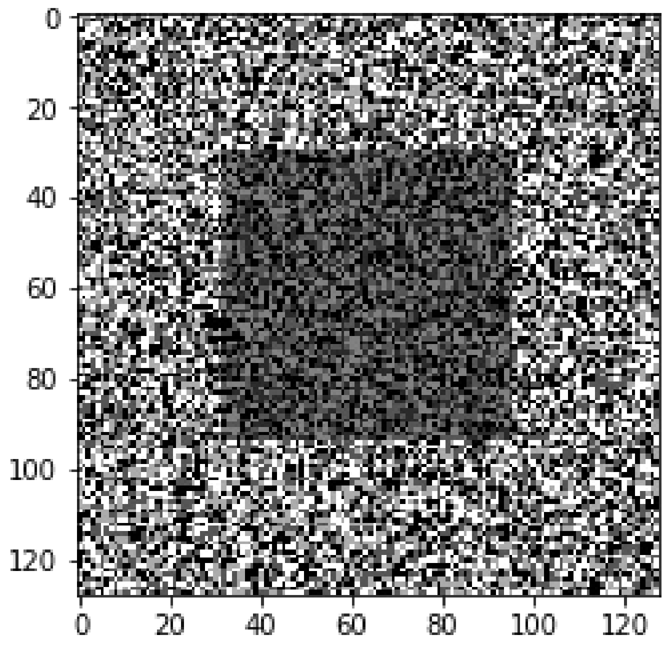
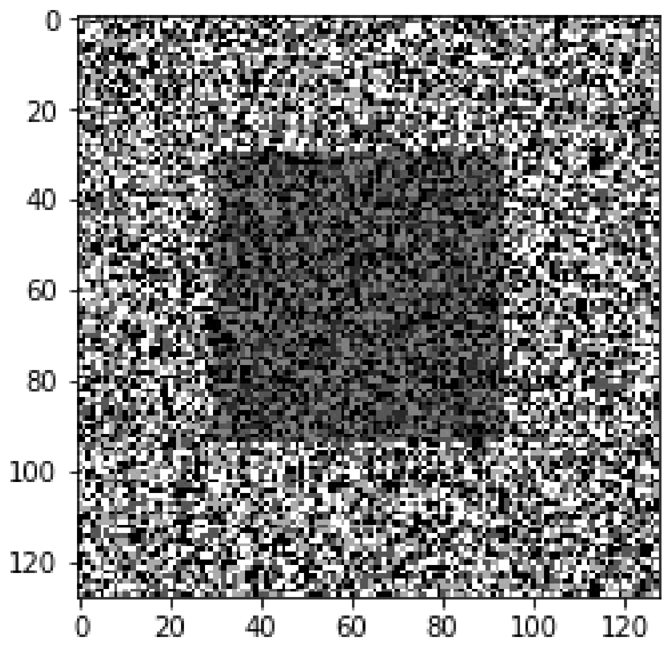
```
# import the libraries
import cv2
import numpy as np
import pylab
from cudamodules import StereoMatchingBasic, StereoMatchingSlow
from tools import PlotTools
```

First we need to load test images

```
img1 = cv2.imread('Data/leftTest.png')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.imread('Data/rightTest.png')
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
PlotTools.display_img(img1)
PlotTools.display_img(img2)
```



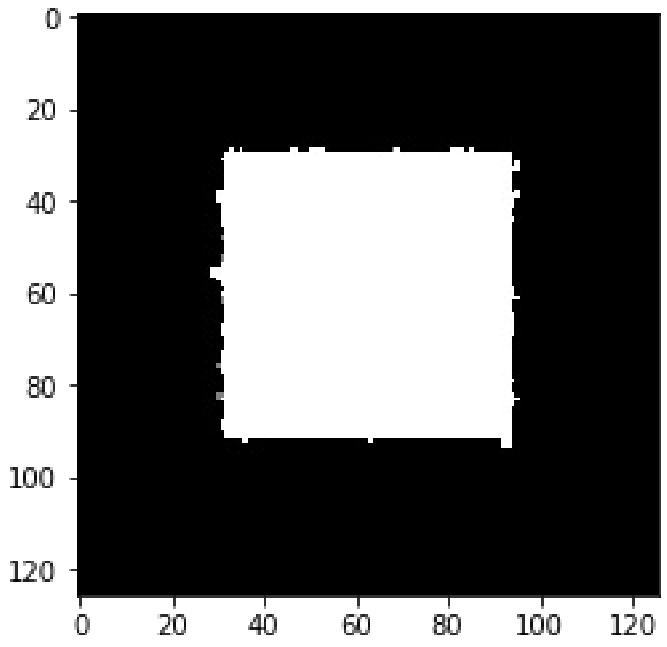


CSS

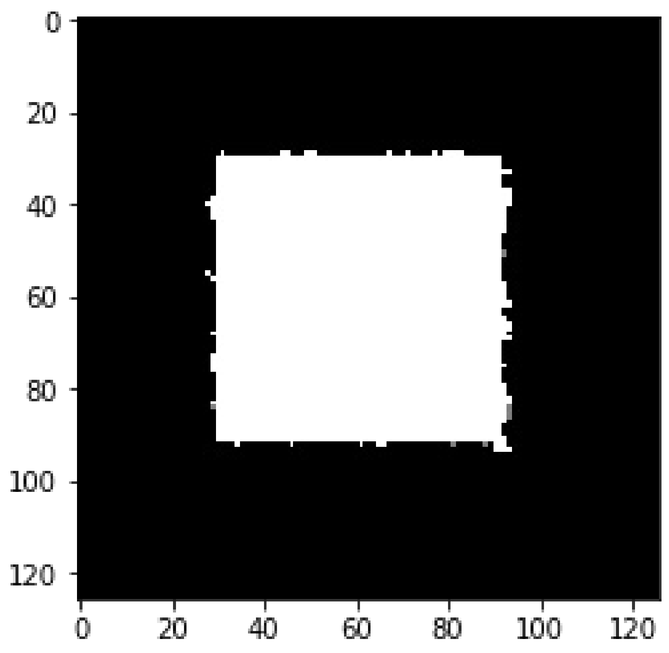
the below images shows the matching with the CSS score function.

