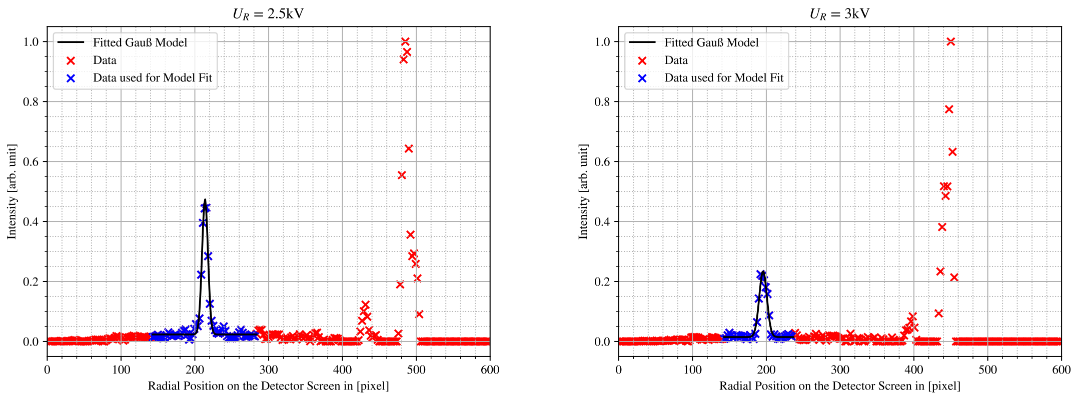
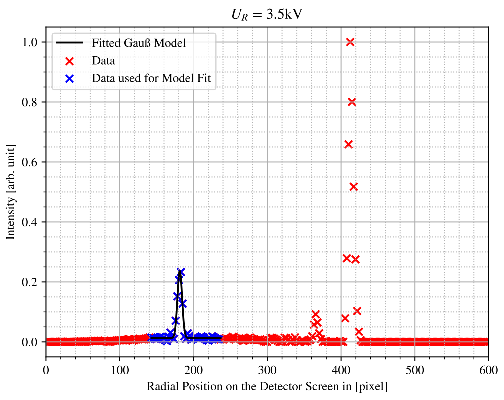


Figure 42: In this figure the data and fits are displayed, which are used to check the dependence of the signals radius in VMI mode on the repeller voltage.



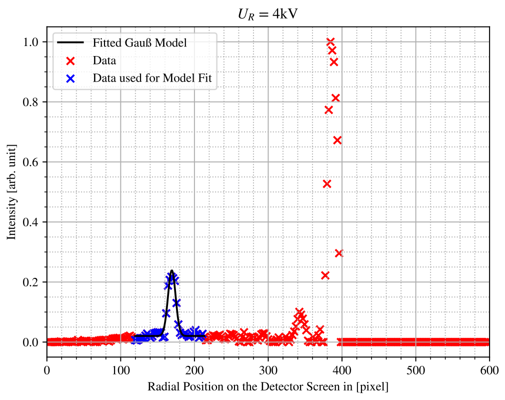


Figure 43: In this figure the data and fits are displayed, which are used to check the dependence of the signals radius in VMI mode on the repeller voltage.

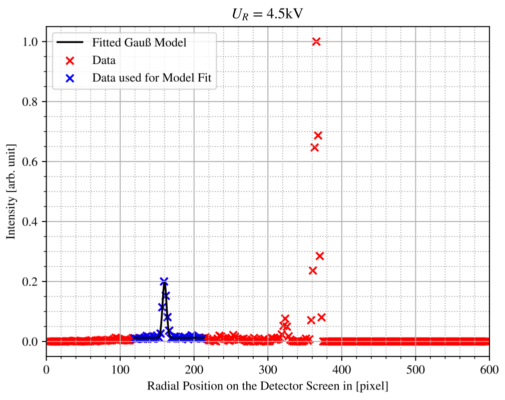


Figure 44: In this figure the data and fits are displayed, which are used to check the dependence of the signals radius in VMI mode on the repeller voltage.

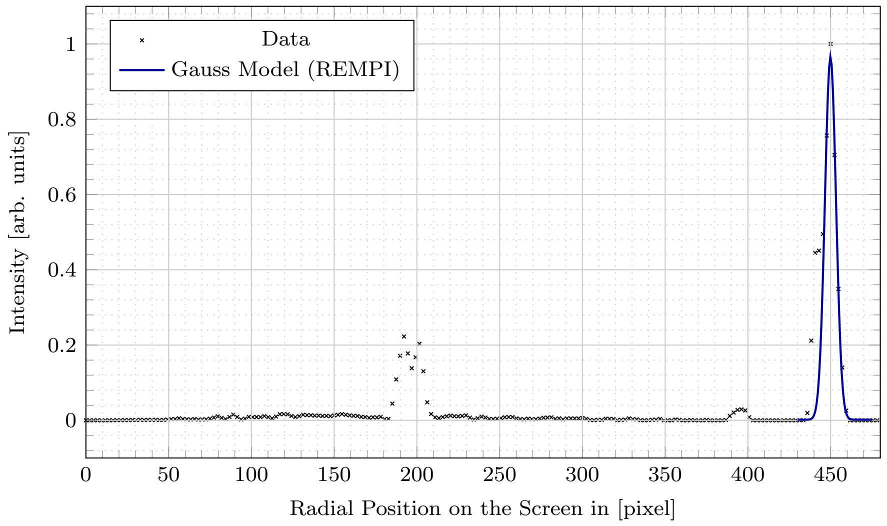


Figure 45: In this figure the radial signal of the spectrometer is shown. The peak with the highest Intensity does correspond to the REMPI process shown in fig. 5. The energy corresponding to the REMPI process is known and therefore used to calibrate the energy axis (x-axis). The data shown is actually averaged over five measurements using the same voltage ratio of $U_E/U_R = 0.715$.

