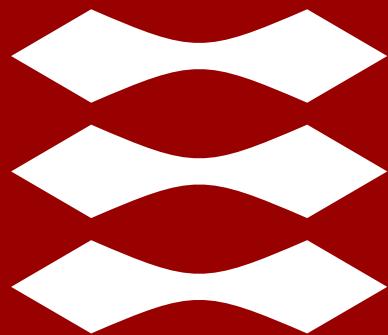


DTU



Lazaros Nalpantidis

Image Processing

- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

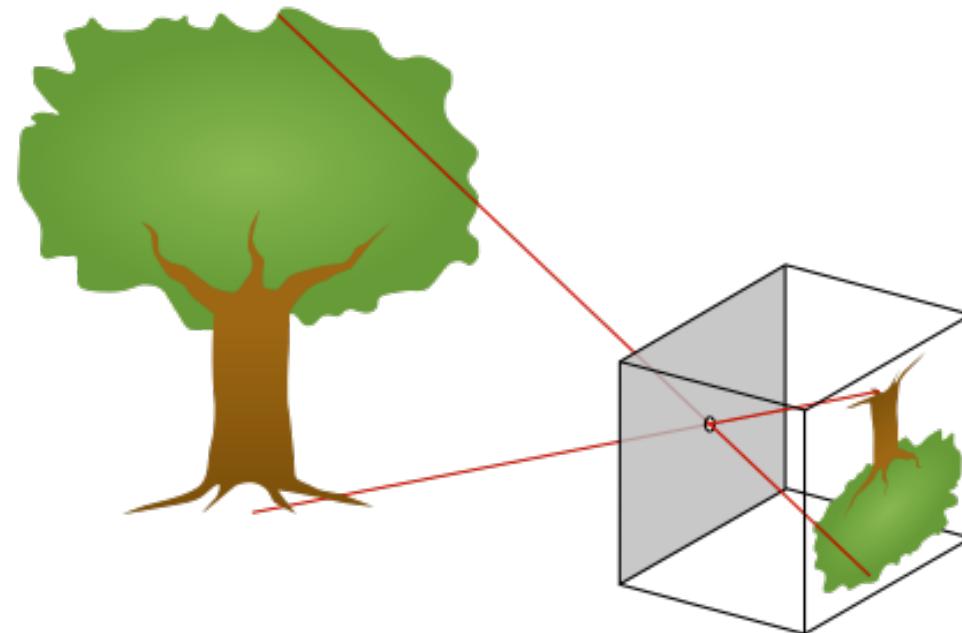
- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

What is Image Processing?

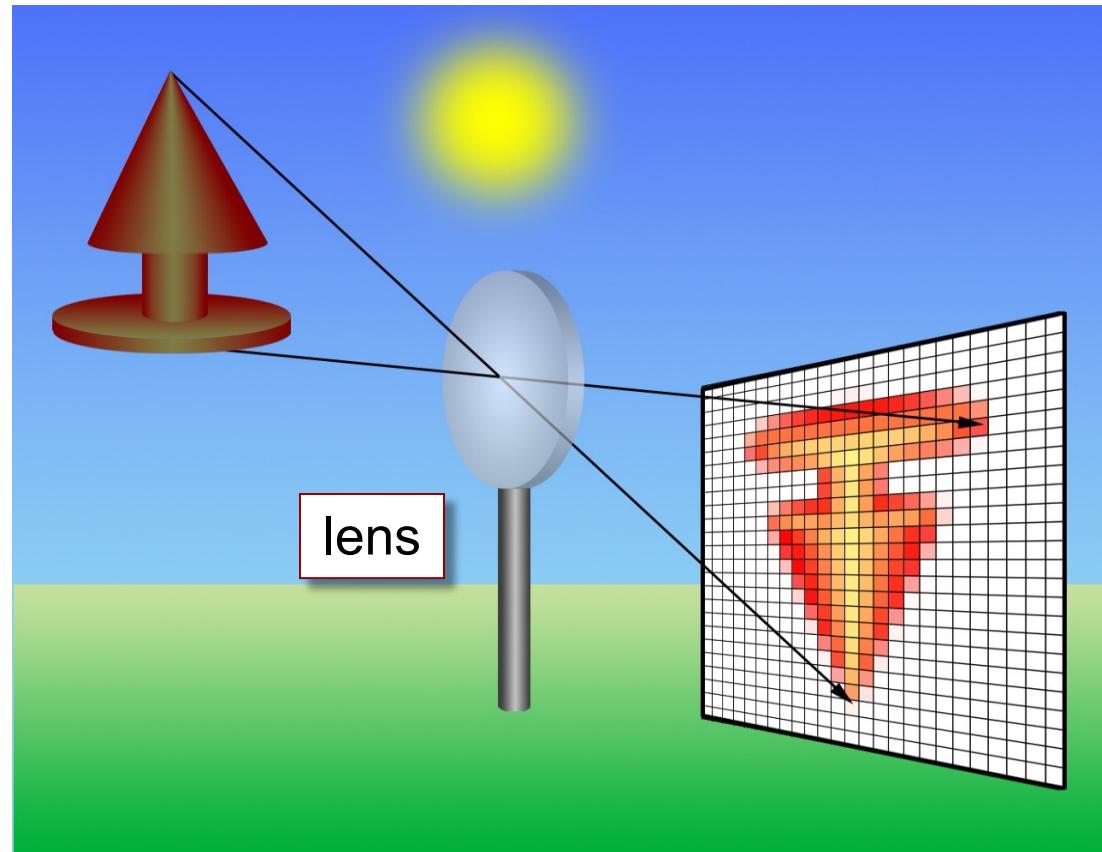
- Image processing is:
 - the operations we perform on an image to change or enhance it and make it suitable for further analysis,
 - so that useful information can be highlighted or get extracted from it.

- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

- Pinhole model

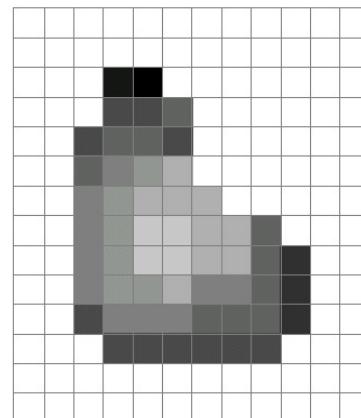
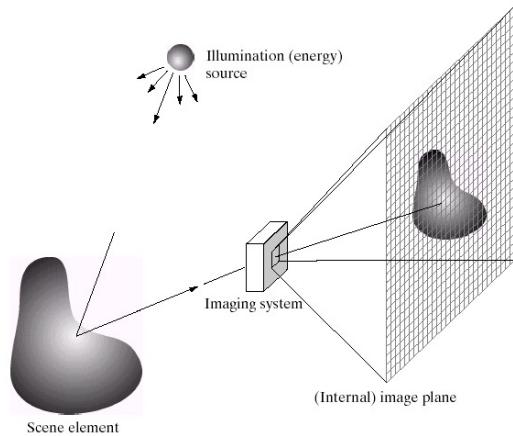


- Image Formation



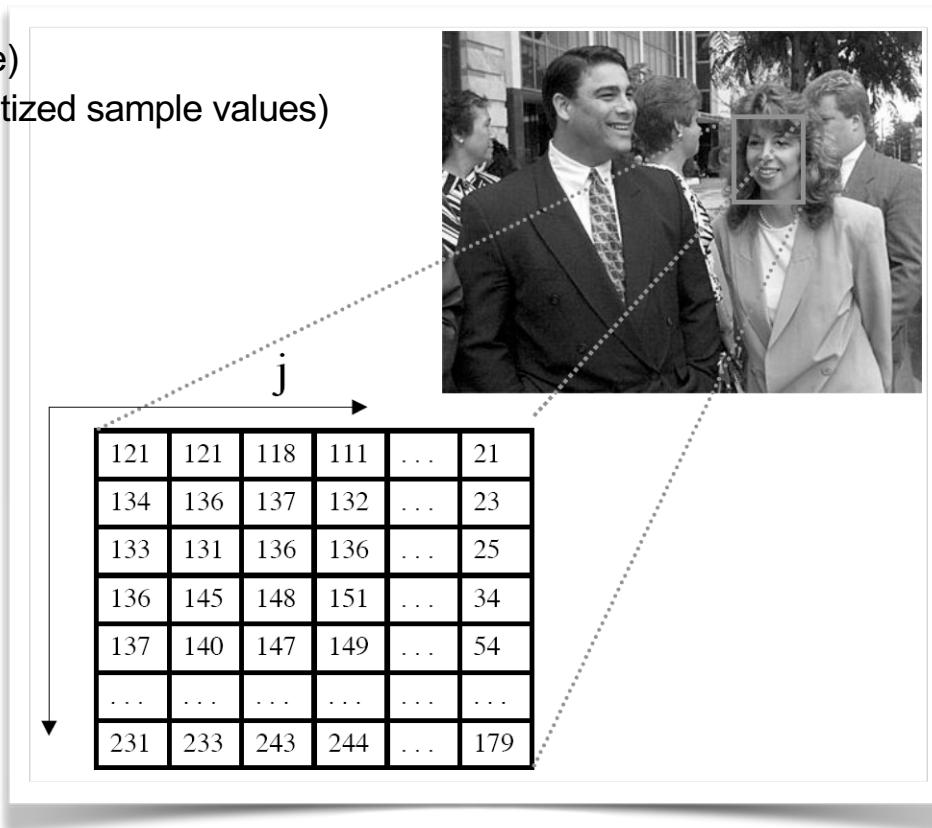
creditis: Richard Alan Peters II

- What is a (digital) image?
 - an image is a matrix/table (discretized 2D space)
 - each cell can take discrete/integer values (quantized sample values)

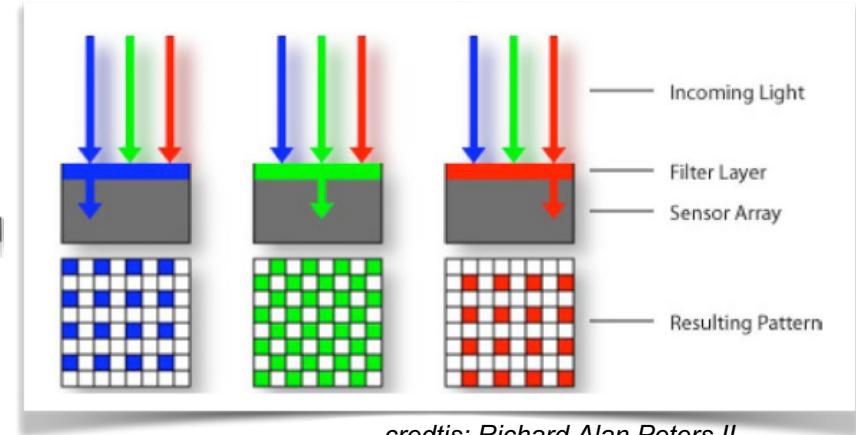
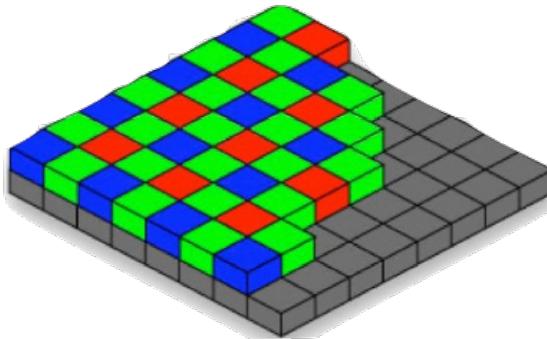
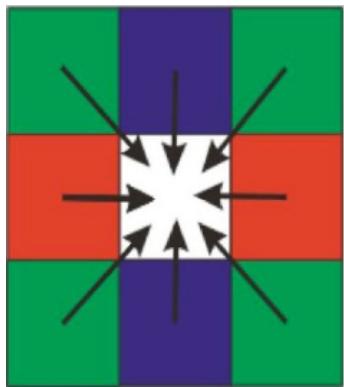
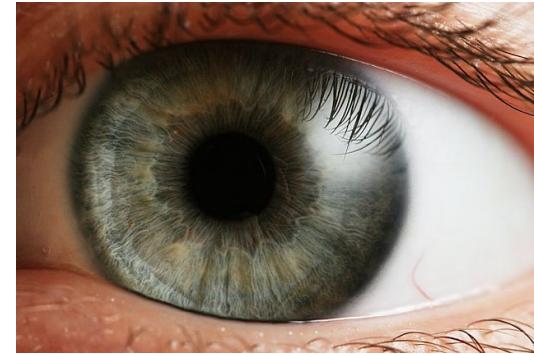


255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	20	0	255	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255	255	255
255	255	127	145	200	200	175	175	175	95	255	255	255	255
255	255	127	145	200	200	175	175	175	95	47	255	255	255
255	255	127	145	145	175	127	127	95	95	47	255	255	255
255	255	74	127	127	127	95	95	95	95	47	255	255	255
255	255	255	74	74	74	74	74	74	74	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255

- What is a (digital) image?
 - an image is a matrix/table (discretized 2D space)
 - each cell can take discrete/integer values (quantized sample values)

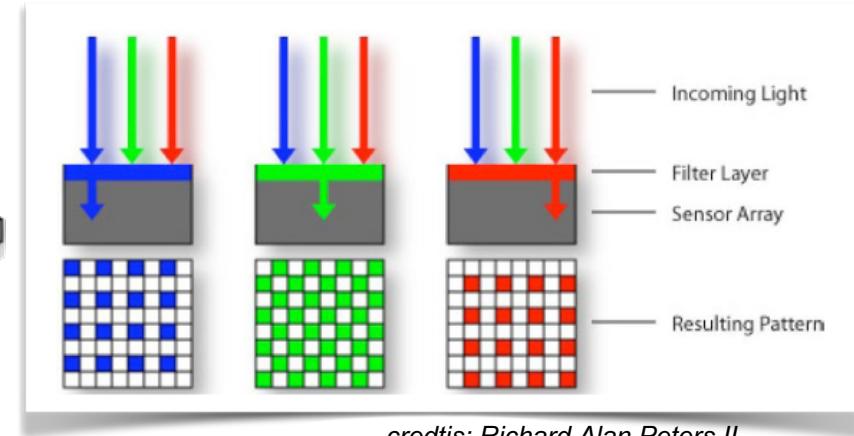
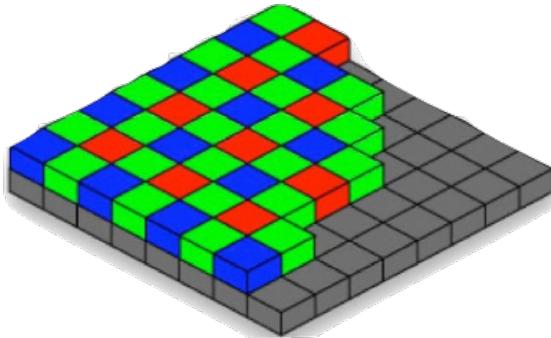
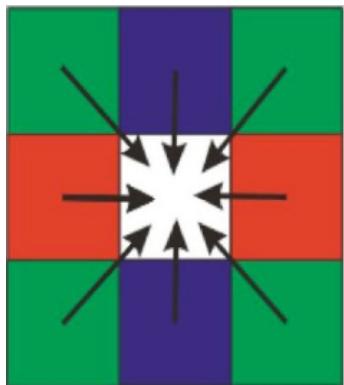
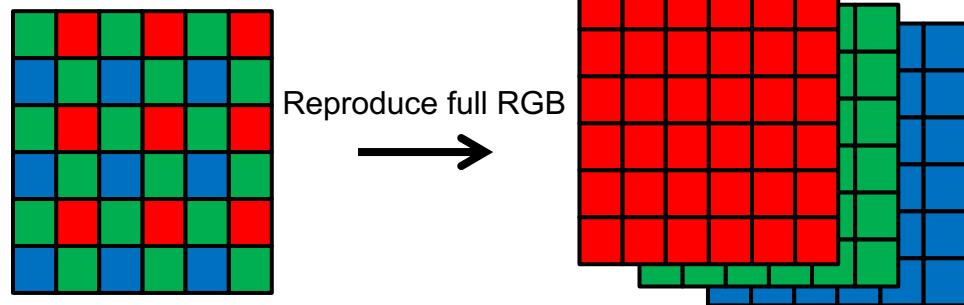


- Color Images
 - Bayer Color Filter Array



credit: Richard Alan Peters II

- Color Images
 - Bayer Color Filter Array

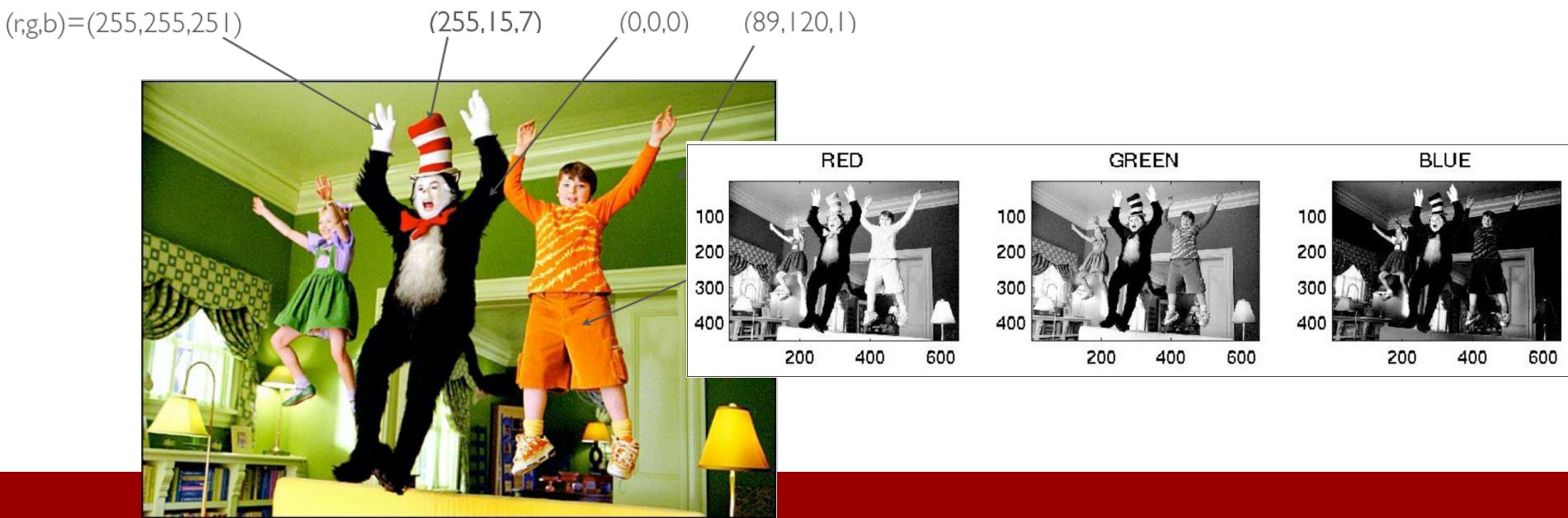


credit: Richard Alan Peters II

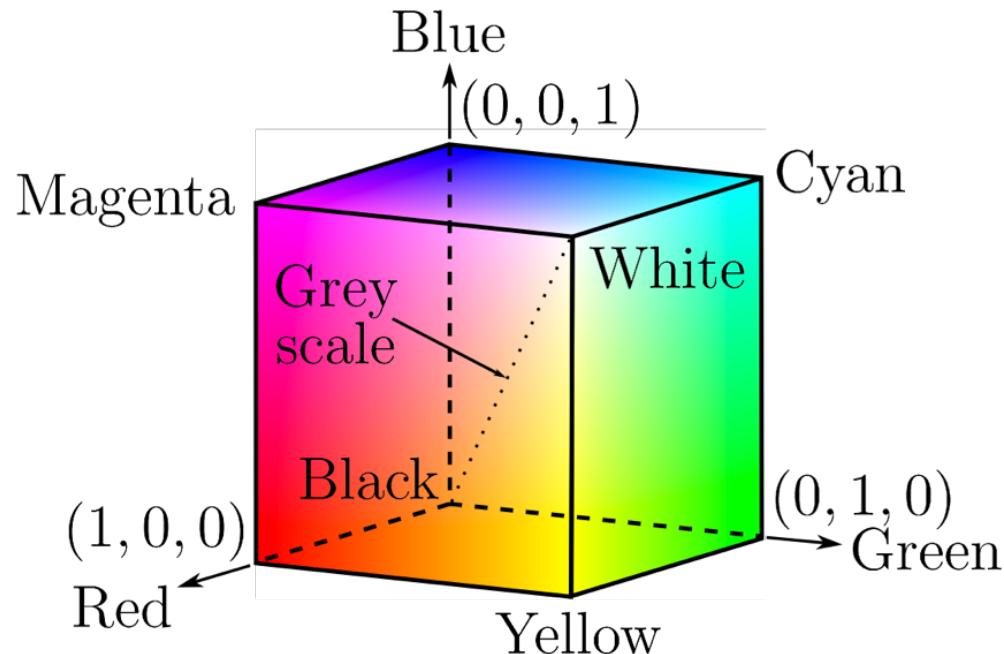
- Color Images
 - are composed of 3 2D images: R(x,y), G(x,y), B(x,y)
 - Each pixel (x,y) of these 3 images consists of values between 0 and 255
 - The values of a specific pixel (x_1, y_1) in the 3 images R, G, B describe the red-ness, green-ness and blue-ness of that particular pixel.



- Color Images
 - are composed of 3 2D images: R(x,y), G(x,y), B(x,y)
 - Each pixel (x,y) of these 3 images consists of values between 0 and 255
 - The values of a specific pixel (x_1, y_1) in the 3 images R, G, B describe the red-ness, green-ness and blue-ness of that particular pixel.



- RGB Color Space
 - all colors can be reproduced by mixing Red, Green and Blue



- Many other Color Spaces exist e.g. L*a*b* color space
- or HSI/HSV Color spaces

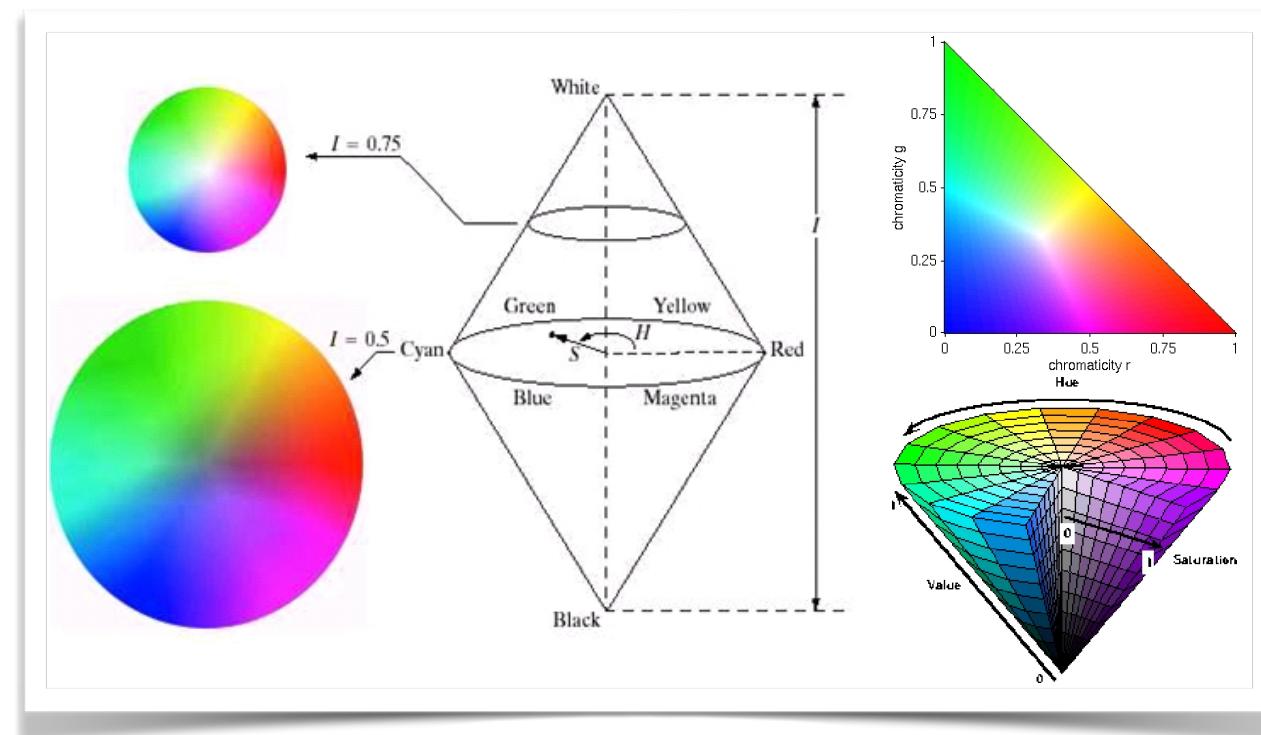
• **Hue:** color, chromatic information



• **Saturation:** purity, amount of white



• **Intensity:**



- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

Linear Filtering

- What are Linear Filters in Image Processing?
- What are they used for?

Noise reduction

- Nearby pixels are likely to belong to same object
 - thus likely to have similar color
- Replace each pixel by *average of neighbors*

0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

$$(0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0)/9 = \\ 6.66$$

Mean filtering

0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 0 + 0 + 0 + 20 + 10 + 40 + 0 + 0 + 20 + 10 + 0 + 0 + 0 + 30 + 20 + 10 + 0 + 0) / 25 = 6.8$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	0	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 10)/9 = 1.11$$

Mean filtering

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0
0	1	4	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 10 + 20)/9 = \\ 4.44$$

Mean filtering

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	0	0	0	0	0	0	0
0	0	10	20	20	0	20	10	40	0	0	0
0	10	20	30	50	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0	0
0	10	20	30	40	30	20	10	0	0	0	0
0	10	20	10	40	30	20	10	0	0	0	0
0	10	20	30	30	20	10	0	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0	0
0	0	0	10	10	10	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0
0	1	4	8	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

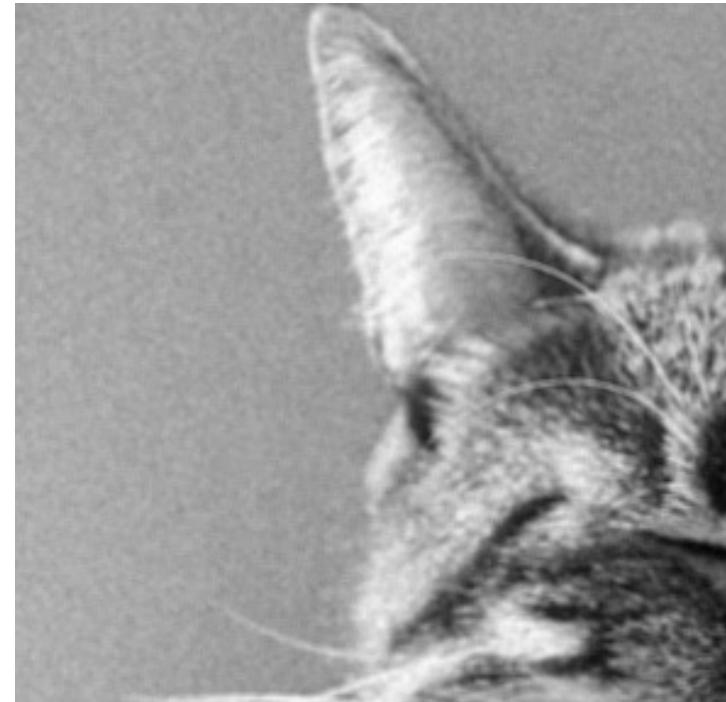
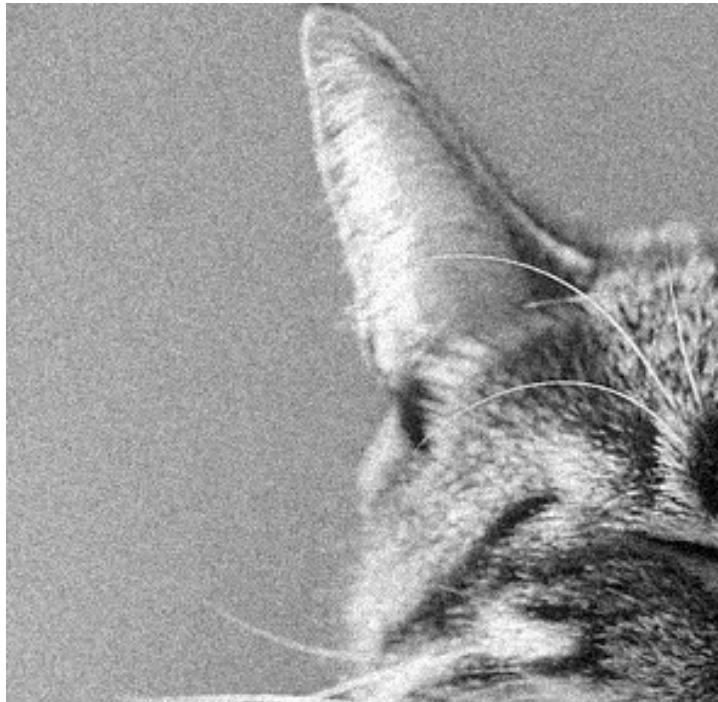
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = \\ 7.77$$

Mean filtering

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	10	10	10	0	0	0	0
0	0	10	0	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0
0	1	4	8	10	8	9	6	4	0	0
0	4	11	13	16	11	12	7	4	0	0
0	6	14	19	23	19	18	10	6	0	0
0	8	18	23	28	23	17	8	2	0	0
0	8	16	26	31	30	20	10	3	0	0
0	10	18	27	29	27	17	8	2	0	0
0	8	14	22	22	20	11	8	3	0	0
0	4	11	17	17	12	6	4	2	0	0
0	0	0	0	0	0	0	0	0	0	0

Noise reduction using mean filtering



Filters

- Filtering
 - Form a new image whose pixels are a combination of the original pixels
- Why?
 - To get useful information from images
 - E.g., extract edges or contours (to understand shape)
 - To enhance the image
 - E.g., to blur to remove noise
 - E.g., to sharpen to “enhance image” a la CSI

- Replace pixel by mean of neighborhood

10	5	3
4	5	1
1	1	7

Local image data

f



		4.1

Modified image data

$S[f]$

$$S[f](m, n) = \sum_{i=-1}^1 \sum_{j=-1}^1 f(m + i, n + j) / 9$$

A more general version

10	5	3
4	5	1
1	1	7



		7

Local image data

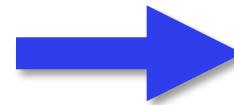
Kernel / filter

$$S[f](m, n) = \sum_{i=-1}^1 \sum_{j=-1}^1 w(i, j) f(m + i, n + j)$$

A more general version

0	10	5	7	0
5	11	6	8	3
9	22	4	5	1
2	9	14	6	7
3	10	15	12	9

Local image data



7

Kernel size = $2k+1$

$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(m + i, n + j)$$

A more general version

$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(m + i, n + j)$$

- $w(i,j) = 1/(2k+1)^2$ for mean filter
- If $w(i,j) \geq 0$ and sum to 1, *weighted mean*
- But $w(i,j)$ can be *arbitrary real numbers!*

Convolution and cross-correlation

- Cross correlation

$$S[f] = w \otimes f$$
$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(m + i, n + j)$$

- Convolution

$$S[f] = w * f$$
$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(\textcolor{red}{m - i}, \textcolor{red}{n - j})$$

Cross-correlation

1	2	3
4	5	6
7	8	9

w

1	2	3
4	5	6
7	8	9

f

$$1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 + 8*8 + \\ 9*9$$

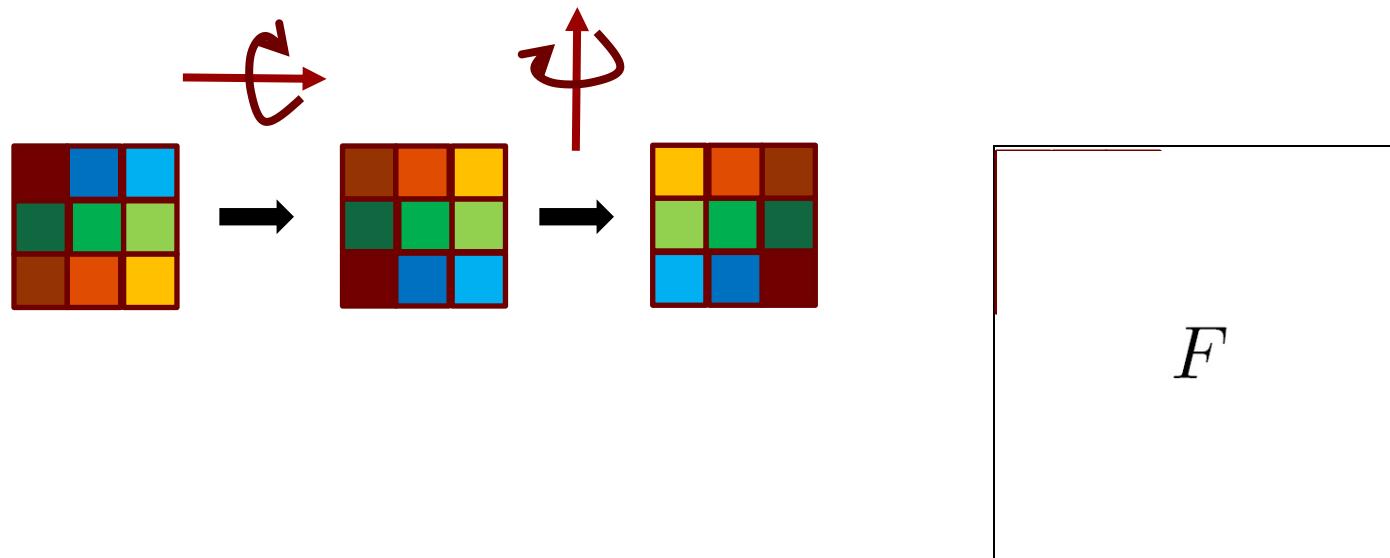
1	2	3
4	5	6
7	8	9

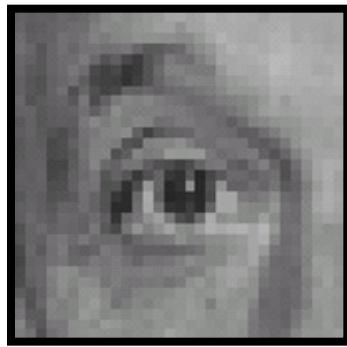
W

1	2	3
4	5	6
7	8	9

f

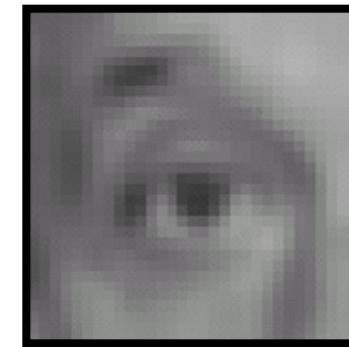
$$1*9 + 2*8 + 3*7 + 4*6 + 5*5 + 6*4 + 7*3 + 8*2 + 9*1$$





Original (f)

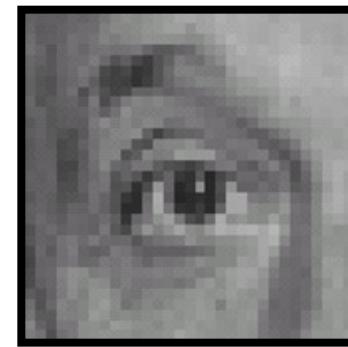
$$\ast \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \text{Kernel (k)}$$

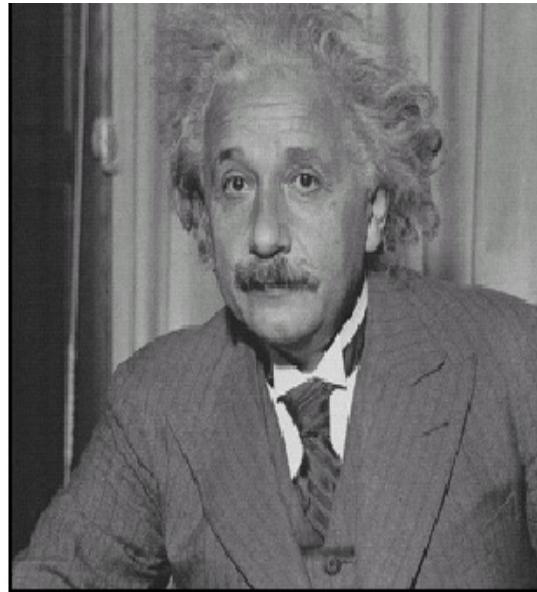


Blur (with a mean filter) (g)

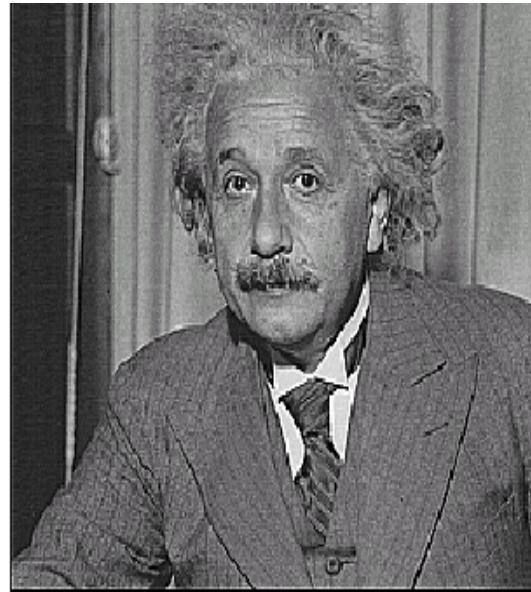
Original (f) $*$

0	0	0
0	1	0
0	0	0

Kernel (k) $=$ Identical image (g)



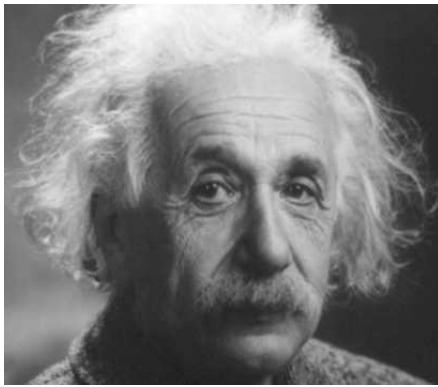
before



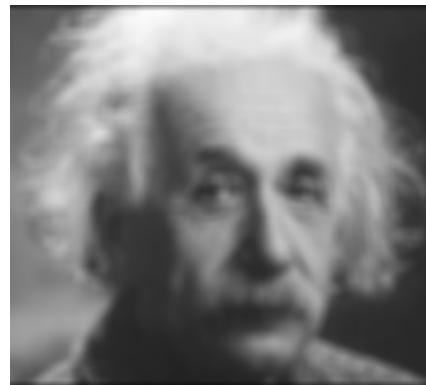
after

Sharpening

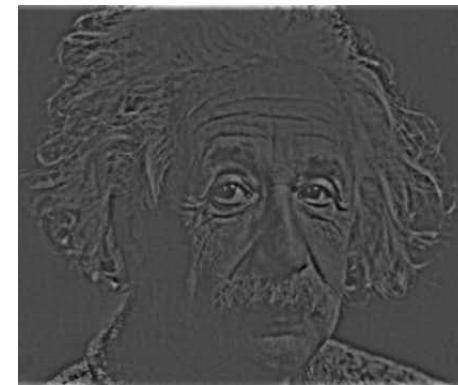
- What does blurring take away?



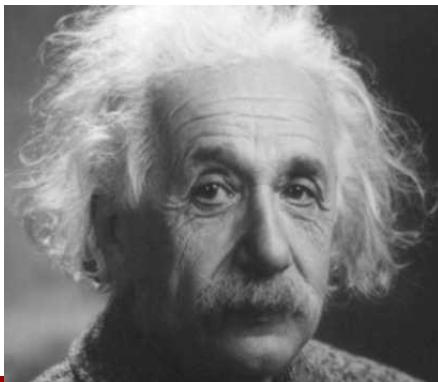
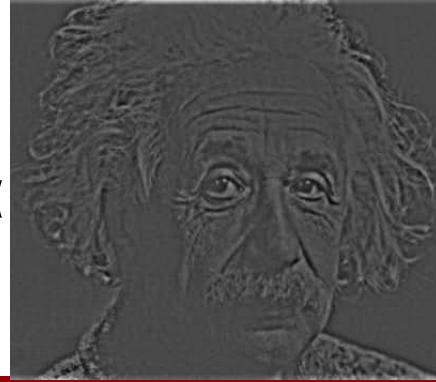
-



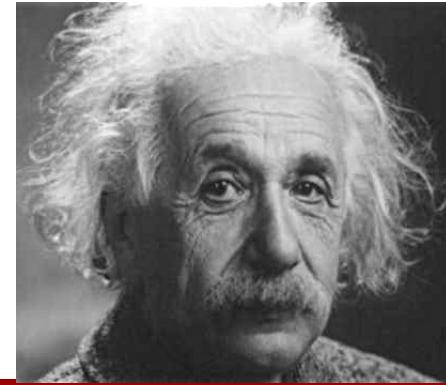
=

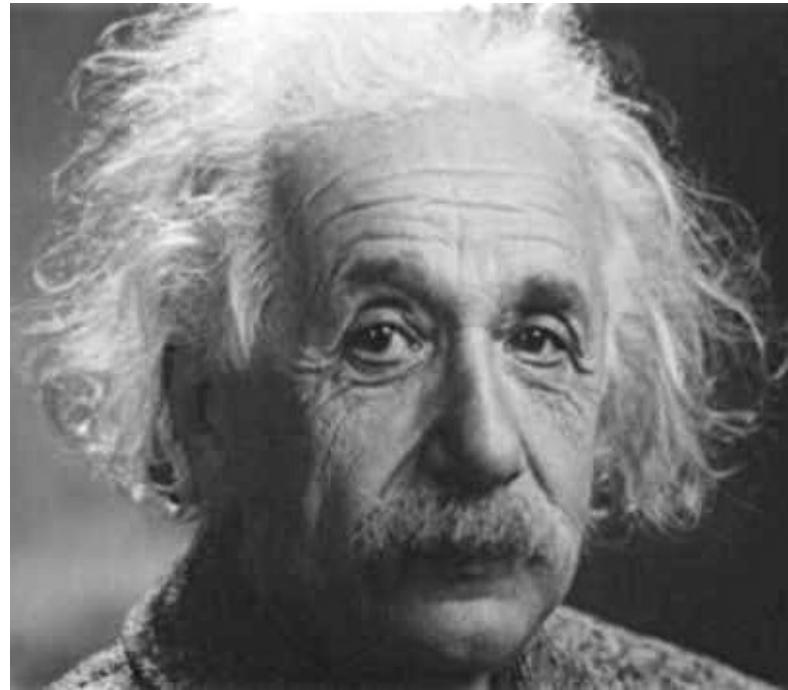


Let's add it back:

 $+ \alpha$ 

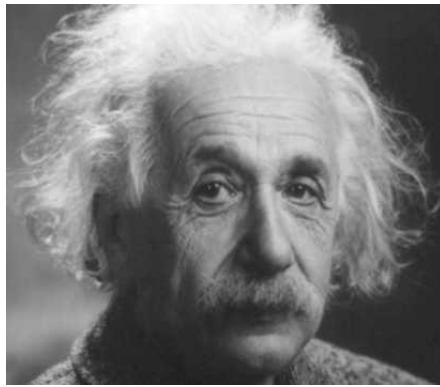
=



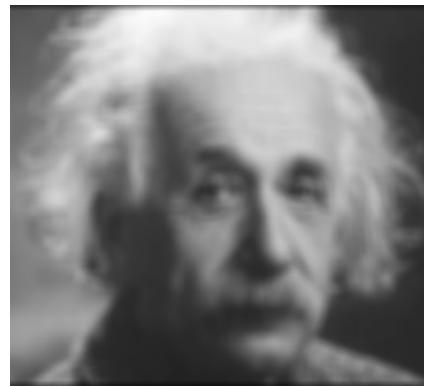


Sharpening

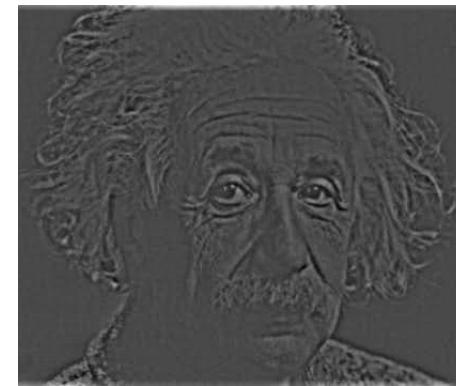
- What does blurring take away?



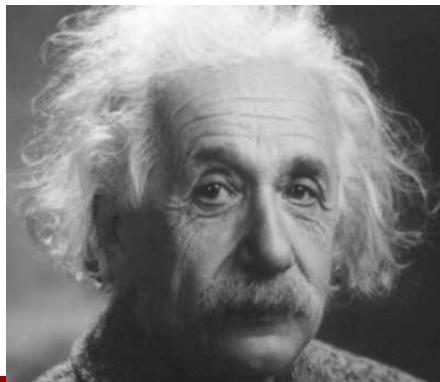
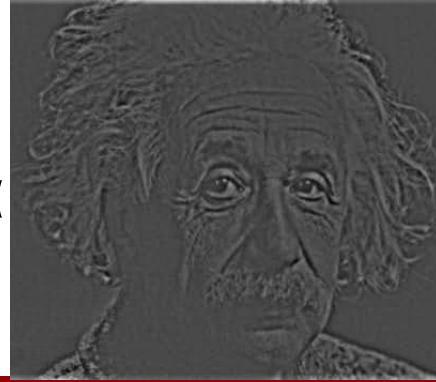
-



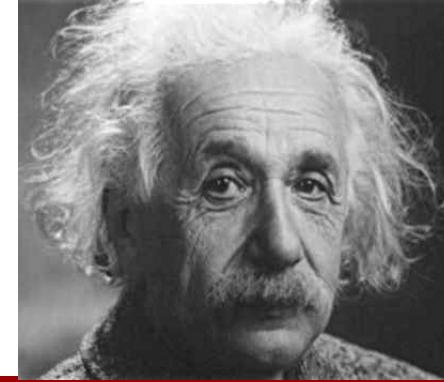
=



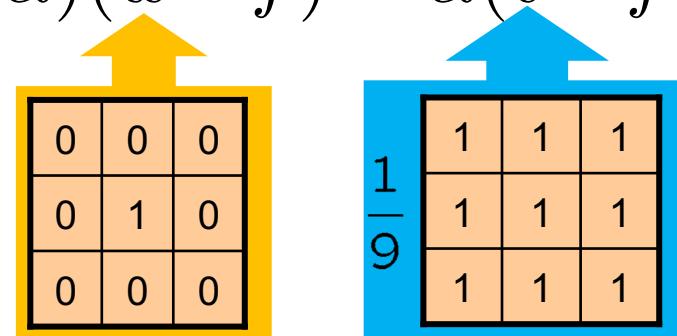
Let's add it back:

 $+ \alpha$ 

=



$$\begin{aligned}f_{sharp} &= f + \alpha(f - f_{blur}) \\&= (1 + \alpha)f - \alpha f_{blur} \\&= (1 + \alpha)(w * f) - \alpha(v * f)\end{aligned}$$



$$= ((1 + \alpha)w - \alpha v) * f$$

Sharpening filter



Original

$$\text{Original} * \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{array} - \frac{1}{9} \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right) = \text{Sharpened Image}$$



Sharpening filter
(accentuates
edges)

- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

Non-linear filters: Thresholding



$$g(m, n) = \begin{cases} 255, & f(m, n) > A \\ 0 & otherwise \end{cases}$$

Non-linear filters: Thresholding

- What if Threshold could be adaptive (instead of a pre-defined value)?
 - Otsu's method performs automatic image thresholding
 - The algorithm exhaustively searches for the threshold that minimizes the intra-class variance, defined as a weighted sum of variances of the two classes

Non-linear filters: Rectification

- $g(m,n) = \max(f(m,n), 0)$
- Crucial component of modern convolutional networks

Non-linear filters

- Sometimes mean filtering does not work



Non-linear filters

- Sometimes mean filtering does not work



Non-linear filters

- Mean is sensitive to outliers
- Median filter: Replace pixel by *median* of neighbors

Non-linear filters

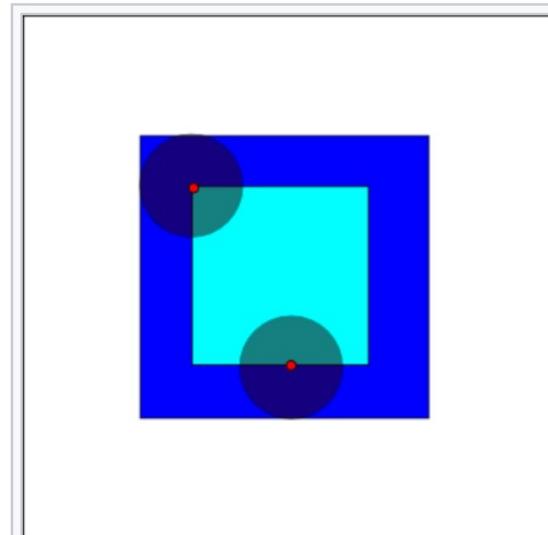


- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

Morphology

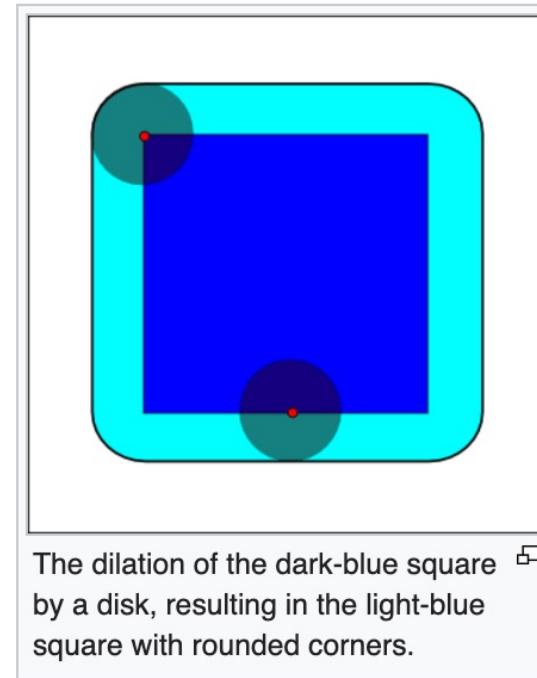
- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
 - Erosion
 - Dilation
 - Opening
 - Closing
 - All of them rely on convolution with a Structuring Element
 - » The Structuring Element can be a disk, rectangle, or of any other shape.
- There are also Grayscale Morphological Operations, apart from Binary ones.

- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
 - Erosion

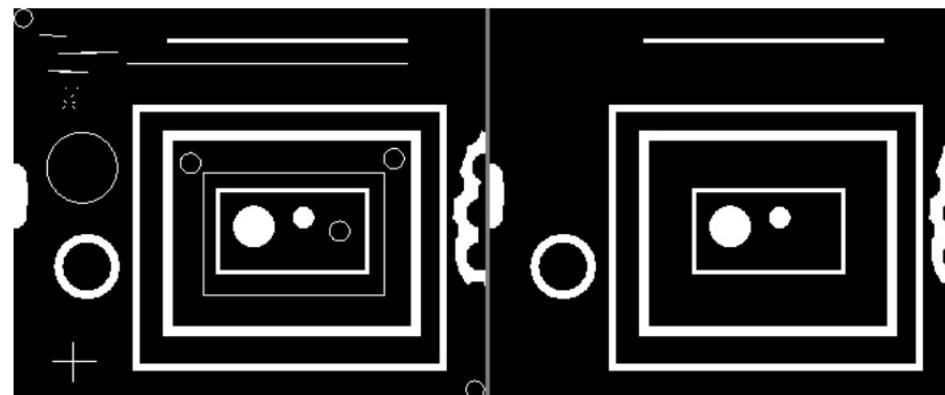


The erosion of the dark-blue square by a disk, resulting in the light-blue square.

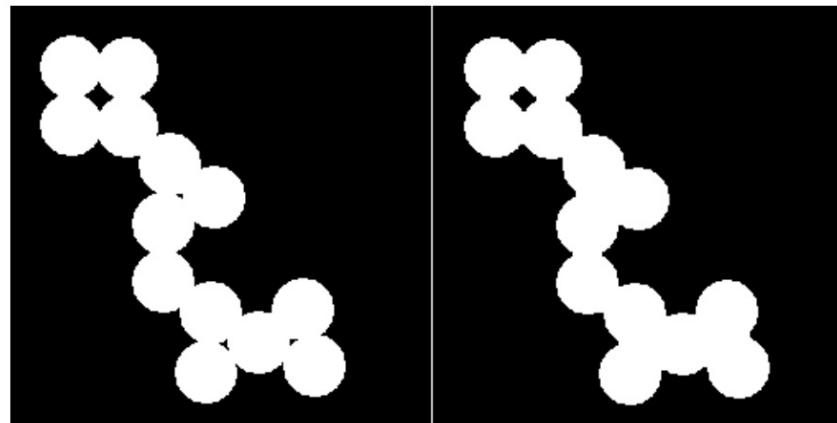
- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
 - Dilation



- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
 - Opening = Erosion and then Dilation
 - » Morphological opening is useful for removing small objects from an image while preserving the shape and size of larger objects in the image.



- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
 - Closing = Dilation and then Erosion
 - » Morphological closing is useful for filling small holes from an image while preserving the shape and size of the objects in the image.



- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

Connected Components Analysis

- Connected Component Analysis checks each pixel of an image for connectivity with its neighboring pixels.
- Each group of connected pixels are considered as one component and are assigned the same label.



- Connectivity is established if two neighboring pixels share same or similar intensity/color value.
- The method works both binary, grayscale, or color images
- Different measures of connectivity are possible (4-connectivity, or 8-connectivity are typical)

Summary

- We discussed about what Image Processing is.
- We learned about :
 - Color
 - RGB, other Color Spaces
 - Linear Filtering
 - convolution, cross-correlation
 - Non-linear Filters / Thresholding
 - Morphology
 - Connected Components Analysis

Lazaros Nalpantidis

Image Processing

Perception for Autonomous Systems

Lecture 1 - Image Processing (01/02/2021)

Introduction Questions

1. What is "Perception"?
 2. What is an "Autonomous System"?
 3. Why Autonomous Systems need Perception?
-

Outline/Content:

- What is Image Processing?
- Image
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

Topics and Reading Sources:

- Color
 - Book A, Chapter 2.3.2
- Linear Filtering
 - Book A, Chapters 3.1, 3.2
- Non-linear Filters, Morphology, Thresholding and Connected Components Analysis
 - Book A, Chapter 3.3

What is Image Processing?

Image processing are operations we perform on an image to change or enhance it and make it suitable for further analysis. So that useful information can be highlighted or get extracted from it.

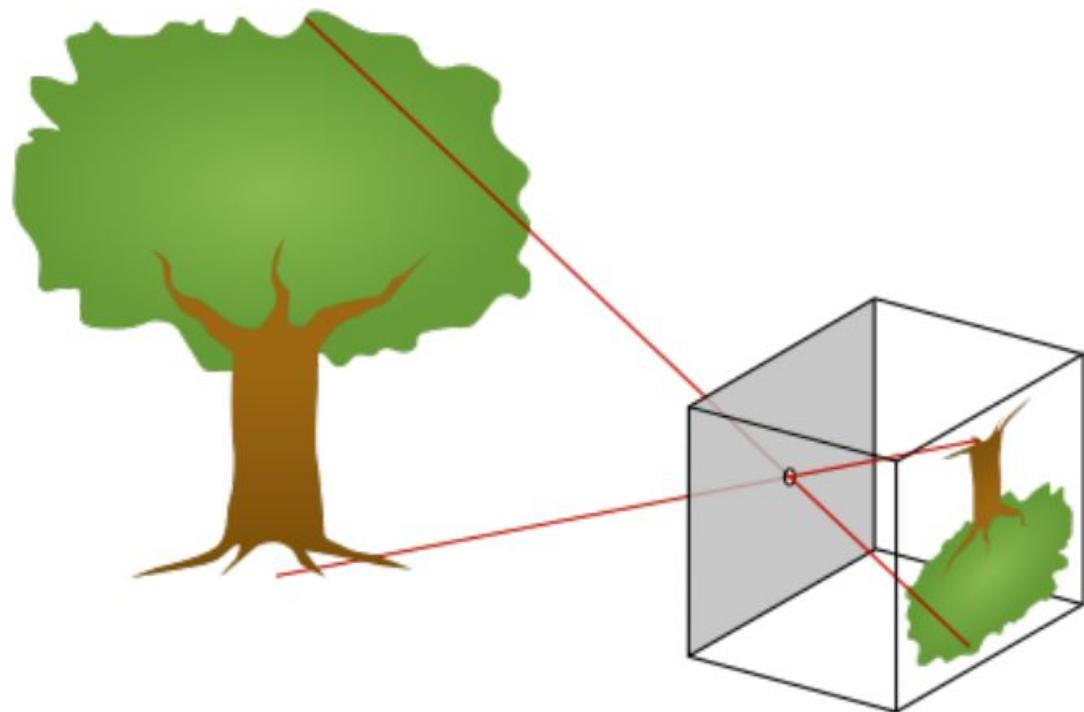
MAYBE ADD MORE POINTS HERE

Image

Pinhole model [Youtube Link](#)

The pinhole model describes the process of reducing the incidence of light to such an extent that the section of the image in front of you can be seen (upside down). This process is used in all given cameras and is known

as "aperture".



Pixels

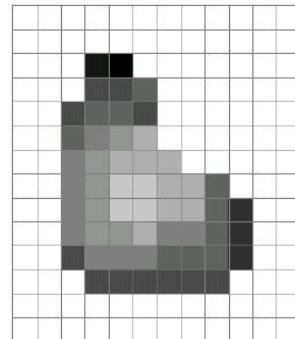
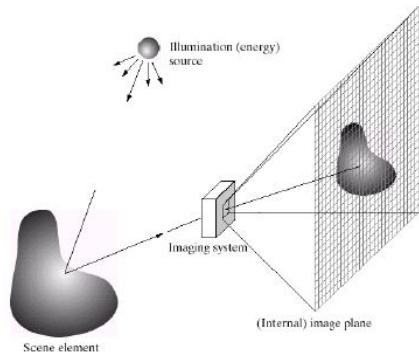
Pixels are the raw building blocks of an image. Each image consists of a set of pixels. There is no finer granularity than the pixel. Normally, a pixel is considered to be the "color" or the "intensity" of the light that appears at a particular location in our image.

Most pixels are represented in two ways:

- Grayscale: single channel
- Color: three channels

What is a (digital) image?

The resolution of an image is given by the representation of columns_number (width) x rows_number (height) of pixels. For example, an image with a resolution of 1,000 x 750 is 1,000 pixels wide and 750 pixels high. We can think of an image as a (multi-dimensional) matrix with the form W x H x D (dimensions/channels). In this case, our matrix has 1,000 columns with 750 rows. Overall, there are $1,000 \times 750 = 750,000$ total pixels in this image. Each cell can take discrete/integer values (quantized samples ranging from 0 to 255).



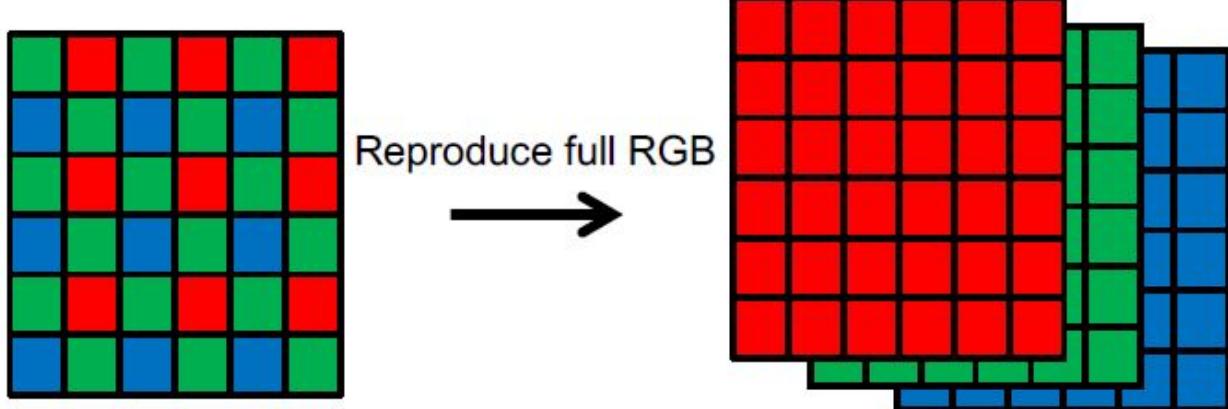
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255	255	255
255	255	127	145	145	175	127	127	95	47	255	255	255	255
255	255	74	127	127	127	95	95	95	47	255	255	255	255
255	255	74	74	74	74	74	74	74	74	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255

HINT: Image processing libraries such as OpenCV and scikit-image represent RGB images as multidimensional NumPy arrays with shape (height, width, depth). So, height and width are reversed.

Color

Bayer Color filter

The Bayer pattern places green filters over half of the sensors (in a checkerboard pattern), and red and blue filters over the remaining ones. The reason that there are twice as many green filters as red and blue is because the luminance signal is mostly determined by green values and the visual system is much more sensitive to high frequency detail in luminance than in chrominance.



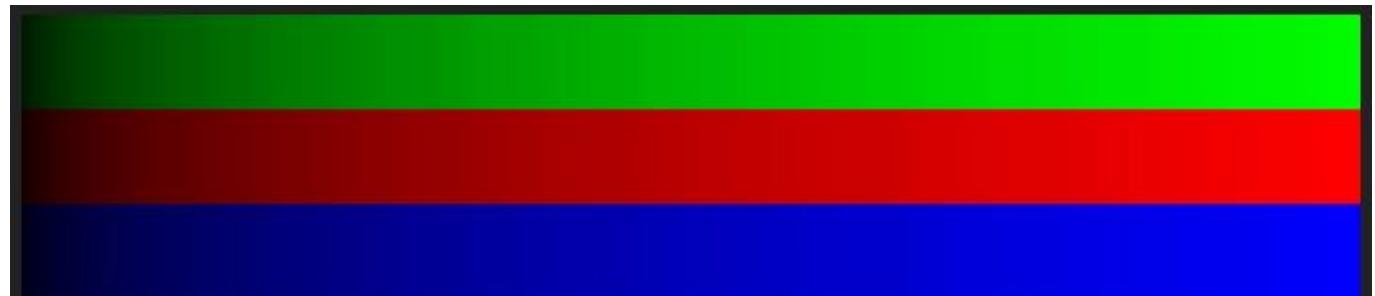
Grayscale

In a grayscale image, each pixel is a scalar (single-channel) value between 0 and 255, where zero corresponds to "black" and 255 to "white". Values in between are different shades of gray, with values closer to 0 being darker and values closer to 255 being lighter.



RGB

In a colored image, each pixel is a representation of three 2D images: $R(x,y)$, $G(x,y)$, $B(x,y)$ with values ranging from 0 to 255. The values of a specific pixel (x,y) in the 3 images R, G, B describe the red-ness, green-ness and blue-ness of that particular pixel. Meaning 0 indicates the absence of the color, therefore "black" whereas 255 indicates the full saturation of that color. Given our three Red, Green, and Blue values, we can combine them into an RGB tuple in the form (red, green, blue). This tuple represents a given color in the RGB color space.



RGB Color Space (additive color space)

- A image consists of 25% red, 50% green and 25% blue pixels
- all colors can be reproduced by mixing Red, Green and Blue
- the more of each color is added, the brighter the pixel becomes and closer to white
- There is a maximum of 16.777.216 unique colors in the RGB color space ($256 \times 256 \times 256$)

Drawbacks of thr RGB Color Space:

- It's additive nature makes it a bit unintuitive for humans to easily define shades of color without using a "color picker" tool
- It doesn't mimic how humans perceive color

However, despite these drawbacks, nearly all images you'll work with will be represented (at least initially) in the RGB color space.

Many other Color Spaces exist e.g. L*a*b* color space or HSI/HSV (Hue-Saturation-Value) Color spaces.

Linear Filtering

Questions to ask are:

- What are Linear Filters in Image Processing?
- What are they used for?

Filters

Filters form a new image whose pixels are a combination of the original pixels - Why?

- To get useful information from images
 - E.g., extract edges or contours (to understand shape)
- To enhance the image
 - E.g., blur to remove noise
 - E.g., sharpen to enhance image

Application of Filters: When using a filter we have two matrices. One big matrix representing the image and a small kernel or convolution matrix representing the filter of choice. Essentially, this tiny kernel sits on top of the image matrix and slides from left-to-right and top-to-bottom, applying a mathematical operation (i.e., a convolution) at each (x;y)-coordinate of the original image.

SIDE NOTE: The Initial starting position within the pixel grid is the left upper corner at (0, 0).

Usually all the kernel cells need to have valid values. Meaning, we don't start at the corner of the images. This will introduce some loss as we are shrinking the image. There are ways to counter that, like zero padding etc.

Kernel

As already mentioned before we're sliding the kernel from left-to-right and top-to-bottom along the original image. At each (x,y)-coordinate of the original image, we stop and examine the neighborhood of pixels located at the center of the image kernel. We then take this neighborhood of pixels, convolve them with the kernel, and obtain a single output value. The output value is stored in the output image at the same (x,y)-coordinates as the center of the kernel.

Odd kernel sizes are required to ensure that there is a valid integer (x,y)-coordinate at the center of the image. Let's take a 3x3 matrix for example. Here you have the center at (1,1 => valid integer) if we start at the left upper corner. Whereas a 2x2 matrix mathematically would have its center point at (0.5, 0.5) which is no valid expression. And visually it's even more obvious that there is no center point in a 2x2 matrix.

131	162	232	84	91	207
104	-1	0	+1	237	109
243	-2	0	+2	135 → 126	
185	-1	0	+1	61	225
157	124	25	14	102	108
5	155	16	218	232	249

Convolution and cross-correlation

Application of Convolution:

- Blurring:
 - average/mean smoothing
 - gaussian smoothing
- Edge detection:
 - Laplacian
 - Sobel
 - Scharr
 - Prewitt
- Sharpen
- other filters

SIDE NOTE: To sharpen an image, you blur the image and subtract it from the original image and then add that result onto the original image again.

Convolutions are one of the most critical, fundamental building-blocks in computer vision and image processing. They're actually quite easy to understand. It's an element-wise multiplication of two matrices followed by a sum.

What's the difference between Convolution and cross-correlation?

Mathematically it's the same equation where simply a sign change took place. This effects the way we access the coordinates of the image (i.e., we don't have to "flip" the kernel relative to the input when applying cross-correlation).

Therefore, speaking about convolution although using cross-correlation is by design.

Cross correlation

$$S[f] = w \otimes f$$

$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j)f(m + i, n + j)$$

Convolution

$$S[f] = w * f$$

$$S[f](m, n) = \sum_{i=-k}^k \sum_{j=-k}^k w(i, j)f(\textcolor{red}{m - i}, \textcolor{red}{n - j})$$

Putting everything together

A convolution requires three components:

1. An input image.
2. A kernel matrix that we are going to apply to the input image.
3. An output image to store the output of the image convolved with the kernel.

Convolution (or cross-correlation) is actually very easy. All we need to do is:

1. Select an (x;y)-coordinate from the original image.
2. Place the center of the kernel at this (x;y)-coordinate.
3. Take the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. The sum of these multiplications is called the kernel output.
4. Use the same (x;y)-coordinates from Step #1, but this time, store the kernel output at the same (x;y)-location as the output image.