Question 1

```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img1,
img2, 3, dmin=0, dmax=2), "PS2-1-a-1")
```



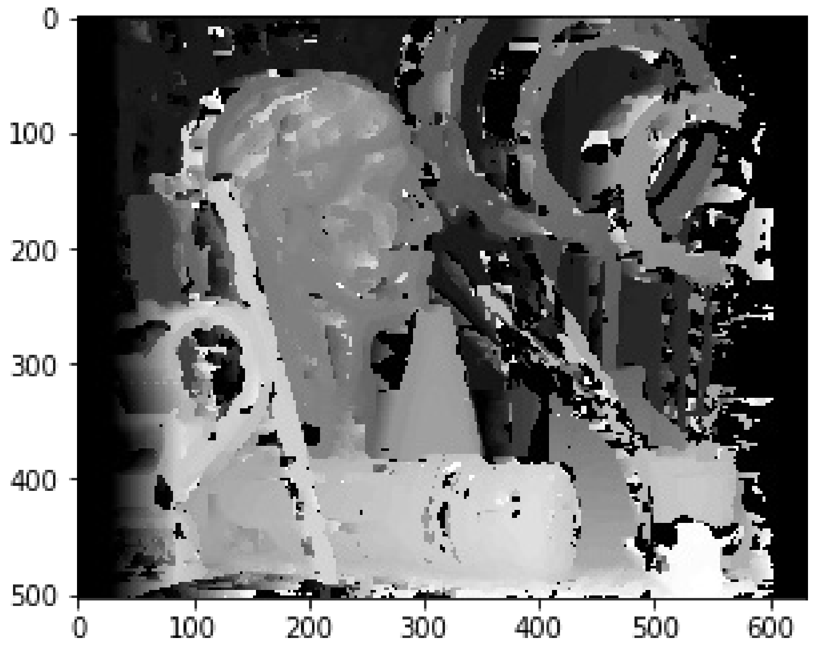
```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img2,
img1, 3, dmin=-2, dmax=0), "PS2-1-a-2")
```



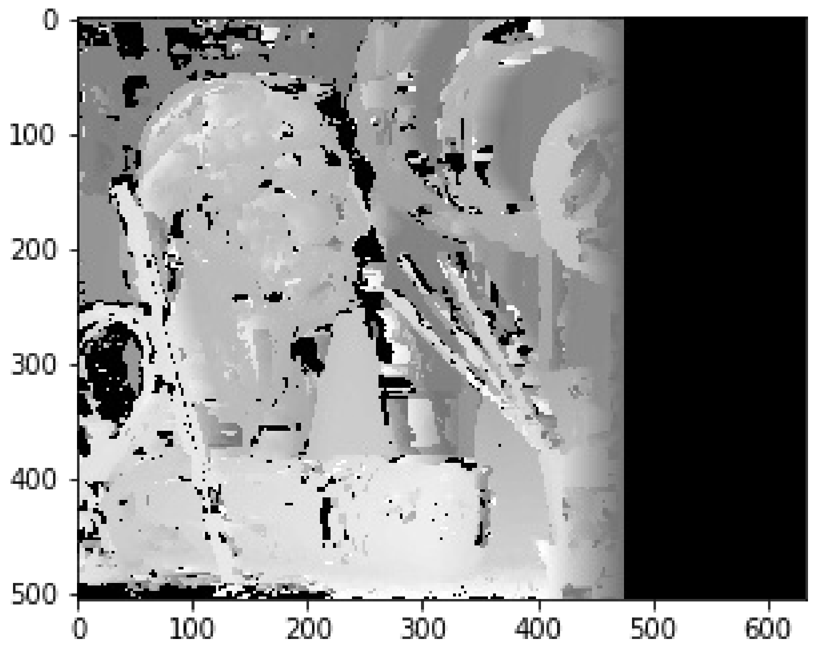
Question 2

```
img1b = cv2.imread('Data/proj2-pair1-L.png')
img1b = cv2.cvtColor(img1b, cv2.COLOR_BGR2GRAY)
img2b = cv2.imread('Data/proj2-pair1-R.png')
img2b = cv2.cvtColor(img2b, cv2.COLOR_BGR2GRAY)
```

```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img1b,
img2b, 9, dmin=30, dmax=100), "PS2-2-a-1")
```



```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img2b,
img1b, 7, dmin=-100, dmax=-30), "PS2-2-a-2")
```



