Robot Sensing & Perception

"Seeing is forgetting the name of what one sees"

– L. Weschler

METR 4202: Advanced Control & **Robotics**

Dr Surya Singh -- Lecture # 6

September 3, 2014

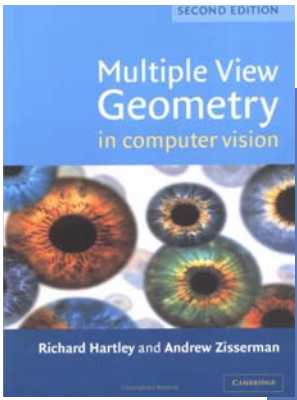
metr4202@itee.uq.edu.au

http://robotics.itee.uq.edu.au/~metr4202/

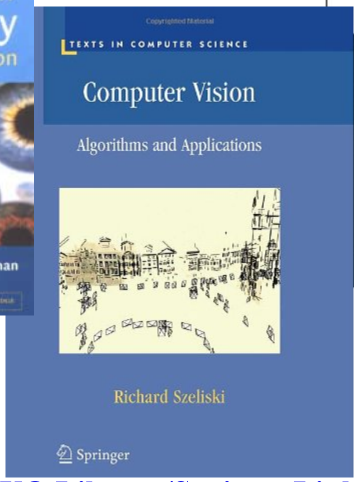
Schedule

Date	Lecture (W: 11:10-12:40, 24-402)
30-Jul	Introduction
6-Aug	Representing Position & Orientation & State (Frames, Transformation Matrices & Affine Transformations)
13-Aug	Robot Kinematics (& Ekka Day)
20-Aug	Robot Dynamics & Control
27-Aug	Robot Motion
3-Sep	**Sensing & Perception**
10-Sep	Multiple View Geometry (Computer Vision)
17-Sep	Navigation & Localization (+ Prof. M. Srinivasan)
24-Sep	Motion Planning + Control
1-Oct	*Study break*
8-Oct	State-Space Modelling
15-Oct	Shaping the Dynamic Response
22-Oct	Linear Observers & LQR
29-Oct	Applications in Industry & Course Review

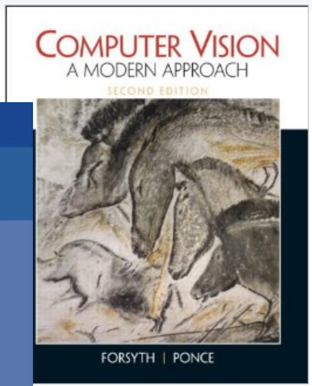
Reference Material



[UQ Library](#)
(ePDF)



[UQ Library/SpringerLink](#)

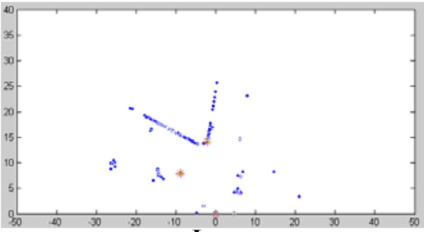


[UQ Library](#)
(Hardcopy)

Quick Outline

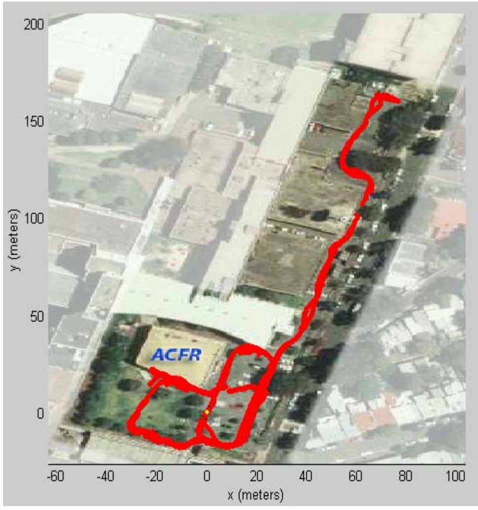
1. **Perception → Camera Sensors**
   1. Image Formation
      → "Computational Photography"
   2. Calibration
   3. Feature extraction
   4. Stereopsis and depth
   5. Optical flow

Sensor Information



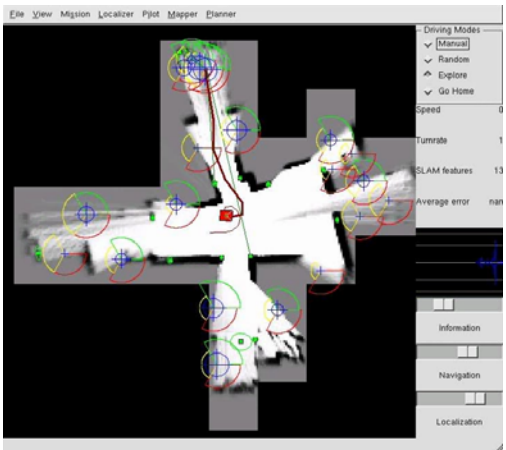
Laser

Vision/Cameras

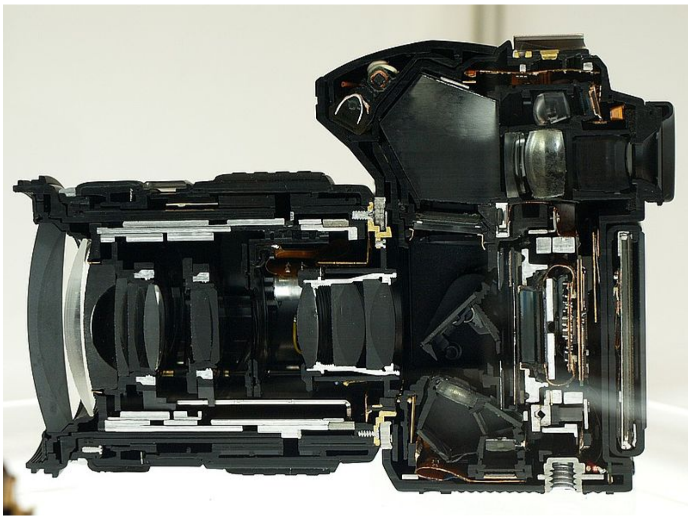


GPS

Mapping: Indoor robots



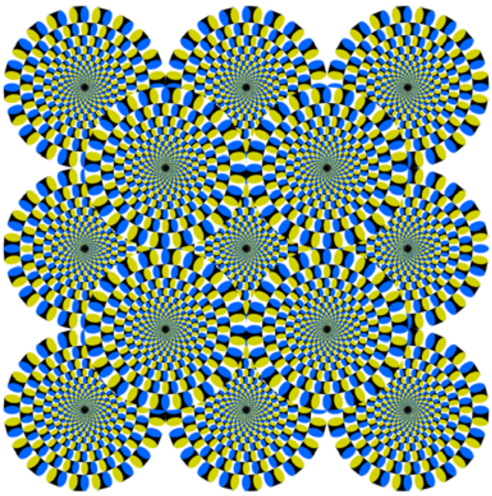
Cameras



Wikipedia, E-30-Cutmodel

Perception

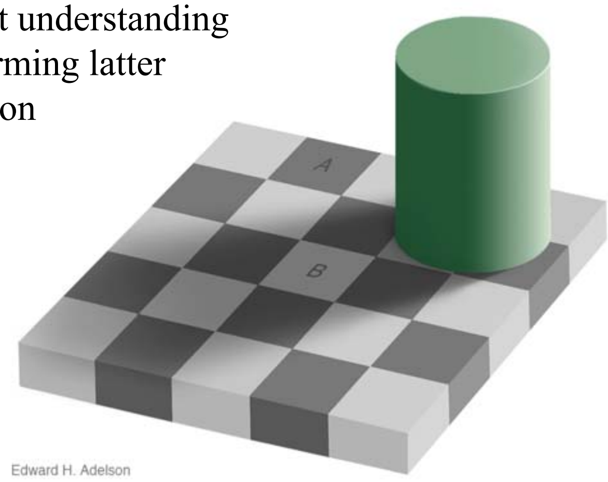
- Making Sense from Sensors



http://www.michaelbach.de/ot/mot_rotsnake/index.html

Perception

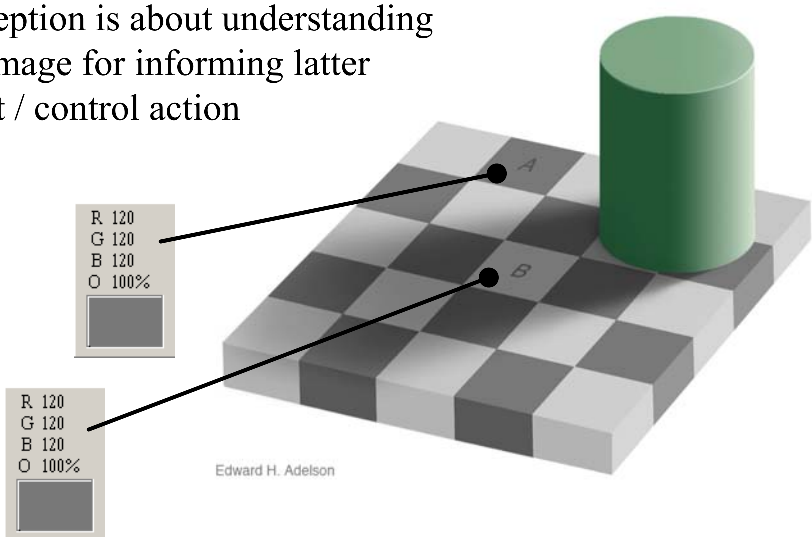
- Perception is about understanding the image for informing latter robot / control action



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

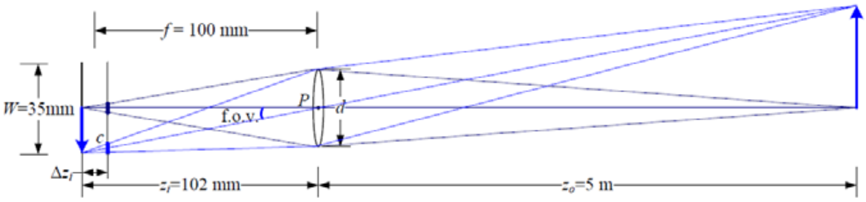
Perception

- Perception is about understanding the image for informing latter robot / control action



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

Image Formation: Lens Optics

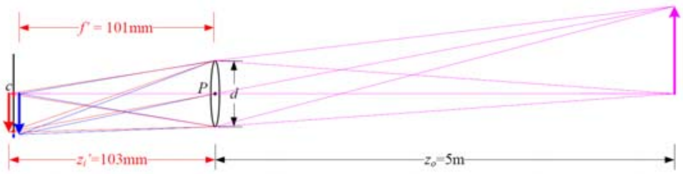


$$\frac{1}{z_0} + \frac{1}{z_1} = \frac{1}{f}$$

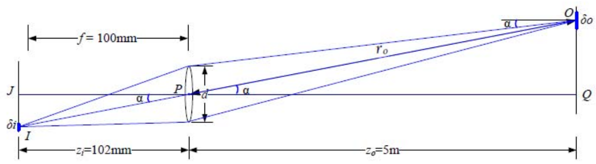
Sec. 2.2 from Szeliski, *Computer Vision: Algorithms and Applications*

Image Formation:
Lens Optics (Chromatic Aberration & Vignetting)

- Chromatic Aberration:



