Robot Sensing
Perception & Linear Observers

METR 4202: **Robotics** & Automation

Dr Surya Singh -- Lecture # 6

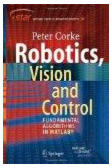
August 31, 2016

metr4202@itee.uq.edu.au
http://robotics.itee.uq.edu.au/~metr4202/ [http://**metr4202.com**]

Schedule of Events

Week	Date	Lecture (W: 12:05-1:50, 50-N202)
1	27-Jul	Introduction
2	3-Aug	Representing Position & Orientation & State (Frames, Transformation Matrices & Affine Transformations)
3	10-Aug	Robot Kinematics Review (*& Ekka Day*)
4	17-Aug	Robot Inverse Kinematics & Kinetics
5	24-Aug	Robot Dynamics (Jacobeans)
6	**31-Aug**	**Robot Sensing: Perception & Linear Observers**
7	7-Sep	Robot Sensing: Multiple View Geometry & Feature Detection
8	14-Sep	Probabilistic Robotics: Localization
9	21-Sep	Probabilistic Robotics: SLAM
	28-Sep	*Study break*
10	5-Oct	Motion Planning
11	12-Oct	State-Space Modelling
12	19-Oct	Shaping the Dynamic Response
13	26-Oct	LQR + Course Review

Follow Along Reading:



Robotics, Vision & Control
by Peter Corke

Also online:SpringerLink

UQ Library eBook:
364220144X

Today

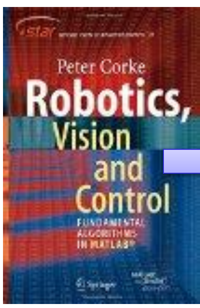
➔ Sensing and Vision 🡐

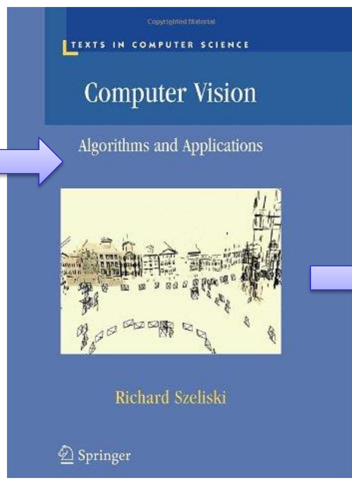
- Vision
  - Chapter 11: Image Formation
  - Chapter 14: Using Multiple Images
    § 14.2 Geometry of Multiple Views

- Multiple View Geometry
  - Hartley & Zisserman:
    Chapter 6: Camera Models
    Chapter 7: Camera Matrix (P)

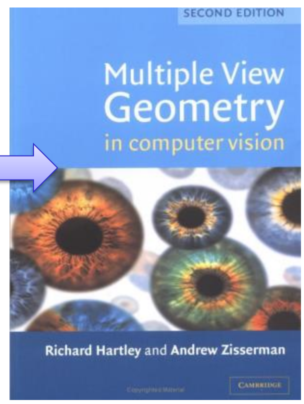
Next Time

Reference Material





UQ Library/ SpringerLink



UQ Library
(ePDF)

Robot Dynamics (leftovers!)

Static Forces

- We can also use the Jacobian to compute the joint torques required to maintain a particular force at the end effector
- Consider the concept of virtual work

$$F \cdot \delta \mathbf{X} = \tau \cdot \delta \theta$$

- Or

$$F^T \delta \mathbf{X} = \tau^T \delta \theta$$

- Earlier we saw that

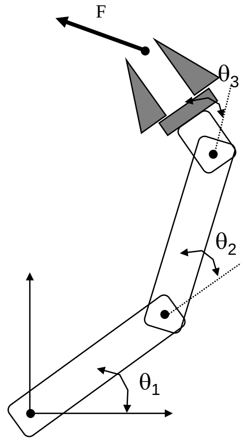
$$\delta \mathbf{X} = \mathbf{J} \delta \theta$$

- So that

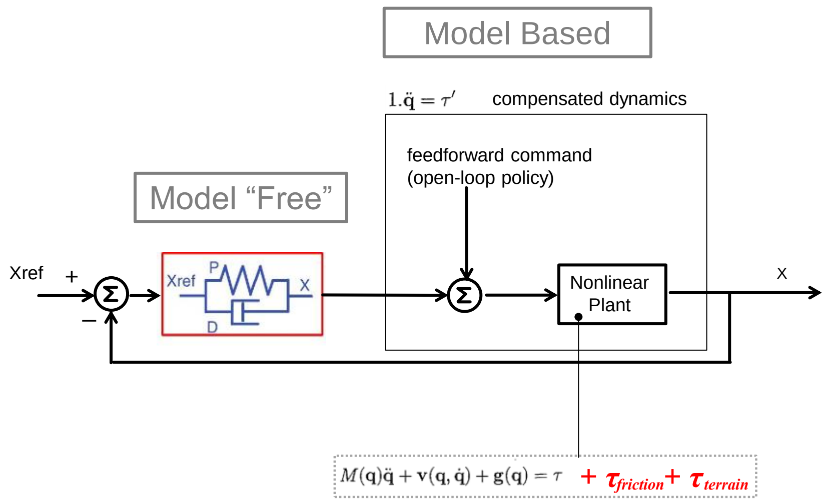
$$F^T \mathbf{J} = \tau^T$$

- Or

$$\tau = \mathbf{J}^T F$$



Operation Space (Computed Torque)