C Code used in the Analysis

C.1 Simulation

VMI-Mode

```python
"""
This module contains data from SimIon simulations. The data is used to find
the optimal ratio of U_E/U_R in VMI mode.
"""

import matplotlib.pyplot as plt
import numpy as np

# data to find minimum of screen distance
z_screen1 = [8.90219e1, 3.73373e1, 8.65263, -9.79813, -2.26675e1, -3.21190e1,
             -3.93121e1, -4.49297e1, -4.94025e1, -5.30162e1, -5.59683e1]  # mm
z_screen2 = [6.79536e1, 2.58166e1, 2.49195, -1.24951e1, -2.29488e1, -3.06334e1,
             -3.64926e1, -4.10810e1, -4.47479e1, -4.77245e1, -5.01705e1]  # mm
u_e = np.array([3000, 2900, 2800, 2700, 2600, 2500, 2400, 2300, 2200, 2100,
                2000])  # V

# closer measurements
z_s1 = [-1.34288e1, -1.67619e1, -1.98317e1, -2.26675e1, -2.52941e1, -2.77329e1]
z_s2 = [-1.55438e1, -1.81510e1, -2.06448e1, -2.29488e1, -2.50834e1, -2.70659e1]
u_e_close = np.array([2675, 2650, 2625, 2600, 2575, 2550])

# even closer measurements
z_1_close = [-2.15596e1, -2.37420e1, -2.47844e1, -2.44750e1]
z_2_close = [-2.20486e1, -2.38219e1, -2.46691e1, -2.44176e1]
u_e_closer = np.array([2610, 2590, 2580, 2583])

# simulation parameters

# distance in source chamber
d_ini = 1  # mm
# repellor voltage
u_r = 3000  # V

# distance on detector screen
d_screen = [np.abs(z1 - z2) for z1, z2 in zip(z_screen1, z_screen2)]  # mm
d_screen_close = [np.abs(z1 - z2) for z1, z2 in zip(z_s1, z_s2)]  # mm
d_screen_closer = [np.abs(z1 - z2) for z1, z2 in zip(z_1_close, z_2_close)]  # mm

# combine all data
d_data = d_screen + d_screen_close + d_screen_closer
u_e_data = np.concatenate((u_e, u_e_close, u_e_closer), axis=None)

# plot data
plt.scatter(u_e_data/u_r, d_data, marker="x", color="red")
plt.xlabel(r"$\frac{U_E}{U_R}$")
plt.ylabel(r"$\frac{d_{Screen}}{d_{Initial}}$")
plt.grid()
plt.show()

# sort data for pgfplots
data = list(zip(u_e_data, d_data))
```

```python
data_sorted = sorted(data, key=lambda tup: tup[0])
u_e_data = [data_sorted[i][0] for i in range(0, len(data_sorted))]
d_data = [data_sorted[i][1] for i in range(0, len(data_sorted))]

# store data for pgfplots
with open("vmi_sim_optimalratio_final.dat", "w") as doc:
    doc_string = ""  # "u_e/u_r\td_screen/d_initial\n"
    for ue, d in zip(u_e_data, d_data):
        doc_string += str(ue/u_r) + "\t" + str(d) + "\n"
    doc.write(doc_string)
```

```python
"""
This module contains code to process the data in vmi_DUCR. We are supposed
to check wether the simulation results in vmi mode depend on the actual value
of the voltages or only on there ratio.
"""
from matplotlib import pyplot as plt

z_particle1 = [-2.94715e+001, -4.13515e+001, -3.39256e+001, -5.78800e+001]
z_particle2 = [-2.95053e+001, -4.14551e+001, -3.39857e+001, -5.80921e+001]

d_screen = [abs(zp1 - zp2) for zp1, zp2 in zip(z_particle1, z_particle2)]
u_r = [2, 1, 1.5, 0.5]

plt.scatter(u_r, d_screen)
plt.show()

with open("ducr_plot_data.dat", "w") as doc:
    doc_string = ""
    for d, u in zip(d_screen, u_r):
        doc_string += str(u) + "\t" + str(d) + "\n"
    doc.write(doc_string)
```

```python
"""
This module contains code to process data generated by SimIon simulations.
In this simulations the VMI mode was used.
"""

from matplotlib import pyplot as plt
import numpy as np


def stddev(data: list) -> float:
    """
    This function determines the standard deviation for a
    list of 1d data.
    """
    mean = sum(data)/len(data)
    s_sq_1 = 0
    for i in range(0, len(data)):
        s_sq_1 += (data[i] - mean)**2
    return np.sqrt(s_sq_1/(len(data) - 1))


YY, ZZ = list(), list()
with open("vmi_bunch/vmi_bunch2", "r") as data:
    doc_raw = data.read()
    lines = doc_raw.split("\n")
    for line in lines[12:-1]:
        line_cut = line.split("\t")
        YY.append(float(line_cut[6]))
        ZZ.append(float(line_cut[7]))

# plot data
plt.scatter(YY[0:500], ZZ[0:500], s=3, label="E=0.1eV")
plt.scatter(YY[500:1000], ZZ[500:1000], s=3, label="E=0.2eV")
plt.scatter(YY[1000:1500], ZZ[1000:1500], s=3, label="E=0.3eV")
plt.xlabel("y")
plt.ylabel("z")
plt.legend(loc=1, framealpha=1)
plt.grid()
plt.show()

# save data for pgfplots
with open("vmi_bunch_p1.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[0:500], ZZ[0:500]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)
with open("vmi_bunch_p2.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[500:1000], ZZ[500:1000]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)
with open("vmi_bunch_p3.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[1000:1500], ZZ[1000:1500]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)

# estimate radii of the circles
p1_data = [np.sqrt(tup[0]**2 + tup[1]**2) for tup in zip(YY[0:500], ZZ[0:500])]
p1_data_sorted = sorted(p1_data, reverse=True)
r1 = sum(p1_data_sorted[0:10])/len(p1_data_sorted[0:10])
sr1 = stddev(p1_data_sorted[0:10])

p2_data = [np.sqrt(tup[0]**2 + tup[1]**2) for tup in zip(YY[500:1000], ZZ[500:1000])
    ]
p2_data_sorted = sorted(p2_data, reverse=True)
r2 = sum(p2_data_sorted[0:10])/len(p2_data_sorted[0:10])
sr2 = stddev(p2_data_sorted[0:10])

p3_data = [np.sqrt(tup[0]**2 + tup[1]**2) for tup in zip(YY[1000:1500], ZZ
    [1000:1500])]
p3_data_sorted = sorted(p3_data, reverse=True)
r3 = sum(p3_data_sorted[0:10])/len(p3_data_sorted[0:10])
sr3 = stddev(p3_data_sorted[0:10])

print("( ", r1, " +- ", sr1, ") mm")
print("( ", r2, " +- ", sr2, ") mm")
print("( ", r3, " +- ", sr3, ") mm")
```

```python
"""
This module contains code to process data generated by SimIon simulations.
In this simulations the VMI mode was used.
"""
```

```python

from matplotlib import pyplot as plt
import numpy as np


def stddev(data: list) -> float:
    """
    This function determines the standard deviation for a
    list of 1d data.
    """
    mean = sum(data)/len(data)
    s_sq_1 = 0
    for i in range(0, len(data)):
        s_sq_1 += (data[i] - mean)**2
    return np.sqrt(s_sq_1/(len(data) - 1))


YY, ZZ = list(), list()
with open("vmi_bunch/vmi_bunch4", "r") as data:
    doc_raw = data.read()
    lines = doc_raw.split("\n")
    for line in lines[12:-1]:
        line_cut = line.split("\t")
        YY.append(float(line_cut[6]))
        ZZ.append(float(line_cut[7]))

# plot data
plt.scatter(YY[0:500], ZZ[0:500], s=3, label="E=0.1eV")
plt.scatter(YY[500:1000], ZZ[500:1000], s=3, label="E=0.2eV")
plt.scatter(YY[1000:1500], ZZ[1000:1500], s=3, label="E=0.3eV")
plt.xlabel("y")
plt.ylabel("z")
plt.legend(loc=1, framealpha=1)
plt.grid()
plt.show()

# save data for pgfplots
with open("vmi_bunch_p1_opt2.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[0:500], ZZ[0:500]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)
with open("vmi_bunch_p2_opt2.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[500:1000], ZZ[500:1000]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)
with open("vmi_bunch_p3_opt2.dat", "w") as doc:
    doc_string = ""  # y pos in [mm]\tz pos in [mm]
    for yy, zz in zip(YY[1000:1500], ZZ[1000:1500]):
        doc_string += str(yy) + "\t" + str(zz) + "\n"
    doc.write(doc_string)
```

SMI-Mode

```python
"""
This module contains data from SimIon simulations. The data is used to find
```

```python
the optimal ratio of U_E/U_R in SMI mode.
"""


import matplotlib.pyplot as plt
import numpy as np

# data to find minimum of screen distance
z_screen1 = [5.89714e1, 2.68545e1, 7.40245, -5.78745, -1.54193e1, -2.28084e1,
             -2.86790e1, -3.34667e1, -3.74514e1, -4.08220e1, -4.37110e1]  # mm
z_screen2 = [-5.89714e1, 2.6854e1, -7.40245, 5.78745, 1.54193e1, 2.28084e1,
             2.86790e1, 3.34667e1, 3.74514e1, 4.08220e1, 4.37110e1]  # mm
u_e = np.array([3000, 2900, 2800, 2700, 2600, 2500, 2400, 2300, 2200, 2100, 2000])
    # kV

#  closer measurements
z_s1 = [-1.18911e1, -9.47808, -6.88210, -4.08013, -1.04474, 2.25680]
z_s2 = [1.18911e1, 9.47808, 6.88210, 4.08013, 1.04474, 2.25680]
u_e_close = np.array([2640, 2665, 2690, 2715, 2740, 2765])

# even closer measurements
z_1_close = [1.57343, -4.06870e-1]
z_2_close = [-1.57343, 4.06870e-1]
u_e_closer = np.array([2760, 2745])

# simulation parameters
u_r = 3000  # kV

# distance on detector screen
d_screen = [np.abs(z1) + np.abs(z2) for z1, z2 in zip(z_screen1, z_screen2)]  # mm
d_screen_close = [np.abs(z1) + np.abs(z2) for z1, z2 in zip(z_s1, z_s2)]  # mm
d_screen_closer = [np.abs(z1) + np.abs(z2) for z1, z2 in zip(z_1_close, z_2_close)]
     # mm

# combine all data
d_data = d_screen + d_screen_close + d_screen_closer
u_e_data = np.concatenate((u_e, u_e_close, u_e_closer), axis=None)

# plot data
plt.scatter(u_e_data/u_r, d_data, marker="x", color="red")
plt.xlabel(r"$\frac{U_E}{U_R}$")
plt.ylabel(r"$d_{Screen}$ in [mm]")
plt.grid()
plt.show()

# sort data for pgfplots
data = list(zip(u_e_data, d_data))
data_sorted = sorted(data, key=lambda tup: tup[0])
u_e_data = [data_sorted[i][0] for i in range(0, len(data_sorted))]
d_data = [data_sorted[i][1] for i in range(0, len(data_sorted))]

# store data for pgfplots
with open("smi_sim_optimalratio_final.dat", "w") as doc:
    doc_string = "u_e/u_r\td_screen [mm]\n"
    for ue, d in zip(u_e_data, d_data):
        doc_string += str(ue/u_r) + "\t" + str(d) + "\n"
    doc.write(doc_string)
```

```python
"""
This module contains code to process data generated by SimIon simulations.
In this simulations the SMI mode was used to map a zylindric volume onto
the detector screen.
"""


from matplotlib import pyplot as plt

YYs = list()
ZZs = list()

for i in ["1", "2", "3", "4"]:
    YY, ZZ = list(), list()
    with open("smi_bunch/smi_bunch" + i, "r") as data:
        doc_raw = data.read()
        lines = doc_raw.split("\n")
        for line in lines[12:-1]:
            line_cut = line.split("\t")
            YY.append(float(line_cut[6]))
            ZZ.append(float(line_cut[7]))
    YYs.append(YY)
    ZZs.append(ZZ)


PARTICLES = list()

for yy, zz in zip(YYs[3:4], ZZs[3:4]):
    YY_P1 = yy[0:500]
    ZZ_P1 = zz[0:500]
    YY_P2 = yy[500:1000]
    ZZ_P2 = zz[500:1000]

with open("smi_bunch_p1_UE2715.txt", "w") as doc:
    doc_string = ""
    for y, z in zip(YY_P1, ZZ_P1):
        doc_string += str(y) + "\t" + str(z) + "\n"
    doc.write(doc_string)
with open("smi_bunch_p2_UE2715.txt", "w") as doc:
    doc_string = ""
    for y, z in zip(YY_P2, ZZ_P2):
        doc_string += str(y) + "\t" + str(z) + "\n"
    doc.write(doc_string)

plt.scatter(YY_P1, ZZ_P1, s=5, label="E=0.2eV")
plt.scatter(YY_P2, ZZ_P2, s=5, label="E=0.1eV")

plt.xlabel("y")
plt.ylabel("z")
plt.legend(loc=1, framealpha=1)
plt.grid()
plt.show()
```

