# Computer Vision

Alessia Paccagnella

October 3, 2019

## 1 Detection

### 1.1 Image gradients

To compute the gradients with the conv2 function (as requested), I applied at first the gaussian filter with the sigma that I chose (sigma = 2) to blur the image. After that I used the sobel matrixes to compute the two gradients:

```
Ix = conv2(img,opx,'same'); Iy = conv2(img,opy,'same');
```

### 1.2 Local auto-correlation matrix

I had to calculate the autocorrelation matrix to proceed with the harris detector. I applied a gaussian filter (with a chosen sigma) to the neighborhood to compute the matrix for each pixel of the image. Through that I found all the components of the matrixes.

### 1.3 Harris response function

I calculated the Harris response function through the formula studied:

$$R = det(H) - k * Track(H)^2$$

Where

$$det(H) = Ix^2Iy^2 - 2Ix^2Iy^2$$

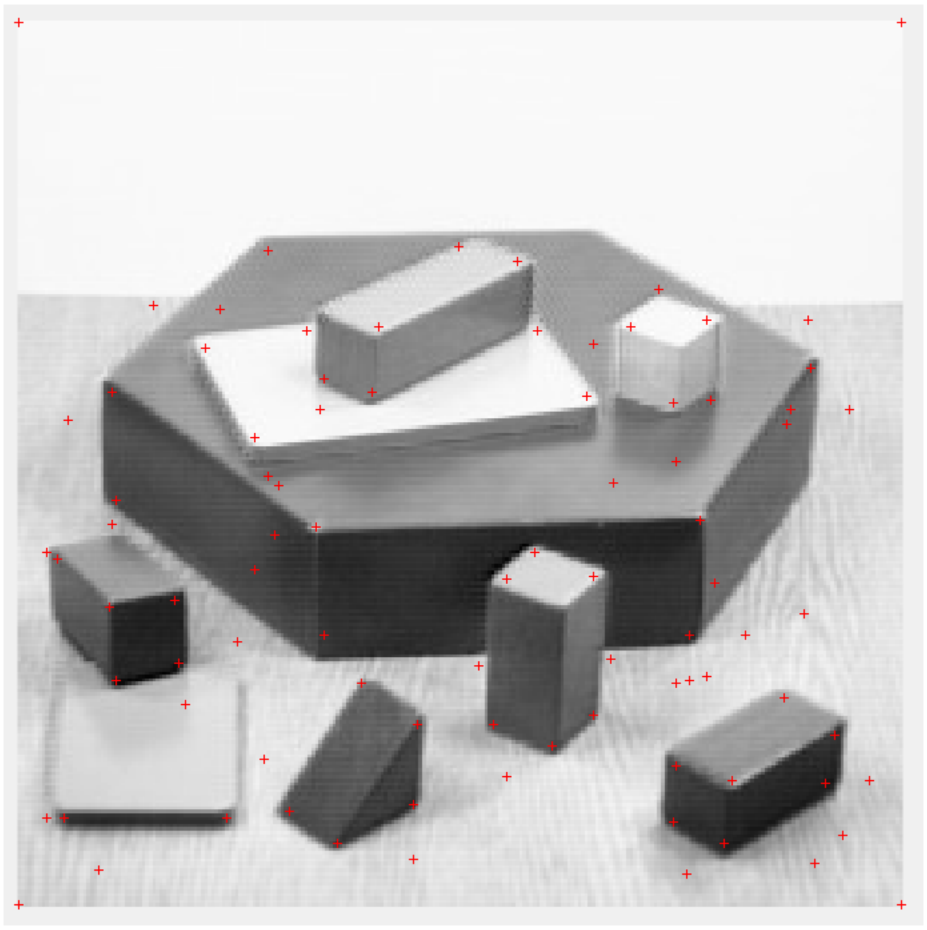
and

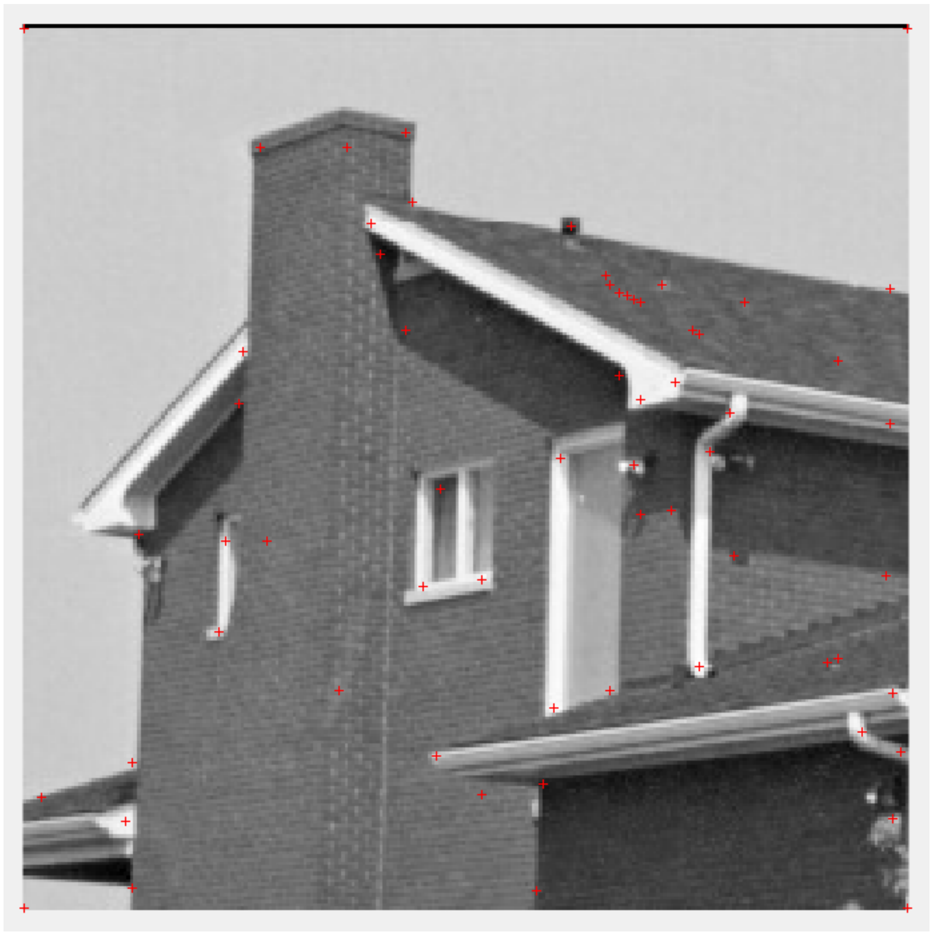
$$Track(H) = Ix^2 + Iy^2$$

## 1.4 Detection criteria

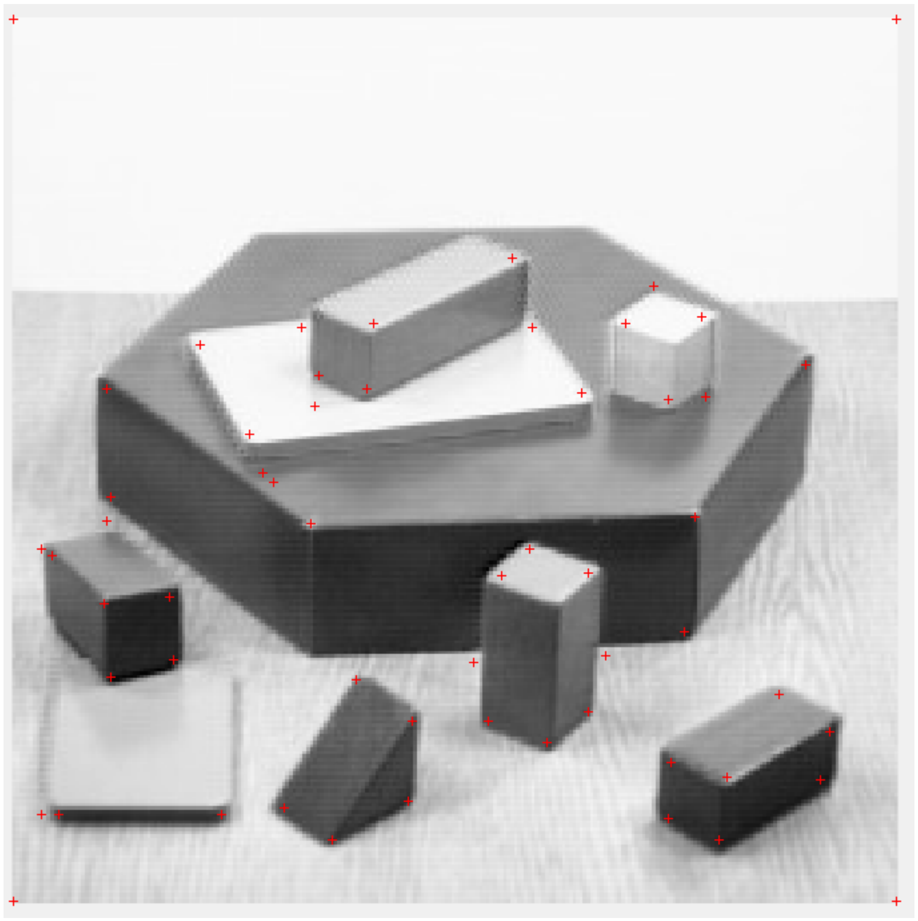
I took account of the detection conditions by thresholding R and maximizing it. The function `imregionalmax` returned me a binary image through which I calculated the new one. This let me find the keypoints positions that I had to return with the function.