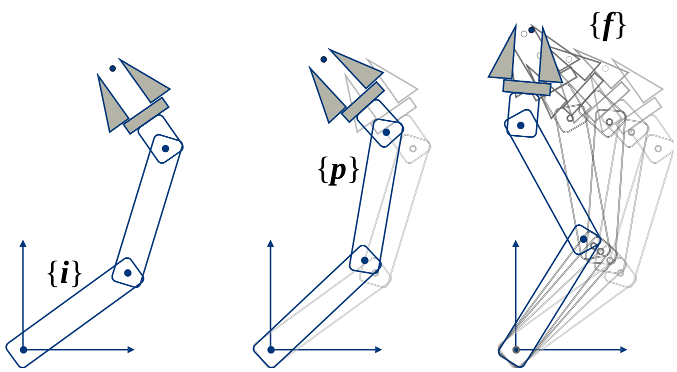
Compensated Manipulation

Dynamics of Parallel Manipulators

- Traditional Newton-Euler formulation:
  - Equations of motion to be written once for each body of a manipulator
  - Large number of equations
- Lagrangian formulation
  - eliminates all of the unwanted reaction forces and moments at the outset.
  - It is more efficient than the Newton- Euler formulation
  - Numerous constraints imposed by closed loops of a parallel manipulator
- To simplify the problem
  - Lagrangian Multipliers are often introduced
  - Principle of virtual work

Trajectory Generation

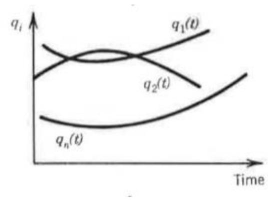
- The goal is to get from an initial position $\{\boldsymbol{i}\}$ to a final position $\{\boldsymbol{f}\}$ via a path points $\{\boldsymbol{p}\}$



Joint Space

Consider only the **joint positions** as a function of time

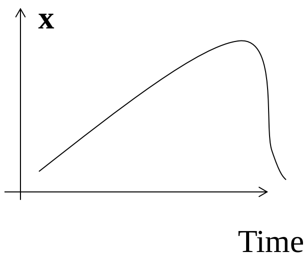
- + Since we control the joints, this is more direct
- -- If we want to follow a particular trajectory, not easy
  - at best lots of intermediate points
  - No guarantee that you can solve the Inverse Kinematics for all path points



Cartesian Workspace

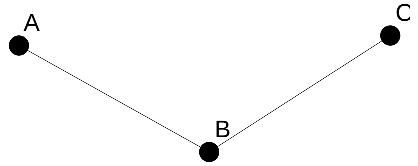
Consider the **Cartesian positions** as a function of time

- + Can track shapes exactly
- -- We need to solve the inverse kinematics and dynamics



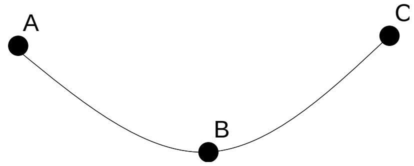
Polynomial Trajectories

- Straight line Trajectories



- Simpler

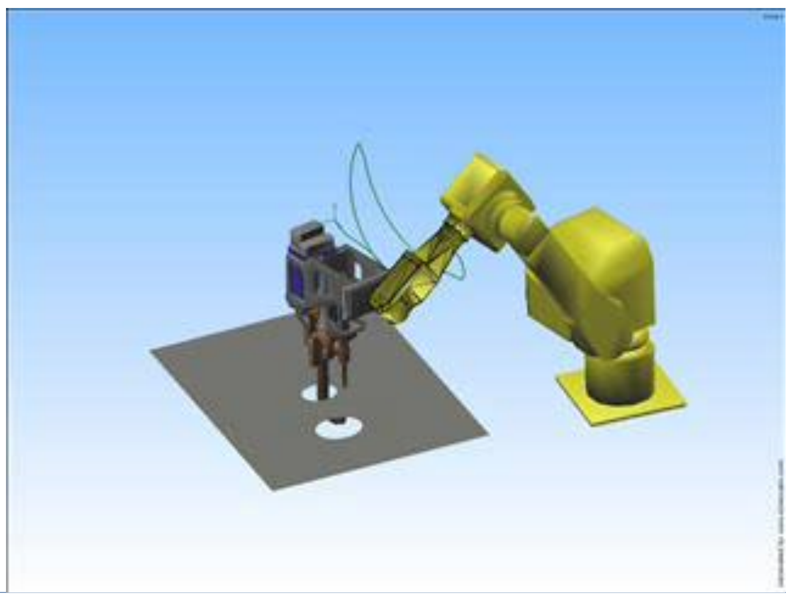
- Polynomial Trajectories



$$u(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

- Parabolic blends are smoother
- Use "pseudo via points"

