SLAC/INFN Summer Exchange Program

SLAC Summer work
Summer 2017

Development of an edge finding algorithm optimized for LSST I&T Sensor Integration

Author:
Giorgio Dho

Supervisor:
Scott Newbry

Abstract

During the installation of the LSST camera CCD sensors some precise measurements will have to be performed to avoid collisionbetween adjacent rafts. During the operation it will be necessary to measure width of gaps between CCDs down to 125 μm with a precision of 25 μm. To accomplish the requirements, software was developed to analyse images using C++ and Python scripts to find the edges of the sensors and measure distance between them. The software, along with a telecentric lens and camera, has been tested using a prototype target, as well as real CCDs mounted into a raft that will be installed, getting promising results.

Introduction

The Large Synoptic Survey Telescope (LSST) is a science project which has the goal to conduct a 10 years survey of the night sky delivering an expected amount of data close to 200 petabytes in order to better understand some of the mysteries of the structure of the cosmos, such as dark energy and dark matter [1].

One of the most important components of the telescope is the camera. Its heart, the focal plane, will be made of 21 modules, called rafts, and will be inside the camera cryostat. Each raft, represented in figure 1, is composed of 9 CCDs which form a square and by all the electronics that is directly connected to the CCD sensors. Each CCD is a detector of 16 Megapixels.

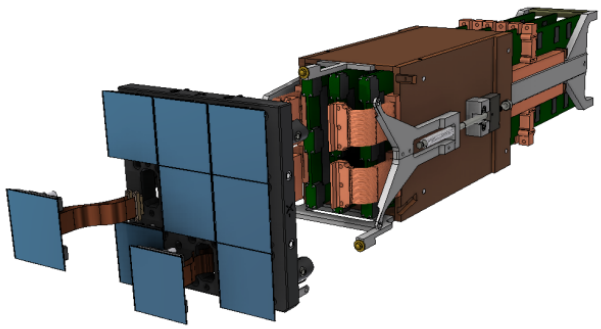

Figure 1: *Schematic drawing of a single raft of the LSST camera*

All the 21 rafts forming the Science sensor will have to be placed one next to the other to form the focal plane as shown in figure 2.

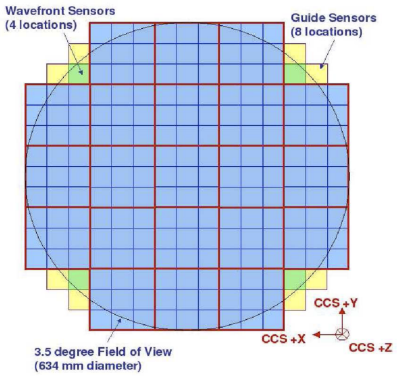

Figure 2: *Schematic representation of the focal plane of the LSST camera*

At the moment of building, which will happen here at SLAC, all the rafts will have to be put in position inside the cryostat in the slots prepared for them. This operation will be done with a system of gantries that will take the rafts one after the other and pull them up to accurately put them in position and lock them there. The rafts will have the side with the CCDs facing down. During these operations it is fundamental to make sure the rafts do not collide each other while being installed. This is due to the fact that CCDs are very fragile and a crash is very likely to break them. A crash would cause causing a loss of hundreds of thousands of dollars, since they are very expensive, and months of time, possibly delaying the completion of the camera. So it is very important tmake sure the rafts do not collide.
To be able to have a fill factor of the focal plane (percentage of active material on the total space covered by the focal plane) as high as possible, the rafts must be placed close each other. The expected distance between them is 500 μm. To be sure to be able to move the rafts without putting them in danger it was decided to be mandatory to measure distances down to 125 μm with an uncertainty of 25 μm. This makes impossible to guide the installation by looking at the rafts with naked eye or simply using a camera to record video in real time. Another complication comes from the fact that, during the operation, the raft that is being pulled up will be closer to the ground than the ones already in position in the cryostat. In particular, imagining to look at the rafts with a camera and have both of them in focus at the same time a depth of field of least 38 mm would be required. Figure 3 shows the problem with images.

