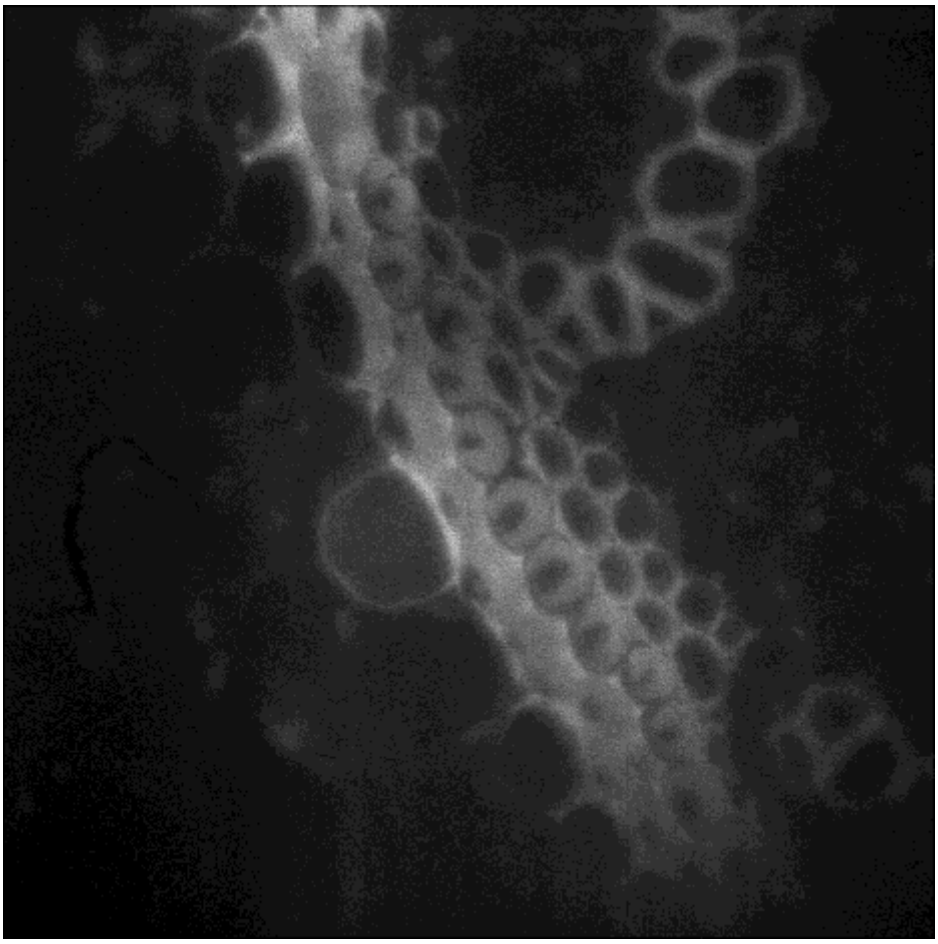
Hanze
University of Applied Sciences
Groningen

Single Photon Imaging with ImageJ



Lonneke Scheffer & Wout van Helvoirt
Bioinformatics | BFV3
Date: June 22 2016
Involved: Martijn Herber

Hanze
University of Applied Sciences
Groningen

Single Photon Imaging with ImageJ

Single Photon Imaging with ImageJ

Lonneke Scheffer & Wout van Helvoirt
Bioinformatics | BFV3
Hanze University of Applied Sciences Groningen
Institute of Life Sciences and Technology
Involved: Martijn Herber
Date: June 22 2016

Summary

A lot of scientific research involves studying objects that are so small that they cannot be perceived by the human eye. Microscope cameras are a very popular way to study this, but unfortunately the images created by such cameras often contain lots of noise. Single Photon Imaging is a technique that measures the positions of single photons and then combines those to create an output image with a lot less noise and more detail. The image intensifier is a device that can be used to record the single photon data, but computer software is needed to create the final images.

ImageJ is an image processing program that is often used to process scientific image data. A plugin has been made for ImageJ to process the recorded single photon data and create high resolution output images. The main goal was to create a plugin that could be used to create output images of a better quality than images created by a regular microscope camera. Other side goals were to make the plugin faster by parallelizing it on a computer cluster, and create an option that could be used to improve the image if not enough input data was given.

The created plugin has four options; Recursive TIFF opener, Photon Image Processor, Image Thresholder and Image Reconstructor. The Photon Image Processor has also been parallelized using Hadoop. The options have been tested with different settings, and with a dataset containing 70,000 input images. The output has been compared to a photo taken by a regular microscope camera.

The final output image created with 70,000 input images was indeed more clear and less noisy than the microscope photo. The quality of the output images increases when using more input images, so if more input images were to be recorded, the output would have been even better. The parallelization in Hadoop was unfortunately not faster when using the same input dataset, although it was a lot faster when using larger images. When not a lot if input images are available, the created Image Reconstructor does seem to improve the quality of the image, and the Image Reconstructor works even better if the noise is first removed using the Image Thresholder.

In conclusion, the created plugin has proven to successfully convert single photon data recorded using an image intensifier to a high resolution output image, that is of a better quality than a photo taken by a microscope camera.

Table of Contents

1. Introduction

Microscopes and cameras are widely used to study small structures like for instance cells. Unfortunately, it is not possible to enlarge these images endlessly, because then the images become blurry. To gain more details in the image you would need a higher resolution too. But lenses have limits and modern cameras can introduce electronic noise. Single photon imaging is a technique that gets around this, because the camera-made image is not enlarged, but single photon events are detected and combined to generate a more detailed picture[1].

Lambert Instruments B.V. is a company that develops, produces and sells cameras and attachments. One of their product is the image intensifier, which can be used to detect single photon events. An image intensifier is a device that improves the light sensitivity of a camera. For every photon that enters the image intensifier, a group of photons is released towards the camera. A camera can usually not detect a single photon, but it can detect a group of photons. The camera is used to generate a series of images containing light spots at the positions where photons would have entered the image intensifier. From these images, a computer program could calculate the exact positions of the photons and combine them in one single output picture. This way almost all camera noise is lost, and it is even possible to create an output image with sub-pixel resolution, because the calculated photon coordinates can be in between pixels.

