



Beam Monitor System for high-energy photons

Claire Van Ngoc Ty

June 2008



Contents

I Introduction

- 1.1 Context
- 1.2 Current Method Measurement

II Production of the photon beam

- 2.1 Photon Tagging System
- 2.2 Tagging Spectrometers
- 2.3 Array of Detectors

III Set-up of the beam monitoring system

- 3.1 Program
- 3.2 Modifications
- 3.3 Testing

IV Camera

- 4.1 Example of the output of the camera
- 4.2 Technical aspect

V Conclusion

- 5.1 Cost of the project
- 5.2 Perspectives for the future

Appendix A: Camera Tests

Appendix B: Instructions for the program

Appendix C: C++ Code

Abstract

An online beam monitor system has been built for a high-energy photon beam. Experiments in nuclear physics are performed on a 100-250 MeV photon beam produced through the bremsstrahlung process at MAX-lab in Lund. A program developed to diagnose the photon beam, was done in a master thesis Ref [1]. The final version of the program was able to treat pictures taken from a CCD video camera and to calculate an average position of the beam. The system includes a converter screen put in front of a scintillating screen viewed by the camera. However the set-up was only successful for the electron beam. The main problem was that the camera used was not sensitive enough when exposed to the high-energy photon, as the scintillator was not giving enough light. [1]

In this current project, the aim is to make the previous system works for the photon beam. Either the program can be improved and/or the general set-up can be modified to get a useful beam spot for position diagnostics.

I Introduction

1.1 Context

In nuclear physics research, an electron beam can be used to produce a tagged photon beam through the bremsstrahlung process. During experiments, the position and the size of the beam are useful information since the measured cross-section of a reaction depends on how the beam hits the target.

1.2 Current Measurement Method

The position is measured before the experiment is started through a Polaroid film in the beam. A few minutes' exposure is needed. Furthermore a thin sheet of metal is put in front of the film in order to convert the photons into an electromagnetic shower. However this operation takes time and doesn't inform the experimenter if something happens during the experiment.[1]

II Production of the Photon Beam

At MAX-lab, the MAX-1 storage ring can not only be used to store the beam. When used for nuclear physics its purpose is to stretch the electron beam to obtain a continuous beam of mono-energetic electrons.

2.1 Photon Tagging System

The tagging system can be seen on fig1.

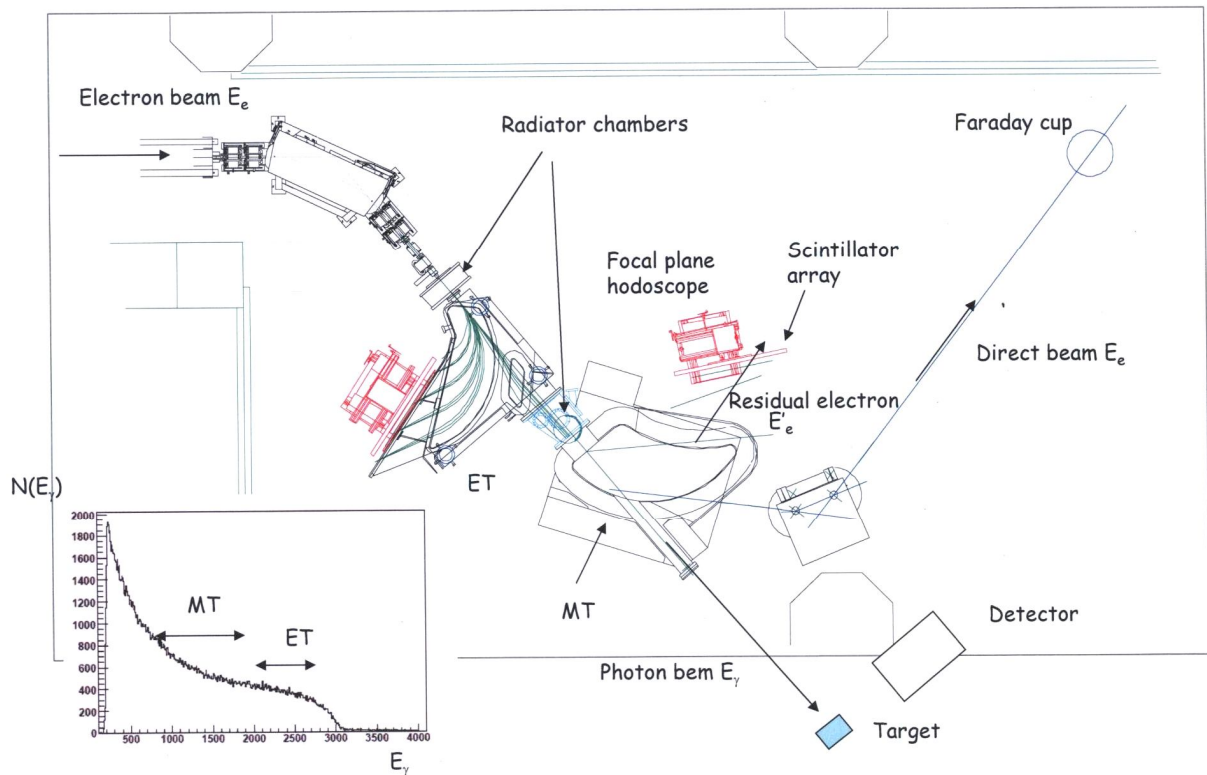


Fig 1: Tagging photon set-up

As said previously the photon beam is produced through the bremsstrahlung process, so the photons form a continuous spectrum. See fig 2.

$$E_{\text{photon}} = E_{e^-} - E_{e^- \text{ measured}}$$

Then, the energy of the photon can be derived by measuring the electron energies after the bremsstrahlung process since we also know the energy of the incoming photon.