- Vignetting:



$$E = L\frac{\pi}{4}\left(\frac{d}{f}\right)^2 \cos^4 \alpha$$

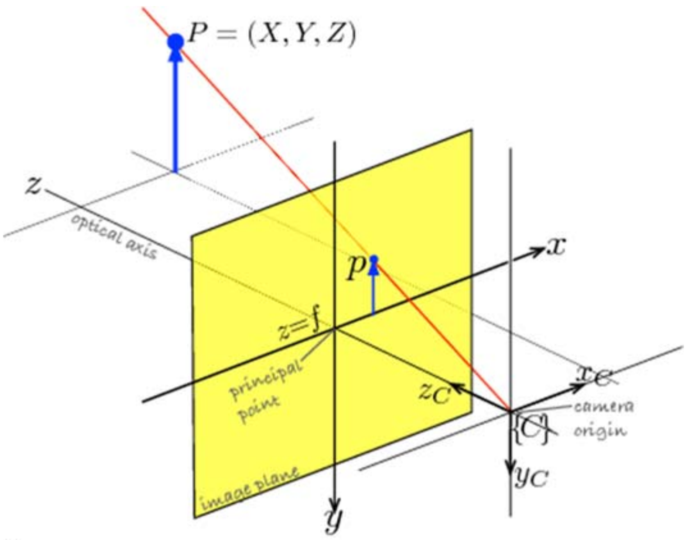
Sec. 2.2 from Szeliski, *Computer Vision: Algorithms and Applications*

Image Formation:
Lens Optics (Aperture / Depth of Field)

$$N = \frac{f}{\#} = \frac{f}{d}$$

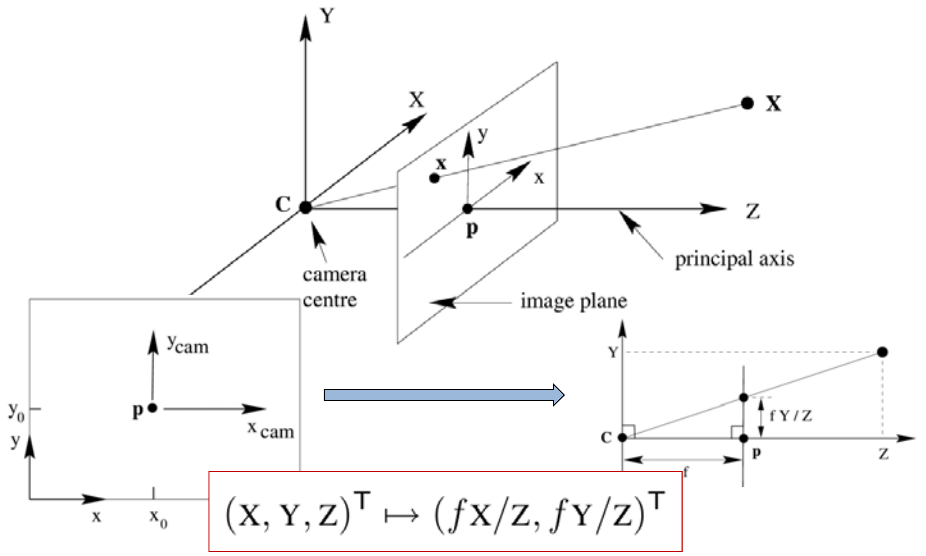
http://en.wikipedia.org/wiki/File:Aperture_in_Canon_50mm_f1.8_II_lens.jpg

Image Formation – Single View Geometry



Corke, Ch. 11

Image Formation – Single View Geometry [II]



$$(X, Y, Z)^{\mathsf{T}} \mapsto (fX/Z, fY/Z)^{\mathsf{T}}$$

Hartly & Zisserman, Ch. 6

Image Formation – Single View Geometry ➔ Camera Projection Matrix

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- x = Image point
- **X** = World point
- *K* = Camera Calibration Matrix

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}$$

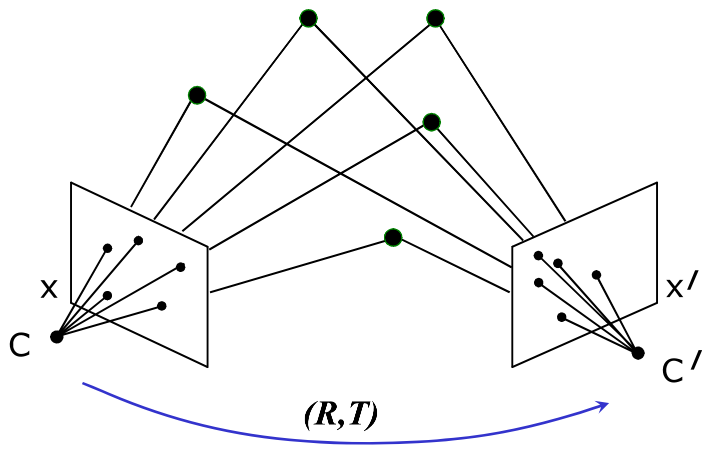
$$\mathbf{x} = K[I \mid \mathbf{0}]\mathbf{X}_{\text{cam}}.$$

➔Perspective Camera as:

where: P is 3×4 and of **rank 3**

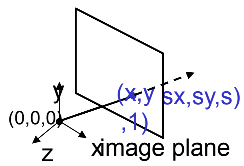
$$P = K[R \mid \mathbf{t}]$$

Image Formation – Two-View Geometry [Stereopsis] ➔ Fundamental Matrix



Planar Projective Transformations

- Perspective projection of a plane
  - lots of names for this:
    - homography, colineation, planar projective map
  - Easily modeled using homogeneous coordinates



$$\begin{bmatrix} sx' \\ sy' \\ s \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

p’ **H** **p**

To apply a homography **H**

- compute **p’** = **Hp**
- **p”** = **p’**/s normalize by dividing by third component

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Transformations

- Forward Warp

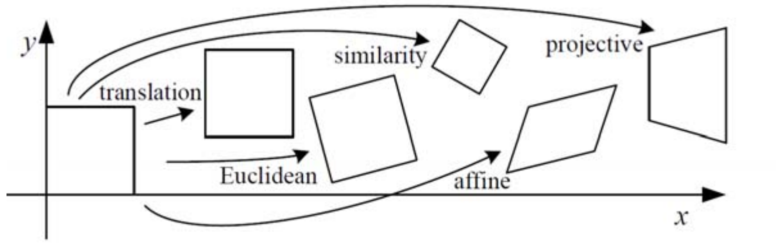
> **procedure** *forwardWarp*$(f, h,$ **out** $g)$:
>
> For every pixel x in $f(x)$
>
> 1. Compute the destination location $x' = h(x)$.
> 2. Copy the pixel $f(x)$ to $g(x')$.

- **<u>Inverse Warp</u>**

> **procedure** *inverseWarp*$(f, h,$ **out** $g)$:
>
> For every pixel x' in $g(x')$
>
> 1. Compute the source location $x = \hat{h}(x')$
> 2. Resample $f(x)$ at location x and copy to $g(x')$

Sec. 3.6 from Szeliski, *Computer Vision: Algorithms and Applications*

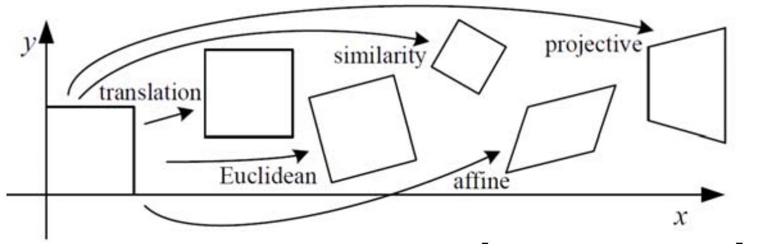
Transformations



- x': New Image & $\underline{x}$: Old Image
- Euclidean: (Distances preserved) $$x' = \begin{bmatrix} R & t \end{bmatrix} \underline{x}$$
- Similarity (Scaled Rotation): (Angles preserved) $$x' = \begin{bmatrix} sR & t \end{bmatrix} \underline{x}$$

Fig. 2.4 from Szeliski, *Computer Vision: Algorithms and Applications*

Transformations [2]



- Affine :
  (|| lines remain ||)

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \underline{x}$$

- Projective:
  (straight lines preserved)
  H: Homogenous 3x3 Matrix

$$x' = \mathbf{H}\underline{x}$$

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}}$$

$$y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$

Fig. 2.4 from Szeliski, Computer Vision: Algorithms and Applications

2-D Transformations

➔ x' = point in the **new** (or 2nd) image
➔ x = point in the old image

- Translation x' = x + t
- Rotation x' = R x + t
- Similarity x' = sR x + t
- Affine x' = A x
- Projective x' = A x

here, x is an inhomogeneous pt (2-vector)
x' is a homogeneous point

2-D Transformations

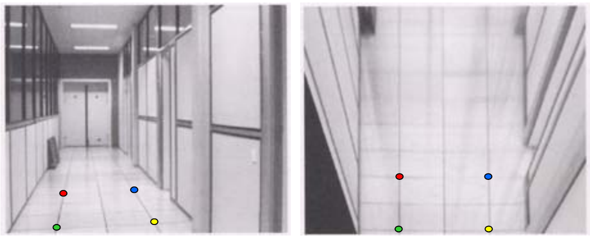
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

3D Transformations

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{3\times 4}$	3	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{3\times 4}$	6	lengths $+\cdots$	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{3\times 4}$	7	angles $+\cdots$	
affine	$\begin{bmatrix} A \end{bmatrix}_{3\times 4}$	12	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{4\times 4}$	15	straight lines	

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Image Rectification



To unwarp (rectify) an image

- solve for **H** given **p”** and **p**
- solve equations of the form: **sp”** = **Hp**
  - linear in unknowns: s and coefficients of **H**
  - need at least 4 points

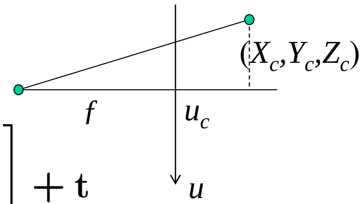
Slide from Szeliski, *Computer Vision: Algorithms and Applications*

3D Projective Geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: P = (X,Y,Z,W)
  - Duality
    - A plane L is also represented by a 4-vector
    - Points and planes are dual in 3D: L P=0
  - Projective transformations
    - Represented by 4x4 matrices T: P’ = TP, L’ = L T-1
  - Lines are a special case…

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

3D → 2D Perspective Projection (Image Formation Equations)



$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} \mathbf{R} \end{bmatrix}_{3\times 3} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

3D → 2D Perspective Projection

- Matrix Projection (camera matrix):

$$\mathbf{p} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{\Pi P}$$

It's useful to decompose $\prod$ into $\mathbf{T} \rightarrow \mathbf{R} \rightarrow$ project $\rightarrow \mathbf{A}$

$$\mathbf{\Pi} \; \underbrace{\begin{matrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1/f \end{matrix}}_{\text{intrinsics}} \; \underbrace{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}}_{\text{projection}} \; \underbrace{\begin{matrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{matrix}}_{\text{orientation}} \; \underbrace{\begin{matrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{matrix}}_{\text{position}}$$

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Calibration matrix

- Is this form of K good enough?
- non-square pixels (digital video)
- skew
- radial distortion

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{K}\ \mathbf{X}_c$$

$$\begin{bmatrix} fa & s & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{K}$$

From Szeliski, *Computer Vision: Algorithms and Applications*

Calibration

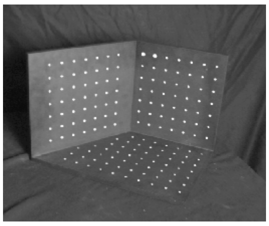
See: *Camera Calibration Toolbox for Matlab*
(http://www.vision.caltech.edu/bouguetj/calib_doc/)

- Intrinsic: Internal Parameters
  - **Focal length:** The focal length in pixels.
  - **Principal point:** The principal point
  - **Skew coefficient:**
    The skew coefficient defining the angle between the x and y pixel axes.
  - **Distortions:** The image distortion coefficients (radial and tangential distortions) (typically two quadratic functions)

- Extrinsics: Where the Camera (image plane) is placed:
  - **Rotations:** A set of 3x3 rotation matrices for each image
  - **Translations:** A set of 3x1 translation vectors for each image

Camera calibration

- Determine camera parameters from known 3D points or calibration object(s)
- internal or intrinsic parameters such as focal length, optical center, aspect ratio: what kind of camera?
- external or extrinsic (pose) parameters: where is the camera?
- How can we do this?



