XIOS HOGESCHOOL LIMBURG
DEPARTEMENT INDUSTRIËLE WETENSCHAPPEN EN TECHNOLOGIE

DATA ACQUISITION FOR MONOLITHIC SCINTILLATION DETECTORS IN PET APPLICATIONS

Robin CUYPERS

Afstudeerwerk ingediend tot het behalen van het diploma van
master in de industriële wetenschappen: nucleaire technologie

Promotoren: dr. J.M. Pérez Morales (CIEMAT)
dr. P. F. Rato Mendes (CIEMAT)
dr. E. Wieërs (XIOS Hogeschool Limburg)

Academiejaar 2008 - 2009

annihilation photon at the surface of the scintillation block. The functioning of those NNs will be discussed later. Both types of scintillation crystals are shown in Figure 3.

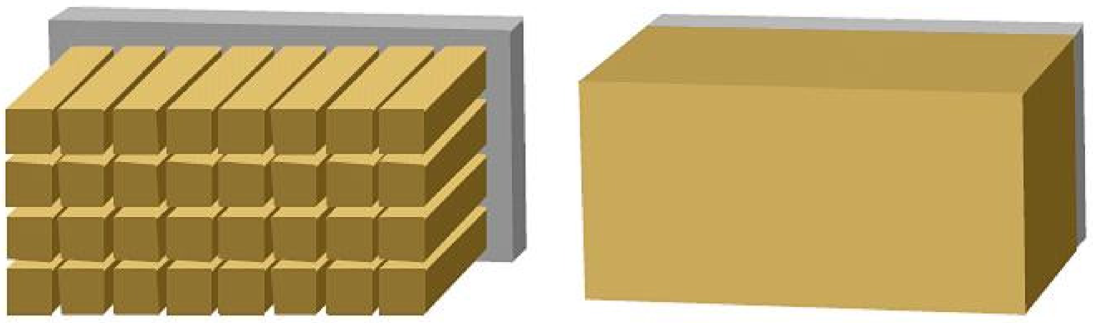


Figure 3: Left: pixilated scintillation crystal; Right: monolithic scintillation crystal [PéBP08].

When using the pixilated scintillation crystal the annihilation photons which interact between the different elements of the scintillation crystal will not be detected. This is because they interact with an isolator material that is placed between the different elements of the pixilated scintillation crystal. This material is placed there to ensure that light produced by one element cannot travel to another element of the crystal block, this to ensure that the calculation of the point of entrance of the photon can be done in a correct way. In Figure 3 this isolation material is not shown. Each element gives its center of gravity as coordinate of interaction. Because the light is trapped in a small volume, the concentration or intensity of the light will be high, making it possible to achieve high energy resolutions. On the other hand you will lose some spatial information due to the death space, when using pixilated scintillation crystals. The spatial data is not continuous, it changes with certain increments.

The monolithic scintillation crystal can detect incident annihilation photons on its entire volume, since there is no isolator material present. Because of this, the efficiency of the detection of annihilation photons will be higher when using monolithic scintillation crystals compared with the use of pixilated scintillation crystals. When looking at the spatial resolution that can be obtained from a scintillation crystal of this type we see that the spatial data will be more like an analog signal, its spatial information is continuous and as a result its spatial resolution may be higher.

So we see that every setup has its own advantages and drawbacks. The energy resolution of a detector build up out of non-monolithic scintillation crystals will be higher than a detector build up out of a monolithic scintillation crystal. On the other hand will the spatial resolution be higher for a monolithic scintillation crystal. The choice of scintillation crystal will depend on the application. In our application, clinical imaging of the human brain, we chose for a good spatial resolution, the monolithic scintillation crystal.

The incident annihilation photon is interacting with the detector crystal on a certain depth, the depth of interaction (DOI). This DOI can be an important issue for the calculation of the correct line of response (LOR). Simulations at CIEMAT show that if we use these data the calculations of the NNs will become too extensive.

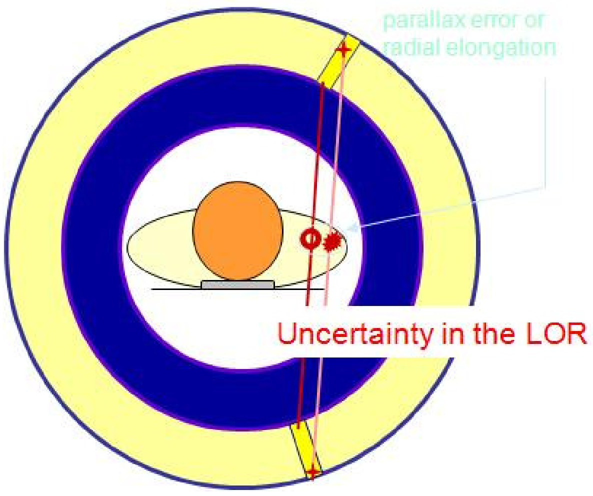


Figure 4: Uncertainty in the line of response created by not using the depth of interaction data [PéBP08].

Because we don’t know whether the interaction of the annihilation photon with the detector crystal happened at the surface or somewhere within the detector crystal, we cannot determine the correct LOR. This error is called the parallax error and is shown in Figure 4.

Although we don’t use the data from the DOI, we are able to calculate the LOR in a justified way. We assume that the annihilation photons are interacting with the surface of the monolithic scintillation crystal. By placing two monolithic scintillation crystals in the opposite direction, both with their own array of APDs we increase the overall efficiency with a good spatial resolution. The goal is to reach a spatial resolution of 1 mm for the final scanner. Figure 5 shows the setup of those monolithic scintillation crystals.

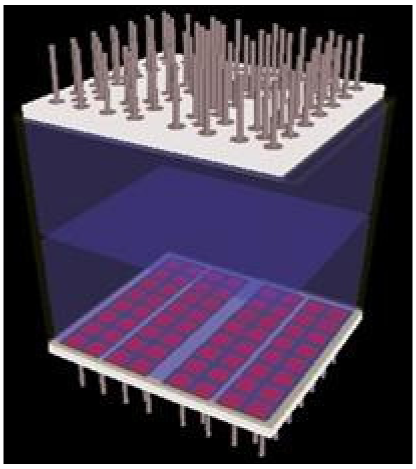


Figure 5: Setup of 2 monolithic scintillation crystals both with their own array of avalanche photodiodes [PéBP08].

In Figure 5 the arrays of APDs are already coupled to the detector crystals, because of the limited size of those APD arrays we need 2 arrays for every detector crystal. The design of the entire ring of detectors is shown in Figure 6. One detector ring consists of 52 pairs of 2 monolithic scintillation crystals. Four detector rings are placed side by side axially as shown in Figure 6. We use LYSO crystals as detector crystals, this crystal will be further discussed in paragraph 2.3.1.

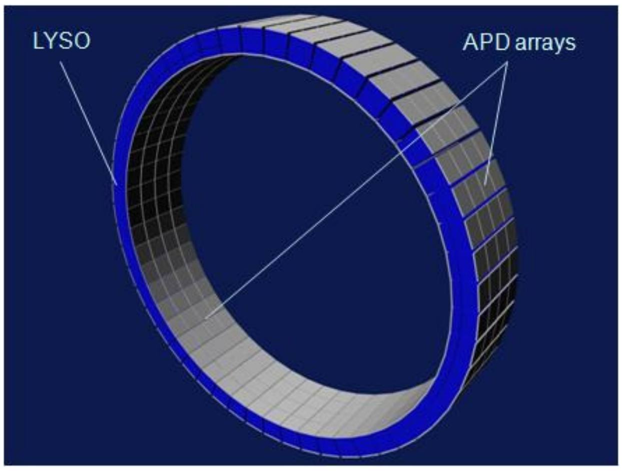


Figure 6: Detector ring setup of a BrainPET scanner [PéBP08].

When the incident annihilation photons interact with the monolithic scintillation crystal, due to the photoelectrical effect, the photon will be converted into an electron. This electron will ionize and excite its surroundings. The ionization will cause light to be generated by scintillation reactions in the crystal. This light is reflected by reflectors placed on the sides of the crystal. Figure 7 shows the reflections of the light in the scintillation crystal.

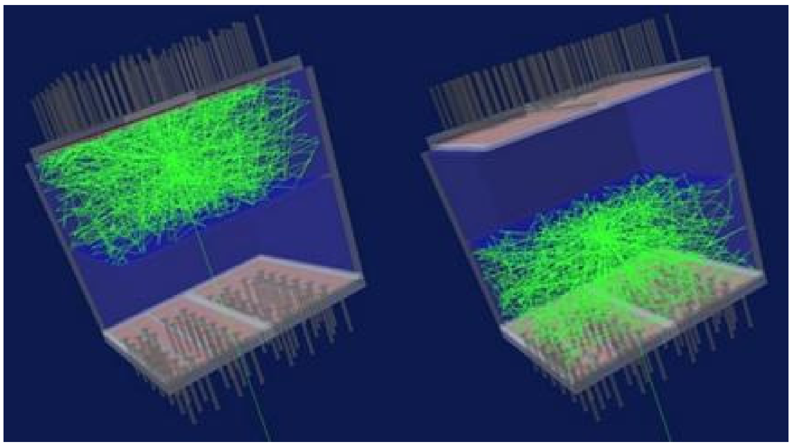


Figure 7: The reflections of the light in the scintillation crystals [PéBP08].

The amount of light is then read out by the APDs and transformed into an electrical signal. The more energy is deposited in the crystal, the more light will be produced and the bigger the electrical signal from the APDs will become. By the placement of the 64 APDs under the detector block, the detector block is divided into eight rows and eight columns, so each pixel has its own APD. The 64 APDs are gathering the light that the monolithic crystal is producing, all covering a certain surface of the crystal.

We are summing the 8 pixel values of each row and each column, this provides us with 16 output values. Eight values corresponding to the columns and eight values corresponding to the rows. We use the sum values instead of the separate pixel values because of the following reasons. First of all it will reduce the noise on the output signal. The signal to noise ratio for the sum of 8 channels will be higher than for 64 single channels. Secondly, it will reduce the number of input channels in the NN. Reducing them from 64 to 8 will reduce the complexity of the algorithm and will reduce the processing time. Thirdly it will reduce the mechanical and electrical complexity of the ASIC circuit boards.

Because we make use of APDs instead of PMTs we will be able to combine the PET scanner with a MRI scanner because the APDs are not influenced by high magnetic fields. In paragraph 2.3.2, those APDs will be discussed further. The advantage of using a PET

Not all the events measured by the detectors are valid. We can make a logic scheme of the trigger in a PET scanner to accept an event, this is shown in Figure 9.

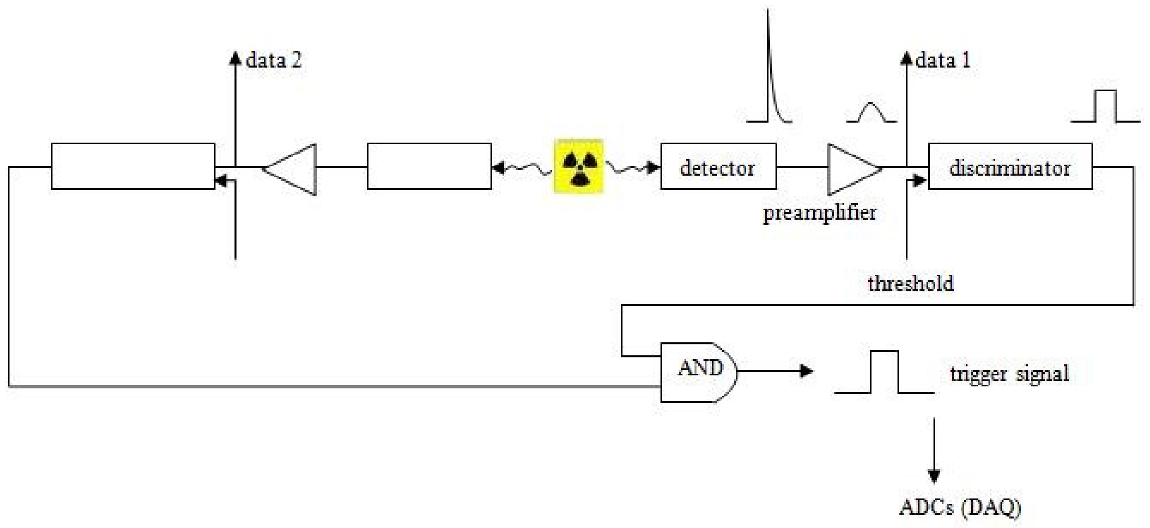


Figure 9: Logic scheme of the positron emission tomography scanner.

An incoming annihilation photon creates a pulse as output of the array of APDs coupled to the scintillation crystals. This pulse is shaped by a pulse shaper and serves then as an input of a discriminator. This pulse shaper is included into the application specific integrated circuit (ASIC) which is connected with the output of the APD arrays. This ASIC will be discusses in more detail in a paragraph 2.3.3. The discriminator compares the incoming pulse with a threshold value. When the energy of the pulse falls above the threshold conveniently chosen below the photo peak of the 511 keV annihilation photon the event is accepted, if not the event will be erased. After the discriminator the signal comes together with a signal produced by a different detector. When both signals are passing through a logic AND gate within a certain time interval, the combination of the two signals is considered as a valid event correlated to an annihilation. The electronics as spoken above checks the *x* and *y* spectrum independent of each other, the new chip sums them first and makes then the necessary comparisons. So the uncertainties should decrease if we make use of this chip.

If an event is accepted the output spectra of the APD arrays are loaded into a NN. One NN is needed to calculate the *x* coordinate and one NN for the *y* coordinate of the position of the interaction of the incident annihilation photon at the crystal surface. With the *x* and *y* coordinate known of two detector blocks in the ring of detectors, a LOR can be calculated.

events that will take place in our PET scanner per second, a fast detection is required. Figure 12 shows the diagram of the response of a LYSO crystal to 511 keV with a R3241 PMT.

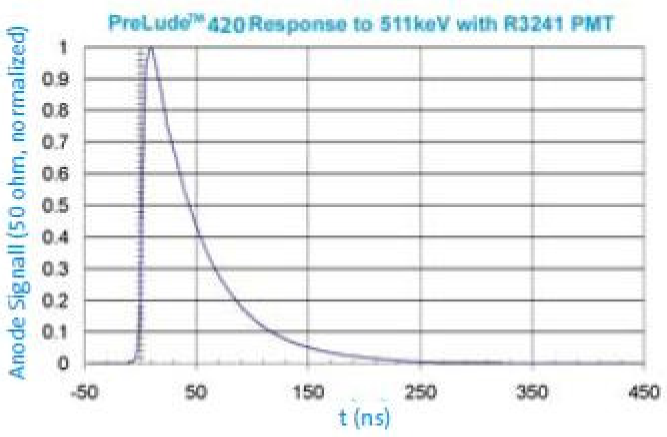


Figure 12: PreLude420 response to 511 keV with R3241 PMT [SG09].

Further we notice a difference in the energy resolutions of the scintillation materials, LYSO has the best energy resolution followed by LSO and BGO. The density and the attenuation length are comparable for the three different scintillation materials, the attenuation length is also called the absorption length. The effective atomic number, together with the density defining the detection efficiency of the detector, for LYSO is worse than for BGO and comparable to LSO. When looking to the temperature coefficients of the scintillation materials we see that LYSO is the least temperature dependent followed by BGO and LSO who have both comparable temperature coefficients.

The principal properties of LYSO, BGO and LSO which we discussed previously are summarized in Table 1.

Property	**PreLude420**	**BGO**	**LSO**
Density (g/cm^3)	7,1	7,1	7,4
Effective atomic number	66	75	65
Attenuation length for 511 keV (cm)	1,2	1,0	1,15
Decay time (ns)	41	300	40
Energy resolution at 511 keV (%)	8,0	12,0	10,0
Light output, photons per keV	32	9	26
Average temperature coefficient 25 to 50 (%/ºC)	-0,28	-1,2	-1,3

Table 1: Comparing principal properties of PreLude420 versus BGO and LSO [SG09][Sm09][OP09].

The main component of PreLude420 is lutetium, this contains the radioactive isotope ^{176}Lu which is a naturally occurring beta emitter. When ^{176}Lu decays it will decay in 99,66 % of the time to ^{176}Hf. This ^{176}Hf will be in an 597 keV excited state then and will decay through a cascade of three gamma rays with corresponding energies of 307, 202 and 88 keV.
A crystal with a diameter of 1 inch and a thickness of 1 inch will absorb nearly 100 % of the beta particles but some photons will be able to escape. This will lead to four peaks of beta and gamma energy distributions, represented in Figure 13 [SG09].

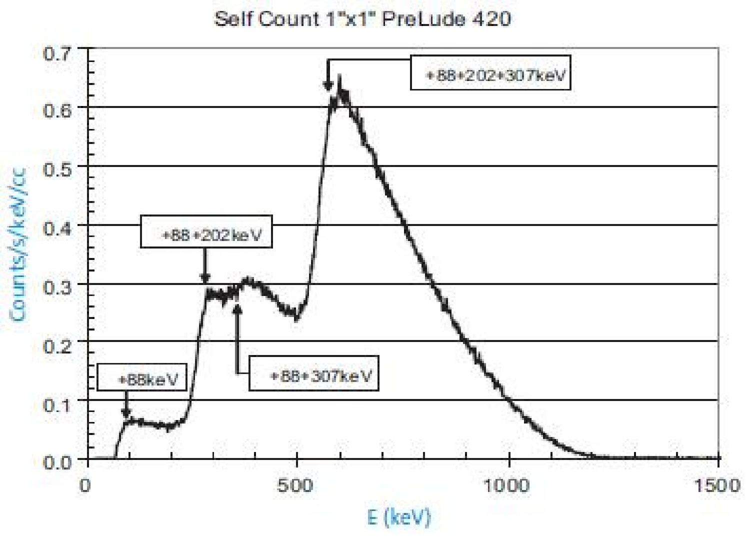


Figure 13: Beta distributions [SG09].

Figure 14 shows the physical dimensions of a pair of monolithic scintillation detector blocks. The shape of the blocks is trapezoidal in one direction. Both blocks have a height of 10 mm, the detector block closest to the center has a ground surface of 21,4 x 18,5 mm and a top surface of 22,4 x 18,5 mm. For the detector block the widest from the center of the detector ring those surfaces are respectively 22,5 x 18,5 mm and 23,5 x 18,5 mm. We chose the trapezoidal shape because previous studies show that the efficiency of this type of detector block is better and more uniform than for a rectangular shaped detector block [Ma05]. For all the experiments discussed in this thesis we used a trapezoidal detector block with the dimensions of a detector block on the inside of the detector ring, so being 18,5 x 21,4/22,4 mm, with 10 mm thickness.

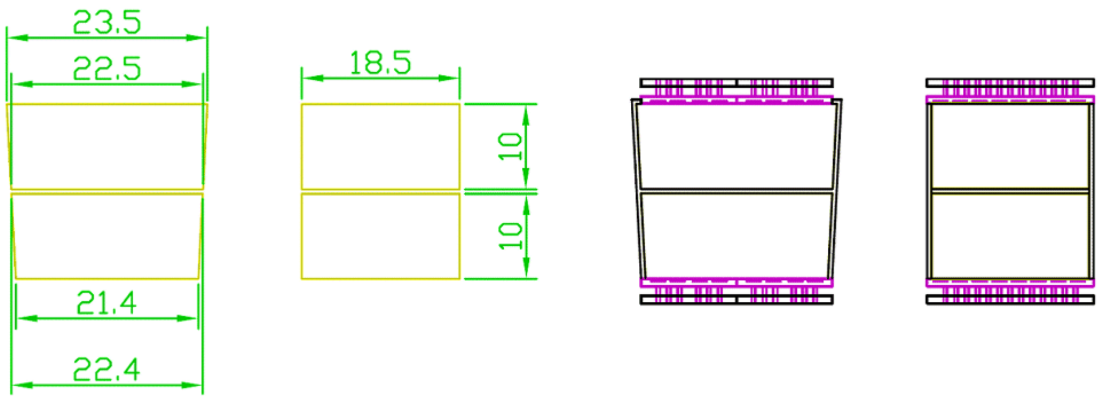


Figure 14: Physical dimensions of a pair of detector blocks.

Figure 15 shows the assembly of the detector block surrounded with Teflon. It is optically glued on the APD arrays and connected to a PCB.

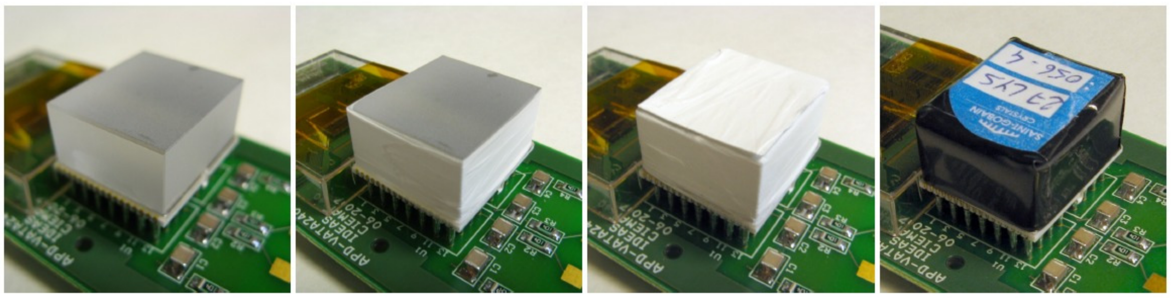


Figure 15: Assembly of the monolithic LYSO detector block surrounded with Teflon.

2.3.2 Avalanche photodiode

Progress in the development process of semiconductors during the years, made it possible to use photodiodes instead of the classic PMTs in some applications. If we make a comparison between the photodiodes and the PMTs then we find a number of advantages of the photodiodes over the PMTs: a higher quantum efficiency, lower power consumption, more compactness and improved ruggedness. The quantum efficiency indicates how well the incident optical photons are absorbed and then transformed to charge carriers. This property may lead to a higher energy resolution, with a photodiode an efficiency of 60-80 % can be achieved. An extra advantage of the photodiodes is that they are insensitive to magnetic fields and so they can be used in experiments where the PMTs cannot be used because of their incompatibility with magnetic fields. Because of the relative small

Light enters through the thin p^+ layer at the left side of the figure and interacts somewhere within the π region that constitutes most of the thickness of the photodiode. Photons from scintillation have a typical energy of around 3-4 eV, enough to create electron-hole pairs in the π region which consist out of a semiconductor with a bandgap of about 1-2 eV. The electrons are drawn to the right side through the drift region and into the multiplying region because of the presence of a high electric field. In this region additional electron-hole pairs are created which enhance the output signal. Gain factors of a few hundred are typical under normal circumstances. This enhancement of the signal is enough to be able to sense much lower levels of scintillation light and results in much better signal to noise ratios [Kn99, 287-292].

We used 2 arrays of APDs connected with the monolithic crystal, both arrays are silicon S8550 APD arrays from HAMAMATSU. The S8550 APD array is a monolithic 8 x 4 pixels structure with an active area of 2,56 mm² for each pixel. In the pulse mode this device allows stable operations at gains up to 74 with a detection efficiency of around 60 % for photons of 420 nm wavelength. They are optimized to have a high sensitivity in the short wavelength region, more precise the region of blue light. We chose these APDs because the light the LYSO detector block is producing has a wavelength of 420 nm, so fitting the region where these APDs are optimized for. The S8550 array has a uniform gain between the different APDs [HA85].
From previous experiments the main drawback of this device seems to be the low internal gain, resulting in a low signal to noise ratio. A second drawback is that surface effects are reducing the efficiency of the APDs for light produced by LSO crystals. These disadvantages are similar as for already existing single devices manufactured by HAMAMATSU. The main advantage of this S8550 APD array is it’s high reliability, as proven in several experiments and applications [Ka]. Figure 17 shows an APD array of the same type we used in our experimental setup.

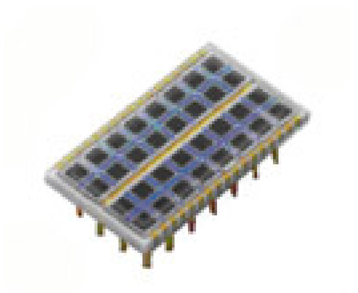


Figure 17: S8550 type APD array [Ve].

The ASIC we used for our application, the VATA 240, is developed in Norway by Gamma Medica-Ideas. It was meant to integrate as much as possible the analog front end electronics required by the readout of the monolithic detector blocks used in the BrainPET project. Figure 19 shows a functional diagram of one column or row of the VATA 240 ASIC.

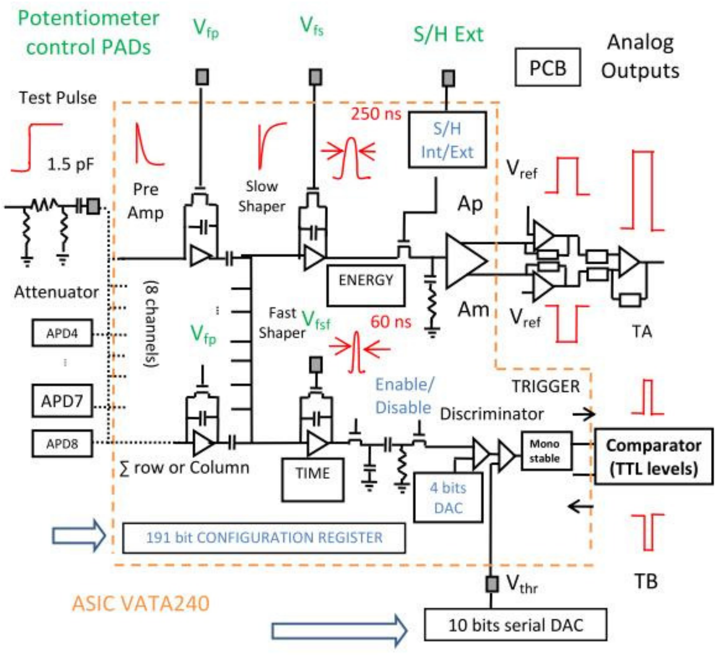


Figure 19: Schematic scheme of the VATA 240 ASIC [Sa08, 2].

The ASIC is a 64 channel low noise charge sensitive preamplifier circuit with a gain of 5 mV/fC, 16 summing shaping circuits with simultaneous sample and hold. The sampling time of the ASIC can be adjusted by a 5 bit DAC. There are 64 channels because in the original experimental setup we needed to be able to read out all the 64 pixels of the detector block, these 64 pixels are shown in Figure 20.

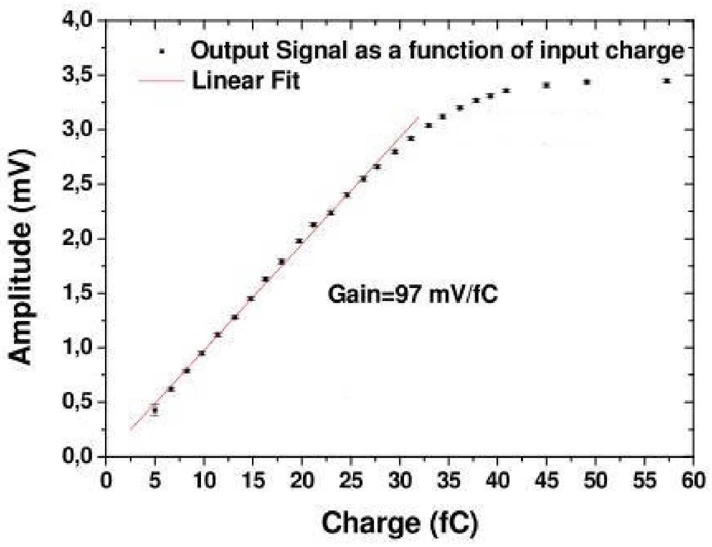


Figure 21: Output of the ASIC as a function of input charge [Sa08, 2].

Figure 22 shows the optimal timing values of the ASIC for an input calibration pulse injected into the ASIC. Due to the use of the fast shaper response and the threshold value we obtain only 60 ns of delay time between the pulse and the trigger output. The time between leading edges of the trigger and the hold output signal is 160 ns, and the jitter of the trigger is less than 2 ns [Sa08].

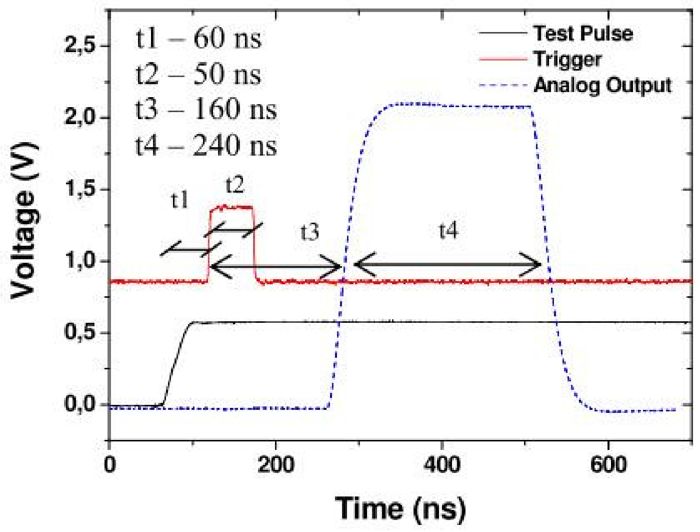


Figure 22: Timing response of the ASIC for a 10 fC calibration test pulse injected into one row [Sa08, 3].

When the gate is open the VME (VERSA Module Eurocard) peak sensing ADC picks the maximum value of its input data coming from the detector box. All the VME-modules are connected with each other through a VME-bus. In this way the data from the peak sensing ADC is send back to the module that created the gate and through this module the data is send to the PC. By using a LabWindows program we were able to see the spectrum of all the columns, rows and the total sum.

More detailed information about the used equipment can be found in the appendix under A.1.