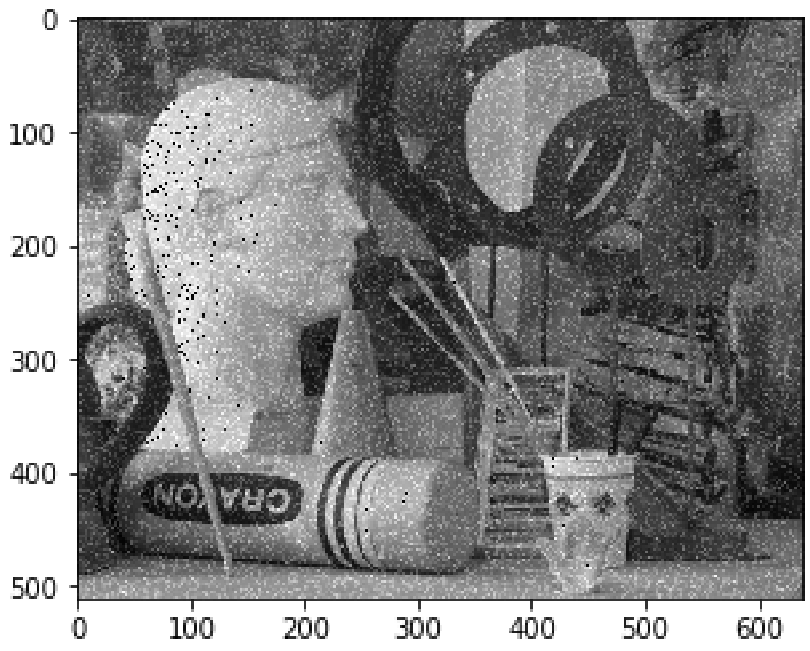
As we can see the images are quite similar to ground results, but it have much noise on it. All the white/black parts corresponds to points that does not have significant matching score withing the given disparity range.

Also we can note that the point at the left/right of the image cannot be matched as there is no such equivalent on the other image.

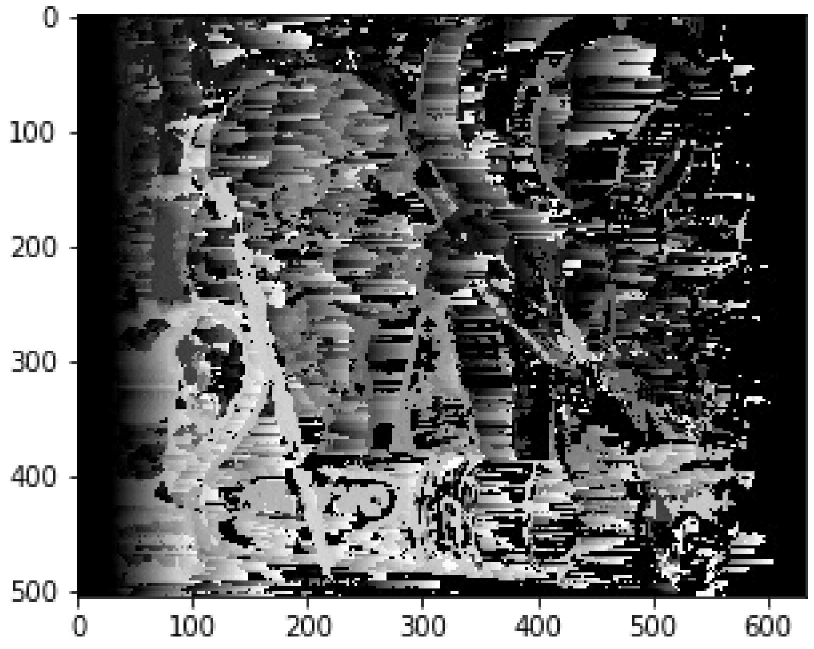
Question 3

```
noise = np.empty_like(img2b)
cv2.randn(noise, 0, 30)
img2b_noised = img2b + noise
pylab.imshow(img2b_noised)
```

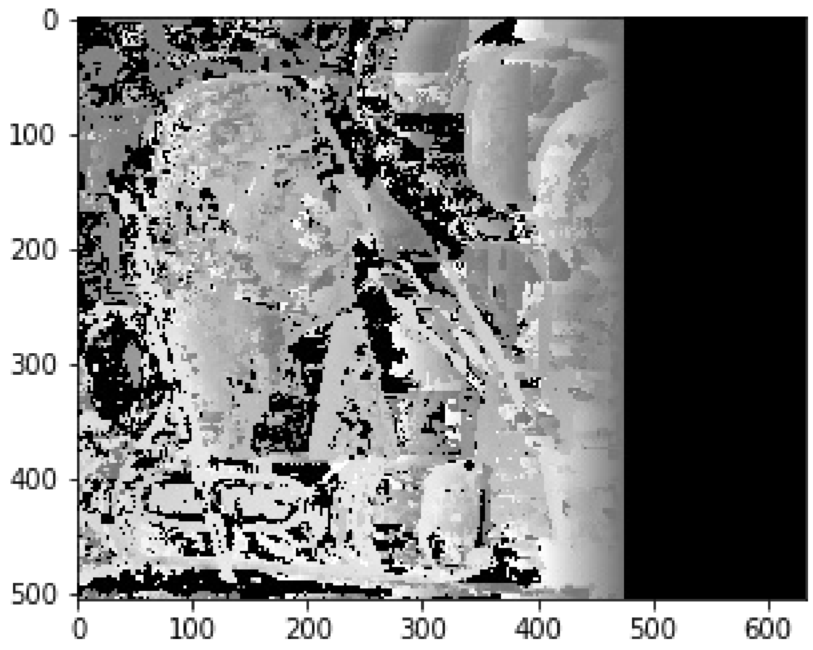
```
<matplotlib.image.AxesImage at 0x7ff0e04d7a50>
```



```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img1b,
img2b_noised, 7, dmin=30, dmax=100), "PS2-3-a-1")
```



```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img2b_nois
ed, img1b, 7, dmin=-100, dmax=-30), "PS2-3-a-2")
```