ImageJ is widely used for scientific imaging, because there are many plugins available for varying purposes. Anyone can create their own plugin for ImageJ, which can then easily be distributed and used by others, like for instance the owners of an image intensifier. Because of this, it would be convenient to have an ImageJ plugin to easily convert the series of created images to a single high resolution output image. Is it possible to create an ImageJ plugin that can convert the data from a camera with an image intensifier to a more detailed picture than a regular microscope camera would be able make? How does it perform? Can the plugin be parallelized on a computer cluster, and does this reduce the overall execution time? And what can be done if there were not enough input images recorded?

2. Materials and Methods

The created plugin consists of multiple parts. Each of these parts is represented as an option in the ImageJ option menu, and can be used separately or in a sequence. There are four options available: Recursive TIFF opener, Photon Image Processor, Image Thresholder and Image Reconstructor. The most important option, Photon Image Processor, has also been parallelized using Hadoop's MapReduce framework, in order to make this option faster. In this section, a description will be given of the different options and their settings, and what experiments have been used to test their quality.

2.1 Recursive TIFF opener

Because the input data often consisted of TIFF files stored in multiple folders, a recursive TIFF file opener has been made. It will search through the underlying directories of a user selected directory to open all the TIFF files as a virtual image stack. Access time of each image file within a virtual stack will be slower, but the image files won't be read into the Random Access Memory at once, like with a normal stack, saving computational memory space. This is often (if not always) necessary since the data sets are too large to load into the Random Access Memory at once.

2.2 Photon Image Processor

The Photon Image Processor option can be used to convert a stack of single photon data (images containing single photon events), and create one high resolution output image. The algorithm searches for the light spots in the images and combines these coordinates in a matrix. The input data can come from different cameras with different settings. For instance, the background color can be very black and even, but it could also be dark gray or contain a lot of (electric) noise. The brightness of the single photon events differs per dataset, and some old cameras might introduce hot pixels. Also, if there is a lot of data available, you might want to be a bit more 'strict' and only accept the very clear and bright light blobs. Whereas if you have very little data, you want to use all the data you have got and do not mind it if a few 'fake' photons slip through.

2.2.1 The algorithm

Because the datasets can be so different, there are several settings that can be adjusted by the user in order to make the most out of their dataset. The plugin can optionally perform pre-processing in order to reduce the noise in the input data. This is advised for images with a lot of noise or hot pixels, since they can be mistaken for single photon events. The 'strictness' of the plugin can be altered by changing the noise tolerance. With a higher noise tolerance, less light points are recognized as photon events.

There are also three processing 'modes' to choose from: simple, accurate and sub-pixel resolution. These methods differ in the way that they choose their exact coordinates for the photons, which is shown in figure 1. Figure 1A shows a part of the raw data that contains a light blob. The simple mode

uses a naive method to gather the positions of the photons. The lightest points with a specific tolerance are used as the photon coordinates (Figure 1C). The accurate method is based on this, but then takes an extra step and compares the found pixel to the pixels around it (Figure 1B). If there are for instance only very dark pixels on the left side of the found pixel, and very light pixels on the right side, then the found midpoint is moved to the right (figure 1D). The sub-pixel resolution mode uses the same method as the accurate mode, but this time the pixels are divided horizontally and vertically, so the found pixel can be one of 4 sub-pixels (figure 1E). This way an output image is created that contains four times as many pixels, but you would also need four times as many photon events to obtain the same amount of saturation.

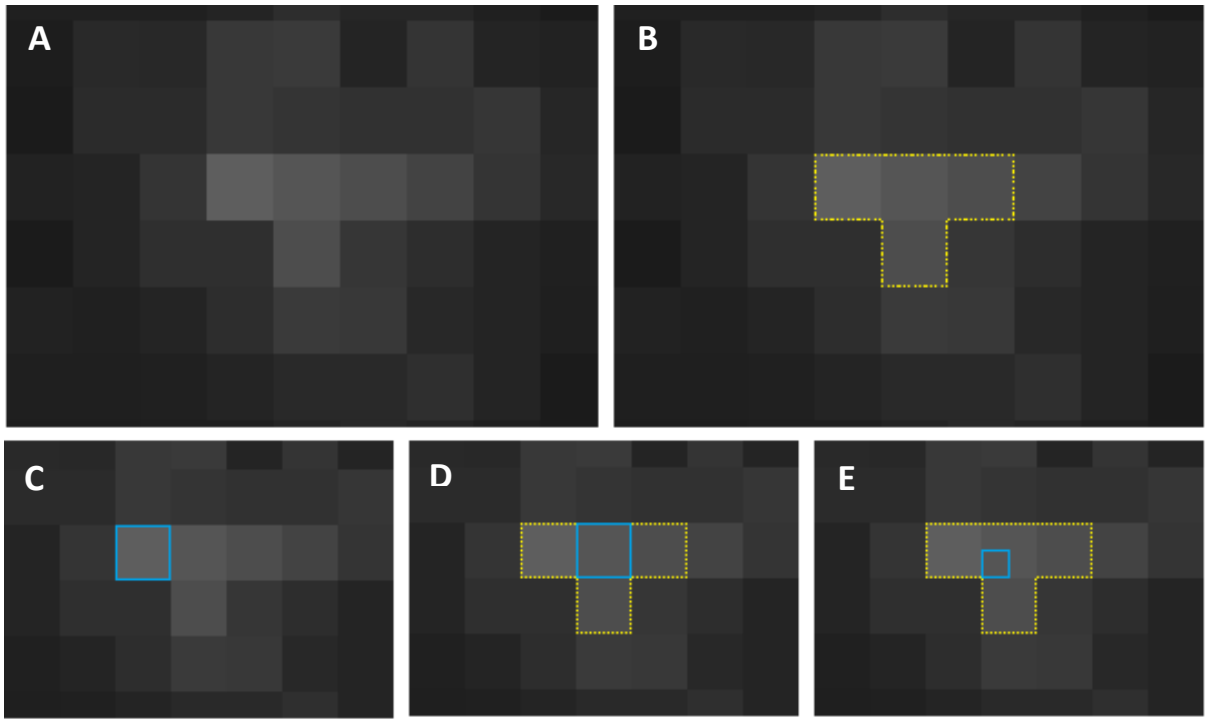


Figure 1: Different methods detect different coordinates. A shows the raw data; B shows the detected photon event outline; C uses Simple mode to detect the lightest pixel; D uses Accurate mode which calculates the midpoint with an outline; E uses sub-pixel resolution and finds partial pixels.