3.1.2 Energy calibration and energy resolution

The output values of the setup are spectrums in ADC units. But because we are interested in the energies the setup is measuring we have to find the relationship between the ADC channels and the energy. In doing so we place two difference sources in the setup, from which we know the energy of the photons they emit. Then we can determine the correlation between the energy and the ADC channels.

The energy calibration was carried out with a ^{22}Na and a ^{137}Cs source. ^{22}Na has a half life time of 2,6 years. The decay of ^{22}Na is shown in Figure 24.

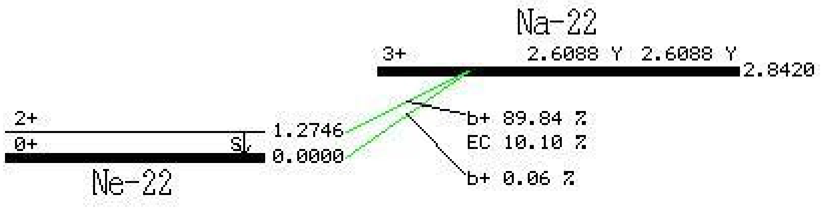


Figure 24: Schematic diagram of the decay of ^{22}Na [En90].

Isotope ^{22}Na decays to the stable element ^{22}Ne. There are three different ways in which ^{22}Na can reach ^{22}Ne. The probabilities of the different reactions are noted in Figure 24. We use the 511 keV photons created by the annihilation reactions following the different β^+ decays and the 1274,6 keV photons that are characteristic from the exited ^{22}Ne.

^{137}Cs has a half life time of 30,04 years. The decay of ^{137}Cs is shown in Figure 25.

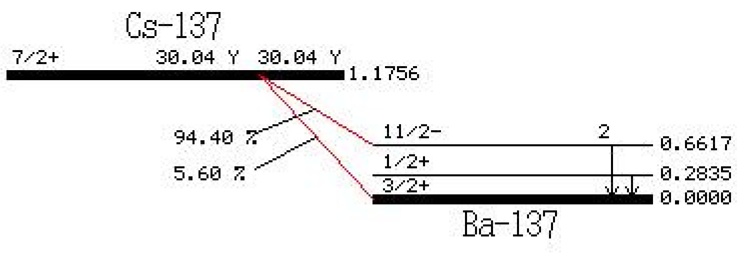


Figure 25: Schematic diagram of the decay of ^{137}Cs [Tu97].

Isotope ^{137}Cs decays to the stable element ^{137}Ba. There are two possible ways for ^{137}Cs to decay to ^{137}Ba, both are β^- reactions. The probabilities of the different reactions are noted in Figure 25. We use the 661,7 keV photons from the most likely reaction.

We performed the energy calibration for 2 different high voltages on the APDs, first by using a high voltage of 417 V followed by an energy calibration using a high voltage of 427 V. To perform these energy calibrations we used the total sum channel. The ADC output of the total sum channel is a spectrum. We obtain one spectrum for ^{22}Na and one for ^{137}Cs. From those two spectra we select the peaks corresponding to 511 keV and 1274,5 keV for ^{22}Na and 661,7 keV for ^{137}Cs. With Origin 7.0 we fit the peaks with a Gaussian function. From the equation of the Gaussian function we obtain the peak position. By knowing the peak positions in ADC channels and knowing the peak positions in energy units we can provide the energy calibration.

From one of the parameters in the equation of the fit shown in Figure 26, the *w* parameter, we can obtain the FWHM values of the peaks. This parameter is defined as followed: $w = 2 \cdot \sigma$ with σ being the standard deviation and FWHM = $2 \cdot \sqrt{2 \cdot \ln 2} \cdot \sigma$. Making use of the energy calibration we can transform the FWHM values expressed in ADC channels into FWHM values in energy units.

Also the FWTM values of the peaks can be determined using the y_o and y_c parameters of the Gaussian functions. Using the energy calibration those values can be expressed in energy units.

Figure 26 shows the definition of the different parameters.

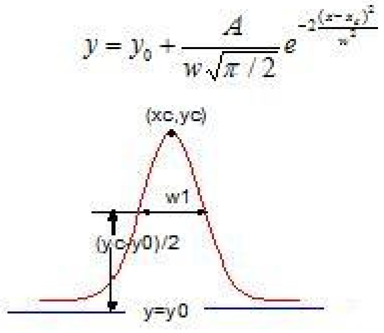
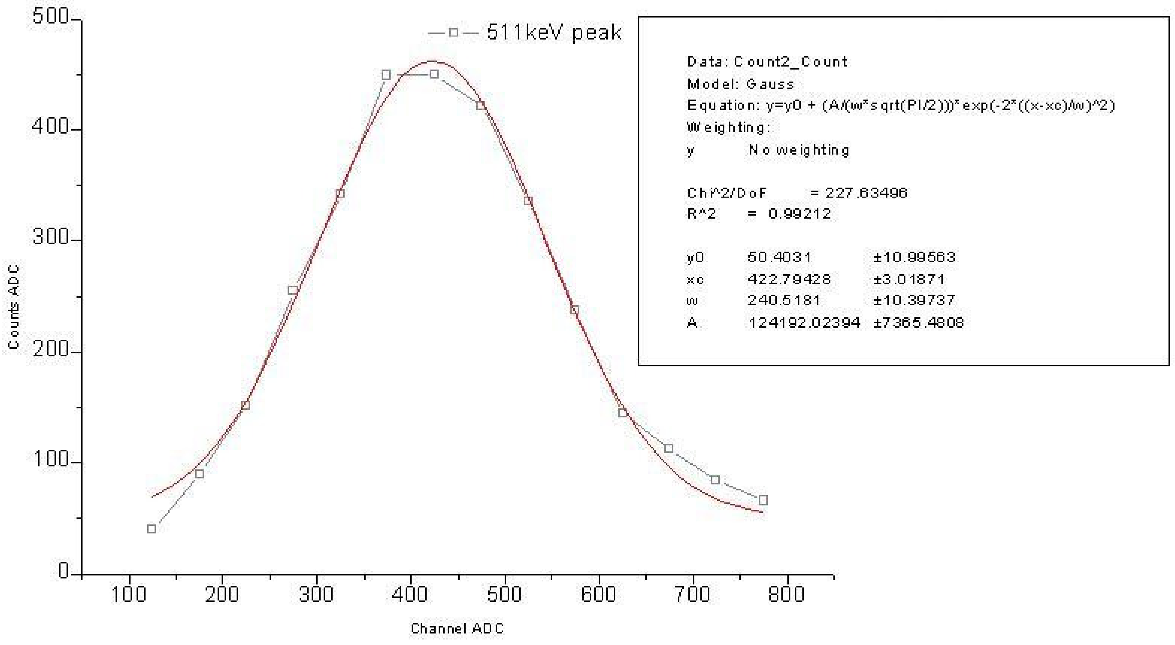

$y = y_0 + \frac{A}{w\sqrt{\pi/2}} e^{-2\frac{(x-x_c)^2}{w^2}}$
(xc,yc)
w1
(yc-y0)/2
y=y0

Figure 26: Definition of the different parameters in the Gaussian fit by OriginPro 7.0.

Finally we can also determine the energy resolution of the setup. We multiply the FWHM value in ADC units with the slope of the energy calibration line and divide this product by the energy of the corresponding peak to obtain this resolution.

3.1.2.1 Results for a high voltage of 417 V

In Figure 27 to Figure 29 the diagrams of the sum channel using a high voltage of 417 V and after fitting the curves with a Gaussian function are shown.

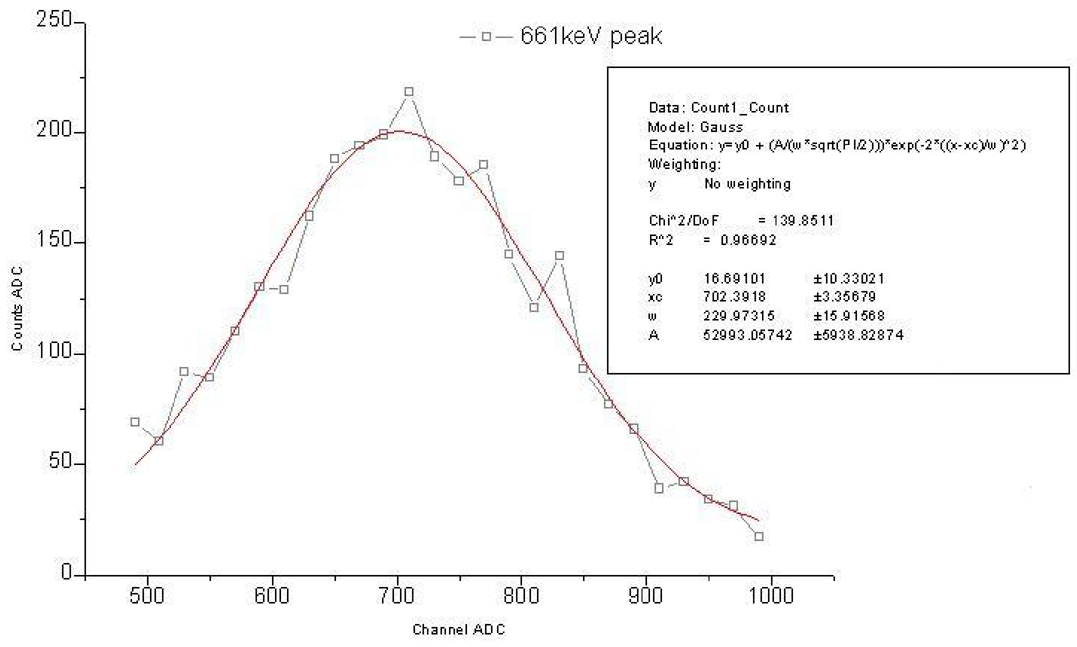


```
Data: Count2_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y          No weighting

Chi^2/DoF        = 227.63496
R^2       =  0.99212

y0       50.4031          ±10.99563
xc       422.79428        ±3.01871
w        240.5181         ±10.39737
A        124192.02394  ±7365.4808
```

Figure 27: Gaussian fit of the 511 keV peak of ^{22}Na for a high voltage of 417 V.

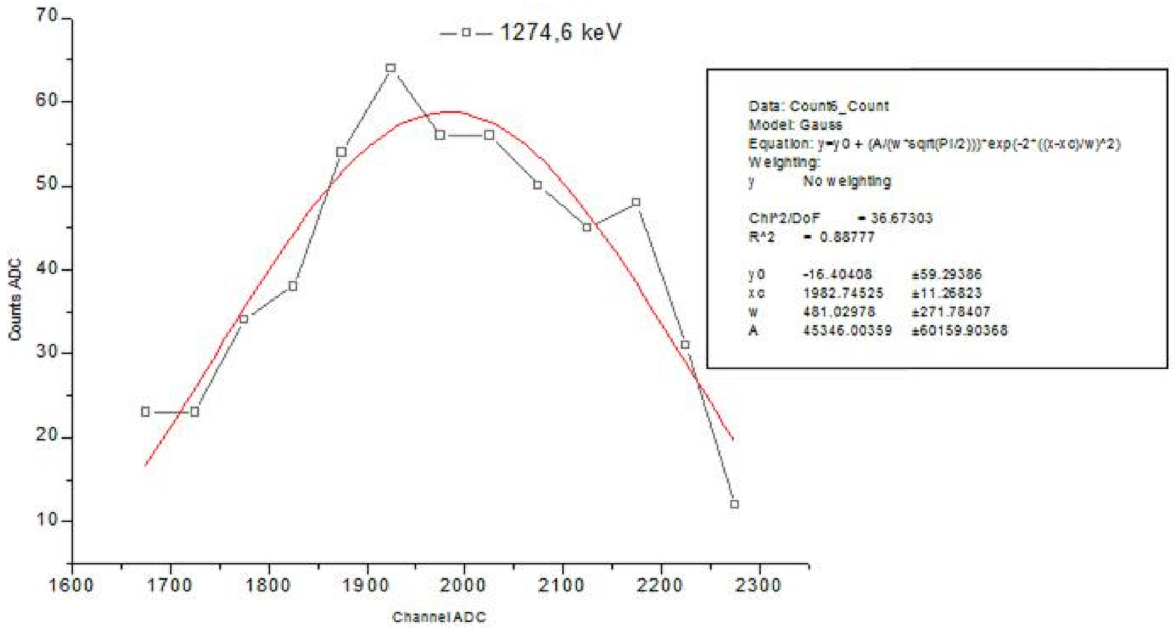


```
Data: Count1_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y          No weighting

Chi^2/DoF        = 139.8511
R^2      =  0.96692

y0         16.69101         ±10.33021
xc         702.3918         ±3.35679
w          229.97315        ±15.91568
A          52993.05742      ±5938.82874
```

Figure 28: Gaussian fit of the 661 keV peak of ^{137}Cs for a high voltage of 417 V.



```
Data: Count6_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y          No weighting

Chi^2/DoF        = 36.67303
R^2      =  0.88777

y0         -16.40408        ±59.29386
xc         1982.74525       ±11.26823
w          481.02978        ±271.78407
A          45346.00359      ±60159.90368
```

Figure 29: Gaussian fit of the 1274 keV peak of ^{22}Na for a high voltage of 417 V.

From the Gaussian curves we can calculate the peak values for the given energies. Table 2 gives these peak values.

Energy (keV)	ADC channel number
511	423
661,7	702
1274,6	1983

Table 2: ADC channel numbers of the corresponding energy peaks for a high voltage of 417 V.

Figure 30 shows the correlation between the energy and the ADC channels, calculated from the data in Table 2.

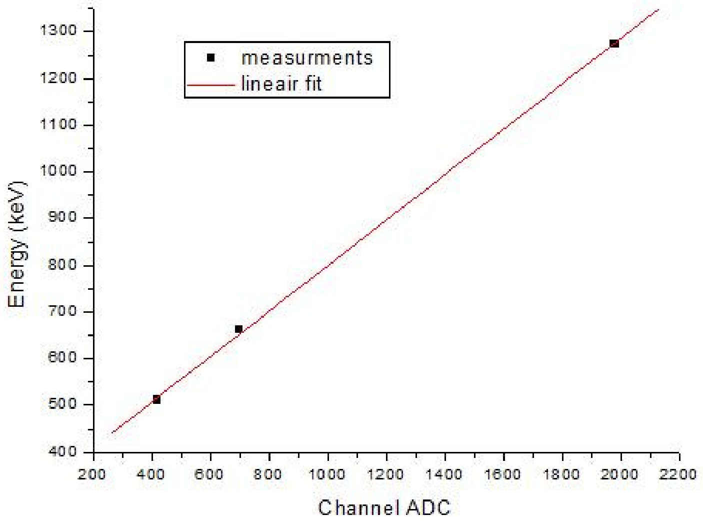


Figure 30: Linear correlation between the energy and the ADC channels for a high voltage of 417 V.

The R^2 value of the linear fit is equal to 0,999 so we can conclude that the behavior of the detector is quite linear. The linear fit is given by the next equation:

$$\text{energy (keV)} = 0{,}486 \cdot \text{ADCchannel} + 312.$$

- Table 3 shows the calculated FWHM values

Energy (keV)	*w* parameter	FWHM value in ADC channels
511	241	284
661,7	230	271
1274,6	481	566

Table 3: FWHM values of the corresponding peaks for a high voltage of 417 V.

- The results of the calculations to obtain the FWTM values are shown in Table 4.

Energy (keV)	FWTM value in ADC channels
511	516
661,7	493
1274,6	1032

Table 4: FWTM values of the corresponding peaks for a high voltage of 417 V.

- The energy resolutions of the corresponding peaks are shown in Table 5.

Energy (keV)	Energy resolution (%)
511	27,0
661,7	19,9
1274,6	21,6

Table 5: Energy resolution of the corresponding energy peaks for a high voltage of 417 V.

3.1.2.2 Results for a high voltage of 427 V

The spectra with the corresponding Gaussian fits for a high voltage of 427 V can be found in the appendix under A.2. Figure 31 shows the linear correlation between the energy and the ADC channels for a high voltage of 427 V as well as for a high voltage of 417 V.

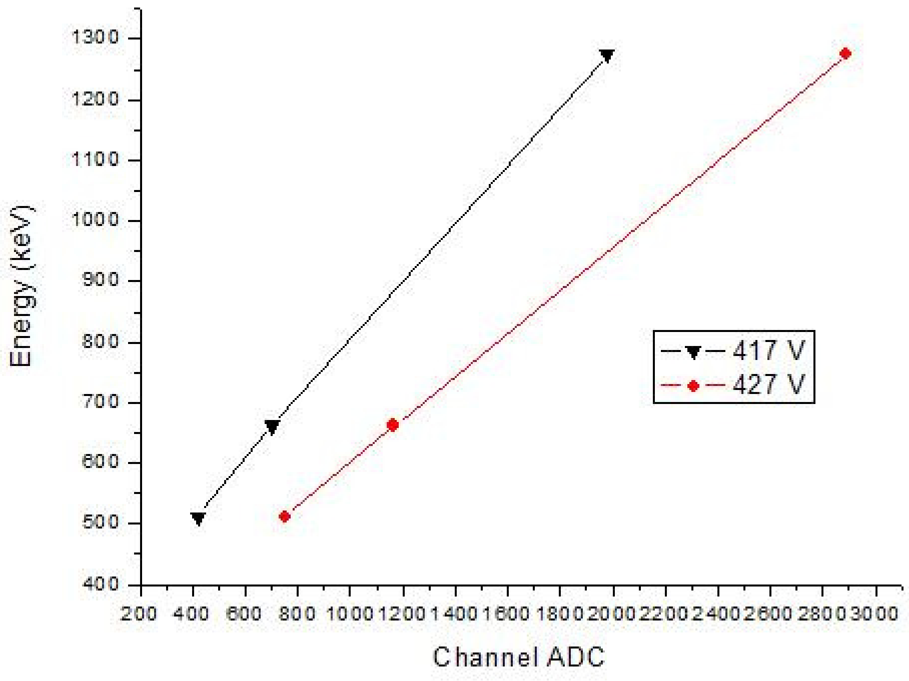


Figure 31: Linear correlation between the energy and the ADC channels for a high voltage of 417 V and 427 V.

We can see on Figure 31 that the slope of the line of the 417 V is bigger than the slope of the 427 V line. We can explain this as follows: with a higher high voltage on the APDs the

With this ‘real’ setup, shown in Figure 32, we are able to use two detector boxes working in coincidence, when the setup is used in this mode the PMT is removed. Each detector box is mounted on a plate which can be moved by three independent electro motors. Two electro motors for the movement of the detector box in the *x*, *y* plane and the third electromotor to make it possible to let the photons incident with a certain angle. In Figure 33 the plate on which the detector box will be mounted is shown.

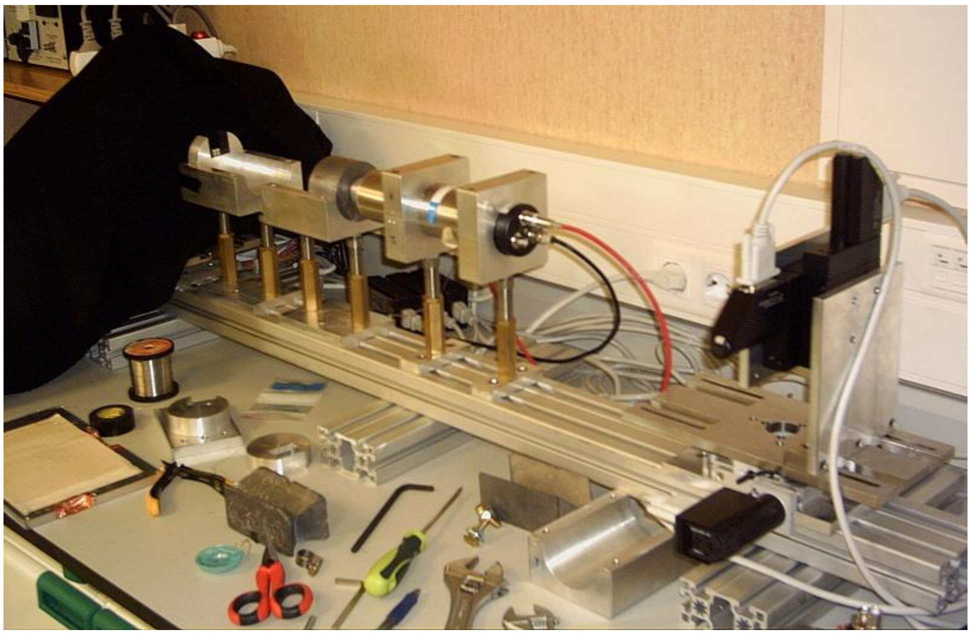


Figure 32: Setup for the two detector boxes.

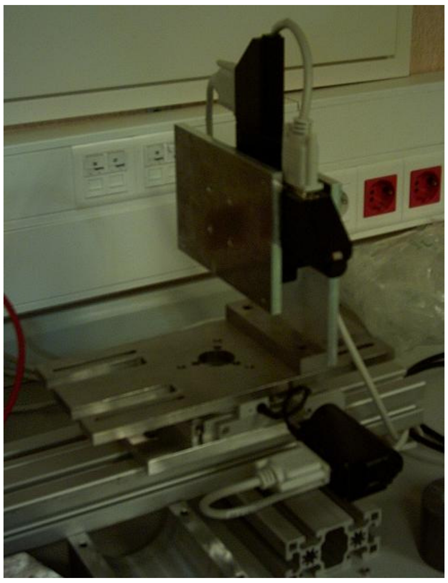
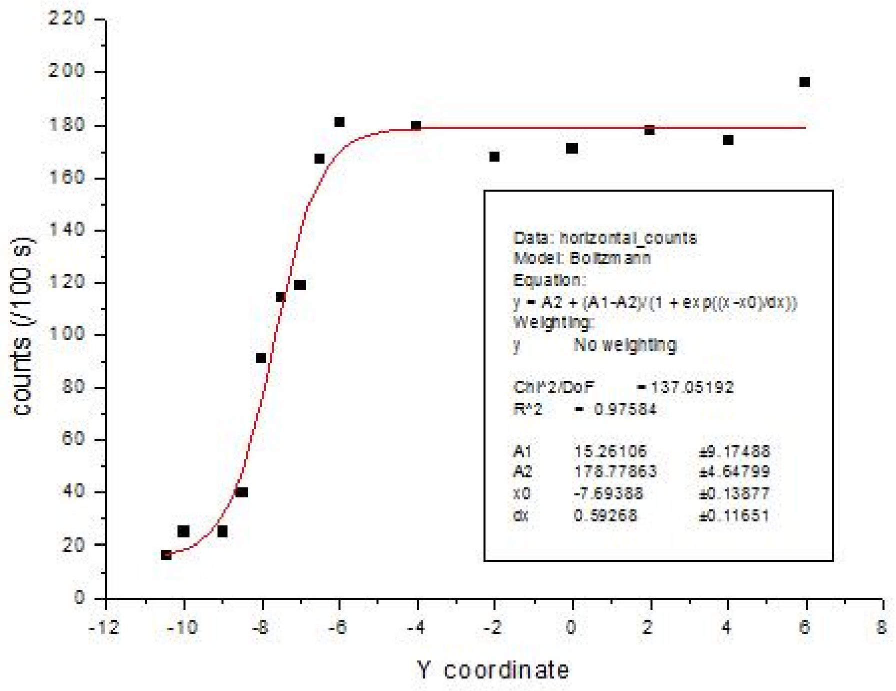


Figure 33: Mounting plate of the detector box, driven by three independent electro motors.

PMT was not correctly centered anymore with the collimator, with a loss of counts as a consequence.

By fitting the left and the right transition from low to high or the other way with a sigmoid function, we can obtain the coordinates of the borders of the detector block. Using those coordinates we can calculate the coordinates of the center of the detector block in the plane created by the electro motors. Figure 35 and Figure 36 are showing the sigmoid fits of the transitions of the plateau of the measurements performed in the *y* direction.



```
Data: horizontal_counts
Model: Boltzmann
Equation:
y = A2 + (A1-A2)/(1 + exp((x-x0)/dx))
Weighting:
y          No weighting

Chi^2/DoF      = 137.05192
R^2     =  0.97584

A1      15.26106      ±9.17488
A2      178.77863     ±4.64799
x0      -7.69388      ±0.13877
dx      0.59268       ±0.11651
```

Figure 35: Sigmoid fit of the transition on the left side of the plateau of the measurement performed in the *y* direction.

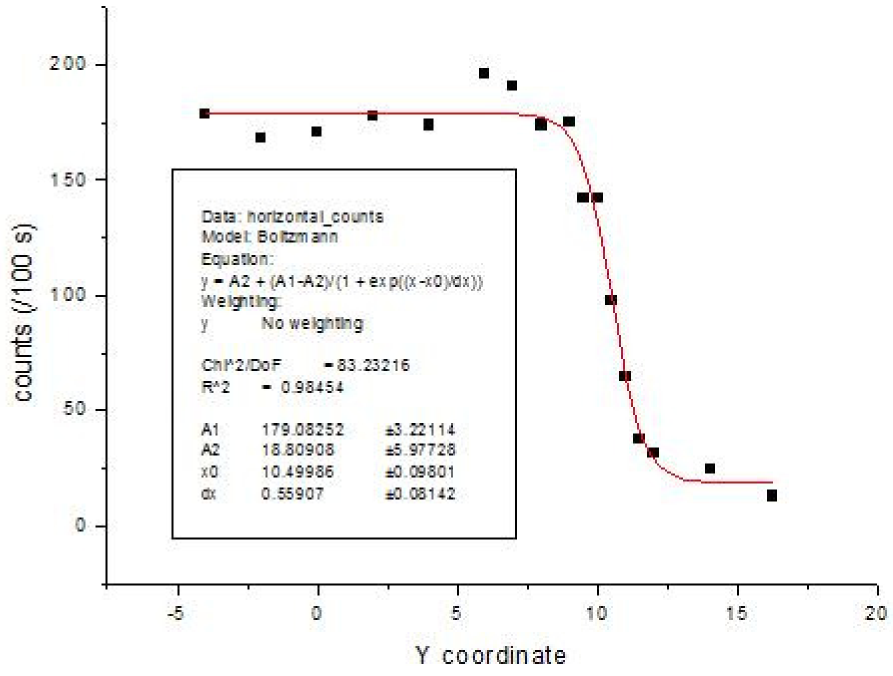


Figure 36: Sigmoid fit of the transition on the right side of the plateau of the measurement performed in the *x* direction.

The sigmoid fits on the plateau obtained in the *x* direction can be found in the appendix in paragraph A.3. In the *y* direction the fits provide us with the border coordinates being -7,96 and 10,5 mm, for the *x* direction those values are -7,84 and 14,3 mm. The center of the detector block in the plane created by the electro motors is thus located at (3,25 ; 1,38 mm). We will have to take those values into account when looking into the results of experiments obtained on this setup.

If the curve makes a transition from a low to a high level or the other way, then halfway this transition the beam is located at the edge of the crystal. Knowing this we are able to estimate the width of the detector block from the sigmoid fits. We obtain 18,1 mm by 22,2 mm what is quite close to the real dimensions being 18,5 mm by 21,4 mm.

3.2.2 Comparison of the energy resolution of the Teflon wrapped and white painted detector block

As shown in paragraph 3.1, we managed to achieve an energy resolution of 27,0 % for the 511 keV peak with a high voltage of 417 V and 21,9 % for 427 V when using the first setup. Both energy resolutions were obtained using a detector block with a Teflon wrapping.

Using the ‘real’ setup we managed to achieve an energy resolution of 13 % for the 511 keV peak when using the same detector block with Teflon wrapping, the same ASIC and a high voltage of 433 V. When we now use a high voltage of 427 V we achieve an energy resolution of 14 %. So we are able to achieve a better energy resolution with this ‘real’ setup than with the first setup, shown in Figure 23.

We also performed measurements on the ‘real’ setup to obtain the energy resolution of the white painted detector block. Figure 37 shows the white painted detector block mounted on the APD arrays.

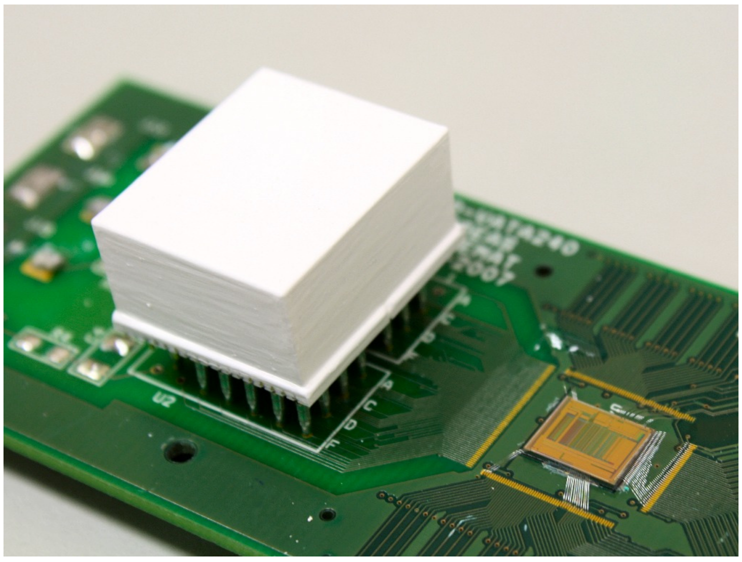


Figure 37: Picture taken of the white painted detector block.

For the white painted detector block we first used a high voltage of 427 V, for this voltage we achieved an energy resolution of the 511 keV peak of 14 %. Afterwards we used a high voltage of 433 V for the same detector block and ASIC and found an energy resolution of 14 %. Unlike for the Teflon wrapped detector block, we now see that the energy resolution of the 511 keV peak remains constant when increasing the high voltage. For the measurements carried out with the Teflon wrapped detector block we used a different ASIC than the one we used for the white painted detector block, but both ASICs have the same gain and characteristics so the results are comparable. We can conclude that regarding to the energy resolution, the Teflon wrapped detector block has a slightly better performance than the white painted detector block.

