ME 598: Introduction to Robotics

Lecture 9: Computer Vision & Image Processing
Sensor Based Navigation

Stevens Institute of Technology
Dr. Mishah U. Salman
Fall 2013

Date:
By:

Slides adapted from Dr. David J. Cappelleri, Dr. M. Ani Hsieh, Drexel University, SAS Lab, Philadelphia, PA, and © R. Siegwart, ETH Zurich – ASL, http://www.mobilerobots.ethz.ch/

Mobile Robot Kinematics

Review

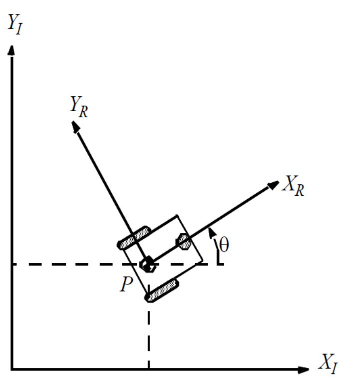
Review: Representing Robot Position

© R. Siegwart, ETH Zurich - ASL

- Representing the robot within an arbitrary initial frame
  - Inertial frame: $\{X_I, Y_I\}$
  - Robot frame: $\{X_R, Y_R\}$
  - Robot pose: $\xi_I = [x \quad y \quad \theta]^T$
  - Mapping between the two frames

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I = R(\theta)\cdot[\dot{x} \quad \dot{y} \quad \dot{\theta}]^T$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

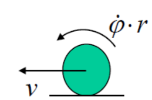


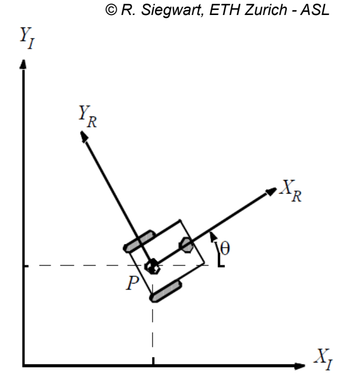
Review: Wheel Kinematic Constraints

© R. Siegwart, ETH Zurich - ASL

Assumptions:

- Movement on a horizontal plane
- Point contact of the wheels
- Wheels not deformable
- Pure rolling
  - $v_c = 0$ at contact point
- No slipping, skidding or sliding
- No friction for rotation around contact point
- Steering axes orthogonal to the surface
- Wheels connected by rigid frame (chassis)



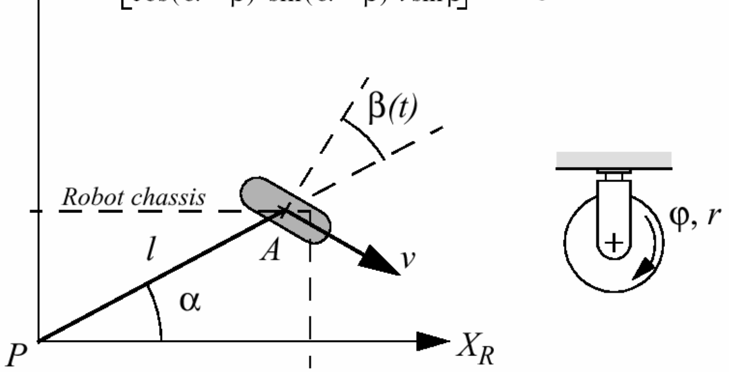


Review:
Wheel Kinematic Constraints- Fixed Standard Wheel

© R. Siegwart, ETH Zurich - ASL

$$\left[\sin(\alpha+\beta) \;\; -\cos(\alpha+\beta) \;\; (-l)\cos\beta\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0$$

$$\left[\cos(\alpha+\beta) \;\; \sin(\alpha+\beta) \;\; l\sin\beta\right] R(\theta)\dot{\xi}_I = 0$$

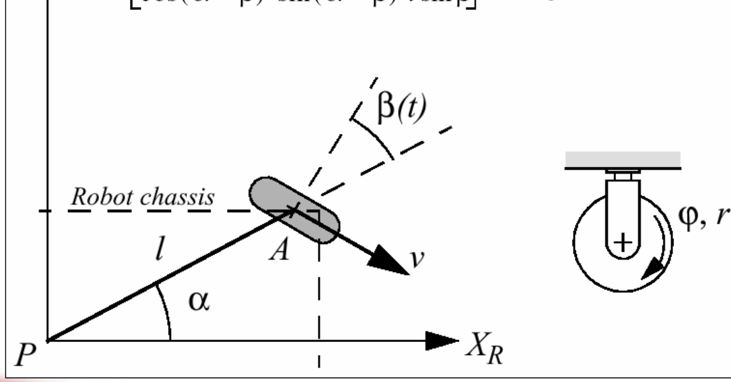


Review:
Wheel Kinematic Constraints- Steered Standard Wheel

© R. Siegwart, ETH Zurich - ASL

$$\left[\sin(\alpha+\beta) \;\; -\cos(\alpha+\beta) \;\; (-l)\cos\beta\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0$$

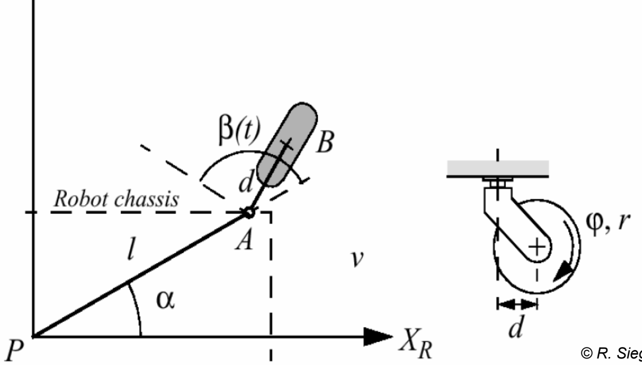
$$\left[\cos(\alpha+\beta) \;\; \sin(\alpha+\beta) \;\; l\sin\beta\right] R(\theta)\dot{\xi}_I = 0$$



Review:
Wheel Kinematic Constraints- Castor Wheel

$$\left[\sin(\alpha+\beta) \;\; -\cos(\alpha+\beta) \;\; (-l)\cos\beta\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0$$

$$\left[\cos(\alpha+\beta) \;\; \sin(\alpha+\beta) \;\; \underline{d+l}\sin\beta\right] R(\theta)\dot{\xi}_I + \underline{d}\dot{\beta} = 0$$

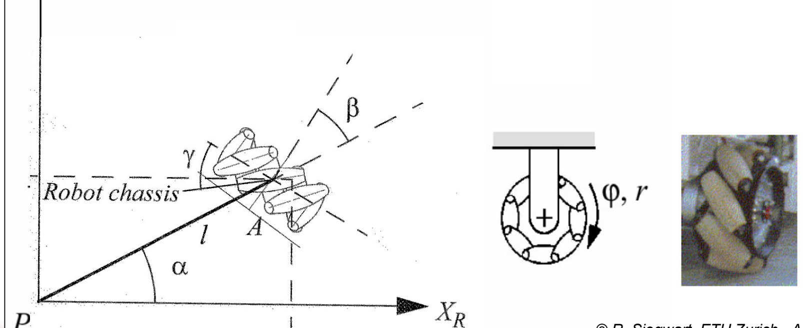


© R. Siegwart, ETH Zurich - ASL

Review:
Wheel Kinematic Constraints- Swedish Wheel

$$\left[\sin(\alpha+\beta+\gamma) \;\; -\cos(\alpha+\beta+\gamma) \;\; (-l)\cos(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi}\cos\gamma = 0$$

$$\left[\cos(\alpha+\beta+\gamma) \;\; \sin(\alpha+\beta+\gamma) \;\; l\sin(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi}\sin\gamma - r_{sw}\dot{\varphi}_{sw} = 0$$



© R. Siegwart, ETH Zurich - ASL

Review: Degrees of Freedom, Holonomy

© R. Siegwart, ETH Zurich - ASL

- DOF *degrees of freedom*:
  - Robots ability to achieve various poses
- DDOF *differentiable degrees of freedom*:
  - Robots ability to achieve various path