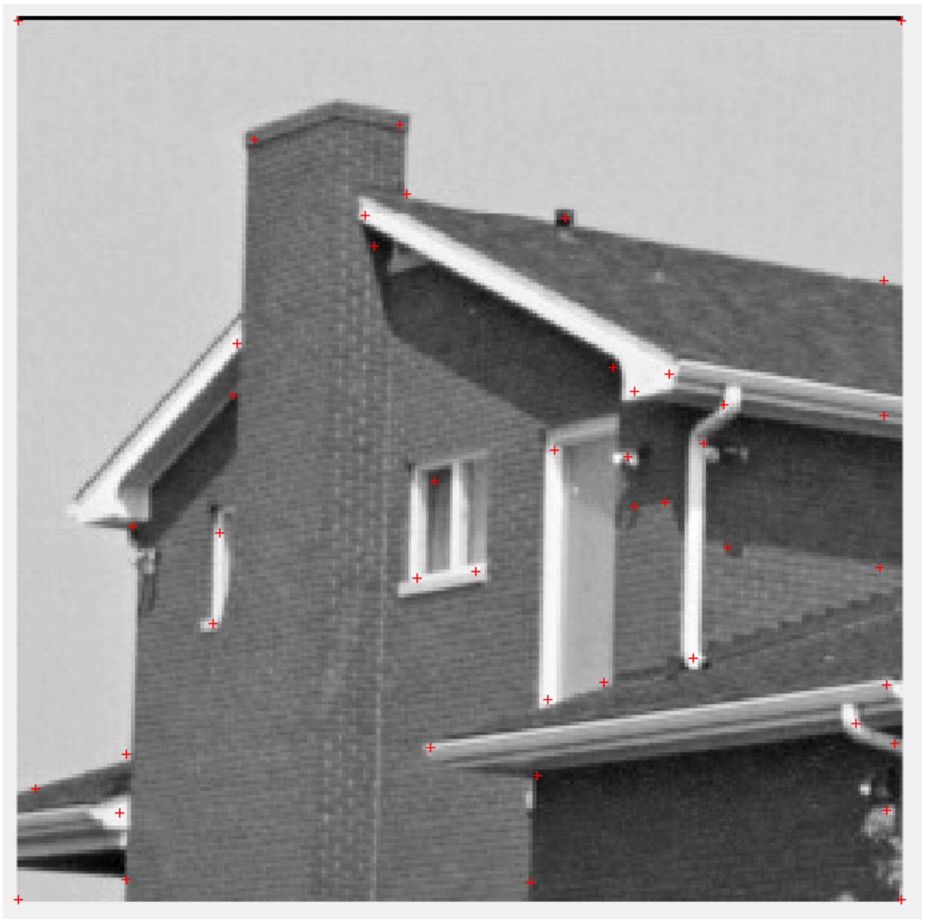
1. By choosing k = 0.06, sigma = 2, thresh = $1x10^{-6}$ :