Below you can find an example of convolving (denoted mathematically as the **star** operator) a 3x3 region of an image with a 3x3 kernel used for blurring:

$$O_{i,j} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 93 & 139 & 101 \\ 26 & 252 & 196 \\ 135 & 230 & 18 \end{bmatrix} = \begin{bmatrix} 1/9 \times 93 & 1/9 \times 139 & 1/9 \times 101 \\ 1/9 \times 26 & 1/9 \times 252 & 1/9 \times 196 \\ 1/9 \times 135 & 1/9 \times 230 & 1/9 \times 18 \end{bmatrix}$$

Therefore,

$$O_{i,j} = \sum \begin{bmatrix} 10.3 & 15.4 & 11.2 \\ 2.8 & 28.0 & 21.7 \\ 15.0 & 25.5 & 2.0 \end{bmatrix} \approx 132.$$

After applying this convolution, we would set the pixel located at the coordinate (i, j) of the output image O to $O_{i, j} = 132$. That's all there is to it! Convolution is simply the sum of element-wise matrix multiplication between the kernel and neighborhood that the kernel covers of the input image.

Mean filtering

Objective: Introduce blur to smoothen the image and remove noise.

Take the sum of each cell within the filter size and divide by the number of cells. For example:

3x3 Filter Size

x1	x2	x3
0	0	0
10	40	0
10	0	0

$$\Rightarrow (0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0) / 9 = 6.66 = 7$$

Pixels only take integer values. Therefore, 6.66 is being rounded to 7.

There are two kinds of operations to apply a filter on a pixel. There are point operators or point processes that manipulate each pixel independently of its neighbors (Brightness, contrast, color correction/transformation). And then there are area-based operators, where each new pixel value depends on the value of its neighbors.

Non-linear filters: Thresholding

By using non-linear filters we set a threshold which acts as our if condition . For example, we could set the threshold to 200 and set everything to 255 if true and 0 otherwise. This will result in a pure black & white image.

What if the threshold could be adaptive (instead of a pre-defined value)?

- Otsu's method performs automatic image thresholding

- The algorithm exhaustively searches for the threshold that minimizes the intra-class variance, defined as a weighted sum of variances of the two classes.

Non-linear filters: Rectification

- $g(m,n) = \max(f(m,n), 0)$
- crucial component of modern convolutional networks

Non-linear filters

- Sometimes mean filters does not work
 - mean is sensitive to outliers
 - median filter: Replace pixel by median of neighbors
-

Morphology [PylImageSearch](#), [Youtube](#)

The most common binary image operations are called Morphological Operations

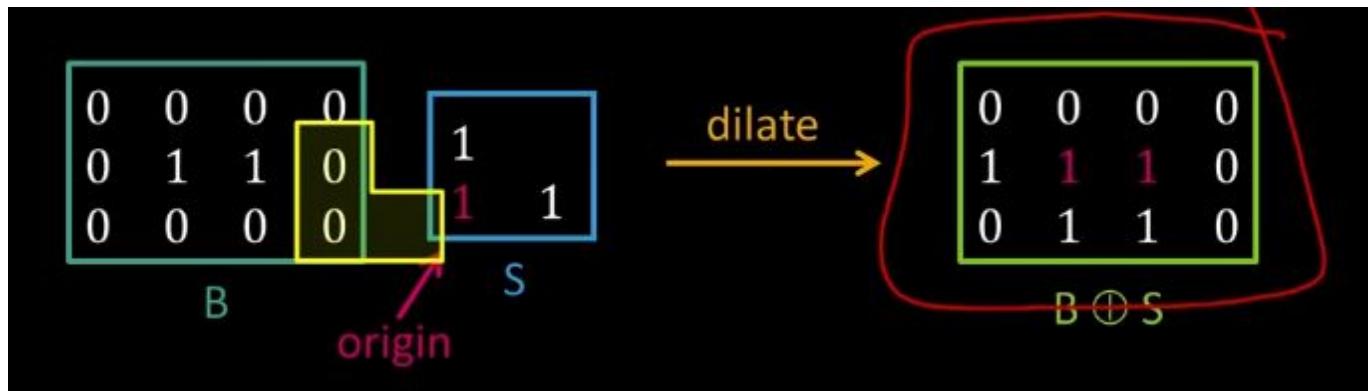
Basic Morphological Operations:

- Dilation
 - Increase white pixels, decrease black pixels
 - Case 1: Perfect match replace origin with 1
 - Case 2: Partial match replace origin with 1
 - Case 3: No match replace origin with 0
- Erosion
 - Decrease white pixels, increase black pixels
 - Case 1: Perfect match replace origin with 1
 - Case 2: Partial match replace origin with 0
 - Case 3: No match replace origin with 0
- Opening
 - Erosion then Dilation
- Closing
 - Dilation then Erosion

Structuring Element A shape mask used in basic morphological ops.

- Any shape, size that is digitally representable (Box, hexagon, disk, rectangle, etc.)
- With a defined origin

Application: The structuring element represents a pixel shape matrix with ones and usually has its origin (highlighted) in the center of the shape. Then slide the shape from left to right, top to bottom and enable or disable pixels at the current position of the origin whenever there is any pixel match depending on the morphological operations.

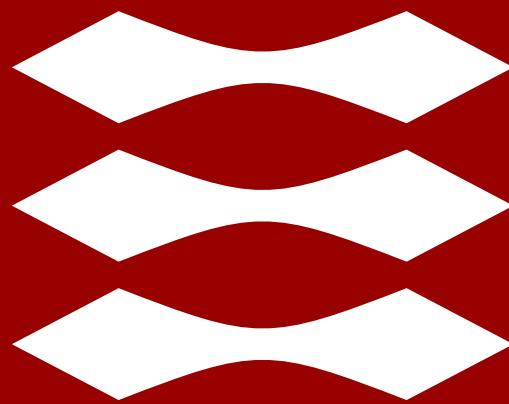


There are also Grayscale Morphological Operations, apart from Binary ones.

Connected Components Analysis

- Connected Component Analysis checks each pixel of an image for connectivity with its neighboring pixels
- Each group of connected pixels are considered as one component and are assigned to the same label
- Connectivity is established if two neighboring pixels share same or similar intensity/color value.
- The method works on binary, grayscale, or color images.
- Different measures of connectivity are possible (4-connectivity, or 8-connectivity are typical)

DTU



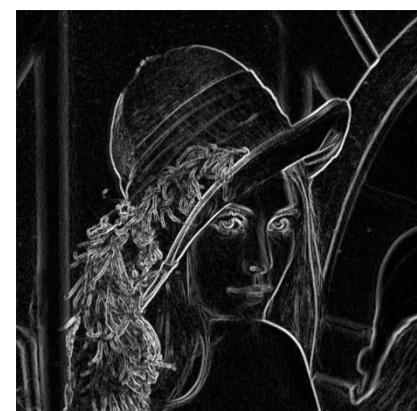
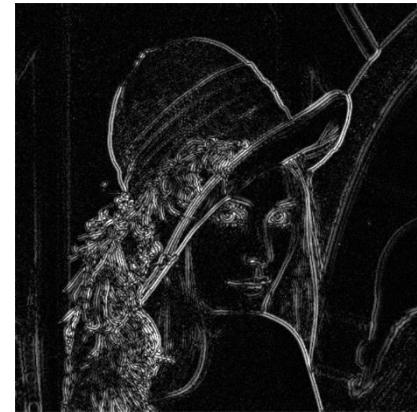
Perception for Autonomous Systems 31392:

Edge Detection

Lecturer: Evangelos Boukas—PhD

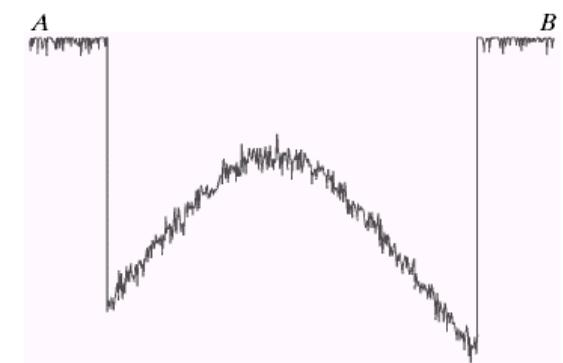
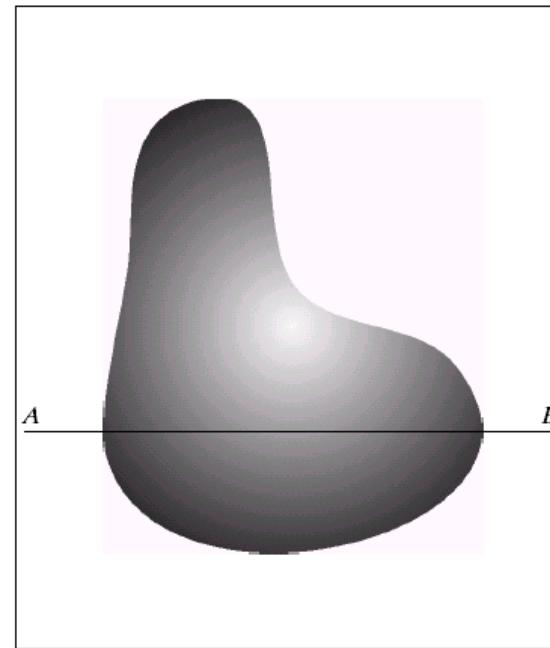
Edge Detection

- What is an Edge?
- Image Derivative
- Gradient
- Sobel
- Laplacian
- Canny



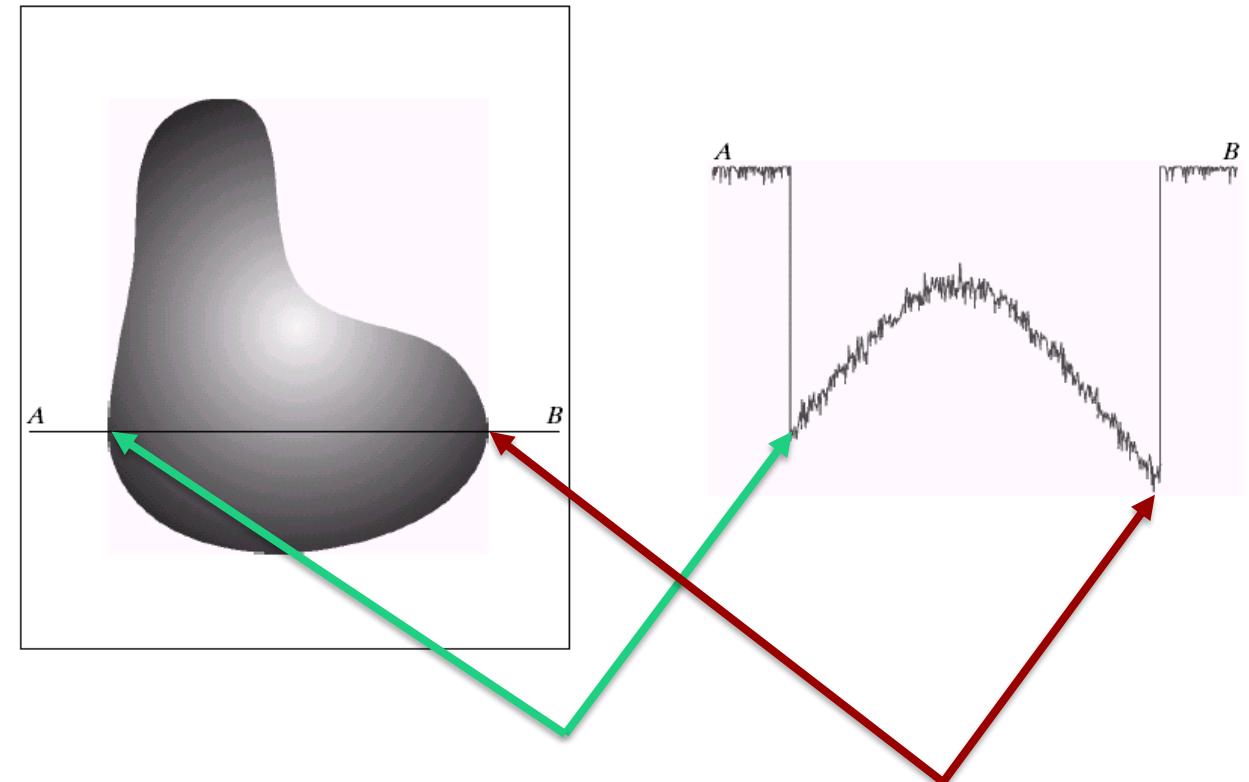
Edge Detection

- What is and edge?



Edge Detection

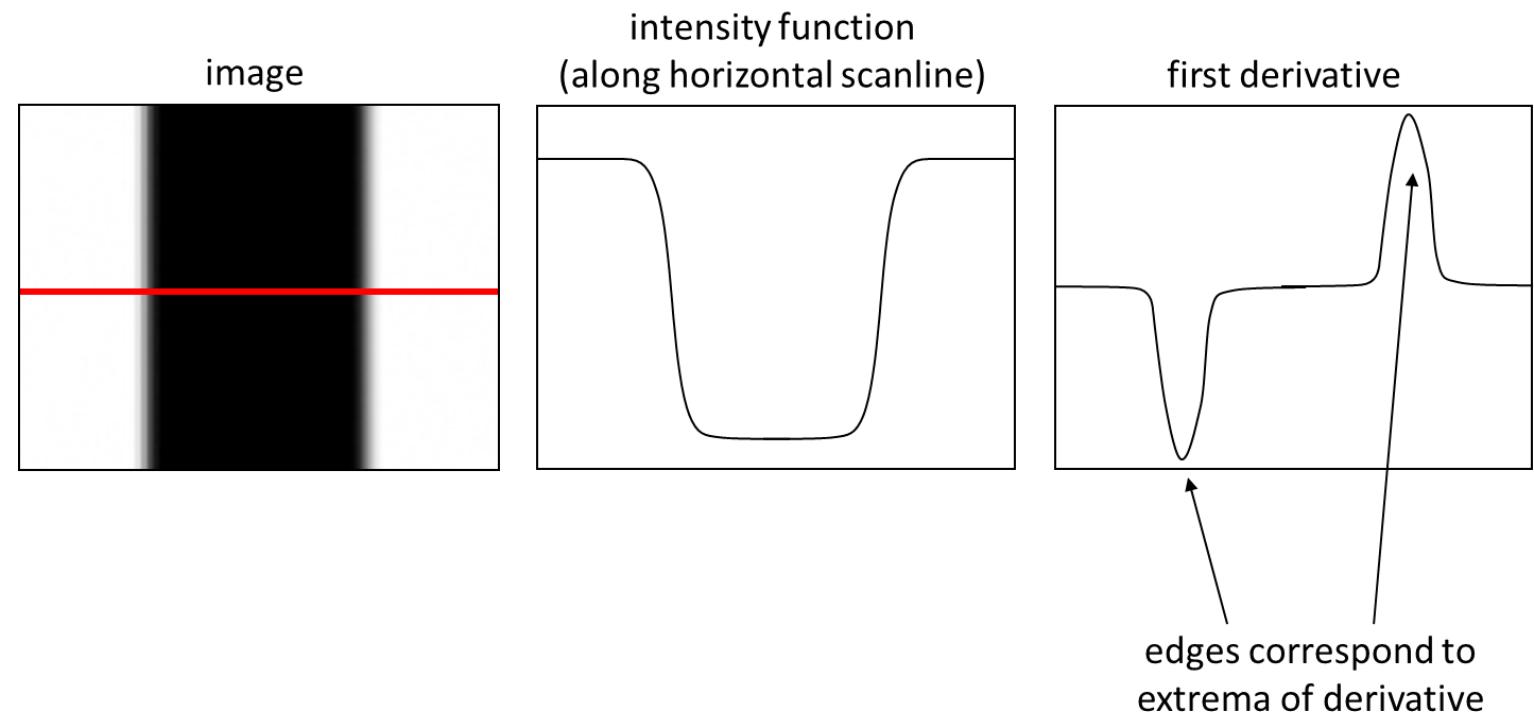
- What is and edge?



Edge Detection

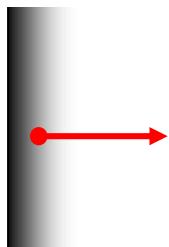
- What is and edge?
- Derivative of an Image:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

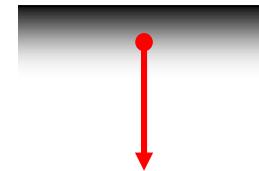


- The gradient is a vector which points in the direction of most rapid change in intensity:

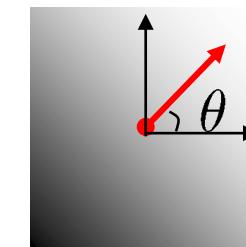
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0 \right]$$

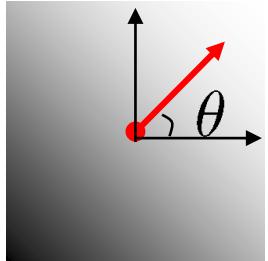


$$\nabla f = \left[0, \frac{\partial f}{\partial y} \right]$$



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

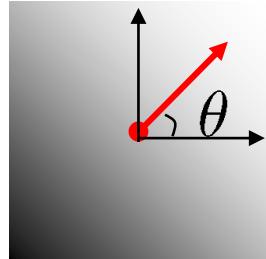
Gradient Simplified



$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \longrightarrow \frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

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

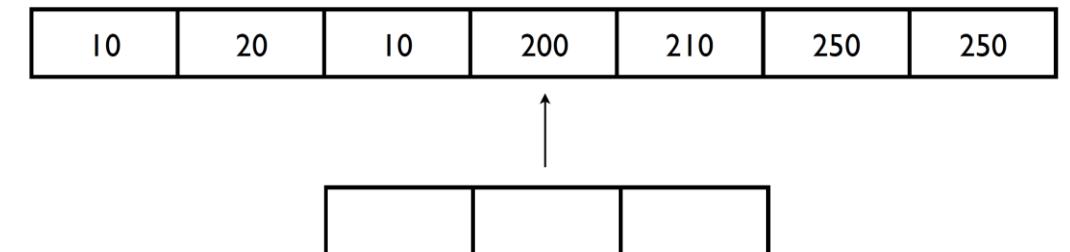
Gradient Simplified



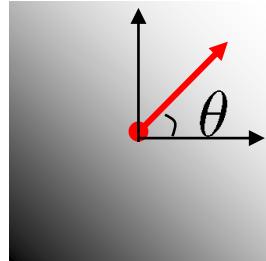
$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \longrightarrow \frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

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

- How would you define the 1D filter of the gradient:



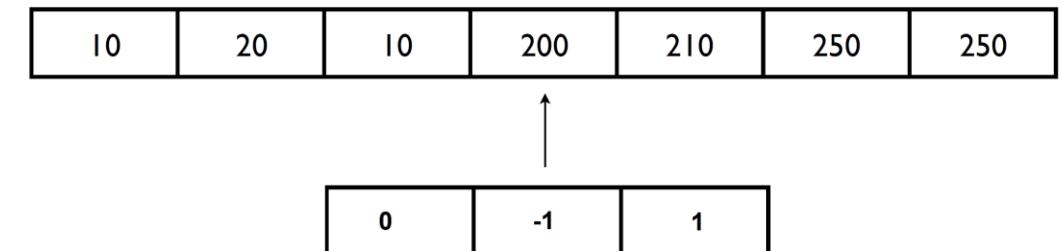
Gradient Simplified



$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \longrightarrow \frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

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

- How would you define the 1D filter of the gradient:



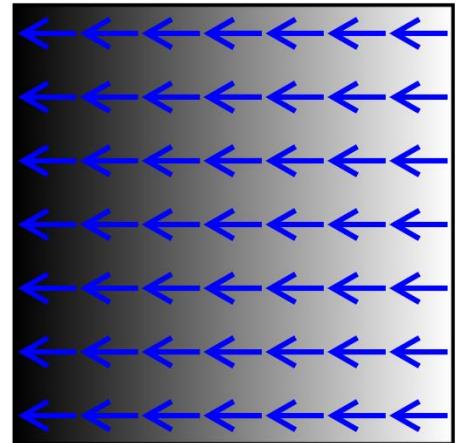
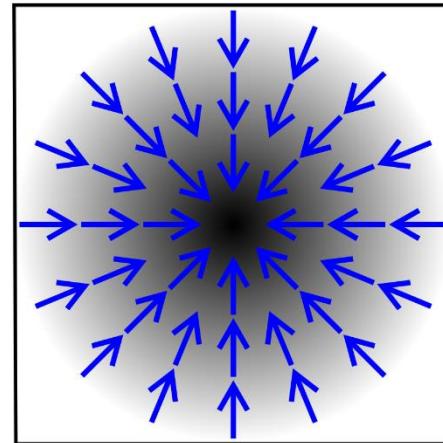
Gradient Simplified

- The gradient is defined by its orientation:

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

- and magnitude:

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



black: 0
white: 1

- Practically:

$$g_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

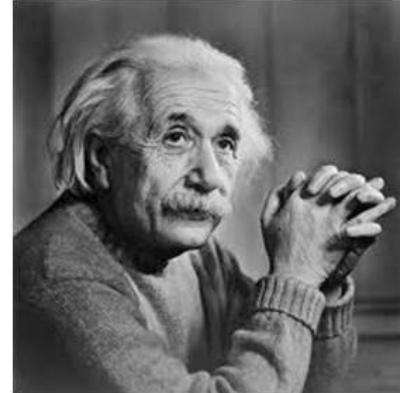
$$g_y = \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

- Magnitude:

$$g = \sqrt{g_x^2 + g_y^2}$$

- Orientation:

$$\Theta = \tan^{-1} \left(\frac{g_y}{g_x} \right)$$



Derivative Filters

applying filter is not done since gradient of an image has a lot of noise

Sobel

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Prewitt

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

straight lines

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Roberts

0	1
-1	0

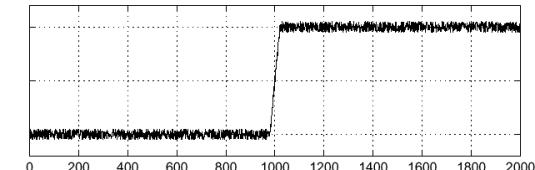
1	0
0	-1

line crossing

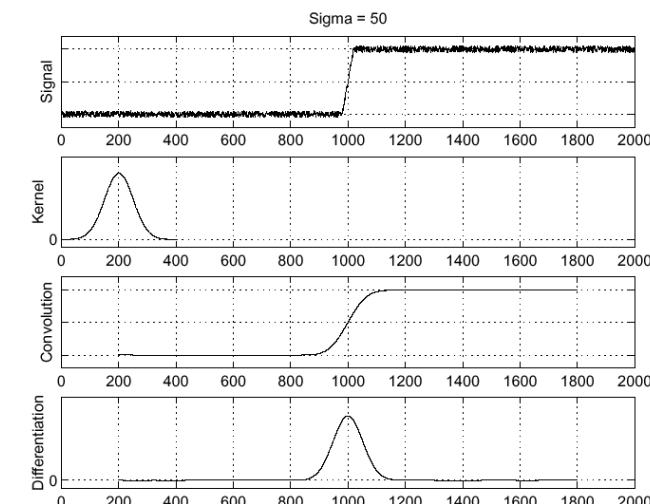
Preprocessing to Edge Detection

- In reality derivatives are very prone to noise
Consider the following example:

$$f(x)$$

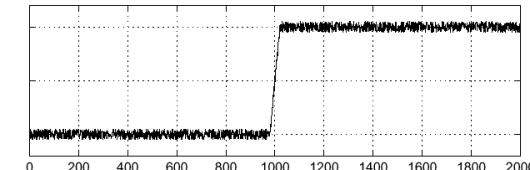
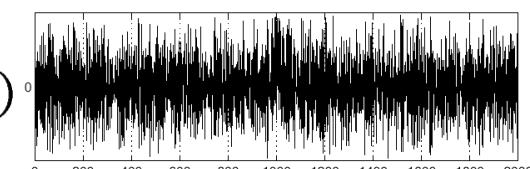


- To overcome this issue we can smooth the signal beforehand



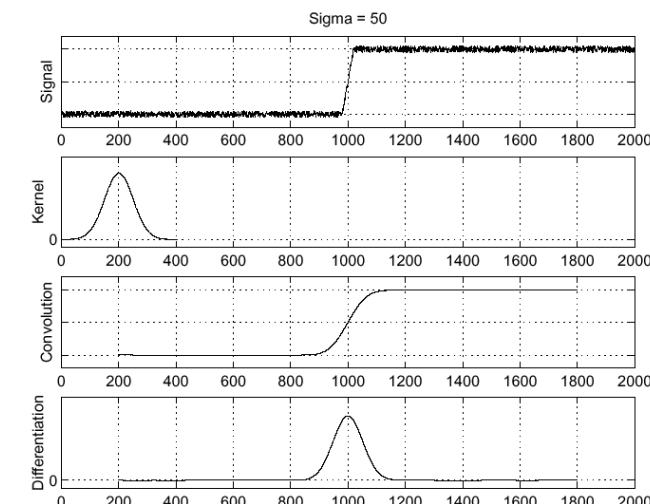
Preprocessing to Edge Detection

- In reality derivatives are very prone to noise
Consider the following example:

 $f(x)$  $\frac{d}{dx}f(x)$ 

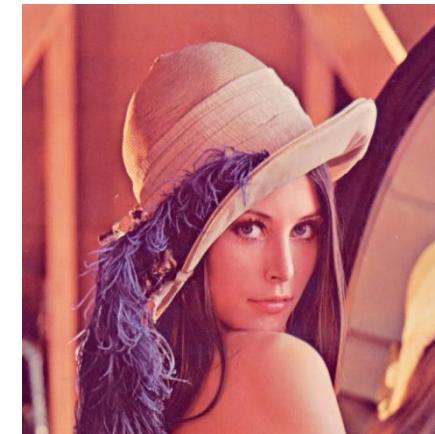
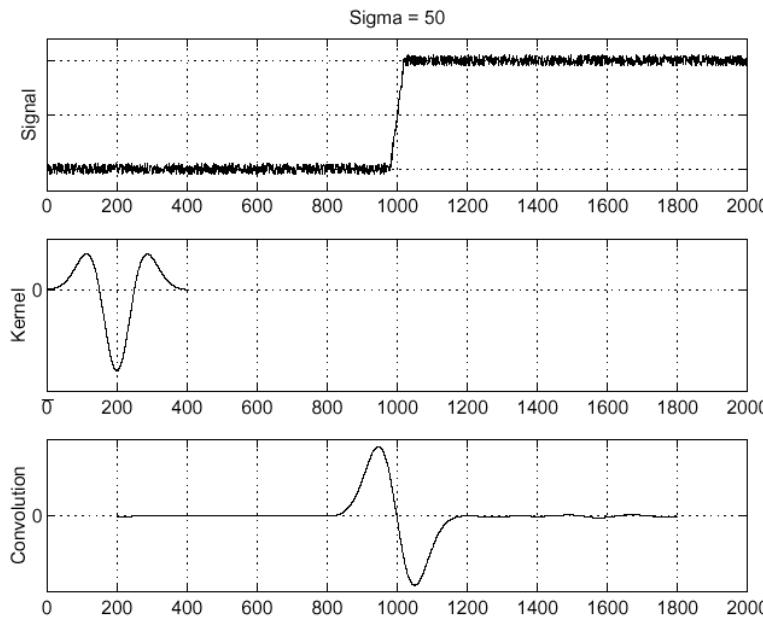
edge detection not found
since pixels are close to each other when
magnified

- To overcome this issue we can smooth the signal beforehand



Laplacian of Gaussian

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



Canny Edge Detection

only binary values

Example of a Complex system:

- Noise reduction
- Gradient calculation
- Non-maximum suppression if not maximum is reduced to zero to keep only edges
- Double threshold
- Edge Tracking by Hysteresis.

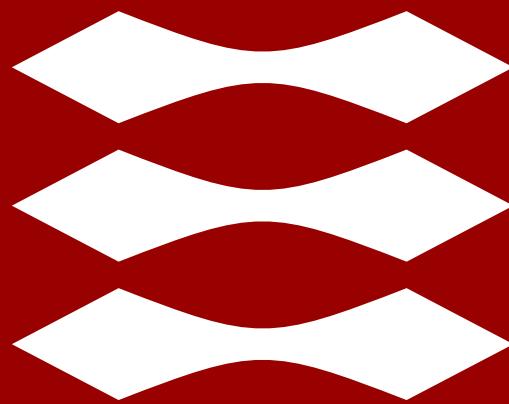


Perception for Autonomous Systems 31392:

Edge Detection

Lecturer: Evangelos Boukas—PhD

DTU



Perception for Autonomous Systems 31392:

Fitting

Lecturer: Evangelos Boukas—PhD

Fitting Data to a Model (Handling Outliers)

- Let's work with the line example
 - Fitting a model without (or with minimum) outliers data points
 - Fitting data through voting (Hough Transform)
 - RANdom SAmple Consensus (RANSAC)
- What about data not in a line?

Least Squares

cannot be solved when outliers are present

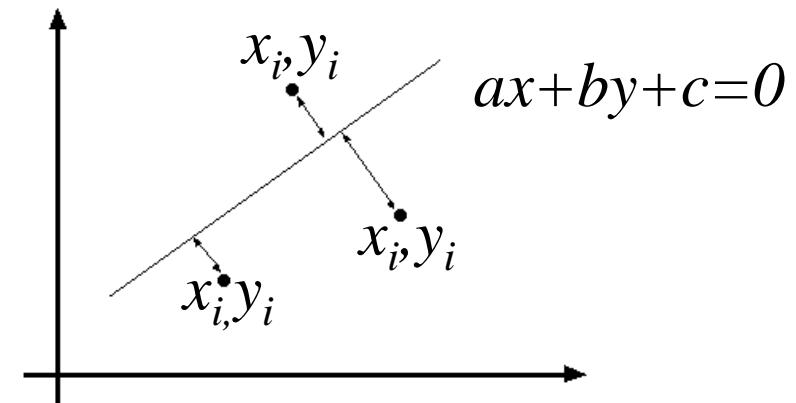
- Let the line depicted here be described by:
 $ax+by+c = 0$
- Then the distance of a point x_i, y_i is defined as:

$$|ax_i + by_i + c|$$

- Therefore, we can find the line that best matches our data by minimizing the following function:

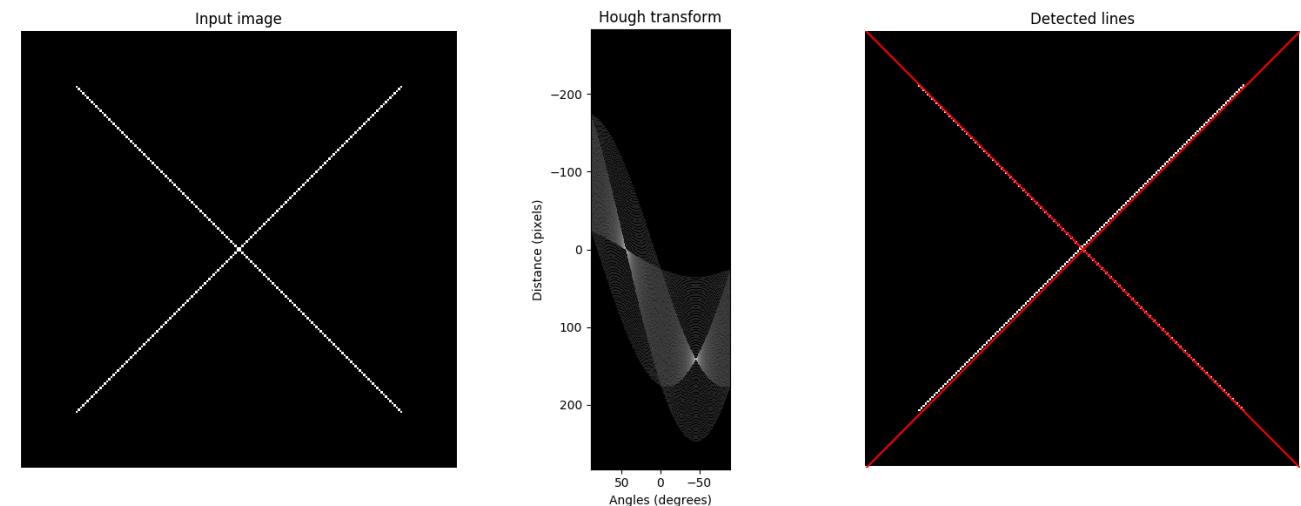
$$E = \sum_{i=1}^n (ax_i + by_i + c)^2$$

- However, in the presence of a lot of outlier data this problem is not directly solvable (in a closed form solution)



Hough Transform

- Assuming we want to fit a line in our data we can use the Hough transform as follows:
 - Formulate the problem as a bounded one
 - Create a grid of parameter values
 - Each data point votes on the grid
 - Find *local*-maxima in the grid and track back to lines in image



Hough Transform (1/4)

- Lets consider the line equation:

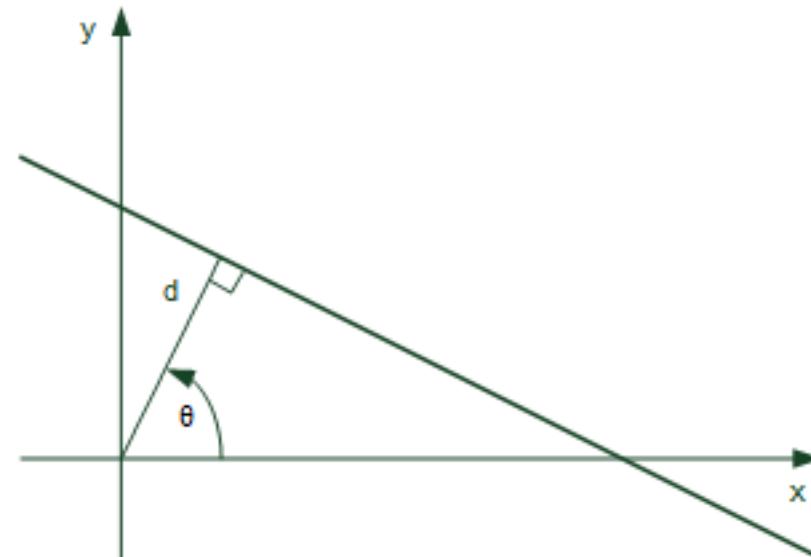
$$Y = aX + b$$

- The problem with the above equation is that a, b are unbounded,
Therefore we consider the following formulation (polar transformation):

$$x \cos \theta - y \sin \theta = d$$

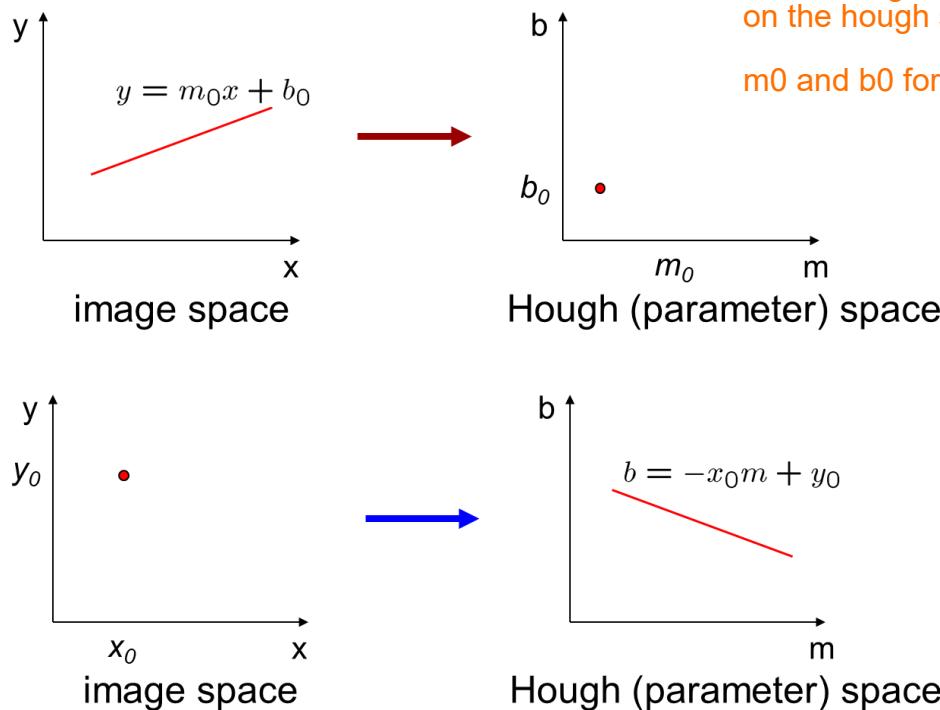
d : perpendicular distance from
line to origin

θ : angle the perpendicular
makes with the x-axis



Hough Transform (2/4)

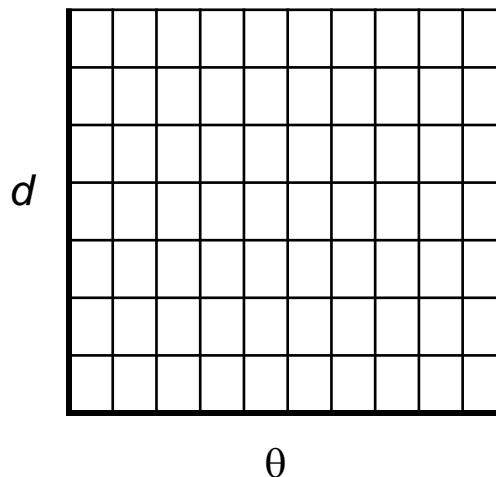
- Next step:
 - Initialize the grid using (d, θ)



line in image space is converted to a dot
on the hough spaces.
 m_0 and b_0 form the line equation $y = m*x + b$

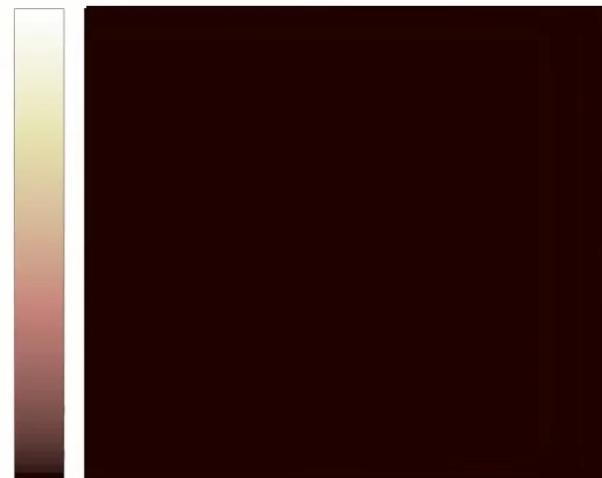
point in image space is a line in hough space
since infinite amount of lines can pass through that point

H : accumulator array (votes)

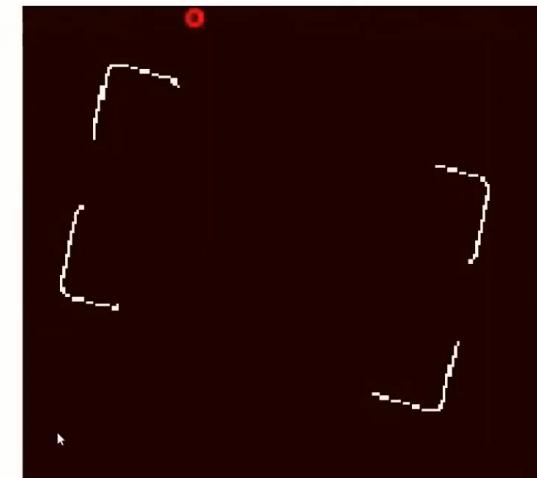


Hough Transform (3/4)

- Populate the grid by passing through the whole image and adding votes.
- See the following video: 



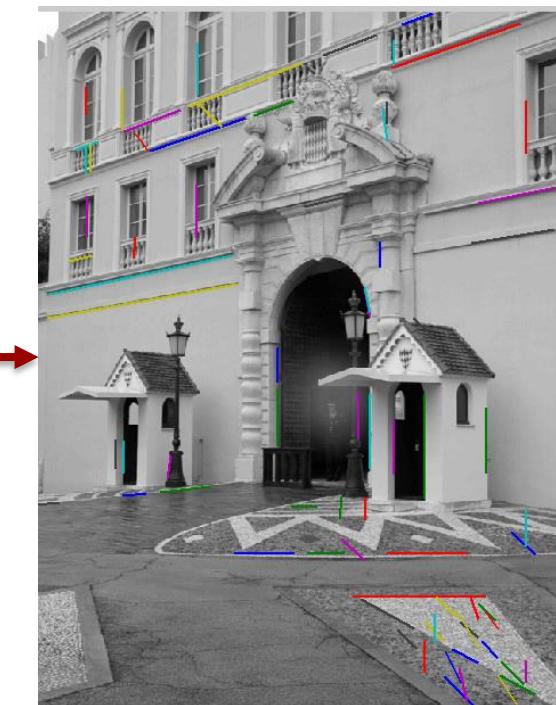
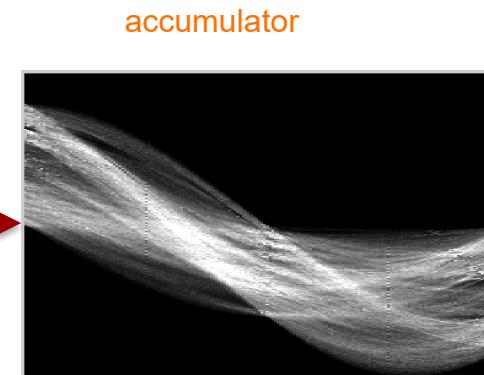
input image



Hough Transform (4/4)

max 4 parameters, else becomes exponential

- Identify maxima and track lines back to image:



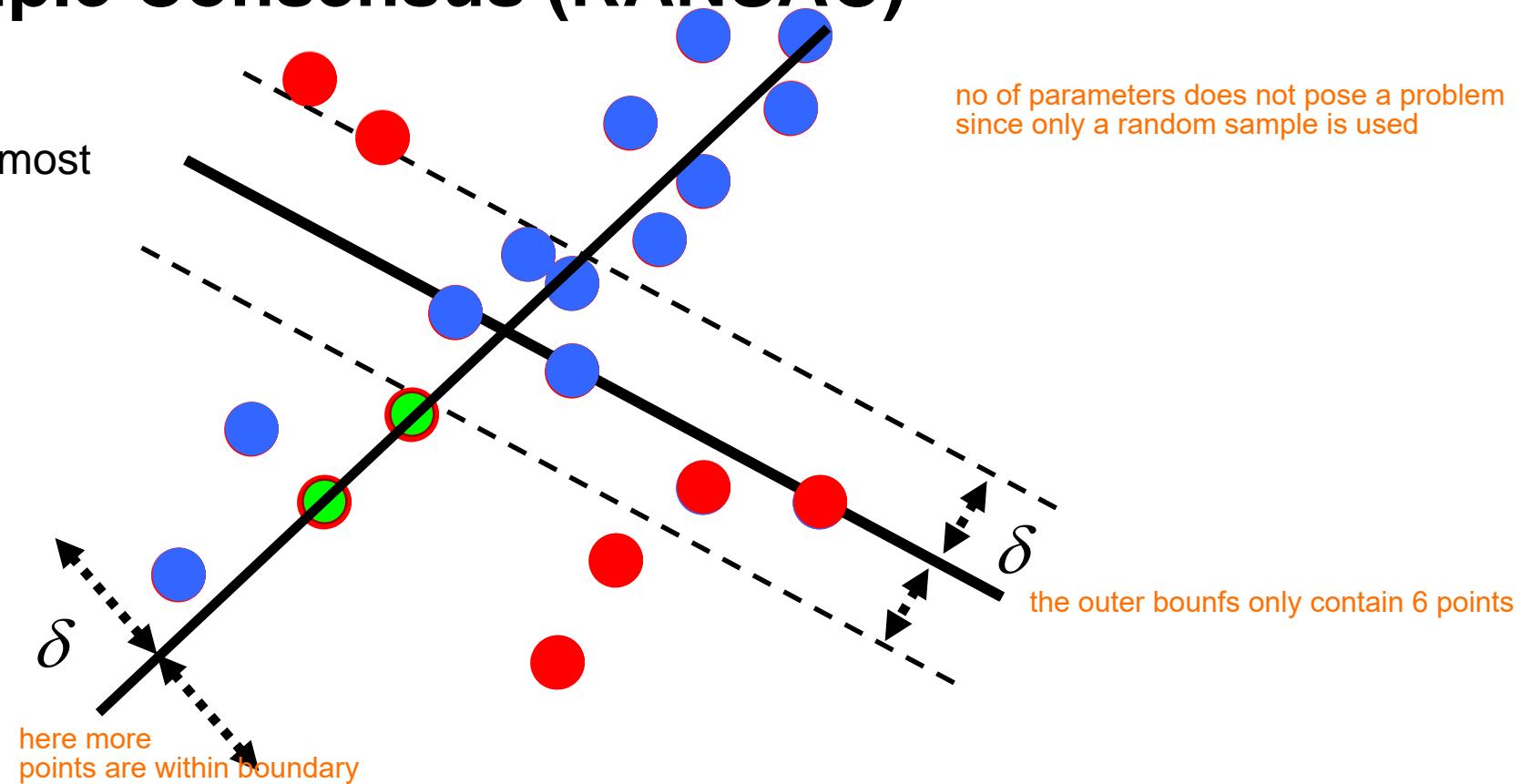
RANdom SAmple Consensus (RANSAC)

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
 2. **Solve** for model parameters using samples
 3. **Score** by the fraction of inliers within a preset threshold of the model
- **Repeat** 1-3 until the best model is found with high confidence
 - δ - is the threshold upon whitch a sample is considered to not fit to the selected model

RANdom SAmple Consensus (RANSAC)

- Select the models with most inliers to create lines



What about data not in a line?

- Same approach is followed for more complex models
eg: circle model is:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

which requires 3 parameters in the Hough grid

- However the complexity grows exponentially
(usually up to 4 parameters is advised)
- Ransac can handle higher order models.

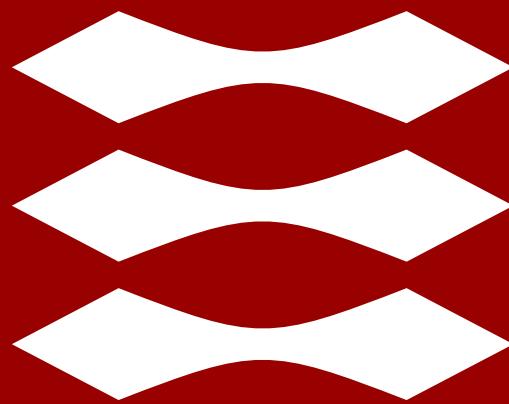


Perception for Autonomous Systems 31392:

Fitting

Lecturer: Evangelos Boukas—PhD

DTU



Perception for Autonomous Systems 31392:

Image Feature Detection and Description

Lecturer: Evangelos Boukas—PhD

Image Features

- Feature Detection:

Find the most “prominent” Points (areas) in an image.

The ones which are likely to be detected in other images, as well

- Feature Description:

Create a “unique” descriptor fingerprint for each Feature point

- Feature Matching:

Find correspondences among different images

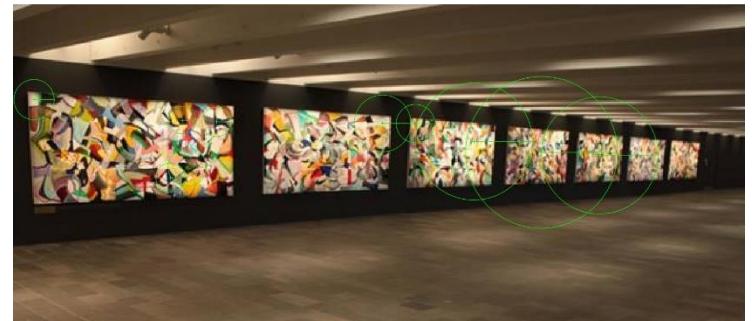
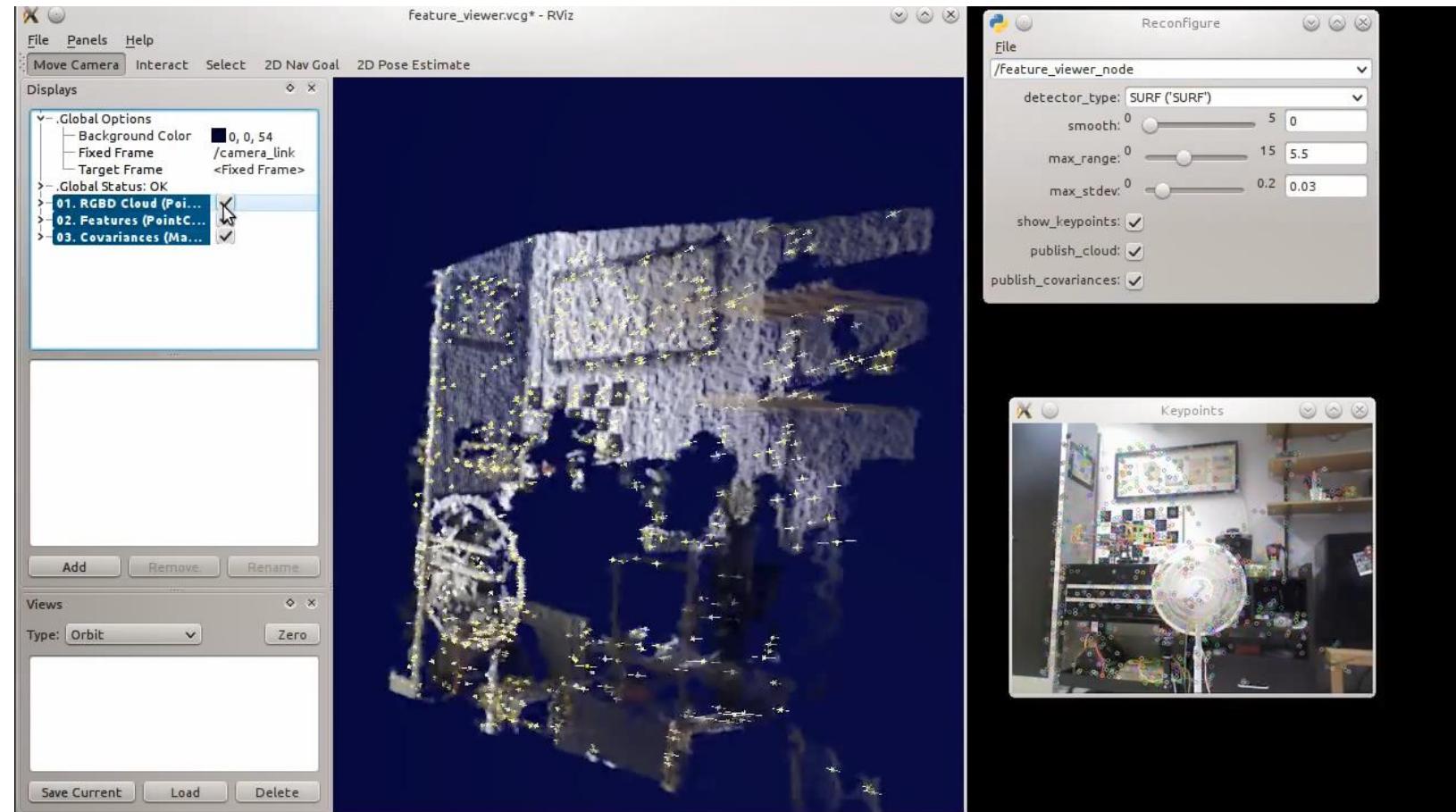


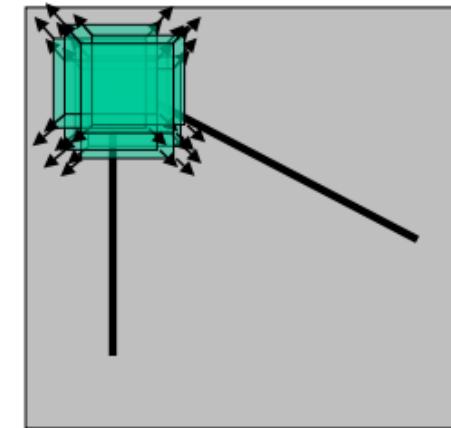
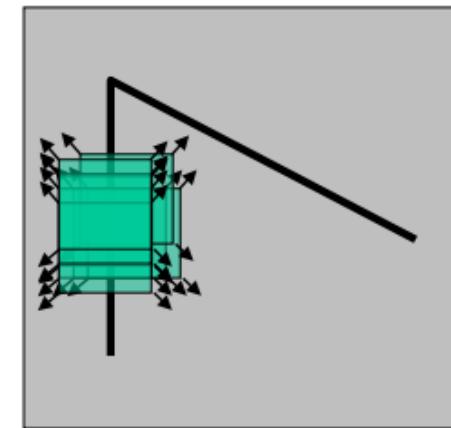
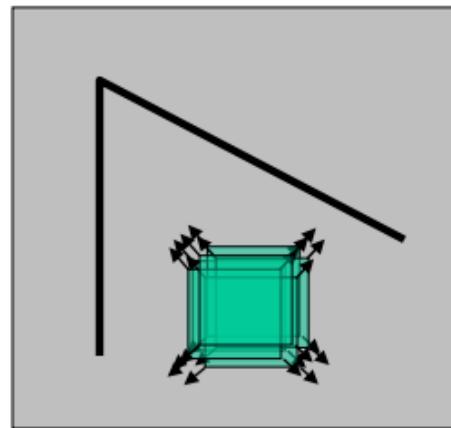
Image Features

- A quick view



Feature Detection: Harris corner detector

- Corners are great for features



- Lines not so, why

line is unbounded

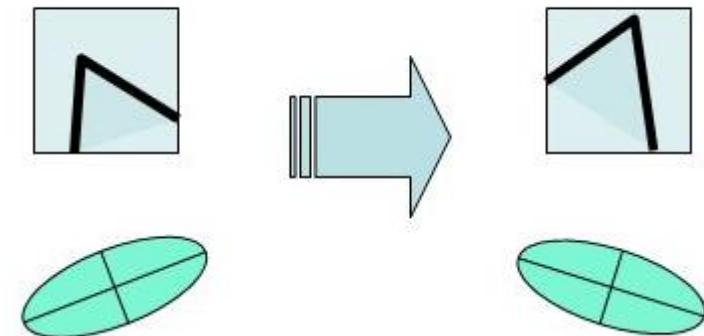
“flat” region:
no change in
all directions

“edge”:
no change along
the edge direction

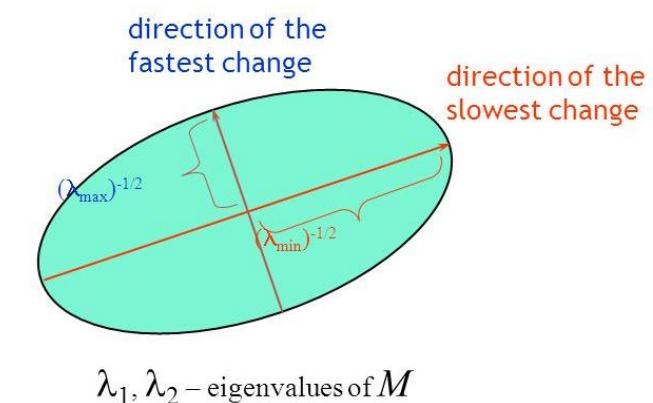
“corner”:
significant change
in all directions

Feature Detection: Harris corner detector

- Harris can discriminate among **edge**, **flat** and **corners**

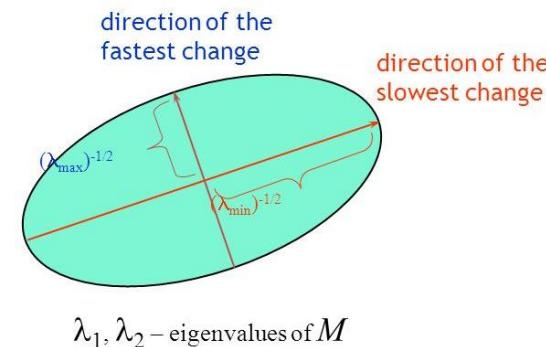


- Notice how by rotating it does not change
- Linear algebra



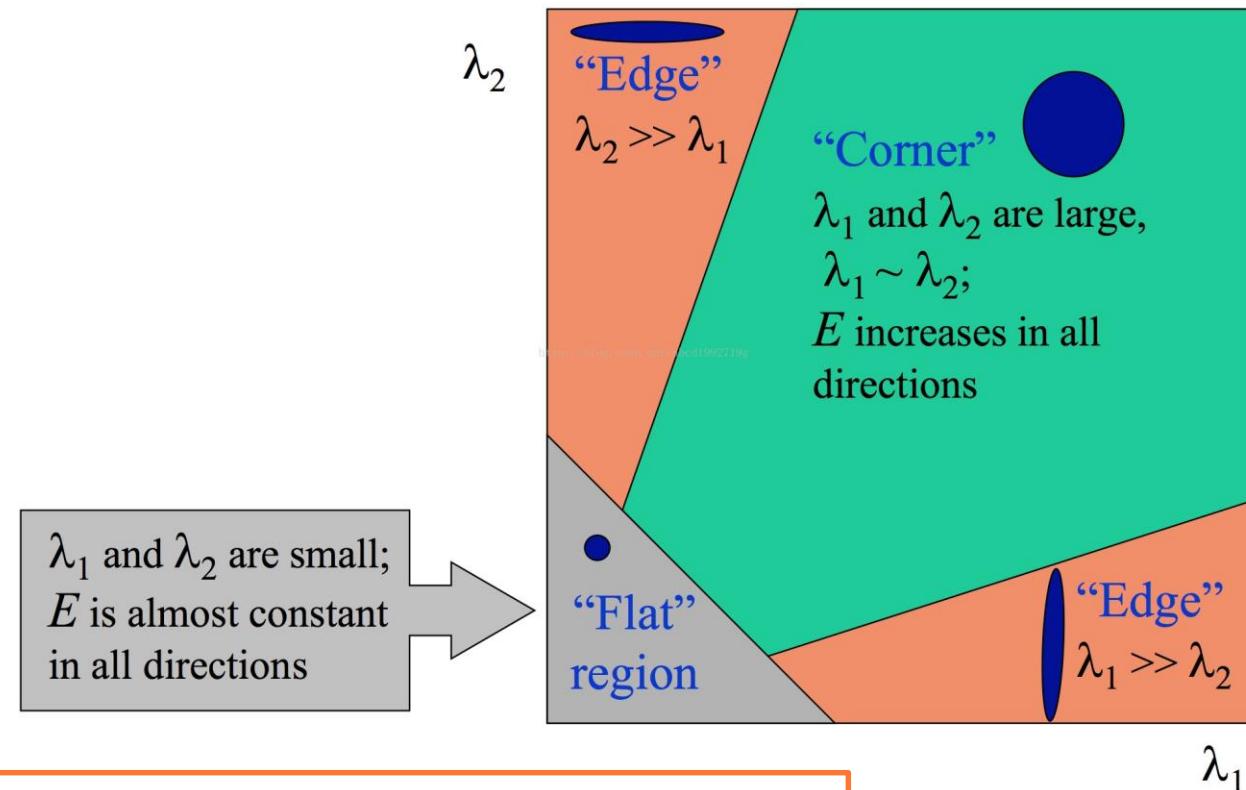
Feature Detection: Harris corner detector

- So how are these eigenvalues useful?



Feature Detection: Harris corner detector

- So how are these eigenvalues useful?

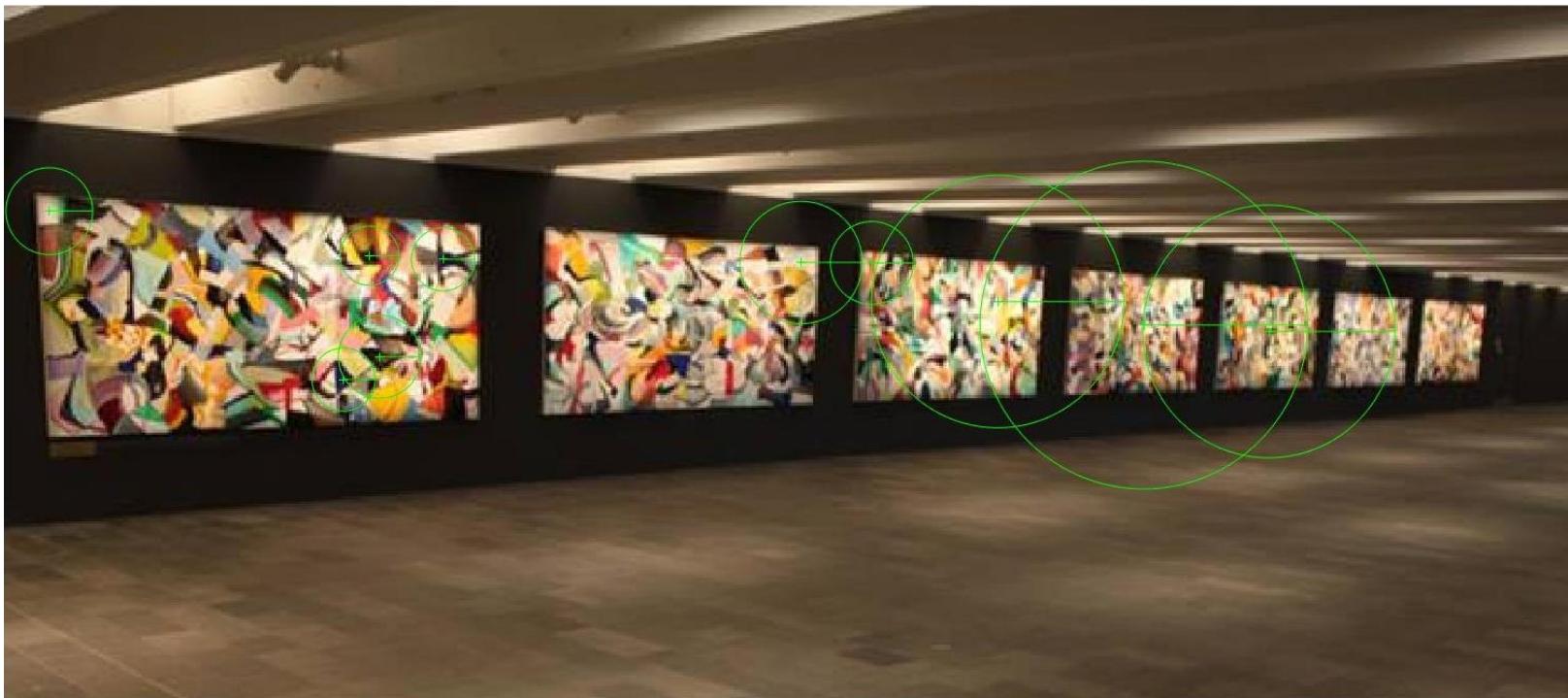


Mikolajczyk, K., and Schmid, C., "A performance evaluation of local descriptors".

IEEE Transactions on Pattern Analysis and Machine Intelligence, 10, 27, pp 1615--1630, 2005.

Feature Detection: Harris corner detector

- So how are these eigenvalues useful?

 λ_1

Mikolajczyk, K., and Schmid, C., "A performance evaluation of local descriptors",
IEEE Transactions on Pattern Analysis and Machine Intelligence, 10, 27, pp 1615--1630, 2005.

Feature Detection: Scale Space Theory

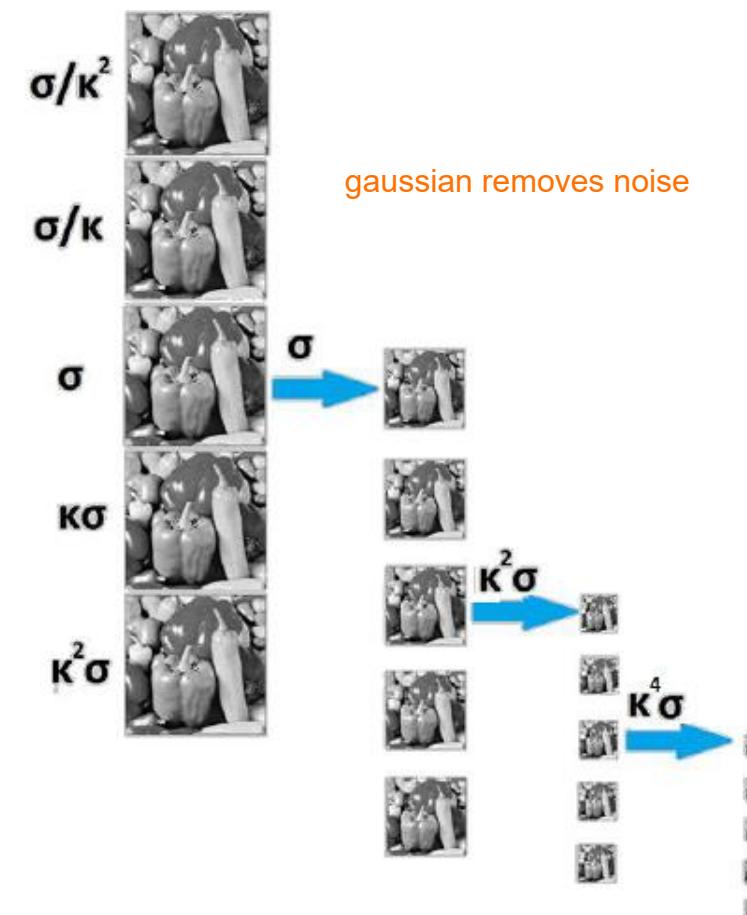
- So we can find corners.
- But how descriptive are these corners?
 - Not really,
 - Think that the roof of a building has corners
 - And your desk has corners...
- Finally a revelation:

Lindeberg, T. 1994. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224-270

Lindeberg, Tony (1998). "Feature detection with automatic scale selection". *International Journal of Computer Vision* 30 (2): 79-116.

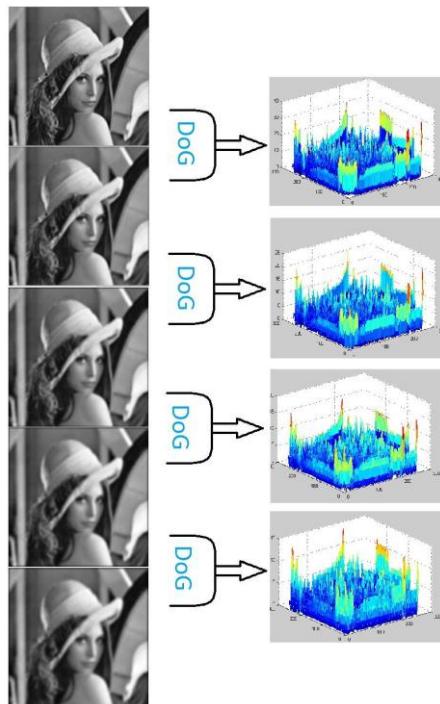
Feature Detection: Scale Space Theory

- We should be able to find these points which are prominent in different scales
- Create octaves with different scale among them
- Different blur level (Gaussian) in them



Feature Detection: Difference of Gaussians

- The main point is the difference of Gaussians
- We can then do this on our scale-space:



finds points that are points of interest

The mother of Features - SIFT

- Ok, we've seen how we can find interest points, but how about matching??
- David Lowe published the most influential paper in computer vision:

Lowe, D. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60, 2 , pp 91-110 (2004).

- How many people do you think have cited this?
- 60000!!!!

