Patricio Beltrán Molas
N1500934H
CZ4003 Computer Vision

Lab 1: Point Processing + Spatial Filtering + Frequency Filtering + Imaging Geometry

All source code included in a separate file.

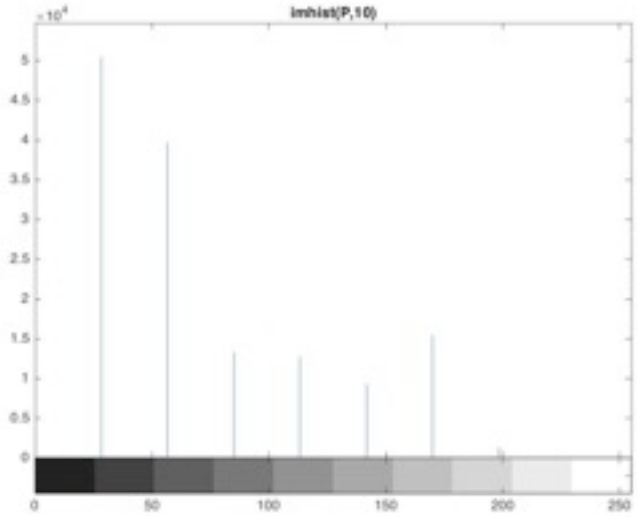
2.1 Contrast stretching

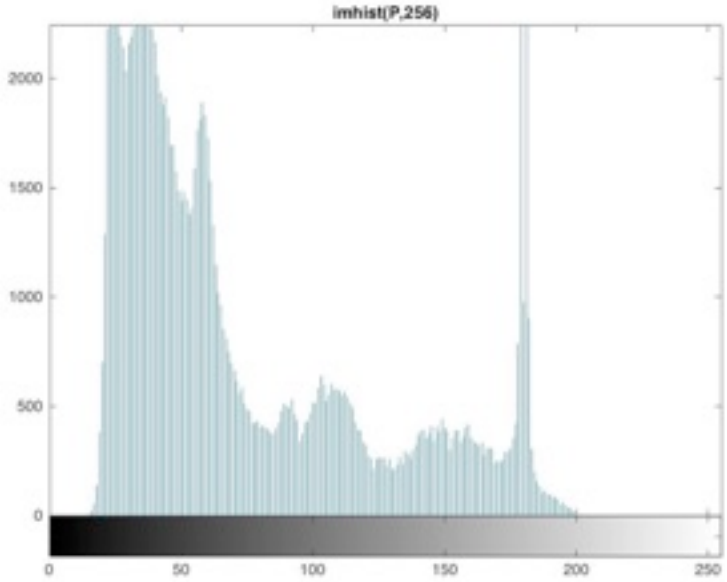
Original image:

After performing contrast stretching:

2.2 Histogram equalisation

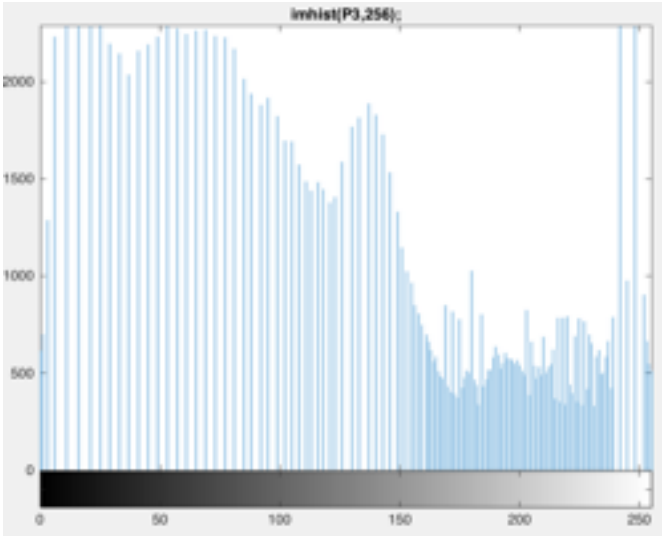
a. displaying the intensity histogram with 10 and 256 bins

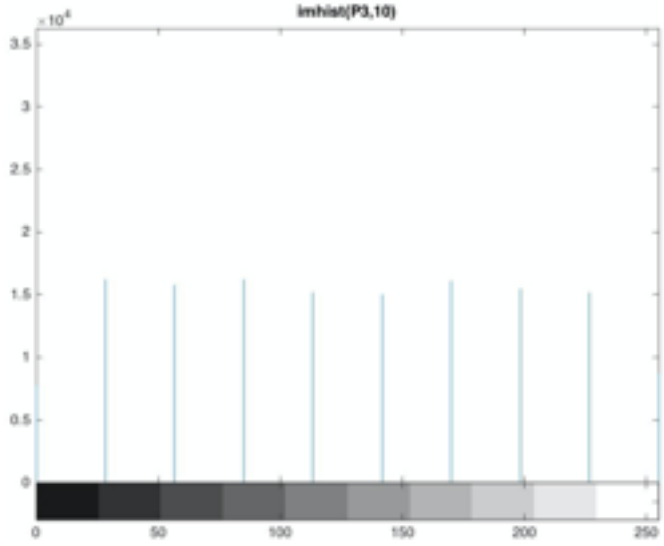




After comparing the two histograms, it can be concluded that the number of bins determines how finely one wants to visualise the data. With 10 bins, pixels are grouped by interval levels of grey, whereas with 256 you get a count of how many pixels there are in each of the 255 levels.

b. Equalize the histograms and we get this:





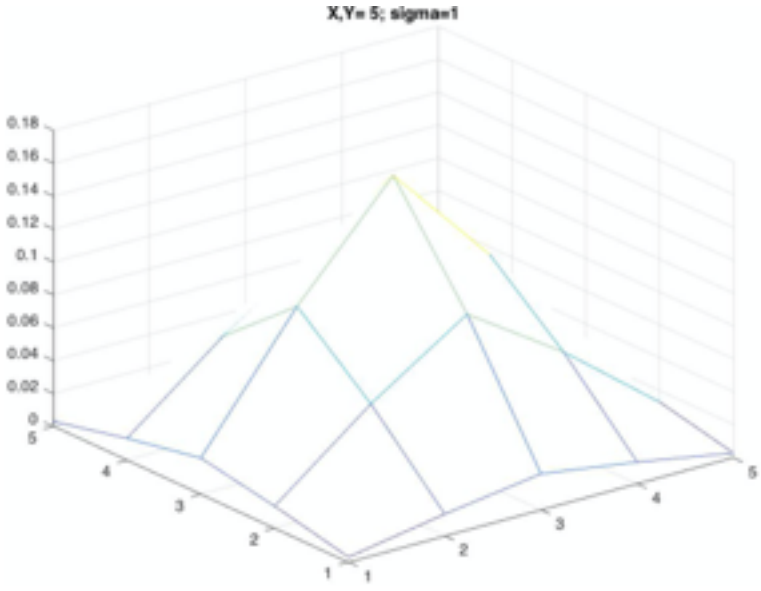
Yes, effectively the histograms were equalised. This is more noticeably in the image on the right, the one with 10 bins. We can see that the pixel distribution is more uniform.

Re-equalising the histograms yielded the exact same histograms. This is because the equalisation algorithm ends up being a series that does not change after two times.

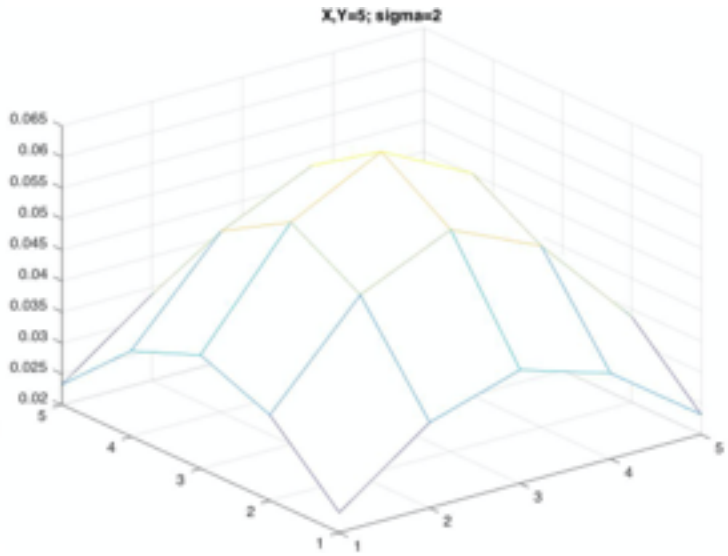
2.3 Linear Spatial Filtering

We create two gaussian filters:

Filter 1:



Filter 2:



And try to use them to filter noise out of this original image with gaussian noise:

After applying the filters, we get interesting results:
Filter 1:

From observing the two images we can conclude that sigma = 2 is a way more aggressive filter that, while effectively removes noise, it also causes a lot of blur that may be undesirable. While sigma = 1 preserves a lot more of detail from the original image but also filters out considerably less noise.
So those are the tradeoffs, we have to be intelligent and choose the priorities correctly when designing or implementing a gaussian filter and choose a radius of blurring adequately.

The following image has added speckle noise, we will try to reduce it using our two gaussian filters:

speckle noise with filter 1

speckle noise with filter 2

After observing the results, we can say that gaussian filters are not effective against speckle noise.

2.4 Median Filtering

Now we try to use median filtering with both 3X3 neighbouring and 5X5 neighbouring to reduce speckle and gaussian noise. (ntu-gn is the one with gaussian noise, while ntu-sp has speckle noise)

ntu-gn with 3x3 median filter

ntu-gn with 5X5 median fiilter

From these two, we can conclude that median filtering is not particularly effective to reduce gaussian noise.