2.2.2 Parallelization with Hadoop

Each image from our single photon event data can be processed independently from the other images. This makes the Photon Image Processor an excellent target for parallelization. ImageJ has built in ways to parallelize the plugin on the cores of the used computer, but using Hadoop's MapReduce framework, the plugin could also be parallelized on a computer cluster. With this framework, each image from the dataset could be passed on to an individual Mapper. These Mappers are able to run simultaneously on a cluster of processing cores. The photon coordinates are gathered afterwards and combined by the Reducer. By using Hadoop, the workload gets divided evenly across many processing cores and this should lower the time needed.
In addition to the standard Photon Image Processor settings, the user must also define the width and height of the output image that will be created.

2.2.3 Photon Image Processor experiments

Several experiments have been done to test the performance of the Photon Image Processor. Default settings can be found in table 1, these settings are used unless stated otherwise.

To compare the results of using different amount of input images to the output of a regular camera, the Photon Image Processor has been tested on 17,500, 35,000 and 70,000 images of the same dataset. Because this dataset contained relatively little noise, no pre-processing has been has been done and the noise tolerance has been set to 5. The image made by the regular camera was an average image consisting of 10 images. This experiment has been done in simple mode, and parallelization has been turned off to measure the elapsed time per 1000 images.

In order to compare the three different modes of the Photon Image Processor to one another, the same dataset of 70,000 images has also been tested with all three different modes (and again noise tolerance 5 and no pre-processing).

To test how the speed of the algorithm changes with different types of input data, 5,000 images have been taken from a dataset with heavy noise and light noise, and the amount of microseconds have been measured. Since both datasets contained images of different sizes, the amount of microseconds have been divided by the amount of pixels in the output image, to get the microseconds per pixel. The light noise dataset has not been preprocessed and noise tolerance 5 has been used. The heavy noise dataset has been preprocessed and run with noise tolerance 22,000. Both simple mode and accurate mode have been used to compare the speed. Since accurate mode uses an extra step to find the exact midpoints, one would think that the simple mode might be faster than the accurate mode.

In order to test the performance of the Hadoop implementation compared to the ImageJ plugin, the following experiment was done. The plugin was run on a dual core Intel Pentium G4400 processor operating at 3,3GHz while the Hadoop enabled cluster had 97 nodes available with 8 cores each. Because Hadoop is meant for parallelization purposes, the plugin has been set by default to use multiple cores as well. There were four different dataset made, all containing 1,000 TIFF files. A dataset using the first 1,000 images from a larger dataset, where each image has a width and height of 800 pixels. The second dataset uses the same images as the first one, but each image has been stretched to a width and height of 8,000 pixels. The third dataset used the first image of a larger dataset with a width and height of 800 pixels, that was duplicated a 1,000 times. The fourth dataset consists of the images found in the third dataset, but each image is replicated 100 times and combined into a grid to create images with a width and height of 8,000 pixels so that each image has hundred times more white dots. The first and second datasets have an average of 82 white dots per image. The third dataset has images with exactly 105 white dots per image, while the images in the fourth dataset have exactly 10,500 white dots each.

2.3 Image Thresholder

The Photon Image Processor creates a high resolution image from the single photon event data. Although this output image does not contain as much noise as a regular picture, it is still not completely noise-free.

2.3.1 The algorithm

A simple way to reduce the noise is to set a threshold on the image colors. Because if there is only one photon counted on a given pixel, there is a bigger chance that this was not a true photon than if there are multiple photons counted on that pixel. The threshold value should be between zero and the amount of different grayscale colors present in the image. For example, when '5' is given as threshold value, the lower five colors are removed from the image. The remaining values will receive new colors in order to balance the image.

2.3.2 Image Thresholder experiment

To show the effect of the Image Thresholder, one of the output images of the Photon Image Processor (70,000 input images, no pre-processing noise tolerance 5, simple mode) has been thresholded with threshold values 3 and 6.

2.4 Image Reconstructor

Sometimes there are not enough input images available, and the output images are not saturated enough. In those cases, it is possible to improve the output image using image reconstruction. This option is based on the algorithm by P. A. Morris et al.[1], it has been implemented as part of the ImageJ plugin but has been altered slightly to fit the purpose better.

2.4.1 The algorithm

When the image is not saturated enough, and the data is too 'sparse', blurring the image can fill up the gaps. The downside of this is that the details of the image get lost. This algorithm combines the best of both worlds by first blurring the image, which makes it less 'sparse'. But then bringing the details back into the image by optimizing the log likelihood of each pixel based on the original color, and simultaneously trying to keep the image sparsity low. This creates a smoother image with less gaps that still contains a lot of the original details.

There are several settings that all have default values but can be changed by the user, in order to make the plugin fit the data better. The 'blur radius' setting is the blur radius for the Gaussian blur, a bigger radius removes more detail from the original image but also closes more gaps between pixels. When the image is blurred, it often becomes darker. To compensate this, the colors of the pixels in the image are multiplied by the value 'multiply image colors'. Also, a dark count rate per pixel can be set, since this might vary per camera. The regularization factor is the factor that indicates how important the log likelihood and image sparsity are compared to one another. A higher regularization factor results in a greater dependency of image sparsity, and a lower regularization

factor makes the log likelihood more important. And at last, the modification threshold that controls how far the algorithm is proceeded. If the algorithm quits too quickly, the image still looks blurry, but if the image is over processed, it starts to look too much like the original image.

2.4.2 Image Reconstructor experiments

The default settings (see table 1) have been used for the following experiments, unless stated otherwise. The Image Reconstructor should be used if there were not enough input images available to make a good output image. To test the Image Reconstructor, the output images of the Photon Image Processor with a low amount of input images (8,750) have been used. These were images from the low noise dataset, so no pre-processing, noise tolerance 5 and the accurate algorithm have been used. To show the effects more clearly only 150 by 150 pixel sized images are cut out of the original images. But the size of the input images does not affect the way the algorithm works.

To show the effect of the Image Reconstructor on this 150 by 150 pixel sized image, it has been reconstructed with multiplication value 2. This multiplication value has been chosen to make the image lighter, because the blurring step causes the image to look darker. To prove that the output was not solely an effect of the multiplication, a version of the original image with all colors multiplied by 2 has also been made.