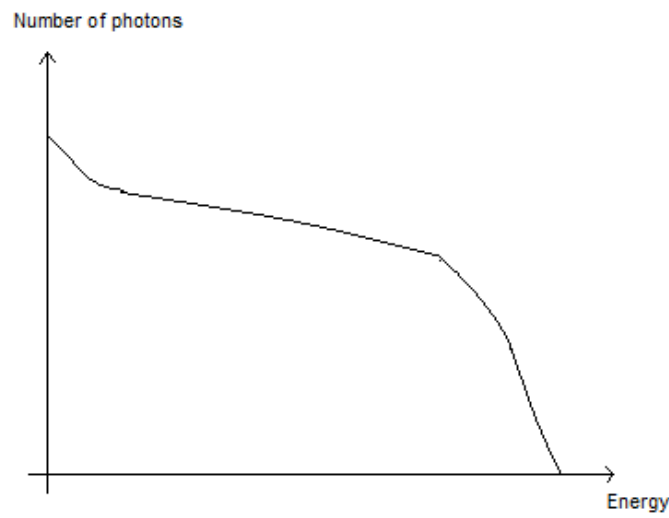


Fig2 : Bremsstrahlung spectrum

The bremsstrahlung process can be started by a radiator. After the electron beam goes through the radiator, to get rid of the electron beam magnets are placed to bend it. Moreover detectors are placed in the focal plane of the spectrometer. As the electrons have different energies, the bent radius will be different.

Finally through coincidence measurements, it is possible to derive the energy of a specific photon which interacts with the target.

2.2 Tagging spectrometers

In this set-up, there are two types of tagging spectrometer. (See fig 1) One is an endpoint tagger (ET) that tags the photons close to the bremsstrahlung endpoint. For an incident electron beam of 250 MeV, it will be able to tag the photons in the energy range of 180-225 MeV. The other one is a main tagger (MT) which will tag photons of a wider range but lower energy (98-163 MeV). [1] The two tagging system can not be used at the same time, so that it is possible to use the same detectors.

2.3 The array of detectors

By putting two series of plastic scintillators which overlap (see fig1), it is possible to obtain a good energy resolution.

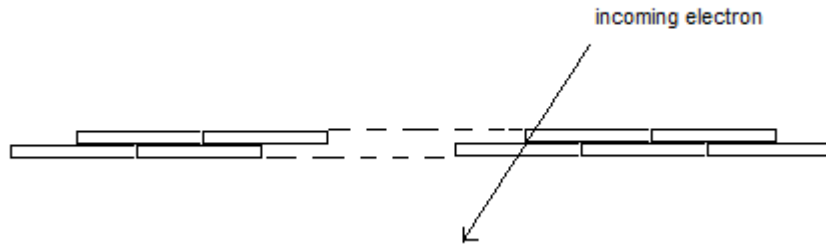


Fig 3: Array of detector

III Set-up of the beam monitoring system

The set-up will follow this principle; the photon beam is converted through a converter in an electromagnetic shower. Then electrons will scintillate in the scintillator. Then the size of this electromagnetic shower is monitored.

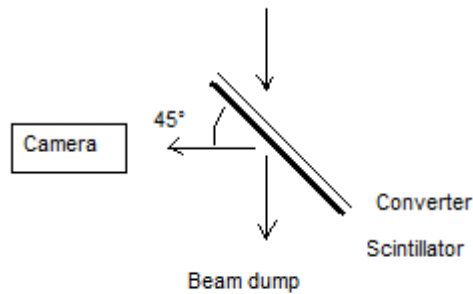


Fig 4: Principle of the set-up

The converter used in our case was a layer of lead which thickness is 3 mm. The scintillator was a "curix orthofine" medium (Gd-compound) from AGFA. This type of scintillator is suitable for high energy gamma. (See appendix A for scintillator/converter tests) Indeed a compromise has to be found to choose the scintillator, since a thin scintillator will give a better precision of the photon beam position but less light whereas a thick scintillator gives electrons with lower energy providing more light to the camera.

3.1 Program

Several versions of the program were available, I am not sure that I picked the best version. So all my comments will be based on the file `getpixel.cc` [1].

From the previous work done on the project, the program was able to take a picture in a jpeg-format (jpeg referring for a compression method for photographic image) from the camera, then to convert it to a bmp-format (photographic format suitable for programming). The picture is treated by the function `ReduceNoise`; this function is looking at the value of the surrounding pixel, and depending of the threshold set for the noise, some pixel might be set to zero. The aim of this function is obviously to try to reduce the noise as much as possible.

Afterwards, the program is using several classes and functions of the ROOT, program package developed by the CERN. For instance the picture is considered as a histogram, so that it is possible to do projections in the X and Y axis.[2] Actually, it is a Gaussian-fitting that is performed, to obtain the mean position on the X and Y axis.

When it comes to the interface, the user was able to see the average mean of X and Y with an error bar.

3.2 Modifications

From the program, I tried to do small improvements. The number of picture taken for the analysis of the beam spot is asked to the user; however the former version did not wait for the user to enter a value. Now the program is really waiting for the user to insert the number of pictures that is treated for a scatterplot. A "counter" variable that appeared to be useless was suppressed. Previously, there were six counters in the program, now there are five counters with more meaningful names.

Though this project is mainly done to have a simple monitoring of the beam, it might be good to try to minimize the timing of the program, in order to have the best monitoring. Indeed, the sleep function is needed in the program otherwise errors might appear. So, instead of having a sleep of 1s, a sleep of 81 ms was used. Moreover, in the previous version one could choose a "loopinterval" to foresee the number of picture taken per minute but since it is better to have as many pictures as possible this variable was taken out.

A color image is also displayed, this give an idea of what the beam may look like. More details will be given in the next part.

As mentioned previously, there is a ReduceNoise function of the ref [1]. But the only access to threshold was to go inside the code. In this version the user is asked to enter a value.

The mean position of the beam is obtained by a Gaussian fitting however the projection might not correspond to a Gaussian. In the last version, the user can at the end of the program chooses if he wants to see the projection obtained and at the same time have a look on how well the fitting where done.

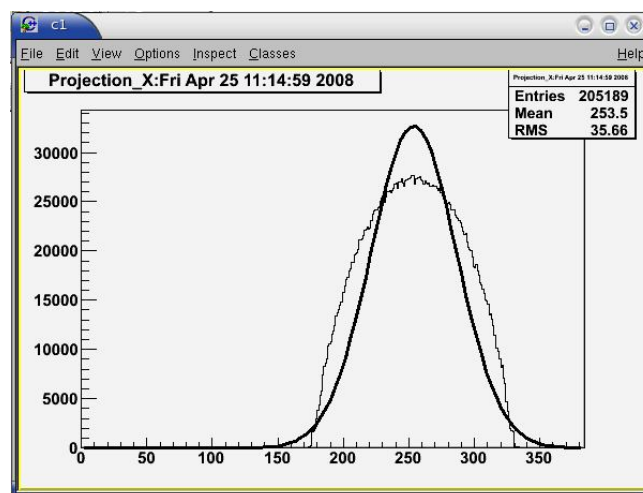


Fig 5: Example of fitting done by the root.

