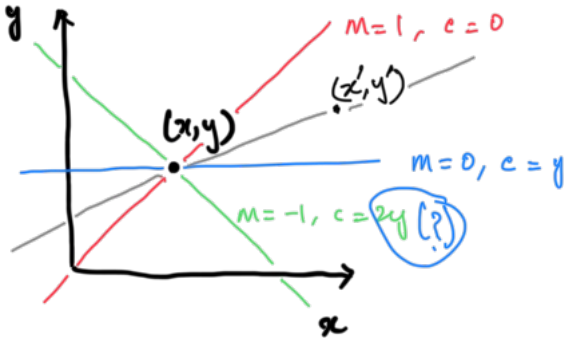
Interest Points, Descriptors and Optical Flow

Computer Vision I (CSCI 3240U)

Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca

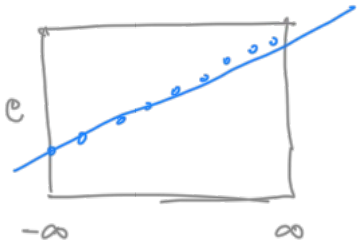


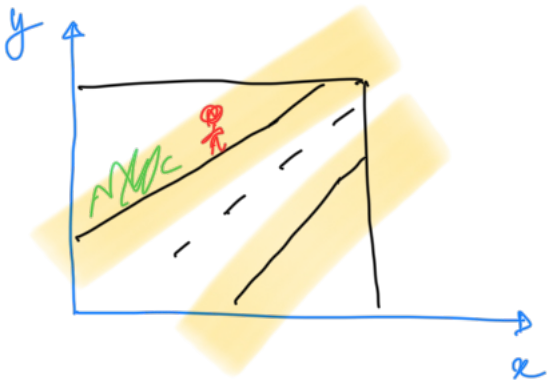


$$y = mx + c$$

$$c = -xm + y$$

$-\infty, \infty$




y
x

Week 8 at a glance

- **Lecture 1: Interest points and descriptors (Sec. 7.1.1–7.1.3)**
  - What makes a point worth finding
  - The structure tensor
  - Harris and Shi-Tomasi
  - Beyond corners: descriptors, briefly (SIFT is self-study)
- **Lecture 2: Optical flow (Sec. 9.3, 7.1.5)**
  - Brightness constancy and the aperture problem
  - Lucas-Kanade, *the same matrix again*
  - Coarse-to-fine tracking

Reading: Szeliski Sec. 7.1.1–7.1.3, 7.1.5, 9.1, 9.3. Exercises `w08a`, `w08b`. **Lab 7** covers corners and optical flow.

Lecture 1: Interest points

Why not use every pixel?

To match two images, or to track between frames, we need **correspondences**.

Using all pixels is hopeless: most of them are not distinctive. A pixel in the middle of the road looks exactly like a million others.

We want points that are:

1. **Repeatable:** found again in another view of the same scene.
2. **Distinctive:** surrounded by structure that identifies them.
3. **Local:** occlusion only kills the points it covers.
4. **Efficient:** few enough to work with.

Correspondences

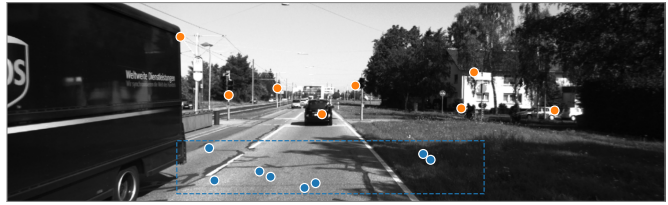
KITTI's left and right cameras, half a metre apart. Each line joins the same scene point in both images: a **correspondence** (40 of 654 SIFT matches).



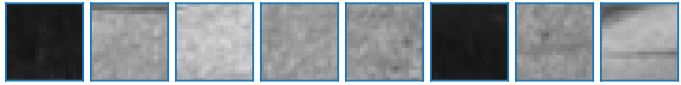
KITTI Road benchmark, um_000032 (Fritsch, Kuehnl and Geiger, ITSC 2013), CC BY-NC-SA 3.0.

Ambiguous versus distinctive

Road patches (top) are indistinguishable; corner patches (bottom) are each unique. Detection is the job of picking the second kind.

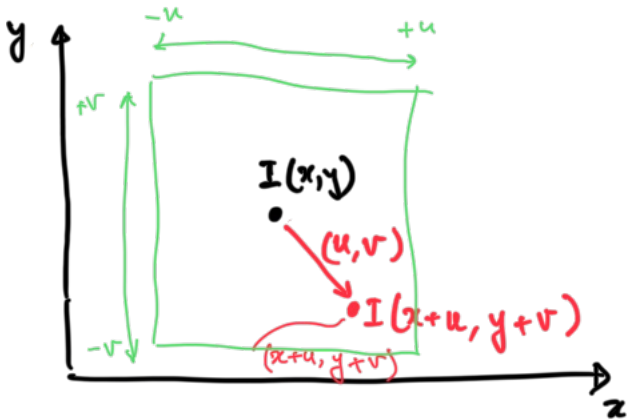


road patches: ambiguous



corner patches: distinctive




y
x
-u
+u
+v
-v
I(x,y)
(u,v)
I(x+u, y+v)
(x+u, y+v)

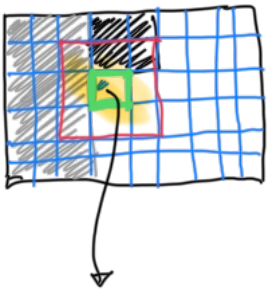
What kind of point is distinctive?

Consider shifting a small window by (u, v) and measuring how much the contents change:

$$E(u,v) = \sum_{(x,y)\in W} w(x,y)\,\big[I(x+u,\ y+v) - I(x,y)\big]^2$$

W is the window around the point. The weight $w(x, y) \geq 0$ says how much each pixel counts: $w = 1$ everywhere (a box window), or a Gaussian centred on the point.

- **Flat region:** E small in *every* direction, nothing to lock onto.
- **Edge:** E small **along** the edge, large across it.
- **Corner:** E large in **every** direction. Uniquely locatable.



$$E(u,v) = \sum_{(x,y) \in W} w(x,y) \left[I(x+u, y+v) - I(x,y) \right]^2$$

(2,3) : 255

↑ LOC ↑ INTENSITY



ERROR SURFACE



$E(u,v)$

127^2	255^2	255^2
127^2	$\varnothing$	$\varnothing$
127^2	$\varnothing$	$\varnothing$

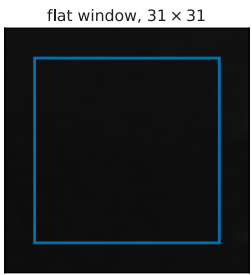
Three real error surfaces

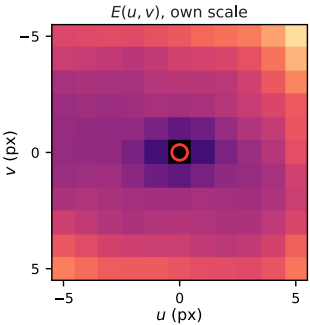
Consider the image below. It shows a flat (blue square), an edge (orange square), and a corner (green square) region. Each region is 31×31. We will compute $E(u, v)$ by shift u and v between -5 and 5, so each $E(u, v)$ will be 11×11 (shown in the following slides).

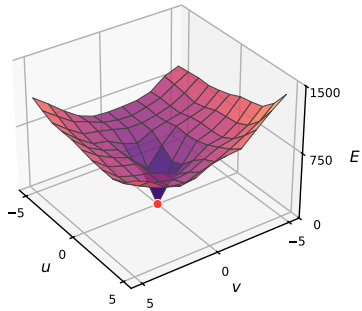


$E(u, v)$ for Flat region: nothing to lock onto

- $E = 0$ only when $u = 0$ and $v = 0$.
- On its own scale (0 to 1,500) E is a shallow bowl, never above 1,393.
- On the shared scale (slide *Flat vs. Edge vs. Corner*) it is flat.



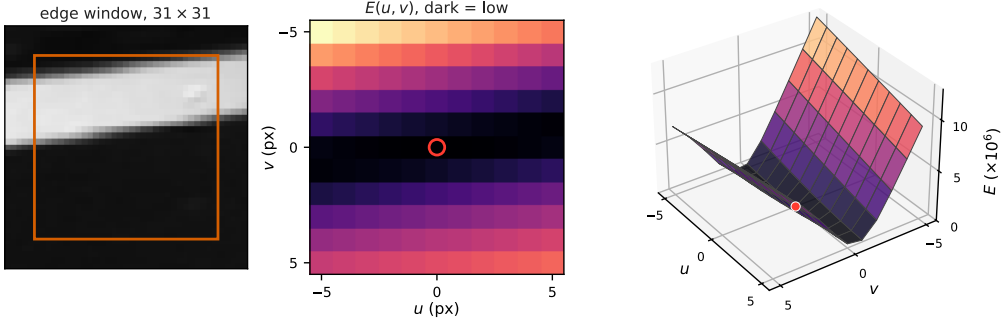




scale 0 to 1,500; max E = 1,393

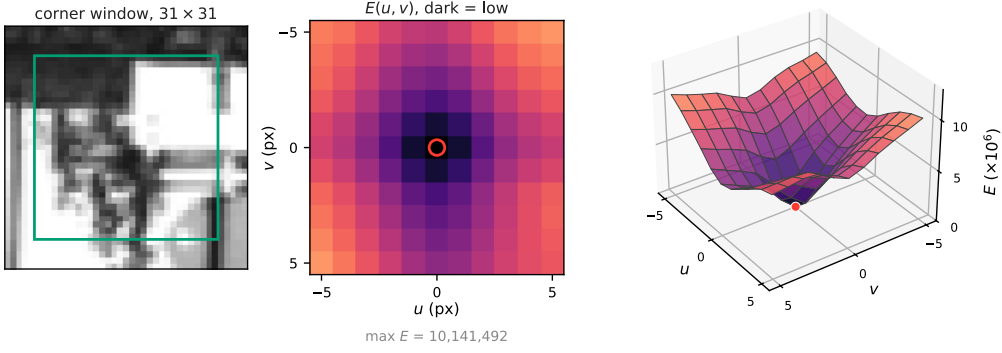
$E(u, v)$ around an Edge: a valley along the edge

- Shifting **along** the edge barely changes E: $(u, v) = (5, 0)$ costs 0.43 million.
- Shifting **across** it costs a lot: $(0, 5)$ costs 7.8 million.