Another interesting issue of comparison between the Teflon wrapped detector block and the white painted detector block is the energy resolution in function of the energy of the incoming radiation. Figure 38 shows this comparison.

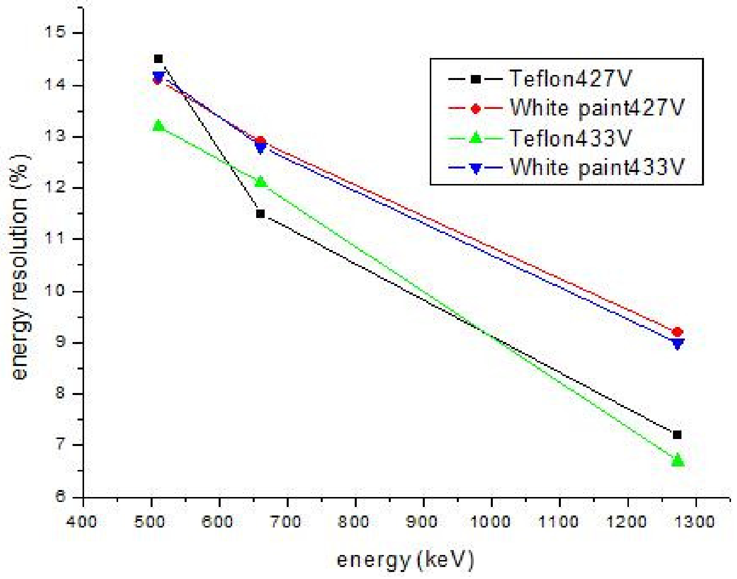


Figure 38: Energy resolution in function of the energy of the incoming radiation for the Teflon wrapped detector block and the white painted detector block, for 427 V and 433 V.

We notice that the Teflon wrapped detector block scores better for almost all the energies and this for both high voltages. Further we see that there is almost no influence of the high voltage on the energy resolution for the different energies when using the white painted detector block. For all the graphs we see clearly that the energy resolution becomes better if the energy of the incoming radiation increases. This comparison confirms that the best combination to use is the Teflon wrapped detector block with a high voltage of 433 V. However if problems are expected for the high voltage stability, the white painted detector block can be a good alternative because its low dependence on the high voltage value.

The peak positions in the ADC unit spectrums are corresponding to the amount of light that is collected by the APDs. Figure 39 shows those peak positions of the corresponding beam positions.

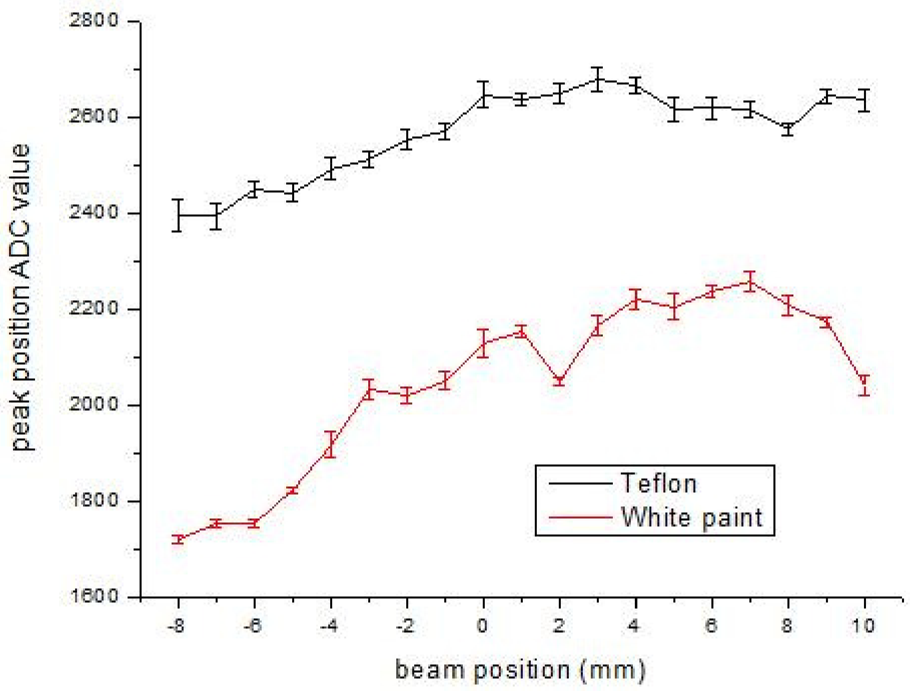


Figure 39: Peak position in ADC units in function of the beam position for the Teflon wrapped detector block and the white painted detector block.

We notice that the peak positions of the spectrums obtained with the white painted detector block are for every beam position smaller than the ones we found when using the Teflon wrapped detector block, further we see that both graphs are increasing from the right to the left. This indicates that the amount of light produced by the Teflon wrapped detector block, that is reaching the APDs, is bigger than when we used the white painted detector block. If we calculate the averaged peak position we find a value of 2569 ADC units for the Teflon wrapped detector bock and a value of 2047 ADC units for the white painted detector block. This means that the white paint is about 0,8 times as reflective as Teflon. The more light is collected by the APDs the better the statistics will get, so for this property the Teflon wrapped detector block scores better than the white painted detector block.

When comparing the peak width values of the spectrum peaks of the corresponding beam positions we obtain Figure 40

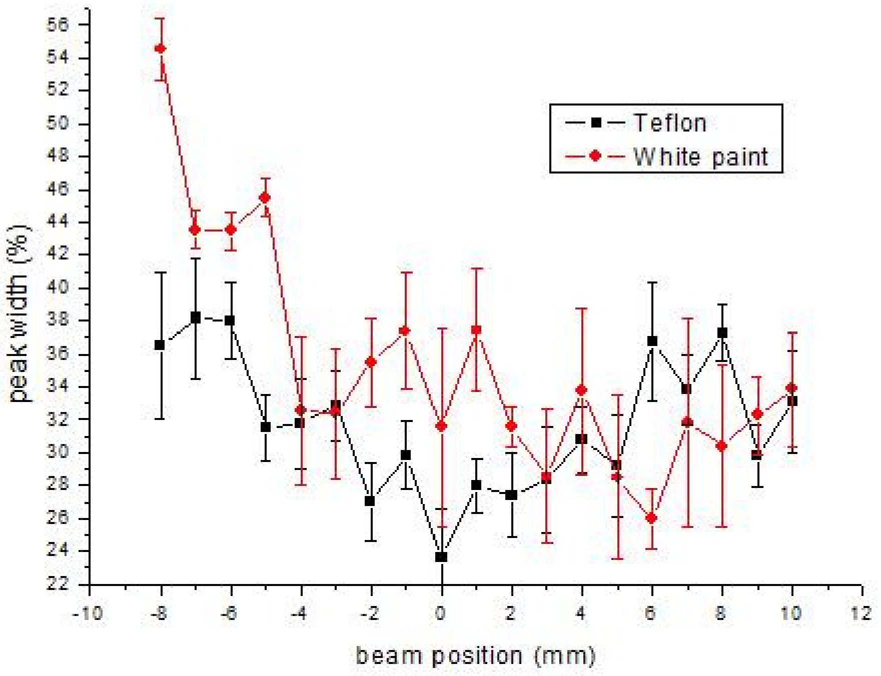


Figure 40: Peak width values for the Teflon wrapped detector block and the white paint detector block in function of the beam position.

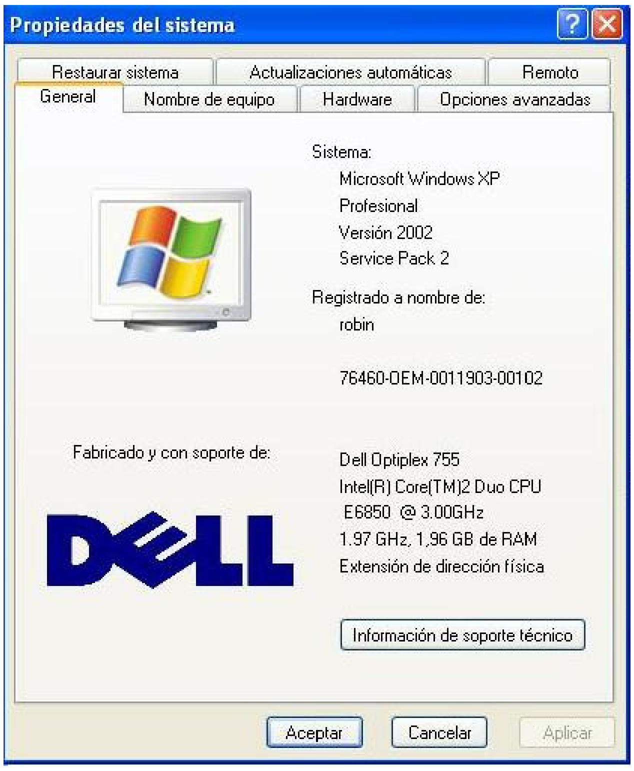
For the Teflon wrapped detector block a pattern appears that shows a decrease of the peak width values in the center, with a minimum value of 24 %. Remember that we are not allowed to compare this value with the obtained energy resolutions. The maximum values are found on the edges.

For the white painted detector block however we see a clearly decreasing trend. In general the peak width values obtained with the Teflon wrapped detector block are the smallest, so preferable. For the averaged peak width values we find 31 % for the Teflon wrapped detector block and 35 % for the white painted detector block. The smaller the peak width values are, the more clear the peaks will be and the more easy it is for the NN to come up with a good estimation of the beam incident position.

4.3.1 Used equipment

A first set of measurements has been carried out on a shared computer grid consisting out of 145 hosts, all with 8 CPUs and running on Linux. With this computer grid, called Euler, we were able to use a maximum of 1160 processors in parallel for our measurements.

Eventually the calculations will probably have to be made on a standard PC. Because of this we made a second set of measurements on a standard fast PC, using two processors in parallel. The maximum clock frequency of those processors is 3 GHz, their nominal clock frequency is slightly less than 2 GHz. Figure 44 shows the system properties of this PC.



Sistema:
Microsoft Windows XP
Profesional
Versión 2002
Service Pack 2

Registrado a nombre de:
robin

76460-OEM-0011903-00102

Fabricado y con soporte de:
Dell Optiplex 755
Intel(R) Core(TM)2 Duo CPU
E6850 @ 3.00GHz
1.97 GHz, 1,96 GB de RAM
Extensión de dirección física

Figure 44: System properties of the PC.

The PC was running on, and the measurements were done while running on, Linux. We used the following Linux program: Fedora release 8 (Werewolf)
Kernel 2.6.23.1-42.fc8 on an x86_64

In paragraph 4.4 we make an introduction in the use of a graphical card as a computing device for our NN application. It was not our intention to present results at the end of the training period, but to take some first steps in what could become a new and more extended project. More detailed theoretical information can be found in the appendix under A.9. The graphical card we used was the GeForce 8600 GT graphical card from NVIDIA. Figure 45 is a picture taken from the graphical card we used. More details about this graphical card can be found in the appendix under A.9.3.

Figure 45: The used GeForce 8600 GT.

4.3.2 Compilers

When writing the neural network software, we have the choice between two different types of compilers to compile our code. First of all there is the gcc compiler, this compiler is downloadable for free from the internet. This compiler can only use one processor at the same time.

On the other hand there are the icc and the icpc compilers, both written by Intel and specially designed to run on an Intel core. The icc compiler is written to compile code in the C language, the icpc compiler is written to compile code written in the C++ language. These compilers can use more than one processor at the same time, but they are not freely available.

Figure 46 and Figure 47 are showing the difference of the performance of the gcc and icc compiling. We see the percentage of the performance of the two processors in function of the time compared with their maximum performance.

Figure 46 shows the CPU history of both processors while running a code three times in a row after the code was compiled with the gcc compiler. We see that during the process there is always only one processor that is working at its maximum rate. In the first process it was processor two that did most of the work, only in the starting up of the process both processors were running at fifty percent. During the second process processor number one did most of the work. In the third process, processor two started and after some time processor one took over. We can conclude that after the gcc compiling there is never a moment where both processors are working at hundred percent.

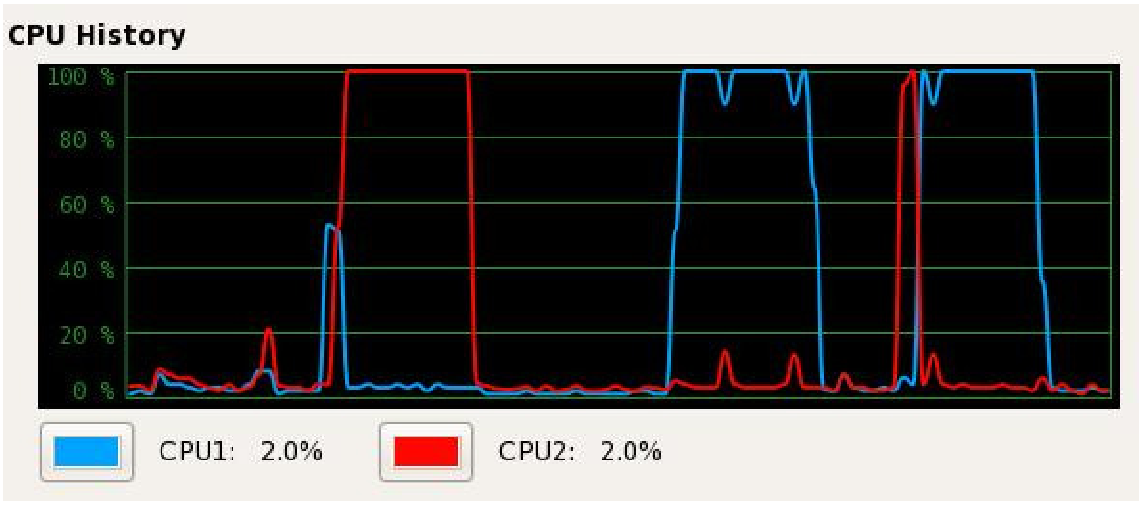


Figure 46: Performance of the two processors after gcc compiling.

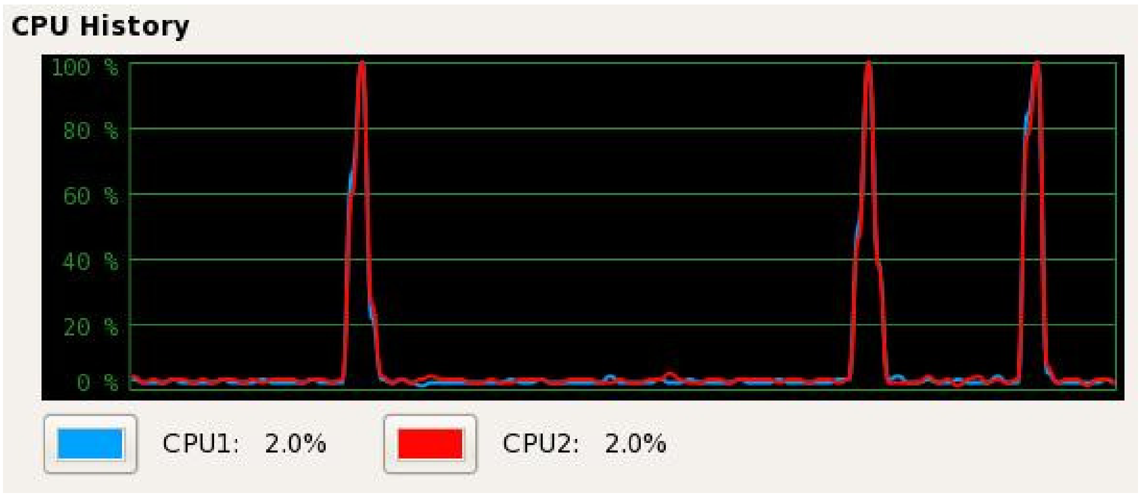


Figure 47: Performance of the two processors after icc compiling.

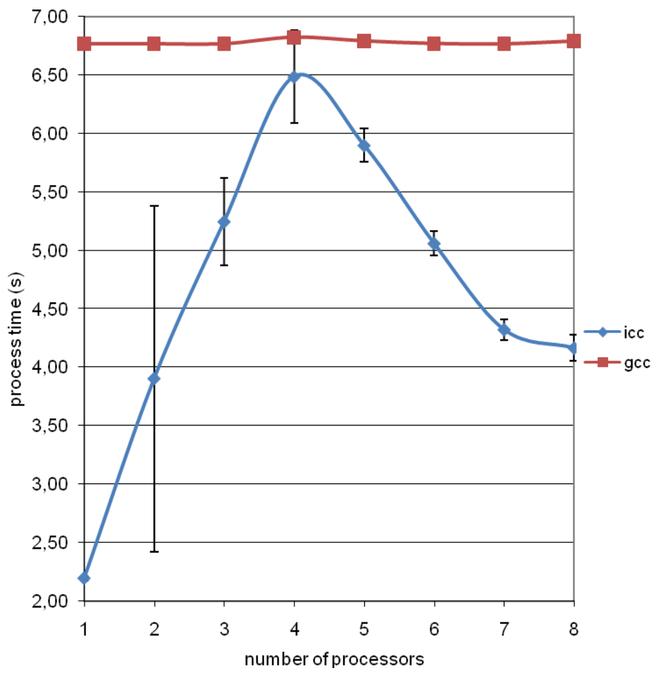


Figure 48: Process time of code A in function of the number of processors for icc and gcc compiling with 2 million NNs on Euler.

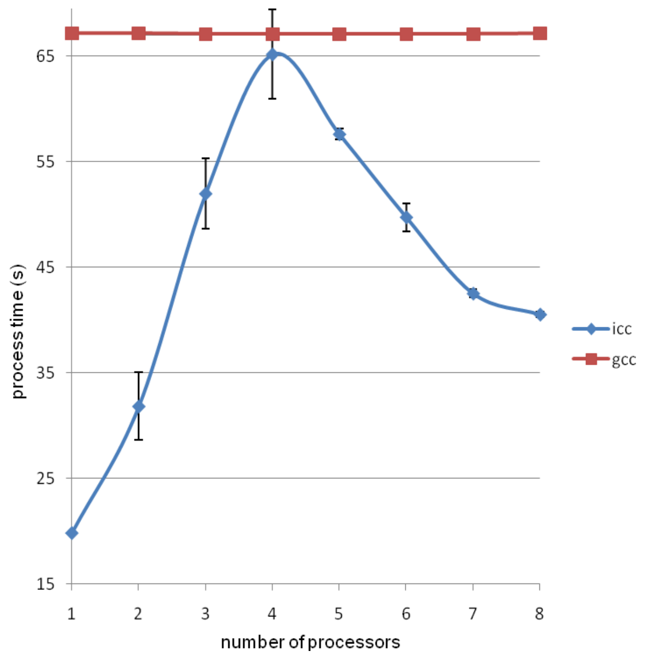


Figure 49: Process time of code A in function of the number of processors for icc and gcc compiling with 20 million NNs on Euler.

In both diagrams the error bars are presenting the standard deviations of the corresponding measurements. In the first diagram the uncertainties of the measurements applied after the gcc compilation are so small that they are invincible in the diagram, the largest uncertainty of those measurements was 0,012 seconds. The uncertainties after the icc compilation are explanation of this difference can be found in a different way of cooperation between the processors in Euler and the PC.

Our goal is to make our code as fast as possible and in the same time getting reliable results. Because we expect that the output process of the code is a major time consumer we performed a measurement to find out what the percentage of the total process time was due to the output process. We found that after icc compiling about 85 % of the total process time is due to the output process, after gcc compiling this becomes 12 %.
So a logic step in the fine tuning process of the code is to make some adaptations in the way the output is given. This adaptation can be found in the appendix under A.6, the code is called 'code B'. Figure 50 show the process time in function of the number of processors for this code.

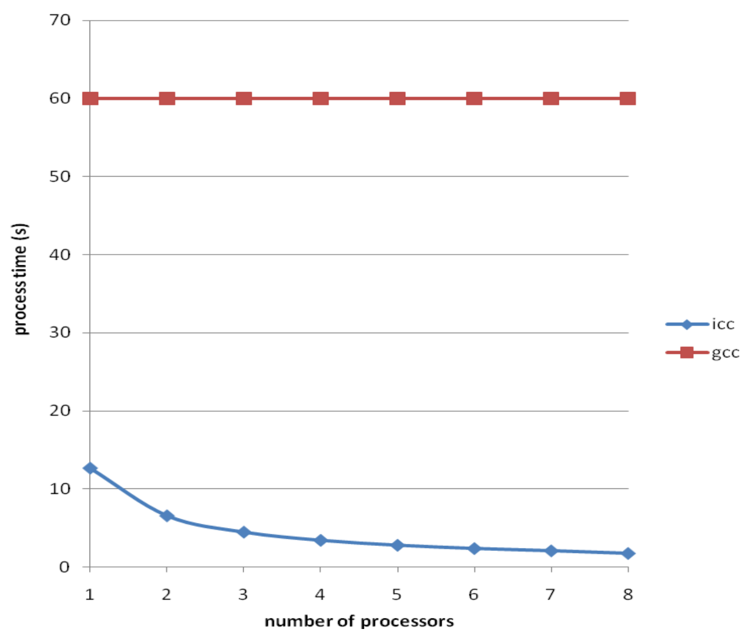


Figure 50: Process time of code B in function of the number of processors for icc and gcc compiling with 20 million NNs on Euler.

On Figure 50 we see a different pattern after the icc compiling than the one we found for the time measurement of code A, shown in Figure 49. Now we are getting the decrease of the process time with the increase of the numbers of processors as we expected. So we can conclude that the shape of the curve of the process time is highly dependent on the code we are compiling.

4.3.4 Sigmoid function

As noted in paragraph 4.2 there is a sigmoid connection between the input layer and the first hidden layer of neurons. And also between the first hidden layer of neurons and the second hidden layer. In the first two versions of the code the computer always had to calculate the sigmoid value, independent of the value of its input. However if we look to the graph of the sigmoid then we can make some simplifications. Figure 51 is the graph of the sigmoid function.

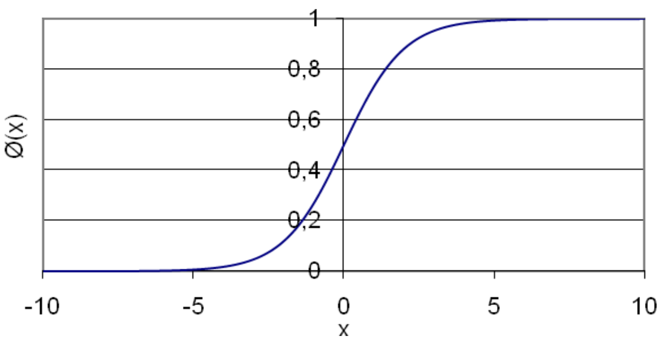


Figure 51: Graph of the sigmoid function.

We see that the sigmoid function is zero for input values smaller than about -5 and one for input values larger than about 5. So we can simplify our code by just letting it calculate the sigmoid function for values between -5 and 5. In the case we have input values bigger than 5, we will set the output of the sigmoid function to one. In the case we have an input value smaller than -5, we will set the output of the sigmoid function to zero. The borders of -5 and 5 are rather arbitrarily. If we calculate the output values for the sigmoid function, then we see that for an input of 4,6 we already have two decimals equal to zero. For the next measurements this value of 4,6 is used as the threshold for the calculation of the sigmoid function. The adaptation of the code can be found in the appendix under A.7, the code is named ‘code C’.

Because with this adaptation of the code there are less calculations that have to be made by the computer, so we expect this code to be faster. We performed some process time measurements in function of the used number of processors, first for Euler afterwards repeated on the PC.

Figure 52 shows a comparison between the results of the measurements of the process time in function of the used number of processors on code B and code C carried out on Euler.

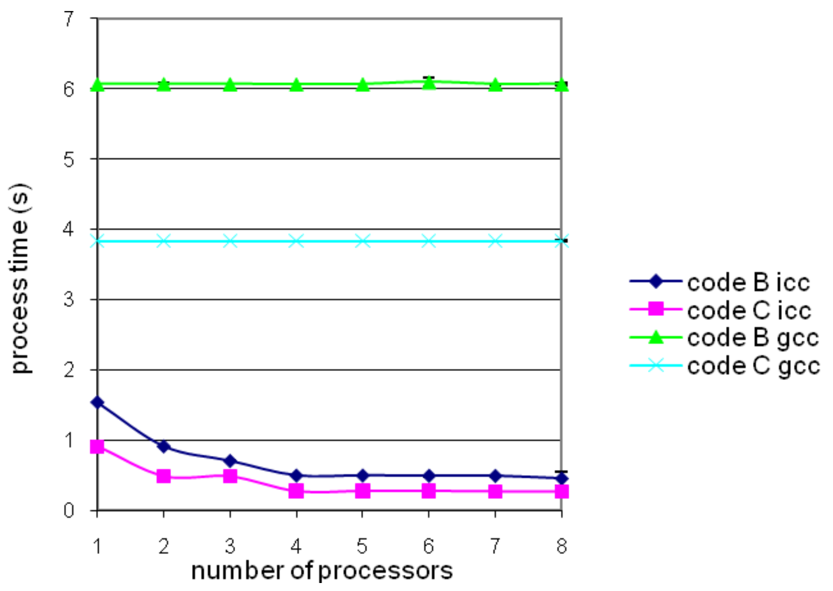


Figure 52: Comparison between code B and code C for icc and gcc compiling and for 2 million NNs on Euler.

In Figure 52 only the results of the measurements done for 2 million NNs are shown. The figure of the measurements done for 20 million NNs is similar, only the numerical values of the process times are bigger and the change of the "code C icc" curve is smoother, so without the jump when using three processors. The uncertainties of the measurements are so small that they are invincible in the graphs.

It is clear that we gain process time if we use code C instead of code B, so the simplification of the code had the desired effect. The gain of the process time because of this adaptation, is shown in Figure 53.

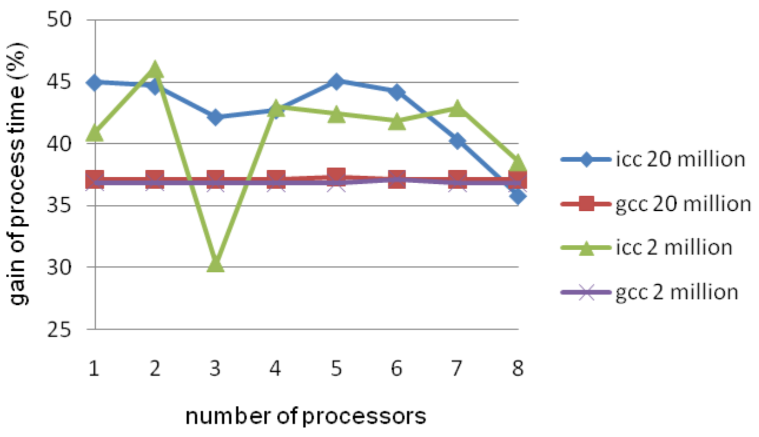


Figure 53: Gain of the process time in percentages as a consequence of the transition from code B to code C for 2 and 20 million NNs.

When we look to the Figure 53 we see that the percentage of process time we are gaining is quite constant when we increase the number of processors in parallel if we use the gcc compiler for 20 million and 2 million NNs. Further we see that the percentage of process time that we are gaining is independent on the number of NNs that we are computing, and is independent of the number of processors we are using if we are using the gcc compiler. For the icc compiler we see a slight decrease in the gain when the number of processors is increasing for 20 million NNs, for 2 million NNs no clear pattern appears.

The performed process time measurements for 2 million NNs were repeated on the PC, the results are shown in the Figure 54 together with the process time results from code A and code B.

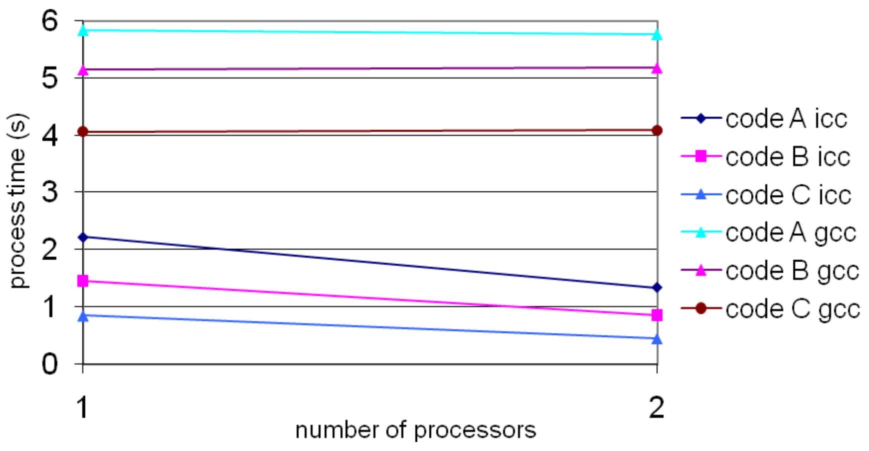


Figure 54: Process time for code A, B and C in function of the number of processors for icc and gcc compiling for 2 million NNs on the PC.

Like Figure 54 shows, we gain indeed a significant amount of process time by using the simplification of the calculation of the sigmoid. When we compare the results of the measurements performed on code C with those of code B we find a gain of process time of 41 % for the icc compiler and one processor, for two processors this value becomes 47 %. So after icc compiling the gain of process time increases when we are using more processors in parallel. After gcc compiling the gain of process time becomes 21 %, independent on the number of used processors.

If we change the threshold value of the sigmoid, the number of calculations that the computer has to make will change. So if we use a threshold of 10 for example the computer will have to make more calculations if we compare it with the used threshold of

4,6. On the other hand if we use a threshold of zero the computer has to make no calculation at all for the sigmoid function and thus we expect him to be faster. We did a measurement to see what the influence was of different values of the threshold of the sigmoid function on the process time. Figure 55 shows the results of this measurement.

