Perception & Vision

METR 4202: Advanced Control & Robotics

Drs Surya Singh, Paul Pounds, and Hanna Kurniawati

Lecture # 7 September 3, 2012

(with Vision Material from Richard Szeliski, *Computer Vision: Algorithms and Applications*)

metr4202@itee.uq.edu.au

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

RoboticsCourseWare Contributor

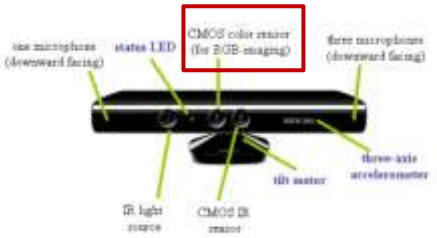
NEW Schedule

Week	Date	Lecture (M: 12-1:30, 43-102)
1	23-Jul	Introduction
2	30-Jul	Representing Position & Orientation & State (Frames, Transformation Matrices & Affine Transformations)
3	6-Aug	Robot Kinematics and Dynamics
4	13-Aug	Robot Dynamics & Control
5	20-Aug	Obstacle Avoidance & Motion Planning
6	27-Aug	Sensors, Measurement and Perception
7	3-Sep	Perception & Vision-based control (+ Steve LaValle)
8	10-Sep	State-space Modelling
9	17-Sep	Controller Design & Observers (+ Mandyam Srinivasan)
	24-Sep	*Study break*
10	1-Oct	*Public Holiday*
11	8-Oct	Control/Planning Under Uncertainty
12	15-Oct	Localization and Navigation
13	22-Oct	Guest Lecture (Peter Corke) & Course Review

Updates: Lab 2 ➔ Free Extension to Oct 2

For Lab 2:

- Group reshuffling is allowed
  - Jared Page will determine new groups at this Friday's lab
- Uses a Kinect mostly as a "Camera"



- Free "extension" to **Tuesday, October 2 at 10am**
  - It is still "due" Sept 21 (Friday at 5 pm)*
  - But no late penalty if it is turned by Oct 2

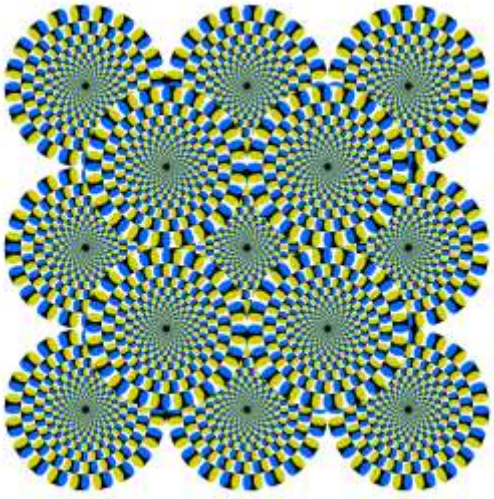
Fig. from Ch. 6 of Davison, *Kinect Open Source Programming Secrets*

Quick Outline

1. **Perception ➔ Camera Sensors**
   1. Image Formation
      ➔ "Computational Photography"
   2. Calibration
   3. Feature extraction
   4. Stereopsis and depth
   5. Optical flow
2. **Kinematics Lab Recap & Stochastic Planning**
   ➔ Guest Lecture (from Norway) by Steve LaValle