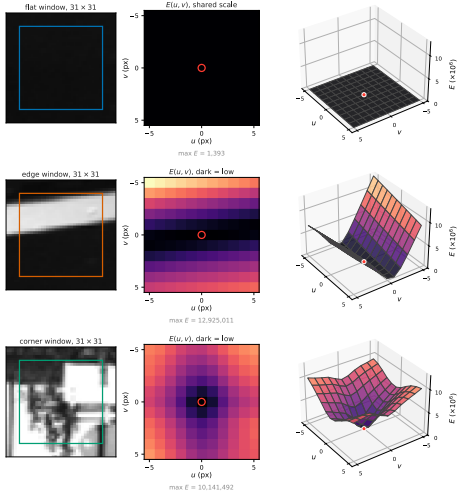


$E(u, v)$ around a Corner: one clear minimum

- E rises in **every** direction from $(0, 0)$.
- Even a one-pixel shift costs at least 1.2 million.
- The minimum is unique, so the window can be located exactly.



Flat vs. Edge vs. Corner: $E(u, v)$ plotted using the same scale



Linearising $E(u,v) = \sum_{(x,y)\in W} w(x,y)\,\big[I(x+u,\,y+v) - I(x,y)\big]^2$

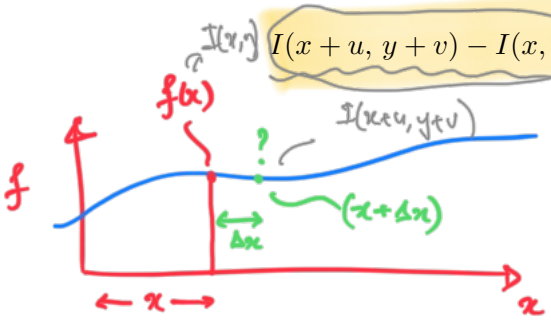
Step 1. For a small shift, expand I to first order (Taylor):

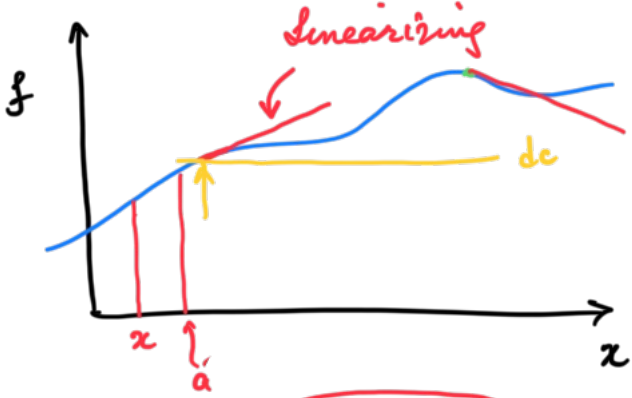
$$I(x+u,\,y+v) \approx I(x,y) + I_x(x,y)\,u + I_y(x,y)\,v$$

$I_x = \partial I/\partial x$ and $I_y = \partial I/\partial y$ are the image gradients of Week 4, taken at (x,y). Together they form the gradient vector $\nabla I = (I_x, I_y)^\top$.

Step 2. Subtract $I(x,y)$; it cancels:

$$I(x+u,\,y+v) - I(x,y) \approx I_x u + I_y v$$





$$f(a) = f(x) + (x-a)f'(x) + \frac{(x-a)^2}{2!}f''(x) + \frac{(x-a)^3}{3!}f'''(x) + \cdots$$

Linearising E: a quadratic in (u, v)

Step 3. Substitute step 2 into E. The weight $w(x, y)$ stays with each pixel; to save space we drop the (x, y) on I_x and I_y:

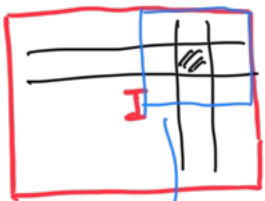
$$E(u, v) \approx \sum_{(x,y)\in W} w(x, y) \left(I_x u + I_y v\right)^2$$

Step 4. Expand the square, then write it as a matrix product:

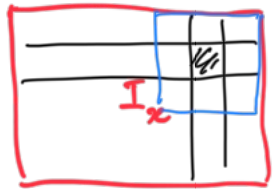
$$\left(I_x u + I_y v\right)^2 = I_x^2\, u^2 + 2 I_x I_y\, uv + I_y^2\, v^2 = \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

So, pixel by pixel, with the weight still in front:

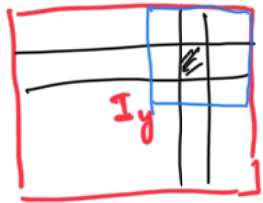
$$E(u, v) \approx \sum_{(x,y)\in W} w(x, y) \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$



$* S_x =$



$* S_y =$



$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Window

⇒ to capture the local structure around that window.

$$E[u,v] = [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix}$$

$I_{xy} = I_x I_y$

computed for every pixel.

$I_x \rightarrow I_x^2$ — square every entry.

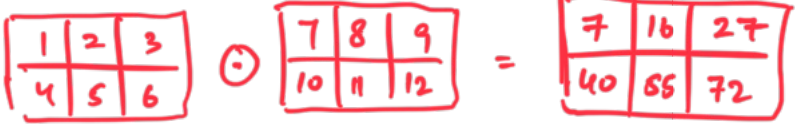
$I_y \rightarrow I_y^2$

$I_x I_y = I_x \odot I_y$ — element-wise multiplication

$I \rightarrow I^2$



$I \odot J$



Linearising E: the matrix $\mathbf{M}$

Step 5. The shift (u, v) is the same at every pixel, so it moves outside the sum; the weight $w(x, y)$ stays inside:

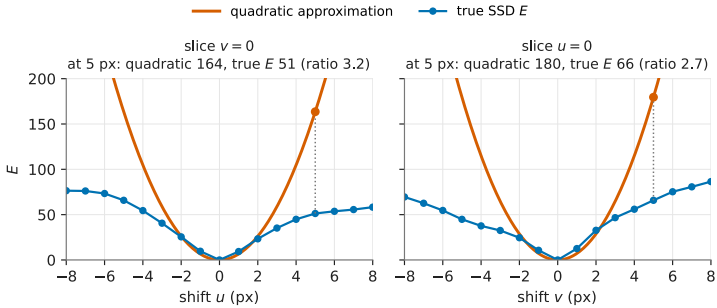
$$E(u,v) \approx \begin{pmatrix} u & v \end{pmatrix} \mathbf{M} \begin{pmatrix} u \\ v \end{pmatrix}, \qquad \boxed{\mathbf{M} = \sum_{(x,y)\in W} w(x,y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}}$$

$\mathbf{M}$ is the **second moment matrix**, also called the **structure tensor**: 2×2, symmetric, and built only from gradients. Since $E \approx \sum_W w\,(I_x u + I_y v)^2 \geq 0$ for every (u, v), $\mathbf{M}$ is positive semi-definite: both eigenvalues are ≥ 0 (matrix types: see the appendix).

Szeliski Sec. 7.1.1, Eqs. (7.3)–(7.8), writes $\Delta\mathbf{u}$ for (u, v) and $\mathbf{A}$ (the auto-correlation matrix) for $\mathbf{M}$. Our $\mathbf{A}$, in Lucas-Kanade, is a different matrix.

How good is the quadratic?

- The true E (blue) and the quadratic $(u\ v)\,\mathbf{M}\,(u\ v)^\top$ (orange) on a 21×21 corner window of a KITTI frame.
- The two stay close for a pixel or two; however, at 5 px the quadratic is 3.2 times too large along u and 2.7 times along v.



$\mathbf{M}$ describes E only near zero shift.

Reading the eigenvalues

$E(u, v)$ is a quadratic form. Its level sets (every shift with the same error) are ellipses whose axes are the eigenvectors of $\mathbf{M}$ and whose extents go as $1/\sqrt{\lambda}$. Throughout, we order the eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$.