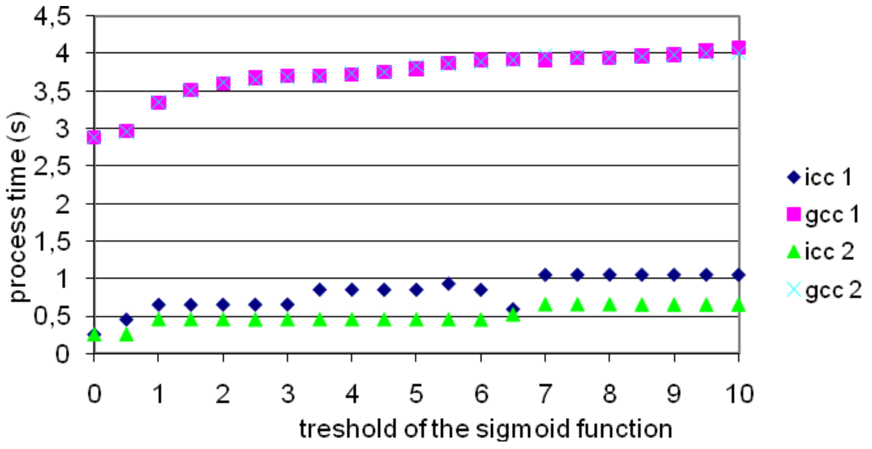


Figure 55: Process time of code C in function of the threshold of the sigmoid function for icc and gcc compiling and for 1 and 2 processors and for 2 million NNs.

The graph shows the process time after the icc and gcc compiling in the case of using one or two processors. The measurement has been carried out on the PC. The results of the measurement were like expected. The process time is indeed decreasing if the threshold of the sigmoid is decreasing and increasing if the threshold value is increasing. If we look for example to the line of the process time after icc compiling and running the code with two processors, we can see that with a threshold of 10 we have a process time of about 0,6 seconds. When we change the threshold to zero, we measure a process time of about 0,2 seconds, so we gain about 66 % of the process time.

But by using the threshold instead of calculating each value of the sigmoid function we introduce a certain error in our calculations. The smaller we make the interval in which we make the calculations the bigger the errors will become. So the errors will reach a maximum if we use a threshold of zero and will reach a minimum in infinity, so equal to not use the threshold value. In our measurements, using the threshold, the minimum error is achieved by using a threshold of ten.

To get an idea of the relative errors due to the use of the threshold instead of the total sigmoid function we adapted the code so it gives both results at the same time. The adaptation of the code, called ‘code D’, can be found in the appendix under A.8.

As can be found in the code, the results of the relative errors are given in a histogram. By fitting the wrapping curve of the histogram with a Gaussian function we obtain the FWHM value of the relative errors. Figure 56 shows the Gaussian fit for a threshold value of 5.

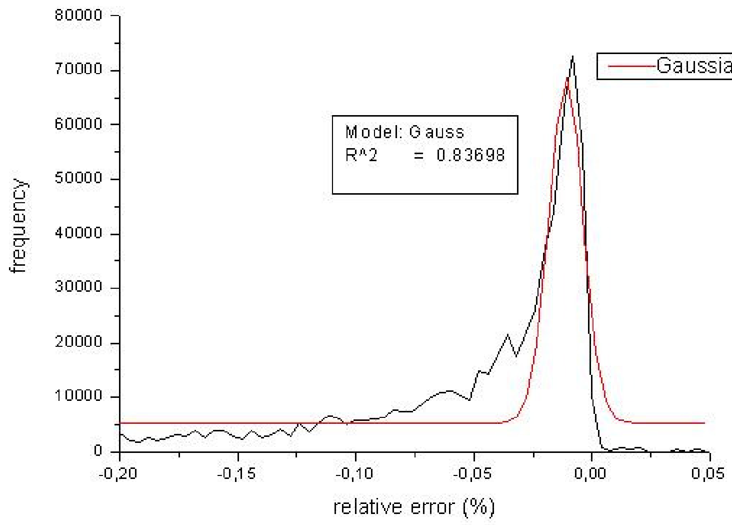


Figure 56: Gaussian fit of the histogram of the relative errors for a threshold of 5.

The results of these histograms were not as expected, we expected a Gaussian shape of the wrapping curve. But on Figure 56 we see that the errors are not symmetrical. There are much more negative relative errors than positive relative errors, meaning that in most cases the code that is using the threshold is producing results that are bigger than the code that is not using the threshold.

128 bits wide, so for example it can hold 4 floats at the same time. This because a float is 32 bits wide. So with this NN we expect that the process time will decrease even more. The code from Peter Bruyndonckx was written in another language, he used assembler. A code written in assembler consists out of series of instructions which correspond to a stream of executable instructions. When these instructions are loaded into an assembler they can be loaded into the memory and executed. Because this type of programming is difficult to learn and rather complicated we decided not to use this type of code but to use the much simpler C++ and C code. In Figure 58 we see a comparison of the speed achievable with the different codes.

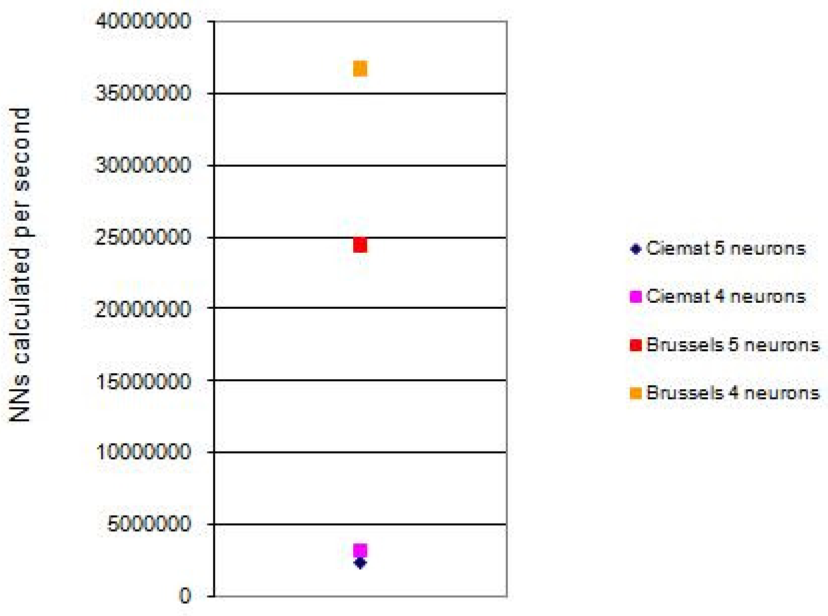


Figure 58: Comparison of the NNs calculated per second for the different codes.

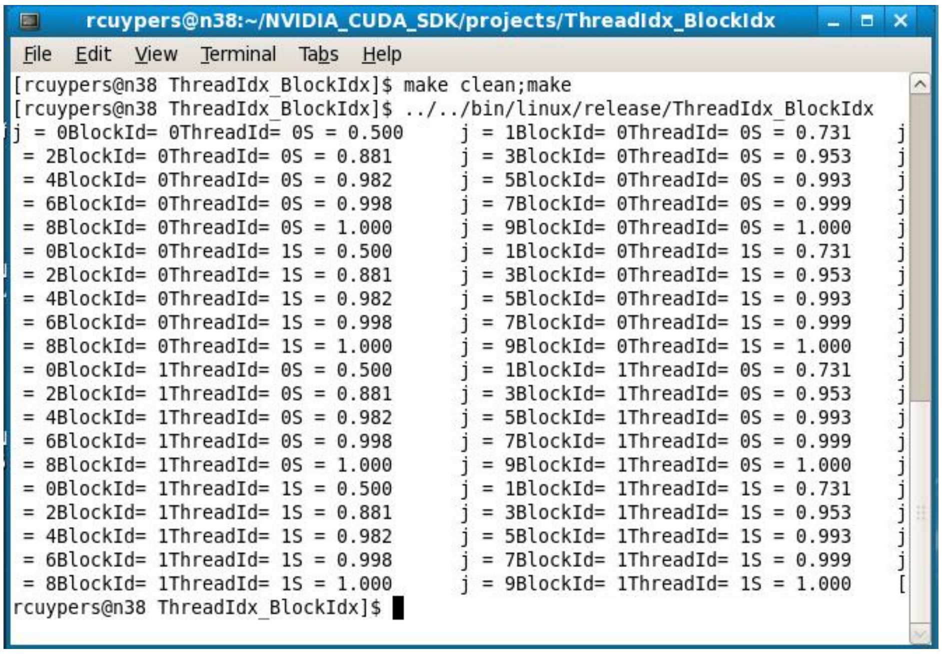
We can see that the codes from Brussels are still a lot faster than our codes, 11,3 times faster to be exact if we compare our fastest code with their fastest code. The difference in speed can be explained by the difference in the codes that were used. The assembler code is much faster than the codes written in C++ or C. If we transfer our fastest code to an assembler code we can achieve a speed 10 times bigger than we can achieve now, so comparable to the speed of the codes from Brussels. Our fastest code, the one using hidden layers of 4 neurons, achieves a speed of 3,2 million NNs per second. So larger than the demand of 2 million NNs per second.

4.4.1 Order of activity of thread blocks and threads

The first measurement we did was with a simple CUDA code. The goal of this measurement was to see how the emulator used the different blocks with their responding threads inside of them. To be more exact to see in what order the emulator was using them. The code can be found in the appendix under A.10.

As can be seen in the code we let the emulator calculate a sigmoid function, there was no special reason why we choose for this function. It was just to make sure the loop had to do some kind of calculation.

By using the commands blockIdx and threadIdx we are able to find out which thread and which thread block were active, every time the loop has been passed trough. Figure 59 gives the output of this program.


rcuypers@n38:~/NVIDIA_CUDA_SDK/projects/ThreadIdx_BlockIdx
File Edit View Terminal Tabs Help

```
[rcuypers@n38 ThreadIdx_BlockIdx]$ make clean;make
[rcuypers@n38 ThreadIdx_BlockIdx]$ ../../bin/linux/release/ThreadIdx_BlockIdx
j = 0BlockId= 0ThreadId= 0S = 0.500     j = 1BlockId= 0ThreadId= 0S = 0.731    j
 = 2BlockId= 0ThreadId= 0S = 0.881     j = 3BlockId= 0ThreadId= 0S = 0.953    j
 = 4BlockId= 0ThreadId= 0S = 0.982     j = 5BlockId= 0ThreadId= 0S = 0.993    j
 = 6BlockId= 0ThreadId= 0S = 0.998     j = 7BlockId= 0ThreadId= 0S = 0.999    j
 = 8BlockId= 0ThreadId= 0S = 1.000     j = 9BlockId= 0ThreadId= 0S = 1.000    j
 = 0BlockId= 0ThreadId= 1S = 0.500     j = 1BlockId= 0ThreadId= 1S = 0.731    j
 = 2BlockId= 0ThreadId= 1S = 0.881     j = 3BlockId= 0ThreadId= 1S = 0.953    j
 = 4BlockId= 0ThreadId= 1S = 0.982     j = 5BlockId= 0ThreadId= 1S = 0.993    j
 = 6BlockId= 0ThreadId= 1S = 0.998     j = 7BlockId= 0ThreadId= 1S = 0.999    j
 = 8BlockId= 0ThreadId= 1S = 1.000     j = 9BlockId= 0ThreadId= 1S = 1.000    j
 = 0BlockId= 1ThreadId= 0S = 0.500     j = 1BlockId= 1ThreadId= 0S = 0.731    j
 = 2BlockId= 1ThreadId= 0S = 0.881     j = 3BlockId= 1ThreadId= 0S = 0.953    j
 = 4BlockId= 1ThreadId= 0S = 0.982     j = 5BlockId= 1ThreadId= 0S = 0.993    j
 = 6BlockId= 1ThreadId= 0S = 0.998     j = 7BlockId= 1ThreadId= 0S = 0.999    j
 = 8BlockId= 1ThreadId= 0S = 1.000     j = 9BlockId= 1ThreadId= 0S = 1.000    j
 = 0BlockId= 1ThreadId= 1S = 0.500     j = 1BlockId= 1ThreadId= 1S = 0.731    j
 = 2BlockId= 1ThreadId= 1S = 0.881     j = 3BlockId= 1ThreadId= 1S = 0.953    j
 = 4BlockId= 1ThreadId= 1S = 0.982     j = 5BlockId= 1ThreadId= 1S = 0.993    j
 = 6BlockId= 1ThreadId= 1S = 0.998     j = 7BlockId= 1ThreadId= 1S = 0.999    j
 = 8BlockId= 1ThreadId= 1S = 1.000     j = 9BlockId= 1ThreadId= 1S = 1.000    [
rcuypers@n38 ThreadIdx_BlockIdx]$
```

Figure 59: Output of the ThreadIdx_BlockIdx code.

We see that the emulator first uses the first thread of the first thread block, followed by the second thread of the first thread block. After the loops are finished for every thread of the first thread block, he switches to the next thread block where he does the same as in the first thread block. This measurement can only be carried out while using the emulator. This because we are using printf commands in the kernel and this is not allowed while we are working with the graphical card instead of the emulator. So while we are using the real graphical card it is not possible to let the code tell you in which order which thread block and which thread in that block is active.

4.4.2 Thread vs. thread block combinations

In the first measurement we saw that with two thread blocks, each consisting out of two threads the kernel was executed 4 times. But to reach this number of executions there are several possibilities, using two blocks with each two threads is just one option. A different option would be to use four threads in one thread block. So if we have to execute our kernel 100 times for a specific application then the number of possible combinations to reach this amount of executions would be 9.

We want our code to be as fast as possible, so we had to find out which combination of threads and thread blocks was the fastest.
We used the code and the settings from the first experiment, but now we made combinations so that the number of executions of the kernel was always equal to 100. We also changed the number of iterations of the for loop. We let every execution of the kernel, execute the for loop 100000 times instead of 10 times, this to have a process time big enough to see any changes. When we plot the results of both measurements in one diagram we obtain Figure 60. The measurement was carried out with and without using the emulator, to be able to compare the results of both measurements.

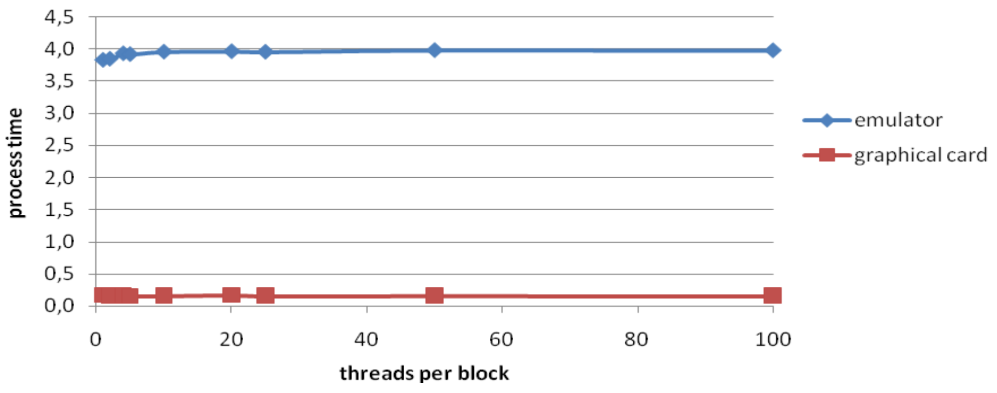


Figure 60: Comparison of the process times for the emulator and the graphical card in function of the number of threads per block.

We see that the process time is as good as a constant value for the measurement carried out with the emulator as well as for the measurement carried out with the graphical card. Further it is clear that the graphical card is much faster than the emulator. The emulator has an average process time of 3,9 seconds where the graphical card only needs an average process time of 0,15 seconds, this is 26 times faster. When we look closer to the results of each measurement we obtain Figure 61 and Figure 62.

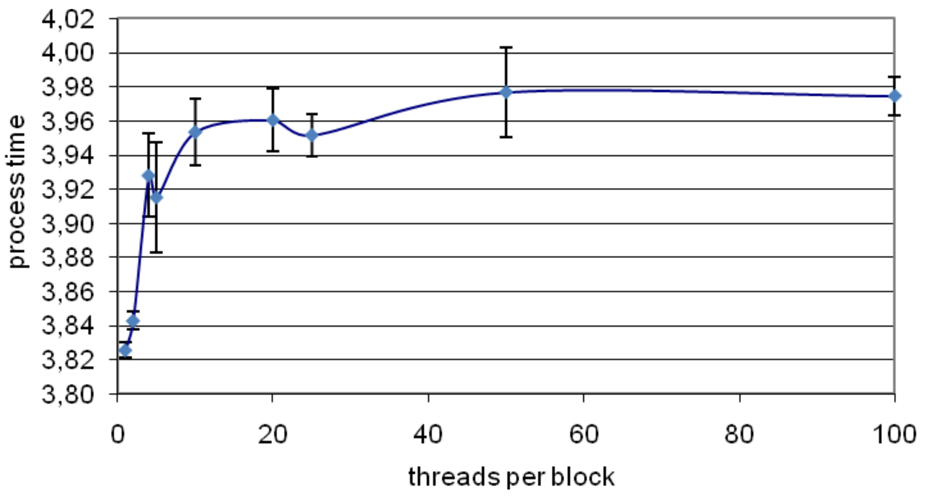


Figure 61: Process time of the emulator in function of the number of threads per block

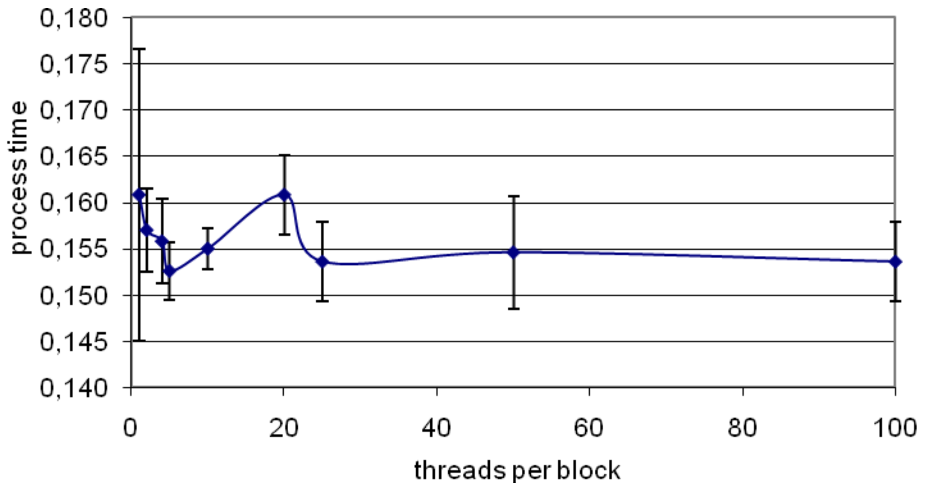


Figure 62: Process time of the graphical card in function of the number of threads per block.

In Figure 61 we see that the process time from the emulator is slightly increasing if we increase the number of threads in each thread block. On the diagram the length of the error and another version for icc, so we could compile it with gcc and with icc alone, making sure the calculations that had to be done were exactly the same to be able to compare the data afterwards. So eventually we had one calculation that could be executed after compiling with 4 different compilers. This gave us the possibility to compare the process time of 6 different ways of running the code, 6 because we can use the nvcc compiler for the emulator and for the real graphical card. Figure 63 shows the results of these measurements.

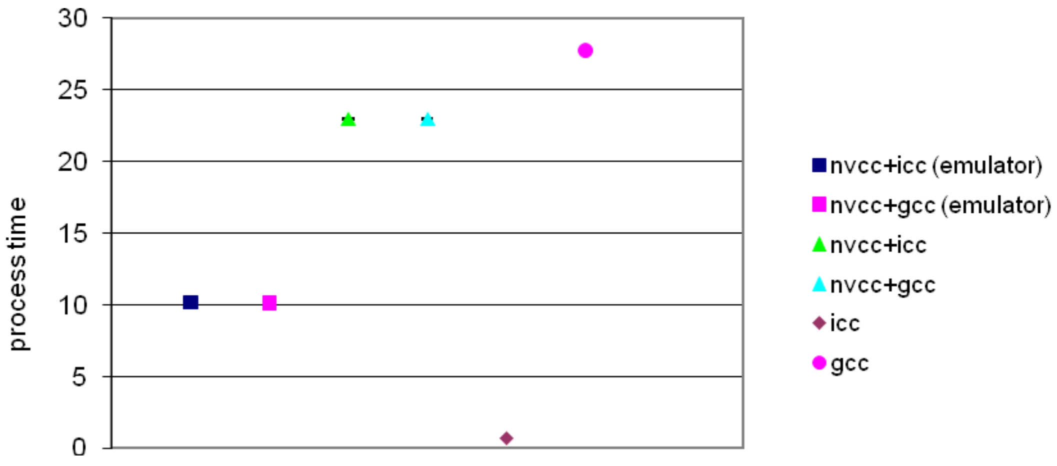


Figure 63: Process time in function of the usage of different compilers.

The result of this measurement was not as we had expected. Because of the previous measurement where it was clear that the real graphical card was faster than the emulator we expected to find something similar. But on the diagram from Figure 63 we see that the process time while using the emulator is smaller than the process time while using the real graphical card. Further we can see that it does not matter which compiler we use in combination with the nvcc compiler, both compilers give the same results. The process time is the smallest after compiling the code with the icc-compiler, the biggest process time we obtain while using the gcc compiler. The uncertainties of these measurements were all smaller than 0,005 seconds, so the error bars are almost invincible on the diagram. With these measurements we have to note that the process times after compiling the code with the different compilers are probably strongly dependent on the source code, so it is dangerous to make general conclusions out of this type of measurements for other codes.

channel analyzer. With this analyzer we can set a window on the incoming signal that we are allowing to pass. By adjusting the different parameters of the channel analyzer we are able to allow only the energy peak of the 511 keV photons to pass through the channel analyzer.

The next device in the channel is the VME optical link, this device will use the signal from the channel analyzer as a trigger to open the gate for the ADCs to acquire the data. From here on the setup works again in the same way as it did in the first setup, see paragraph 3.1.

More detailed information about the equipment used in this setup can be found in the appendix under A.13.

To see if there is a correlation between the channel values of the different rows and columns and the position of a radioactive source on top of the detector box we build a setup as shown in Figure 65. Figure 66 is a picture taken from the experimental setup itself.

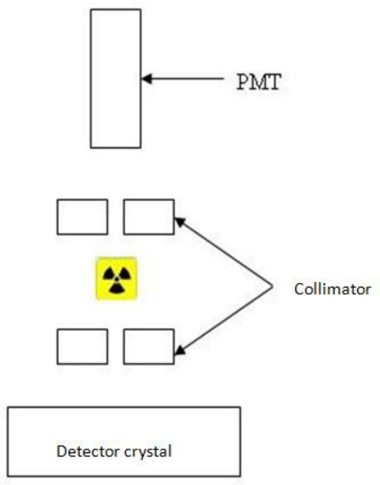


Figure 65: Schematic diagram of the PMT, the collimator and the radioactive source.

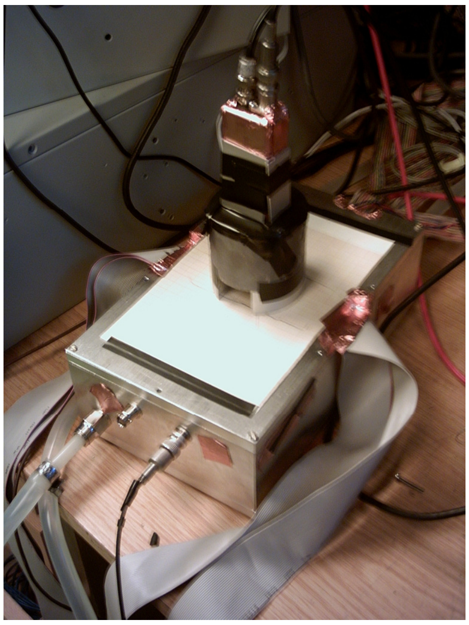


Figure 66: Picture of the experimental setup.

The collimator is built in such a way that we are able to place a radioactive source inside of it and that we are able to fix a PMT on top of it. By moving the collimator, with the radioactive source inside of it and the PMT on top of it, over the detector box and study the variations in the values of the different ADC channels, we hope to see a correlation between the spatial position of the radioactive source and the values in the different channels. We moved the combination in 5 steps of 1 mm over the detector box, making a cross over the center of the detector box, as shown in Figure 67.

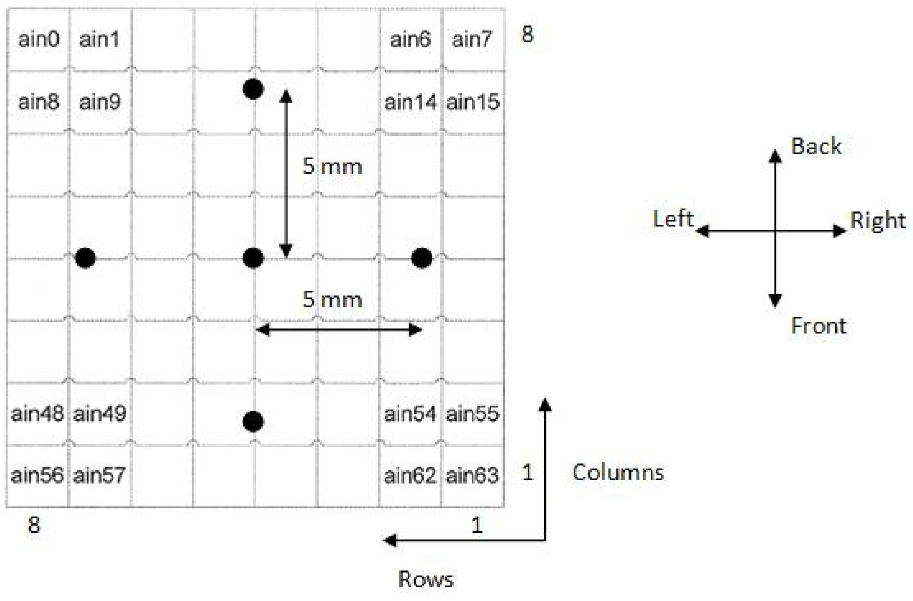


Figure 67: Mapping between the detector and the ASIC, and the direction of the steps, based on [Pe07, 8].

4.5.2 Correlation between ADC channels and the position of the radioactive source

The mean of the output values of the different columns and rows in a diagram for the different beam positions is shown in Figure 68 for the beam positioned in the front, in Figure 69 for the beam positioned in the back, in Figure 70 for the left side, in Figure 71 for the right side and in Figure 72 for the beam positioned in the center.

- Front

Figure 68 is showing the values of the ADC channels for the 5 steps of 1 mm to the front of the detector box.

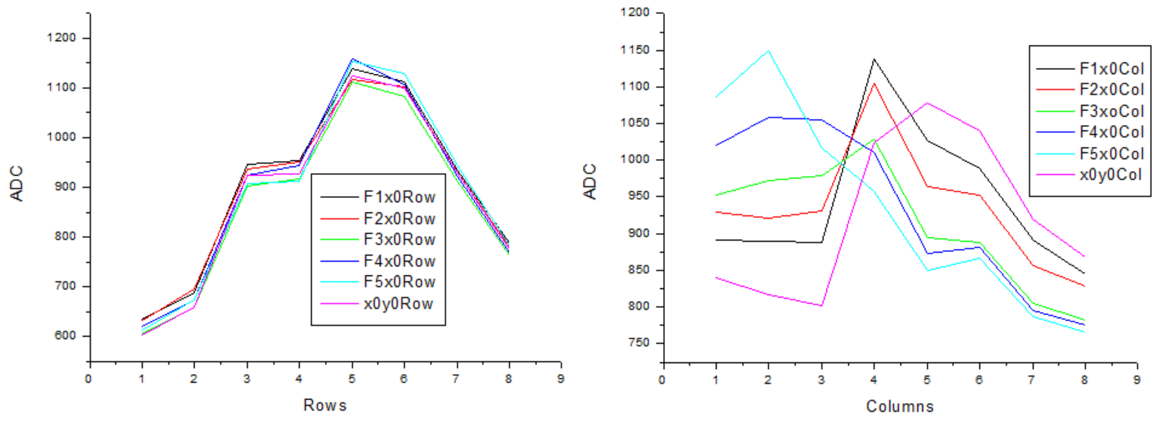


Figure 68: ADC values of the different rows and columns in the "front" position.

When we look to the diagram of the rows, we see that there is a clear difference between the left part of the graphs and the right part of the graphs. We were expecting that those two parts were similar because the source remains in the middle of the rows. This can be explained by the fact that the APDs under the detector block are split up into two arrays of APDs and both arrays have a slightly different gain if the same voltage is applied to them. In this case the right array of APDs appears to have a slightly higher gain than the left array of APDs. In the next versions of the detector box we will make sure we choose 2 arrays of APDs which have a comparable gain, so that this difference is eliminated. Further it is logic that the graphs of the different steps are matching because the source is placed in steps of 1 mm to the front so it remains in the middle of all the rows. We can note that we achieved similar patterns like the theoretical patterns discussed in paragraph 2.3.

The columns values are clearly changing when we move the source. We see that the peak is moving from the center to the left of the graph exactly like we expected. Further we notice that the peak of the center is not matching the middle column but that it is slightly shifted to the right. This is probably due to not very accurate placing of the collimator. After all the collimator was placed on top of a paper with separations of one millimeter by hand.

- Back

In Figure 69 the results of the measurements performed while moving the source in steps of 1 mm to the back of the detector box are shown.