Scale Invariant Feature Transform - SIFT

Contains all:

- Detection
- Description
- Matching

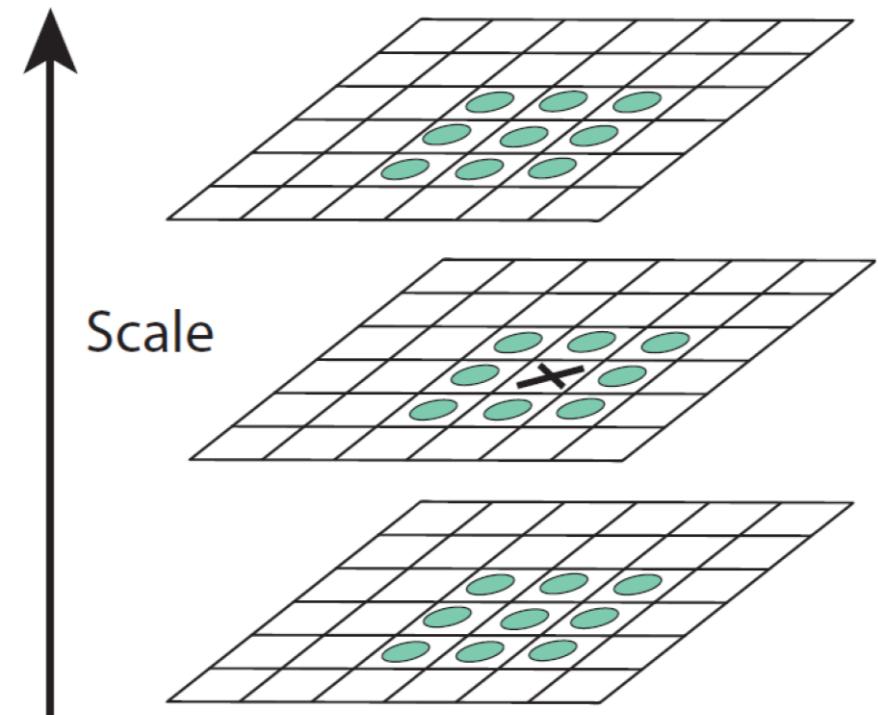
Main Feature is that it is robust in:

- Change of Translation
- Change in Scale
- Change in Rotation
- Change in 3D View Point
- Change in Illumination

Scale Invariant Feature Transform - SIFT

Detection

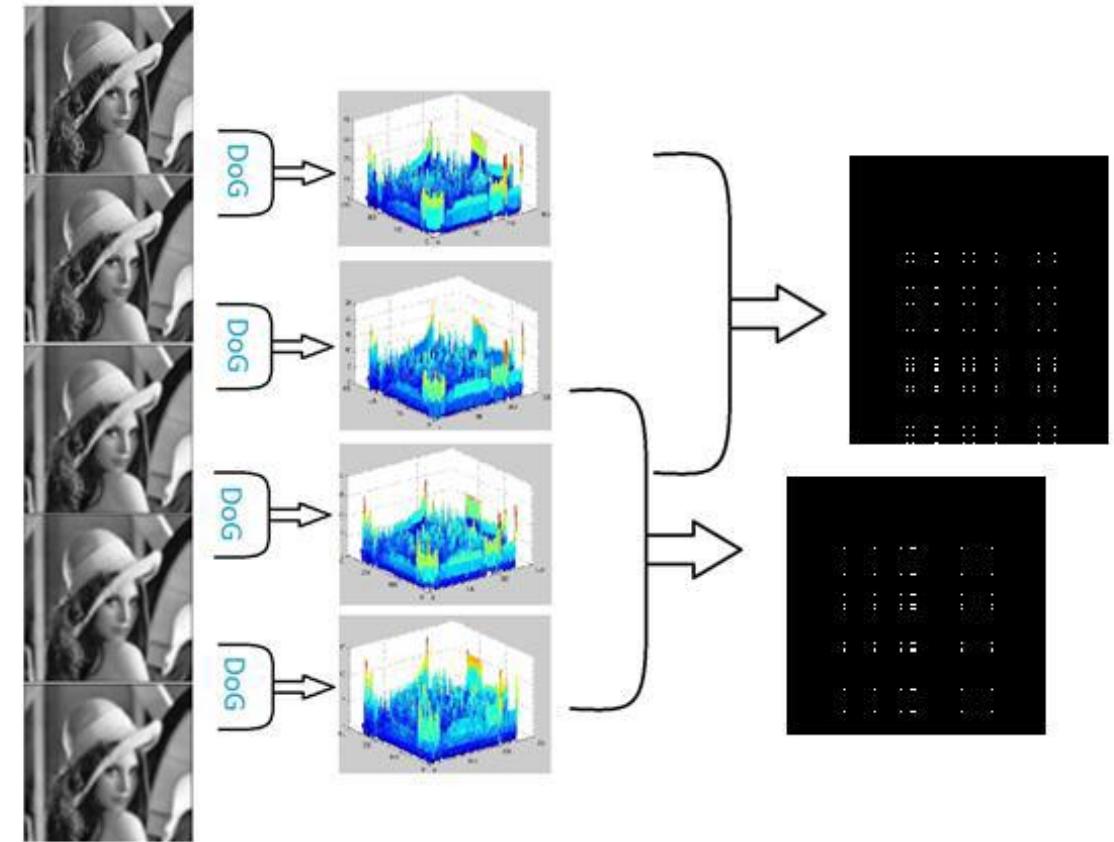
- Use DoG on Scale Space
 - Only the maximum or minimum in a neighborhood are considered
 - All octaves are investigated



Scale Invariant Feature Transform - SIFT

Detection

- Use DoG on Scale Space
 - Only the maximum or minimum in a neighborhood are considered
 - All octaves are investigated
- Specifically,
 - In groups of 3
 - 2 Set of Points from each octave
 - 4 octaves -> 8 set of Points



Scale Invariant Feature Transform - SIFT

Description

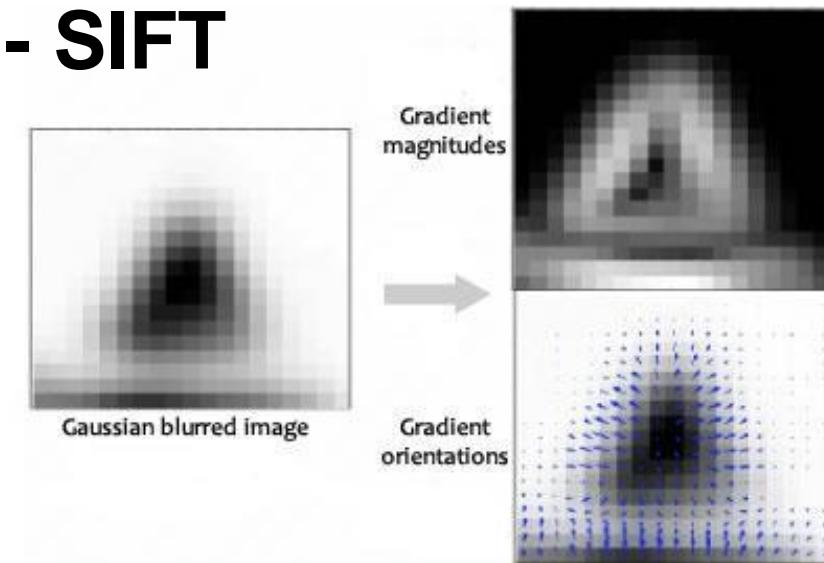
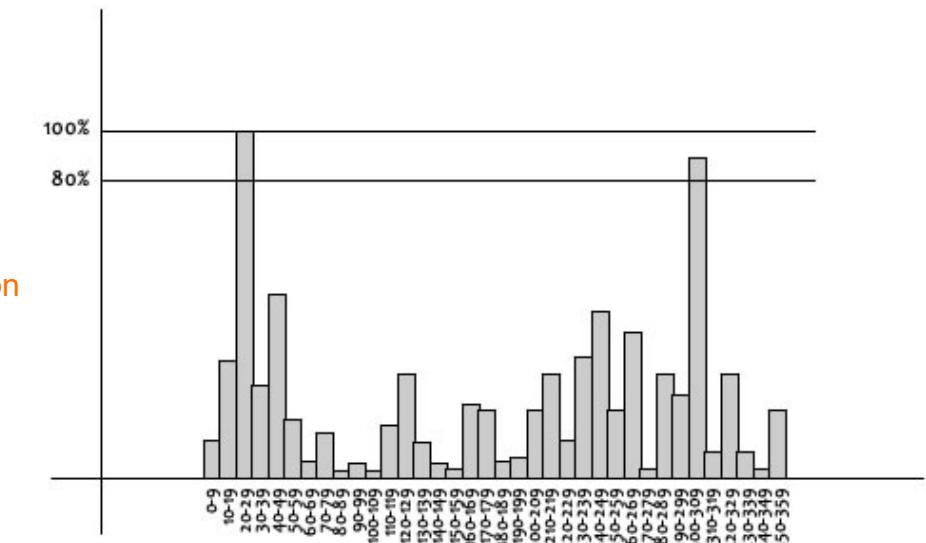
- Orientation:
 - Based on the gradient

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

- Create a histogram of orientations
Consisting of 36 bins (every 10 degrees)
- Fingerprint

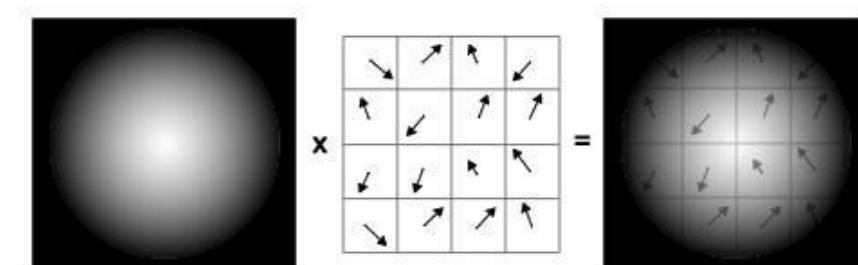
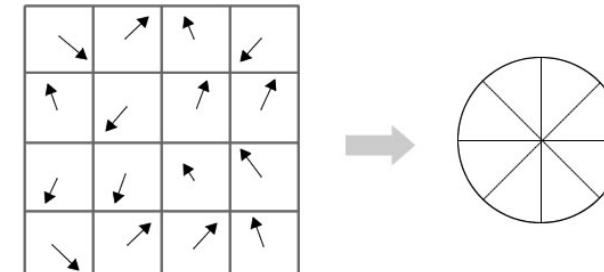
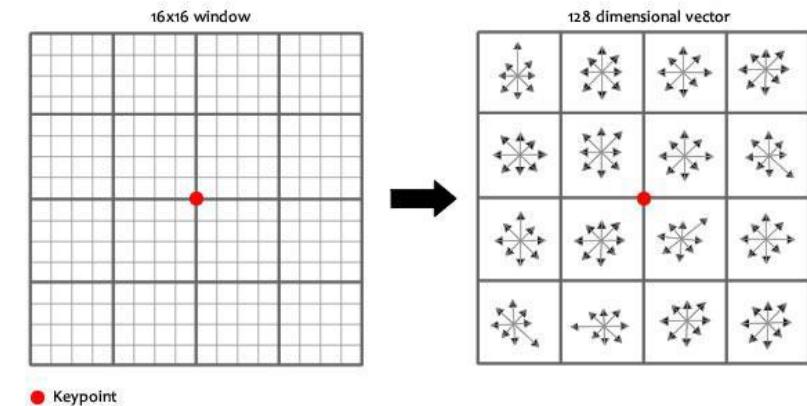
orientation of feature is the orientation
of the max histogram



Scale Invariant Feature Transform - SIFT

Description

- Orientation
- Fingerprint:
 - Assume a 16×16 area around each Key Point
 - Create histogram with 8 bins (as before)
 - This gives out 128 values
 - Note that the values are scaled for proximity



Scale Invariant Feature Transform - SIFT

Matching

- Given some features from (let's say) 2 Images:
- Lowe proposed the ALL-ALL Euclidean distance:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2}$$

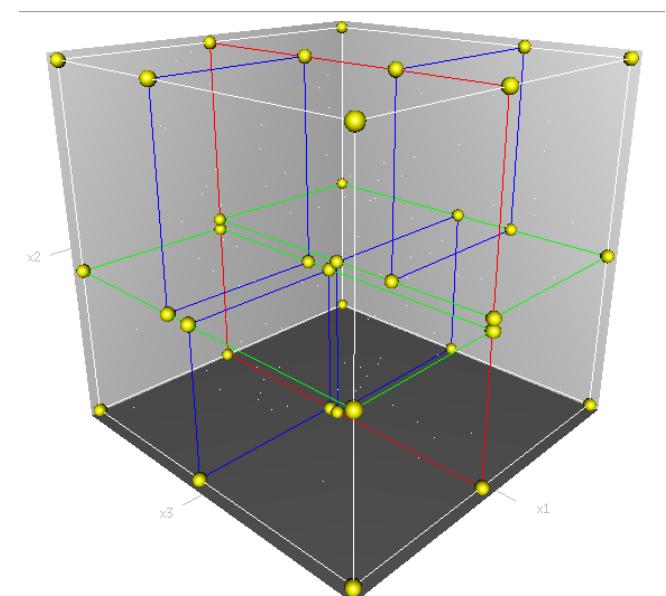
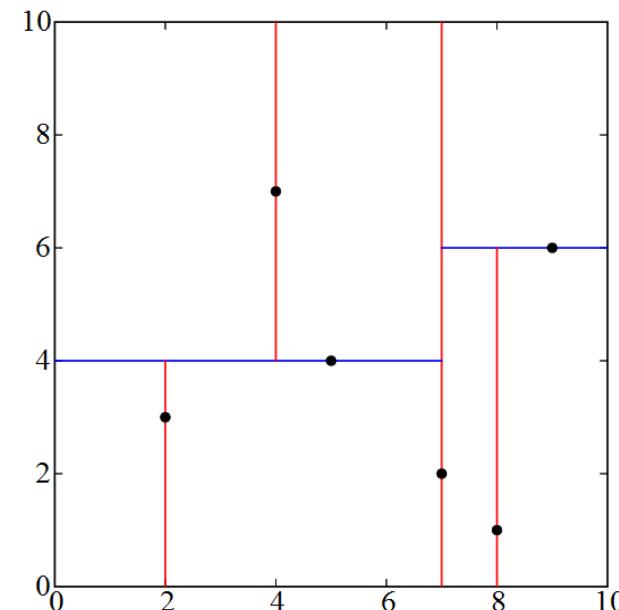
- For a 640x640 image we can get even 1000 features
- With 128 values each feature, you see how this can get ugly quickly

Scale Invariant Feature Transform - SIFT

Matching

- So our hero, proposed the usage of Kd-Trees:
- Creation:
 - You split the space in half based on distance
 - Then again
 - Then again,....
- Search:
 - Start from top
 - Is it closer to 1 than 2?
 - Go in 1
 - Is it closer to 1.1 than 1.2?
 - Go in 1.1
 - GO On until you find a feature

a mid point is chosen taht splits the values to left and right and then further splits happen till no of points becomes less



speeds up process 100x

matching is the most computationally intensive

Feature points and areas

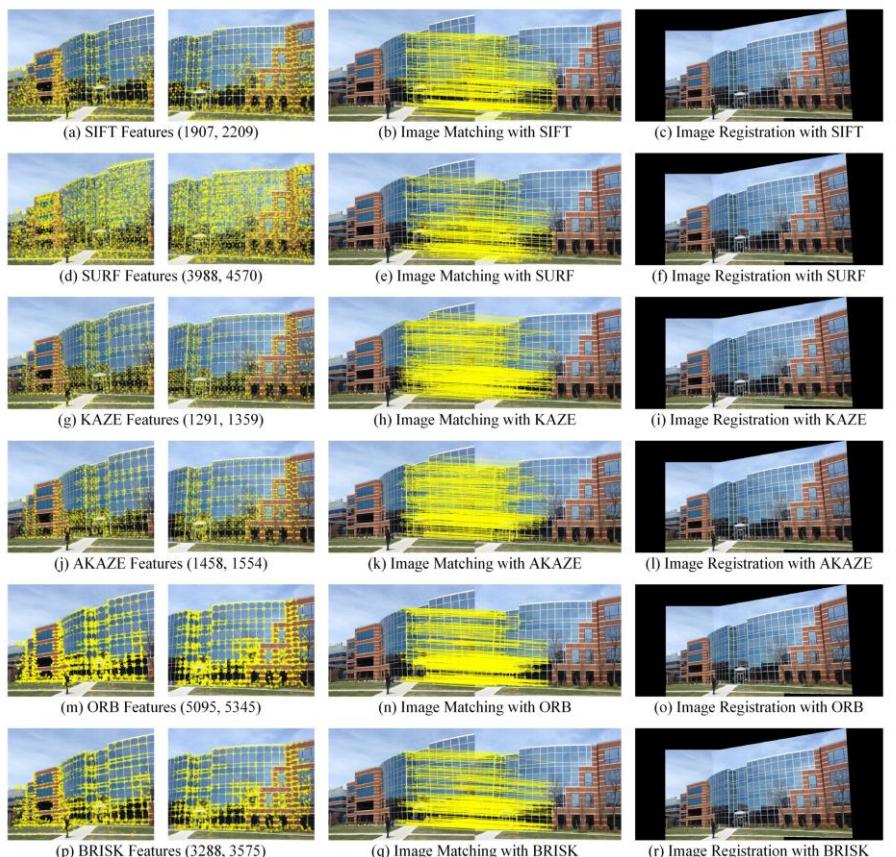
- A multitude of features have been proposed (specific for each application):
 - SIFT
 - SURF
 - BRISK
 - FREAK
 - MSER
 - ORB
 - ...

Feature points and areas

- A multitude of features have been proposed (specific for each application):

- SIFT
- SURF
- BRISK
- FREAK
- MSER
- ORB
- ...

A special case to research



What applications do the Features have?

- More or less everything..
- Motion Estimation
- Localization
- Mapping
- Photogrammetry
- Image Retrieval
- Machine Learning
 - Object Detection
 - & Recognition
- Autonomous Driving
-
- More or less everything!

What applications do the Features have?

- More or less everything..
- Motion Estimation
- Localization
- Mapping
- Photogrammetry
- Image Retrieval
- Machine Learning
 - Object Detection
 - & Recognition
- Autonomous Driving
-
- More or less everything!



Image Feature Detection and Description

- What did we learn?
 - Is that a good feature?
 - How to get scale and rotational invariance in the features we get?
 - How to detect points of interest?
 - How to describe the feature of points so,
 - We can match them across multiple images.
 - How can we use features?

optical flow: flow of points in image

high speed higher/longer vectors

Perception for Autonomous Systems 31392:

Image Feature Detection and Description

Lecturer: Evangelos Boukas—PhD

Perception for Autonomous Systems

Lecture 2 - Image Processing (08/02/2021)

Outline/Content:

- Edge Detection
 - What is an Edge?
 - Image Derivative
 - Gradient
 - Sobel
 - Laplacian
 - Canny
- Image Feature Detection and Description
- Feature Description
- Feature Matching
- Fitting Data to a Model (Handling Outliers)
 - Least Squares
 - Hough Transform
 - RANdom Sample Consensus (RANSAC)

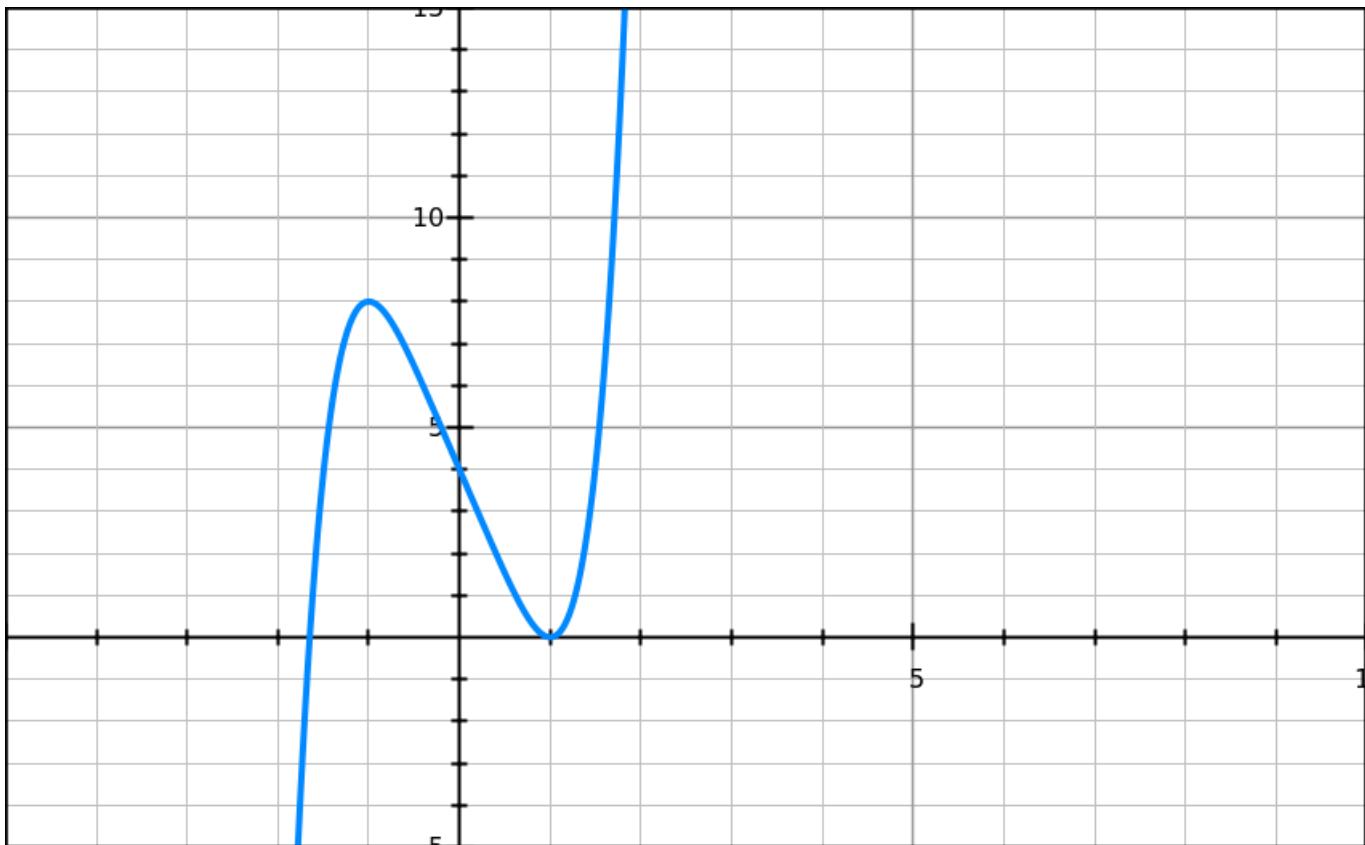
Topics and reading material

- Feature detection
 - Book A, Chapter 4.1.1
- Feature description and Matching
 - Book A, Chapters 4.1.2 - 4.1.3
- Feature Tracking
 - Book A, Chapter 4.1.4
- Hough transform, RANSAC
 - Book A, Chapter 4.3.2

Edge Detection

What is an edge?

Edge detection embodies mathematical methods to find points in an image where the brightness of pixel intensities changes distinctly. Simply put, edges occur at the boundaries between areas of different color, intensity, or texture.

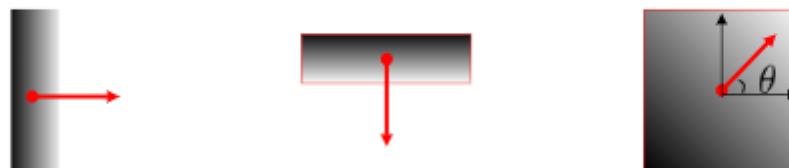


Visually speaking, do edges occur at locations of steep slopes, or equivalently, in regions of closely packed contour lines (extremas). The edge detection is calculated based on the derivative of an image.

Gradient

The gradient is a vector which points in the direction of most rapid change in intensity:

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



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0 \right] \quad \nabla f = \left[0, \frac{\partial f}{\partial y} \right] \quad \nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

Sobel

Sobel is a edge extractor operator, which is a seperable combination of a horizontal central difference (so called because the horizontal derivative is centered on the pixel) and a vertical filter (to smooth the results).

- Practically:

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Magnitude:

$$g = \sqrt{g_x^2 + g_y^2}$$

- Orientation:

$$\Theta = \tan^{-1} \left(\frac{g_y}{g_x} \right)$$

Derivative Filters

Sobel

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

Scharr

$$\begin{array}{|c|c|c|} \hline 3 & 0 & -3 \\ \hline 10 & 0 & -10 \\ \hline 3 & 0 & -3 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 3 & 10 & 3 \\ \hline 0 & 0 & 0 \\ \hline -3 & -10 & -3 \\ \hline \end{array}$$

Prewitt

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

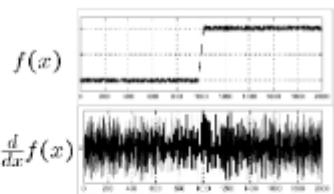
Roberts

$$\begin{array}{|c|c|} \hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$$

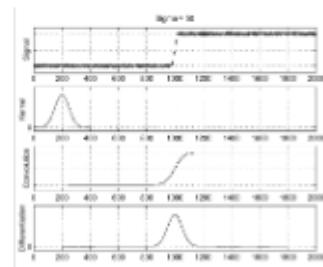
Preprocessing to Edge Detection

Taking image derivatives (required for edge detection) accentuates high frequencies and hence amplifies noise, since the proportion of noise to signal is larger at high frequencies. It is therefore prudent to smooth the image with a low-pass filter prior to computing the gradient.

- In reality derivatives are very prone to noise
Consider the following example:



- To overcome this issue we can smooth the signal beforehand



Because it's desirable for the response to be independent of orientation it's necessary to use a circularly symmetric smoothing filter.

Laplacian of Gaussian

The Laplacian of Gaussian (LoG) is the convolution kernel of the Laplacian (gradient operator dot product with the gradient).

Canny Edge Detection

The Canny edge detector was developed way back in 1986 by John F. Canny. And it's still widely used today as one of the default edge detectors in image processing.

The Canny edge detection algorithm can be broken down into 5 steps:

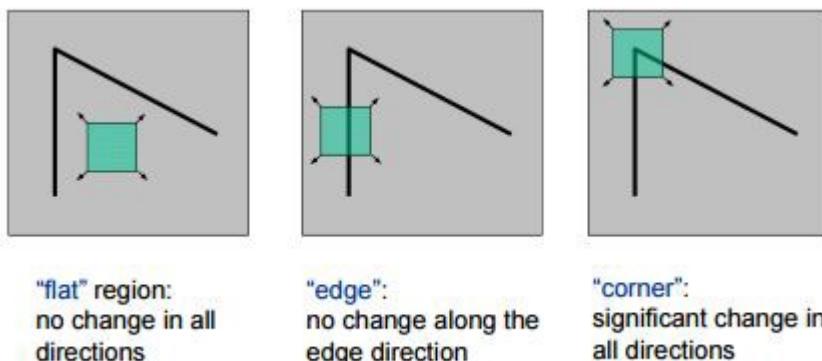
- Step 1: Smooth the image using a Gaussian filter to remove high frequency noise.
- Step 2: Compute the gradient intensity representations of the image.
- Step 3: Apply non-maximum suppression to remove "false" responses to edge detection.
- Step 4: Apply thresholding using a lower and upper boundary on the gradient values.
- Step 5: Track edges using hysteresis by suppressing weak edges that are not connected to strong edges.

Image Features

- Feature Detection:
 - Find the most "prominent" Points (areas) in an images
- Feature Description:
 - Create a "unique" descriptor fingerprint for each Feature point
- Feature matching:
 - Find correspondences among different images

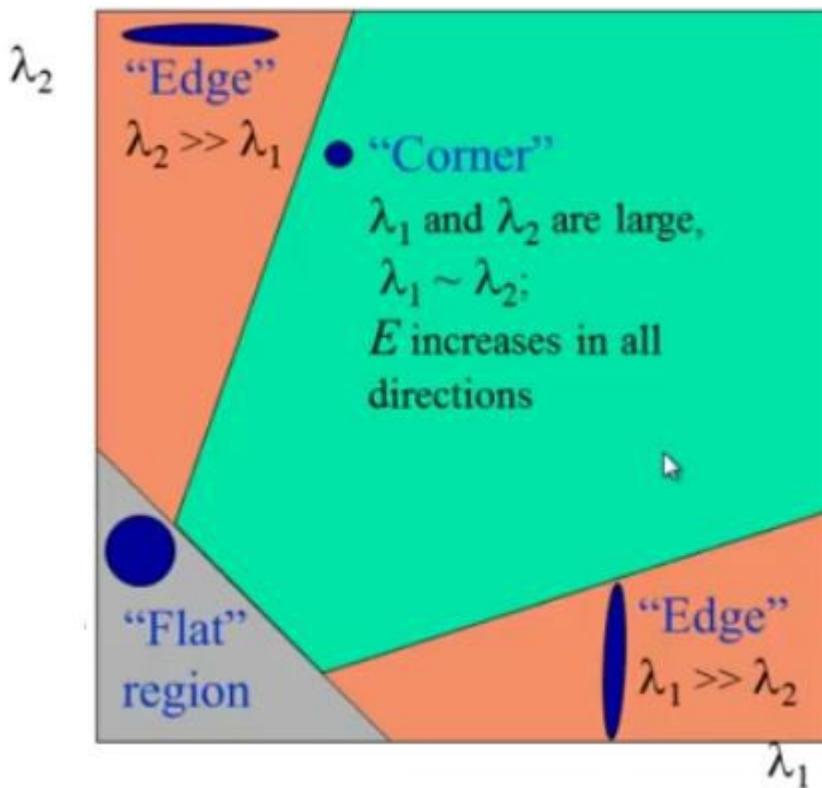
Feature Detection: Harris corner detector Medium

A corner is a point whose local neighborhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness. Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.



So let's understand why corners are considered better features or good for patch mapping. In the above figure, if we take the flat region then no gradient change is observed in any direction. Similarly, in the edge region, no gradient change is observed along the edge direction. So both flat region and edge region are bad

for patch matching since they are not very distinctive (there are many similar patches in along edge in edge region). While in corner region we observe a significant gradient change in all direction. Due to this corners are considered good for patch matching (shifting the window in any direction yield a large change in appearance) and generally more stable over the change of viewpoint.



The values of these eigenvalues (lambdas) decide whether a region is a corner, edge or flat. **[More information is to be found in the linked medium article.](#)**

[Feature Detection: Scale Space Theory Expert: Tony Lindeberg | Scale-space theory: A basic tool for analysing structures at different scales, Tony Lindeberg](#)

A theory of multi-scale representation of sensory data developed by the image processing and computer vision communities. The purpose is to represent signals at multiple scales in such a way that fine scale structures are successively suppressed, and a scale parameter t is associated with each level in the multi-scale representation.

Scale space is a collection of images having different scales, generated from a single image. For more information check the linked source.

[The mother of Features - SIFT Article](#)

SIFT, or Scale Invariant Feature Transform, is a feature detection algorithm in Computer Vision.

It helps to locate the local features in an image, commonly known as the 'keypoints' of the image. These keypoints are scale & rotation invariant that can be used for various computer vision applications, like image matching, object detection, scene detection, etc.

This implies that simple corner detectors are not able to match features for images with different scales and rotations. That's when SIFT comes into play.

Characteristics of the SIFT algorithm: Contains:

- Detection
- Description
- Matching

It's robust in:

- Change of Translation
- Change in Scale
- Change in Rotation
- Change in 3D View Point
- Change in Illumination

Scale Invariant Feature Transform - SIFT

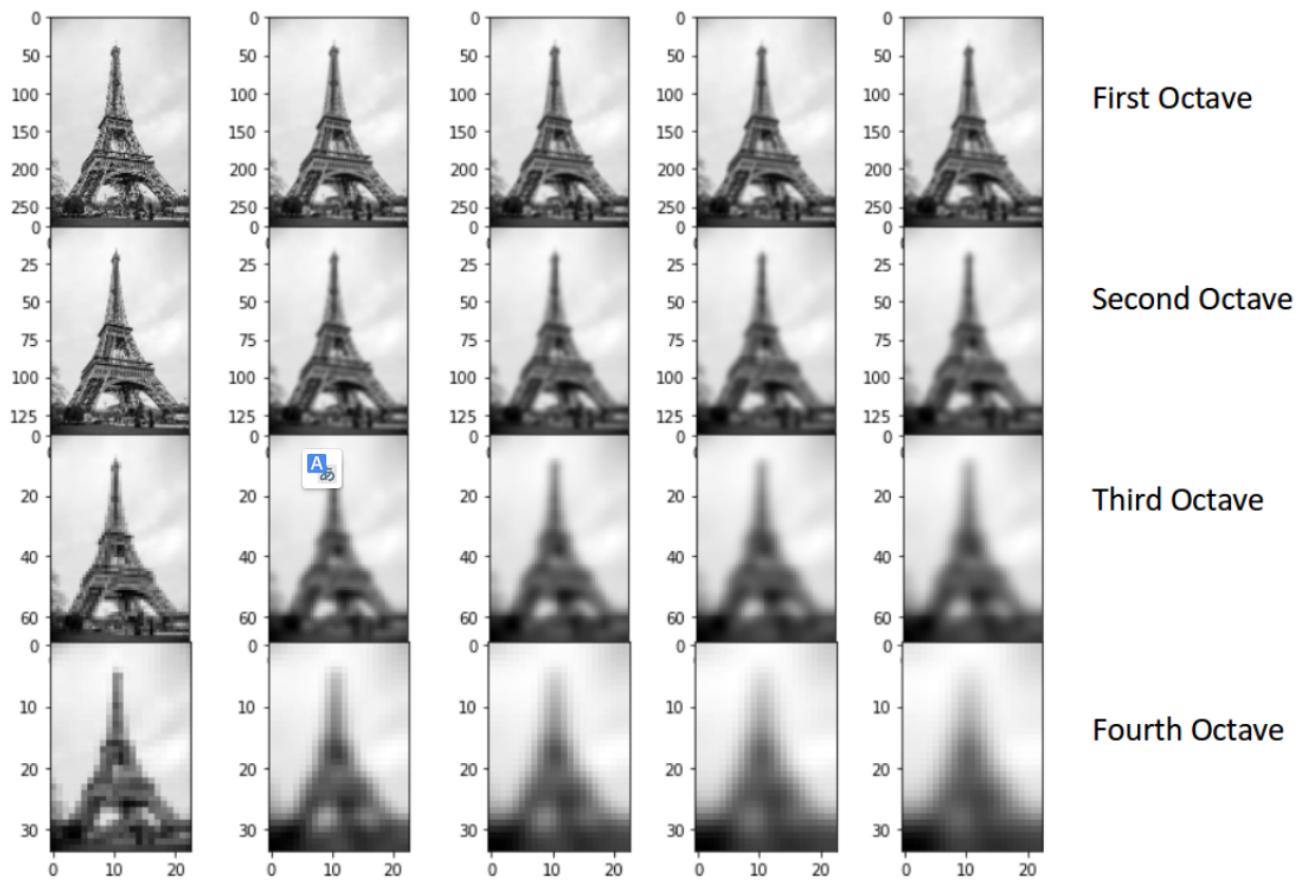
The entire process of that algorithm can be divided into 4 parts:

- Constructing a Scale Space: To make sure that features are scale-independent
- Keypoint Localisation: Identifying the suitable features or keypoints
- Orientation Assignment: Ensure the keypoints are rotation invariant
- Keypoint Descriptor: Assign a unique fingerprint to each keypoint

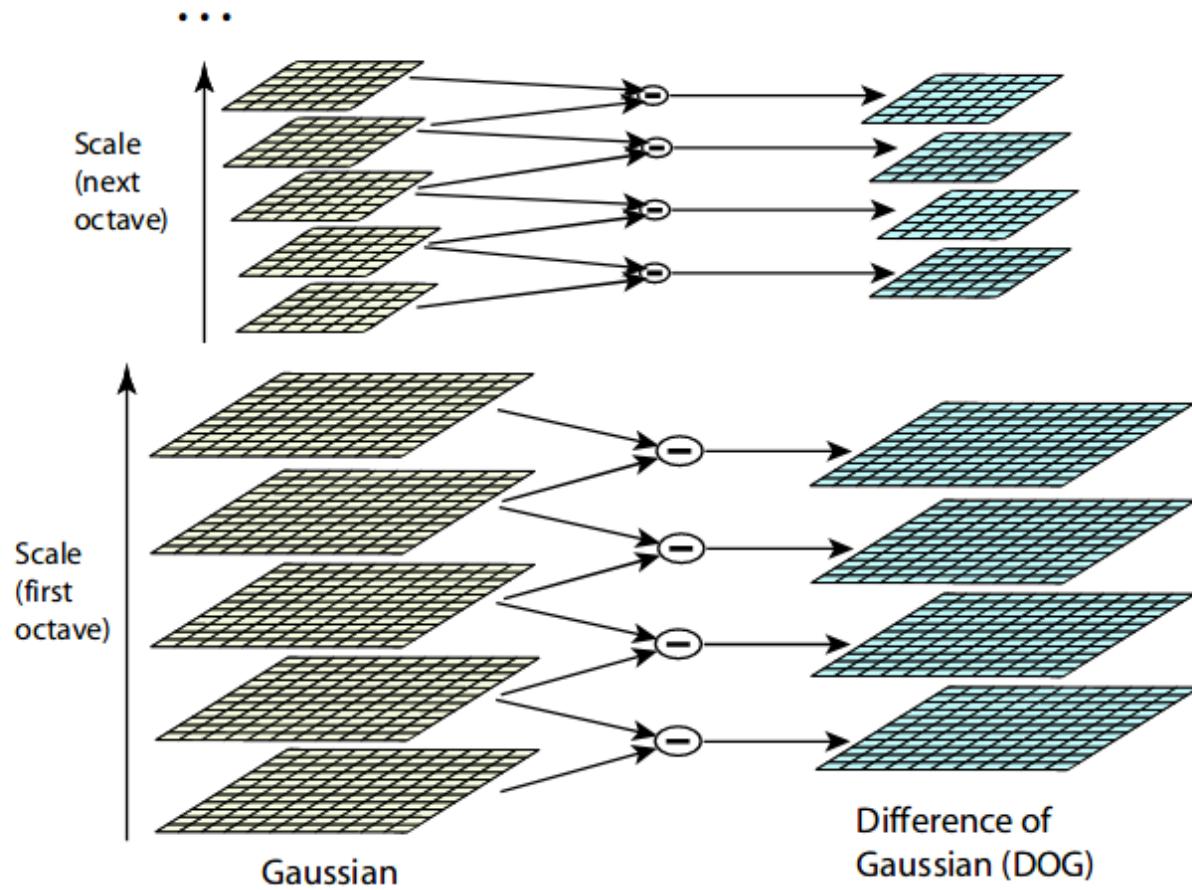
Applied Scale space

The first step of the sift algorithm is to create a number of octaves (images of different scales) and blur out all of those images repeatedly.

The ideal number of octaves should be four, and for each octave, the number of blur images should be five (recommended by the authors).



The second step is to enhance the images. DoG creates another set of images, for each octave, by subtracting every image from the previous image in the same scale. Here is a visual explanation of how DoG is implemented:



Difference of Gaussian is a feature enhancement algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original.

The next step is to find the important keypoints from the image that can be used for feature matching. The idea is to find the local maxima and minima for the images. This part is divided into two steps:

- Find the local maxima and minima
- Remove low contrast keypoints (keypoint selection)

Fourth step is ensuring the orientation invariance. At this stage, we have a set of stable keypoints for the images. We will now assign an orientation to each of these keypoints so that they are invariant to rotation. We can again divide this step into two smaller steps:

- Calculate the magnitude and orientation
- Create a histogram for magnitude and orientation

For more information check the linked article.

Feature points and areas

Feature matching algorithms:

- SIFT
- SURF
- BRISK
- FREAK
- MSER

- ORB

What applications do the Features have

More or less everything

- Motion estimation
- Localization
- Mapping
- Photogrammetry
- Image Retrieval
- Machine Learning:
 - Object Detection
 - Recognition
- Autonomous Driving
- etc.

Fitting Data to a Model (Handling Outliers)

Least Squares

Least Squares describes the process of minimizing the loss function by finding the partial derivative of that loss function.

Hough Transform Article

Hough transform is a feature extraction method for detecting simple shapes such as circles, lines etc in an image.

The main advantage of using the Hough transform is that it is insensitive to occlusion.

It's important to use polar coordinates when applying the hough transform method, to have bounded parameters. Otherwise we could have unbounded parameters ranging from -infinity - infinity.

Initialization of the hough space

RANSAC Article

The Random Sample Consensus (RANSAC) algorithm proposed by Fischler and Bolles is a general parameter estimation approach designed to cope with a large proportion of outliers in the input data. Its basic operations are:

- Select sample set
- Compute model
- Compute and count inliers
- Repeat until sufficiently confident

The RANSAC steps in more details are:

- Select randomly the minimum number of points required to determine the model parameters.
- Solve for the parameters of the model.

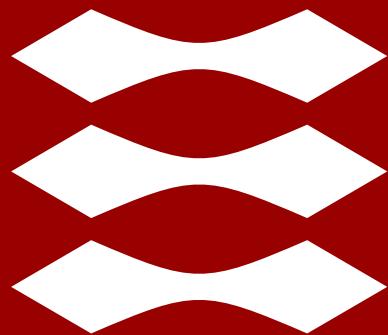
- Determine how many points from the set of all points fit with a predefined tolerance.
- If the fraction of the number of inliers over the total number of points in the set exceeds a predefined threshold, re-estimate the model parameters using all the identified inliers and terminate.
- Otherwise, repeat steps 1 through 4 (maximum of N times).

Briefly, RANSAC uniformly at random selects a subset of data samples and uses it to estimate model parameters. Then it determines the samples that are within an error tolerance of the generated model.

These samples are considered as agreed with the generated model and called as consensus set of the chosen data samples. Here, the data samples in the consensus are behaved as inliers and the rest as outliers by RANSAC. If the count of the samples in the consensus is high enough, it trains the final model of the consensus by using them.

It repeats this process for a number of iterations and returns the model that has the smallest average error among the generated models. As a randomized algorithm, RANSAC does not guarantee to find the optimal parametric model with respect to the inliers. However, the probability of reaching the optimal solution can be kept over a lower bound by assigning suitable values to algorithm parameters.

DTU



Lazaros Nalpantidis

Stereo Vision

some slides borrowed or adapted from:

- Noah Snavely
- Aaron Bobick
- Antonio Torralba

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

What is Stereo Vision?



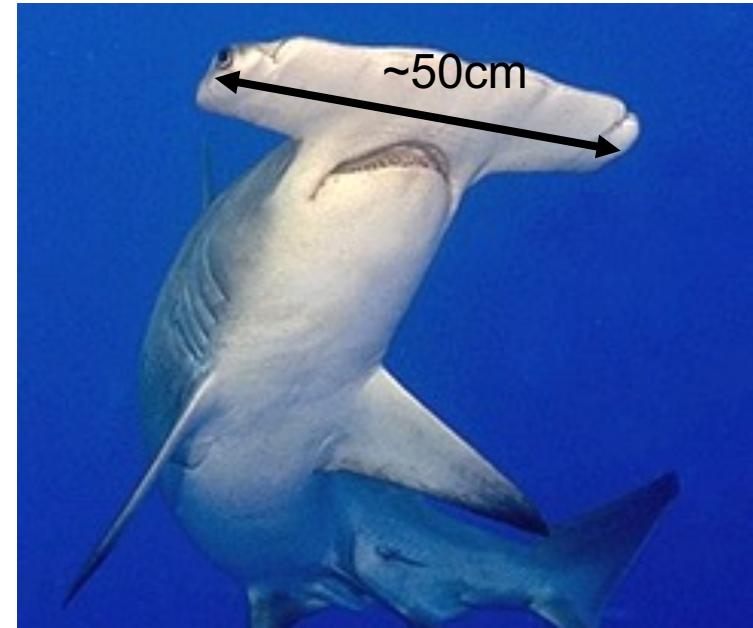
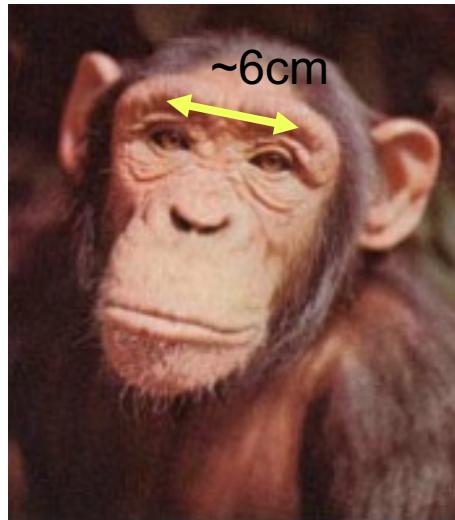
two eyes is minimum for depth perception

What is Stereo Vision?

- 3D cinema
- 3D television
- ...



What is Stereo Vision?

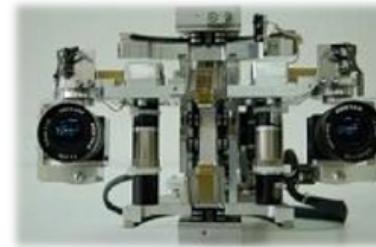
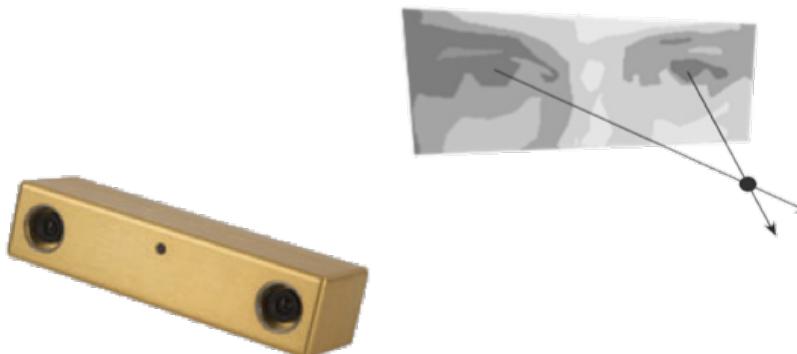


What is Stereo Vision?

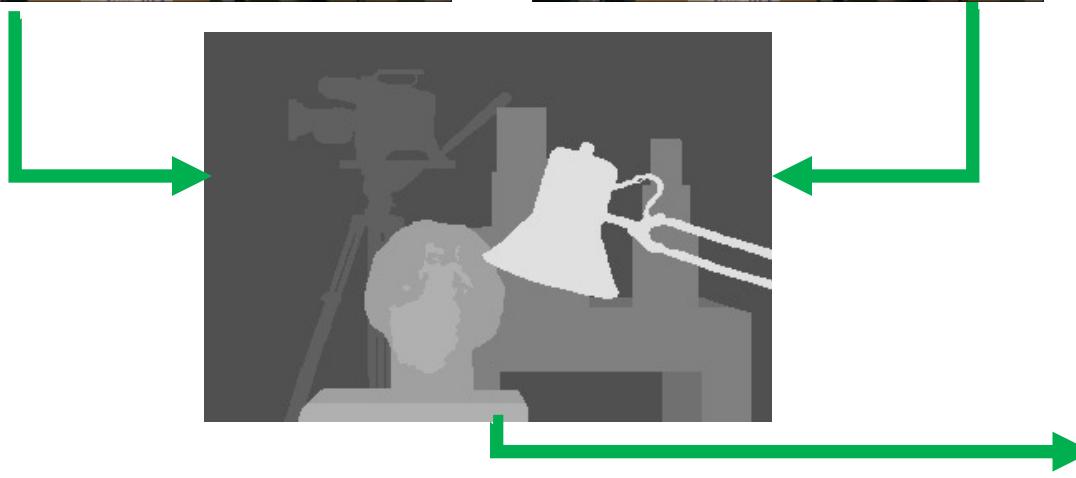
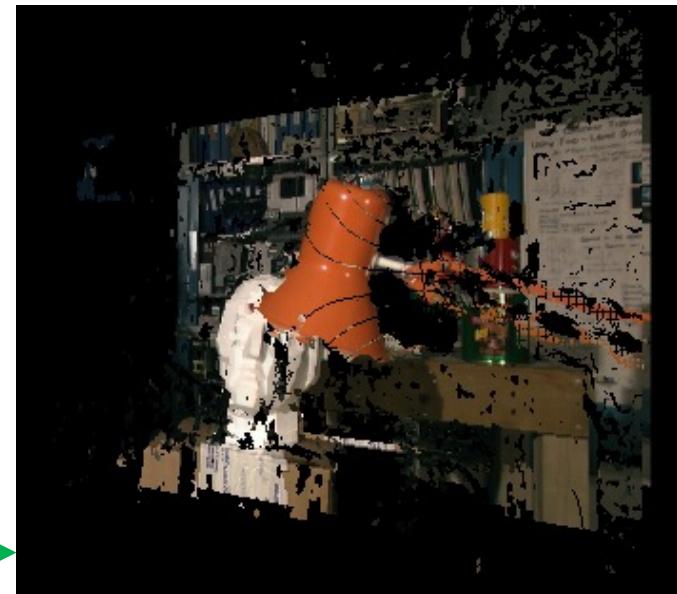


”... the mind perceives an object of three-dimensions by means of the two dissimilar pictures projected by it on the two retinae...”

Sir Charles Wheatstone, 1838

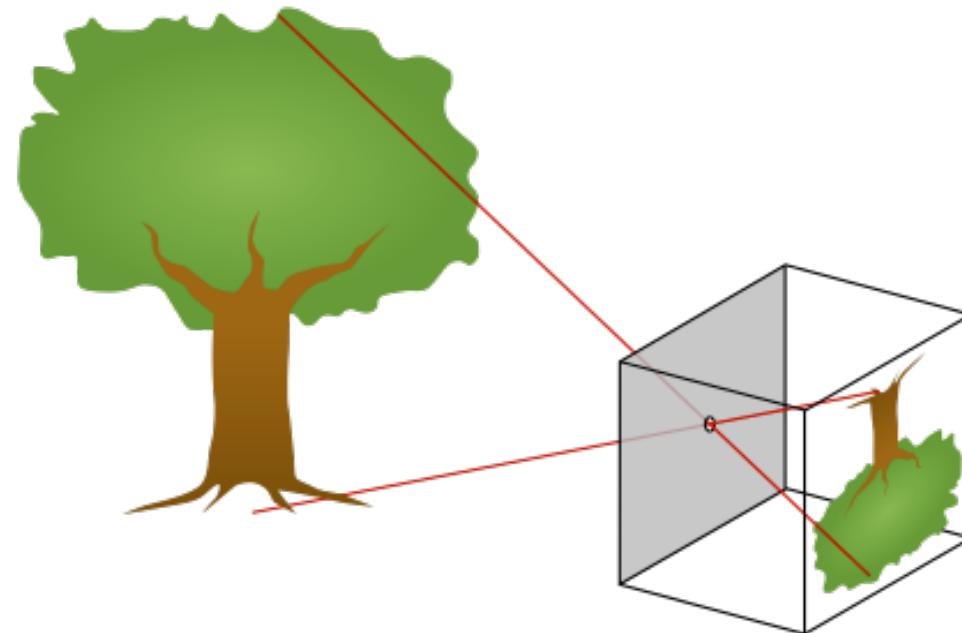


Stereo Vision Computation



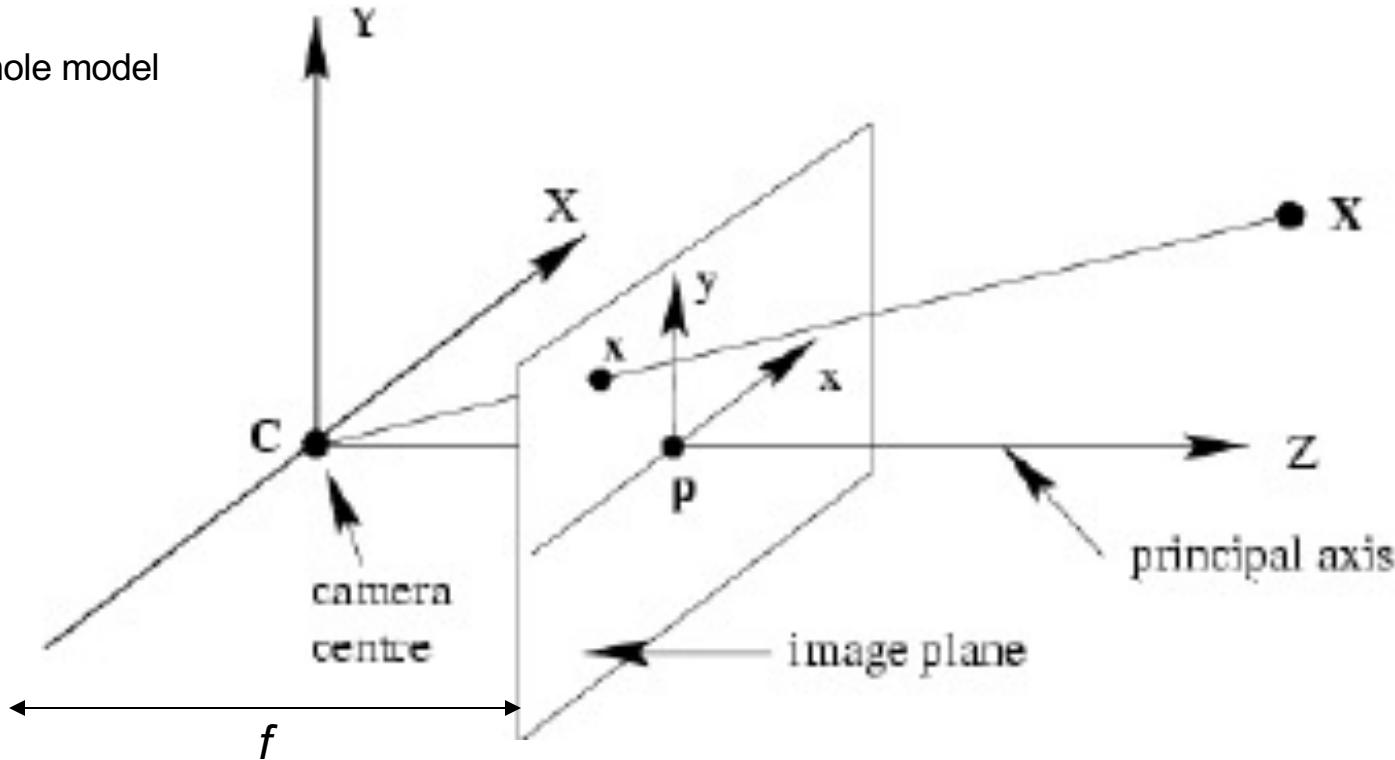
- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

- Pinhole model

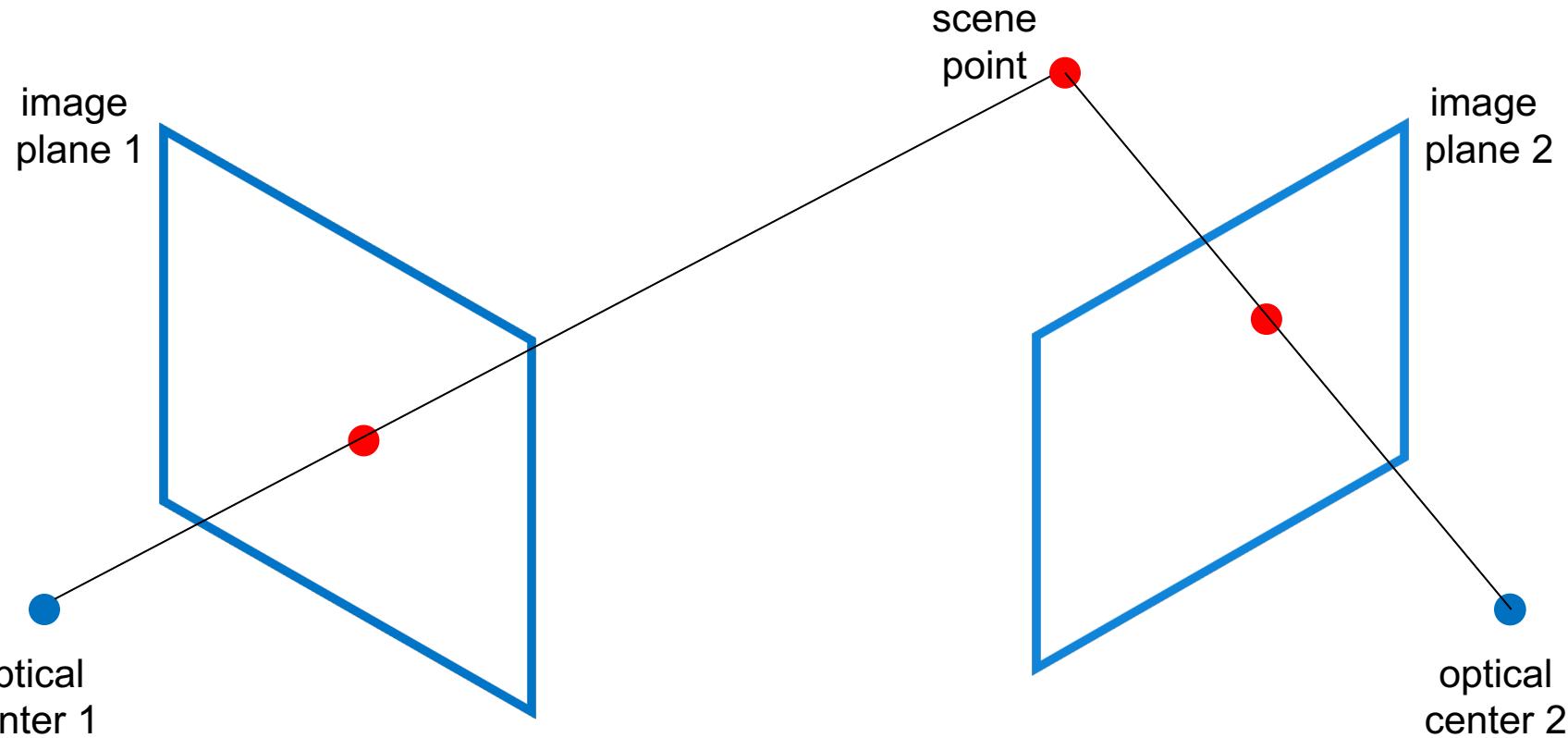


Camera Model

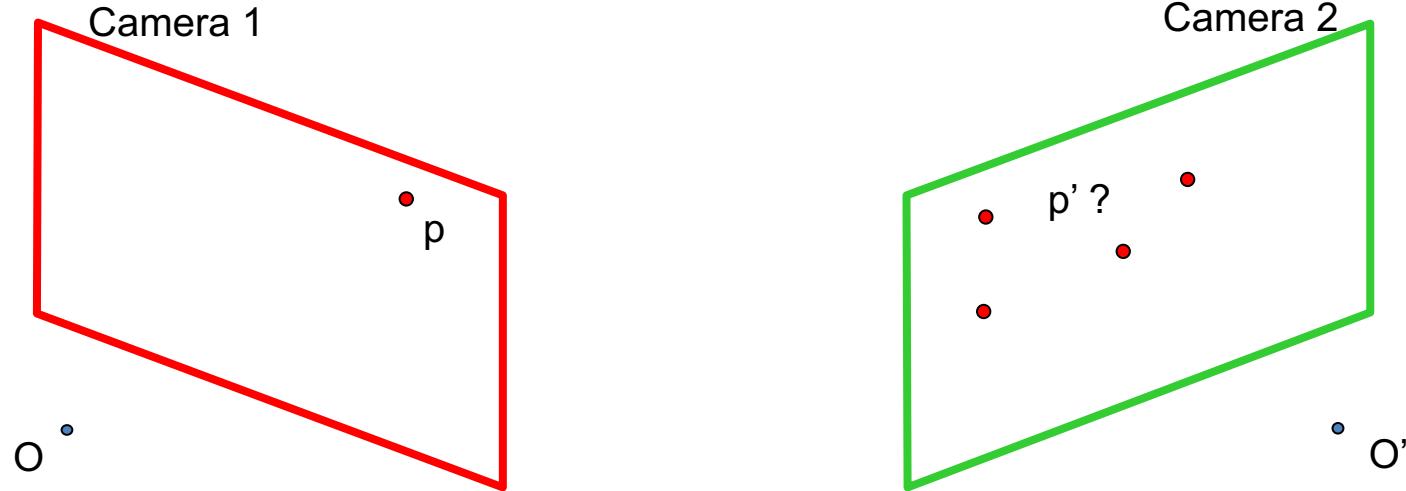
- Pinhole model



Epipolar Geometry

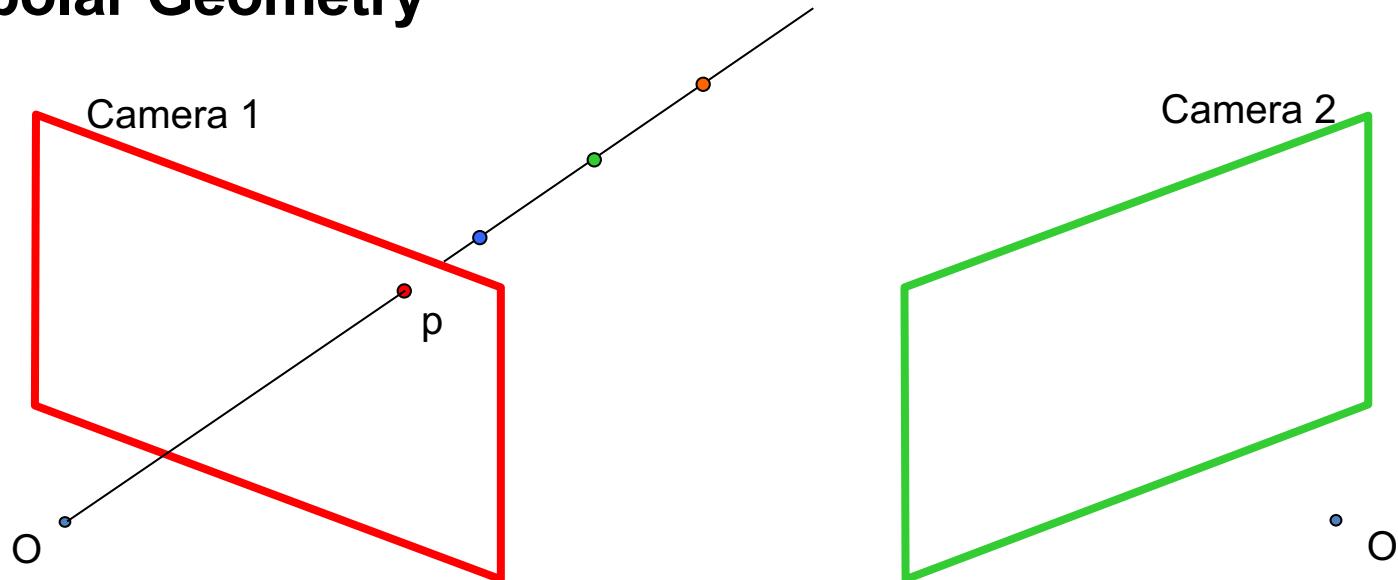


Epipolar Geometry



If we see a point in camera 1, are there any constraints on where we will find it on camera 2?

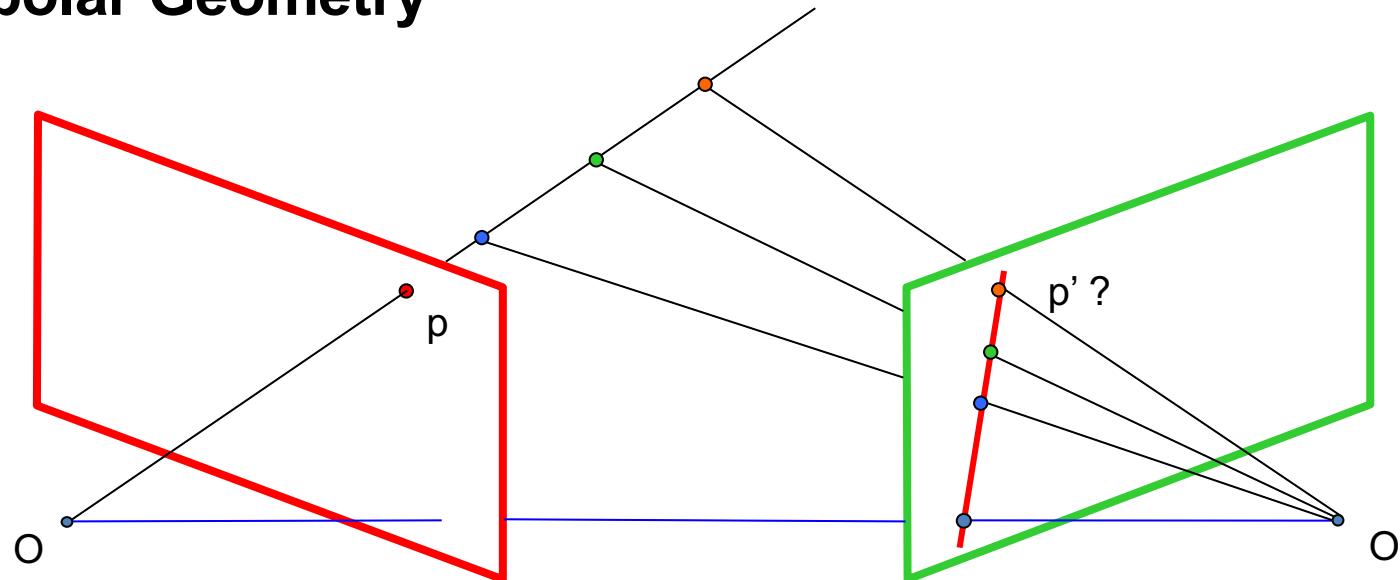
Epipolar Geometry

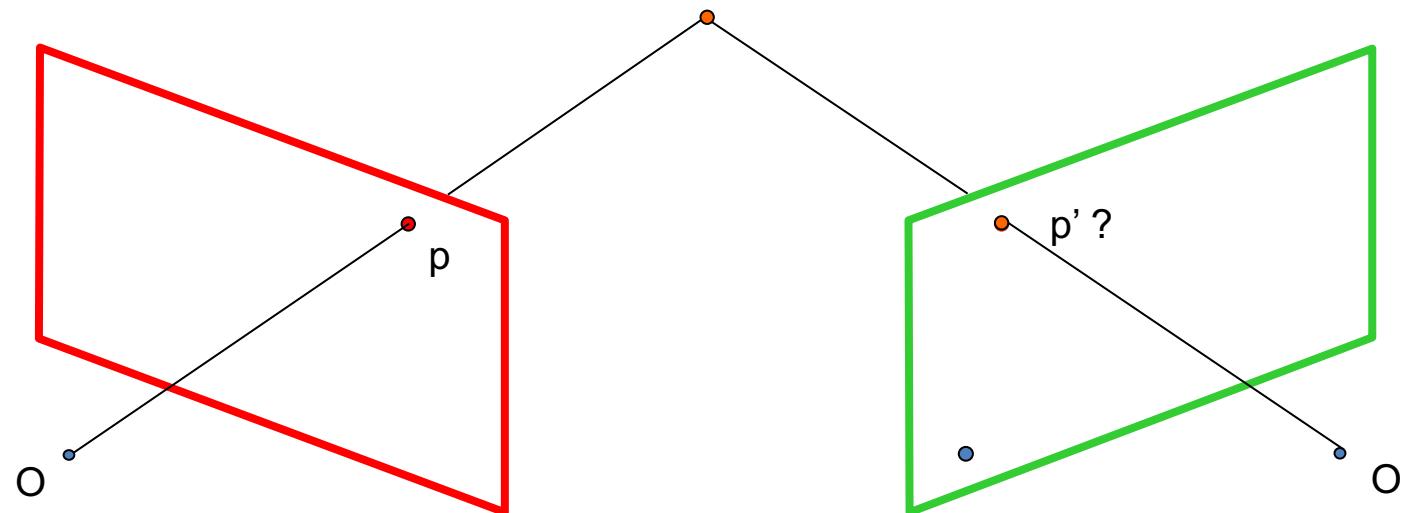


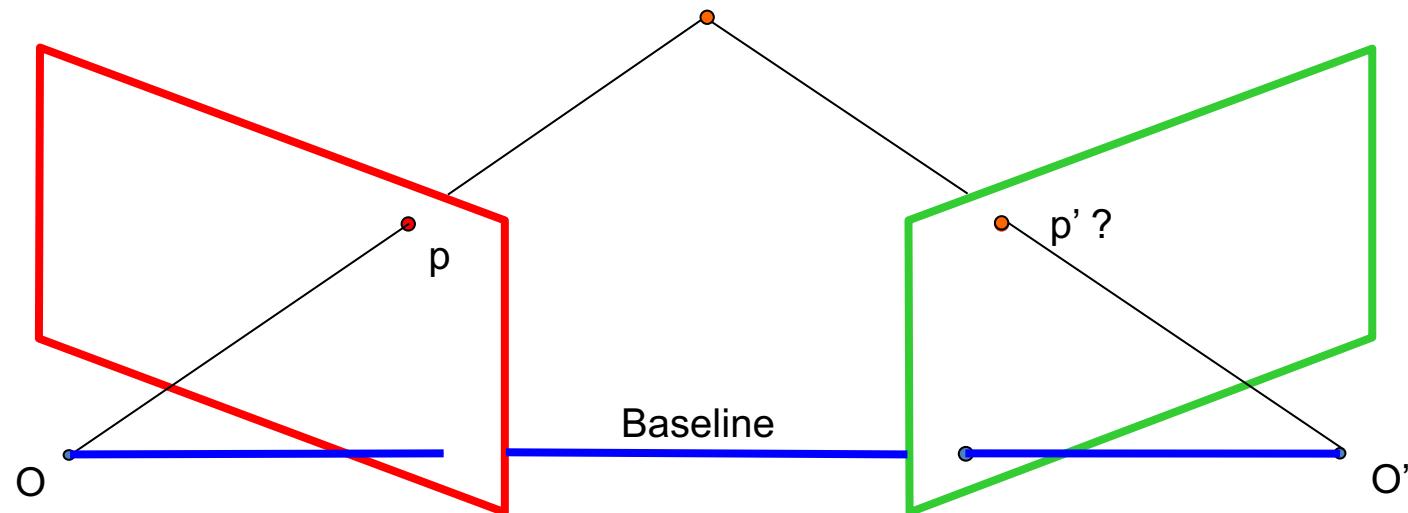
If we see a point in camera 1, are there any constraints on where we will find it on camera 2?



