Homography and Epipolar Geometry

Computer Vision I (CSCI 3240U)

Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca



Week 9 at a glance

- **Lecture 1: Homography (Sec. 8.1, 8.2.1)**
  - 2D transformations and their degrees of freedom
  - When one matrix relates two views
  - Estimating $\mathbf{H}$: the direct linear transform (DLT)
  - Warping, done backwards, and stitching
- **Lecture 2: Epipolar geometry (Sec. 12.1, 11.3.1)**
  - Two views of a general scene
  - The epipolar constraint
  - Essential and fundamental matrices
  - Rectification

Reading: Szeliski Sec. 8.1, 8.2.1, 12.1, and the eight-point algorithm in 11.3.1. Exercise `w09a`. **Lab 8** is this week's payoff.

Notation used in this deck

- $\mathbf{p} = (x, y)$ is a pixel; $\tilde{\mathbf{p}} = (x, y, 1)^\top$ is the same point in **homogeneous** coordinates. In Lecture 1, a tilde always means homogeneous.
- $\mathbf{a} \sim \mathbf{b}$ means **equal up to a nonzero scale**: $\mathbf{a} = \lambda \mathbf{b}$ for some $\lambda \neq 0$.
- $\pi(u, v, w) = (u/w,\ v/w)$ turns a homogeneous 3-vector back into a pixel: **divide by the third coordinate**.
- A prime marks the second image or camera: $\tilde{\mathbf{p}}'$, $\mathbf{K}'$, $\mathbf{X}'$.
- $\mathbf{X} = (X, Y, Z)^\top$ is a 3D world point in camera-1 coordinates, with depth Z; $\mathbf{X}'$ is the same point in camera-2 coordinates.
- $\hat{\mathbf{p}} = \mathbf{K}^{-1}\tilde{\mathbf{p}}$ is in **normalised camera coordinates** (a hat): the direction of the ray through the pixel, with the intrinsics $\mathbf{K}$ removed.
- Two letters do double duty. Bold $\mathbf{e}$ is an epipole; italic e is RANSAC's outlier fraction. d is RANSAC's inlier threshold; d_Π is a plane's distance from the camera.
- **Lecture 2 switches to Szeliski's symbols** and opens with its own table.

Lecture 1: Homography

A hierarchy of 2D transformations

Transformation	Matrix	DOF	Preserves
Translation	$[\mathbf{I} \mid \mathbf{t}]$	2	orientation, lengths, angles
Euclidean (rigid)	$[\mathbf{R} \mid \mathbf{t}]$	3	lengths, angles
Similarity	$[s\mathbf{R} \mid \mathbf{t}]$	4	angles, ratios of lengths
Affine	$[\mathbf{A} \mid \mathbf{t}]$	6	parallelism, ratios along a line
Projective (homography)	$\mathbf{H}$	**8**	**straight lines**

$\mathbf{I}$, $\mathbf{R}$, $\mathbf{A}$ are 2×2, $\mathbf{t}$ is 2×1; each 2×3 form gains a bottom row $(0\ 0\ 1)$. Counting: $\mathbf{t}$ gives 2; a rotation adds one angle θ (3); a scale s adds 1 (4); a general $\mathbf{A}$ has 4 entries (6); $\mathbf{H}$ has 9 entries, less 1 for scale (8).

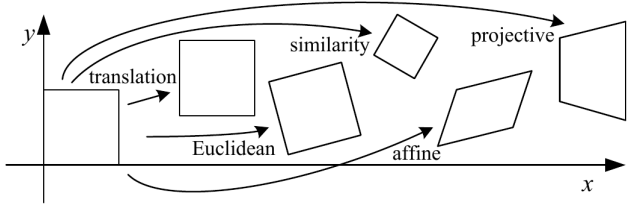
All five are 3×3 matrices; they differ only in how constrained that matrix is. The projective case preserves almost nothing, but it does preserve straightness, and that is enough to be useful.

Szeliski Sec. 2.1.1, Table 2.1.

A hierarchy of 2D transformations

Each model applied to the same square:

- **Translation** moves it;
- **Euclidean** (rigid) also turns (rotates) it;
- **similarity** also scales it;
- **affine** skews it into a parallelogram, so opposite sides stay parallel; and
- **projective** makes a general quadrilateral, and only the straightness of the sides survives.



Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 2.4.

Recap: the intrinsic camera matrix, $\mathbf{K}$

$$\mathbf{K} = \begin{pmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- α_x, α_y: focal length in pixels; (c_x, c_y): principal point; s: skew, in practice 0.
- $\mathbf{K}$ takes a point in **camera coordinates** to its pixel: $\tilde{\mathbf{p}} \sim \mathbf{K}\mathbf{X}$.
- $\mathbf{K}^{-1}$ goes back: $\mathbf{K}^{-1}\tilde{\mathbf{p}}$ is the direction of the ray through the pixel ($\hat{\mathbf{p}}$ in this deck's notation).

$\mathbf{K}$ is **intrinsic**: moving or turning the camera leaves it unchanged; zooming changes α_x, α_y. The second camera's intrinsics are $\mathbf{K}'$, equal to $\mathbf{K}$ for the same camera at the same zoom.

Week 2, "The calibration matrix". Szeliski Sec. 2.1.4, Eq. (2.57), writes f_x, f_y for α_x, α_y.

Recap: the inverse of $\mathbf{K}$

Recall for square matrices: $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$

With $s = 0$, $\mathbf{K}$ is a scaling followed by a shift. Undo them in the opposite order:

$$\mathbf{K} = \underbrace{\begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{shift}} \underbrace{\begin{pmatrix} \alpha_x & 0 & 0 \\ 0 & \alpha_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{scale}} \qquad \mathbf{K}^{-1} = \begin{pmatrix} 1/\alpha_x & 0 & 0 \\ 0 & 1/\alpha_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix}$$

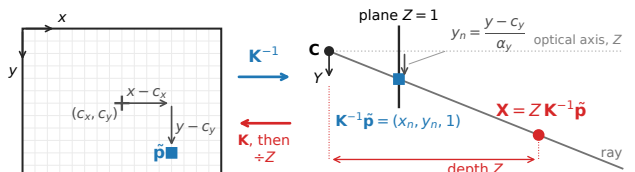
$$\boxed{\mathbf{K}^{-1} = \begin{pmatrix} 1/\alpha_x & 0 & -c_x/\alpha_x \\ 0 & 1/\alpha_y & -c_y/\alpha_y \\ 0 & 0 & 1 \end{pmatrix}}$$

$\mathbf{K}^{-1}$ always exists: $\det \mathbf{K} = \alpha_x \alpha_y \neq 0$.

Recap: $\mathbf{K}^{-1}$ and $\mathbf{K}$: pixels to rays and back

$\mathbf{X}$ is a point in camera coordinates.

- **Point to pixel:** $\mathbf{K}\mathbf{X} = Z\,\tilde{\mathbf{p}}$
  - To project onto the image plane, apply $\mathbf{K}$ to $\mathbf{X}$, then divide by Z.
- **Pixel to ray:** $\mathbf{K}^{-1}\tilde{\mathbf{p}} = (x_n, y_n, 1)^\top$, the point at depth $Z = 1$ on the ray from the camera centre $\mathbf{C}$ (the origin) through the pixel.
  - Every nonzero multiple of $\mathbf{K}^{-1}\tilde{\mathbf{p}}$ is also a point on this ray.
- **Ray to point:** the depth picks the point, $\mathbf{X} = Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}} = (Zx_n,\ Zy_n,\ Z)^\top$.



image, in pixels

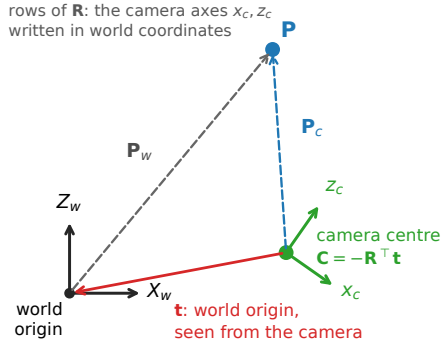
camera from the side (*x* works the same way)

Recap: the camera pose $\mathbf{R}$, $\mathbf{t}$

- **From world coordinates to camera coordinates:**

$$\mathbf{P}_c = \mathbf{R}\mathbf{P}_w + \mathbf{t}$$

- $\mathbf{R}$ is a 3×3 rotation: $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$ and $\det \mathbf{R} = 1$, so $\mathbf{R}^{-1} = \mathbf{R}^\top$.
- $\mathbf{t}$ is a 3-vector: the world origin, seen from the camera. It is **not** the camera's position. The camera centre is where $\mathbf{P}_c = \mathbf{0}$: $\mathbf{C} = -\mathbf{R}^\top \mathbf{t}$.



Week 2, "The camera is not at the origin". Szeliski Sec. 2.1.2, Eqs. (2.24), (2.25); Sec. 2.1.4, Eq. (2.56).

From camera 1 to camera 2: the relative pose

Each camera has its own world-to-camera pose:

$$\mathbf{X} = \mathbf{R}_1 \mathbf{P}_w + \mathbf{t}_1 \qquad\qquad \mathbf{X}' = \mathbf{R}_2 \mathbf{P}_w + \mathbf{t}_2$$

Undo the first, $\mathbf{P}_w = \mathbf{R}_1^\top (\mathbf{X} - \mathbf{t}_1)$, then apply the second:

$$\mathbf{X}' = \underbrace{\mathbf{R}_2 \mathbf{R}_1^\top}_{\mathbf{R}} \mathbf{X} + \underbrace{\mathbf{t}_2 - \mathbf{R}_2 \mathbf{R}_1^\top \mathbf{t}_1}_{\mathbf{t}} \qquad \Longrightarrow \qquad \boxed{\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}}$$

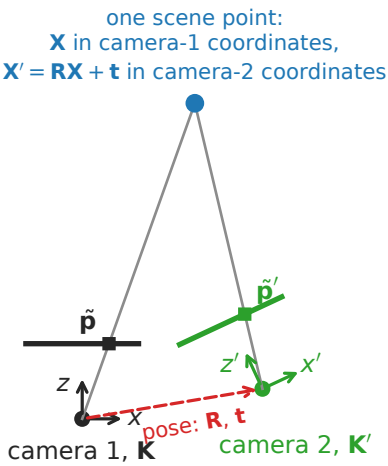
- This $\mathbf{R}, \mathbf{t}$ is the **relative pose**: camera 2 relative to camera 1. In general it is **not** either camera's world-to-camera pose.
- This week we put the world frame at camera 1: $\mathbf{R}_1 = \mathbf{I}$, $\mathbf{t}_1 = \mathbf{0}$. Then $\mathbf{R} = \mathbf{R}_2$, $\mathbf{t} = \mathbf{t}_2$.
- Homographies and $\mathbf{F}$ depend on the cameras only through $\mathbf{K}$, $\mathbf{K}'$ and the relative pose, never on where the world origin is.

Recap: $\mathbf{K}$, $\mathbf{R}$, $\mathbf{t}$ together

From a pixel in camera 1 to the pixel in camera 2:

1. $\mathbf{K}^{-1}$: pixel $\tilde{\mathbf{p}}$ to the ray through it.
2. $\times Z$: ray to the point $\mathbf{X} = Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}}$. **This step needs the depth.**
3. $\mathbf{R}, \mathbf{t}$: camera-1 to camera-2 coordinates, $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$.
4. $\mathbf{K}'$: point to pixel, $\tilde{\mathbf{p}}' \sim \mathbf{K}'\mathbf{X}'$.

$\mathbf{K}$ and $\mathbf{K}'$ convert between pixels and rays; $\mathbf{R}$ and $\mathbf{t}$ move between cameras.