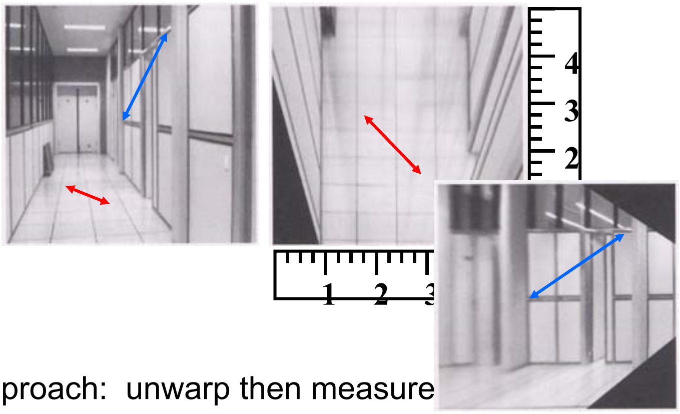
From Szeliski, *Computer Vision: Algorithms and Applications*

Camera calibration – approaches

- Possible approaches:
  - linear regression (least squares)
  - non-linear optimization
  - vanishing points
  - multiple planar patterns
  - panoramas (rotational motion)

From Szeliski, *Computer Vision: Algorithms and Applications*

Measurements on Planes
(You can not just add a tape measure!)



Approach: unwarp then measure

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Projection Models

- Orthographic
- Weak Perspective
- Affine
- Perspective
- Projective

$$\Pi = \begin{bmatrix} i_x & i_y & i_z & t_x \\ j_x & j_y & j_z & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Pi = f \begin{bmatrix} i_x & i_y & i_z & t_x \\ j_x & j_y & j_z & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Pi = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Pi = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$$

$$\Pi = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

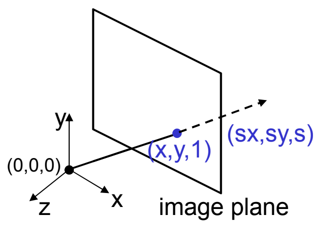
Properties of Projection

- Preserves
  - Lines and conics
  - Incidence
  - Invariants (cross-ratio)

- Does not preserve
  - Lengths
  - Angles
  - Parallelism

The Projective Plane

- Why do we need homogeneous coordinates?
  - Represent points at infinity, homographies, perspective projection, multi-view relationships
- What is the geometric intuition?
  - A point in the image is a ray in projective space

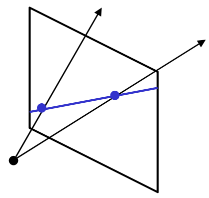


- Each *point* (x,y) on the plane is represented by a *ray* (sx,sy,s)
  - all points on the ray are equivalent: $(x, y, 1) \cong (sx, sy, s)$

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Projective Lines

- What is a line in projective space?



- A line is a *plane* of rays through origin
  - all rays (x,y,z) satisfying: ax + by + cz = 0

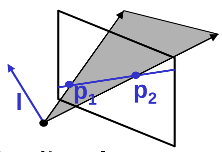
$$\text{in vector notation: } \quad 0 = \underbrace{\begin{bmatrix} a & b & c \end{bmatrix}}_{\mathbf{l}^T} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\mathbf{p}}$$

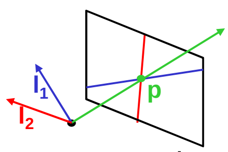
- A line is represented as a homogeneous 3-vector **l**

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Point and Line Duality

– A line l is a homogeneous 3-vector (a ray)
– It is ⊥ to every point (ray) p on the line: lT p=0

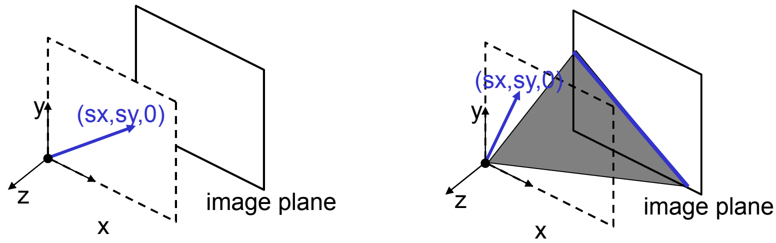




- What is the line **l** spanned by rays $\mathbf{p_1}$ and $\mathbf{p_2}$?
  - **l** is ⊥ to $\mathbf{p_1}$ and $\mathbf{p_2}$ ⇒ $\mathbf{l} = \mathbf{p_1} \times \mathbf{p_2}$ (**l** is the plane normal)
- What is the intersection of two lines $\mathbf{l_1}$ and $\mathbf{l_2}$?
  - **p** is ⊥ to $\mathbf{l_1}$ and $\mathbf{l_2}$ ⇒ $\mathbf{p} = \mathbf{l_1} \times \mathbf{l_2}$
- Points and lines are *dual* in projective space
  - every property of points also applies to lines

Ideal points and lines

- Ideal point ("point at infinity")
  - p ≅ (x, y, 0) – parallel to image plane
  - It has infinite image coordinates

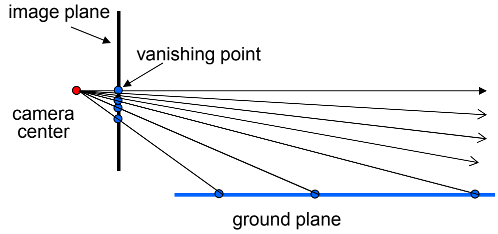


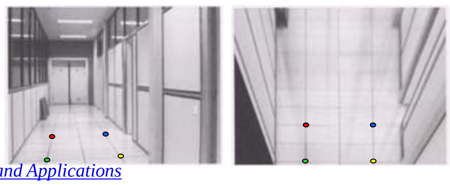
Line at infinity

- $l_\infty \cong (0, 0, 1)$ – parallel to image plane
- Contains all ideal points

Vanishing Points

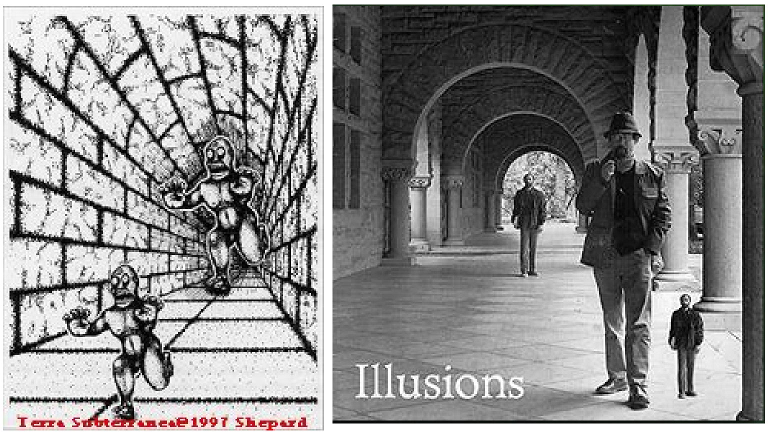
- Vanishing point
  - projection of a point at infinity
  - whiteboard capture, architecture,…





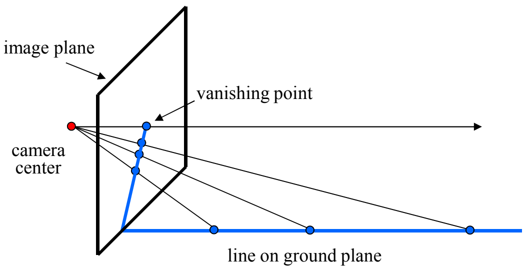
Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Fun With Vanishing Points

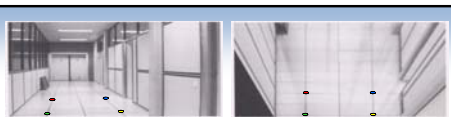


Slide from Szeliski, *Computer Vision: Algorithms and Applications*

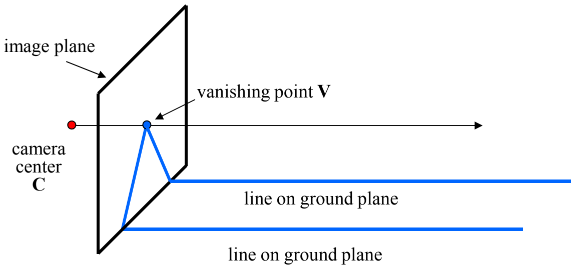
Vanishing Points (2D)



Vanishing Points

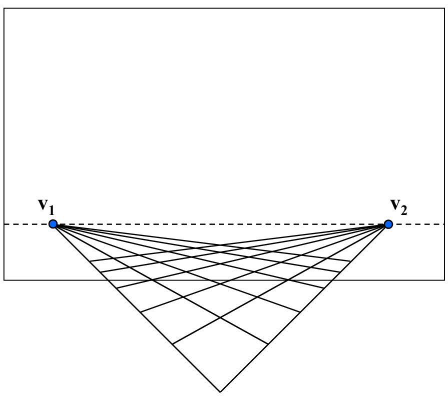


- Properties
  - Any two parallel lines have the same vanishing point
  - The ray from C through v point is parallel to the lines
  - An image may have more than one vanishing point

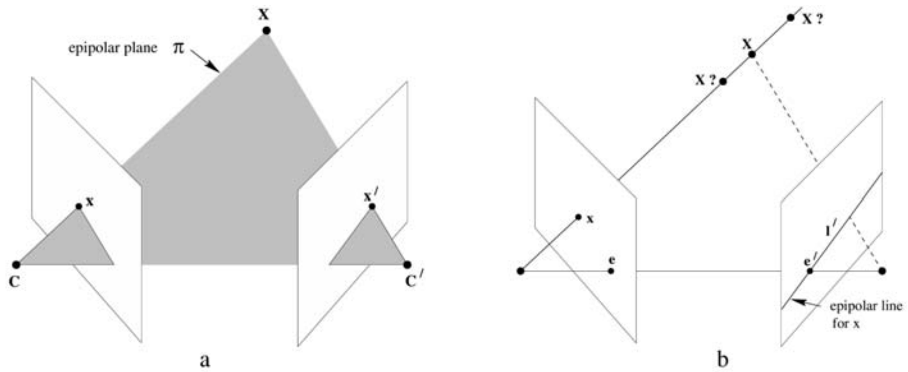


Vanishing Lines

- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point
  - The union of all of these vanishing points is the horizon line



Two-View Geometry: Epipolar Plane



- **Epipole:** The *point* of intersection of the line joining the camera centres (the baseline) with the image plane. Equivalently, the epipole is the image in one view of the camera centre of the other view.
- **Epipolar plane** is a plane containing the baseline. There is a one-parameter family (a pencil) of epipolar planes
- **Epipolar line** is the intersection of an epipolar plane with the image plane. All epipolar lines intersect at the epipole. An epipolar plane intersects the left and right image planes in epipolar lines, and defines the correspondence between the lines.