C.2 Analysis

Oven Measurement

```python
import matplotlib.pyplot as plt
import numpy as np


temp = []
flux = []

temperr = []
fluxerr = []

with open("ovendata.dat", "r") as data:
    raw_data = data.read()
    lines = raw_data.split("\n")
    for line in lines[1:-1]:
        entries = line.split(",")
        temp.append(float(entries[0]))
        flux.append((float(entries[1]) - 0.031) * 10**(-9) / (6 * 1.602*10**(-19)))
        temperr.append(float(entries[2]))
        fluxerr.append((np.sqrt(2) * float(entries[3])) * 10**(-9) / (6 *
    1.602*10**(-19)))


with open("offsetdata.dat", "w") as off_data:
    off_data.write("temt\tflux\temperr\tfluxerr\n")
    for t, f, te, fe in zip(temp, flux, temperr, fluxerr):
        off_data.write(str(t) + "\t" + str(f) + "\t" + str(te) + "\t" + str(fe)
                       + "\n")

plt.errorbar(temp, flux, xerr=temperr, yerr=fluxerr)
plt.show()
```

SMI

```python
'''
This module takes the experimental SMI-data and calculated the best voltageratio
U_E / U_R
'''
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


def read_smi_data(filename):
    with open(filename, "r") as smidata:
        data = smidata.read()
        lines = data.split('\n')
        pixel = [[float(entry) for entry in line.split(",")] for line in
                 lines[0:-1]]
    return pixel


def clear_noise(pixellist, backlist):
    return [[d - b for d, b in zip(datlines, backlines)] for datlines,
            backlines in zip(pixellist, backlist)]


def gauss(x, A, mu, sigma, C):
    return A * np.exp(-((x - mu)**2) / (2 * sigma**2)) + C


def fwhm(sigma, sigma_err):
    return 2.355 * sigma, 2.355 * sigma_err


ratio_list = np.linspace(86, 94, 9)

pixel_background = read_smi_data("../smi_ratio/ion_smi_background.csv")

pixel_1 = clear_noise(read_smi_data("ion_smi_volrat_86.csv"), pixel_background)
pixel_2 = clear_noise(read_smi_data("ion_smi_volrat_87.csv"), pixel_background)
pixel_3 = clear_noise(read_smi_data("ion_smi_volrat_88.csv"), pixel_background)
pixel_4 = clear_noise(read_smi_data("ion_smi_volrat_89.csv"), pixel_background)
pixel_5 = clear_noise(read_smi_data("ion_smi_volrat_90.csv"), pixel_background)
pixel_6 = clear_noise(read_smi_data("ion_smi_volrat_91.csv"), pixel_background)
pixel_7 = clear_noise(read_smi_data("ion_smi_volrat_92.csv"), pixel_background)
pixel_8 = clear_noise(read_smi_data("ion_smi_volrat_93.csv"), pixel_background)
pixel_9 = clear_noise(read_smi_data("ion_smi_volrat_94.csv"), pixel_background)

big_pixellist = [pixel_1, pixel_2, pixel_3, pixel_4, pixel_5, pixel_6, pixel_7,
                 pixel_8, pixel_9]
big_eind_pixellist = []

for pixellist in big_pixellist:
    big_eind_pixellist.append([np.sum(line) for line in pixellist])


xx = np.linspace(0, len(big_eind_pixellist[0]), len(big_eind_pixellist[0]))

plt.scatter(xx[500:700], big_eind_pixellist[0][500:700])
popt0, pcov0 = curve_fit(gauss, xx[500:700], big_eind_pixellist[0][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt0))

plt.scatter(xx[500:700], big_eind_pixellist[1][500:700])
popt1, pcov1 = curve_fit(gauss, xx[500:700], big_eind_pixellist[1][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt1))

plt.scatter(xx[500:700], big_eind_pixellist[2][500:700])
popt2, pcov2 = curve_fit(gauss, xx[500:700], big_eind_pixellist[2][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt2))

plt.scatter(xx[500:700], big_eind_pixellist[3][500:700])
popt3, pcov3 = curve_fit(gauss, xx[500:700], big_eind_pixellist[3][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt3))

plt.scatter(xx[500:700], big_eind_pixellist[4][500:700])
popt4, pcov4 = curve_fit(gauss, xx[500:700], big_eind_pixellist[4][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt4))

plt.scatter(xx[500:700], big_eind_pixellist[5][500:700])
popt5, pcov5 = curve_fit(gauss, xx[500:700], big_eind_pixellist[5][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt5))

plt.scatter(xx[500:700], big_eind_pixellist[6][500:700])
popt6, pcov6 = curve_fit(gauss, xx[500:700], big_eind_pixellist[6][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt6))

plt.scatter(xx[500:700], big_eind_pixellist[7][500:700])
popt7, pcov7 = curve_fit(gauss, xx[500:700], big_eind_pixellist[7][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt7))

plt.scatter(xx[500:700], big_eind_pixellist[8][500:700])
popt8, pcov8 = curve_fit(gauss, xx[500:700], big_eind_pixellist[8][500:700],
                         p0=[220000, 600, 10, 60000])
plt.plot(xx[500:700], gauss(xx[500:700], *popt8))


sigma_list = [popt0[2], popt1[2], popt2[2], popt3[2], popt4[2], popt5[2],
              popt6[2], popt7[2], popt8[2]]

sigma_err_list = [pcov0[2][2], pcov1[2][2], pcov2[2][2], pcov3[2][2],
                  pcov4[2][2], pcov5[2][2], pcov6[2][2], pcov7[2][2],
                  pcov8[2][2]]

with open("smi_ratio.dat", "w") as datafile:
    datafile.write("ratio\tsigma\terr\n")
    for ratio, sigma, err in zip(ratio_list, sigma_list, sigma_err_list):
        datafile.write(str(ratio) + "\t" + str(sigma) + "\t" +
                       str(np.sqrt(err)) + "\n")


#  determine the variance for the best voltageratio with ue/ur = 90:
#  use pixel_5

maximum = 0
i, j = 0, 0
counti = 0

for line in pixel_5:
    countj = 0
    for entry in line:
        if entry > maximum:
            maximum = entry
            i = counti
            j = countj
        countj += 1
    counti += 1

max_list = []
for line in pixel_5:
    max_list.append(line[705])

plt.clf()
plt.scatter(xx[500:700], max_list[500:700])
poptmax, pcovmax = curve_fit(gauss, xx[500:700], max_list[500:700],
                         p0=[220, 600, 10, 0])
```

```
plt.plot(xx[500:700], gauss(xx[500:700], *poptmax))
plt.show()
print(poptmax[2], np.sqrt(pcovmax[2][2]))
```

```
'''
This module takes SMI-data and calculates the image ratio.
'''
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


def read_smi_data(filename):
    with open(filename, "r") as smidata:
        data = smidata.read()
        lines = data.split('\n')
        pixel = [[float(entry) for entry in line.split(",")] for line in
                 lines[0:-1]]
    return pixel


def clear_noise(pixellist, backlist):
    return [[d - b for d, b in zip(datlines, backlines)] for datlines,
            backlines in zip(pixellist, backlist)]


def gauss(x, A, mu, sigma, C):
    return A * np.exp(-((x - mu)**2) / (2 * sigma**2)) + C

def linear(x, m, b):
    return m * x + b

def write_file(filename, xx, pix_list, popt):
    xfit = np.linspace(xx[0], xx[-1], 100)
    with open(filename + "data.dat", "w") as file:
        file.write("x\tpix\n")
        for x, pix in zip(xx, pix_list):
            file.write(str(x) + "\t" + str(pix) + "\n")
    with open(filename + "fit.dat", "w") as fitfile:
        fitfile.write("xfit\tfit\n")
        for x in xfit:
            fitfile.write(str(x) + "\t" + str(gauss(x, *popt)) + "\n")


lense_pos = np.linspace(7, 4, 7)  # in mm

pixel_background = read_smi_data("ion_smi_background.csv")

pixel_1 = clear_noise(read_smi_data("ion_smi1.csv"), pixel_background)
pixel_2 = clear_noise(read_smi_data("ion_smi2.csv"), pixel_background)
pixel_3 = clear_noise(read_smi_data("ion_smi3.csv"), pixel_background)
pixel_4 = clear_noise(read_smi_data("ion_smi4.csv"), pixel_background)
pixel_5 = clear_noise(read_smi_data("ion_smi5.csv"), pixel_background)
pixel_6 = clear_noise(read_smi_data("ion_smi6.csv"), pixel_background)
pixel_7 = clear_noise(read_smi_data("ion_smi7.csv"), pixel_background)


summed_pixel = [[p1 + p2 + p3 + p4 + p5 + p6 + p7 for p1, p2, p3, p4, p5, p6,
    p7 in zip(l1, l2, l3, l4, l5, l6, l7)] for l1, l2, l3, l4, l5, l6, l7 in
    zip(pixel_1, pixel_2, pixel_3, pixel_4, pixel_5, pixel_6, pixel_7)]

plt.xlabel(r"$x$-position in pixel")
plt.ylabel(r"$y$-position in pixel")
plt.imshow(summed_pixel, cmap="gray")
plt.savefig("ion_ratio.pdf")


big_pixellist = [pixel_1, pixel_2, pixel_3, pixel_4, pixel_5, pixel_6, pixel_7]
big_eind_pixellist = []

for pixellist in big_pixellist:
    big_eind_pixellist.append([np.sum(line) for line in pixellist])


xx = np.linspace(0, len(big_eind_pixellist[0]), len(big_eind_pixellist[0]))

plt.scatter(xx[850:900], big_eind_pixellist[0][850:900])
popt0, pcov0 = curve_fit(gauss, xx[850:900], big_eind_pixellist[0][850:900],
        p0=[220000, 880, 10, 60000])
plt.plot(xx[850:900], gauss(xx[850:900], *popt0))

plt.scatter(xx[800:850], big_eind_pixellist[1][800:850])
popt1, pcov1 = curve_fit(gauss, xx[800:850], big_eind_pixellist[1][800:850],
        p0=[220000, 830, 10, 60000])
plt.plot(xx[800:850], gauss(xx[800:850], *popt1))

plt.scatter(xx[750:800], big_eind_pixellist[2][750:800])
popt2, pcov2 = curve_fit(gauss, xx[750:800], big_eind_pixellist[2][750:800],
        p0=[220000, 800, 10, 60000])
plt.plot(xx[750:800], gauss(xx[750:800], *popt2))

plt.scatter(xx[700:750], big_eind_pixellist[3][700:750])
popt3, pcov3 = curve_fit(gauss, xx[700:750], big_eind_pixellist[3][700:750],
        p0=[220000, 730, 10, 60000])
plt.plot(xx[700:750], gauss(xx[700:750], *popt3))

plt.scatter(xx[650:720], big_eind_pixellist[4][650:720])
popt4, pcov4 = curve_fit(gauss, xx[650:720], big_eind_pixellist[4][650:720],
        p0=[180000, 680, 10, 60000])
plt.plot(xx[650:720], gauss(xx[650:720], *popt4))

plt.scatter(xx[600:680], big_eind_pixellist[5][600:680])
popt5, pcov5 = curve_fit(gauss, xx[600:680], big_eind_pixellist[5][600:680],
        p0=[180000, 640, 10, 60000])
plt.plot(xx[600:680], gauss(xx[600:680], *popt5))

plt.scatter(xx[550:640], big_eind_pixellist[6][550:640])
popt6, pcov6 = curve_fit(gauss, xx[550:640], big_eind_pixellist[6][550:640],
        p0=[180000, 600, 10, 60000])
plt.plot(xx[550:640], gauss(xx[550:640], *popt6))

peak_pos = [popt0[1], popt1[1], popt2[1], popt3[1], popt4[1], popt5[1],
            popt6[1]]
peak_err = [pcov0[2][2], pcov1[2][2], pcov2[2][2], pcov3[2][2], pcov4[2][2],
            pcov5[2][2], pcov6[2][2]]