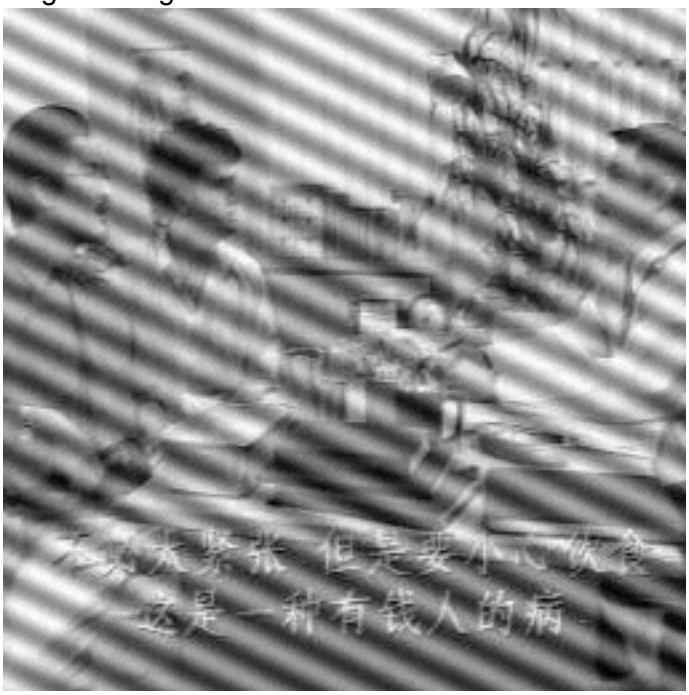
ntu-sp with 3X3 median filter

ntu-sp with 5x5 median filter

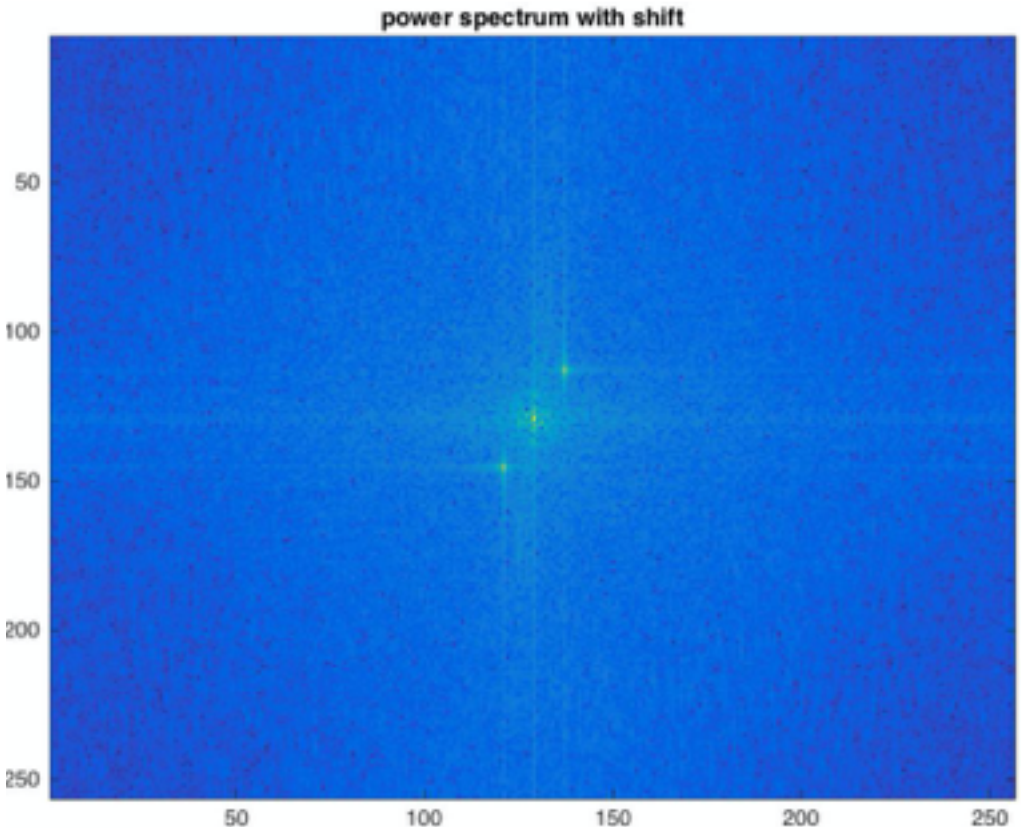
However, the filtering worked amazingly good to remove speckle noise. So we can observe that the 3X3 neighbouring filter is more delicate in removing the noise but does not cause too much blurring as compared to the 5X5 one, that, while it removes almost all noise, it causes too much blurring. I believe Adobe Lightroom 5 uses median filtering for noise reduction. So then again, there are tradeoffs and it depends on the purpose of our application how we choose our neighbouring or radiuses. Median filtering works for speckle noise and gaussian filtering works for gaussian noise.

2.5 Suppressing Noise Interference Patterns

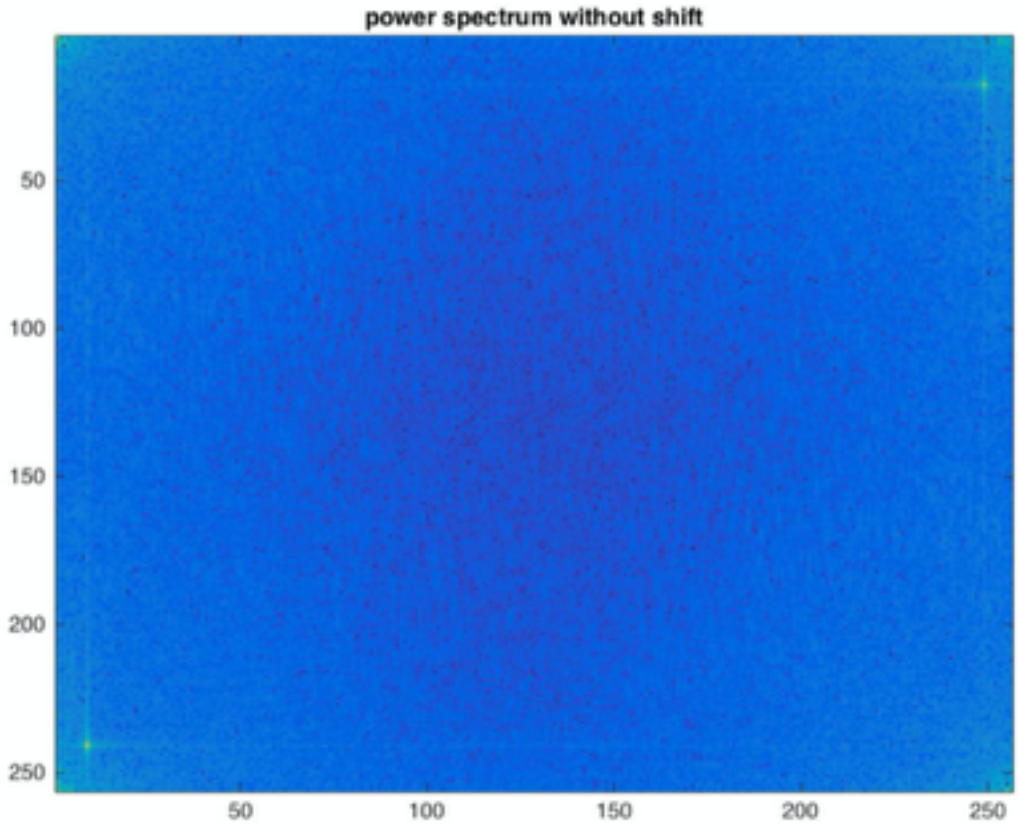
Original image:



Calculate the power spectrum and we get :