Trajectory Generation & Planning



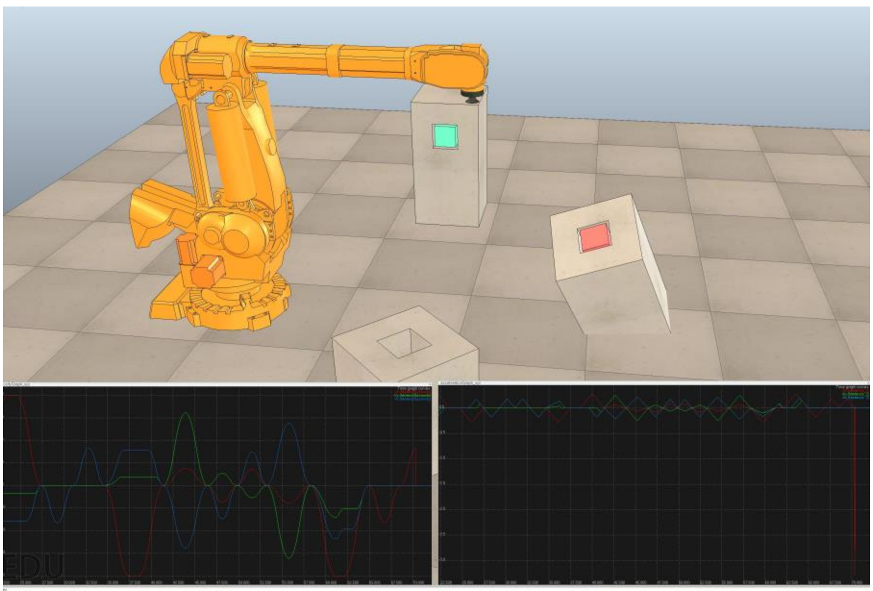
Dynamic Simulation Software

- v-rep virtual robot experimentation platform
- Reflexxes

http://www.coppeliarobotics.com/

http://www.reflexxes.com/

Trajectory Generation & Planning



Sensing:
Image Formation / Single-View Geometry

Quick Outline

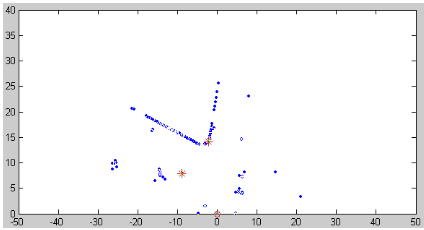
- Frames
- Kinematics

➔ "Sensing Frames" (in space) ➔ Geometry in Vision

1. <u>Perception ➔ Camera Sensors</u>

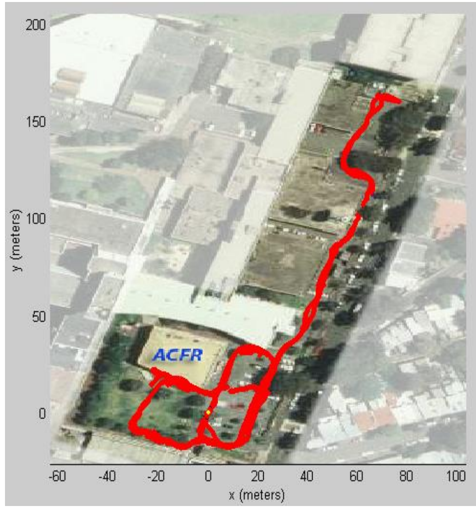
1. Image Formation
   ➔ "Computational Photography"
2. Calibration
3. Features
4. Stereopsis and depth
5. Optical flow

Sensor Information: Mostly **(but not only)** Cameras!



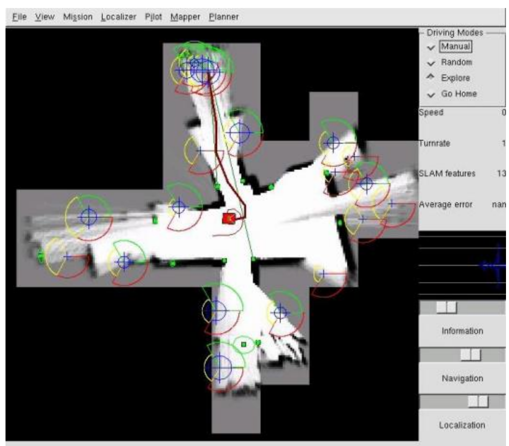
Laser

Vision/Cameras



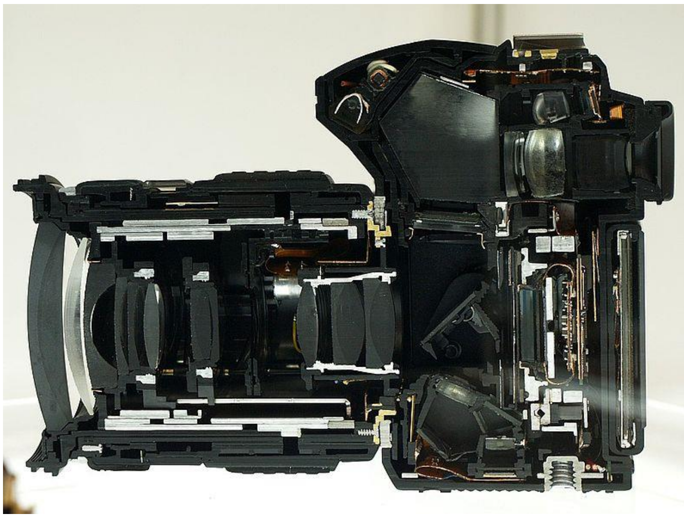
GPS

Mapping: Indoor robots



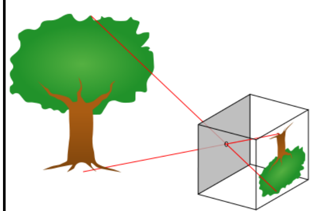
ACFR, IROS 2002

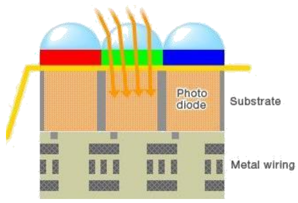
Cameras



Wikipedia, E-30-Cutmodel

Cameras: A 3D ⇒ 2D Photon Counting Sensor*





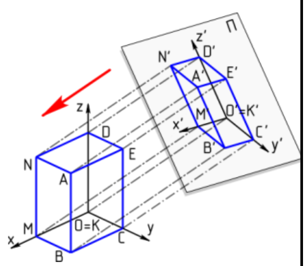
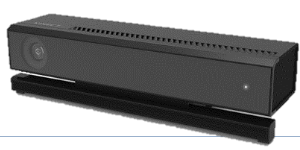
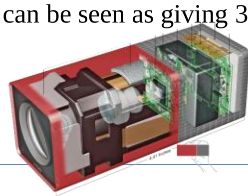