In addition, for each name chosen by the user, the last picture used for the analysis is saved.

In this version of the program, it can be obtained up to 9-10 scatterplots per minute with a « live-image ». Without this image one could have up to 13-14 scatterplots per minute.

3.3) Testing

During testing, it was realized that it might be good to have a “live-image of the beam”. Actually it is the treated image that is shown to the user with the help of a standard colour palette. However this function reduces the maximum number of photo that can be treated by the computer. Therefore, this function was at first an option of the program. However, there is currently no TV-screen to visualize what is seen from the camera so it is now always displayed.

The Reducenoise function can treat the picture but a threshold is needed, the first value tried was 90. Trough different tests, the level have been set to 110, 150, 180, using different thicknesses for the radiator: 300 μ m silicon, 50 μ m aluminum. All this factors will affect the mean value of the position obtained by the fitting.

As the program is also there to diagnose if there are any problems with the beam, the beam was steered in order to see if the mean value of the position was changing and the live image were changing. Depending on the radiator thickness, the size of beam spot will be small or larger. However, it is easier to see any changes when the beam spot is smaller.

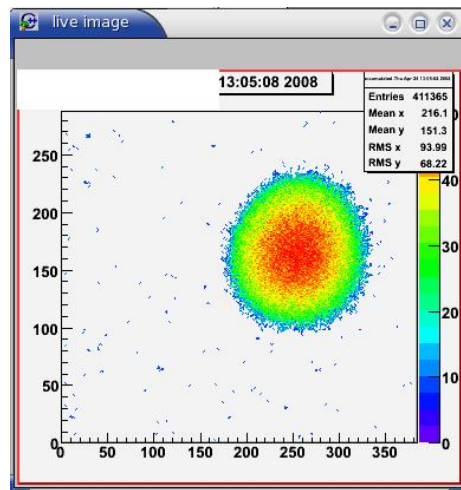


Fig 6: Coloured image of the beam with a noise at 90.

In addition, the program is saving all the colored pictures in the root file with the projections. A way to access to them is to look offline when the program is asking at the end.

From the different tests, it is advice to the user to always allow some noise, since it helps to see that the beam spot is moving.

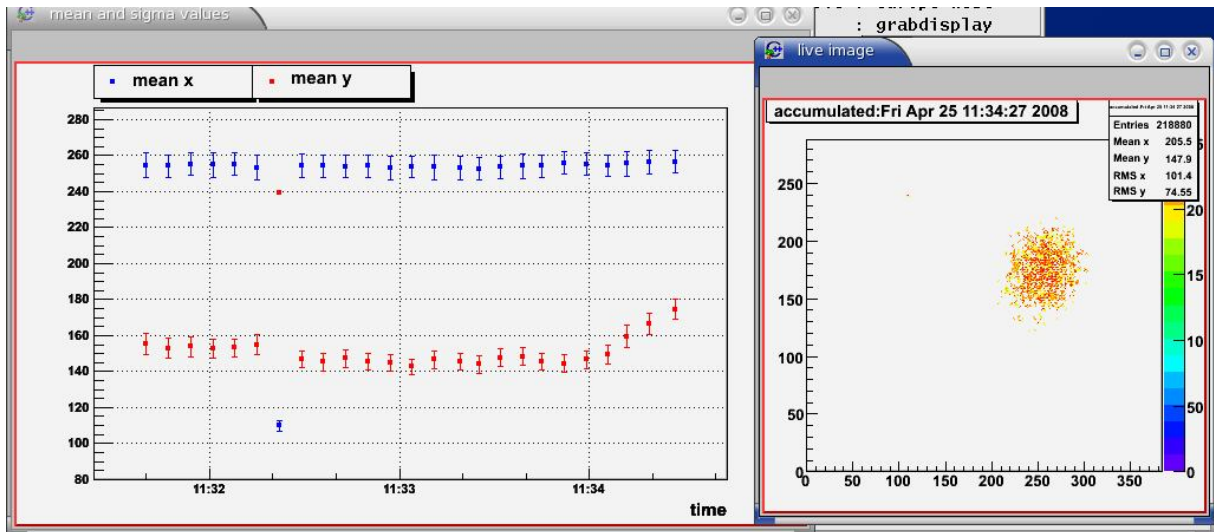


Fig 7: Evolution of the mean y position while steering with a high threshold for the noise level (180).

As can be seen on the fig.7, fluctuations in the intensity of the gamma-beam can occur from the picture taken (7th value).

3.4 Influence of parameters

Two parameters are asked to the user, one of them is the number of photos accumulated by the program. The more picture frames are accumulated, the better the statistics are, but one will lose in the timing efficiency.

The other parameter will affect the mean position in a different way; pixels containing a low value (below the threshold) will be set to zero. This helps for the fitting, below one can see different levels of noise (fig 8 a and b).

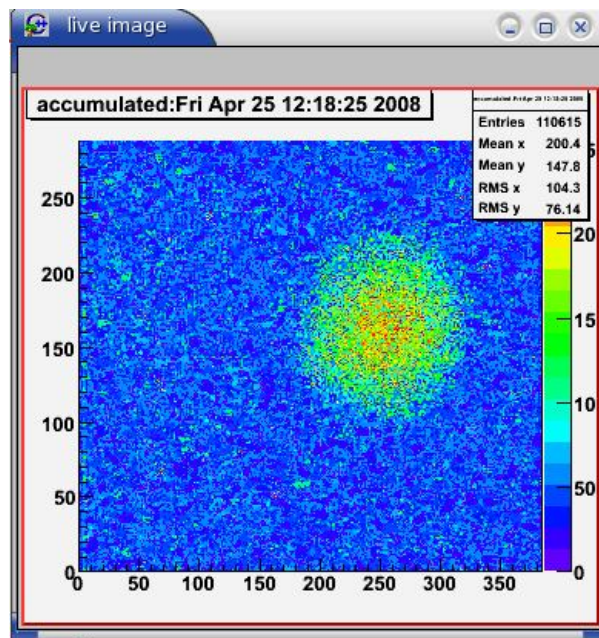


Fig 8: a) Beam spot with the noise level set to zero and with the 50 μm radiator.