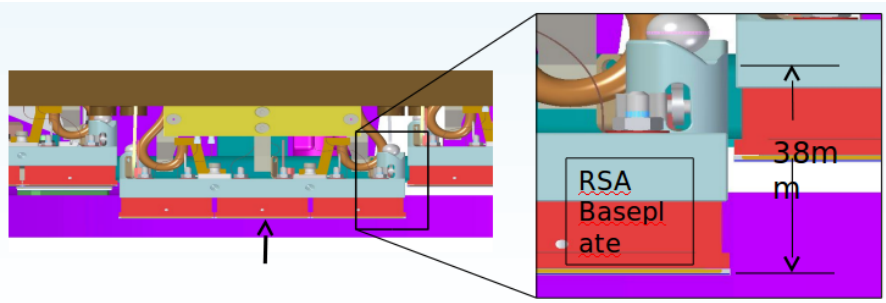

Figure 3: *Schematic representation of raft being positioned close to other two. The spaces left between adjacent CCDs are very small (close to 500 μm)*

Since there was not any commercial item that had the characteristics to accomplish the task, the group had to come up with a system to solve this problem. They thought to use four digital cameras equipped with telecentric lenses, each one staring at a corner of the raft to be installed, to be able to see where it is located compared to the others already mounted. A telecentric lens is a lens with a light stopper positioned behind the glass of the lens, but before the camera. The stopper blocks off-axis light and permits collimated ight to reach the camera. As a result, in a specific region of space the objects framed by the camera are magnified the same way, independently from the distance from camera. This feature will allow to see the rafts in different distances from the camera, magnified the same way, to be immediately able to recognize if the raft is going to collide against another one or not.

Despite being able to build this system, it is still necessary to create software to read the images from the cameras and translate them into information to guide the process of installation. The pictures taken by the camera will be cropped to reduce the size and to focus only on the region where the edges of the rafts are located. After that a program will use the cropped image to give back information about the location of the edges and the distance between the rafts involved in the picture.
My work focused on this last part, of developing an algorithm to return information of this kind from the cropped images, having the possibility to use a cleanroom to take pictures of some targets.

Work with plastic blocks

At the beginning, the equipment in the clean room at disposal for me and my supervisor, which are shown in figure 4, were:

- A gray scale camera RT-mvBF3-2051aG
- A colour camera EO-10012C
- Telecentric lenses TC23064
- Plastic black blocks, to simulate CCDs

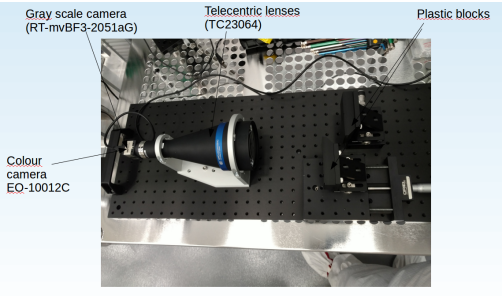

Figure 4: *Picture of the table we worked on*

It was possible to precisely move the blocks in the x and y directions and measure the displacement with the use of a micrometer. Some images were taken and cropped down to 128x128 pixels with the gap between the two blocks in them. I used GIMP editor to crop the images. It allowed me to decide exactly the dimensions of the cropped image and pixel position where to start the resizing. The figure 5 shows the result

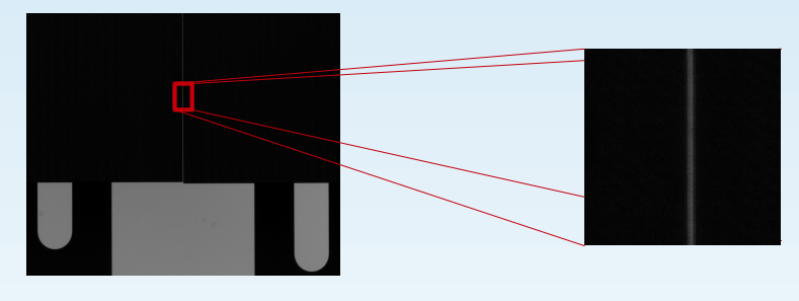

Figure 5: *Result of the process of cropping.*

C++ macros for Root were used and after completing the work a Python3 translation was produced, using Root libraries, scipy and numpy. Using these languages, the image could be easily read by some Root functions and be translated into a matrix of numbers containing RGB values (intensity of the light captured by the camera) for each pixel of the image. Looking at a row of pixels in the direction orthogonal to the edgeline on the picture, a plot like the one in figure 6 could be displayed:

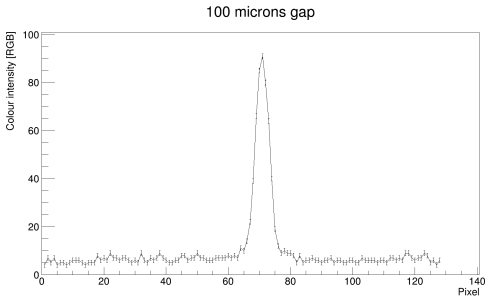

Figure 6: *Example of a row of pixels in a 100 μm gap image. The peak corresponds to the brightest part of the image, meaning the background light*

By varying the width of the gap it was immediately clear that the shape of the data from the rows of pixels changed drastically. Two different cases were highlighted: plateau or not plateau. The latter was the case when the gap looked like a peak, as in figure 6, the former when it seemed there was a plateau instead of a single point on the top, like in figure 7.

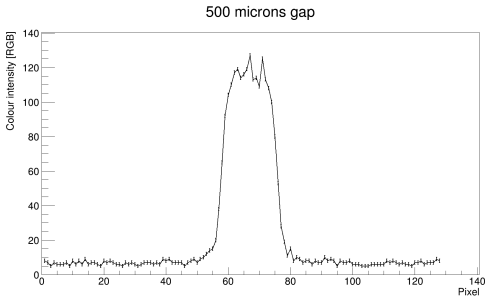

Figure 7: *Example of a row of pixels in a 500 μm gap image. The plateau is visible.*

Therefore it was very important for the software to understand what kind of shape the data had. The algorithm to sort this out behaves as it follows: first, it looks for the maximum y value, where the peak or the plateau should be located; to avoid picking a noisy pixel instead of the the maximum wanted, the program checks the two pixels next to the candidate to become the new maximum and if at least one of the two is too much lower in RGB value, then the conclusion is that it is noise and the program continues looking for another max. Once the maximum is found it counts the number of points with similar values of RGB and if they exceed 8 a plateau is considered to be discovered, because there are many points close to the highest value of RGB found. The number 8, as well as the definition of close to the top or too much lower are all arbitrary at this point. They depend deeply on the brightness of the image and so on RGB values reached. All these things should be calibrated during the last steps of preparation, when the final setup for the integration will be ready.

Fit function for the edge

To find out where the edges of the blocks are located from the rows of data, it was important to understand how the transition from a black region to a bright region happens. For the diffraction of light when looking close to the edge, the passage from dark to bright is gradual. When hitting the edge if the light diffracts equally on both directions, then the edge should be located close to the middle of this transition region. As a consequence it was needed a function with a shape similar to an S (starting from a plateau and rising or decerasing to another one) that had a parameter directly related

to the position of the middle, our edge position. This way it could have been possible to minimize the propagation of errors to get to the information desired. Two functions were mainly tried: the integral of a gaussian and an S-function. Results on real data gave consistent results and the two had the same number of parameters, but the S-function had parameters that could be handled more easily. Therefore this function was chosen:

$$f(x) = a + \frac{b}{1 + (\frac{c}{x})^d}$$

where:

- a is the vertical position the lower plateau;
- b is the amplitude, the difference in y between the higher and the lower plateau;
- c is the position half way from one plain to the other is. Here is where the edge is thought to be located;
- d is a shape factor that descibes the steepness of the curve.

A representation of it is on the following figure.