```

```python
popt_lin, pcov_lin = curve_fit(linear, lense_pos, peak_pos, sigma=peak_err)

with open("lin_pixel_lense.dat", "w") as data:
    data.write("%" + str(popt_lin[1]) + "\t" + str(np.sqrt(pcov_lin[1][1])) + "\n")
    data.write("lense\tpeak\tpeakerr\n")
    for lense, peak, err in zip(lense_pos, peak_pos, peak_err):
        data.write(str(lense) + "\t" + str(peak) + "\t" + str(np.sqrt(err)) + "\n")

plt.clf()
plt.scatter(lense_pos, peak_pos)
plt.plot(lense_pos, linear(lense_pos, *popt_lin))

print(popt_lin[0], popt_lin[1], pcov_lin[0][0])

write_file("pixel0", xx[850:900], big_eind_pixellist[0][850:900], popt0)
write_file("pixel1", xx[800:850], big_eind_pixellist[1][800:850], popt1)
write_file("pixel2", xx[750:800], big_eind_pixellist[2][750:800], popt2)
write_file("pixel3", xx[700:750], big_eind_pixellist[3][700:750], popt3)
write_file("pixel4", xx[650:720], big_eind_pixellist[4][650:720], popt4)
write_file("pixel5", xx[600:680], big_eind_pixellist[5][600:680], popt5)
write_file("pixel6", xx[550:640], big_eind_pixellist[6][550:640], popt6)
```

