Particle detection with a CMOS camera

ICTP-SAIFR Second School on Dark Matter and Neutrino Detection

June 8-19, 2024

All the materials in this course were developed and put together by the **LAMBDA laboratory** at Buenos Aires University and **Fermilab**. For comments and questions, you can contact me directly at botti.anamartina@gmail.com

Abstract

This course is divided into two parts. At the beginning of each part, I will present a summary and guidelines for the activity (see slides), and you will find discussion questions to consider during the activity. We will use the last 15 to 20 minutes of each block to share the answers to these questions. Part 1 consists of setting up and running a system to calibrate a pixeled silicon sensor with a Raspberry Pi 5, LEDs, and a Sony IMX477 camera. Part 2 consists of defining operation parameters and taking data with the camera to detect air shower muons. You will work with pre-compiled datasets and use your calibration to define quality cuts and selection criteria to search for muons in your dataset. An extra data set of 241Am X-rays is available to reconstruct and interpret their energy spectrum.

Contents

1 WARNING notes before starting

- Contacts at the bottom of the Raspberry Pi and Camera are exposed. Do not remove the isolating tape from the camera or the raspberry from the case.
- Before connecting or disconnecting anything to the Raspberry, please shut it down, connect, and then on again. Any misplace during installation can produce a short and burn the device (trust me, I already burnt a few.)
- Probably this is not the latest version of this guide, which is updated daily. Get the latest version from **here** . Materials for this course can be found **here**, including code, exercises and student data.

2 Part 1: calibration and photon transfer curve

For calibrating the camera, we will use a fundamental principle that states that if we perform several repetitions of an experiment to count photons emitted by an ideal light source, the measured number of photons will follow a Poissonian distribution. This means that the mean and variance of the distribution are equal. When we illuminate the camera with light, the photons produce electron-hole pairs in the silicon. We will measure a signal proportional to the number of charges, and in turn, to the number of impinging photons. However, since the signal is represented in digital counts, we still need to understand how many digital counts we have per photon. The goal of this part is to determine what this relation is.

2.1 Discussion questions

- How can we detect light with silicon?
- Can we detect any wavelength?
- How does a CMOS-camera work?
- What does it mean to calibrate a CMOS sensor?
- What are the sources of noise and background?
- Can we measure 1 photon? Can we measure 1000000 photons?

2.2 Plug everything up and turn it on

Connect the CMOS camera to a display port in the raspberry, the Fan (or cooler) to the fan port, the LED to the GPIO pins, the monitor to the mini HDMI, and the keyboard and mouse to the USB ports:

