VIP: Assignment 2

Due on Wednessday, December 13th, 2017

Therese Darum, zbl558; Cecilie Novak, bqk662; Christoffer Belhage, DZP300

Contents

Programming language and used libraries

For this assignment `Matlab` and its `Image processing toolbox` have been used. Throughout the assignment code snippets and parts of functions have been reused. We have only reused code that we have written by ourselves.

1: Detecting interest points (Features)

For the assignment we have chosen to use the Harris corner detector. The implementation has been made by Cecilie for her bachelor's project back in 2016. More precisely, two files with code has been borrowed from the bachelor project: *Harris_corner_function.m* and *nms.m*.

Implementation

The Harris corner detector has been implemented in the file *Harris_corner_function.m* using gaussian derivatives for the smoothing of the images, prior to the corner detection.

The computation of the Harris corner measures are implemented as in the original paper on the detector (making use of the constant $k = 0.04$). After computing the Harris measures, non-maxima suppression and thresholding is performed on the Harris measures, such that corners are only detected once and vaque corners, which possibly could be other structures than corners, are thrown away. The non-maximal suppression has been implemented separately in the file *nms.m*.

The threshold used for discarding vague corners has been set as a parameter that specifies how big a measure must be relatively to the maximal Harris measure in the image (i.e., if the threshold parameter is set to $t = 0.5$, only measures bigger than half the maximal Harris measure in the image is kept).

Experimentation and results

We have experimented with the parameter settings for the implemented Harris corner detector, using the following pairs of values:

- t = 0.001, k = 0.04
- t = 0.0001, k = 0.04
- t = 0.00001, k = 0.04
- t = 0.001, k = 0.15
- t = 0.0001, k = 0.15
- t = 0.00001, k = 0.15
- t = 0.001, k = 0.4
- t = 0.0001, k = 0.4
- t = 0.00001, k = 0.4

Out of these, we experienced that the value pair giving the best results were $t = 0.001$, $k = 0.15$. The values for k has been chosen based on the recommended values from the slides and the original paper about the detector. The detection points found using the "best" value pair are shown on top of the images in figure 1-3.

1: Detecting interest points (Features) continued on next page...

In general we experienced that the number of corner points detected went up, when lowering the threshold variable t, while we could only see a slight change in the number of "false" corners when changing the k value.

A lot of the detected corners are correctly detected, but there are also a lot of false corners caused by texture patterns such as the roofs, se for example the edge of the light roof approximately in the middle of the images.

Figure 1: Interest points detected with the implemented Harris corner detector for image *Img001_diffuse_smallgray.png*. Parameters used for the detection was $\sigma = 1$, square kernelsize $sz = 25$, threshold $t = 0.001$ and $k = 0.15$.

Figure 2: Interest points detected with the implemented Harris corner detector for image *Img002_diffuse_smallgray.png*. Parameters used for the detection was $\sigma = 1$, square kernelsize $sz = 25$, threshold $t = 0.001$ and $k = 0.15$.

Figure 3: Interest points detected with the implemented Harris corner detector for image *Img003_diffuse_smallgray.png*. Parameters used for the detection was $\sigma = 1$, square kernelsize $sz = 25$, threshold $t = 0.001$ and $k = 0.15$.

2: Simple matching of features

Implementation

Extracting points
For each detected point of interest we must extract information to be used as a descriptor. The most simple extraction is placing a square $N \times N$ window centered over the detected pixel and using every pixel within the window as a descriptor. We pad the image with N zeros, in case we wish to extract a descriptor from a pixel less than $\lfloor \frac{N}{2} \rfloor$ pixels from the border. Then for every extracted image patch we reshape it into a $N \cdot N$ vector. Every vector is then collected and placed into a matrix.

Matching
The matching was performed by calculating the difference between all the descriptors across the two images, using sum of squared intensity difference:

$$SSID(u, v) = \sum_{i=1}^{n} (u_i - v_i)^2$$

Then for each descriptor in the first image, the descriptor from the second image with the smallest difference is logged as the "match". Just doing this produced a lot of unwanted matches, so we needed to filter the matches. Besides, multiple descriptors from the first image might match a single descriptor from the second image, which naturally does not make sense.

The first filtering applied is checking whether the second best match for any given descriptor is within some scale of the same difference to the best match. So, if the second best match is not far worse than the best match, then we consider the matches too ambiguous and discard it entirely.This reduced the number of matches, depending on what scale was used to determine "too ambigous".

