Harris Corner Detection and Keypoint Tracking

Contents

1 Preliminaries **1**
- 1.1 Outline of the exercise . . . 1
- 1.2 Provided code . . . 2
- 1.3 Conventions . . . 2

2 Part 1: Calculate Corner Response Functions **2**

3 Part 2: Select keypoints **3**

4 Part 3: Describe keypoints **4**

5 Part 4 and 5: Match descriptors **4**
- 5.1 Matching quality . . . 5
- 5.2 Optimizing your code the smart way - profiling . . . 5

6 Numerical Exercises **6**

The goal of this laboratory session is to get you familiarized with feature detection, description and matching.

1 Preliminaries

1.1 Outline of the exercise

In this exercise, you will implement Harris corner detection and keypoint tracking, by only using the knowledge you have so far obtained in the class. You will run this keypoint tracker on the first 200 frames of the popular KITTI dataset.



Figure 1: Matching features from the first two frames in the KITTI dataset.

You will achieve this with the following steps: First, you will evaluate the Harris score for each pixel of the input image. Then, you will select keypoints based on the Harris scores. In a next step, you will evaluate simple image patch descriptors at the selected keypoint locations. Finally, you will match these descriptors using the sum of squared differences (SSD) in order to find feature correspondences between frames.

1.2 Provided code

We provide you with skeletal Matlab code (`main.m`) and Python code (`main.py`) which loads the images for you, provides you with good initial parameters, and does the plotting for you. Your job will be to implement the code that does the actual logic. We also provide the functions stubs with some comments about the input and output formats, so if these are not clear from this pdf, they should be clear from the function stubs. *You do not need to reproduce the reference outputs exactly*, as slight implementation differences might affect the outputs. In the Matlab code, the `main` script is subdivided into an initialization part and the parts below, you can execute the separate parts by hitting `Ctrl+Enter` while your cursor is in the corresponding part.

1.3 Conventions

In our code, we use the naming convention that `patch_size` represents the size of a square patch in one dimension and `patch_radius` the amount of pixels between the center pixel of an odd-sized patch and its border. Thus, `patch_size` = `patch_radius` $\cdot 2+1$ and a patch has `patch_size`^{2} pixels.

Furthermore, for all conversions between matrices and vectors, unless otherwise stated, we assume column-wise order. This means that e.g. a matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ gets transormed into $\mathbf{a} = \begin{bmatrix} 1 & 3 & 2 & 4 \end{bmatrix}^T$ and vice-versa. Please refer to the Matlab command `reshape` and Python command `numpy.reshape`.

2 Part 1: Calculate Corner Response Functions

We detect features using a Corner Response Function. Two well known Corner Response Functions are the Shi-Tomasi score and Harris score. As seen in the lecture, the Harris score for a given pixel (u, v) is:

$$R_{\text{harris}}(u, v) = \det(M(u, v)) - \kappa \text{trace}^2(M(u, v)) \tag{1}$$

and the Shi-Tomasi score is

$$R_{\text{shi-tomasi}}(u, v) = \min(\lambda_1(u, v), \lambda_2(u, v)) \tag{2}$$

where λ_1 and λ_2 are the two eigen-values of M. The matrix M is defined as

$$M(u, v) = \begin{bmatrix} \sum I_x^2(u, v) & \sum I_x(u, v) I_y(u, v) \\ \sum I_x(u, v) I_y(u, v) & \sum I_y^2(u, v) \end{bmatrix}, \tag{3}$$

where the sums are performed over a fixed-size image patch around the point for which the score is evaluated, and I_x and I_y are the image derivatives in the x and y direction respectively. These can be obtained by convolving the image with the Sobel filter:

$$I_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I, \quad I_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I \tag{4}$$

In the lecture, we have observed that it is sensible to combine smoothing and derivative filters if we have noisy images. In this exercise, however, the given images are free of noise and we can directly use derivative filters (e.g. Sobel filter) without smoothing. You should obtain roughly the scores shown in Fig. 2. Some hints:

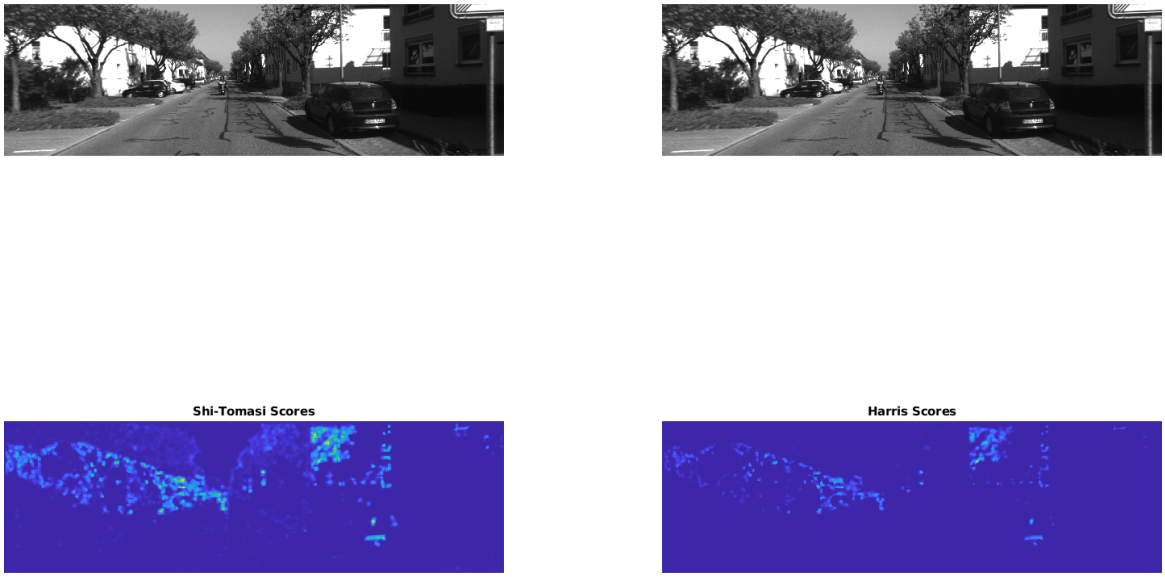


Figure 2: Corner response functions for Harris and Shi-Tomasi on the first frame of the KITTI dataset.

- When applying an odd-sized filter without padding, values cannot be calculated for pixels within $(patch_size - 1)/2$ of the image border. This reduces the size of the resulting matrices by $patch_size - 1$ in each dimension. For example, if an image I has size $h \times w$, convolution with a patch size of 3 will then result in a matrix of size $(h-2) \times (w-2)$. A second convolution with an odd patch size p will result in a matrix of size $(h - 2 - (p - 1)) \times (w - 2 - (p - 1))$.
- The fact that the Sobel filter is both linear and separable (i.e. it can be expressed as the outer product of two 1D filters) can be used to decrease the computational cost of the convolutions.
- You should have intermediate matrices containing as coefficients $I_x, I_y, I_x^2, I_y^2, I_x I_y, \sum I_x^2, \sum I_y^2$ and $\sum I_x I_y$ for each pixel *where they are defined*.
- You should be able to evaluate both $R_{\text{shi-tomasi}}$ and R_{harris} without any for-loop. This can be done by deriving closed-form expressions for the determinant, trace, and eigenvalues of M *as functions of its individual entries* and leveraging element-wise operators (in Matlab `.*` and `.^` or in numpy `*` and `**`).
- The final matrix R can be restored to the origianl size by padding with zeros using `padarray` or `numpy.pad`.
- The image in Fig. 2 has been obtained by setting all negative scores to 0. You can do this since in the remainder of this exercise we will only use the positive values.

3 Part 2: Select keypoints

One method for selecting keypoints is to simply find all scores above a threshold. However, a "good" threshold can depend on many changing factors, which makes it difficult to choose. Another approach is to select the k points with highest score instead. We will pursue this approach here to avoid manually setting a threshold. In both cases, pixels with high scores tend to have neighbors also with high scores. To prevent selecting neighboring pixels as keypoints, we perform non-maximum suppression around the highest-scoring pixels, setting all scores within a radius (or square box) to 0.