VMI

```python
"""
Module to analyse the measured energy spectrum of potassium. All taken data
is averaged and afterwards the averaged spectrum is analysed.
"""

from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

# Global variable to set with ratio is analysed
RATIO = "71"

# Set output header
print("====================================")
print("The ratio " + RATIO + " is analysed!")
print("====================================")
print("")

# This handles the used fonts in the plot to make it more or less consistent
# with the standard latex font.
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['mathtext.rm'] = 'Bitstream Vera Sans'
plt.rcParams['mathtext.it'] = 'Bitstream Vera Sans:italic'
plt.rcParams['mathtext.bf'] = 'Bitstream Vera Sans:bold'


def fwhm(sigma: float) -> float:
    """
    This function takes the standard deviation of a gaussian function
    and determines its [F]ull [W]idth at [H]alf [M]aximum
    """
    return 2 * np.sqrt(2 * np.log(2)) * sigma
```

```python


def average_lists(list_initial: list) -> list:
    """
    This function takes a list of equally sized float lists as an argument.
    The function returns a list of floats where every entry is the mean of
    the entrys of the sublists at the same position.

    Example:
        Input:  list_initial = [[1,2], [3, 4]]
        Output: list_final = [(1 + 3)/2, (2 + 4)/2]

    list_initial:   list of float lists
    list_final:     list of floats
    """
    list_final = list()
    for i in range(0, len(list_initial[0])):
        entry_sum = 0
        for j in range(0, len(list_initial)):
            entry_sum += list_initial[j][i]
        list_final.append(entry_sum/len(list_initial))
    return list_final


def gauss(x: float, mu: float, sigma: float, amp: float, off: float) -> float:
    """
    Gauss function used to fit data.

    x: x-Position
    mu: Mean
    sigma: Standard Deviation
    amp: Amplitude
    off: Offset on y-Axis
    """
    return amp * np.exp(-(x - mu)**2/(2 * sigma**2)) + off


def energy(r_pix: float, mu_cal, e_cal) -> float:
    """
    Function which calculates the energy dependent on the radius
    (in pixels) on the screen.

    r_pix: radius in pixel
    mu_cal: radius of calibration signal
    e_cal: energy of calibration signal
    """
    return e_cal * ((r_pix**2) / (mu_cal**2))


def final_energy(ekin: float) -> float:
    """
    Calculates the level energy with given kinetic energy of the measured
    electron.
    Used values are taken from the Instruction.pdf and calculations found in
    the mathematica notebook.

    ekin: kinetic energy of e- in [eV]
    """
    eion = 4.34066354  # eV
    egamma = 3.06491  # eV
    return ekin + eion - egamma


# Container for data: All measurements
IIs = list()
RRs = list()

SETS71 = ["a", "b", "c", "d", "e", "f", "g"]
SETS715 = ["a", "b", "c", "d", "e"]

if RATIO == "71":
    SETS = SETS71
else:
    SETS = SETS715

for i in SETS:
    # Container for data: One measurement
    # NOTE: II and RR get redefined later.
    II = list()
    RR = list()

    # Read data
    with open("./data" + RATIO + "/el_vmi_"
              + RATIO + i + "_pes.dat", "r") as doc:
        data_raw = doc.read()
        lines = data_raw.split("\n")
        for line in lines[0:-1]:
            entries = line.split("\t")
            RR.append(float(entries[0]))
            II.append(float(entries[1]))

    # NOTE: Save Plot of every Dataset (Set 0 if not wanted)
    if 0:
        plt.scatter(RR, II, marker="x", s=5, c="red")
        plt.xlim(0, 600)
        plt.xlabel("Radial Position on the Detector Screen in [pixel]")
        plt.ylabel("Intensity [arb. unit]")
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":")
        plt.grid(which="major", linestyle="-")
        plt.savefig("./plots_raw/ang_dataset_" + i + "ratio" + RATIO + ".pdf",
                    format="pdf")
        plt.clf()

    # Append Dataset to Container
    IIs.append(II)
    RRs.append(RR)

# Raw Data
# Redefine II and RR
RR = RRs[0]
II = average_lists(IIs)

# Calibrate energy axis
# ====================

# Fit Gauss model (PEAK: 1) to data for calibration
INITIALGUESS1 = [450, 10, 0, 0]
popt1, pcov1 = curve_fit(gauss, RR[191:215], II[191:215], p0=INITIALGUESS1)
XX_GAUSS1 = np.linspace(430, 475, 60)
YY_GAUSS1 = gauss(XX_GAUSS1, *popt1)

# Get fit parameters
mu1 = popt1[0]
sigma1 = popt1[1]
mu1_error = np.sqrt(pcov1[0][0])


# Calibrated x-axis
energy1 = 1.78915  # eV Used for calibration
EE = [energy(r_pix, mu1, energy1) for r_pix in RRs[0]]

# Save energy axis for Mali
with open("energy_axis.txt", "w") as data:
    doc_str = ""
    for e in EE:
        doc_str += str(e) + "\n"
    data.write(doc_str)

# Plot raw data (x - axis not calibrated)
# =======================================
# NOTE: Non integer pixel radii due to inverse abel transform
plt.scatter(RR, II, marker="x", s=5, c="red", label="data")
plt.plot(XX_GAUSS1, YY_GAUSS1, label="fit1: mu=" + str(popt1[0]))
# plt.ylim(0, 0.03)
# plt.xlim(0, 600)
plt.xlabel("Radial Position on the Detector Screen [pixels]")
plt.ylabel("Intensity [arb. unit]")
plt.legend(loc=1)
plt.grid(which="both")
plt.show()
plt.clf()

# Save raw data
# =============
# NOTE: Save raw data for pgfplots
with open("pse_plot_data_cal" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for r, i in zip(RR, II):
        doc_string += str(r) + "\t" + str(i) + "\n"
    doc.write(doc_string)
with open("pse_gaussfit1_data_cal" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for r, i in zip(XX_GAUSS1, YY_GAUSS1):
        doc_string += str(r) + "\t" + str(i) + "\n"
    doc.write(doc_string)

# Determine Fit Parameters
# ========================

# Fit Gauss model (PEAK: 1) to data
INITIALGUESS1 = [1.75, 0.2, 0, 0]
popt1, pcov1 = curve_fit(gauss, EE[191:215], II[191:215], p0=INITIALGUESS1)
XX_GAUSS1 = np.linspace(energy(430, mu1, energy1),
                        energy(475, mu1, energy1), 60)
YY_GAUSS1 = gauss(XX_GAUSS1, *popt1)

# Fit Gauss model (PEAK: 2) to data
INITIALGUESS2 = [1.4, 0.1, 0, 0]
popt2, pcov2 = curve_fit(gauss, EE[166:176], II[166:176], p0=INITIALGUESS2)
XX_GAUSS2 = np.linspace(energy(380, mu1, energy1),
                        energy(410, mu1, energy1), 60)
YY_GAUSS2 = gauss(XX_GAUSS2, *popt2)

# Fit Gauss model (PEAK: 3) to data
INITIALGUESS3 = [0.4, 0.1, 0, 0]
popt3, pcov3 = curve_fit(gauss, EE[74:91], II[74:91], p0=INITIALGUESS3)
XX_GAUSS3 = np.linspace(energy(170, mu1, energy1),
                        energy(230, mu1, energy1), 60)
YY_GAUSS3 = gauss(XX_GAUSS3, *popt3)

# Plot data (calibrated x-axis)
# ============================

plt.scatter(EE, II, marker="x", s=5, c="red", label="data")
plt.plot(XX_GAUSS1, YY_GAUSS1, label="fit1: mu=" + str(popt1[0]))
plt.plot(XX_GAUSS2, YY_GAUSS2, label="fit2: mu=" + str(popt2[0]))
plt.plot(XX_GAUSS3, YY_GAUSS3, label="fit3: mu=" + str(popt3[0]))
# plt.ylim(0, 0.03)
# plt.xlim(0, 2)
plt.xlabel("Energy [eV]")
plt.ylabel("Intensity [arb. unit]")
plt.legend(loc=1)
plt.grid(which="both")
plt.show()

# Save data
# =========
# NOTE: Save data for pgfplots
with open("pse_plot_data" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for e, i in zip(EE, II):
        doc_string += str(e) + "\t" + str(i) + "\n"
    doc.write(doc_string)
with open("pse_gaussfit1_data" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for e, i in zip(XX_GAUSS1, YY_GAUSS1):
        doc_string += str(e) + "\t" + str(i) + "\n"
    doc.write(doc_string)
with open("pse_gaussfit2_data" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for e, i in zip(XX_GAUSS2, YY_GAUSS2):
        doc_string += str(e) + "\t" + str(i) + "\n"
    doc.write(doc_string)
with open("pse_gaussfit3_data" + RATIO + ".dat", "w") as doc:
    doc_string = ""
    for e, i in zip(XX_GAUSS3, YY_GAUSS3):
        doc_string += str(e) + "\t" + str(i) + "\n"
    doc.write(doc_string)

# Calculate Energies
# ==================

# Get fit parameters
mu1 = popt1[0]
sigma1 = popt1[1]
mu1_error = np.sqrt(pcov1[0][0])
sigma1_error = np.sqrt(pcov1[1][1])

mu2 = popt2[0]
sigma2 = popt2[1]
mu2_error = np.sqrt(pcov2[0][0])
sigma2_error = np.sqrt(pcov2[1][1])

mu3 = popt3[0]
sigma3 = popt3[1]
mu3_error = np.sqrt(pcov3[0][0])
sigma3_error = np.sqrt(pcov3[1][1])

# Get list index of values for beta-determination
# ==============================================
# NOTE: This ain't nice, but it works...
# Determine index intervals for FWHM around peaks; Brute Force
AA = list()
BB = list()
for mu, width in zip([mu1, mu2, mu3],
                     [fwhm(sigma1), fwhm(sigma2), fwhm(sigma3)]):
    i = 0
    while True:
        if EE[i] < (mu - width/2) and EE[i+1] > (mu - width/2):
            a = i
            break
        i += 1
    i = 0
    while True:
        if EE[i] < (mu + width/2) and EE[i+1] > (mu + width/2):
            b = i
            break
        i += 1
    AA.append(a)
    BB.append(b)
# Save index intervals
with open("indices_beta.txt", "w") as data:
    doc_str = "[a, b]\n"
    for a, b in zip(AA, BB):
        doc_str += str(a) + "\t" + str(b) + "\n"
    data.write(doc_str)

print("Kinetic Energies:\n=================")
print("E_{kin,1} = (", round(mu1, 5), "+-", round(mu1_error, 8), ") eV")