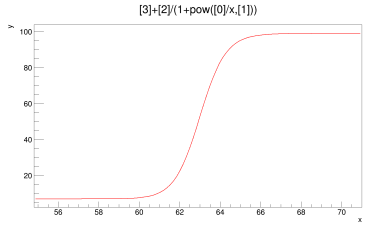

Figure 8: *S-function.*

Algorithm

As already explained a way was implemented to discriminate whether the data took the shape of a plateau or not and the function to use to fit the data in the transition region. This last piece is found by searching for the location where the data points rise from the black baseline. This one is calculated from the first points averaging their value and calculating their standard deviation

($\sigma_{baseline}$). The beginning of the transition region is defined as the moment when three consecutive points are out of the interval $baseline + k \cdot \sigma_{baseline}$. The value of k has to be calibrated, as pointed out previously. The end of the transition region is found differently for the two cases. If there is a plateau the program looks for the three consecutive points that get back inside the interval delimited by the value of the previous point plus or minus $2k \cdot \sigma_{baseline}$. Here 2 times k is used because with more light the noise increases as well, leading to more fluctuations. If instead, there is no plateau, then the region stops at the point where the maximum is, which was already found to determine the existence of the plateau. The use of three consecutive points aims to avoid mistaking noise for the transition region. When three points are going upward in value, this is the point the program must look for.
Repeating the same kind of commands but on reverse, the fit region for the second edge can be found. The results are shown in figure 9

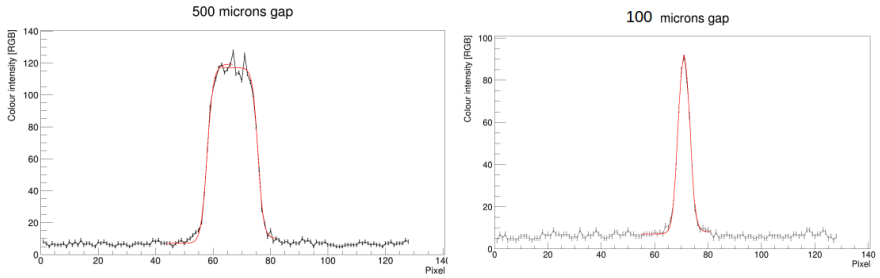

Figure 9: *On the left the result of the fits for a plateau condition while on the right the same with the other possibility.*

These operations are iterated for each row of pixels, to save for each row the position of the edges of the blocks from the parameter c of the function. However, it is possible that a particular annoying noise or the shape of the points too far from the one of the function prevent the algorithm of minimization used by Root or the algorithm that finds the region of transition from working. In these scenarios the values returned for the edge positions, if any, are expected to be wrong. Therefore a series of controls on the proper fulfillment of the fit and of the results were conceived. In particular it is considered a good fit one whose χ^2 divided by the number of degrees of freedom is less than 13. Usually for this purpose 3 should be a correct value, but it is clear that the S-function is not the function the points follow in the first

place, so the threshold value was kept higher.
The operation can be done with each single row of pixels or using an average of several rows.

At the end, to approximate the location of the edge in each point in the space, a linear fit is performed with the points gotten from the fits. When the linear fits are done for both edges, the minimum distance between the two and their relative angle can be inferred. The uncertainty for these measurements comes from the propagation of the errors of the parameters.

Results

Camera choice

As described, two cameras were available to take pictures with. The colour one has a better resolution because of the size of each pixel, 1.67 μm, while the BF3 has 3.45 μm pixels. Nevertheless the BF3 gave better outcomes. In fact the data from the colour one were much more noisy which made the fit algorithm to fail more often or to give values much less precise. The figure 12 shows the difference of the same conditions framed with the two different cameras. The reduction in the dimension of the pixels was overcome by severe noise. This is likely due to the kind of filters applied over the CMOS detectors of the camera to make it able to read colours. The filters are often disposed as a chessboard over the pixels, with different colours next to one another. When using this camera looking at pixel after pixel, despite averaging the values from each colour, it is visible when one of the base colour (red, blue or green) matches the colour of the filter, resulting frequently in spikes as visible in the figure.