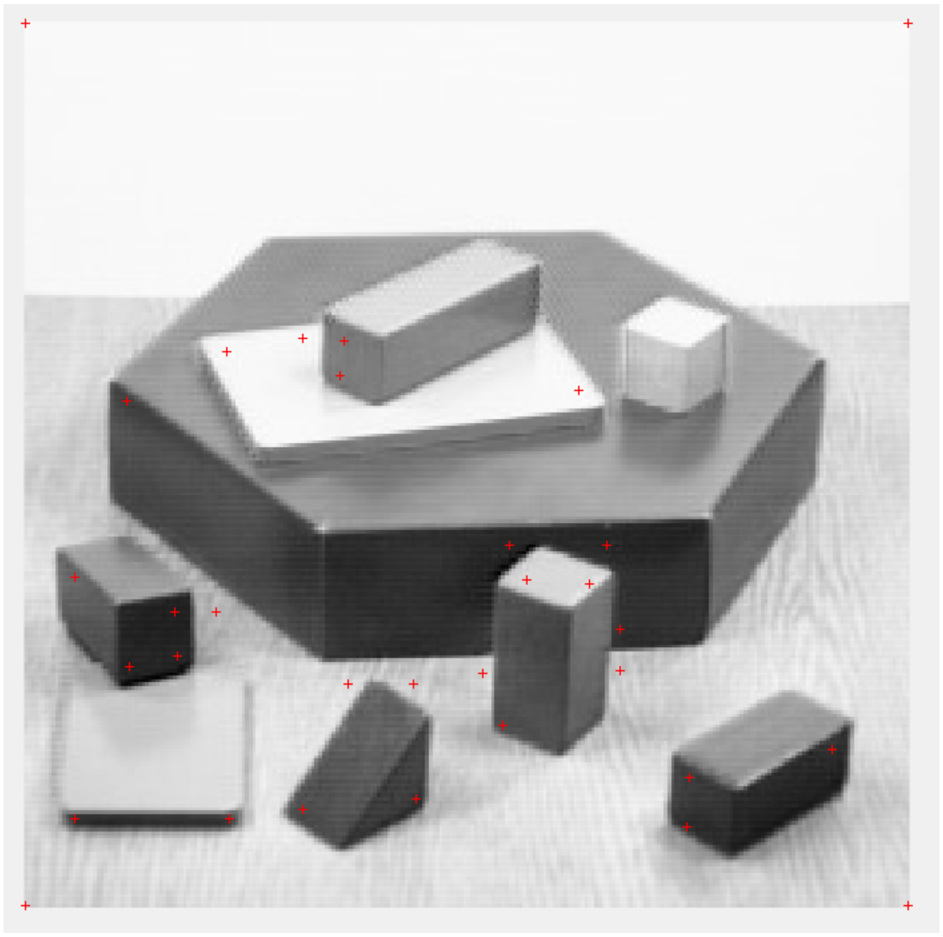


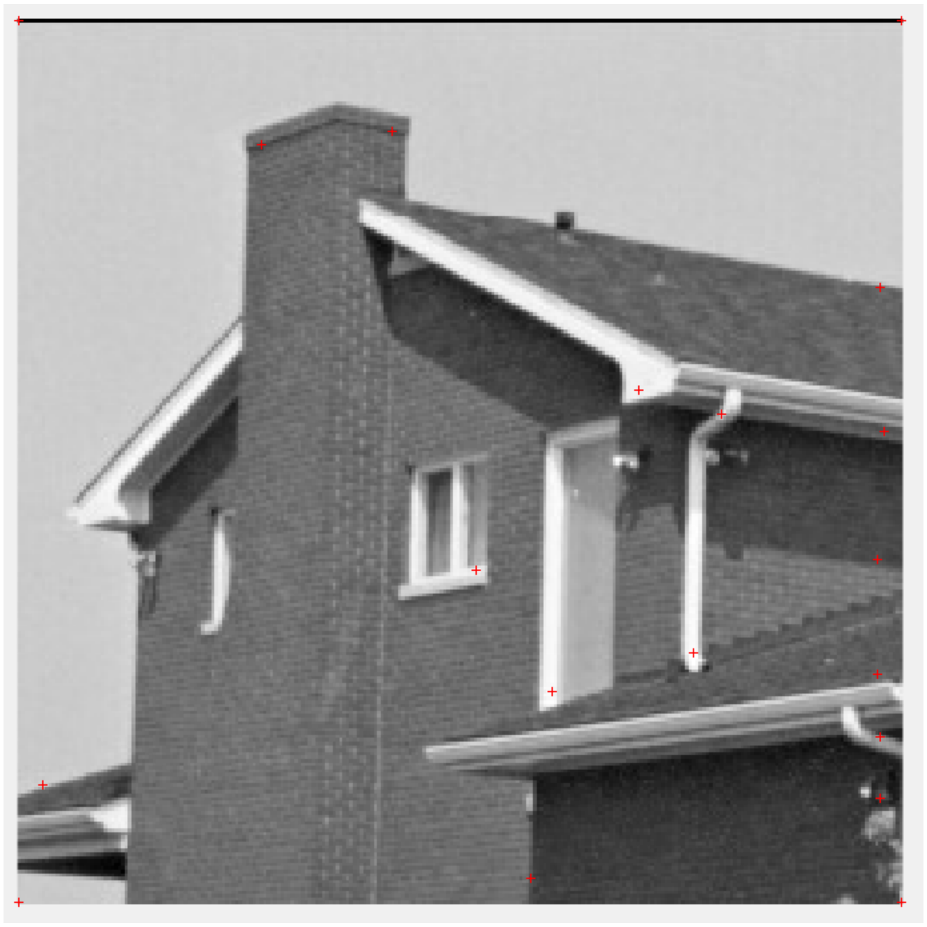
2. By choosing a higher thresh I get less corners (thresh = $1x10^{-4}$) :