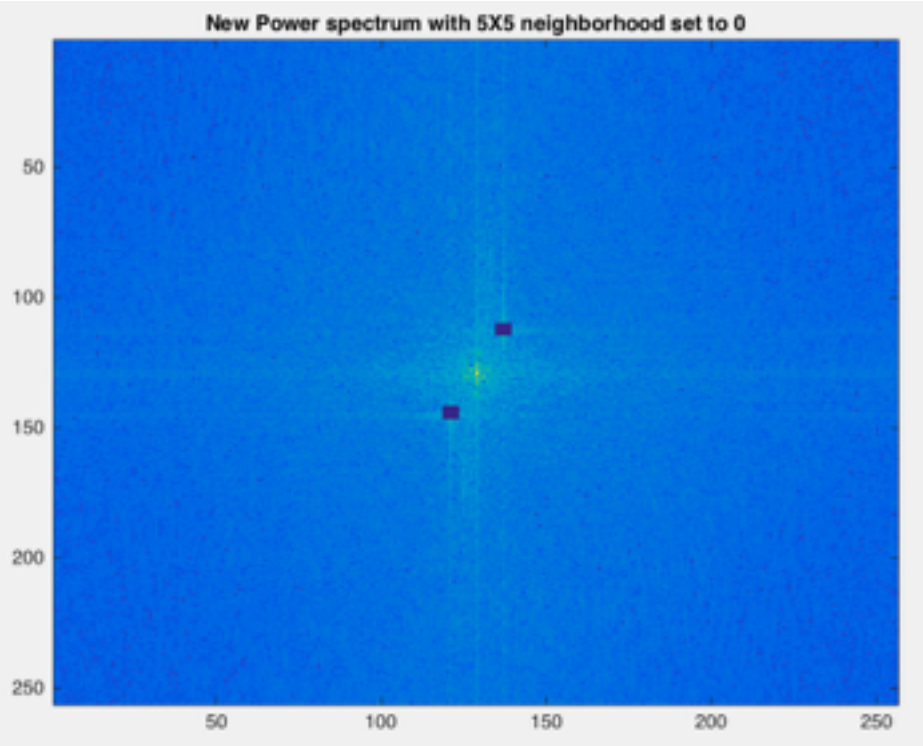


And:



fftshift is needed to change the origin to the centre.

After setting the neighbour elements of the interference peaks to 0, we get the following resulting spectrum:



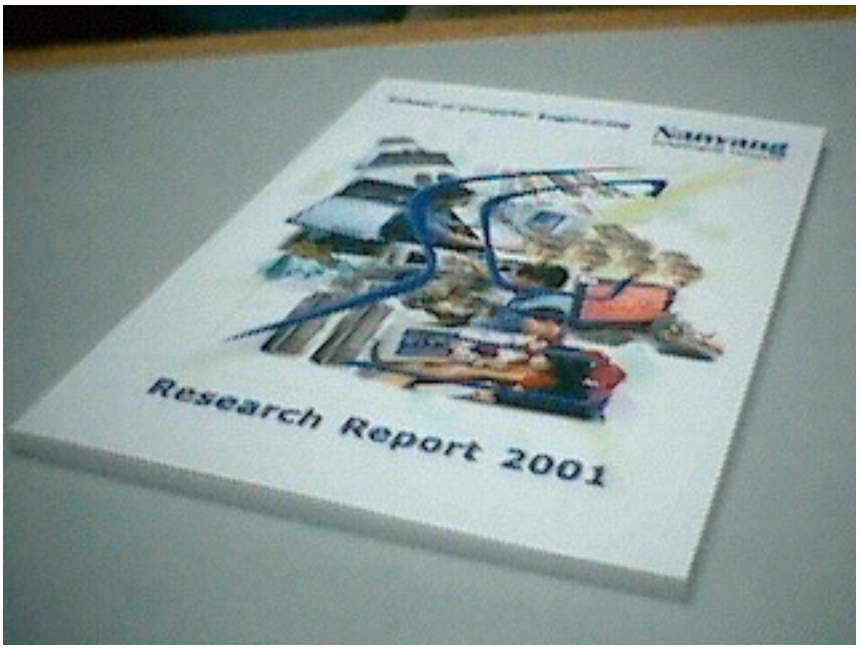
And after doing the inverse Fourier transform on that matrix, we display the final image:

Which has a lot less interference noise pattern.

When we set the neighbourhood pixels to zero in the power spectrum matrix, we are removing the most intense interference patterns. As we can see, the interference remains on the edges but this might need a more meticulous selection of the areas we want to set to zero, probably. Increasing the neighbourhood area and setting it to zero would be an effective way to remove more noise.

2.6 Undoing perspective distortion

We have the original image of a book that has perspective distortion and we'll attempt to get a front looking image from the input.



And we do the transformation process to finally arrive to this result:

Which is an impressive transformation and a great result. It is true that the upper part of the book looks distorted, but the quality of the initial image was pretty bad form the start. It would be interesting to see how the transformation starts with higher resolution images.

Patricio Beltrán Molas
N1500934H
CZ4003 Computer Vision

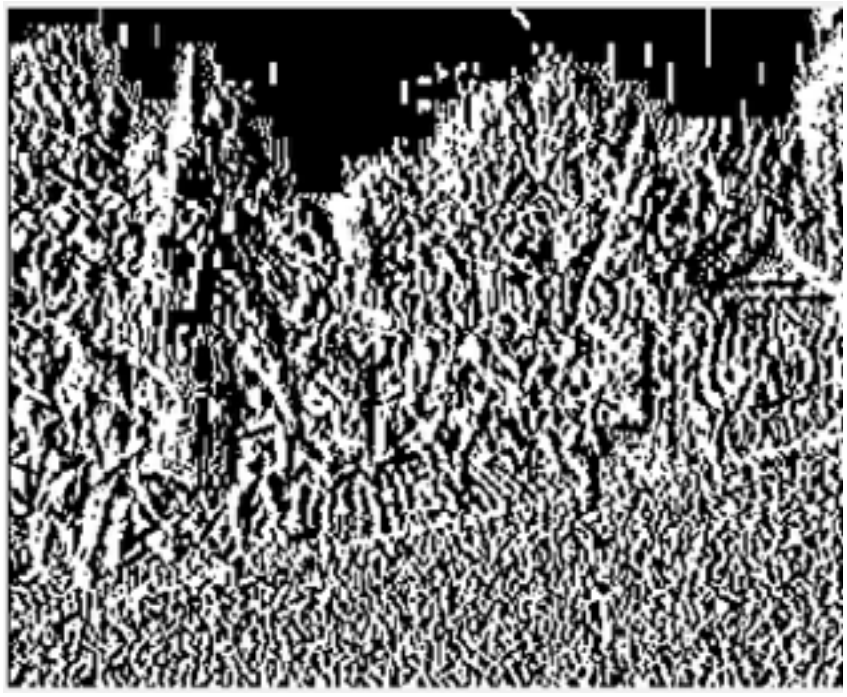
Computer Vision — Lab 2 Report

All source code is included in a separate file.