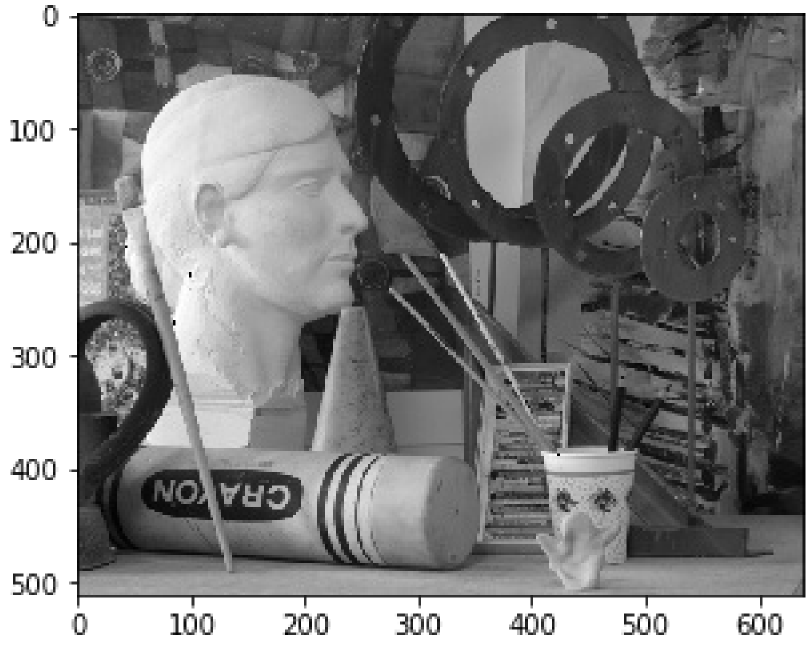


Without any surprise adding noise to inputs, make the algorithm less efficient. We can also note that the flat surfaces a more affected than edge. Also finding an noisy window into a normal image seems to lead to better results than finding a "regular" window into a noisy image. (This is not trivial as this operation is supposed to be commutative) This may due to some algorithmic specificities (especially when finding the minima of the disparities scores).

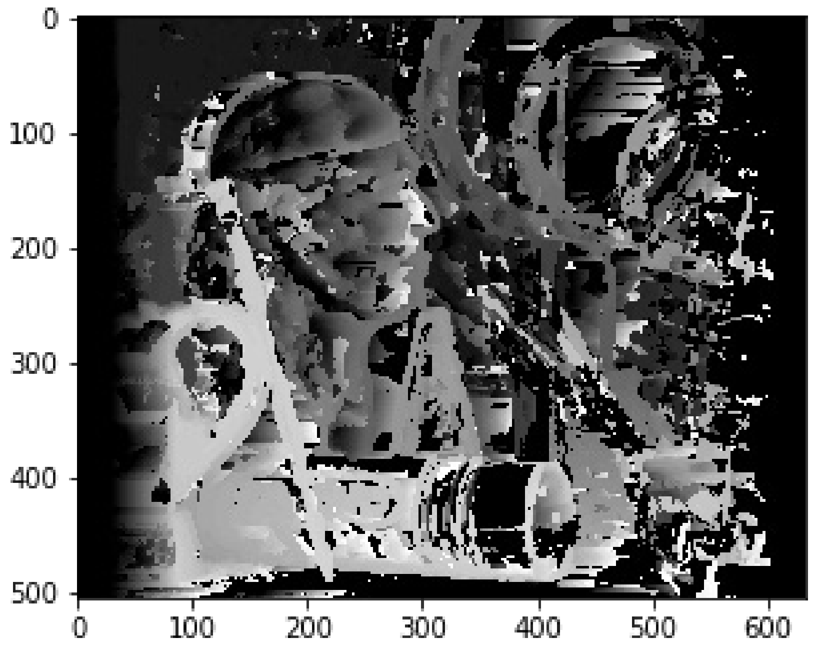
question 3.b

```
img2b_contrast = img2b * 1.1
img2b_contrast = img2b_contrast.astype(np.uint8)
pylab.imshow(img2b_contrast)
```

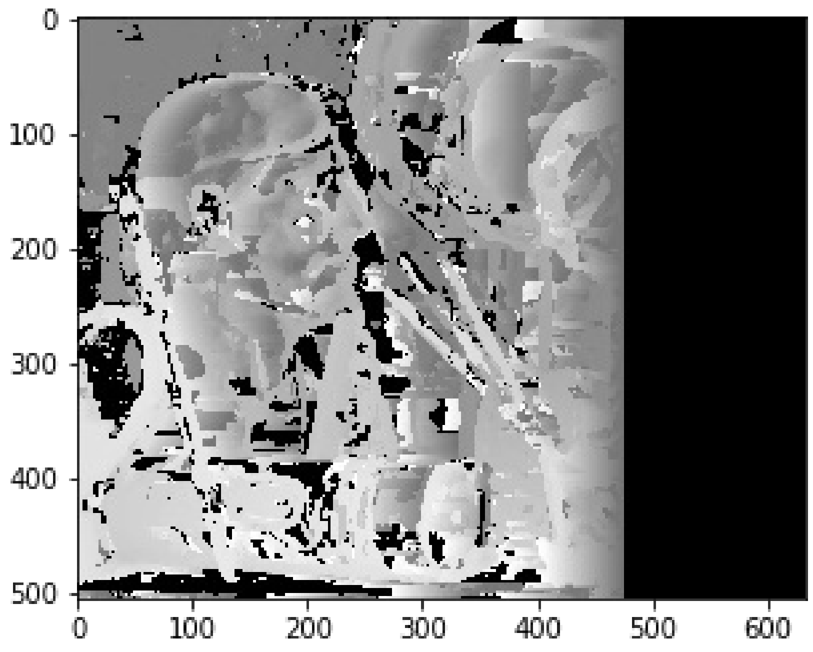
```
<matplotlib.image.AxesImage at 0x7ff0fb77d790>
```



```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img1b,
img2b_contrast, 7, dmin=30, dmax=100), "PS2-3-b-1")
```



```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img2b_cont
rast, img1b, 7, dmin=-100, dmax=-30), "PS2-3-b-2")
```