Perception

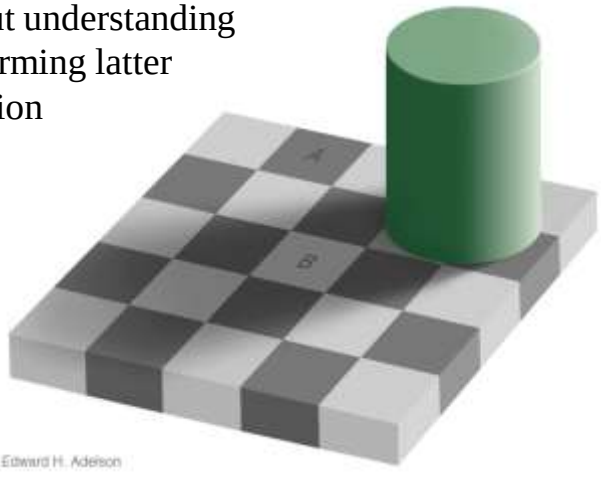
- Making Sense from Sensors



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

Perception

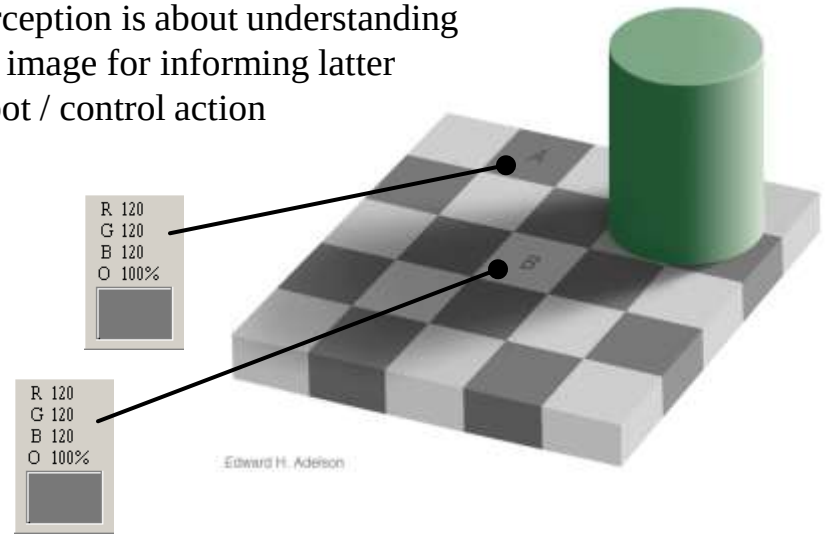
- Perception is about understanding the image for informing latter robot / control action



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

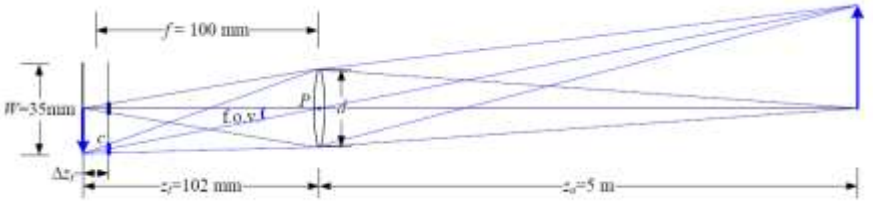
Perception

- 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



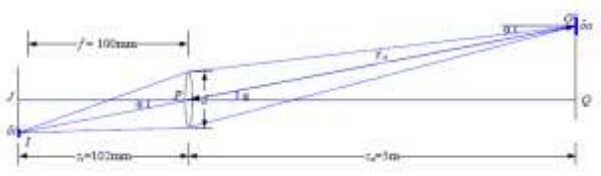
$$\frac{1}{z_0} + \frac{1}{z_1} = \frac{1}{f}$$

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*

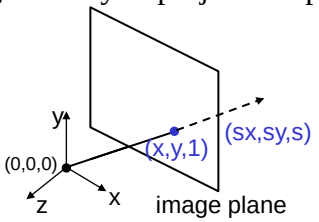
Image Formation:
Lens Optics (Aperture / Depth of Field)

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

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

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

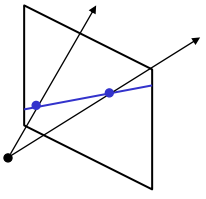


- Each *point* (x,y) on the plane is represented by a *ray* (sx,sy,s)
  - all points on the ray are equivalent: (x, y, 1) ≅ (sx, sy, s)

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

$$\text{in vector notation:} \quad 0 = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

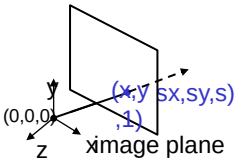
$$\mathbf{l}^T \quad \mathbf{p}$$

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

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

Planar Projective Transformations

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



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

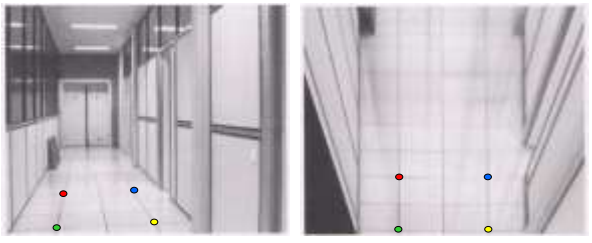
$\mathbf{p'}$ $\quad$ $\mathbf{H}$ $\quad$ $\mathbf{p}$

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*

Image Rectification



To unwarp (rectify) an image

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

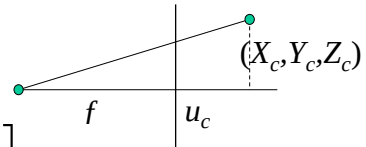
Slide from Szeliski, *Computer Vision: Algorithms and Applications*

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



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

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

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 $\rightarrow \mathbf{A}$

$$\mathbf{\Pi} = \begin{bmatrix} s_x & 0 & -t_x \\ 0 & s_y & -t_y \\ 0 & 0 & 1/f \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}$$

intrinsics projection orientation position

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

3D Transformations

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{3\times4}$	3	orientation + ⋯	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{3\times4}$	6	lengths + ⋯	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{3\times4}$	7	angles + ⋯	
affine	$\begin{bmatrix} A \end{bmatrix}_{3\times4}$	12	parallelism + ⋯	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{4\times4}$	15	straight lines	