Four steps, one matrix

Chain the four steps, applied right to left:

$$\tilde{\mathbf{p}}' \sim \underbrace{\mathbf{K}'}_{\text{step 4}} \Big(\underbrace{\mathbf{R}}_{\text{step 3}} \underbrace{Z}_{\text{step 2}} \underbrace{\mathbf{K}^{-1}}_{\text{step 1}} \tilde{\mathbf{p}} + \underbrace{\mathbf{t}}_{\text{step 3}} \Big)$$

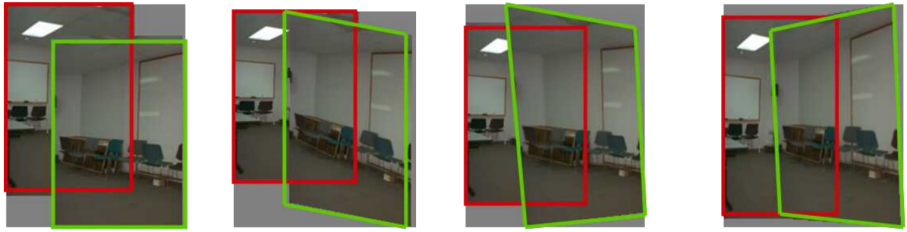
Let the camera only turn, $\mathbf{t} = \mathbf{0}$. Z is a scalar, so it moves to the front, and $\sim$ ignores a nonzero scale:

$$\tilde{\mathbf{p}}' \sim \mathbf{K}'\mathbf{R}\, Z\, \mathbf{K}^{-1}\tilde{\mathbf{p}} = Z\,(\mathbf{K}'\mathbf{R}\mathbf{K}^{-1})\,\tilde{\mathbf{p}} \sim \underbrace{\mathbf{K}'\mathbf{R}\mathbf{K}^{-1}}_{3\times 3}\,\tilde{\mathbf{p}}$$

- Matrix products are associative, so the three matrices multiply into **one** 3×3 **matrix**, computed once and used for every pixel.
- The depth has gone: we never need to know Z. This is the "3D rotation" model on the next slide.
- With $\mathbf{t} \neq \mathbf{0}$ the $+\mathbf{t}$ does not scale with Z, so Z cannot be pulled out. Appendix A works out what happens then.

The same models, aligning two photos

Each model maps the second photo (green frame) onto the first (red). Translation only shifts the frame; affine keeps opposite sides parallel; the perspective model makes it a general quadrilateral. 3D rotation is the homography $\mathbf{KRK}^{-1}$ of the previous slide, a homography restricted to 3 rotation angles, plus the focal length if it is unknown: hence "3+" rather than 8.



(a) translation [2 dof] (b) affine [6 dof] (c) perspective [8 dof] (d) 3D rotation [3+ dof]

Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 8.4 and Sec. 8.2.3.

The homography

- $\mathbf{p} = (x, y)$: a pixel in **image 1**.
- $\mathbf{p}' = (x', y')$: the pixel in **image 2** where the same scene point appears. $\mathbf{p}$ and $\mathbf{p}'$ form a **correspondence**.
- $\tilde{\mathbf{p}} = (x, y, 1)^\top$ and $\tilde{\mathbf{p}}' = (x', y', 1)^\top$ are the same pixels in homogeneous coordinates.

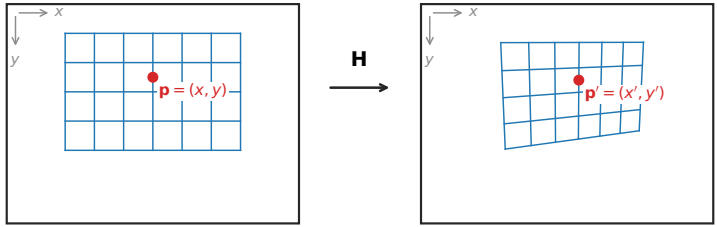
A homography maps every $\mathbf{p}$ to its $\mathbf{p}'$ with one 3×3 matrix:

$$\tilde{\mathbf{p}}' \sim \mathbf{H}\,\tilde{\mathbf{p}}, \qquad \mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

The $\sim$ hides a division: multiply, then divide by the third coordinate, $\mathbf{p}' = \pi(\mathbf{H}\tilde{\mathbf{p}})$.

The homography, drawn

A square grid in image 1 and its image under one $\mathbf{H}$. Straight lines stay straight; parallel lines need not stay parallel.



Given:

$$\mathbf{H} = \begin{pmatrix} 1.2 & 0.1 & 20 \\ 0.05 & 1.1 & 10 \\ 0.001 & 0.0005 & 1 \end{pmatrix}$$

For the pixel marked:

$$\mathbf{H}\tilde{\mathbf{p}} = \mathbf{H}\,(200,\ 100,\ 1)^\top = (270,\ 130,\ 1.25)^\top$$
$$\mathbf{p}' = (270/1.25,\ 130/1.25) = (216,\ 104)$$

The homography: degrees of freedom

Nine entries, but homogeneous coordinates are defined **up to scale**, so $\mathbf{H}$ and $\lambda\mathbf{H}$ are the same map.

$$\boxed{8 \text{ degrees of freedom}}$$

Each point correspondence gives two equations (one for x', one for y'), so

$$8/2 = \boxed{4 \text{ correspondences}}$$

with no three collinear. The next two slides show why.

Four correspondences fix $\mathbf{H}$

- Four numbered points in image 1 and their matches in image 2.
- $\mathbf{H}$ was estimated from **these four pairs alone**, then applied to the whole grid.
- Four pairs give $4 \times 2 = 8$ equations for the 8 degrees of freedom.
  - The $\mathbf{H}$ that fits them is unique up to scale.

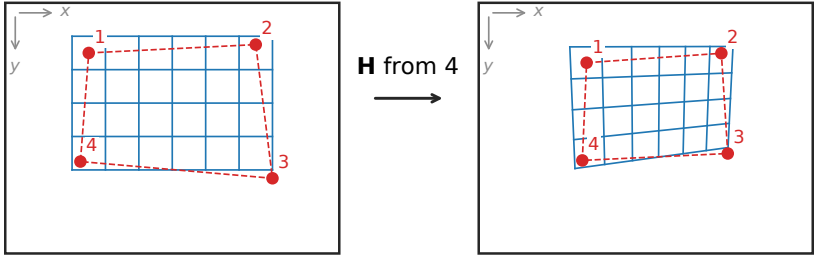


image 1 image 2

How we estimate $\mathbf{H}$ from four correspondences comes later.

Three collinear points are not enough

- **Centre:** image 1, with points 1, 2 and 3 on one line.
- **Left and right:** image 2, mapped by two different homographies, $\mathbf{H}$ and $\mathbf{H}'$ (Appendix B shows how $\mathbf{H}'$ was built).
- Both map the four points exactly, but they send the rest of the grid, and the grey cell, to different places.
  - Why: a homography always maps a line to a line, so point 2 adds one equation, not two. That leaves 7 equations for 8 degrees of freedom, so $\mathbf{H}$ is not pinned down.
  - Need at least four correspondences, no three of them collinear.

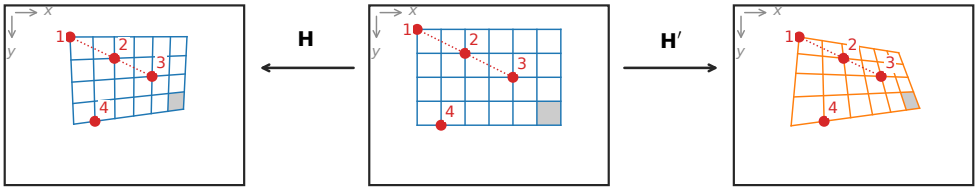


image 2, by **H** | image 1 | image 2, by **H**′

When does a homography apply?

A single $\mathbf{H}$ relates two views in exactly two situations.

- **1. The scene is planar.** Two views of a plane are always related by a homography, however the camera moved: a wall, a document, the road, a building facade.
- **2. The camera rotates about its optical centre**. The mapping is then independent of depth, so it holds for any scene. The camera may even zoom between shots.

In every other case when a camera translates while viewing a non-planar scene, the homography cannot relate the two images.

- Why? Due to **parallax** where near points shift more than far points.
- Caveat: **parallax** shrinks as the scene gets far away compared with how far the camera moved, so then a homography is a good approximation.

Advice for budding photographers: rotate in place when you shoot a panorama.

Appendix A derives both cases and shows parallax in a real mosaic.

Estimating $\mathbf{H}$: setting it up

Given: $n \geq 4$ correspondences $\mathbf{p}_i = (x_i, y_i) \leftrightarrow \mathbf{p}'_i = (x'_i, y'_i)$, $i = 1, \ldots, n$.

Unknown: the nine entries of $\mathbf{H}$.

Method: the **direct linear transform (DLT)**. Rewrite each correspondence as equations linear in the entries of $\mathbf{H}$, stack them, and solve with the SVD.

Each correspondence must satisfy $\tilde{\mathbf{p}}'_i \sim \mathbf{H}\tilde{\mathbf{p}}_i$. Drop the subscript i and write out the product:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11}x + h_{12}y + h_{13} \\ h_{21}x + h_{22}y + h_{23} \\ h_{31}x + h_{32}y + h_{33} \end{pmatrix}$$

$(u, v, w)^\top$ is $\tilde{\mathbf{p}}'$ up to scale, so the pixel is $\mathbf{p}' = \pi(u, v, w) = (u/w,\ v/w)$.

Estimating $\mathbf{H}$: x' and y'

Divide by the third coordinate:

$$x' = \frac{u}{w} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \qquad y' = \frac{v}{w} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

These are not linear in the h_{jk}: the unknowns appear in the denominator. Multiply out the denominators. This makes the equations **linear in the entries of $\mathbf{H}$**:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13}$$
$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}$$

Two equations per correspondence, in the 9 unknowns $h_{11}, \ldots, h_{33}$.

The two rows of $\mathbf{A}$

Move everything to one side. With $\mathbf{h} = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^\top$, correspondence i gives

$$\begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{pmatrix} \mathbf{h} = \mathbf{0}$$

- Keep the order of $\mathbf{h}$ fixed, so reshaping back to 3×3 puts each entry in its place.

Stacking all n correspondences

Each correspondence adds two rows. Stack them:

$$\underbrace{\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 & -y'_2 \\ \vdots & & & & \vdots & & & & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{pmatrix}}_{\mathbf{A},\ 2n\times 9} \underbrace{\begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{pmatrix}}_{\mathbf{h},\ 9\times 1} = \underbrace{\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{2n\times 1}$$

With $n = 4$, $\mathbf{A}$ is 8×9: 8 equations for the 8 degrees of freedom.

Solving $\mathbf{Ah} = \mathbf{0}$

$\mathbf{h} = \mathbf{0}$ satisfies $\mathbf{Ah} = \mathbf{0}$ but is useless. Impose $\|\mathbf{h}\| = 1$, which is legitimate precisely because $\mathbf{H}$ is only defined up to scale.

Problem setup: Minimise $\|\mathbf{Ah}\|$ subject to $\|\mathbf{h}\| = 1$.

Solution: $\mathbf{h}$ is the right singular vector of $\mathbf{A}$ for the SMALLEST singular value: the last column of $\mathbf{V}$ in $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$.

Same structure as total least squares when we were fitting a line $ax + by + c = 0$ to the data.

More than four points

With $n > 4$ noisy points the system is overdetermined, and the smallest singular value is no longer zero.

- The SVD solution is then the **least-squares** fit of the **algebraic error** $\|\mathbf{Ah}\|$ (note: this does not represent distance in pixels.)
- The smallest singular value is a useful quality measure.
- The estimate becomes more stable.