As seen on the previous images leads to poor results, especially on flat surfaces. Anyway the edges seems to have better matching in this case.

Normalized Cross Correlation

the above images shows the matches with the cross correlation score function.

here is the kernel used to compute the cross correlation, note that the computation has been splitted in two steps (computation of *sa* and *sb* and then computation of *d*). This was done to avoid enventual overflowing on big windows sizes.

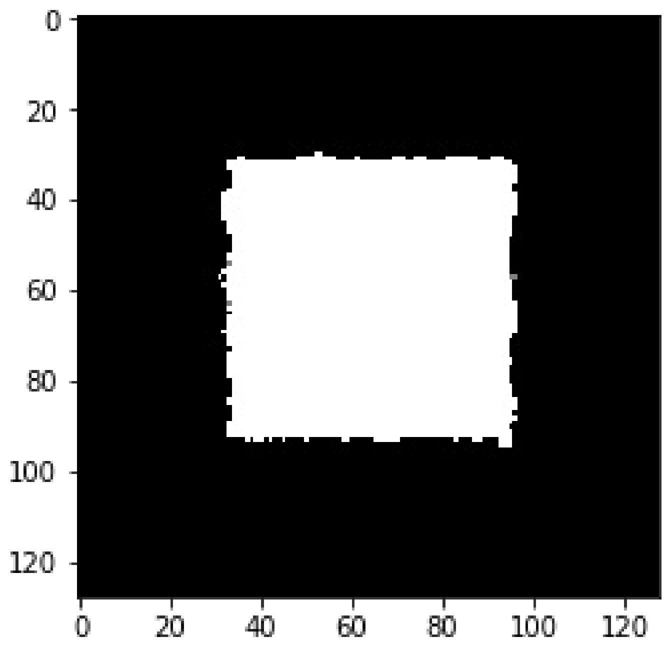
EDIT:

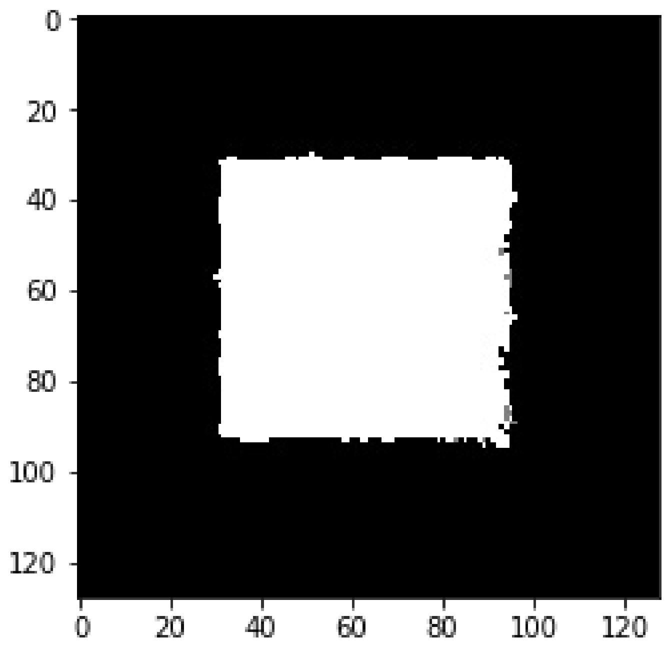
For some obscure reason, this code wont work, i decided to switch to a fully-python algorithm that can use the cv2 function for this

```
kernel = StereoMatchingBasic.KERNEL_CROSS_CORR
print(kernel)
```