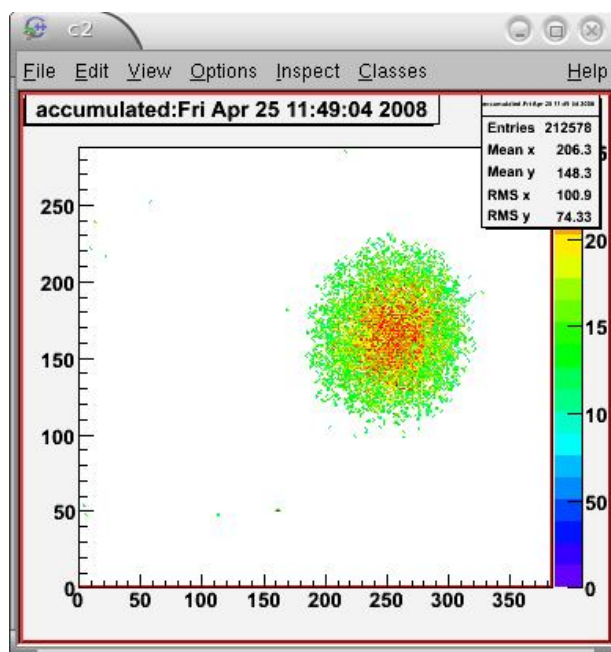


Fig 8: b) Beam spot with the noise level set to 110 and with the 50 μm radiator.

IV Camera

4.1) Example of output of the camera

As said previously, the main problem comes from the sensitivity of the camera. With the old cameras, there were always dots on pictures. As the spot that one can get is faint, it was quite hard to get accurate projection on the x and y axis.

As the photon beam is not always available, it was good to simulate it for the camera. Through grey filters and a laser pen it was possible to simulate a spot.



Fig 9: Simulation of beam with old camera with a 40- filter and a ruler in front of the laser

4.2) Technical comparison

In the previous project done, a MTV261CM was chosen. However in this project, the final test was made with a StellaCam II. Another camera was also used during the project.

MTV261CM: its sensitivity is down to 0.05 lux. It can endure temperature from -20 to 50 °C.

TVCCD-240 by MONACOR: this camera was developed for surveillance system. This camera has 500x582 pixels. Its sensitivity is lower than 0.001 lux. It can endure temperature from 0 to 40 °C.

The new camera was actually recommended in ref [1]. It has the following characteristics.

StellaCam II: The minimum illumination required is 0,00002 lux. It can withstand a temperature between -10 to 40 °C. The camera is given with a remote control; several functions are available, for instance the camera can do frame accumulation. Ref [3]

The new camera is much more sensitive, it is this one that was used in the latest test. During tests, the frame accumulation was set to 4.

While the tests were done, some constant dots were appearing in the camera. This comes from the radiation inside the set-up. This may become a problem later on.

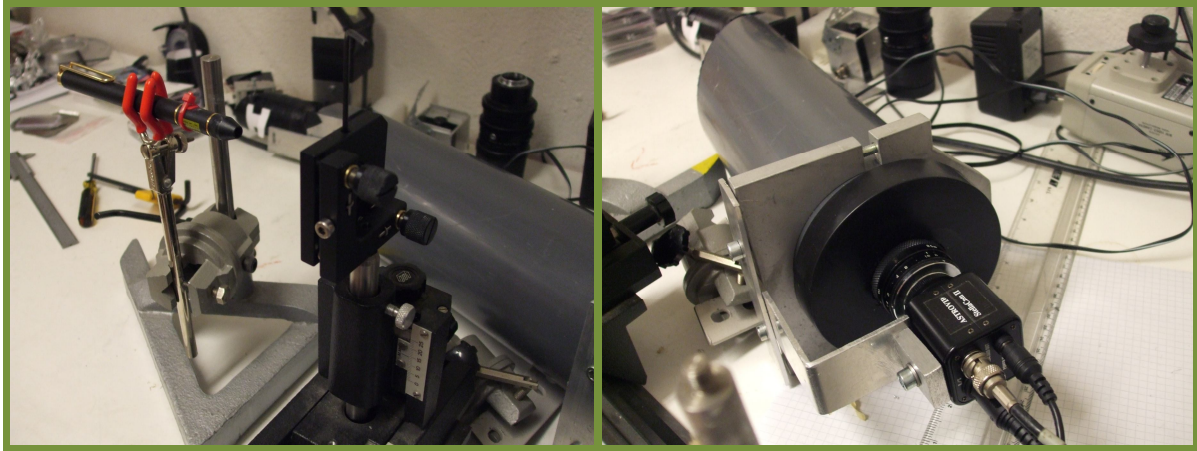


Fig 10: Laser pen and grey filters (right picture) / camera set-up

V Conclusion

I felt that it was hard to go through all the different stages of the program, as I had to learn C++ and also understand how the program was organized at the same time. During the project, there were only two weeks where the photon beam was on.

From the tests done, it can be said that with energetic photons one is able to see a clear beam spot. And the program is able to treat this picture. The user should be able to see the movement of beam either from the mean x and y values or the live image.

Comparing to the current method of measurement, much time is saving for the same result. This technique is efficient but of course can be improved in order to have a monitoring during the whole experiment behind targets. It may as well help in the data cuts since it is possible to see when the intensity of the beam was low or when the position has completely changed.

5.1 Cost of the project.

The main cost of the project is due to the purchase of the new camera. It cost around 5000 SEK. The screen used as a scintillator was from a hospital.

5.2 Perspectives for the future:

It should be mentioned that the program was run on a quite old computer. It is possible to improve the timing of the program, indeed several times the sleep function is called in order to give the computer time to respond correctly. Perhaps if the program is moved on a new machine, some of the sleep function could be taken out.

Moreover as said earlier, the fitting is made by a Gaussian but this fitting is not the best depending on the beam shape. It would be nice to find another fitting function more suitable.

Of course, the way the code is written could be improved. This may also improve the timing of the program.

Acknowledgments

First of all, I would like to thank Bernt Schröder, Kurt Hansen, and Lennart Isaksson for giving the opportunity to work on this project. And for all the advices and explanations they gave to me. Many thanks also for the staff of Max-lab, because I really enjoyed being here.