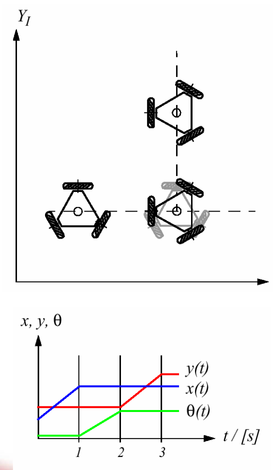
How many DOF can be controlled by just changing wheel velocities

$$DDOF \le \delta_m \le DOF$$

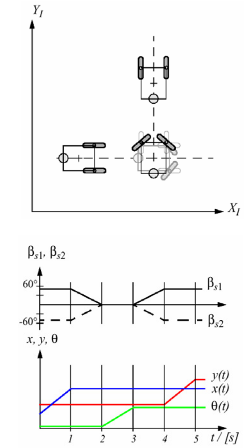
- Holonomic Robots
  - A holonomic kinematic constraint can be expressed a an explicit function of position variables only
  - A non-holonomic constraint requires a different relationship, such as the derivative of a position variable
  - *Fixed and steered standard wheels impose non-holonomic constraints*

Review: Path / Trajectory Considerations

Omnidirectional Drive

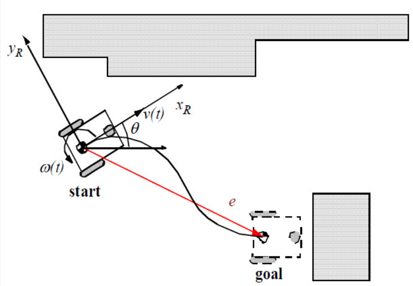


Two-Steer Drive



© R. Siegwart, ETH Zurich - ASL

Review: Motion Control- Feedback Control



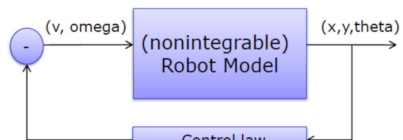
- Find a control matrix K, if exists

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix}$$

with $k_{ij}=k(t,e)$

- such that the control of $v(t)$ and $\omega(t)$

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = K \cdot e = K \cdot {}^{R}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$



- drives the error e to zero

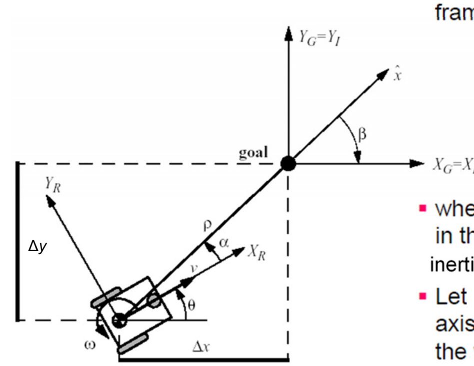
$$\lim_{t \to \infty} e(t) = 0$$

- MIMO state feedback control

© R. Siegwart, ETH Zurich - ASL

Review: Motion Control- Kinematic Model

© R. Siegwart, ETH Zurich - ASL



- The kinematics of a differential drive mobile robot described in the inertial frame $\{x_I, y_I, \theta\}$ is given by,

$${}^{I}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- where $\dot{x}$ and $\dot{y}$ are the linear velocities in the direction of the x_I and y_I of the inertial frame.
- Let α denote the angle between the x_R axis of the robots reference frame and the vector connecting the center of the axle of the wheels with the final position.

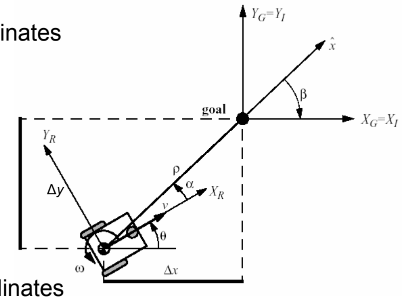
Review:
Kinematic Model: Coordinate Transformation

- Coordinate transformation into polar coordinates with its origin at goal position:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x)$$

$$\beta = -\theta - \alpha$$



- System description, in the new polar coordinates

$$\begin{bmatrix}\dot\rho\\ \dot\alpha\\ \dot\beta\end{bmatrix} = \begin{bmatrix}-\cos\alpha & 0\\ \frac{\sin\alpha}{\rho} & -1\\ -\frac{\sin\alpha}{\rho} & 0\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

for $\alpha \in I_1 = \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$

$$\begin{bmatrix}\dot\rho\\ \dot\alpha\\ \dot\beta\end{bmatrix} = \begin{bmatrix}\cos\alpha & 0\\ -\frac{\sin\alpha}{\rho} & -1\\ \frac{\sin\alpha}{\rho} & 0\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

for $\alpha \in I_2 = (-\pi, -\pi/2] \cup (\pi/2, \pi]$

Review:
Kinematic Position Control- Control Law

- It can be shown, that with

$$v = k_\rho \rho \qquad \omega = k_\alpha \alpha + k_\beta \beta$$

the feedback controlled system

$$\begin{bmatrix}\dot\rho\\ \dot\alpha\\ \dot\beta\end{bmatrix} = \begin{bmatrix}-k_\rho \rho \cos\alpha\\ k_\rho \sin\alpha - k_\alpha \alpha - k_\beta \beta\\ -k_\rho \sin\alpha\end{bmatrix}$$

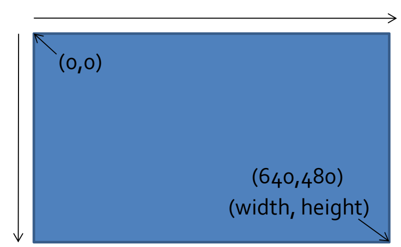
will drive the robot to $(\rho, \alpha, \beta) = (0, 0, 0)$

- The control signal v has always constant sign,
  - the direction of movement is kept positive or negative during movement
  - parking maneuver is performed always in the most natural way and without ever inverting its motion.

Computer Vision & Image Processing

Computer Vision & Image Processing:
Representing Images

- Images: width x height (pixels)

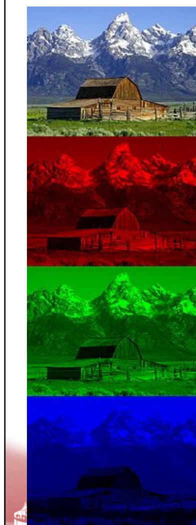


- In MATLAB: image = matrix
  - Height = # rows (y coordinate)
  - Width = # columns (x coordinate)
  - I(y,x) → pixel value according to image x,y coordinate

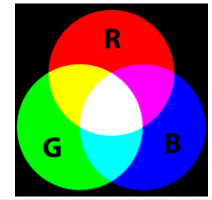
Computer Vision & Image Processing: Grayscale Images

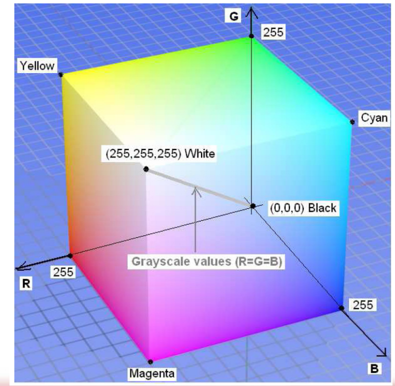
- Grayscale images (black and white)
  - size(I) = rows x columns = height x width
  - Pixel values: 0 (black) → 255 (white)

Computer Vision & Image Processing: Color Images

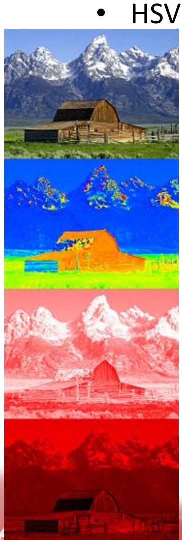


- RGB color model
  - Size(I) = rows x columns x 3
  - Red, green, and blue layers (RGB)
  - Pixel values: 0 → 255
    - (0, 0, 0) is black
    - (255, 255, 255) is white
    - (255, 0, 0) is red
    - (0, 255, 0) is green
    - (0, 0, 255) is blue
    - (255, 255, 0) is yellow
    - (0, 255, 255) is cyan
    - (255, 0, 255) is magenta

