Implementing and Testing the Position Sensitive Anode

Sven Kiefer

Examiner: Prof. Dr. André Rubbia
Supervisor: Dr. Paolo Crivelli
Lars Frieder Gerchow

March 12, 2019

Abstract

The position sensitive anode is an element of a position sensitive detector which uses a resistive layer to divide incoming charges and thus enables a position reconstruction. Its slim design and compact form makes it less fragile and easier to implement compared to other anodes. The anode was tested in 3 different set up and a python script to evaluate the measurements was developed. During the testing, a design flaw of the anode was found. The measured events can create currents of up to 3 mA which damage the anode. Because of an already low signal to noise ratio of 1.5, the currents could not be reduced. Thus, the current design of the anode needs to be improved to use it as position sensitive detector. With a simulation, the limitations of the current anode design were confirmed. The measured events are always either to weak too find the position via triangulation or to strong and would damage the carbon layer.

Contents

1 Introduction

1.1 General remarks

Accelerators are powerful tools to advance our knowledge of particle physics to a fundamental level. The Large Hadron Collider at CERN is probably the most famous one because of its recent ground-breaking results. It can accelerate particles up to several TeV, which made it possible to search and find the Higgs boson. Even though the LHC is impressive, there exist many other smaller accelerators which can probe our knowledge at lower energies. At ETH, the Laboratory for Positron and Positronium Physics (LPPP) uses several low energy positron beams to experiment with positrons and positronium. These accelerators are constantly altered to fit the experiments. If necessary, some parts are replaced or updated to achieve even better results.

A very important diagnostic tool is the position sensitive detector (PSD). To detect a single positron, a micro channel plate (MCP) is used to amplify the event [1]. A single positron can be amplified to a signal of 10^6 electrons, which then hits the PSD. The detection is currently done with a phosphor plate and a CCD camera. After the event hits the phosphor plate, it illuminates and is then recorded by the camera. This implementation needs a lot of space and is difficult to install. Other detectors are all either large or fragile and thus difficult to mount.

There already exists designs for smaller PSAs, without having the disadvantage of being too large or fragile. Unfortunately, most of them trade a good readout signal for a smaller detection area which again is not very practical to use in a vacuum chamber. Last semester a similar detector element was developed at the LPPP. It combines the advantage of the old designs with a larger detection area. In the end, this detector should optimize the precision of the position measurement, maximize the detection area and minimize the space needed in the vacuum tube.

1.2 Theory of the position sensitive anode

The position sensitive anode (PSA) was developed by Jonas Kunath during a semester thesis [2] at the LPPP (see figure 6). The idea was to replace the phosphor screen with a new PSA which is easier to install without losing precision. To achieve this goal, a small carbon layer was sputtered onto an aluminium oxide plate. The carbon layer ensures a high, uniform resistance on the whole detection area. An electron pulse will hit the carbon layer and create an electric potential. The resulting voltage can then be measured at 3 to 6 readouts at the edges of the carbon layer. Afterwards, the position of the event can be reconstructed via triangulation by comparing the different voltage curves at the readouts.

To simplify the calculation and reduce the distortion effects, the PSA was built to have a linear dependence between the distance (between events and readouts) and the resistance of the layer. An infinite sheet of uniform resistance would generate the desired linearity, but for a finite sheet we need to consider boundary effects. To nevertheless achieve the linearity, an analogy of Gauss's Law can be used:

1. An infinite sheet of a uniform material (with resistivity r) will have a linear dependence between distance and resistance.
2. If there are holes in this sheet (with radius a) which have a line resistance of $R = \frac{r}{a}$ at its boundaries then the linearity still holds.
3. The linearity still holds, if the holes "cut out" a finite part of the infinite sheet.

There exist designs with round boundaries which fulfil this condition, but they have a considerably smaller detection area. To optimize both, the limit of $a \to \infty$ was considered. This also means that $R \to 0$. However, the linearity will still hold to a good approximation if we have a reasonably high resistance ratio $Z = \frac{r}{R}$ [3]. In the improved design, the ratio is approximately $Z \approx 20$ which holds in this approximation.

2 Preparation

2.1 Cabel Holder

A detailed description of the PSA can be found in Kunath's thesis [2]. The anode was designed to be implemented in the pre-existing MCP set up. This implementation was possible without further complications. However, a direct connection of the cables to the small copper layers on the back of the anode is not possible. The copper layers are too thin to attach the cables by using solder or silver glue. The connection would be too fragile and small movements could either detach the cable or the copper from the plate.

The solution was to build a cable holder as an additional structure right next to the PSA (figure 2). The copper layer and the cable holder are connected via small copper plates. These plates are C-shaped and lightly pressed onto the copper layers of the PSA and fixed with silver glue to ensure a stable connection.

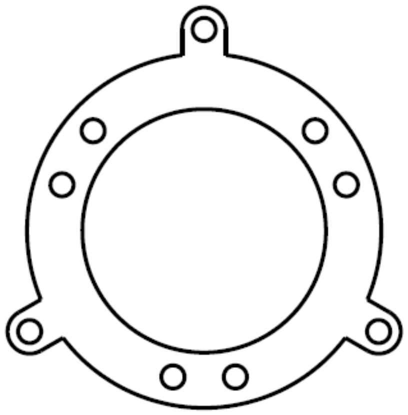


Figure 1: Concept of the cable holder.

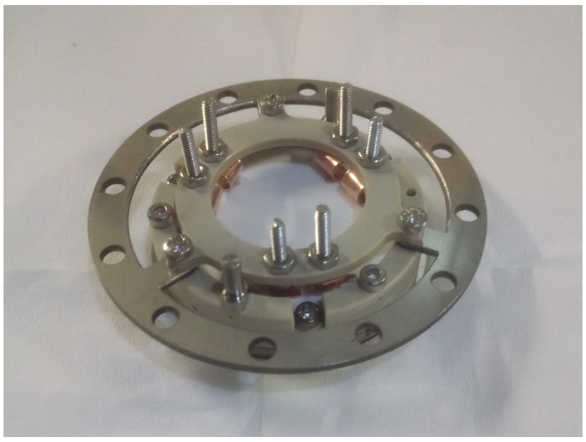


Figure 2: Implemented cable holder with C-shaped copper plates glued to the PSA.

2.2 Pinhole mask

To test the PSA, a pinhole mask was placed between the incoming beam and the MCP. With this set up, events can only occur under the pinholes. After the triangulation, the shape of the masking structure should be reconstructed. This set up allows to test the triangulation algorithm as well as distortion effects. During the experiments two different types of pinhole masks were used, which can be seen in figure 3 and 4. A detailed description of the masks can be found in the appendix.

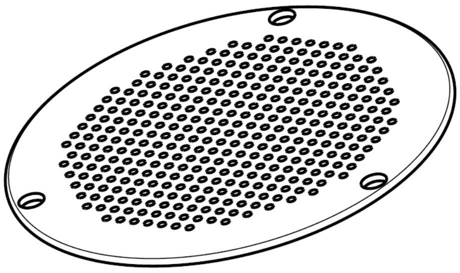


Figure 3: Uniform pinhole mask.

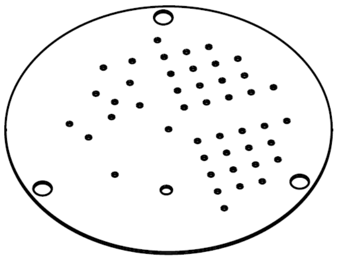
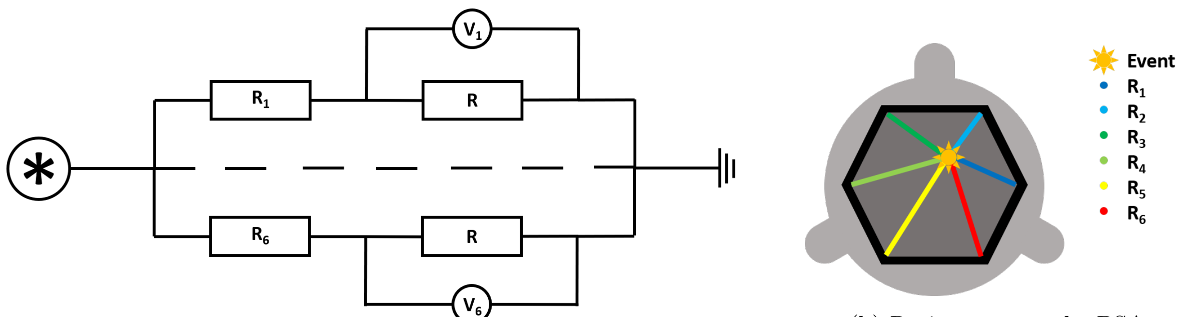


Figure 4: Differentiated pinhole mask.

2.3 Distance measurement

The PSA has six readouts on which the voltage pulses of an event can be measured. In the evaluation of the measurements, the linear resistance on the carbon layer of the anode is used to reconstruct the position of the event via triangulation. The circuit in figure 5a represents the readout schematics. Resistance R_1 to R_6 are the internal resistances of the PSA produced by the carbon layer. The voltages V_1 to V_6 are measured with oscilloscopes connected to the readouts where R is the internal resistance of the measurement device. V_{cc} is the total voltage over one read out and has the same value for all readouts. The following relation holds:



(a) Schematic circuit

(b) Resistances on the PSA

Figure 5: Conceptual circuit of the PSA. The resistance 1 to 6 are from the carbon layer of the anode. The resistance R is the internal resistance of the oscilloscopes.