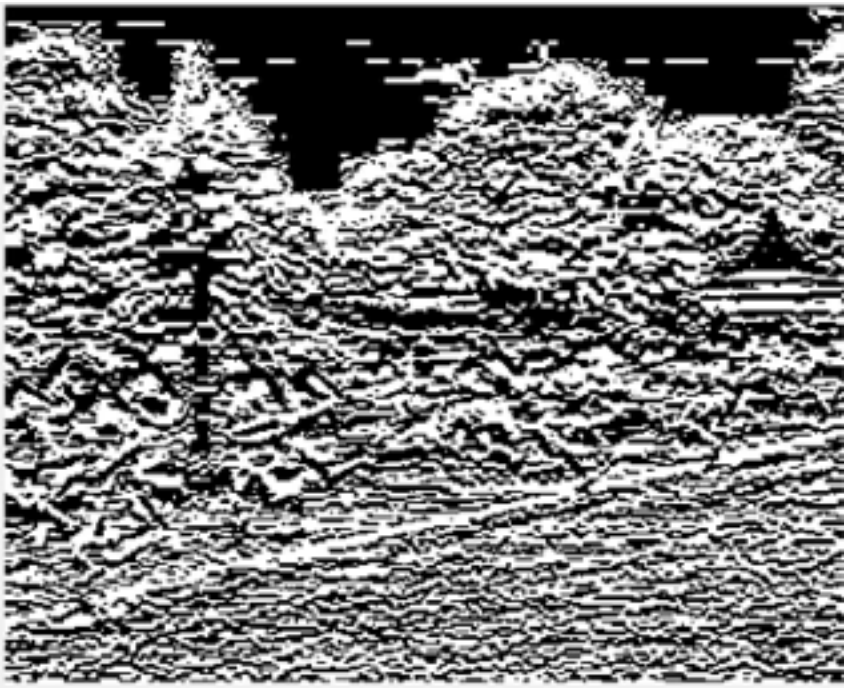
3.1 Edge Detection

We have the following original image:

And we want to detect it's edges, so we use the basic method of creating vertical and horizontal Sobel masks and filtering the image using those masks. We get the following results:
Vertical Sobel mask filtered:



Horizontal:



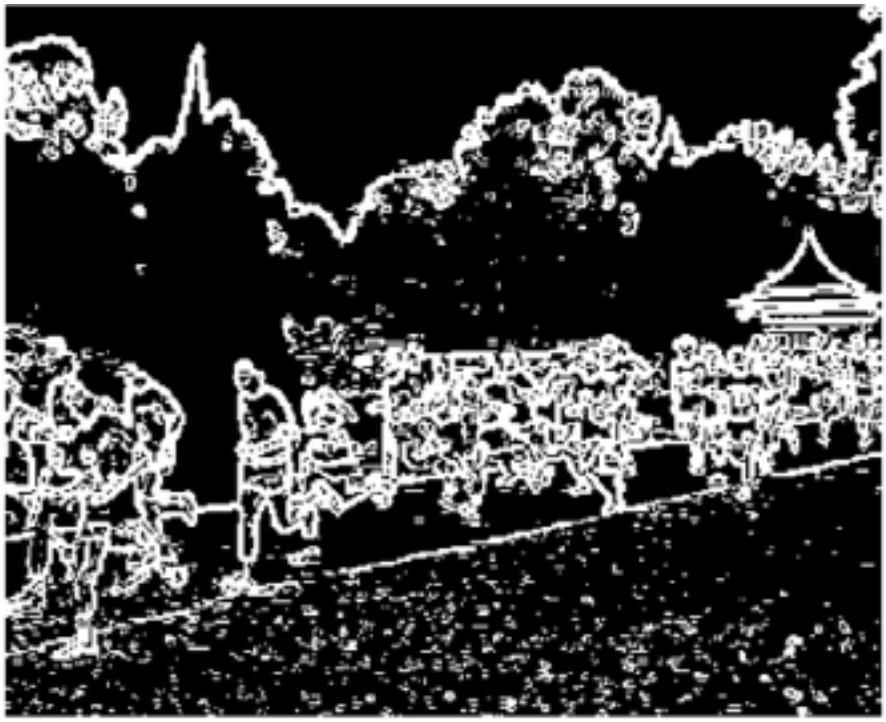
We can notice that each sober mask detects only a particular direction of edges. (horizontal or vertical). We need to combine them to get the full detection.

And to arrive to a final image, we have to square both matrices and add them and then take the square root of the result. (NOTE: the manual instructs just to square and add them, but actually the square root is necessary). We have to square them because given the Sobel masks we used, we ended up having negative values in the horizontal and vertical matrices.

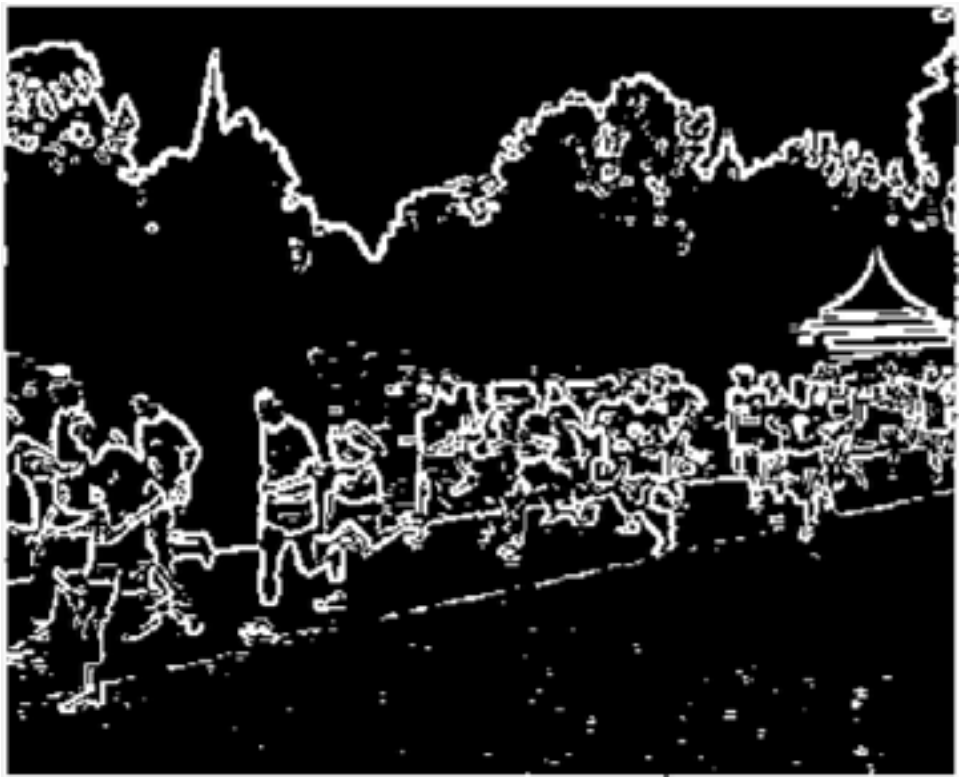
We get the following result:

We can see that both edge directions are well defined.