In NumPy:

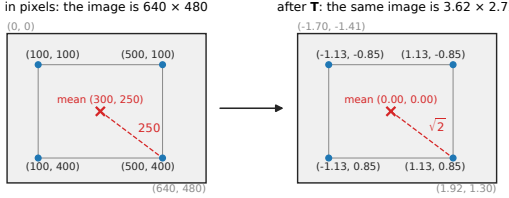
```
U, S, Vt = np.linalg.svd(A)     # A is 2n x 9
h = Vt[-1]                      # last ROW of V^T = last column of V
H = h.reshape(3, 3)
```

With $n = 4$, `S` holds only 8 values; the ninth is zero and not listed. `Vt` is still 9×9, so `Vt[-1]` is the answer.

Practical considerations: Hartley's fix

Normalise the correspondences' coordinates to have 0 mean and mean distance $\sqrt{2}$ from the origin before estimating $\mathbf{H}$

- Four correspondences in a 640×480 image. Compute $\mathbf{T}$ to transform the correspondence coordinates to have zero mean and mean distance $\sqrt{2}$ from the origin.
  - Important: $\mathbf{T}$ is computed over correspondence coordinates and not on image size.
- Applying $\mathbf{T}$ changes the positions only. No pixels are moved.



- Homography: the error drops by a factor of 1.2 to 1.6 when the matches are crowded, but hardly changes when they are spread across the image.
- Fundamental matrix (later): the error drops by a factor of 6 to 8.

Hartley's fix: the recipe for $\mathbf{T}$

$\mathbf{p}_i = (x_i, y_i)$ are the n correspondences in image 1 that are used to estimate the homography $\mathbf{H}$.

Compute mean and scale

Step	In general	In this example
1. Mean	$\mathbf{m} = \frac{1}{n}\sum_i \mathbf{p}_i$	$(300,\ 250)$
2. Mean distance	$r = \frac{1}{n}\sum_i \lVert\mathbf{p}_i - \mathbf{m}\rVert$	$\sqrt{200^2 + 150^2} = 250$ for every point
3. Scale	$s = \sqrt{2}/r$	$\sqrt{2}/250 = 0.00566$
4. Shift	$-s\,m_x,\ -s\,m_y$	$-1.697,\ -1.414$

Assemble $\mathbf{T}$

$$\mathbf{T} = \begin{pmatrix} s & 0 & -s\,m_x \\ 0 & s & -s\,m_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.00566 & 0 & -1.697 \\ 0 & 0.00566 & -1.414 \\ 0 & 0 & 1 \end{pmatrix}$$

Hartley's fix: $\mathbf{T}$ and $\mathbf{T}'$

Each image gets its own matrix, from its half of the correspondences:

- Image 1: $(100, 100)$, $(500, 100)$, $(500, 400)$, $(100, 400)$
- Image 2: $(120, 90)$, $(560, 130)$, $(520, 430)$, $(80, 430)$, in matching order

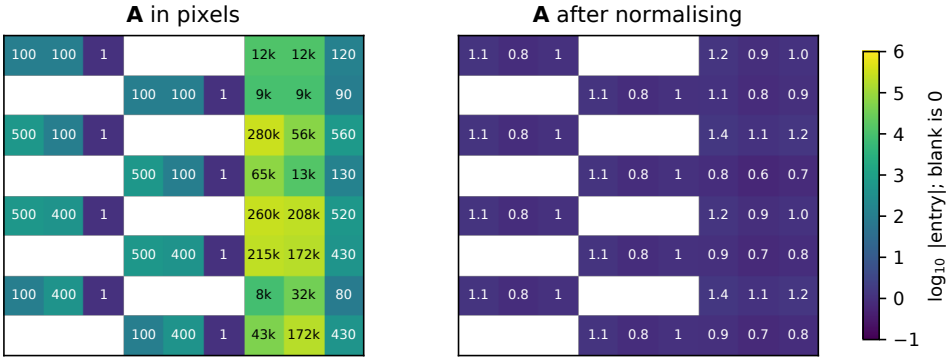
	Image 1	Image 2
Mean	$(300,\ 250)$	$(320,\ 270)$
Mean distance	250	$(269.1 + 277.8 + 256.1 + 288.4)/4 = 272.9$
Scale	$s = \sqrt{2}/250 = 0.00566$	$s' = \sqrt{2}/272.9 = 0.00518$

$$\mathbf{T} = \begin{pmatrix} 0.00566 & 0 & -1.697 \\ 0 & 0.00566 & -1.414 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}' = \begin{pmatrix} 0.00518 & 0 & -1.658 \\ 0 & 0.00518 & -1.399 \\ 0 & 0 & 1 \end{pmatrix}$$

Different points give different matrices. With RANSAC, the final refit computes both from the inliers alone, because those are the points going into the DLT.

Hartley's fix: solving for $\mathbf{H}_n$

Run the usual DLT on the normalised points $\mathbf{T}\tilde{\mathbf{p}}_i$ and $\mathbf{T}'\tilde{\mathbf{p}}'_i$: build $\mathbf{A}$, take the last right singular vector, reshape it into $\mathbf{H}_n$. Each cell is the size of one entry of $\mathbf{A}$:



Hartley's fix: from $\mathbf{H}_n$ back to $\mathbf{H}$

$\mathbf{H}_n$ maps normalised points to normalised points:

$$\mathbf{T}'\tilde{\mathbf{p}}' \sim \mathbf{H}_n\,\mathbf{T}\tilde{\mathbf{p}}$$

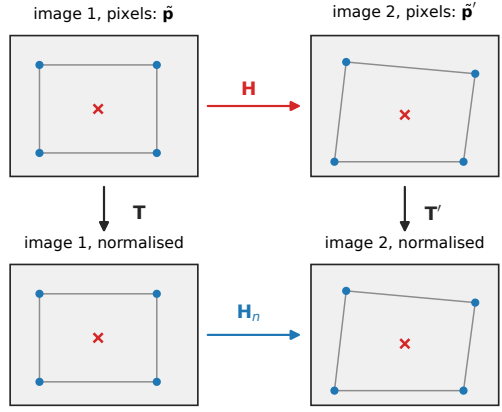
Multiply both sides by $\mathbf{T}'^{-1}$:

$$\tilde{\mathbf{p}}' \sim \mathbf{T}'^{-1}\mathbf{H}_n\mathbf{T}\,\tilde{\mathbf{p}}$$

This holds for every point, so

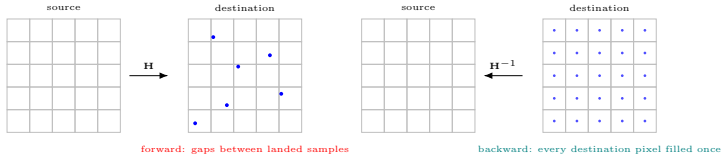
$$\boxed{\mathbf{H} = \mathbf{T}'^{-1}\,\mathbf{H}_n\,\mathbf{T}}$$

Read it right to left: normalise in image 1, apply $\mathbf{H}_n$, then undo image 2's normalisation.



Warping: Given $\mathbf{H}$ stitch two images

Given two images $\mathbf{I}$ and $\mathbf{I}'$ and $\mathbf{H}$ that maps coordinates from $\mathbf{I}$ (source) to $\mathbf{I}'$ (destination).



Forward warping: maps each source (x, y) to destination (x', y'), leaving holes in the destination: some destination (x', y') receive no source (x, y), while others receive several.

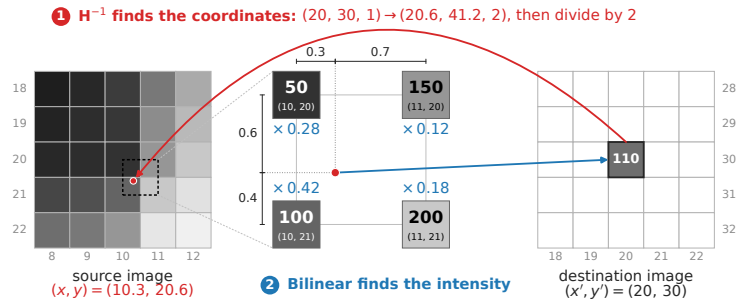
Backward warping: for each destination (x', y') use $\mathbf{H}^{-1}$ to find the corresponding source (x, y). This guarantees that each destination (x', y') whose source falls inside the image is filled.

Always warp backwards.

Backward warping

When a destination (x', y') falls on a non-integer source (x, y), i.e., not on a source pixel, interpolate intensities in the source to copy back to destination (x', y').

Pixel $(20, 30)$ in the destination maps to $(10.3, 20.6)$ in the source. Use source pixels at $(10, 20)$, $(11, 20)$, $(10, 21)$ and $(11, 21)$ to compute the intensity for the destination pixel.



Stitching two images (Part 1/2)

Goal: Set **image 1 as the destination** and warp image 2 into its frame.

Recipe

Estimate $\mathbf{H}$ with image 2 points as $\tilde{\mathbf{p}}$ and image 1 points as $\tilde{\mathbf{p}}'$: then $\mathbf{H}$ carries image 2 into image 1.

Image 2 may land partly left of or above image 1, at negative coordinates. So:

1. Map image 2's four corners with $\mathbf{H}$, and divide by the third coordinate.
2. Take the bounding box of those four points and image 1's own four corners to compute $x_{\min}, x_{\max}, y_{\min}, y_{\max}$.
3. Make a canvas $(x_{\max} - x_{\min})$ wide and $(y_{\max} - y_{\min})$ tall, and shift everything into it with $\mathbf{T} = \begin{pmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{pmatrix}$, so all coordinates are positive.
4. Copy image 1 in at offset $(-x_{\min}, -y_{\min})$.

Stitching two images (Part 2/2)

For each canvas pixel $\tilde{\mathbf{q}}$, warp backwards:

$$\mathbf{p} = \pi\Big(\mathbf{H}^{-1}\,\mathbf{T}^{-1}\,\tilde{\mathbf{q}}\Big)\,,$$

then sample image 2 at $\mathbf{p}$ by bilinear interpolation. Skip the pixel if $\mathbf{p}$ falls outside image 2.

Where both images cover a canvas pixel, **blend** them:

- **Average:** $(I_1 + I_2)/2$. Simple, but the seam can show where the two exposures differ.
- **Feather:** weight each image by its pixel's distance α_1, α_2 to that image's own border, $(\alpha_1 I_1 + \alpha_2 I_2)/(\alpha_1 + \alpha_2)$. Each image fades out towards its edge, which hides the seam.

The full pipeline

1. **Detect** interest points in both images.
2. **Describe** and **match**, with the ratio test.
3. **RANSAC** to reject wrong matches, $s = 4$. A match is an inlier if $\|\mathbf{p}' - \pi(\mathbf{H}\tilde{\mathbf{p}})\| < d$ pixels.
4. **Refit** $\mathbf{H}$ on all inliers by SVD.
5. **Warp backwards** onto the canvas and blend.

Step 3 is not optional. Least squares over all putative matches, outliers included, produces a wildly wrong $\mathbf{H}$, typically warping the second image into an extreme trapezoid.