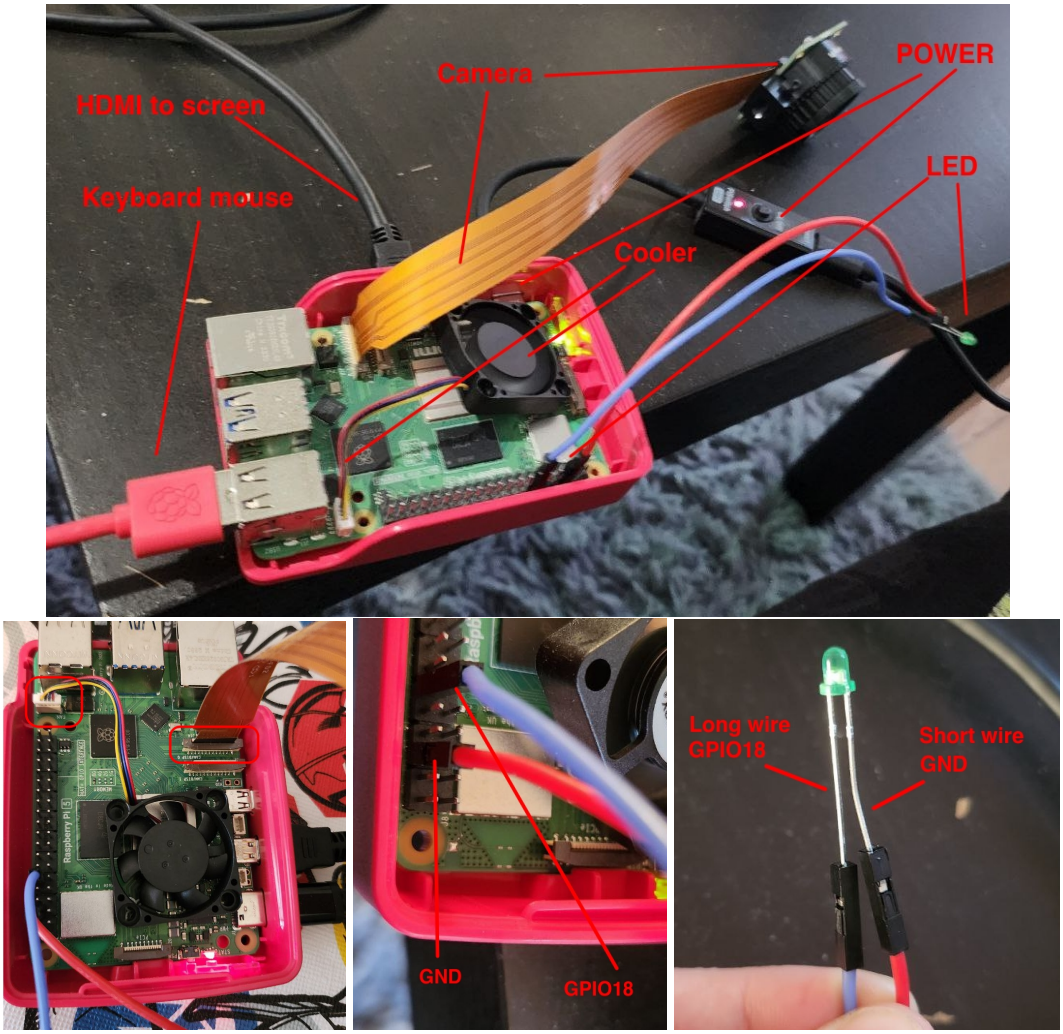


Figure 1: System cabling. The class code will only work with this setting. You can change the setting, but you must modify the code. Watch out for the LED polarity. If you have doubts, you can use the 3V3 pin instead of the GPIO18, which should light the LED as soon as you connect (see 4.1 for the Raspberry pinout to locate the 3V3 pin). I recommend starting with a white LED.

For technical details on the Raspberry Pi and pinout, you can check 4.1. Connect LAST the source power to turn it on (it will automatically turn on). You will be logged into the Raspberry OS graphic interface. At the bottom, you will have the taskbar, with three icons you can open:

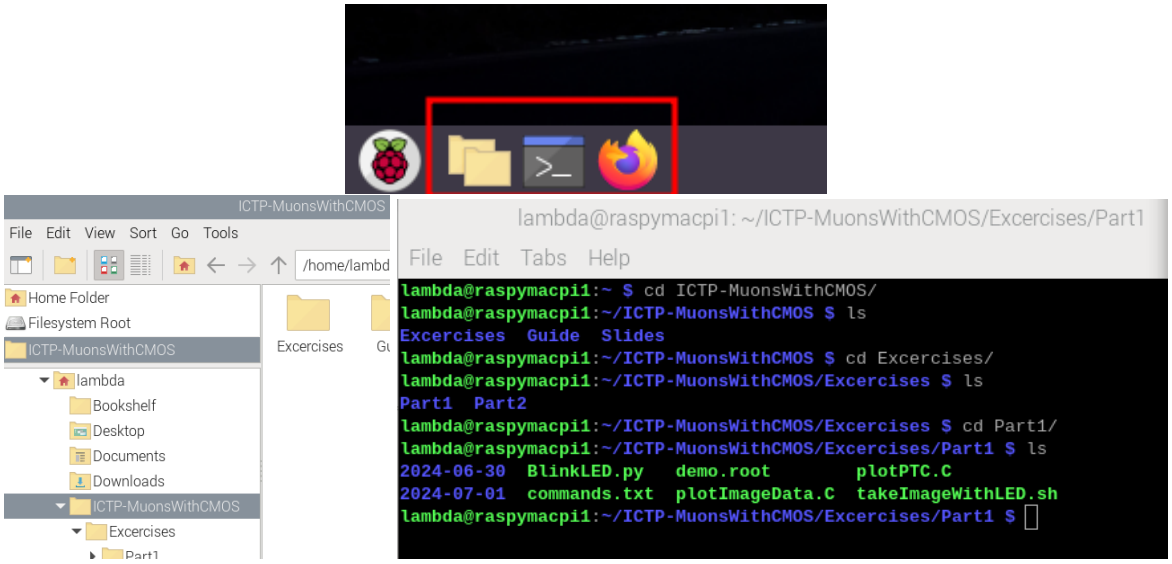


Figure 2: In order from left to right: The folder explorer with quick access to the course script (in case you would like to modify anything, you shouldn't have to). The terminal launcher with the navigation instructions to the exercise folder. *ls* to list files in a folder *cd folder* to enter a folder and *cd ..* to go one folder back; you don't need to write the whole folder name, the tab key will auto-complete. The Firefox launcher will automatically open a Google Drive with the course material, including an updated version of this guide. All materials are already uploaded into the Raspberry, but you can have this in hand if needed.

2.3 Check everything is well connected

First, check the camera is well connected using this command from any location in the terminal:

```
libcamera-hello --list-cameras
```

You should get an output similar to:

```
    Available cameras
-----------------
0 : imx477 [4056x3040 12-bit RGGB] (/base/axi/pcie@120000/rp1/i2c@88000/
    imx477@1a)
        Modes: 'SRGGB10_CSI2P' : 1332x990 [120.05 fps - (696, 528)/2664
            x1980 crop]
        'SRGGB12_CSI2P' : 2028x1080 [50.03 fps - (0, 440)/4056x2160 crop]
                          2028x1520 [40.01 fps - (0, 0)/4056x3040
                              crop]
                          4056x3040 [10.00 fps - (0, 0)/4056x3040
                              crop]
```

Note: if the CMOS camera is not detected, shut down the Raspberry and reconnect to the other port. Let me know if this problem continues.

We are going to use pulse width modulation (PWM) to control the LED; this allows us to define a duty_cycle or intensity, a time window during which the LED will be ON, and an

ON/OFF frequency. You can check some technical details about PWM in 4.2. Check that the LED is well connected and running:

```
BlinkLED.py 50 1000 2000
BlinkLED.py 100 10 3000
```

This will first blink the LED with half intensity for 2000 ms (2s) with a frequency of 1000 Hz and then with full intensity with a frequency of 10 Hz for 3000 ms (3 s). For details of usage, you can run *BlinkLED.py* and get:

```
Usage: python BlinkLED.py <duty_cycle> <frequency> <sleep_timems>
Usage: Duty cycle needs to be between 0 and 100
Usage: frequency maximum is: 10000
Usage: sleep_time is in ms
```

On the terminal, you can use *ls* to list files in a folder *cd folder* to enter a folder and *cd ..* to go one folder back; you don't need to write the whole folder name, the tab key will auto-complete.

Now, navigate on the terminal to the first part of this course by running:

```
cd ~/ICTP-MuonsWithCMOS/Excercises/Part1
```

Finally, keep the cover of the camera on, and check take an image to check if it is working fine using:

```
./takeImageWithLED.sh
Usage: ./takeImageWithLED.sh <LED_duty_0_to_100> <LED_time_ms>
<shutter_time_ms> [number_images=1]

./takeImageWithLED.sh 100 1000 2000
```

This will take an image with 10 seconds of exposure and turn on the LED with full power during 1 s. For more details about the camera functioning and parameters, you can check 4.3. The script will output raw files (.jpg and .dng) and two processed files (.fits and .root) in the folder ./Date/LED/. You will only be working with the processed files. To open the fits files with ds9, you can run:

```
ds9z ./TodayDate/LED/YourFilename.fits
```

ds9 is a software widely used in astronomy that allows us to visualize fit images and perform some basic analysis. Using the mouse wheel, you can zoom in and out in the image. By hovering over the pixels you can see the value in charge of the pixel and the position of the pixel (x,y). A camera in the dark should give a mean value of about 256. Check 4.4 for technical notes on how to use ds9.