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

Projection Models

- Orthographic

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

- Weak Perspective

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

- Affine

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

- Perspective

$$\Pi = [\mathbf{R} \quad \mathbf{t}]$$

- Projective

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

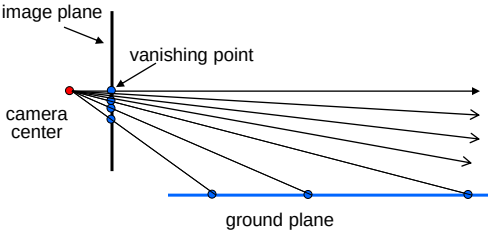
Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Properties of Projection

- Preserves
  - Lines and conics
  - Incidence
  - Invariants (cross-ratio)
- Does not preserve
  - Lengths
  - Angles
  - Parallelism

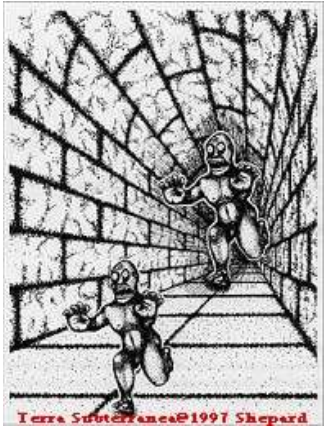
Vanishing Points

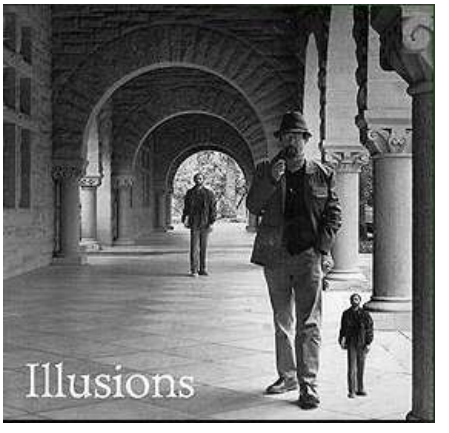
- Vanishing point
  - projection of a point at infinity
  - whiteboard capture, architecture,…



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

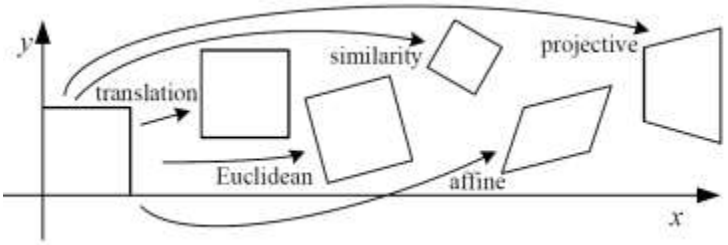
Fun With Vanishing Points





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

Transformations



- ***x'*: New Image & ***x*** : Old Image**
- Euclidean:
  (Distances preserved)

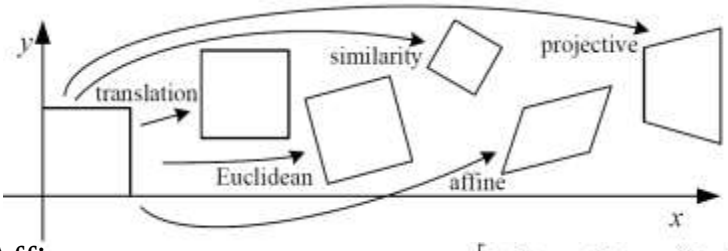
$$x' = \begin{bmatrix} R & t \end{bmatrix} \underline{x}$$

- 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

Transformations [3]

- 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 $g(x')$.

- **Inverse Warp**

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*

Calibration

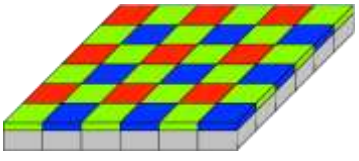
See: *Camera Calibration Toolbox for Matlab*
(http://www.vision.caltech.edu/bouguetj/calib_doc/)