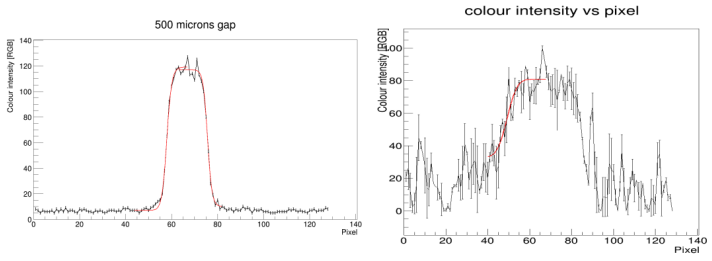

Figure 10: *On the left the result using BF3 camera while on the right the same with the colour one.*

Hence the camera that will be used is the BF3. In particular for this camera, knowing the dimension of the pixels and the magnification of the lenses used (0.138x), it is posible to calculate the real dimension of a pixel on an image: it corresponds to 25 μm.
The result for a single edgeline is exemplified in the following figure.

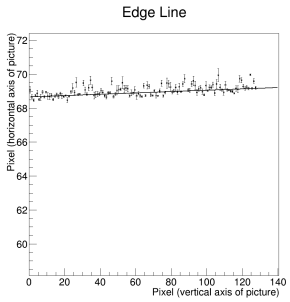

Figure 11: *Best linear fit for an example of an edgeline.*

The χ^2 divided by the number of degrees of freedom is hardly ever below 10. Nevertheless the points seem to reside up and down the line, like oscillating, so it seems there is not any systematic effect that draws the line away from where the real edge is. With this camera the result of the program analysing an image with 500 μm gap was very accurate, as the figure shows.

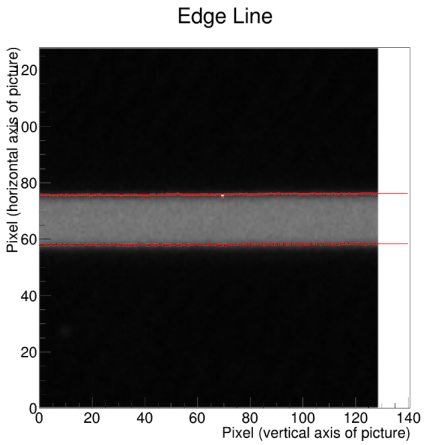

Figure 12: *The image and the result of the application of the algorithm are shown superimposed.*

Repetition

One of the first thing which was tested, was the ability of the system to give the same results when taking images in different moments of the same setup. The answer turned out to be negative. Little oscillations, trembling and camera shot noise made the outcome of each image, cropped in the same position, always slightly different (couple of microns). It became clear that it was compulsory to take many images and average the results to have robust information. Acting this way it was possible to have a distribution of distances and angles to get the final answer from. The most important aspect of this test is the fact that is possible to associate to a measurement a consistent uncertainty. The mean uncertainty for the angle turned out to be around 0.03 degrees, while for the distance it was less than a micron. However for the purpose of the installation, the uncertainty given to the width of the gap should be an indication of the distribution of these quantities, leading to the idea of associating the standard deviation of the distribution as the final uncertainty. It was possible to perform only one test with 9 images, so the errors should actually be 0.09 degrees and 2 μm. It is expected these values to change but it is very unlikely that the uncertainty on the distance shall pass 5 μm.

Minimum distance

An important feature of the algorithm is the minimum distance that it is able to resolve. For this to be tested, pictures were taken with different widths of the gap between the blocks. It was not known the real distance between them but the relative increase in width was measured.
When the two sides of the blocks come too close one another, since it is expected that the diffraction works the same way, the functions representing the transition phase from black to bright overlap, preventing the transition region from being seen completely. So the position of the edges are expected to be closer to the peak in the picture instead of in the middle of the transition region. To take into account this effect, the background level of RGB values reached with wider gaps was studied. This way the parameter of the amplitude was forced to stay close to the value gotten from the background study. Using this trick, the minimum distance measured by the algorithm that was still consistent in relative decrease of width of the gap with the real measurements was 56 μm. It was not possible to perform multiple measurements so the absolute value is more or less an indication of what it should really be. In spite of this it is confident enough to say that the algorithm can measure reliably down to 60 μm which is less than the minimum value required. The figure 13 represents a row of pixels of the lowest gap measured with success.