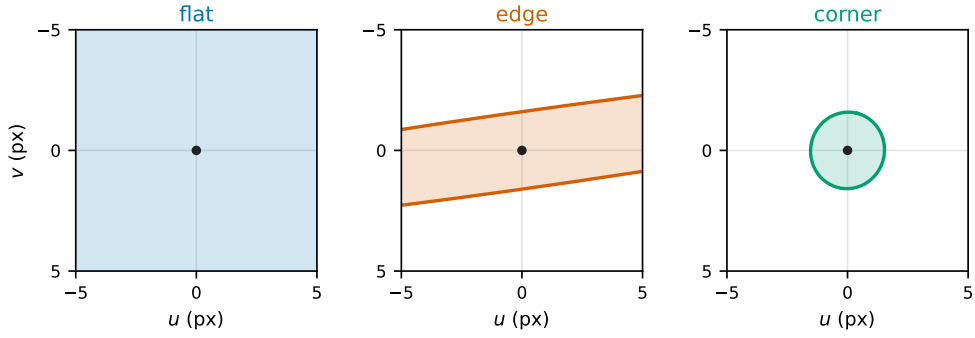
Region	λ_1	λ_2	Meaning
Flat	≈ 0	≈ 0	no structure
Edge	large	≈ 0	structure in one direction
Corner	large	large	structure in two directions

Here ≈ 0 means small compared with λ_1 of a textured window.

The whole of corner detection is: are BOTH eigenvalues large?

The ellipses of our three windows

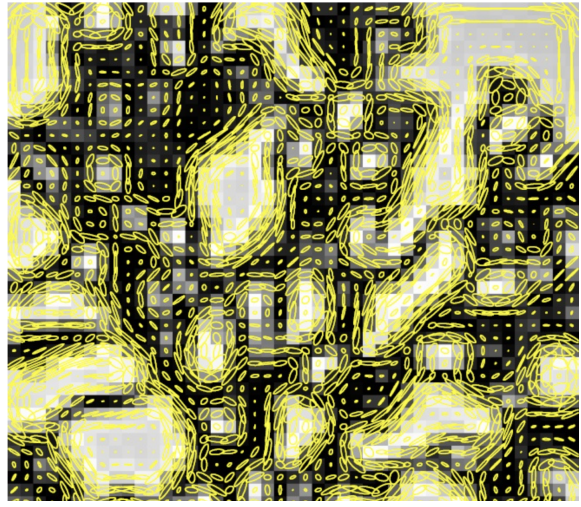
Corner: a small near-circle, pinned down in every direction. **Edge:** a band with half-axes 1.6 px across and 25 px along the edge. **Flat:** half-axes over 100 px, so the whole panel lies inside.



One ellipse per pixel

Compute $\mathbf{M}$ in a window around every pixel, then draw its level set there.

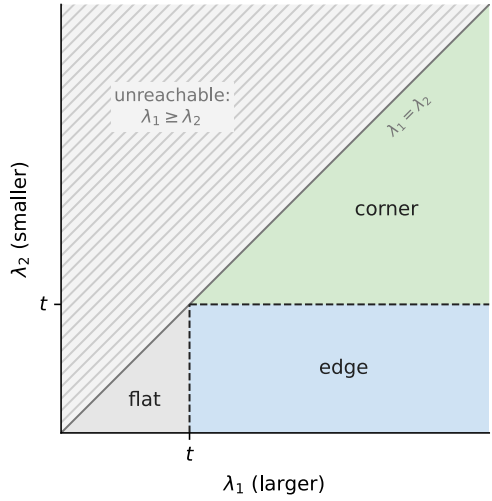
- **Small, round:** both λ large, a corner.
- **Long, thin:** one λ small, an edge; the long axis runs along the edge.
- **Large:** both λ small, a flat patch.



Courtesy K. Grauman; via F. Z. Qureshi, *Computer Vision Notes*, notebook 09.

The λ_1-λ_2 plane

- $\lambda_1 \geq \lambda_2$, so only the half below the diagonal is reachable.
- The threshold t is a choice.
- Shi-Tomasi's test for corner region: $\lambda_2 \geq t$.



Shi-Tomasi

Corner response

$$
\begin{aligned}
R &= \min(\lambda_1, \lambda_2) && (1)\\
&= \lambda_2 && (2)\\
&= \tfrac{1}{2}\operatorname{tr}\mathbf{M} - \sqrt{\left(\tfrac{1}{2}\operatorname{tr}\mathbf{M}\right)^2 - \det\mathbf{M}} && (3)
\end{aligned}
$$

Cheap for a 2×2 matrix. It is also the number that decides how well a patch can be tracked (Lecture 2).

Shi-Tomasi's test for corner detection: $R \geq t$, where t is a user-defined threshold.

Aside: The $\min$ above is redundant, since we assume that eigenvalues are ordered, i.e., $\lambda_1 \geq \lambda_2$.

Harris

Corner Response: no eigenvalues needed.

$$R = \det(\mathbf{M}) - k\left(\operatorname{tr}\mathbf{M}\right)^2 \tag{4}$$

$$= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{5}$$

with $k \approx 0.04$–0.06.

- Large positive R: corner
- Negative R: edge
- Small $|R|$: flat

Shi-Tomasi and Harris corner detection

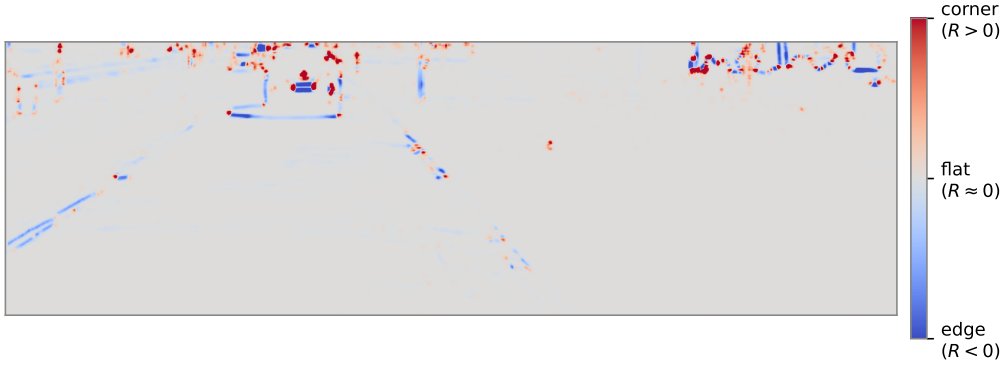
Both need **non-maximum suppression** (NMS) afterwards: keep a pixel only if its score is above a threshold and is the largest in its neighbourhood (say 9×9).



(a) shows $\sqrt{R}$ for $R > 0$. (b) Each corner fires as a blob. (c) NMS keeps only the largest pixel in each 9×9 window.

Harris on a real frame: the response

The Harris response R on a crop of a KITTI frame ($k = 0.05$). Red is corner ($R > 0$), blue is edge ($R < 0$), grey is flat.



Notice the long straight lane markings and the car's bottom edge come out blue (edges).

Harris on a real frame: the corners

The 60 strongest responses after 9×9 non-maximum suppression. They sit on the corners of the car's outline, in the clutter of the tree line, and where lane markings end or shadows cross them. Almost none fall on the plain road surface or along the middle of a straight marking.



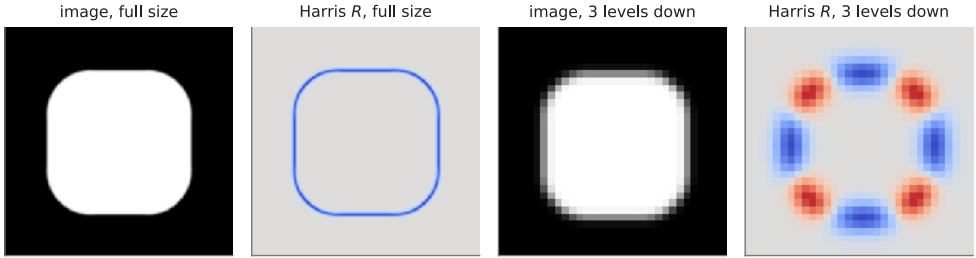
Invariance: what survives?

- **Translation:** yes.
- **Rotation:** yes, with a circular (Gaussian) window. Rotating the image rotates $\mathbf{M}$'s eigenvectors and leaves its eigenvalues alone. (Detected *locations* still shift slightly, from interpolation and the discrete grid.)
- **Intensity shift $I + b$:** yes; gradients are unaffected.
- **Intensity scale aI:** $\mathbf{M}$ scales by a^2 (Harris R by a^4), so a *fixed* threshold is not invariant.
- **Scale:** no. Magnify a rounded corner and a fixed window sees only a gently curved edge.

Scale invariance requires searching over scale (pyramids again), which is what SIFT does (appendix). Rotation **repeatability** measures how well rotation survives in practice: the fraction of points found in one image that are found again, within a few pixels of the right place, in the other. On the four Lab 7 photos at 30° we measured 55–80%, not 100%.

Scale breaks Harris

The same Harris window (Gaussian, $\sigma = 2$ px) on a rounded square. At full size each corner is a gentle arc, and the response there is negative (blue, edge-like). Three pyramid levels down, the arcs are small compared with the window, and the four corners fire (red).



Which points you find depends on the scale you look at.

Beyond corners: descriptors

Detection gives *where*. To match a point across images we also need *what it looks like*: a **descriptor**, a vector that summarises the patch.

Interest points are not just corners:

- **Blobs.** SIFT's detector finds extrema of difference-of-Gaussian images, across position *and* scale, so each point comes with its own size.
- **Regions.** MSER finds regions that stay stable as an intensity threshold changes.