3. By choosing a higher sigma (sigma = 5), I get less corners as the image is more blurred.





I noticed that by varying the k value the detection was not changing that much.

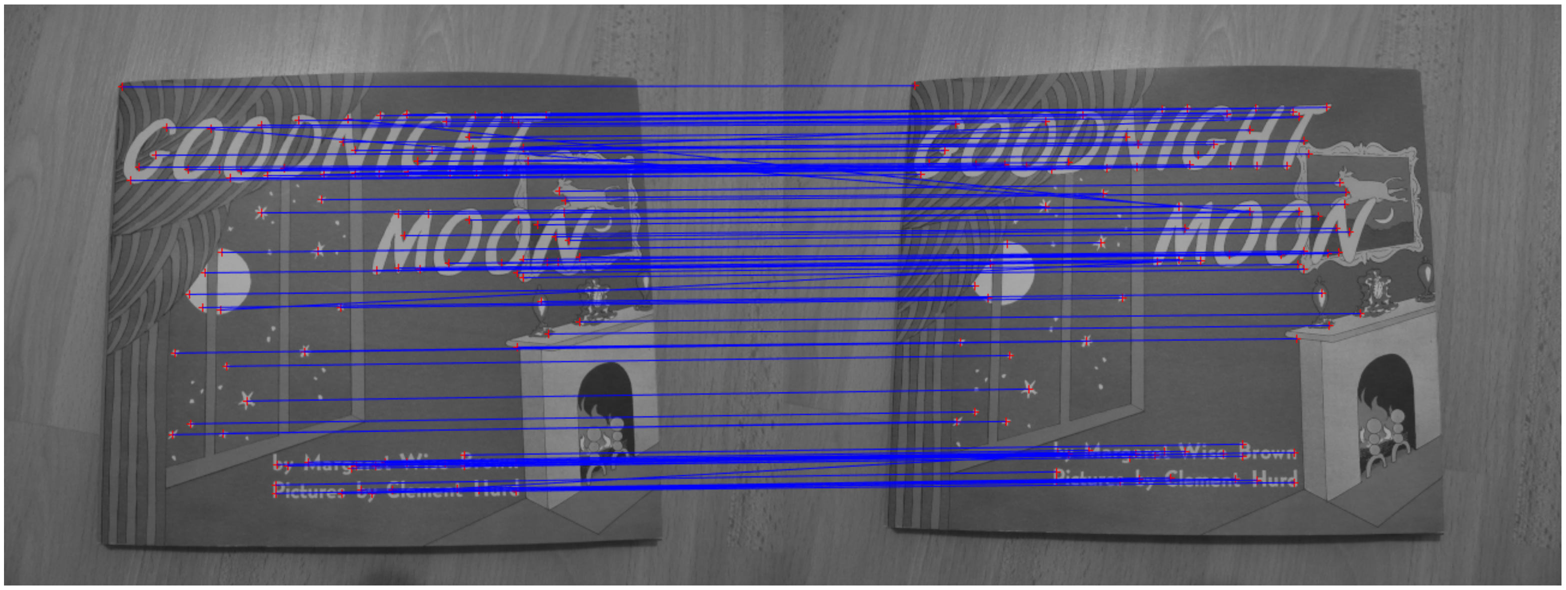
## 2 Description Matching

### 2.1 Local descriptors

At first I found the keypoints that were at shorter distance than 4 from the boundaries and deleted them from the keypoints matrix.
Then i used the function `extractPatches` to get the descriptors, by specifying the dimension of the patch. Use the provided extractPatches function to extract 9x9 patches around the detected keypoints which will be used as descriptor. Make sure to first filter out the keypoints that are too close to the image boundaries so as to avoid out-of-bounds issues.

