Lecture 5-2: Feature Detection and Tracking

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor(s).*

This lecture discusses how to:

- extract point features in an image,
- track features across two images, and
- perform descriptor-based feature matching (we mainly provide pointers).

The presentation mostly follows Chapter 4 in [4].

5.2 Corner detection

Consider a pixel $\bar{\boldsymbol{x}} = [u, v]^\mathsf{T}$ in an image and call $W(\bar{\boldsymbol{x}})$ a rectangular window of given size centered at $\bar{\boldsymbol{x}}$. Moreover, call $\mathcal{I}(\bar{\boldsymbol{x}})$ the intensity at pixel $\bar{\boldsymbol{x}}$. Then, $\bar{\boldsymbol{x}}$ is a good "corner", if shifting the window $W(\bar{\boldsymbol{x}})$ in any direction $\boldsymbol{\delta} = [\delta_u \ \delta_v]^\mathsf{T}$ produces a new window $W(\bar{\boldsymbol{x}} + \boldsymbol{\delta})$ which is different from $W(\bar{\boldsymbol{x}})$. Mathematically, a good corner is such that:

$$\min_{\|\boldsymbol{\delta}\|=\epsilon} \sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \|\mathcal{I}(\boldsymbol{x} + \boldsymbol{\delta}) - \mathcal{I}(\boldsymbol{x})\|^2 > 0 \quad \text{(and possibly as large as possible)} \tag{5.15}$$

since if the minimum is zero, then there is a direction $\boldsymbol{\delta} \neq 0$ (we constrain $\|\boldsymbol{\delta}\| = \epsilon$, for some small scalar $\epsilon > 0$) that does not cause any change of appearance.

For a small displacement $\boldsymbol{\delta}$, we can take a first-order Taylor expansion of the cost and write:

$$\min_{\|\boldsymbol{\delta}\|=\epsilon} \sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \|\mathcal{I}(\boldsymbol{x}) + \nabla\mathcal{I}(\boldsymbol{x})^\mathsf{T}\boldsymbol{\delta} - \mathcal{I}(\boldsymbol{x})\|^2 = \min_{\|\boldsymbol{\delta}\|=\epsilon} \sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \|\nabla\mathcal{I}(\boldsymbol{x})^\mathsf{T}\boldsymbol{\delta}\|^2 = \tag{5.16}$$

$$\min_{\|\boldsymbol{\delta}\|=\epsilon} \sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \boldsymbol{\delta}^\mathsf{T} \left(\nabla\mathcal{I}(\boldsymbol{x})\nabla\mathcal{I}(\boldsymbol{x})^\mathsf{T}\right) \boldsymbol{\delta} = \min_{\|\boldsymbol{\delta}\|=\epsilon} \boldsymbol{\delta}^\mathsf{T} \left(\sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \nabla\mathcal{I}(\boldsymbol{x})\nabla\mathcal{I}(\boldsymbol{x})^\mathsf{T} \right) \boldsymbol{\delta} \tag{5.17}$$

where $\nabla\mathcal{I}(\boldsymbol{x}) \in \mathbb{R}^{2\times 1}$ is the gradient of the intensity function $\mathcal{I}(\cdot)$ at $\boldsymbol{x}$ (see also paragraph about "Image Gradients" below for practical considerations). Let us now define:

$$\boldsymbol{G} = \sum_{\boldsymbol{x} \in W(\bar{\boldsymbol{x}})} \nabla\mathcal{I}(\boldsymbol{x})\nabla\mathcal{I}(\boldsymbol{x})^\mathsf{T} \in \mathbb{R}^{2\times 2} \tag{5.18}$$

Then, since $\boldsymbol{\delta}$ is a vector of magnitude ϵ, we write it as $\boldsymbol{\delta} = \epsilon \boldsymbol{d}$ where $\boldsymbol{d}$ is a unit-norm vector. Substituting $\boldsymbol{\delta} = \epsilon \boldsymbol{d}$ and the definition of $\boldsymbol{G}$ in (5.18) we get:

$$\min_{\|\boldsymbol{\delta}\|=\epsilon} \boldsymbol{\delta}^\mathsf{T}\boldsymbol{G}\boldsymbol{\delta} = \min_{\|\boldsymbol{d}\|=1} \epsilon^2 \boldsymbol{d}^\mathsf{T}\boldsymbol{G}\boldsymbol{d} = \epsilon^2 \, \lambda_{\min}(\boldsymbol{G}) \tag{5.19}$$

where $\lambda_{\min}(\boldsymbol{G})$ is the smallest eigenvalue of $\boldsymbol{G}$ (if you do not understand how we get the smallest eigenvalue from the minimization, read this: https://en.wikipedia.org/wiki/Rayleigh_quotient).