For the triangulation the distance between the event and each readout needs to be calculated. The formula for the distances l_i can be derived from the current, voltage, resistance relation $V = RI$.

$$I_i = \frac{V_{cc}}{R_i + R} = \frac{V_i}{R} \tag{1}$$

$$R_i = \left(\frac{V_{cc}}{V_i} - 1\right) R \tag{2}$$

To arrive at l_i the linear resistance κ with $[\kappa] = \frac{m}{\Omega}$ needs to be measured. With this constant we arrive at:

$$l_i = \kappa R_i = \kappa \left(\frac{V_{cc}}{V_i} - 1\right) R \tag{3}$$

The hit position of the event is reconstructed via triangulation [4]. This is done by defining the residual vector as the difference between the distance measurement d_i and the reconstructed distance and minimising it, using the least square method.

$$f(x, y) = l_i^2 - (x - x_i)^2 - (y - y_i)^2 \tag{4}$$

Where (x_i, y_i) is the position of the i^{th} readout and (x, y) is the position of the event. This is a six-dimensional vector with a minimum at the position of the event.

3 Experiment

The linearity of the carbon layer is sensitive to irregularities and scratches and thus it is not possible to measure the linearity on the layer itself without damaging the PSA. To calculate the β coefficient one can measure the resistance between the readouts and compare it with the distances. The measurements can be seen in table 2. This measurement was taken before the PSA was used in an experiment.

	1	2	3	4	5	6
dist [mm]	x	14.78	25.6	29.56	25.6	14.78

Table 1: Distance from readout 1, symmetric for all other readouts

	1	2	3	4	5	6
1	x	1.07	1.74	1.95	1.87	1.44
2		x	1.12	1.68	1.93	1.88
3			x	0.95	1.54	1.90
4				x	0.87	1.57
5					x	1.00
6						x

Table 2: Resistance between readouts in kΩ before the anode was used

	1	2	3	4	5	6
1	x	1.3	180	∞	66	66
2		x	180	∞	67	66
3			x	∞	180	181
4				x	∞	∞
5					x	1.1
6						x

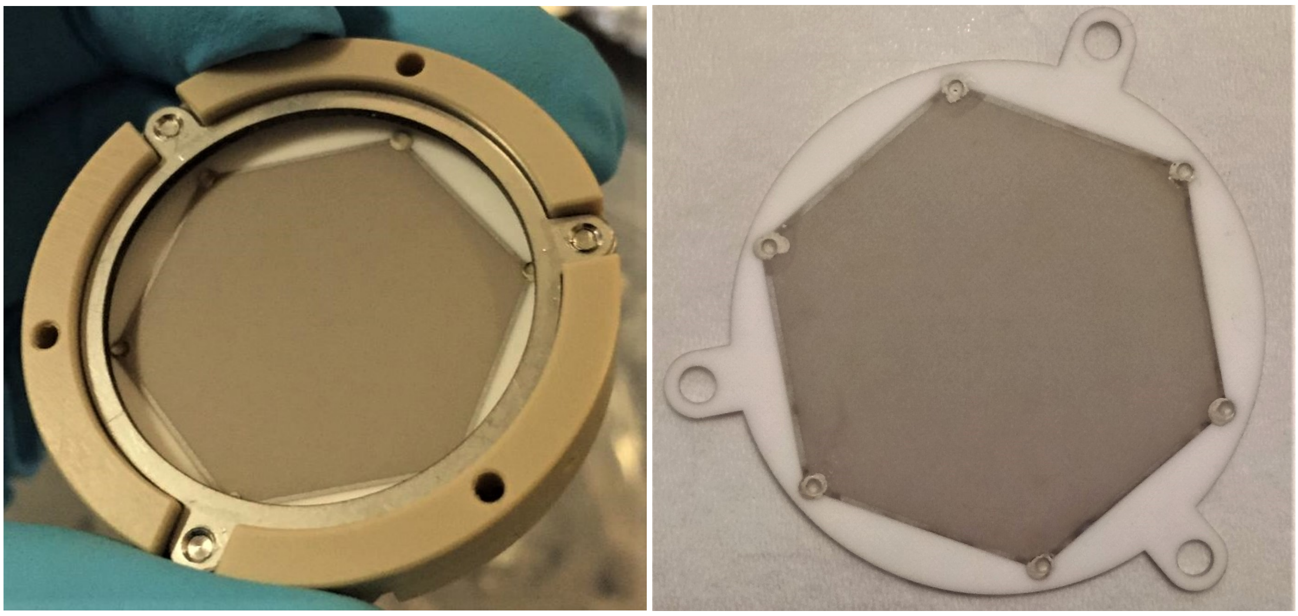
Table 3: Resistance between readouts in kΩ after the anode was used

The distances are symmetric for all other readouts. This leads to a beta value of:

$$\kappa = 14.72 \pm 0.72 \ \frac{mm}{k\Omega} \tag{5}$$

As a next step, the PSA was used in the experiment described in chapter 7.1.3. Approximately 100'000 measurements were taken and the PSA was exposed to a positron beam for 100 hours. Afterwards the linear resistance was measured again. The result can be seen in table 3.

After the anode was used, the linear resistance changed by different amounts per readout. After this measurement, the PSA was demounted and analysed. On a closer look, the carbon layer and the tungsten boundary had discoloured patches near the readouts. The differences can be seen in figure 6.



(a) Before the experiment

(b) After the experiment

Figure 6: PSA before and after the experiment with visible discolourations after the experiment

The discolourations indicate that the carbon layer was damaged (greatly increasing the resistance) and even disconnected the readout 4. The same holds for the tungsten boundary. A possible explanation for the damage is events that occur at the edge of the PSA. During the experiments, events with up to 200 mV where measured. If such an event would occur less than 1 mm from a readout, the resulting current would be:

$$I = \frac{V}{R} = \frac{\kappa}{d} V \approx 3 \ mA \tag{6}$$

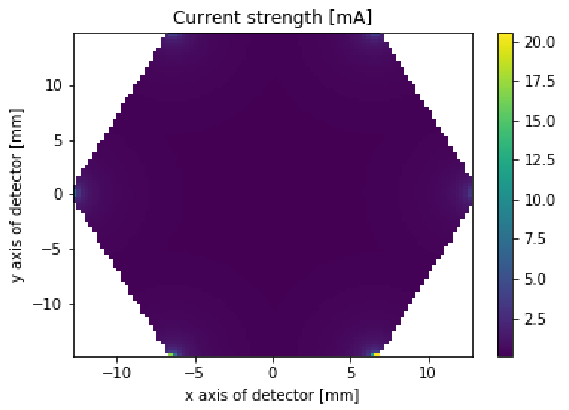
This is a high current for the very thin carbon layer, thus it is possible that many such events have caused the damage on the PSA.

4 Simulation

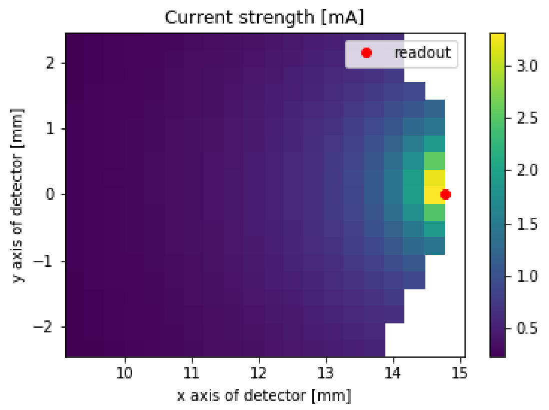
To improve the design of the PSA a simulation was developed. The goal was to find the maximum current on different parts on the carbon layer. Simultaneously the triangulation algorithm was tested by assuming a perfectly working anode and adding a Gaussian noise $\mathcal{N}(0, 0.0005)$ on the simulation which corresponds to the noise measured during the experiments (see chapter 7.1.1, 7.1.2 and 7.1.3).

4.1 Maximum Current

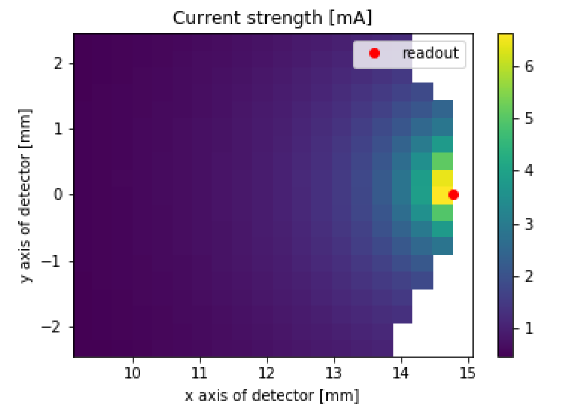
To calculate the maximum current at different parts of the anode, a program was used to simulate events randomly distributed over the whole anode. Then the maximal current was calculated by assuming a linear resistance of the carbon layer according to equation 5. The simulation was done using different strengths for the incoming signal. The results can be seen in figure 7.



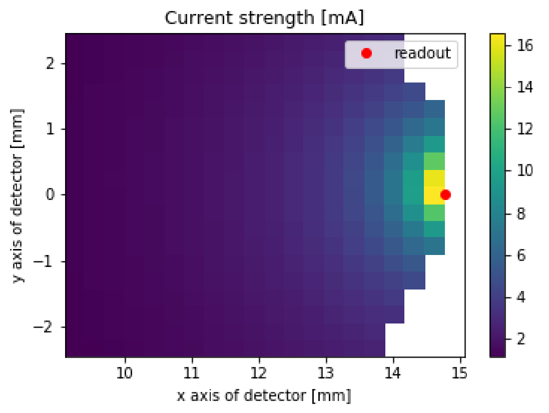
(a) Whole PSA with Signal 0.1 V



(b) Signal: 0.1 V



(c) Signal: 0.2 V



(d) Signal: 0.5 V

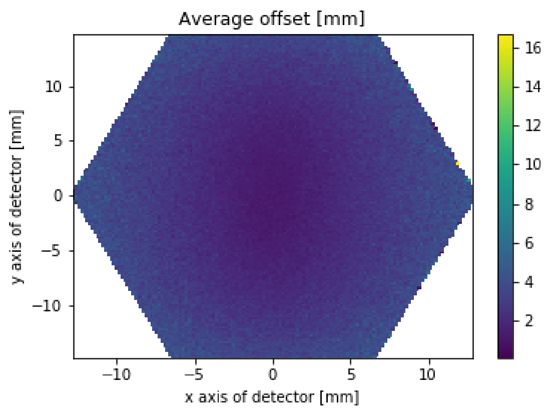
Figure 7: Simulation of the maximum current with 10^6 events

The overview in figure 7a shows that the inner region of the PSA has a large enough resistance and will not be problematic. In figures 7b, 7c and 7d the right most readout is shown. During the experiments, a typical signal strength of 0.2 V was measured. The readouts were damaged 1 mm around the readouts. Figure 7c shows that this corresponds to a current of approximately 2.5 mA.