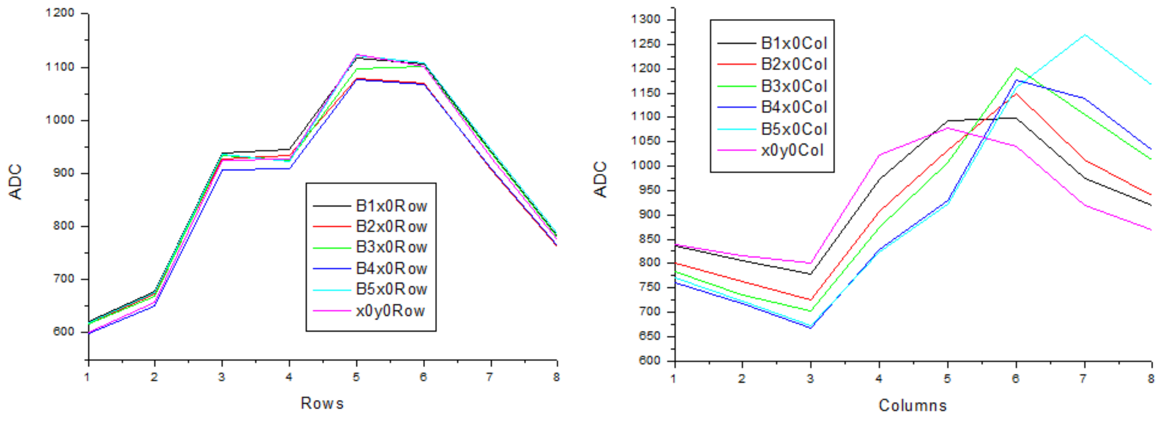


Figure 69: ADC values of the different rows and columns in the “back” position.

The remarks we made for the steps to the front are also true for the steps to the back. In the diagram of the columns the shift of the peak to the side is more clearly seen than in the diagram of the columns of the steps to the front.

- Left

Figure 70 shows both the diagrams of the values of the ADC channels from the columns and the rows for the 5 steps of 1 mm to the left of the detector box.

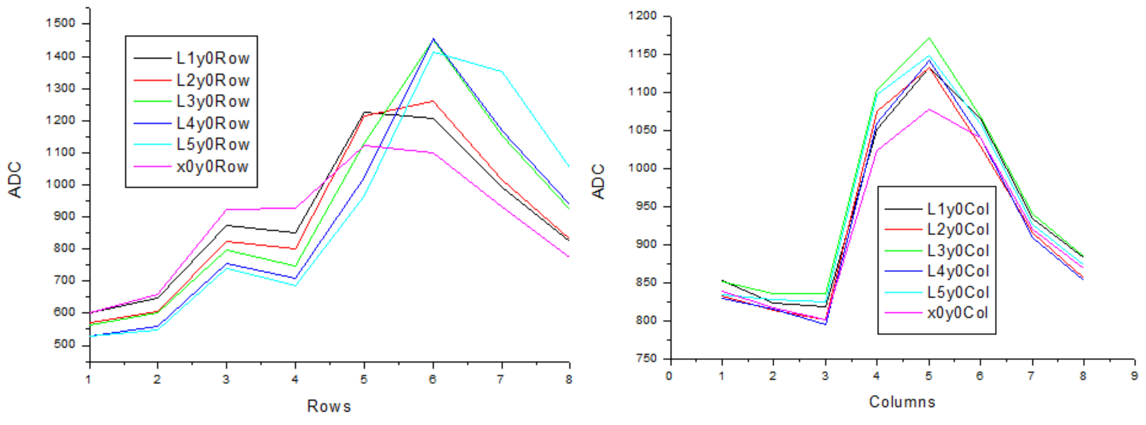


Figure 70: ADC values of the different rows and columns in the "left" position.

Now we notice a shift of the peak in the diagram of the rows, this because we are moving the source in the direction of the rows. Again we see the effect of the difference in gain between the two APD arrays in the diagram of the rows. And as expected, the values of the ADCs corresponding to the columns remain approximately constant.

- Right

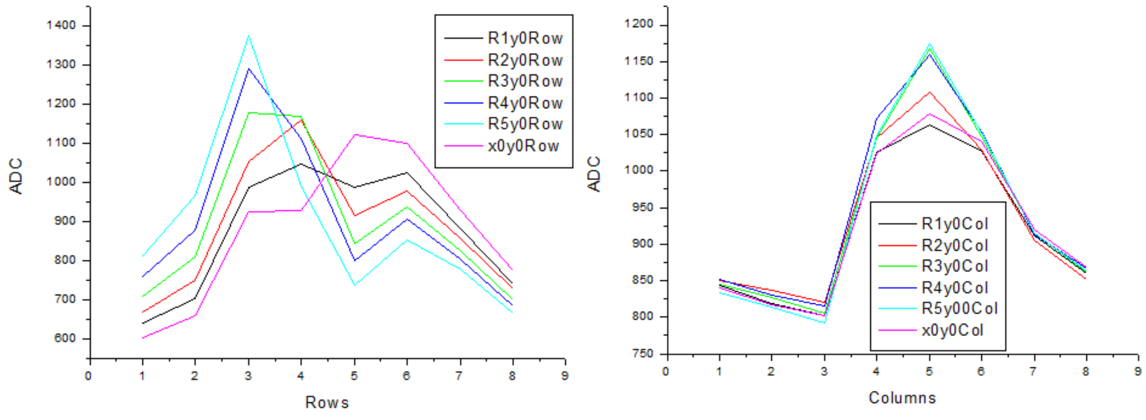


Figure 71: ADC values of the different rows and columns in the "right" position.

When moving the collimator together with the radioactive source and the PMT to the right of the detector box the same remarks can be made as for the movement to the left.

- Center

To see if there was a difference between the values of the ADCs in time, we performed an identical measurement twice, once in the beginning of all the measurements and the second one at the end of all the measurements. Figure 72 shows the results of those two measurements.

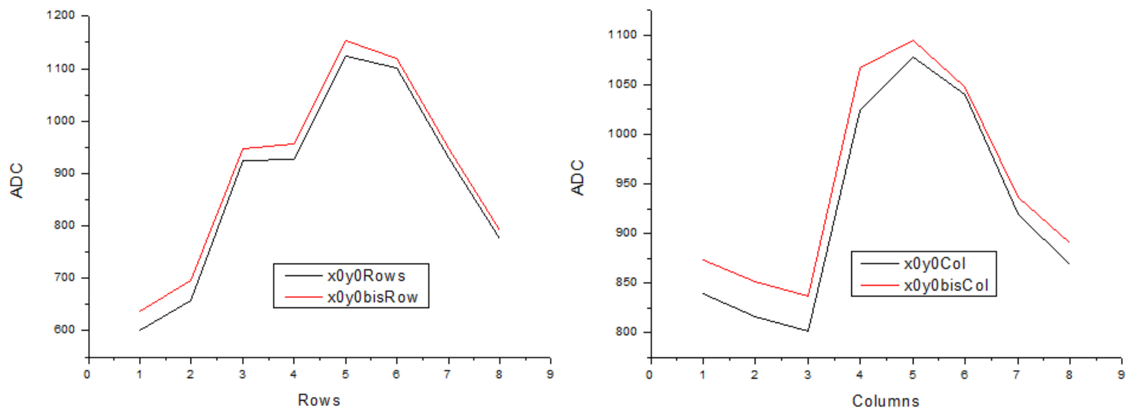


Figure 72: ADC values of the different rows and columns in the center.

The measurements were carried out with the radioactive source placed on top of the center of the detector box. Both diagrams give comparable results, with the ADC outputs being slightly higher in the second measurement. The reason for this is probably due to a different temperature in the detector box during the measurement, although the detector box is cooled. Such a different temperature leads to an increase of the gain of the APDs.

From all these measurements we can conclude that there is indeed a correlation between the spatial position of the radioactive source and the outputs of the ADC values of row and column data. After all when the source is moved on top of the detector box we can tell by looking to the outputs of the ADCs in what direction the source has moved.

4.5.3 Spatial resolution of both setups

The data gathered from experiments like those discussed in paragraph 4.5.2, are used to train the NN further and to control the correct functioning of the NN. One NN uses the values of the 8 rows to provide us with an estimation of the *x* coordinate of the impinging photon, a second NN uses the 8 columns to provides us an estimation of the *y* coordinate.

In this paragraph the relative errors of the estimations of the NNs will be discussed for the setup shown in Figure 66 and the 'real' setup, shown in Figure 32.

4.5.3.1 Spatial resolution of the first setup

When we make a histogram of the relative errors of all the coordinates provide by the NN in the *x* direction we obtain Figure 73.

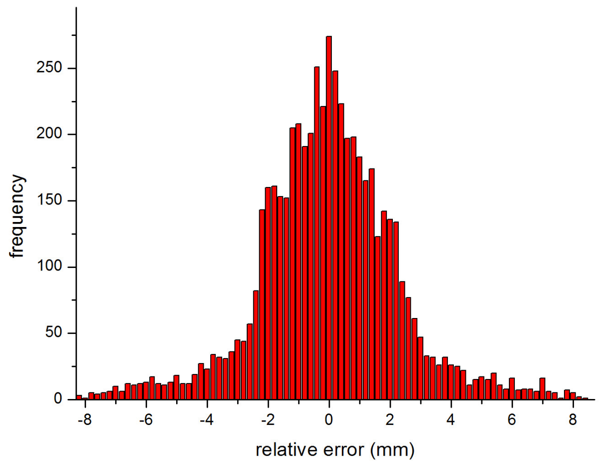


Figure 73: Histogram of all the relative errors of the estimations of the beam position by the NN in the *x* direction.

When we do the same for the *y* coordinates we obtain Figure 74.

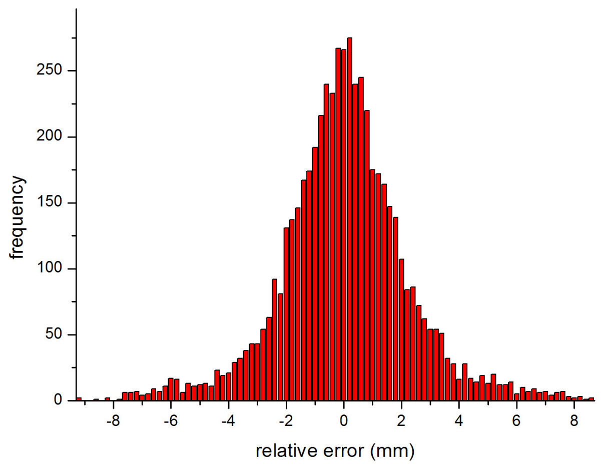


Figure 74: Histogram of all the relative errors of the estimations of the beam position by the NN in the *y* direction.

After fitting these histograms with a Gaussian curve we find a FWHM value of 4,0 mm in the *x* direction and a FWHM value of 3,7 mm in the *y* direction. Which are quite bad if you consider that we worked in all the measurements in steps of 1 mm. A shift in the position of the centre of the distribution of the different histograms of the relative errors is the cause of this wide FWHM value.

However if we look to the FWHM values of the relative errors of the estimations of the different measurements we see that in general they are smaller. Figure 75 shows how the FWHM values of the relative errors are distributed in function of the beam position on the detector block. Figure 76 shows the shift of the peak in the histogram of the relative errors.

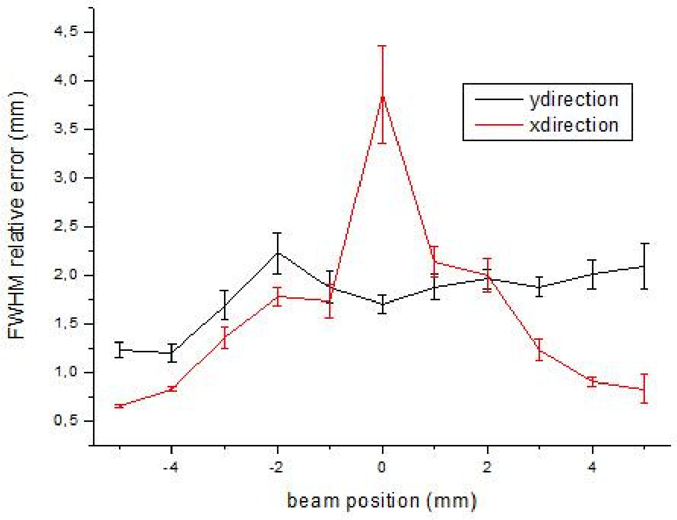


Figure 75: FWHM of the relative errors in function of the beam position in the *x* and *y* direction.

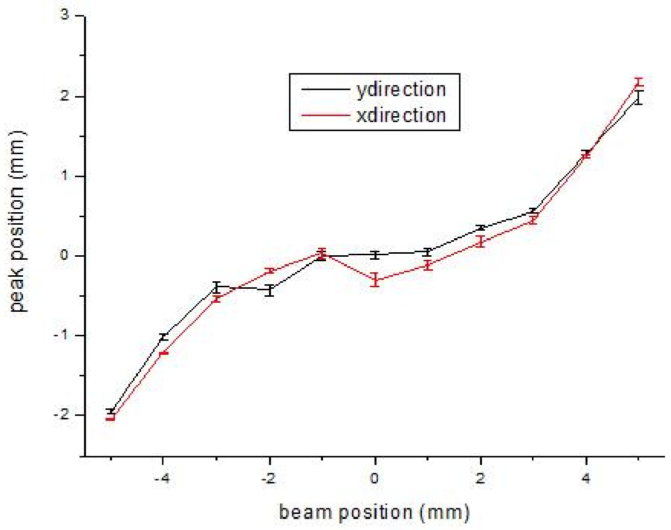


Figure 76: Position of the peak of the histogram of the relative errors in function of the beam position in the *x* and *y* direction.

The amplitude of the FWHM values of the relative errors are in both cases of a comparable size, most of them have an amplitude around 2 mm or less. In the *x* direction we obtain an average FWHM value of 1,6 mm, for the *y* direction this value becomes 1,9 mm. Knowing that the setup positioning was not very accurate, these results are satisfying.

It is clear that the shift of the peaks of the histograms of the relative errors increases when we move further away from the centre of the detector block, in the centre this shift is minimal. When this shift is known we can compensate for it, so it does not necessarily means a big problem.

4.5.3.2 Spatial resolution of the 'real' setup

We can perform the same study of the NN estimations when the NN is using the data obtained from the 'real' setup, shown in Figure 32. Again the scans were performed using a detector block with a Teflon coating. When we look into the results of the estimations of the NN we can compare the performances of both setups. Figure 77 and Figure 78 are showing the results of the NN estimations based on the data obtained on the 'real' setup.

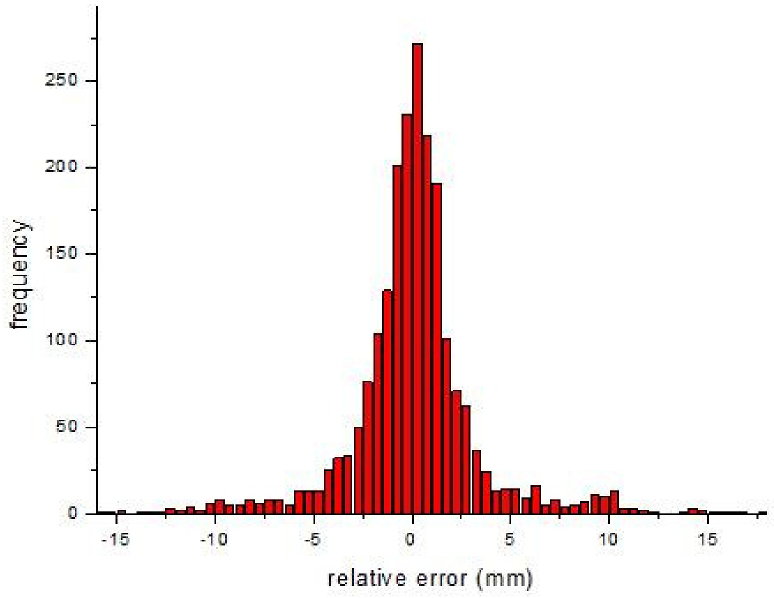


Figure 77: Histogram of all the relative errors of the estimations of the beam position by the NN in the *x* direction.

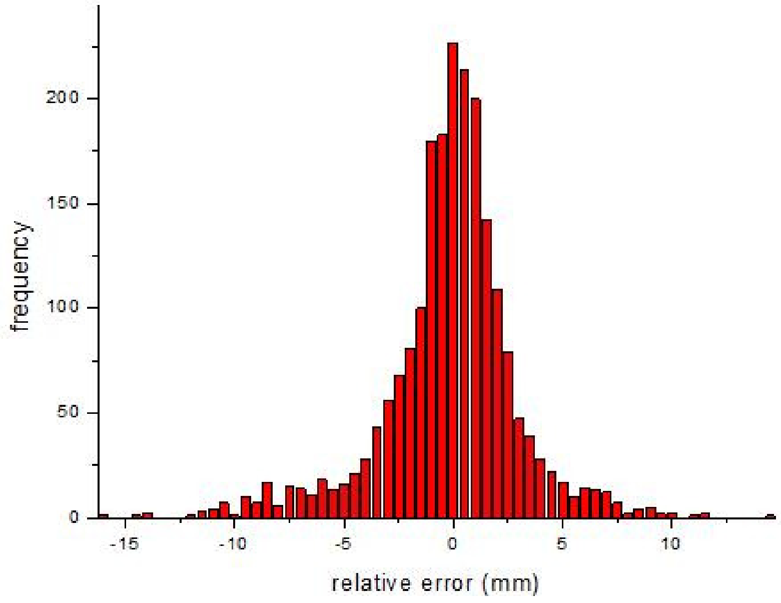


Figure 78: Histogram of all the relative errors of the estimations of the beam position by the NN in the *y* direction.

When fitting these histograms with a Gaussian curve we find in the *x* direction a FWHM of the relative errors of 3,7 mm and in the *y* direction this becomes 3,0 mm. These values are slightly better than the values obtained with the other setup, being 4,0 mm in the *x* direction and 3,7 mm in the *y* direction.

When we study the FWHM values of the relative errors of the estimations for the different beam positions we obtain Figure 79.

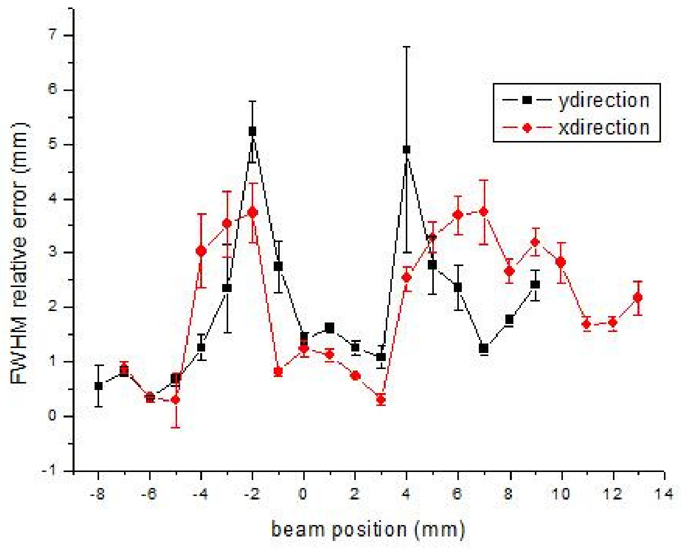


Figure 79: FWHM of the relative errors in function of the beam position in the *x* and *y* direction.

We notice that the FWHM values of the relative errors of the estimations performed by the NN in the *y* direction are wider spread than those in the *x* direction. The average values are 2,0 mm for the *x* direction and 1,9 mm in the *y* direction. For the *x* direction as well as for the *y* direction we see that we have a wider variation in the FWHM values when we use this setup compared with the values discussed in paragraph 4.5.3.1. The average FWHM values of both directions are comparable for both setups, 1,6 mm instead of 2,0 mm for the *x* direction and 1,9 mm for both setups in the *y* direction.

For the ‘real’ setup we can also compare the variations in the position of the peaks of the different histograms of the relative errors in function of the beam position, in doing so we obtain Figure 80.

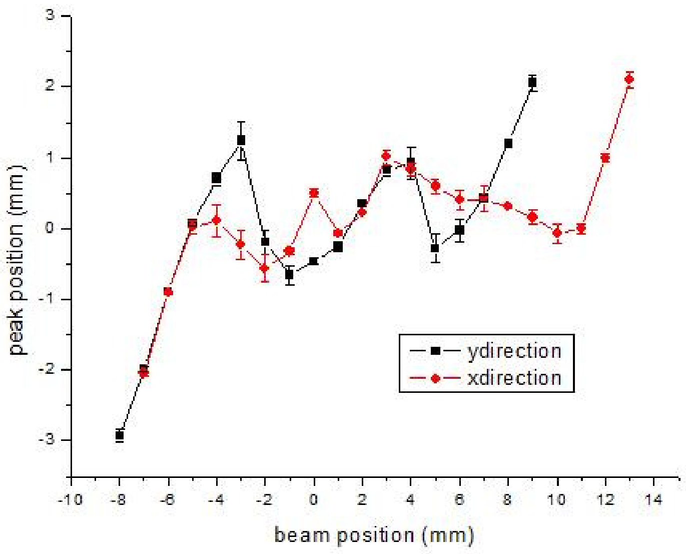


Figure 80: Position of the peak of the histogram of the relative errors in function of the beam position in the *x* and *y* direction.

When comparing Figure 80 with Figure 76 we notice that the curves have essentially the same shape. Although the curves in Figure 76 are smoother and the pattern is more clear, we see the same effect. The shift of the peak positions increases the further we move away from the centre of the detector block. With this ‘real’ setup it will be more difficult to compensate for these non linearities in the estimations of the NN because the shape of the curve is not that clear, so the deviations of the estimations from the true values will be harder to predict.

4.6 Comparison of the spatial resolution of the Teflon wrapped detector block and the white painted detector block

From the data obtained by the scan in the *y* direction performed on the detector block wrapped in Teflon and the white painted detector block, we can use NNs to make the corresponding estimations of the beam positions and study the differences between the results for the two detector blocks. The measurements were performed on the 'real' setup which allows us to use two detector boxes in coincidence, this setup is shown on Figure 32.

Figure 81 shows the histograms of the relative errors of the estimations of the beam positions of the full scans for both detector blocks.

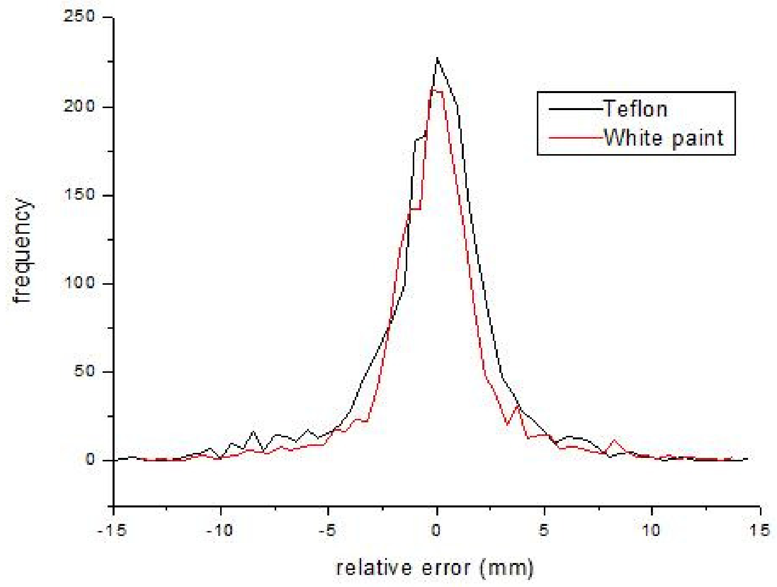


Figure 81: Histograms of the relative errors of the estimations of the beam positions by the NN for the white painted detector block and the Teflon wrapped detector block.

When we fit those graphs with a Gaussian function we find a FWHM value of 3,8 mm for the Teflon wrapped detector block and a value of 3,4 for the FWHM value of the white painted detector block. So the white painted detector block provides us with slightly better results when we look at these results.

When we study the FWHM values of the relative error histograms for all the different beam positions we obtain Figure 82.

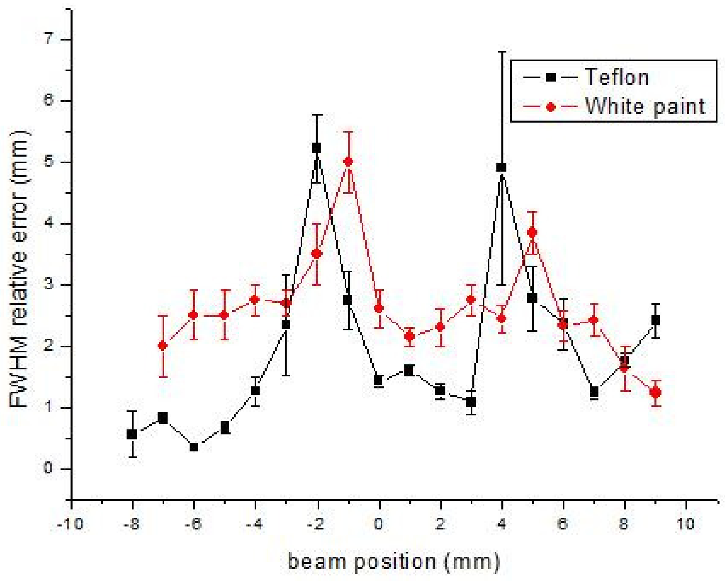


Figure 82: FWHM of the relative errors in function of the beam position for the Teflon wrapped detector block and the white painted detector block.

When we look to the graphs we notice that for most beam positions the FWHM values of the white painted detector block are bigger than the ones we found with the Teflon wrapped detector block for the corresponding beam position. The average of the FWHM values for the Teflon wrapped detector block is 2,0 mm and for the white painted detector block this average value is 2,6 mm. So unlike for the results we found in the histograms for the relative error of the full scan where the white painted block had the better results, we find now that the Teflon wrapped detector block gives better FWHM values of the relative errors.

Another parameter useful for comparison purposes is the shift in the peak position of the histograms of the relative errors for every peak position. Figure 83 shows this shift in the peak positions.

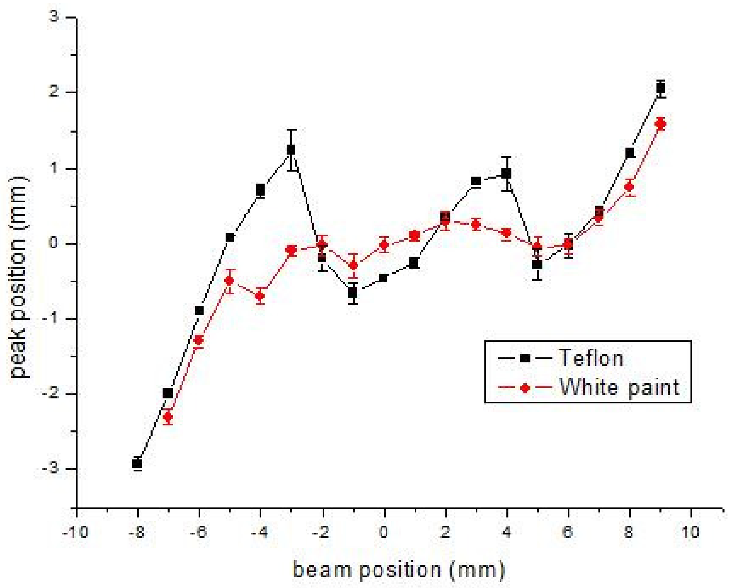


Figure 83: Position of the peak of the histogram of the relative errors in function of the beam position for the Teflon wrapped detector block and the white painted detector block.

We see that the shift of the peak positions of the white painted detector block gives a much smoother pattern than we find for the Teflon wrapped detector block. So it will be much easier to compensate for the nonlinearities when using the white painted detector block than for the Teflon wrapped detector block.

However the compensation for the nonlinearities for the white painted detector block is the easiest, in the end the Teflon wrapped detector block gives the best overall spatial resolution. It provides us with about 23 % better spatial resolution than the white painted detector block.

5.1 Scan in the *y* direction

Figure 84 shows the results of the comparison between the FWHM values of the peaks in the ADC outputs of both measurements in the *y* direction.

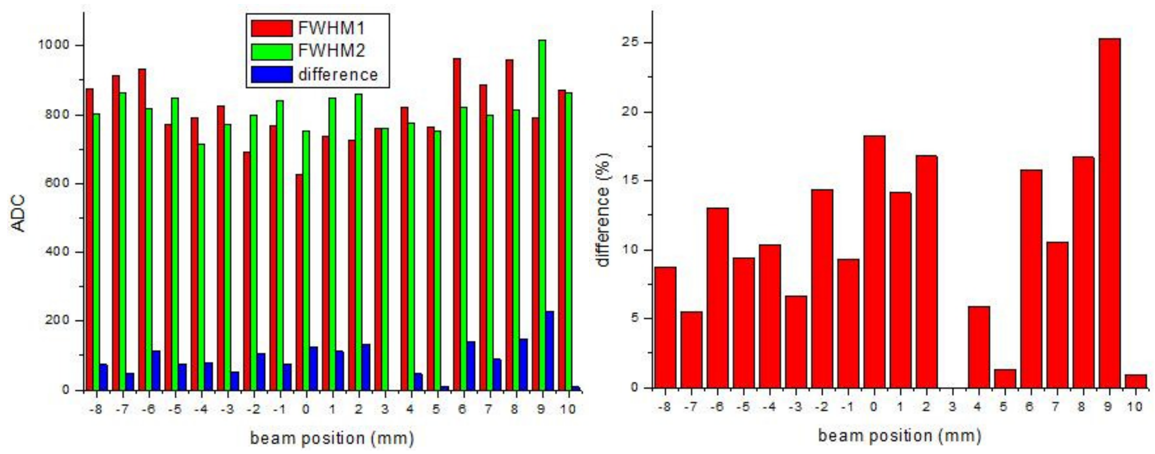


Figure 84: Left: FWHM values of both measurements and the difference between these two values in function of the beam position in the *y* direction; Right: The difference between the FWHM values of both measurements as a percentage of the average value for every beam position in the *y* direction.