Image Formation ⇒ Image Sensing ⇒ (Re)Projection

Sources: Wikipedia, Pinhole Camera, Sony sensor, Wikipedia, Graphical projection,

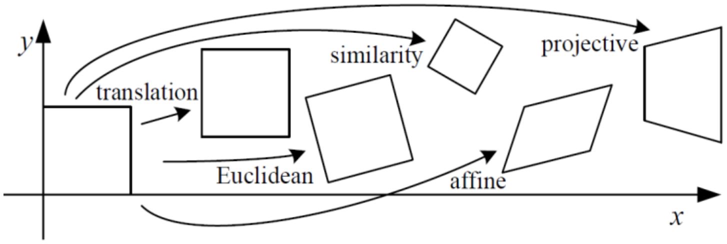
* Well Almost… RGB-D and Light-Field cameras can be seen as giving 3D ⇒ 3D





Camera Image Formation: 3D↦2D ⇒ **Perspective!**

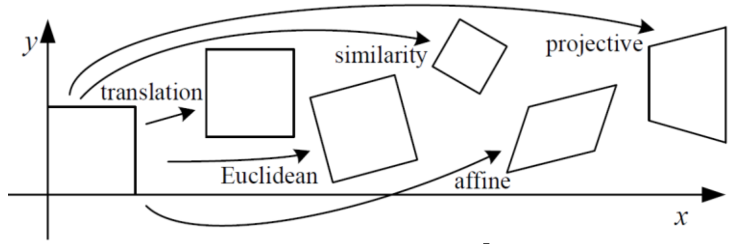
Transformations



- ***x'*: New Image & <u>**x**</u> : 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

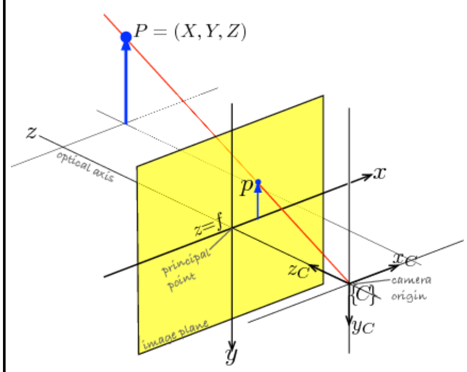
➔ x = point in the old image

• 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

Image Formation – Single View Geometry



$$\frac{Y}{Z} = \frac{y}{f} \qquad \frac{X}{Z} = \frac{x}{f}$$

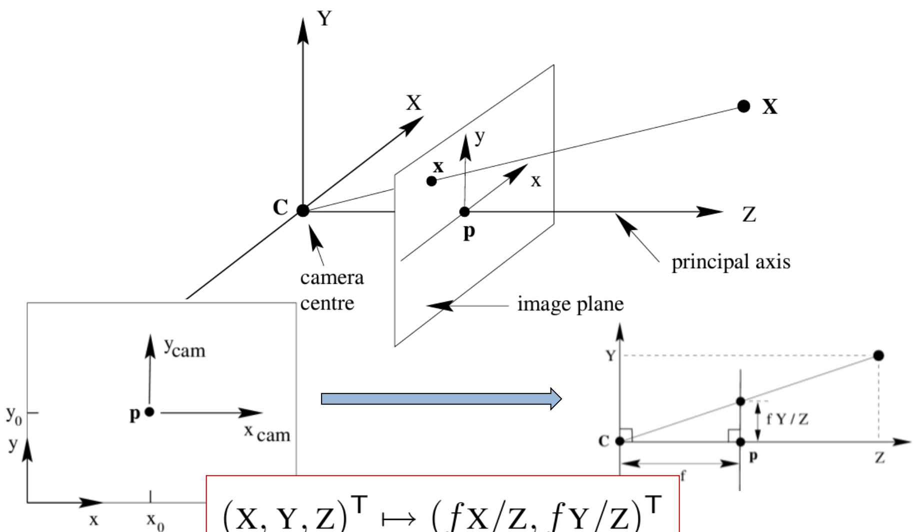
$$x = \frac{fX}{Z}, y = \frac{fY}{Z}$$

$$(X, Y, Z) \mapsto (x, y)$$

$$\mathbb{R}^3 \mapsto \mathbb{R}^2$$

Image and Slide from: Corke, Ch. 11

Image Formation – Single View Geometry [I]



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

Hartly & Zisserman, Ch. 6

Image Formation – Single View Geometry [II]

➔ Camera Projection Matrix

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

world camera Intrinsics

- x = Image point
- $\mathbf{X}$ = World point
- K = Camera Calibration Matrix

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

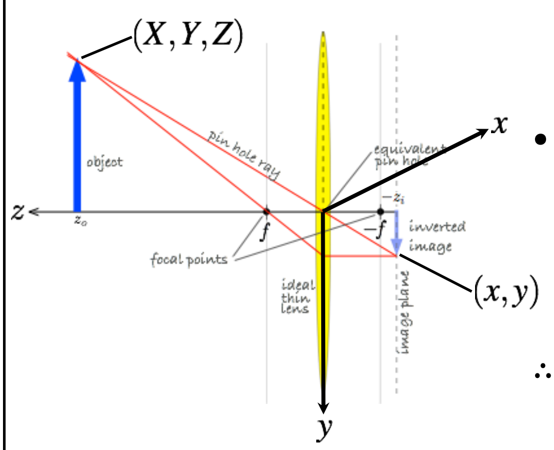
$$\mathbf{x} = K[I \mid \mathbf{0}]\mathbf{X}_{\text{cam}}.$$

➔Perspective Camera as:

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

where: P is **3**×4 and of **rank 3**

Image Formation: (Thin-Lens) Projection model

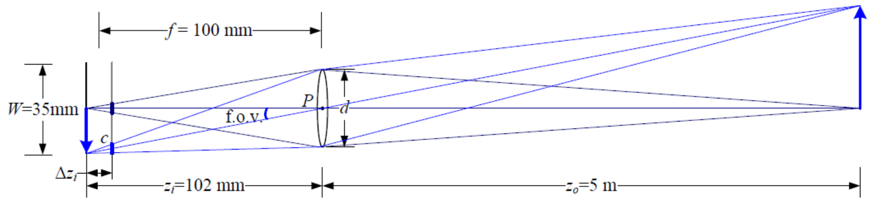


- $x = \frac{fX}{Z}, y = \frac{fY}{Z}$
- $\frac{1}{z_0} + \frac{1}{z_1} = \frac{1}{f}$

$\therefore$ as $z_0 \to \infty$, $z_i \to f$

Image and Slide from: Corke, Ch. 11

Image Formation: Simple Lens Optics ≅ Thin-Lens



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

Sec. 2.2 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

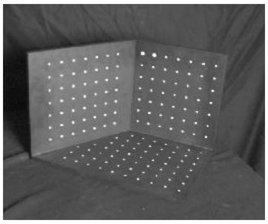
See: *Camera Calibration Toolbox for Matlab*
(http://www.vision.caltech.edu/bouguetj/calib_doc/)

- 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

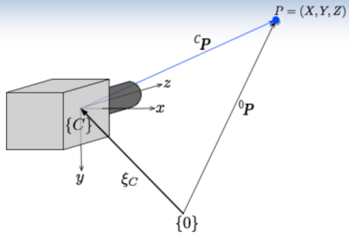
Camera calibration

- 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*

Complete camera model



$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

extrinsic parameters

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \underbrace{\underbrace{\begin{pmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{K}} \begin{pmatrix} \mathbf{R} & t \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}^{-1}}_{\mathbf{C}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

intrinsic parameters

camera matrix

Image and Slide from: Corke, Ch. 11

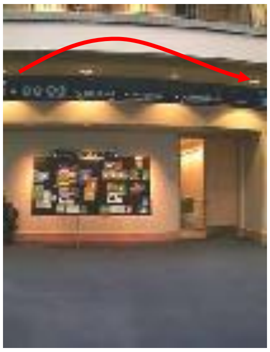
Camera Image Formation "Aberrations"[I]: Lens Optics (Aperture / Depth of Field)

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

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

Camera Image Formation "Aberrations"[II]: Lens Distortions

Barrel



Pincushion

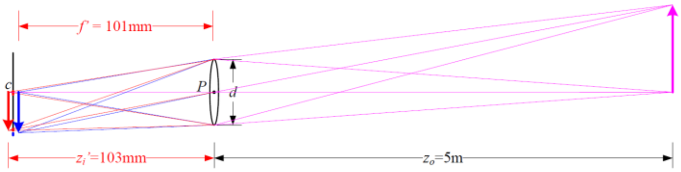
Fisheye

➔ Explore these with `visualize_distortions` in the [Camera Calibration Toolbox]

Fig. 2.1.3 from Szeliski, *Computer Vision: Algorithms and Applications*

Camera Image Formation "Aberrations" [II]: Lens Optics: Chromatic Aberration

- Chromatic Aberration:

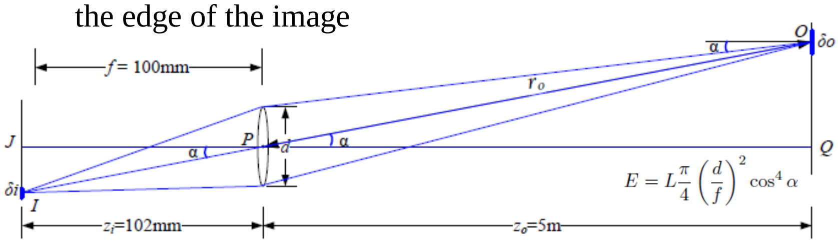


  - In a lens subject to chromatic aberration, light at different wavelengths (e.g., the red and blur arrows) is focused with a different focal length f' and hence a different depth z_i, resulting in both a geometric (in-plane) displacement and a loss of focus

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

Camera Image Formation “Aberrations” [III]: Lens Optics: Vignetting

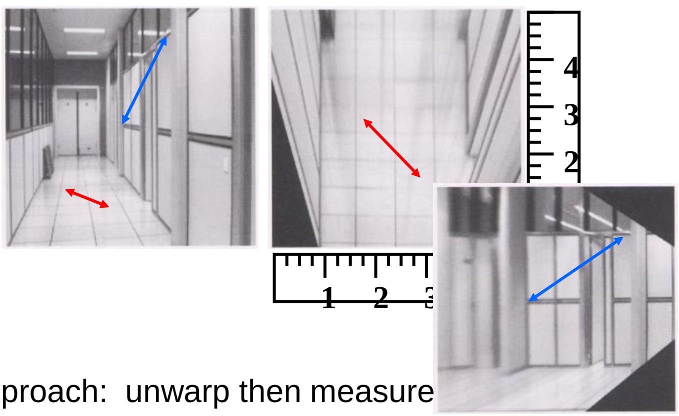
- Vignetting:
  - The tendency for the brightness of the image to fall off towards the edge of the image



  - The amount of light hitting a pixel of surface area δi depends on the square of the ratio of the aperture diameter d to the focal length f, as well as the fourth power of the off-axis angle α, $\cos^4 \alpha$