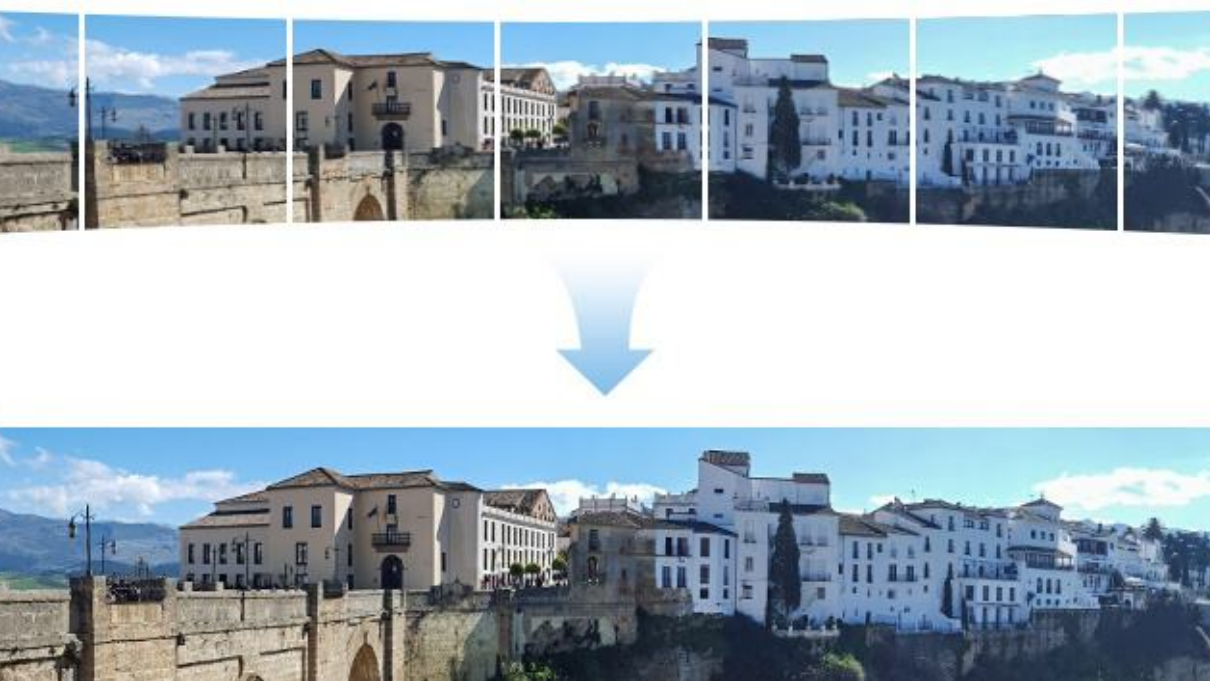


Image credit (previous slide)

Courtesy: //www.baeldung.com/cs/image-stitching

Lecture 1 recap

- Homography: 3×3, 8 DOF, 4 correspondences, preserves straight lines.
- Applies when the scene is planar **or** the camera only rotates.
- Parallax (near points shift more than far ones) is what breaks it.
- DLT: build $\mathbf{A}$, solve $\mathbf{A}\mathbf{h} = \mathbf{0}$ by SVD, **smallest** singular vector.
- Normalise the coordinates first.
- Warp backwards, interpolate, size the canvas from the warped corners.

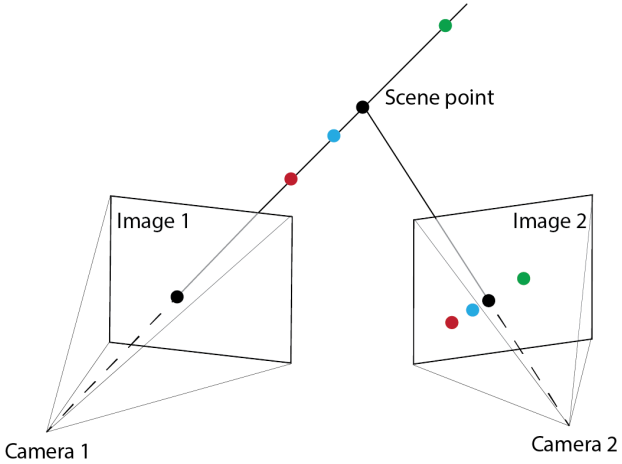
Lecture 2: Epipolar geometry

Notation

- Bold is a vector, italic is a number, e.g., $\mathbf{x}_0 = (x_0, y_0, 1)^\top$.
- Two cameras: camera 0 and camera 1

	This lecture	vs. Homography
Cameras and images	**0 and 1**	1 and 2
Camera centres	$\mathbf{c}_0,\ \mathbf{c}_1$	$\mathbf{C},\ \mathbf{C}'$
3D point; in camera 0's and camera 1's coordinates	$\mathbf{p}$; $\mathbf{p}_0,\ \mathbf{p}_1$	$\mathbf{X},\ \mathbf{X}'$
Its depth in each camera	$d_0,\ d_1$	$Z,\ Z'$
Pixel, always written $(x, y, 1)^\top$	$\mathbf{x}_0,\ \mathbf{x}_1$	$\tilde{\mathbf{p}},\ \tilde{\mathbf{p}}'$
Ray through the pixel, $\mathbf{K}_j^{-1}\mathbf{x}_j$	$\hat{\mathbf{x}}_0,\ \hat{\mathbf{x}}_1$	$\hat{\mathbf{p}},\ \hat{\mathbf{p}}'$
Intrinsics	$\mathbf{K}_0,\ \mathbf{K}_1$	$\mathbf{K},\ \mathbf{K}'$
Relative pose	$\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t}$	$\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$

Two views



Two views

Homography fails if the camera translates and the scene has depth: a pixel in image 0 lands on different pixels in image 1, depending upon its depth.

Observation: for any pixel in image 0, its match in image 1 lies on a line.

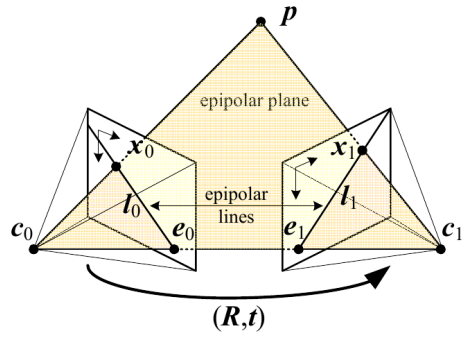


Searching for a match on a line vs. in the entire image makes dense matching of pixels between two images affordable.

Photographs: R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004.

The geometry: two cameras and one 3D point $\mathbf{p}$

- **Baseline:** the line joining $\mathbf{c}_0$ and $\mathbf{c}_1$.
- **Epipolar plane:** the plane through $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{p}$.
- **Epipolar lines** l_0, l_1: where the epipolar plane cuts each image plane.
- **Epipoles** $\mathbf{e}_0, \mathbf{e}_1$: where the baseline pierces each image plane. So $\mathbf{e}_1$ is the image of $\mathbf{c}_0$ in camera 1, and $\mathbf{e}_0$ the image of $\mathbf{c}_1$ in camera 0.



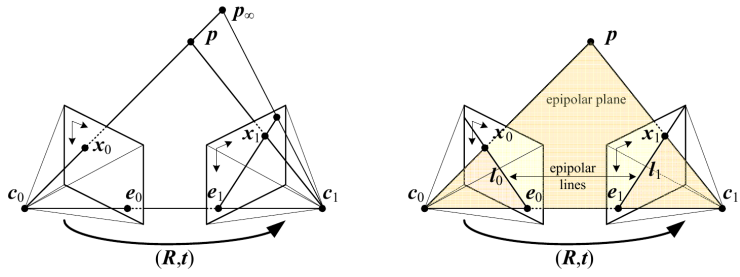
Every epipolar plane contains the baseline, so **every epipolar line passes through the epipole**.

An epipole can fall outside the image frame: the image plane goes on beyond the part the sensor records.

Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 11.11.

The epipolar constraint

- Every candidate for $\mathbf{p}$ lies on the ray from $\mathbf{c}_0$ through $\mathbf{x}_0$, out to $\mathbf{p}_\infty$, infinitely far away.
- Camera 1 sees that ray as a segment of the epipolar line, starting at the epipole $\mathbf{e}_1$.
- The epipolar plane through $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{p}$ cuts each image in its epipolar line, $\boldsymbol{l}_0$ and $\boldsymbol{l}_1$.



Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 12.3.

The epipolar constraint: an example (Part 1)

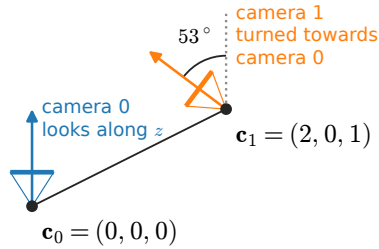
Both cameras: images 640×480, focal length 500 px, principal point $(320, 240)$:

$$\mathbf{K} = \begin{pmatrix} 500 & 0 & 320 \\ 0 & 500 & 240 \\ 0 & 0 & 1 \end{pmatrix}$$

Camera 0 sits at the origin, looking along z.
Camera 1 sits at $\mathbf{c}_1 = (2, 0, 1)^\top$, turned $53°$ towards camera 0.
Relative pose: $\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t}$ where

$$\mathbf{R} = \begin{pmatrix} 0.6 & 0 & 0.8 \\ 0 & 1 & 0 \\ -0.8 & 0 & 0.6 \end{pmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} -2 \\ 0 \\ 1 \end{pmatrix}$$

Aside: $\mathbf{t} = -\mathbf{R}\mathbf{c}_1$, because camera 1's own centre must land at $\mathbf{p}_1 = \mathbf{0}$.



Seen from above. The image planes are drawn close to the centres, to fit.

The epipolar constraint: an example (Part 2)

Take one scene point, $(0, 1, 5)^\top$ in camera 0's coordinates.

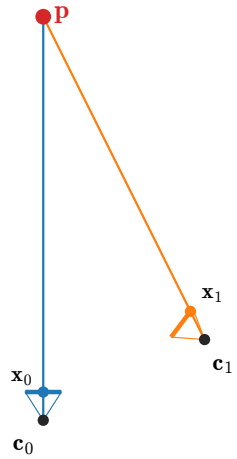
In camera 0:

- Point: $\mathbf{p}_0 = (0, 1, 5)^\top$, so its depth is $d_0 = 5$.
- Pixel: $\mathbf{K}\mathbf{p}_0 = (1600,\ 1700,\ 5)^\top$. Divide by the depth: $\mathbf{x}_0 = (320,\ 340,\ 1)^\top$.

In camera 1:

- Point: $\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t} = (2, 1, 4)^\top$, so its depth is $d_1 = 4$.
- Pixel: $\mathbf{K}\mathbf{p}_1 = (2280,\ 1460,\ 4)^\top$. Divide by the depth: $\mathbf{x}_1 = (570,\ 365,\ 1)^\top$.

Pixel $(320, 340)$ in camera 0 $\leftrightarrow$ pixel $(570, 365)$ in camera 1.



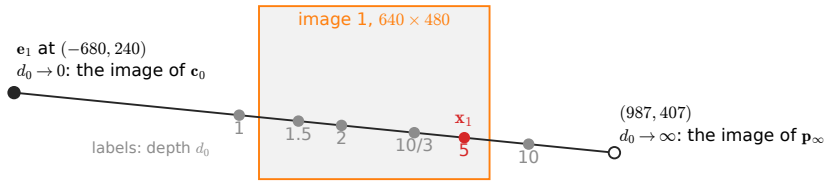
The epipolar constraint: an example (Part 3)

In camera 0, we only see $\mathbf{x}_0$. Its ray is $\hat{\mathbf{x}}_0 = \mathbf{K}^{-1}\mathbf{x}_0 = (0,\ 0.2,\ 1)^\top$. A point on this ray at depth d_0 is $d_0\hat{\mathbf{x}}_0$.

A point at depth d_0 in camera 0 maps to $\mathbf{p}_1 = d_0\mathbf{R}\hat{\mathbf{x}}_0 + \mathbf{t}$ in camera 1. Here, $\mathbf{R}\hat{\mathbf{x}}_0 = (0.8,\ 0.2,\ 0.6)^\top$.

Project the points at different d_0 into camera 1 using $\mathbf{K}$.

d_0	$\to 0$	1	1.5	2	10/3	5	10	$\to \infty$
x_1	-680	-55	109	229	431	570	749	987
y_1	240	303	319	331	351	365	383	407



Points at depths d_0 in camera 0 all project onto one line, $y = 0.1x + 308$ in camera 1. Image 0 alone cannot say which depth is right; image 1 finds it, by searching along the line.