To run the data analysis on the image, you can run (watch out for inverted commas; probably copy/paste will not work):

```
root 'plotImageData.C("2024-07-01/LED/YourFile.fits.root",0.0)'
```

This will take a few seconds and open two ROOT windows. One will show the spatial distribution of the charge and the projections in the X and Y directions; to calibrate the sensor with the LED, you will need to arrange your setup to get a uniform spatial distribution. In this case, since the cover was on, we only expect to see the detector noise and baseline offset, naturally no asymmetries in the charge distribution. The panels will look like this:

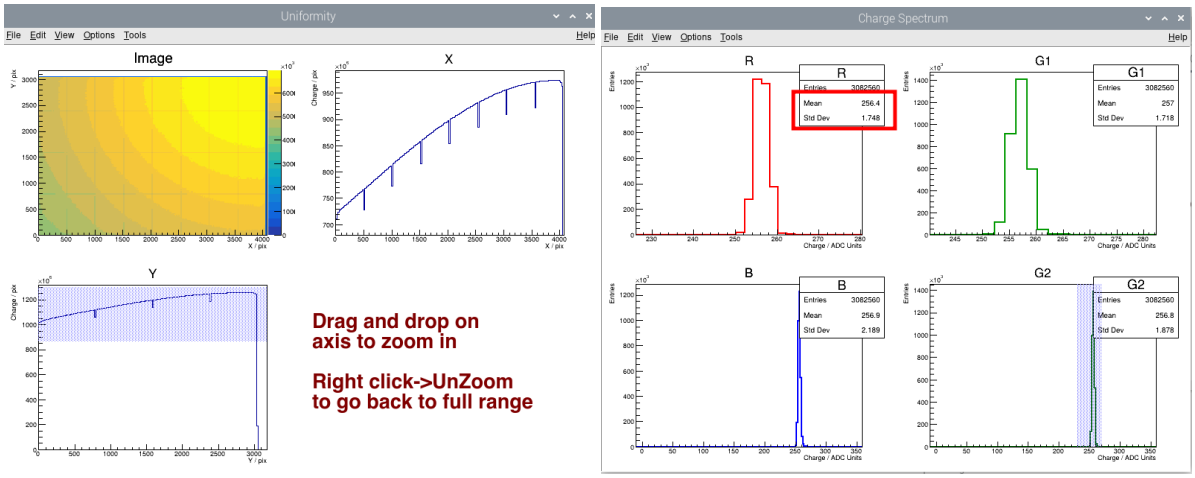


Figure 3: (Left) uniformity plots. Top-left shows the image, and the color coding indicates the signal level in each pixel. Top-right is the projection of the previous plot on the X-axis, and the Bottom-right is the same on the Y-axis. You can zoom in on the plots by dragging and dropping on the axis. For more details on how to navigate the root panels, you can check 4.5. In this case, the upper-right corner has way more light than the bottom-left; this is not good calibration data. This image was taken with an LED on, the first dark image you take, you will see a flat distribution (Right) Histogram plots for each pixel color. Each entry in this histogram is one pixel and the value in x is the signal or charge in that pixel. You will be using the label on the top-right to get the mean and std. Dev. (which is the $\sqrt{Var}$) of your data. You can zoom in on the x-axis to cut the histogram

The histogram window is divided by 4, corresponding to the four-pixel colors: Red (R), Blue (N), Green 1 (G1) and Green 2 (G2). The label on the top right indicates the mean and variance of the histogram. You can use your dark image to get the baseline level (mean), which should be close to 256. You can draw the plot subtracting the baseline by running:

```
root 'plotImageData.C("2024-07-01/LED/YourFile.fits.root",256.0)'
```

2.4 Build your setup to take images with LED

The objective now is to illuminate the camera with the LED in the most uniform way possible, which implies that in the uniformity plots, you want to see the whole image in the same color and the projections as flat as possible. Ideally, you want enough light to be far above the camera read-out noise but not too much to saturate it.