print("E_{kin,2} = (", round(mu2, 4), "+-", round(mu2_error, 8), ") eV")
print("E_{kin,3} = (", round(mu3, 4), "+-", round(mu3_error, 8), ") eV")
print("")

# Calculate level energies
energy1 = final_energy(mu1)
energy2 = final_energy(mu2)
energy3 = final_energy(mu3)

# Calculate level energy errors
```

```python
e_gamma_err = 0.00378825  # eV, if lambda_err = 0.5 nm
s_energy1 = 0  # Used for calibration
s_energy2 = np.sqrt(mu2_error**2 + e_gamma_err**2)
s_energy3 = np.sqrt(mu3_error**2 + e_gamma_err**2)

print("Level Energies:\n=================")
print("E_1 = (", round(energy1, 5), "+-", round(s_energy1, 5), ") eV")
print("E_2 = (", round(energy2, 4), "+-", round(s_energy2, 4), ") eV")
print("E_3 = (", round(energy3, 4), "+-", round(s_energy3, 4), ") eV")
print("")

# TODO: Something is wrong here... :(
# Calculate energy resolution
deltaE1 = fwhm(sigma1)/mu1
deltaE2 = fwhm(sigma2)/mu2
deltaE3 = fwhm(sigma3)/mu3

# Calculate resolution errors
s_deltaE1 = np.sqrt(((fwhm(sigma1)/mu1**2)*mu1_error)**2 +
                    ((1/mu1)*sigma1_error)**2)
s_deltaE2 = np.sqrt(((fwhm(sigma2)/mu2**2)*mu2_error)**2 +
                    ((1/mu2)*sigma2_error)**2)
s_deltaE3 = np.sqrt(((fwhm(sigma3)/mu3**2)*mu3_error)**2 +
                    ((1/mu3)*sigma3_error)**2)

print("Energy Resolution:\n=================")
print("dE_1 = (", deltaE1, "+-", s_deltaE1, ") %")
print("dE_2 = (", deltaE2, "+-", s_deltaE2, ") %")
print("dE_3 = (", deltaE3, "+-", s_deltaE3, ") %")
print("")
```

```python
"""
This module contains code to check how the VMI signal radius
changes wit different repellor voltages while the ratio stays
the same.
"""

import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import curve_fit

# Set to True if you want to see all the plots
PLOT_ALL = True

# This handles the used fonts in the plot to make it more or less consistent
# with the standard latex font.
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['mathtext.rm'] = 'Bitstream Vera Sans'
plt.rcParams['mathtext.it'] = 'Bitstream Vera Sans:italic'
plt.rcParams['mathtext.bf'] = 'Bitstream Vera Sans:bold'


def gauss(x: float, mu: float, sigma: float, amp: float, off: float) -> float:
    """
    Gauss function used to fit data.

    x: x-Position
    mu: Mean
    sigma: Standard Deviation
    amp: Amplitude
    off: Offset on y-Axis
    """
    return amp * np.exp(-(x - mu)**2/(2 * sigma**2)) + off


# Define repellor voltage lists and mirror them afterwards because i am dumb af
U_R = [1.5, 2, 2.5, 3, 3.5, 4, 4.5]
U_R = U_R[::-1]
U_R_STR = ["15", "2", "25", "3", "35", "4", "45"]
U_R_STR = U_R_STR[::-1]

# Initial guess for every fit seperatly, I fuggin hate it
INIT1 = [160, 5, 0.2, 0]
INIT2 = [170, 10, 0.2, 0]
INIT3 = [180, 10, 0.2, 0]
INIT4 = [200, 10, 0.2, 0]
INIT5 = [200, 10, 0.4, 0]
INIT6 = [200, 30, 1.0, 0]
INIT7 = [290, 30, 1.0, 0]
INITS = [INIT1, INIT2, INIT3, INIT4, INIT5, INIT6, INIT7]

# Fit intervall for every fit seperatly, end my suffering...
aa = [50, 50, 60, 60, 60, 70, 80]
bb = [90, 90, 100, 100, 120, 140, 150]

# Container for radii determined by fitting the lowest energy peak
RADII = list()
S_RADII = list()

# Get data fit gauss to data, usual buisness u know :(
for i in range(len(U_R)):

    # tetra paks for radii and intensity of every measurement,
    # gets recycled within every iteration.
    radius = list()
    intensity = list()

    # Get data, yeah boiii
    with open("el_vmi_" + U_R_STR[i] + "_pes.dat", "r") as data:
        lines = data.read().split("\n")
        for line in lines[1:-1]:
            radius.append(float(line.split("\t")[0]))
            intensity.append(float(line.split("\t")[1]))

    # Fit that hoe
    a = aa[i]
    b = bb[i]
    popt, pcov = curve_fit(gauss, radius[a:b], intensity[a:b], p0=INITS[i])

    # Save the radii and its errors
    RADII.append(popt[0])
    S_RADII.append(np.sqrt(pcov[0][0]))

    # Generate data to plot gauss fit, AGAIN
    xx = np.linspace(radius[a], radius[b], 100)
```

```python
    yy = gauss(xx, *popt)

    if PLOT_ALL:
        # Plot the data because everybody loves plots, looks as expected though
        plt.title(r"$U_R$ = " + str(U_R[i]) + "kV")
        plt.xlim(0, 600)
        plt.xlabel("Radial Position on the Detector Screen in [pixel]")
        plt.ylabel("Intensity [arb. unit]")
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":")
        plt.grid(which="major", linestyle="-")
        plt.scatter(radius, intensity, marker="x", color="red", label="Data")
        plt.scatter(radius[a:b], intensity[a:b], marker="x", color="blue",
                    label="Data used for Model Fit")
        plt.plot(xx, yy, color="black", label="Gauss")
        plt.legend(loc=2)
        plt.savefig("./raw_data/vmi_ur_dependence" + U_R_STR[i] + ".pdf",
                    format="PDF")
        plt.show()

# Plot the final data, because everybody loves plots
plt.xlabel(r"Repellor Voltage $U_R$ in [kV]")
plt.ylabel("Radial Position of the Peak on the Detector Screen in [pixel]")
plt.minorticks_on()
plt.grid(which="minor", linestyle=":")
plt.grid(which="major", linestyle="-")
plt.scatter(U_R, RADII, marker="x", color="red")
plt.show()

with open("vmi_ur_dependence.dat", "w") as data:
    doc_str = "u\tr\ts\n"
    for u, r, sr in zip(U_R, RADII, S_RADII):
        doc_str += str(u) + "\t" + str(r) + "\t" + str(sr) + "\n"
    data.write(doc_str)
```

```python
"""
Module to analyse the measured energy spectrum of potassium.
In this module we check the FWHM of the first peak (inner circle)
for different voltage ratios, to justify our chosen ratio.
"""

from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

# This handles the used fonts in the plot to make it more or less consistent
# with the standard latex font.
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['mathtext.rm'] = 'Bitstream Vera Sans'
plt.rcParams['mathtext.it'] = 'Bitstream Vera Sans:italic'
plt.rcParams['mathtext.bf'] = 'Bitstream Vera Sans:bold'


def fwhm(sigma: float) -> float:
    """
    This function takes the standard deviation of a gaussian function
    and determines its [F]ull [W]idth at [H]alf [M]aximum.
    See e.g. https://en.wikipedia.org/wiki/Full_width_at_half_maximum
    """
    return 2 * np.sqrt(2 * np.log(2)) * sigma


def gauss(x: float, mu: float, sigma: float, amp: float, off: float) -> float:
    """
    Gaussian function used to fit to the data. Gaussian model might not be
    totally justified by the theory, but is a good enough method to find the
    position of a "gauss-like" shaped peak.

    x: x-Position
    mu: Mean
    sigma: Standard Deviation
    amp: Amplitude
    off: Offset on y-Axis
    """
    return amp * np.exp(-(x - mu)**2/(2 * sigma**2)) + off


# Container for data: All measurements
IIs = list()
RRs = list()
MUs = list()
SIGMAs = list()
S_MUs = list()
S_SIGMAs = list()

for i in ["69", "70", "71", "715", "72", "73"]:
    # Container for data: One measurement
    II = list()
    RR = list()

    # Read data
    with open("./el_vmi_" + i + "_pes.dat", "r") as doc:
        data_raw = doc.read()
        lines = data_raw.split("\n")
        for line in lines[0:-1]:
            entries = line.split("\t")
            RR.append(float(entries[0]))
            II.append(float(entries[1]))

    # Fit Gauss model (PEAK: 1) to data
    INITIALGUESS = [200, 100, 0, 0]
    popt, pcov = curve_fit(gauss, RR[0:160], II[0:160], p0=INITIALGUESS)

    # Get fit parameter
    mu, sigma = popt[0], popt[1]
    s_mu, s_sigma = np.sqrt(pcov[0][0]), np.sqrt(pcov[1][1])

    # Get fit data to plot it
    XX_GAUSS = np.linspace(RR[0], RR[160], 80)
    YY_GAUSS = gauss(XX_GAUSS, *popt)

    # NOTE: Save Plot of every Dataset (Set 0 if not wanted)
    if 1:
        plt.scatter(RR, II, marker="x", s=5, c="red", label="Data")
        plt.plot(XX_GAUSS, YY_GAUSS, c="blue", label="Gauss Model")
```

```python
        plt.xlim(0, 600)
        plt.xlabel("Radial Position on the Detector Screen in [pixel]")
        plt.ylabel("Intensity [arb. unit]")
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":")
        plt.grid(which="major", linestyle="-")
        plt.legend(loc=2)
        plt.savefig("./plots_raw/ang_dataset_" + i + ".pdf", format="pdf")
        plt.show()
        plt.clf()

    # Append Data to corresponding Container
    IIs.append(II)
    RRs.append(RR)
    MUs.append(mu)
    SIGMAs.append(sigma)
    S_MUs.append(s_mu)
    S_SIGMAs.append(s_sigma)

# Calculate desired quantities
FWHM = [fwhm(sigma) for sigma in SIGMAs]
S_FWHM = [fwhm(s_sigma) for s_sigma in S_SIGMAs]
RATIO = [69, 70, 71, 71.5, 72, 73]
S_RATIO = [0.2 for _ in RATIO]

# Plot the voltage ratio against the FWHM
plt.scatter(RATIO, FWHM, c="red")
plt.errorbar(RATIO, FWHM, xerr=S_RATIO, yerr=S_FWHM, ecolor="black",
             capsize=2.5, fmt="none")
plt.minorticks_on()
plt.xlabel(r"Voltage ratio $U_E / U_R$")
plt.ylabel(r"FWHM [pixel]")
plt.grid(which="minor", linestyle=":")
plt.grid(which="major", linestyle="-")
plt.show()

with open("vmi_ratio_data.dat", "w") as data:
    doc_str = "fwhm\tsfwhm\tr\tsr\n"
    for f, sf, r, sr in zip(FWHM, S_FWHM, RATIO, S_RATIO):
        doc_str += str(f) + "\t" + str(sf) + "\t" + str(r) + "\t" + str(sr) +\
                "\n"
    data.write(doc_str)
```