It is clear that $\min_{\boldsymbol{\delta}} \boldsymbol{\delta}^{\mathsf{T}} \boldsymbol{G} \boldsymbol{\delta}$ is nonzero (for a nonzero $\boldsymbol{\delta}$) if and only if $\boldsymbol{G}$ is positive definite. Moreover, the larger the eigenvalues of $\boldsymbol{G}$, the larger the minimum. Therefore, a pixel $\bar{\boldsymbol{x}}$ is a good corner, if the corresponding matrix $\boldsymbol{G}$ exhibits large eigenvalues.

Different corner detectors use different scores to quantify how "large" the eigenvalues of $\boldsymbol{G}$ are. The popular *Harris corner detector* uses the following quantity (for a given small scalar k)

$$C(\boldsymbol{G}) = \det(\boldsymbol{G}) - k \operatorname{tr}(\boldsymbol{G})^2 \qquad \text{(Harris corner score)} \tag{5.20}$$

to score the quality (or "cornerness") of a pixel [2].

Another popular choice of the cornerness score is simply:

$$S(\boldsymbol{G}) = \lambda_{\min}(\boldsymbol{G}) \qquad \text{(Shi-Tomasi score)} \tag{5.21}$$

and has been proposed by Shi and Tomasi in [5].

An intuitive explanation of the Harris corner score Calling λ_1 and λ_2 the two eigenvalues of $\boldsymbol{G}$ and recalling that $\det(\boldsymbol{G}) = \lambda_1 \lambda_2$ and $\operatorname{tr}(\boldsymbol{G}) = \lambda_1 + \lambda_2$, the Harris corner score can be rewritten as:

$$C(\boldsymbol{G}) = \lambda_1 \lambda_2 - k\lambda_1^2 - k\lambda_2^2 - 2k\lambda_1\lambda_2 = (1-2k)\lambda_1\lambda_2 - k(\lambda_1^2 + \lambda_2^2) \tag{5.22}$$

Since k is typically small, it is clear that when λ_1, λ_2 are small, the corner score will be small, consistently with what we expect.

Now consider the case in which $\lambda_1 \gg \lambda_2$. In this case we can write:

$$C(\boldsymbol{G}) = (1-2k)\lambda_1\lambda_2 - k(\lambda_1^2 + \lambda_2^2) \approx -k\lambda_1^2 \tag{5.23}$$

and the score becomes negative. This is the case in which we have an *edge*, rather than a corner.

Finally, consider the case in which $\lambda_1 \approx \lambda_2 \gg 0$:

$$C(\boldsymbol{G}) = (1-2k)\lambda_1\lambda_2 - k(\lambda_1^2 + \lambda_2^2) \approx (1-2k)\lambda_1^2 - 2k\lambda_1^2 = (1-4k)\lambda_1^2 \tag{5.24}$$

which is large, since we assumed $\lambda_1 \gg 0$ and small k. Therefore, the score rewards large eigenvalues of $\boldsymbol{G}$ when they have similar magnitude.

Image gradients When taking the Taylor expansion in (5.16) we treated the intensity as a continuous function over the pixel domain, and adopted the standard definition of gradient (that we assume to be a column vector):

$$\nabla \mathcal{I}(\boldsymbol{x}) = \nabla \mathcal{I}(u,v) = \begin{bmatrix} \frac{\partial \mathcal{I}(u,v)}{\partial u} \\ \frac{\partial \mathcal{I}(u,v)}{\partial v} \end{bmatrix} \tag{5.25}$$

However, in practice the image if formed by discrete pixels, and the notion of derivative needs be replaced by a discrete counterpart. A common choice is to use *finite differences*:

$$\nabla \mathcal{I}(\boldsymbol{x}) = \nabla \mathcal{I}(u,v) \approx \begin{bmatrix} \frac{\mathcal{I}(u+h,v) - \mathcal{I}(u,v)}{h} \\ \frac{\mathcal{I}(u,v+h) - \mathcal{I}(u,v)}{h} \end{bmatrix} \tag{5.26}$$

If you want to learn more about alternative choices, please take a look at [4, p. 99].

5.3 Feature Tracking

Assume we extracted N corner features from an image $\mathcal{I}_1$. Consider one of such corner features at pixel position $\boldsymbol{x}_1$ and call $\boldsymbol{p}$ the 3D point which projects to $\boldsymbol{x}_1$. The question we answer in this section is: *given a second (temporally continuous) image $\mathcal{I}_2$, compute the pixel projection of $\boldsymbol{p}$ in $\mathcal{I}_2$, namely $\boldsymbol{x}_2$*. Note that in this problem we are only given the images $\mathcal{I}_1$ and $\mathcal{I}_2$ and the pixel $\boldsymbol{x}_1$ so we cannot use standard perspective projection to compute $\boldsymbol{x}_2$ (e.g., the coordinates of $\boldsymbol{p}$, and the intrinsic and extrinsic parameters of the cameras picturing $\mathcal{I}_1$ and $\mathcal{I}_2$ are unknown).