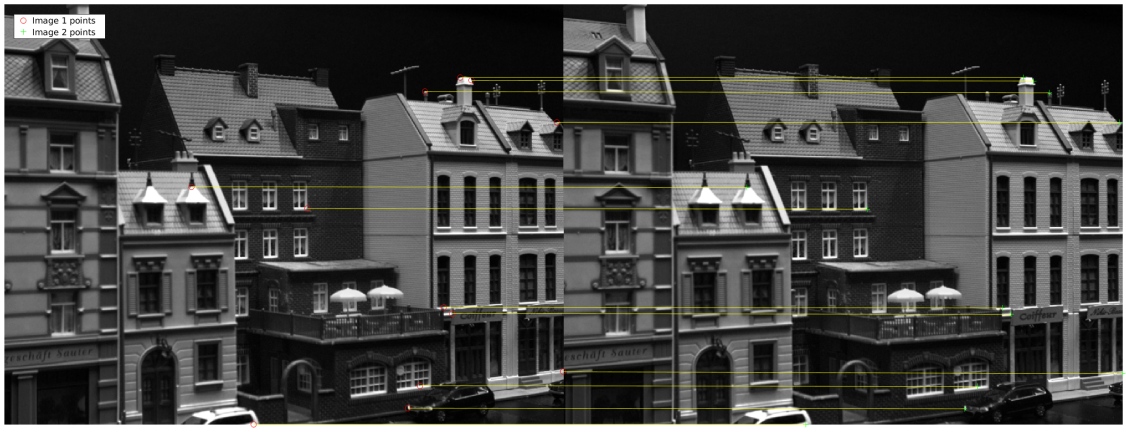
The next filter added is doing the exact same as described above, but for all descriptors in the second image compared to the descriptors in the first image. If a descriptor from the first image points to a descriptor from the second image, which in turn points to the same descriptor in the first image, then it is kept as a match, else discarded. That it, descriptors must match two-way.

Lastly, as a variable parameter, a hard threshold for how similar the matches should be before we keep them. This might be a less useful filter than the previous two; The statistics will show whether this is the case.

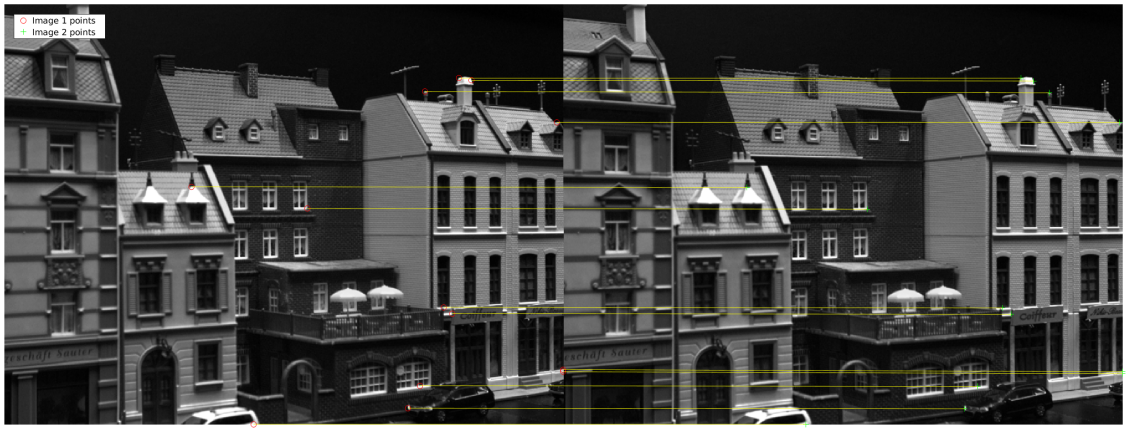
2: Simple matching of features continued on next page. . .