Since this algorithm is mostly about enhancing the signal that is given, and not so much about reducing noise, it is expected that the algorithm will do better on data that have first been treated by the Image Thresholder. To test this, the same image as before has been thresholded with threshold value 2. This image looks darker because of the thresholding, so it is reconstructed with multiplication value 3 instead of 2. Again, the original image multiplied by the same multiplication value has been made for comparison.

An interesting parameter is the regularization factor. This factor is used to set the balance between the importance of the matrix sparsity and the log likelihood. In the original article[1] this factor played a large role, so big and small values will be tested for this factor in order to see whether the Java implementation behaves the same. A low regularization factor makes the log likelihood more important, so the output image should look more like the input image. A high regularization factor makes a low matrix sparsity favorable, so the image should look smoother. To test this effect, the Image Reconstructor has been run with a very high regularization factor (20), and a very low regularization factor (0.000,0001). To show the effect of the regularization factor more clearly, a lighter input image (made of 17,500 input images) has been used, so the multiplication value could be left at 1.

Table 1: Default settings for all algorithms.

Algorithm	Setting	Default value
Photon Image Processor	Pre-processing	True (enabled)
	Noise tolerance	100
	Mode	Simple
Image Thresholder	Threshold value	1
Image Reconstructor	Modification Threshold	25%
	Dark count rate per pixel	0.1
	Regularization factor	0.5
	Multiply image color	1 (no multiplication happens)
	Blur radius	2

3. Results

3.1 Photon Image Processor

The Photon Image Processor can be used to create a single high resolution image from single photon event data. A number of tests have been run to demonstrate its performance.

3.1.1 Photon Image Processor versus a regular microscope camera

The Photon Image Processor has been run on 17,500; 35,00 and 70,000 images of the same dataset to compare the output to a photo taken with a regular microscope camera. The results are shown in figure 2. Figure 2A shows the average of 10 images taken by the camera. Figures 2B - 2D show the output of Photon Image Processor with different amounts of input images. The output becomes more clear when more input images are being used.

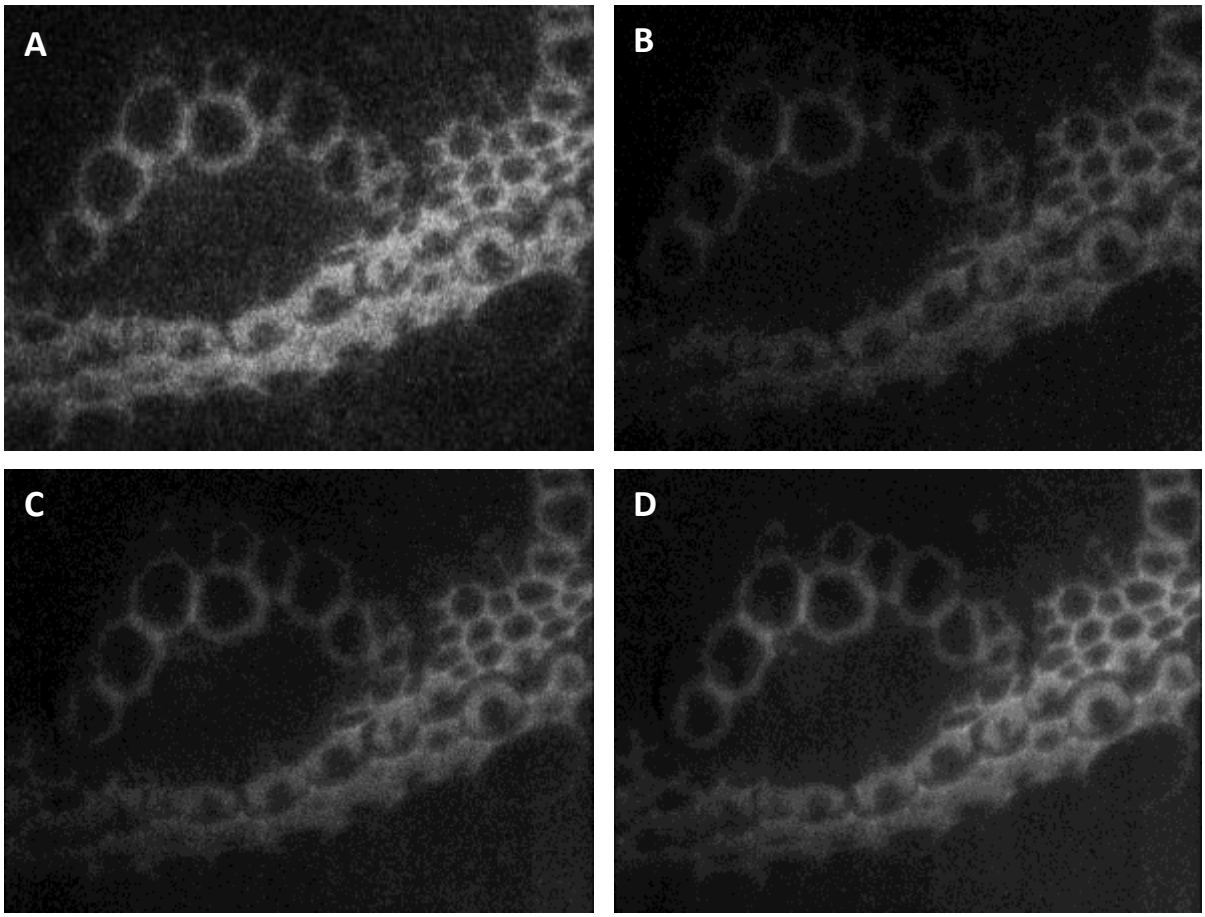


Figure 2: The results of Photon Image Processor with different amounts of input images, compared to a photo taken with a regular microscope camera. A is taken with a regular camera; B is the result of 17,500 input images; C is the result of 35,000 input images; D is the result of 70,000 input images.

3.1.2 The different modes of Photon Image Processor

The whole dataset of 70,000 images has also been run with the three different modes: simple, accurate and sub-pixel resolution. The outputs of those three modes are compared in figure 3. Figure 3A was made using the simple mode which uses a naive method to gather the positions of the photons. The resulting image of accurate mode is shown in figure 2B. Figure 2C uses sub-pixel resolution mode, this output image is twice as high and wide as the other two images. The differences between simple and accurate method are not very big, the images look very similar on the first sight, but in reality they are not exactly the same. The output of sub-pixel resolution mode does not look as saturated as the output of the other two images.

