Maastricht University

HOMEWORK 1

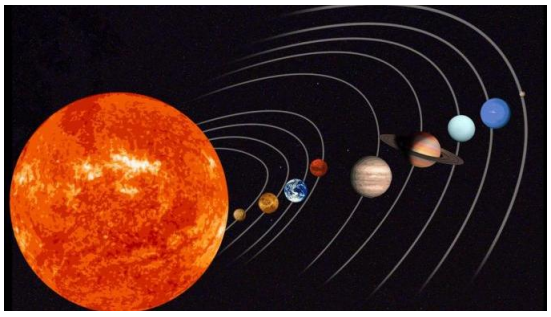
Eliott Simon
i6184618
Department of Data Science and Knowledge Engineering
Introduction to Image and Video Processing

April 19, 2020

1 Images

- Image with high edge-directionality:

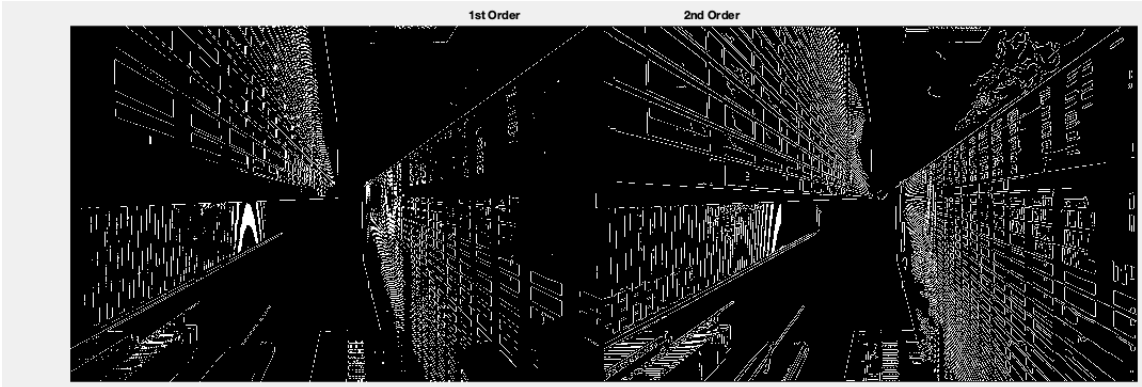
- Image with strong circular symmetry:



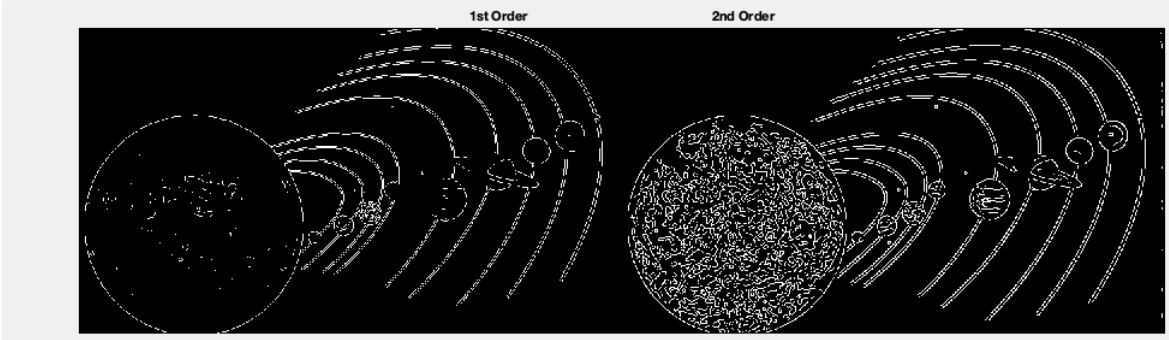
NB: we have one greyscale and one RGB image.

2 Exercise 1

For this exercise we have to apply some gradient edge detection to both images.
I kept it quite simple and used the Matlab formula 'edge' to get the following results:

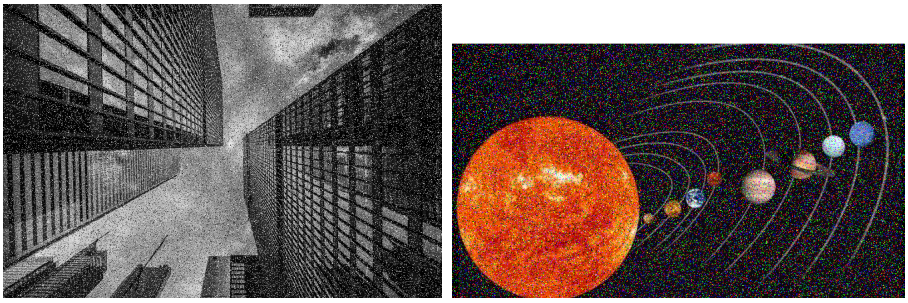


As expected, the 2nd-order (canny) gradient gives better results than first-order(I used Sobel in this case). For example, we can see on the right of both images that there are many more edge points detected in the second one.
For the second image:

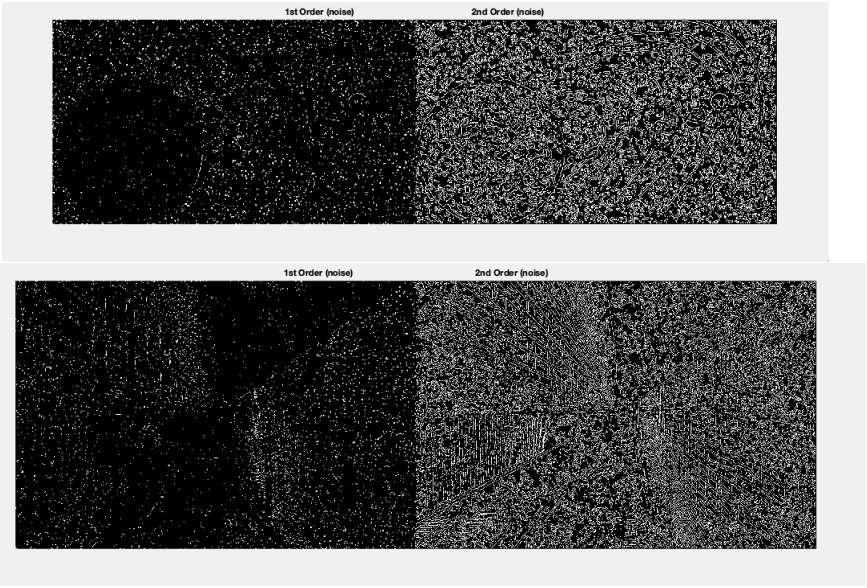


Here we observe the same thing. I used the Prewitt method as first order and we still see that canny gives better results.

Now let's add some noise to our image. I decided to use the salt and pepper noise method, cause the name sounds cool. No seriously because it looked like the best option to test spatial filtering. Our noised image now look like this:

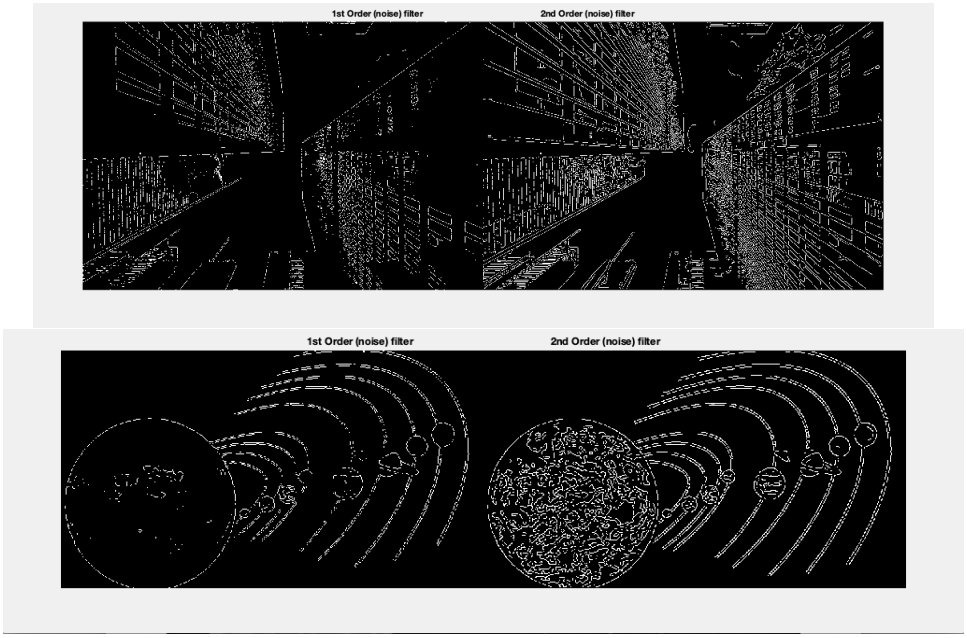


Obviously, when I use the edge detection on these two noisy images I get horrible results such as:



It is quite obvious that we get this nasty result since the noise adds white pixels randomly and pretty much everywhere on the screen (for the first image) and this is very bad for the edge detection since it's going to consider all these pixels as 'edges'.

I used the *median* method to filter the image and get rid of the noise (as much as possible). Although using the median reduces the noise, we still lose a lot of information on the original image.
When I apply the 1st and second order gradient methods AFTER applying the median filter to my images I obtain this:



The results obtained here look very similar to the results computed at first (without noise) which means that the median filter did a great job on reducing the noise on the image.