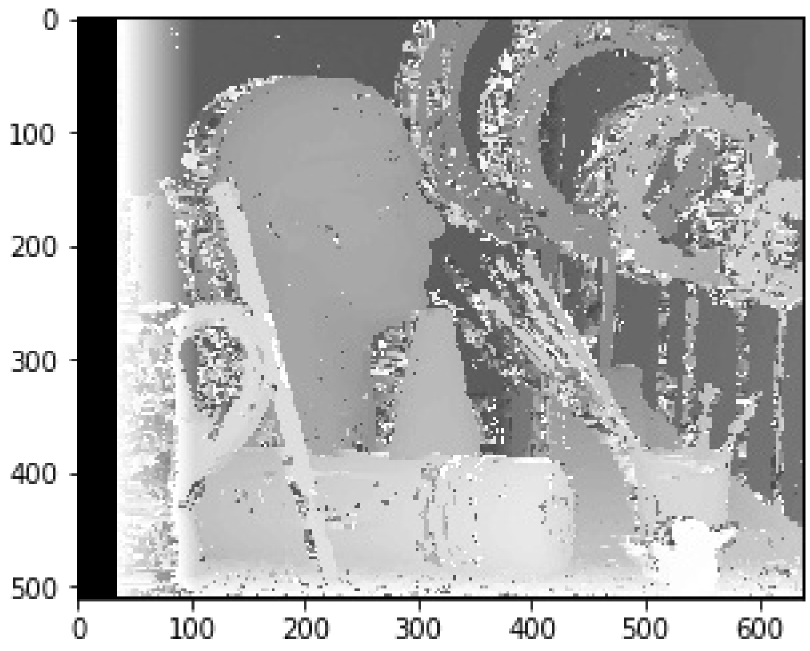
```
register float best;

__shared__ float scores[resdx];
__syncthreads();

int64_t aij;
int64_t bij;
int64_t sa = 0;
int64_t sb = 0;
float d = 0.0;
float div = 0.0;
// split the computation in two steps to avoid overflow
for( i=-wdiam; i<=wdiam; i++){
    for( j=-wdiam; j<=wdiam; j++){
        aij = a[(ay+i)+((ax+j)*imy)];
        bij = b[(bx+i)+((ax+j)*imy)];
        sa = sa + (aij*aij);
        sb = sb + (bij*bij);
    }
}
div = sqrtf((float)sa)*sqrtf((float)sb);
for( i=-wdiam; i<=wdiam; i++){
    for( j=-wdiam; j<=wdiam; j++){
        aij = a[(ay+i)+((ax+j)*imy)];
        bij = b[(bx+i)+((ax+j)*imy)];
        d  = d  + ((aij*bij) / div);
    }
}
scores[tix] =  d;
```

```
PlotTools.display_img(StereoMatchingSlow.cross_corr_match(img1, img2, 5, -2,
0))
PlotTools.display_img(StereoMatchingSlow.cross_corr_match(img2, img1, 5, 0, 2))
```

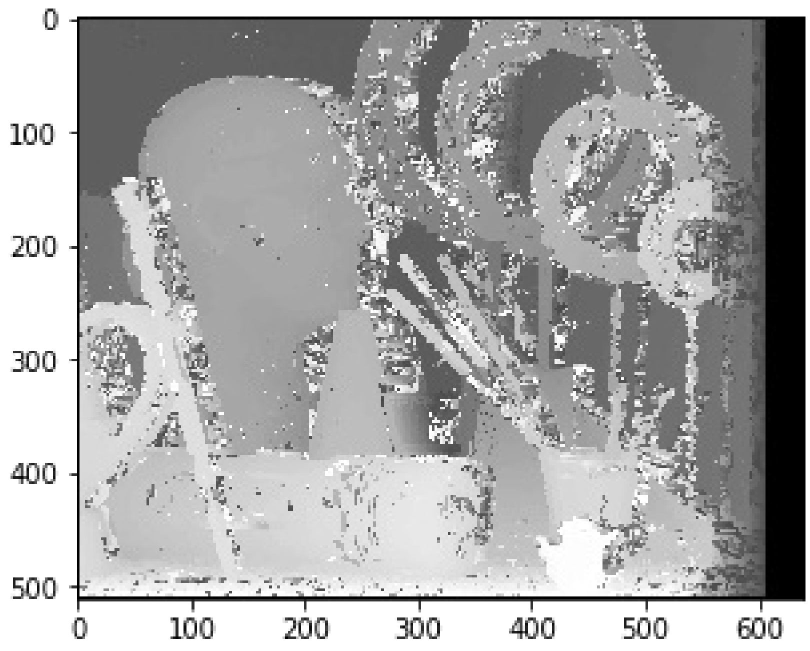




```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img1b,
    img2b,
    7, -100, -30),
    "PS2-4-a-1"
)
```

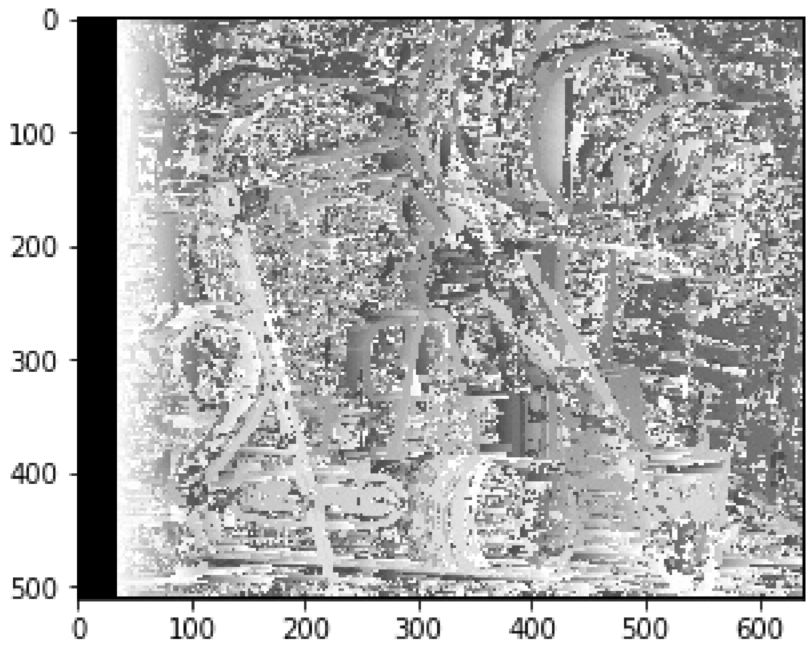


```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img2b,
    img1b,
    7, 30, 100),
    "PS2-4-a-2"
)
```