We see in Figure 84 on the left side that both measurements show a similar pattern, the smallest FWHM values occur in the center, the largest ones appear on the edges. Further we see that there is no clear pattern in the differences between both measurements. The difference is in most cases limited to less than 15 %, with some exceptions until a maximum of 25 %.

Because the FWHM values are proportional to the parameter peak width, defined as discussed in paragraph 3.2.3, we expect a similar pattern for this parameter. The required parameters of the setup remained stable during the period of both measurements so in this case we can use the peak width parameter for comparison. Figure 85 shows the peak width values at their corresponding beam positions.

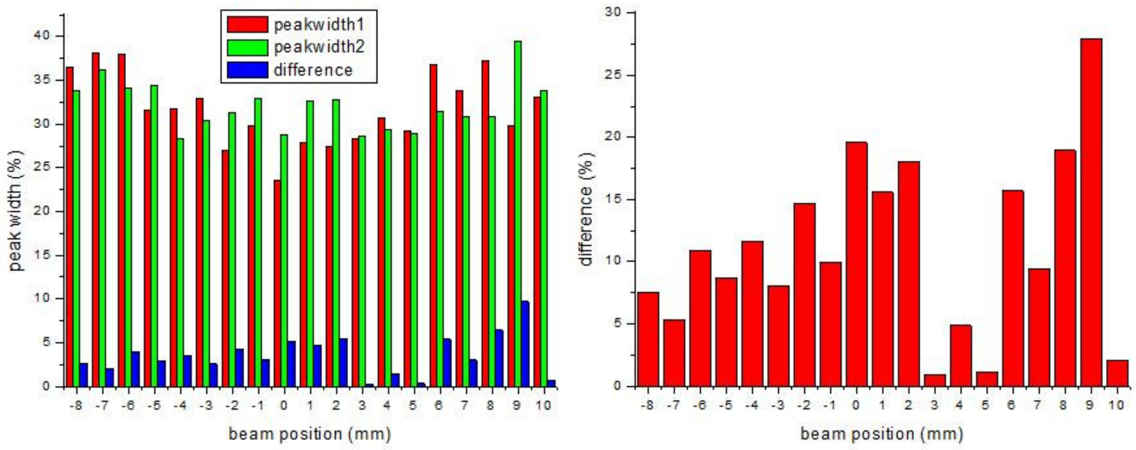


Figure 85: Left: Peak width values of both measurements and the difference between these two values in function of the beam position in the *y* direction; Right: The difference between the peak width values of both measurements as a percentage of the average peak width for every beam position in the *y* direction.

As expected we see indeed the same pattern appear in the diagrams. In general we can say that the difference expressed in percentages of the peak width values for the different beam positions are comparable to the differences of the FWHM values also expressed in percentages. The difference in the peak width values between both measurements is in most cases below 10 %. Knowing that the peak width values are of the order 30 %, the variations are within a region of 3-4 % which is quite reasonable.

Another important parameter for the NN to base its estimations on is the position of the sum peaks. A decrease of those peaks could mean a decrease of the collected light whereas the opposite, an increase of the peak position could be a result of an increase of the collected light. When we look to the difference in the peak positions in the sum of the column channels of both scans in the *y* direction we obtain Figure 86.

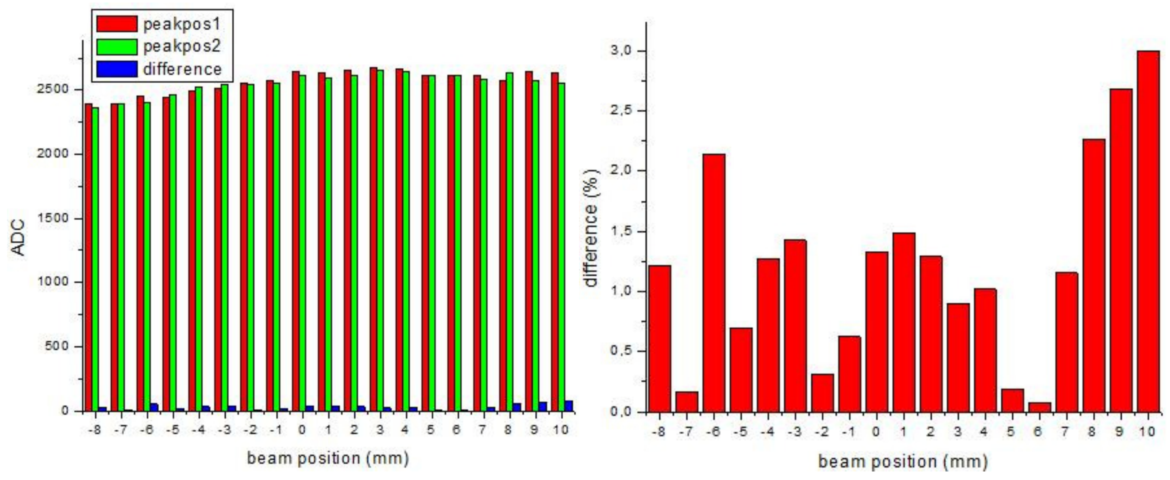


Figure 86: Left: Peak positions of both measurements and the difference between these two values in function of the beam position in the *y* direction; Right: The difference between the peak positions of both measurements as a percentage of the average value for every beam position in the *y* direction.

We notice that the peak positions of both measurements are quite similar over the entire surface of the detector block. The percentage of the differences between the peak values of both measurements are in all cases smaller than 3 %. Knowing that there was a time interval of about 4 days between both measurements this is quite satisfying.

We can also compare the mean values of the columns and rows from both measurements. For the scans in the *y* direction the values in the columns are representative, Figure 87 shows these representative values for the *y* direction.

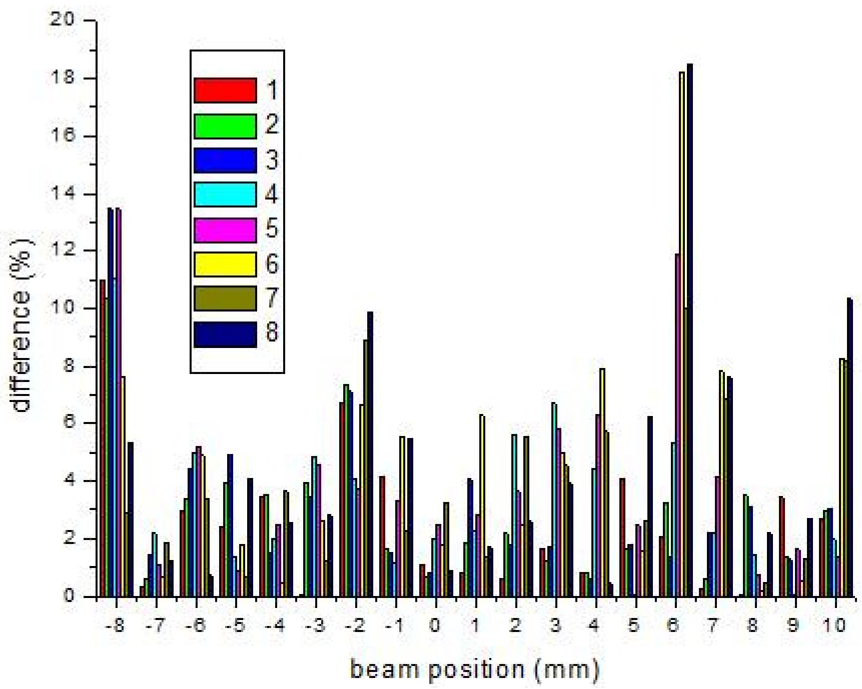


Figure 87: Difference in percentages between the mean values of the different 8 columns for every beam position along the *y* direction.

We notice that the differences between the mean values of the columns are in general smaller than 6 % with exceptions at the borders of the detector block and at beam positions -2 and 6. Because the amplitude of the ADC outputs at the edges of the detector block are smaller, a small variation in the outputs will provide us with a great difference when we look at it in percentages. At positions -2 and 6 we can explain the exceptional great percentages by local fluctuations in the ADC output values.

When we compare the estimations performed by the NN for this direction we can study the FWHM values of the peaks in the histograms of the relative errors and the positions of those peaks. Figure 88 shows the comparison of the FWHM values of the peaks in the histograms of the relative errors.

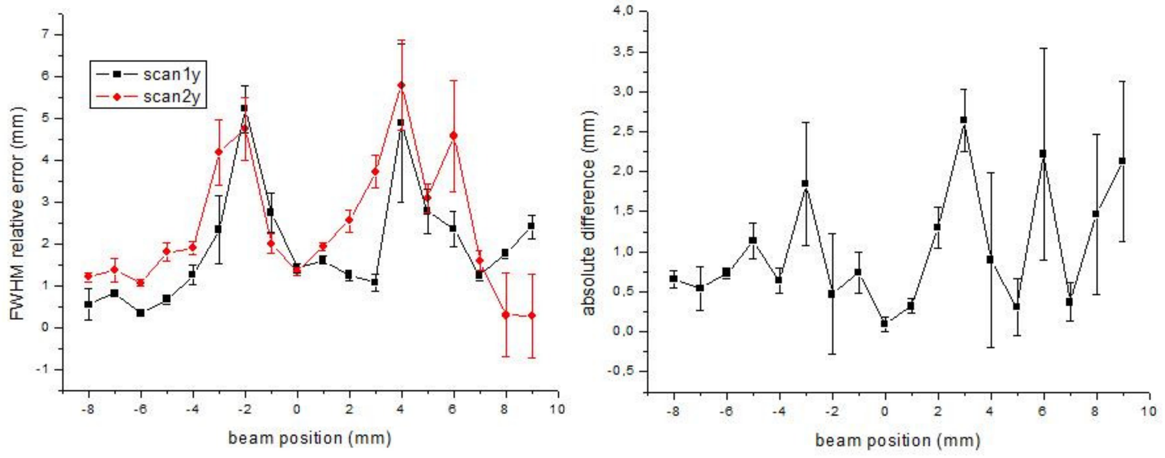


Figure 88: Left: FWHM of the relative errors of the NN estimations of both scans in function of the beam position for the scan in the *y* direction; Right: The difference between both estimations in absolute values in function of the beam positions.

We notice that the same pattern occurs in the graph of the FWHM values for both scans. The absolute value of the difference between the FWHM values of both scans for every beam position is at maximum 2,6 mm. Most differences however are smaller than 1mm. The FWHM value of the entire relative error histogram has a FWHM value of 3,8 mm for the first scan and 3,5 mm for the second scan. So the spatial resolution of the setup in the *y* direction is quite stable. In Figure 89 the peak positions of the NN errors for both scans and the difference between those values are shown.

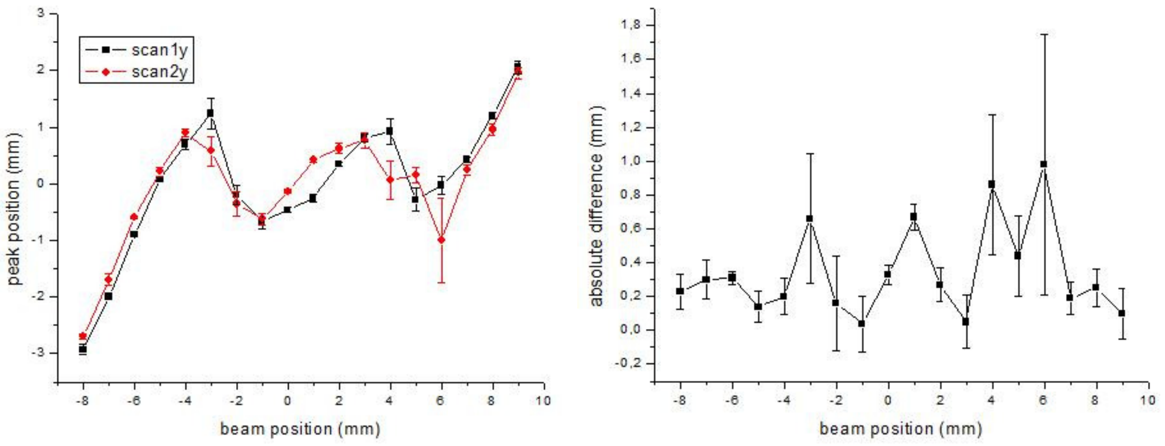


Figure 89: Left: peak positions of the NN errors of both scans in function of the beam position for the scan in the *y* direction; Right: The difference between both peak positions in absolute values in function of the beam positions.

In both scans the same pattern occurs when we look to the peak position of the relative errors of the NN estimations in function of the beam position. The maximum absolute difference between the two scans for a certain beam position is 0,98 mm. Most differences are even smaller than 0,5 mm. So after the FWHM values of the relative errors of the NN we can say that also the peak positions of those relative errors in the *y* direction are quite stable.

5.2 Scan in the *x* direction

For the scan in the *x* direction we can examine the same parameters as we did for the *y* direction. Figure 90 shows the FWHM values of the sum peaks in the *x* direction.

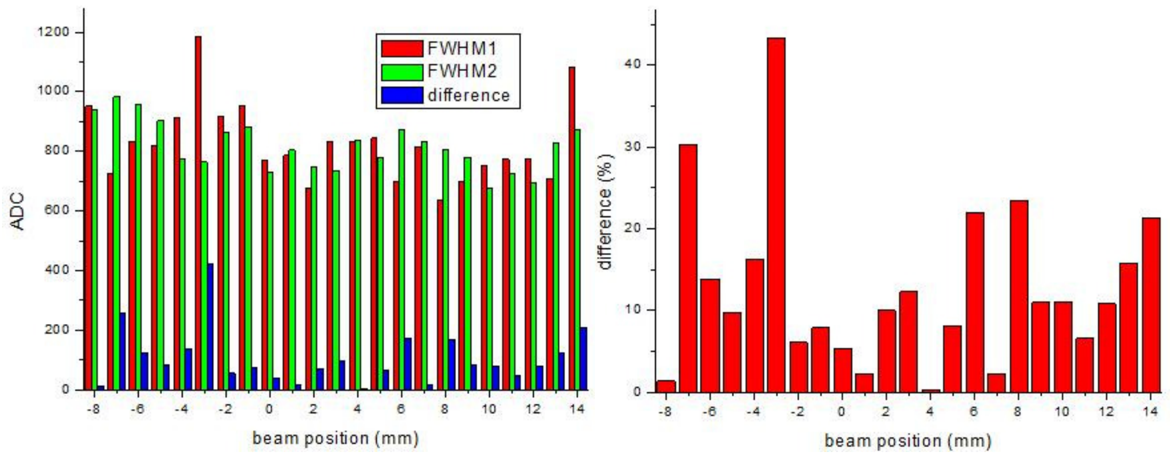


Figure 90: Left: FWHM values of the both measurements and the difference between these two values in function of the beam position in the *x* direction; Right: The difference between the FWHM values of both measurements as a percentage of the average value for every beam position in the *x* direction.

We expect to see the same pattern appear in the left part of Figure 90 as we saw in Figure 84. But the same pattern doesn't occur. The FWHM values of both scans however are in the *x* as well as in the *y* direction all located around 800. Most of the differences in the FWHM values in the *x* direction are smaller than 15 % with some exceptions until a maximum of around 45 %. These results are in accordance to those found along the *y* direction.

The peak width values of the scans in the *x* direction are shown in Figure 91.

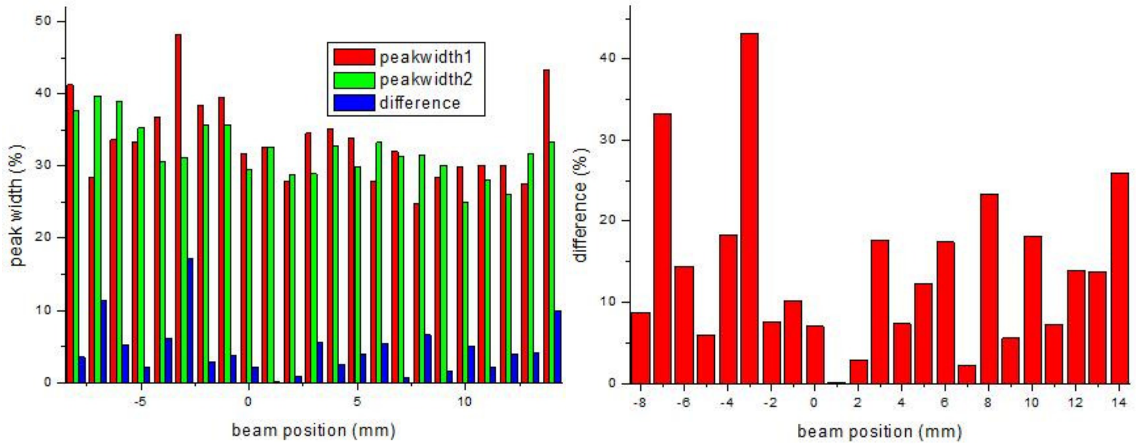


Figure 91: Left: Peak width values of both measurements and the difference between these two values in function of the beam position in the *x* direction; Right: The difference between the peak width values of both measurements as a percentage of the average peak width for every beam position in the *x* direction.

We see comparable results as we saw in the *y* direction, again the peak width values are located around 30 %. The differences between the both scans are in this case somewhat higher than in the *y* direction, most differences are now located below 15 % instead of the 10 % we saw in the *y* direction.

Figure 92 shows the positions of the sum peaks for the *x* direction.

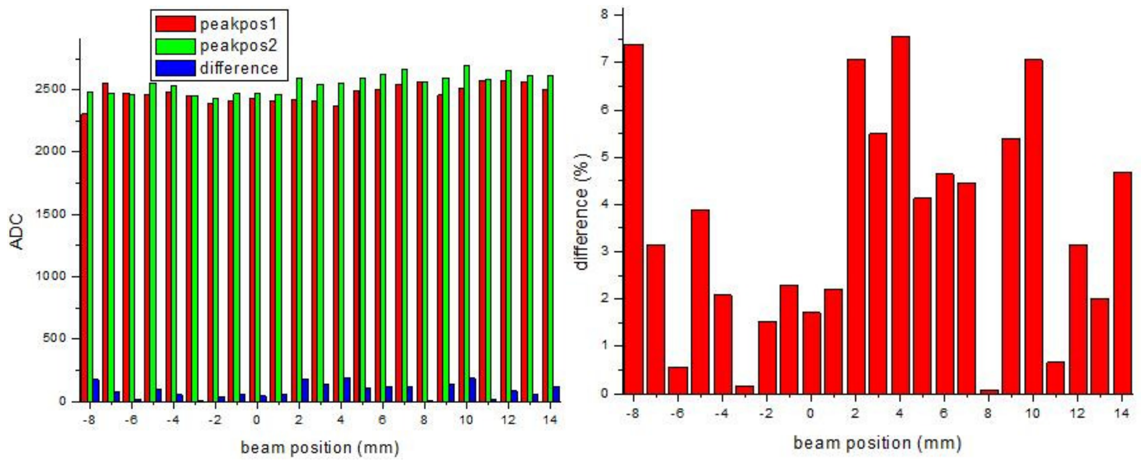


Figure 92: Left: Peak positions of the both measurements and the difference between these two values in function of the beam position in the *x* direction; Right: The difference between the peak positions of both measurements as a percentage of the average value for every beam position in the *x* direction.

We see the stability of the peak position over the entire surface of the detector block like we saw in the *y* direction as well. In both directions the peak positions are located around 2500. The difference between the peak positions in the *x* direction of both measurements is larger than we noticed for the *y* direction. Most differences are smaller than 5 %, but with exceptions until a maximum of around 7,5 %, where the values in the *y* direction where all located below 3 %.

When we study the differences between the ADC output values in the *x* direction, the values of the rows are representative. Figure 93 shows those results.

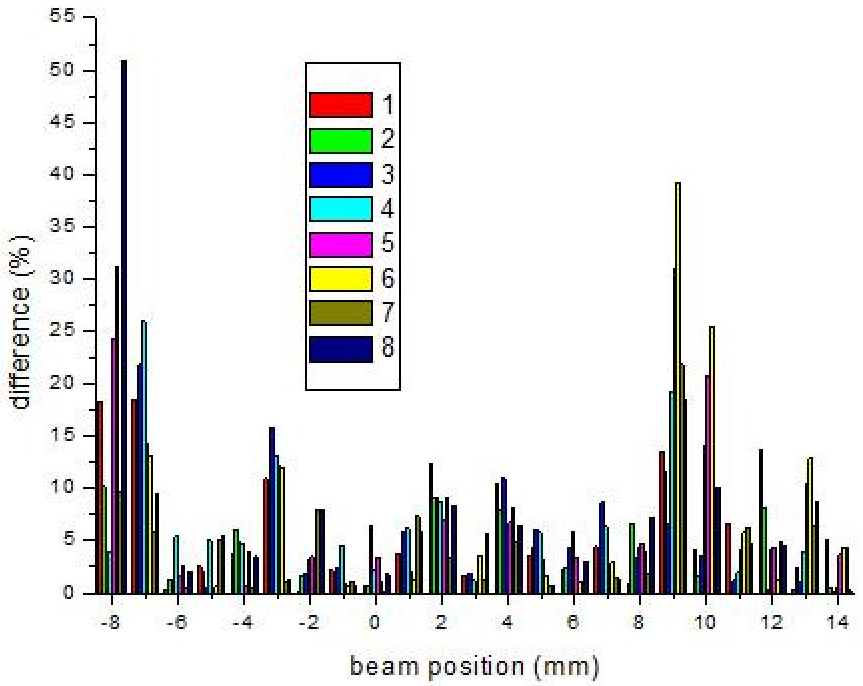


Figure 93: Difference in percentages between the mean values of the different 8 rows for every beam position in the *x* direction.

We notice that the differences in percentages in the *x* direction are in general bigger than the percentages in the *y* direction. In general the percentages of the differences of the mean values of the rows in the *x* direction are less than 10 % with some exceptions up to a maximum of 52 %, where for the *y* direction most differences were located below 6 %. We notice that only on the left edge of the detector block is where the bigger differences are located, this could be due to a slight displacement of the detector block in one of the two measurements. For most of the beam positions we noticed that row 6 gives us quite a big difference compared with the other rows. This can be explained by the fact that row 6 provides us with outputs that are clearly smaller in amplitude than the other rows, because of this, variations will result in greater differences when we look at them in percentages. Other local major differences can again be due to local variations in the ADC output values.

An explanation to those local variations could probably be the placement of the APD arrays relative to the beam. This placement is shown in Figure 94.

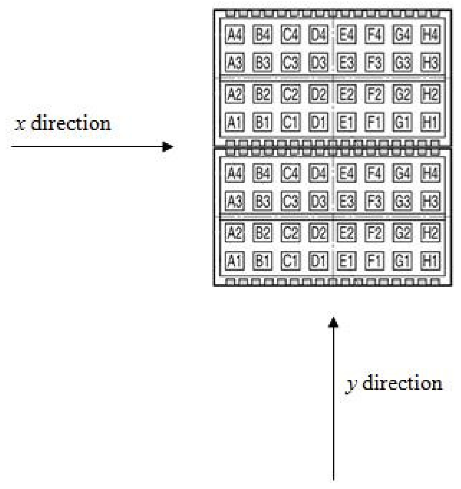


Figure 94: Placement of the APD arrays relative to the beam, based on [HA85].

We see that the distances between the 8 APDs in the *y* direction is constant. Between the centers of every APD pixel the distance is 2,3 mm, the physical dimensions of the APD are shown in Figure 18. In the *x* direction not all the distances between the APDs are equal. In one APD array the distance between A1 and A2 is 2,3 mm equal to the distance between A3 en A4. The distance between A2 and A3 is 2,6 mm and the distance between A1 from APD array 1 and A4 from APD array 2 is equal to 4 mm. Those differences in space between the APDs could be an explanation of local variations in the mean values.

The FWHM values of the peaks in the histograms of the relative errors of the estimations of the NN are shown in Figure 95.

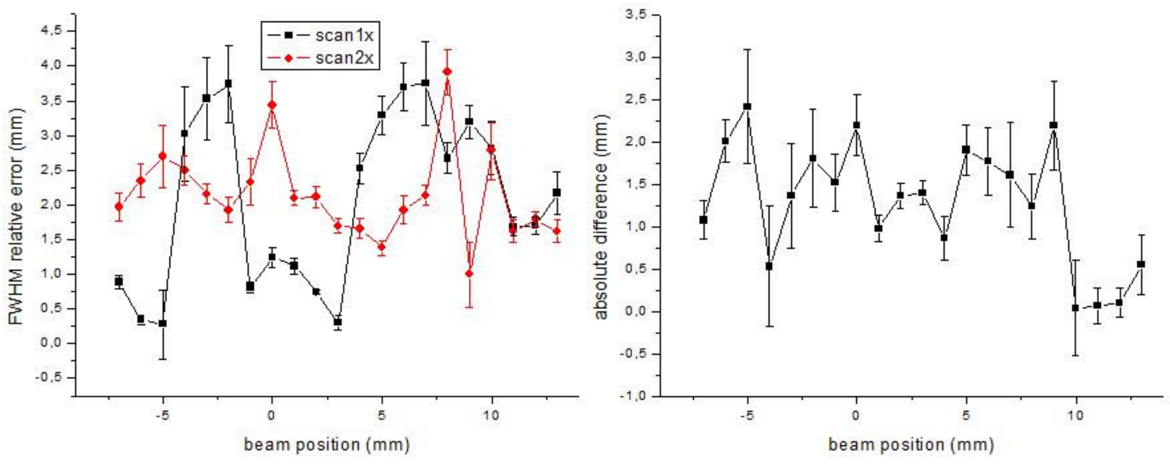


Figure 95: Left: FWHM of the relative errors of the NN estimations of both scans in function of the beam position for the scan in the *x* direction; Right: The difference between both estimations in absolute values in function of the beam positions.

For the scans performed in the *y* direction we noticed that the pattern of the FWHM values of both scans was similar. For the scans in the *x* direction this is clearly not the case. The absolute differences are mostly smaller than 1,5 mm, the maximum difference is 2,4 mm. The FWHM value of the entire relative error histogram in the *x* direction has a FWHM value of 3,3 mm for the first scan and 3,1 mm for the second scan. So like the *y* direction, the *x* direction is quite stable.

Figure 96 shows the peak positions of the NN errors for both scans and the difference between those values.

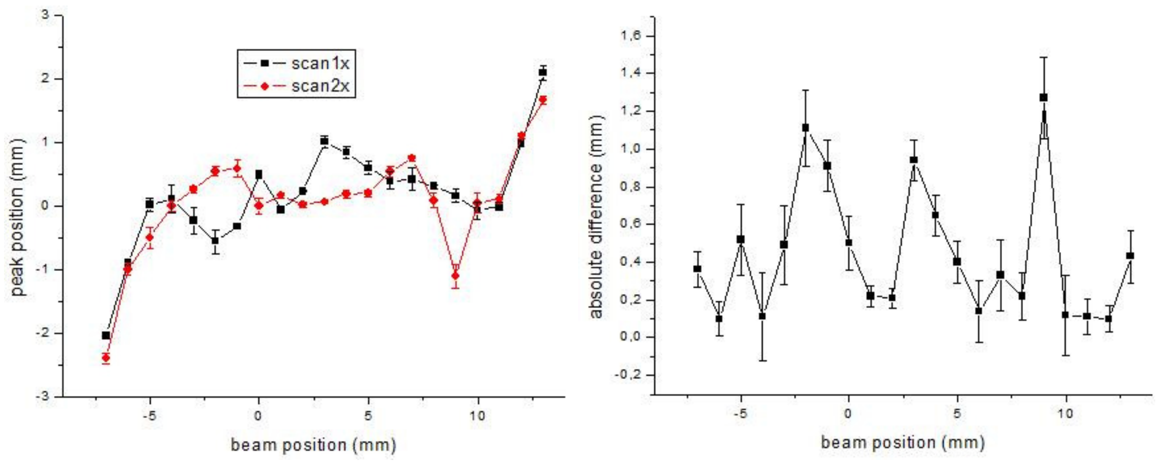


Figure 96: Left: peak positions of the NN errors of both scans in function of the beam position for the scan in the *x* direction; Right: the difference between both peak positions in absolute values in function of the beam positions.

The same pattern shows up when we look to the peak positions in function of the different beam positions for both scans. The maximum difference between the peak positions of both scans is 1,3 mm and most values are smaller than 0,6 mm.

Appendix

A.1 Used equipment, debugging setup

- DC Power Supply

The DC power supply we used was the E3631A 80-watt triple-output from AGILENT. This power supply can deliver three different dc outputs:

  - 0 to 6 V with a maximum current of 5 A;
  - 0 to 25 V with a maximum current of 1 A;
  - 0 to -25 V with a maximum current of 1 A;

We used the 0 to 25 and 0 to -25 V outputs to be able to give the electronically circuits a power supply of 7 V and 1 A. Figure 97 shows a power supply of the same type as we used in our setup.

Figure 97: E3631A DC power supply [TS].

- High Voltage Supply

The BERTAN 215 is the high voltage power supply we used in our application. We used the power supply to add 427 V to our APDs to get the desired gain. Figure 98 is a picture taken of the high voltage power supply we used in our setup.

Figure 98: BERTRAN 215 high voltage supply.

o Detector Box

The detector box is made of aluminum, the thickness of the box is 10 mm. Above the position of the monolithic crystal there is a thinner widow. This window has a thickness of 3 mm. The goal of this thinner window is to let the radiation pass easier, so we could use a radioactive source which was not too strong. This also allowed us to limit the duration of our experiments. Figure 99 is a picture taken from the lid of the detector box, it shows clearly the thinner window and its position in the lid.