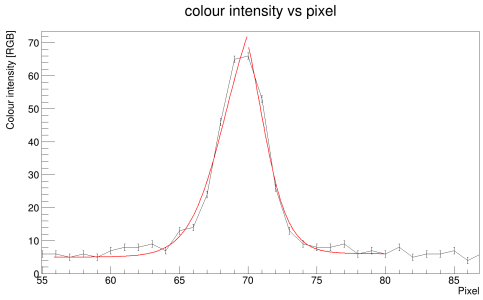

Figure 13: *Since the gap is small it is possible to see that the fit functions would have the higher plateau at bigger RGB values, This is the result of the force bond given to the amplitude.*

Light effect

Another important characteritic of the setup is how the light is shone on the target, and how it affects the recording of the image. Four pictures were taken with a fixed gap: two illuminating the blocks from the front, but placing the light source in two different positions, and two illuminating from the rear in the same way the two on the front were settled.
For the ones illuminated from the front, the direct light highlighted surface texture of the prototype blocks pattern that reflected too much localized light so that it was almost impossible from the data to recognize where the gap was.
For the two illuminated from the rear, some data are shown in figure 14

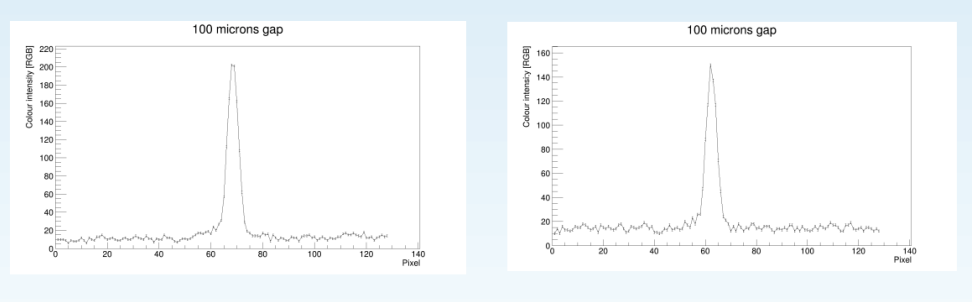

Figure 14: *The same row of the two pictures taken with different light directions.*

Whilst the width of the gap did not change the position of the edgelines was moved by 150 μm. This difference could not be explained by vibration or oscillations, so it had to do with the inclination of the light hitting the blocks. This suggested that the most important variable to control in the final systemis probably the illumination.

CCDs

In the mid of my permanence a real raft with the kind of detectors that will eventually be installed arrived. LSST will have two types of CCDs on the raft: some with a golden layer around the silicon and some without. For what concerns this work, the ones without the layer should behave as the black blocks, so the skeleton of the code ought to be the same. The ones with the golden edge pose some difficulties instead. In fact the edge that has to be found is the one between the object and the empty space and in this case it corresponds to the one between the gold and the background. Gold is very reflective, making it even brighter than the backgound, as figure 15 shows.

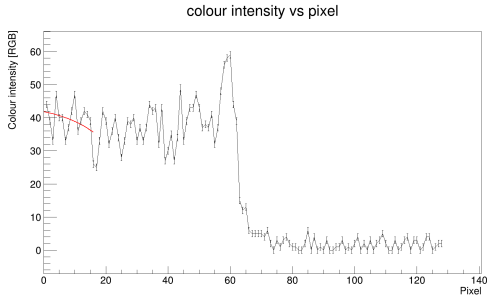

Figure 15: *The part with low values of RGB corresponds to the black anti-reflection coated surface of the CCD. On the left there is the noisy background light from empty space behind the CCD. Between the two is a spike which corresponds to the golden layer.*

Algorithm

This algorithm has been only written in Python3 scripts. Because the raft was inside a cryostat for testing, the pictures were taken looking through

a glass which increased the reflected light adding more noise to the data. Moreover the edge was now to be found in a much more illuminated region making the presence of noise even worse. To smooth the data, cleaning it a little from the noise an open source Python library was used[2]. From that library a gaussian blur was applied to the picture prior the translation to the form of a matrix of numbers. The effect of this blur can be seen in figure 16.