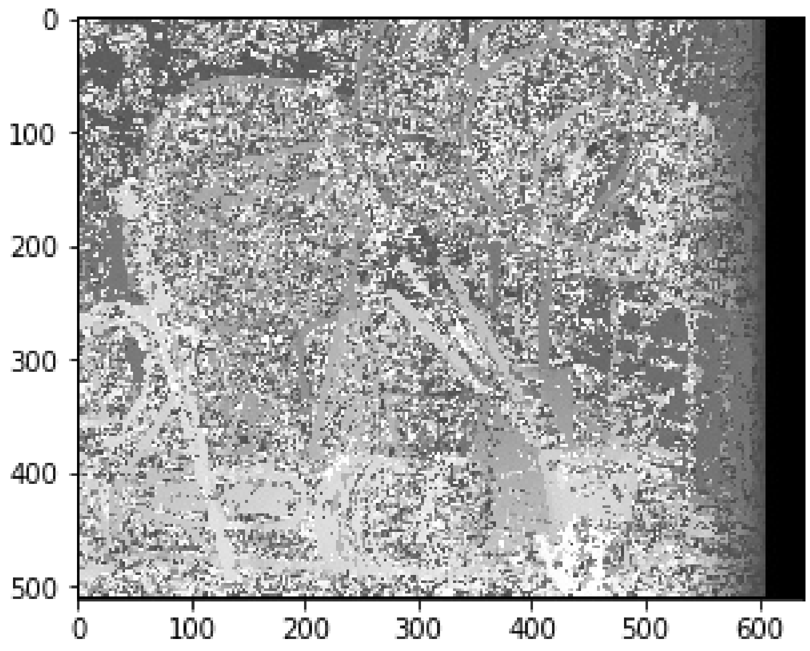


question b: gaussian noise and contrast

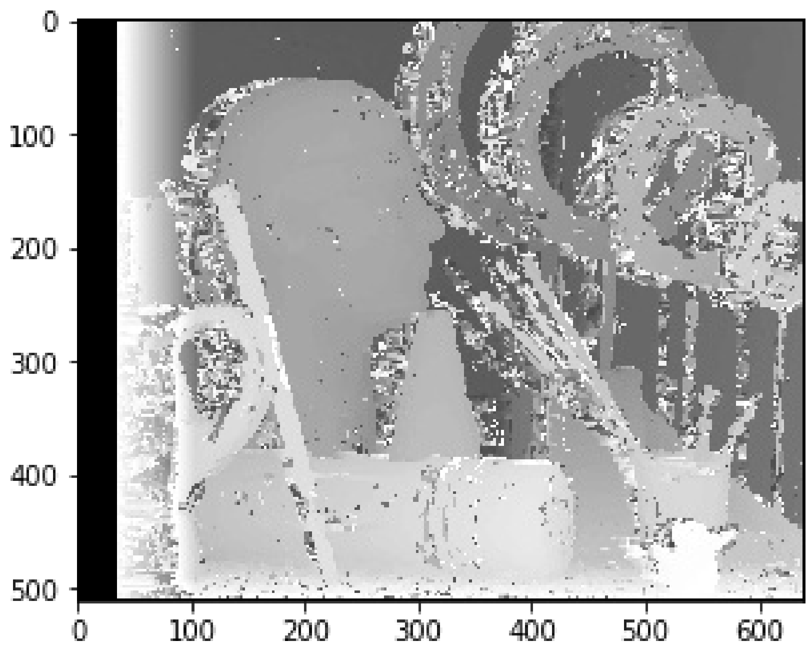
```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img1b,
    img2b_noised,
    7, -100, -30),
    "PS2-4-b-noise-1"
)
```



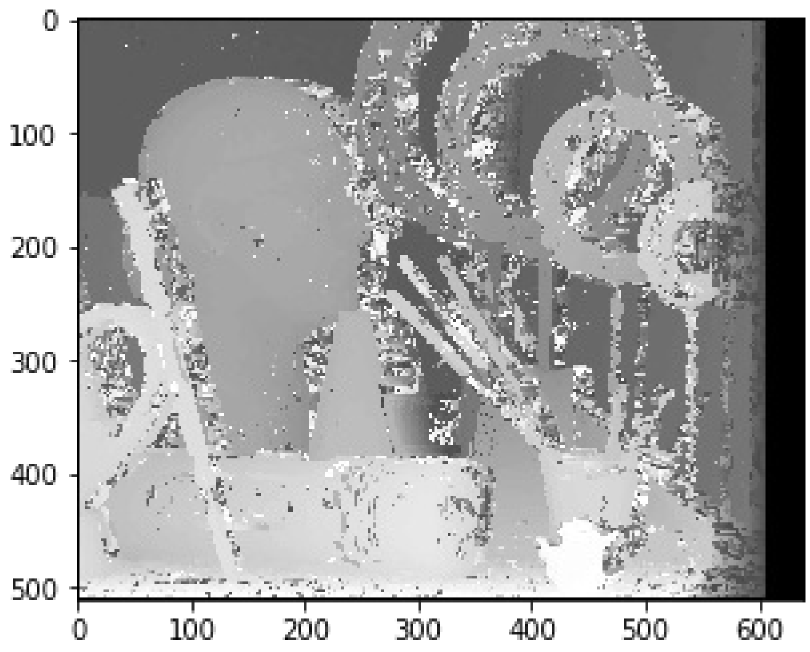
```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img2b_noised,
    img1b,
    7, 30, 100),
    "PS2-4-b-noise-2"
)
```



```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img1b,
    img2b_contrast,
    7, -100, -30),
    "PS2-4-b-contrast-1"
)
```



```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img2b_contrast,
    img1b,
    7, 30, 100),
    "PS2-4-b-contrast-2"
)
```