- 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

Features

- Colour
- Corners
- Edges
- Lines
- Statistics on Edges: SIFT, SURF

Features -- Colour Features



Bayer Patterns

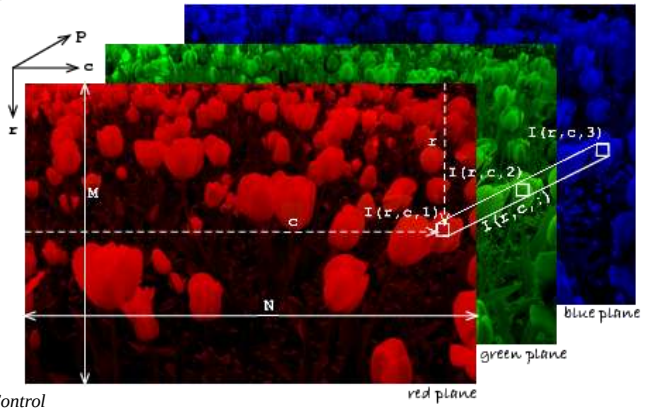


Fig: Ch. 10, *Robotics Vision and Control*

Edge Detection

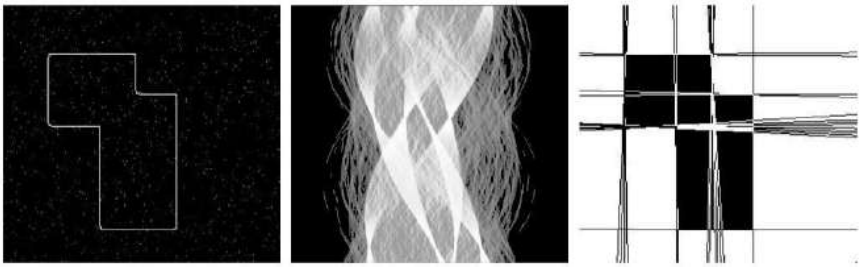
- Canny edge detector:

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

Edge Detection

- Canny edge detector:

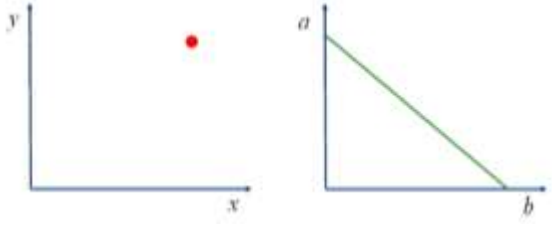
Hough Transform



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

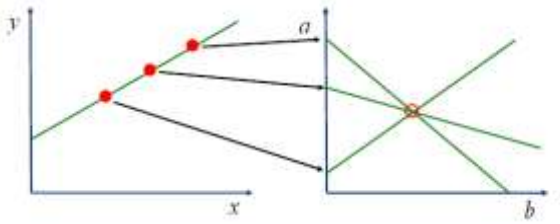
Hough Transform: Voting Space

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



- 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.

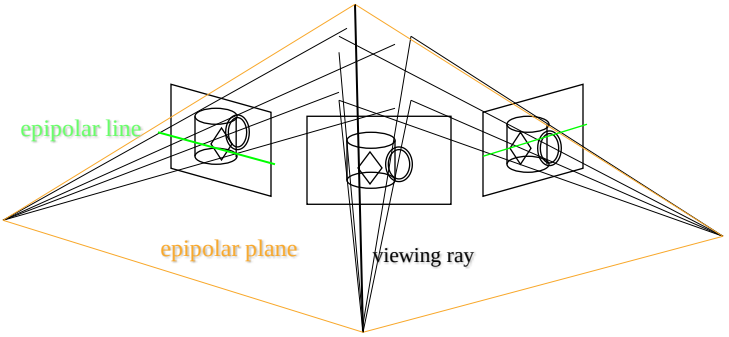
2. Assume that each cell in the parameter space is an accumulator. Initialize all cells to zero.

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.

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*

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*

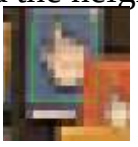
Image registration (revisited)

- 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?

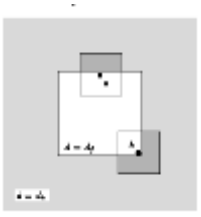



Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Neighborhood size

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

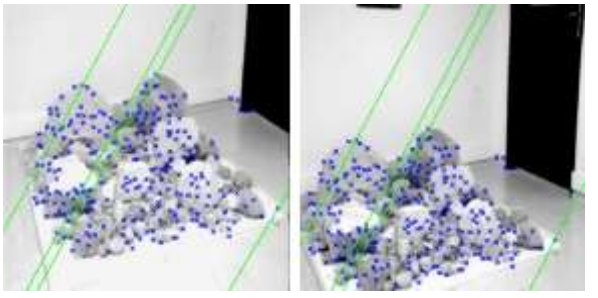




Slide from Szeliski, *Computer Vision: Algorithms and Applications*

Feature-based stereo

- Match “corner” (interest) points



- Interpolate complete solution

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

Cool Robotics Share

D. Wedge, *The Fundamental Matrix Song*