Anisotropy

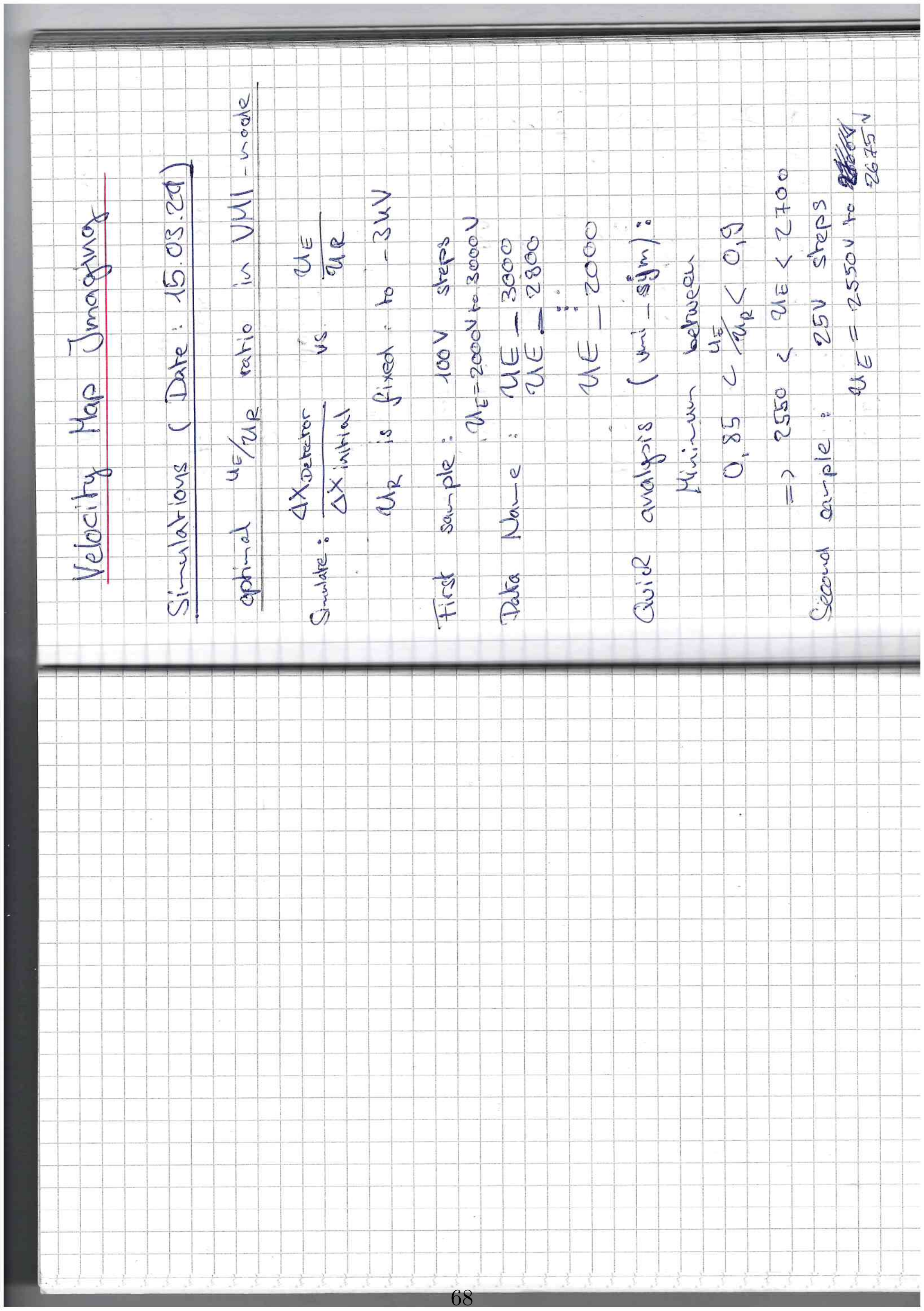
```python
'''
This module takes angular and radial VMI-data and calculated the anisotropy
    parameters.
'''
import numpy as np
import matplotlib.pyplot as plt

r = []

with open("../energy_analysis/energy_axis.txt", "r") as data:
    raw_data = data.read()
    lines = raw_data.split("\n")
    for line in lines[0:-1]:
        r.append(float(line))

def read_data(filename):
    with open(filename + "ang.dat", "r") as data:
        b1 = []
        b2 = []
        raw_data = data.read()
        lines = raw_data.split("\n")
        for line in lines[0:-1]:
            entries = line.split("\t")
            b1.append(float(entries[1]))
            b2.append(float(entries[2]))
    with open(filename + "pes.dat", "r") as data:
        pes = []
        raw_data = data.read()
        lines = raw_data.split("\n")
        for line in lines[0:-1]:
            pes.append(float(line.split("\t")[1]))
    weight_b1 = [b1 * p / np.sum(pes) for b1, p in zip(b1, pes)]
    weight_b2 = [b2 * p / np.sum(pes) for b2, p in zip(b2, pes)]
    return weight_b1, weight_b2


b1a, b2a = read_data("el_vmi_715_")
b1b, b2b = read_data("el_vmi_715a_")
b1c, b2c = read_data("el_vmi_715b_")
b1d, b2d = read_data("el_vmi_715c_")
b1e, b2e = read_data("el_vmi_715d_")


beta1_list = b1a + b1b + b1c + b1d + b1e
beta2_list = b2a + b2b + b2c + b2d + b2e
r_long = 5 * r


plt.ylim(-1, 2)
plt.scatter(r_long, beta1_list)
plt.scatter(r_long, beta2_list)
plt.scatter(r[190], 1)
plt.scatter(r[194], 1)
plt.scatter(r[166], 1)
plt.scatter(r[171], 1)
plt.scatter(r[79], 1)
plt.scatter(r[88], 1)
plt.show()


# calculate beta2
beta_1_left_l = b1a[79:87] + b1b[79:87] + b1c[79:87] + b1d[79:87] + b1e[79:87]
beta_1_middle_l = b1a[166:170] + b1b[166:170] + b1c[166:170] +\
                  b1d[166:170] + b1e[166:170]
beta_1_right_l = b1a[190:193] + b1b[190:193] + b1c[190:193] + b1d[190:193] +\
                 b1e[190:193]

beta_1_left = np.mean(beta_1_left_l)
beta_1_left_err = np.std(beta_1_left_l)
beta_1_middle = np.sum(beta_1_middle_l) / len(beta_1_middle_l)
```

```python
beta_1_middle_err = np.std(beta_1_middle_l)
beta_1_right = np.sum(beta_1_right_l) / len(beta_1_right_l)
beta_1_right_err = np.std(beta_1_right_l)

print("beta 2")
print(beta_1_left, beta_1_middle, beta_1_right)
print("beta 2 err")
print(beta_1_left_err, beta_1_middle_err, beta_1_right_err)

# calculate beta4
beta_2_left_l = b2a[79:87] + b2b[79:87] + b2c[79:87] + b2d[79:87] + b2e[79:87]
beta_2_middle_l = b2a[166:170] + b2b[166:170] + b2c[166:170] +\
                  b2d[166:170] + b2e[166:170]
beta_2_right_l = b2a[190:193] + b2b[190:193] + b2c[190:193] + b2d[190:193] +\
                 b2e[190:193]

beta_2_left = np.sum(beta_2_left_l) / len(beta_2_left_l)
beta_2_left_err = np.std(beta_2_left_l)
beta_2_middle = np.sum(beta_2_middle_l) / len(beta_2_middle_l)
beta_2_middle_err = np.std(beta_2_middle_l)
beta_2_right = np.sum(beta_2_right_l) / len(beta_2_right_l)
beta_2_right_err = np.std(beta_2_right_l)

print("beta 4")
print(beta_2_left, beta_2_middle, beta_2_right)
print("beta 4 err")
print(beta_2_left_err, beta_2_middle_err, beta_2_right_err)


with open("anisotropy.dat", "w") as data:
    data.write("energy\tbeta2\tbeta4\n")
    for e, b2, b4 in zip(r, beta1_list, beta2_list):
        data.write(str(e) + "\t" + str(b2) + "\t" + str(b4) + "\n")

print(r[79], r[87], r[166], r[170], r[190], r[193])
```

```python
'''
This module takes the experimental and the reference anisotropy parameters and
    calculates
the possible Able inversed picture.
'''
import numpy as np
import matplotlib.pyplot as plt


# reference values
beta_2 = [0.17, 0.86, 1.07]
beta_4 = [0, 0, 0.52]

# experimental values
# beta_2 = [0.002, 0.0097, 0.4]
# beta_4 = [0.006, 0.002, 0.3]
#
radius = [[79, 80, 81, 82, 83, 84, 85, 86, 87],
          [166, 167, 168, 169, 170],
          [190, 191, 192, 193]]


```

```
def legendre(theta, b2, b4):
    return (1 + b2 * (1.5*(np.cos(theta))**2 - 0.5) + b4 *
            (4.375*(np.cos(theta))**4 - 3.75*(np.cos(theta))**2 + 0.375))


dataarray = np.ones([200, 360])

for transition, tb2, tb4 in zip(radius, beta_2, beta_4):
    for r_ind in transition:
        for angle in range(0, 360):
            dataarray[r_ind, angle] = legendre((angle + 90) * np.pi / 180, tb2, tb4)

rbins = np.linspace(0, 200, 200)
phibins = np.linspace(0, 2*np.pi, 360)

ang, rad = np.meshgrid(phibins, rbins)
fig, ax = plt.subplots(subplot_kw=dict(projection="polar"))
ax.set_yticks([])
ax.set_xticks([])
pc = ax.pcolormesh(ang, rad, dataarray, cmap="viridis")

plt.savefig("anisot_exp.png")
plt.show()
```