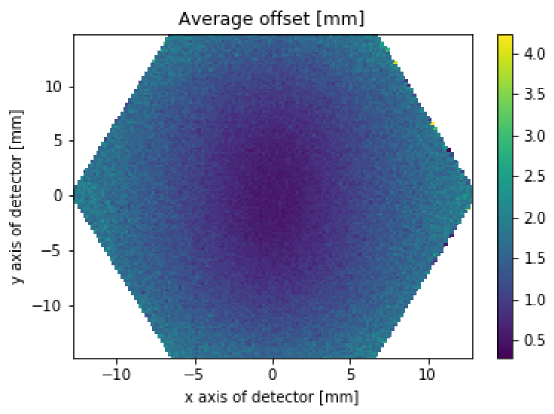
In the simulations with a signal strength of 0.1 V, the current only exceeds 2.5 mA if the events hit the PSA closer than 0.5 mm. Because the mounts for the readouts are approximately of this size, the events will not hit the carbon layer directly and thus be unproblematic for the PSA. In the simulations with a signal strength of 0.5 V, the current exceeds 2.5 mA in a larger region around the readouts. This will cause damage to an even larger area of the carbon layer.

4.2 Triangulation

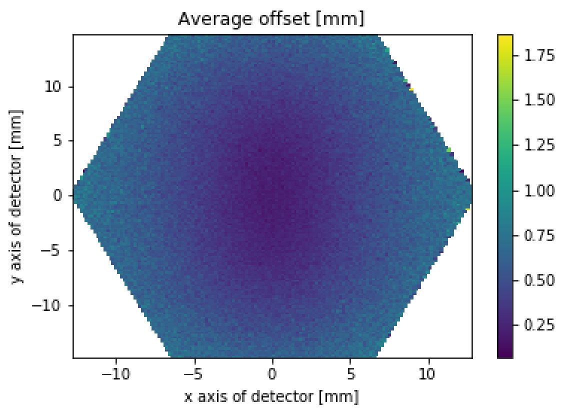
To test the accuracy of the triangulation, the same event generator as for the maximum current was used. The events then were used to generate a simulated data set as it would be produced in an experiment. A Gaussian error was added to the data according to the noise encountered during the experiments. Then the position of the events was reconstructed using the triangulation algorithm. The real position of the event could then be compared with the reconstructed one. Over 10^6 simulations, the average offset of the triangulation algorithm at different regions of the anode could then be derived. The results for an initial signal strength of 0.1 V, 0.2 V and 0.5 V can be seen in figure 8.



(a) Signal: 0.1 V



(b) Signal: 0.2 V



(c) Signal: 0.5 V

Figure 8: Simulation of the precision of the triangulation algorithm with 10^6 events

Figure 8b corresponds to the maximal signal value encountered during the experiments. The offset is smaller in the middle region and gets larger towards the readouts. This was expected because an event closer to one readout has a weaker signal in all others and thus the signal to noise ratio is smaller. The values range from 0.5 mm offset in the central region to 3 mm close to the readouts.

For signals of 0.1 V were at the lower end of our threshold. As one can see, even in the central region the triangulation has an average offset of approximately 2 mm. In the outer region the offset is around 6 mm and thus a position determination is not accurate. In the case of signals with an initial strength of 0.5 V, the offset is less than 1.5 mm on the whole PSA, except very close to the readouts. Thus, it is possible to reconstruct events with such a strong signal via triangulation.

5 Conclusion

The current design of the position sensitive anode did not work. The currents on the carbon layer, caused by the events one wishes to measure, generate large enough currents to cause damage by overheating. Thus, the PSA needs to be redesigned with materials which can withstand the created currents. The simulation shows that this current can get as high as 4 mA.

With the simulations, the conditions for a new design were found. With the current noise level, a signal of 0.5 V or higher would be needed to successfully determine the position via triangulation. Such a signal would cause damage to the carbon layer. Thus, it is not possible to adjust the signal strength to use the PSA. Either the signal is to weak and the triangulation is not precise enough, or it is to strong and will cause damage to the PSA.

6 Outlook

The theory behind the PSA should still hold, but the material flaws need to be corrected. The carbon layer and the tungsten would preferably withstand currents of up to 20 mA. This would allow to increase the signal strength and thus enables a better triangulation. The following considerations would be a good start for an improved PSA design:

- Design a larger boundary. Tungsten is known to be heat resistance and thus a good material choice, but the experiments showed that it is too thin. A larger boundary would make the material more heat resistance without losing the linearity of the carbon layer.
- Design a thicker carbon layer. Not only the tungsten but also the carbon layer was damaged by overheating. A thicker carbon layer would make it more heat resistance, but it would also have a negative influence on the linearity. A thicker carbon layer has more irregularities and thus would create more distortion effects.
- Reduce the active area. By implementing a circular mask between the MCP and the PSA one could eliminate dangerous events close to the readouts. This means also a smaller detection area. If the radius of the mask was chosen to eliminate all events closer than 2 mm to the event, one would lose 9.6% active area.
- Reduce the noise. With a lower noise, the triangulation would still be successful even with a lower signal strength. The simulation with a ten times lower noise and a signal of 0.1 V can be seen in figure 9. In this case, the errors would be at the same level as using a 0.5 V signal strength with the current set up. The source of the noise and how to reduce it is not yet determined.

Overall the PSA would be a great improvement for all measurements relying on an MCP. With an improved design and experimental set up it should be possible to use the PSA as intended. The simulation showed, if the anode could resist slightly stronger signals it could reconstruct the position of events generated by the MCP. If the anode is strong enough, an amplification of the event signal could increase the precision even further. Thus, even if this design failed, the PSA is promising and should be progressed.

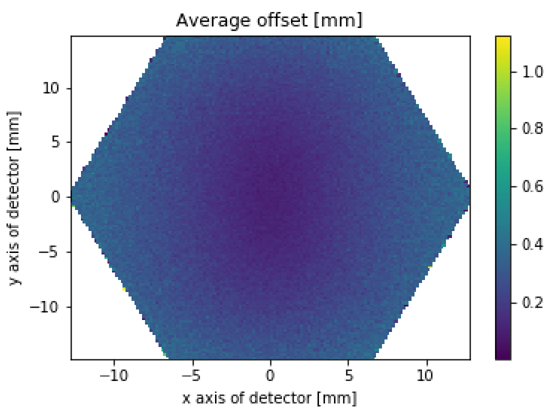


Figure 9: Simulation of 10^6 events with a signal of 0.2 V and a ten times smaller noise.

Acknowledgements

I would like to thank Lars Gerchow for his support during the whole project. He created a work environment in which I could learn and execute most of the steps by myself. In the end it was thanks to his detailed introductions and trust in my skills that i got hands on experience on the set up and testing of accelerator experiments.

Furthermore, I want to thank Vanessa Sennrich for her help with the simulations. Only because of her help, it was possible to develop and use the simulations.

Finally, I want to thank Dr. Paolo Crivelli for his engagement to find a project which perfectly fitted my interests. Because of him I was able to explore many different aspects of particle physics.

References

[1] J. Ladislas Wiza, "Microchannel plate detectors," *Nuclear Instruments and Methods*, vol. 162, no. 1, pp. 587–601, Jun. 1979. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0029554X79907341

[2] J. R. Kunath, "Design for a Position Sensitive Microchannel Plate," Sep. 2018.

[3] X.-D. Ju, M.-Y. Dong, Y.-C. Zhao, C.-X. Zhou, and O.-Y. Qun, "Design and optimization of resistive anode for a two-dimensional imaging GEM detector," *Chinese Physics C*, vol. 40, no. 8, p. 086004, 2016. [Online]. Available: http://stacks.iop.org/1674-1137/40/i=8/a=086004

[4] Y. Wang, and, and a. L. Cuthbert, "Bluetooth positioning using RSSI and triangulation methods," in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, Jan. 2013, pp. 837–842.

7 Appendix

7.1 Experimental Set Up

Before the failure of the PSA was found, three different experiments were conducted. The same PSA and MCP was used for all three experiments. A second Anode and a second MCP were installed during the third experiment to measure the κ value. It is not known when the carbon layer of the first anode started to get damage and thus the state of the anode during the first two experiments remains unclear.

7.1.1 Dark Counts

For a first measurement, the MCP with the built in PSA was placed inside a simple vacuum tube. This was done to test the set-up, improve the python code and to collect a first set of data. The data could have been used a dark correction in the images taken if the PSA worked properly. A schematic of the set up can be seen in figure 10.