Another way to look at the problem is to think that the camera is moving between the capture of image $\mathcal{I}_1$ and $\mathcal{I}_2$, and we want to compute the corresponding displacement of the pixel $\boldsymbol{x}_1$ induced by the camera motion. Therefore, computing $\boldsymbol{x}_2$ is the same as tracking the motion $\boldsymbol{\delta} = [\delta_u, \delta_v]^\mathsf{T}$ of pixel $\boldsymbol{x}_1$ between image $\mathcal{I}_1$ and $\mathcal{I}_2$:

$$\boldsymbol{x}_2 = \boldsymbol{x}_1 + \boldsymbol{\delta} \tag{5.27}$$

Since we do not have any geometric information, our only hope is to compute the motion $\boldsymbol{\delta}$ by comparing pixel intensities in the two images. More specifically, if we build a rectangular window $W(\boldsymbol{x}_1)$ collecting a set of pixels around pixel $\boldsymbol{x}_1$ in $\mathcal{I}_1$, we can try to compute $\boldsymbol{x}_2$ by looking for a patch in $\mathcal{I}_2$ whose appearance is similar to $W(\boldsymbol{x}_1)$.[1] Reasoning in terms of pixel displacement $\boldsymbol{\delta}$, this is equivalent to computing:

$$\min_{\boldsymbol{\delta}} \sum_{\boldsymbol{y} \in W(\boldsymbol{x}_1)} \|\mathcal{I}_1(\boldsymbol{y}) - \mathcal{I}_2(\boldsymbol{y} + \boldsymbol{\delta})\|^2 \tag{5.28}$$

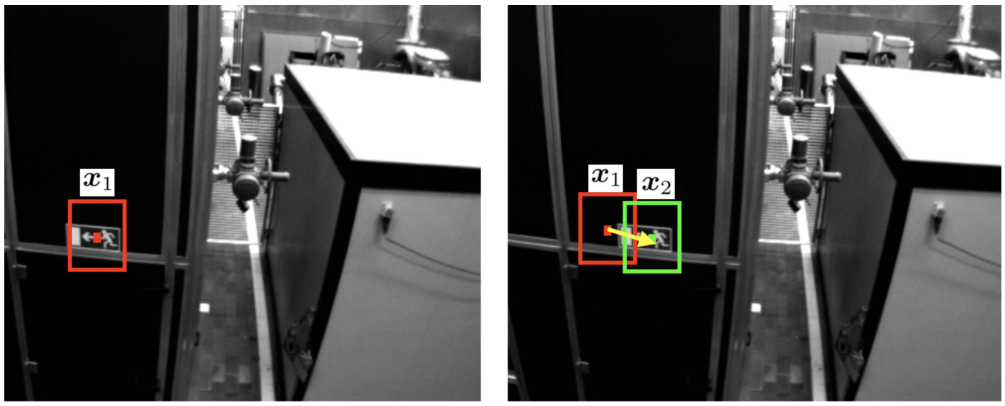

Figure 5.2: The goal of feature tracking is to compute the displacement of a given pixel $\boldsymbol{x}_1$ (displacement shown as yellow arrow) between two images, due to the camera motion.

However, the minimization (5.28) already took a strong assumption: every pixel in $W(\boldsymbol{x}_1)$ is moving by the same amount $\boldsymbol{\delta}$. Unfortunately, this assumption, typically referred to as a *translational motion model* [4, Section 4.2.1], is only true if $W(\boldsymbol{x}_1)$ is picturing a flat surface parallel to the image and the camera is moving parallel to it. This justifies the use of a more general deformation model, which assumes a more complex mapping of the pixels:

$$\min_{\boldsymbol{A}, \boldsymbol{\delta}} \sum_{\boldsymbol{y} \in W(\boldsymbol{x}_1)} \|\mathcal{I}_1(\boldsymbol{y}) - \mathcal{I}_2(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{\delta})\|^2 \tag{5.29}$$

[1] Note: this assumes that the illumination changes due to the motion of the camera are small, an assumption known as "brightness constancy" constraint.