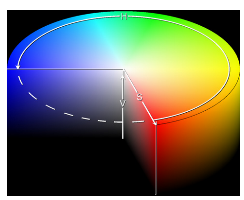




Computer Vision & Image Processing: Color Images



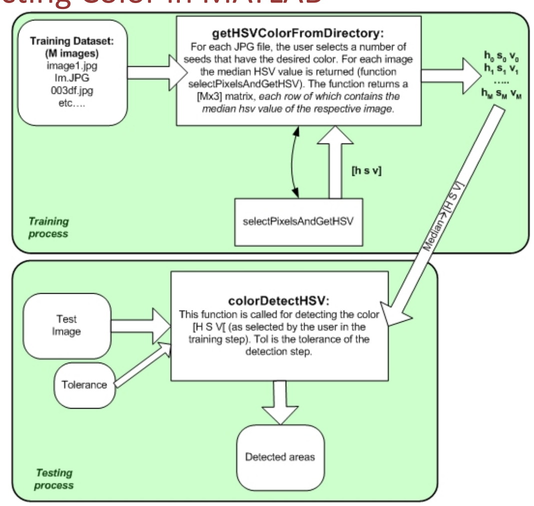
- HSV color model
  - Size(I) = rows x columns x 3
  - Hue, Saturation, and Value layers (HSV)
  - Colors as points in a cylinder
    - Central axis ranges from black at the bottom to white at the top with neutral colors between them
    - Angle around the axis corresponds to "hue"
    - Distance from the axis corresponds to "saturation"
    - Distance along the axis corresponds to "value"
  - H: Hue represents color, angle from 0° to 360° (can be normalized between 0 and 1)
  - S: Saturation indicates the grey in color space, 0 to 100% (or 0 to 1); 0 = grey, 1 = primary color
  - V: Value is brightness of the color and varies with saturation; ranges from 0 to 100% (0 = totally black)



Angle	Color
0-60	Red
60-120	Yellow
120-180	Green
180-240	Cyan
240-300	Blue
300-360	Magenta

Computer Vision & Image Processing: Detecting Color in MATLAB

Code from MATLAB File Exchange:
Theodoros Giannakopoulos
Department of Informatics and Telecommunications
University of Athens, Greece



http://www.mathworks.com/matlabcentral/fileexchange/18440

Computer Vision & Image Processing: Detecting Color in MATLAB

- example.m
  - Create Training data set of images to determine HSV values that you are looking for and place in training directory (‘train’)

```
% STEP 1: Use getHSVColorFromDirectory(dirName) in order to estimate the
% average HSV values of your objects of interest.

HSV = getHSVColorFromDirectory('train');

%
% The above function call will let the user choose manually (through simple
% mouse clicks) several "seeds" from each image.
% At the end the HSV matrix contains M rows (M is the total number of jpeg files
% in dirName): each row corresponds to the average HSV value of the
% selected seeds in the respective image.
% The average (or median) value of this matrix (column-wise) can be used,
% in the sequence for detecting the speficic color values.
%

% STEP 2: Use the estimated (average) hsv value for detecting the specified
% color in a specific image.

colorDetectHSV('test/face01.jpg', median(HSV), [0.05 0.05 0.2]);
```


Computer Vision & Image Processing: Detecting Color in MATLAB

- Training data
  - Left-click in color region you want to detect (at least 10 in each image)
  - Right-click once you have finished selecting all points in the image
  - Repeat for all images in training directory

Computer Vision & Image Processing: Detecting Color in MATLAB

- example.m
  - Step 2. Use the estimated HSV value for detecting the specified color in a particular image

```
% STEP 1: Use getHSVColorFromDirectory(dirName) in order to estimate the
% average HSV values of your objects of interest.

HSV = getHSVColorFromDirectory('train');

%
% The above function call will let the user choose manually (through simple
% mouse clicks) several "seeds" from each image.
% At the end the HSV matrix contains M rows (M is the total number of jpeg files
% in dirName): each row corresponds to the average HSV value of the
% selected seeds in the respective image.
% The average (or median) value of this matrix (column-wise) can be used,
% in the sequence for detecting the speficic color values.
%

% STEP 2: Use the estimated (average) hsv value for detecting the specified
% color in a specific image.

colorDetectHSV('test/face01.jpg', median(HSV), [0.05 0.05 0.2]);
```


Computer Vision & Image Processing: Detecting Color

- colorDetectHSV.m
  - Inputs:
    - image filename
    - hsvValue
    - Tolerance
  - Calculates HSV values for all pixels in image
  - Compares with HSV values that you’re searching for
  - New image initialized as all black pixels; If HSV difference < tol, turns respective pixel white
  - Output:
    - Figure with:
      - Original image
      - Color detected image

```
function colorDetectHSV(fileName, hsvVal, tol)

%
% function colorDetectHSV(fileName, hsvVal, tol)
%
% This function is used for detecting a specified hsv value in images.
%
% ARGUMENTS:
% fileName: the name of the jpg file to be loaded
% hsvVal: 3x1 array containing the HSV values to be detected
% tol: 1x1 or 2x1 or 3x1 array containing the tolerance (i.e. the maximum
% distance - in each hsv coefficient - of each pixel from hsvVal).
%
% Example:
% colorDetectHSV('train/face07.jpg', median(HSV), [0.05 0.05 0.1]);
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Theodoros Giannakopoulos - January 2008
% www.di.uoa.gr/~tyiannak
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


RGB = imread(fileName);

HSV = rgb2hsv(RGB);

% find the difference between required and real H value:
diffH = abs(HSV(:,:,1) - hsvVal(1));
```

Color Detection Demo

Computer Vision & Image Processing: Blob Analysis- Opening

bwareaopen

Morphologically open binary image (remove small objects)

Syntax

```
BW2 = bwareaopen(BW,P)
BW2 = bwareaopen(BW,P,conn)
```

Description

`BW2 = bwareaopen(BW,P)` removes from a binary image all connected components (objects) that have fewer than `P` pixels, producing another binary image, `BW2`. The default connectivity is 8 for two dimensions, 26 for three dimensions, and `conndef(ndims(BW),'maximal')` for higher dimensions.

`BW2 = bwareaopen(BW,P,conn)` specifies the desired connectivity. `conn` can have any of the following scalar values.

Value	Meaning
Two-dimensional connectivities	
4	4-connected neighborhood
8	8-connected neighborhood
Three-dimensional connectivities	
6	6-connected neighborhood
18	18-connected neighborhood
26	26-connected neighborhood

Connectivity can be defined in a more general way for any dimension by using for `conn` a 3-by-3-by-...-by-3 matrix of 0's and 1's. The 1-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element.

Computer Vision & Image Processing: Blob Analysis- Opening

CD = Original BW image

CDfiltered = bwareaopen(CD, 200);

Removes connected pixel regions less than 200 pixels in size

Computer Vision & Image Processing: Blob Analysis- Filling

imfill

Fill image regions and holes

Syntax

```
BW2 = imfill(BW)
[BW2,locations] = imfill(BW)
BW2 = imfill(BW,locations)
BW2 = imfill(BW,'holes')
I2 = imfill(I)
BW2 = imfill(BW,locations,conn)
```

Description

`BW2 = imfill(BW)` displays the binary image `BW` on the screen and lets you define the region to fill by selecting points interactively on using the mouse. To use this interactive syntax,`BW` must be a 2-D image. Press **Backspace** or **Delete** to remove the previously selected point. A shift-click, right-click, or double-click selects a final point and starts the fill operation. Pressing **Return** finishes the selection without adding a point.

`[BW2,locations] = imfill(BW)` returns the locations of points selected interactively in `locations`. `locations` is a vector of linear indices into the input image. To use this interactive syntax,`BW` must be a 2-D image.

`BW2 = imfill(BW,locations)` performs a flood-fill operation on background pixels of the binary image `BW`, starting from the points specified in `locations`. If `locations` is a P-by-1 vector, it contains the linear indices of the starting locations. If `locations` is a P-by-`ndims(BW)` matrix, each row contains the array indices of one of the starting locations.

`BW2 = imfill(BW,'holes')` fills holes in the binary image `BW`. A hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

`I2 = imfill(I)` fills holes in the grayscale image `I`. In this syntax, a hole is defined as an area of dark pixels surrounded by lighter pixels.

`BW2 = imfill(BW,locations,conn)` fills the area defined by `locations`, where `conn` specifies the connectivity. `conn` can have any of the following scalar values.