SIFT pairs its detector with a descriptor built from gradient orientations: 128 numbers per point, normalised for brightness, contrast and rotation. Matches are found by comparing descriptors.

The details are self-study: see the SIFT appendix. Lab 8 uses OpenCV's SIFT to find correspondences.

Szeliski Sec. 7.1.1 (difference-of-Gaussian, MSER), 7.1.2 (descriptors), 7.1.3 (matching).

Lecture 1 recap

- Good interest points are repeatable, distinctive, local.
- $\mathbf{M} = \sum_W w \nabla I \nabla I^\top$, the structure tensor.
- Both eigenvalues large $\Rightarrow$ corner. Harris and Shi-Tomasi are two scores on the same matrix.
- Non-maximum suppression is required. Neither detector is scale invariant.
- Matching needs descriptors too. Interest points include blobs and regions, not just corners. SIFT and the ratio test: self-study, in the appendix.

Lecture 2: Optical flow

The problem

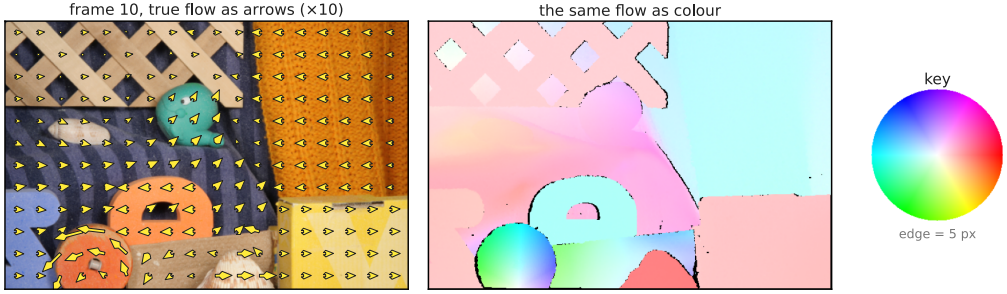
Given two frames, estimate the motion (u, v) of each point.

Applications: tracking, video stabilisation, structure from motion, compression, and, in Lab 7, separating moving vehicles from a static background.

We need an assumption to make it tractable.

What optical flow looks like

Optical flow gives every pixel a motion (u, v) between two frames. Left: the true flow as arrows, drawn 10 times longer. Middle: the same flow as colour. The hue gives the direction and the saturation the speed; white means no motion. The spinning wheel (bottom right) shows every direction at once.



Middlebury optical flow benchmark, RubberWhale, frames 10 and 11, with ground-truth flow (Baker et al., IJCV 2011).

Lucas-Kanade method for optical flow estimation

Rests on three assumptions.

1. **Brightness constancy.** A point looks the same in both frames.
2. **Small motion.** Points move only a little between frames.
3. **Spatial coherence.** A point moves like its neighbours.

Assumption 1 gives one equation per pixel. Assumption 2 lets us linearise it. Assumption 3 lets us pool the equations over a window.





$$I_t \overset{?}{=} \frac{\partial I}{\partial t}$$

$$\frac{I_{t+1} - I_t}{t+1-t} = I_{t+1} - I_t = I_t$$

Optical flow constraint equation

A pixel keeps its intensity as it moves (**brightness constancy**):

$$I(x, y, t) = I(x + u,\ y + v,\ t + 1)$$

intensity of pixel at (x,y) at frame t

intensity of the same pixel at frame t+1.

First-order Taylor expansion of the right-hand side (**Small motion**):

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

giving the **optical flow constraint equation**

$$\boxed{I_x u + I_y v + I_t = 0}$$

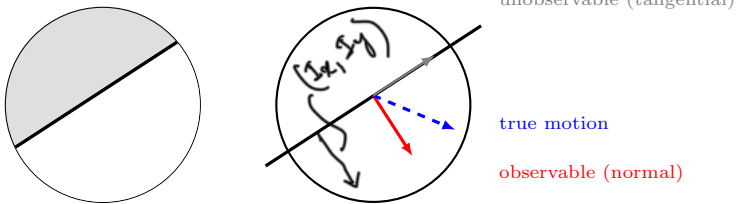
I_x, I_y are the spatial gradients of Week 4, taken in the first frame.
$I_t = I(x, y, t + 1) - I(x, y, t)$ is the difference between the two frames; time is counted in frames, so the time step is 1.

The aperture problem

- One equation, two unknowns, **per pixel**.
- The constraint says $\nabla I \cdot (u, v) = -I_t$.

$(I_x, I_y) \cdot (u, v) = -I_t$

Through a small window on a long straight edge, you cannot tell whether the edge is moving perpendicular to itself or sliding along.



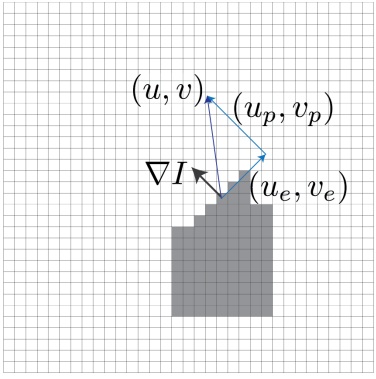
This is not a defect of the method. The information is genuinely absent.

Normal flow

Split the motion into a part **along the edge**, (u_e, v_e), and a part **perpendicular** to it, (u_p, v_p), which points along ∇I:

$$(u, v) = (u_e, v_e) + (u_p, v_p)$$

In the figure the dark arrow is (u, v) and the two light arrows are its parts.



∇I is perpendicular to the edge, so $\nabla I \cdot (u_e, v_e) = 0$ and the constraint becomes $\nabla I \cdot (u_p, v_p) = -I_t$.

We can recover (u_p, v_p), often called **normal flow**, but we can never recover (u_e, v_e), which never appears in the equation.

Lucas-Kanade

Assume the flow is **constant over a small window** W (**spatial coherence**). Then every pixel in W gives one equation:

$$I_x(\mathbf{p}_i)\, u + I_y(\mathbf{p}_i)\, v = -I_t(\mathbf{p}_i), \qquad \mathbf{p}_i \in W$$

Stack them. A 5×5 window gives 25 equations in 2 unknowns:

$$\underbrace{\begin{pmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{pmatrix}}_{\mathbf{A}} \begin{pmatrix} u \\ v \end{pmatrix} = \underbrace{- \begin{pmatrix} I_t(\mathbf{p}_1) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{pmatrix}}_{\mathbf{b}}$$

An overdetermined system. Solve it by least squares (Week 6).

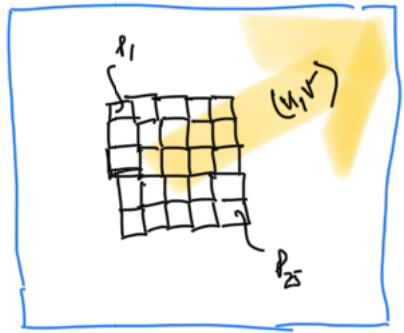
Window size is a trade-off. A bigger window gives more equations and more texture, so a better-conditioned system. But it is more likely to cover two different motions, which breaks the constant-flow assumption.

$$I_x u + I_y v + I_t = 0$$

$$\Rightarrow (I_x, I_y) \cdot (u, v) = -I_t$$

$$\Rightarrow \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t \end{bmatrix}$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t(p_1) \\ \\ I_t(p_{25}) \end{bmatrix}$$



$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ -I_t(p_3) \end{bmatrix}$$

$$A \qquad x \qquad b$$

$$A^T A = \begin{bmatrix} I_x(p_1) & I_x(p_2) & I_x(p_3) \\ I_y(p_1) & I_y(p_2) & I_y(p_3) \end{bmatrix} \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \sum_{p_1, p_2, p_3} I_x^2 & \sum_{p_1, p_2, p_3} I_x I_y \\ \sum_{p_1, p_2, p_3} I_x I_y & \sum_{p_1, p_2, p_3} I_y^2 \end{bmatrix}$$

The normal equations

Least squares gives $\mathbf{A}^\top \mathbf{A}\,(u\ v)^\top = \mathbf{A}^\top \mathbf{b}$, that is

$$\mathbf{M}\begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}, \qquad \mathbf{M} = \mathbf{A}^\top \mathbf{A} = \sum_W \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

$\mathbf{M}$ is the structure tensor from Lecture 1. It is exactly the same matrix when Lecture 1 uses a box window ($w = 1$). For a Gaussian window, weight each pixel's equation by w (weighted least squares), and you get $\sum_W w\,\nabla I\,\nabla I^\top$, Lecture 1's $\mathbf{M}$.

I_x, I_y are the gradients of the first frame; I_t is frame 2 minus frame 1.