Your entire function should contain only one for-loop used to interatively select the highest-scoring keypoint and suppress its neighbors. Take a closer look at the ways in which the `max` function can be used. `ind2sub` in Matlab and `numpy.unravel_index` in Python might also be helpful. Fig. (3) shows what you should roughly get.



Figure 3: Keypoints selected in the first frame of the KITTI dataset.

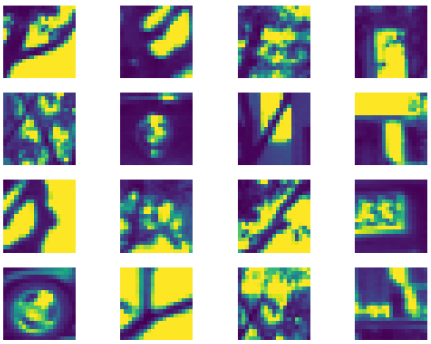


Figure 4: The 16 patch-based descriptors with the highest Harris scores with our code.

4 Part 3: Describe keypoints

We describe keypoints using just the intensity values of the pixels around the keypoint. Trivial, right? Still, it is intepretable, which makes it easy toverify that your output makes sense after this step.

Your function should return a $d \times k$ matrix, where d is the descriptor dimension (total amount of pixels in patch) and k the number of keypoints. The ith column of the matrix should contain the patch intensities around the ith keypoint, stored in column-wise order (see section 1.3). You might want to check out the `reshape` function.

Also, we recommend to `padarray` or `numpy.pad` the input matrix so that you don't have to deal with special cases if a keypoint is closer to the image border than the descriptor patch radius. You should use at most one for-loop, to iterate over keypoints.

Once the descriptors are evaluated, you should be able to tell where in the image they come from based on the plotting we implemented for you. Our 16 "best" descriptors (highest Harris score) are shown in Fig. (4). Assuming that the keypoint selection code started with the keypoint with highest Harris score and put it at the first place (etc.), the 16 "best" descriptors should be stored in the first 16 columns.

5 Part 4 and 5: Match descriptors

In this final step, we will match descriptors between two images. We will simply do this by matching the image patches that differ the least. While this is not quite state-of-the-art, we can already use this to track keypoints across frames! In part 4, you will use the first two frames of KITTI to develop and debug your descriptors matching. Then in part 5 you will apply your tracking to the first 200

frames of the KITTI dataset, and if all goes well, you should obtain behavior as shown in the preview video (if your code is efficient, you should be able to achieve a similar speed).

Note that you can use the Matlab function `pdist2` or scipy function `cdist` in Python to compute the SSD between two vectors, which will significantly boost performance. Familiarize yourself with this command and use it for this exercise. Your function should take as input two descriptor matrices, query_descriptors (descriptors from current frame) and database_descriptors (descriptors from previous frame). It should then return a vector with the same length as the query_descriptors matrix, such that the i-th *element* of the vector is the index of the *database* descriptor correspoinding to the i-th *query* descriptor. As you can imagine, not all query descriptors will be matched. A naive approach would be to look for a descriptor distance threshold, such that you only match descriptors which have a smaller descriptor distance between them. However, since we are looking at subsequent frames of a trajectory, we know that there should be at least one match, and so we will use an adaptive threshold

$$\delta = \lambda \cdot d_{\min}, \tag{5}$$

where $d_{\min}$ is the smallest non-zero distance between two descriptors from the two descriptor sets. Write code that matches a database descriptor at most once, i.e. no database descriptor should be matched to more than one query descriptor. If all goes well, the plot resulting from executing the code from part 4 should look like Fig. 1.1. You are then ready to run part 5 and compare with the preview video.

5.1 Matching quality

Note that the matching is not quite perfect - many keypoints are not matched, and there are occasional outliers (you can tweak λ to trade off between false negatives and false positives - give it a try!). In a later exercise, we will see how we can filter outliers using geometric verification. Apart from geometric verification, can you think of a simple way to filter outliers in the given dataset (hint: camera motion is not too big between frames)?