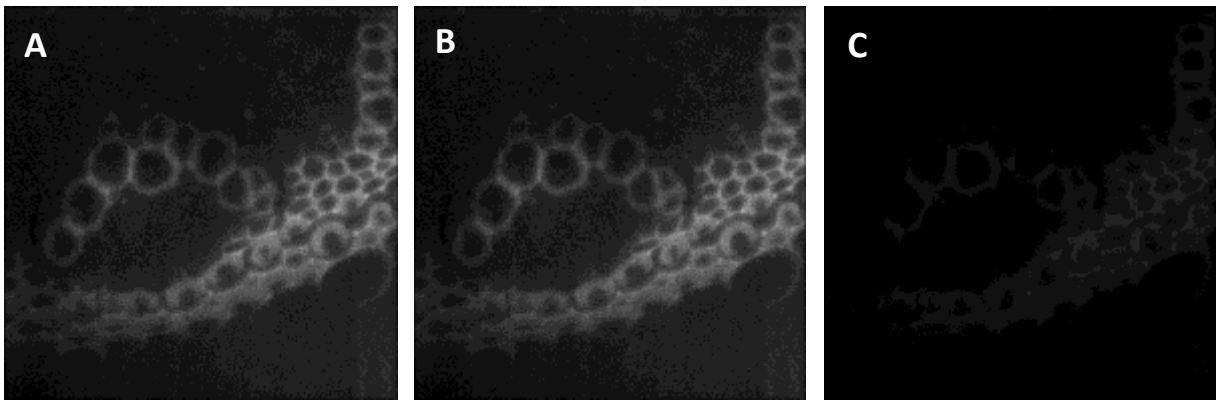


Figure 3: The results of Photon Image Processor using different processing modes. A is the result using simple mode; B is the result using accurate mode; C is the result using sub-pixel resolution mode.

3.1.3 Speed of the algorithm modes using two different datasets

To show how the duration of the algorithm differs when using different kinds of input data, a dataset with a lot of noise has been compared to a dataset containing very little noise. This has been done using both simple and accurate method. The speed has been measured in microseconds per pixel, to compensate for the difference in size between the images of both the datasets. Note that parallelization has been turned off for this experiment, so in real life the algorithm would run faster. Results for this experiment are shown in figure 4. It is clear that the processing of the data with heavy noise takes more than twice as long as the data with very little noise. Interestingly, there does not really seem to be a difference between the simple and accurate algorithm, for both datasets. So speed should not be a reason to prefer the simple algorithm over the accurate algorithm.

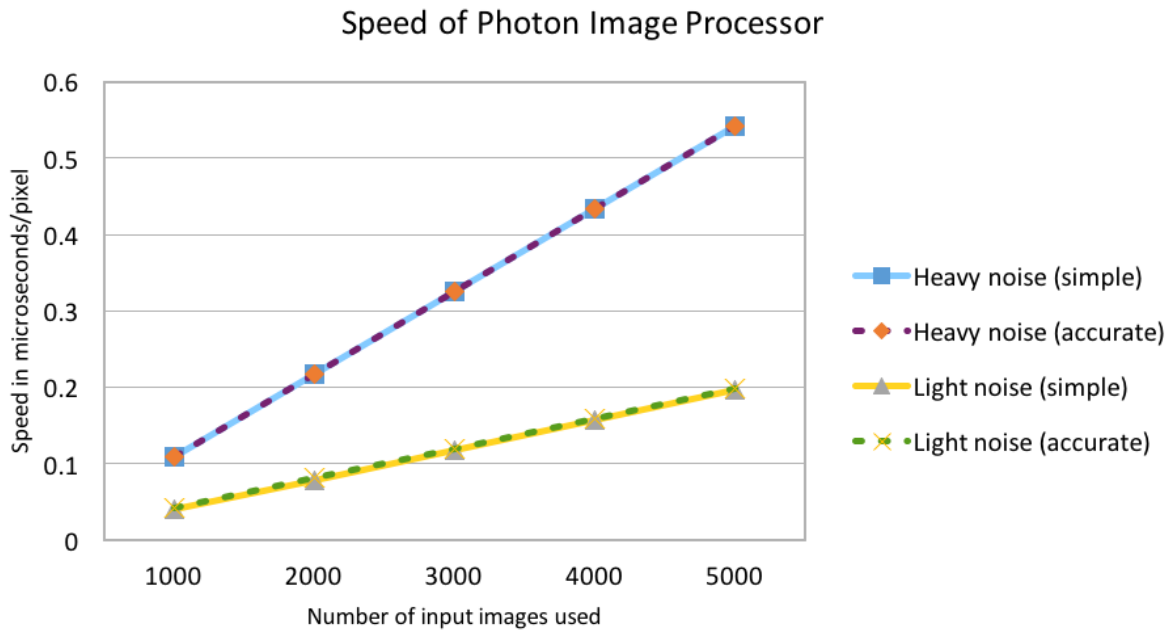


Figure 4: Speed comparison between simple and accurate mode of Photon Image Processor using images with light and heavy noise.

3.1.4 Parallelized Photon Image Processor in Hadoop

Figure 5 displays two graphs, each of which show the amount of time in seconds it takes to complete the task. In both figures, the blue bars represent the original Photon Image Processor using ImageJ, while the yellow bars represent the Hadoop cluster. Figure 5A uses for both tests the first 1,000 images with a width and height of 800 pixels from a larger dataset. These have been stretched out to create images with a width and height of 8,000 pixels.

Figure 5B uses for both tests one image of 800 by 800 pixels which has been duplicated a 1,000 times. The same dataset has been used so that each image has been replicated a hundred times and combined into a grid with a width and height of 8,000 pixels.

In both graphs, the smaller sized images perform better using the ImageJ plugin, while the Hadoop implementation gains a massive improvement using the larger sized images. The plugin slows down tremendously when the required processing for each image increases.