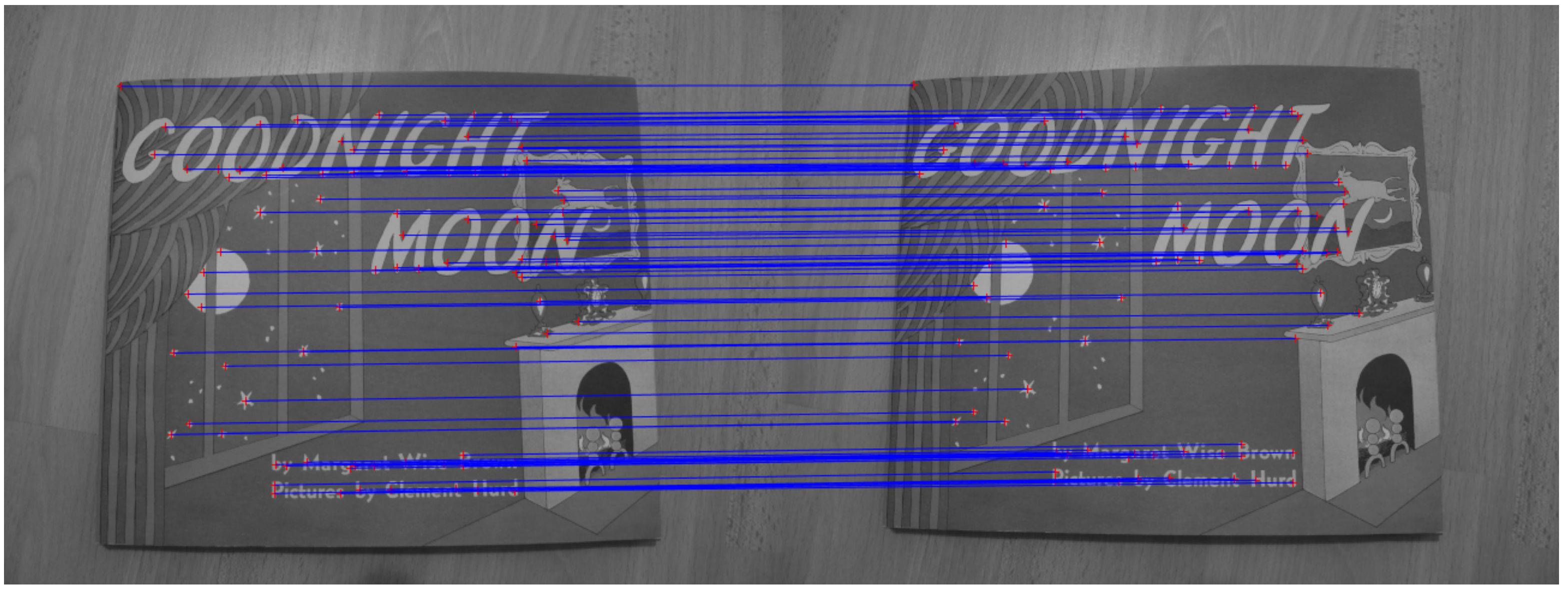
### 2.2 SSD one-way nearest neighbors matching

I calculated the SSD between the descriptors of the first image and the second one through the pdist2 function.
Then, to find the one-way matches I checked, for each of the descriptors for the first image, the one with the minimum ssd from it.



### 2.3 Mutual nearest neighbors / Ratio test

For the mutual case I calculated the ssd from the descriptors of the second image to the first one and I found the intersection between the matches. In this way, I obtained a mapping that was also valid when swapping the images.



For the ratio test instead I repeated the same process as the one-way test, but I also checked that the ratio between the first to minimum ssd was minor than 0.5. If not, I didn't put it in the matches matrix.