The essential matrix

Given two **calibrated** cameras with intrinsic matrices $\mathbf{K}_0$ and $\mathbf{K}_1$, respectively.

A pixel $\mathbf{x}_j$ in camera j gives the ray $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$.

A 3D point (seen in both cameras) at depths d_0 and d_1 lies on both rays:

$$\mathbf{p}_0 = d_0\hat{\mathbf{x}}_0, \qquad \mathbf{p}_1 = d_1\hat{\mathbf{x}}_1$$

Substitute both into $\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t}$:

$$d_1\hat{\mathbf{x}}_1 = d_0\mathbf{R}\hat{\mathbf{x}}_0 + \mathbf{t}$$

In the example from previous slides:

$$\underbrace{4\,(0.5,\ 0.25,\ 1)^\top}_{(2,\ 1,\ 4)^\top} = \underbrace{5\,(0.8,\ 0.2,\ 0.6)^\top}_{(4,\ 1,\ 3)^\top} + (-2,\ 0,\ 1)^\top \ \checkmark$$

Szeliski Sec. 11.3.1, Eq. (11.27).

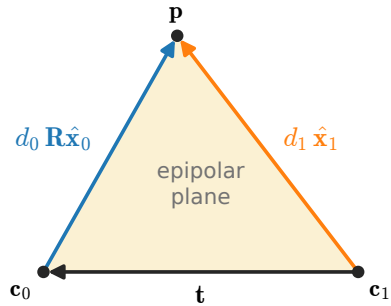
The essential matrix

Two ways to get to $\mathbf{p}$ from $\mathbf{c}_1$, all in camera 1's coordinates:

Directly: along camera 1's ray $d_1\hat{\mathbf{x}}_1$.

Via camera 0: first $\mathbf{t}$, then $d_0\mathbf{R}\hat{\mathbf{x}}_0$.

- $\mathbf{t}$ is where camera 0 sits, seen from camera 1: put $\mathbf{p}_0 = \mathbf{0}$ and you get $\mathbf{p}_1 = \mathbf{t}$.
- $d_0\mathbf{R}\hat{\mathbf{x}}_0$ is camera 0's ray, turned into camera 1's axes.



The sides of a triangle lie in one plane: the epipolar plane.

The caption of Szeliski's Fig. 11.11 writes $\mathbf{t} = \mathbf{c}_1 - \mathbf{c}_0$, but his Eqs. (11.27) and (11.32) give the direction used here.

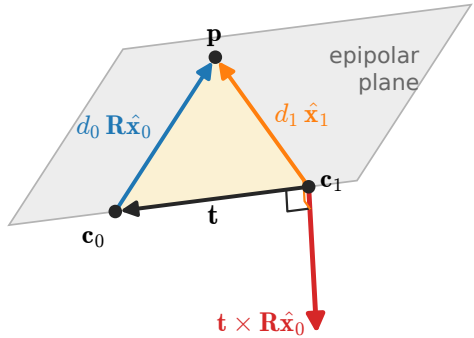
The essential matrix

$\mathbf{t}$, $d_0\mathbf{R}\hat{\mathbf{x}}_0$, and $d_1\hat{\mathbf{x}}_1$ lie in the epipolar plane.

d_0 and d_1 are scalars and just stretch/shrink the vectors.

$\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0$ (the red arrow) is perpendicular to the epipolar plane.

Therefore, $\hat{\mathbf{x}}_1^\top (\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0) = 0$ since $\hat{\mathbf{x}}_1$ lies on the epipolar plane.



In our example from previous slides

$\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0 = (-0.2,\ 2,\ -0.4)^\top$ and $\hat{\mathbf{x}}_1 = (0.5,\ 0.25,\ 1)^\top$:

$$0.5(-0.2) + 0.25(2) + 1(-0.4) = 0 \checkmark$$

Szeliski Sec. 11.3.1, Eq. (11.32).

The essential matrix $\mathbf{E}$

Write the cross product as a matrix-vector product, $\mathbf{t} \times \mathbf{v} = [\mathbf{t}]_\times \mathbf{v}$, with

$$[\mathbf{t}]_\times = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}$$

$$\begin{aligned} \hat{\mathbf{x}}_1^\top (\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0) &= 0 \\ \hat{\mathbf{x}}_1^\top [\mathbf{t}]_\times \mathbf{R}\, \hat{\mathbf{x}}_0 &= 0 \\ \hat{\mathbf{x}}_1^\top \mathbf{E}\, \hat{\mathbf{x}}_0 &= 0 \end{aligned}$$

Here $\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$ is the essential matrix.

Szeliski Sec. 11.3.1, Eqs. (11.30)–(11.32); $[\,\cdot\,]_\times$ is Eq. (2.32). Eqs. (11.28), (11.29) reach the same result by algebra: cross $d_1 \hat{\mathbf{x}}_1 = d_0 \mathbf{R}\hat{\mathbf{x}}_0 + \mathbf{t}$ with $\mathbf{t}$, then dot with $\hat{\mathbf{x}}_1$.

Essential matrix

- $\hat{\mathbf{x}}_1^\top \mathbf{E}\, \hat{\mathbf{x}}_0 = 0$ holds wherever the point lies on its rays.
- Needs translation. If the camera only rotates, we have a homography.
- **5 DOF:** 3 for the rotation, 2 for the direction of $\mathbf{t}$.
- $\mathbf{E}$ has rank 2 because $\mathbf{E}\mathbf{v}$ is always perpendicular to $\mathbf{t}$ so it has only two independent directions.

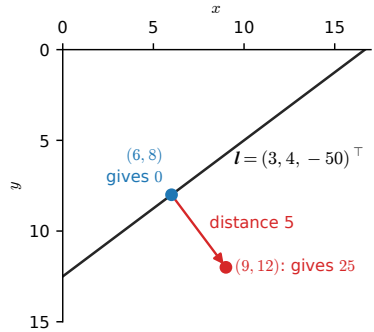
A line is three numbers

Consider a line $ax + by + c = 0$. We represent it with the vector $\boldsymbol{l} = (a, b, c)^\top$.

$\boldsymbol{l}$ and $\lambda\boldsymbol{l}$ are the same line. (a, b) is perpendicular to the line.

A point (x, y) lies on this line when $(x, y, 1)\,\boldsymbol{l} = 0$.

The distance between a point (x, y) and the line $\boldsymbol{l}$ is $|ax + by + c|/\sqrt{a^2 + b^2}$.



Therefore $\hat{\mathbf{x}}_1^\top\,(\mathbf{E}\hat{\mathbf{x}}_0) = 0$ says that $\hat{\mathbf{x}}_1$ lies on the line $\boldsymbol{l}_1 = \mathbf{E}\hat{\mathbf{x}}_0$.

The product is one number, so it equals its own transpose, $\hat{\mathbf{x}}_0^\top(\mathbf{E}^\top\hat{\mathbf{x}}_1) = 0$: $\hat{\mathbf{x}}_0$ lies on $\boldsymbol{l}_0 = \mathbf{E}^\top\hat{\mathbf{x}}_1$.

Szeliski Sec. 2.1, Eq. (2.3); Sec. 11.3.1.

The fundamental matrix

Plug $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$ into $\hat{\mathbf{x}}_1^\top \mathbf{E}\hat{\mathbf{x}}_0 = 0$.

$$
\begin{aligned}
\hat{\mathbf{x}}_1^\top \mathbf{E}\hat{\mathbf{x}}_0 &= 0 \\
\left(\mathbf{K}_1^{-1}\mathbf{x}_1\right)^\top \mathbf{E} \left(\mathbf{K}_0^{-1}\mathbf{x}_0\right) &= 0 \\
\left(\mathbf{x}_1^\top \mathbf{K}_1^{-\top}\right) \mathbf{E} \left(\mathbf{K}_0^{-1}\mathbf{x}_0\right) &= 0 \\
\mathbf{x}_1^\top \underbrace{\mathbf{K}_1^{-\top}\mathbf{E}\mathbf{K}_0^{-1}}_{\mathbf{F}} \mathbf{x}_0 &= 0 \\
\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 &= 0
\end{aligned}
$$

$\mathbf{F}$ is called the **fundamental matrix**. It has 7 degrees of freedom. It has rank 2. It is used for uncalibrated cameras, since we do not need intrinsic matrices $\mathbf{K}_0$ and $\mathbf{K}_1$. We can estimate $\mathbf{F}$ directly from pixel matches.

$\mathbf{E}$ or $\mathbf{F}$?

	Essential matrix $\mathbf{E}$	Fundamental matrix $\mathbf{F}$
Works on	rays $\hat{\mathbf{x}}_j$	pixels $\mathbf{x}_j$
Constraint	$\hat{\mathbf{x}}_1^\top \mathbf{E} \hat{\mathbf{x}}_0 = 0$	$\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 = 0$
Line in image 1	$\boldsymbol{l}_1 = \mathbf{E}\hat{\mathbf{x}}_0$, in ray coordinates	$\boldsymbol{l}_1 = \mathbf{F}\mathbf{x}_0$, in pixels
What is inside	$[\mathbf{t}]_\times \mathbf{R}$	$\mathbf{K}_1^{-\top} \mathbf{E} \mathbf{K}_0^{-1}$
You must know	$\mathbf{K}_0, \mathbf{K}_1$	nothing
You estimate it from	matches, turned into rays	matches, in pixels
Degrees of freedom	5	7
Singular values	$(\sigma, \sigma, 0)$	$(\sigma_0, \sigma_1, 0)$
It tells you	$\mathbf{R}$ and the direction of $\mathbf{t}$	the epipolar lines; not the pose

Estimate $\mathbf{E}$ and $\mathbf{F}$ from matches. Factor $\mathbf{E}$ to get $\mathbf{R}$ and $\mathbf{t}$.

Factoring $\mathbf{E}$ into $\mathbf{R}$ and $\mathbf{t}$: Szeliski Sec. 11.3.1, Eqs. (11.40)–(11.45), beyond this course.

Consequences of $\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 = 0$

- $\mathbf{x}_1$ lies on $\boldsymbol{l}_1 = \mathbf{F}\mathbf{x}_0$
- $\mathbf{x}_0$ lies on $\boldsymbol{l}_0 = \mathbf{F}^\top \mathbf{x}_1$. (Transpose both sides of the above equation.)
- Every line in image 1 passes through $\mathbf{e}_1$, so $\mathbf{e}_1^\top \mathbf{F}\mathbf{x}_0 = 0$ for **every** $\mathbf{x}_0$. A row vector that gives 0 against every pixel must be all zeros:

$$\mathbf{e}_1^\top \mathbf{F} = \mathbf{0}^\top, \qquad \text{and in the same way, in image 0,} \qquad \mathbf{F}\mathbf{e}_0 = \mathbf{0}.$$

- All the epipolar lines can meet at one point since $\mathbf{F}$ flattens the direction $\mathbf{e}_0$ to zero.
- $\mathbf{F}\mathbf{e}_0 = \mathbf{0}$ is no line at all. The ray through $\mathbf{e}_0$ passes through $\mathbf{c}_1$, so camera 1 sees it as a single point.

Szeliski Sec. 11.3.3.

Finding the epipoles

From $\mathbf{F}$.

$\mathbf{F}\mathbf{e}_0 = \mathbf{0}$ has the same form as $\mathbf{A}\mathbf{h} = \mathbf{0}$. With $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$:

- $\mathbf{e}_0$ is the last right singular vector: the last column of $\mathbf{V}$.
- $\mathbf{e}_1$ is the same thing for $\mathbf{F}^\top$, whose SVD swaps $\mathbf{U}$ and $\mathbf{V}$: the last column of $\mathbf{U}$.