Why $\mathbf{A}^\top\mathbf{A} = \mathbf{M}$

Row i of $\mathbf{A}$ is pixel $\mathbf{p}_i$'s gradient, $\mathbf{a}_i = \big(I_x(\mathbf{p}_i),\ I_y(\mathbf{p}_i)\big)$. Multiply out:

$$\mathbf{A}^\top\mathbf{A} = \begin{pmatrix} I_x(\mathbf{p}_1) & \cdots & I_x(\mathbf{p}_N) \\ I_y(\mathbf{p}_1) & \cdots & I_y(\mathbf{p}_N) \end{pmatrix} \begin{pmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ \vdots & \vdots \\ I_x(\mathbf{p}_N) & I_y(\mathbf{p}_N) \end{pmatrix} = \begin{pmatrix} \sum_i I_x^2 & \sum_i I_x I_y \\ \sum_i I_x I_y & \sum_i I_y^2 \end{pmatrix}$$

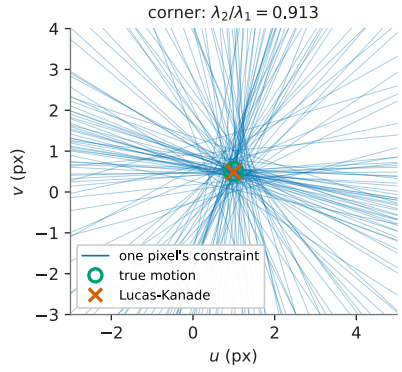
Each entry is a row of $\mathbf{A}^\top$ times a column of $\mathbf{A}$: a sum over the window. Equivalently, one outer product per pixel: $\mathbf{A}^\top\mathbf{A} = \sum_i \mathbf{a}_i^\top \mathbf{a}_i = \sum_i \nabla I(\mathbf{p}_i)\, \nabla I(\mathbf{p}_i)^\top$.

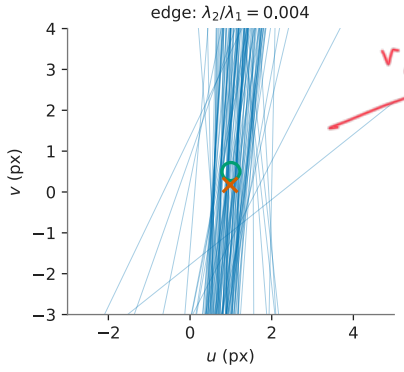
The right-hand side works the same way: $\mathbf{A}^\top\mathbf{b} = -\big(\sum_i I_x I_t,\ \sum_i I_y I_t\big)^\top$.

Gaussian window: weight pixel i's equation by w_i. Then $\mathbf{A}^\top \operatorname{diag}(w_1, \ldots, w_N)\, \mathbf{A} = \sum_i w_i\, \nabla I(\mathbf{p}_i)\, \nabla I(\mathbf{p}_i)^\top$, Lecture 1's $\mathbf{M}$ exactly.

Each pixel is a line

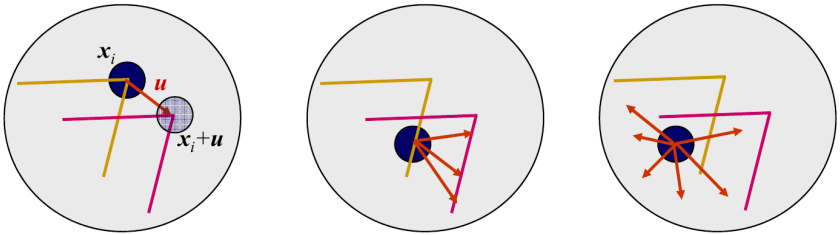
Each pixel's constraint $I_x u + I_y v + I_t = 0$ is a line in the (u, v) plane. Here the true motion is $(1, 0.5)$ px. Corner: the lines cross at one point, and Lucas-Kanade finds it. Edge: the lines are almost parallel, so v is poorly pinned down; Lucas-Kanade returns $v = 0.17$.





Three windows, three answers

Two frames overlaid: the first drawn in yellow, the second in red. The dark circle is the window. $\mathbf{x}_i$ is a pixel position and $\mathbf{u}$ the displacement, our (u, v). (a) A corner: one displacement fits. (b) An edge: a whole family of displacements fits equally well. (c) No texture: any displacement fits. In (b) and (c) each arrow is a different displacement that fits equally well; in (b) they differ only along the edge.



Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 7.4.

The connection between corner detection and optical flow

$\mathbf{M}$ must be **invertible and well conditioned** for the flow to be recoverable (both terms are defined in the appendix).

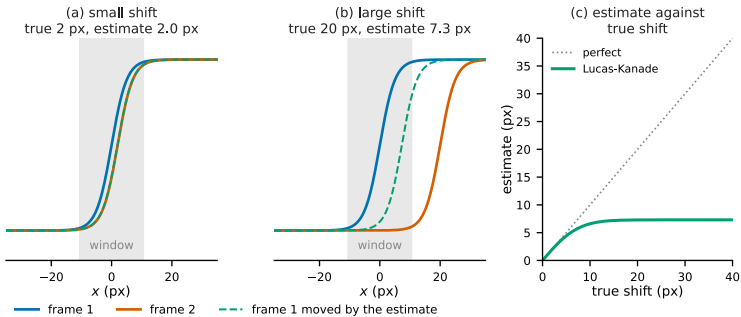
Window	λ_1, λ_2	Corner score	Flow
Flat	both ≈ 0	low	none recoverable
Edge	one large, one ≈ 0	low	normal component only
Corner	both large	high	fully recoverable

The corner criterion and the aperture problem are the same statement about $\mathbf{M}$.

Large λ_2 means a patch you can track. That is Shi and Tomasi's score, $\min(\lambda_1, \lambda_2) = \lambda_2$, and the title of their paper, *Good Features to Track*.

Dealing with large motions

Optical flow estimated by Lucas-Kanade with a 21-pixel window. The estimate is correct when the true motion is 2 px. It is inaccurate (7.3 px) when the actual motion is 20 px.

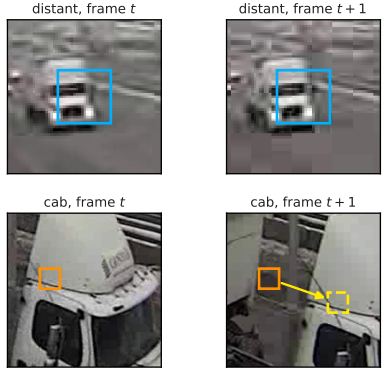


The derivation used a **first-order Taylor expansion**. It is valid only while the displacement is small relative to the window and the local texture scale.

Small and large motion in one video

Two consecutive frames. **Top:** a distant vehicle moves 0.5 px, so the 21×21 Lucas-Kanade window (blue) covers the same patch in both frames. **Bottom:** the truck cab moves 76 px. In frame $t+1$ the window (orange) now sees a pole and the road; the patch is far away (dashed).

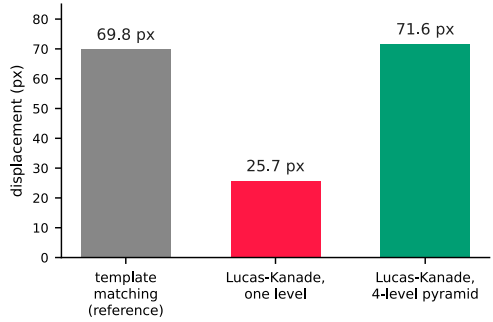


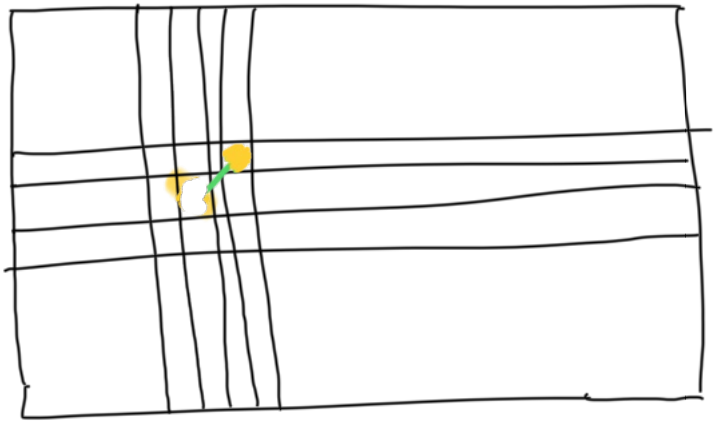


Lab 1 traffic video, frames 2999 and 3000. Displacements measured by template matching and pyramidal Lucas-Kanade.

Dealing with large motions

- **True motion**: ~ 75 px
- **Template matching**: try every shift of the patch in frame 2 (`cv.matchTemplate`) and keep the best normalised correlation (1 is a perfect match; here 0.76). No small-motion assumption.
- **Lucas-Kanade, one level**: about a third of the truth.
- **Lucas-Kanade, 4-level pyramid**: best estimate.





Iterative Lucas-Kanade, one level

Call the frames I (first) and J (second), so $J(x, y) = I(x, y, t + 1)$, and the current estimate $\mathbf{d} = (u, v) = (0, 0)$.

1. Compute I_x, I_y on the window in I, and $\mathbf{M}$, **once**. They do not change.
2. **Warp:** sample J at $(x + u,\ y + v)$ for every window pixel (x, y). These positions are not whole pixels, so use **bilinear interpolation**: a weighted average of the four nearest pixels, each weighted by how close it is.
3. Recompute $I_t = J(x + u,\ y + v) - I(x, y)$ over the window.
4. Solve the normal equations for an update $\Delta\mathbf{d}$, and set $\mathbf{d} \leftarrow \mathbf{d} + \Delta\mathbf{d}$.
5. Stop when $\|\Delta\mathbf{d}\| < 0.01$ px, or after 30 iterations. Otherwise go to 2.

Each pass removes most of what is left, as long as what is left is small.