D Lab Notes



Velocity Map Imaging

Simulations (Date: 15.03.21)

optimal u_E/u_R ratio in VMI-mode

Simulate: $\frac{\Delta X_{\text{Detector}}}{\Delta X_{\text{initial}}}$ vs $\frac{u_E}{u_R}$

u_R is fixed to -3kV

First sample: 100V steps

$u_E = 2000$V to 3000V

Data Name: UE_3000

UE_2800

⋮

UE_2000

Quick analysis (vmi_sim):

Minimum between

$0{,}85 < \frac{u_E}{u_R} < 0{,}9$

$\Rightarrow 2550 < u_E < 2700$

Second sample: 25V steps

$u_E = 2550$V to 2675V

Third sample: 10V steps

$\Longrightarrow 2580V < U_E < 2610V$

Extra sample at 2583V

Estimated optimal ratio for

VMI: $\left(\frac{U_E}{U_R}\right)_{opt} \approx 0{,}862 \pm 0{,}002$

$\Longrightarrow (U_E)_{opt} \approx (2586 \pm 6)\,V$

<u>different U_R, constant ratio?</u>

$U_R = 2kV$, $U_E = 1724V$

File: DUCR1

$U_R = 1kV$ $U_E = 862\,V$

File: DUCR2

$U_R = 1{,}5kV$ $U_E = 1293\,V$

File: DUCR3

$U_R = 0{,}5kV$ $U_E = 431V$

<u>Addition (date: 17.03)</u>

For $U_R = 2kV$: $U_E = 1784V$
$\Rightarrow$ DUCR1_r
$U_E = 1664\,V$
$\Rightarrow$ DUCR1_l

<u>VMI – Bunch</u>

Run 1: $E_{kin} = 0{,}1\ eV$

$\vec{v}$ direction: θ, φ uniform distributed

Data: vmi_bunch1

Run 2: $E_{kin} \in \{0{,}1eV; 0{,}2eV; 0{,}3eV\}$

$\vec{v}$ direction: θ, φ uniform distributed

Data: vmi_bunch2

Used $U_E = 2586V$
$U_R = -3kV$

<u>VMI – Bunch different U_E, U_R</u>

Run 3: $E_{kin} \in \{0{,}1eV; 0{,}2eV; 0{,}3eV\}$

$\vec{v}$ direction: θ, φ uniform distributed

Data: vmi_bunch3

Used $U_E = 434\,V$
$U_R = -500\,V$

Run 4: $E_{kin} \in \{0{,}1eV; 0{,}2eV; 0{,}3eV\}$

$\vec{v}$ direction: θ, φ uniform distributed

Data: vmi_bunch4

Used $U_E = 4340\,V$
$U_R = -5000\,V$

optimal U_E/U_R ratio in SMI-mode

Used Kalium mass 39,0883 u
(source: ciaaw.org)

First sample: 100 V steps
U_E = 2000 V to 3000 V

Data Name UE_3000

Quick analysis:
Minimum between
$0{,}88 < \frac{U_E}{U_R} <$ ~~[illegible]~~ $0{,}93$

SMI: $\left(\frac{U_E}{U_R}\right)_{opt} = 0{,}915 \pm 0{,}02$

$\left(U_E\right)_{opt} = (2745 \pm 6)\,V$

QMI - Bunch

Calculate Rayleigh-Length

$$z_R = \frac{\pi \cdot w_0^2}{\lambda}$$

w_0 : Beam waist
λ : wavelength

$$w_0 = \frac{\lambda f}{\pi w_e}$$

$\lambda = 404{,}52847$ nm (Instructions)
$f = 150$ mm (Instructions)
$w_e = 1$ mm (Instructions)

$\Longrightarrow w_0 = 1{,}9315 \cdot 10^{-5}$ m
$= 19{,}315$ µm

$\Longrightarrow z_R = 0{,}002897$ m
$= 2{,}897$ mm

~~Run 1: Line distribution~~

~~Source position : $[-z_R, z_R]$ uniform distribution~~

~~$\vec{v}$ distribution : φ, Θ uniform distributed~~

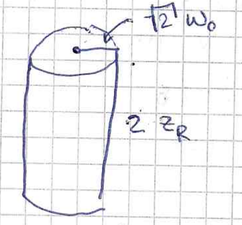
~~$E_{kin} = 0,1 eV ; 0,2 eV$~~

~~$U_E = 2745 V$ $U_R = 3 kV$~~

~~$\Rightarrow$ Not exactly the same diameter on the screen~~

~~$\Rightarrow$ Fine tuning the ratio~~

Run 1 : Cylindrical distribution



$E_{kin} \in \{0,1 eV ; 0,2 eV\}$

"optimal" bunch 1

$U_E = 2775 V$ bunch 2

$U_E = 2500 V$ bunch 3

$U_E = 2715 V$ bunch 4

Adjusting the Laser - Beam (16.03)

Beam path adjusted.

Laser wavelength tuned.

$\Rightarrow$ Picture of spectrum taken.

First Use of the spectrometer

(SMI - Mode with K^{+} Ions)

Laser :	50,62500 mA	(CC)
	1,575000 V	(PC)

MCP : 1600 V

Phosphor : 3,4 kV

JK - Oven : 157,2 °C

(Voltage 32,9 V)

$U_E / U_R = 0,9$

Varying lense position

~~pos~~ 0 : 6,29 mm

lense position

1	7mm
2	6,5mm
3	6,0mm
4	5,5mm
5	5,0mm
6	4,5mm
7	4,0mm

filename ion_sim .ppm
(sim_ratio

Varying $u_E/u_R = R$

in sim_voltageratio

R = 86%
87
⋮
94%

VMI - Mode (e^-)

153°C - Oven, while measurements

MCP : 1610 V
Phosphor: 3000 V
Laser (R - adjusted)

Oven measurement

Optimal voltages:

$U_E = 151,5$ V
$U_R = 279,5$ V
} offset/weird display but ratio should not be affected

T [°C]	I_{FC} [10 µA]	
155,4	3,022	
155,1	2,280	Big error
154,4	2,200	
153,9	2,130	
153,7	2,080	
153,6	2,020	
153,2	1,960	
153,0	1,910	
152,3	1,880	
151,8	1,850	
151,6	1,770	
151,0	1,740	
150,0	1,680	
149,5	1,620	
149,4	1,580	
149,1	1,555	
148,0	[illegible]	

147,7	1,444
147,2	1,380
146,5	1,340
146,1	1,300
145,5	1,270
145,0	1,230
144,6	1,200
144,0	1,180
143,5	1,160
143,0	1,130
142,7	1,100
142,5	1,090
142,0	1,060
141,2	1,030
140,6	1,000
140,3	0,990
139,8	0,960
139,0	0,940
138,6	0,915
138,3	0,892

138,0	0,870
137,2	0,840
136,5	0,830
136,2	0,810
136,0	0,790
135,5	0,770
135,0	0,750
134,3	0,735
133,8	0,720
133,2	0,700
132,4	0,680
131,9	0,666
131,2	0,650
130,5	0,620
130,0	0,600
129,7	0,580
129,4	0,570

128,0	0,550
127,6	0,530
127,3	0,520
126,6	0,500
126,0	0,495
125,3	0,480
125,0	0,470
124,6	0,460
124,0	0,450
123,0	0,430
122,5	0,420
122,0	0,420
121,2	0,400
120,7	0,385
120,0	0,370
119,5	0,360
119,0	0,350

118,5	0,345
118,0	0,335
117,5	0,330
116,5	0,320
116,0	0,315
115,5	0,310
115,0	0,300
114,5	0,300
114,0	0,280
113,0	0,279
112,5	0,272
112,3	0,266
112,0	0,261
111,5	0,255
111,0	0,248
110,5	0,244
110,0	0,237
109,0	0,228

smaller error

108,0	0,217
107,0	0,208
106,0	0,202
105,0	0,192
104,0	0,186
103,0	0,177
102,0	0,172
101,0	0,164
100,0	0,157
	[illegible]
97,0	0,140
94,0	0,125
91,0	0,113
88,0	0,100
85,0	0,090
82,0	0,083
79,0	0,077
76,0	0,071

73,0	0,065
70,0	0,060
67,0	0,057
64,0	0,053
61,0	0,050
58,0	0,046
55,0	0,044
52,0	0,042
49,0	0,040
46,0	
43,0	0,036
40,0	0,035
37,0	0,034
34,0	0,031

References

[1] URL: http://www-dick.chemie.uni-regensburg.de/IonImag.html.

[2] URL: https://commons.wikimedia.org/w/index.php?curid=6002103.

[3] URL: https://periodictable.com/Isotopes/019.39/index.html.

[4] URL: https://www.rp-photonics.com/microchannel_plates.html.

[5] Lutz Fechter. “Aufbau eines Velocity-Map-Imaging-Spektrometers und winkelaufgelöste Spektroskopie an Rubidium-dotierten Helium-Nanotröpfchen”. *Diplomarbeit* (2011).

[6] D. Manura and D. Dahl. *SIMION (R) 8.1 User Manual*. 2008. URL: http://simion.com/.

[7] A. Wituschek. *Versuchsanleitung: Velocity Map Imaging*.

[8] A. Wituschek et al. “A simple photoionization scheme for characterizing electron and ion spectrometers”. *Review of Scientific Instruments* (2016).