5.2 Optimizing your code the smart way - profiling

If you want to make your code run faster, you might be tempted to make a guess at what part of your code is slowest and then try to optimize that - don't do that, that's ineffective! The most effective way to optimize your code is through profiling. In Matlab, it's very simple to profile code. Type `profile on` in the console, then run part 5. Once part 5 is complete, type `profile viewer` and inspect the profile summary. Is it what you would have expected it to be? Identify one function that you have written and click on its corresponding entry. You can scroll down and see how much time each line of code has taken. For Python, there are several nice options like `cProfile` or `line_profiler`.

6 Numerical Exercises

1. Compute the distance for each of the following vectors a, b, c with vector d according to the following similarity measures:

$$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \qquad c = \begin{bmatrix} 6 \\ 5 \\ 4 \end{bmatrix} \qquad d = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$$

   (a) Cross-correlation

   (b) Normalized Cross Correlation (NCC)

   (c) Sum of Squared Differences (SDD)

   (d) Sum of Absolute Differences (SAD)

2. Compute the census transform for the the following patches A and B and the hamming distance between the two transforms.

$$A = \begin{bmatrix} 2 & 3 & 7 \\ 0 & 5 & 6 \\ 1 & 9 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 9 & 4 \\ 2 & 4 & 3 \\ 5 & 0 & 6 \end{bmatrix}$$

3. Compute the cornerness measure for the Moravec Corner detector for the following patch C. To avoid border artifacts, you can neglect padding.

$$C = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

4. Compute the second moment matrix for the following patch D. To avoid border artifacts, you can neglect padding.

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 4 & 4 & 4 & 5 \\ 5 & 5 & 5 & 6 \end{bmatrix}$$

5. Which of the following matrices correspond to the second moment matrix M approximating the sum of squared differences for the given patch A?
   Assume the following conditions $\lambda_1 \neq 0, \lambda_2 \neq 0$, $M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$

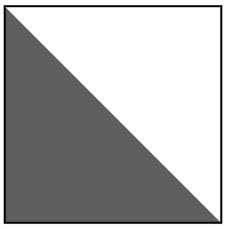


Figure 5: Patch A

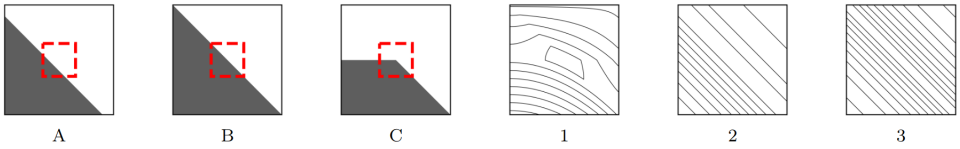
   (a) $M = \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix}$

   (b) $M = \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix}$

(c) $M = \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} \\ -\sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix}$

(d) $M = \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} \\ -\sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix}$

6. Consider the image patches within the red dashed squares. Which image patch [A, B, C] corresponds to which SSD cost landscape [1, 2, 3]? Curves in the SSD landscapes [1, 2, 3] indicate points of constant SSD values, i.e., all points on one curve have the same SSD cost with respect to the center patch.



(a) A - 3, B - 2, C - 1

(b) A - 3, B - 1, C - 2

(c) A - 2, B - 3, C - 1

(d) A - 2, B - 1, C - 3

7. Discuss the impact of image resolution on Harris corner detection. How might this affect the performance of a vision-based robot operating in environments with varying distances to objects?

8. A robotics company is developing a vision system for a warehouse robot that needs to identify and navigate around various objects. They decide to use Harris corner detection as part of their object recognition pipeline. What are two potential challenges they might face when implementing this algorithm in a warehouse environment, and how could they address each challenge?

9. A self-driving car manufacturer wants to incorporate Harris corner detection in their vehicle's computer vision system for lane detection and obstacle avoidance. How might the choice of parameters for the Harris corner detector affect the system's performance in various driving scenarios (e.g., urban vs. highway, day vs. night)? Suggest two specific parameter adjustments and explain their potential impacts.