Two-frame methods

- Two main variants:
- Calibrated: "Essential matrix" E
  use ray directions (xi, xi')
- Uncalibrated: "Fundamental matrix" F

- [Hartley & Zisserman 2000]

From Szeliski, *Computer Vision: Algorithms and Applications*

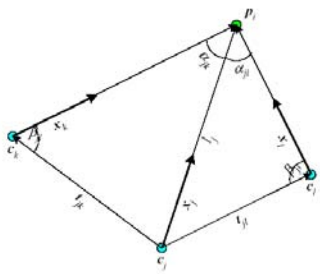
Fundamental matrix

- Camera calibrations are unknown
- x' F x = 0 with F = [e]× H = K'[t]× R K-1
- Solve for F using least squares (SVD)
  - re-scale (xi, xi') so that |xi|≈1/2 [Hartley]
- e (epipole) is still the least singular vector of F
- H obtained from the other two s.v.s
- "plane + parallax" (projective) reconstruction
- use self-calibration to determine K [Pollefeys]

From Szeliski, *Computer Vision: Algorithms and Applications*

Essential matrix

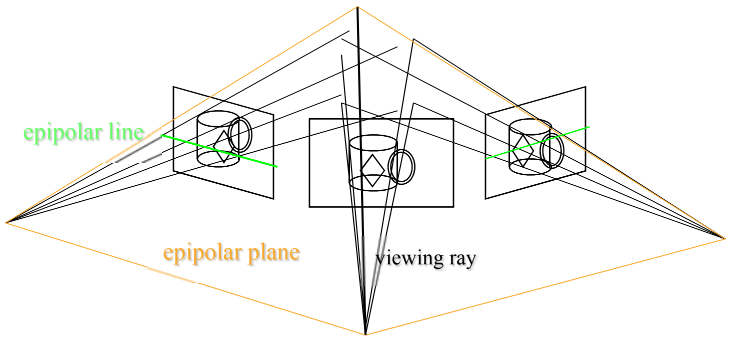
- Co-planarity constraint:
- x' ≈ R x + t
- [t]× x' ≈ [t]× R x
- x' [t]× x' ≈ x' [t]× R x
- x' E x = 0 with E =[t]× R
- Solve for E using least squares (SVD)
- t is the least singular vector of E
- R obtained from the other two s.v.s



From Szeliski, *Computer Vision: Algorithms and Applications*

Stereo: Epipolar geometry

- Match features along epipolar lines



Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Stereo: epipolar geometry

- for two images (or images with collinear camera centers), can find epipolar lines
- epipolar lines are the projection of the pencil of planes passing through the centers

- Rectification: warping the input images (perspective transformation) so that epipolar lines are horizontal

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Fundamental Matrix

- The fundamental matrix is the algebraic representation of epipolar geometry.

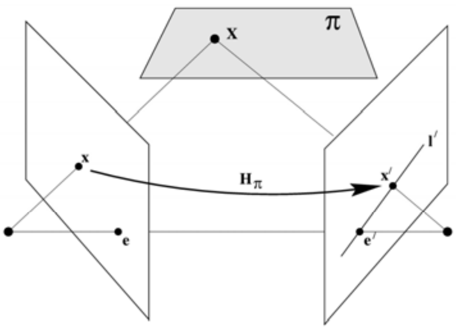


Fig. 9.5. *A point* $\mathbf{x}$ *in one image is transferred via the plane* π *to a matching point* $\mathbf{x}'$ *in the second image. The epipolar line through* $\mathbf{x}'$ *is obtained by joining* $\mathbf{x}'$ *to the epipole* $\mathbf{e}'$*. In symbols one may write* $\mathbf{x}' = \mathtt{H}_\pi \mathbf{x}$ *and* $\mathbf{l}' = [\mathbf{e}']_\times \mathbf{x}' = [\mathbf{e}']_\times \mathtt{H}_\pi \mathbf{x} = \mathtt{F}\mathbf{x}$ *where* $\mathtt{F} = [\mathbf{e}']_\times \mathtt{H}_\pi$ *is the fundamental matrix.*

Fundamental Matrix Example

- Suppose the camera matrices are those of a calibrated stereo rig with the world origin at the first camera

$$\mathtt{P} = \mathtt{K}[\mathtt{I} \mid \mathbf{0}] \qquad \mathtt{P}' = \mathtt{K}'[\mathtt{R} \mid \mathbf{t}].$$

- Then:

$$\mathtt{P}^+ = \begin{bmatrix} \mathtt{K}^{-1} \\ \mathbf{0}^\mathsf{T} \end{bmatrix} \qquad \mathbf{C} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}$$

- Epipoles are at:

$$\mathbf{e} = \mathtt{P}\begin{pmatrix} -\mathtt{R}^\mathsf{T}\mathbf{t} \\ 1 \end{pmatrix} = \mathtt{K}\mathtt{R}^\mathsf{T}\mathbf{t} \qquad \mathbf{e}' = \mathtt{P}'\begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix} = \mathtt{K}'\mathbf{t}.$$

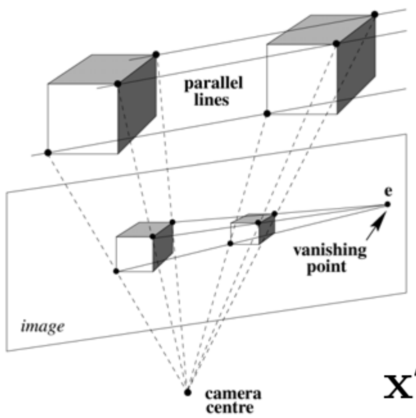
∴

$$\mathtt{F} = [\mathbf{e}']_\times \mathtt{K}'\mathtt{R}\mathtt{K}^{-1} = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1} = \mathtt{K}'^{-\mathsf{T}}\mathtt{R}[\mathtt{R}^\mathsf{T}\mathbf{t}]_\times \mathtt{K}^{-1} = \mathtt{K}'^{-\mathsf{T}}\mathtt{R}\mathtt{K}^\mathsf{T}[\mathbf{e}]_\times$$

Summary of fundamental matrix properties

- F is a rank 2 homogeneous matrix with 7 degrees of freedom.
- **Point correspondence**: If $\mathbf{x}$ and $\mathbf{x}'$ are corresponding image points, then $\mathbf{x}'^{\mathsf{T}} F \mathbf{x} = 0$.
- **Epipolar lines**:
  - $\mathbf{l}' = F\mathbf{x}$ is the epipolar line corresponding to $\mathbf{x}$.
  - $\mathbf{l} = F^{\mathsf{T}}\mathbf{x}'$ is the epipolar line corresponding to $\mathbf{x}'$.
- **Epipoles**:
  - $F\mathbf{e} = \mathbf{0}$.
  - $F^{\mathsf{T}}\mathbf{e}' = \mathbf{0}$.
- **Computation from camera matrices** P, P':
  - General cameras,
    $F = [\mathbf{e}']_\times P'P^+$, where P^+ is the pseudo-inverse of P, and $\mathbf{e}' = P'\mathbf{C}$, with $P\mathbf{C} = \mathbf{0}$.
  - Canonical cameras, $P = [I \mid \mathbf{0}]$, $P' = [M \mid \mathbf{m}]$,
    $F = [\mathbf{e}']_\times M = M^{-\mathsf{T}}[\mathbf{e}]_\times$, where $\mathbf{e}' = \mathbf{m}$ and $\mathbf{e} = M^{-1}\mathbf{m}$.
  - Cameras not at infinity $P = K[I \mid \mathbf{0}]$, $P' = K'[R \mid \mathbf{t}]$,
    $F = K'^{-\mathsf{T}}[\mathbf{t}]_\times RK^{-1} = [K'\mathbf{t}]_\times K'RK^{-1} = K'^{-\mathsf{T}}RK^{\mathsf{T}}[KR^{\mathsf{T}}\mathbf{t}]_\times$.

Fundamental Matrix & Motion



$$\mathbf{x}'^{\mathsf{T}} F \mathbf{x} = 0.$$

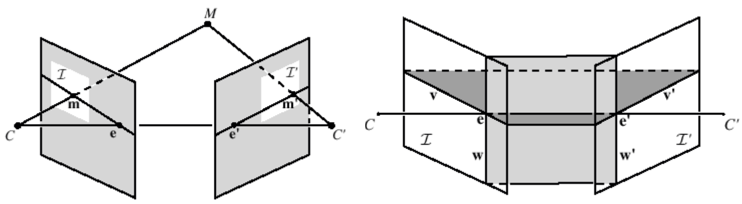
- Under a pure translational camera motion, 3D points appear to slide along parallel rails. The images of these parallel lines intersect in a vanishing point corresponding to the translation direction. The epipole **e** is the vanishing point.

Cool Robotics Share

D. Wedge, *The Fundamental Matrix Song*

Rectification

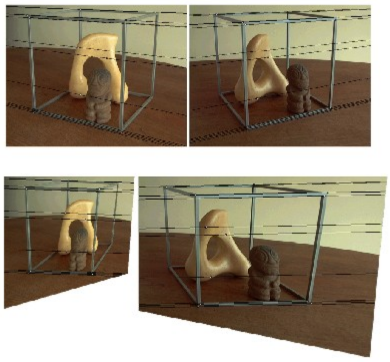
- Project each image onto same plane, which is parallel to the epipole
- Resample lines (and shear/stretch) to place lines in correspondence, and minimize distortion



- [Zhang and Loop, MSR-TR-99-21]

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Rectification



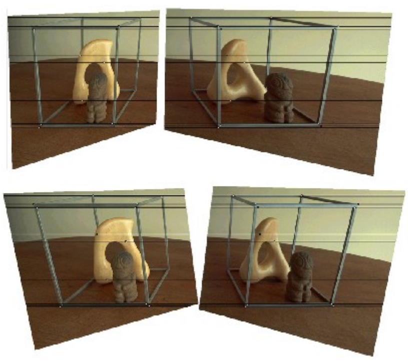
(a) Original image pair overlayed with several epipolar lines.

(b) Image pair transformed by the specialized projective mapping $\mathbf{H}_p$ and $\mathbf{H}'_p$. Note that the epipolar lines are now parallel to each other in each image.

BAD!

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Rectification



(c) Image pair transformed by the similarity $\mathbf{H}_r$ and $\mathbf{H}'_r$. Note that the image pair is now rectified (the epipolar lines are horizontally aligned).

(d) Final image rectification after shearing transform $\mathbf{H}_s$ and $\mathbf{H}'_s$. Note that the image pair remains rectified, but the horizontal distortion is reduced.

GOOD!