I have to say thank you to Jennie, since she did the hardest part of the programming. Thanks to this project, it gave me back some hopes about computer science.

Special thanks for Kurt Hansen for explaining and do the set-up for the project.

I would like also to thank Magnus Lundin for the time he spent helping me to understand how the program was working and also how to improve it.

Special thanks also for Lund University Hospital, Gunnilla Holje, who has given to MAX-lab the scintillator we used.

Finally, I would like to thank my family especially my mother for her supporting, and my fiancé Thomas who spent time with me to get more familiar to Linux.

Appendix A

Camera tests 15/3/2008 by Kurt Hansen

The tests were made to prepare for the final efforts to get a video image of the high-energy photon beam from the Tagging system at MAX-lab. All tests were made with the cameras positioned between the vacuum exit window and the collimator used to define the beam size to the experiments. The aim was to determine the best compromise between light output from the scintillator and the sharpness of the beam spot on the scintillator. Two cameras were used. One was the same used in Jennie Bergs master thesis, the MTV261CM with 0.05 Lux sensitivity, and the other was a new camera from Monacor, TVCCD-624E with a sensitivity of 0.001 Lux.

The tests were made with 144MeV, 8-10 mA current in the MAX-1 ring, around 3-4 nA current in the Faraday cup and with the focal plane counting around 2 Mhz, with a 300 micrometer radiator.

1. The MTV261CM and the screen made of a 3 mm thick powder scintillator specifically made for high energy gammas and a 3 mm lead converter in front of the screen. Result; an easily seen beam spot with rather fuzzy edges, visible also with 150 micrometer radiator.
2. Same camera as 1. but with a plastic scintillator 2.5 mm, used for particle detectors and a 3 mm lead converter in front of the scintillator. Result; a very diffuse light ('aurora borealis'-type) mainly visible at the edges.
3. As above but with a thin steel foil, 50 micrometers, covered with same scintillation powder as used in test 1. Result; also very diffuse light. But with a 2 mm screen as radiator, creating around 5 times more gammas in the gamma monitor detector downstream from the collimator and with the focal plane detectors shut down, there was a visible beam spot on the screen, but diffuse.
4. Test with the Monacor camera, same screen as in test 1, and also a 3 mm lead converter. Result; clearly visible light spot, somewhat diffuse, possibly better than test 1., spot visible also with 150, 100 and 50 micrometer radiator.
5. Same as above but without lead converter. Result; barely visible spot with 150 micrometer radiator.

Conclusions; The lead converter is essential for a good light spot, but the largest contribution to the light output is the thick fluorescent screen, unfortunately this also makes the spot more diffuse.

Appendix B Instructions

To run the program:

Xawtv must be run with the following configuration:

TV-source: composite1
TV-format: NTSC
Capture mode: grabdisplay

Then, please enter `./getpixel` in a console.

1. Enter a name for the file
2. The program is expected a value to reduce the noise, a good value to take away some background is 90.
3. Choose the number of accumulated picture for the scatterplot, for instance 1,2 ...

Projections will be calculated from the accumulation of the scatterplot.

The program can display a colored image from the file "accumulated".

This live image can give an idea of the relative intensity of the beam.

4a) To quit the program press: q

Then, you can choose to have a look on the projection done by the program. Press (1) for a straight exit, (2) to launch a root window.

4b) If one is inside the root, one should go to the directory chosen and double-click on the file. This file is then on the "ROOT files" folder. All projections and accumulated plot are inside this folder. To quit the root, one should press: `.q` to leave the root part.

If the program is not responding, the user can use: `ctrl + c`

Appendix C

Getpixel.cc

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>

// Root stuff
#include <TROOT.h>
#include <TH1.h>
#include <TH2.h>
#include <TFile.h>
#include <TMath.h>
#include <TF1.h>
#include <TF2.h>
#include <TProfile.h>
#include <TCanvas.h>
#include <TGraph.h>
#include <TSystem.h>
#include <TApplication.h>
#include <TAttFill.h>
#include <TFile.h>
#include <TStyle.h>
#include <THistPainter.h>
#include <TProfile2D.h>
#include <TAttMarker.h>
#include <TMultiGraph.h>
#include <TAxis.h>
#include <TLegend.h>
#include <TGraphErrors.h>
#include <Rtypes.h>
#include "reducenoise.h"

#pragma pack(1)
struct BITMAPFILEHEADER
{ unsigned short bfType;
  unsigned long bfSize;
  unsigned short bfReserved1;
  unsigned short bfReserved2;
  unsigned long bfOffBits;
};
#pragma pack()

#pragma pack(1)
struct BITMAPINFOHEADER
{
  unsigned long bsize;
  unsigned long biwidth;
  unsigned long biheight;
  unsigned short biplanes;
  unsigned short bibitcount;
  unsigned long bicompression;
```

```

unsigned long bsizeimage;
unsigned long bixpelspermeter;
unsigned long biypelspermeter;
unsigned long biclrused;
unsigned long biclimportant;

};
#pragma pack()

#define DEBUG 1

#define STRINGLENGTH 128

//Some def.
unsigned short irow;
unsigned short icol;
unsigned short nrow;
unsigned short ncol;
char logname[10];
char buff[STRINGLENGTH];
ssize_t bytes;
unsigned long int counter = 0;
unsigned int counter2 = 0;
unsigned int counter_picture = 0;
unsigned int counter_total_pict = 0;
Double_t seconds;
char str[128];
unsigned int datapoints = 0;
unsigned int pos = 0;
unsigned int binnumber;
unsigned int savehisto;
unsigned int errormin;
unsigned int errorplus;
int response =0;
Double_t gausYmin;
Double_t gausYmax;
Double_t gausXmin;
Double_t gausXmax;
unsigned int NumberOfPictures = 0;

//For function "ReduceNoise"
Int_t *Xmax = new Int_t();
Int_t *Ymax = new Int_t();
Int_t *noiselevel= new Int_t();
Int_t reducepixel;

int main(int argc,char* argv[])
{
TApplication theApp("App", &argc, argv);

//This will probably not be of use for the beams and therefore can be set to
//1, but i kept it anyway. When the .bmp image is plotted in a histogram
//(scatterplot) it will for each pixel in the scatterplot plot a number of

```