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

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

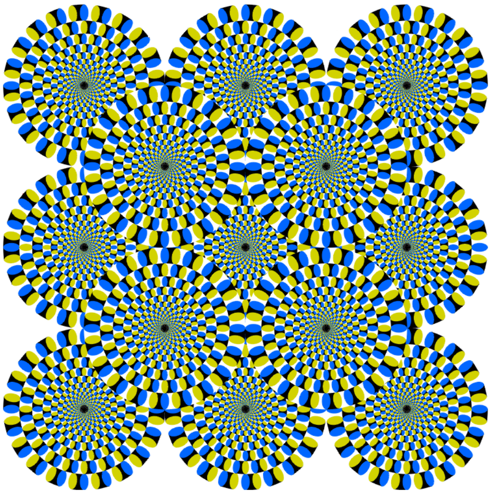


Approach: unwarp then measure

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

Perception

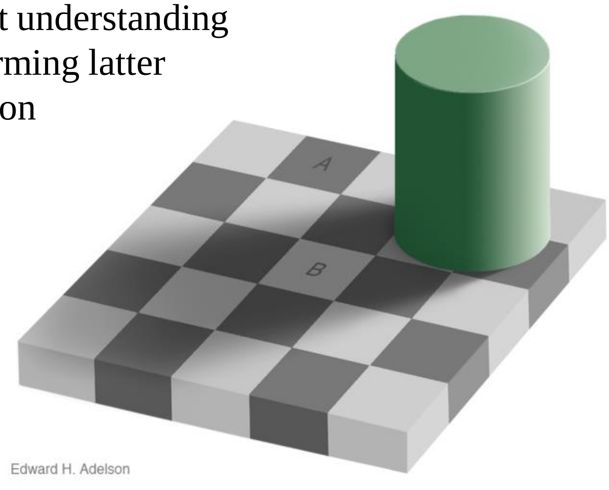
- Making Sense from Sensors



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

Perception

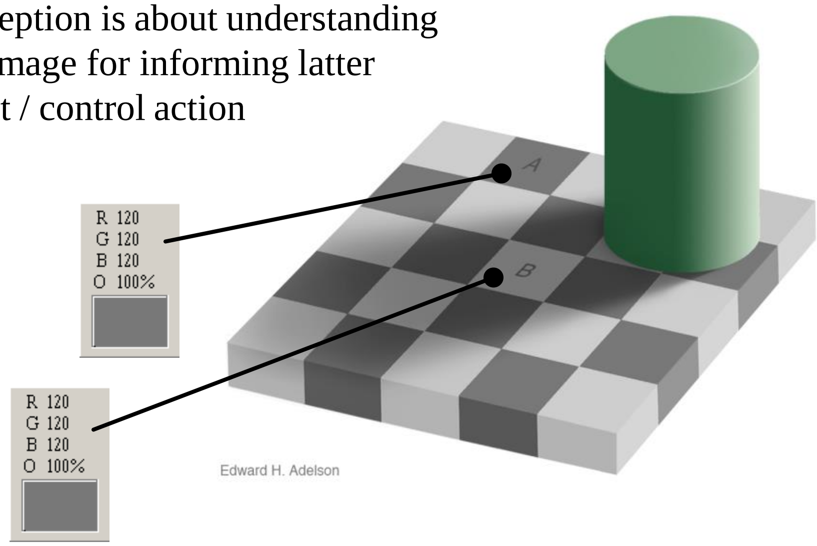
- Perception is about understanding the image for informing latter robot / control action



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

Basic Features:

~~Colour~~

Edges & Lines

Features -- Colour Features

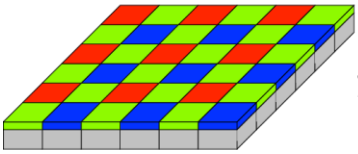
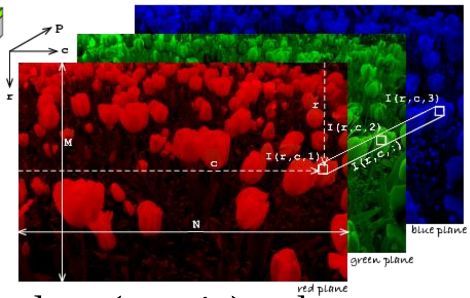


Fig: Ch. 10, *Robotics Vision and Control*



Bayer Patterns

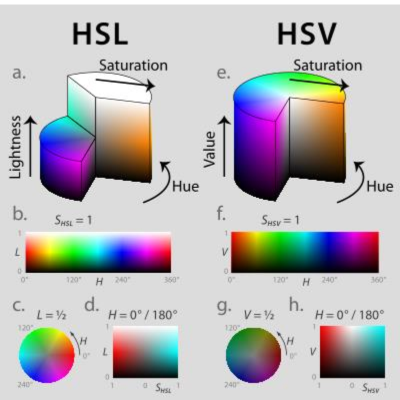
- RGB is **NOT** an absolute (metric) colour space

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

Colour Spaces

- HSV



Source: Wikipedia – HSV and YCrCb

- YCrCb

➔Gamma Corrected Luma (Y) + Chrominance

➔BW → Colour TVs : Just add the Chrominance

➔γ Correction: CRTs γ=2.2-2.5

$$\begin{aligned} Y' &= 16 + (65.481 \cdot R' + 128.553 \cdot G' + 24.966 \cdot B') \\ C_B &= 128 + (-37.797 \cdot R' - 74.203 \cdot G' + 112.0 \cdot B') \\ C_R &= 128 + (112.0 \cdot R' - 93.786 \cdot G' - 18.214 \cdot B') \end{aligned}$$

- L*ab

How to get the Features? **Still** MANY Ways

- Canny edge detector:

Subtractive (CMYK) & Uniform (L*ab) Color Spaces

- $C = W - R$
- $M = W - G$
- $Y = W - B$

- $K = -W$ ☺

- A Uniform color space is one in which the distance in coordinate space is a fair guide to the significance of the difference between the two colors