Computer Vision & Image Processing: Blob Analysis- Filling

CDfiltered = bwareaopen(CD, 200);

CDfilled = imfill(CDfiltered, 'holes');

Computer Vision & Image Processing: Blob Analysis- Labelling

bwlabel

Label connected components in binary image

Syntax

```
L = bwlabel(BW,n)
[L,num] = bwlabel(BW,n)
```

Description

`L = bwlabel(BW,n)` returns a matrix `L`, of the same size as `BW`, containing labels for the connected objects in `BW`. `n` can have a value of either 4 or 8, where 4 specifies 4-connected objects and 8 specifies 8-connected objects; if the argument is omitted, it defaults to 8.

The elements of `L` are integer values greater than or equal to 0. The pixels labeled 0 are the background. The pixels labeled 1 make up one object, the pixels labeled 2 make up a second object, and so on.

`[L,num] = bwlabel(BW,n)` returns in `num` the number of connected objects found in `BW`.

Computer Vision & Image Processing: Blob Analysis- Measuring Properties

regionprops

Measure properties of image regions (blob analysis)

Syntax

```
STATS = regionprops(L, properties)
STATS = regionprops(L, I,properties)
```

Description

`STATS = regionprops(L, properties)` measures a set of properties for each labeled region `L`. `L` can be a label matrix or a multidimensional array. When`L` is a label matrix, positive integer elements of `L` correspond to different regions. For example, the set of elements of `L` equal to `1` corresponds to region 1; the set of elements of `L` equal to `2` corresponds to region 2; and so on. The return value `STATS` is a structure array of length `max(L(:))`. The fields of the structure array denote different measurements for each region, as specified by *properties*. See Properties for a list of valid property strings.

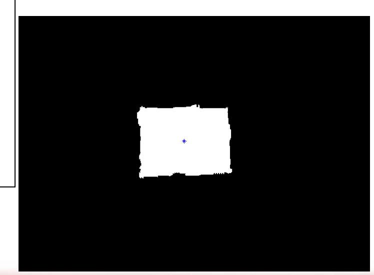
`STATS = regionprops(L, I,properties)` measures a set of properties for each labeled region in the 2-D or N-D grayscale image `I`. `L` is a label matrix that identifies the regions in `I` and must have the same size as `I`.

Properties

`properties` can be a comma-separated list of strings, a cell array containing strings, the single string `'all'`, or the string `'basic'`. If `properties` is the string `'all'`, `regionprops` computes all the shape measurements, listed in Shape Measurements. If called with a grayscale image, `regionprops` also returns the pixel value measurements, listed in Pixel Value Measurements. If `properties` is not specified or if it is the string `'basic'`, `regionprops` computes only the `'Area'`, `'Centroid'`, and `'BoundingBox'` measurements. The following properties can be calculated on N-D label matrices: `'Area'`, `'BoundingBox'`,`'Centroid'`, `'FilledArea'`, `'FilledImage'`, `'Image'`,`'PixelIdxList'`, `'PixelList'`, and `'SubarrayIdx'`.

Computer Vision & Image Processing: Blob Analysis Example

```
% Label connected components
L = bwlabel(CDfilled);
% Calculate region properties for connected components
s  = regionprops(L);
% Concatenate an array of all the regions 'area' values
areas = cat(1, s.Area);
% Concatenate an array of all the regions 'centroid' values
centroids = cat(1, s.Centroid);
% Identify largest area
max_area = max(areas);
% Find the index in the 'areas' array corresponding to max_area
idx = find(areas == max_area);
% Get the centroid value for the region with the largest area
centroidX = centroids(idx,1);
centroidY = centroids(idx,2);
% Select the connected component corresponding to this region
BW2 = ismember(L,idx);
% Plot the image of the largest connected region
figure(3)
imshow(BW2)
hold on
% Plot a blue star in centroid of region
plot(centroidX, centroidY, 'b*')
```



Blob Analysis Demo

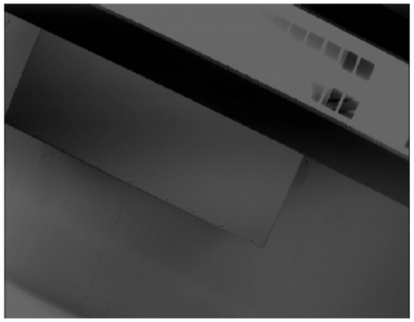
Computer Vision & Image Processing: Edge Detection

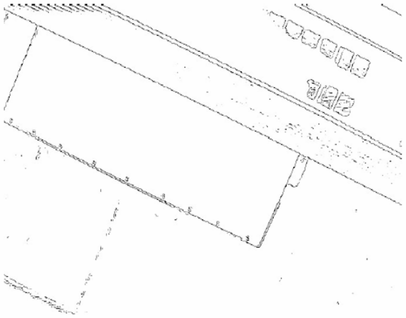
M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Edge Detection

- Ultimate goal of edge detection
  - an idealized line drawing.
- Edge contours in the image correspond to important scene contours.

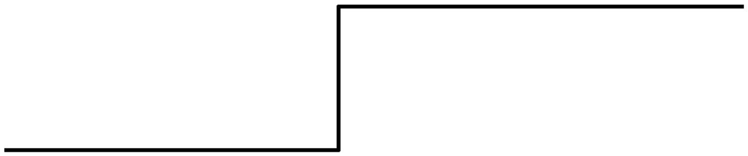
© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL





Computer Vision & Image Processing: Edge Detection

- Edges correspond to sharp changes of intensity
- Change is measured by 1st derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.



© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

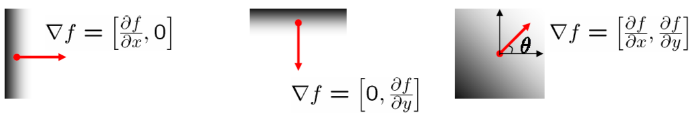
Computer Vision & Image Processing: Edge Detection

- The gradient of an image:

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient points in the direction of most rapid change in intensity



- The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

  - how does this relate to the direction of the edge? ← perpendicular!
- The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Computer Vision & Image Processing: Edge Detection

- How can we differentiate a *digital* image f[x,y]?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx f[x+1, y] - f[x, y]$$

Convolution

*A convolution is an integral that expresses the amount of overlap of one function **g** as it is shifted over another function **f***

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

Computer Vision & Image Processing: Convolution

- Definition
  - Continuous

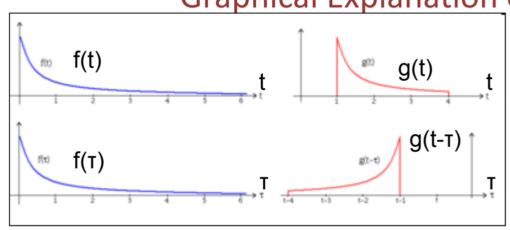
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t-\tau) d\tau$$

  - Discrete

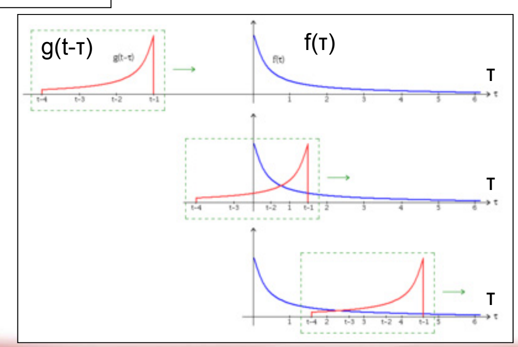
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m]$$

M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Graphical Explanation of Convolution



M. Ani Hsieh, Drexel University, SAS Lab



Computer Vision & Image Processing: Convolution By Applying Masks to Images

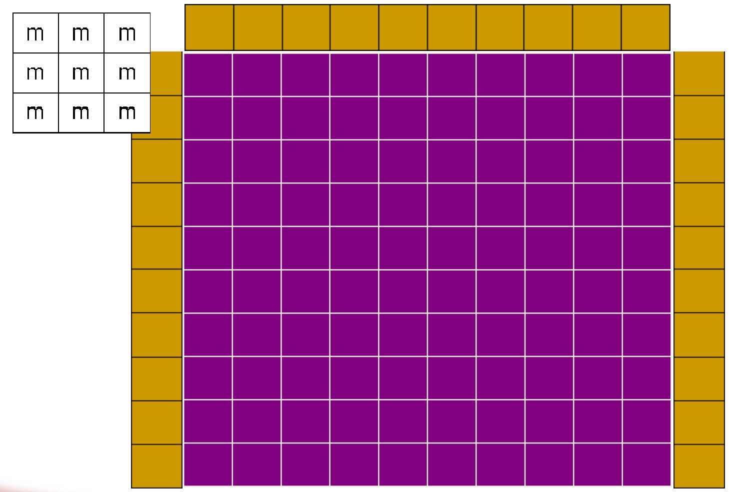
1	1	1
1	1	1
1	1	1

1	2	1
2	4	2
1	2	1

1
1
1
1
1
1

- Convolution of the image w/ another "signal"
- *Masks* have origins
  - Symmetric masks – origins are the center pixels