```
//black dots corresponding to the gray-scale number in the picture. If the
//.bmp image has low contrasts the scatterplot might be black (or white if
//too many pixels are removed by ReduceNoise) This problem might be solved
//through putting a value >1 to reducepixel. (The value zero is forbidden!)
reducepixel = 1;
```

```
//Here you can change the number of events showed in the online graph
//(the one called "mean and sigma values" showed on the display when the
//program is running)
//Maximum value allowed is 999
binnumber = 25;
```

```
//Here you can choose at what loop interval to save the histograms and their
//projections in the root-file. If you, for example, set this number to
//30 they will be saved every 30:th time the program loops. (For the other
//29 loops this information will not be saved.)
savehisto = 1;
```

```
//This numbers scale the size of the errorbars in the online graph. If you
//want to make them smaller, put errorplus = 1 and use errormin (for which
//the number 0 is forbidden!) and if you want to make them bigger put
//errormin = 1 and use errorplus. If you don't want any errorbars, set
//errorplus = 0.
errormin = 3;
errorplus = 1;
```

```
//If the beam is only a thin peak on the projection on the X-axis, it might
//be preferable to do the gaussian fit over a smaller range. These
//coordinates can be set here. For this Kjell & co card, and the camera
//I use, there are 384 pixels (= bins in plot) on the x-axis and 288 on
//the y-axis if you HAVE NOT changed the size of the online camera display
//window.
gausXmin = 0;
gausXmax = 384;
```

```
gausYmin = 0;
gausYmax = 288;
```

```
//-----
```

```
TH2F * accumulated = NULL;
// TH2F * copyacc = NULL;
```

```
//Pointers to the files where the information are stored
FILE *pGAUSS = NULL;
FILE *pFILE = NULL;
```

```
Double_t meanX[1000];
Double_t sigmaX[1000];
Double_t meanY[1000];
Double_t sigmaY[1000];
Double_t loopnr[1000];
Double_t errorx[1000];
Double_t errory[1000];
```

```

struct BITMAPFILEHEADER bmfh;
struct BITMAPINFOHEADER bmih;
//pixel and row is used if you want to print values of bitmap-pixels
// to screen (see below)
// unsigned int pixel, row;
unsigned char *buffer = NULL;

/*Here te user is asked to type the name of the file where the data
and histograms will be stored. Two files are created and opened for
writing using this name, one root file "logname.root" for the histograms,
and one text file "logname" for the data*/
printf("To start the program, type desired name of file (maximum 8 char), followed by return.\nIf the name of
an already existing file is chosen, the data will be added to the end of this file.\n");

scanf("%s", &logname); //assign input from keyboard to the variable "logname"
pGAUSS = fopen(logname,"a"); //Textfile

sprintf(str, "%s.root",logname); //root file
TFile *hfile = new TFile(str, "UPDATE", "histograms"); //root file

//Some commands for the while (1) loop
fcntl(STDIN_FILENO, F_SETFL, fcntl(STDIN_FILENO, F_GETFL) | O_NONBLOCK);

//Output: information for the user when the program continues into the loop
printf("Program is running. Press q followed by return to quit program.\nPlease wait...\n\n");

//Create canvas to draw "online graph" within
TCanvas *c1 = new TCanvas("c1", "mean and sigma values",0,350, 800,500);
c1->SetGridx();
c1->SetGridy();

//Here the program ask the user how many pictures should be accumulated in
//each scatterplot. This number is assigned to the vaiable NumberOfPictures
//The same number is used in the background scatterplot.

printf("please give a value for the noiselevel .\n");
int response1 =0;
while (response1 == 0)
{scanf("%i", &response1);
}
noiselevel = &response1;
printf("The program is now ready to make scatterplot. \nHow many images do you want to accumulate in each
scatterplot?\n\nType the desired number followed by return.\n\n");
///Chec that the number is not zero
while(NumberOfPictures == 0)
{
// scanf("%i", &NumberOfPictures);
// printf("Invalid. The number must be at least 1.\n\nTry again.\n\n");
scanf("%i", &NumberOfPictures);
}

printf("accumulating the scatterplot.\n\n");
//Here starts the loop that will continue until the user press q return.
while(1)
{

```

```

counter++; //for use when saving histograms at certain loop intervals

// usleep(10);

//get current time and date (for use later in the program)
//See www.cplusplus.com/ref/ctime
//These lines may not be moved anywhere else in program by two reasons.
//If the definitions are moved outside the loop, strange dating will occur
//in the files. The second last line is the one that get the current time,
//and should therefore be just after the photo is taken.
time_t rawtime;
struct tm * timeinfo;
time(&rawtime);
timeinfo = localtime (&rawtime);
static char currenttime[30];
strcpy(currenttime, asctime (timeinfo));
//Previous line gets the current time and assigns it to "currenttime"
currenttime[24] = 0; //remove newline (\n) from "currenttime"

seconds = time(NULL); //Get time in seconds since (UNIX time)

//This code is to extract the hour from the time string above
/* char hour[3];
hour[0] = currenttime[11];
hour[1] = currenttime[12];
hour[3] = '\0';
printf("hour %s\n", hour);
int h;
h = atoi(hour);
printf("h is %d\n", h);*/

char *tmpname = "last_saved_picture";

//fprintf( stdout, "je suis la\n");
for(counter_picture = 0; counter_picture < NumberOfPictures; counter_picture++)
{
    // Snap a photo

    sprintf( str, "xawtv-remote snap jpeg win %s%u.jpeg", tmpname, counter_picture);
    system( str);
    usleep(81000);

    sprintf( str, "cp /home/kfoto/%s%u.jpeg .", tmpname, counter_picture);
    system( str);

    // This is not elegant, but one has to wait otherwise convert below won't
    // understand that the jpeg file is there. Could poll instead.
    //sleep(1);

    // Convert to bmp format
    sprintf( str, "convert -depth 8 -colors 256 -compress none %s%u.jpeg %s%u.bmp", tmpname,
counter_picture, tmpname, counter_picture);
    system( str);

    sprintf( str, "%s%u.bmp", tmpname, counter_picture);

```

```

pFILE = fopen( str, "rb");
if( pFILE == NULL)
    {
        printf("error opening %s", str);
        exit(1);
    }