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

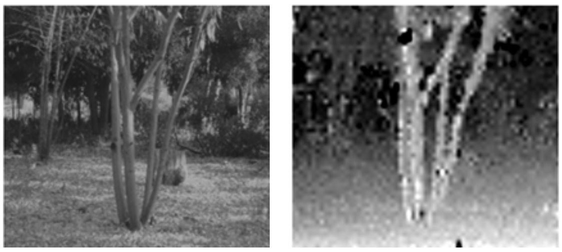
Matching criteria

- Raw pixel values (correlation)
- Band-pass filtered images [Jones & Malik 92]
- "Corner" like features [Zhang, …]
- Edges [many people…]
- Gradients [Seitz 89; Scharstein 94]
- Rank statistics [Zabih & Woodfill 94]

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Finding correspondences

- Apply feature matching criterion (e.g., correlation or Lucas-Kanade) at all pixels simultaneously
- Search only over epipolar lines (many fewer candidate positions)



Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Image registration (revisited)

- How do we determine correspondences?
  - block matching or SSD (sum squared differences)

  d is the disparity (horizontal motion)

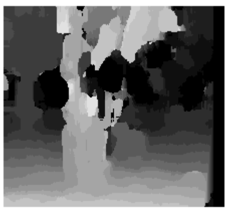
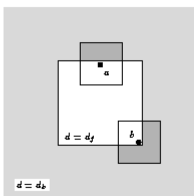
$$E(x,y;d) = \sum_{(x',y') \in N(x,y)} [I_L(x'+d,y') - I_R(x',y')]^2$$

- How big should the neighborhood be?


Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Neighborhood size

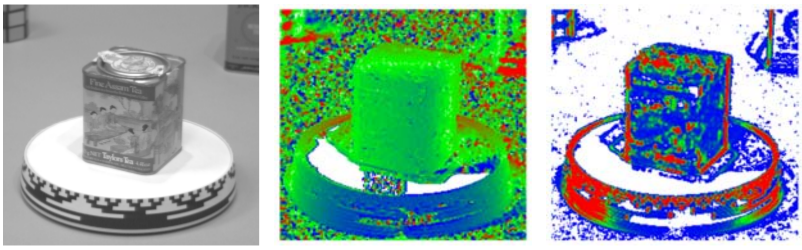
- Smaller neighborhood: more details
- Larger neighborhood: fewer isolated mistakes

W 



Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Stereo: certainty modeling

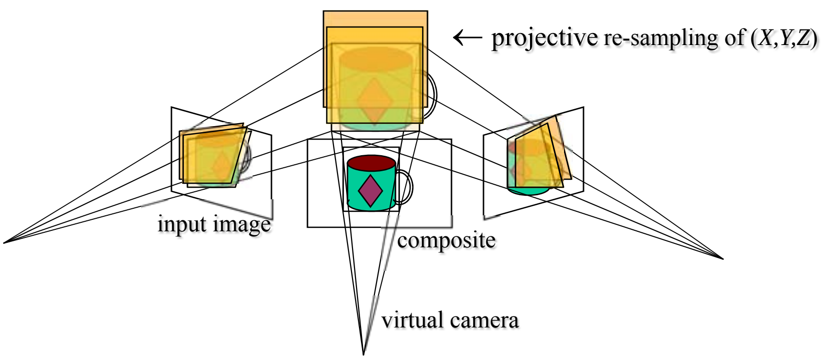
- Compute certainty map from correlations



input depth map certainty map

Plane Sweep Stereo

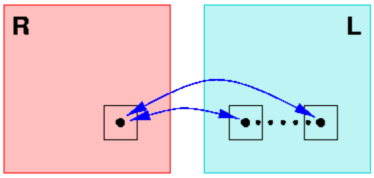
- Sweep family of planes through volume

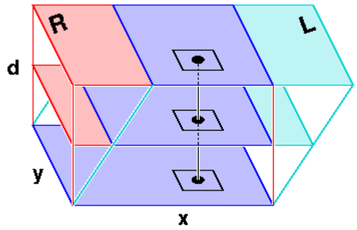


- each plane defines an image ⇒ composite homography

Plane sweep stereo

- Re-order (pixel / disparity) evaluation loops





for every pixel,
  for every disparity
    compute cost

for every disparity
  for every pixel
compute cost

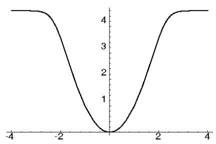
Stereo matching framework

- For every disparity, compute raw matching costs

Why use a robust function?
  – occlusions, other outliers

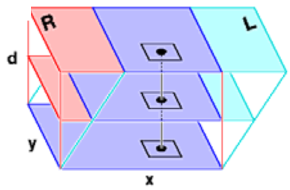
$$E_0(x, y; d) = \rho(I_L(x' + d, y') - I_R(x', y'))$$

- Can also use alternative match criteria



Stereo matching framework

- Aggregate costs spatially
- Here, $E(x, y; d) = \sum_{(x',y') \in N(x,y)} E_0(x', y', d)$ (efficient moving average implementation)
- Can also use weighted average, [non-linear] diffusion…

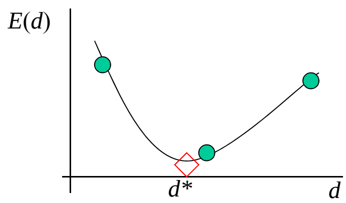


Stereo matching framework

- Choose winning disparity at each pixel

$$d(x, y) = \arg\min_d E(x, y; d)$$

- Interpolate to sub-pixel accuracy



Traditional Stereo Matching

- Advantages:
  - gives detailed surface estimates
  - fast algorithms based on moving averages
  - sub-pixel disparity estimates and confidence

- Limitations:
  - narrow baseline $\Rightarrow$ noisy estimates
  - fails in textureless areas
  - gets confused near occlusion boundaries

Stereo with Non-Linear Diffusion

- Problem with traditional approach:
  - gets confused near discontinuities
- New approach:
  - use iterative (non-linear) aggregation to obtain better estimate
  - provably equivalent to mean-field estimate of Markov Random Field

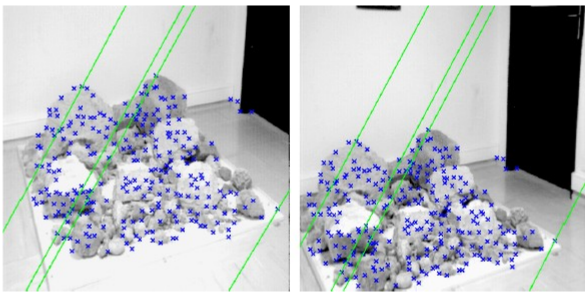
How to get Matching Points? **Features**

- ~~Colour~~
- Corners
- Edges
- Lines
- Statistics on Edges: SIFT, SURF, ORB…

In OpenCV: The following detector types are supported:
- "FAST" – FastFeatureDetector
- "STAR" – StarFeatureDetector
- "SIFT" – SIFT (nonfree module)
- "SURF" – SURF (nonfree module)
- "ORB" – ORB
- "BRISK" – BRISK
- "MSER" – MSER
- "GFTT" – GoodFeaturesToTrackDetector
- "HARRIS" – GoodFeaturesToTrackDetector with Harris detector enabled
- "Dense" – DenseFeatureDetector
- "SimpleBlob" – SimpleBlobDetector

Feature-based stereo

- Match "corner" (interest) points



- Interpolate complete solution

Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Features -- Colour Features

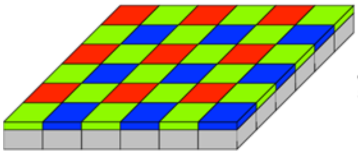
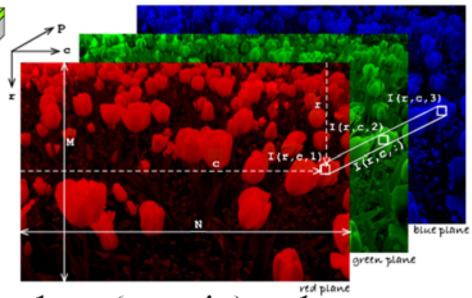


Fig: Ch. 10, *Robotics Vision and Control*

Bayer Patterns



- RGB is **NOT** an absolute (metric) colour space

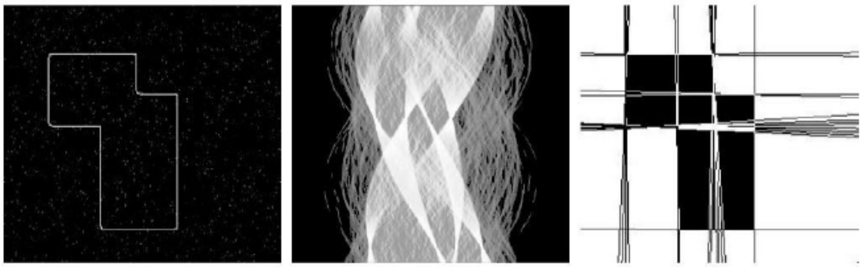
Also!

- **RGB** (display or additive colour) does not map to **CYMK** (printing or subtractive colour) without calibration
- **Y-Cr-Cb** or **HSV** does not solve this either

How to get the Features? Still MANY Ways

- Canny edge detector:

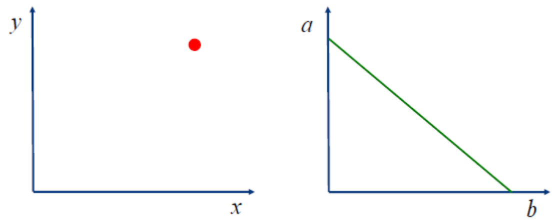
Hough Transform



- Uses a voting mechanism
- Can be used for other lines and shapes (not just straight lines)

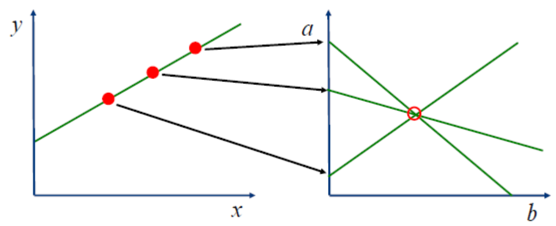
Hough Transform: Voting Space

$$y = ax + b \qquad\qquad a = -\frac{1}{x}b + \frac{y}{x}$$



- Count the number of lines that can go through a point and move it from the "*x-y*" plane to the "a-b" plane
- There is only a one-"infinite" number (a line!) of solutions (not a two-"infinite" set – a plane)

Hough Transform: Voting Space



- In practice, the polar form is often used

$$a = x \cos a + y \sin b$$

- This avoids problems with lines that are nearly vertical

Hough Transform: Algorithm

1. Quantize the parameter space appropriately.

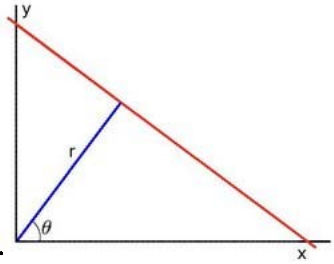
2. Assume that each cell in the parameter space is an accumulator. Initialize all cells to zero.

3. For each point (x,y) in the (visual & range) image space, increment by 1 each of the accumulators that satisfy the equation.