Epipolar Geometry

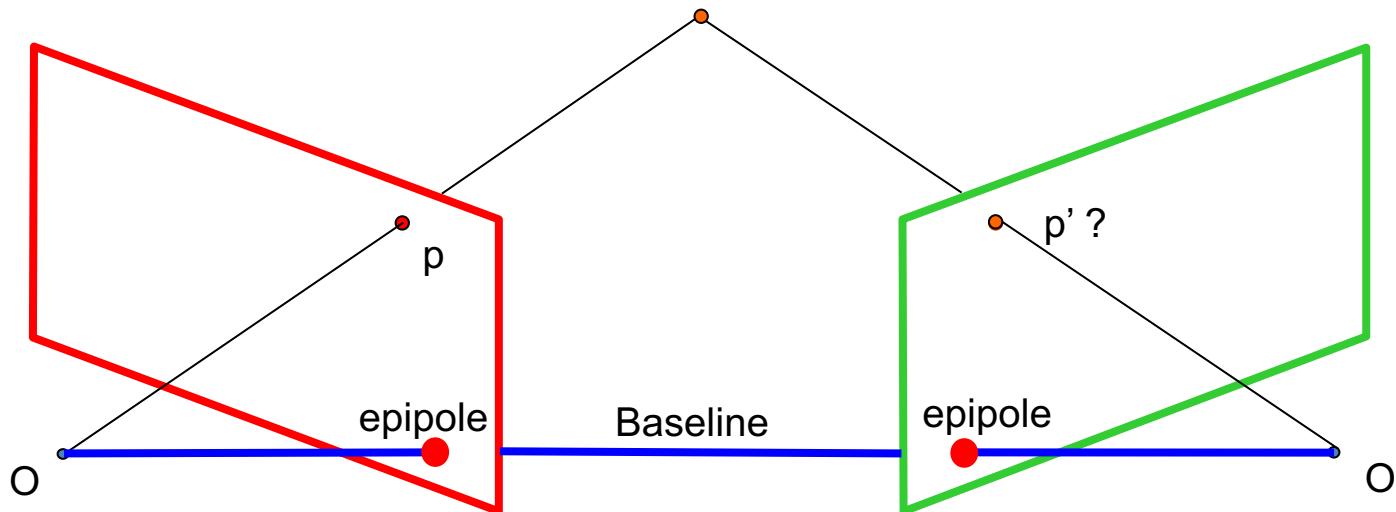






Baseline: the line connecting the two camera centers

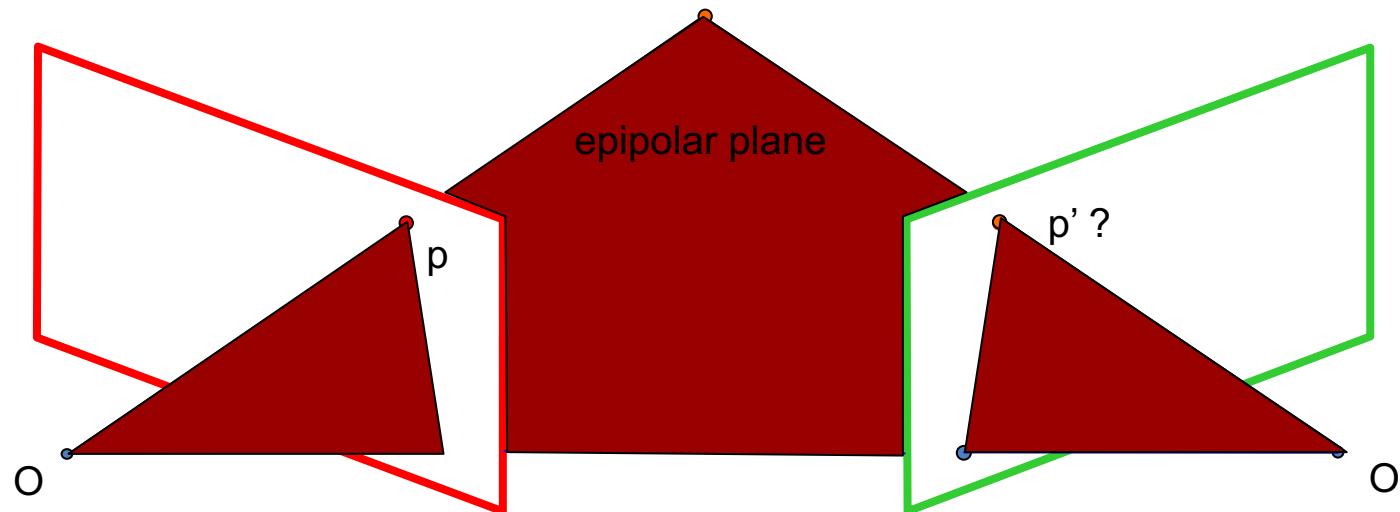
Epipolar Geometry



Baseline: the line connecting the two camera centers

Epipole: point of intersection of *baseline* with the image plane

Epipolar Geometry

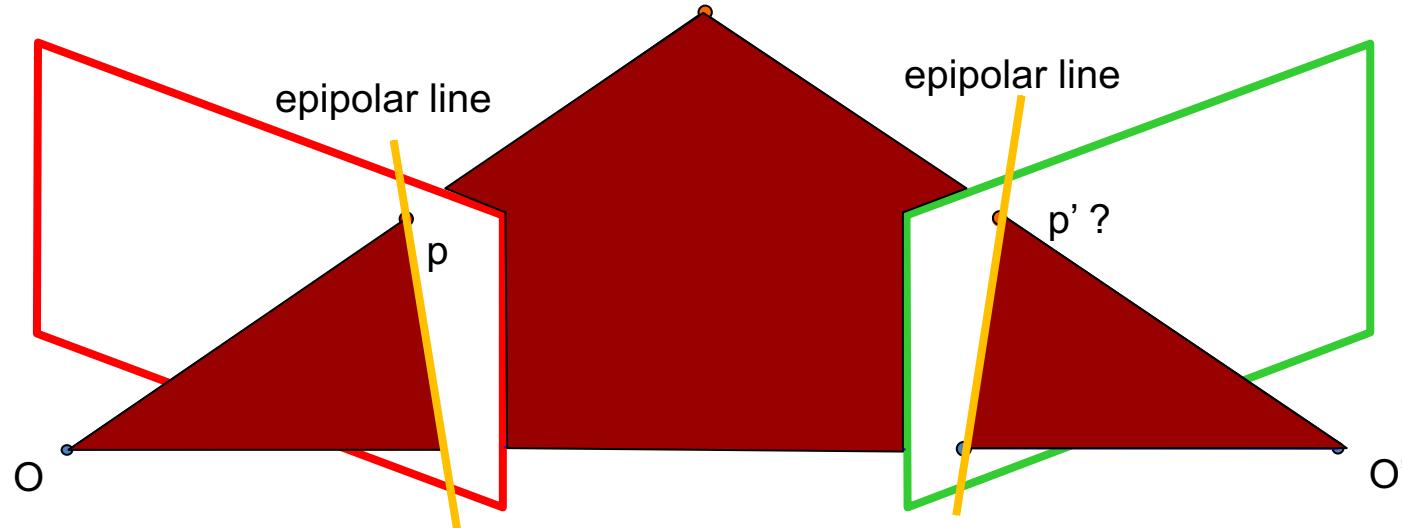


Baseline: the line connecting the two camera centers

Epipole: point of intersection of *baseline* with the image plane

Epipolar plane: the plane that contains the two camera centers and a 3D point in the world

Epipolar Geometry



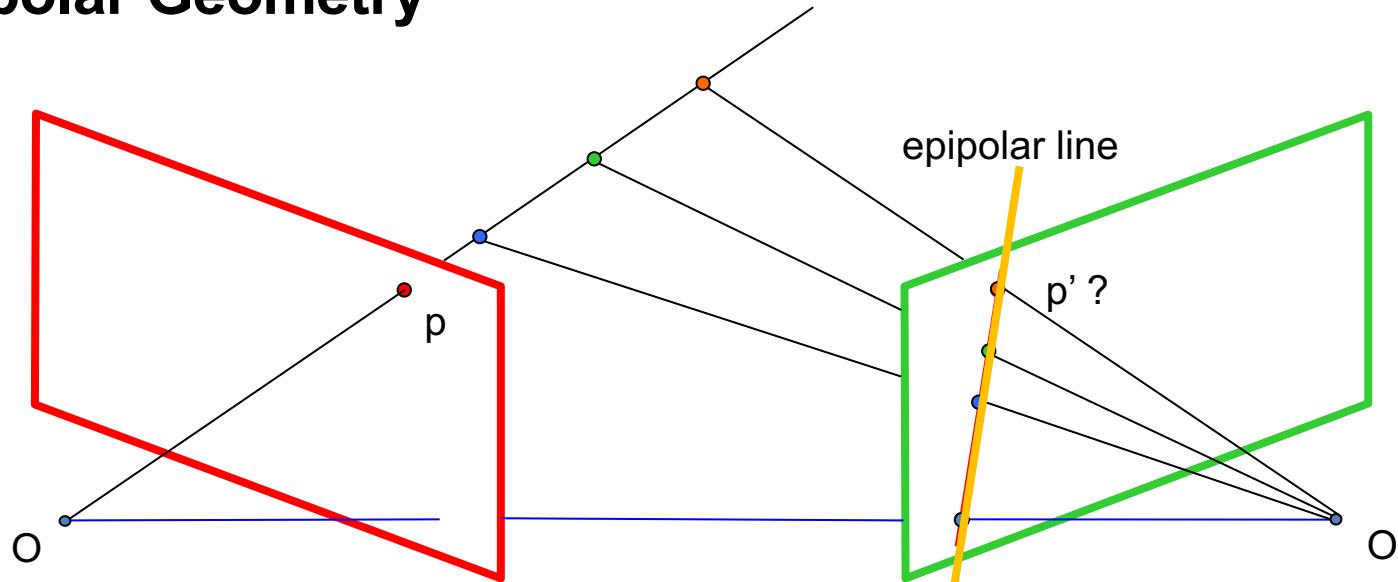
Baseline: the line connecting the two camera centers

Epipole: point of intersection of *baseline* with the image plane

Epipolar plane: the plane that contains the two camera centers and a 3D point in the world

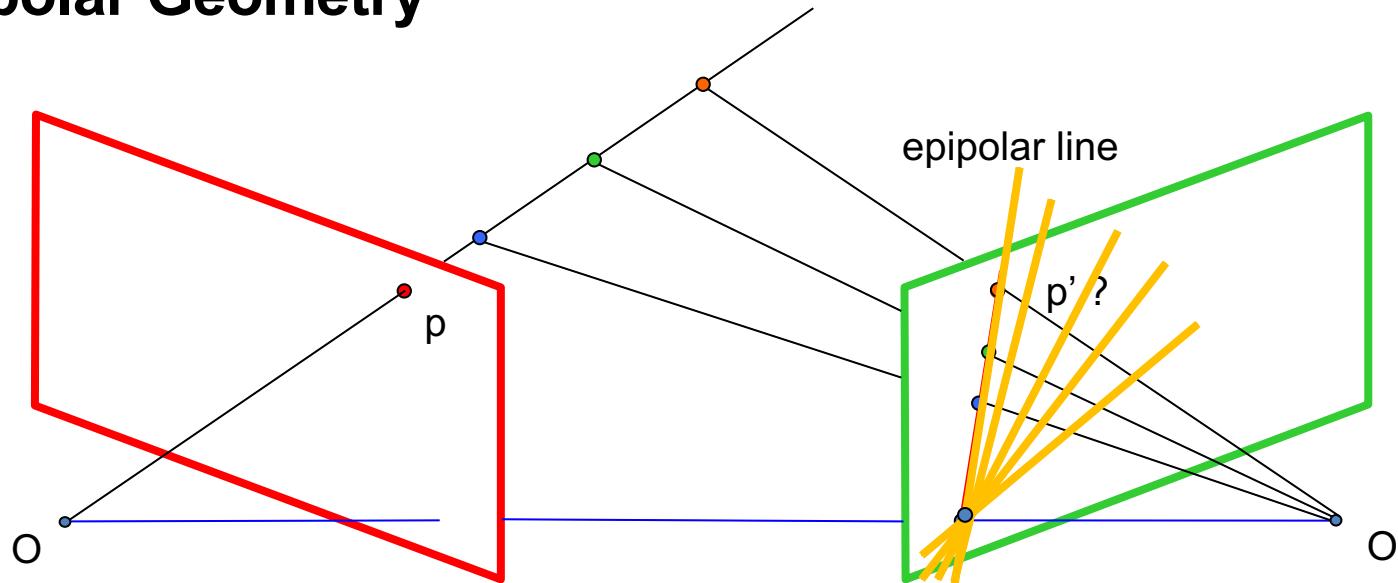
Epipolar line: intersection of the *epipolar plane* with each image plane

Epipolar Geometry

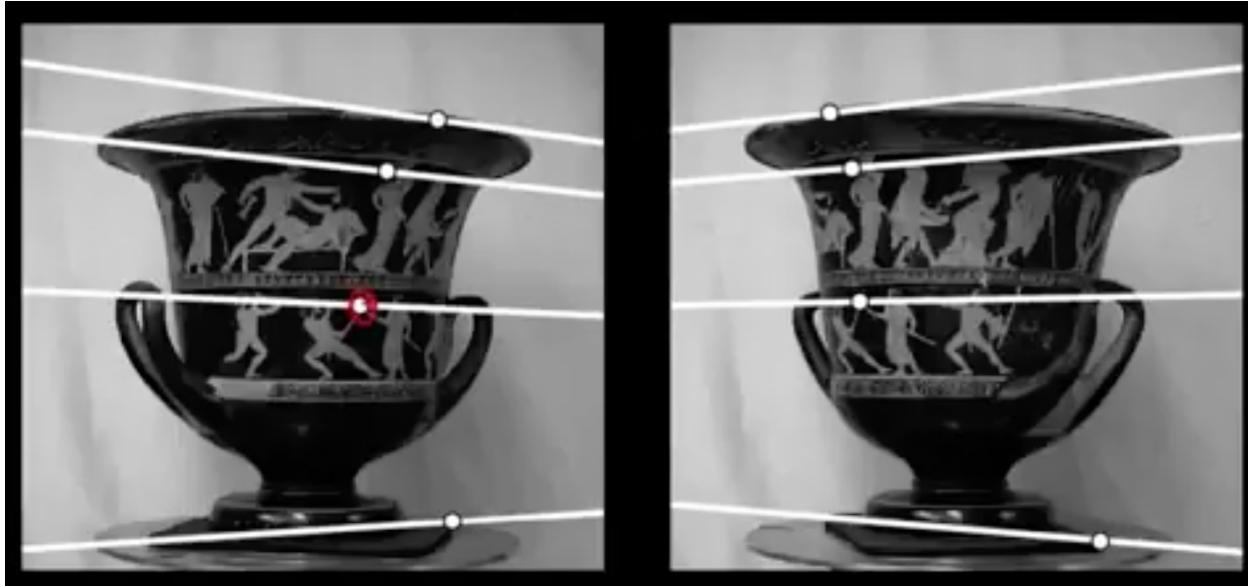


- We can search for matches across epipolar lines.
 - Search space for correspondences reduces to a 1D problem!

Epipolar Geometry



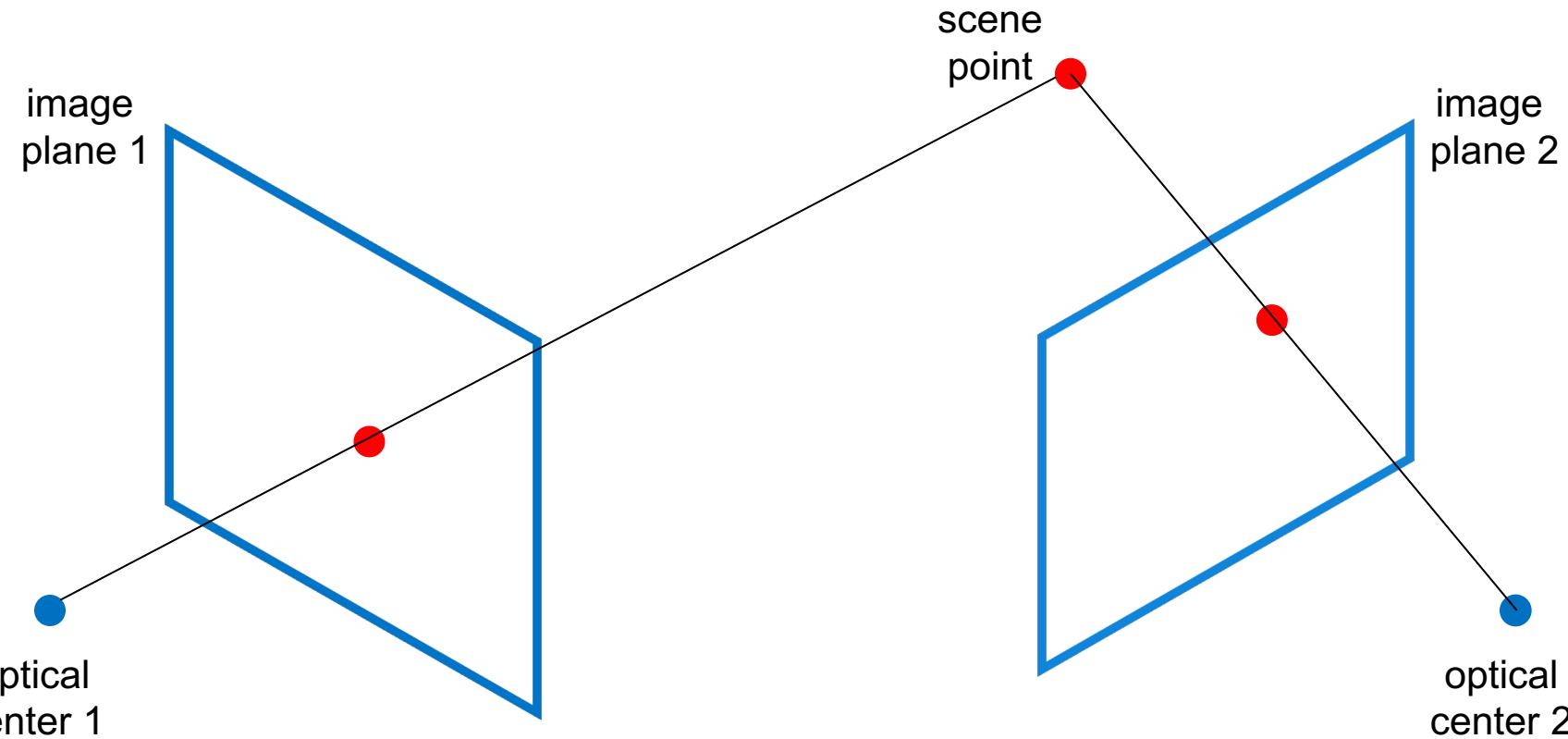
- We can search for matches across epipolar lines
 - Search space for correspondences reduces to a 1D problem!
- All epipolar lines intersect at the epipoles



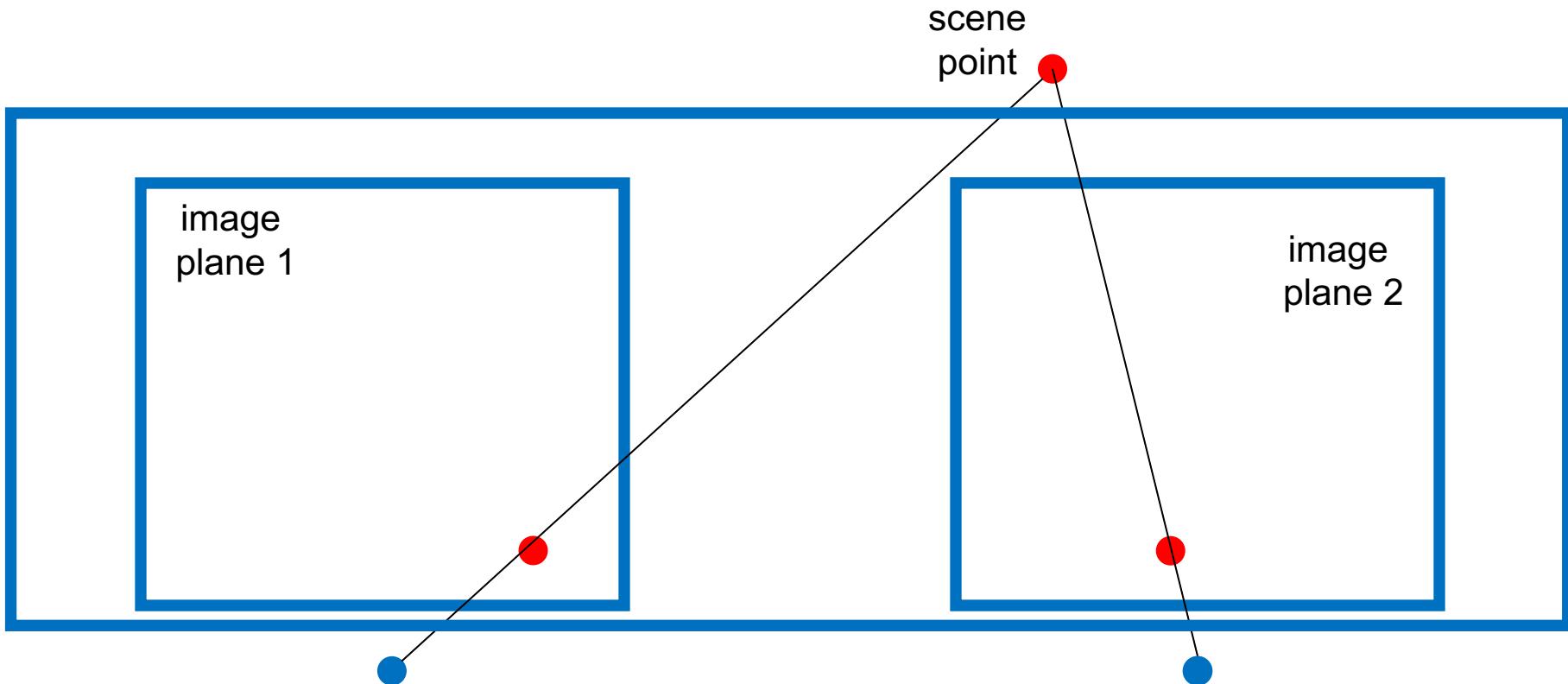
- We can search for matches across epipolar lines
 - Search space for correspondences reduces to a 1D problem!
- All epipolar lines intersect at the epipoles
 - **Where are the epipoles in this case?**

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

Rectified Stereo – a simpler case

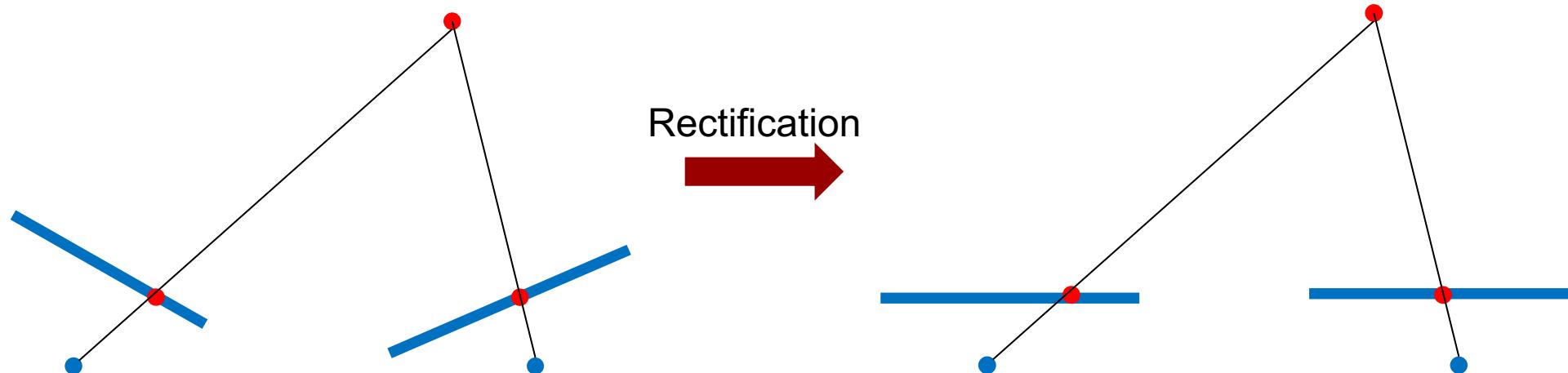


Rectified Stereo – a simpler case

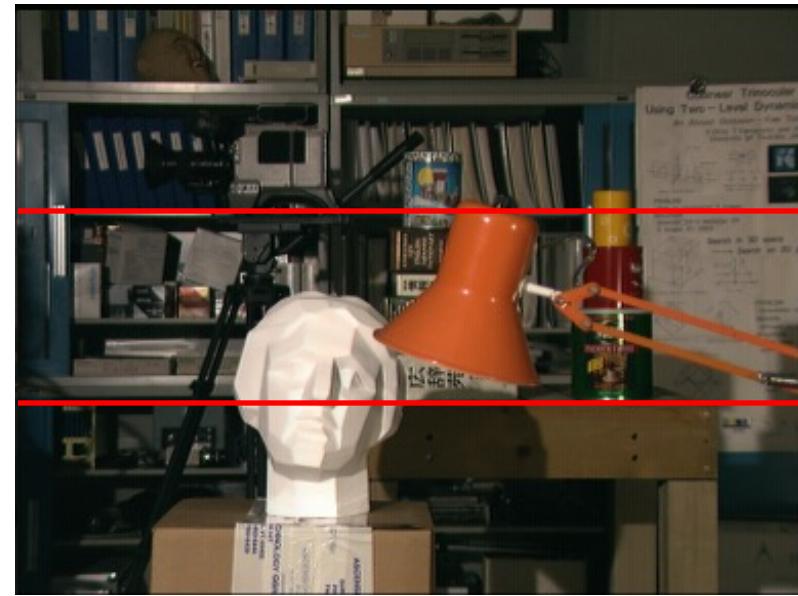
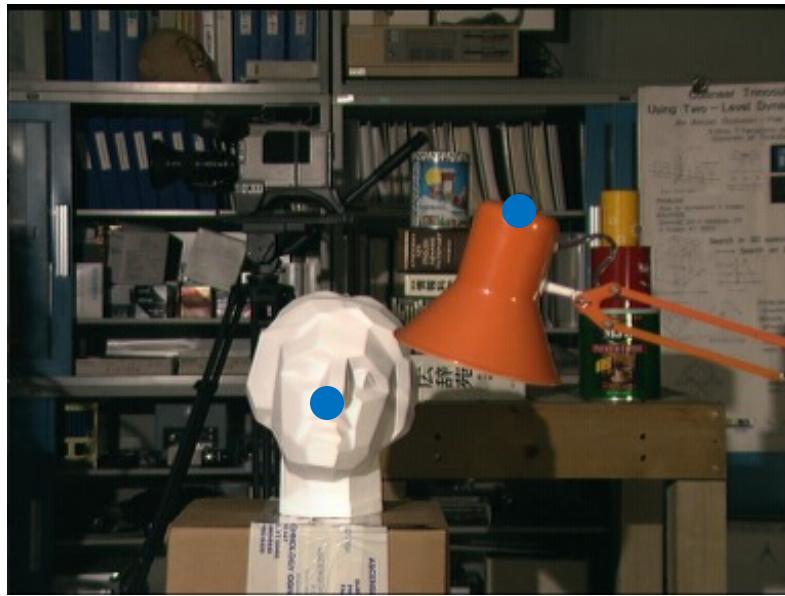


Rectified Stereo – a simpler case

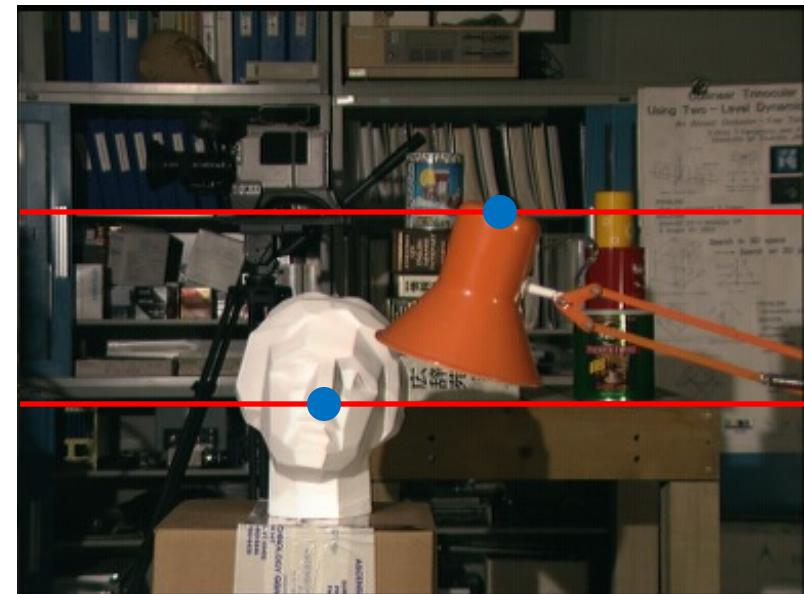
- Rectification:
 - The initial images are reprojected on a common plane that is parallel to the baseline B joining the optical centers of the initial images.
 - Epipolar lines become parallel (and under certain conditions they become also horizontal)



Rectified Stereo



Rectified Stereo

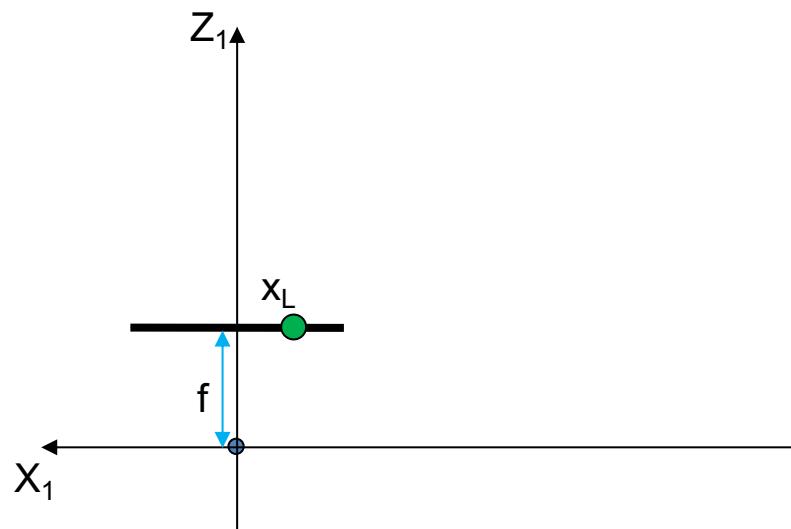


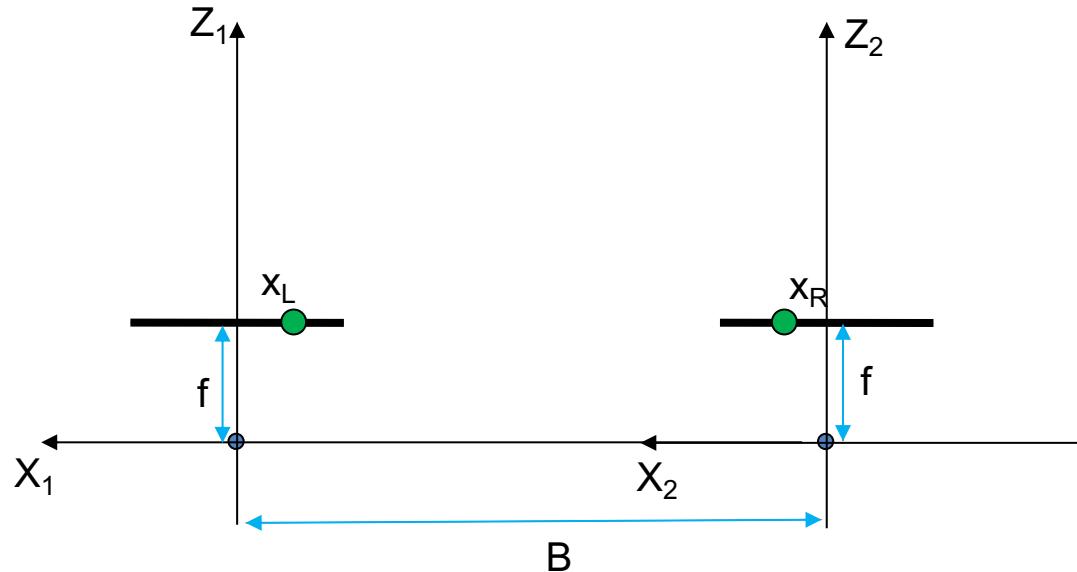
- “All epipolar lines intersect at the epipoles”
 - **Where are the epipoles in this case?**

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

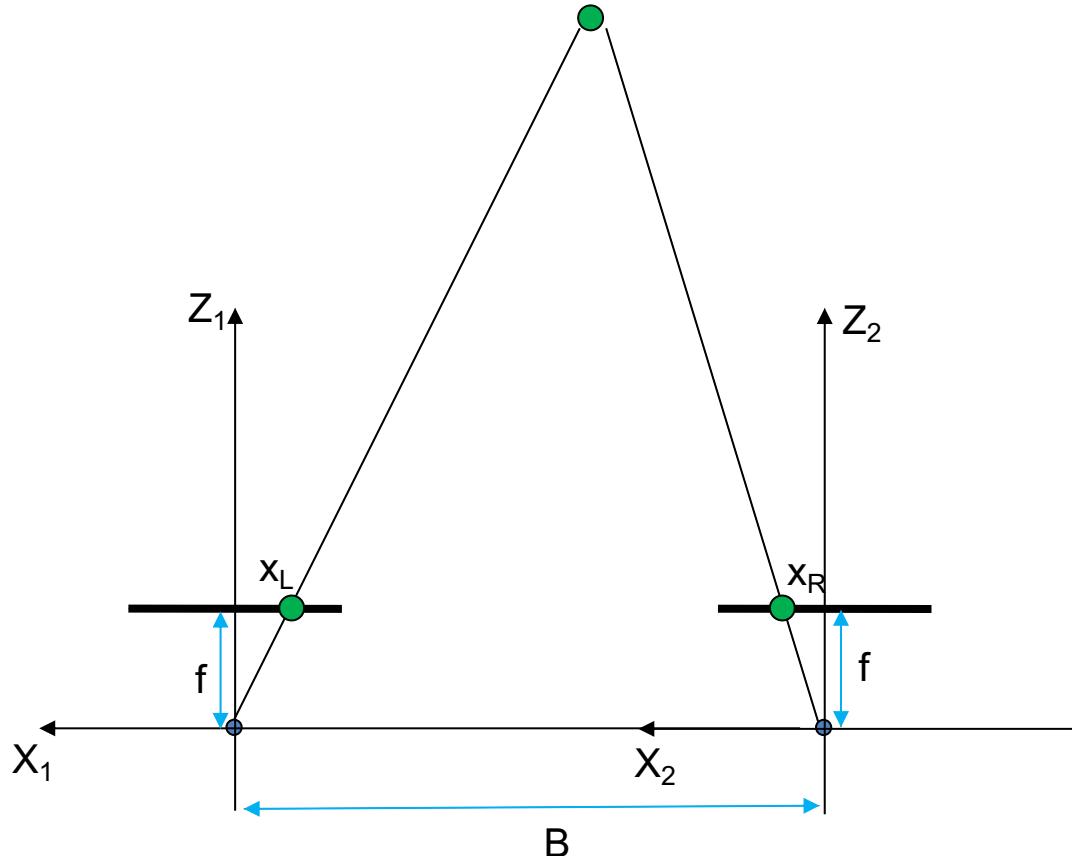
Depth from Stereo Matches

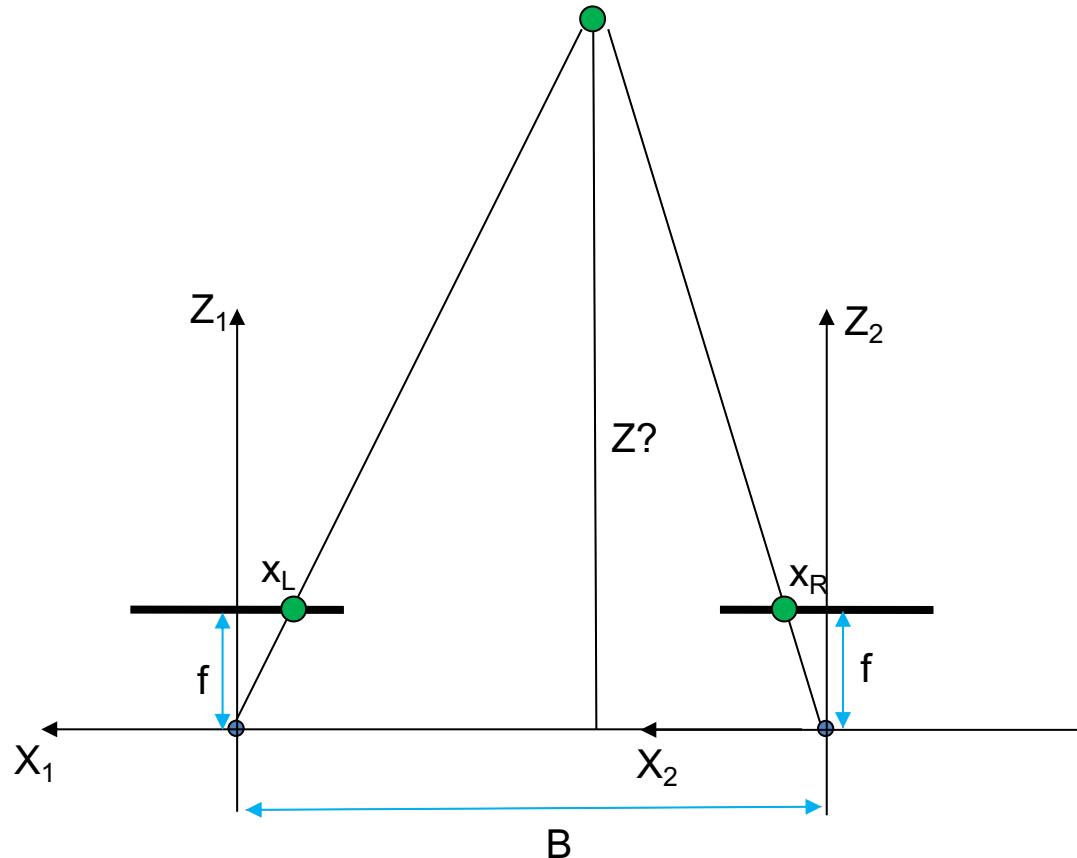
- Let us assume (for now!!) that:
 - we can check the points along the epipolar line and
 - we can find the point (on the right image) that is most similar to our reference point (in the left image), i.e. we can solve the correspondence problem!



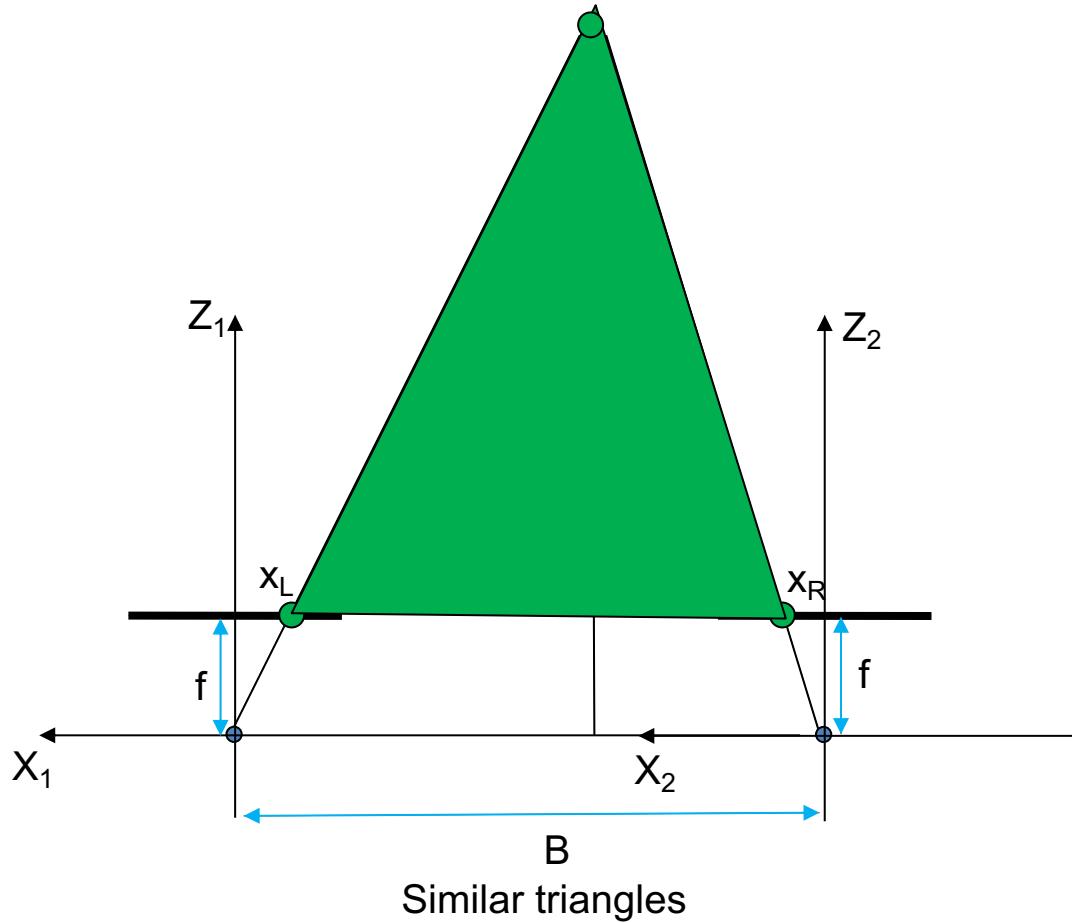


Depth from Stereo Matches

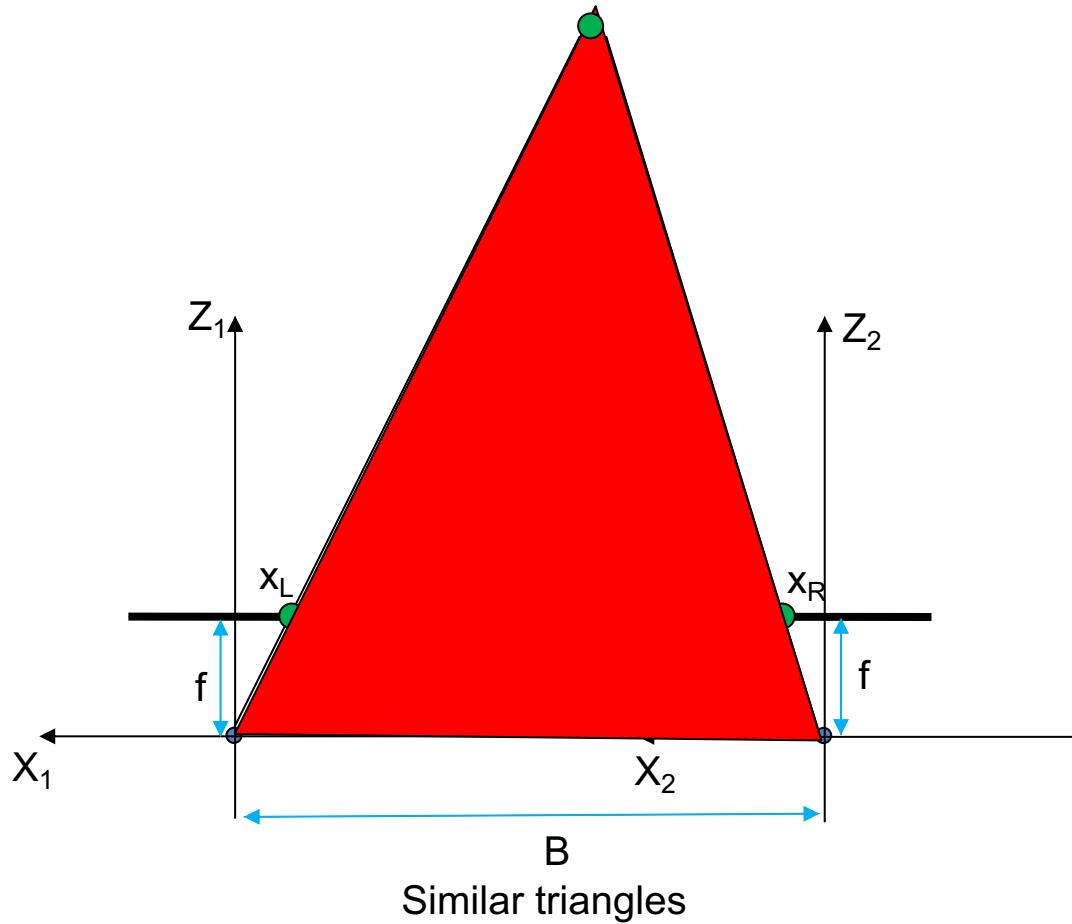




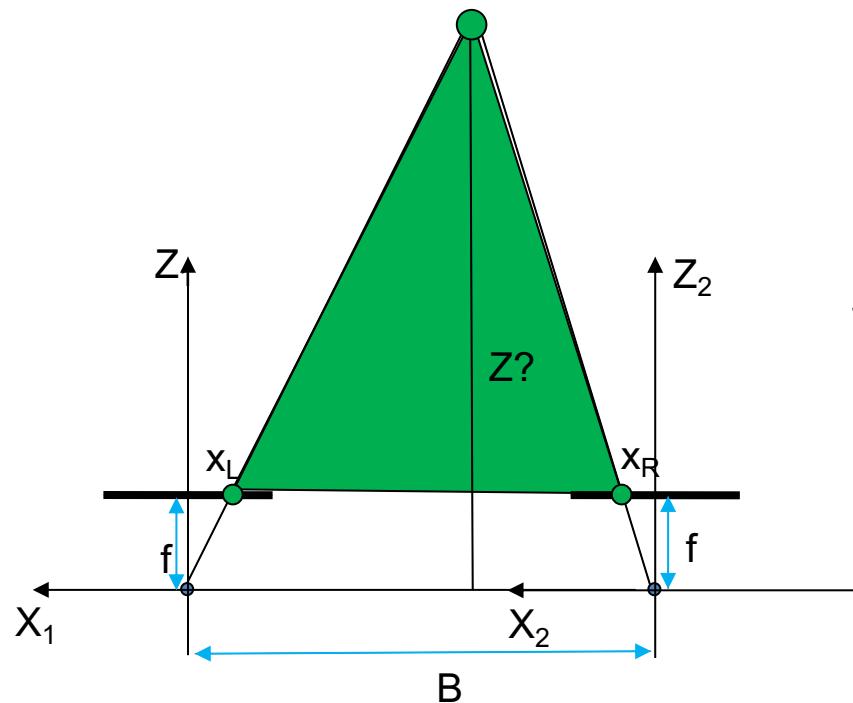
Depth from Stereo Matches



Depth from Stereo Matches



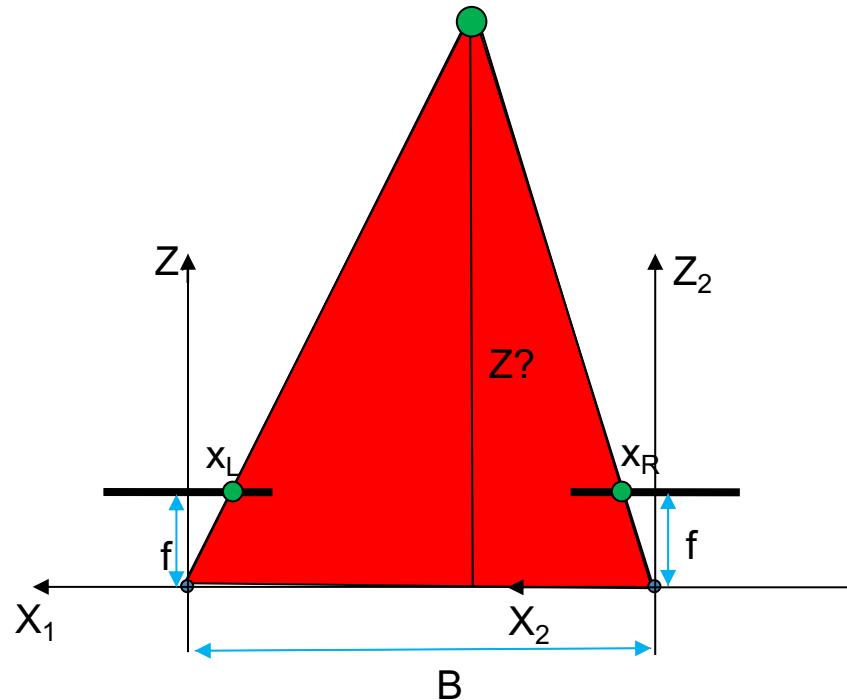
Depth from Stereo Matches



Similar triangles:

$$\frac{B - X_L + X_R}{Z - f} =$$

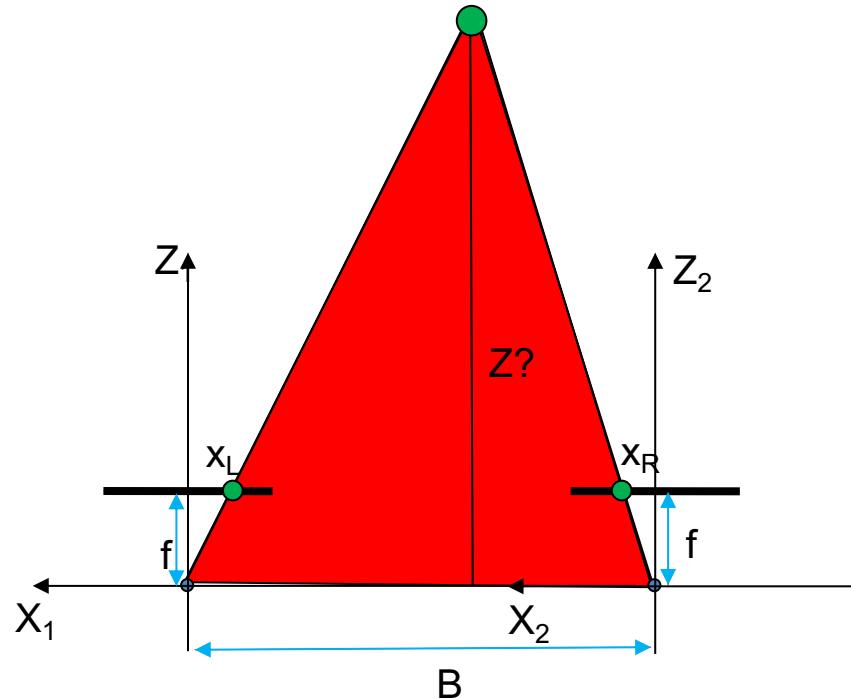
Depth from Stereo Matches



Similar triangles:

$$\frac{B - x_L + x_R}{Z - f} = \frac{B}{Z}$$

Depth from Stereo Matches



Similar triangles:

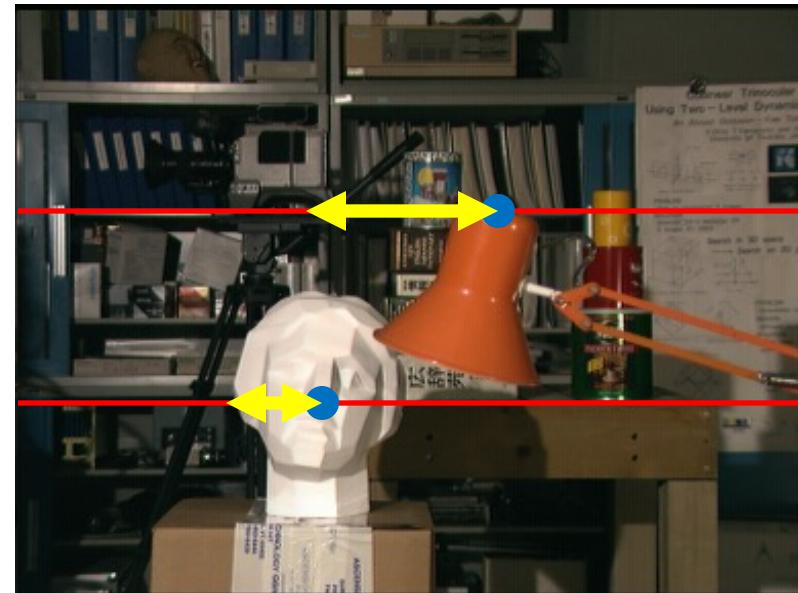
$$\frac{B - x_L + x_R}{Z - f} = \frac{B}{Z}$$

Solving for Z :

$$Z = f \frac{B}{x_L - x_R}$$

Disparity

Depth from Stereo Matches



Depth from Stereo Matches

depth map will be the opposite
closer is dark, further is white

Reference Image



Disparity Map



$$Z = f \frac{B}{X_L - X_R}$$

Disparity

A diagram illustrating the relationship between disparity and depth. It shows a red elliptical object representing a scene patch. A green arrow points from the center of the ellipse towards the bottom left, labeled "Disparity". Above the ellipse, the formula $Z = f \frac{B}{X_L - X_R}$ is written, where Z is depth, f is focal length, B is baseline, and $X_L - X_R$ is the horizontal disparity.

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

Correspondence Problem

- We have assumed (up to now!!) that:
 - we can check the points along the epipolar line and
 - we can find the point (on the right image) that is **most similar** to our reference point (in the left image), i.e. we can **solve the correspondence problem!**
- *How can we indeed match corresponding pixels between the two stereo images?*

Correspondence Problem

- Beyond the hard constraint of epipolar geometry, there are “soft” constraints to help identify corresponding points
 - Similarity
 - Uniqueness
 - Ordering
 - Disparity gradient is limited

Correspondence Problem

- To find matches in the image pair, we will assume
 - Most scene points visible from both views
 - Image regions for the matches are similar in appearance

Correspondence Problem

- It depends!
 - Do we need **dense** or **sparse** stereo matching?

Stereo Vision

Sparse output

Dense output

- Local Methods (Area-based)
- Global Methods (Energy-based)
- Other Methods

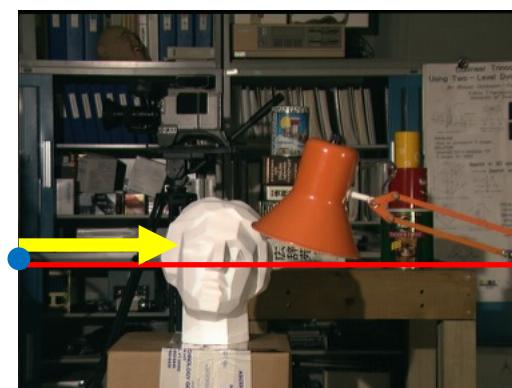
Sparse Stereo Correspondence

- Extract features (e.g. SIFT, SURF, Harris,...) and match them!

- Pros? works
- Cons? computationally heavy

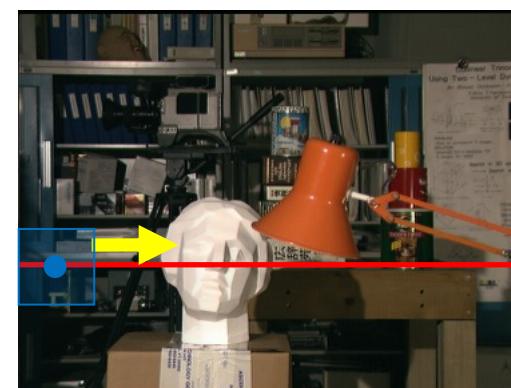
Dense Stereo Correspondence : Local Methods

- Try to find correspondences for all the pixels of the reference image.
- For each epipolar line
 - For each pixel in the left image
 - Compare with every pixel on same epipolar line in right image
 - Choose the pixel that maximizes a similarity metric (or minimizes a dissimilarity metric!).



Dense Stereo Correspondence

- Try to find correspondences for all the pixels of the reference image.
- For each epipolar line
 - For each pixel in the left image
 - Compare with every pixel on same epipolar line in right image
 - Choose the pixel that maximizes a similarity metric (or minimizes a dissimilarity metric!).
- Improvement: don't match individual pixels, but rather match windows!



Stereo Correspondence Metrics

- Sum of Absolute Differences (SAD)

$$SAD(x, y, d) = \sum_{x, y \in W} |I_l(x, y) - I_r(x, y - d)|$$

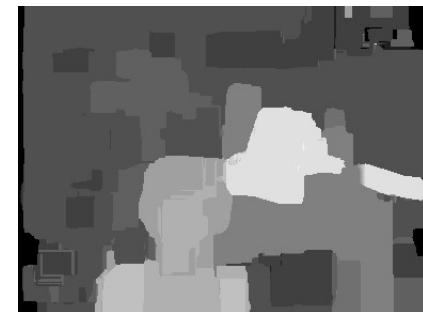
- Sum of Squared Differences (SSD)

$$SSD(x, y, d) = \sum_{x, y \in W} (I_l(x, y) - I_r(x, y - d))^2$$

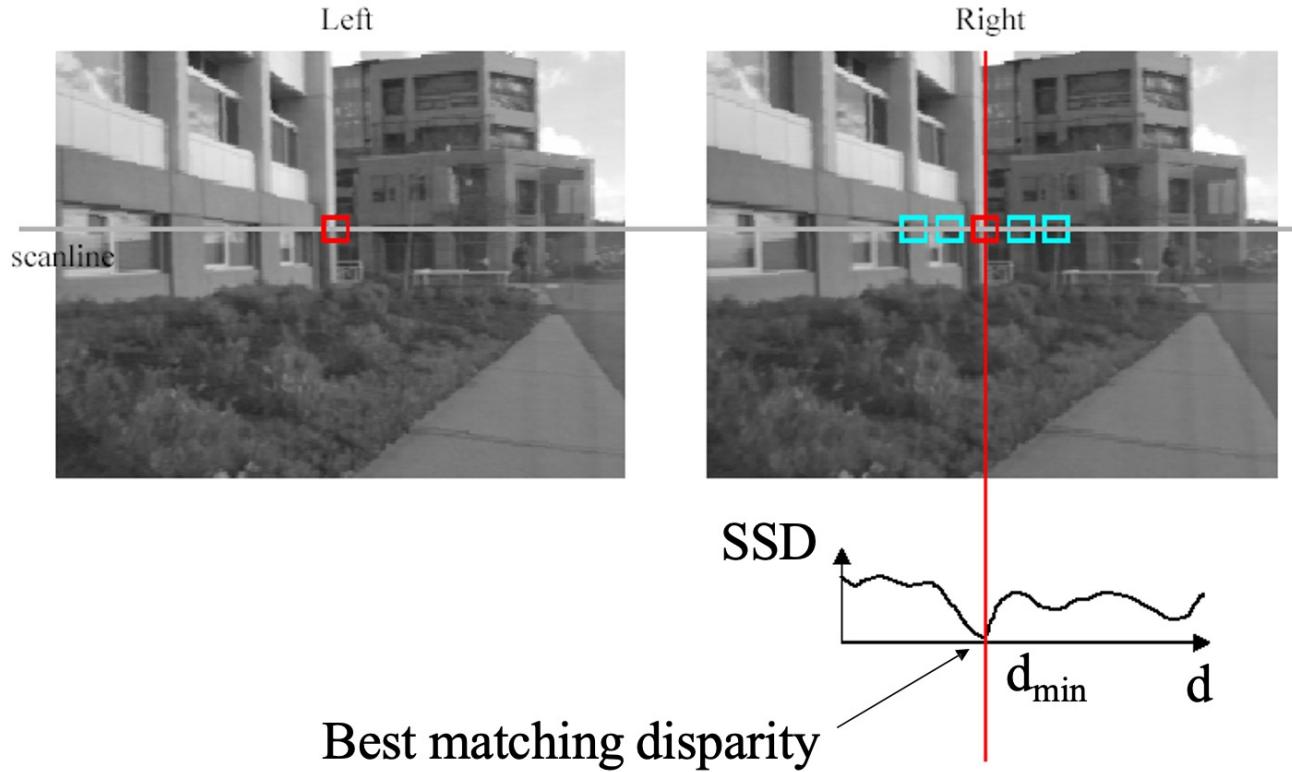
- Normalized Cross-Correlation

$$NCC(x, y, d) = \frac{\sum_{x, y \in W} I_l(x, y) \cdot I_r(x, y - d)}{\sqrt{\sum_{x, y \in W} I_l^2(x, y) \cdot \sum_{x, y \in W} I_r^2(x, y - d)}}$$

- ...many many more!!!



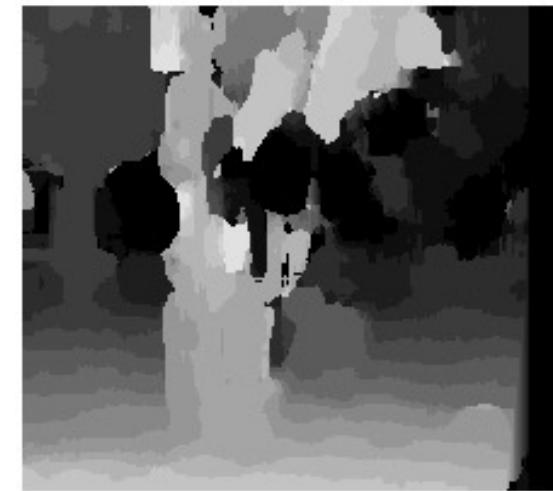
Stereo Correspondence Metrics: SSD



Stereo Correspondence Metrics: SSD on various windows



$$W = 3$$



$$W = 20$$

- Small vs Big windows smaller less significant features
- What are their Pros and Cons? feature definition

Stereo Correspondence Metrics: Good/Bad areas



- In this stereo image pair:
 - what would be good areas to match? boundaries btw colours
 - where would you expect to face problems and why? white, horizontal lines, boring windows

Global Stereo Correspondence

- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
 - Thus, those methods are called "local" or "area-based" methods.
- Example: Result of a **local** SSD algorithm with $W=21$:



Global Stereo Correspondence

- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
 - Thus, those methods are called "local" or "area-based" methods.
- Global methods find better solutions in expense of more computations
 - Optimize jointly the disparity values of all the pixels of each scanline (e.g. Dynamic Programming)



Global Stereo Correspondence

- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
 - Thus, those methods are called "local" or "area-based" methods.
- Global methods find better solutions in expense of more computations
 - Optimize jointly the disparity values of all the pixels of each scanline (e.g. Dynamic Programming)
 - Optimize jointly the disparity values of all the pixels of the image (e.g. graph cuts)

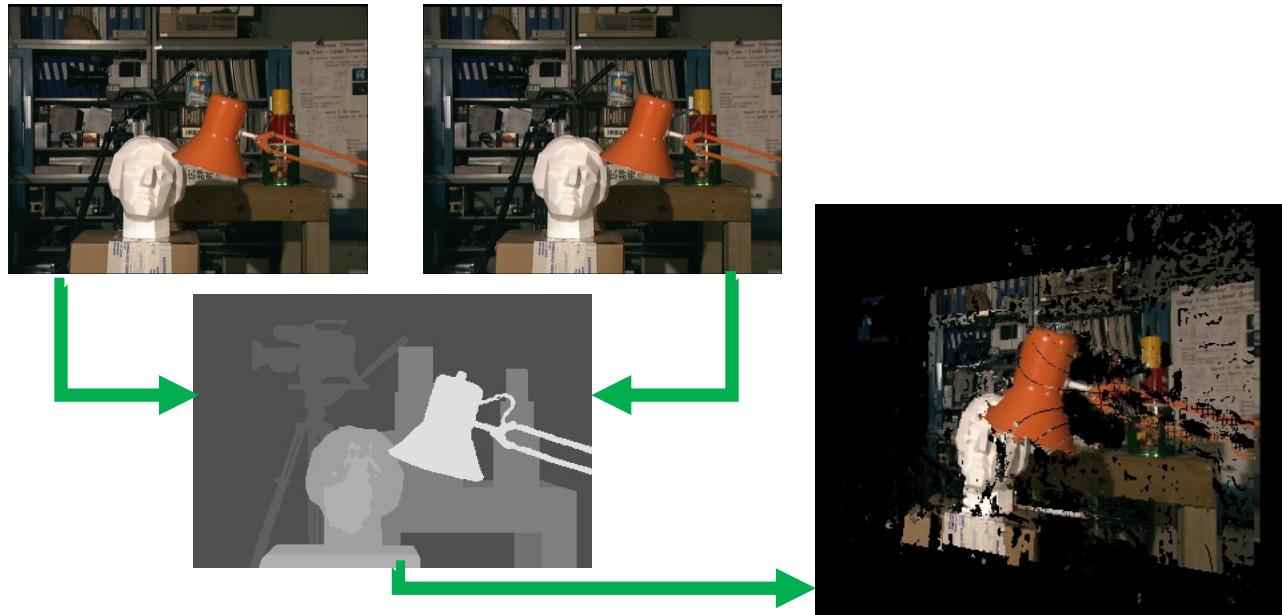


Global Stereo Correspondence

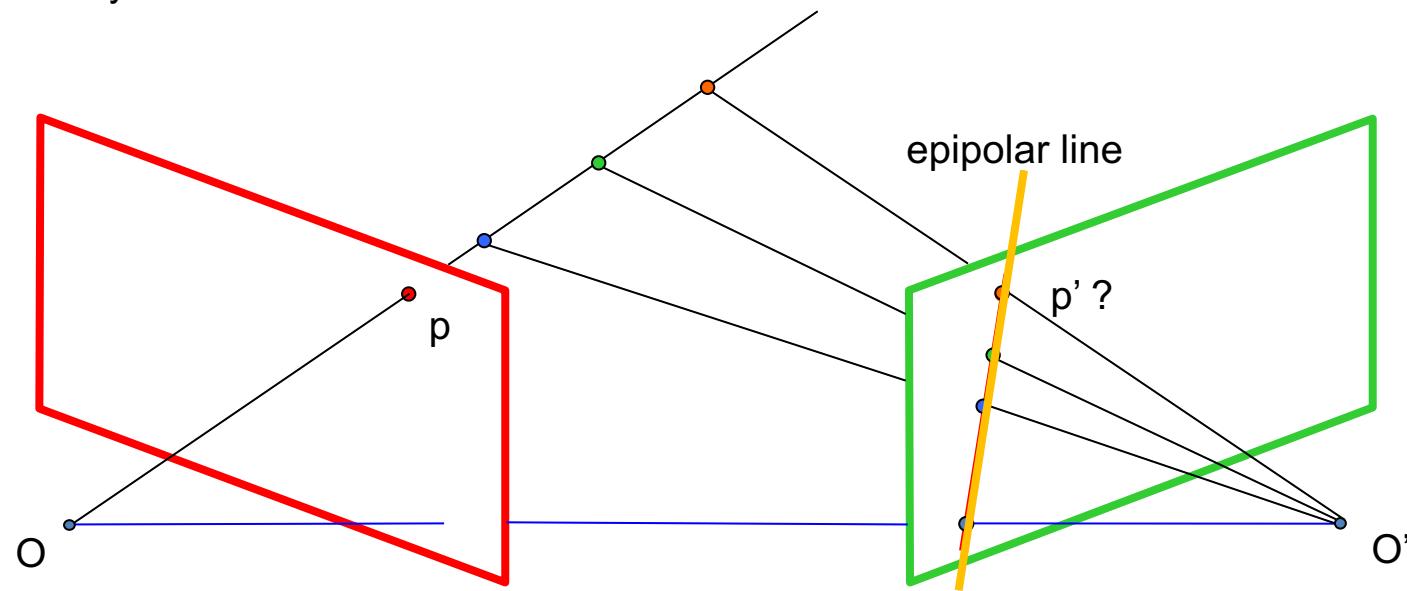
- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
 - Thus, those methods are called "local" or "area-based" methods.
- Global methods find better solutions in expense of more computations
 - Optimize jointly the disparity values of all the pixels of each scanline (e.g. Dynamic Programming)
 - Optimize jointly the disparity values of all the pixels of the image (e.g. graph cuts)
- *In global algorithms, stereo correspondence is formulated as an energy function minimization problem, consisting of data and smoothness terms.*

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures
- Summary

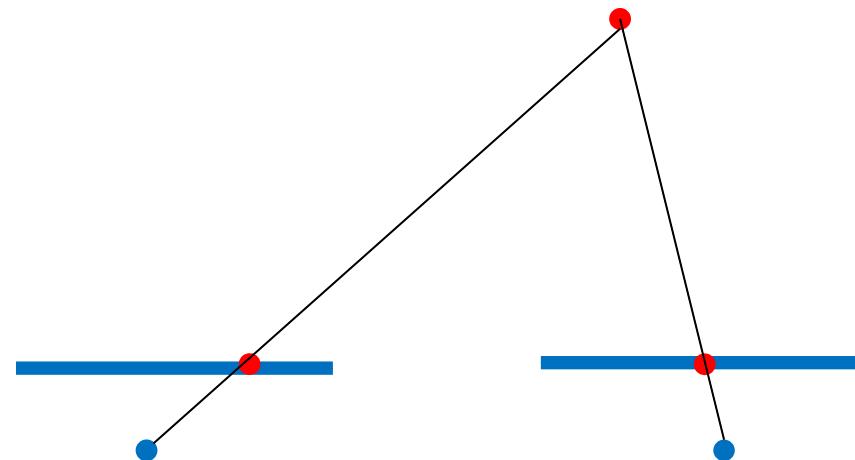
- We discussed about what Stereo Vision is.