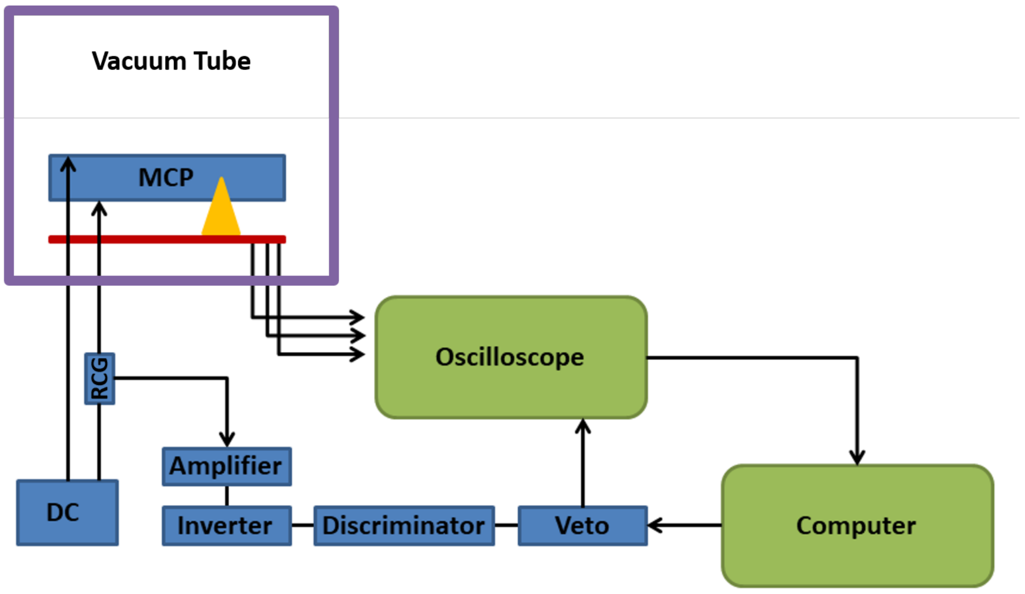


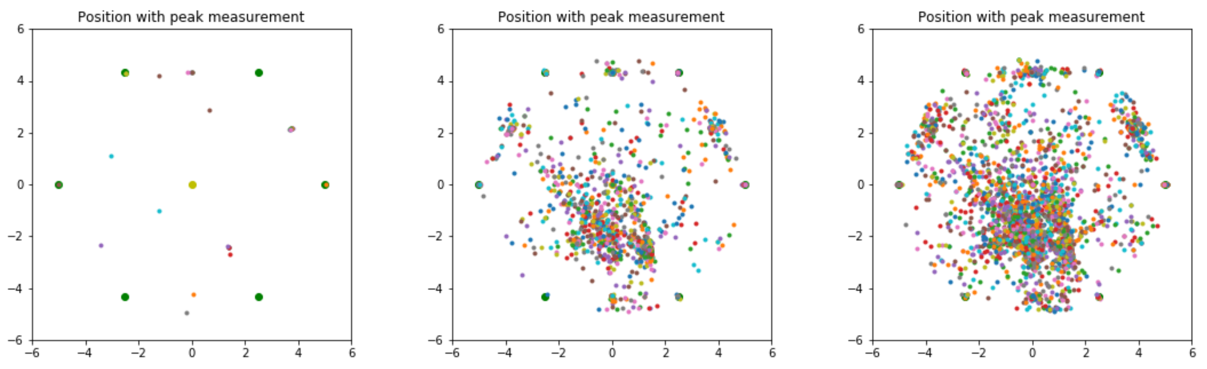
Figure 10: Schematic set up for the dark counts.

To collect the dark count data, the evaluation program was run three times: once with 250 events, once with 3'500 and once with 10'000. The positions of the events were recreated afterwards with a first version of the triangulation algorithm. This program used the simplified relation of resistance and distance:

$$R_i \propto \left(\frac{1}{V_i}\right) \tag{7}$$

Even though this relation is not precise, it was useful to check the functionalities of the python code and the set-up, before it was implemented in a more advanced experiment including a positron source. The position recreation of the measurements can be seen in figure 11. A single event is imaged only if at least three of the readouts had a signal to noise of at least 1.5. At least three distinguishable readout measurements are needed to perform the triangulation.

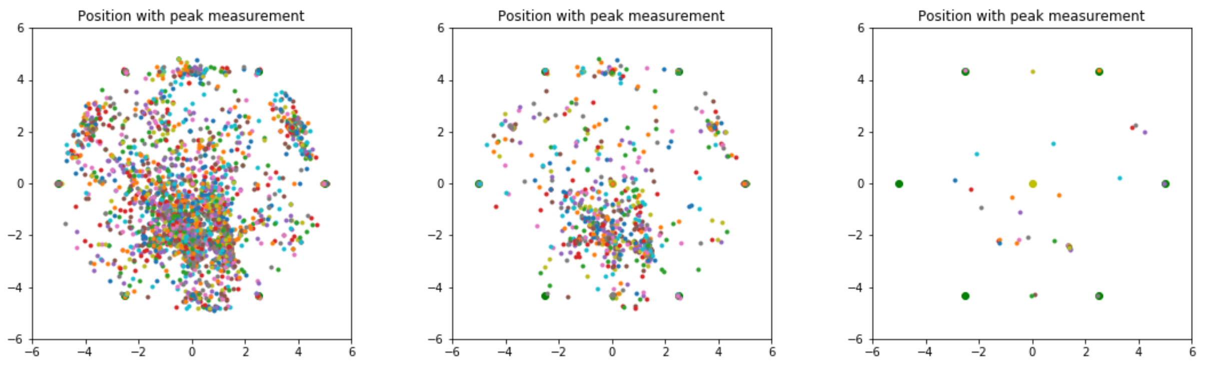
Events which lead to a dark count can happen anywhere in the MCP. If they occur closer to the PSA the signal will be significantly weaker. In our test ~40% of the events were too weak to create a detectable signal on at least 3 readouts. This was largely due to a higher background noise than previously expected. The noise can be explained by the 6 readouts which all create oscillation in their cables. These backscatter to the PSA, creating a higher background noise. Additionally, the signal gets split into 6 parts which further reduces the signal. Because the dark counts are generated randomly (and in various layers of the MCP), they cannot be used to improve the triangulation algorithm nor detect distortion effects caused by the PSA. To detect these, a source is needed to image the pinhole mask in front of the MCP.



(a) 250 events (b) 3'500 events (c) 10'000 events

Figure 11: Calculated position with a first version of the algorithm.

Additionally, the influence of different number of valid readouts was analysed. One could assume that a measurement with only 3 detectable signals is more influenced by errors then a measurement with 5 detectable readouts. To analyse this with the dark count data taken, only points with a certain amount of valid readouts were plotted. Because events with 5 detectable signals are rare, the measurement with 10'000 points was chosen to be evaluated. The result can be seen in figure 12. For at least 3 detectable signals, 38.93% of the events were filtered out. For 4 detectable signals 86.48% and for 5 detectable signals 99.58%.



(a) 3 valid readouts (b) 4 valid readouts (c) 5 valid readouts

Figure 12: calculation with different number of minimal valid readouts required

All 3 plots show similar distributions and thus it was assumed that 3 detectable signals are enough to give a good position estimation in the case of the dark counts. Points near the readouts are more likely to have only a few valid read outs (because the signal is weaker for the other readouts) and thus are less likely to give more than 3 detectable signals. This analysis is only a first approximation for the real experiment and cannot yet be used to conclude the properties of the PSA.

7.1.2 Simple Radioactive Source

The dark count set up was useful for a first testing of the MCP and the PSA but not to test the algorithm and image the pinhole mask. To further improve the triangulation, a source is needed. In this second set up, a simple radioactive positron source, 22Na, A 10MBq, paired with tungsten mesh moderator was placed before the pinhole mask. In the end the holes of the mask should appear in the reconstructed image of the events. If they are blurred or distorted, the algorithm needs to be improved.

The Set up for the simple radioactive source can be seen in figure 13. The source is placed 10cm before the MCP. A tungsten grid is placed in front of the source. Both gutters can be charged and create a very small accelerator. Additionally, this set up spreads the positrons from the source over the a larger area of the MCP. The evaluation of the data from the simple radioactive source can be seen in figure 14.

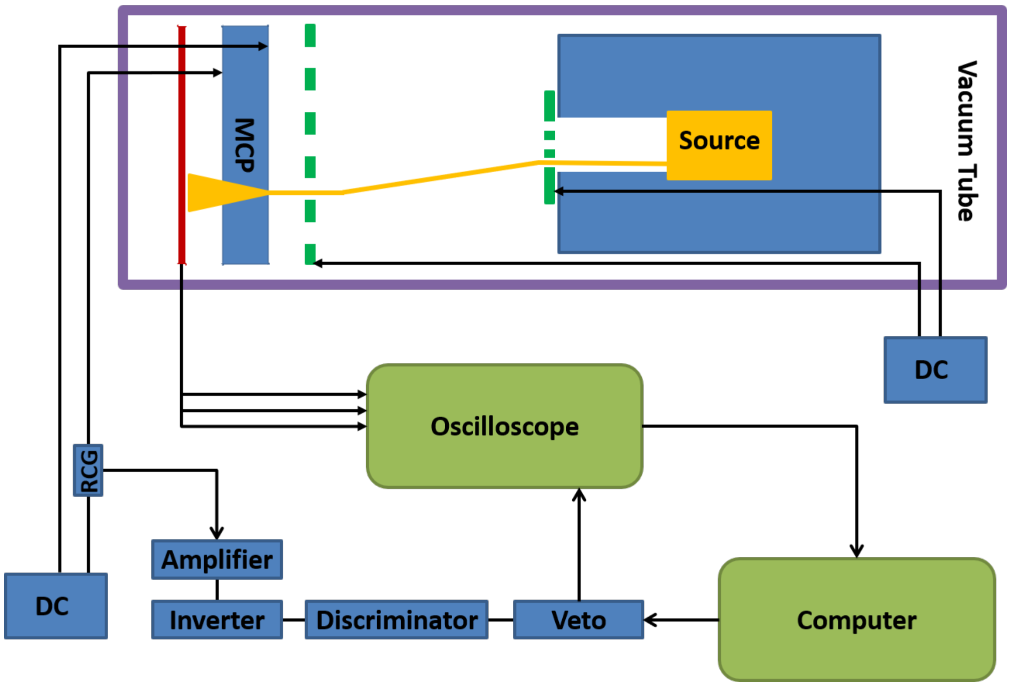


Figure 13: Schematic set up for the test with a simple radioactive source.