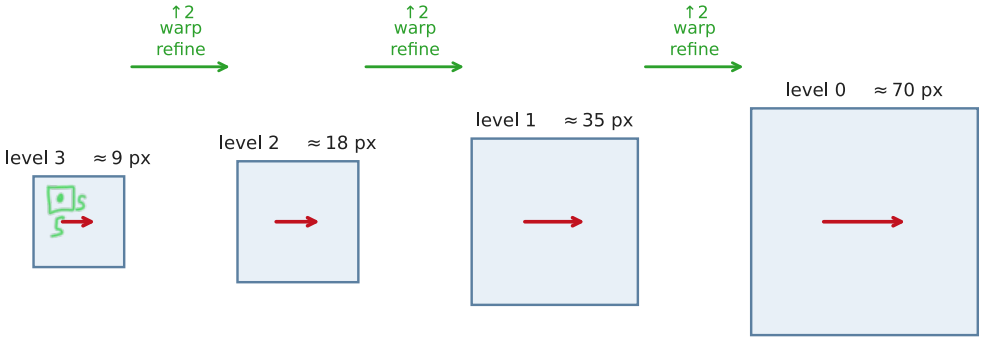
$$d = \begin{bmatrix} u \\ v \end{bmatrix}$$

Coarse-to-fine Lucas-Kanade

Gaussian pyramid: build it for **each** frame: blur with a Gaussian, keep every second row and column, repeat. At level k a 70 px motion is $70/2^k$ px; at level 3 that is about 9 px, which *is* small enough.

1. At the coarsest level, start from $\mathbf{d} = (0, 0)$. The point sits at $(x, y)/2^k$ on level k.
2. Run iterative Lucas-Kanade there, starting from $\mathbf{d}$.
3. Move one level finer. Pixels there are half the size, so **double the vector**: $\mathbf{d} \leftarrow 2\mathbf{d}$. The warp now starts from this guess, and only a small residual is left.
4. Repeat down to level 0.

Coarse-to-fine, as a picture



Lab 7 numbers, one traffic sequence:

true displacement ≈ 70 px (template matching)

single-scale Lucas-Kanade ≈ 26 px (patch has left the window)

pyramidal Lucas-Kanade ≈ 72 px (each level's residual is small)

1. Compute I_x, I_y, I_t
2. Compute M (use 5x5 window)
3. Compute: $M\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t \end{bmatrix}$ → solve $\begin{bmatrix} u \\ v \end{bmatrix}$.
4. $J[x+u, y+v] = I[x, y]$ (warping)
5. I_t (J, I)
6. Estimate (u, v)
7. Check $|u, v| < 0.01$

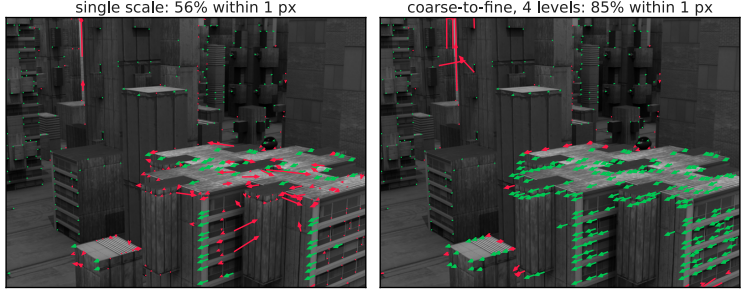
I^0 I^1

$I_t = I^1 - I^0$

$I_t = I^1 - J^0$

Coarse-to-fine on a real sequence

Lucas-Kanade (15×15 window) at 400 Shi-Tomasi corners, checked against the true flow. Green: within 1 px; red: wrong. Motions reach 22 px. Of the 188 corners that move more than 5 px, single scale gets 32% right; coarse-to-fine gets 94%. Why only 85% overall? Most remaining misses sit on a motion boundary: their window holds two motions, so constant flow fails at any scale. Corners whose window holds one motion are 97% right.



Middlebury optical flow benchmark, Urban2 (Baker et al., IJCV 2011).

Tracking a feature through a film

A traffic sign tracked through 21 frames of Woody Allen's *Manhattan*.
The camera moves, so the sign shifts and grows.



Figure 1: Three frame details from Woody Allen's *Manhattan*. The details are from the 1st, 11th, and 21st frames of a subsequence from the movie.

Bottom row: each tracked window warped back with an affine model (shift plus stretch). The windows match, so the tracker stayed on the sign.



Figure 2: The traffic sign windows from frames 1,6,11,16,21 as tracked (top), and warped by the computed deformation matrices (bottom).

J. Shi and C. Tomasi, "Good Features to Track," CVPR 1994, Figs. 1 and 2; via F. Z. Qureshi, *Computer Vision Notes*, notebook 20.

Failure modes worth knowing

- **Textureless regions:** $\mathbf{M} \approx 0$, nothing recoverable. The eigenvalues warn you in advance.
- **Repetitive structure:** a railing, a picket fence. The cost has several near-equal minima, so the estimate can be confidently wrong. $\mathbf{M}$ looks healthy here, which is what makes this dangerous.
- **Occlusion and disocclusion:** the point genuinely has no correspondence.
- **Brightness changes:** the constancy assumption simply fails.

A tracker that does not know when it is wrong is worse than one that reports fewer, reliable matches.

Week 8 recap

- Interest points: repeatable, distinctive, local.
- $\mathbf{M} = \sum_W w \nabla I \nabla I^\top$ decides cornerness *and* flow recoverability.
- Harris and Shi-Tomasi; non-maximum suppression; not scale invariant.
- Descriptors and matching (SIFT, ratio test): self-study, in the appendix.
- Brightness constancy $\Rightarrow I_x u + I_y v + I_t = 0$; the aperture problem.
- Lucas-Kanade solves it over a window, using the same $\mathbf{M}$.
- Large motion needs iteration and coarse-to-fine.

This week's work

- **Reading:** Szeliski Sec. 7.1.1–7.1.3, 7.1.5, 9.3.
- **Self-study:** the SIFT appendix, before Lab 8.
- **Exercises:** `w08a` (corner detection), `w08b` (optical flow).
- **Lab 7:** Harris and Shi-Tomasi, then Lucas-Kanade on the Lab 1 traffic video (fixed-camera highway footage).

Next week: with correspondences in hand, we can finally relate two views geometrically: homographies and epipolar geometry.

Appendix: SIFT, for self-study

These slides are not covered in lecture. Read them before Lab 8, where you match SIFT descriptors with the ratio test.

- How SIFT finds points at their own scale
- Constructing the 128-number descriptor
- Why the sampling grid must rotate
- Matching descriptors, and Lowe's ratio test

Reading: Szeliski Sec. 7.1.1–7.1.3.

How SIFT handles scale

SIFT's detector searches over scale as well as position.

1. Blur the image with Gaussians of increasing σ, and halve the image size after each doubling of σ: a pyramid.
2. Subtract neighbouring blurs. This gives difference-of-Gaussian images.
3. Keep a point if it is larger, or smaller, than all 26 neighbours: 8 in its own image and 9 in each of the two neighbouring scales.

Each point then comes with its own scale, and the descriptor is computed at that scale (Lowe 2004, Sec. 3; Szeliski Sec. 7.1.1). OpenCV provides it as `cv.SIFT_create`.

Finding the point is not enough

Detection gives *where*. Matching needs *what it looks like*: a **descriptor**, a vector summarising the patch.

Raw pixels are a poor descriptor: sensitive to brightness, contrast and rotation.

Better: describe the patch by its **gradient orientations**, which are already invariant to additive brightness change, and can be normalised for contrast.

A SIFT-like descriptor

1. Sample a 16×16 grid of points around the point (step 4 rotates it, so gradients are read from a region of at least 23×23).
2. Compute gradient magnitude and orientation at every pixel.
3. Weight magnitudes by a Gaussian centred on the patch, with σ half the patch width ($\sigma = 8$ px).
4. Find the **dominant orientation** θ_{dom}. Rotate the sampling grid by θ_{dom}, and subtract it from every gradient orientation.
5. Split into a 4×4 grid of cells; in each, build an 8-bin orientation histogram ($45°$ per bin) weighted by magnitude.
6. Concatenate: $4 \times 4 \times 8 = 128$ values. Normalise to unit length.

Step 4 buys rotation invariance and the normalisation in Step 6 handles contrast.

Step 4 in detail

Finding θ_{dom}. Build a 36-bin histogram of gradient orientations (10° per bin) over the unrotated 16×16 patch. Each pixel votes with its gradient magnitude times step 3's Gaussian ($\sigma = 8$ px). θ_{dom} is the highest peak.

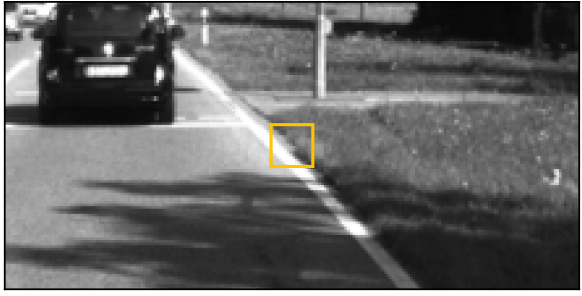
Rotating the grid. Place the 16×16 sample points on a grid rotated by θ_{dom} about the keypoint. They no longer land on pixel centres, so read the gradients there with bilinear interpolation. The rotated corners reach $8\sqrt{2} \approx 11.3$ px from the centre, so read from a region of at least 23×23 pixels, or the corners come out empty.