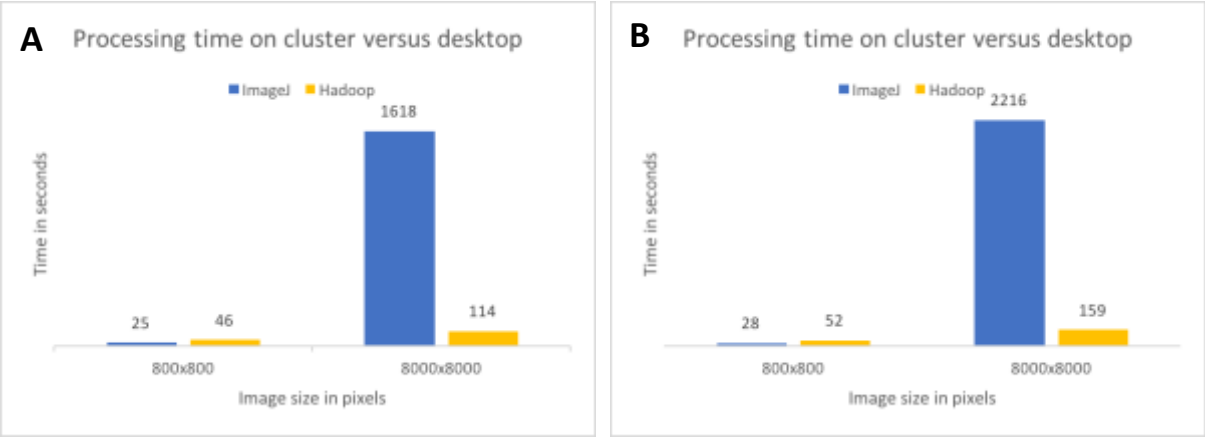


Figure 5: Comparison between ImageJ plugin versus Hadoop implementation. Graph A uses 1,000 different images (width and height of 800 pixels) as small dataset and a dataset where the images from the small dataset are stretched (width and height of 8,000 pixels). Graph B uses 1,000 copies of one image (width and height of 800 pixels) as small dataset and a dataset where each image in the small dataset has been transformed into a grid of 100 times the image (width and height of 8,000 pixels).

3.2 Image Thresholder

One experiment has been done to show the results of using the Image Thresholder on the output images of Photon Image Reconstructor. Figure 6 shows this original output image compared to the thresholded images with threshold values 3 and 6. It can be seen that the Image Thresholder does not simply make the whole image darker. Instead, the remaining colors of the images are rescaled, so the lightest pixels in the image will stay the same color, while the darkest pixels become even darker.

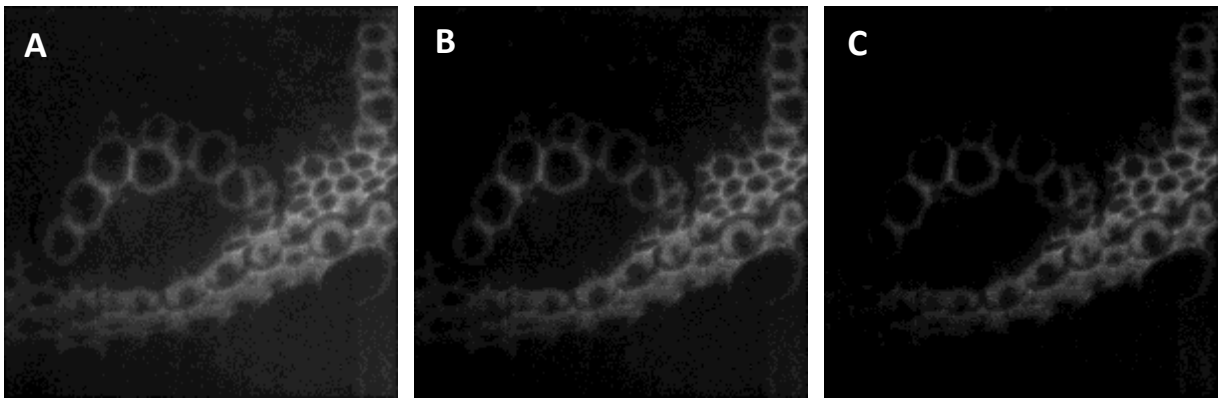


Figure 6: A comparison between thresholded and non-thresholded images. A is the original image; B is the result using a threshold value of 3; C is the result using a threshold value of 6.

3.3 Image Reconstructor

The quality of the output image is highly dependent on the amount of input images given. Sometimes there are not a lot of input images available, and the Image Reconstructor can be used to enhance the signal that is present.

3.3.1 Reconstructing the original image

To demonstrate the algorithm directly on the output of the Photon Image Processor, the output image of the 'accurate' algorithm with a small amount (8,750) of input images has been used for image reconstruction. Figure 7 shows the results of this experiment. Figure 7A shows the image before reconstruction, and figure 7B after reconstruction. For comparison, figure 7C shows the image multiplied by 2 (the same multiplication value was used for reconstruction), to show that this does not give the same results as using the Image Reconstructor. The multiplied image looks a lot harsher and granular than the reconstructed image.

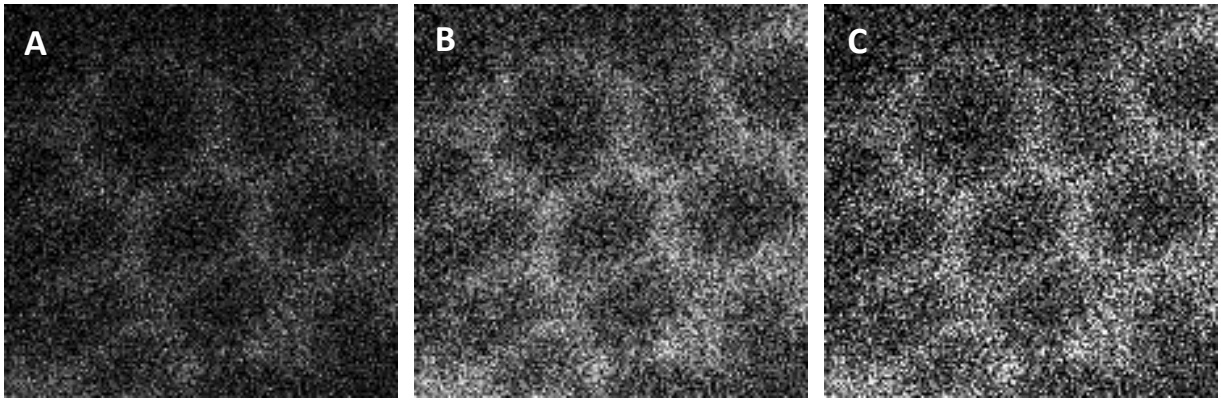


Figure 7: Image Reconstructor using the original image. A is the original non-reconstructed image; B is a reconstructed image with a multiplication value of 2; C is the original image multiplied by 2.