Now you can:

1. Remove the cover from the camera
2. Add some paper to cover the camera.
3. Locate the LED not too close to the camera as centered as possible.
4. cover both with a camera and LED blackout to block the environmental light.
5. takes two or three images with different LED intensity, time, and exposure. I would recommend using exposures of less than 5 seconds and LED time below 1 (you don't want to saturate the image)
6. rearrange the setup until you get uniform illumination
7. Once you get a uniform image, you can look at the histograms to get the mean and variance
8. try to get about 5 to 10 of these with different levels of illumination

In this photo-counting experiment, if the illumination on the camera is uniform, each pixel serves as one repetition of the experiment, so the histogram of the number of photons produced in the image putting all the pixels together (separated by pixel color) should follow a Poisson distribution and the mean and variance should be equal. Now, since our data is in digital units, this is a little bit different:

Var(electrons) = Mean(electrons) = Var(gain * signal) = Mean (gain * signal) = gain * Mean(signal)

if Var(gain*signal) = Std. Dev.(gain*signal)^2 $\Rightarrow$ gain^2 Var(signal) = gain * Mean(signal)

$\therefore$ Var(signal) = Mean(signal) / gain

This part is optional Ideally, you want to get the gain from many images with different levels of illumination to improve the statistics and reduce uncertainties. There is a script available: plotPTC.C, which you can open and edit with your data points.

```
double xG1[] = {1, 2, 3, 4, 5}; //Mean for Green 1
double yG1[] = {1.1, 2.2, 3.3, 4.2, 5.1}; //Var for Green 1

double xR[] = {1, 2, 3, 4, 5};  //Mean for R
double yR[] = {0.9, 2.1, 3.2, 4.1, 5.3}; //Var for R

double xG2[] = {1, 2, 3, 4, 5};  //Mean for Green 2
double yG2[] = {0.8, 2.0, 3.1, 4.0, 5.2}; //Var for Green 2

double xB[] = {1, 2, 3, 4, 5};  //Mean for Blue
double yB[] = {1.0, 1.9, 3.0, 3.9, 4.8}; //Var for Blue
```

By running

```
root plotPTC.C
```

You can plot your points with the corresponding fits to obtain the gain.

3 Part 2: particles in silicon

In this part, you will use the calibration to reconstruct particle tracks and to look for air shower muons. Muons are one of the most significant backgrounds in dark matter and neutrino experiments, so we need to know how to remove them from the data. You will use the camera with the cover on to obtain a dataset that can potentially contain muon signals. You need to decide how to run this experiment. Is it better to take fewer images with longer exposure or the other way around?

3.1 Discussion questions

- What kind of particles can we detect with silicon?
  - What kind of signal we expect to see for different particle species?
  - Can we measure neutral particles with silicon?
- X-rays, gamma-rays, infra-red and optics are all photons. Do they all interact in the same way?
- How can we measure the energy a particle deposits in the silicon target?
- Is the gain calibration enough to measure a particles energy?
- Can we use a CMOS sensor to detect dark-matter in this room?

3.2 Building the dataset

Now, navigate on the terminal to the second part of this course by running:

```
cd ~/ICTP-MuonsWithCMOS/Excercises/Part2
```

Define the operation parameters and run

```
./takeImageParticles.sh
Usage: takeImageParticles.sh <shutter_time_ms> [number_images=1]

./takeImageParticles.sh 5 1
```

This will take one image with a shutter time of 5 milliseconds. Define a shutter time and number of images to obtain about 0.5 hour of integrated exposure. Don't take more than 1000 images, or you will run out of storage, and the run will crash. Think also about the position of the camera: you want it vertical to get long traces and horizontal to maximize the number of muons hitting the camera.