- We discussed about what Stereo Vision is.
- We learned about :
 - Stereo/Epipolar Geometry



- We discussed about what Stereo Vision is.
- We learned about :
 - Stereo/Epipolar Geometry
 - Rectified Stereo Case



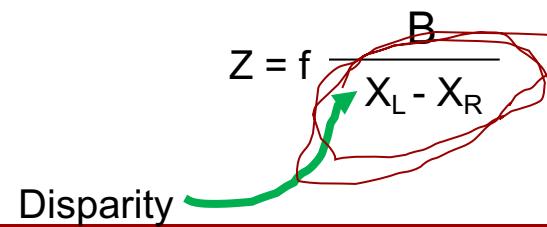
- We discussed about what Stereo Vision is.
- We learned about :
 - Stereo/Epipolar Geometry
 - Rectified Stereo Case
 - Depth from Stereo Matches

small depth big disparity
vice versa

Reference Image



Disparity Map



- We discussed about what Stereo Vision is.
- We learned about :
 - Stereo/Epipolar Geometry
 - Rectified Stereo Case
 - Depth from Stereo Matches
- Correspondence Problem
 - **Dense** vs Sparse Correspondence
 - **Local** vs Global Correspondence
 - (Dis)-Similarity Measures

Stereo Vision

Sparse output

Dense output

- Local Methods (Area-based)
- Global Methods (Energy-based)
- Other Methods

Lazaros Nalpantidis

Stereo Vision

Perception for Autonomous Systems

Lecture 3 - Stereo Vision (15/02/2021)

Outline/Content:

- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
 - Dense vs Sparse Correspondence
 - Local vs Global Correspondence
 - (Dis-)Similarity Measures
- Summary

Topics and Reading Sources:

- Optional Additional Reading Material: D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," International Journal of Computer Vision, vol. 47, no. 1-3, pp. 7–42, 2002.
- L. Nalpantidis, G. C. Sirakoulis, and A. Gasteratos, "Review of stereo vision algorithms: from software to hardware," International Journal of Optomechatronics, vol. 2, no. 4, pp. 435–462, 2008.

What is Stereo Vision?

Stereovision (aka., stereoscopic vision or stereopsis) describes the visual perception in three dimensions.

Two sensory inputs (both eyes for humans or two cameras for machines) which capture individual images are being used in this process. These two images have overlapping areas of common visual information, but also some unique visual information. The images are then being merged together, by matching up the similarities and adding in the small differences. By doing that we obtain a three-dimensional stereo picture.

This implies that at least two visual based sensory inputs are needed to obtain a stereo picture.

Stereo/Epipolar Geometry Article

There are two key requirements, that need to be fulfilled to calculate the 3D structure out of two images:

- **The position of cameras:** To obtain the positions of the cameras, it is necessary to calculate the 3D points by taking one of the camera positions as the origin. Then a calibration pattern is needed to calibrate the two cameras, which allows to find the 3D points.
- **The point correspondence:** All point correspondences between the two images must be calculated for each 3D point in the scene.

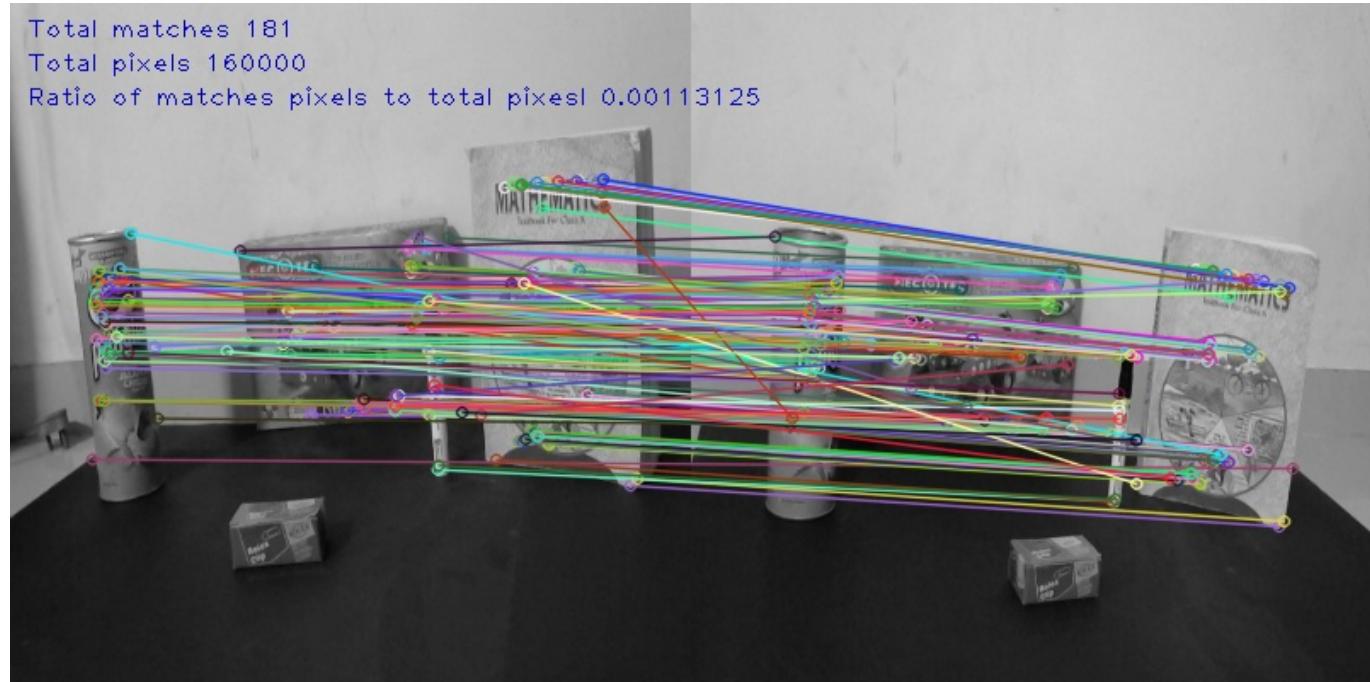
Point correspondence is finding the corresponding pixel of one image in the other.

Point correspondence: Feature matching

Feature matching algorithms such as SIFT, SURF, or ORB are valid methods for matching features between the left and right stereo images.

However, these algorithms will most likely result in a very sparsely reconstructed 3D scene due to the poor ratio of total pixels to found pixels.

Those approaches are also very computationally heavy, and would check each pixel, although we only need to track pixels on our epipolar line.



Epipolar geometry to the rescue!

Epipolar geometry is crucial for stereo matching (the process of stereo vision) as it reduces the search space for point correspondence by eliminating false matches.

Stereo matching is only feasible to do, if there's structure in the image. Therefore, stereo matching on a white wall performs very poorly.

Terminology

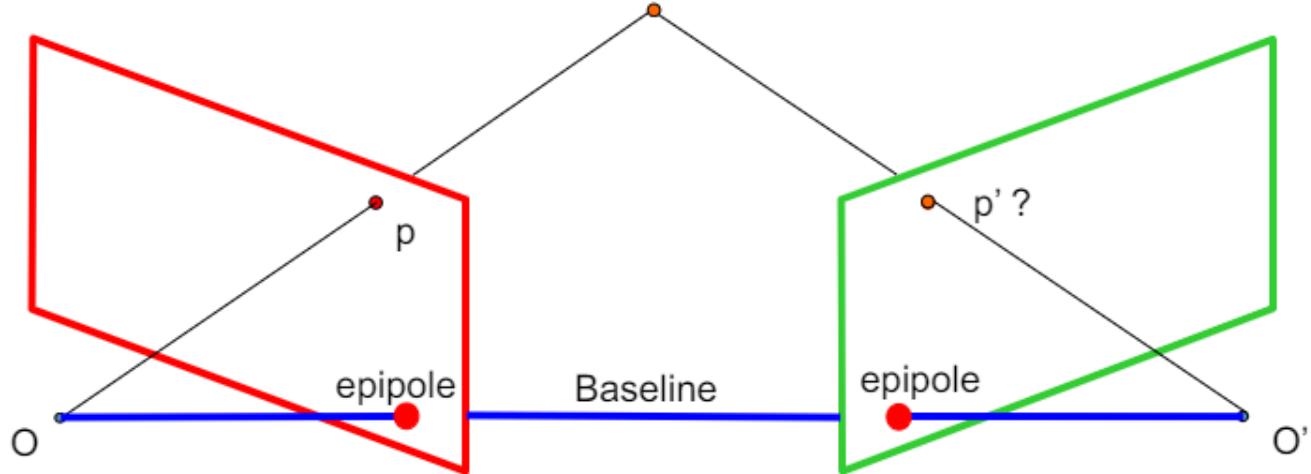
- *Baseline*: The line connecting the two camera centers.
- *Epipole*: Point of intersection of baseline with the image plane.
- *Epipolar plane*: The plane that contains the two camera centers and a 3D point in the world.
- *Epipolar line*: Intersection of the epipolar plane with each image plane.

How does it work?

We have two rigidly fixed cameras with different viewpoints capturing pictures. Now we want to know which pixels in an image pair (containing the left and right stereo images) corresponds to each other. If both cameras produce rays that pass through the red dots (projections of the 3D scene point onto the image plane), then they should intersect at that exact point, since each line passes through the scene point.

The ray from one image is reflected onto the image plane of the other image - displayed by the epipolar line that connects the epipole with the projection of the scene point in that image plane (red dot).

This way, the possible location of the right_image for example is constrained to a single line. Which reduces the search space for a pixel corresponding to a pixel of left_image.

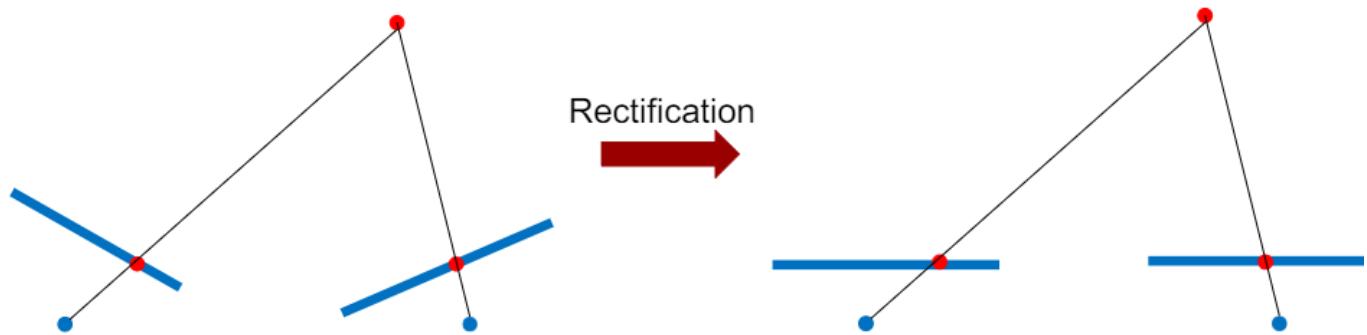


We always spoke about finding the correspondence for one single pixel. To create a 3D structure of an image it's necessary to apply this method on each pixel of those image pairs.

Rectified Stereo Case

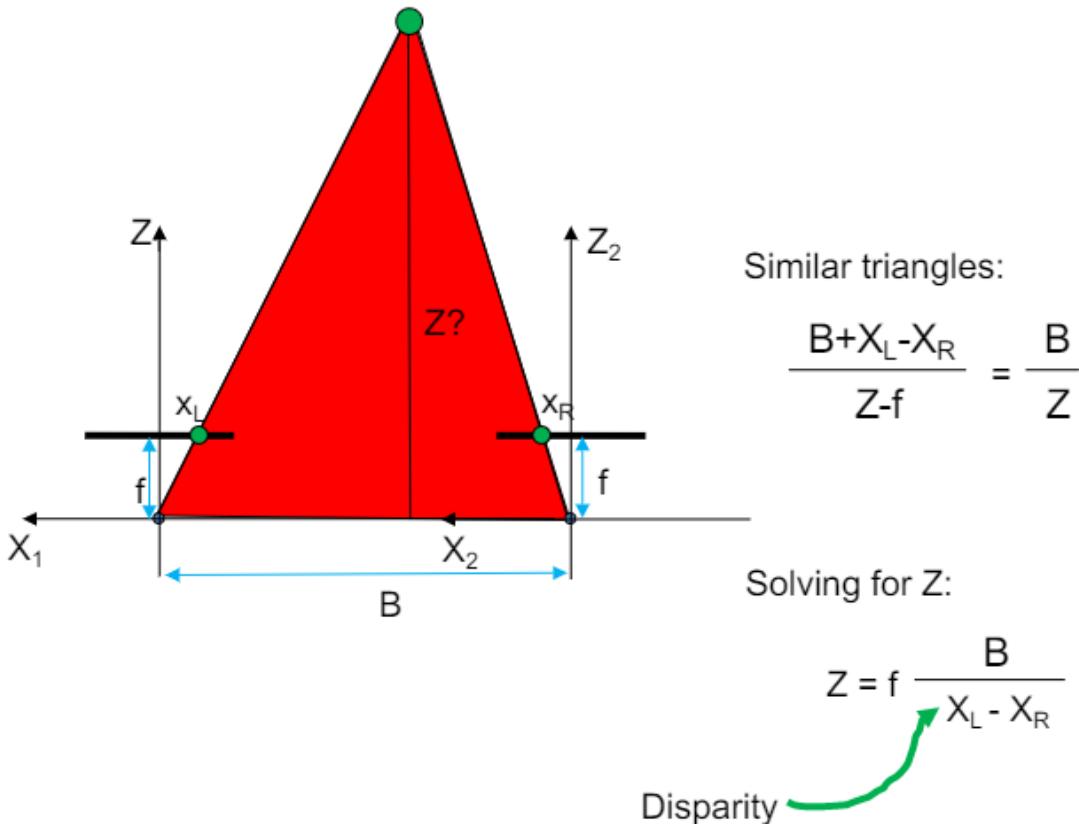
Rectification:

- The source images are projected onto a common plane parallel to the baseline B connecting the optical centers of the images
- Epipolar lines become parallel (and under certain conditions they become also horizontal)



Depth from Stereo Matches

It is important to note that Z extends from the scene point to the baseline. Therefore, we need to exploit similar triangles to formalize the following equation:



1. The depth of the stereo image "Z" is proportional to the baseline of the point and inversely proportional to the difference of the corresponding points of the two images. This means that a small depth will result in a large difference or jump between the two images and vice versa.
2. The difference of the corresponding points of the two images is called the disparity.
3. A depth map is not the same as a disparity map. It's possible to convert from one to another as long as the focal length is known, but the difference between the two is that the disparity map is inverted and scaled, as only the pixel values are known. However, the information contained information is similar.

Correspondence Problem

Definition: The correspondence problem refers to the problem of determining which parts (clusters of pixels or individual pixels) of one image correspond to parts of another image. Most often, the differences arise from the movement of the camera, the flow of time and/or the movement of objects in the photos.

Beyond the hard constraint of epipolar geometry, there are "soft" constraints to help identify corresponding points:

- Similarity: The image pairs should be relatively similar in a sense, that they contain the same objects, colors, etc.
- Uniqueness: Objects in an image are assumed to contain unique features that are easily identifiable. These unique features should not differ in the two images, e.g. a person with a nose in the source image will most likely also have only one nose in the target image.
- Ordering: Let us assume that in a source image there is an apple, a coke and a teddy bear. We assume that this order remains the same in the target image, which is most likely the case. But there are exceptions to this rule.

- Disparity gradient is limited: The depth values should be continuous for neighboring pixels. There will be jumps in depth and disparity, but they should be limited.

What is meant by hard constraint of epipolar geometry? This refers to the strict rule, that correspondences HAVE to be on the epipolar line. Soft constraints aren't always true, but most of the time they are.

To find matches in the image pair, we will assume:

- most scene points are visible in both images
- image regions for the matches are similar in appearance

Do we need dense or sparse stereo matching?

Before this question can be answered, we need to make sure that we understand the differences:

What's the difference between sparse or dense stereo matching? The difference lies in the number of calculations of your correspondences. With dense stereo matching, you need to calculate as many correspondences as possible, whereas with sparse stereo matching, only a small number of correspondences need to be calculated to meet the requirements of the application.

To come back to the question of which ones we need - it depends on the use case. For example, dense stereo matching is important for autonomous vehicles because you can't neglect any information to make sure it correctly assesses the situation and reacts accordingly. A face tracking system, on the other hand, needs only sparse stereo matching because it doesn't care about anything other than the face.

Stereo Vision

Sparse output

Dense output

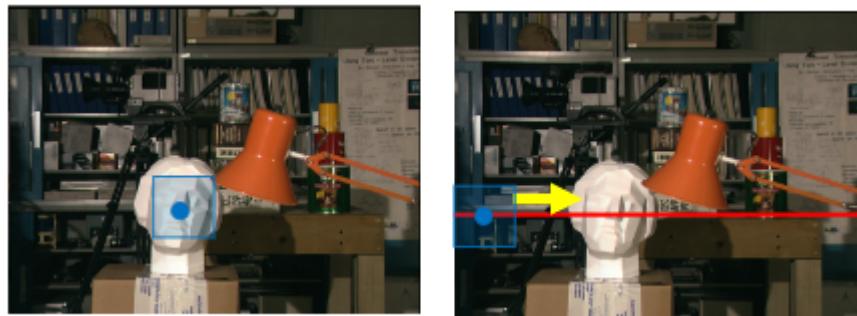
- Local Methods (Area-based)
- Global Methods (Energy-based)
- Other Methods

Dense Stereo Correspondence: Local Methods

Try to find correspondences for all the pixels of the reference image.

- For each epipolar line
 - For each pixel in the left image
 - Compare with every pixel on same epipolar line in right image
 - Choose the pixel that maximizes a similarity metric (or minimizes a dissimilarity metric!).

- Improvement: don't match individual pixels, but rather match windows!



The proposed windows method slides from left to right along the epipolar line. To minimize dissimilarity while maximizing similarity.

Stereo Correspondence Metrics [Different Metrics](#)



Stereo Correspondence Metrics

- Sum of Absolute Differences (SAD)

$$SAD(x, y, d) = \sum_{x, y \in W} |I_l(x, y) - I_r(x, y - d)|$$

- Sum of Squared Differences (SSD)

$$SSD(x, y, d) = \sum_{x, y \in W} (I_l(x, y) - I_r(x, y - d))^2$$



- Normalized Cross-Correlation

$$NCC(x, y, d) = \frac{\sum_{x, y \in W} I_l(x, y) \cdot I_r(x, y - d)}{\sqrt{\sum_{x, y \in W} I_l^2(x, y) \cdot \sum_{x, y \in W} I_r^2(x, y - d)}}$$

- ...many many more!!!

SAD Example:

$$A = \begin{bmatrix} 2 & -10 & -2 \\ 14 & 12 & 10 \\ 4 & -2 & 2 \end{bmatrix}; B = \begin{bmatrix} 6 & 10 & -2 \\ 0 & -12 & -4 \\ -5 & 2 & -2 \end{bmatrix};$$

$$C = \text{abs}(A - B) = \begin{bmatrix} |2 - 6| & |(-10) - 10| & |(-2) - (-2)| \\ |14 - 0| & |12 - (-12)| & |10 - (-4)| \\ |4 - (-5)| & |(-2) - 2| & |2 - (-2)| \end{bmatrix} = \begin{bmatrix} 4 & 20 & 0 \\ 14 & 24 & 14 \\ 9 & 4 & 4 \end{bmatrix};$$

$$\text{SAD} = \text{sum}(C) 4 + 20 + 0 + 14 + 24 + 14 + 9 + 4 + 4 = 93$$

The closer the similarity metric calculated by the SAD is to 0, the stronger the similarity between these two images.

$$A = \begin{bmatrix} 2 & -10 & -2 \\ 14 & 12 & 10 \\ 4 & -2 & 2 \end{bmatrix}; B = \begin{bmatrix} 6 & 10 & -2 \\ 0 & -12 & -4 \\ -5 & 2 & -2 \end{bmatrix};$$

$$C = A-B \odot A - B = \begin{bmatrix} (2-6)^2 & ((-10)-10)^2 & ((-2)-(-2))^2 \\ (14-0)^2 & (12-(-12))^2 & (10-(-4))^2 \\ (4-(-5))^2 & ((-2)-2)^2 & (2-(-2))^2 \end{bmatrix}$$

$$= \begin{bmatrix} 16 & 400 & 0 \\ 196 & 576 & 196 \\ 81 & 16 & 16 \end{bmatrix};$$

$$SSD = \text{sum}(C) = 16 + 400 + 0 + 196 + 576 + 196 + 81 + 16 + 16 = 1497$$

SSD Example:

The closer the similarity metric calculated by the SSD is to 0, the stronger the similarity between these two images. Note that SSD is generally only used due to its simplicity and relatively low computational cost - in general better results are achievable by using Normalized Cross Correlation.

Normalized Cross-Correlation Example:

$$A = \begin{bmatrix} 2 & -10 & -2 \\ 14 & 12 & 10 \\ 4 & -2 & 2 \end{bmatrix}; B = \begin{bmatrix} 6 & 10 & -2 \\ 0 & -12 & -4 \\ -5 & 2 & -2 \end{bmatrix};$$

$$A^2 = \begin{bmatrix} 4 & 100 & 4 \\ 196 & 144 & 100 \\ 16 & 4 & 4 \end{bmatrix}; B^2 = \begin{bmatrix} 36 & 100 & 4 \\ 0 & 144 & 16 \\ 25 & 4 & 4 \end{bmatrix};$$

$$\text{nom} = A \odot B = \begin{bmatrix} 2 * 6 & ((-10) * 10) & ((-2) * (-2)) \\ 14 * 0 & (12 * (-12)) & (10 * (-4)) \\ (4 * (-5)) & ((-2) * 2) & (2 * (-2)) \end{bmatrix}$$

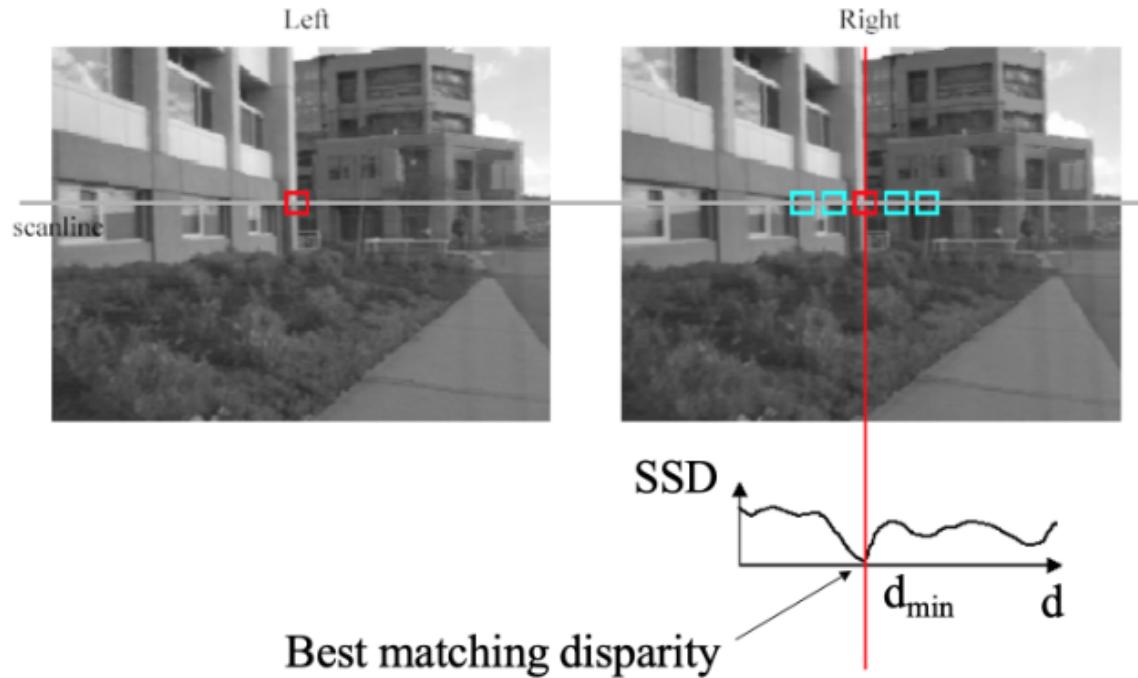
$$= \begin{bmatrix} 12 & -100 & 4 \\ 0 & -144 & -40 \\ -20 & -4 & -4 \end{bmatrix};$$

$$\text{dom1} = A \odot A = \begin{bmatrix} 4 & 100 & 4 \\ 196 & 144 & 100 \\ 1644 & & \end{bmatrix}$$

$$\text{dom2} = B \odot B = \begin{bmatrix} 26 & 100 & 4 \\ 0 & 144 & 16 \\ 25 & 4 & 4 \end{bmatrix}$$

$$\text{NormalizedCross-Correlation} = \frac{\text{nom}}{\sqrt{\text{dom1}} * \sqrt{\text{dom2}}} = \frac{-296}{\sqrt{572} * \sqrt{333}} = -0.678$$

Stereo Correspondence Metrics: SSD



Visual illustration that the lowest SSD value found for a target pixel on the epipolar line indicates that this pixel is most similar to the source pixel and that there is therefore a correspondence between these pixels.

Windows Sizes

Large windows suppress noise better than smaller windows. However, if the window is too large, the image loses its fine granularity. With smaller windows, the structure is better preserved, but at the expense of a gain in noise.

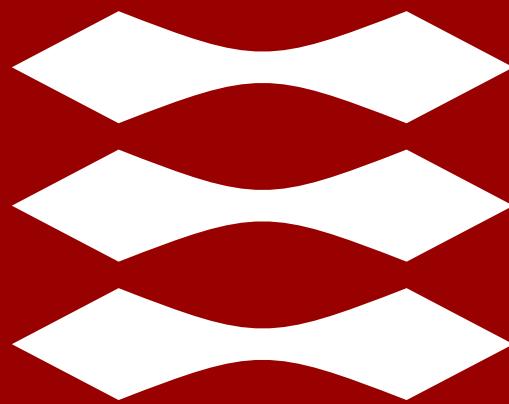
Global Stereo Correspondence

Global Stereo Correspondence

- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
 - Thus, those methods are called "local" or "area-based" methods.
- Global methods find better solutions in expense of more computations
 - Optimize jointly the disparity values of all the pixels of each scanline (e.g. Dynamic Programming)
 - Optimize jointly the disparity values of all the pixels of the image (e.g. graph cuts)
- *In global algorithms, stereo correspondence is formulated as an energy function minimization problem, consisting of data and smoothness terms.*



DTU



Perception for Autonomous Systems 31392:

Camera Matrix and Camera Calibration

Lecturer: Evangelos Boukas—PhD

Outline

- Image formation
- Camera model – Project form World to sensor
- Camera calibration – The 3D case
- Camera calibration – The realistic case
- Camera radial distortion

Image Formation

- Let's look at
the most simplistic case.
What's weird about it?

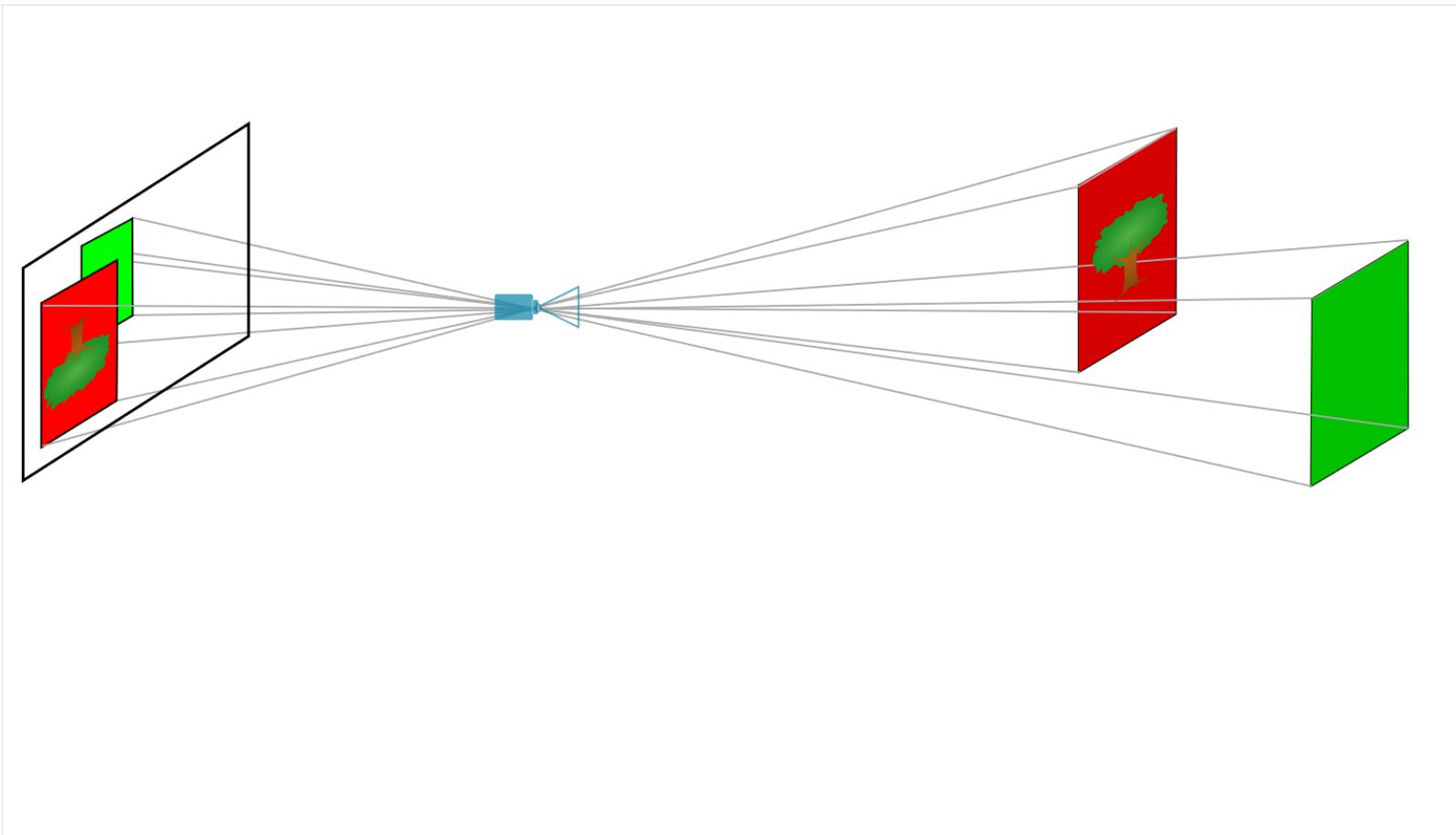


Image Formation

- Let's look at
the most simplistic case.
What's weird about it?
- Let's Fix that

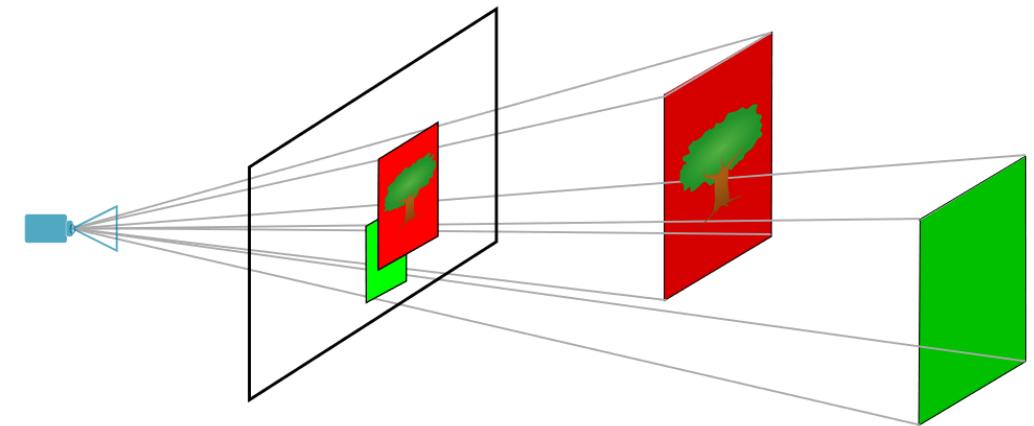


Image Formation

- Let's look at
the most simplistic case.
What's weird about it?
- Let's Fix that

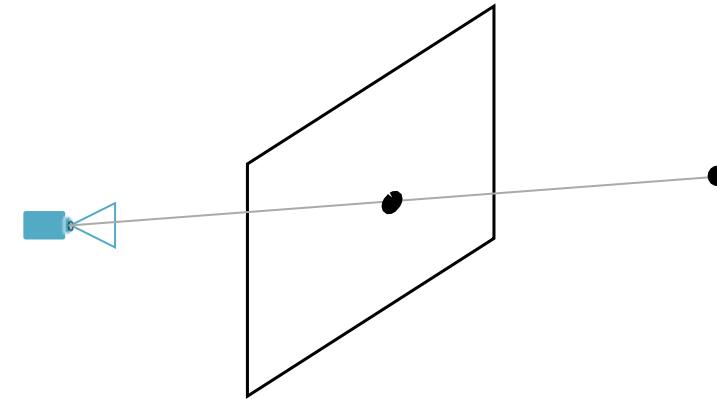
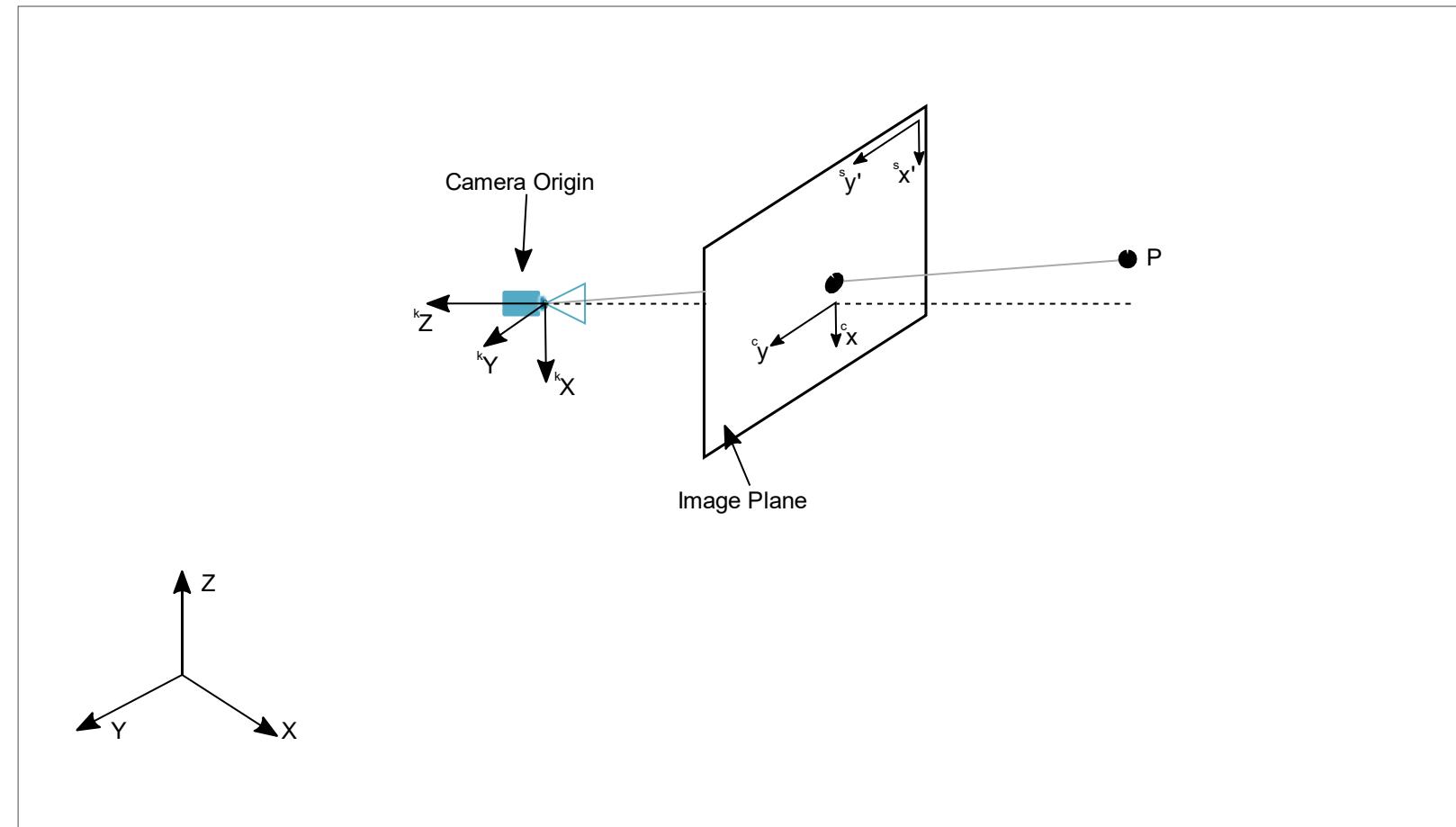


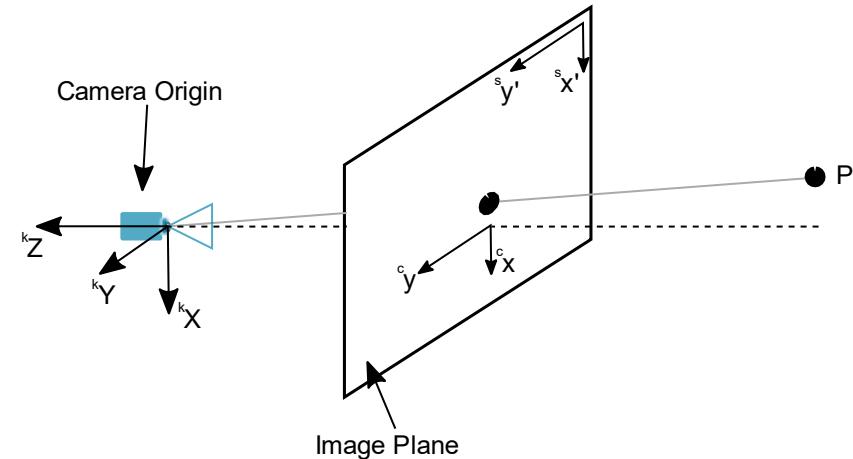
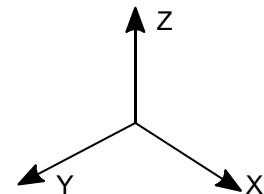
Image Formation

- Let's look at the most simplistic case.
What's weird about it?
- Let's Fix that
- Let's get some notation



Camera Extrinsics and Instrisics

- The orientation of the camera wrt Word coordinates is called extrinsics
- The rest of Parameters are called intrinsics
- We will see how points in 3D are projected on the sensor



Camera Extrinsics

- We need to express a 3D point P
 $x_p = [x_p \ y_p \ z_p]^T$
 in the camera coordinates

$$^k x_p = [^k x_p \ ^k y_p \ ^k z_p]^T$$

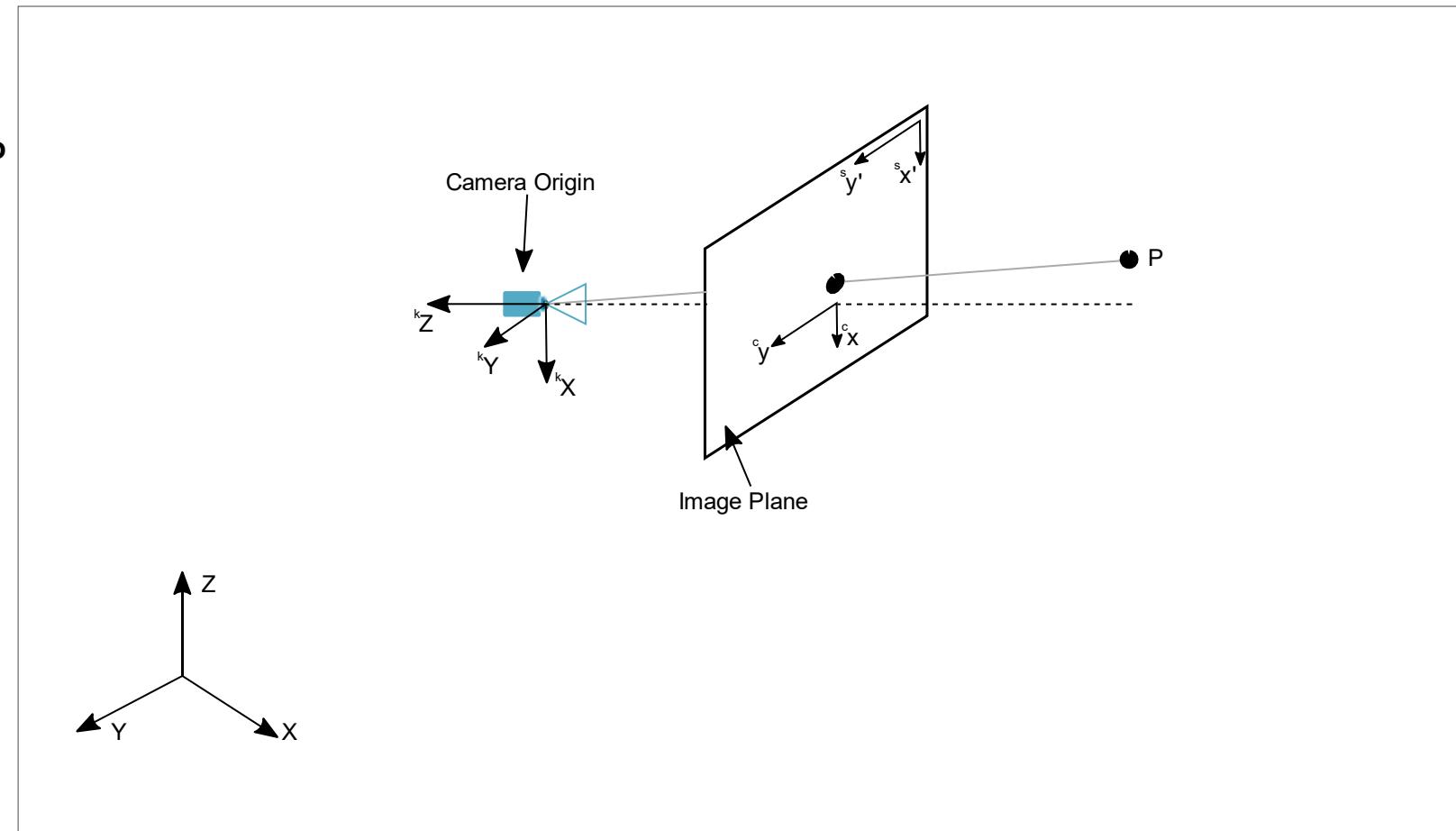
- To do so we need to use
 the camera Origin

$$^k x_c = [^k x_c \ ^k y_c \ ^k z_c]^T$$

- Then:

$$\begin{bmatrix} x_p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^w R_c & {}^w t_c \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} ^k x_p \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} ^k x_p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^w R_c^T & -{}^w R_c^T {}^w t_c \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} x_p \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} ^k x_p \\ 1 \end{bmatrix} = \begin{bmatrix} R & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} I_{3 \times 3} & \mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} x_p \\ 1 \end{bmatrix}$$

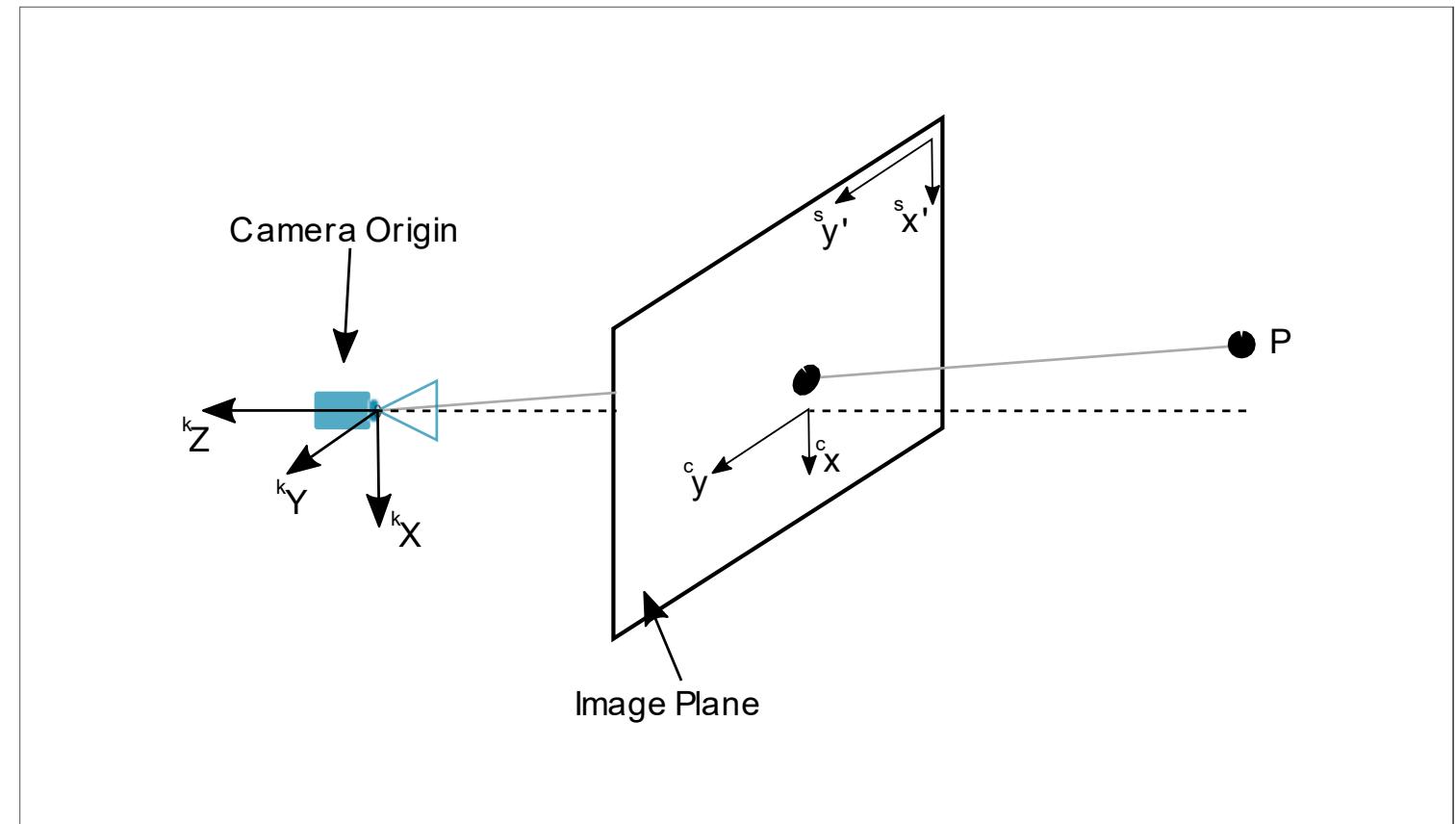


Camera Extrinsics

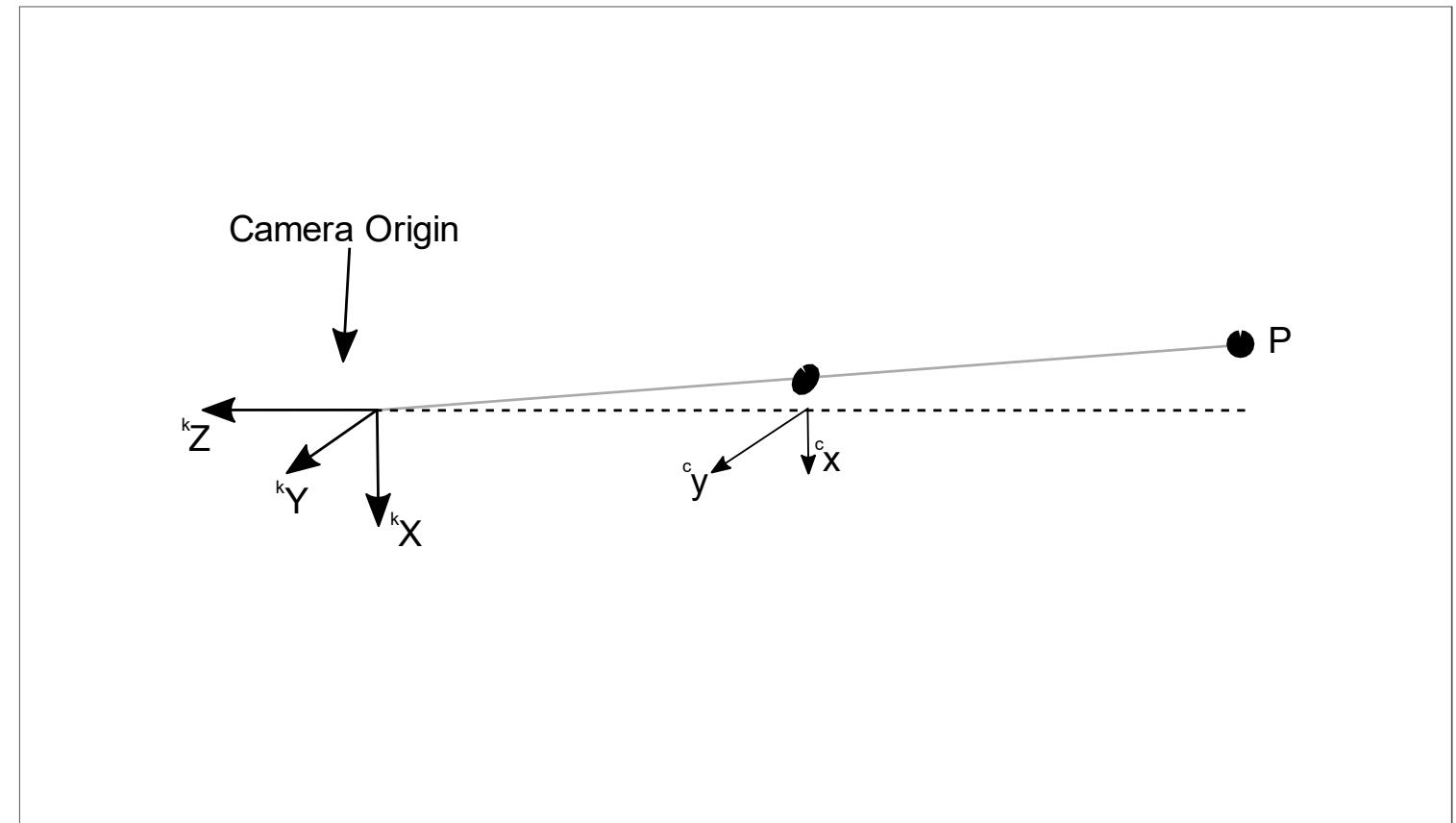
- So now we have a description for the 3D point in camera frame

- To get to the image frame we will have to drop a dimension

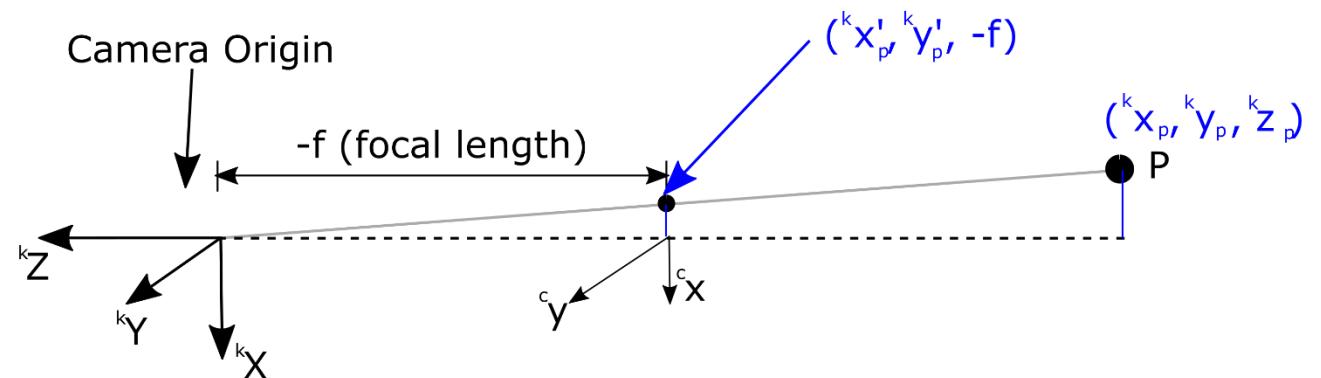
- Let's get rid of things we don't need



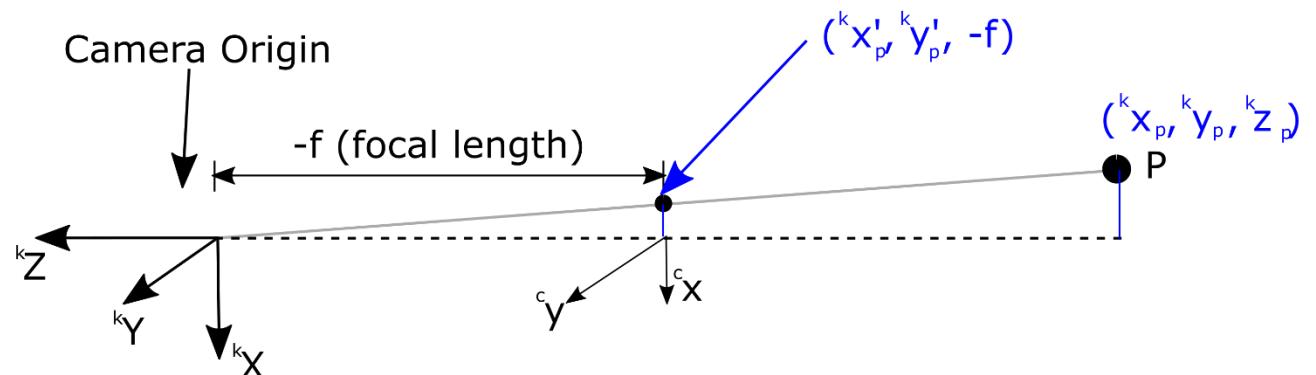
- Let's get rid of things we don't need



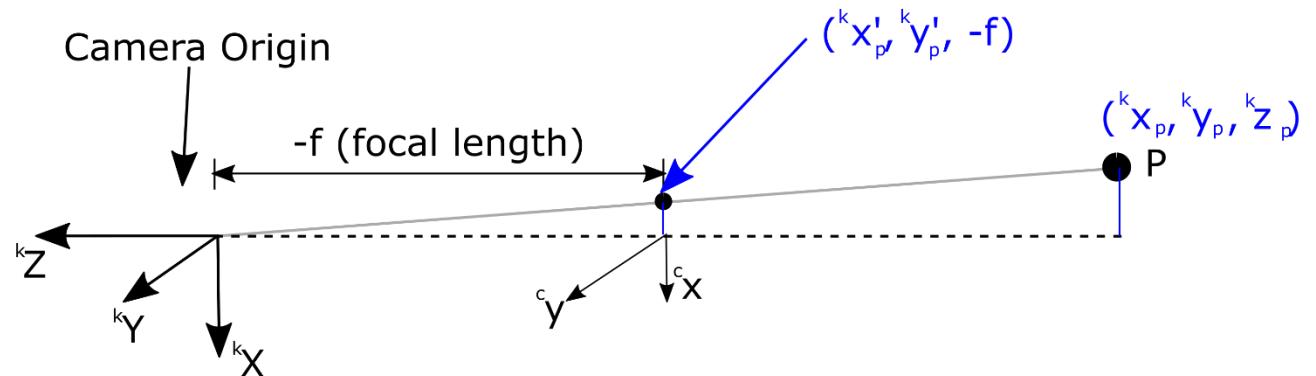
- Let's get rid of things we don't need



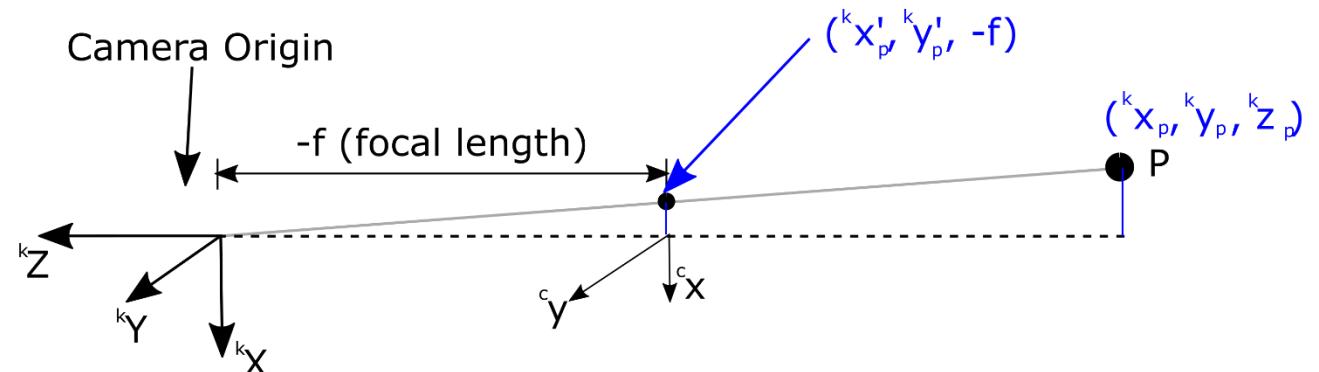
- Let's get rid of things we don't need
- So it is clear that the position of the projection on the image frame is dependent on the ratio x/z



- Let's get rid of things we don't need
- So it is clear that the position of the projection on the image frame is dependent on the ratio x/z
- In fact ${}^c x_p = -f * {}^k x_p / {}^k z_p$ and ${}^c y_p = -f * {}^k y_p / {}^k z_p$



- Let's get rid of things we don't need
- So it is clear that the position of the projection on the image frame is dependent on the ratio x/z
- In fact ${}^c x_p = -f * {}^k x_p / {}^k z_p$ and ${}^c y_p = -f * {}^k y_p / {}^k z_p$
- Additionally we can add the transformation to the sensor frame



Projection

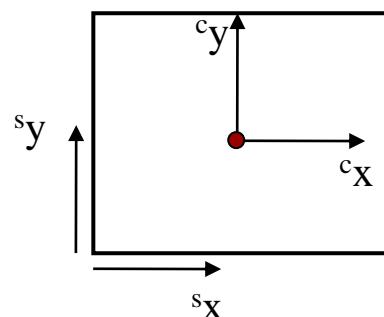
- From the previous

- We get to the following

- Where W is the scale
 - **This is the analytic version of the projection matrix**

Projection Matrix

- We can define the Projection Matrix as follows

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$


- **This is great news for Calibration!**
- It means that our problem is a system of linear equations
- How many unknowns?
- 11: (5 + 6) Intrinsic+Extrinsic

Notation Disclaimer: In this slide X Y Z refer to 3D world points (not matrices) and x,y to 2D camera points

Calibration – Known 3D positions

- We defined our camera projection as follows:

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- If we “sample” a set of points in 3D and 2D: $[X, Y, Z] \rightarrow [x, y]$
we get the following analytic equations:

$$\lambda x = P_{11}X + P_{12}Y + P_{13}Z + P_{14}$$

$$\lambda y = P_{21}X + P_{22}Y + P_{23}Z + P_{24}$$

$$\lambda = P_{31}X + P_{32}Y + P_{33}Z + P_{34}$$

- However, λ is used for scale and therefore does not provide a 3rd equation

Notation Disclaimer: In this slide X Y Z refer to 3D world points (not matrices) and x,y to 2D camera points

Calibration – Known 3D positions

- So for every 3D to 2D correspondence we get 2 equations:

$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})x = P_{11}X + P_{12}Y + P_{13}Z + P_{14}$$

$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})y = P_{21}X + P_{22}Y + P_{23}Z + P_{24}$$

- How many points do we need?
- 6 Points x 2 equations

Notation Disclaimer: In this slide X Y Z refer to 3D world points (not matrices) and x,y to 2D camera points

- For every point we get the following

$$0 = p_{11}X + p_{12}Y + p_{13}Z + p - p_{31}xX - p_{32}xY - p_{33}xZ - p_{34}x$$

$$0 = p_{21}X + p_{22}Y + p_{23}Z + p_{24} - p_{31}yX - p_{32}yY - p_{33}yZ - p_{34}y$$

- We can format all the linear equations in a single matrix: $Ap=0$

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Notation Disclaimer: In this slide X Y Z refer to 3D world points (not matrices) and x,y to 2D camera points

Calibration – Known 3D positions

- The solution is given by the following process which is itself called DLT:
 - Calculate the Singular Value decomposition (SVD)

$$\text{SVD : } A = UDV^T$$

- *Get the last column of V*

$P = V_{\text{smallest}}$ (*column of V corr. to smallest singular value*)

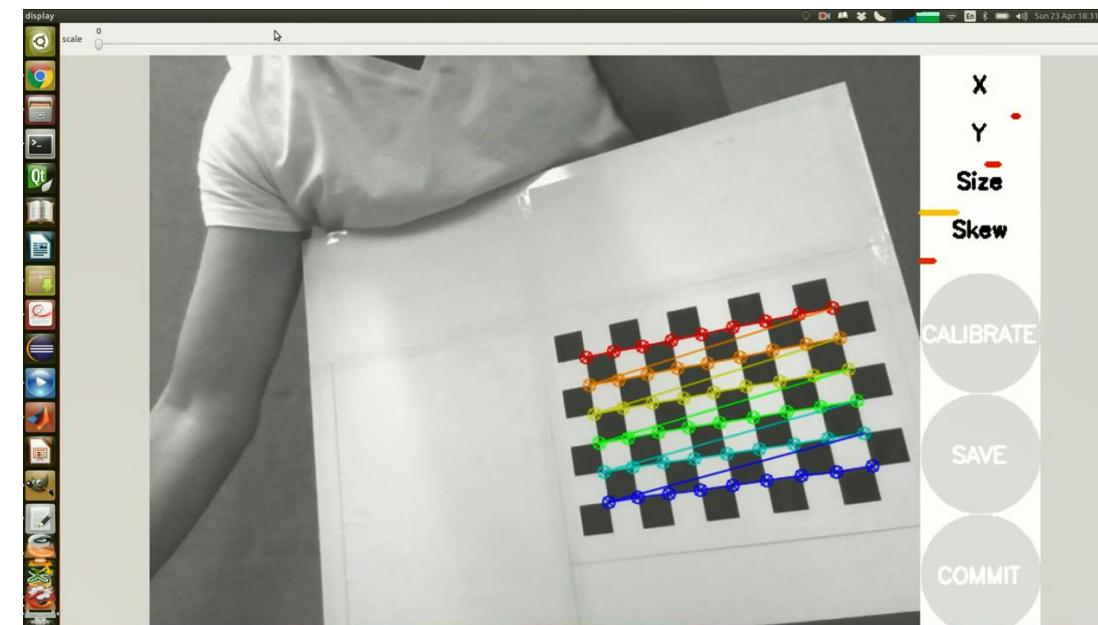
-Reshape into the P matrix

Calibration – Known 3D positions

- To summarize:
 - Get min 6 3D – 2D correspondences
 - Form matrix A ($AP=0$)
 - Calculate SVD
 - Get Last column of V which corresponds to the values of P

Calibration in real world

- In Real World we do not have 3D points.
- Or do we?
- What if we could use some sort of predefined setup which would allow us to know the relative position of 3D points?
- It turns out that if we have a calibration pattern we can know their relative positions of all points as they are on a known configuration



Calibration using homography

- Since $Z = 0$ we lose one degree of freedom:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

- We can then redefine our P as:

$$\equiv \begin{bmatrix} P_{11} & P_{12} & P_{14} \\ P_{21} & P_{22} & P_{24} \\ P_{31} & P_{32} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

- This is called homography and allows us to project points from one plane to another

Notation Disclaimer: In this slide X Y Z refer to 3D world points (not matrices) and x,y to 2D camera points

Calibration using homography

- How to calculate the homography:

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \quad \mathbf{x}' = \begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

- The homography has 9 elements. However, similar to before due to the scale there are only 8 unknowns
- Therefore, we need minimum 4 correspondences to setup the problem as a DLT

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & xx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & yx' & yy' & y' \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \mathbf{0}$$

Calibration using homography

Steps

- With a static camera move the pattern and get images.
 - Make sure to not present only one orientation, as there will be a problem with the estimation of the homography
- Calculate correspondences using the pattern
- Solve using **Normalized DLT**
 - Normalize coordinates for each image
 - Translate for zero mean
 - Scale so that average distance to origin is $\sim\sqrt{2}$
 - Form Matrix A
 - Solve using SVD
 - Calculate SVD
 - Use last column of V to get the homography.
 - Denormalize

Calibration using homography

- However the DLT solution is known to be prone to outliers.
 - Other approaches exist to solving the calibration:
 - Nonlinear least squares
 - RANSAC
 - RANSAC:
 1. Choose number of samples N
 2. Choose 4 random potential matches
 3. Compute \mathbf{H} using normalized DLT
 4. Project points from \mathbf{x} to \mathbf{x}' for each potentially matching pair:
 5. Count points with projected distance $< t$
 6. Repeat steps 2-5 N times
- Choose \mathbf{H} with most inliers

Calibration - Distortion

- Ok, but still, What about the lens Distortion?