4. Maxima in the accumulator array correspond to the parameters of model instances.

Line Detection – Hough Lines [1]

- A line in an image can be expressed as two variables:
  - **Cartesian coordinate system:** m,b
  - **Polar coordinate system:** r, θ
    - ➔ **avoids problems with vert. lines**



y=**m**x+**b** ➔

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

- For each point (x_1, y_1) we can write:

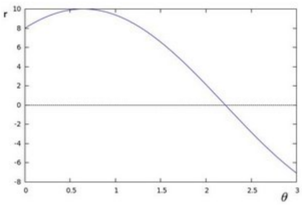
$$r = x_1 \cos\theta + y_1 \sin\theta$$

- Each pair (r,θ) represents a line that passes through (x_1, y_1)

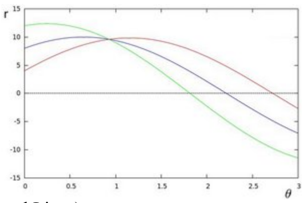
See also OpenCV documentation (cv::HoughLines)

Line Detection – Hough Lines [2]

- Thus a given point gives a sinusoid



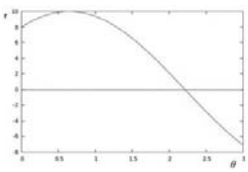
- Repeating for all points on the image



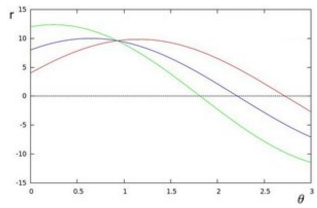
See also OpenCV documentation (cv::HoughLines)

Line Detection – Hough Lines [3]

- Thus a given point gives a sinusoid



- Repeating for all points on the image

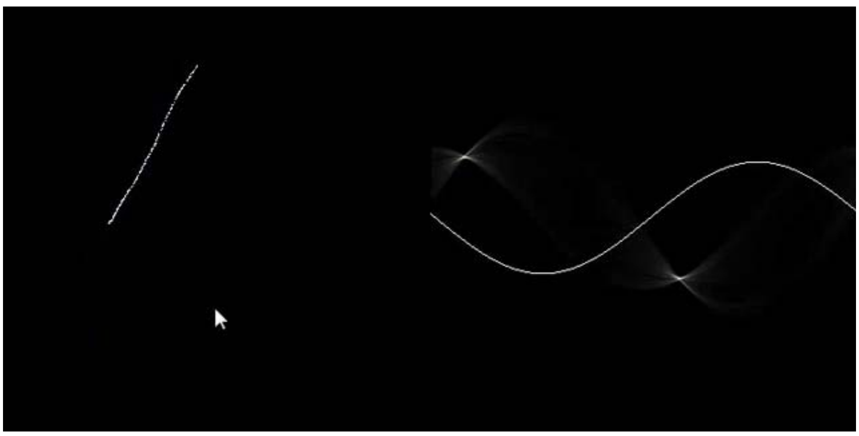


- NOTE that an intersection of sinusoids represents (**a point**) represents **a line** in which pixel points lay.

➔ Thus, a line can be *detected* by finding the number of Intersections between curves

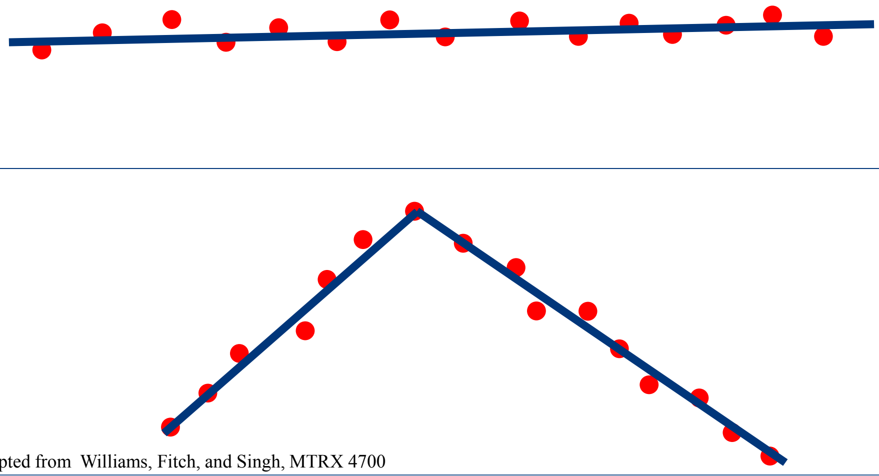
See also OpenCV documentation (cv::HoughLines)

"Cool Robotics Share" -- Hough Transform



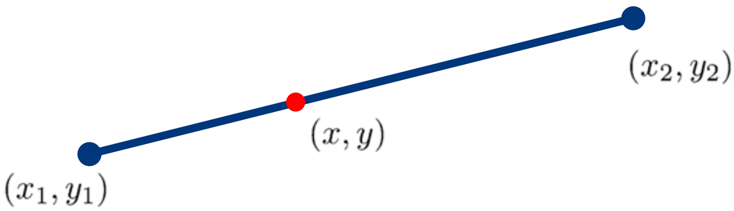
- http://www.activovision.com/octavi/doku.php?id=hough_transform

Line Extraction and Segmentation



Adopted from Williams, Fitch, and Singh, MTRX 4700

Line Formula

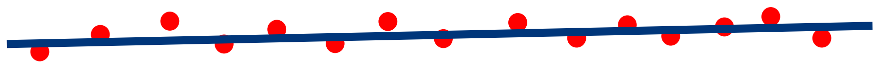


$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\mathbf{y} = m\mathbf{x} + b$$

Adopted from Williams, Fitch, and Singh, MTRX 4700

Line Estimation

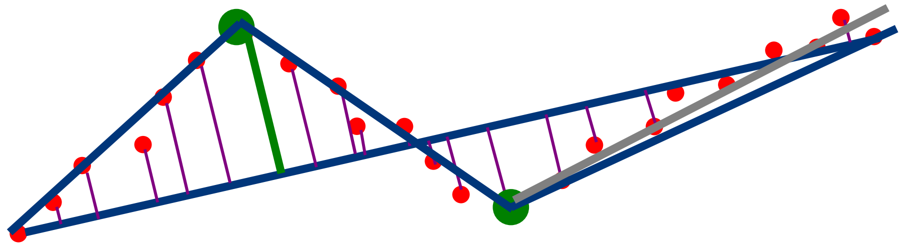


Least squares minimization of the line:

- Line Equation: $\mathbf{y} - m\mathbf{x} - b = 0$
- Error in Fit: $\sum_i (y_i - mx_i - b)^2$
- Solution: $\begin{pmatrix} \overline{xy} \\ \bar{y} \end{pmatrix} = \begin{pmatrix} \overline{x^2} & \bar{x} \\ \bar{x} & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix}$

Adopted from Williams, Fitch, and Singh, MTRX 4700

Line Splitting / Segmentation



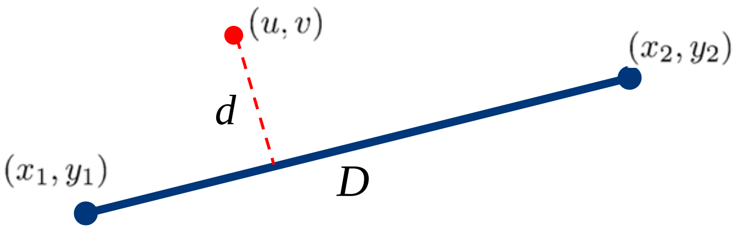
- What about corners?

➔Split into multiple lines (via expectation maximization)

1. Expect (assume) a number of lines N (say 3)
2. Find "breakpoints" by finding nearest neighbours upto a threshold or simply at random (RANSAC)
3. How to know N? (Also RANSAC)

Adopted from Williams, Fitch, and Singh, MTRX 4700

⊥ of a Point from a Line Segment



$$r = u(y_1 - y_2) + v(x_2 - x_1) + y_2 x_1 - y_1 x_2$$

$$d = \frac{r}{D}$$

Adopted from Williams, Fitch, and Singh, MTRX 4700

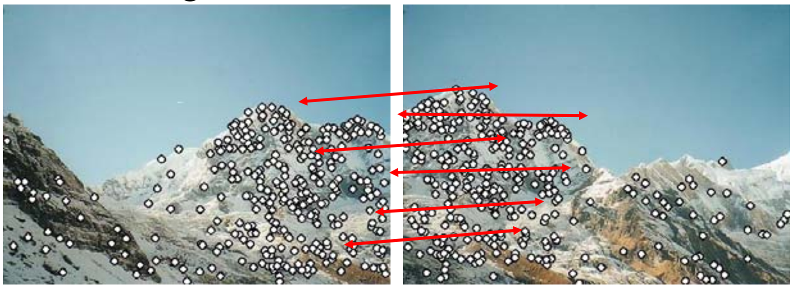
Edge Detection

- Canny edge detector:
  - Pepsi Sequence:

Image Data: http://www.cs.brown.edu/~black/mixtureOF.html and Szeliski, CS223B-L9
See also: Use of Temporal information to aid segmentation:
http://www.cs.toronto.edu/~babalex/SpatiotemporalClosure/supplementary_material.html

Why extract features?

- **Object detection**
- Robot Navigation
- Scene Recognition



- Steps:
  - Extract Features
  - Match Features

Adopted drom S. Lazebnik, Gang Hua (CS 558)

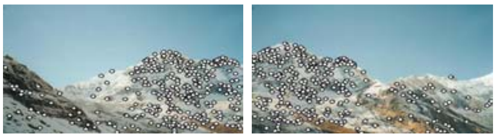
Why extract features? [2]

- Panorama stitching…
  - → Step 3: Align images

Adopted from S. Lazebnik, Gang Hua (CS 558)

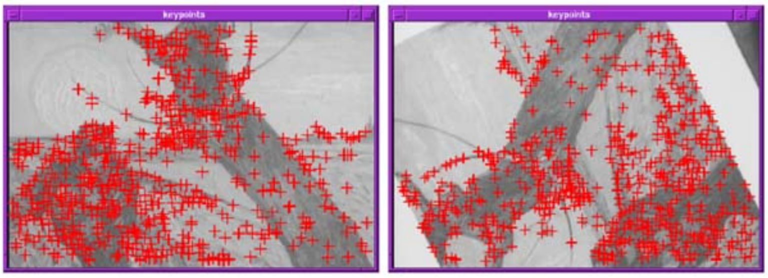
Characteristics of good features

- Repeatability
  - The same feature can be found in several images despite geometric and photometric transformations
- Saliency
  - Each feature is distinctive
- Compactness and efficiency
  - Many fewer features than image pixels
- Locality
  - A feature occupies a relatively small area of the image; robust to clutter and occlusion



Adopted from S. Lazebnik, Gang Hua (CS 558)

Finding Corners



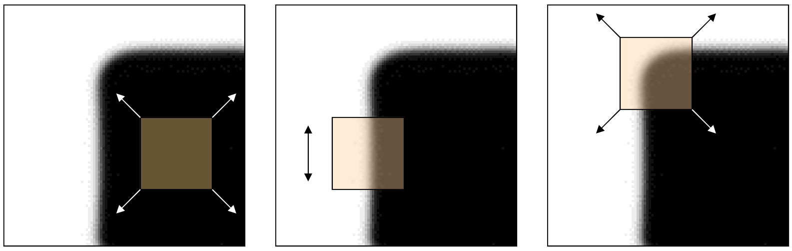
- Key property: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and distinctive

C.Harris and M.Stephens. "A Combined Corner and Edge Detector.“ *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Adopted from S. Lazebnik, Gang Hua (CS 558)

Corner Detection: Basic Idea

- Look through a window
- Shifting a window in any direction should give a large change in intensity



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

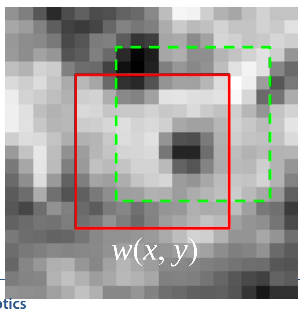
Source: A. Efros

Corner Detection: Mathematics

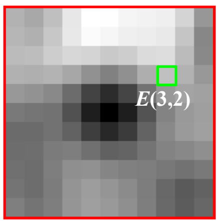
Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