Computer Vision & Image Processing: Applying Masks to Images



Computer Vision & Image Processing: Linear Functions & Filters

- Simplest: linear filters
  - Key idea: replace each pixel by a linear combination of its neighbors
- The prescription for the linear combination is called the convolution kernel

10	5	3
4	5	1
1	1	7

Local image data

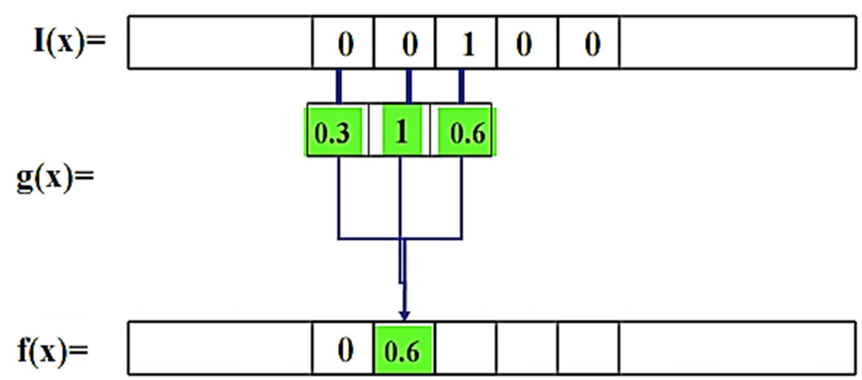
0	0	0
0	0.5	0
0	1	0.5

kernel

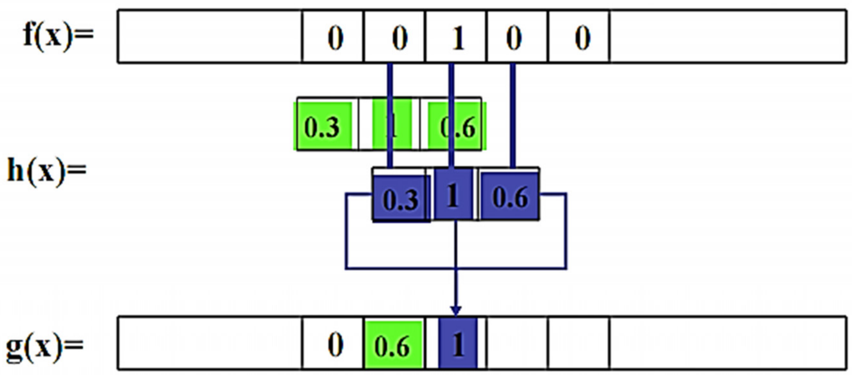
	7	

Modified image data

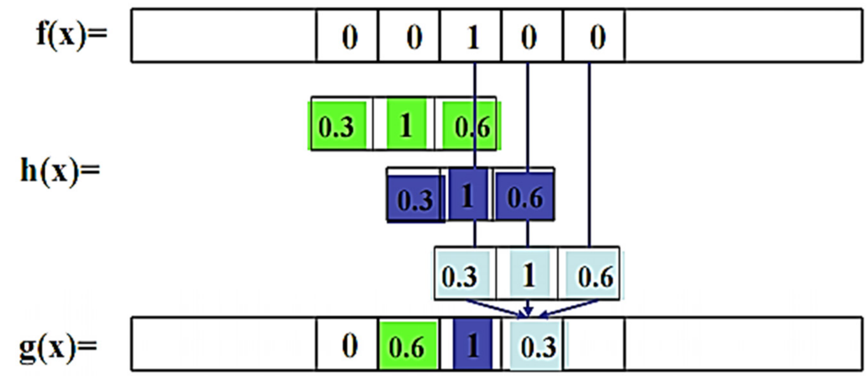
Computer Vision & Image Processing: Linear Functions Example- 1D Linear Filter



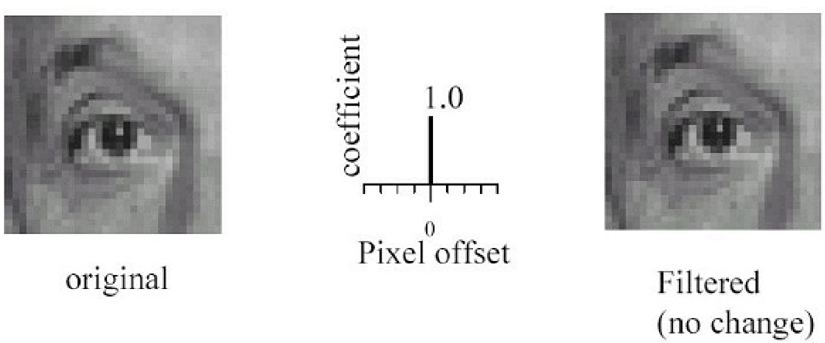
Computer Vision & Image Processing: Linear Functions Example- 1D Linear Filter



Computer Vision & Image Processing: Linear Functions Example- 1D Linear Filter

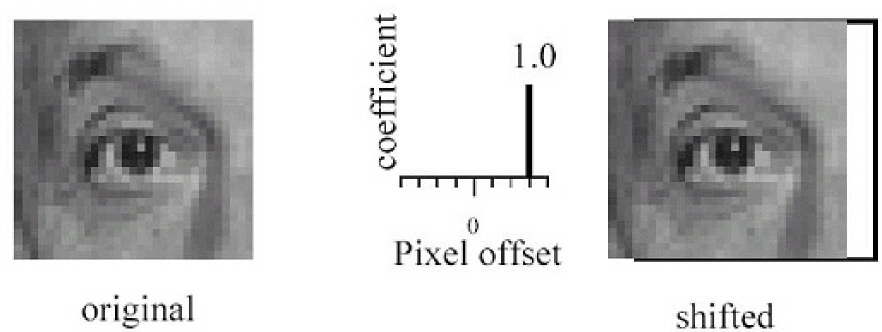


Computer Vision & Image Processing: Linear Filtering- Exercise I

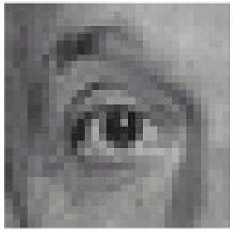


(Following examples taken from B. Freeman)

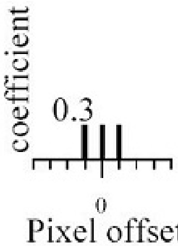
Computer Vision & Image Processing: Linear Filtering- Exercise II

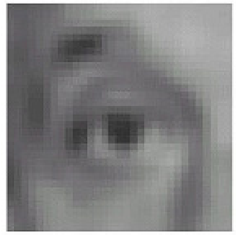


Computer Vision & Image Processing: Linear Filtering- Exercise III



original

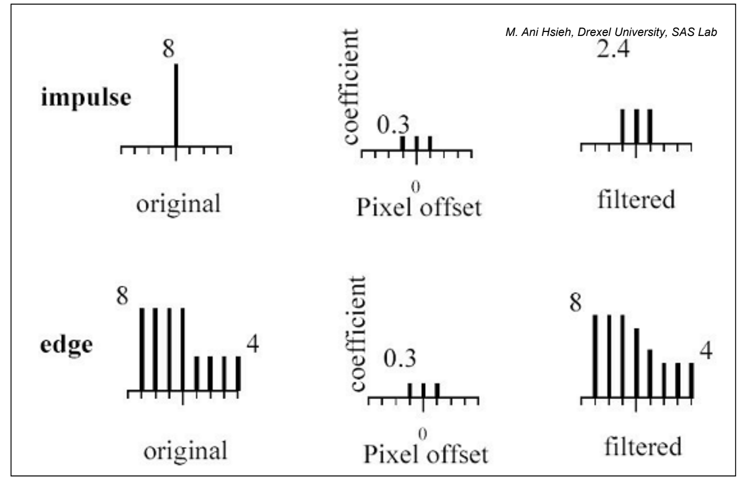




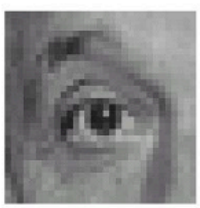
Blurred (filter applied in both dimensions).