We can threshold the image using MATLAB's included im2bw function (using a t level of 0.498) we obtain the following image:



Or with t=.800:



So we can see that the threshold value determines how visible edges are in the image. As mentioned in the previous report, it is all a matter of the purpose we need the image for. We need to find a balance that suits our needs, try and test.

We can also perform the edge detection with the more sophisticated Canny algorithm, that comes within the Computer Vision Toolbox.

Using values of tl=.04, th=.01 and sigma= 1 we get this image:

We play around with the sigma value, raise it to 2.5.

and sigma = 5.0:

We can conclude that sigma determines how permissive the algorithm is when connecting pixels. Lower sigma values detect smaller edges while bigger values detect only the more drastic ones.

We can also play with the tl value, here we have an image with sigma=1, tl=.01 and th=.01

and with tl = .08

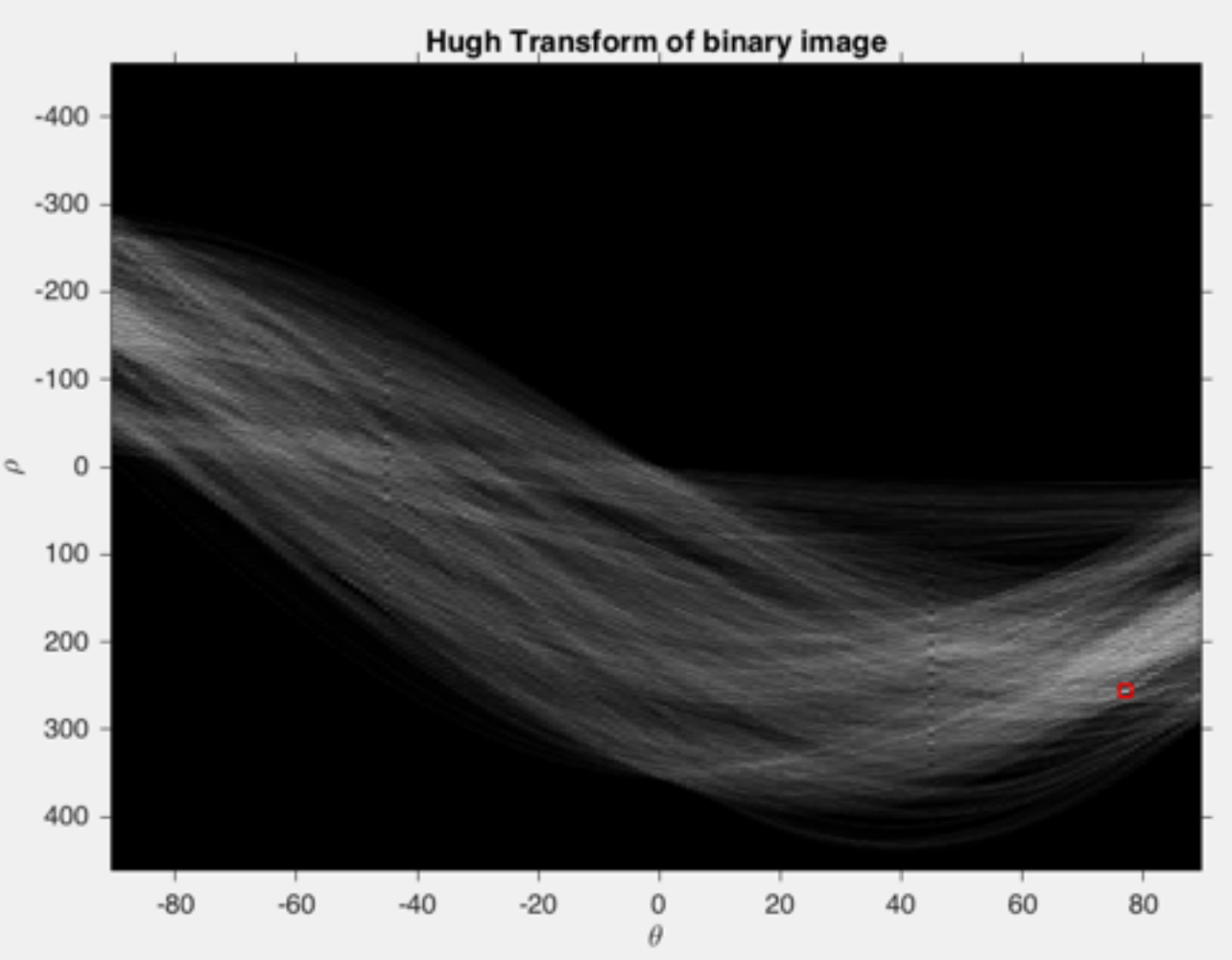
The th and tl values are threshold values. If they are equal, we get a pretty noisy image as more pixels are allowed to be connected due to the nature of the algorithm.

3.2 Line detection with Hough transform

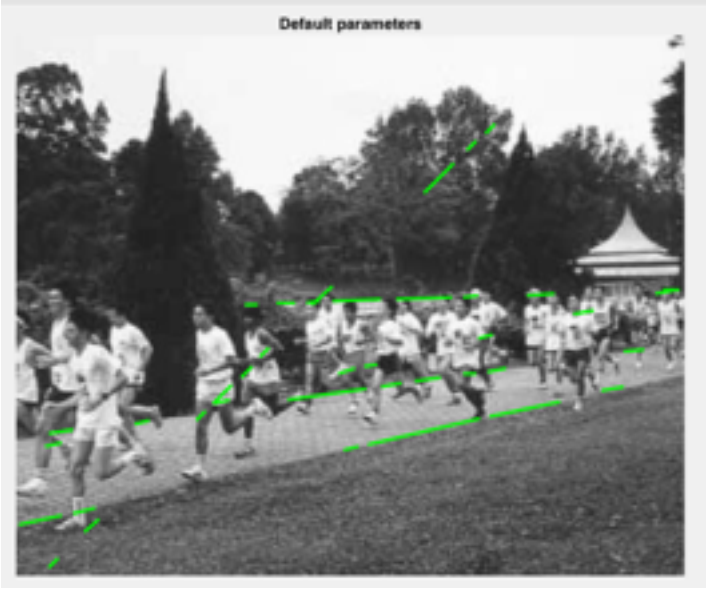
There is a Hough transform function in MATLAB r2015b, so due to my lack of proficiency in MATLAB, and in the spirit of trying out the new feature, the following was done with the toolbox.