Barrel



Pincushion



Fisheye

Calibration - Distortion

- So how do we model it:

Calibration - Distortion

- So how do we model it:
- Usually a quartic (biquadratic) polynomial is used:

$$\begin{aligned}\hat{x}_c &= x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \\ \hat{y}_c &= y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)\end{aligned}$$

- OR higher order

Calibration - Distortion

- So how do we model it:
- Usually a quartic (biquadratic) polynomial is used:

• OR h



Distorted

+

+



Undistorted

Calibration - Distortion

- How to calibrate for the distortion coefficients:
 - First do DLT for the Projection Matrix
 - Then form a non linear least square problem that includes both the linear projection and the non-linear distortion using the Levenberg Marquardt algorithm.

Conclusion

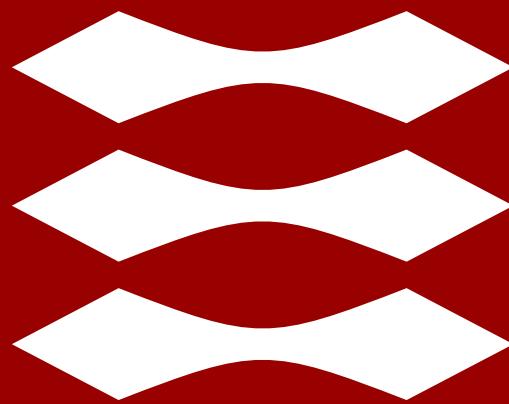
- What did we learn?

Perception for Autonomous Systems 31392:

Camera Matrix and Camera Calibration

Lecturer: Evangelos Boukas—PhD

DTU



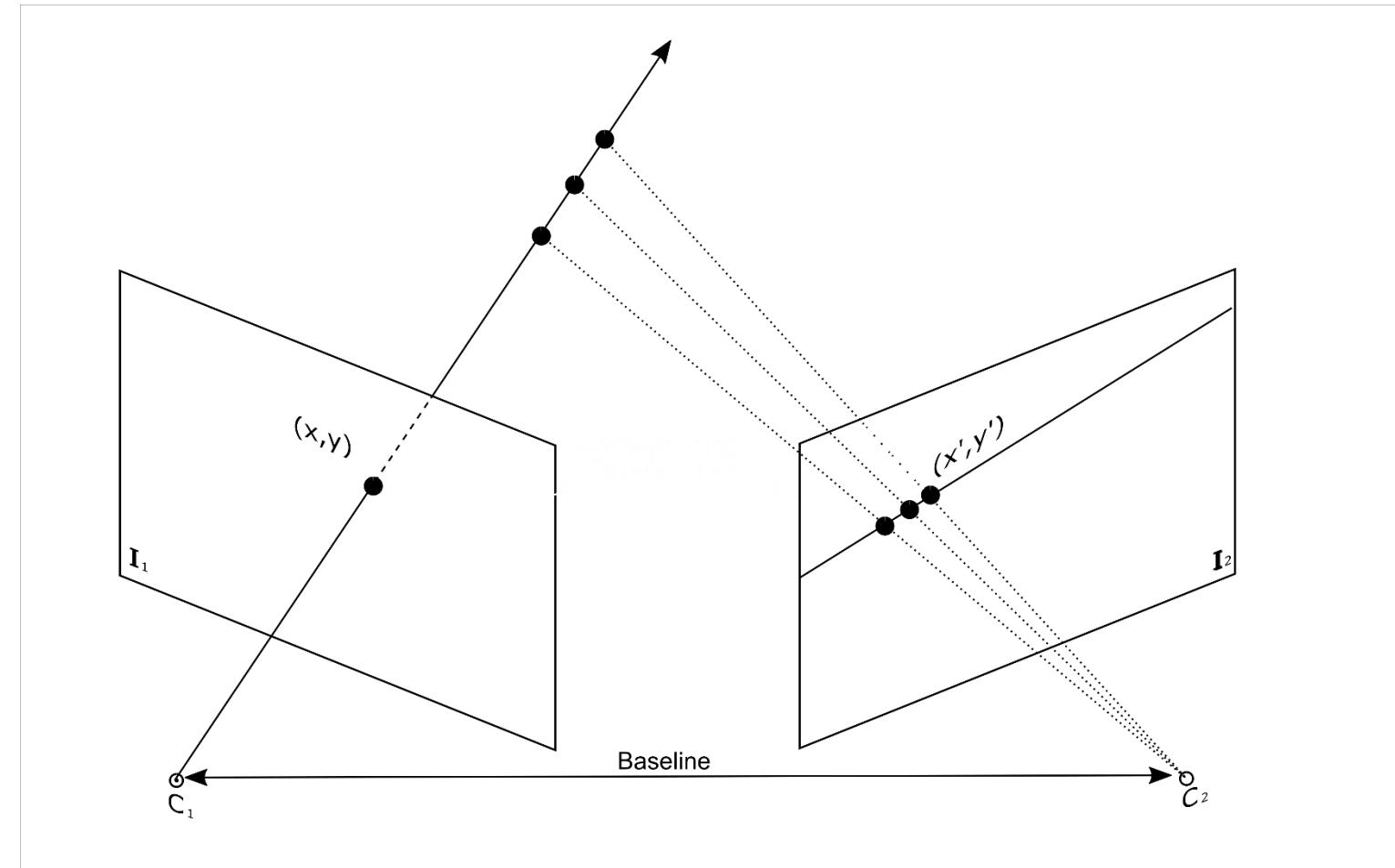
Perception for Autonomous Systems 31392:

Epipolar Geometry and the Fundamental Matrix

Lecturer: Evangelos Boukas—PhD

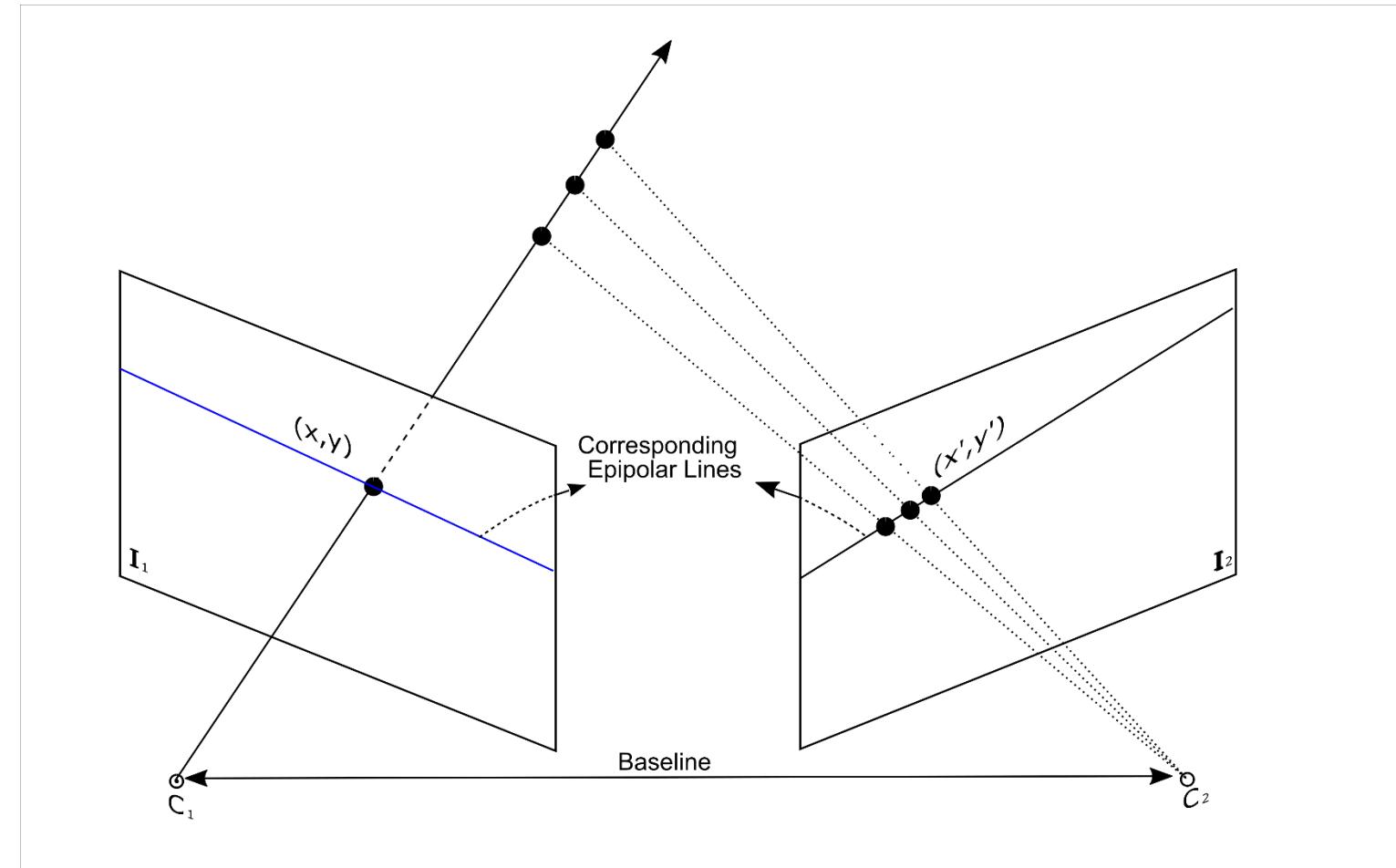
Epipolar Geometry: General Case

- Assuming:
 - 2 Camera Views
 - A ray passing through the camera center



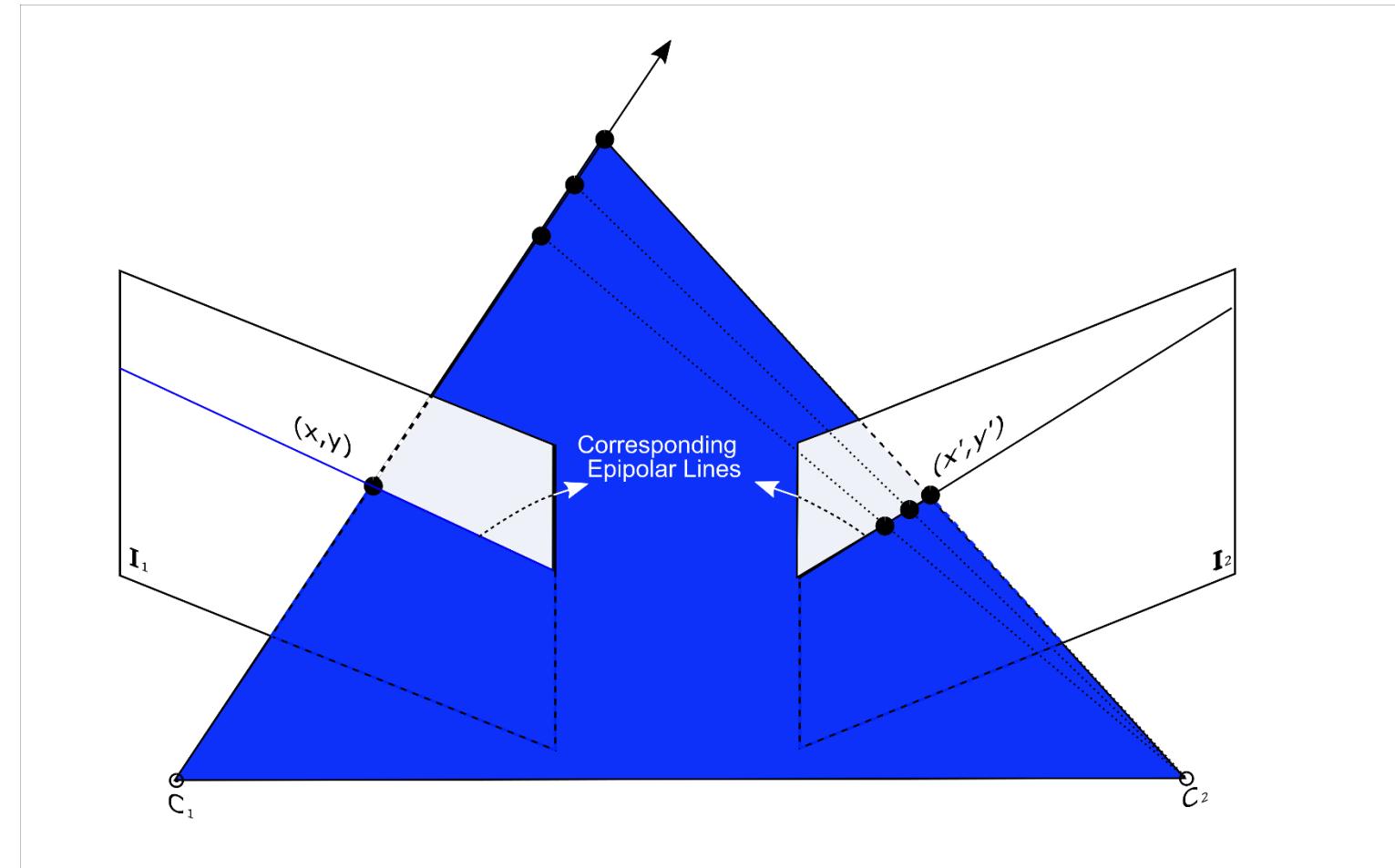
Epipolar Geometry: General Case

- Assuming:
 - 2 Camera Views
 - A ray passing through the camera center
- We can find the:
 - baseline and
 - the epipolar lines



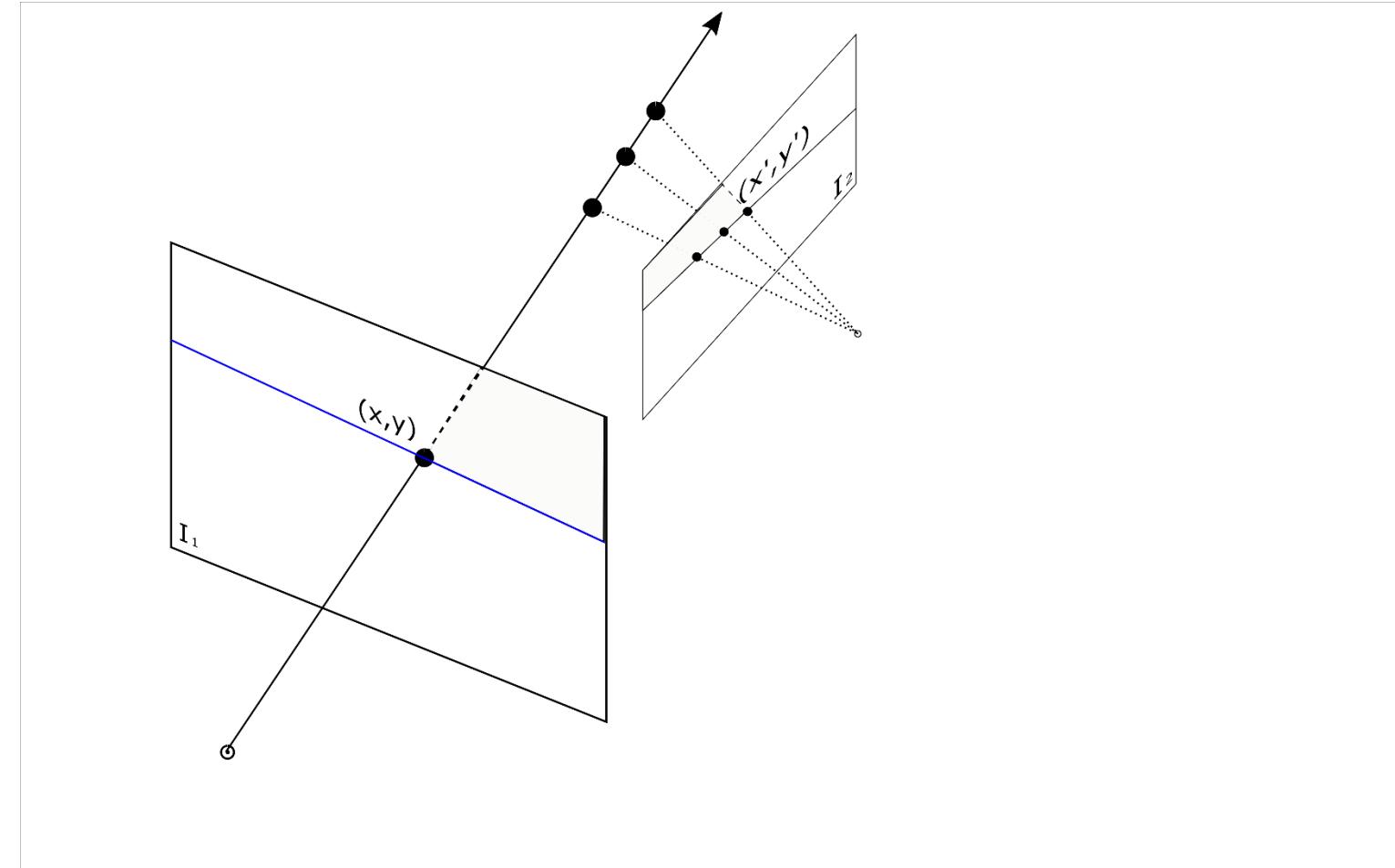
Epipolar Geometry: General Case

- Assuming:
 - 2 Camera Views
 - A ray passing through the camera center
- We can find the:
 - baseline and
 - the epipolar lines
 - Through the epipolar plane



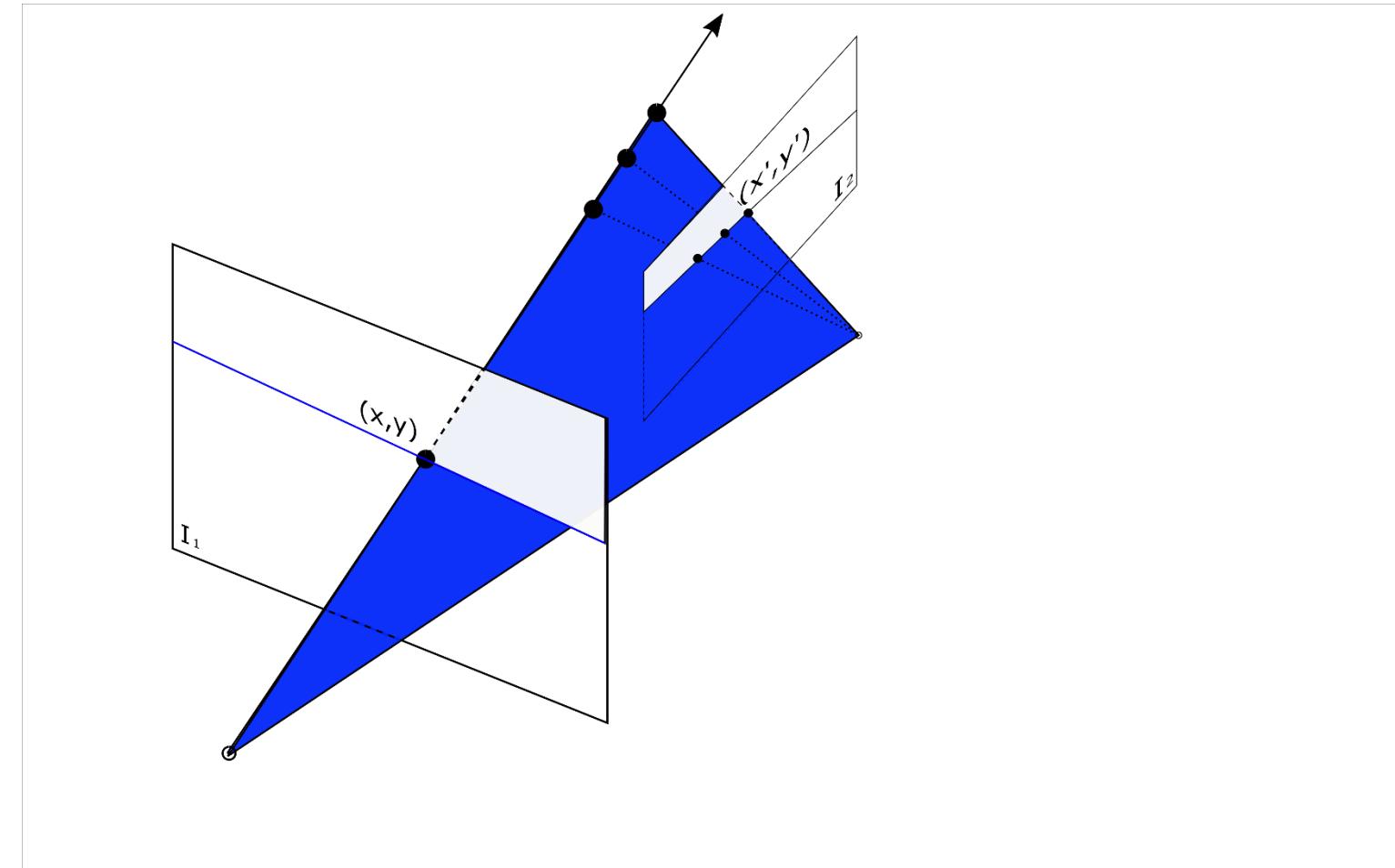
Epipolar Geometry

- What about this case?



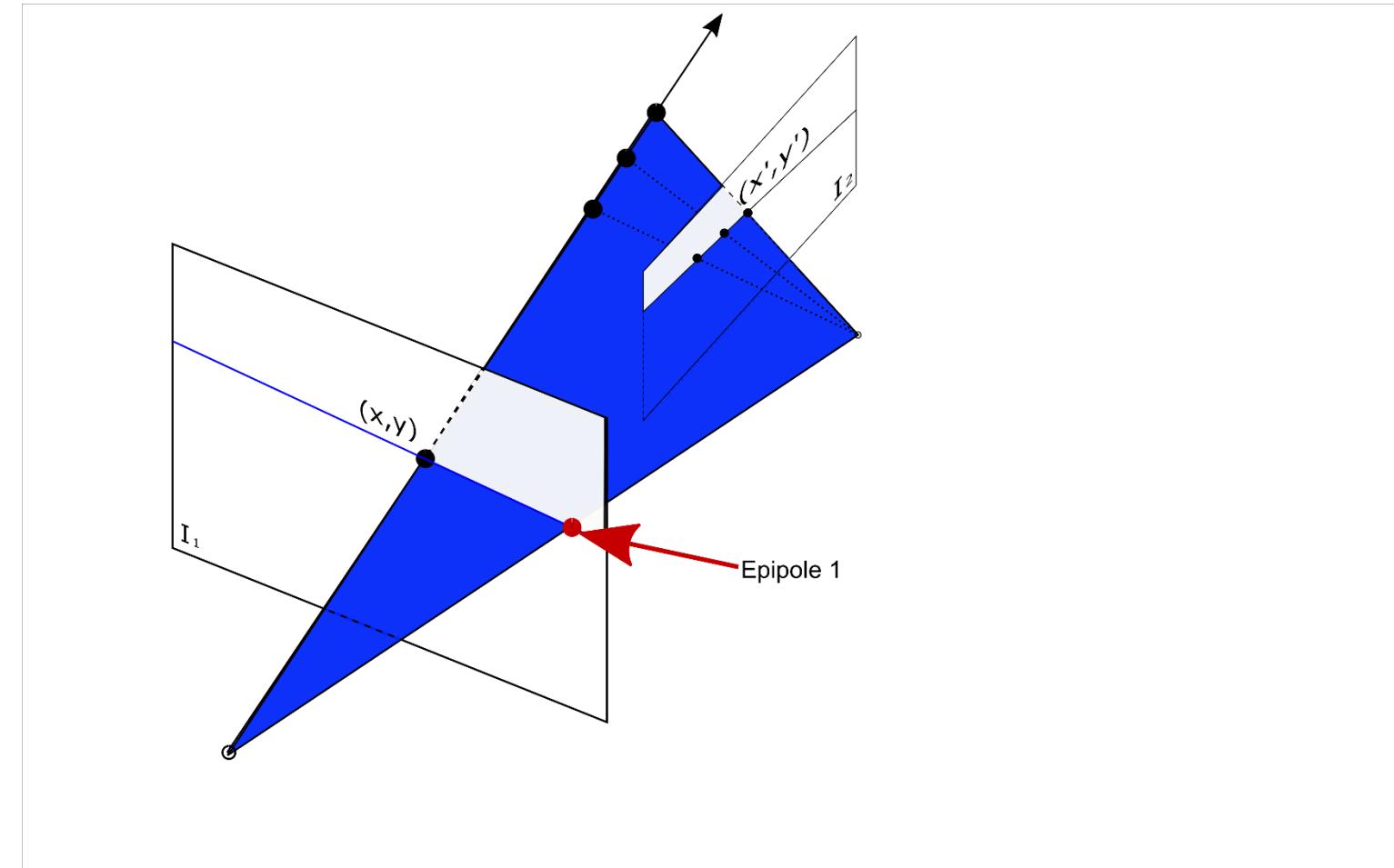
Epipolar Geometry

- What about this case?
- What's the epipolar plane?



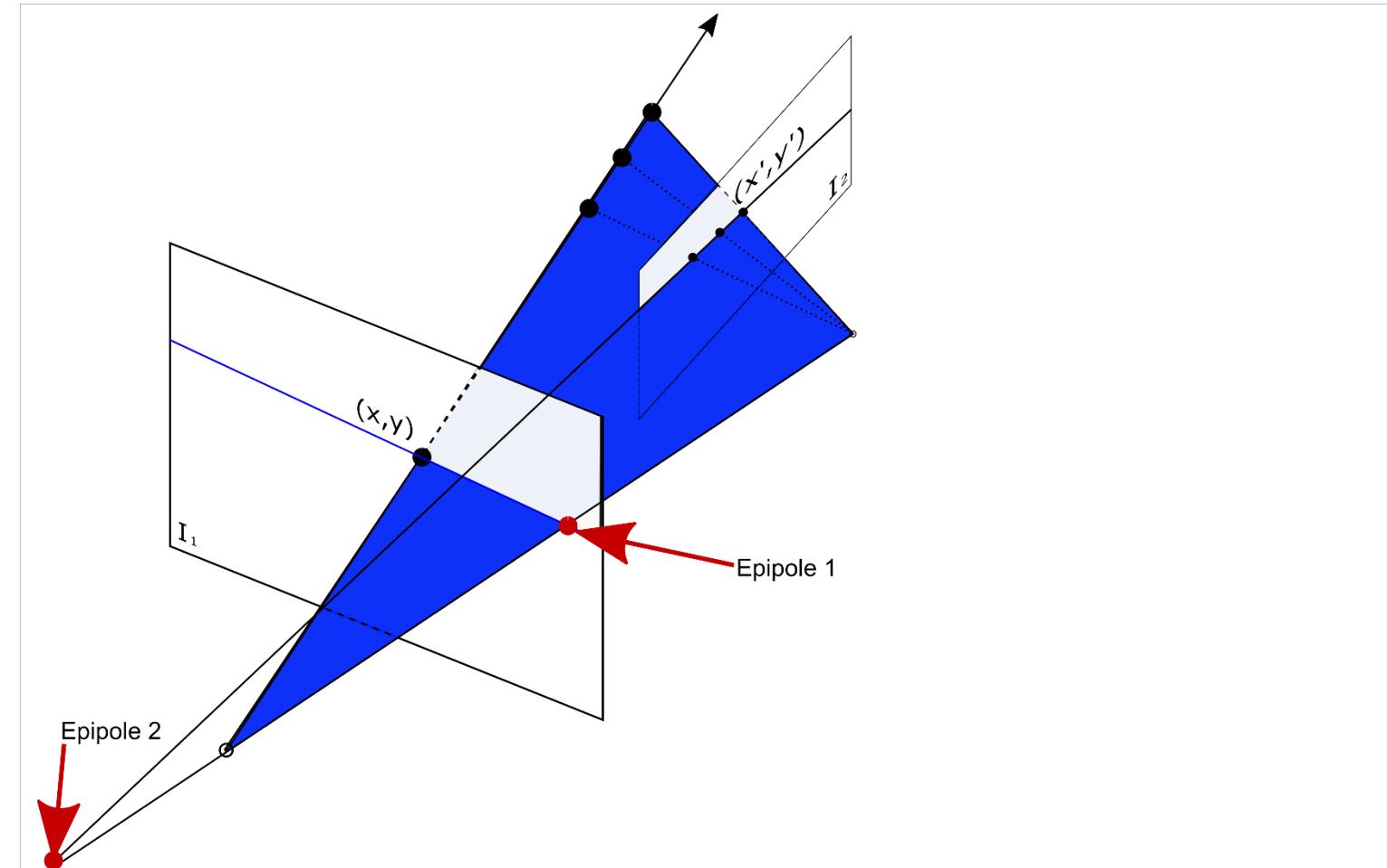
Epipolar Geometry

- What about this case?
- What's the epipolar plane?
- Can we find the epipole of I_1 ?



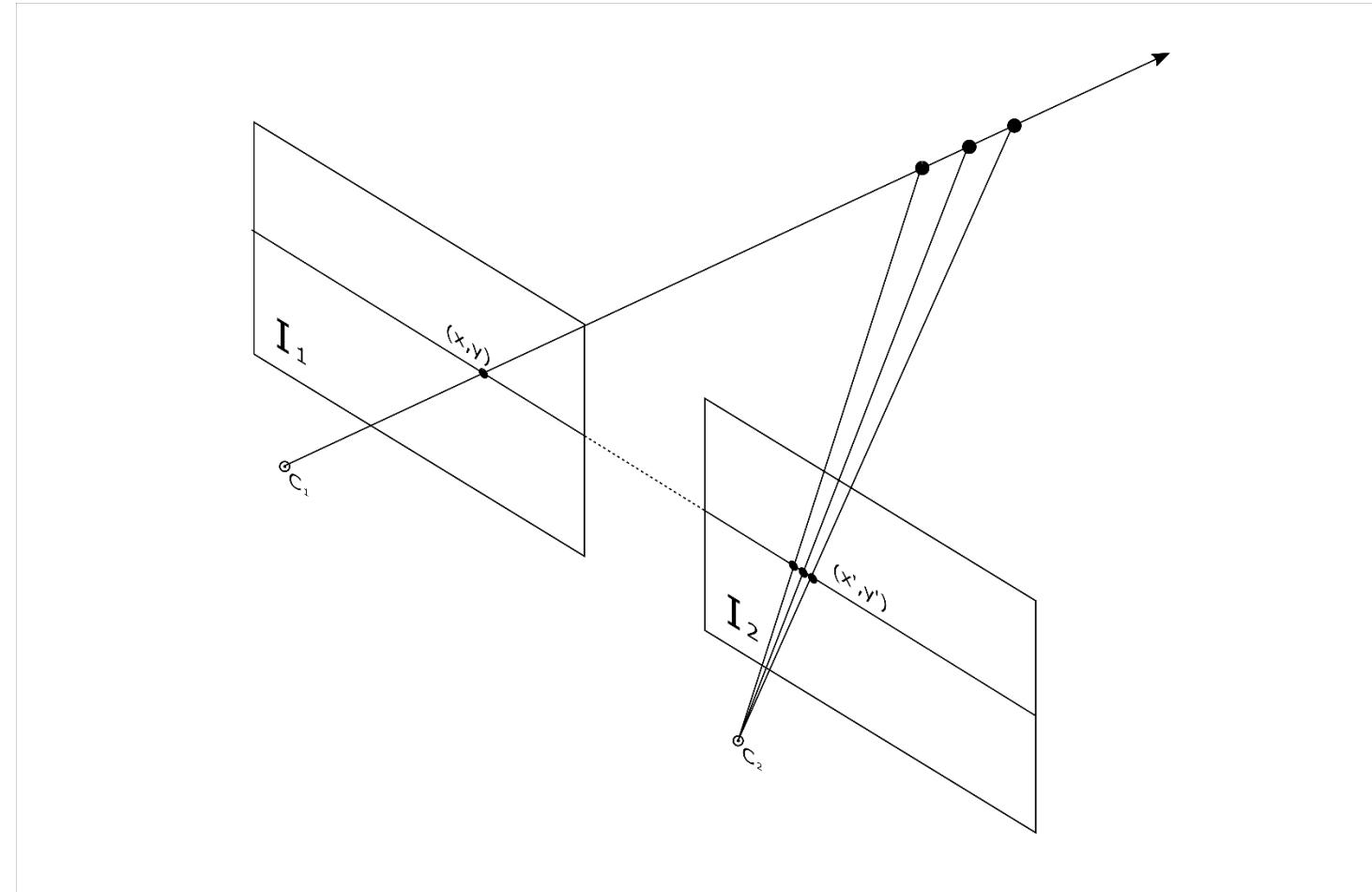
Epipolar Geometry

- What about this case?
- What's the epipolar plane?
- Can we find the epipole of I_1 ?
- What about the epipole of I_2 ?



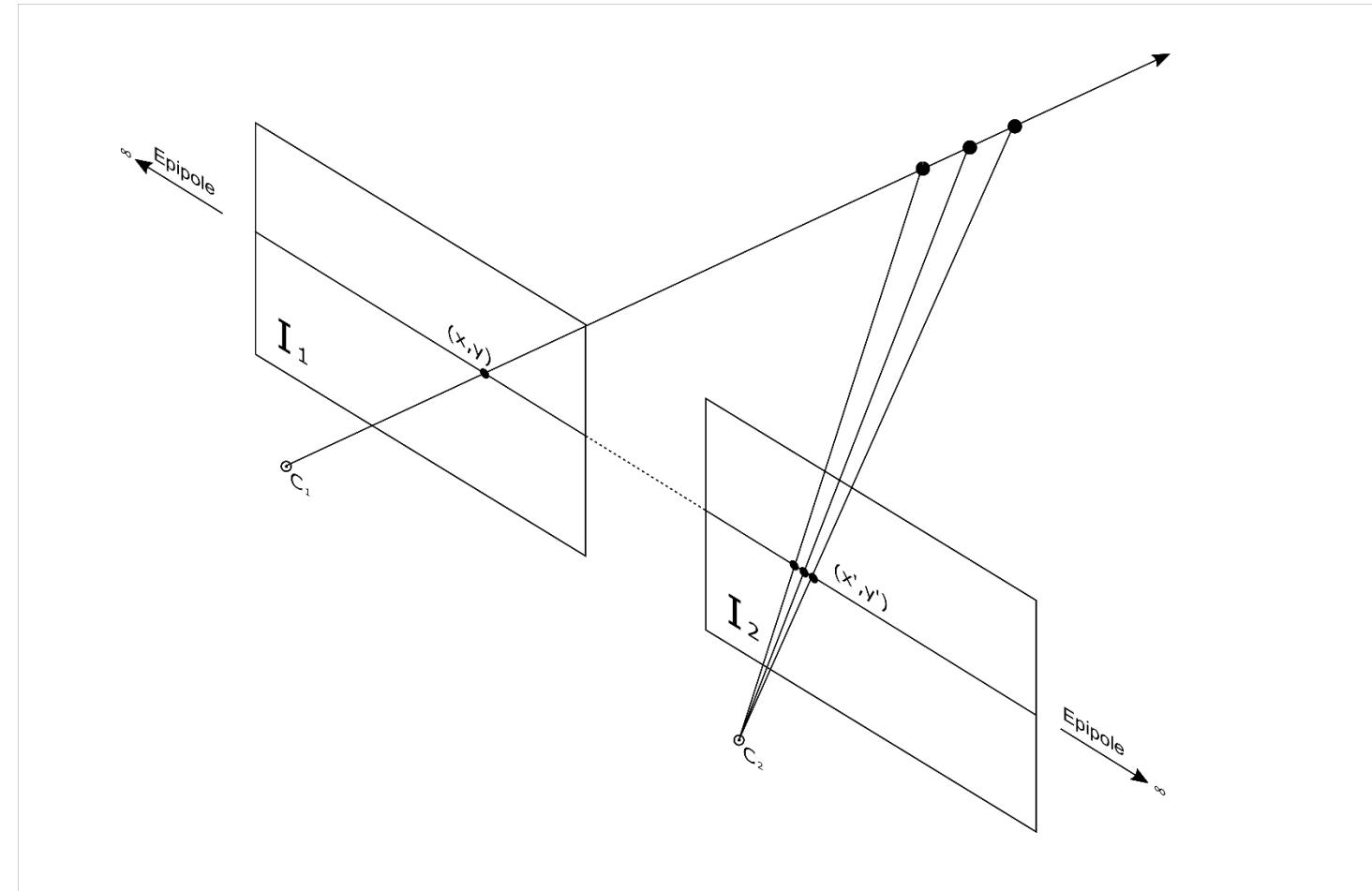
Epipolar Geometry:

- How about this case?



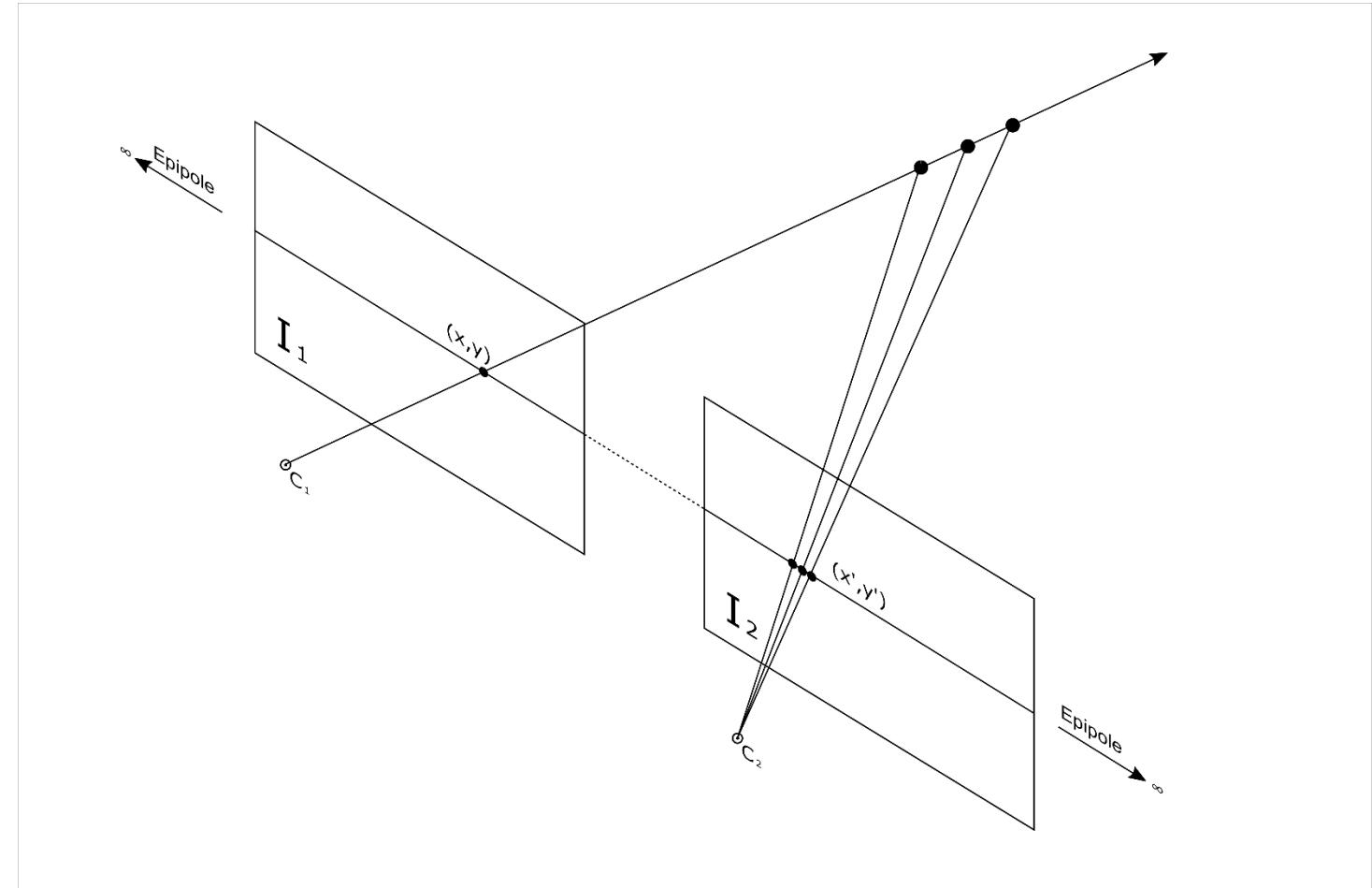
Epipolar Geometry:

- How about this case?
- Where are the epipoles?



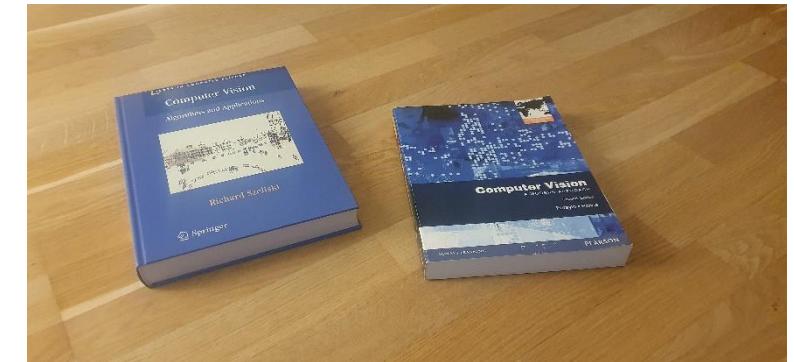
Epipolar Geometry:

- How about this case?
 - Where are the epipoles?
-
- How would such a case be useful
 - The scanline is used in Disparity calculation



Example of Epipolar Lines “in the wild”

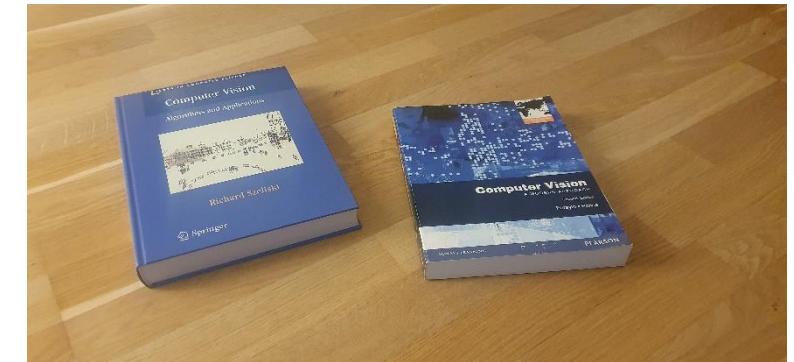
- Let's think of this example:



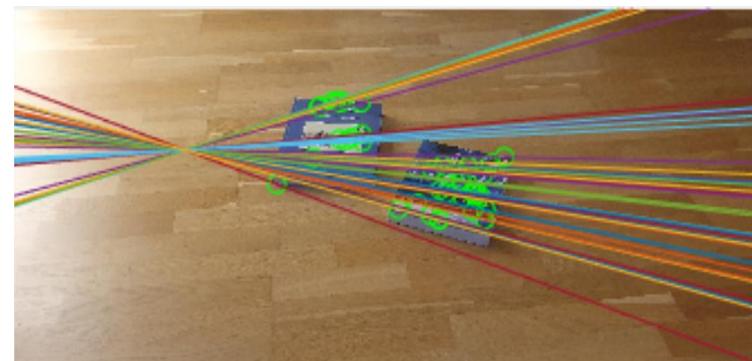
- Where should the epipole be?

Example of Epipolar Lines “in the wild”

- Let's think of this example:

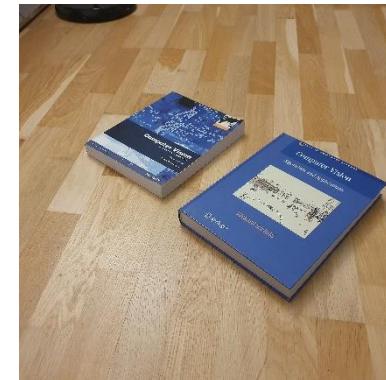
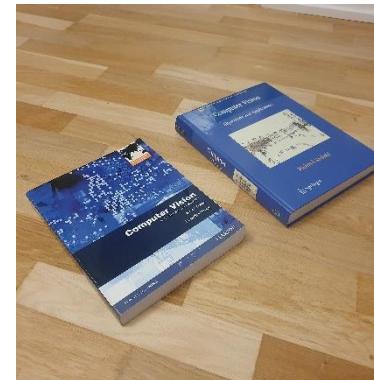


- Where should the epipole be?



Example of Epipolar Lines “in the wild”

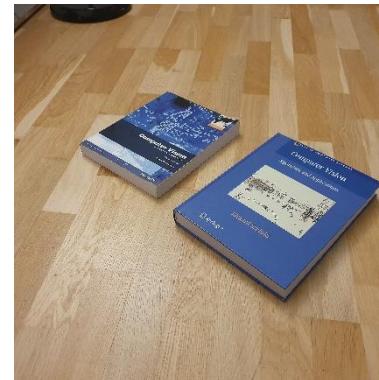
- What about this:



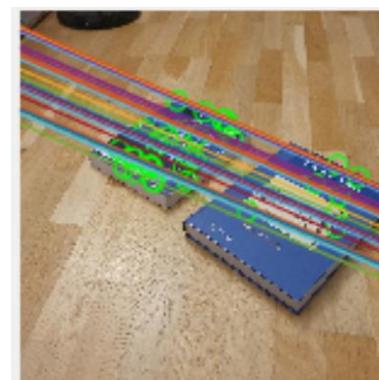
- How should the epipolar Lines look like?

Example of Epipolar Lines “in the wild”

- What about this:



- How should the epipolar Lines look like?

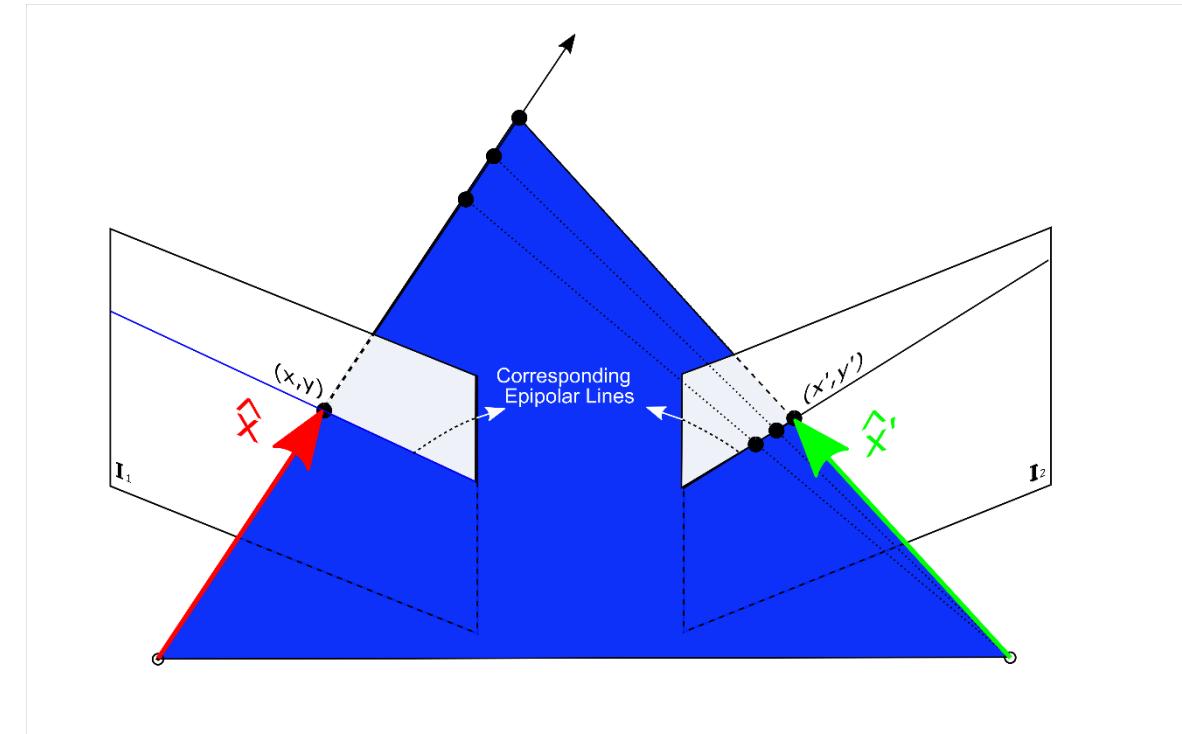


Essential Matrix

- Assuming two calibrated stereo pairs:
- We can express the points x, y from the image plane to homogeneous coordinates \hat{x} and \hat{x}' using the inverse of the camera matrix

$$\hat{x} = K^{-1}x = X$$

$$\hat{x}' = K'^{-1}x' = X'$$



Essential Matrix

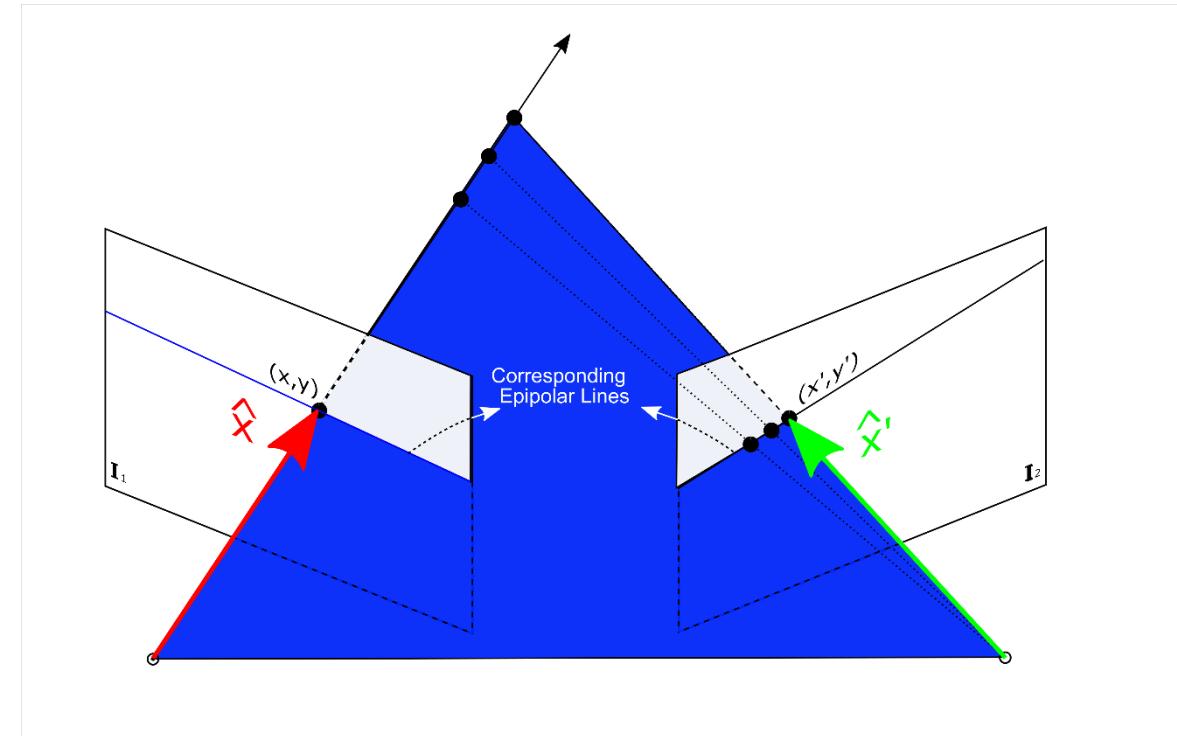
- We can express the left homogeneous point to the right one:
- $\hat{x} = R * \hat{x}' + T$, where R is the Rotation Matrix and T the translation vector
- Then we can prove that there is a Matrix connecting the two points \hat{x}, \hat{x}'
- Trying to eliminate the left side by Applying cross product and then dot product

$$T \mathbf{x} \hat{x} = T \mathbf{x} R * \hat{x}' + T \mathbf{x} T$$

$$\hat{x} \cdot T \mathbf{x} \hat{x} = \hat{x} \cdot T \mathbf{x} R * \hat{x}' + 0$$

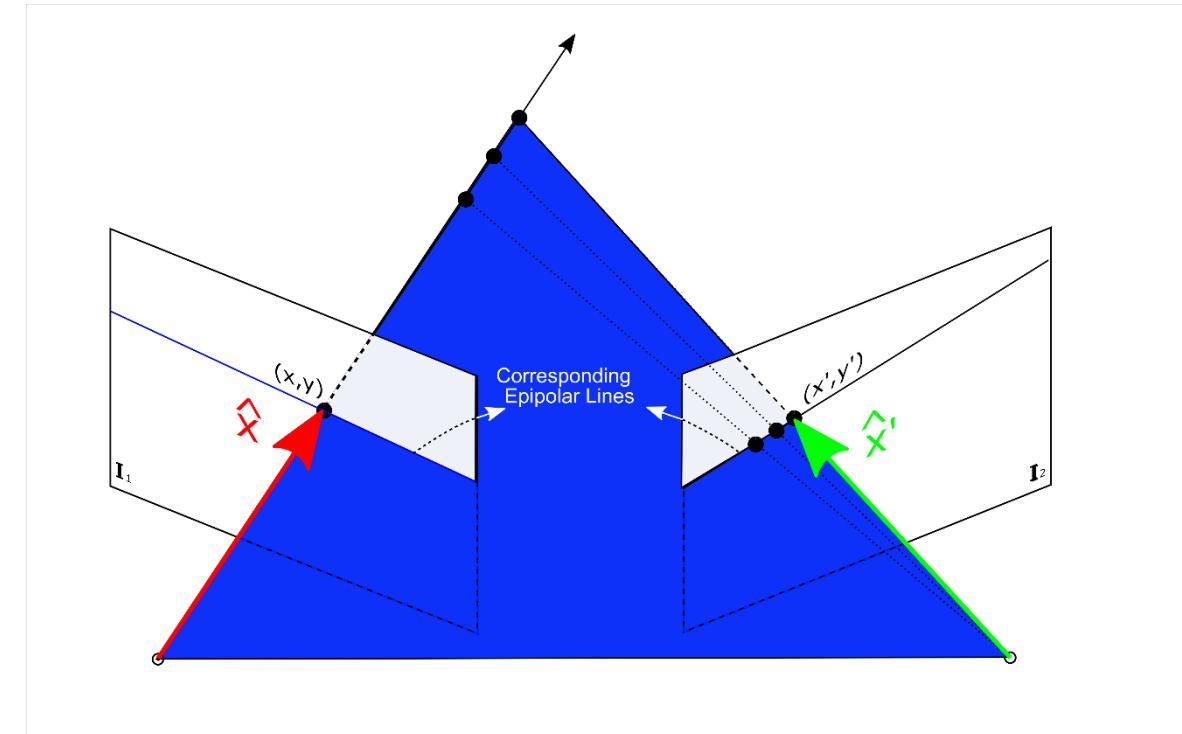
$$0 = \hat{x} \cdot T \mathbf{x} R * \hat{x}'$$

- Or we can write it in matrix form: $\hat{x}^T E \hat{x}' = 0$ with $E = [t]_x R$, where $[t]_x$ is the skew symmetric matrix



Essential Matrix

- The essential matrix $E = [t]_x R$ is a 3×3 matrix, for which:
 - $E x'$ is the epipolar line associated with x' ($l = E x'$)
 - $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
 - $E e' = 0$ and $E^T e = 0$
 - E is singular (rank two)
 - E has five degrees of freedom



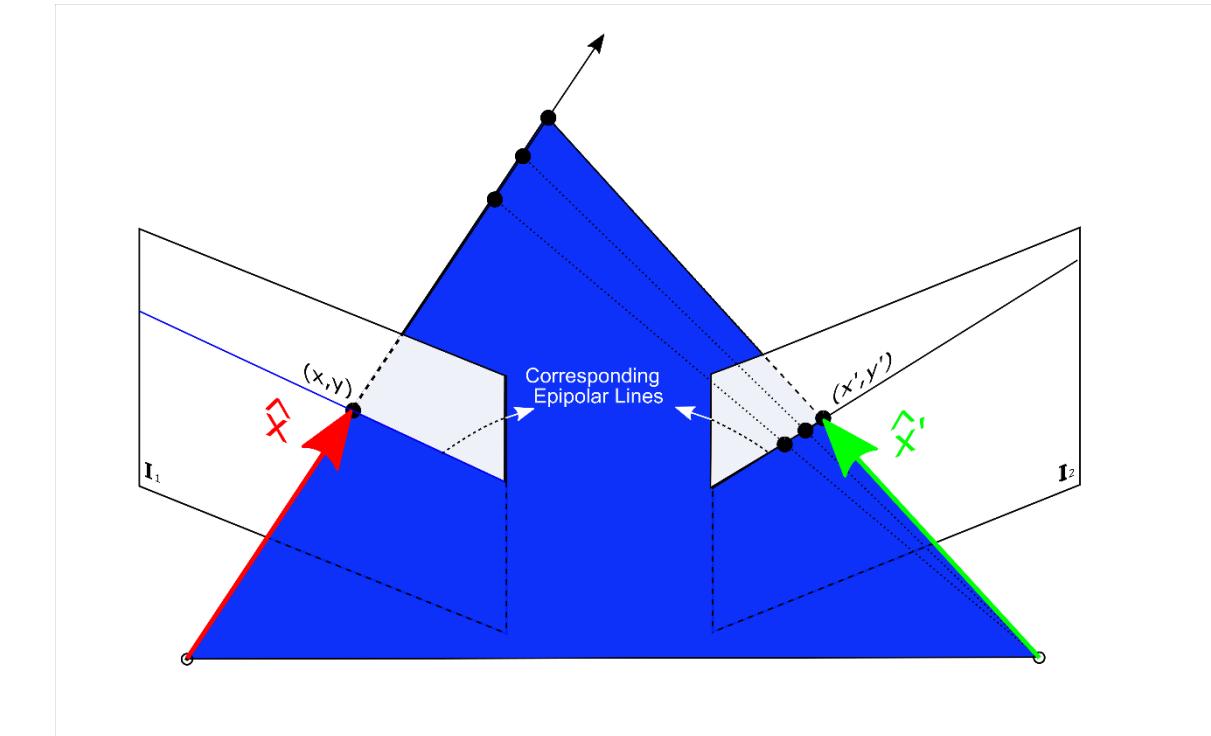
Fundamental Matrix

- We know how to get from a homogeneous point in one camera to another
- How can we get directly from one image to another?

$$\hat{x}^T E \hat{x}' = 0$$

$$\hat{x} = K^{-1}x \quad \rightarrow \quad x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1}$$

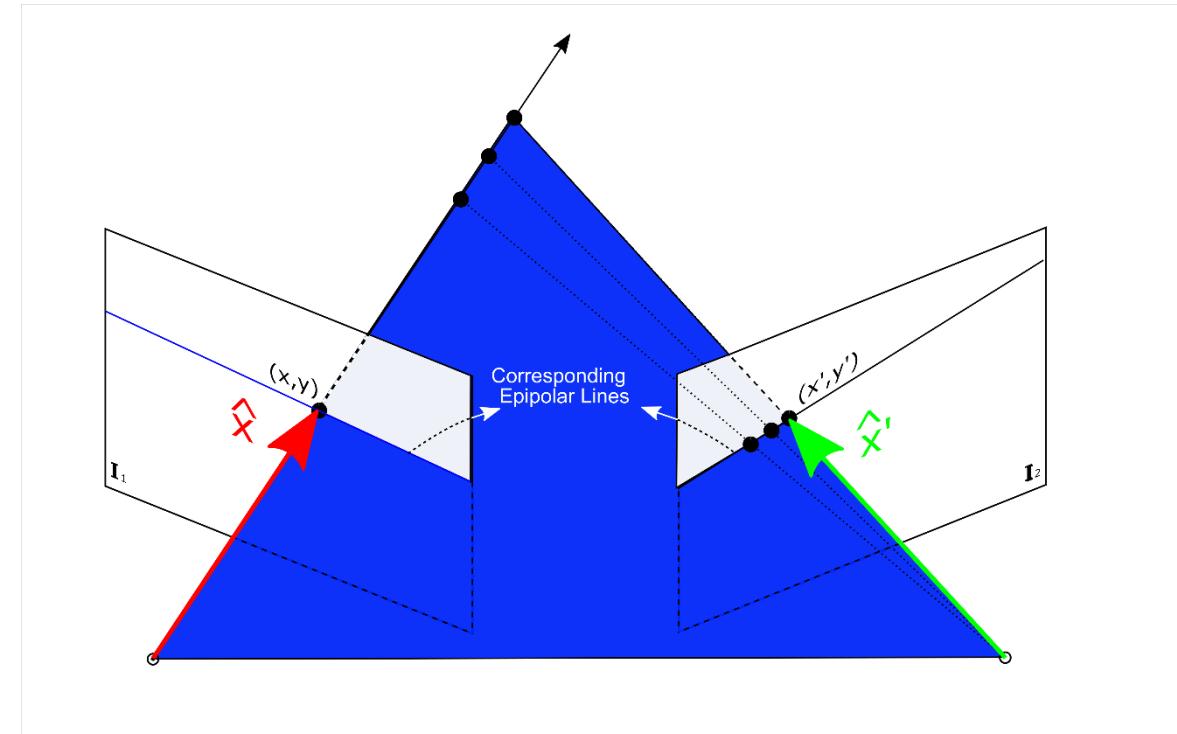
$$\hat{x}' = K'^{-1}x'$$



Which is the fundamental matrix

Fundamental Matrix

- The fundamental matrix $F = K^{-T} E K'^{-1}$ is a 3×3 matrix, for which:
 - Fx' is the epipolar line associated with x'
 - F^Tx is the epipolar line associated with x
 - $Fe' = 0$ and $F^Te = 0$
 - F is singular (rank two): $\det(F)=0$
 - F has seven degrees of freedom:



How to compute the ~~Homography~~ Fundamental Matrix

- Similar to DLT method as before

It's called the 8 point algorithm as we need 8 points to solve it

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

$$xx'f_{11} + xy'f_{12} + xf_{13} + yx'f_{21} + yy'f_{22} + yf_{23} + x'f_{31} + y'f_{32} + f_{33} = 0$$

$$A\mathbf{f} = \begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots \\ x_ny_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

How to compute the Fundamental Matrix

- Homography (No Translation)

- Correspondence Relation

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \Rightarrow \mathbf{x}' \times \mathbf{H}\mathbf{x} = \mathbf{0}$$

1. Normalize image coordinates

$$\tilde{\mathbf{x}} = \mathbf{T}\mathbf{x} \quad \tilde{\mathbf{x}}' = \mathbf{T}'\mathbf{x}'$$

2. RANSAC with 4 points

- Solution via SVD

3. De-normalize:

$$\mathbf{H} = \mathbf{T}'^{-1} \tilde{\mathbf{H}} \mathbf{T}$$

- Fundamental Matrix (Translation)

- Correspondence Relation

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

1. Normalize image coordinates

$$\tilde{\mathbf{x}} = \mathbf{T}\mathbf{x} \quad \tilde{\mathbf{x}}' = \mathbf{T}'\mathbf{x}'$$

2. RANSAC with 8 points

- Initial solution via SVD

- Enforce $\det(\tilde{\mathbf{F}}) = 0$ by SVD

3. De-normalize:

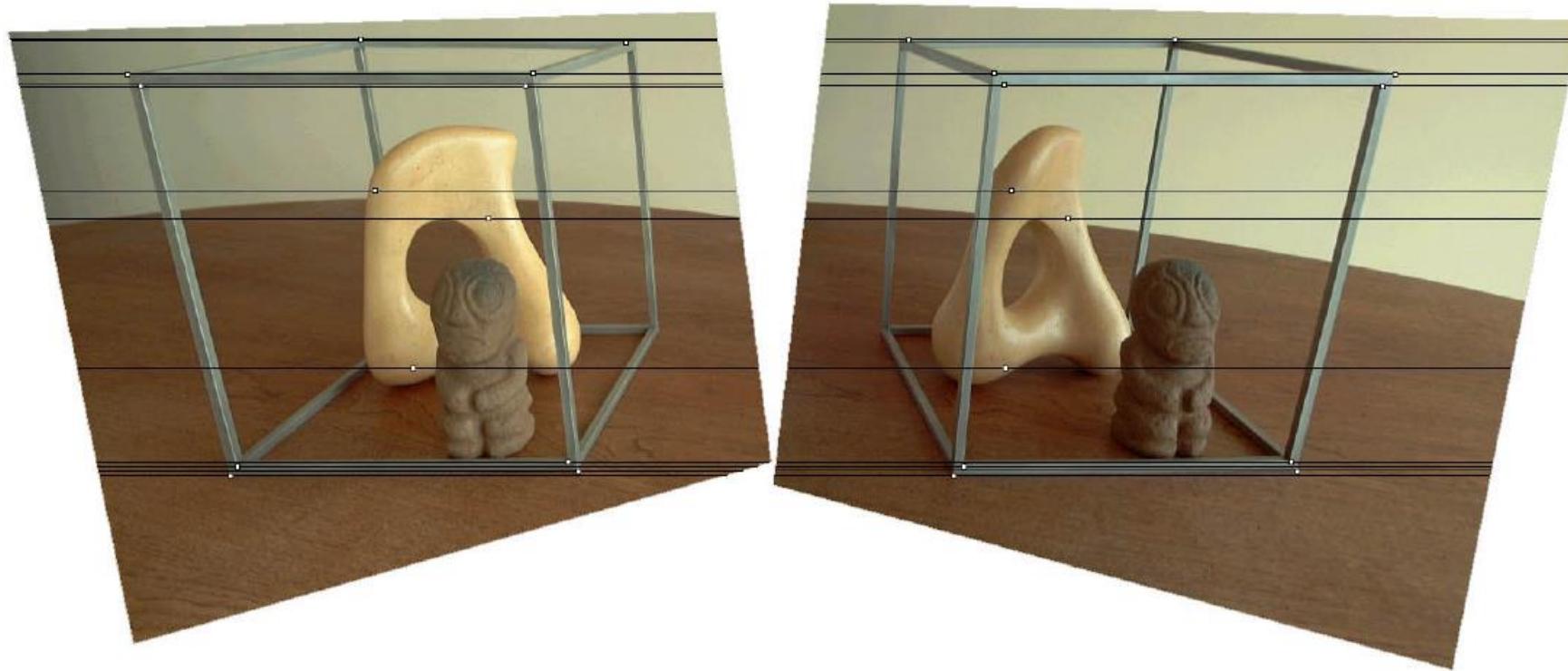
$$\mathbf{F} = \mathbf{T}'^T \tilde{\mathbf{F}} \mathbf{T}$$

Epipolar Geometry to Rectified Epipolar Geometry

- To finally go full circle:
 - To be able to use the stereo in a block matching algorithm to produce 3D points we must first **convert to rectified stereo**
 - **How would you do that?**