The script will take the images and process them, subtract the baselines, and build a catalog of events by running a clusterization algorithm; this will put together pixels with non-zero charges into one event. When the run is finished, you will get an output like this:

```
    ==================================================
DONE!
The catalog file is .//home/lambda/ICTP-MuonsWithCMOS/Excercises/Part2
   /2024-07-01/Particles//proc/events.root
Run eventViewer .//home/lambda/ICTP-MuonsWithCMOS/Excercises/Part2
   /2024-07-01/Particles//proc/events.root
==================================================
```

3.3 Visualizing events

You can run

```
eventViewer.exe ./2024-07-01/Particles/proc/events.root
```

To open the event visualizer that looks like this

Figure 4: Event visualizer. The middle panel shows the event on the camera. The textbox on the top allows us to define selection cuts separated with &&. The left panel contains a list of all the events that you can select to display in the middle panel. The top-right panel shows the event properties, which includes the number of pixels, charge variance, and integrated signal (E0), which is proportional to the number of electrons in the event and, thus, to the energy deposited in the silicon. The display parameter text box allows you to choose which parameter is being used to build the histogram on the bottom-right panel. For example, for looking at the event energy, you want to Select E0 * gain and use the gain you obtained in the calibration.

The help in the eventviewer should show:

```
You can use any combination of the variables that are defined below. To
    combine them
for the selection cuts you should use " && ". You can also use
    transformations of the
variables such as pow(x,y), sqrt(x), etc.

EG for cuts: flag==0 && E0>1 && E0<5
EG for histogram, mean energy per pixel: E0/n0
EG for histogram, track length: sqrt( pow(xMax-xMin, 2) + pow(yMax-yMin, 2)
    )

List of available variables:
  nSat:  number of saturated pixels
  flag:  error code
```

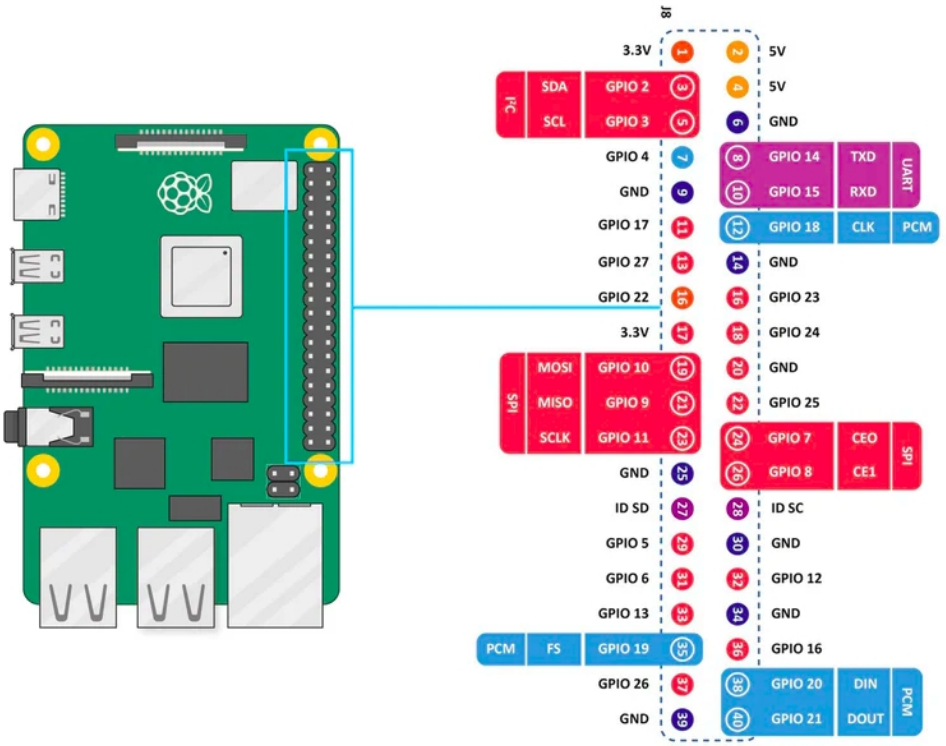
```
xMin:  x coordinate of the leftmost pixel
xMax:  x coordinate of the rightmost pixel
yMin:  y coordinate of the bottommost pixel
yMax:  y coordinate of the topmost pixel

E0:    total energy of the "level 0" pixels
n0:    number of "level 0" pixels
xBary0: x coordinate of the signal weighted barycenter using "level 0"
   pixels
yBary0: y coordinate of the signal weighted barycenter using "level 0"
   pixels
xVar0:  signal weighted variance along the x direction using "level 0"
   pixels
yVar0:  signal weighted variance along the y direction using "level 0"
   pixels
E1, n1, xBary1, yBary1, xVar1, yVar1: same as avobe but including "level
   1" pixels
E2, n2, xBary2, yBary2, xVar2, yVar2: same as avobe but including "level
   2" pixels
E3, n3, xBary3, yBary3, xVar3, yVar3: same as avobe but including "level
   3" pixels
```