- Start with RGB → CIE XYZ (Under Illuminant D65)

$$L^\star = 116(Y/Y_n)^{(1/3)} - 16$$
$$a^\star = 500\left[(X/X_n)^{(1/3)} - (Y/Y_n)^{(1/3)}\right]$$
$$b^\star = 200\left[(Y/Y_n)^{(1/3)} - (Z/Z_n)^{(1/3)}\right]$$

Edge Detection

- Canny edge detector:
  - Pepsi Sequence:

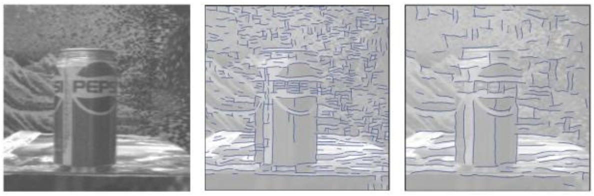
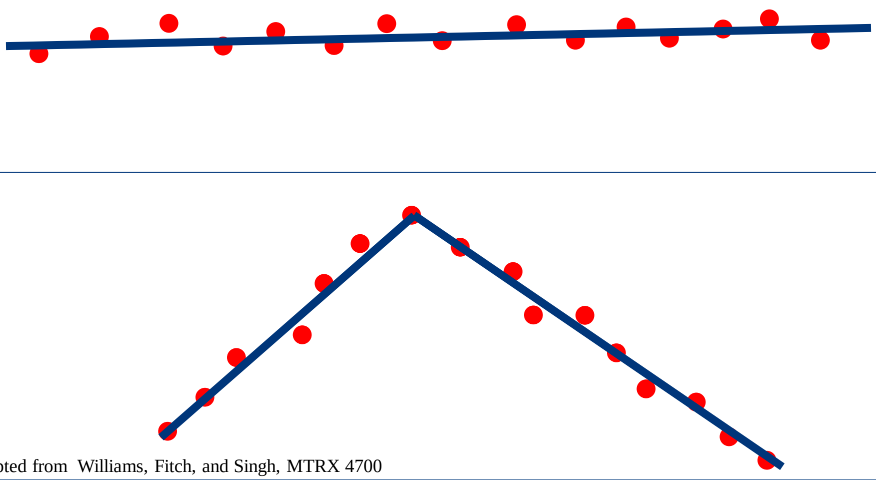


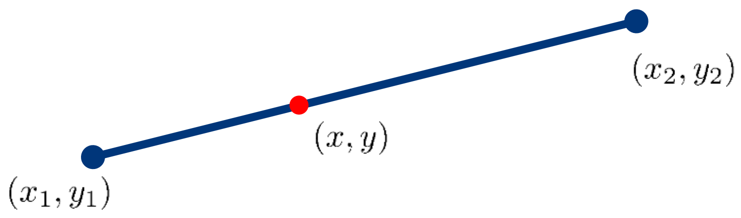
Image Data: http://www.cs.brown.edu/~black/mixtureOF.html and Szeliski, CS223B-L9
See also: Use of Temporal information to aid segmentation:
http://www.cs.toronto.edu/~babalex/SpatiotemporalClosure/supplementary_material.html

Line Extraction and Segmentation



Adopted from Williams, Fitch, and Singh, MTRX 4700

Line Formula

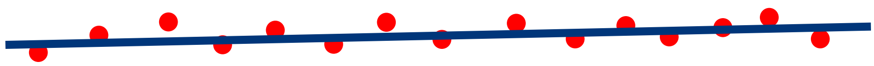


$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

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

Adopted from Williams, Fitch, and Singh, MTRX 4700

Line Estimation

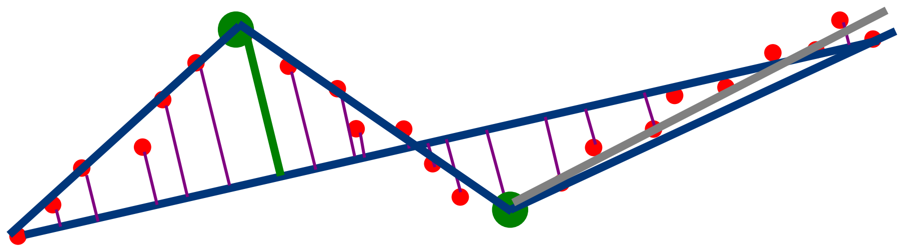


Least squares minimization of the line:

- 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



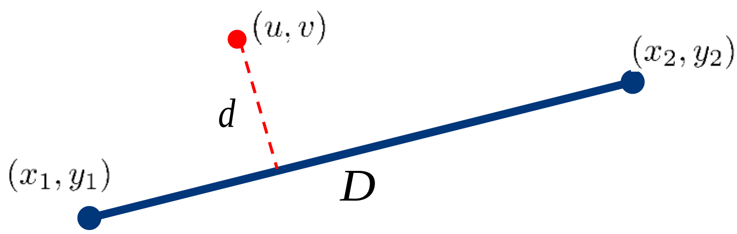
- What about corners?