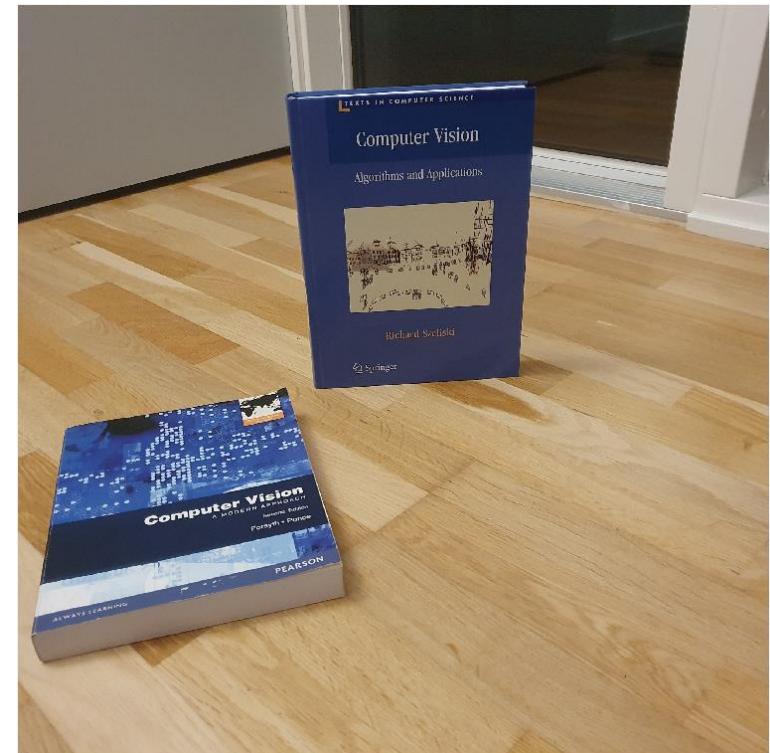


Rectified Epipolar Geometry



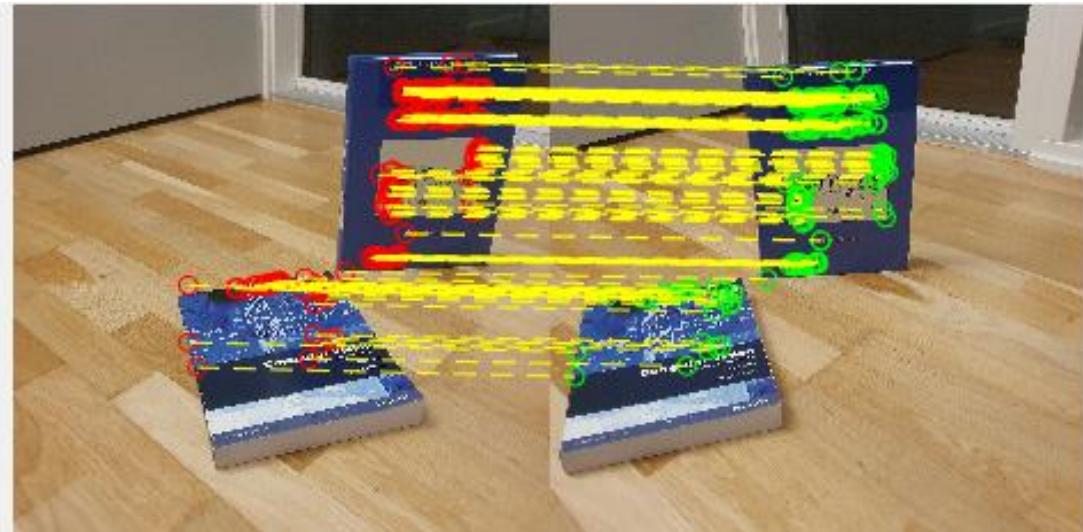
An Example

- Assume these two images

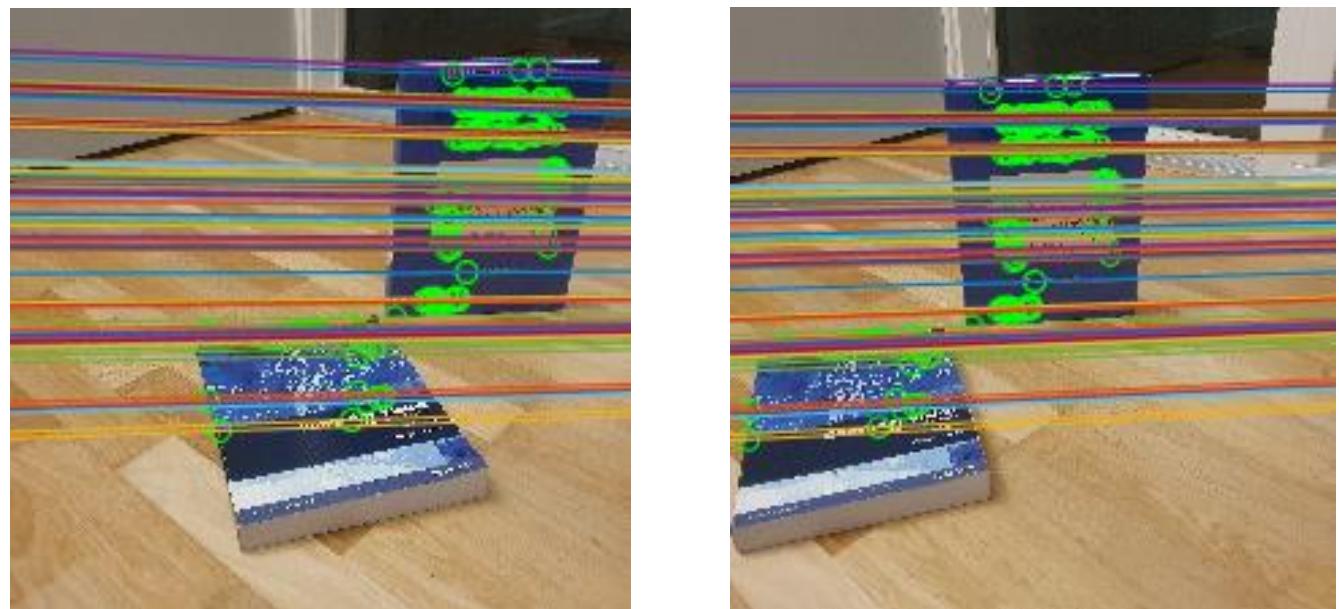


An Example

- First step is to match some points

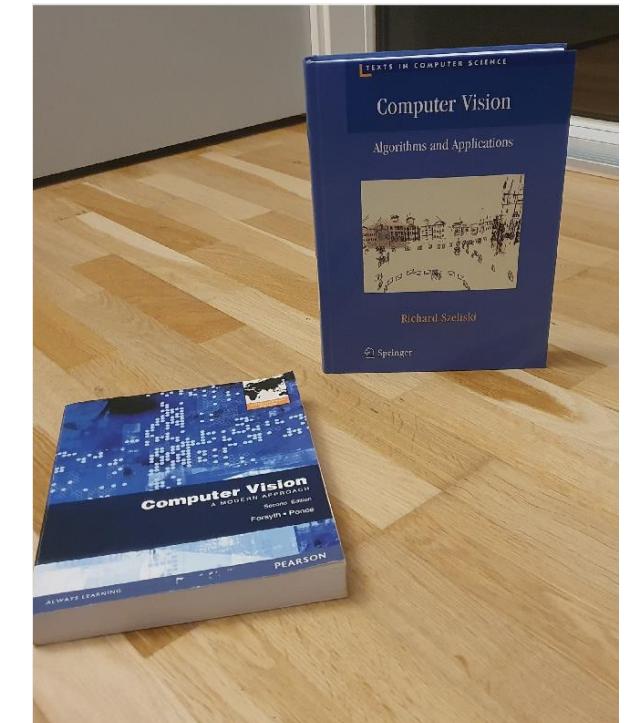
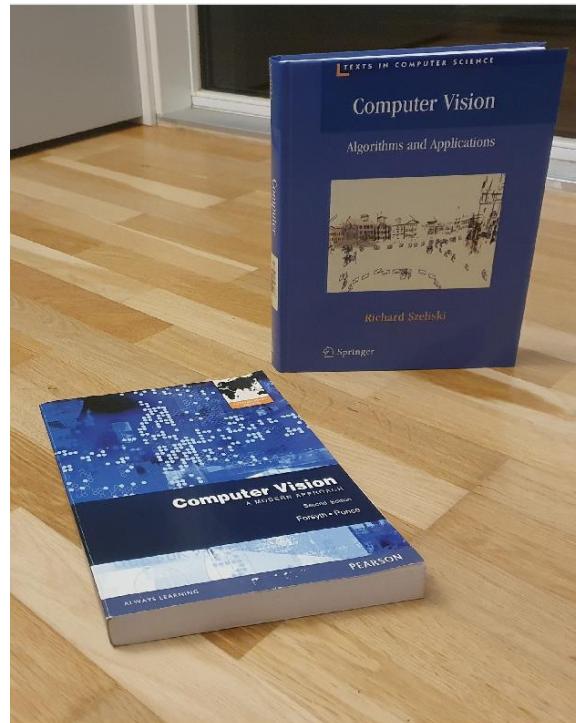


- Next, calculate the fundamental matrix



An Example

- Finally calculate the rectified images



- Notice how the images are now scan line



Perception for Autonomous Systems 31392:

Epipolar Geometry and the Fundamental Matrix

Lecturer: Evangelos Boukas—PhD

Perception for Autonomous Systems

Lecture 4 - Calibration and Stereo Vision 2 (22/02/2021)

Outline/Content:

- Camera matrix (Book A 2.1.5)
- Distortion coefficients (Book A 2.1.6)
- Calibration (Paper A)
- Undistortion
- Algebraic Derivation of Stereo Vision (Book B 7.1)
 - (Essential and) Fundamental Matrix
- Stereo Calibration and Rectification
 - Ranging

Additional Reading Material: Paper A; Z. Zhang, "A flexible new technique for camera calibration," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, pp. 1330-1334, Nov. 2000. doi: 10.1109/34.888718

Epipolar Geometry: General Case

In the general case, the following assumptions can be made for the determination of the components in the epipolar geometry:

- **2 Angled** camera views
- Rays passing through the camera center

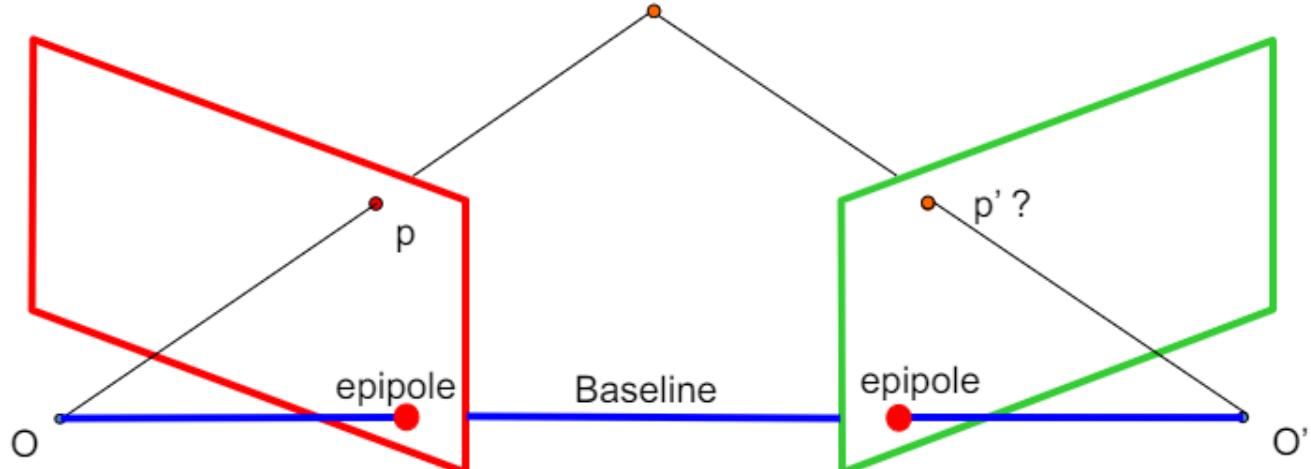
NOTE: If the camera views are parallel, we don't

With this knowledge, we can easily obtain the following two components (as shown in the next figure):

- Baseline
- Epipolar lines
- Epipolar plane
- Epipoles

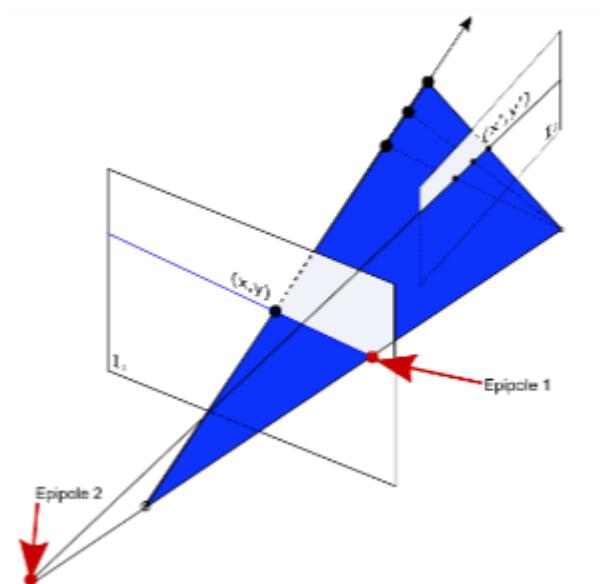
Necessary knowledge from the previous lecture:

- *Baseline*: The line connecting the two camera centers.
- *Epipole*: Point of intersection of baseline with the image plane.
- *Epipolar plane*: The plane that contains the two camera centers and a 3D point in the world.
- *Epipolar line*: Intersection of the epipolar plane with each image plane.

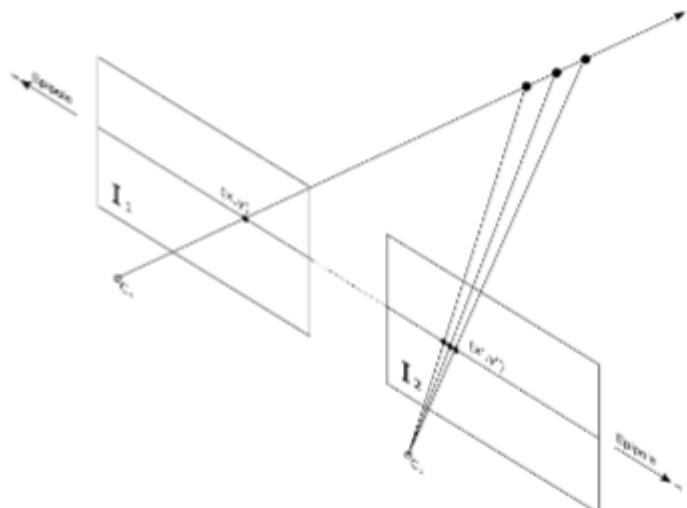


Epipolar Geometry: Special Case

For special cases it's not always possible to determine the epipolar geometry components.



In the first case, we have two staggered images. We're able to determine the **Baseline**, **Epipolar plane**, **Epipolar lines** and the **Epipoles**. However, Due to the placement of the cameras it's noticeable that one of the epipolar poles is not part of the epipolar plane.



Example 2:

In the second case we have vertically aligned images. We can determine the **baseline**, **epipolar plane**, **epipolar lines** but NOT the **epipoles**. This is because the alignment of the two images makes them parallel to the baseline. Therefore, an intersection between the baseline and the epipolar line is not possible.

This case can be useful by using the scanline to calculate the disparity

Essential Matrix [Lecture Slides - Carnegie University](#)

The Essential Matrix E is a 3×3 matrix, and it relates corresponding image points between both cameras, given the rotation and translation.

1. Given a point in one image, multiplying by the essential matrix will tell us the epipolar line in the second view.
2. It's assumed that both cameras are calibrated. For uncalibrated cameras, see section Fundamental Matrix.
3. See the link for the lecture slides of the Carnegie Mellon university, as it is perfectly explained, but unfeasible to include into this section.

properties of the E matrix

Longuet-Higgins equation

$$\mathbf{x}'^\top \mathbf{E} \mathbf{x} = 0$$

Epipolar lines

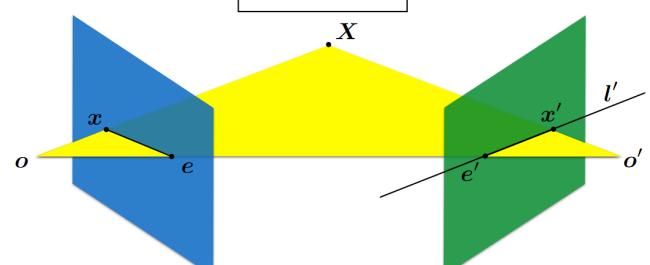
$$\begin{aligned} \mathbf{x}^\top \mathbf{l} &= 0 & \mathbf{x}'^\top \mathbf{l}' &= 0 \\ \mathbf{l}' &= \mathbf{E} \mathbf{x} & \mathbf{l} &= \mathbf{E}^T \mathbf{x}' \end{aligned}$$

Epipoles

$$\mathbf{e}'^\top \mathbf{E} = \mathbf{0} \quad \mathbf{E} \mathbf{e} = \mathbf{0}$$

(points in normalized camera coordinates)

$$\mathbf{E} \mathbf{x} = \mathbf{l}'$$



Characteristics:

- $E x'$ is the epipolar line associated with x' ($l = E x'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $E e' = 0$ and $E^T e = 0$
- E is singular (rank two)
- E has five degrees of freedom

Fundamental Matrix [Lecture Slides - Carnegie University](#)

The 3x3 Fundamental matrix is a generalization of the Essential matrix, where the assumption of calibrated cameras is removed.

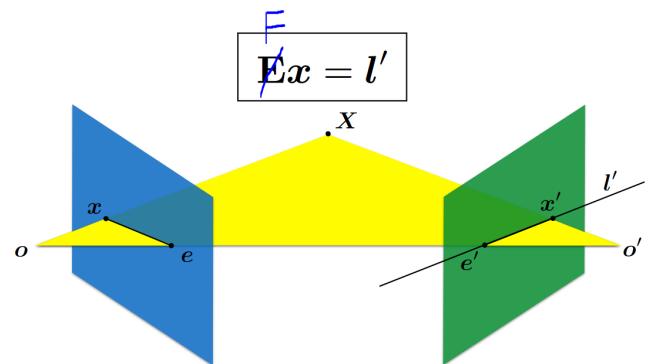
properties of the \mathbf{F} matrix

Longuet-Higgins equation $\mathbf{x}'^\top \mathbf{E} \mathbf{x} = 0$

Epipolar lines $\mathbf{x}^\top \mathbf{l} = 0$ $\mathbf{x}'^\top \mathbf{l}' = 0$
 $\mathbf{l}' = \mathbf{E} \mathbf{x}$ $\mathbf{l} = \mathbf{E}^T \mathbf{x}'$

Epipoles $\mathbf{e}'^\top \mathbf{E} = 0$ $\mathbf{E} \mathbf{e} = 0$

(points in **image** coordinates)



Characteristics:

- $\mathbf{F} x'$ is the epipolar line associated with x'
- $\mathbf{F}^T x$ is the epipolar line associated with x
- $\mathbf{F} e' = 0$ and $\mathbf{F}^T e = 0$
- \mathbf{F} is singular (rank two): $\det(\mathbf{F})=0$
- \mathbf{F} has seven degrees of freedom:

See the link for the lecture slides of the Carnegie Mellon university, as it's well explained, but unfeasible to include into this section.

How to compute the Fundamental Matrix [Lecture Slides - Carnegie University](#)

The method for computing the Fundamental Matrix is called the 8 point algorithm, as 8 points are needed to solve it.

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

$$xx'f_{11} + xy'f_{12} + xf_{13} + yx'f_{21} + yy'f_{22} + yf_{23} + x'f_{31} + y'f_{32} + f_{33} = 0$$

$$Af = \begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots \\ x_ny_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

- Homography (No Translation)
- Correspondence Relation $\mathbf{x}' = \mathbf{H}\mathbf{x} \Rightarrow \mathbf{x}' \times \mathbf{H}\mathbf{x} = \mathbf{0}$
- 1. Normalize image coordinates
 $\tilde{\mathbf{x}} = \mathbf{T}\mathbf{x}$ $\tilde{\mathbf{x}}' = \mathbf{T}'\mathbf{x}'$
- 2. RANSAC with 4 points
 - Solution via SVD
- 3. De-normalize:
 $\mathbf{H} = \mathbf{T}'^{-1} \tilde{\mathbf{H}} \mathbf{T}$
- Fundamental Matrix (Translation)
- Correspondence Relation $\mathbf{x}'^T \mathbf{F} \mathbf{x} = 0$
- 1. Normalize image coordinates
 $\tilde{\mathbf{x}} = \mathbf{T}\mathbf{x}$ $\tilde{\mathbf{x}}' = \mathbf{T}'\mathbf{x}'$
- 2. RANSAC with 8 points
 - Initial solution via SVD
 - Enforce $\det(\tilde{\mathbf{F}}) = 0$ by SVD
- 3. De-normalize:
 $\mathbf{F} = \mathbf{T}'^T \tilde{\mathbf{F}} \mathbf{T}$

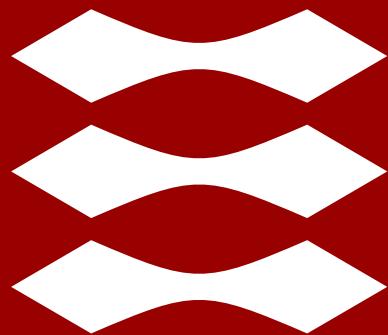
3. See the link for the lecture slides of the Carnegie Mellon university, as it is perfectly explained, but unfeasible to include into this section.

Epipolar Geometry to Rectified Epipolar Geometry

Necessary steps:

1. Match some points
2. Calculate the fundamental matrix
3. Calculate the rectified images

DTU



Lazaros Nalpantidis

3D Point Cloud Processing - Pose Estimation

- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

Why do we need 3D Point Clouds?

Why do we need 3D Point Clouds?

- World is 3D
 - Objects are not entirely described by 2D images
 - 3D geometry and shape are often important
- *However,*
 - *operations in 3D are computationally heavy*
 - *SIFT/SURF/... do not work in point clouds*

Bonus:

<https://github.com/dataarts/radiohead>

What is “Pose”?

What is “Pose”?

- Pose
 - the transformation (translation + rotation) needed to map one point cloud (model) to another point cloud (model) of the same (fully or partially) object or scene.
- ...or equivalently
 - “...determining a camera’s position relative to a known 3D object or scene...” (Szelinski).

Why is 3D Pose Estimation Important?

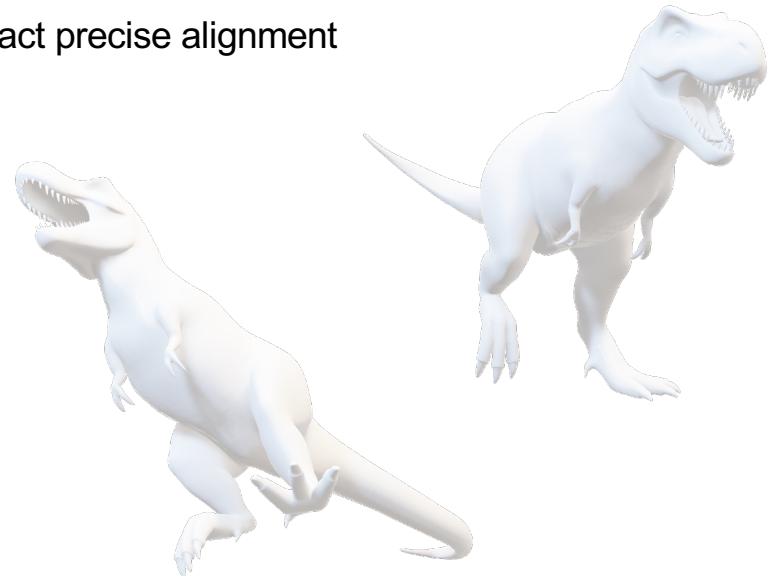
Why is 3D Pose Estimation Important?

- Many applications in Autonomous Systems
 - Robot Grasping/Manipulation
 - Quality Inspection
 - Augmented reality
 - Progressive map building

- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

Point Cloud Registration

- Generally 2 steps are needed to register 2 given point clouds of the same object (fully or partially)
 - Global alignment
 - The 2 models are "roughly aligned"
 - Local alignment
 - Starting from a "rough" initial alignment, find the exact precise alignment



- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

Local Alignment

- Consider 2 models (point clouds) of the same object (fully or partially) that are almost aligned.
 - What is the difference of their pose?
 - ...or equivalently...
 - What is the transformation that can fully align them?



Local Alignment - ICP

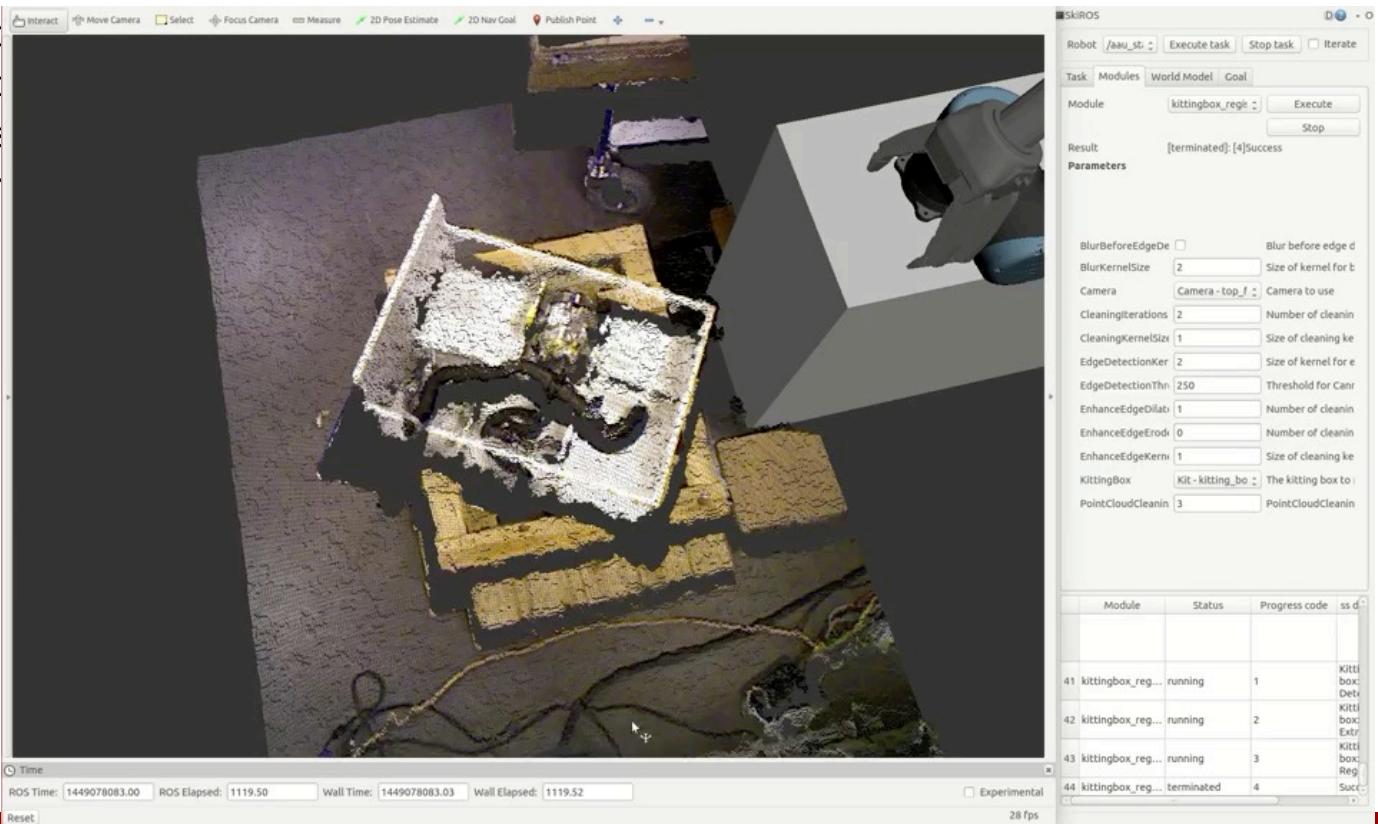
- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239-256, Feb. 1992.

Local Alignment - ICP

- Iterative Closest Point (ICP)

1. For each object
2. Use all pairs (points)
3. Apply the transformation
4. Repeat steps



Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. **For each object point p in model 1, find the nearest point q in model 2**
 2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. **For each object point p in model 1, find the nearest point q in model 2**
 2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

– How to compare all possible pairs for proximity, in an efficient manner?

- If there are X points in model 1 and Y points in model 2, then XY distance calculations are required
- However, we can significantly speed up this search by using k-d trees!

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. **Use all pairs (p,q) to estimate the transformation from model 1 to model 2**
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. **Use all pairs (p,q) to estimate the transformation from model 1 to model 2**
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

– How to estimate the transformation from model 1 to model 2, given pairs (p,q) ?

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. **Use all pairs (p,q) to estimate the transformation from model 1 to model 2**
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

- How to estimate the transformation from model 1 to model 2, given pairs (p,q) ?
 - Kabsch Algorithm / Procrustes Analysis:
 - Translate the centroids of both models to the origin of the coordinate system $(0,0,0)$.
 - » Compute centroids c_p , c_q of both models
 - » Subtract from each point coordinates the coordinates of its corresponding centroid:
 $p' = p - c_p$ and $q' = q - c_c$
 - Compute Covariance Matrix: $C_{pq} = \Sigma p'q'^T$
 - Compute the optimal rotation
 - » Calculate the Singular Value Decomposition (SVD) of the covariance matrix: $C_{pq} = USV^T$
 - » Calculate rotation matrix: $R = UV^T$
 - Compute the optimal translation: $T = c_q - Rc_p$

Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. **Use all pairs (p,q) to estimate the transformation from model 1 to model 2**
 3. Apply the transformation to the points of Model 1
 4. Repeat steps 1-3 until convergence/stop criterion is met

- How to estimate the transformation from model 1 to model 2, given pairs (p,q) ?
 - Kabsch Algorithm / Procrustes Analysis

- QUESTION:
 - Why ICP needs to employ Kabsch algorithm at every iteration?

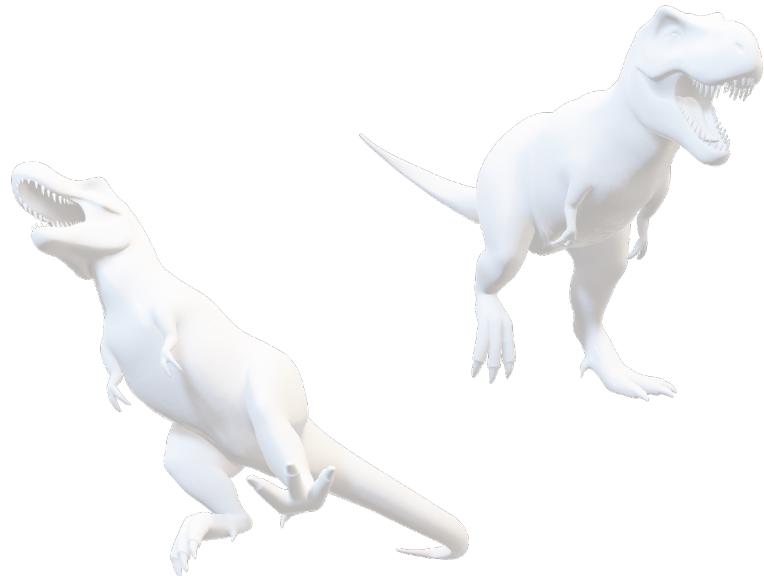
Local Alignment - ICP

- Iterative Closest Point (ICP)
 1. For each object point p in model 1, find the nearest point q in model 2
 2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
 3. **Apply the transformation to the points of Model 1**
 4. **Repeat steps 1-3 until convergence/stop criterion is met**

- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

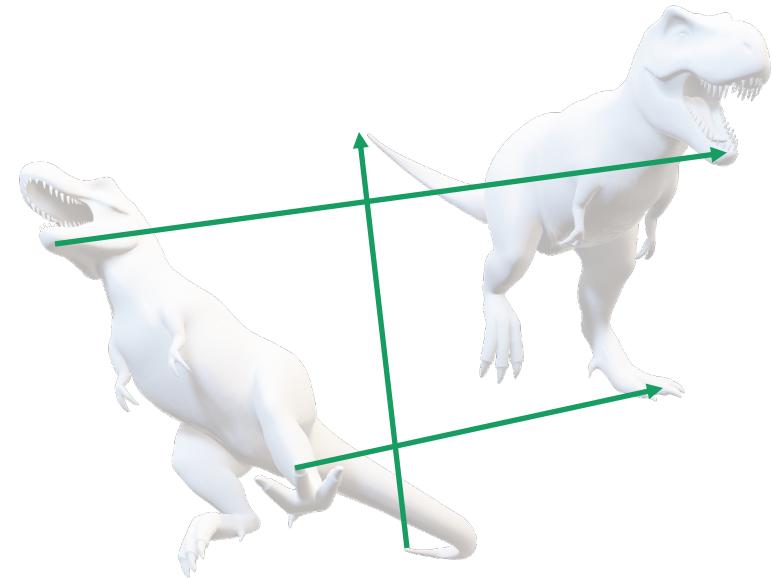
Global Alignment

- Consider 2 models (point clouds) of the same object (fully or partially)
 - How can we “roughly” align them?



Global Alignment

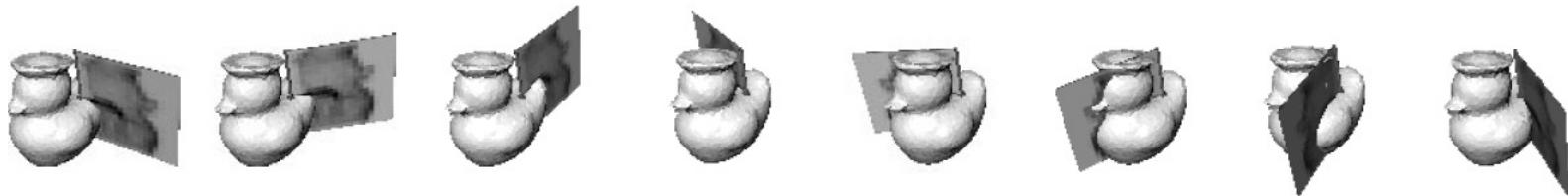
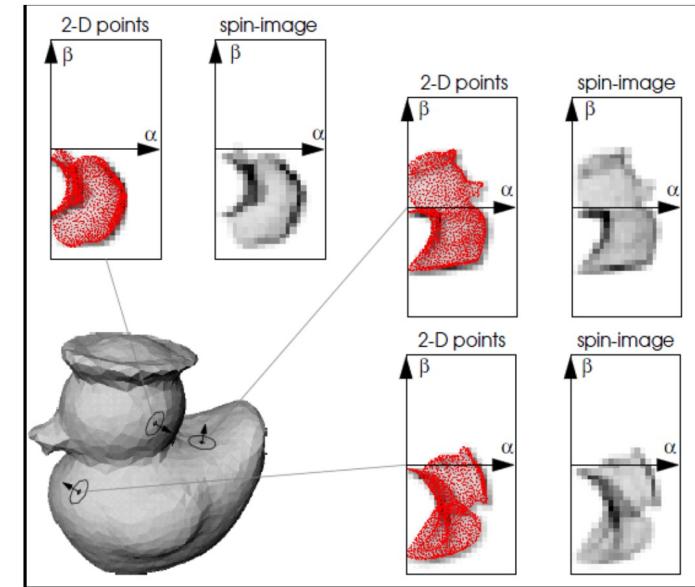
- Consider 2 models (point clouds) of the same object (fully or partially)
 - How can we “roughly” align them?
 - We cannot use ICP, if the initial poses are too different.
 - Can we use some kind of (3D) “features” and match them?



Global Alignment – 3D Feature Descriptors

- **Spin Images**

- “Spin” a discretized 2D grid around the surface normal of a point.
- Accumulate neighboring pixels in the grid bins, while spinning.
- The descriptor is the 2D grid (image) where each element (pixel) contains the number of accumulated points.



A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433-449, May 1999.

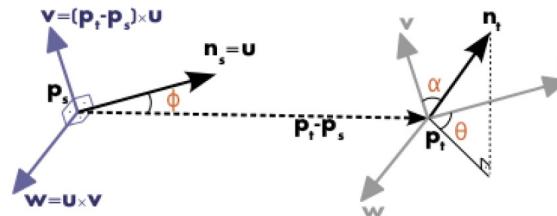
Global Alignment – 3D Feature Descriptors

- PFH (Point Feature Histograms)

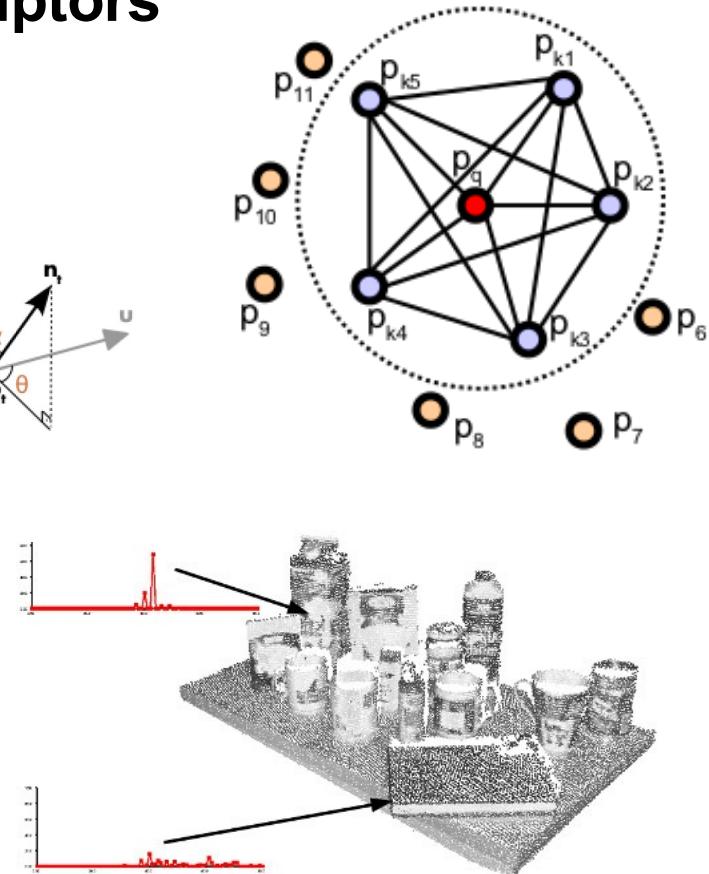
$$\alpha = \arccos(v \cdot n_t)$$

$$\phi = \arccos\left(u \cdot \frac{(p_t - p_s)}{\|p_t - p_s\|_2}\right)$$

$$\theta = \arctan(w \cdot n_t, u \cdot n_t)$$



- Calculate these 3 values for all pairs of points within a radius r from the considered point (maybe also their distance d)
- The set of all triplets/quadruplets are binned in a histogram – This is the multi-dimensional descriptor

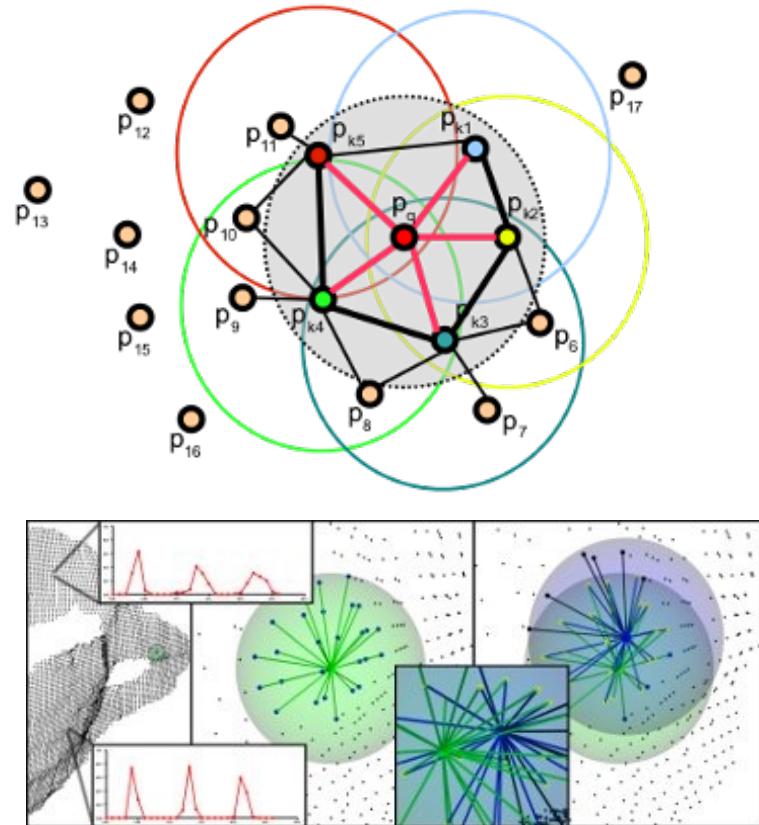


R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.

R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning Point Cloud Views using Persistent Feature Histograms," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

Global Alignment – 3D Feature Descriptors

- **FPFH (Fast Point Feature Histograms)**
 - Simplification/Approximation of the PFH formulation
 - reduces the computational complexity
 - retains most of the discriminative power of PFH.
 - Algorithm:
 - Find all oriented points in a spherical neighborhood of radius r around each point (k-d tree)
 - Compute relative angles using surface normals and direction vector from the source point to each neighbor
 - The descriptor is a multi-dimensional histogram

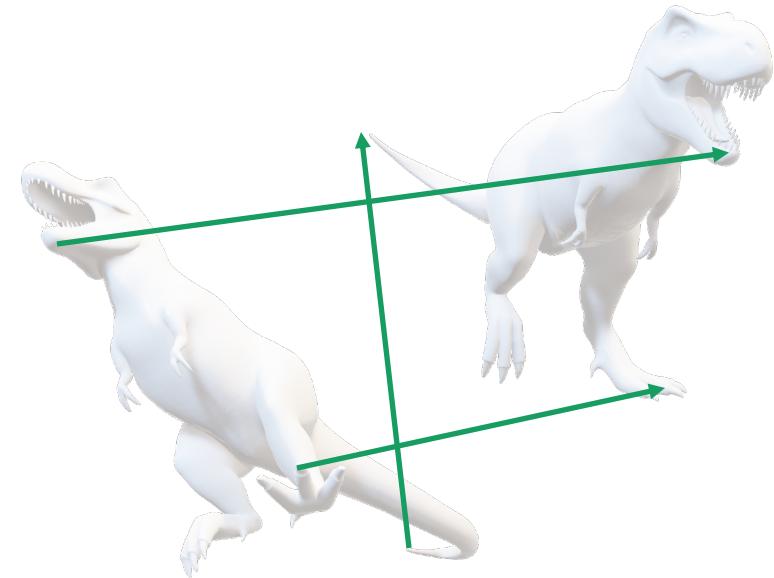


R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.

R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning Point Cloud Views using Persistent Feature Histograms," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

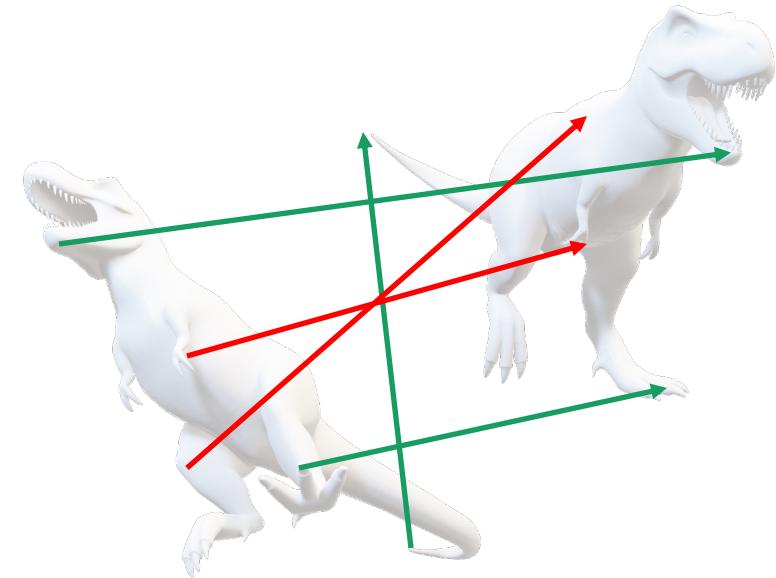
Global Alignment

- Consider 2 models (point clouds) of the same object (fully or partially)
 - How can we “roughly” align them?
 - We cannot use ICP, if the initial poses are too different.
 - Can we use some kind of (3D) “features” and match them?



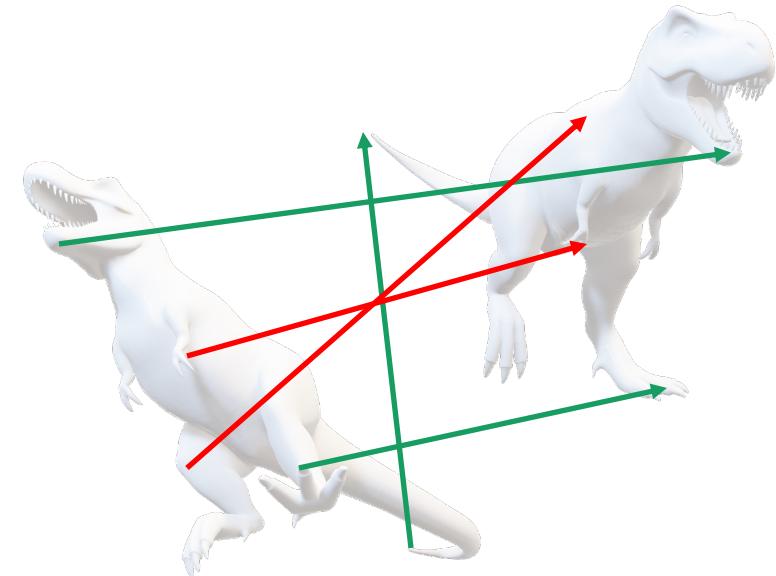
Global Alignment

- Matching 3D features generally results in many false-matches
 - Big additional computational cost, brings only small increase in the matching accuracy 😞
 - Need for an algorithm that is robust to outliers.



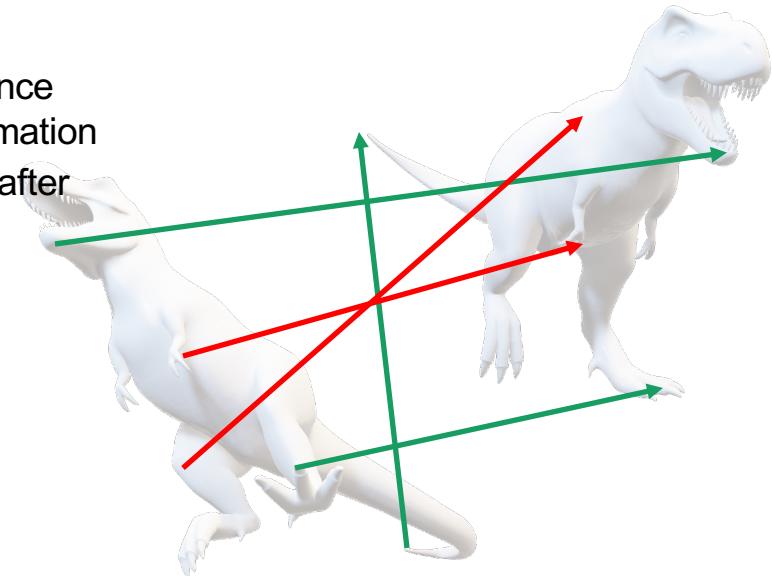
Global Alignment – RANSAC in 3D

- Matching 3D features generally results in many false-matches
 - Big additional computational cost, brings only small increase in the matching accuracy 😞
 - Need for an algorithm that is robust to outliers.
- RANSAC
 - We can use:
 - Model 1 (point cloud with normals)
 - Model 2 (point cloud with normals)
 - Feature matches (containing many outliers)
 - In order to:
 - Estimate the “rough” pose difference between the 2 models



Global Alignment – RANSAC in 3D

- RANSAC for Pose Estimation
 - Randomly choose 3 pairs of matched points
 - Estimate the relative pose
 - Kabsch/ Procrustes
 - Apply the transformation and assess its “validity”:
 - number of inliers: matched point pairs whose distance became “small” after applying the specific transformation
 - Sum of distances between all matched point pairs after transformation
 - ...
 - Keep the transformation with most inliers
 - Repeat random sampling



- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D
- Summary

Summary

- Point clouds provide the geometry of the scene/object (and might also combine it with color information)
- Pose Estimation very important for many tasks of Autonomous Systems
- Point Cloud Registration:
 1. “Rough” alignment (Global)
 - 3D Feature Descriptors (Spin Images, FPH, FPFH, ...)
 - RANSAC for robust model fitting
 2. “Fine-tuning” of alignment (Local)
 - ICP
 - » Kabsch / Procrustes

Lazaros Nalpantidis

3D Point Cloud Processing - Pose Estimation

Perception for Autonomous Systems

Lecture 5 - 3D Point Cloud Processing - Pose Estimation (01/03/2021)

Outline/Content:

- Why do we need 3D Point Clouds?
- Why is Pose Estimation in 3D important?
- Point Cloud Registration
 - Local Alignment
 - Iterative Closest Point (ICP) algorithm
 - Global Alignment
 - 3D Feature Descriptors
 - Spin Images
 - PFH
 - FPFH
 - Random Sample Consensus (RANSAC) in 3D

Reading Material:

- Book A, Sections 6.1 and 6.2
- [Paper 1 - ICP] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, Feb. 1992.
- [Paper 2 - Spin Images] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 433-449, May 1999.
- [Paper 3 - PFH] R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.
- [Paper 4 - FPFH] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning Point Cloud Views using Persistent Feature Histograms," in Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

What is a 3D Point Cloud? [Point Cloud Expert: Florent Poux Medium](#)

A point cloud is a set of data points in a three-dimensional coordinate system. These points are spatially defined by [x, y, z] coordinates and often represent a surface of a physical asset or object. Reality capture devices obtain the external surface in its three dimensions to generate the point cloud. These are commonly obtained through Photogrammetry, LiDAR (Terrestrial Laser Scanning), Mobile Mapping, Aerial LiDAR, depth sensing, sonar, and more recently deep learning through Generative Adversarial Networks.

Photogrammetry

Aerial LiDAR

Photogrammetry**Aerial LiDAR****Why do we need 3D Point Clouds? [Article](#)****The world is in 3D**

- Objects are not entirely described by 2D images
- 3D geometry and shape are often important

However,

- operations in 3D are computationally heavy
- SIFT/SURF/... do not work in point clouds

What is "Pose"

Pose

- the transformation (translation + rotation) needed to map one point cloud (model) to another point cloud (model) of the same (fully or partially) object or scene.

or equivalently

- ... determining a camera's position relative to known 3D object or scene... "(Szelenisk)

Why is 3D Pose Estimation Important? [Article](#)

Many applications in Autonomous Systems

- Robot Grasping/Manipulation
- Quality Inspection
- Augmented reality
- Progressive map building (real-time)
- Localization (real-time)

Other applications:

- rescue and recovery
- explore dense forests

- Rescue and recovery: Robots or drones with LiDAR sensors can reach dangerous locations and scan or generate useful data helping humans to navigate safely at such risky places and avoid mishaps. Natural disasters, unknown hidden dense forests and dark caves are the best examples of such dangerous locations.
- Explore dense forests: A sprawling maya network in the guatemala jungle has been discovered with the help of Aerial LiDAR, which mapped the dense forest and discovered subtle sign of the archaeological wonder.

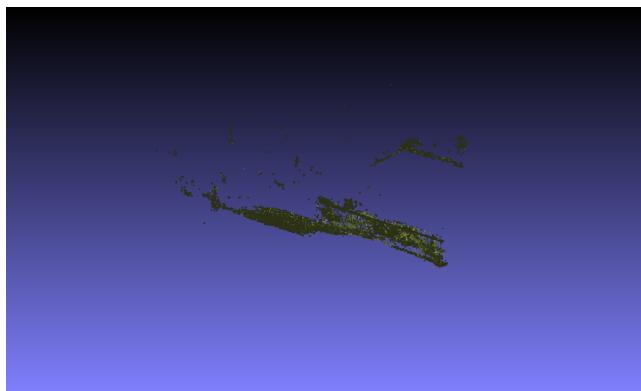
Point Cloud registration [Paper](#)

Definition: Point Cloud Registration is a fundamental problem in 3D computer vision and photogrammetry. Given several sets of points in different coordinate systems, the aim of registration is to find the transformation that best aligns all of them into a common coordinate system. Point Cloud Registration plays a significant role in many vision applications such as 3D model reconstruction, cultural heritage management, landslide monitoring and solar energy analysis.

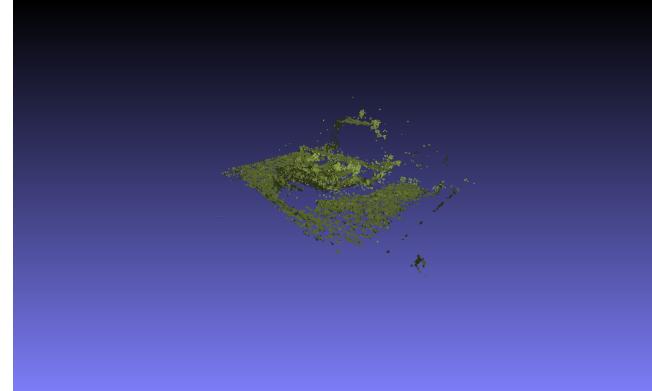
Generally 2 steps are needed to register 2 given point clouds of the same object (fully or partially)

- Global alignment (3D Feature Descriptors)
 - The 2 models are "roughly aligned"
- Local alignment (ICP)
 - Starting from a "rough" initial alignment, find the exact precise alignment

Before Registration



After Registration



Local Alignment [Jupyter-Notebook](#)

Having two scans (point clouds) of the same object $P=\{p_i\}$ and $Q=\{q_i\}$ we want to find a transformation (rotation R and translation t) to apply to P to match Q as good as possible (alignment).

This leads to the following two questions:

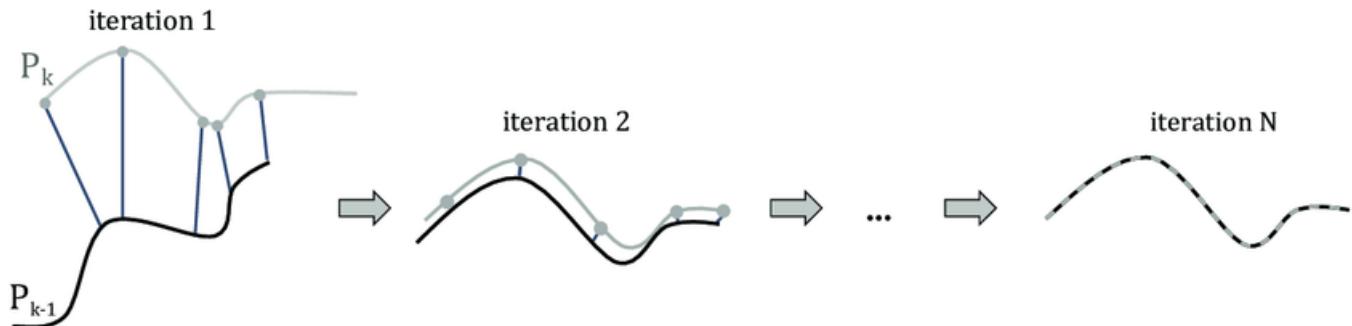
- What is the difference of theirs pose? ... or equivalently..
- What is the transformation that can fully align them?

Checkout the linked jupyter notebook to see the ICP algorithm hands on

Iterative Closest Point (ICP)

1. For each object point p in model 1, find the nearest point q in model 2
2. Use all pairs (p,q) to estimate the transformation from model 1 to model 2
3. Apply the transformation to the points of model 1
4. Repeat steps 1-3 until convergence/stop criterion is met

Visual representation of the IPC algorithm



Elaborating the bullet points 1 and 2 in greater detail

For each object point p in model 1, find the nearest point q in model 2: How to compute all possible pairs for proximity?

Simple k-nearest neighbor approach, is being used to find correspondences (similar points) in the two point clouds. However, this approach can be computationally heavy and slow. A better and more suitable approach is K-d trees.

Use all pairs (p,q) to estimate the transformation from model 1 to model 2: How to estimate the transformation from model 1 to model 2, given pairs (p,q)

Kabsch Algorithm / Procrustes Analysis:

- Translate the centroids of both models to the origin of the coordinate system (0,0,0)
 - Compute centroids c_p, c_q of both models
 - Subtract from each point coordinates the coordinates of its corresponding centroid:
 - $p' = p - c_p$ and $q' = q - c_q$
- Compute Covariance Matrix: $C_{pq} = \text{Sum}(p' * q'^T)$
- Compute the optimal rotation
 - Calculate the singular value decomposition (SVD) of the covariance matrix: $C_{pq} = USV'$
 - Calculate rotation matrix: $R = UV^T$
- Compute the optimal translation: $T = c_q - R c_p$

Why does the Kabsch Algorithm needs to be applied at every iteration?

The reason for the Kabsch Algorithm being applied at every iteration is that our proximity approach is not able to initially find all correspondences for each (p,q) pair. Thus, the algorithm needs to revalue its estimation based on the new values it's provided with, until it converges or a stop criterion is met.

Global Alignment

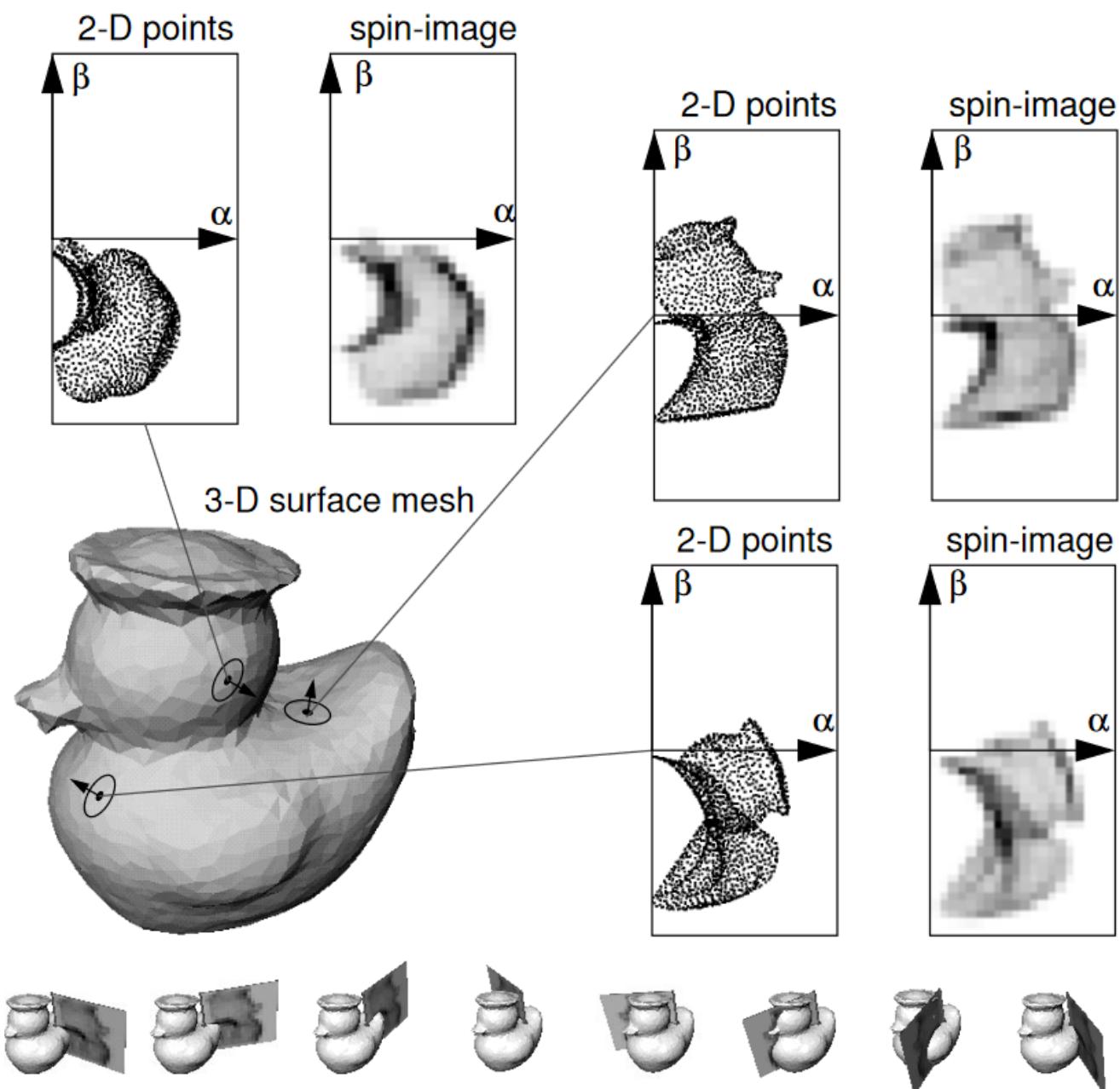
Consider 2 models (point clouds) of the same object (fully or partially)

- How can we "roughly" align them?
 - We cannot use ICP, if the initial poses are too different
 - Can we use some kind of (3D) "features" and match them?

Spin Images: Paper

- "Spin" a discretized 2D grid around the surface normal of a point.
- Accumulate neighboring pixels in the grid bins, while spinning.
- The descriptor is the 2D grid (image) where each element (pixel) contains the number of accumulated points.

Visualization:



Spin-images from points on one surface are compared by computing correlation coefficient with spin-images from points on another surface; when two spin-images are highly correlated, a point correspondence between the surfaces is established. More specifically, before matching, all of the spin-images from one surface (the model) are constructed and stored in a spin-image stack. Next, a vertex is selected at random from the other

surface (the scene) and its spin-image is computed. Point correspondences are then established between the selected point and the points with best matching spin-images on the other surface. This procedure is repeated for many points resulting in a sizeable set of point correspondences (~100). Point correspondences are then grouped and outliers are eliminated using geometric consistency. Groups of geometrically consistent correspondences are then used to calculate rigid transformations that aligns one surface with the other. After alignment, surface matches are verified using a modified iterative closest point algorithm. The best match is selected as the one with the greatest overlap between surfaces.

[PHP \(Point Feature Histograms\) Tutorial](#)

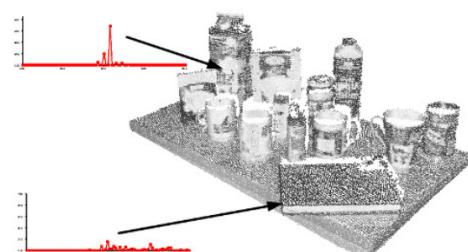
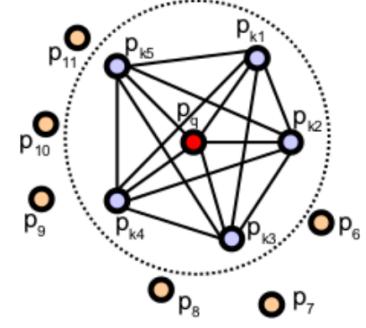
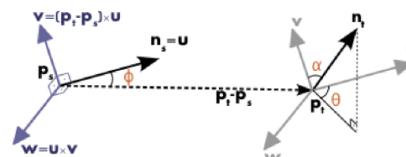
Global Alignment – 3D Feature Descriptors

- PFH (Point Feature Histograms)

$$\alpha = \arccos(v \cdot n_t)$$

$$\phi = \arccos\left(u \cdot \frac{(p_t - p_s)}{\|p_t - p_s\|_2}\right)$$

$$\theta = \arctan(w \cdot n_t, u \cdot n_t)$$



[FPFH \(Fast Point Feature Histograms\) Tutorial](#)

Global Alignment – 3D Feature Descriptors

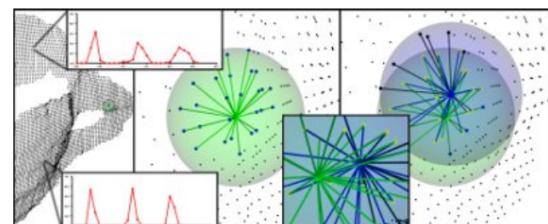
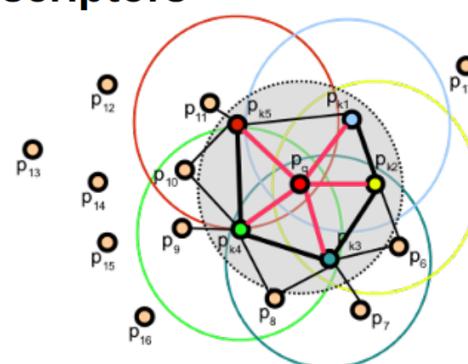
- FPFH (Fast Point Feature Histograms)

- Simplification/Approximation of the PFH formulation
 - reduces the computational complexity
 - retains most of the discriminative power of PFH.

- Algorithm:

- Find all oriented points in a spherical neighborhood of radius r around each point (k-d tree)
- Compute relative angles using surface normals and direction vector from the source point to each neighbor

- The descriptor is a multi-dimensional histogram



[RANSAC in 3D Youtube](#)

Matching 3D features generally results in many false-matches

- Big additional computational cost, brings only small increase in the matching accuracy
- Need for an algorithm that is robust to outliers

RANSAC to the rescue

We can use:

- Model 1 (point cloud with normals)
- Model 2 (point cloud with normals)
- Feature matches (containing many outliers)

In order to:

- Estimate the "rough" pose difference between the 2 models

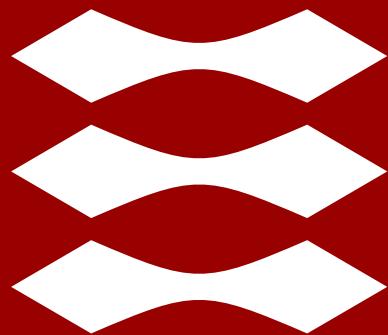
RANSAC for Pose Estimation

- Randomly choose 3 pairs of matched points
- Estimate the relative pose
 - Kabsch/ Procrustes
- Apply the transformation and assess its "validity":
 - number of inliers: matched point pairs whose distance became "small" after applying the specific transformation
 - Sum of distances between all matched point pairs after transformation
 - ...
- Keep the transformation with most inliers
- Repeat random sampling

Pros:

1. RANSAC often produces better results, even in noiseless cases
2. RANSAC does not require a good initial estimate of the transform between the two data sets
3. RANSAC can be used with data sets that do not have as many local features

DTU



Lazaros Nalpantidis

3D Point Cloud Processing - Clustering & Regression

Outline

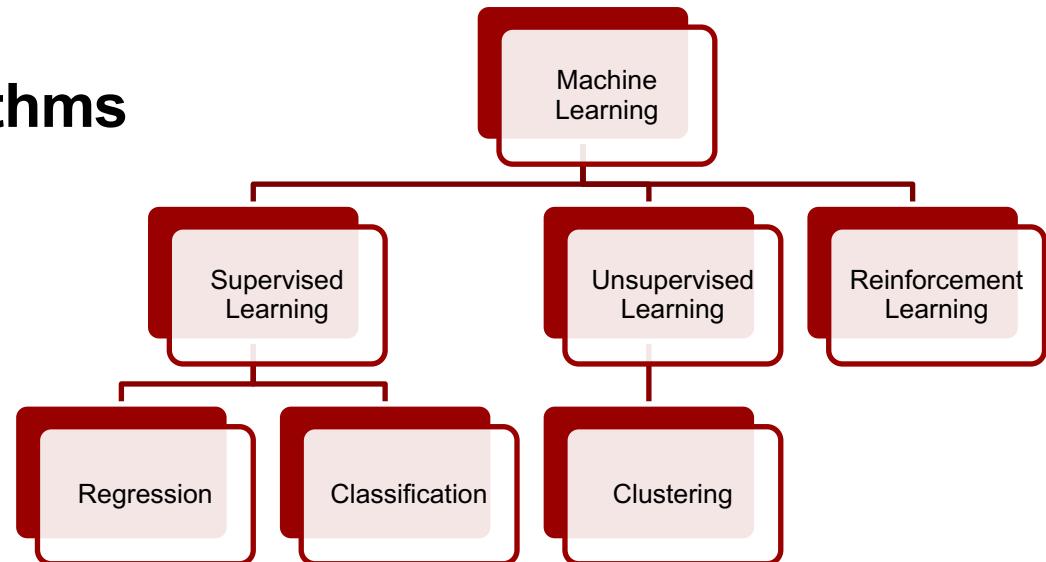
- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

Outline

- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

What are Clustering and Regression?