Once you start the data taking, you can open another terminal and navigate to Part 2. In the folder DATA, you will find some precompiled datasets with muons and x-rays. While you build the data set, you can use these to define the selection criteria you will apply to your dataset. Try to obtain the energy spectrum of the X-rays on "events_241Am_g1_50s.root" you can try to find **here** what elements are producing the peaks you see.

4 Technical notes

4.1 Raspberry pi 5

You can find all possible information and technical details at the **Raspberry Pi website**, with a lot of super fun projects to learn microcontroller programming. In the scope of this course, you might only need the GPIO block pinout:



Watch out for the orientation of the drawing concerning the pictures in Part 1 (they are inverted). We are using for the LED one GND pin and GPIO18. You can remove the connection to GPIO18 to one of the 3V3 pings to test that the polarization of the LED is correct.

4.2 Pulse width modulated (PWM) LED

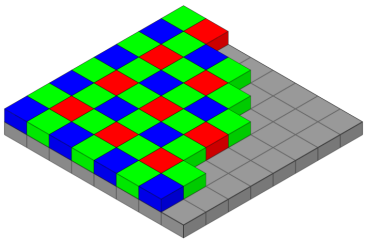
Pulse Width Modulation (PWM) is a widely used technique for controlling the intensity of LEDs with a digital signal. It works by varying the amount of time that the LED is turned on versus the time it is turned off. This on-off cycling happens so quickly that the human eye perceives it as a change in brightness rather than a flickering light. Key Concepts of PWM:

- **Duty Cycle:** The duty cycle of a PWM signal is the percentage of one period in which the signal is active (on). For example, a 50% duty cycle means the LED is on half the time and off half the time. A 100% duty cycle means the LED is always on, while a 0% duty cycle means the LED is always off.
- **Frequency:** The frequency of the PWM signal is how fast the PWM cycle repeats per second. A higher frequency results in smoother perceived changes in brightness, while a lower frequency might result in visible flickering.

4.3 IMX477 and and the libcamera-raw command

The IMX477 is a high-quality camera module developed by Sony and designed for use with the Raspberry Pi single-board computers. It is commonly known as the ”High-Quality Camera” and offers superior imaging performance compared to earlier Raspberry Pi camera modules. The IMX477 features a larger sensor size, higher resolution, and improved low-light performance, making it ideal for a range of applications including photography, videography, scientific imaging, and more. It has 12.3 megapixels (4056 x 3040 pixels) and a total sensor size of 7.9 mm (diagonal). It is one of the few off-the-shell CMOS that allows us to obtain raw data.

To build colored images the CMOS uses a color filter array, which means that each of the pixels has a color filter right on top of it as in:



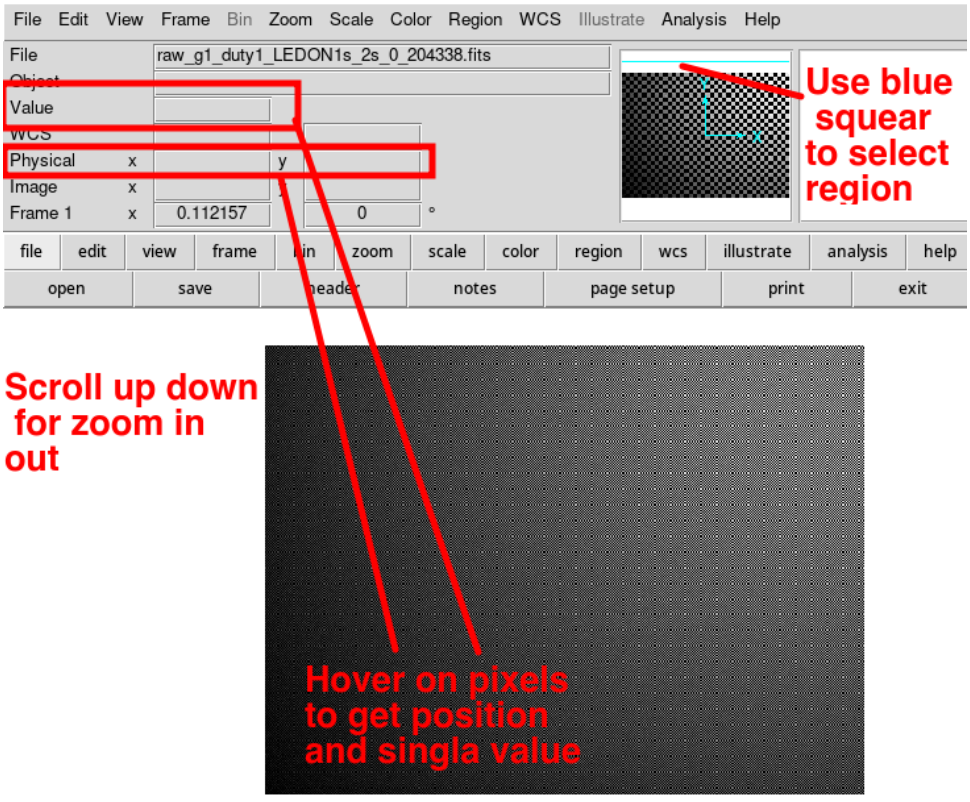
For using the calibration technique described in this guide, we need to separate the analysis per pixel color, since each filter will allow different photons in. This will produce a non-Poissonian fluctuation that breaks the analysis.

The libcamera-raw is the command that allows us to take raw data, meaning that the image is not compressed and processed inside the chip to remove background noise, which is what we want to see. The most important parameter for us is the shutter time, but by running libcamera-raw only you will access all the documentation with information on the different parameters.

4.4 ds9

DS9, also known as SAOImage ds9, is an astronomical imaging and data visualization application. It is widely used in the field of astrophysics for analyzing and visualizing data from telescopes. It is quite a powerful analysis tool but not very intuitive.

You can hover over the image with the mouse to obtain information on the pixels and use the mouse wheel to zoom in and out:



You can also use this tool to build charge histograms, however this will group all pixels in one. For this, you need to create regions (e.g., circles, rectangles, ellipses) by selecting Region $>$ Shape from the menu bar and drawing them on the image. And then to use the regions to extract and analyze data you can right-click on a region to access options like Get Information, Plot, or Statistics.

You can also check the help and documentation by accessing the built-in help by selecting Help from the menu bar. The official DS9 User Manual provides comprehensive information on all features.

4.5 ROOT Analysis framework

ROOT is an analysis framework widely used in high-energy physics well beyond the LHC. It has thousands of libraries and utilities to do pretty much anything.

In the scope of this course, there are some other things to do with the ROOT panels, such as changing the binning of the histograms and fitting. To access the histogram options you need to Right-click on the histogram displayed on the canvas, and from the context menu, select Set Binning. In the Set Binning window, enter the desired number of bins and the range.

For fitting data, you can use the Graphical Interface by right-clicking on the histogram again and selecting the fit panel from the context menu to open the fitting options. In the Fit Panel, select the desired fit function from the list (e.g., Gaussian, polynomial). Adjust the fit range if needed by specifying the From and To values. And optionally, click on the Parameters tab to set initial values and limits for the fit parameters. Finally, click fit to perform the fit. The fit results will be displayed on the canvas, and the fit parameters will be shown in the Fit Panel.

You can check the **ROOT's User Manual** for A LOT of information and example codes to learn how to use ROOT.