$I(x, y)$



$E(u, v)$



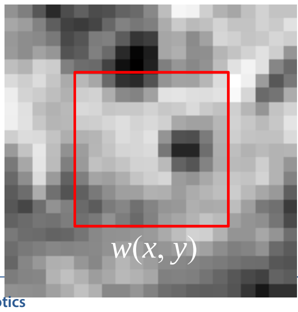
Adopted from S. Lazebnik, Gang Hua (CS 558)

Corner Detection: Mathematics

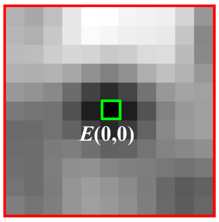
Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

$I(x, y)$



$E(u, v)$



Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Corner Detection: Mathematics

Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

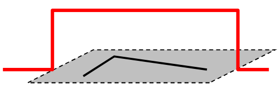
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

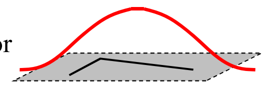
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Source: R. Szeliski

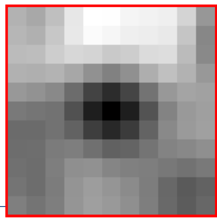
Corner Detection: Mathematics

Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

We want to find out how this function behaves for small shifts

$E(u, v)$



Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Corner Detection: Mathematics

Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

We want to find out how this function behaves for small shifts

$$E(u,v) \approx E(0,0) + [u \;\; v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u \;\; v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

Local quadratic approximation of $E(u,v)$ in the neighborhood of (0,0) is given by the *second-order Taylor expansion*:

Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Corner Detection: Mathematics

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

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx E(0,0) + [u \;\; v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u \;\; v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

$$E_u(u,v) = \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right]I_x(x+u,y+v)$$

$$E_{uu}(u,v) = \sum_{x,y} 2w(x,y)I_x(x+u,y+v)I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right]I_{xx}(x+u,y+v)$$

$$E_{uv}(u,v) = \sum_{x,y} 2w(x,y)I_y(x+u,y+v)I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right]I_{xy}(x+u,y+v)$$

Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Corner Detection: Mathematics

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

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx [u \;\; v]\begin{bmatrix} \sum_{x,y} w(x,y)I_x^2(x,y) & \sum_{x,y} w(x,y)I_x(x,y)I_y(x,y) \\ \sum_{x,y} w(x,y)I_x(x,y)I_y(x,y) & \sum_{x,y} w(x,y)I_y^2(x,y) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(0,0) = 0$$

$$E_u(0,0) = 0$$

$$E_v(0,0) = 0$$

$$E_{uu}(0,0) = \sum_{x,y} 2w(x,y)I_x(x,y)I_x(x,y)$$

$$E_{vv}(0,0) = \sum_{x,y} 2w(x,y)I_y(x,y)I_y(x,y)$$

$$E_{uv}(0,0) = \sum_{x,y} 2w(x,y)I_x(x,y)I_y(x,y)$$

Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Harris detector: Steps

- Compute Gaussian derivatives at each pixel
- Compute second moment matrix M in a Gaussian window around each pixel
- Compute corner response function R
- Threshold R
- Find local maxima of response function (nonmaximum suppression)

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

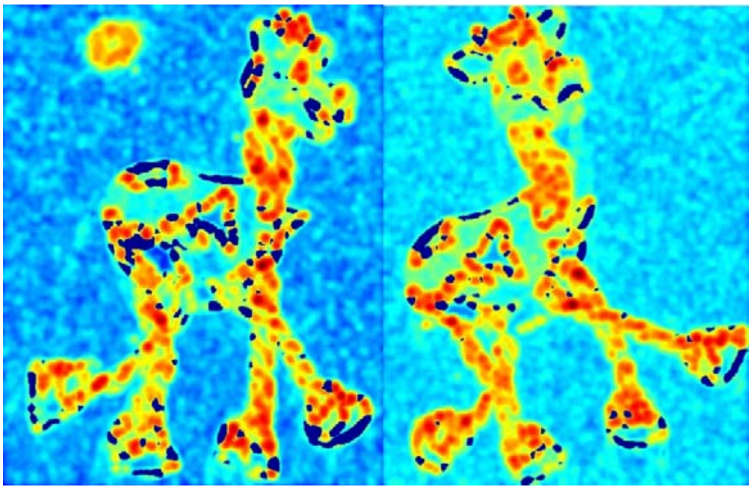
Adopted from
S. Lazebnik,
Gang Hua (CS 558)

Harris Detector: Steps

Adopted from S. Lazebnik, Gang Hua (CS 558)

Harris Detector: Steps

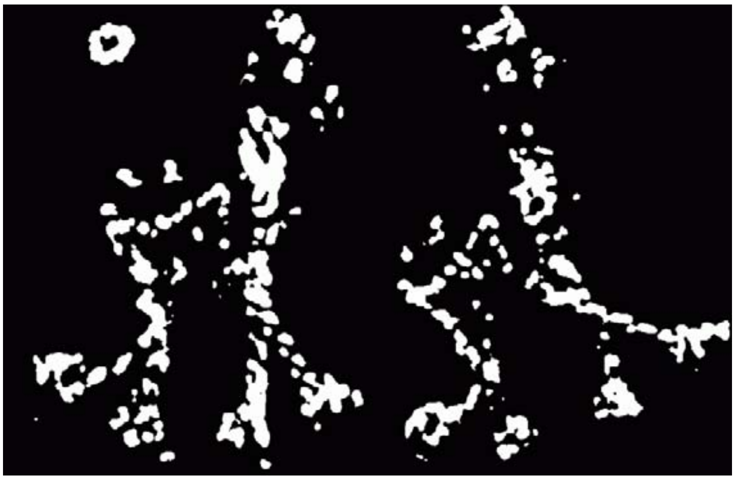
Compute corner response R



Adopted from S. Lazebnik, Gang Hua (CS 558)

Harris Detector: Steps

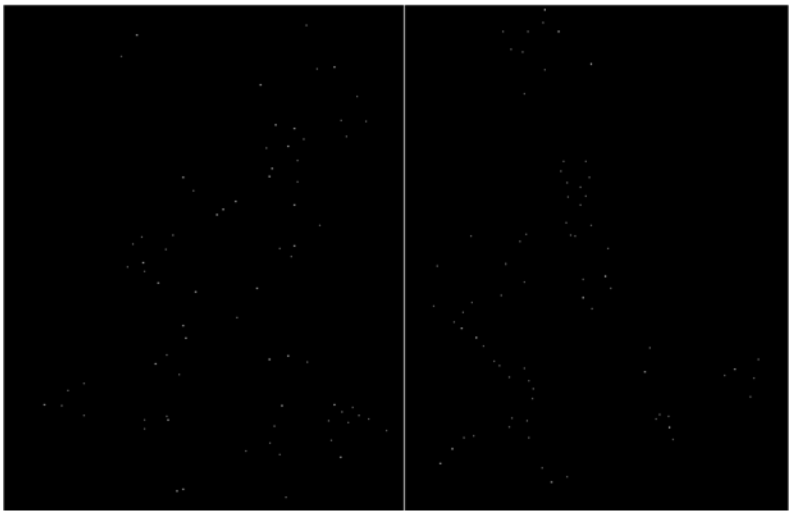
Find points with large corner response: R>threshold



Adopted from S. Lazebnik, Gang Hua (CS 558)

Harris Detector: Steps

Take only the points of local maxima of *R*



Adopted from S. Lazebnik, Gang Hua (CS 558)

Harris Detector: Steps

Adopted from S. Lazebnik, Gang Hua (CS 558)

Invariance and covariance

- We want corner locations to be invariant to photometric transformations and covariant to geometric transformations
  - Invariance: image is transformed and corner locations do not change
  - Covariance: if we have two transformed versions of the same image, features should be detected in corresponding locations

Adopted from S. Lazebnik, Gang Hua (CS 558)

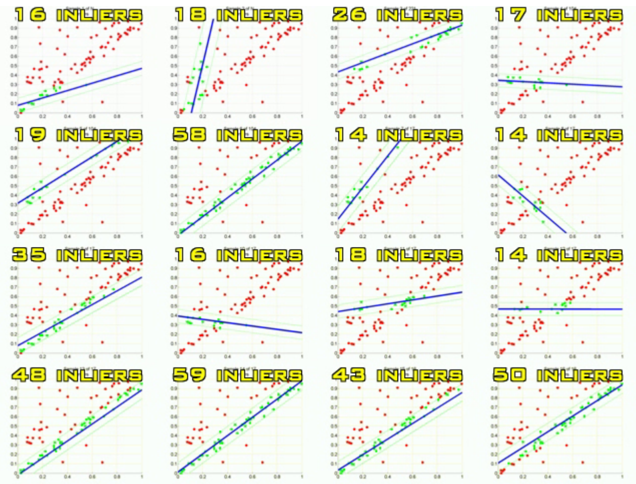
RANdom SAmple Consensus

1. Repeatedly select a small (minimal) subset of correspondences
2. Estimate a solution (in this case a the line)
3. Count the number of "inliers", $|e|<\Theta$ (for LMS, estimate med($|e|$))
4. Pick the *best* subset of inliers
5. Find a complete least-squares solution

- Related to least median squares
- See also: MAPSAC (Maximum *A Posteriori* SAmple Consensus)

From Szeliski, *Computer Vision: Algorithms and Applications*

Cool Robotics Share Time!

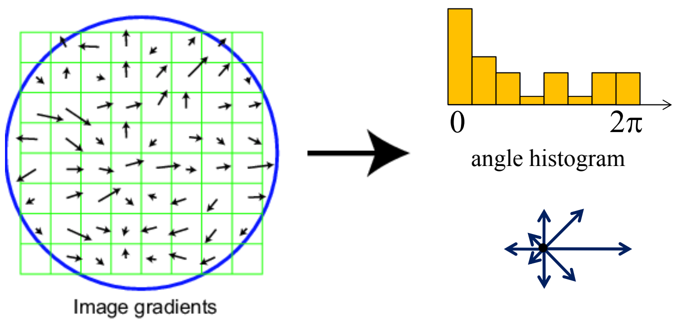


D. Wedge, *The RANSAC Song*

Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

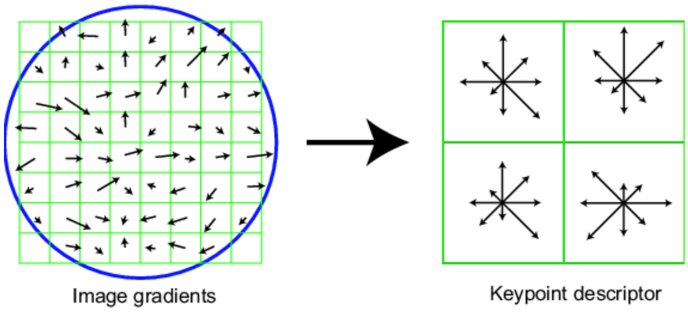


Adapted from slide by David Lowe

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Adapted from slide by David Lowe

Properties of SIFT

- Extraordinarily robust matching technique
  - Can handle changes in viewpoint
    - Up to about 60 degree out of plane rotation
  - Can handle significant changes in illumination
    - Sometimes even day vs. night (below)
  - Fast and efficient—can run in real time
  - Lots of code available
    - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

From David Lowe and Szeliski, *Computer Vision: Algorithms and Applications*

Feature matching

- Given a feature in I_1, how to find the best match in I_2?
  1. Define distance function that compares two descriptors
  2. Test all the features in I_2, find the one with min distance

From Szeliski, *Computer Vision: Algorithms and Applications*

Feature distance

- How to define the difference between two features f_1, f_2?
  - Simple approach is SSD(f_1, f_2)
    - sum of square differences between entries of the two descriptors
    - can give good scores to very ambiguous (bad) matches