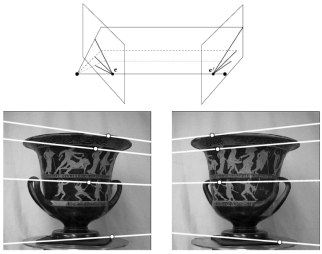
From the pose.

An epipole is the image of the other camera's centre:

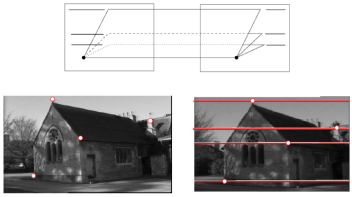
- Camera 0's centre is $\mathbf{t}$ in camera 1's coordinates, so $\mathbf{e}_1 \sim \mathbf{K}_1\mathbf{t}$.
- Camera 1's centre is $\mathbf{c}_1$ in camera 0's coordinates, so $\mathbf{e}_0 \sim \mathbf{K}_0\mathbf{c}_1$.

Szeliski Sec. 11.3.3, Eq. (11.50).

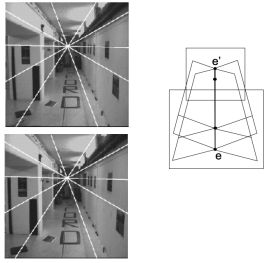
Where the epipole lands



Cameras turned inward. Lines slant and meet at the epipole, off to one side.



Sideways, no turn.
$\mathbf{t} = (t_x, 0, 0)^\top$, so $\mathbf{e}_j \sim (1, 0, 0)^\top$. We get x and y by dividing by the third number; with 0 there, no pixel exists. This **point at infinity** is a direction. The lines are parallel.



Forward, no turn.
$\mathbf{t} = (0, 0, t_z)^\top$, so $\mathbf{e}_j \sim (c_x, c_y, 1)^\top$, inside the image and the same in both. The lines radiate from it: the *focus of expansion*.

R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004, Ch. 9, via F. Z. Qureshi, *Computer Vision Notes*, notebook 21; their $\mathbf{e}, \mathbf{e}'$ are our $\mathbf{e}_0, \mathbf{e}_1$. Points at infinity: Szeliski Sec. 2.1. Focus of expansion: Fig. 11.12.

Estimating $\mathbf{F}$

Given: $n \geq 8$ matches $(x_{i0}, y_{i0}) \leftrightarrow (x_{i1}, y_{i1})$, $i = 1, \ldots, n$. **Unknown:** the nine entries of $\mathbf{F}$.

Each match must satisfy $\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 = 0$. Drop the subscript i and write it out:

$$\begin{pmatrix} x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix} = 0$$

Multiply out and we get

$$x_1(f_{00}x_0 + f_{01}y_0 + f_{02}) + y_1(f_{10}x_0 + f_{11}y_0 + f_{12}) + (f_{20}x_0 + f_{21}y_0 + f_{22}) = 0$$

which is linear in the entries of $\mathbf{F}$. We get one such equation per match.

Estimating $\mathbf{F}$

Re-arrange equations for n matches as

$$\underbrace{\begin{pmatrix} x_{10}x_{11} & y_{10}x_{11} & x_{11} & x_{10}y_{11} & y_{10}y_{11} & y_{11} & x_{10} & y_{10} & 1 \\ x_{20}x_{21} & y_{20}x_{21} & x_{21} & x_{20}y_{21} & y_{20}y_{21} & y_{21} & x_{20} & y_{20} & 1 \\ \vdots & & & & \vdots & & & & \vdots \\ x_{n0}x_{n1} & y_{n0}x_{n1} & x_{n1} & x_{n0}y_{n1} & y_{n0}y_{n1} & y_{n1} & x_{n0} & y_{n0} & 1 \end{pmatrix}}_{\mathbf{A},\ n\times 9} \underbrace{\begin{pmatrix} f_{00} \\ f_{01} \\ \vdots \\ f_{22} \end{pmatrix}}_{\mathbf{f},\ 9\times 1} = \underbrace{\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{n\times 1}$$

Problem setup: Minimise $\|\mathbf{A}\mathbf{f}\|$ subject to $\|\mathbf{f}\| = 1$. Recall that $\mathbf{F}$ is only known up to some scale.

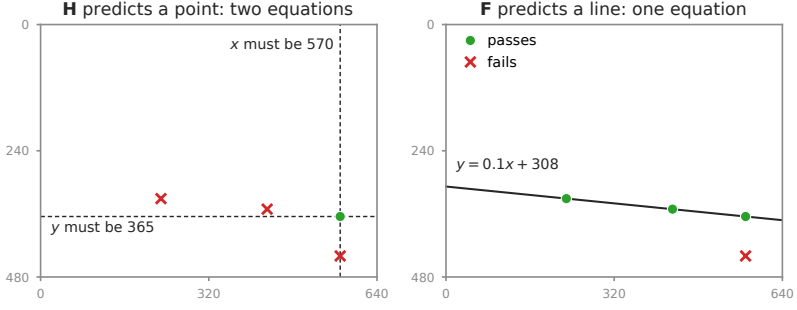
Solution: $\mathbf{f}$ is the right singular vector of $\mathbf{A}$ for the SMALLEST singular value: the last column of $\mathbf{V}$ in $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$.

Reshape $\mathbf{f}$ to 3×3 to get $\mathbf{F}$.

How many equations per match: homography vs. fundamental matrix

Each match gives one equation for estimating $\mathbf{F}$, but gives two equations for estimating $\mathbf{H}$.

- $\mathbf{H}$ predicts the exact pixel, so x and y must both agree: two equations.
- $\mathbf{F}$ predicts only a line, and the match can sit anywhere on it. "On the line" is one condition: one equation.



Nine unknowns, but only eight matches

Only the ratios matter. $\mathbf{F}$ and $2\mathbf{F}$ give the same constraint, because multiplying $\mathbf{x}_1^\top \mathbf{F}\mathbf{x}_0 = 0$ by 2 changes nothing. So the nine entries hold eight numbers to find.

Count the rows. Each independent row of $\mathbf{A}$ removes one direction from the solutions of $\mathbf{A}\mathbf{f} = \mathbf{0}$:

$$9 \text{ unknowns} - 8 \text{ rows} = 1 \text{ direction left}$$

That direction is the line of solutions $\lambda\mathbf{f}$, and all of them are the same $\mathbf{F}$.

	Entries	Up to scale	Equations per match	Matches needed
$\mathbf{H}$	9	8	2	4
$\mathbf{F}$, eight-point	9	8	1	8

With more than eight noisy matches, no exact solution exists. The SVD then finds the best fit. Eight is this algorithm's minimum, not the goal: use every match you have.

The normalised eight-point algorithm

Given $n \geq 8$ matches $\mathbf{x}_{i0} \leftrightarrow \mathbf{x}_{i1}$. A tilde means **Hartley-normalised**, as in Szeliski.

1. **Normalise.** Build $\mathbf{T}_0$ from image 0's points and $\mathbf{T}_1$ from image 1's, by Lecture 1's recipe (its $\mathbf{T}$ and $\mathbf{T}'$). Let $\tilde{\mathbf{x}}_{ij} = \mathbf{T}_j \mathbf{x}_{ij}$.
2. **Build** $\mathbf{A}$, one row per match, from the normalised points.
3. **Solve** by SVD: the last right singular vector of $\mathbf{A}$, reshaped row by row into $\tilde{\mathbf{F}}$.
4. **Enforce rank 2.** Take the SVD of the matrix itself, $\tilde{\mathbf{F}} = \mathbf{U}\,\mathrm{diag}(\sigma_0, \sigma_1, \sigma_2)\,\mathbf{V}^\top$, and rebuild $\tilde{\mathbf{F}} \leftarrow \mathbf{U}\,\mathrm{diag}(\sigma_0, \sigma_1, 0)\,\mathbf{V}^\top$. Rank counts the nonzero singular values, so this has rank 2; it is also the closest rank-2 matrix to $\tilde{\mathbf{F}}$ (Frobenius norm).
5. **Denormalise:** $\mathbf{F} = \mathbf{T}_1^\top \tilde{\mathbf{F}}\, \mathbf{T}_0$.

Why transposes in step 5, when homography estimation used an inverse? $\tilde{\mathbf{F}}$ sits *between* two points:

$$\tilde{\mathbf{x}}_1^\top \tilde{\mathbf{F}} \tilde{\mathbf{x}}_0 = (\mathbf{T}_1 \mathbf{x}_1)^\top \tilde{\mathbf{F}}\, (\mathbf{T}_0 \mathbf{x}_0) = \mathbf{x}_1^\top (\mathbf{T}_1^\top \tilde{\mathbf{F}}\, \mathbf{T}_0) \mathbf{x}_0 = 0.$$

$\mathbf{H}$ instead maps a point to a point, so it needed homography estimation's $\mathbf{T}'^{-1}$ to undo the second image's normalisation.

R. I. Hartley, "In defense of the eight-point algorithm," *IEEE TPAMI* 19(6), 1997. Szeliski Sec. 11.3.1, Eq. (11.37); he notes that scaling to unit variance works about as well as Hartley's mean distance of $\sqrt{2}$.

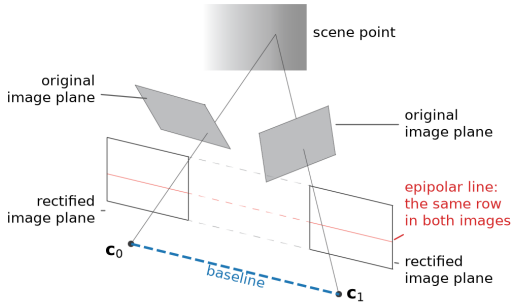
Rectification

Slanted epipolar lines are awkward to search along. **Rectification** warps both images so that:

- epipolar lines become **horizontal**, and
- each pair of lines lies on the **same image row**.

Then a match differs only in x (disparity).

2D search $\rightarrow$ 1D search $\rightarrow$ one number



The two rectified image planes lie in one plane, parallel to the baseline. The red epipolar line is the same row in both.

Szeliski Sec. 12.1.1. Diagram: F. Z. Qureshi, *Computer Vision Notes*, notebook 21, CC BY-NC 4.0; labels added for this course.

Rectification: the recipe

Given: the calibration $\mathbf{K}_0, \mathbf{K}_1$ and the relative pose $\mathbf{R}, \mathbf{t}$, with $\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t}$.

1. **New x-axis: the baseline.** In camera 0's coordinates, $\mathbf{c}_1 = -\mathbf{R}^\top \mathbf{t}$, so $\mathbf{r}_x = \mathbf{c}_1 / \|\mathbf{c}_1\|$. If its first entry is negative (camera 1 on the left), negate it, or the images come out upside down.
2. **New y- and z-axes.** Average the two optical axes to keep the turns small: $\mathbf{a} = \frac{1}{2}\left[(0,0,1)^\top + \mathbf{R}^\top (0,0,1)^\top\right]$. Then $\mathbf{r}_y = \mathbf{a} \times \mathbf{r}_x / \|\mathbf{a} \times \mathbf{r}_x\|$ and $\mathbf{r}_z = \mathbf{r}_x \times \mathbf{r}_y$.
3. **Rotations.** Stack the axes as rows: $\mathbf{R}_{r0} = \begin{pmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z \end{pmatrix}^\top$. Camera 1 first goes back to camera 0's axes: $\mathbf{R}_{r1} = \mathbf{R}_{r0}\mathbf{R}^\top$. (Neither is the relative pose $\mathbf{R}$.)
4. **One $\mathbf{K}$ for both.** For example, the average of $\mathbf{K}_0$ and $\mathbf{K}_1$.
5. **The two warps.** From pixels to rays, turn the rays, back to pixels: $\mathbf{H}_j = \mathbf{K}\,\mathbf{R}_{rj}\,\mathbf{K}_j^{-1}$, for $j = 0, 1$.
6. **Warp each image backwards**, as in Lecture 1: for each new pixel, apply $\mathbf{H}_j^{-1}$ and interpolate. If the result leaves the frame, shift both images by the same amount vertically.