which assumes an affine (pixel) motion model for the pixels in the window $W(\boldsymbol{x}_1)$. This model is a good approximation for small planar patches parallel to the image plane and undergoing small rotations [4, Section 4.2.1]. In order to compute the optimal displacement $\boldsymbol{\delta}$, we can take a Taylor expansion around the point $\boldsymbol{A} = \mathbf{I}_2$ and $\boldsymbol{\delta} = \mathbf{0}_2$ of the intensity $\mathcal{I}_2(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{\delta})$ (thought as a function over a continuous pixel space):

$$\mathcal{I}_2(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{\delta}) \approx \mathcal{I}_2(\boldsymbol{y}) + \nabla\mathcal{I}_2(\boldsymbol{y})^\mathsf{T}[(\boldsymbol{A} - \mathbf{I}_2)\boldsymbol{y} + \boldsymbol{\delta}] \tag{5.30}$$

which after substituting back into (5.29) leads to a linear least squares model:

$$\min_{\boldsymbol{A},\boldsymbol{\delta}} \sum_{\boldsymbol{y}\in W(\boldsymbol{x}_1)} \|\mathcal{I}_1(\boldsymbol{y}) - \mathcal{I}_2(\boldsymbol{y}) - \nabla\mathcal{I}_2(\boldsymbol{y})^\mathsf{T}[(\boldsymbol{A} - \mathbf{I}_2)\boldsymbol{y} + \boldsymbol{\delta}]\|^2 \tag{5.31}$$

from which one can compute $\boldsymbol{A}$ and $\boldsymbol{\delta}$ in closed form.

In oder to get some more insight, let us go back to the purely translational deformation model, whose Taylor expansion gives:

$$\mathcal{I}_2(\boldsymbol{y} + \boldsymbol{\delta}) \approx \mathcal{I}_2(\boldsymbol{y}) + \nabla\mathcal{I}_2(\boldsymbol{y})^\mathsf{T}\boldsymbol{\delta} \tag{5.32}$$

After substituting (5.32) back into (5.29) we obtain:

$$\min_{\boldsymbol{\delta}} \sum_{\boldsymbol{y}\in W(\boldsymbol{x}_1)} \|\mathcal{I}_1(\boldsymbol{y}) - \mathcal{I}_2(\boldsymbol{y}) - \nabla\mathcal{I}_2(\boldsymbol{y})^\mathsf{T}\boldsymbol{\delta}\|^2 \tag{5.33}$$

whose solution can be computed using linear least squares as:

$$\boldsymbol{\delta}^\star = \left(\sum_{\boldsymbol{y}\in W(\boldsymbol{x}_1)} \nabla\mathcal{I}_2(\boldsymbol{y})\nabla\mathcal{I}_2(\boldsymbol{y})^\mathsf{T}\right)^{-1} \sum_{\boldsymbol{y}\in W(\boldsymbol{x}_1)} \nabla\mathcal{I}_2(\boldsymbol{y})(\mathcal{I}_1(\boldsymbol{y}) - \mathcal{I}_2(\boldsymbol{y})) \tag{5.34}$$

Note that the matrix to be inverted is the same as the $\boldsymbol{G}$ we computed to extract good corners in the previous section. Therefore, by construction, good corners are such that the matrix in (5.34) is invertible.

5.4 Descriptor-based Feature Matching

Feature tracking typically works well if the images are not very different (e.g., collected by the same camera at consecutive time instants). However, feature tracking does not work well when the images are taken by radically different viewpoints. In these case, a *Descriptor-based Feature Matching* is typically preferred, where for each keypoint, one computes a *descriptor* vector. The descriptor can be understood as a unique "signature" of the feature.

Therefore, *descriptor-based feature matching* works as follows:

- given images $\mathcal{I}_1$ and $\mathcal{I}_2$, it computes corners in both images as well as the corresponding descriptors for each corner;
- it matches corners in the two images if the corresponding descriptors are close enough (note: each descriptor is simply a large vector)

Many of these descriptors also come with a *feature detector*, which is typically a more sophisticated version of the Harris corners we saw earlier in this document. However, it is also common to attach detectors to Harris or Shi-Tomasi corners.

Common detector-descriptors are:

- SIFT [3]: https://docs.opencv.org/3.4.3/da/df5/tutorial_py_sift_intro.html (this nice video provides an in-depth explanation: https://www.youtube.com/watch?time_continue=3964&v=NPcMS49V5hg)
- SURF [1]: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html
- ORB: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html

All these (and more) are available in OpenCV, the most popular library for computer vision.

References

[1] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: speeded up robust features. In *European Conf. on Computer Vision (ECCV)*, 2006.

[2] C. Harris and M. Stephens. A combined corner and edge detector. *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, August 1988.

[3] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. J. of Computer Vision*, 60(2):91–110, 2004.

[4] Y. Ma, S. Soatto, J. Kosecka, and S.S. Sastry. *An Invitation to 3-D Vision*. Springer, 2004.

[5] J. Shi and C. Tomasi. Good features to track. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.