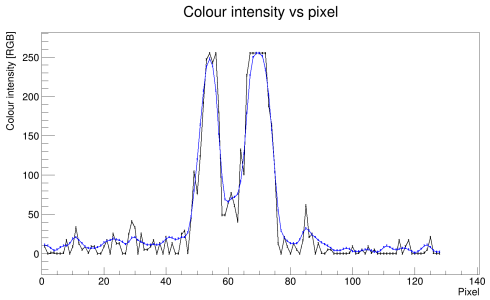

Figure 16: *The black point are the original data, while the blue are the ones after blurring.*

After this, the program looks for the max value in RGB, which should be one of the two peaks of the golden layers. To be sure it is the kind of maximum needed, similar to what was described in the search for the plateau, the number of points with values close to the maximum candidate are counted and if they are more than 4 (we are not looking for a plateau, just a thick spike), then the point is accepted. Afterwards it looks for the second maximum in a range that excludes all the points too close to the first: this way if the second golden spike is much lower than the first it will be found anyway, instead of picking a point on the transition region of the first.
Once the two maxima are found, they define an interval in which the program searches the minimum, which should be undoubtely close to the middle between the two golden layers. From that point it goes backward to find where to make the first fit start analysing the slope, calculated as difference of the RGB values of two consecutive pixels. As soon as the increase in slope is less than 30% than the one calculated with the previous points, it stops. This because it means that the search is arriving to the top where the slope is zero. An extra point can be added to increase the number of degrees of freedom if its RGB value is very similar to the one considered to be the beginning of the region for the first fit.

The procedure is repeated similarly to find the end of the region for the fit for the second edge. Then the fits are performed with the same function. If the fit has badly terminated or is too unprecise (χ^2 too high), the region for the fit is shrunk by one point and retried. If it gets worse or not better enough (χ^2 again) then it is discarded. A result is in figure 17.

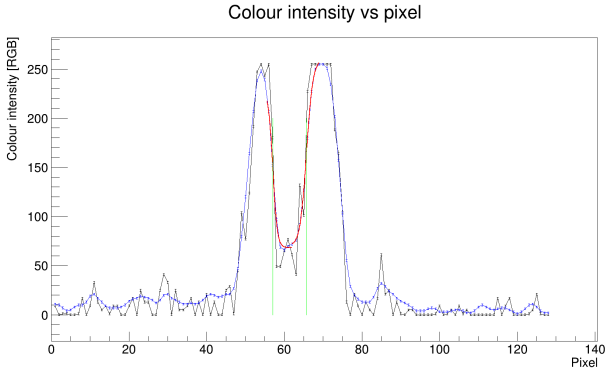

Figure 17: *The black point are the original data, while the blue are the ones after blurring. The red lines are the functions fitting the transition and the green lines are the location of the edges*

.

After this the same linear fit of the edge location for each row is performed for each side. Then the points that are too far from the best fit line are erased and the fit is done agian. The outcome is shown in the figure 18

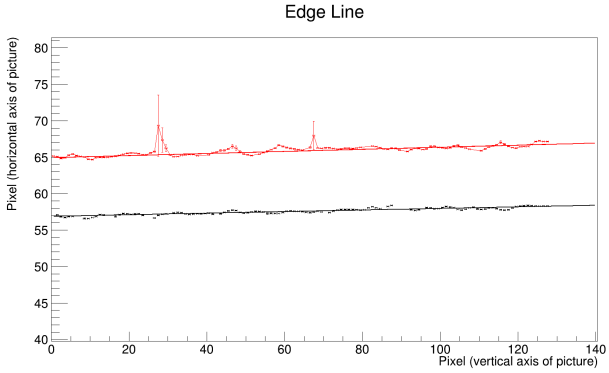

Figure 18: *The black point are the original data, while the blue are the ones after blurring. The red lines are the functions fitting the transition and the green lines are the location of the edges*

The program in the end calculates the minimum distance, repeats all the steps for the number of images available, and give the averaged result in a txt file.