Corresponding points now share a row. The next slides run this recipe on our example.

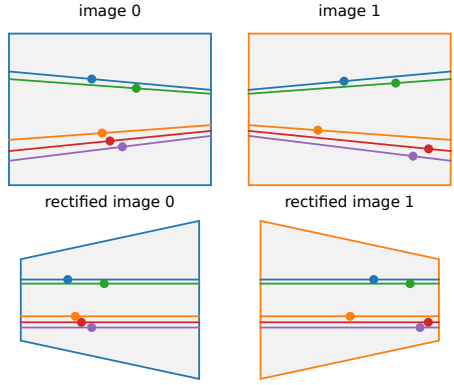
After Szeliski Sec. 12.1.1 and its footnote 2. Both cameras must take the same axes: looking perpendicular to the baseline leaves a free tilt about it, and a different tilt in each camera gives horizontal lines on different rows.

Rectification

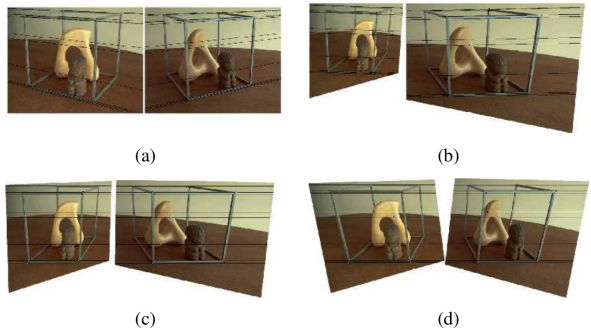
Top: the lines slant towards the epipoles.

Bottom: the lines are horizontal, and each pair shares a row.

The frames become trapezoids, because a homography does not keep a rectangle. This pair turns by a lot, so they are extreme.



Rectification on a real pair



(a) The original pair, with epipolar lines. **(b)** Warped so the lines are parallel. **(c)** Warped so the lines are horizontal and on matching rows. **(d)** The same, with less distortion. Loop and Zhang's method needs only the epipolar geometry, not the pose.

Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 12.4, from Loop and Zhang (1999), © 1999 IEEE.

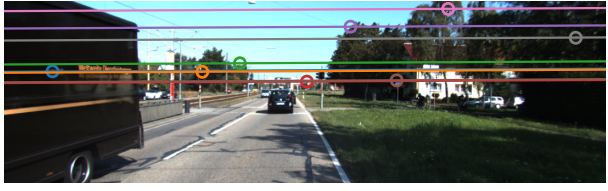
KITTI epipolar lines

Eight of the 593 inliers, one colour each. A **circle** marks the point in that image; the **line** of the same colour is the epipolar line of its partner.

image 0 (left): epipolar lines $\boldsymbol{l}_0 = \mathbf{F}^\top \mathbf{x}_1$



image 1 (right): epipolar lines $\boldsymbol{l}_1 = \mathbf{F}\mathbf{x}_0$



Lines from the calibrated $\mathbf{F}$. KITTI Road benchmark, um_000032 (Fritsch, Kuehnl and Geiger, ITSC 2013), CC BY-NC-SA 3.0.

Week 9 recap

- Homography: 8 DOF, 4 points, planar scenes or pure rotation. Parallax breaks it.
- A match must lie on its epipolar line: 2D search $\rightarrow$ 1D. All such lines pass through the epipole.
- $\hat{\mathbf{x}}_1^\top \mathbf{E} \hat{\mathbf{x}}_0 = 0$ on rays (calibrated, 5 DOF); $\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 = 0$ on pixels (uncalibrated, 7 DOF). Both have rank 2.
- $\boldsymbol{l}_1 = \mathbf{F}\mathbf{x}_0$, $\boldsymbol{l}_0 = \mathbf{F}^\top \mathbf{x}_1$, $\mathbf{F}\mathbf{e}_0 = \mathbf{0}$, $\mathbf{e}_1^\top \mathbf{F} = \mathbf{0}^\top$.
- The DLT and the eight-point algorithm are the same recipe: a linear system, the smallest singular vector, coordinates normalised first. $\mathbf{F}$ adds a rank-2 step.
- Warp backwards.
- Rectification makes epipolar lines horizontal: $\mathbf{x}_1^\top \mathbf{F} \mathbf{x}_0 \propto y_0 - y_1$.

This week's work

- **Reading:** Szeliski Sec. 8.1, 8.2.1, 12.1, and the eight-point algorithm in 11.3.1.
- **Exercise:** `w09a`.
- **Lab 8:** estimate a homography with your own DLT from supplied matches, then automatically with SIFT and RANSAC, and stitch.

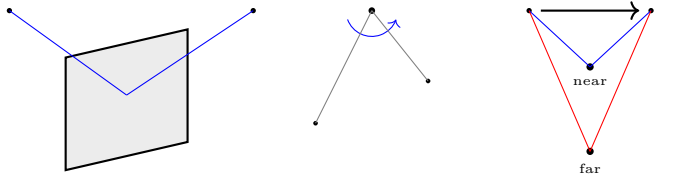
Appendix A: why a plane or a rotation gives a homography

These slides are not covered in lecture. They derive the two cases of "When does a homography apply?" from $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{t}$:

- pure rotation, where depth drops out;
- a plane, where $1/Z$ is linear in the pixel and folds into $\mathbf{H}$;
- translation with depth, where parallax breaks the single-matrix model.

The three cases, drawn

Black dots at the top are camera centres, the other dots scene points; lines are viewing rays.



(a) planar scene — homography works

(b) pure rotation — homography works

(c) translation + depth — parallax: fails

- **(a)** Two cameras see a point on the grey plane (blue rays).
- **(b)** One camera turns in place (blue arrow). Every ray turns by the same amount, whatever the depth of the point on it (grey rays).
- **(c)** The camera translates (black arrow). The rays to the near point (blue) swing through a larger angle than the rays to the far point (red). That difference is parallax.

Why only a plane or a rotation? A pixel is a ray

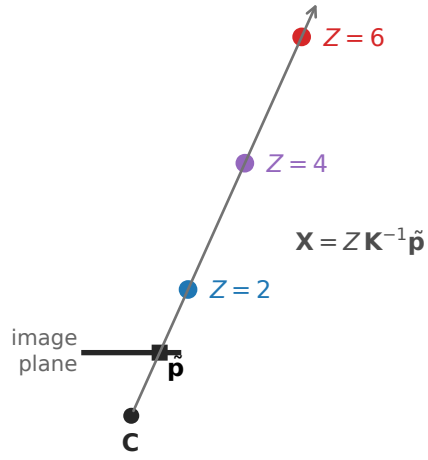
Put camera 1 at the origin. It sees the world point $\mathbf{X} = (X, Y, Z)^\top$ at

$$\tilde{\mathbf{p}} \sim \mathbf{K}\mathbf{X}.$$

The bottom row of $\mathbf{K}$ is $(0, 0, 1)$, so the third entry of $\mathbf{K}\mathbf{X}$ is exactly Z. Dividing by it gives the pixel, so $\mathbf{K}\mathbf{X} = Z\,\tilde{\mathbf{p}}$, and

$$\boxed{\mathbf{X} = Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}}}$$

$\mathbf{K}^{-1}\tilde{\mathbf{p}}$ is the **ray** through the pixel; the depth Z picks the point on it. The points at $Z = 2, 4, 6$ all give the same $\tilde{\mathbf{p}}$: one image cannot tell them apart.



Why only a plane or a rotation? Into camera 2

Camera 2 sees the same point at $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$, so $\tilde{\mathbf{p}}' \sim \mathbf{K}'\mathbf{X}'$. Substitute $\mathbf{X} = Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}}$:

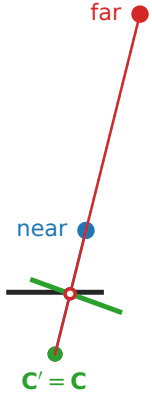
$$\tilde{\mathbf{p}}' \sim \mathbf{K}'\left(Z\,\mathbf{R}\,\mathbf{K}^{-1}\tilde{\mathbf{p}} + \mathbf{t}\right) = Z\,\mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1}\tilde{\mathbf{p}} + \mathbf{K}'\mathbf{t}.$$

Divide by Z, which $\sim$ allows because it ignores a nonzero scale:

$$\tilde{\mathbf{p}}' \sim \underbrace{\mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1}\,\tilde{\mathbf{p}}}_{\text{one matrix for every pixel}} + \underbrace{\frac{\mathbf{K}'\mathbf{t}}{Z}}_{\text{depends on each point's depth}}$$

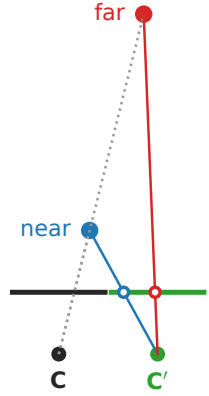
- The first term is a 3×3 matrix applied to $\tilde{\mathbf{p}}$: a homography.
- The second shifts each point by an amount that falls off as $1/Z$. Near points shift more than far ones: that is **parallax**.
- One matrix describes the whole mapping only if the second term vanishes (next slide), or is itself linear in $\tilde{\mathbf{p}}$ (a plane, after that).

Pure rotation: depth drops out



camera 2 rotates about the same centre

same pixel in camera 2: depth does not matter



camera 2 translates

different pixels in camera 2: parallax

A near and a far point on one ray of camera 1 (dotted). Black: camera 1's image plane; green: camera 2's.

Pure rotation, in symbols

Left panel, $\mathbf{t} = \mathbf{0}$: the parallax term $\mathbf{K}'\mathbf{t}/Z$ vanishes, and

$$\tilde{\mathbf{p}}' \sim \mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1}\,\tilde{\mathbf{p}}, \qquad \boxed{\mathbf{H} = \mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1}}$$

Both points land on one pixel, whatever their depth. $\mathbf{K}'$ may differ from $\mathbf{K}$: the camera may even zoom between shots.

Right panel, $\mathbf{t} \neq \mathbf{0}$: $\mathbf{K}'\mathbf{t}/Z$ is larger for the near point than for the far one, so they land on different pixels. No single matrix can send one pixel of camera 1 to two different pixels of camera 2.

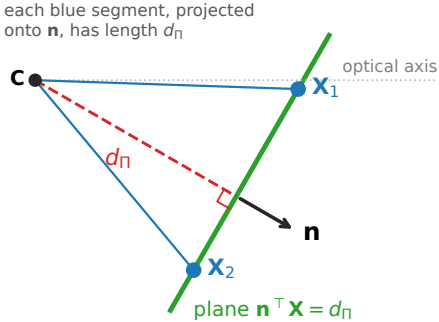
Pure-rotation form: Szeliski Sec. 8.2.3, Eq. (8.38).

The plane equation, opened up

A plane in camera-1 coordinates:

$$\mathbf{n}^\top \mathbf{X} = d_\Pi \quad \Longleftrightarrow \quad n_x X + n_y Y + n_z Z = d_\Pi$$

- $\mathbf{n} = (n_x, n_y, n_z)^\top$: the plane's **unit normal**.
- $d_\Pi > 0$: its **distance** from the camera centre $\mathbf{C}$.
- $\mathbf{C}$ is the origin and $\|\mathbf{n}\| = 1$, so $\mathbf{n}^\top \mathbf{X} = \|\mathbf{X}\| \cos\theta$ (θ the angle between them) is the length of the vector $\mathbf{C} \to \mathbf{X}$ **projected onto** $\mathbf{n}$. For every point on the plane it is d_Π (figure).