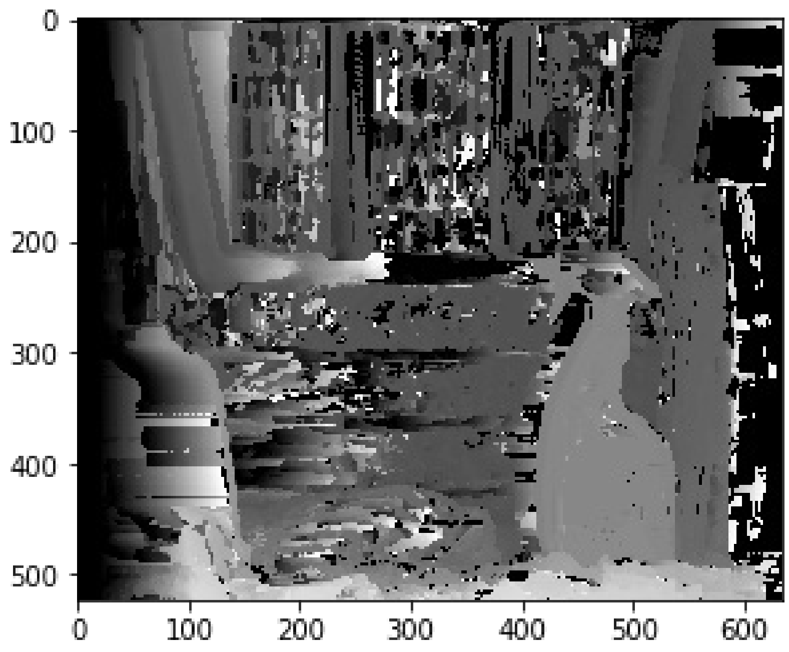


as we can see the cross correlation is very sensitve to noise altough it is not sensitive to variation in luminance/contrast. Applying a gaussian blur can be a great idea with this filter.

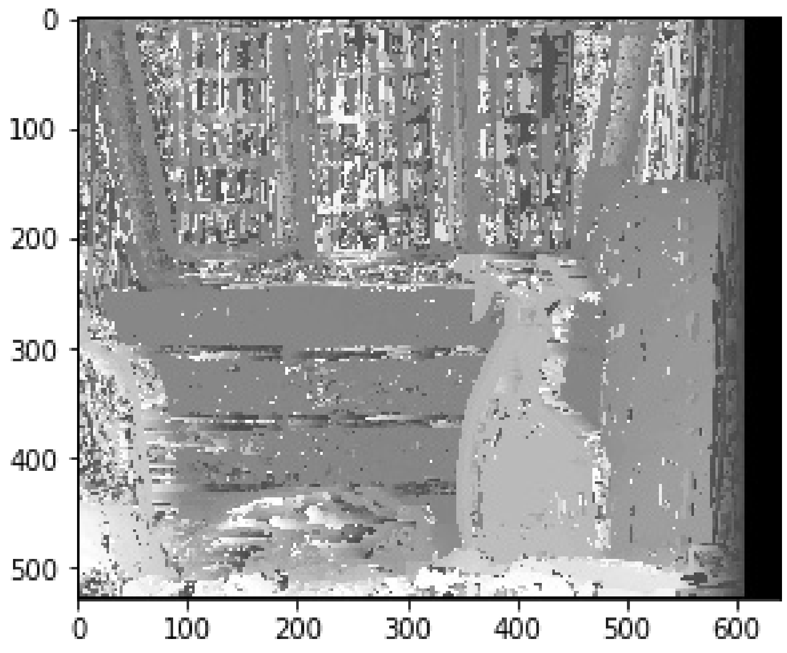
Question 5:

```
img1c = cv2.imread('Data/proj2-pair2-L.png')
img1c = cv2.cvtColor(img1c, cv2.COLOR_BGR2GRAY)
img2c = cv2.imread('Data/proj2-pair2-R.png')
img2c = cv2.cvtColor(img2c, cv2.COLOR_BGR2GRAY)
```

```
PlotTools.display_and_save(StereoMatchingBasic.stereo_matching_basic(img1c,
img2c, 7, dmin=20, dmax=120), "PS2-5-a-css-1")
```



```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img2c,
    img1c,
    7, 30, 100),
    "PS2-5-a-cc-2"
)
```



```
img1c_blur = cv2.GaussianBlur( src=img1c, ksize=(7,7), sigmaX=3, sigmaY=3,
borderType=cv2.BORDER_REFLECT)
img2c_blur = cv2.GaussianBlur( src=img2c, ksize=(7,7), sigmaX=3, sigmaY=3,
borderType=cv2.BORDER_REFLECT)
```

```
PlotTools.display_and_save(StereoMatchingSlow.cross_corr_match(
    img1c_blur,
    img2c_blur,
    7, -100, -30),
    "PS2-5-a-cc-blur"
)
```