I_1 I_2

From Szeliski, *Computer Vision: Algorithms and Applications*

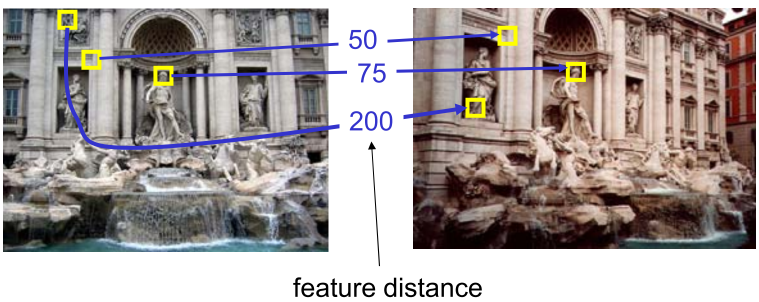
Feature distance

- How to define the difference between two features f_1, f_2?
  - Better approach: ratio distance = SSD(f_1, f_2) / SSD(f_1, f_2')
    - f_2 is best SSD match to f_1 in I_2
    - f_2' is 2^{nd} best SSD match to f_1 in I_2
    - gives small values for ambiguous matches

From Szeliski, *Computer Vision: Algorithms and Applications*

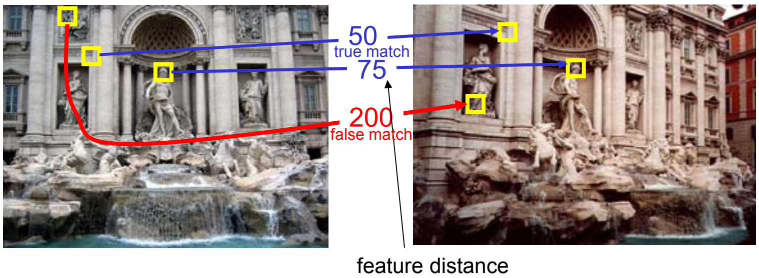
Evaluating the results

- How can we measure the performance of a feature matcher?



From Szeliski, *Computer Vision: Algorithms and Applications*

True/false positives



- The distance threshold affects performance
  - True positives = # of detected matches that are correct
    - Suppose we want to maximize these—how to choose threshold?
  - False positives = # of detected matches that are incorrect
    - Suppose we want to minimize these—how to choose threshold?

From Szeliski, *Computer Vision: Algorithms and Applications*

Levenberg-Marquardt

- Iterative non-linear least squares [Press'92]
  - Linearize measurement equations

$$\hat{u}_i = f(\mathbf{m}, \mathbf{x}_i) + \frac{\partial f}{\partial \mathbf{m}} \Delta \mathbf{m}$$
$$\hat{v}_i = g(\mathbf{m}, \mathbf{x}_i) + \frac{\partial g}{\partial \mathbf{m}} \Delta \mathbf{m}$$

  - Substitute into log-likelihood equation: quadratic cost function in Dm

$$\sum_i \sigma_i^{-2} (\hat{u}_i - u_i + \frac{\partial f}{\partial \mathbf{m}} \Delta \mathbf{m})^2 + \cdots$$

From Szeliski, *Computer Vision: Algorithms and Applications*

Levenberg-Marquardt

- What if it doesn't converge?
  - Multiply diagonal by (1 + l), increase l until it does
  - Halve the step size Dm (my favorite)
  - Use line search
  - Other ideas?
- Uncertainty analysis: covariance S = A-1
- Is maximum likelihood the best idea?
- How to start in vicinity of global minimum?

From Szeliski, *Computer Vision: Algorithms and Applications*

Camera matrix calibration

- Advantages:
  - very simple to formulate and solve
  - can recover K [R | t] from M using QR decomposition [Golub & VanLoan 96]
- Disadvantages:
  - doesn't compute internal parameters
  - more unknowns than true degrees of freedom
  - need a separate camera matrix for each new view

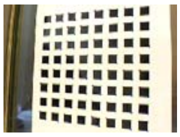
From Szeliski, *Computer Vision: Algorithms and Applications*

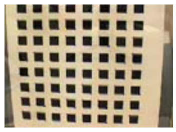
Multi-plane calibration

- Use several images of planar target held at unknown orientations [Zhang 99]
  - Compute plane homographies

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \sim \mathbf{K} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \mathbf{HX}$$

  - Solve for K-TK-1 from Hk's
    - 1plane if only f unknown
    - 2 planes if (f,uc,vc) unknown
    - 3+ planes for full K
  - Code available from Zhang and OpenCV





From Szeliski, *Computer Vision: Algorithms and Applications*

Rotational motion

- Use pure rotation (large scene) to estimate f
  - estimate f from pairwise homographies
  - re-estimate f from 360º "gap"
  - optimize over all {K,Rj} parameters
    [Stein 95; Hartley '97; Shum & Szeliski '00; Kang & Weiss '99]

- Most accurate way to get f, short of surveying distant points

From Szeliski, *Computer Vision: Algorithms and Applications*

SFM: Structure from Motion
(& Cool Robotics Share (this week))

Structure [from] Motion

- Given a set of feature tracks,
  estimate the 3D structure and 3D (camera) motion.
- Assumption: orthographic projection
- Tracks: (u_{fp},v_{fp}), f: frame, p: point
- Subtract out **mean** 2D position…

  $\mathbf{i}_f$: rotation, $\mathbf{s}_p$: position

$$u_{fp} = i_f^T s_p, v_{fp} = j_f^T s_p$$

From Szeliski, *Computer Vision: Algorithms and Applications*

Structure from motion

- How many points do we need to match?
- 2 frames:
  - (R,t): 5 dof + 3n point locations ≤
  - 4n point measurements ⇒
  - n ≥ 5

$$\hat{u}_{ij} = f(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i)$$
$$\hat{v}_{ij} = g(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i)$$

- k frames:
  - 6(k–1)-1 + 3n ≤ 2kn
- always want to use many more

From Szeliski, *Computer Vision: Algorithms and Applications*

Measurement equations

- Measurement equations

$$u_{fp} = \boldsymbol{i}_f^T \boldsymbol{s}_p \qquad \boldsymbol{i}_f\text{: rotation, } \boldsymbol{s}_p\text{: position}$$
$$v_{fp} = \boldsymbol{j}_f^T \boldsymbol{s}_p$$

- Stack them up…

$$\boldsymbol{W} = \boldsymbol{R}\,\boldsymbol{S}$$
$$\boldsymbol{R} = (\boldsymbol{i}_1,\ldots,\boldsymbol{i}_F, \boldsymbol{j}_1,\ldots,\boldsymbol{j}_F)^T$$
$$\boldsymbol{S} = (\boldsymbol{s}_1,\ldots,\boldsymbol{s}_P)$$

From Szeliski, *Computer Vision: Algorithms and Applications*

Factorization

$$\boldsymbol{W} = \boldsymbol{R}_{2F\times 3}\,\boldsymbol{S}_{3\times P}$$

SVD

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V} \quad \boldsymbol{\Lambda} \textbf{ must be rank 3}$$

$$\boldsymbol{W'} = (\boldsymbol{U}\boldsymbol{\Lambda}^{1/2})(\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}) \quad = \boldsymbol{U'}\boldsymbol{V'}$$

Make ***R*** orthogonal

$$\boldsymbol{R} = \boldsymbol{QU'},\ \boldsymbol{S} = \boldsymbol{Q}^{-1}\boldsymbol{V'}$$

$$\boldsymbol{i}_f^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{i}_f = 1 \ \ldots$$

From Szeliski, *Computer Vision: Algorithms and Applications*

Results

- Look at paper figures…

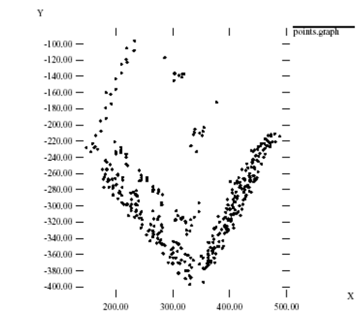


Figure 4.5: A view of the computed shape from approximately above the building (compare with figure 4.6).

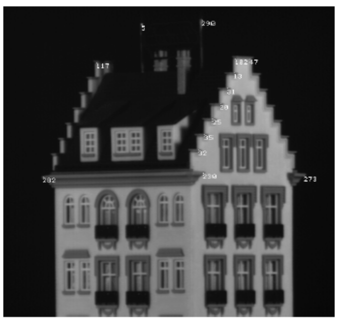


Figure 4.7: For a quantitative evaluation, distances between the features shown in the picture were measured on the actual model, and compared with the computed results. The comparison is shown in figure 4.8.

From Szeliski, *Computer Vision: Algorithms and Applications*

Bundle Adjustment

- What makes this non-linear minimization hard?
  - many more parameters: potentially slow
  - poorer conditioning (high correlation)
  - potentially lots of outliers
  - gauge (coordinate) freedom

$$\begin{aligned} \hat{u}_{ij} &= f(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i) \\ \hat{v}_{ij} &= g(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i) \end{aligned}$$

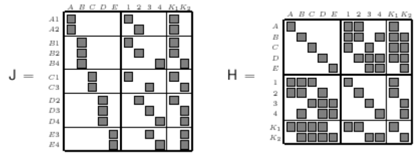
From Szeliski, *Computer Vision: Algorithms and Applications*

Lots of parameters: sparsity

- Only a few entries in Jacobian are non-zero

$$\begin{aligned} \hat{u}_{ij} &= f(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i) \\ \hat{v}_{ij} &= g(\mathrm{K}, \mathrm{R}_j, \mathrm{t}_j, \mathrm{x}_i) \end{aligned}$$

$$\frac{\partial \hat{u}_{ij}}{\partial \mathrm{K}}, \quad \frac{\partial \hat{u}_{ij}}{\partial \mathrm{R}_j}, \quad \frac{\partial \hat{u}_{ij}}{\partial \mathrm{t}_j}, \quad \frac{\partial \hat{u}_{ij}}{\partial \mathrm{x}_i},$$

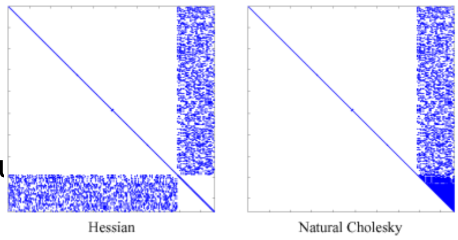


From Szeliski, *Computer Vision: Algorithms and Applications*

Sparse Cholesky (skyline)

- First used in finite element analysis
- Applied to SfM by [Szeliski & Kang 1994]

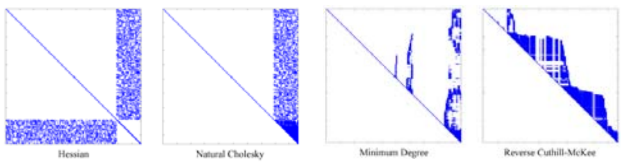
structu



From Szeliski, *Computer Vision: Algorithms and Applications*

Conditioning and gauge freedom

- Poor conditioning:
  - use 2nd order method
  - use Cholesky decomposition



- Gauge freedom
  - fix certain parameters (orientation) or
  - zero out last few rows in Cholesky decomposition

From Szeliski, *Computer Vision: Algorithms and Applications*

More Cool Robotics Share!

Cool Robotics Share (IV)

Source: Youtube: Wired, How the Tesla Model S is Made