The goal in this section is to extract the line long edge of the path in the macritchie image.
Original image:

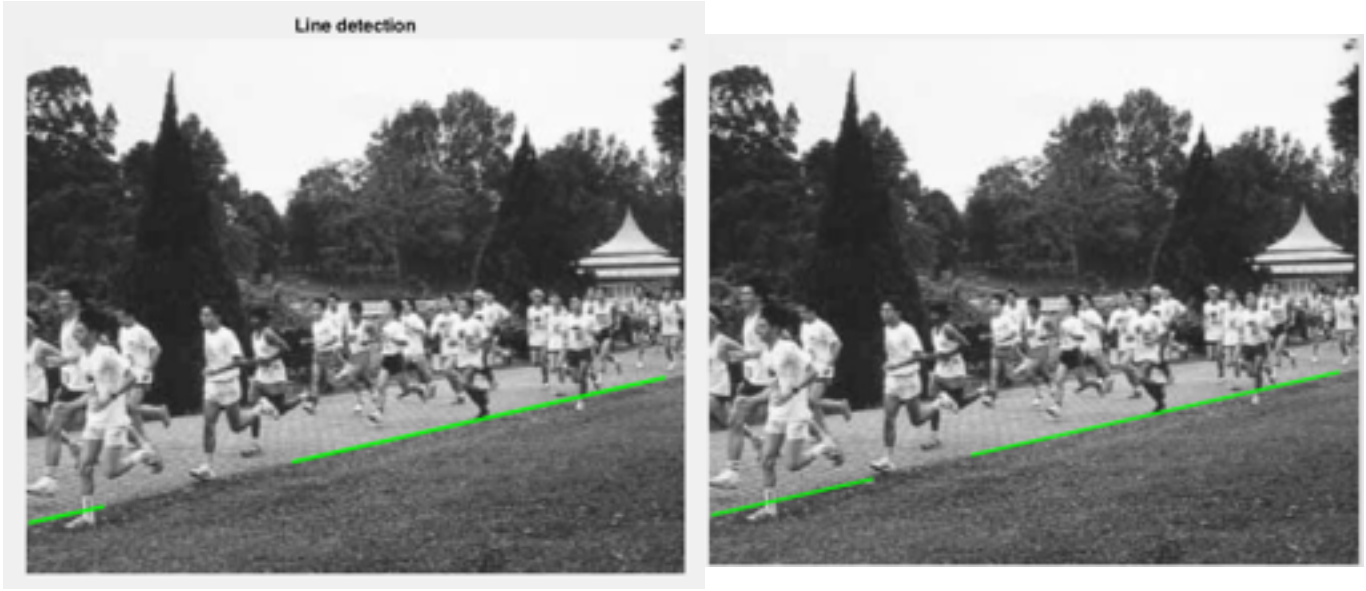
First we have to compute the Hough transform of the image, and find the peaks of the transform, which are highlighted in red. This is done with the commands hough and houghpeaks respectively. Houghpeaks can take several arguments, for example, the amount of peaks to be found and the minimum intensity of a peak to be included in the result. I decided to find just one peak (which should be the strongest line. We get the following result (peak highlighted in red).



From there, we use the command houghlines, which takes in the peaks from the transform and returns x1, y1, x2, y2… corresponding to the image lines. Houghlines also takes optional arguments, such as the minimum amount of pixels between a line and another to be considered merged in the same line.
At first, using default arguments yielded incorrect results:



But upon reading the manual of hough transform and changing parameters such as the fill ago and several others, I got closer and closer:



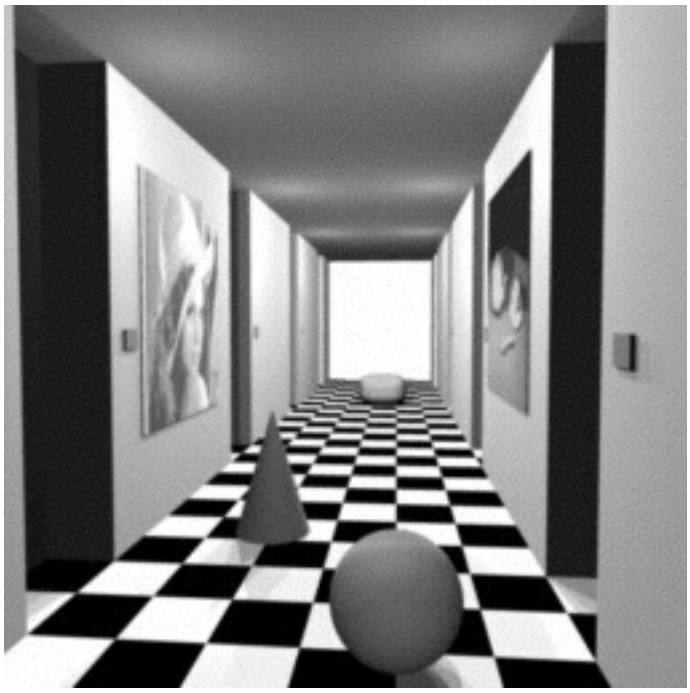
And eventually I arrived to the desired result:

We can see that this is a very useful tool once we figure put the adequate parameters to use. For this, it is important to fully red and understand the algorithm and equations laying behind the functions.