// get the file and info headers
if ( fread(&bmfh,sizeof(struct BITMAPFILEHEADER),1,pFILE) != 1)
    {
        printf("error reading fileheader\n");
        exit(1);
    }
if ( fread(&bmih,sizeof(struct BITMAPINFOHEADER),1,pFILE) != 1)
    {
        printf("error reading infoheader\n");
        exit(1);
    }

// fprintf( stdout, "je suis la1\n");

buffer = (unsigned char *)malloc(bmih.bisizeimage);
if ( buffer == NULL)
    {
        printf("error allocating buffer\n");
        exit(1);
    }

// skip forward to where the data begins
if ( fseek( pFILE, bmfh.bfOffBits, SEEK_SET) != 0)
    {
        printf("error advancing in file\n");
        exit(1);
    }

if ( fread( buffer, bmih.bisizeimage, 1, pFILE) != 1)
    {
        printf("error reading from file (image)\n");
        exit(1);
    }
fclose(pFILE);

// fprintf( stdout, "je suis la2\n");

//Set some variables that will be used to fill scatterplot
nrow = bmih.biheight;
ncol = bmih.biwidth;

//Create new TH2F histogram
sprintf(str, "scatterplot:%u%s",counter_picture, currenttime);
TH2F *scatterplot = new TH2F(str,str,ncol,0,ncol,nrow,0,nrow);

//Fill TH2F histogram with the data extracted from the bitmap-file

```

```

for (irow = 0; irow < nrow; irow++)
{
    for (icol = 0; icol < ncol; icol++)
    {
        scatterplot->Fill(icol, irow,(buffer[(irow*ncol + icol)])/reducepixel);

    }
}

```

//Get the number of bins on X and Y axis for use in the functions below

```

* Xmax = scatterplot->GetNbinsX();
* Ymax = scatterplot->GetNbinsY();

```

```

if(counter_total_pict % NumberOfPictures == 0)
{
    //histogram to add scatterplots within
    sprintf(str, "accumulated:%s", currenttime);
    accumulated = new TH2F(str,str,*Xmax,0,*Xmax,*Ymax,0,*Ymax);
}

```

```
counter_total_pict++;
```

//This function, which is defined outside main, reduces noise from the
//camera.

```

ReduceNoise(scatterplot, Xmax, Ymax, noiselevel);
accumulated->Add(scatterplot,1);
scatterplot->Delete();

```

```
}
```

```

//Make a projection of the scatterplot on the X-axis, TH1D histogram
sprintf(str, "Projection_X:%s", currenttime);
TH1D*Projection_X = new TH1D(str,str,*Xmax,0,*Xmax);
accumulated->ProjectionX(str,0,*Ymax);

```

```

//Fit gaussian to the projection. The options of the function "Fit" are
//specified in the root homepage; root.cern.ch
sprintf(str, "f1:%s", currenttime);
TF1*f1 = new TF1(str,"gaus",gausXmin, gausXmax);
Projection_X->Fit(str,"RQ");

```

```

//Get the mean and sigma values of the fit
Double_t f1mean = f1->GetParameter("Mean");
Double_t f1sigma = f1->GetParameter("Sigma");

```

```

//Make a projection of the scatterplot on the Y-axis, TH1D histogram
sprintf(str, "Projection_Y:%s", currenttime);
TH1D*Projection_Y = new TH1D(str,str,*Ymax,0,*Ymax);
accumulated->ProjectionY(str,0,*Xmax);

```

```

//Fit gaussian to the projection. The options of the function "Fit" are
//specified on the root homepage; root.cern.ch
sprintf(str, "f2:%s", currenttime);
TF1 *f2 = new TF1(str, "gaus", gausYmin, gausYmax);
Projection_Y->Fit(str, "RQ");

//Get the mean and sigma values of the fit
Double_t f2mean = f2->GetParameter("Mean");
Double_t f2sigma = f2->GetParameter("Sigma");

// accumulated->copy(%copyacc);
// gStyle->SetPalette(1);
// copyacc->UseCurrentStyle();
// gROOT->ForceStyle()

//Save the scatterplot and projections in the root file "logfile.root",
//every time the quote "counter/savehisto" equals to zero. (counter
//counts the number of loops, starting from one when program is started.
if (counter % savehisto == 0) //Here the number can be changed
{
    hfile->Write();
}

//This writes the data to "logfile" together with "curenttime"
fprintf(pGAUSS, "\n %s\n %lf %lf %lf %lf\n", currenttime, f1sigma, f1mean, f2sigma, f2mean);

//Write the values sigma and mean to arrays, for use in histograms.

meanX[counter2] = f1mean;
sigmaX[counter2] = f1sigma/errormin*errorplus;
meanY[counter2] = f2mean;
sigmaY[counter2] = f2sigma/errormin*errorplus;
loopnr[counter2] = seconds;
errorx[counter2] = 0;
errorY[counter2] = 0;

//The if statement below is used to set the number of points in the graphs
//below. It has to be the same as the actual numbers of values in the strings
//that are plotted, else the x-axis will look funny.
if(counter < binnumber)
{
    datapoints = counter;
}
else
{
    datapoints = binnumber;
}

//Search minimum and maximum value of arrays to scale the graf y-axis
//in the online graph
float minimum = meanX[0];
float maximum = meanX[0];

```

```

for (pos = 0; pos < datapoints; pos++)
{
    if(meanX[pos] < minimum)
        minimum = meanX[pos];
    if(meanX[pos] > maximum)
        maximum = meanX[pos];
}

```

```

for (pos = 0; pos < datapoints; pos++)
{
    if(meanY[pos] < minimum)
        minimum = meanY[pos];
    if(meanY[pos] > maximum)
        maximum = meanY[pos];
}

```

counter2++; //Used for writing values to arrays

//Create two graphs for the fit values, with sigma values as error bars,
//a multigraph to put them in and clear the canvas to draw it all in.
//The multigraph is draw and c1 updated

c1->Clear();

sprintf(str, "onlinegraph:%s", currenttime);
TMultiGraph *multigraph = new TMultiGraph(str, "values from fit");