Two planes you have met:

- A wall facing the camera, $\mathbf{n} = (0, 0, 1)^\top$: $Z = d_\Pi$.
- The road in Week 2 (Y points down), $\mathbf{n} = (0, 1, 0)^\top$: $Y = h$, the camera height.

A plane gives a homography: in 3D

The scene is the plane $\mathbf{n}^\top \mathbf{X} = d_\Pi$. **Goal:** write $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$ as one matrix times $\mathbf{X}$, with no separate $+\mathbf{t}$.

1. On the plane, $\mathbf{n}^\top \mathbf{X} = d_\Pi$, so the **scalar** $\mathbf{n}^\top \mathbf{X}/d_\Pi$ equals 1.
2. Multiply $\mathbf{t}$ by that 1; nothing changes: $\mathbf{t} = \mathbf{t}\,\dfrac{\mathbf{n}^\top \mathbf{X}}{d_\Pi}$.
3. Regroup: $\mathbf{t}\,(\mathbf{n}^\top \mathbf{X}) = (\mathbf{t}\,\mathbf{n}^\top)\,\mathbf{X}$, because matrix products are associative. $\mathbf{t}\,\mathbf{n}^\top$ is a column times a row: a 3×3 matrix.
4. Substitute into $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$ and factor out $\mathbf{X}$:

$$\mathbf{X}' = \mathbf{R}\mathbf{X} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\,\mathbf{X} = \boxed{\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right)\mathbf{X}}$$

Step 1 holds only on the plane. For a point off it, $\mathbf{n}^\top \mathbf{X}/d_\Pi \neq 1$, and the trick fails.

A plane gives a homography: in pixels

Start from the 3D result, $\mathbf{X}' = \left(\mathbf{R} + \mathbf{t}\,\mathbf{n}^\top / d_\Pi\right)\mathbf{X}$, and work from pixel to pixel, one step per line:

$$\tilde{\mathbf{p}}' \sim \mathbf{K}'\,\mathbf{X}' \qquad \text{camera 2 projects } \mathbf{X}'$$

$$= \mathbf{K}'\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right)\mathbf{X} \qquad \text{the 3D result}$$

$$= \mathbf{K}'\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right) Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}} \qquad \mathbf{X} = Z\,\mathbf{K}^{-1}\tilde{\mathbf{p}} \text{ (pixel to point)}$$

$$= Z\,\mathbf{K}'\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right)\mathbf{K}^{-1}\tilde{\mathbf{p}} \qquad Z \text{ is a scalar: move it to the front}$$

$$\sim \boxed{\mathbf{K}'\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right)\mathbf{K}^{-1}}\,\tilde{\mathbf{p}} = \mathbf{H}\,\tilde{\mathbf{p}} \qquad \sim \text{ ignores the nonzero scale } Z$$

With $\mathbf{t} = \mathbf{0}$ this is $\mathbf{K}'\mathbf{R}\mathbf{K}^{-1}$, the pure-rotation homography.

On a plane, $1/Z$ is linear in the pixel

Put $\mathbf{X} = Z\,(x_n, y_n, 1)^\top$ into the plane equation $\mathbf{n}^\top \mathbf{X} = d_\Pi$ and use $x_n = (x - c_x)/\alpha_x$, $y_n = (y - c_y)/\alpha_y$:

$$\frac{1}{Z} = \frac{n_x x_n + n_y y_n + n_z}{d_\Pi} = \frac{1}{d_\Pi}\left(n_x \frac{x - c_x}{\alpha_x} + n_y \frac{y - c_y}{\alpha_y} + n_z \right) = a\,x + b\,y + c$$

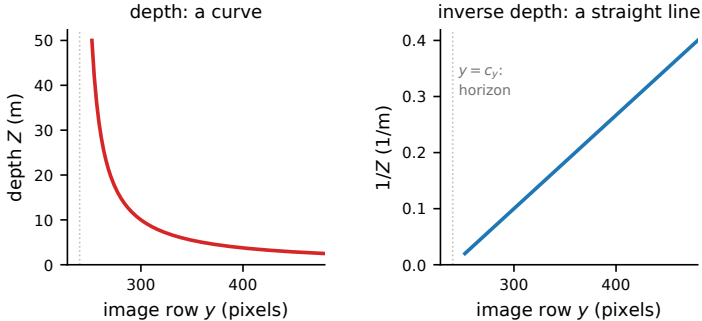
- a, b, c depend only on the plane and $\mathbf{K}$, **not on the pixel**. So $1/Z = (a, b, c)\,\tilde{\mathbf{p}}$: a fixed row of 3 numbers times $\tilde{\mathbf{p}}$. That is what "linear in the pixel" means.
- Off a plane there are no such a, b, c: each pixel's Z is whatever the scene happens to be there.

Example: the road

The road below the camera is the plane $Y = h$: $\mathbf{n} = (0, 1, 0)^\top$, $d_\Pi = h$, the camera height. Then $n_x = n_z = 0$ and

$$\frac{1}{Z} = \frac{y - c_y}{\alpha_y \, h}$$

Depth against the image row is a curve; inverse depth is a **straight line**. It is zero at the horizon row $y = c_y$ and grows steadily down the image.



Illustrative values: $\alpha_y = 400$ px, $c_y = 240$, $h = 1.5$ m.

Folding the parallax term into $\mathbf{H}$

"Into camera 2" left a term that depends on depth:

$$\tilde{\mathbf{p}}' \sim \mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1}\tilde{\mathbf{p}} + \mathbf{K}'\mathbf{t}\cdot\frac{1}{Z}$$

On the plane, $\dfrac{1}{Z} = \dfrac{\mathbf{n}^\top\mathbf{K}^{-1}}{d_\Pi}\,\tilde{\mathbf{p}}$, a fixed row times $\tilde{\mathbf{p}}$. Substitute:

$$\mathbf{K}'\mathbf{t}\cdot\frac{1}{Z} = \mathbf{K}'\mathbf{t}\left(\frac{\mathbf{n}^\top\mathbf{K}^{-1}}{d_\Pi}\,\tilde{\mathbf{p}}\right) = \underbrace{\left(\frac{\mathbf{K}'\,\mathbf{t}\,\mathbf{n}^\top\mathbf{K}^{-1}}{d_\Pi}\right)}_{\text{column}\times\text{row} = 3\times 3}\tilde{\mathbf{p}}$$

Now both terms are a matrix times $\tilde{\mathbf{p}}$, so they add:

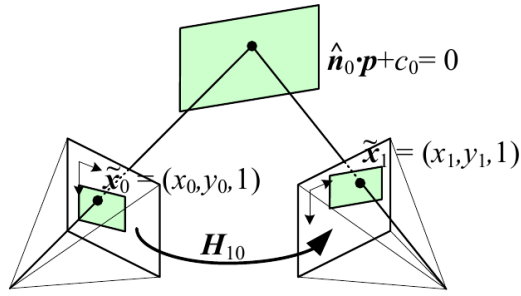
$$\tilde{\mathbf{p}}' \sim \left(\mathbf{K}'\mathbf{R}\,\mathbf{K}^{-1} + \frac{\mathbf{K}'\,\mathbf{t}\,\mathbf{n}^\top\mathbf{K}^{-1}}{d_\Pi}\right)\tilde{\mathbf{p}} = \mathbf{K}'\left(\mathbf{R} + \frac{\mathbf{t}\,\mathbf{n}^\top}{d_\Pi}\right)\mathbf{K}^{-1}\tilde{\mathbf{p}}$$

The same $\mathbf{H}$. Off the plane, $1/Z$ is not a fixed row times $\tilde{\mathbf{p}}$, and no single matrix absorbs it.

A plane gives a homography, drawn

Two cameras see points on one plane (green). Wherever the cameras are, $\mathbf{H}_{10}$ carries the plane's image in one camera to its image in the other.

The figure's notation differs from ours: $\tilde{\mathbf{x}}_0$ and $\tilde{\mathbf{x}}_1$ are our $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{p}}'$, $\mathbf{H}_{10}$ is our $\mathbf{H}$, and its plane $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + c_0 = 0$ is our $\mathbf{n}^\top \mathbf{X} = d_\Pi$, with $c_0 = -d_\Pi$.



Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 2.12(b).

Parallax: why translation breaks it

If the camera translates, a point's image displacement depends on its **depth**: the term $\mathbf{K}'\mathbf{t}/Z$.

- Near objects (small Z) shift a lot.
- Far objects (large Z) shift a little.

A homography assigns one displacement per image position, with no depth term. It cannot represent a depth-dependent mapping.

Practical consequence (try it in Lab 8's optional part): stitch two photos taken while walking sideways and the far background aligns while near objects **ghost**, or vice versa. You can align one depth, not both.

Rotate in place when you shoot a panorama.

Parallax in a real mosaic



(a) (b) (c)

Left, a mosaic with parallax: the stop sign appears twice. Middle and right, a local correction on top of the global alignment removes the ghost (Szeliski Sec. 8.3.2). That repair is beyond this course; rotating in place avoids it.

Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., 2022, Fig. 8.10, from Shum and Szeliski (2000), © 2000 IEEE.

Appendix B: a family of solutions

"Three collinear points are not enough" showed two homographies that fit the same four matches. This slide shows where the second one comes from.

Write the line through points 1, 2 and 3 as $ax + by + c = 0$, or $\mathbf{l}^\top \tilde{\mathbf{p}} = 0$ with $\mathbf{l} = (a, b, c)^\top$. Given the true $\mathbf{H}$, build

$$\mathbf{H}' = \mathbf{H} + \gamma\,(\mathbf{H}\tilde{\mathbf{p}}_4)\,\mathbf{l}^\top \qquad \text{for any } \gamma \text{ with } 1 + \gamma\,\mathbf{l}^\top \tilde{\mathbf{p}}_4 \neq 0.$$

- **Points 1, 2, 3** lie on the line, so $\mathbf{l}^\top \tilde{\mathbf{p}}_i = 0$ and the added term vanishes: $\mathbf{H}'\tilde{\mathbf{p}}_i = \mathbf{H}\tilde{\mathbf{p}}_i$.
- **Point 4**: $\mathbf{H}'\tilde{\mathbf{p}}_4 = (1 + \gamma\,\mathbf{l}^\top \tilde{\mathbf{p}}_4)\,\mathbf{H}\tilde{\mathbf{p}}_4$, the same homogeneous vector rescaled, so the same pixel.

Every γ gives a homography that fits all four matches: a one-parameter family. The figure uses the γ that rescales $\mathbf{H}\tilde{\mathbf{p}}_4$ by 0.7.

What the DLT returns. $\mathbf{A}$ is 8×9 with rank 7, so its null space is **two**-dimensional: the last two right singular vectors both solve $\mathbf{A}\mathbf{h} = \mathbf{0}$, and the SVD hands back an arbitrary member of the family. With real, nearly collinear points the system is not exactly singular but badly conditioned: small noise swings the estimate a long way. Choose correspondences spread across the image.

Figure credits

- Figs. 2.4, 2.12(b), 8.4, 8.10, 11.11, 12.3 and 12.4: R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., Springer, 2022. Fig. 8.10 originally from H.-Y. Shum and R. Szeliski, *IJCV* 36(2), 2000 (credited © 2000 IEEE in the book); Fig. 12.4 from C. Loop and Z. Zhang, CVPR 1999, © 1999 IEEE. Reproduced for teaching under fair dealing; not covered by this deck's licence.
- KITTI images: KITTI Road benchmark; J. Fritsch, T. Kuehnl and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," ITSC 2013. CC BY-NC-SA 3.0.
- The rectification diagram: F. Z. Qureshi, *Computer Vision Notes*, CC BY-NC 4.0.
- The church photographs and "Where the epipole lands": R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004, Ch. 9, via *Computer Vision Notes*, notebook 21. Reproduced for teaching under fair dealing; not covered by this deck's licence.
- All other figures: drawn or computed for this course.

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.