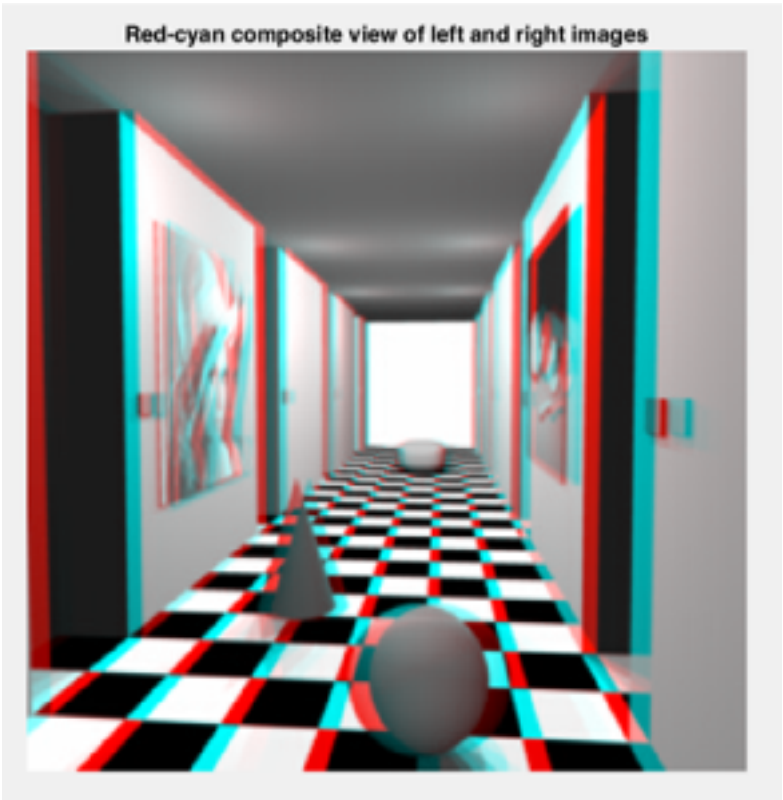
3.3 3D Stereo

Once again, I am not proficient enough in MATLAB to code the disparity script but luckily there is a disparity map function included in the computer vision toolbox, which I'll be using in this section.

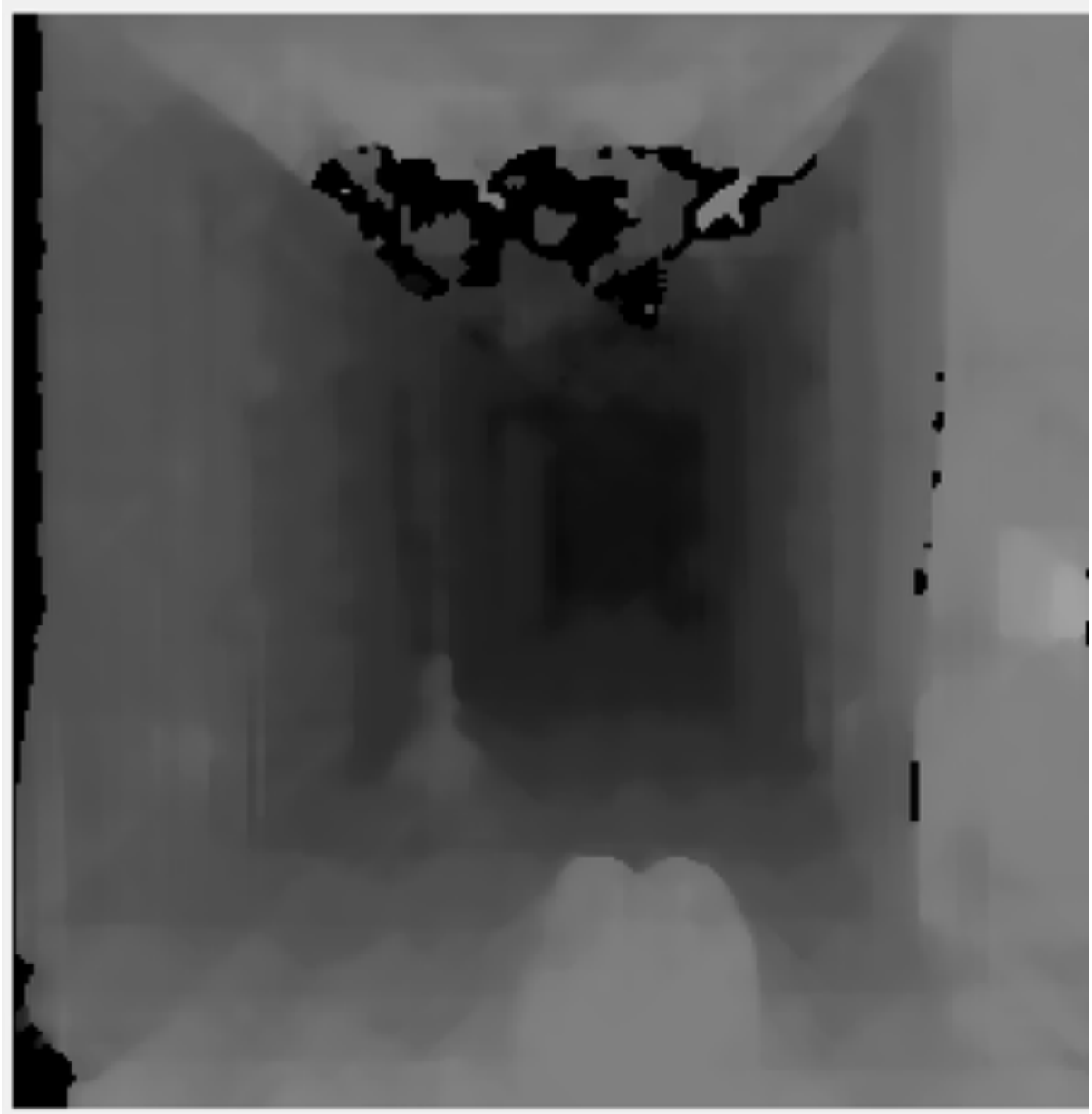
We have two images from two cameras located next to each other, left and right:



To the naked eye they seem equal, but there are minor differences due to the parallax effect. the difference is greater in the closer objects. Similar to closing one eye and then switch to the other eye, you can notice how closer objects seem to shift position.
To evidence this, we can create a composite image of the two images above:



After calculating the disparity using the conveniently called disparity command, we use the disparityMap command and get this:



We can see that the furthest parts of the scene are shown in darker shades of grey.