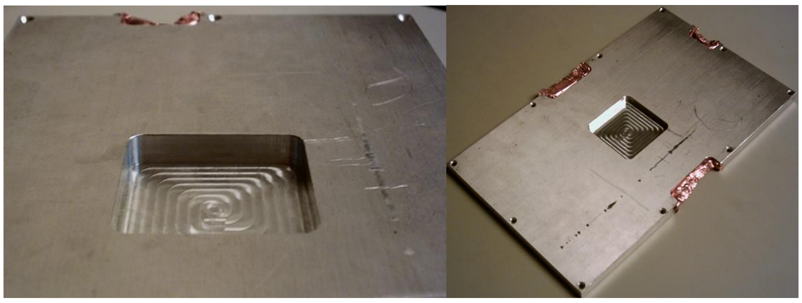


Figure 99: Lid of the detector box.

Figure 100 is a picture taken from the detector box while the lid is removed. We can see that the detector block itself was not present in the box at the time the picture was taken.

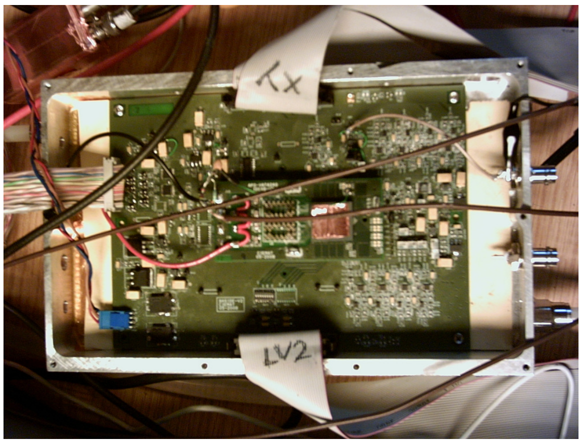


Figure 100: Detector box without the lid.

o Inverter

To change the positive trigger-pulse into a negative trigger-pulse we used the module: ORTEC 533 Dual Sum and Invert Amplifier. From this module we only used the input A1 and the output A connections with a gain of 1. While connecting the input from the detector box with the A1 input we used a T-connector of the BNC-type, at the other side of the T-connector we placed a 50 Ω impedance, this to avoid oscillations. Figure 101 shows on the left side an inverter of the same type we used in our setup.

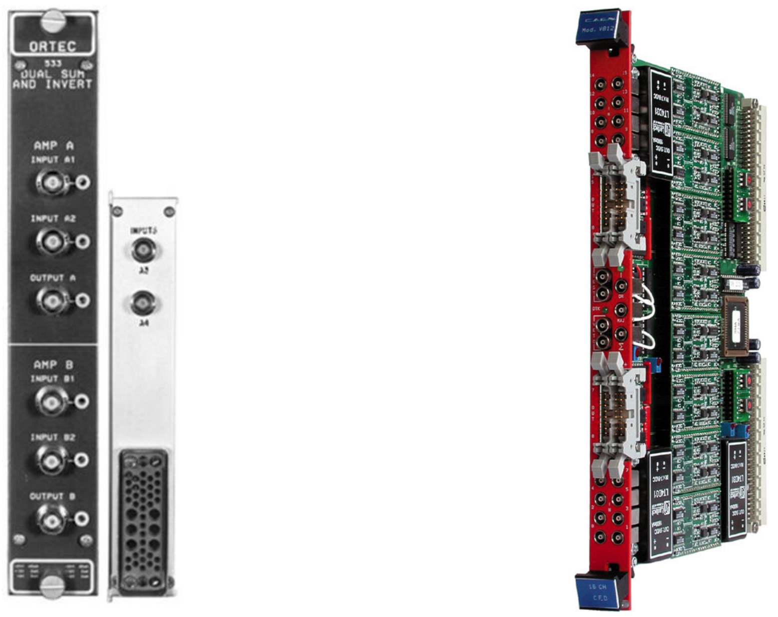


Figure 101: Left: ORTEC 533 Dual Sum and Invert Amplifier [0R]; Right: CAEN V812 [CA812].

o Discriminator

The discriminator we used to transform the negative TTL-trigger into a pulse of the NIM-type is the CAEN V812 module. From this module we only used one of the 16 available inputs and the OR-output. Because there is only one input active the output of the OR gate will be exactly the same as the input signal but now in the form of a NIM signal instead of a TTL signal. At the back of the module the module is connected with the VME-bus connecting all the VME-modules. In Figure 101 on the right side, we see a V812 module similar as the one we used in our setup.

o Optical Link

To create the gate we used the CAEN V2718 module. From this module we are using one input and one output connection. Further this module is connected with the second computer equipped with a PCI controller card, in this way we can modify the parameters of the window like the duration and the delay. The connection between the computer and the V2718 module takes place through an optical fiber cable. The VME bus activity can be monitored through the LED displays on the module. Figure 102 shows on the left side a V2718 module similar as the one we used in our setup.

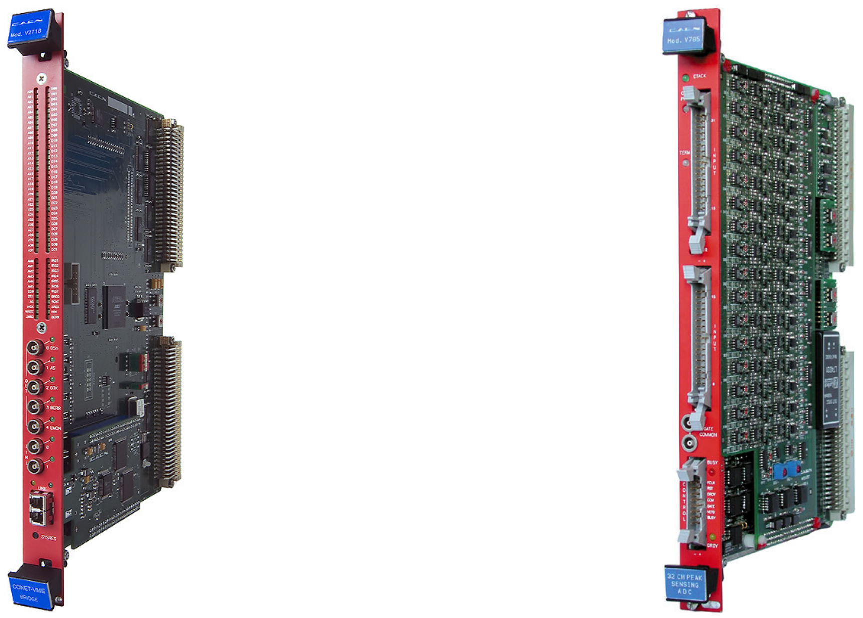


Figure 102: Left: CAEN V2718 [CA2718]; Right: CAEN V785 [CA785].

- Peak Sensing ADC

The module that gives us eventually the maximum count within a certain time window is the CAEN V785 module. This module is like the other modules connected with the VME-bus, in this way it is able to give its output back to the V2718 which can pass it to the computer. This module receives 2 kinds of input signals: on one hand it receives the data from the ASICs and on the other hand it receives a time window to tell the module when it has to search for a maximum in the received data. In Figure 102 is a module of this type shown on the right side.

- Connections

To make the connections between the different devices we used two different kinds of connections:

  - The first kind of connectors are the so called BNC connectors (Bayonet Neill Concelman). These connectors have an impedance of 50 Ω. Figure 103 shows some BNC-connectors.

Figure 103: BNC connectors [NU].

- The second kind of connectors we used are the LEMO connectors, named after the company fabricating these cables. This type of connectors is a standard when using NIM-type signals. The cables we used had a time delay of 15 or 10 ns. The following figure shows a LEMO-connector.

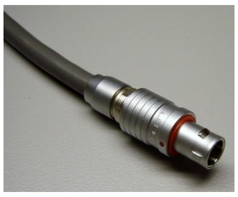
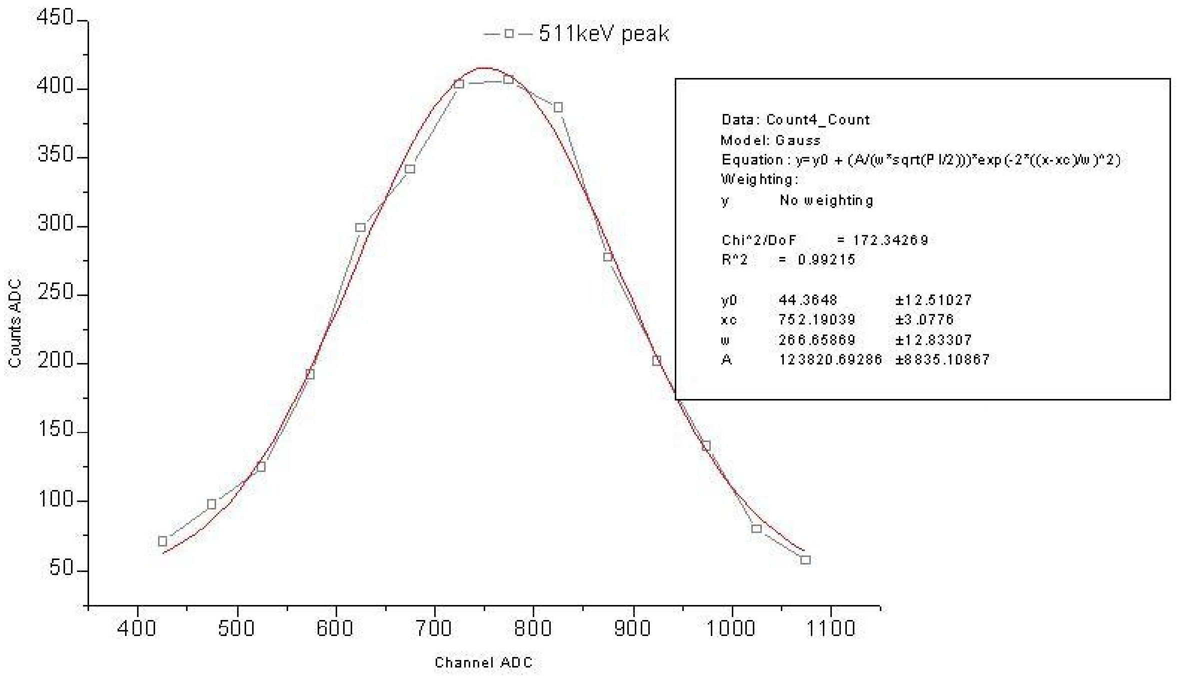


Figure 104: LEMO connector [Pa].

A.2 Spectrums for a high voltage of 427 V

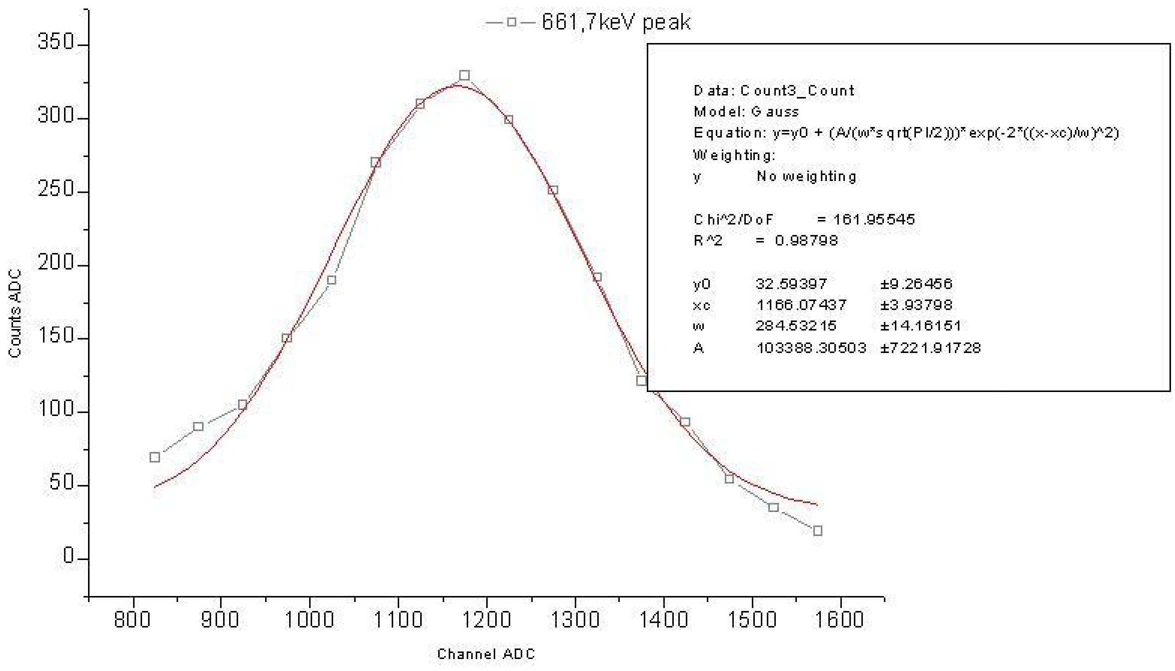


```
Data: Count4_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y         No weighting

Chi^2/DoF      = 172.34269
R^2     =  0.99215

y0        44.3648          ±12.51027
xc        752.19039        ±3.0776
w         266.65869        ±12.83307
A         123820.69286     ±8835.10867
```

Figure 105: Gaussian fit of the 511 keV peak of ^{22}Na for a high voltage of 427 V.

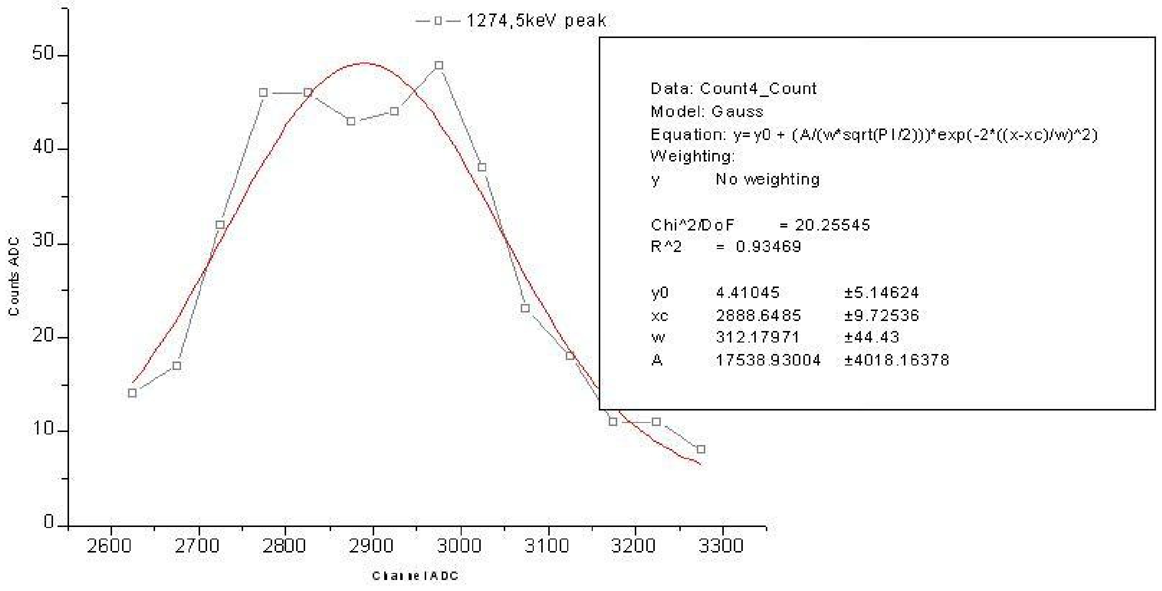


```
Data: Count3_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y          No weighting

Chi^2/DoF        = 161.95545
R^2      =  0.98798

y0        32.59397          ±9.26456
xc        1166.07437        ±3.93798
w         284.53215         ±14.16151
A         103388.30503      ±7221.91728
```

Figure 106: Gaussian fit of the 611,7 keV peak of ^{137}Cs for a high voltage of 427 V.



```
Data: Count4_Count
Model: Gauss
Equation: y=y0 + (A/(w*sqrt(PI/2)))*exp(-2*((x-xc)/w)^2)
Weighting:
y          No weighting

Chi^2/DoF        = 20.25545
R^2      =  0.93469

y0        4.41045           ±5.14624
xc        2888.6485         ±9.72536
w         312.17971         ±44.43
A         17538.93004       ±4018.16378
```

Figure 107: Gaussian fit of the 1274,5 keV peak of ^{22}Na for a high voltage of 427 V.

A.3 Measured plateau and sigmoid fits in the *x* direction

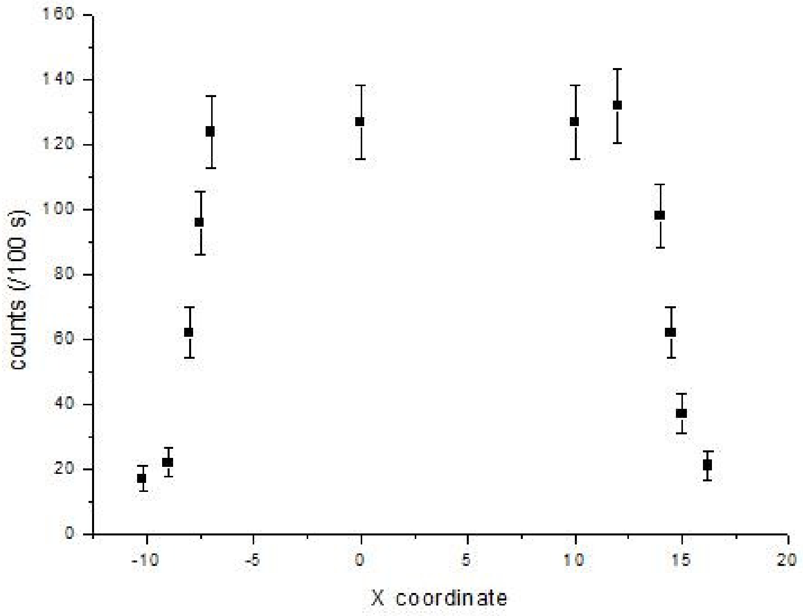
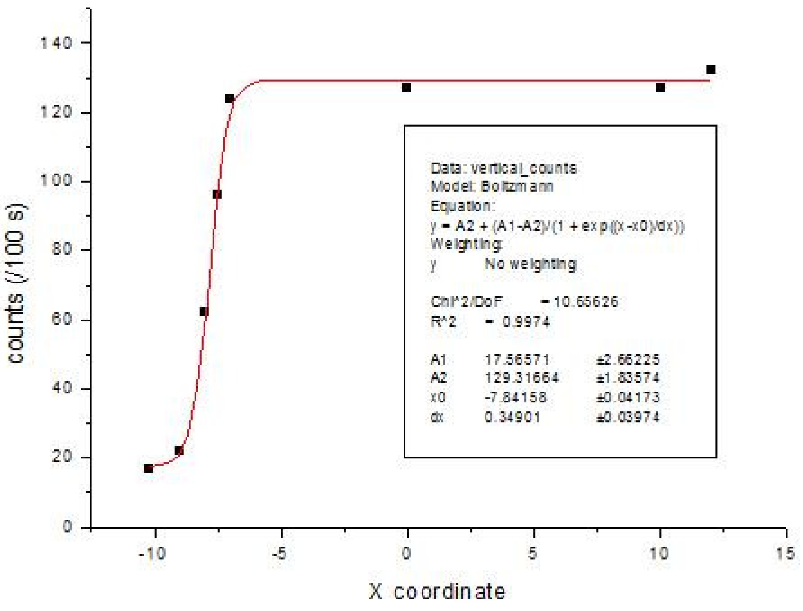


Figure 108: Measured plateau in the *x* direction.

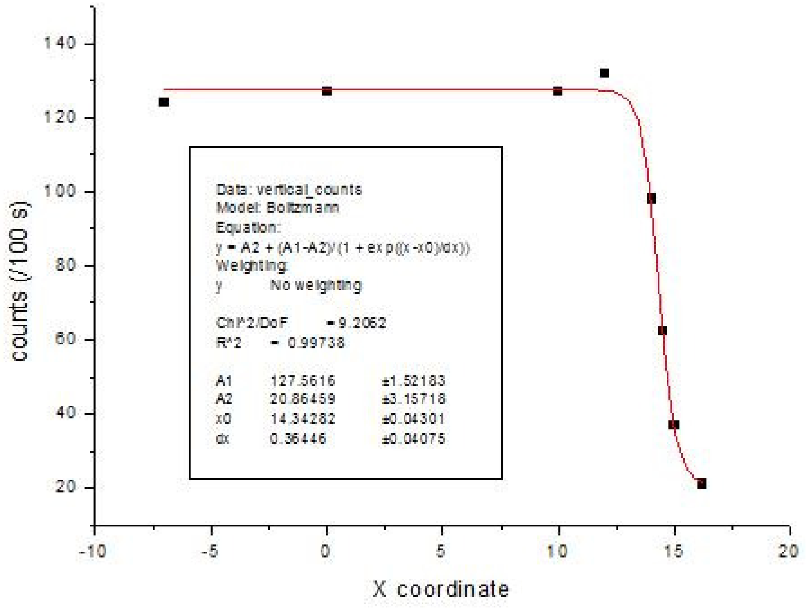


```
Data: vertical_counts
Model: Boltzmann
Equation:
y = A2 + (A1-A2)/(1 + exp((x-x0)/dx))
Weighting:
y        No weighting

Chi^2/DoF      = 10.65626
R^2     = 0.9974

A1      17.56571       ±2.66225
A2      129.31664      ±1.83574
x0      -7.84158       ±0.04173
dx      0.34901        ±0.03974
```

Figure 109: Sigmoid fit of the transition on the left side of the plateau of the measurement performed in the *x* direction.



```
Data: vertical_counts
Model: Boltzmann
Equation:
y = A2 + (A1-A2)/(1 + exp((x-x0)/dx))
Weighting:
y          No weighting

Chi^2/DoF       = 9.2062
R^2       = 0.99738

A1        127.5616        ±1.52183
A2        20.86459        ±3.15718
x0        14.34282        ±0.04301
dx        0.36446         ±0.04075
```

Figure 110: Sigmoid fit of the transition on the right side of the plateau of the measurement performed in the *x* direction.

A.4 The Levenberg-Marquardt algorithm

The function is transformed to the following format:

$$f: \quad R^m \rightarrow R^n;$$

$$f_l\left(\vec{p}\right) = x_l \text{ , } l = 0, \ldots, n-1\text{; } m = 81.$$

We consider that we have a function that provides us n outputs while it needs m inputs. The vector parameter array $\vec{p} \equiv (p_0 \ p_1 \ \ldots \ p_{80})^T$ corresponds in a compact notation to the parameter set $(b_0 \ldots b_4, a_0 \ldots a_4, u_{00} \ldots u_{44}, w_{00} \ldots w_{47}, v_0 \ldots v_4, c)$. The number of training events n must be sufficiently large to be able to have a proper convergence efficiency.

The Levenberg-Marquardt algorithm requires the following input data:

- The dimensions of the data set: n;
- The dimensions of the parameter set: $m = 81$;
- The initial estimation of $\vec{p} \equiv (p_0 \ p_1 \ \ldots \ p_{80})^T$;
- The measurement vector $\vec{x} \equiv (x_0 \ \ldots \ x_l \ \ldots \ x_{n-1})^T$;

In the case of using two hidden layers of each five neurons, the first five q-parameters of the set of eighty-one parameters, are the constants for the neurons of the second hidden layer, the next five parameters are the constants of the neurons of the first hidden layer. The next twenty-five parameters are the ones making the connections between the first and the second hidden layer, the next forty parameters are the ones making the connections between the input layer and the first hidden layer of neurons. The next five parameters are the ones connecting the second hidden layer of neurons with the output and the last parameter is the constant of the output neuron.

A.5 Code A

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
//Here we include the different headerfiles needed in the code, these
header files include the different libraries that are used. In the code
we tried to use in most cases pointers, which point to the place in
memory of the concerned value, becaues this works faster than using the
real values.

float p[80];
int m;
//Implement an array p with a size of eighty-one places for the
parameters and an integer m that will be needed in the following code as
a counter for the parameters.

void load_coeff_p(){
//We made a class with the name load_coef_p which we can call from the
main class. In this class the eighty-one coefficients are loaded for the
neural network. We implement a File type object pointer fp to be able to
control the stream of data from an inputfile. We make a char_data array
with a size of two hundred places, this should be sufficient enough for
our inputfile. The inputfile is scanned for data as long as the integer
eof is different from zero. If the end of the file is reached, this
integer is set to zero and the scanning stops. Each number extracted from
```

```
the data is coppied to the float_data and then placed in the array p. A
counter m_i makes sure that the number is places in the correct place in
the array. At the end of this class the value of the integer m is set to
the total number of coefficients for the next steps in the code.

  FILE    *fp;
  float   float_data;
  char    char_data[200];
  int     eof=1;
  int     m_i=0;

  if ((fp=fopen("coeff_p_y_0.dataset","r"))==NULL){
    printf("Error opening coeff p Y file \n");
  }
  else{
    eof=fscanf(fp," %s",char_data);
    while (eof==1){
      fscanf(fp," %f ",&float_data);
      if(eof==1){
      p[m_i]=float_data;
      m_i++;
      eof=fscanf(fp," %s",char_data);
      }
    }
    fclose(fp);
  }
  m=m_i;
}

inline float sigmoid(float *x){
//In this class the calculation of the sigmoid is being executed. We use
an inline function to suggest to the compiler that the code generated by
the function body is inserted at each point in the code where the
function is called, instead of being inserted only once and perform a
regular call to it, which generally involves some additional overhead in
process time.

  return 1.0/(1.0+exp(-(*x)));
}

float nn_BP_float(float *d_l){
```

```
//In this class the calculations of the neural networks itself are being
executed. The float alpha_j holds the constants of the 5 neurons of the
first hidden layer, float dzeta_k holds the constants of the 5 neurons in
the second hidden layer and the float sum_k holds the result. The
inputvalue of the class is a pointer of an array of 8 values named d_l.
The integers i,j and k are respectively the indexes of the inputlayer,
the first hidden layer of neurons and the second hidden layer. On the
other hand the integers ji, kj and jj are holding respectively the
weights between the inputlayer and the first hidden layer, the weights
between the first and the second hidden layers and the weights between
the second hidden layer and the output. At the end we sum the result with
the last value of the array p, this is the constant of the output neuron.

  int i,j,k;
  int ji,kj,jj,k_M;
  float alpha_j,dzeta_k,sum_k;
  sum_k=0.0;
  for(k=0; k<5; k++){
    dzeta_k=0.0;
    for(j=0; j<5; j++){
      alpha_j=0;
      for(i=0; i<8; i++){
      ji=35+8*j+i;
      alpha_j+=p[ji]*d_l[i];
      }
      kj=10+5*k+j;
      jj=j+5;
      alpha_j+=p[jj];
      dzeta_k+=p[kj]*sigmoid(&alpha_j);
    }
    dzeta_k+=p[k];
    k_M=75+k;
    sum_k+=p[k_M]*sigmoid(&dzeta_k);
  }
  return sum_k+p[m-1];
}
int main(){
//From the inputfile, trset_y_deg_0_0.dat, the data is readed in and
stored in datasets of eight values. For this purpose the two dimensional
float array is inplemented. Then the input data is normalised, this
happens in this case by deviding every of the eight values by the maximum
```

```
of the eight values. By using the parallel command we can use the two
cores of the computer at the same time. Depending on which compiler you
use both processors or one processor will now calculate the number of
NN's you want. In the present form of the code two million NNs are being
calculated.

  int i,l;
  int ret;
  float p[81];
  float x[36000];
  float data[36000][8];
  float d_l[8];
  int m, n;
  FILE   *fp;
  float  float_data;
  int    out=0;
  int    k,j,ji,kj,jj,k_M;
  float sum_k,dzeta_k,alpha_j;
  int    tst_q;
  float  tst_res;
  float nn_res;
  float int_nn_res;
  int    int_data;
  int    error=0;
  int    i_tr,n_tr,tot_tr;
  char   char_data[200];
  int    eof=1;
  float  dat_xy[9];
  int    sum_max, data_max;
  int ll;
  float res;
  m=81;
  load_coeff_p();
  n=0;
  n_tr=20;
  i_tr=0;
  tot_tr=20;
  sum_max=0;
  data_max=0;

  if ((fp=fopen("trset_y_deg_0_0.dat","r"))==NULL){
    printf("Error opening train set  file \n");
  }
  else{
    printf("Reading train data set file \n");
    fscanf(fp," %s %s %s %s",char_data,char_data,char_data,char_data);
    while (eof==1){
      eof=fscanf(fp," %f ",&float_data);
      if(eof==1){
      x[n]=float_data;
      fscanf(fp," %f ",&float_data);
      for (i=0;i<8;i++){
        fscanf(fp," %d ",&int_data);
        dat_xy[i]=(float)int_data;
        if ( dat_xy[i]> data_max) data_max=dat_xy[8];
      }
      fscanf(fp," %d ",&int_data);
      dat_xy[8]=(float)int_data;
      if(i_tr<n_tr){
        for (i=0;i<8;i++){
          data[n][i]=dat_xy[i]/dat_xy[8];
        }
        n++;
      }
      i_tr++;
      if(i_tr==tot_tr){
        i_tr=0;
      }
      }
    }
    fclose(fp);
  }

  printf("num de datos: %d. >Data_max. %d  Sum_max: %d
\n",n,data_max,sum_max);

  tst_q=0;

  #pragma omp parallel for private(ll,l)
  for (ll=0;ll<200;ll++){
    for (l=tst_q;l<(tst_q+10000);l++){
      for (i=0;i<8;i++){
```

```
      d_l[i]=data[l][i];
      }
      res= nn_BP_float(d_l);
      printf("%.3f  \n",res);
    }
  }
  return 0;
}
```

A.6 Code B

```
#pragma omp parallel for private(ll,l)
  for (ll=0;ll<200;ll++){
    for (l=tst_q;l<(tst_q+10000);l++){
      for (i=0;i<8;i++){
      d_l[i]=data[l][i];
      }
      res= nn_BP_float(d_l);
      //printf("%.3f  \n",res);
    }
  }
  printf("%.3f  \n",res);
  return 0;
```

A.7 Code C