Experimentation and results

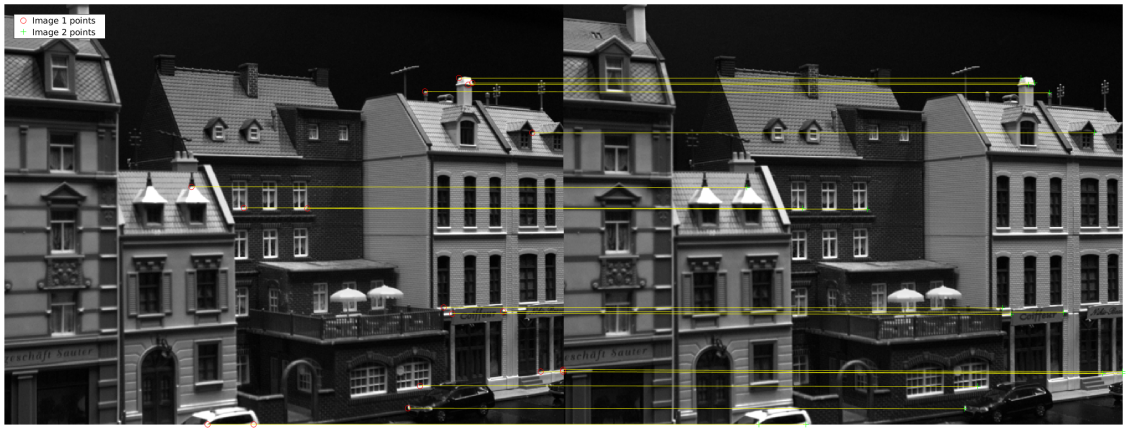
Matchings



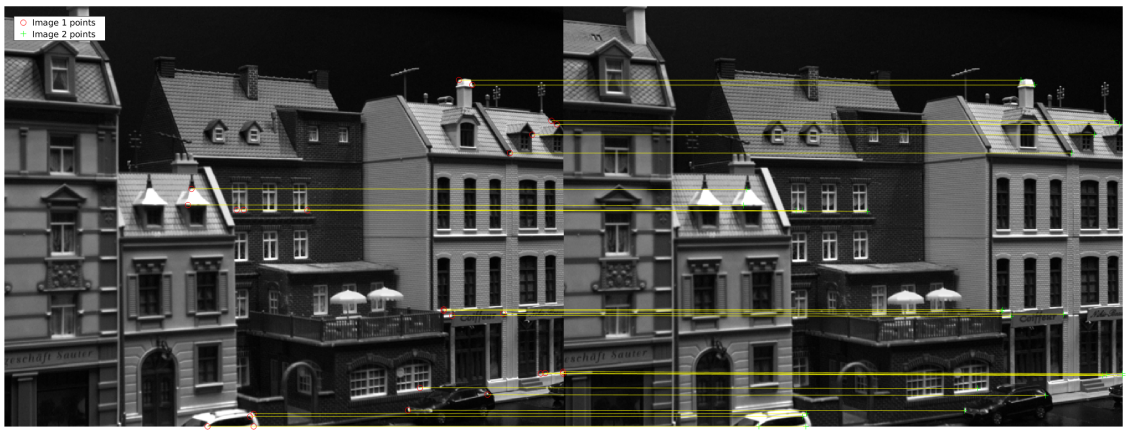
(a) Matches using a descriptor field of width and height 5.



(b) Matches using a descriptor field of width and height 7.



(c) Matches using a descriptor field of width and height 9.



(d) Matches using a descriptor field of width and height 13.

Figure 4: Matches found in the used test images. The parameters for the corner detection was as for figure 1 - 3.

Looking at the matches produced for descriptor sizes 5 and 7 in figure 4, it seems that many matches have been filtered out. We will be able to see which of our 3 filtering methods caused most of this effect in the statistics section.

In no of the figures showing the matches, we have seen tilted lines. The tilted lines would have indicated wrong matchings, which we would also have seen by visually inspecting the positions of the interest points. All of the lines are fairly parallel, which is a good sign, because the lines should be parallel to each other, assuming all the matchings are correct.

Lastly, there seem to be a lot more matches being accepted for when using descriptor fields of size 9 and 13. We will look at what the reason for this is in the statistics section below.

Statistics

In table 1 we see the statistics of our matching. We made 176144 comparisons in total, and we had 389 and 438 interest points in image 1 and 2 respectively. We can see that the first filtering we do reduced the

number of possible correspondences significantly.

We discard more points for small N because we only have a little neighborhood. The "pattern" of the neighborhood is more likely to be seen in other parts of the image, whereas larger neighborhoods are more "complex" and thus "rarer".

The number of potential matches is reduced even further, when we check that a match from point y to point x is two-sided. Again we see that for the larger N we have more potential matches than for small N. This is natural since the more potential matches we have, the higher the chance of them being true matches.

As we theorized the final thresholding does not have a noteworthy effect. The hard threshold we used was 100000. The thresholding seems to have a larger effect on large N and practically none on the smaller. This is because the sum of squared intensity differences of the smaller window sizes cannot sum up to the threshold, unless the best matching descriptors are still very different, in which case they should have been discarded during the first filtering. Therefore the hard threshold should probably have been chosen based on N.

N	5	7	9	13
Number of interest points, image 1	202	202	202	202
Number of interest points, image 2	218	218	218	218
Number of correspondences after filtering, image 1	26	24	26	34
Number of correspondences after filtering, image 2	24	22	26	29
Number of two-way accepted correspondences	12	13	18	26
Number of correspondences after thresholding	12	13	18	26
Worst possible dissimilarity between patches	1625625	3186225	5267025	10989225
Average dissimilarity	1877.5	5560	11256	24640
Standard deviation	1751.1	5102	9752.7	19287

Table 1: Statistics of matching

We would like to have a small average dissimilarity, since small dissimilarity means that two points are a good match. We would also like for the standard deviation to be small, since we would like for all of our matchings to be good. A large standard deviation could mean that we have a few exceptionally good matchings or a few exceptionally bad matchings. From the means and standard deviations we see in table 1 we can see that we do not get consistently good/bad matches. However the matches we do get seems to be very good, if we take into consideration the worst possible dissimilarity between two patches, calculated by 255^2N^2. Based on worst case scenario, average dissimilarity, standard deviation and number of accepted matches, it would seem that a very large patch sizes yield the best true case to worst case ratio. This is again because large patches are more difficult to find at other image positions.