1. Introduction

In this exercise we going to implement a Harris corner detector. It is used to extract features from an image and then match these features between two similar images. The two images represent a similar scene but there can be illumination and point of view changes.

2. Extraction of Harris corners

To extract the corners, we begin by applying a slight blur to the image to reduce the effect of noise. Afterwards, we calculate the gradient of intensity at each point using the internal MATLAB command. We are then ready to compute the Harris response at each point. The first implementation that I tried was using a nested for loop. Because of the image size these loops took a long time (a few minutes). I have then replaced the loops by a much more efficient method. First, we precompute the needed gradient matrices:

$$I_x^2, \quad I_y^2, \quad I_x, \quad I_y$$

Then compute the elements of H using a moving sum:

```
H11=movsum(movsum(Ixsq,Wsize,1),Wsize,2);
H12=movsum(movsum(IxIy,Wsize,1),Wsize,2);
H22=movsum(movsum(Iysq,Wsize,1),Wsize,2);
```

Finally compute K elementwise using the definition of trace() and det():

```
K=(H11.*H22-H12.^2)./(H11+H22);
```

This approach is much faster. It takes less than a second.

Sadly, we still need to go in a nested for loop to remove the non-maximum points. For convenience the threshold check is done at the same time. First if the response of a point is less than the threshold it is set to 0 in K. Then if the response is not the maximum of its neighbors it is also set to 0 in K. If the point passes both tests it is added to the list of corners.

3. Extraction of descriptors

In our case a descriptor is a 1x81 vector representing the 9x9 window of pixel intensities around a given point.
Once again, we start by blurring the image. We also pad it with 4 pixels on each side. These pixels are set to the same value as the outermost pixel in their line/column.
For each corner we simply take a small part (the 9x9 window) of the black and white image and reshape it into the descriptor vector. We then return the array of descriptors.

4. Descriptors Matching

We perform steps 1. And 2. On two images the goal now is to match the extracted features between the two images. We do this by comparing each descriptor of the first image to

each one of the second. The comparison is done by the sum of differences squared (SSD). The smaller the SSD the more similar the two descriptors are. We can have two issues: First, not all corners have their match in the two images. Second, a descriptor can appear very similar, but it actually comes from completely different part of the image.

To resolve the first issue we set a threshold and only keep the corners that are matched well-enough (SSD< threshold). Detecting the second issue is more complicated. A good solution is to remove a corner when it has two matches with a similar SSD (+- 20% for example). This means that the corner could be matched to two different places and as such the probability of an error is too big. It is better to not consider this corner. The matches are stored in an array that maps the indices of both corner sets.

5. Effects of different parameters

5.1 Parameters

We have the following parameters to tune the algorithm:

- **Amount of blur**: Increasing the amount of blur reduces the effect of noise but if it is too high we won't be able to detect the corners anymore. The same applies to the descriptors. If the image is too blurry we won't be able to match properly.
  Final value: sigma = 0.5
- **Threshold for corner detection**: The higher this threshold the less corners we are going to consider. This is a tradeoff between quantity and quality. It is better to take more corners at this step to have a better chance of matching them afterwards.
  Final value: 4e-4
- **Threshold for corner matching**: The lower this threshold the harder it is too match two corners, so we will have less matches. This is again tradeoff between quantity and quality. Here it is better to reduce the number of matches to avoid errors.
  Final value: 0.05
- **Threshold for similarity of two matches**: the higher we set this the more a match must be better than all others to be considered. We want to set it to a value where the minimum number of false matches is let trough but not too many good matches are removed.
  Final value: 20%

6. Results of The Harris detector

After tuning the parameters correctly, the final result is reasonably good. We still see some errors, but the percentage of correct matches seems too be good. There are obvious confusions like between stars or letter, but these errors are normal in this context.

Figure 2: Best result

6.2 effect of non-maximum suppression

From the comparison below we see that without non-maximum suppression we get multiple corners in every actual corner. This is not wanted and as such this technique is very useful. Actually, the number of corners without non-maximum suppression is simply too high for the matching step.

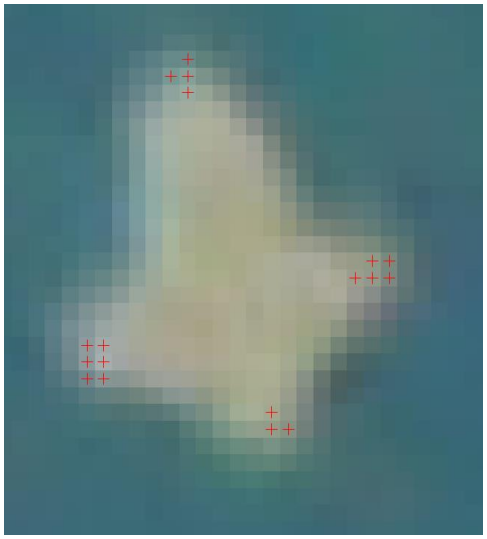
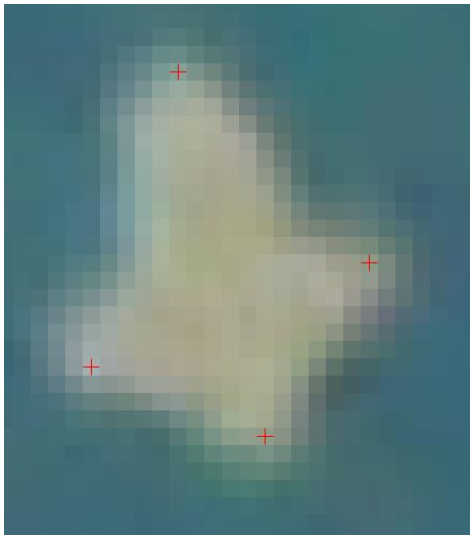


Figure 1: Without non-maximum Suppression



With non-maximum Suppression

6.3 effect of similarity check between matches.

The image below shows the result of not using the technique described above (removing corners that could be matched to two different candidates. We can see that there are more matched corners, but also more errors.

Figure 3: Without Similar SSD check

7. Comparison to SIFT

Using the Vlfeat toolbox we can easily apply the SIFT detector to our image (after some conversion and scaling). There are two parameters that can be tuned: the detection threshold and the edge threshold. I left the second untouched but tuned the first one. The best result was obtained with it being set to 13. We can see that this detector outperforms our previous results with no apparent error. We also see that the location of these features is different from the Harris detector. Very few come from the text and most features come from more unique points such as the clock. It is worth noting that after increasing the threshold we get more matches, but errors appear.

Figure 4: SIFT detector from toolbox