```
inline float sigmoid(float *x)
{
      if(*x < -4.6){
            return 0;
      }
      else if(*x > 4.6){
            return 1;
      }
      else{
            return (1.0/(1.0+exp(-(*x))));
      }
}
```

A.8 Code D

```
inline float sigmoid_fast(float *x){
      if(*x < -10){
            return 0;
      }
      else if(*x > 10){
            return 1;
      }
      else{
            return (1.0/(1.0+exp(-(*x))));
      }
}

inline float sigmoid(float *x){
    return (1.0/(1.0+exp(-(*x))));
}

float nn_BP_float(float *d_l){
  int i,j,k;
  int ji,kj,jj,k_M;
  float alpha_j,dzeta_k,sum_k;
  sum_k=0.0;
  for(k=0; k<5; k++){
    dzeta_k=0.0;
    for(j=0; j<5; j++){
      alpha_j=0;
      for(i=0; i<8; i++){
            ji=35+8*j+i;
            alpha_j+=p[ji]*d_l[i];
      }
      kj=10+5*k+j;
      jj=j+5;
      alpha_j+=p[jj];
      dzeta_k+=p[kj]*sigmoid(&alpha_j);
    }
    dzeta_k+=p[k];
    k_M=75+k;
    sum_k+=p[k_M]*sigmoid(&dzeta_k);
  }
  return sum_k+p[m-1];
}

float nn_BP_float_fast(float *d_l){
  int i,j,k;
  int ji,kj,jj,k_M;
  float alpha_j,dzeta_k,sum_k;
  sum_k=0.0;
  for(k=0; k<5; k++){
    dzeta_k=0.0;
    for(j=0; j<5; j++){
      alpha_j=0;
      for(i=0; i<8; i++){
            ji=35+8*j+i;
            alpha_j+=p[ji]*d_l[i];
      }
      kj=10+5*k+j;
      jj=j+5;
      alpha_j+=p[jj];
      dzeta_k+=p[kj]*sigmoid_fast(&alpha_j);
    }
    dzeta_k+=p[k];
    k_M=75+k;
    sum_k+=p[k_M]*sigmoid_fast(&dzeta_k);
  }
  return sum_k+p[m-1];
}

int hist[1001];
float offset=2.0;
int ch;
float arrerror[2000001];
int counter=0;
```

```
#pragma omp parallel for private(ll,l)
  for(ll=0;ll<200;ll++){
    for (l=tst_q;l<(tst_q+10000);l++){
      res= nn_BP_float(data[l]);
      res2= nn_BP_float_fast(data[l]);
      relerror = (res-res2)/res*100;
      arrerror[counter]=relerror;
      counter++;
    }
  }
  int w=0;
  for(w;w<2000000;w++){
    if(arrerror[w]>= -2 && arrerror[w]<2){
      int ch=(arrerror[w]+offset)*250;
      hist[ch]=(hist[ch])+1;
    }
  }
  int b = 0;
  for(b;b<1001;b++){
    printf("%d \n", hist[b]);
  }
  return 0;
```

A.9 Neural networks with CUDA

A.9.1 The graphical processor unit as a data-parallel computing device

Within a time period of just a few years the programmable GPU's have become known as an absolute computing workhorse, as illustrated in Figure 111.

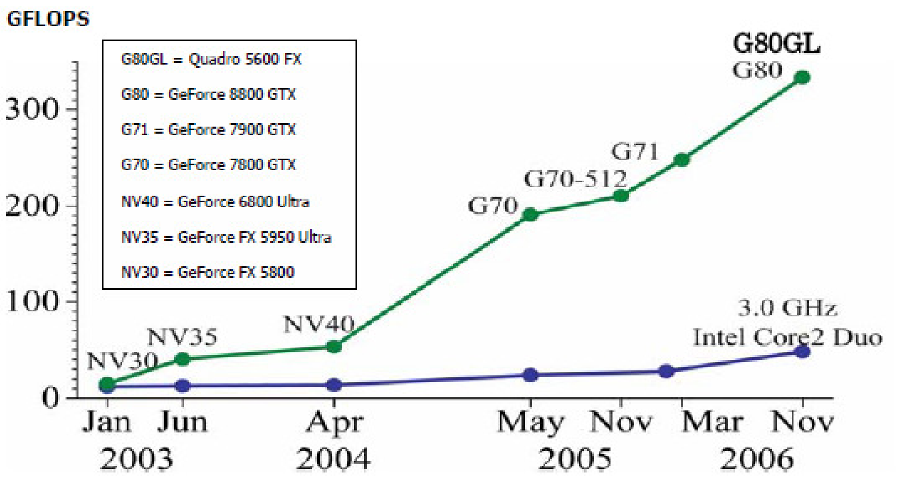


Figure 111: Graphical floating point operations per second for the CPU and GPU [NV07, 1].

In Figure 111 we see the GFLOPS performances, what stand for Graphical Floating point Operations Per Second, for some GPUs and CPUs in function of the time. It is clear that the performances of the GPUs are better than the performances of the CPUs and that the difference is increasing with the time.

The main reason behind this evolution is the design of the GPUs, they are specially designed for compute-insensitive, highly parallel computations. For this reason there are more transistors devoted to data processing rather than to data catching and flow control. This difference between the CPUs and the GPUs is schematically illustrated in Figure 112.

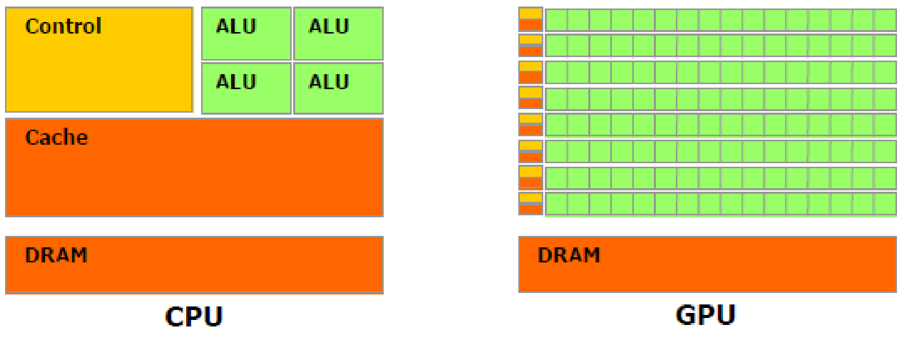


Figure 112: Schematically illustration of the difference between the CPUs and GPUs [NV07, 2].

We can see that the flow control and the cache blocks are less important in the GPU than in the CPU structure. The cache is a smaller, faster memory which is used to store copies of data from the most frequently used main memory locations. When a processor needs to read data from or write data to the main memory, it first checks if there is a copy of the data available in the cache. If this is the case, the processor will read from or write to the cache because this is much faster than writing to or reading from the main memory. The DRAM block is in the CPU and GPU of the same importance, it stands for the Dynamic Random Access Memory. This DRAM stores each bit of the data in a separate capacitor within an integrated circuit. Because real capacitors leak charge, the information eventually fades unless the capacitor charge is refreshed periodically. Because of this refresh requirement, it is a dynamic memory.

On the other hand the ALU block, which stands for Arithmetic Logic Unit, is becoming more important. This ALU is a fundamental building block of the CPU or GPU, it performs arithmetic and logical operations.

The explanations of these differences can be found in the fact that GPU is especially designed to address problems that can be expressed as data-parallel computations. In this case the program is executed on many data elements in parallel; these programs have a high ratio of arithmetic operations to memory operations. Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control; and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be done by calculations instead of big data caches.

Data-parallel processing maps data elements to parallel processing threads. Many applications that process large data sets such as arrays can use a data-parallel programming model to speed up the computation. So our problem of the calculation of the neural networks is extremely suited to be executed by this data-parallel programming.

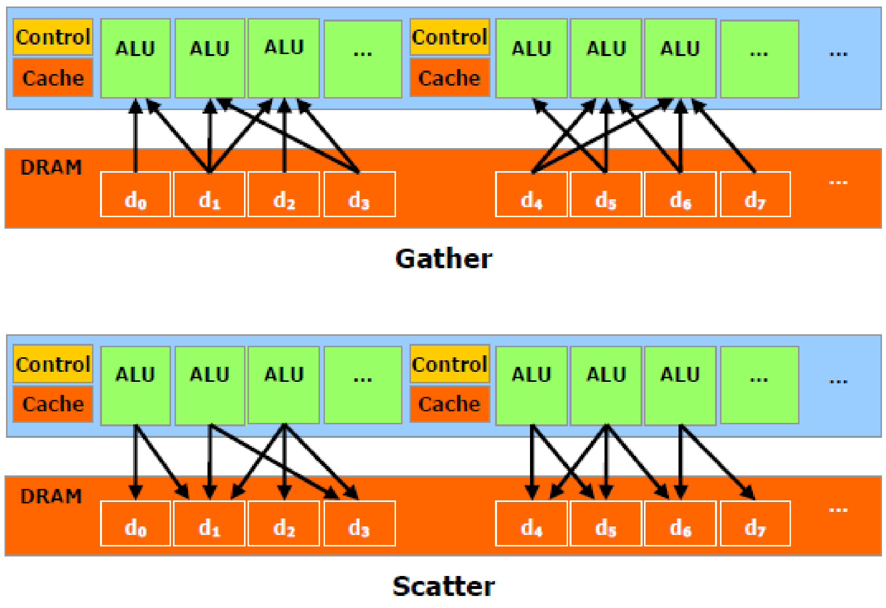


Figure 113: The gather and scatter memory operations [NV07, 4].

From a programming perspective, this translates into the ability to read and write data at any location in DRAM, just like on a CPU. Further CUDA features a parallel data cache or on-chip shared memory with very fast general read and write access. This memory is used by the threads to share data with each other. Applications can take advantage of this on-chip memory by minimizing over fetch and round-trips to DRAM and therefore becoming less dependent on DRAM memory bandwidth. Figure 114 shows the use of this on-chip memory.

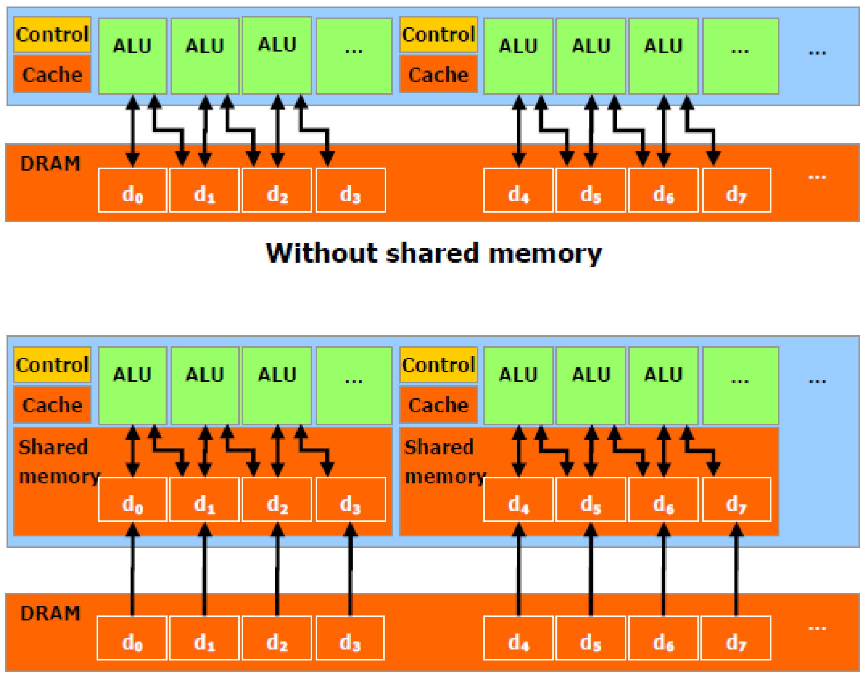


Figure 114: The use of on chip shared memory [NV07, 5].

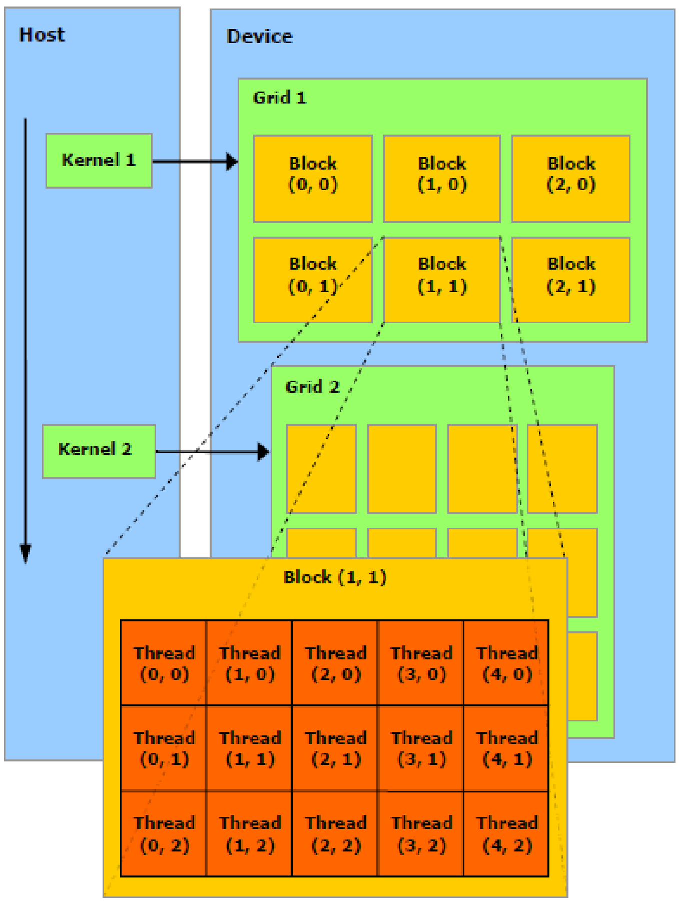
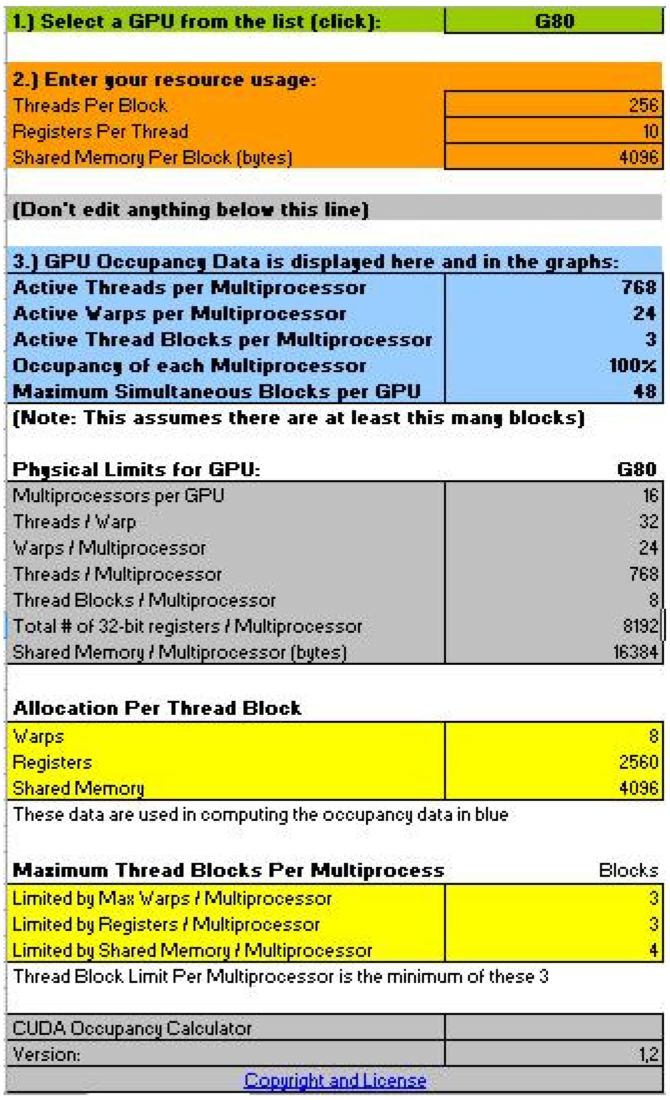


Figure 115: Structure of the kernel, thread batching [NV07, 9].

We can see that each kernel consist out of a grid, this grid on its turn consists out of a number of thread blocks. These thread blocks on their turn consist out of a number of threads.

The batch of threads forming a thread block can cooperate together by efficiently sharing data through some fast shared memory and synchronizing their execution to coordinate memory access. More precisely, it is possible to set some synchronization points in the kernel. The threads in the thread blocks are suspended until they all reach the synchronization point. Every thread has its own unique thread ID, which is the thread number within the block. These thread IDs are only unique within their own thread block. To help with complex addressing based on the thread ID, an application can also specify a block as a two- or three-dimensional array.
There is a maximum number of threads that a thread block can contain. However, thread blocks with the same dimensions executing the same kernel can be batched together into a grid of blocks. In this way the total number of threads dedicated to one kernel can become



1.) Select a GPU from the list (click):	G80
2.) Enter your resource usage:	
Threads Per Block	256
Registers Per Thread	10
Shared Memory Per Block (bytes)	4096
(Don't edit anything below this line)	
3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	768
Active Warps per Multiprocessor	24
Active Thread Blocks per Multiprocessor	3
Occupancy of each Multiprocessor	100%
Maximum Simultaneous Blocks per GPU	48
(Note: This assumes there are at least this many blocks)	
Physical Limits for GPU:	G80
Multiprocessors per GPU	16
Threads / Warp	32
Warps / Multiprocessor	24
Threads / Multiprocessor	768
Thread Blocks / Multiprocessor	8
Total # of 32-bit registers / Multiprocessor	8192
Shared Memory / Multiprocessor (bytes)	16384
Allocation Per Thread Block	
Warps	8
Registers	2560
Shared Memory	4096
These data are used in computing the occupancy data in blue	
Maximum Thread Blocks Per Multiprocess	Blocks
Limited by Max Warps / Multiprocessor	3
Limited by Registers / Multiprocessor	3
Limited by Shared Memory / Multiprocessor	4
Thread Block Limit Per Multiprocessor is the minimum of these 3	
CUDA Occupancy Calculator	
Version:	1,2
Copyright and License	

Figure 116: Spreadsheet from the CUDA Occupancy Calculator [NV1.2].

In this spreadsheet you can choose the graphical card you are using in your application. The next step is to make a decision about the number of threads available in each thread block. Then we fill in the registers that can be used by each thread and the shared memory available for each thread block. Now the excel-sheet will calculate the occupancy of your settings and give the output-data.

A.9.3 The GeForce 8600 GT graphical card

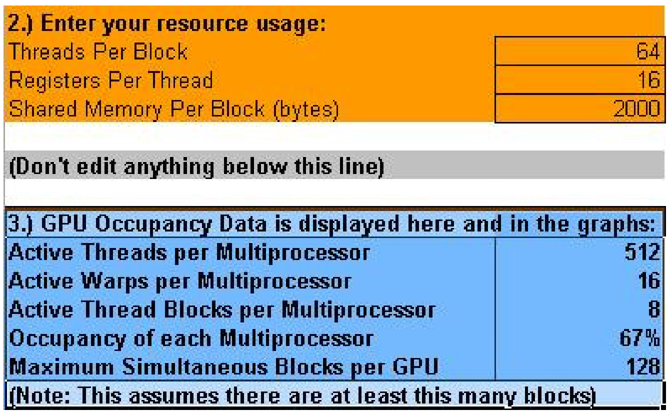
Our graphical card was the GeForce 8600 GT graphical card from NVIDIA, we choose a graphical card from NVIDIA because they are the most advanced company in using the graphical card as a parallel computing device. Out of the list of possible graphical cards from NVIDIA that were compatible with our computer we chosen the GeForce 8600 GT because this one is in everything an average of all the graphical cards available, its price is averaged and also the number of stream processors available is averaged. Figure 117 is a picture taken from our graphical card.

Figure 117: The used GeForce 8600 GT.

Specifications of the GeForce 8600 GT [NV07, 63-64]:

- Chipset: GeForce 8600 GT
- Graphic bus technology: PCI-Express 16 Lane;
- Core clock speed: 540 MHz;
- Shader clock speed: 1180 MHz;
- Memory clock speed: 700 MHz;
- Memory amount: 512 MB;
- Memory technology: DDR 2;
- Memory bus: 128 bit;
- Peak memory bandwidth: 16 GB/sec;
- Graphic engine: 256 bit;

- The total number of threads becomes 1024 which is smaller than the maximum number of threads that was set to 65535 threads per grid;
- For every multiprocessor the number of threads becomes 512, this is smaller than the maximum value of 768;
- The number of registers that are available for every thread can then be calculated by the following formula, explained in the previous chapter: $\frac{8192}{8 \cdot 64} = 16$, this should be enough to make sure that the kernel is able to compile;
- If we want to check our settings with the CUDA Occupancy Calculator, we also have to calculate the available shared memory per block. There are 16000 bytes available per multiprocessor so that gives a total of 32000 bytes. Divided over 16 thread blocks this gives 2000 bytes per thread block;
- When we give in our settings in the occupancy calculator we see that we achieve an occupancy of each multiprocessor of 67 percent. Figure 118 shows the output of the calculator.



2.) Enter your resource usage:	
Threads Per Block	64
Registers Per Thread	16
Shared Memory Per Block (bytes)	2000

(Don't edit anything below this line)

3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	512
Active Warps per Multiprocessor	16
Active Thread Blocks per Multiprocessor	8
Occupancy of each Multiprocessor	67%
Maximum Simultaneous Blocks per GPU	128

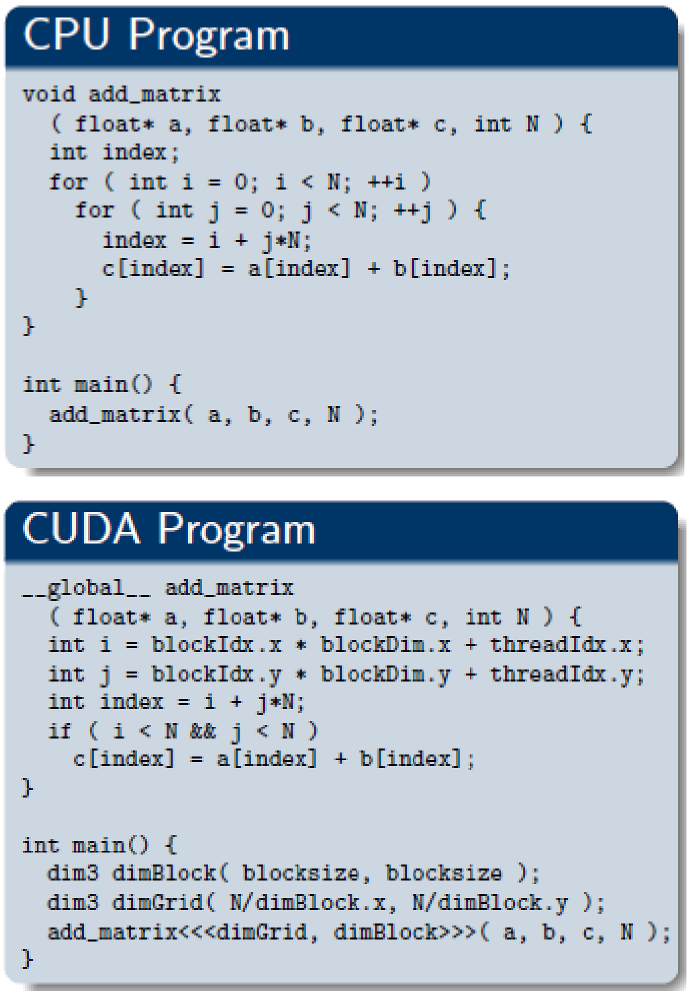
(Note: This assumes there are at least this many blocks)

Figure 118: Output of the occupancy calculator for our application.

A.9.4 CUDA-Code

Together with the CUDA software a nvcc compiler is downloadable, it is this compiler that converts your C code into an executable that will run on a GPU or an emulator.

Figure 119 gives a schematically impression of the difference between a simple code written in C++ and written in CUDA.



CPU Program

```
void add_matrix
  ( float* a, float* b, float* c, int N ) {
  int index;
  for ( int i = 0; i < N; ++i )
    for ( int j = 0; j < N; ++j ) {
      index = i + j*N;
      c[index] = a[index] + b[index];
    }
}

int main() {
  add_matrix( a, b, c, N );
}
```

CUDA Program

```
__global__ add_matrix
  ( float* a, float* b, float* c, int N ) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if ( i < N && j < N )
    c[index] = a[index] + b[index];
}

int main() {
  dim3 dimBlock( blocksize, blocksize );
  dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
  add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```

Figure 119: Difference between a code in C++ and CUDA [Se08, 7].

As you can see in Figure 119 the structure is quite similar, both languages are using a main function to call the function which contains the declaring of the calculation. In the CUDA program however this function is executed on the graphical card and in the CPU program it is executed on the CPU's of the computer. You can see the difference if you look at the way the function is called. The typical way of calling a function in the CUDA code is by using the "Func <<< Dg, Db, Ns >>> parameter" structure. Where Func stands for the name of the function you are calling, Dg stands for the dimensions of the grid you want to use, so the number of blocks to be more precise. Db is standing for the dimensions of the blocks you want to use, so the number of threads where the blocks are consisting off. Ns is of type size_t and it specifies the number of bytes in the shared memory that is dynamically allocated per block for this call in addition to the statically allocated memory, this is an optional argument which defaults to 0. Then there are the parameters needed for the function you want to execute. This call executes a function that is declared in the

A.12 Used equipment, spatial resolution

Most of the equipment used for this experimental setup is the same as used in the debugging setup. The next elements are added to this setup to complete the experimental setup to determine if there is a spatial correlation.

- PMT

The photomultiplier tube we used was the R7400U-01 of HAMAMATSU.

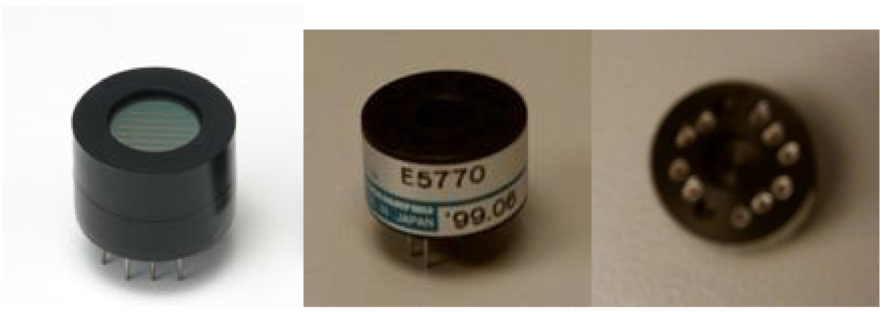


Figure 120: Left: PMT R7400U-01 [HA74]; Right: pictures taken from the active socket.

The PMT is mounted on one sight with optical glue to a NaI:Tl crystal and on the other side it is plugged into an active socket of the type E5770. It is an active socket because it divides the high voltage you apply on it into the voltages suited for the pins of the PMT. The PMT can only be connected with the socket in one correct way because of the placement of the holes in the socket.

Figure 121 gives a schematically impression of the PMT in combination with the detector crystal and the active socket.

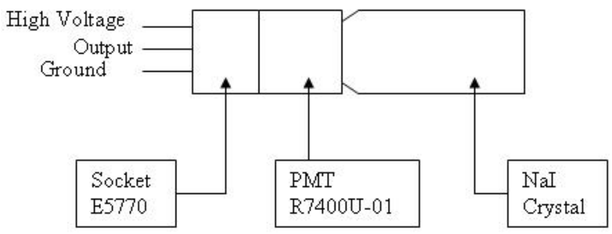


Figure 121: Left: Schematic diagram of the NaI:Tl crystal, PMT and socket; Right: picture of the box.

This combination is placed in a small box wrapped in copper paper for safety reasons, because of the high voltage we have to apply to the PMT and to isolate the PMT from noise. This box is shown in Figure 121, the picture shows clearly the connections for the high voltage and the output.

- Amplifier

The amplifier we used to amplify the signal produced by the PMT is the ORTEC 450, shown in Figure 122.

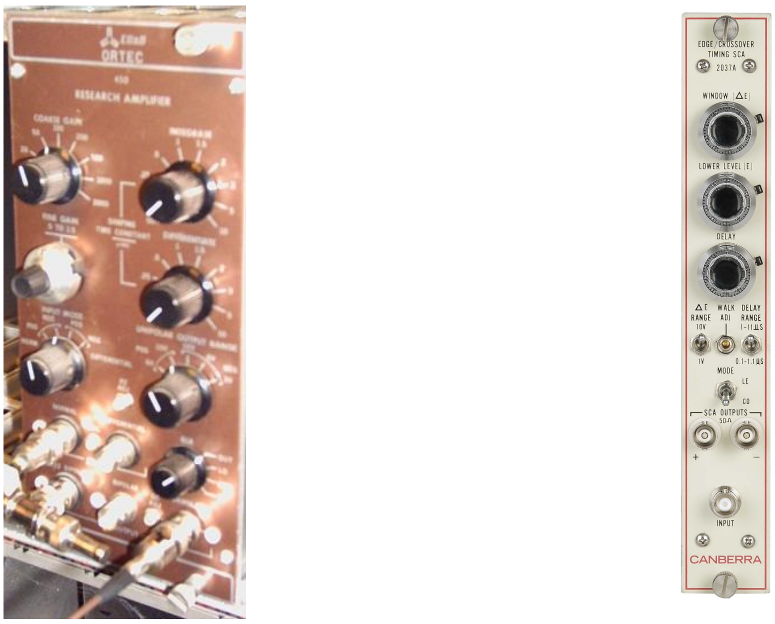


Figure 122: Left: picture of the ORTEC450 Right: single channel analyzer 2037 A [CA09].

- Single channel analyzer

We used a 2037 A module from CANBERRA to select the 511 keV peak from the signal that the PMT was providing us. Figure 122 shows a single channel analyzer of the type we used in our setup.