Results

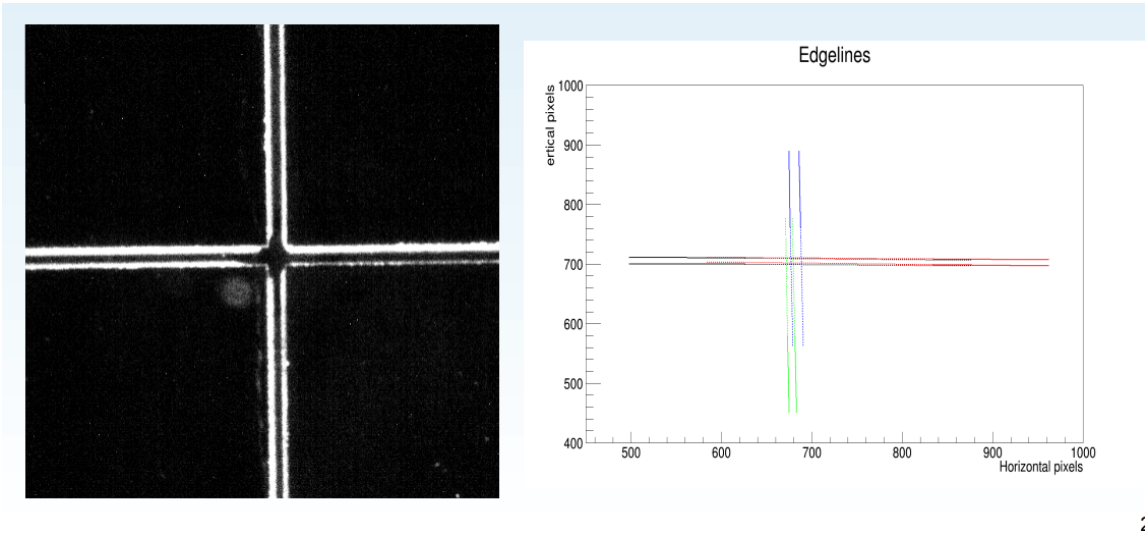

Figure 19

The previous figure shows on the left a portion of one of the sixteen photographs that were taken of the raft centered at the corners of four different CCDs of the same raft. From each of these pictures four smaller were extracted, each one to analyse the edges in every direction from the centre. On the right there is the result of the application of the algorithm to these smaller images. The full lines are the average of the sixteen best fits of each edgeline, while the dashed are their prolongation towards the centre.
It is clear that the lines which were expected to be parallel are not and moreover the widths of the gaps vary too, going from 205 to 280 μm. At a first glance this seems to be the result of an unreliable analysis. The expected gap was measured in the past months and should have been around 250 μm. However when zooming the original image it can be observed that the edgelines are not parallel and the gaps differ too. This means that the algorithm worked properly, translating truthfully the images into information corresponding to what the camera captured.

The uncertaities on the distance obtained by averaging the images are around 4 μm, using the standard deviation of the distribution.

A major concern is the control of the illumination. As shown before, differences in how the light is shone on the target can severely affect the position of the edgelines. In these photographs only the diffuse room light was used. To be sure that the edgelines are not displaced by light effects it will be fundamental to perfectly control the illumination. Most likely the best option would be to make it the most homogeneous possible, at least around the region of interest.
To make the linear fits more robust, another improvement would be to increase the dimension of the cropped image, enhancing the number of points to be used to perform the fit.

Conclusion

During these two months an algorithm to find edges was developed. It evolved a lot eluding more and more problems that arose. It is capable of working with black blocks that do not have special edges and with CCDs with the golden edge. The results seem to be promising especially looking at the constraints needed to be used. The 125 μm minimum gap and 25 μm error are completely respected in the case of the black blocks, even though they will be only used as developing tools. Since there were not more rafts it was not possible to test the minimum distance that could be resolved with real CCDs, but the work with them seems promising.
By making accurate measurements, this software will greatly reduce the risk of a CCD collision that would cause costly damage and delays

Acknolegment

I would like to thank all the LSST group who made me feel part of the team and were very kind and accomodating. In particular I thank Scott Newbry and Kevin Reil for their patience and support they gave me. I certainly acquired a lot of experience and skills in programming and in lab enviroment and all of this was thanks to the SLAC/INFN Exchange Program.

Bibliography

[1] "LSST Camera Conceptual Design Report CD-1 Internal", LSST collaboration 2006. URL: LSST.org

[2] openCV library, openCV.org