Then subtract θ_{dom} from each sampled orientation modulo 360° and build the histograms of step 5.

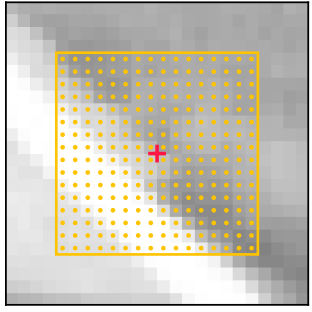
SIFT step 1: the sample grid

A keypoint at the end of a lane marking in a KITTI frame. Its 16×16 sample points sit on the pixel centres around it.

the keypoint, in the frame



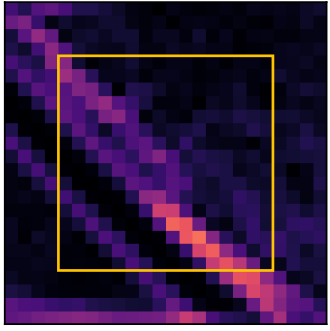
its 16×16 sample points



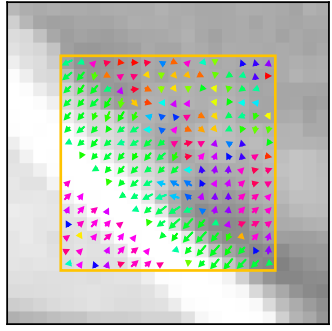
SIFT step 2: a gradient at every sample

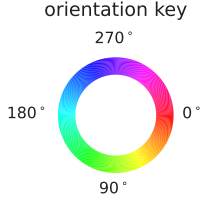
Gradients by central differences on the whole image. Each arrow is one sample's gradient; its colour gives the orientation ($90°$ points down). Arrow length goes as the square root of the magnitude, so weak gradients show.

gradient magnitude



gradient at each sample

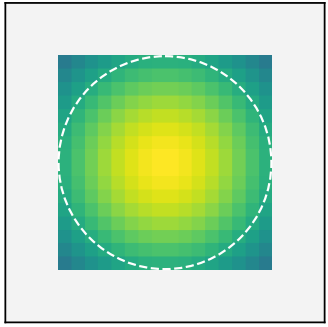




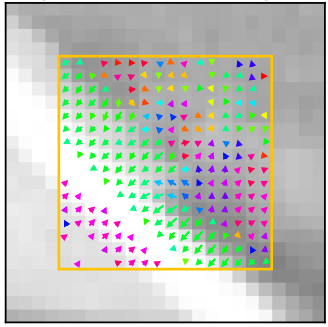
SIFT step 3: weight by a Gaussian

Each magnitude is multiplied by a Gaussian with $\sigma = 8$ px. Samples near the keypoint count most. Those near the edge, most hurt by a slightly misplaced window, count least, and a small shift of the window changes the descriptor only a little.

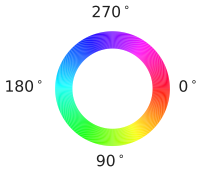
Gaussian weight, $\sigma = 8$ px (dashed)



gradients times the weight

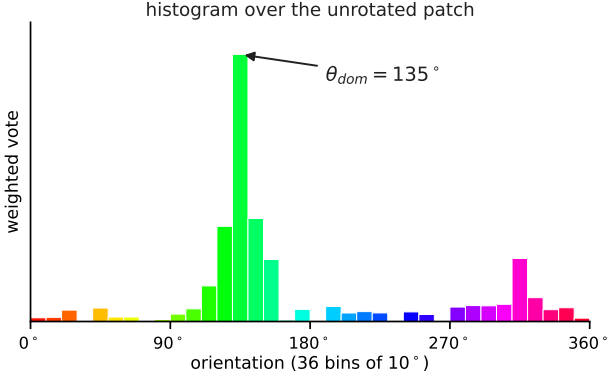


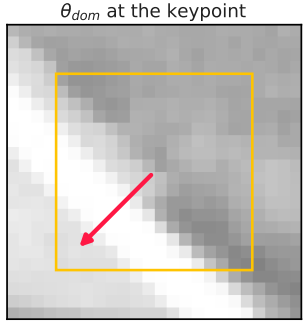
orientation key



SIFT step 4: the dominant orientation

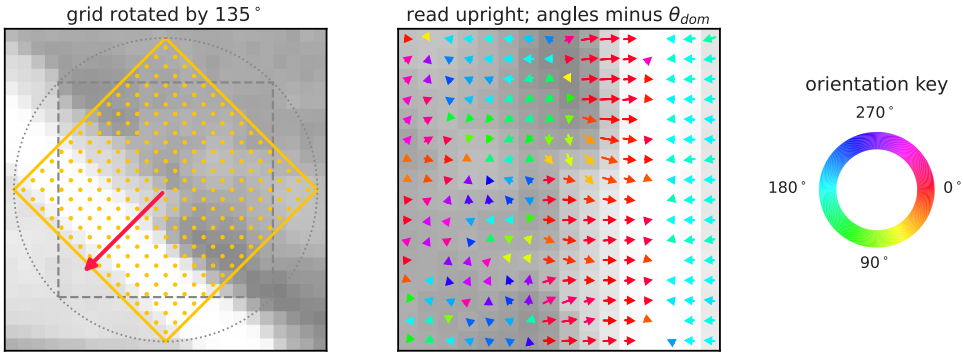
The 36-bin histogram peaks in the bin $[130°, 140°)$, so $\theta_{\text{dom}} = 135°$: the gradient across the lane marking's edge.





SIFT step 4: rotate the grid

Left: the grid rotated by 135° about the keypoint, read by bilinear interpolation; its corners reach $8\sqrt{2} \approx 11.3$ px (dotted circle). Right: what the rotated grid reads, drawn upright, with θ_{dom} subtracted. The edge's gradients now point at 0°.

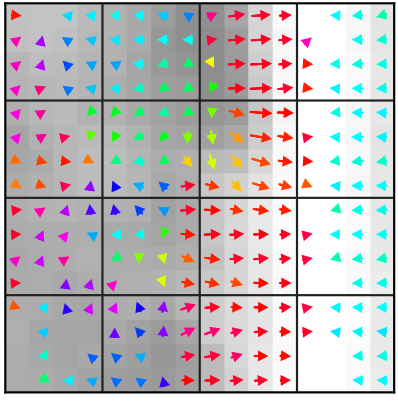


Aside: θ_{dom} snaps to a 10° bin and votes jump between bins, which Lowe smooths by interpolation.

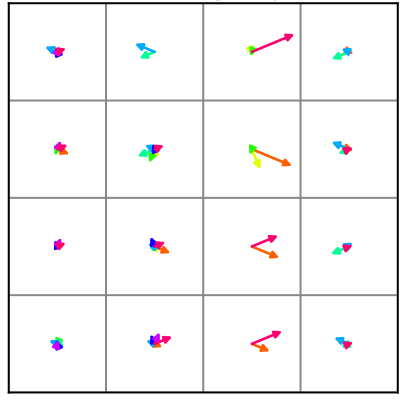
SIFT step 5: 4×4 cells, 8 bins each

The 16 samples in each cell vote into 8 bins of $45°$, each vote weighted as in step 3. Right: each arrow is one bin, pointing at its angle, with length going as the square root of its vote.

4 × 4 cells of 4 × 4 samples

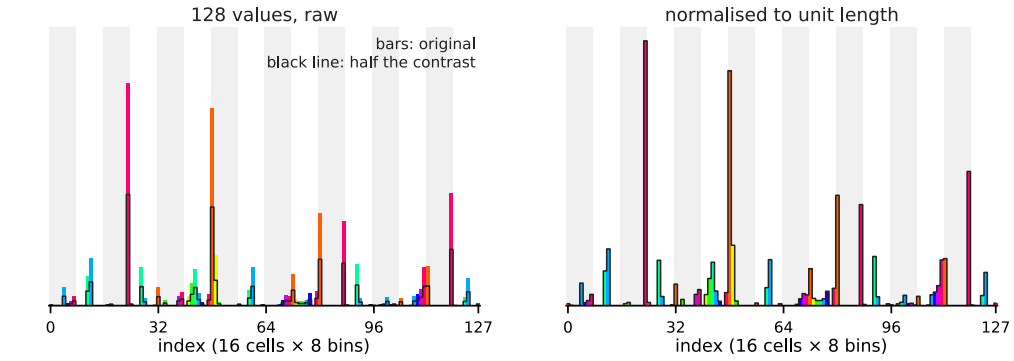


one 8-bin histogram per cell



SIFT step 6: 128 numbers

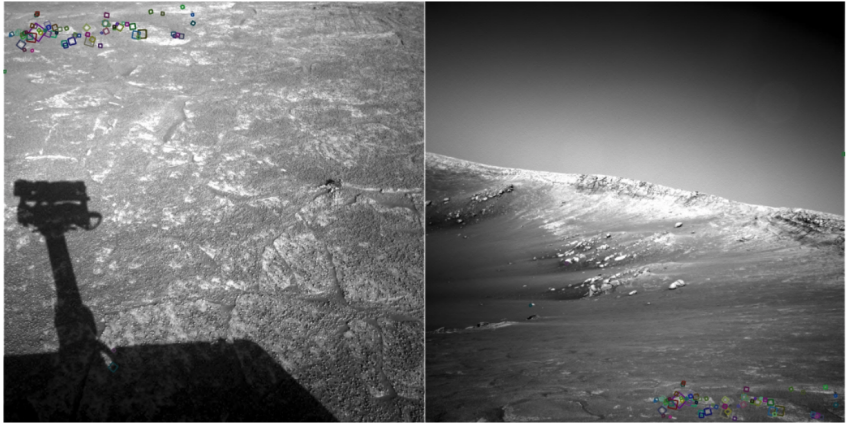
Concatenate the 16 histograms: $16 \times 8 = 128$ values (grey bands mark alternate cells; bar colour is the bin's angle).