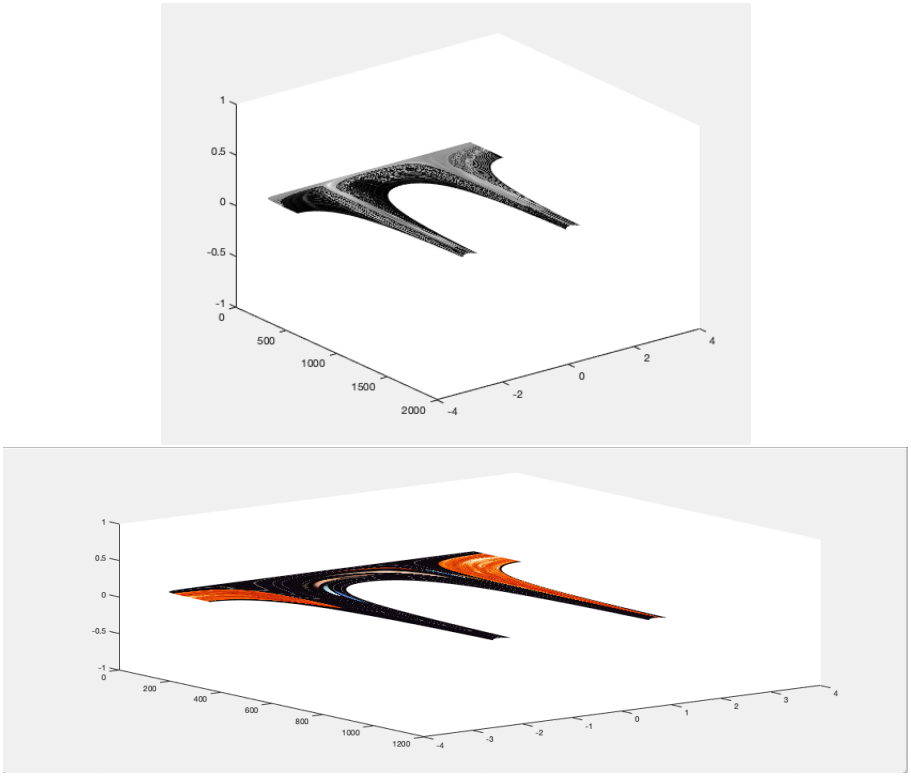
3 Exercise 2

To warp these two images I used the function warp(*polar coordinates*) to get the plot of the warped image. The important steps are:

- Compute the center point of the image, say the (0,0) point of the cartesian plane.
- Next we convert our rows and cols into a single 2D space using the meshgrid() function

- Convert our 2D space from cartesian to polar form using cart2pol() and retrieving our ρ and θ
- Unfortunately, we have to add a Z value to make the function work (I'm not sure why..) thus we have a 3d space with Z = 0, anyway.

As results:



Unfortunately I didn't find a way to visualize the 2D representations of these two plots but from what I see it looks correct, I would like to know if there is a way of doing it though. For the first image, the lines look more straight than the second one which has a very curled shape (duh). When we see circular shapes in the cartesian plane, it would also look circular in the polar plane.

For the last exercise I used this picture:

1) The first (and easiest) image warping that I implemented is a rotation/stretching. I used the A-matrix form and played with its values (see code) to obtain:

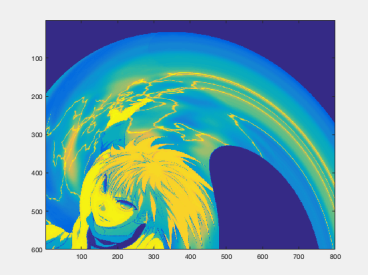
2) The second image warping that I implemented is a swirl effect, for which I made my own implementation. the steps are as follow:

- Choose the swirling constant K. The higher it is the less the less the image is going to be swirled. I set it to 100.
- We have to transform our plane to polar coordinates, which means that we have to find the center of the image first.
- Then we iterate over x and y, and we get a pixel which is at distance K of our point and at angle θ.
- We do this repeatedly, for each x we calculate all the y values using ρ and θ using the cart2pol() function.

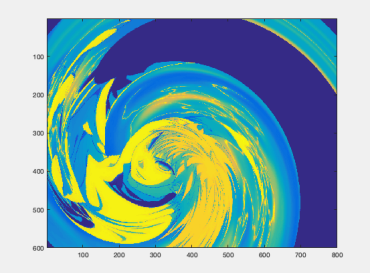
- Once we have our ρ and θ we can recompute our x and y values using pol2cart() to get their new position in the cartesian plane.

I had a bug for 3 days because I would get negative values and value that exceed the image size thus I make sure to only take values from range [1 img.size] and then I take the (heatmap) **imagesc** of the new image to see the results.

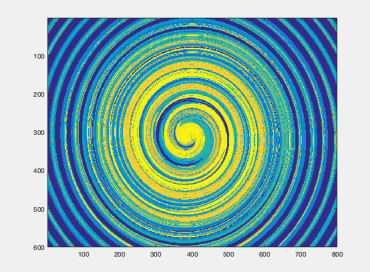
- Constant K = 200



- Constant K = 100



- Constant K = 10



As expected, when K is close to 0, the points are selected very close to the center which means that the spirals are gonna be more 'tight'. I am quite happy about this image warping since it took me a lot of time and I think it's a cool method.