A line in the lower part of figure 14 can be seen which has a lot more counts then the rest. By comparing this result with the one from section 7.1.1, one finds that this region has many dark counts. An additional source for this error could be the cable to the pinhole mask. As shown in figure 13, the cable passes the accelerator area and if not perfectly aligned at the wall can influence the result.

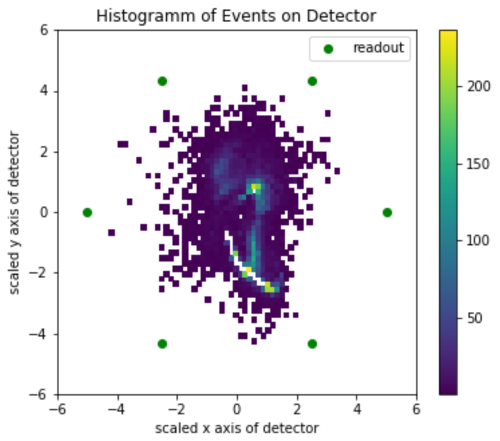


Figure 14: Evaluation of the first simple source. Only points with a detectable signal on at least 5 readouts were considered.

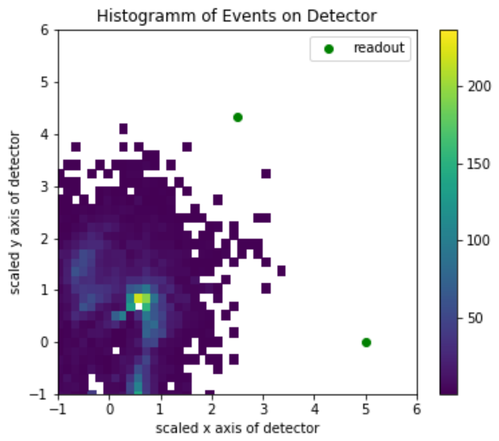


Figure 15: Same Histogram as in but zoomed on the upper right part and a upper limit for the bin values was chosen to remove the error source.

Compared to the dark count set up, a lot more events were detected in the upper part of the detector. In a next step only a section of the upper left part was considered to see if the gutter structure of the pinhole mask could be seen, but figure 15 shows no such structure. This can be explained by the source emitting fast positrons which can interact with the aluminium and create detectable photons. Thus, it is not possible to make further corrections in the triangulation algorithm.

7.1.3 Particle Accelerator

To improve the set up from the last experiment, a particle accelerator was chosen instead of a simple radioactive source. This eliminates fast positrons and thus should allow to image the structure of the pinhole mask. The set up can be seen in figure 16. Additionally, the pinhole mask was changed to the second version as shown in figure 4.

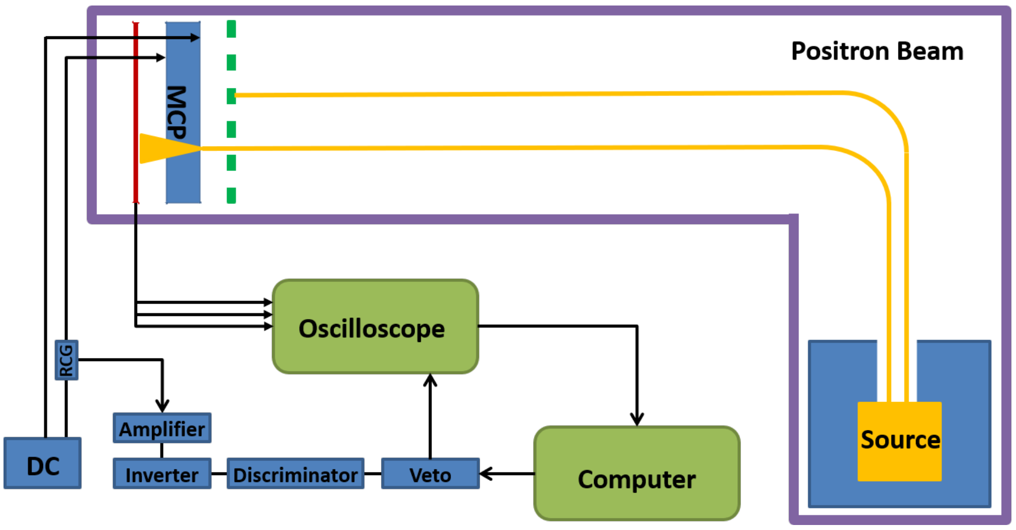
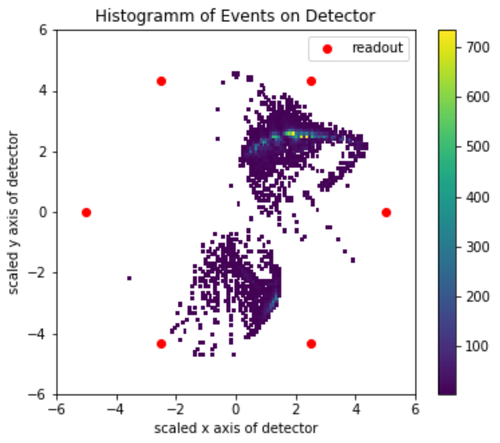
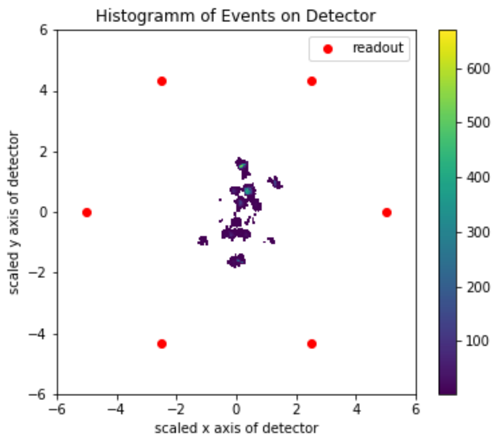


Figure 16: Schematic set up for the test with a particle accelerator.

After the first 52'000 measurement points the data was evaluated with the simple triangulation algorithm and the improved version. The differences between the reconstructed events can be seen in figure 17a.



(a) Simple algorithm



(b) Improved algorithm

Figure 17: Evaluation of 52'000 data points with different algorithms.

At this point the linear resistance of the anode was checked. The results can be seen in table 4. Especially the infinity resistance values are a strong hint that the anode was damaged. Because the κ value of this anode was not measured before the experiments it was not clear if the experiment itself damaged the anode or if it was a manufacturing flaw. To test this, the κ of a second anode was measured, then used in the particle accelerator set up, then measured again. A detailed description of this measurements can be found in chapter 3.

	1	2	3	4	5	6
1	x	∞	11.5	9.54	7.48	5.12
2		x	∞	∞	∞	∞
3			x	1.98	4.04	6.41
4				x	2.06	4.43
5					x	2.37
6						x

Table 4: Resistance between readouts in kΩ of the first anode used in all previous experiments

7.2 CATIA drawings

During the experiments different additional parts had to be created. The design process is described in chapter 2.2 and 2.1.

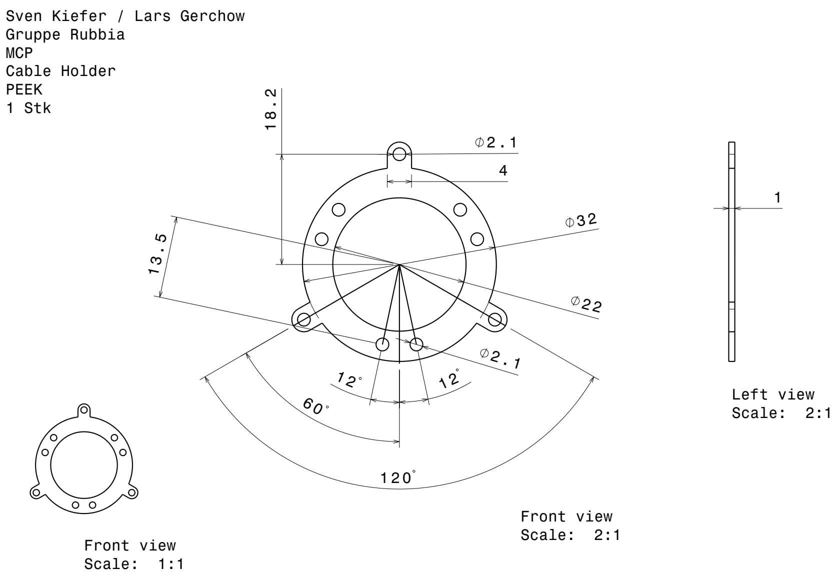


Figure 18: CATIA drawing of the cable holder.

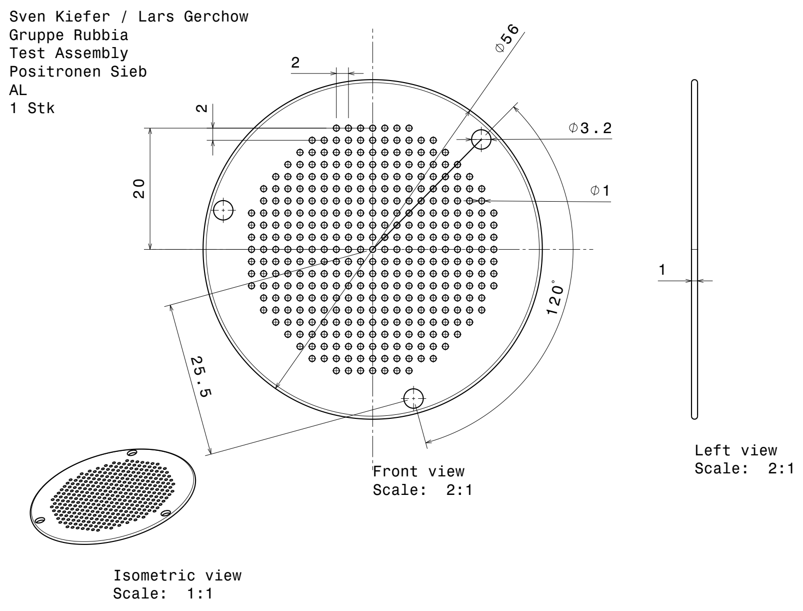


Figure 19: CATIA drawing of the pinhole mask.