3.3.2 Reconstructing the thresholded image

It was expected that the reconstruction algorithm would work better with thresholded images, since it does not handle noise very well. Figure 8A shows the image from the last experiment thresholded with threshold value 2, 8B shows the reconstructed version of this image with multiplication value 3, and 8C shows the results of multiplying all colors in image 8A by 3. Again, the reconstructed image is smoother than the multiplied version. But this time it looks like even more gaps between the pixels have been closed up by reconstruction, and the structures are more connected, while the noise has not been enhanced as much. The noise has not been removed, but it looks like independent dots instead of connected structures.

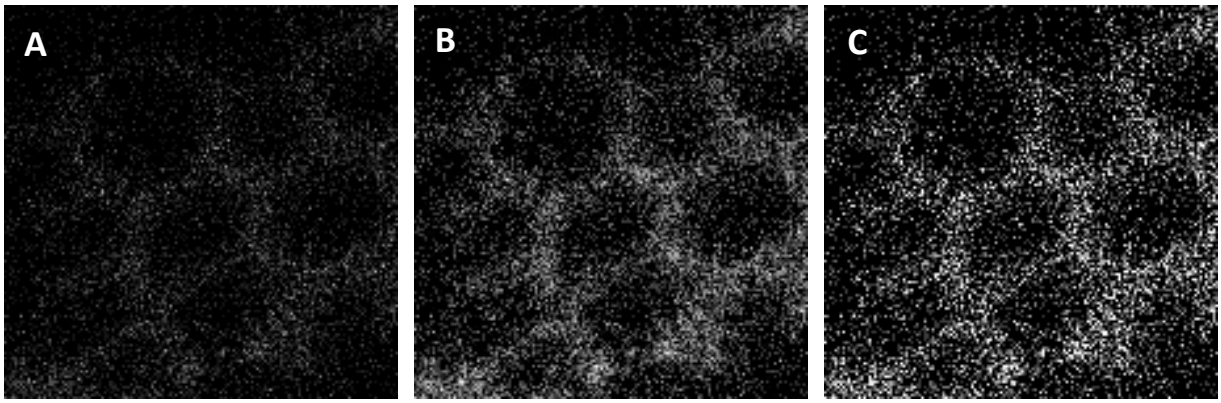


Figure 8: Image Reconstructor using a thresholded image. A is a thresholded image with a threshold of 2; B is a reconstructed thresholded image with a multiplication value of 3; C is a thresholded image multiplied by 3.

3.3.3 The effect of the regularization factor

Different values for the regularization factor have been tested in order to see if it behaves the same way as in the original article. Regularization factors 20 and 0.000,000,1 have been tested for the same input image, and the results are shown in figure 9A and figure 9B respectively. The image with the higher regularization factor, looks blurrier, and the image with the low regularization factor looks more defined. This is the behavior that was expected, but it is not as strong as seen in the original article[1].

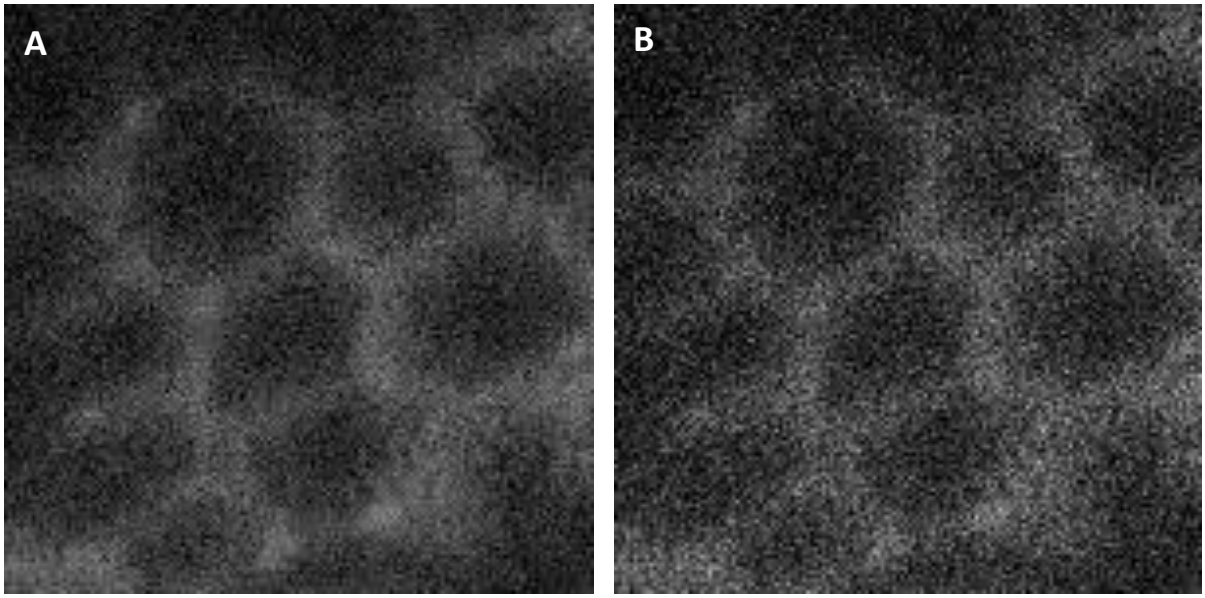


Figure 9: Image Reconstructor using different regularization factors. A is a reconstructed image with a regularization factor of 20; B is a reconstructed image with a regularization factor of 0.000,000,1.

4. Discussion

The results show that it is indeed possible to use the ImageJ plugin to create better images than using a regular microscope camera. Figure 2 shows the comparison of the camera image and the results of Photon Image Processor using different amounts of input images. The images clearly get better when using more input images. More photons in the input data create a smoother output image with a larger variety of pixel values. When using 70,000 input images, the output image is a lot less noisy and harsh than the image made by the microscope camera. When using more input images, the output gets even better.