M. Ani Hsieh, Drexel University, SAS Lab

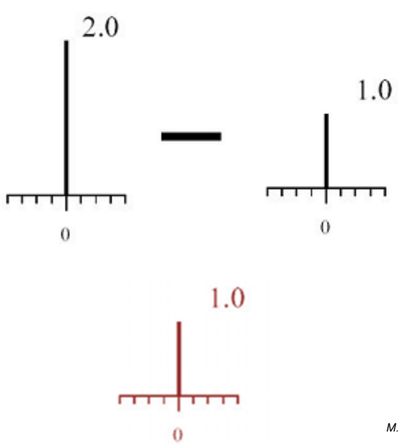
Computer Vision & Image Processing: Linear Filtering- Blur Examples



Computer Vision & Image Processing: Linear Filtering- Exercise IV



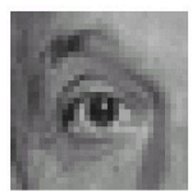
original



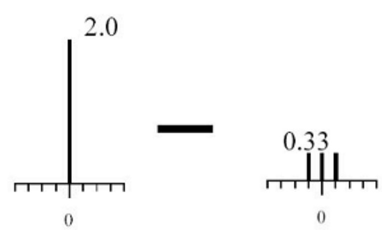
Filtered (no change)

M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Linear Filtering- Exercise V



original

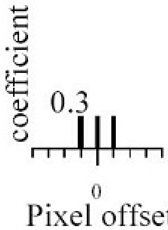


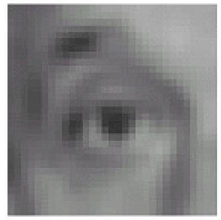
?

M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Linear Filtering- Exercise V (Remember Blurring?)

original

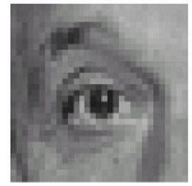




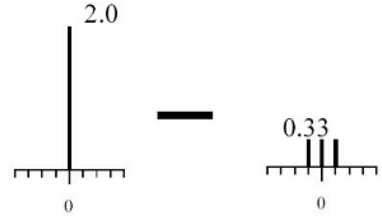
Blurred (filter applied in both dimensions).

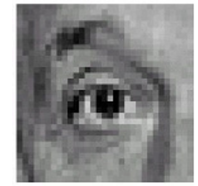
M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Linear Filtering- Exercise V



original

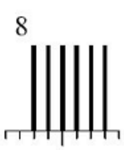




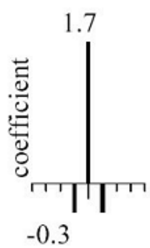
Sharpened original

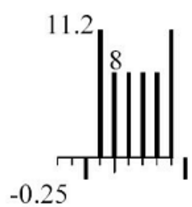
M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Linear Filtering- Sharpening Examples



original

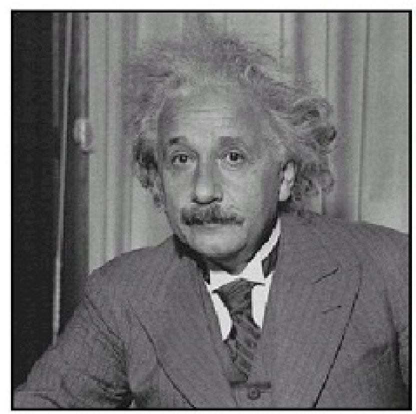




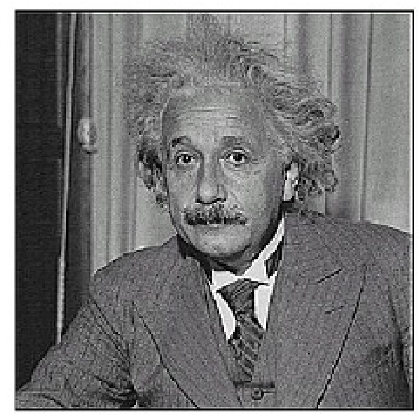
Sharpened (differences are accentuated; constant areas are left untouched).

M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Linear Filtering- Sharpening Example



before



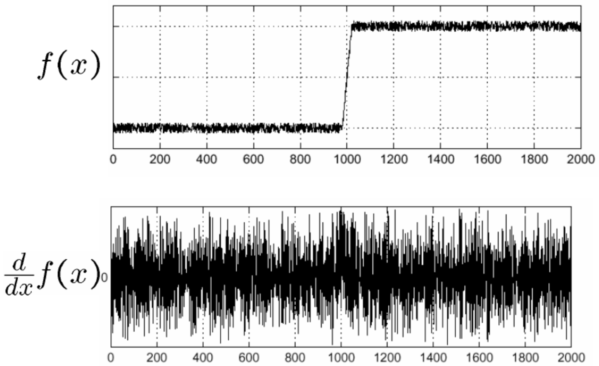
after

M. Ani Hsieh, Drexel University, SAS Lab

Computer Vision & Image Processing: Edge Detection

- Noise Effects:

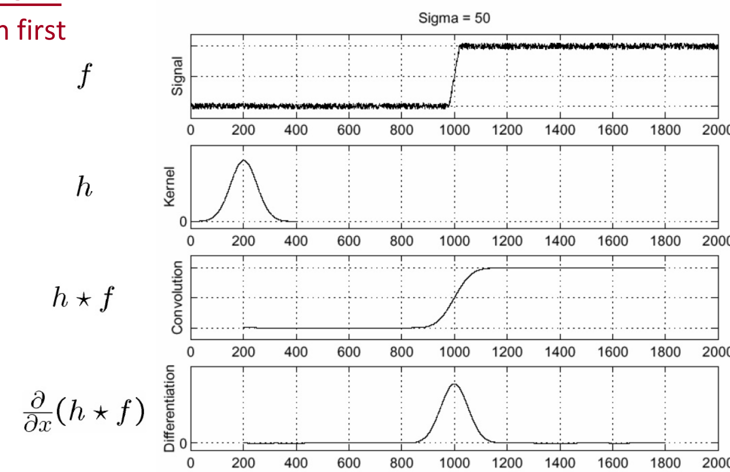
- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



- Where is the edge?

Computer Vision & Image Processing: Edge Detection

- Solution:
  Smooth first



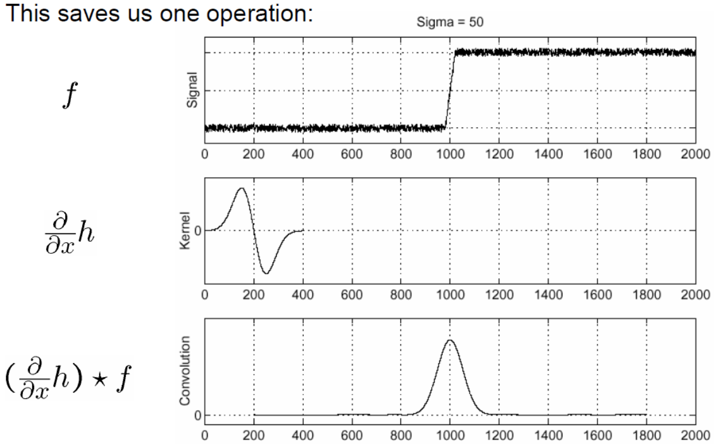
- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Computer Vision & Image Processing: Edge Detection

Derivative theorem of convolution:

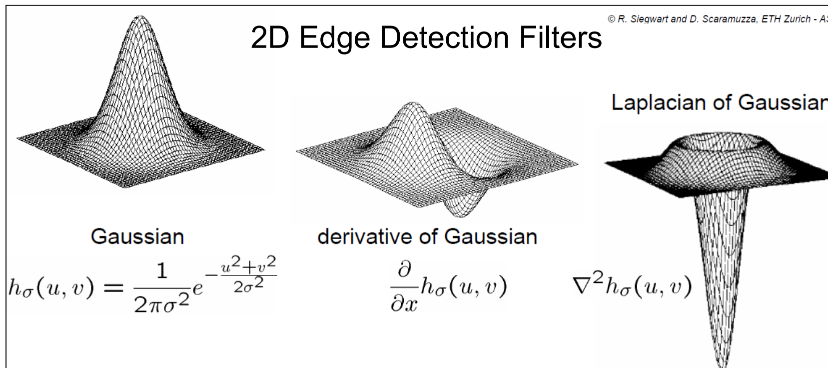
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:



Computer Vision & Image Processing: 2D Edge Detection

2D Edge Detection Filters



$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{2\sigma^2}}$ $\quad \frac{\partial}{\partial x}h_\sigma(u,v)$ $\quad \nabla^2 h_\sigma(u,v)$

- ∇^2 is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Computer Vision & Image Processing: Optimal Edge Detection- Canny

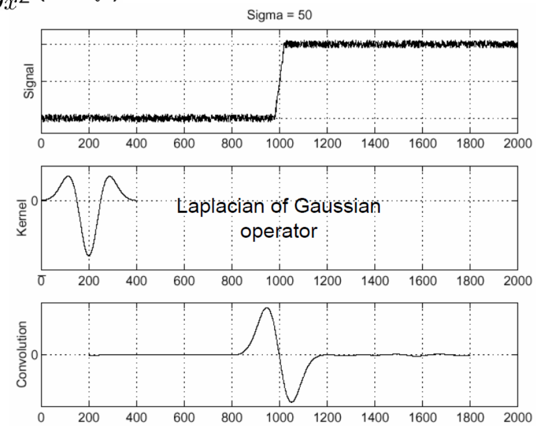
© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

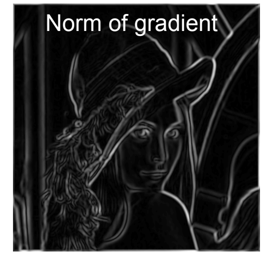
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$

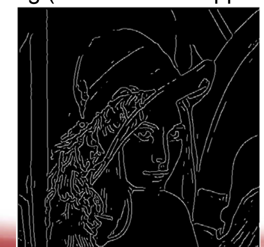


- Where is the edge?
- Zero-crossings of bottom graph

Computer Vision & Image Processing: Edge Detection Example- Canny Edge Detector



Thinning (non-maxima suppression)



© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

Computer Vision & Image Processing: Gradient Edge Detectors

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

- **Roberts**

$$|G| \cong \sqrt{r_1^2 + r_2^2}\ ;\quad r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\ ;\quad r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

- **Prewitt**

$$|G| \cong \sqrt{p_1^2 + p_2^2}\ ;\quad \theta \cong \operatorname{atan}\left(\frac{p_1}{p_2}\right)\ ;\quad p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}\ ;\quad p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

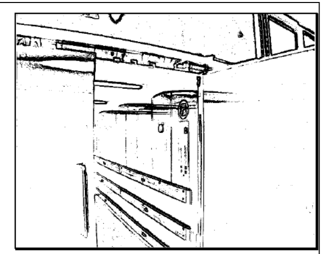
- **Sobel**

$$|G| \cong \sqrt{s_1^2 + s_2^2}\ ;\quad \theta \cong \operatorname{atan}\left(\frac{s_1}{s_2}\right)\ ;\quad s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}\ ;\quad s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Computer Vision & Image Processing: Edge Detection- Nonmaxima Suppression

Nonmaxima Suppression

- Output of an edge detector is usually a b/w image where the pixels with gradient magnitude above a predefined threshold are black and all the others are white
- Nonmaxima suppression generates contours described with only one pixel thinness



© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

Computer Vision & Image Processing: Edge Detection Example- Sobel Filter

a) Raw image

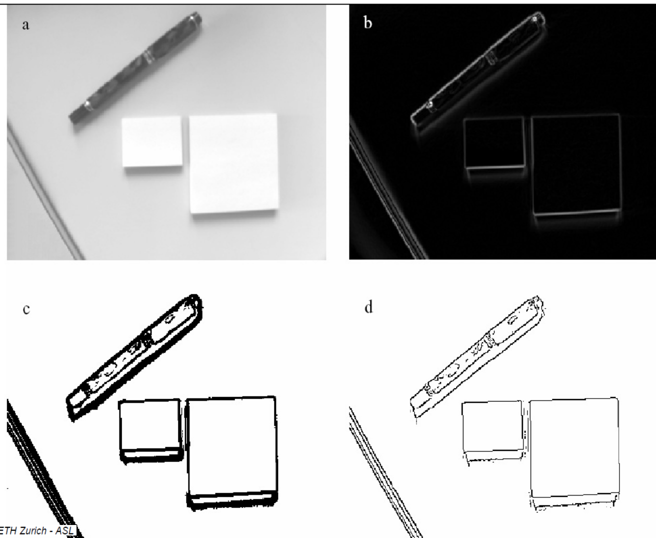
b) Filtered (Sobel)

c) Thresholding

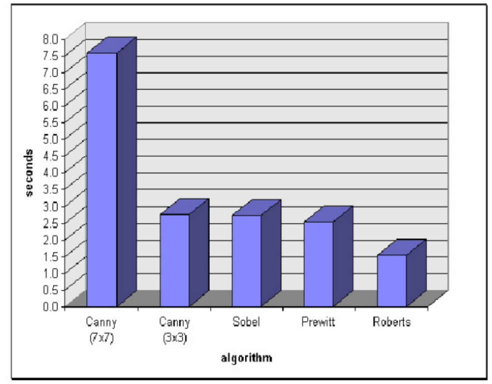
d) Nonmaxima suppression

$$s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Computer Vision & Image Processing: Comparison of Edge Detection Methods

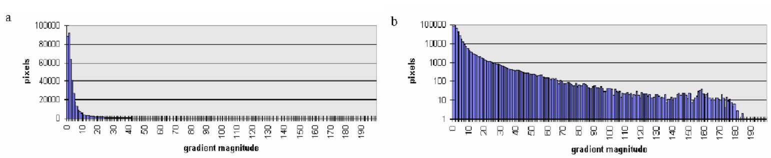


- Average time required to compute the edge figure of a 780 x 560 pixels image.
- The times required to compute an edge image are proportional with the accuracy of the resulting edge images

Computer Vision & Image Processing: Edge Detection- Dynamic Thresholding

Dynamic Thresholding for Unstructured Environments

- Changing illumination
  - Constant threshold level in edge detection is not suitable
- Dynamically adapt the threshold level
  - consider only the n pixels with the highest gradient magnitude for further calculation steps.



(a) Number of pixels with a specific gradient magnitude in the image of Figure 1.2(b).
(b) Same as (a), but with logarithmic scale

Computer Vision & Image Processing: Line Detection

- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- Option 2:
  - Use a voting scheme: Hough transform

Computer Vision & Image Processing: Hough Transform- Straight-Line Detection

- All points p on a straight-line edge must satisfy $y_p = m_1\, x_p + b_1$.
- Each point (x_p, y_p) that is part of this line constraints the parameter m_1 and b_1.
- The Hough transform finds the line (line-parameters m, b) that gets most "votes" from the edge pixels in the image.
- This is realized by four steps
  1. Create a 2D array A *[*m,b*]* with axes that tessellate the values of m and b.
  2. Initialize the array A to zero.
  3. For each edge pixel (x_p, y_p) in the image, loop over all values of m and b: if $y_p = m_1\, x_p + b_1$ then A*[*m,b*]*+=1
  4. Search cells in A with largest value. They correspond to extracted straight-line edge in the image.