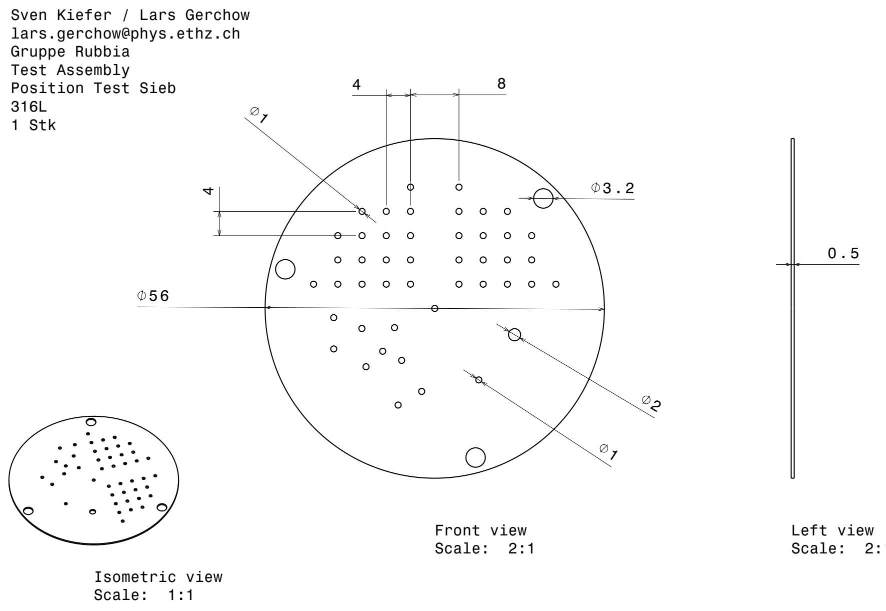


Figure 20: CATIA drawing of the pinhole mask.

7.3 Pyhton Code

Figure 21: Python functions for the triangulation.

.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 12 09:04:16 2019

@author: Sven Kiefer
File: PMCP.py
"""

import numpy as np
import scipy.optimize as so
import matplotlib.pylab as plt



#----------------------------------------------- Part 1: Additional Functions


def __Residual_Vector(_loc, _peak, _max, _readout_loc, _linear_res,
                      _oscilo_res, _min_signal):
    # Generates the residual Vector which is to be minimized,
    # ignors not detectable signals
    # :I: Intensitys at readouts [V]
    # :readout_Loc: Locations of the readouts [m]
    # :loc: current Location of the particle [m]
    # :linear_res: Sheet resistance of the anode [Ohm/m]
    # :oscilo_res: internal resistance of the osciloscope [Ohm]
    # :min_signal: Minimum signal at a readout to count as detection [V]
    # :return Fr: residual vectors [m^2]

    d_meas = (_max/_peak - 1)*_oscilo_res/_linear_res

    Fr = ((d_meas)**2 - (_readout_loc[:,0] - _loc[0])**2 -
          (_readout_loc[:,1] - _loc[1])**2)*(_peak > _min_signal)
    return Fr



#----------------------------------------------- Part 2: Evaluation Function

def triangulation(data, total_signal, linear_res, oscilo_res, readout_loc,
                  min_signal = 5, min_readouts = 3, min_total_signal = 65, s
                  tart_loc = np.array([0,0]), hist_bins = 40, prints = True,
                  visualisations = True):
    # Takes a set of data and total_signal and calculates the position
    # of the events via triangulation
    # :data: n x k array, where n is the number of readouts, k the number of
    #        measurements with the peak value of the readouts  (V_i) [V]
    # :total_signal: k array with the peak value of incominc signal (V_cc) [V]
    # :linear_res: linear resistance of the anode [Ohm/m]
    # :oscilo_res: Internal resistance of the Osciloscope [Ohm]
    # :readout_loc: Position of the n readouts as n x (x,y) array [m]
    # :min_signal: Minimum signal at a readout to count as detection [V]
    # :min_readouts: Minimum number of readouts with a detection to do the
    #                triangulation (must be >= 3) []
    # :min_signal_strength: Threshhold for incoming signals [V]
    # :start_loc: Initial condition for the least square methode, 2 array [m]
    # :hist_bins: Number of bins squared in the histogram
    # :prints: prints information about the triangulation if true
    # :visualisations: plots usefull information about the triangulation
    # :return pos_data: Positions of the events found via triangulation

    if prints:
        print("Triangulation Started")
        print("---------------------")




    #Step1: Filter out events which have a signal too weak
    if prints:
        print("STATE: Filtering of the Signal")

    mask_1 = total_signal >= min_total_signal
    mask_2 = (np.sum(data[:,:] >= min_signal, axis=0) >= min_signal)* mask_1
    if prints:
        print("    Percantage filtered out: " +
              str(100 - 100*np.sum(mask_2)/len(data[0,:])) + "%")
    r_data = data[:,mask_2]
    r_maxv = total_signal[mask_2]




    #Step2: Calculation of position with least squares
    if prints:
        print("STATE: Triangulation")

    pos_data = np.zeros((len(r_data[0,:]), 2))
    for i in range(len(r_data[0,:])):
        pos = so.least_squares(__Residual_Vector, start_loc,
                               args=(r_data[:,i], r_maxv[i], readout_loc,
                                     linear_res, oscilo_res, min_signal))
        pos_data[i,0] = pos.x[0]
        pos_data[i,1] = pos.x[1]




    #Step3: Plotting
    if visualisations:

        #overview of the result
        fig, axi = plt.subplots(2,2, figsize=(10,10))
        axi[0,0].set_title("Position with peak measurement")
        axi[0,0].set_xlabel('x axis of detector')
        axi[0,0].set_ylabel('y axis of detector')
        axi[0,0].plot(0,0,'yo')
        #plot readout position
        axi[0,0].plot(readout_loc[:,0], readout_loc[:,1], 'ro',label="readout")
        for i in range(len(pos_data[:,0])):
            #plot calculated position
            axi[0,0].plot(pos_data[i,0], pos_data[i,1], '.')
        axi[0,0].set_xlim(axi[0,0].dataLim.x0*1.05, axi[0,0].dataLim.x1*1.05)
        axi[0,0].set_ylim(axi[0,0].dataLim.y0*1.05, axi[0,0].dataLim.y1*1.05)
        axi[0,0].legend()

        axi[0,1].set_title("Histogramm of Events on Detector")
```

```
        axi[0,1].set_xlabel('x axis of detector')
        axi[0,1].set_ylabel('y axis of detector')
        axi[0,1].hist2d(pos_data[:,0], pos_data[:,1], bins = hist_bins, cmin=1)
        #plot readout position
        axi[0,1].plot(readout_loc[:,0], readout_loc[:,1], 'r.',label="readout")
        axi[0,1].set_xlim(axi[0,0].dataLim.x0*1.05, axi[0,0].dataLim.x1*1.05)
        axi[0,1].set_ylim(axi[0,0].dataLim.y0*1.05, axi[0,0].dataLim.y1*1.05)

        axi[1,0].set_title("Histogram of total signal strength")
        axi[1,0].hist(r_maxv)
        axi[1,0].set_xlabel("Total signal strength")

        axi[1,1].set_title("Correlation Histogram")
        axi[1,1].hist2d(r_maxv, np.sum(r_data,axis=0), cmin = 1, bins = 40)
        axi[1,1].set_xlabel("Total signal strength")
        axi[1,1].set_ylabel("Sum over all readouts")

        fig.savefig("Scatterplot.png")
        fig.show()


    if prints:
        print("END: Triangulation finished succesfully")