```

TGraphErrors *graph1 = new TGraphErrors(datapoints, loopnr, meanX,errorx,sigmaX);
graph1->GetYaxis()->SetRangeUser(minimum - 25,maximum + 25);
graph1->GetYaxis()->SetLabelSize(0.03);
graph1->GetXaxis()->SetTimeDisplay(1);
graph1->GetXaxis()->SetTimeFormat("%H:%M");
graph1->GetXaxis()->SetLabelSize(0.03);
graph1->GetXaxis()->SetTitle("time");
graph1->SetMarkerStyle(21);
graph1->SetMarkerSize(0.5);
graph1->SetMarkerColor(4);
graph1->SetLineColor(4);
graph1->SetTitle("");
multigraph->Add(graph1,"AP");
multigraph->Draw();
c1->Update();

```

```

TGraphErrors *graph2 = new TGraphErrors(datapoints, loopnr, meanY,errorY,sigmaY);
graph2->SetMarkerStyle(21);
graph2->SetMarkerSize(0.5);
graph2->SetMarkerColor(2);
graph2->SetLineColor(2);
multigraph->Add(graph2,"P");
multigraph->Draw();
c1->Update();

```

//Create ledgends for the multigraph. I've made two separate

```

//ones instead of one, because I didn't understand howe to get them on
//one line. (And since it is of no importance I didn't care...)
TLegend *leg1 = new TLegend(0.1,0.99,0.3,0.92,"");
TLegend *leg2 = new TLegend(0.3,0.99,0.5,0.92,"");

leg1->AddEntry(graph1,"mean x","p");
leg2->AddEntry(graph2,"mean y","p");

leg1->SetTextSize(0.04);
leg1->SetTextFont(2);
leg2->SetTextSize(0.04);
leg2->SetTextFont(2);

leg1->Draw();
leg2->Draw();

c1->Update();
if (counter % binnumber == 0)
{
    sprintf(str, "onlinegraph:%s", currenttime);
c1->Write(str);
}

if(counter2 == binnumber)
{
counter2 = 0; //reset to rewrite arrays
}

//This is to avoid saving the same plots again next loop,
//since they are written to strings before saved.
Projection_X->Delete();
Projection_Y->Delete();
f1->Delete();
f2->Delete();

// Live coloured image part

if (response == 0){ printf("This question will be only asked once :Press :(1) Don't show the coloured image
(2) See coloured image of the beam\n");
}
while (response ==0)
{
scanf("%i",&response);}

if (response == 2)
{ TCanvas *c2 = new TCanvas("c2","live image",850,370,450,450);
gStyle->SetPalette(1);
accumulated->UseCurrentStyle();
accumulated->Draw("colz");
accumulated->Write();
c2->Update();
c2->UseCurrentStyle();
c2->Write();
sleep(2);
}

```

```

    accumulated->Delete();

    //Here the program look for an input, q followed by return, which
    //breaks the loop. If there is no input, it loops again.
    bytes = read(STDIN_FILENO, &buff, STRINGLENGTH);

    if (strstr(buff, "q") !=NULL)
    {
        break;
    }
}

int value=0;

printf("(1) Don't show modified picture, (2) See modified picture and projections\n");
while (value == 0)
{

    scanf("%i", &value) ;

}

switch (value)
{
    case 1 : { break;}
    case 2 : {
        printf( "Now you can go to your directory and click on your file, then go to root file to have a look on
histograms\n");
        sprintf(str,"root -l mkmkm.C");
        system (str);
        break;}
    default : {break;}
}

printf("End of program\n");

return (0);
}

```

Reducenoise.cc

```

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>

// Root stuff
#include <TROOT.h>
#include <TH1.h>
#include <TH2.h>
#include <TFile.h>
#include <TMath.h>
#include <TF1.h>

```

```

#include <TF2.h>
#include <TProfile.h>
#include <TCanvas.h>
#include <TGraph.h>
#include <TSystem.h>
#include <TApplication.h>
#include <TAttFill.h>
#include <TFile.h>
#include <TStyle.h>
#include <THistPainter.h>
#include <TProfile2D.h>
#include <TAttMarker.h>
#include <TMultiGraph.h>
#include <TAxis.h>
#include <TLegend.h>
#include <TGraphErrors.h>

```

```

void ReduceNoise( TH2F *scatterplot, Int_t* Xmax,Int_t *Ymax )
{
//-----

```

```

//Below user can change some parameters in the program. Remark that
//they all have to be positive integer numbers! (Some might be zero)

```

```

//The function "ReduceNoise" removes information (noise from the camera)
//form the scatterplot before any other treatment of the data
//by setting the pixel value to zero if the closest pixels (four pixels)
//all have a value below the number "badpixel" that you can set here.
//(If you set "badpixel" to zero, the scatterplot will not change.)

```

```

Int_t badpixel;
badpixel =60;

```

```

//The function "ReduceNoise" also replace the value in a bin in the
//scatterplot by zero if it is lower than "noiselevel". If you set
//"noiselevel" to 0 it means that no values are replaced.

```

```

Int_t noiselevel;
noiselevel =60;

```

```

Int_t binx;
Int_t biny;
Int_t minmin;
Int_t minplus;
Int_t plusmin;
Int_t plusplus;
Int_t binxmin;
Int_t binxplus;
Int_t binymin;
Int_t binyplus;
Int_t thispixel = 0;

```

```

for(binx = 1; binx <= *Xmax; binx++)
{
for(biny = 1; biny <= *Ymax; biny++)
{
binxmin = binx - 1;
binxplus = binx + 1;
binymin = biny - 1;
binyplus = biny + 1;

```

```
minmin = (Int_t)scatterplot->GetBinContent(binxmin,binymin);
minplus = (Int_t)scatterplot->GetBinContent(binxmin,binyplus);
plusmin = (Int_t)scatterplot->GetBinContent(binxplus,binymin);
plusplus = (Int_t)scatterplot->GetBinContent(binxplus,binyplus);
thispixel = (Int_t)scatterplot->GetBinContent(binx,biny);

if((minmin<badpixel && minplus<badpixel && plusmin<badpixel && plusplus<badpixel) || thispixel < noiselevel )
{
scatterplot->SetBinContent(binx,biny,0);
}
}
}
}
```

References

[1] Jennie Berg, Master Thesis : « Beam Monitor System » for the 250 MeV Photon beam at MAX-lab.

[2] Root: <http://root.cern.ch>

[3] Website of Astrovid for the StellaCam II <http://www.astrovid.com>