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

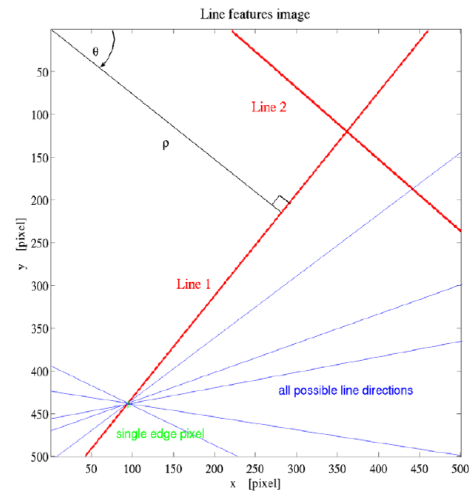
Computer Vision & Image Processing: Hough Transform- Straight-Line Detection

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

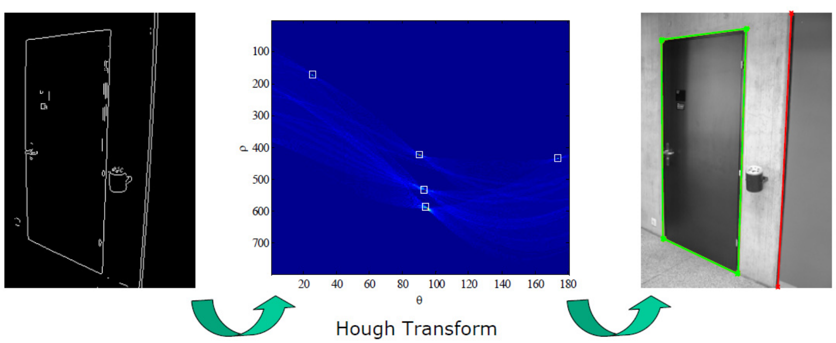
- Curve function and parameters:

$$f(\mathbf{x}, \mathbf{a}) = x\cos(\theta) + y\sin(\theta) - \rho$$
$$\mathbf{a} = (\rho, \theta)^T$$

- Hough transformation of a single edge pixel is a sine wave in parameter space.
- If the edge pixel direction is available a pixel transforms into a single accumulator cell.



Computer Vision & Image Processing: Hough Transform- Straight-Line Detection Example

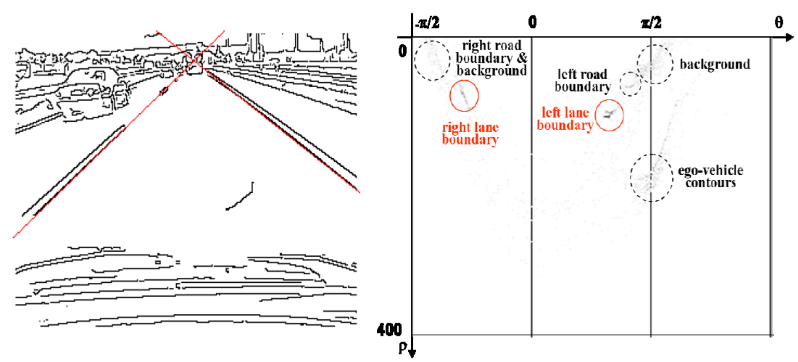


© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

Computer Vision & Image Processing: Edge Detection + Line Detection

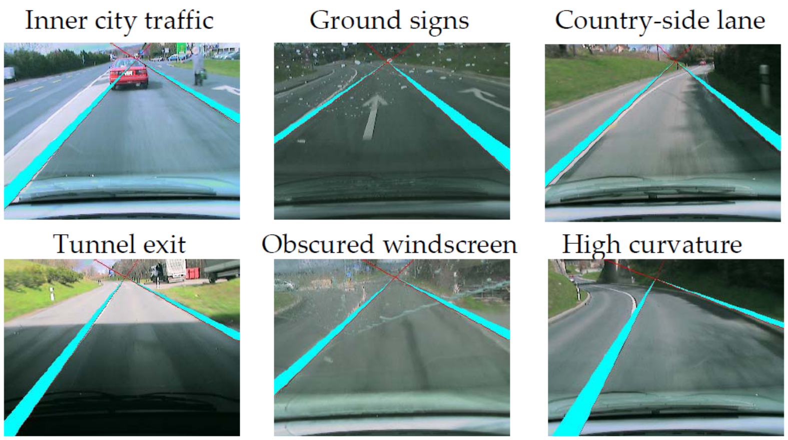
- A Canny edge pixel map and the resulting Hough transform accumulator array:

© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL



- Lane boundaries: $(\theta_l, \rho_l) = \{59°, 105\}$, $(\theta_r, \rho_r) = \{-51°, 78\}$.

Computer Vision & Image Processing: Edge Detection + Line Detection

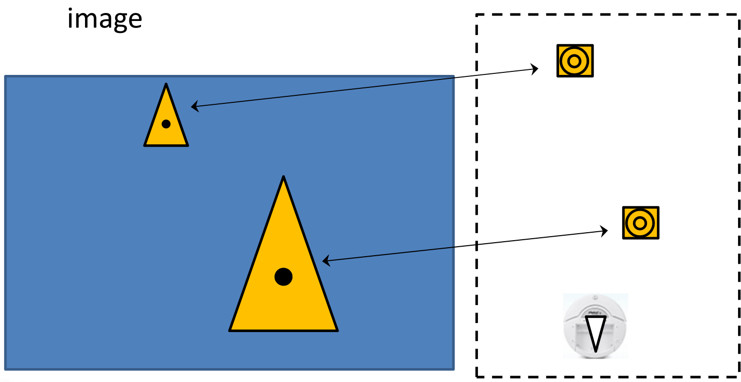


© R. Siegwart and D. Scaramuzza, ETH Zurich - ASL

Sensor Based Navigation

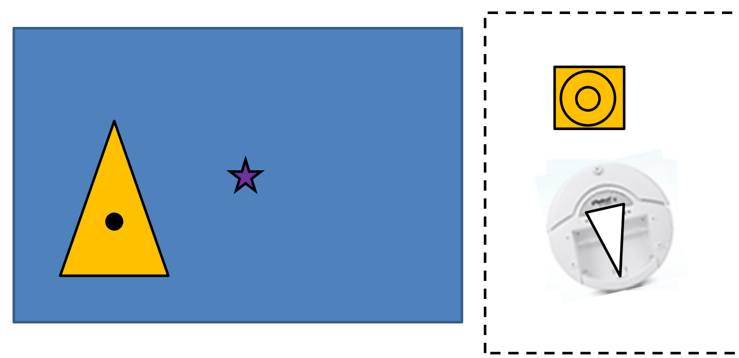
Sensor Based Navigation: Vision-Based Navigation

- Distance sensor
  - Distance from object proportional to area of object in image



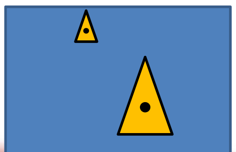
Sensor Based Navigation: Vision-Based Navigation

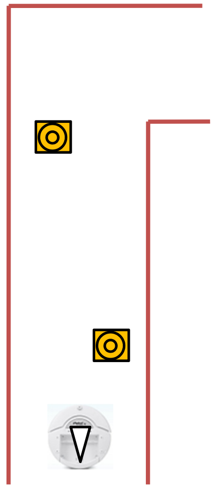
- Track target/landmark
  - Rotate robot to keep target in center of image



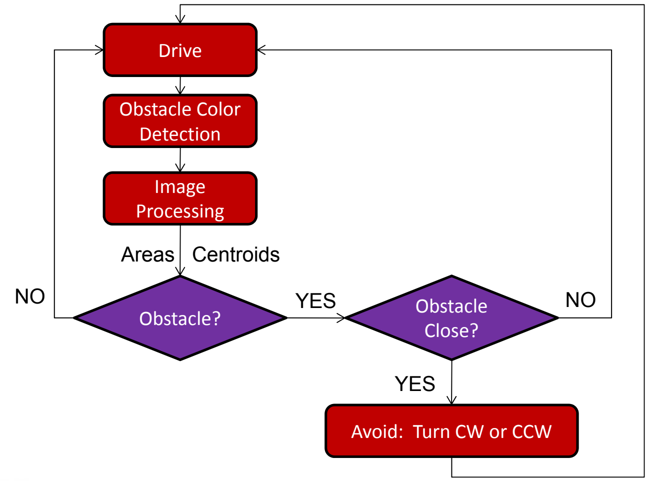
Sensor Based Navigation: Vision-Based Navigation

- Obstacle Avoidance
  - Distance sensing
    - Object close → avoid
    - Object far → OK
  - Avoidance: opposite of target tracking
    - Rotate robot so object is ***not*** in center of image





Sensor Based Navigation: Vision-Based Navigation Scheme



Sensor Based Navigation: Vision + Sensor Based Navigation Scheme