    return pos_data




#"""
```

Figure 22: Python script to take measurements. Needs an arduino and two tektronix TDS 2024c.

.

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 12 11:59:44 2019

@author: Sven Kiefer
File: Data_Collect.py
"""

import numpy as np
import matplotlib.pylab as plt

import time
import datetime
import visa
import serial
from struct import unpack




#------------------------------------------------  Additional
# To run this program, additional parts are needed:
#   1) An arduino connected via USB and running the correct running
#   2) Two Tektronix TDS 2024C connected via USB
#   3) Three folders in the same folder as this code, labeled:
#      Data, Pics, Temp
#
# The first Osciloscope (osc) should be connected to readout 1 - 3, the second
# should be connected to the readouts 4 - 6 in ascending order.

#------------------------------------------------  Variables

#Fix variables
#Instrument number of first osc
instr_name_1 = "USB0::0x0699::0x03A6::C046163::INSTR"
#Instrument number of second osc
instr_name_2 = "USB0::0x0699::0x03A6::C046149::INSTR"
ard_loc = 'com3'         #USB output of arduino

su_trig_val = "-0.4"     #Trigger value in Volt (not specifically needed)
su_trig_chan = "EXT"     #Trigger Channel

su_hor_pos = "-20E-8"    #Horizontal Position [s]
su_hor_scale = "5E-9"    #Horizontal Scale [s]

su_ch_probe = "1"        #probe value of channels
su_ch_pos = "2"          #position of channels in divs
su_ch_scale = "0.01"     #scale of channels [V]

su_chm_probe = "1"       #probe value of triger signal
su_chm_pos = "2"         #position of triger signal in divs
su_chm_scale = "0.1"     #scale of triger signal in [V]

data_per_run = 2507      #Nuber of data points per measurement (given by osc)
cut_of = 50              #rejected number of data points at start and end
```

```
#Edit variables

sig_filter = 90       #min signal on triger value to count as event
min_detect = 5        #min signal of channel to count as detection
min_read = 4          #min number of detections to count as event

skip_su = 1           #Skipp set up if true
visualizations = 1    #Show visualisation if true

nr_events = 10000     #Number of measurements taken
save_after = 2000     #Save after this measurement (carefull: effects runtime)




#-----------------------------------------------  Part 1: Set Up

print("STATE: Set Up")


#find instruments
ser = serial.Serial(ard_loc, 9600)
rm = visa.ResourceManager()
#print(rm.list_resources()) #find Instrument numbers
i1 = rm.open_resource(instr_name_1)
i2 = rm.open_resource(instr_name_2)


def Set_dev(dev):
    dev.write("ACQ:STATE OFF")

    #Trigger Settings
    dev.write("TRIG:MAI:EDGE:SLO FALL")
    dev.write("TRIG:MAI:MODE NORM")
    dev.write("TRIG:MAI:EDGE:SOU EXT")
    dev.write("TRIG:MAI:LEV " + su_trig_val)

    #Horizontal Settings
    dev.write("HOR:MAI:POS " + su_hor_pos)
    dev.write("HOR:MAI:SCA " + su_hor_scale)
    #Data settings
    dev.write("DAT:ENC RIB")
    dev.write("DAT:WID 1")

    #Verticals Settings
    def Set_CH(dev, i):
        dev.write("SEL:CH" + str(i) + " ON")
        dev.write("CH" + str(i) + ":PRO " + su_ch_probe)
        dev.write("CH" + str(i) + ":INV OFF")
        dev.write("CH" + str(i) + ":POS " + su_ch_pos)
        dev.write("CH" + str(i) + ":SCA " + su_ch_scale)

    for j in range(1,4):
        Set_CH(dev,j)
    #Trigger Channel Settings
    dev.write("SEL:CH4 ON")
    dev.write("CH4:PRO " + su_chm_probe)
    dev.write("CH4:INV ON")
    dev.write("CH4:POS " + su_chm_pos)
    dev.write("CH4:SCA " + su_chm_scale)



if (not skip_su):
    Set_dev(i1)
    Set_dev(i2)
    print("    Done")

else:
    print("    Skipped")






#-----------------------------------------------  Part 2: Collect data

print("STATE: Collecting Data")


#initaly turn data gathering off
i1.write("ACQ:STATE OFF")
i2.write("ACQ:STATE OFF")
i1.write("ACQ:STOPA SEQ")
i2.write("ACQ:STOPA SEQ")

#Variable set up
#array with: read_out / events / measuremnts
raw_data = np.zeros((8, nr_events, data_per_run))
m = 0


#initializing
start_time = time.time()
term_time = time.time()
i1.write("ACQ:STATE ON")   #Ready Osc1
i2.write("ACQ:STATE ON")   #Ready Osc2
ser.write(b'1')            #Deactivate Veto


#Gather data
while (m < nr_events):
    if (i1.query("TRIG:STATE?")[:4] == str("SAVE")):        #Check for trigger
        ser.write(b'0')                                     #Reactivate Veto
        for n in range(4):
            i1.write("DAT:SOU CH" + str(n+1))               #Set redout source
            i1.write("CURV?")                               #Prepare data
            mn_raw = i1.read_raw()                          #Read out data
            mn_len = len(mn_raw)                            #Get binary length
            new_data = np.array(unpack('b'*mn_len, mn_raw)) #Convert binary
            raw_data[n,m,:] = new_data[:]                   #Ad data to string

            i2.write("DAT:SOU CH" + str(n+1))               #Same as above
            i2.write("CURV?")
            mn_raw = i2.read_raw()
            mn_len = len(mn_raw)
            new_data = np.array(unpack('b'*mn_len, mn_raw))
            raw_data[n+4,m,:] = new_data[:]

        m += 1


        #Save inbetween to prevent data loss on system crash
        if m%save_after == 0:
            results_file = open("Temp/tempData_"
                + str(datetime.datetime.now().strftime("%Y-%m-%d-%H-%M"))
                + ".txt", "w")
            for ms in range(len(raw_data[0,:,0])):
                for rs in range(len(raw_data[:,0,0])):
                    for ds in range(len(raw_data[0,0,:])):
                        results_file.write(str(raw_data[rs,ms,ds]) + ",")
                    results_file.write("_")
                results_file.write("\n")

            results_file.close()

        #reactivate osc
        i1.write("ACQ:STATE ON")   #Ready Osz1
        i2.write("ACQ:STATE ON")   #Ready Osz2

        #visual
        print("    " + str(m) + ": Elaps time: "
              + str(time.time() - start_time))
        start_time = time.time()


    if(i1.query("ACQ:STATE?")[:-1] == str(1)
        and i2.query("ACQ:STATE?")[:-1] == str(1)):
        ser.write(b'1')            #Deactivate Veto



ser.close()  #Close Arduino Connection (if code crashed, execute this manualy)

print("    Total data collection time: " + str(time.time()-term_time))



#Save data to file
print("STATE: Save full Data")

results_file = open("Temp/rawData_"
                    + str(datetime.datetime.now().strftime("%Y-%m-%d-%H-%M"))
                    + ".txt", "w")
for m in range(len(raw_data[0,:,0])):
    for r in range(len(raw_data[:,0,0])):
        for d in range(len(raw_data[0,0,:])):
            results_file.write(str(raw_data[r,m,d]) + ",")
        results_file.write("_")
    results_file.write("\n")

results_file.close()
```

```python
#------------------------------------------------ Part 4: Calculate Peak Value
print("STATE: Peak calculation")


#Variable set up
nr_events = len(raw_data[0,:,0])

#Determin total signal strength with respect to background
max_val = np.zeros_like(raw_data[6,:,0])
for i in range(len(raw_data[7,:,0])):
    max_val[i] = abs(np.median(raw_data[7,i,cut_of:-cut_of])-
           np.min(raw_data[7,i,cut_of:-cut_of]))


#applie cut off to data
#array with: read_out / events / measuremnts
r_data = np.zeros((6, nr_events, 2507 - 2*cut_of))
r_data[:3,:,:] = raw_data[:3 ,:,cut_of:-cut_of]
r_data[3:,:,:] = raw_data[4:7,:,cut_of:-cut_of]


#first visualization of signals
prints = 10
if (visualizations):
    for k in range(prints):
        z = int(nr_events/prints)*k
        plt.figure()
        plt.title("Measurment "+ str(z)+"; peak: " +str(max_val[z]))
        for i in range(6):
            plt.plot(r_data[i,z,:])
        plt.savefig("Pics/Testplot_" + str(z) + ".png")
    plt.show()




#Peak detection
peak_n = np.zeros_like(r_data[:,:,0]) #array with: read_out / events
for i in range(len(r_data[:,0,0])):
    for j in range(len(r_data[0,:,0])):
        #calc peak with respect to background
        peak_n[i,j] = abs(np.median(r_data[i,j,cut_of:-cut_of])-
              np.min(r_data[i,j, cut_of:-cut_of]))




#--------------------------------------------------- Part 5: Save Peak Value
#This section was not tested yet
print("STATE: Save Peak Value")

peak_file = open("Data/peakData_"
                 + str(datetime.datetime.now().strftime("%Y-%m-%d-%H-%M"))
                 + ".txt", "w")
for m in range(len(peak_n[0,:])):
    peak_file.write(str(max_val[m]) + ",")
    for r in range(len(peak_n[:,0])):
        peak_file.write(str(peak_n[r,m]) + ",")
    peak_file.write("\n")

results_file.close()



print("END: Program finished succesfully")

#"""
```

Figure 23: Python script to evaluate the data taken with the script in figure 21.

.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 25 18:31:54 2018
e
@author: Sven Kiefer
File:Evaluation.py
"""
import numpy as np

import PMCP
import datetime
import glob


#---------------------------------------- variables
#Analysis variables
sig_filter = 65         #min signal on triger value to count as event
min_detect = 5          #min signal of chanal to count as measurement
min_read = 4            #min number of dist
linear_res = 1/14.7212  #linear resistance [kohm/mm]
osc_res = 0.05          #Internal resistance [kohm]
#position of redouts
readout_loc = np.array([[5,0], [2.5,4.33], [-2.5,4.33],
                        [-5,0], [-2.5,-4.33], [2.5,-4.33]])/10*29.56
start_loc = np.array([0,0]) #initial condition for least square


#Settings
collect_data = 1
    #if 1: all files from Data are collected and attached
    #if 2: a results file next to this code is loaded










#---------------------------------------------------- Part 1: Load Peak Data
#The load was not tested with this formating of text file
#if errors occure compare with save of Data_Collect

peak_data = np.empty((6,0))
maxv_data = np.empty((1,0))

if(collect_data):
    #load data from files
    print("STATE: Attache Data")

    path = glob.glob("Data/*.txt")
    for run in path:
        data_file = open(run,"r")
        nr_r = len(data_file.readlines())
        data_file.close()
        data_file = open(run,"r")
        nr_m = len(data_file.readline().split(",")) - 1
        data_file.close()
        data_file = open(run,"r")

        ro_data = np.zeros((6,0))
        line_data = np.zeros((6,1))
        r_count = -1
        for r in data_file:
            r_count += 1
            m_split = r.split(",")
            maxv_data = np.append(maxv_data, np.array([m_split[0]]))
            for m in range(1, len(m_split)-1):
                line_data[m-1,0] = float(m_split[m])
            ro_data = np.append(ro_data, line_data, axis=1)


        data_file.close()
        peak_data = np.append(peak_data, ro_data, axis=1)
        print("    Added number of measurements: " + str(ro_data.shape[1]))

    print("    Finished Collecting with Number of Events: "
          + str(peak_data.shape[1]))


    #Save data to file
    print("    Saving Combination...")