At half the contrast every value halves (left). After normalising to unit length the two coincide (right). This improves invariance to changes in contrast.

SIFT on Mars

Two images from a NASA Mars rover. The coloured squares are SIFT features matched between them.



NASA Mars rover images; figure by N. Snavely; via F. Z. Qureshi, *Computer Vision Notes*, notebook 11.

Matching descriptors

Compare by Euclidean distance. But a nearest neighbour always exists, even when there is no true match.

Lowe's ratio test: accept a match only if

$$\frac{d_1}{d_2} < 0.7\text{–}0.8$$

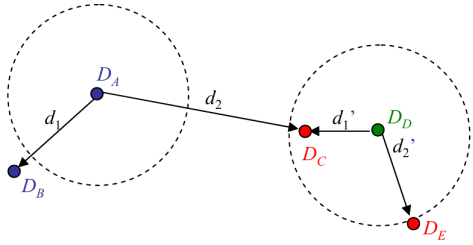
where d_1, d_2 are the distances to the best and second-best candidates.

The reasoning: a *correct* match should be clearly better than the runner-up. If the two are similar, the region is ambiguous (repeated texture, say) and the match is unreliable.

The ratio test, visually

D_B is D_A's true match; D_D has no true match. A fixed distance threshold (dashed circles) fails both ways: D_A misses D_B, and D_D matches both D_C and D_E.

- D_A matches with D_B since ratio d_1/d_2 is small
- D_D does not match with D_C or D_E since ratio d_1'/d_2' is close to 1.0



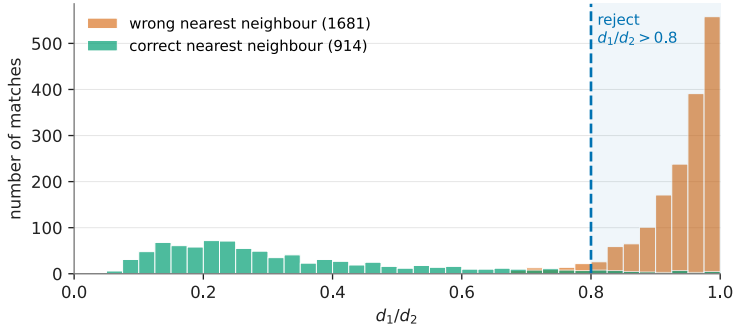
Despite this wrong matches can get through. Therefore, we need RANSAC. Recall that wrong matches act as *outliers*.

Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 7.23.

An experiment to see the efficacy of the ratio test

Setup: take a KITTI frame I. Now create a version J by rotating I by 30°, scaling by 0.8, and adding noise. Match descriptors across I and J. We know the true matches, since we constructed J.

Results: With a ratio threshold of 0.8 the test keeps 95% of the correct matches and removes 96% of the wrong ones. Still, 72 of the 944 survivors are wrong.



Appendix: types of matrix

$\mathbf{B}$ is a square matrix; $\mathbf{x}$ is a non-zero vector.

Type	Definition	Eigenvalues	Example
Symmetric	$\mathbf{B}^\top = \mathbf{B}$	real	$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
Positive definite	$\mathbf{x}^\top \mathbf{B}\, \mathbf{x} > 0$ for all $\mathbf{x}$	all > 0	$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$
Positive semi-definite	$\mathbf{x}^\top \mathbf{B}\, \mathbf{x} \geq 0$ for all $\mathbf{x}$	all ≥ 0	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$
Indefinite	$\mathbf{x}^\top \mathbf{B}\, \mathbf{x}$ takes both signs	some > 0, some < 0	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Singular	$\mathbf{B}\mathbf{x} = \mathbf{0}$ for some $\mathbf{x}$	at least one $= 0$	$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$

A symmetric matrix's eigenvectors can be chosen orthogonal. The eigenvalue tests for definiteness hold for symmetric matrices. A matrix can be of several types at once: the third and fifth examples are symmetric, positive semi-definite and singular.

Appendix: what the types mean for $\mathbf{M}$

$\mathbf{M}$ is always **symmetric** and **positive semi-definite**: its two eigenvalues are real and $\lambda_1 \geq \lambda_2 \geq 0$. Which further type it is tells us the kind of window:

- **Corner:** λ_2 large, well away from singular. E rises in every direction.
- **Edge:** nearly singular, $\lambda_2 \approx 0$. Along the eigenvector of λ_2, which points along the edge, $E \approx 0$.
- **Flat:** $\mathbf{M} \approx \mathbf{0}$, $\lambda_1 \approx \lambda_2 \approx 0$. $E \approx 0$ in every direction.

$\mathbf{M}$ is never indefinite: its quadratic form is exactly a weighted sum of squares, $(u\ v)\,\mathbf{M}\,(u\ v)^\top = \sum_W w\,(I_x u + I_y v)^2 \geq 0$, since $w \geq 0$.

Appendix: determinant and trace

For a 2×2 matrix $\mathbf{B} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

$$\det \mathbf{B} = ad - bc, \qquad \operatorname{tr} \mathbf{B} = a + d \quad \text{(the sum of the diagonal)}$$

Both are tied to the eigenvalues. The eigenvalues solve $\det(\mathbf{B} - \lambda \mathbf{I}) = (a - \lambda)(d - \lambda) - bc = 0$, that is

$$\lambda^2 - (\operatorname{tr} \mathbf{B})\, \lambda + \det \mathbf{B} = 0 \quad \Longrightarrow \quad \det \mathbf{B} = \lambda_1 \lambda_2, \qquad \operatorname{tr} \mathbf{B} = \lambda_1 + \lambda_2$$

Example: $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ has $\det = 3$, $\operatorname{tr} = 4$, and eigenvalues 3 and 1.

For $\mathbf{M}$: $\det \mathbf{M} = \big(\sum_W w I_x^2\big)\big(\sum_W w I_y^2\big) - \big(\sum_W w I_x I_y\big)^2$ and $\operatorname{tr} \mathbf{M} = \sum_W w\,(I_x^2 + I_y^2)$. Harris's R needs only these two. Solving the quadratic gives $\lambda = \frac{1}{2} \operatorname{tr} \mathbf{M} \pm \sqrt{(\frac{1}{2} \operatorname{tr} \mathbf{M})^2 - \det \mathbf{M}}$; the smaller root (minus sign) is the Shi-Tomasi response.

Appendix: invertible and well-conditioned

Invertible. A square matrix $\mathbf{B}$ is invertible if some $\mathbf{B}^{-1}$ gives $\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}$. Then $\mathbf{B}\mathbf{x} = \mathbf{c}$ has exactly one solution, $\mathbf{x} = \mathbf{B}^{-1}\mathbf{c}$. Equivalently: $\det \mathbf{B} \neq 0$, and no eigenvalue is zero. A matrix that is not invertible is **singular**.

Well-conditioned. Invertible is yes or no; conditioning asks *how safely*. For a symmetric positive definite matrix the **condition number** is

$$\kappa(\mathbf{B}) = \frac{\lambda_1}{\lambda_2} \geq 1 \qquad \text{(largest over smallest eigenvalue)}$$

In general $\kappa = \sigma_{\max}/\sigma_{\min}$, the ratio of singular values (deck 06); for such a matrix the two agree. A small relative error in $\mathbf{c}$ can grow by up to a factor of κ in the solution $\mathbf{x}$. κ near 1: well-conditioned. κ large: ill-conditioned, so noise is amplified.

Example. $\mathbf{B} = \begin{pmatrix} 1 & 0 \\ 0 & 0.001 \end{pmatrix}$ is invertible, but $\kappa = 1000$. For $\mathbf{c} = (1,\ 0.001)$, $\mathbf{x} = (1,\ 1)$. Change c_2 by just 0.001, to 0.002, and x_2 doubles, to 2. A 0.1% change in $\mathbf{c}$ became a 71% change in $\mathbf{x}$: a factor of about 707, within the bound of 1000.

Appendix: conditioning and $\mathbf{M}$

Lucas-Kanade solves $\mathbf{M}\,(u\ v)^\top = -\big(\sum I_x I_t,\ \sum I_y I_t\big)^\top$. Both $\mathbf{M}$ and the right-hand side are built from noisy gradients, so $\mathbf{M}$ must be invertible **and** well-conditioned. Our three windows (31×31, $w = 1$, intensities 0 to 255):

Window	λ_1	λ_2	$\kappa = \lambda_1/\lambda_2$	Flow
Corner	854,000	791,000	1.08	reliable
Edge	791,000	3,250	243	noisy along the edge
Flat	173	131	1.32	unreliable

The flat window shows that κ alone is not enough. It is well-conditioned, but both eigenvalues are so small that image noise swamps them. We need λ_2 itself to be large: exactly the Shi-Tomasi criterion.

Copyright and License

©Faisal Z. Qureshi



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Third-party figures are not covered by this licence. Each is credited where it appears, and remains under its owner's terms.