When testing the different modes of the Photon Image Processor, the results between simple mode and accurate mode turn out to be very similar, as seen in figure 3. On the first sight they look completely the same, but when comparing the images pixel by pixel there are some differences. One does not seem to be better than the other. Even when comparing the speed (figure 4) it does not really make a difference.
The sub-pixel resolution output does not look very good for this dataset, it has a line pattern. The explanation for this is simple; the input data contained very little noise and very condensed light blobs. These light blobs often only consisted of one relatively bright pixel. In those cases, the sub-pixel resolution algorithm only counts a pixel in the upper left corner of the pixel. The upper right, lower left and lower right corners are only 'counted' when the center of the light blob was stretched over multiple pixels. Which was not often the case in this dataset, but when using a dataset where the light blobs are more defined, it might definitely work.

The Hadoop implementation can be used for parallelization of the Photon Image Processor. In both figures, the ImageJ plugin performed slightly better with the 800 by 800 sized images, while the cluster had an incremental improvement over the plugin with 8,000 by 8,000 sized images. So why is the plugin faster on the smaller sized images? This is because of the file size of the images. The datasets containing small images have image sizes of around 650 Kb, while the datasets with larger images have images with an image size of 60 Mb. The Hadoop cluster creates the same amount of map functions for both the large and small images, which is 1,000 map functions. Each Mapper processes one image, and while an image of 60 Mb takes a bit of time to process, an image of 650 Kb is completed within milliseconds, creating an overhead.
The plugin takes a different approach; it only creates the necessary classes once. These classes process each image one by one and multithreading is used to speed up the runtime.
The datasets with large images create a hard drive bottleneck on the computer running the plugin with only two processing cores, while the cluster is able to use a lot more of these cores at once.

The Hadoop version of the Photon Image Processor works great on large files, but needs to be improved when using small files. A solution would be to use Hadoop Image Processing Interface (HIPI)[2] for the pre-processing of the input dataset. This interface creates an HIPI input bundle, containing all the image file including metadata. The specialized input formatting function is able to divide the images within the bundle into multiple stacks of images. Each stack is given to one map function. This automatic process should greatly reduce the amount of overhead, initializing less

Mappers which perform their tasks on multiple images each. There is only one big downside of HIPI, which is the missing TIFF file support. All our datasets consist out of TIFF image files, which made plain Hadoop a better solution. It would also be great to see how the Hadoop implementation performs on 100,000 images of 650 Kb each in comparison to the plugin. We were not able to test this, because of network limitations. It takes quite a lot of time to transfer datasets to the cluster its filesystem. This process is so slow that the dual core system could have already processed the data and create an output image in the same time.

The Image Thresholder is able to reduce unwanted noise from images created by the photon image processor. Figure 6 shows that thresholding the image sets the darkest colors in the image to black, and then scales the remaining colors so the lightest parts of the image do not get darker, while the darker parts of the image become a lot darker. This makes the contrast between the object of interest and the background higher. Thresholding the image is a way to reduce noise since a pixel that has counted only a few photons is more likely to have counted false positives than a pixel with a very light color (equals many photons). The downside is that this obviously also removes the information of a lot of real photons. Thresholding the image has also been shown to be a good way to prepare the image for reconstruction.

The Image Reconstructor can be used to improve the quality of the image if there is not a lot of data given. The reconstructed images are smoother than the original images, but the details are not blurred out. Figure 8 shows that the Image Reconstructor is capable of closing 'gaps' in sparse-looking images. Image Reconstruction is clearly not the same as just multiplying the original values by a factor, because this last method creates images that look harsher and granular (see figures 7C and 8C). The results look even better if the noise is first removed from the image by the Image Thresholder, this can be seen when comparing the images in figure 7 to the thresholded images in figure 8. In the non-thresholded reconstructed image the signal of the noise has also been enhanced. The results of the reconstructed thresholded image look even slightly more clear than the original, non-thresholded image.

The algorithm used for image reconstruction was based on the algorithm by P. A. Morris et al.[1] Their results showed that the regularization factor had a very big impact the way their output images looked. A high regularization factor makes the algorithm prefer a low sparsity (= smooth image), while a low regularization factor makes the log likelihood (= similar to the input image) more important. The Java implementation showed the expected behavior, but the effect in figure 9 is not as strong as the effect shown in figure 3 of 'Imaging with a small number of photons'[1].
The original algorithm was created in the visual programming platform LabVIEW. LabVIEW knows a lot of built in calculations (like for instance the Direct Cosine Transform that is used to calculate the image matrix sparsity), and ImageJ does not standard have this. Besides that, their raw data consisted of black and white images without grayscale, and with a lot less false positive photons because of the way their data is collected. These are all factors that could contribute to the fact that the algorithms do not seem to behave exactly the same. However, that does not mean the ImageJ version cannot be useful.

5. Conclusion

The main goal was to create an ImageJ plugin that is able to convert the single photon data from a camera with an image intensifier to more detailed, combined output image. Other goals were to parallelize this plugin so it could run on a computer cluster, and implement something to improve the images if not enough input images were recorded.

The Photon Image Processor is indeed capable of creating images that are of better quality than the camera-recorded input images. The images look smoother and also slightly more detailed, because they contain a lot less noise. The more input images, the better the results. So if a larger amount of input images would be used, the output would be even more detailed and saturated. A big advantage of the Photon Image Processor is that it works on a lot of types of input data. It can both handle noisy data and noiseless data. Unfortunately, the sub-pixel resolution mode could not be tested so well with the given dataset, since the photon blobs were too concentrated.

The plugin has been parallelized using Hadoop's MapReduce framework. While the performance is better with large files, the parallelized version is not faster when using the smaller, unmodified input images. This is because it is currently only possible to divide all single input images over the Mappers, instead of giving each Mapper a stack of images. This problem could be solved by using HIPI, but this has not been implemented because HIPI cannot use TIFF input files.

Finally, if there are not enough input images given to create a good output image, the Image Reconstructor can be used to improve the output image. To get a better result from the Image Reconstructor, the Image Thresholder can first be used to reduce the noise in the image. The effect of the Image Reconstructor is not as strong as observed in the original algorithm[1], but it can still be used to improve unsaturated images.

As for now, we can say that our goals for this project have been reached. While the Hadoop version of the program could still be improved, the ImageJ plugin does bring the necessary functionality to customers of Lambert Instruments B.V.

References

1. P. A. Morris *et al.* 'Imaging with a small number of photons.' *Nat. Commun.* 6:5913 doi: 10.1038/ncomms6913 (2015).

2. 2. S. Arietta, J. Lawrence, L. Liu, C. Sweeney, 'HIPI, Hadoop Image Processing Interface', http://hipi.cs.virginia.edu/, (20-06-2016)