    results_file = open("Collection_"
                        + str(datetime.datetime.now().strftime("%Y-%m-%d-%H-%M"))
                        + ".txt", "w")
    for m in range(len(peak_data[0,:])):
        results_file.write(str(maxv_data[m]) + ",")
        for r in range(len(peak_data[:,0])):
            results_file.write(str(peak_data[r,m]) + ",")
        results_file.write("\n")

    results_file.close()



else:
    print("STATE: Load Data")
    #load data from one file
    data_file = open("results.txt","r")
    nr_r = len(data_file.readlines())
    data_file.close()
    data_file = open("results.txt","r")
    nr_m = len(data_file.readline().split(",")) - 1
    data_file.close()
    data_file = open("results.txt","r")

    ro_data = np.zeros((6, nr_r))
    r_count = -1
    for r in data_file:
        r_count += 1
        m_split = r.split(",")
        maxv_data = np.append(maxv_data, np.array([m_split[0]]))
        for m in range(1, len(m_split)-1):
            ro_data[m-1,0] = float(m_split[m])
    data_file.close()
    peak_data = ro_data

    print("    Finished Collecting with Number of Events: "
          + str(peak_data.shape[1]))




#--------------------------------------------------------- Part 2: Analysis
pos_data = PMCP.triangulation(peak_data, maxv_data, linear_res, osc_res,
                readout_loc, min_signal=min_detect, min_readouts=min_read,
                min_total_signal=sig_filter, start_loc=start_loc)



print("END: Program finished succesfully")
#"""
```

Figure 24: Python script used for the simulations in chapter 4.

.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 14 07:35:26 2019

@author: Sven Kiefer, Vanessa Sennrich
File: Simulations.py
"""

import numpy as np
import matplotlib.pylab as plt

import PMCP


#----------------------------------------------------- Variables

#fix variables
sig_filter = 0          #min signal on triger value to count as event
min_detect = 0          #min signal of chanal to count as measurement
min_read = 4            #min number of dist
osc_res = 0.05          #Internal resistance [kohm]
linear_res = 1/14.7212  #Linear resistance [kohm/mm]
#position of redouts [mm]
readout_loc = np.array([[5,0], [2.5,4.33], [-2.5,4.33],
                        [-5,0], [-2.5,-4.33], [2.5,-4.33]])/10*29.56


#edit variables
vcc_l_mc = np.array([0.1, 0.2, 0.5]) # incoming signal [V]
events_mc = 1000000
bins_mc = 100

vcc_l_ss = np.array([0.1, 0.2, 0.5]) # incoming signal [V]
events_ss = 1000000
bins_ss = 150
sigma_ss = 0.0005 #experimentaly measured error

#----------------------------------------------------- Part 1: Sim Function

def randhex(d, n):
    # creates an (n,2)-array of pseudo-random points located within a
    # regular hexagon of outer radius d
    # :d = length of each hexagon side
    # :n = amount of random points within hexagon created
    # return: (n,2)-array containing pseudo-random points inside the hexagon
    # to be solved: edges might not be included, numpy random.rand excludes 1

    #variables
    vecs = np.array([[-d,0.], [d*0.5, d*np.sqrt(3.)/2.],
                     [d*0.5, -d*np.sqrt(3.)/2.]])
    #three edges of the hexagon -> opposite points on the three parallelograms
    #that make up the regular hexagon
    para = np.random.randint(0,3,n) #random ints 0, 1, or 2
    x = np.random.rand(n,2) #random floats [0,1)

    #calculations
    #(2, n, 2)-array with: point//n//coord
    v = np.array([vecs[para,:], vecs[(para + 1)%3, :]])
    #(n,2)-array with: n//coord
    o = np.array([x[:,0]*v[0,:,0]+x[:,1]*v[1,:,0],
                  x[:,0]*v[0,:,1]+x[:,1]*v[1,:,1]]).T
    return o



def sim(_exactpos, _loccorner, _Vcc, _beta=1, _R=0.05, _sigma=1):
    # (n,m)-array of the simulated voltage at each readout (m, at each corner
    # of the anode) of each event (centered electron dribble onto the position
    # sensitive anode) with Gaussian errors
    # :_exactpos = (n,2)-array with: n//coord, contains the positions where
    #              an event takes place
    # :_loccorner = (e,2)-array with: e//coord, contains the location of each
    #               corner of the anode (readouts, measurements taken at each
    #               corner)
    # :_beta = constant, linear resistance, default = 1
    # :_R = internal resistance of the oscilloscope (measurement device)
    # :_sigma = constant, standard deviation, Gaussian error, simulating
    #           noise on data
    # :_Vcc = voltage, always within a range given by the #MCP?
    # return: v1 tilde aka a (n,m)-array with: n = number of events,
    #         m = number of readouts (corners), containing the simulated
    #         voltage at each readout for each event with a Gaussian error

    #variables
    (a,b) = _loccorner.shape
    (c,d) = _exactpos.shape
    d = np.zeros((c,a)) # a = number of corners, c = number of events
    #(c,a)-array, Gaussian distribution around 0 with standard deviation _sigma
    noise = np.random.normal(0, _sigma,[c,a])

    #calculations
    #d = (n,m)-array with: n = number of events, m = number of
    #corners/readouts; contains distance from event to each corner
    for i in range(a):
        #array containing the distance between the events and each readout
        d[:,i] = np.sqrt((_loccorner[i,0]-_exactpos[:,0])**2 +
         (_loccorner[i,1]-_exactpos[:,1])**2)
    r1 = _beta*d #calculation of the resistance, approx linear
    #simulated voltage at each readout for each event with noise
    v1_ = (_R/(_R + r1))*_Vcc + noise

    return v1_



def plotter3d_mean(_x, _y, _z, bins = 200):
    # 2d color plot with z avaraged over the bin
    # :_x = x coord (n-array)
    # :_y = y coord (n-array)
    # :_z = value at x,y (n-array)
    # :bins = number of bins (quadratic)
    # return: x grid coord, y grid coord, averaged z value

    #create grid
    b = bins
    x_min = np.min(_x)
    x_max = np.max(_x)
    x_step = (x_max - x_min)/b*1.000001
    y_min = np.min(_y)
    y_max = np.max(_y)
    y_step = (y_max - y_min)/b*1.000001

    grid = np.zeros((b,b))
    count = np.zeros((b,b))
    mask_x = ((_x - x_min)/x_step).astype(int)
    mask_y = ((_y - y_min)/y_step).astype(int)


    #set z values and avarage over them
    for i in range(len(_x)):
        grid[mask_x[i],mask_y[i]] += _z[i]
        count[mask_x[i],mask_y[i]] += 1

    #corrections
    count_mask = (count==0)
    count[count_mask] = 1
    grid_avg = grid/count
    grid_avg_mask = (grid_avg==0)
    grid_avg[grid_avg_mask] = np.NaN


    #plot
    plt.figure()
    plt.pcolormesh(np.linspace(y_min,y_max,b),
                   np.linspace(x_min,x_max,b), grid_avg.T)
    plt.xlabel("x axis of detector")
    plt.ylabel("y axis of detector")
    plt.colorbar()

    return np.linspace(x_min,x_max,b), np.linspace(y_min,y_max,b), grid_avg



#------------------------------------------------------- Part 2: Max Current

#generate random events
mc_events = randhex(readout_loc[0,0], events_mc)

#Min Resistance calculation
mc_d = np.zeros((events_mc,6))
for i in range(6):
    mc_d[:,i] = np.sqrt((readout_loc[i,0]-mc_events[:,0])**2 +
        (readout_loc[i,1]-mc_events[:,1])**2)

mc_minDis = np.min(mc_d, axis=1)
mc_minRes = mc_minDis*linear_res


#Max current calculation
for vcc in vcc_l_mc:
    #actal cur and plotter
    mc_maxCur = vcc/mc_minRes
    mc_x, mc_y, mc_grid = plotter3d_mean(mc_events[:,0], mc_events[:,1],
                                         mc_maxCur, bins_mc)

    #plot whole figure
    plt.figure()
    plt.title("Current strength [mA]")
    plt.pcolormesh(mc_y, mc_x, mc_grid.T)
    plt.xlabel("x axis of detector [mm]")
    plt.ylabel("y axis of detector [mm]")
    plt.colorbar()
    plt.savefig("MC_Whole_VCC_" + str(vcc) + ".png")

    #plot zoomed figure
    plt.figure()
    plt.title("Current strength [mA]")
    plt.pcolormesh(mc_x[-20:], mc_y[bins_mc//2 - 10 : bins_mc//2 + 10],
                   mc_grid[-20:,bins_mc//2 - 10 : bins_mc//2 + 10].T)
    plt.plot(readout_loc[0,0],readout_loc[0,1],'ro',label="readout")
    plt.xlabel("x axis of detector [mm]")
    plt.ylabel("y axis of detector [mm]")
    plt.legend()
    plt.colorbar()
    plt.savefig("MC_Zoom_VCC_" + str(vcc) + ".png")




#---------------------------------------------------- Part 2: Sigma simulation
vccs = vcc_l_ss[1]

#generate random events
ss_events = randhex(readout_loc[0,0], events_ss)

for vccs in vcc_l_ss:
    #generate readout data
    sim_data = sim(ss_events, readout_loc, vccs, linear_res, osc_res, sigma_ss)
    sim_maxv = np.ones_like(sim_data[:,1])*vccs

    #reconstruct position via triangulation
    rec_pos = PMCP.triangulation(sim_data.T, sim_maxv, linear_res, osc_res,
                readout_loc, min_signal=min_detect, min_readouts=min_read,
                min_total_signal=sig_filter, hist_bins=bins_ss,
                visualisations=False)

    #calculate distance between real and reconstructed event
    rec_dist = np.sqrt((ss_events[:,0]-rec_pos[:,0])**2 +
                       (ss_events[:,0]-rec_pos[:,0])**2)

    #plot the result
    ss_x, ss_y, ss_grid = plotter3d_mean(ss_events[:,0], ss_events[:,1],
                                         rec_dist, bins_ss)

    #plot whole figure
    plt.figure()
    plt.title("Average offset [mm]")
    plt.pcolormesh(ss_y, ss_x, ss_grid.T)
    plt.xlabel("x axis of detector [mm]")
    plt.ylabel("y axis of detector [mm]")
    plt.colorbar()
    plt.savefig("Sigma_VCC_" + str(vccs) + ".png")
```