➔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

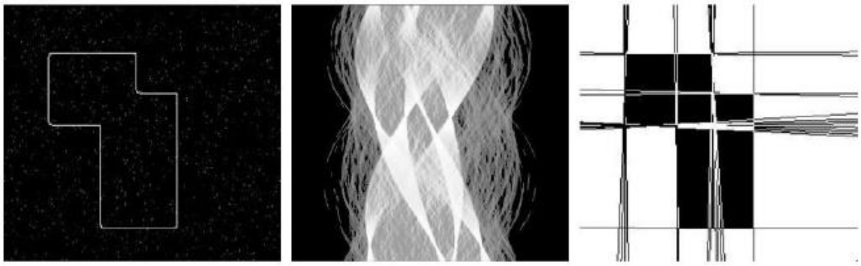


$$r = u(y_1 - y_2) + v(x_2 - x_1) + y_2 x_1 - y_1 x_2$$

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

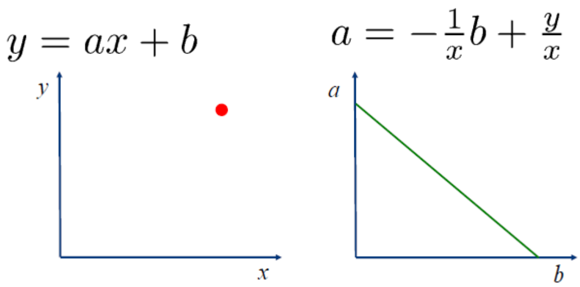
Adopted from Williams, Fitch, and Singh, MTRX 4700

Hough Transform



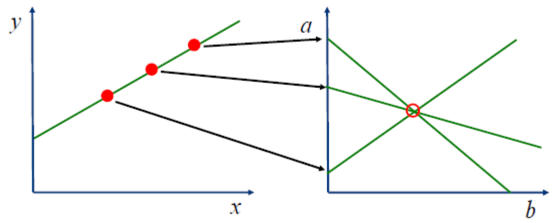
- Uses a voting mechanism
- Can be used for other lines and shapes (not just straight lines)

Hough Transform: Voting Space



- 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.

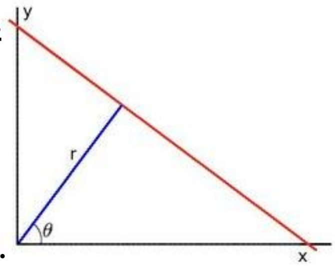
2. Assume that each cell in the parameter space is an accumulator. Initialize all cells to zero.

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:

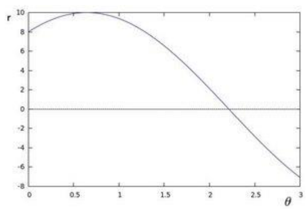
$$r = x_1 \cos\theta + y_1 \sin\theta$$

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

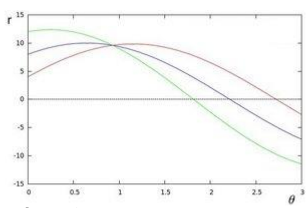
See also OpenCV documentation (cv::HoughLines)

Line Detection – Hough Lines [2]

- Thus a given point gives a sinusoid



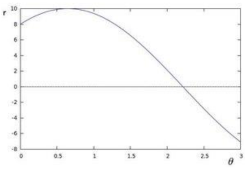
- Repeating for all points on the image



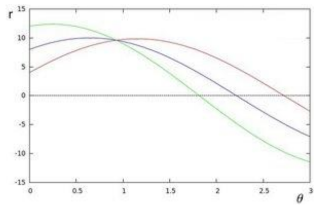
See also OpenCV documentation (cv::HoughLines)

Line Detection – Hough Lines [3]

- Thus a given point gives a sinusoid



- Repeating for all points on the image

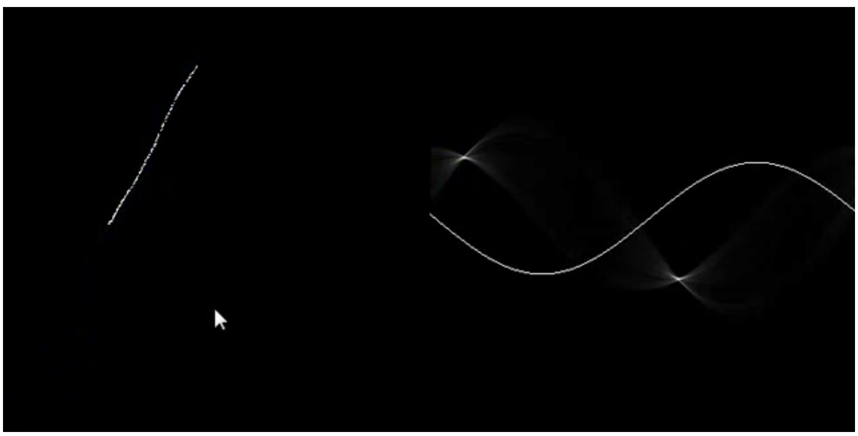


- NOTE that an intersection of sinusoids represents (**a point**) represents **a line** in which pixel points lay.

➔ 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

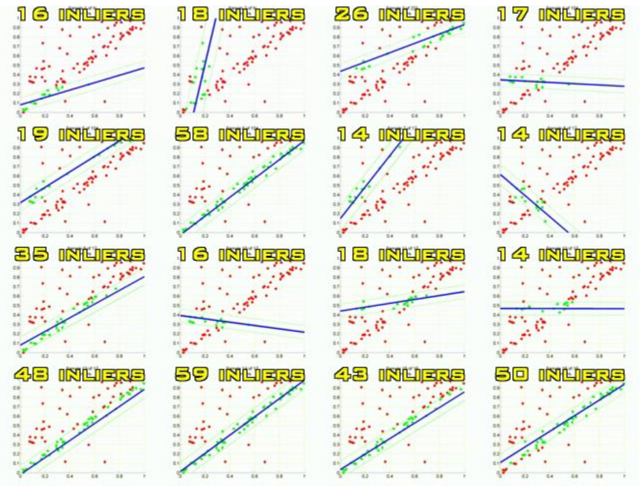
RANdom SAmple Consensus

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*