A taxonomy of ML algorithms

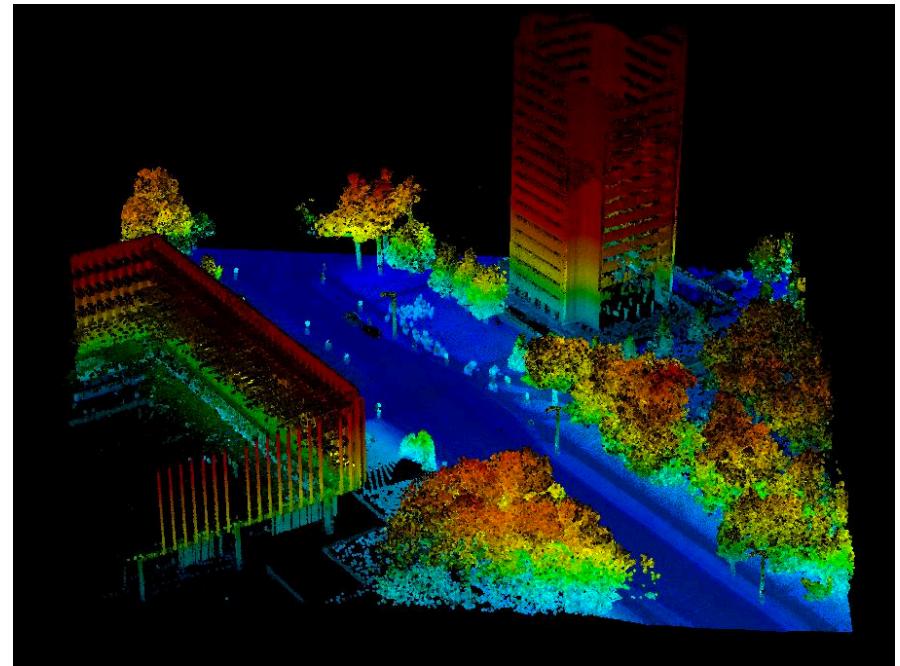


- **Supervised Learning** – The target values are known
 - Regression – The target value is numeric
 - Classification – The target value is nominal
- **Unsupervised Learning** – The target values are unknown
 - Clustering – Group together similar instances
- **Reinforcement Learning** – Interacting with a dynamic environment the system must perform a certain goal.

- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

Why is ML important for Perception of Autonomous Systems?

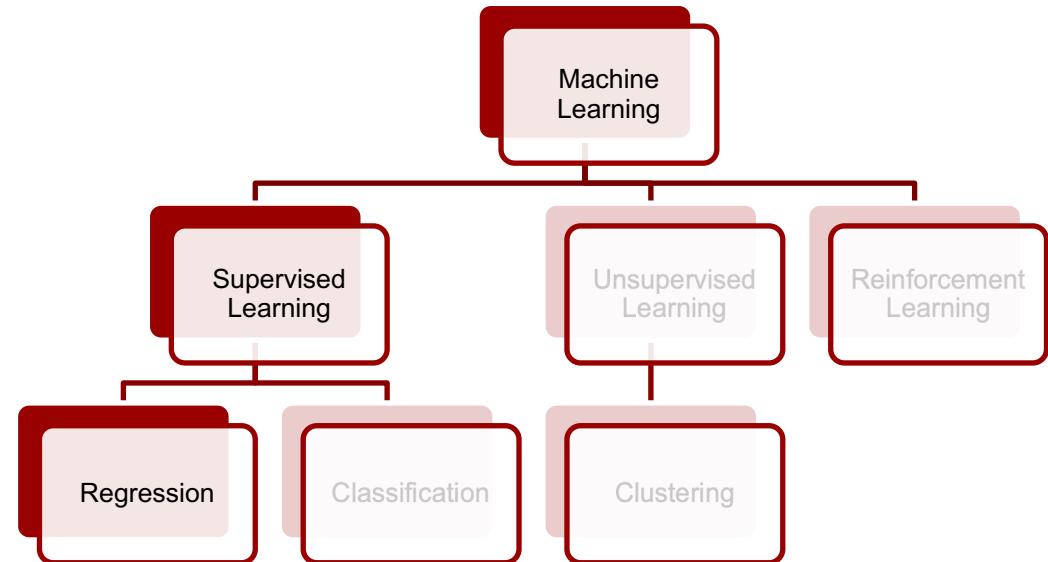
- Create initial object hypotheses
 - input to pose estimation?
- Abstract representation of scene
- ...what else?



Outline

- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

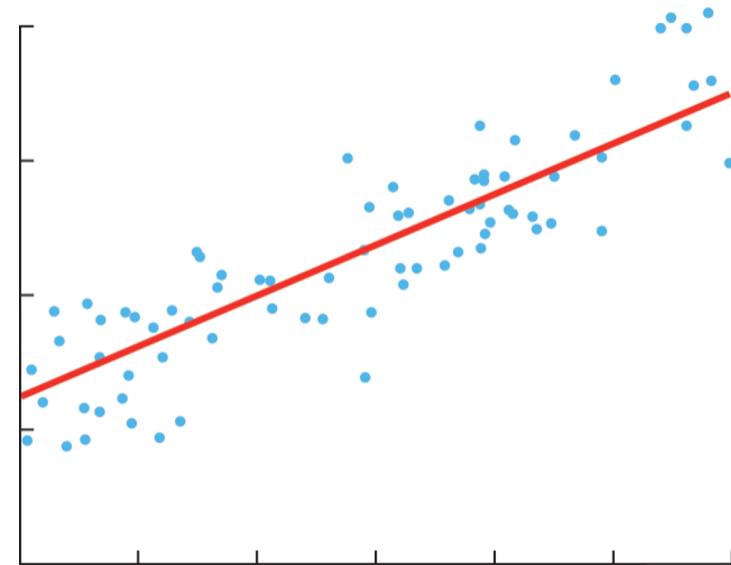
Regression



- **Supervised Learning** – The target values are known
 - Regression – The target value is numeric

Regression

- Regression tries to assign correct significance (weights) to input variables and formulate a weighted linear combination of them that can predict the output variables.
- The number of input and output variables can vary depending on the specific problem.
- The final output of the algorithm is a regression line



Regression

- The goal is to find an equation $f()$ that models the relationship between input x and output y such as:

$$y = f(x) + \varepsilon$$

- Regression Models:
 - Linear Regression: Assumes that function f is linear.
 - Locally Weighted Regression: Creates multiple linear models on small neighbor data.

Regression

- The performance for the linear regression method is evaluated by an error function.
- The most used is the Mean Squared Error (MSE) :

$$MSE = \frac{1}{N} \sum_{n=1}^N (t - y)^2$$

...where t is the true value of the learned output and y is the predicted value.

Regression

- **Linear Regression**
 - Linear Regression assumes that $f(x)$ is a linear combination between the inputs x and a set of regression coefficients b :
$$y = f(b, x) = b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n + c = b^T x + c$$
 - The goal of linear regression is to find regression coefficients b that minimize the squared error between the true values of the output and the predicted values.

- **Polynomial Regression**

- In the case that the mapping between the inputs x and the output is not expressed well by a straight line we can use polynomial regression.
- This is done by adding dimensions to the input data, where n is the degree of polynomial.

$$X = [x, x^2, x^3, \dots, x^n]$$

- The degree of polynomial can be found by cross-validation.

- **Polynomial Regression**

- In the case that the mapping between the inputs x and the output is not expressed well by a straight line we can use polynomial regression.
- This is done by adding dimensions to the input data, where n is the degree of polynomial.

$$X = [x, x^2, x^3, \dots, x^n]$$

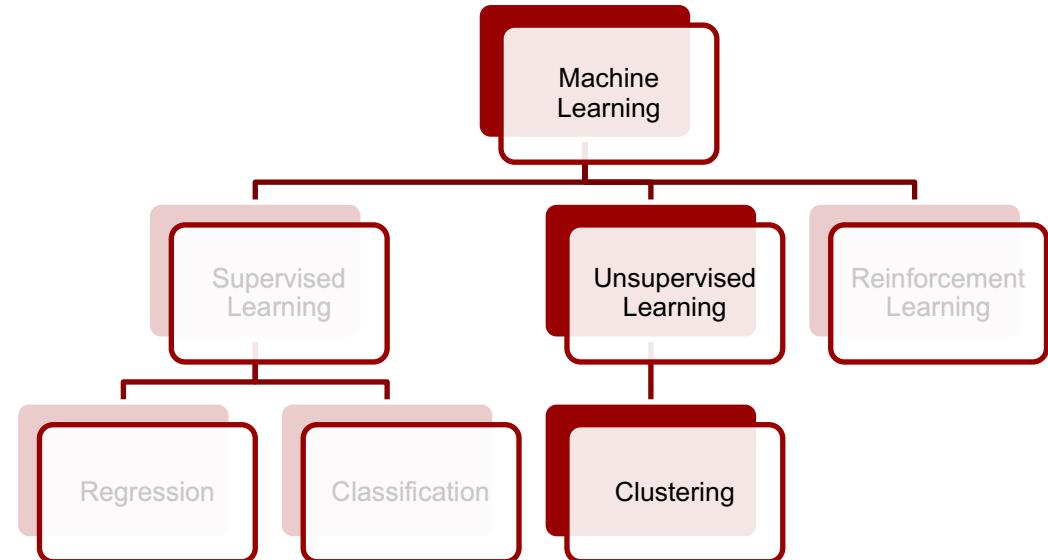
- The degree of polynomial can be found by cross-validation.

Regression

- Can we use Regression on our 3D point clouds?
- What problems could we tackle?
 - Line fitting
 - Plane fitting
 - Fitting other (non-linear) surfaces
 - ...what else?

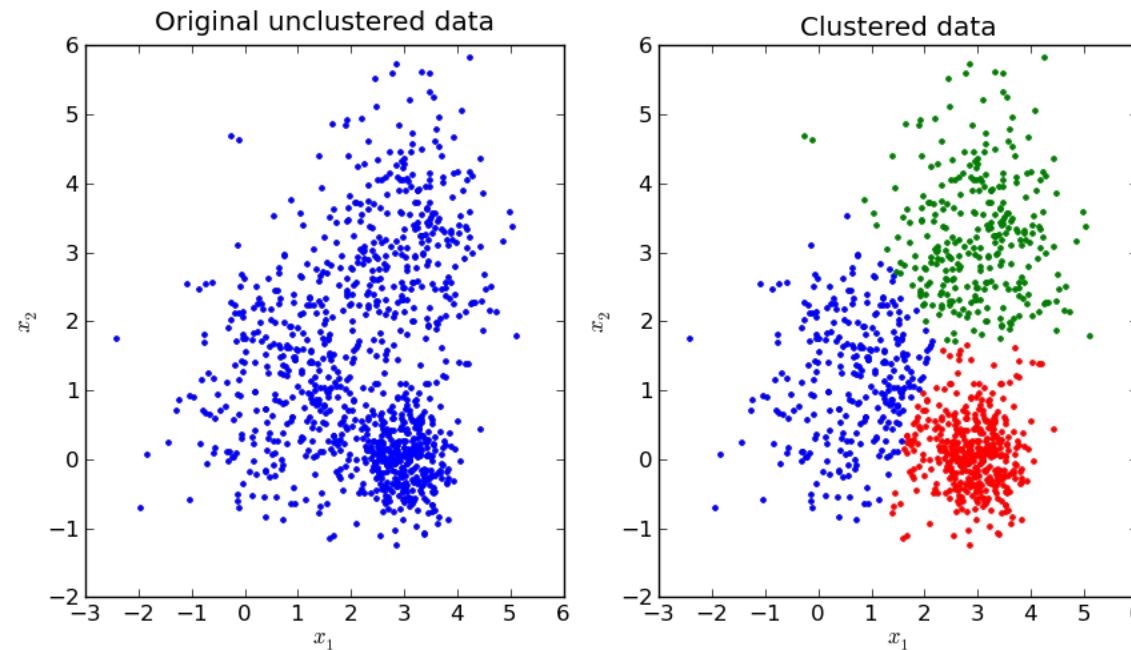
- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

Clustering



- **Unsupervised Learning** – The target values are unknown
 - Clustering – Group together similar instances

- Partitions a point cloud into groups of (similar) points, i.e. clusters,
- Finds the underlying structure of the point cloud.



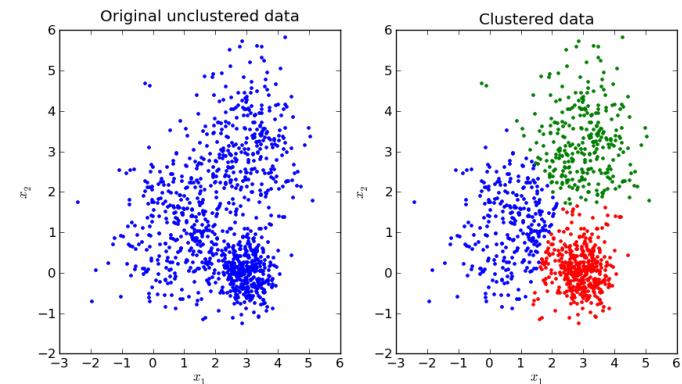
Clustering – k-means

- Partitions a point cloud into k clusters, such that each point belongs to one cluster
- Assigns point to clusters, so as to minimize the Sum of Squared Euclidean Distances (Distortion) between points and their assigned cluster centers.

$$J = \sum_k \sum_{n \in k} d(x_n, \mu_k)^2$$

...where x is a vector representing the n^{th} data point assigned to cluster k and μ_k is the centroid the cluster k .
The function $d()$ is the Euclidean distance between x and μ .

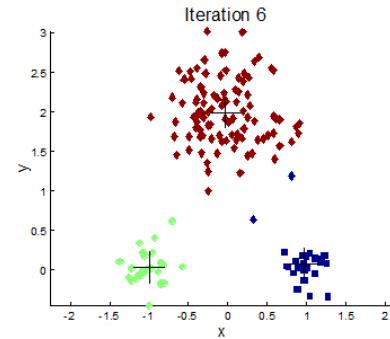
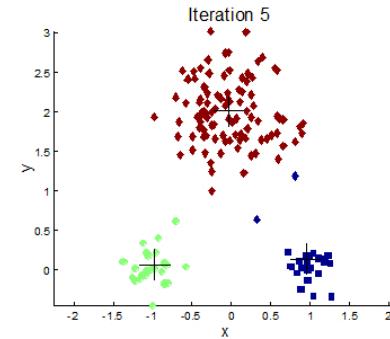
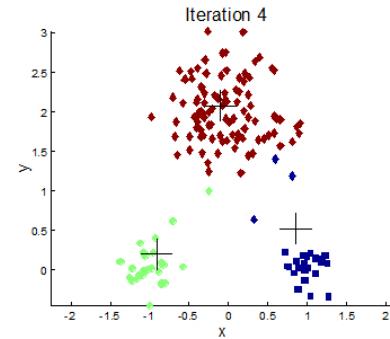
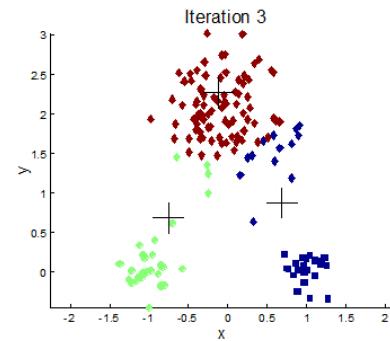
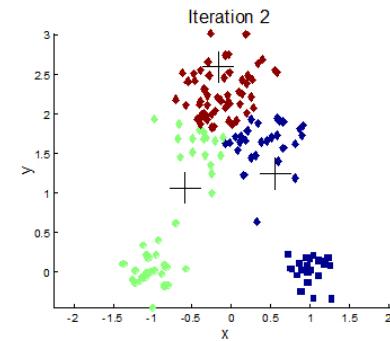
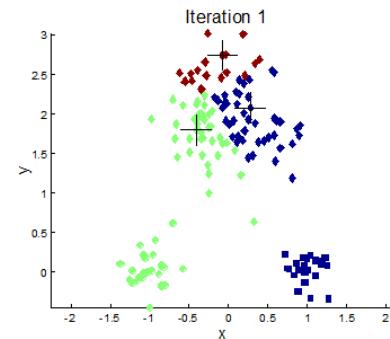
- **Important:** K-means requires known number of clusters k



Clustering – k-means

It is an iterative process:

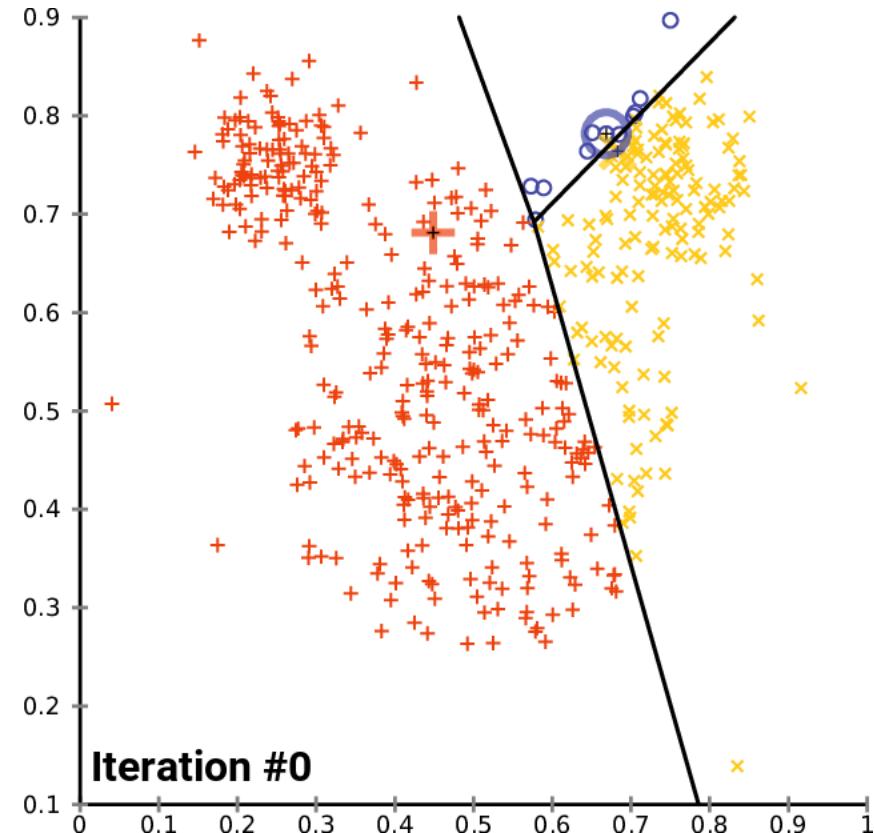
- Step 1:
 - Randomly initialize k cluster centers
- Step 2:
 - Assign each to its nearest cluster center
- Step 3:
 - Re-compute each cluster-center as the centroid of all points assigned to each cluster
- Step 4:
 - Check convergence/stop criterion, otherwise repeat Steps 2-4



Clustering – k-means

It is an iterative process:

- Step 1:
 - Randomly initialize k cluster centers
- Step 2:
 - Assign each to its nearest cluster center
- Step 3:
 - Re-compute each cluster-center as the centroid of all points assigned to each cluster
- Step 4:
 - Check convergence/stop criterion, otherwise repeat Steps 2-4

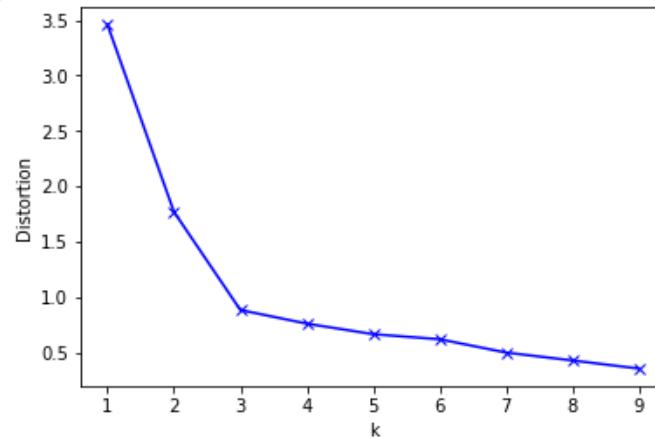


Clustering – k-means

How to define k ?

– Elbow method

- Run k-means for several k and calculate distortion for each k
 - » Distortion: sum of squared distances of each point to the center of the closest cluster
- Plot distortion vs k
- Look for k where the curve stops decreasing rapidly

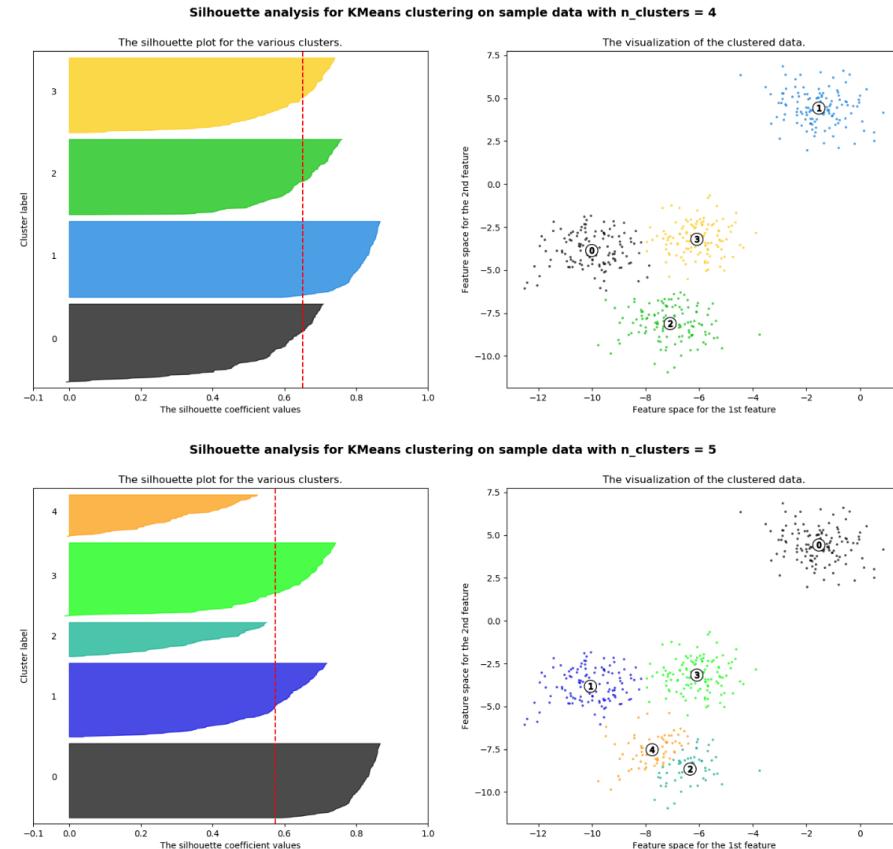


Clustering – k-means

How to define k ?

– Silhouette Analysis

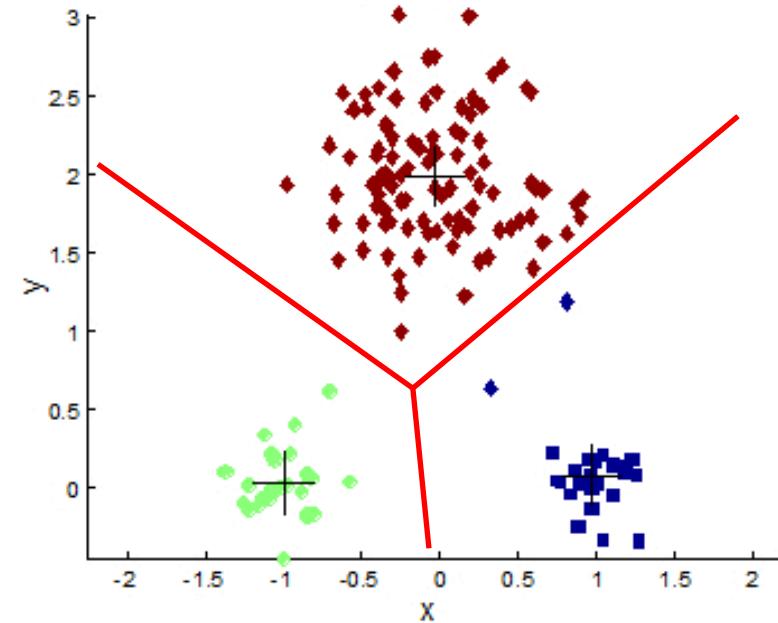
- Calculate Silhouettes for various k
 - Silhouette coefficient shows how far each sample is from other cluster centers
 - » +1: far from others
 - » 0: at the boundary
 - » -1: sample is on average closer to the points of another cluster
- Thickness shows cluster size (number of points assigned to cluster)
- Avoid k that leads to:
 - Scores for clusters < average
 - Individual scores < 0
 - Big differences in cluster sizes (thickness)



Clustering – k-means

Issues

- Need to define k
- Final cluster assignments depend on initialization
 - » Cluster assignments may be different on different runs
 - » K-means may not achieve the global optimum
- Can describe only convex cluster geometries
- Computationally demanding as the number of points and dimensions increase



3D Point Cloud Segmentation by k-means

Which space to apply Clustering? Is it your point cloud's 3D space?

What should be the dimensionality?

- dimensionality = 3 ? → X, Y, Z
- dimensionality = 6 ? → X, Y, Z, R, G, B
- dimensionality = 6 ? → X, Y, Z, L, U, V
- dimensionality = 6 ? → X, Y, Z, H, S, V
- dimensionality > 6 ? → X, Y, Z, H, S, V, gradients, texture, ... ??

Clustering – Mean Shift

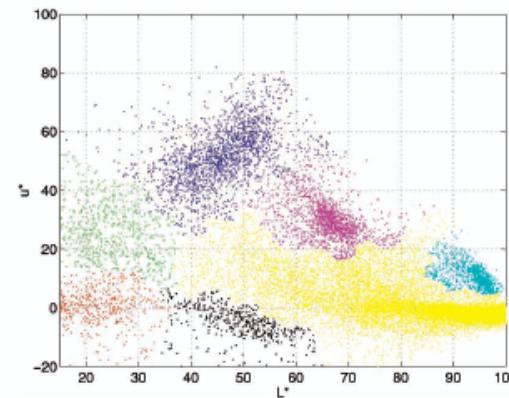
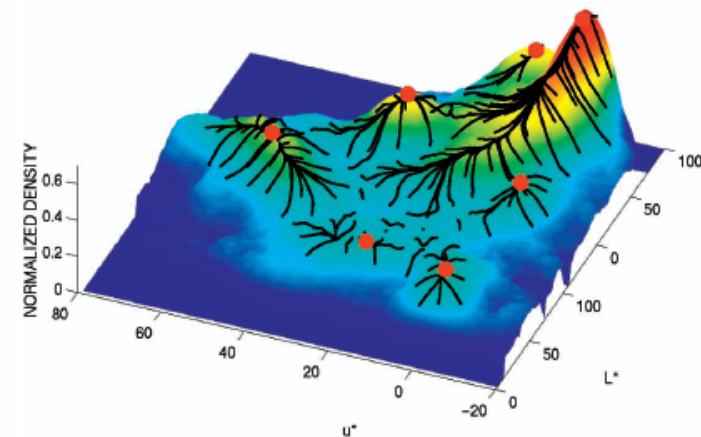
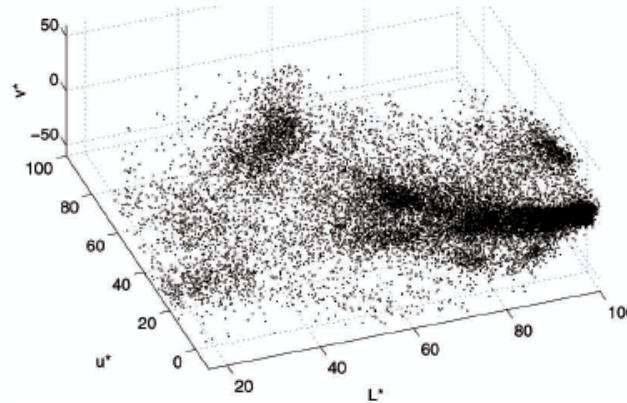
- **Mean Shift** is an algorithm that finds the maxima—the modes—of a density function given discrete data sampled from that function. Thus, it is a density hill climbing algorithm.
 - Every point gives rise to a cluster.
 - Each cluster is defined by the radius, a.k.a. “bandwidth”, h of its region, and a kernel function K that is used to calculate the contribution of the points included within this radius.
 - In the end, only unique clusters are considered.
- So, we do not need to set the number of clusters!
 - However, we need to define the bandwidth (and the kernel function).

Dorin Comaniciu and Peter Meer, “Mean Shift: A robust approach toward feature space analysis,” IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. pp. 603-619.

Yizong Cheng, “Mean shift, mode seeking, and clustering,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, pp. 790–799, Aug 1995.

Mean shift algorithm

- Try to find *modes* of this non-parametric density



can be as fast as K-means
although heavier by parallelising

Kernel density estimation

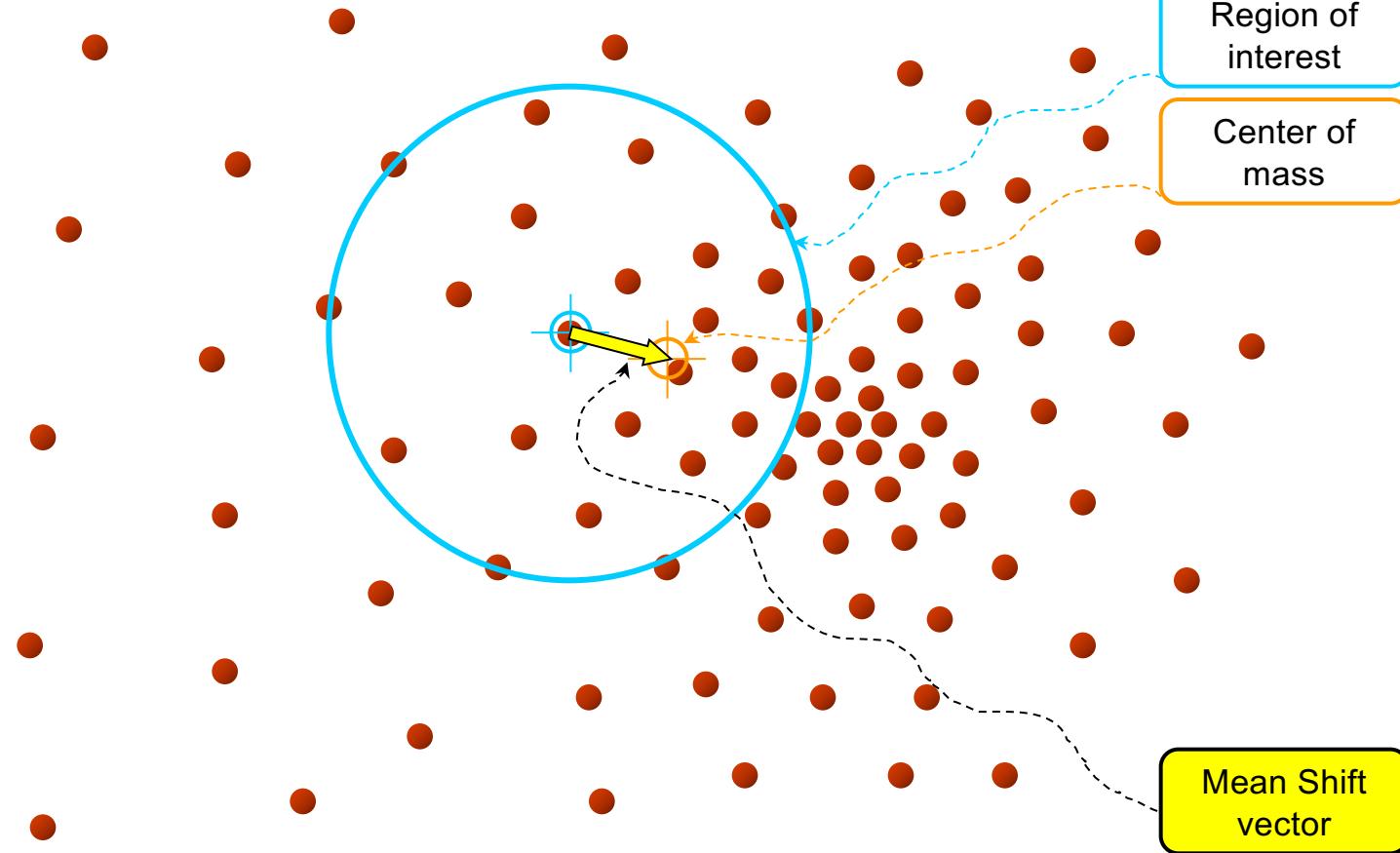
Kernel density estimation function

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$$

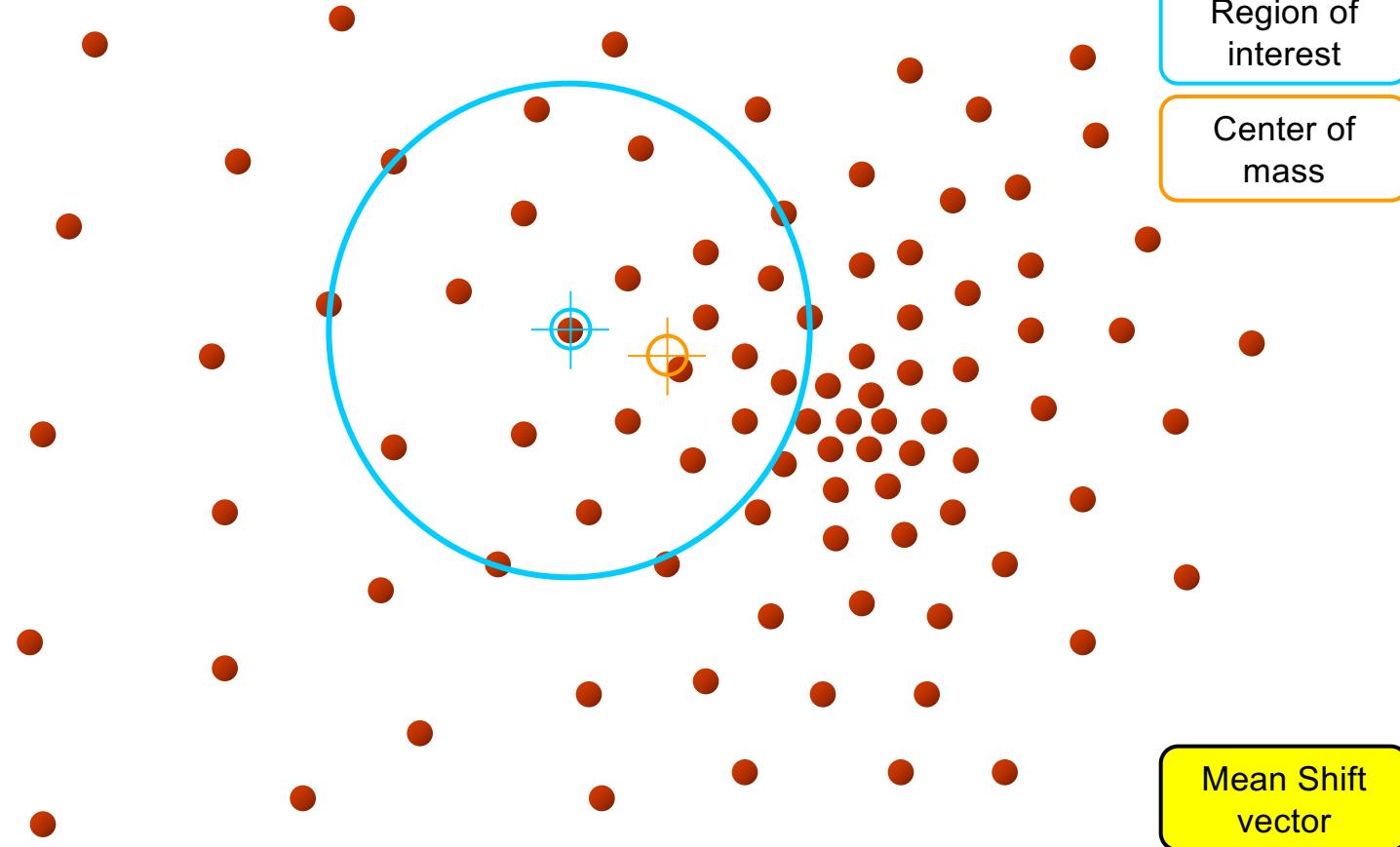
Gaussian kernel (typically used)

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}.$$

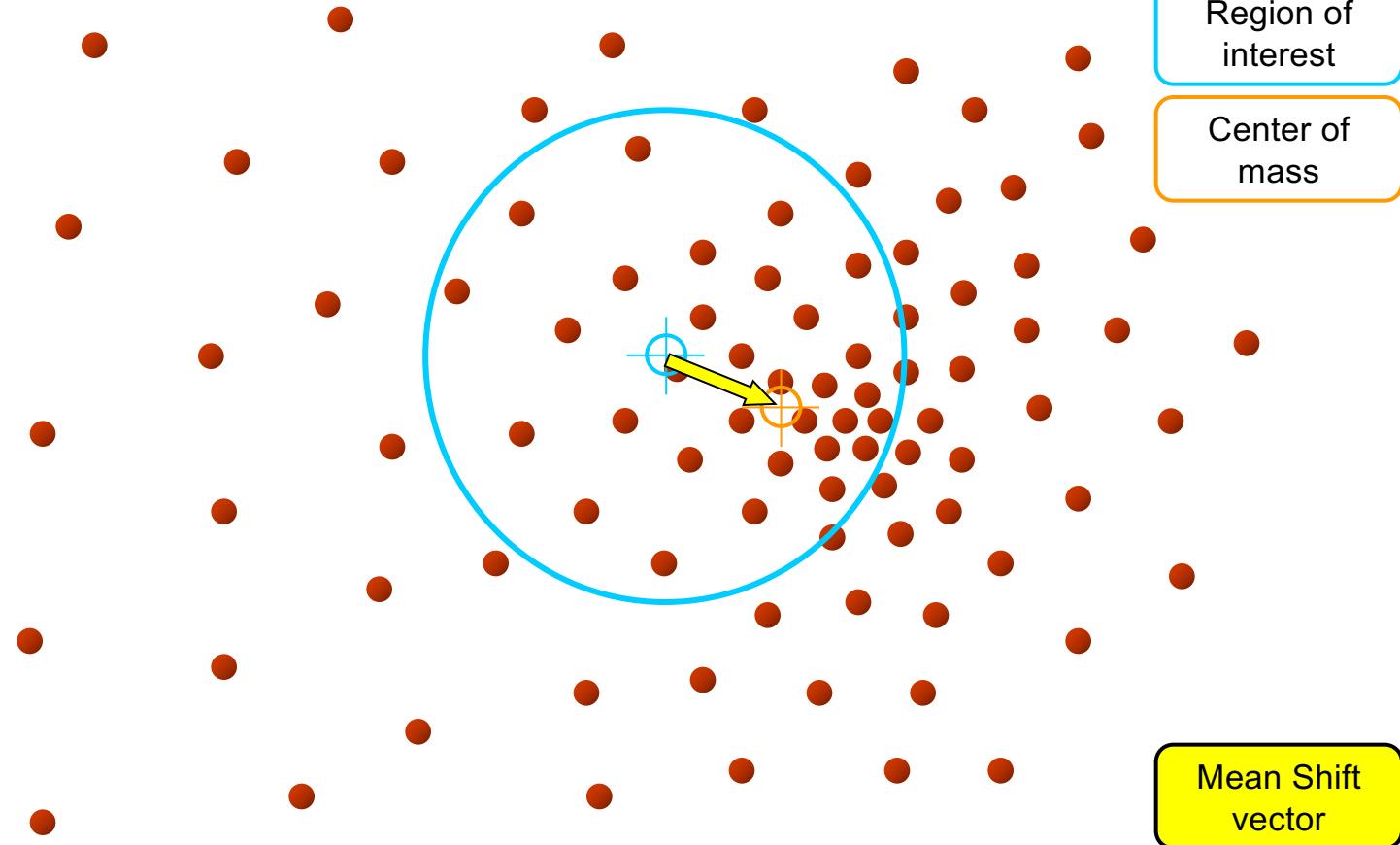
Mean shift



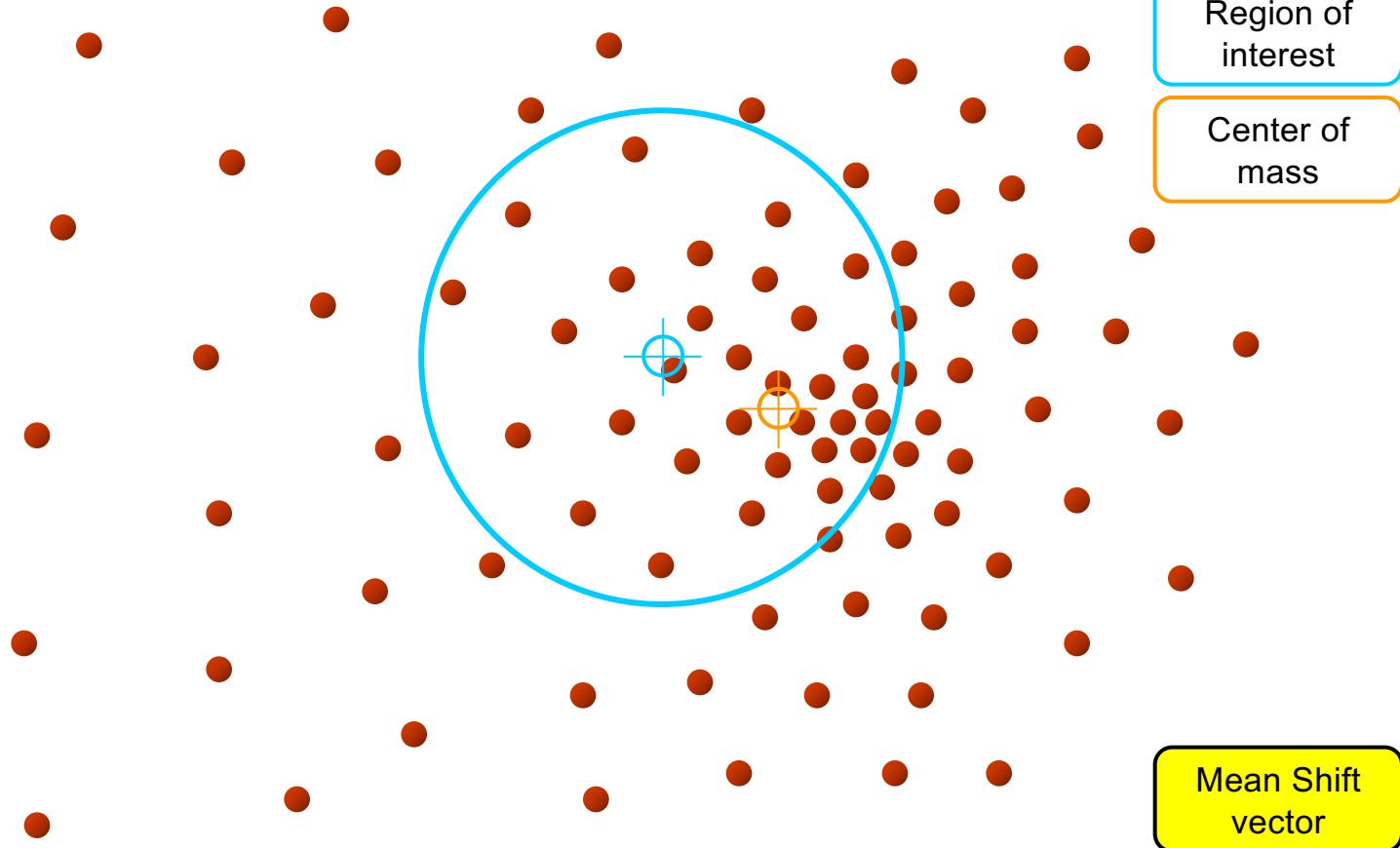
Mean shift



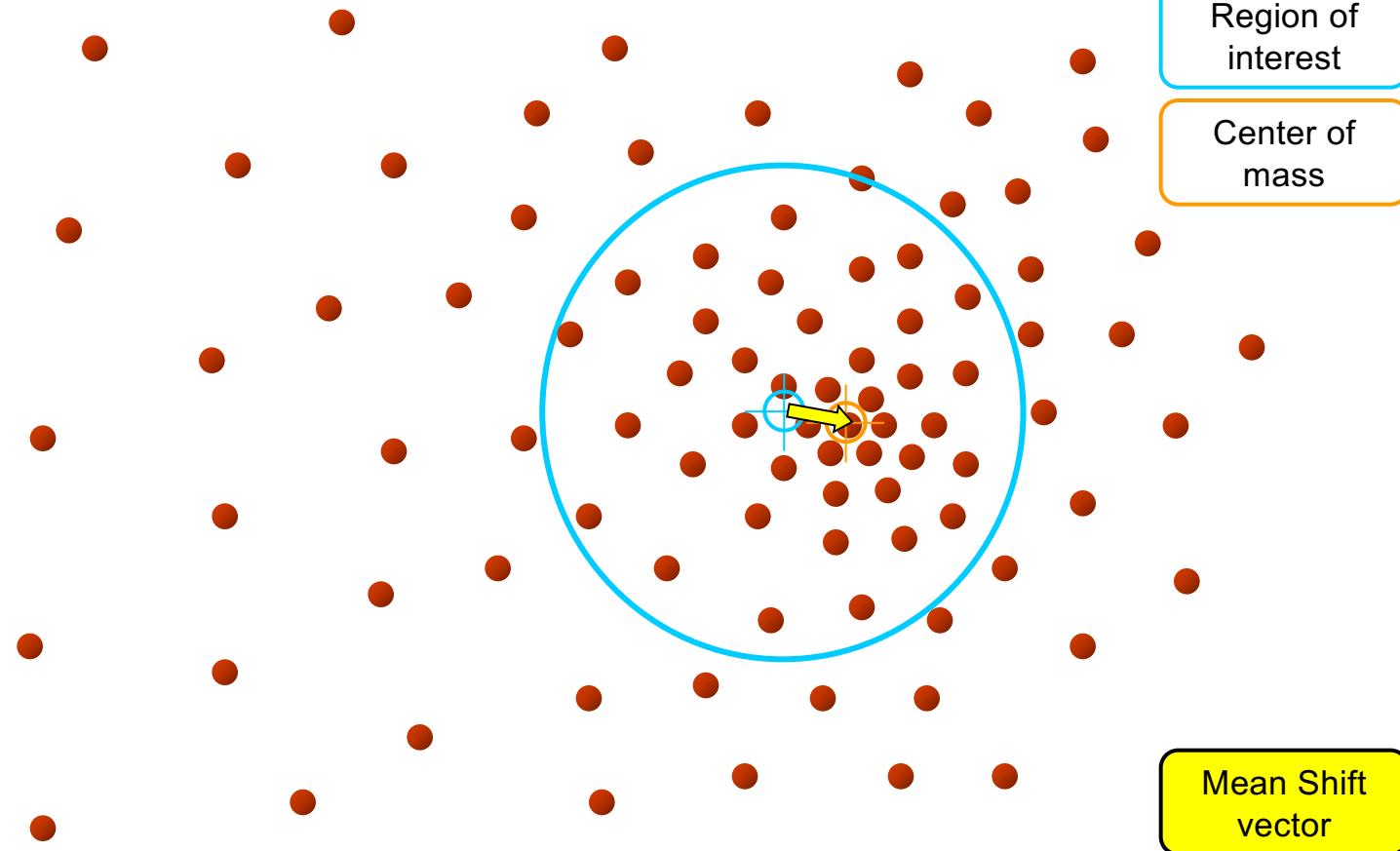
Mean shift



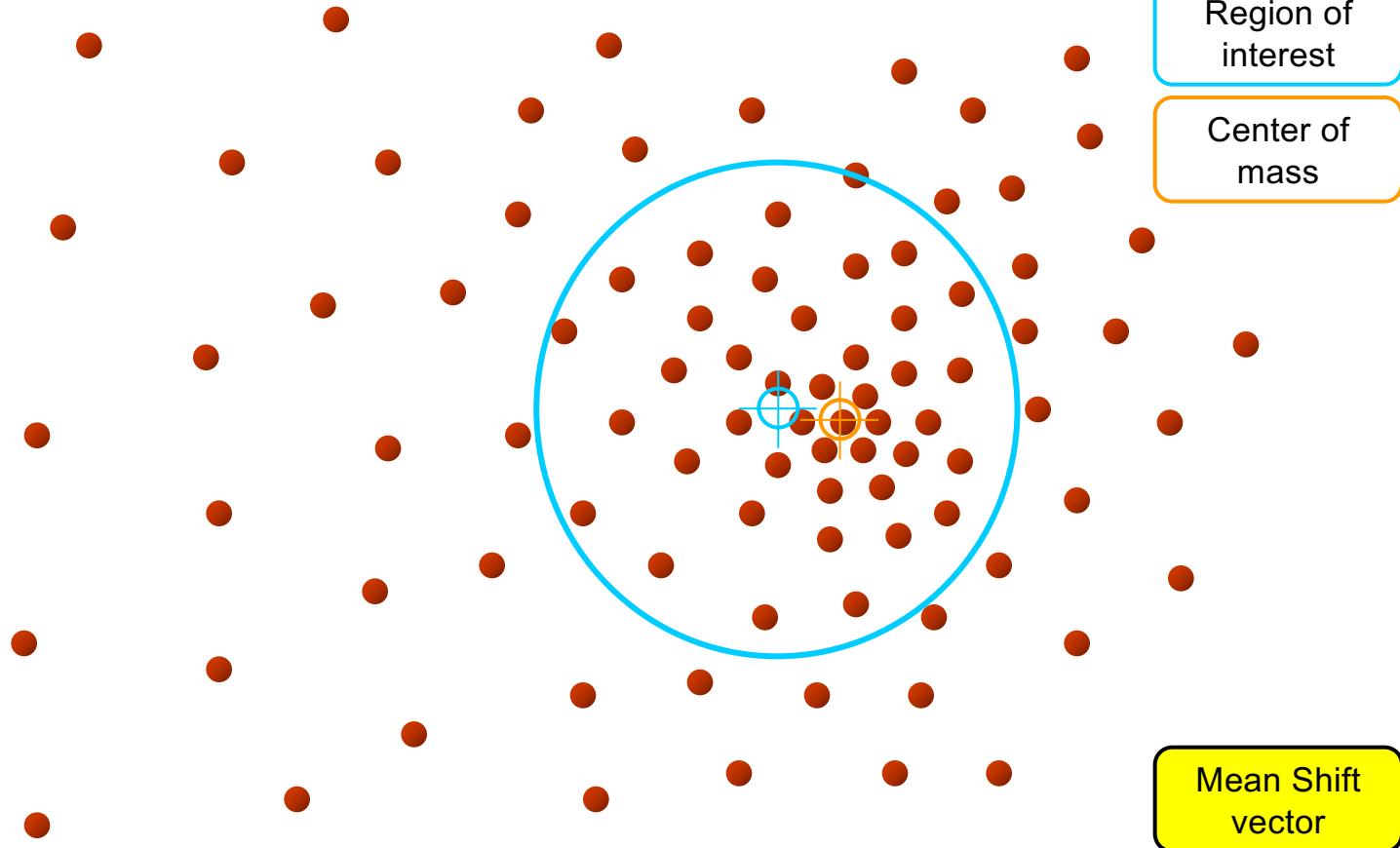
Mean shift



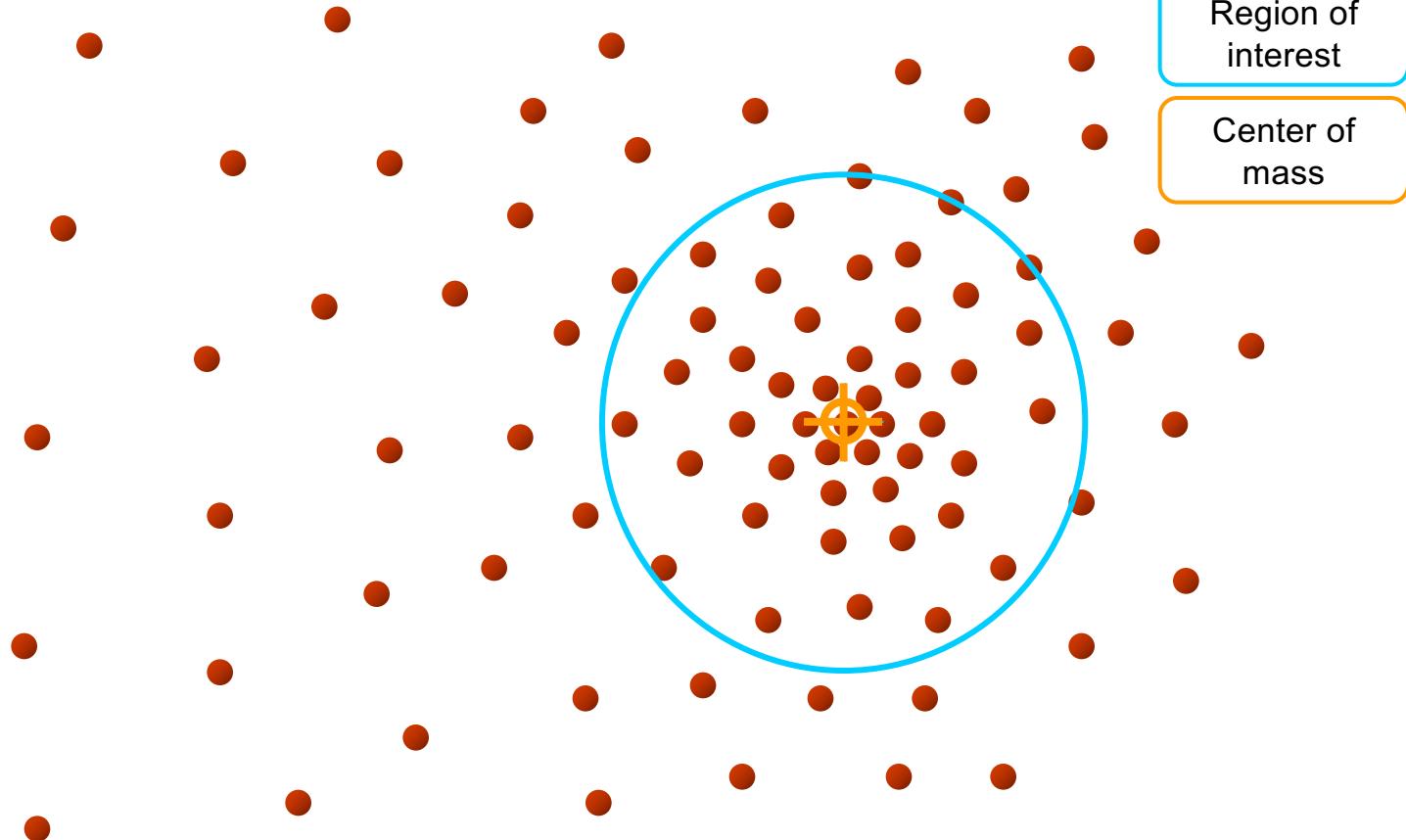
Mean shift



Mean shift

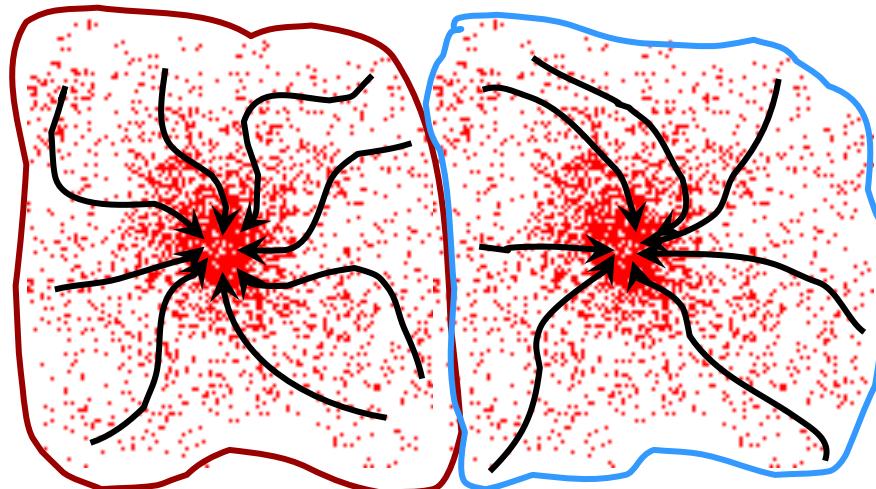


Mean shift

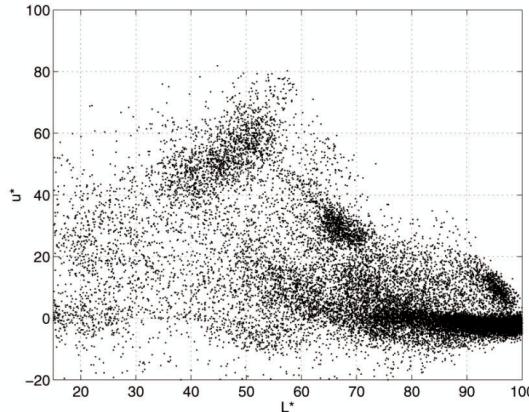


Attraction basin

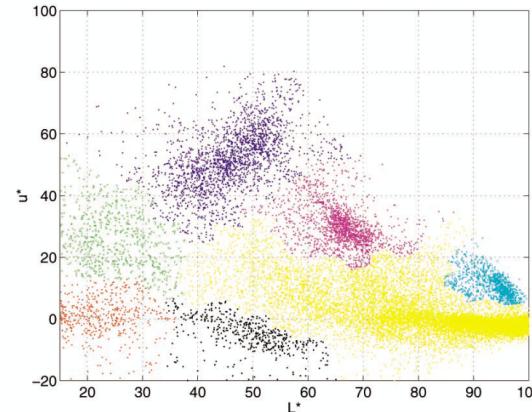
- Attraction basin: the region for which all trajectories lead to the same mode
- Cluster: all data points in the attraction basin of a mode



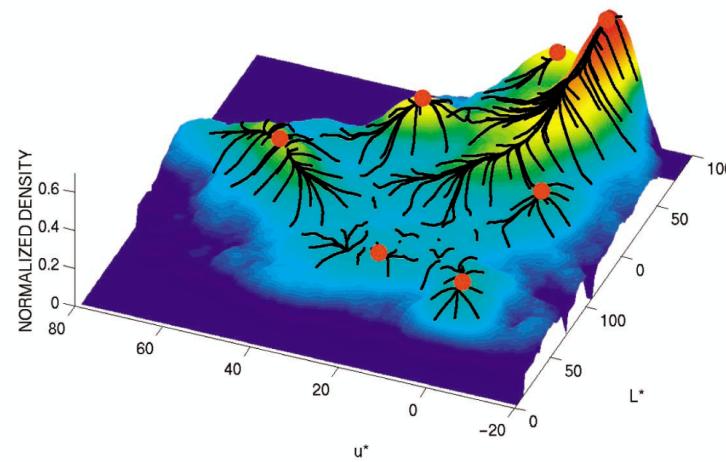
Attraction basin



(a)



(b)



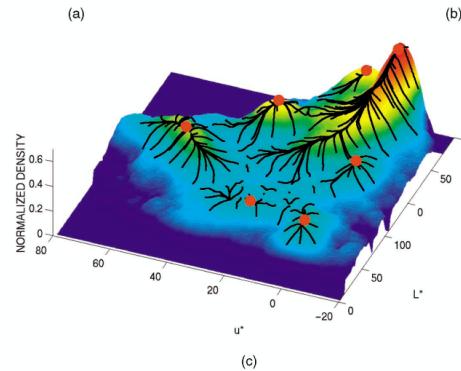
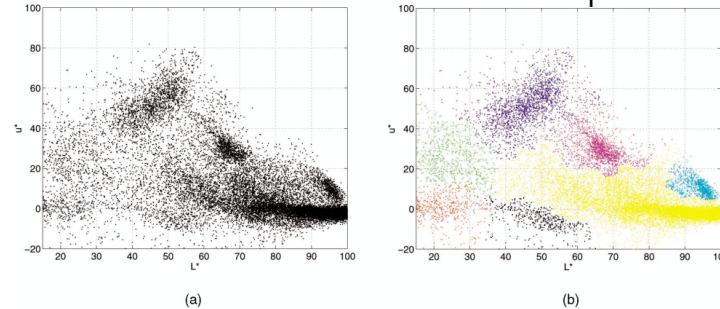
Mean shift clustering

The mean shift algorithm seeks *modes* of the given set of points

1. Choose kernel and bandwidth
2. For each point:
 - a) Center a window on that point
 - b) Compute the mean of the data in the search window
 - c) Center the search window at the new mean location
 - d) Repeat (b,c) until convergence
3. Assign points that lead to nearby modes to the same cluster

2D Image Segmentation by Mean Shift

- Compute features for each pixel (color, gradients, texture, etc)
- Set kernel size for features K_f and position K_s
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- Merge windows that are within width of K_f and K_s



3D Point Cloud Segmentation by Mean Shift

Which space to apply Clustering? Is it your point cloud's 3D space?

What should be the dimensionality?

- dimensionality = 3 ? → X, Y, Z
- dimensionality = 6 ? → X, Y, Z, R, G, B
- dimensionality = 6 ? → X, Y, Z, L, U, V
- dimensionality = 6 ? → X, Y, Z, H, S, V
- dimensionality > 6 ? → X, Y, Z, H, S, V, gradients, texture, ... ??

Mean shift segmentation results



Clustering - DBSCAN

- DBSCAN is a density-based clustering algorithm
- In density-based clustering:
 - we partition points into dense regions separated by not-so-dense regions.
 - a cluster is defined as a maximal set of density-connected points
 - we can discover clusters of arbitrary shape
- Density definition:
 - Density at point p is defined as the number of points within a circle/sphere of radius ε
 - A region is dense if the circle/sphere of radius ε contains at least $MinPts$ points

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in International Conference on Knowledge Discovery and Data Mining, 1996.

Clustering - DBSCAN

Types of points:

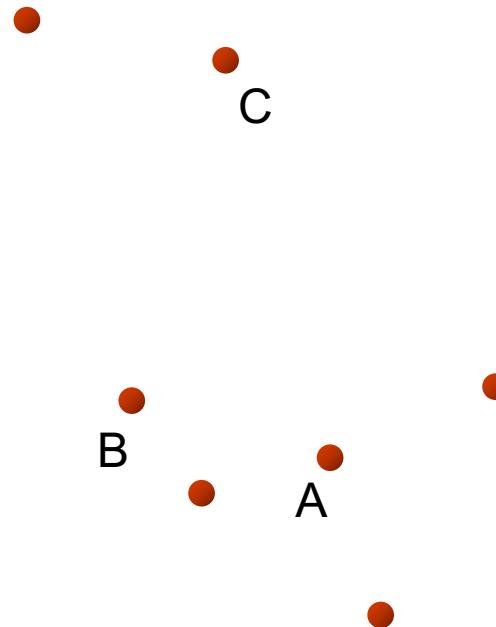
- A **core point** has more than a specified number of points ($MinPts$) within ε
- A **border point** has fewer than $MinPts$ within ε , but is in the neighborhood of a core point.
- A **noise point** is any point that is not a core point or a border point.

Clustering - DBSCAN

Assume:

$$\varepsilon=1$$

$$MinPts=4$$



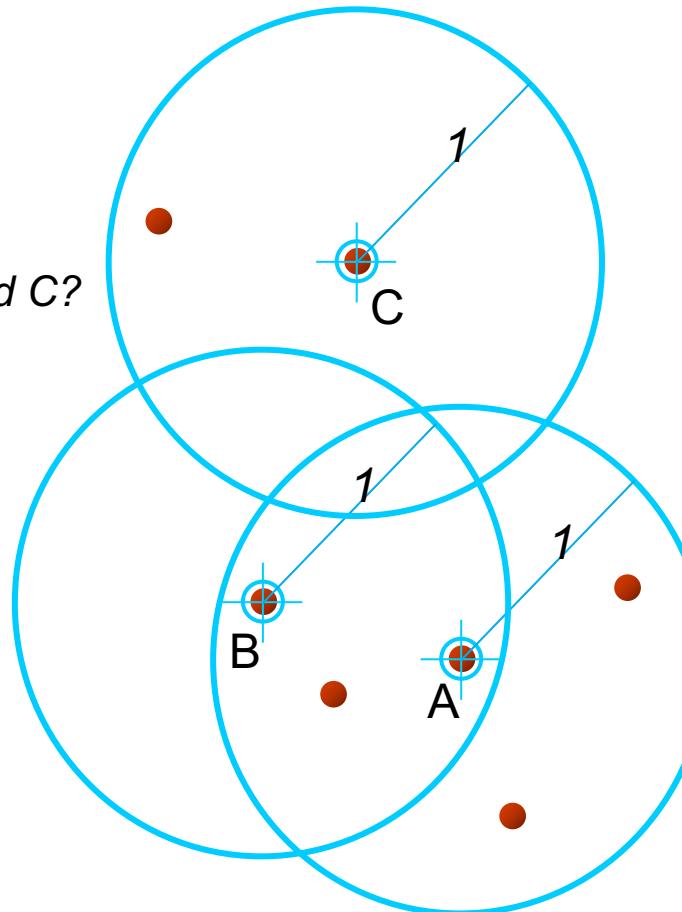
Clustering - DBSCAN

Assume:

$$\varepsilon=1$$

$$\text{MinPts}=4$$

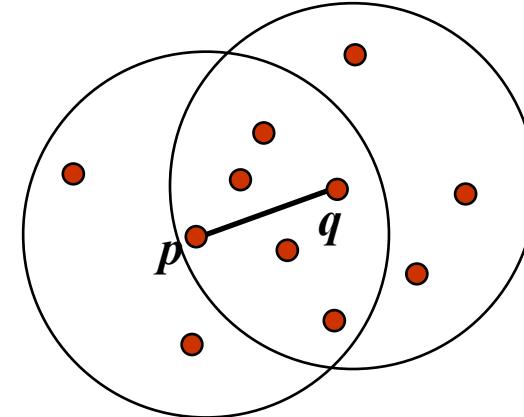
What is the type of points A, B and C?



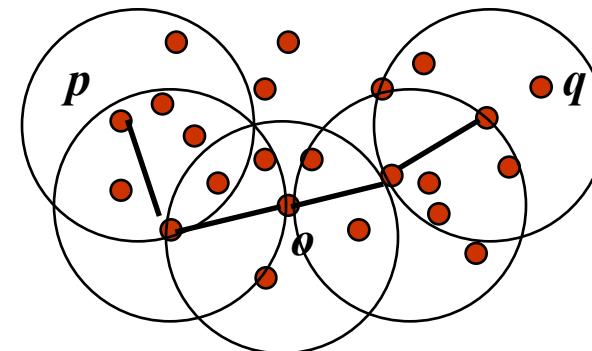
Clustering - DBSCAN

Some more definitions:

- **Density edge**
 - We place an edge between two core points q and p if they are within distance ε



- **Density-connected**
 - A point p is density-connected to a point q if there is a path of edges from p to q



Clustering - DBSCAN

DBSCAN algorithm

1. Label points as core, border and noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster
 - Create a new cluster with the point p and all the points that are density-connected to p.
4. Assign border points to the cluster of the closest core point.

A cluster contains:

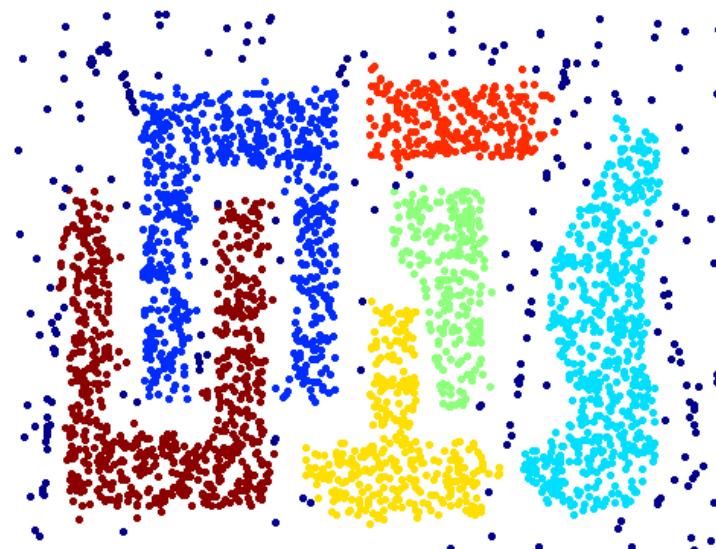
- all core points that can be reached by following a sequence of density-connected core points
- border points that are closest to one of the above-clustered core points

Clustering - DBSCAN

Original Points



Clusters



Clustering - DBSCAN

Pros

- A cluster can have non-convex shape
- No need to define the number of clusters
- Resistant to noise

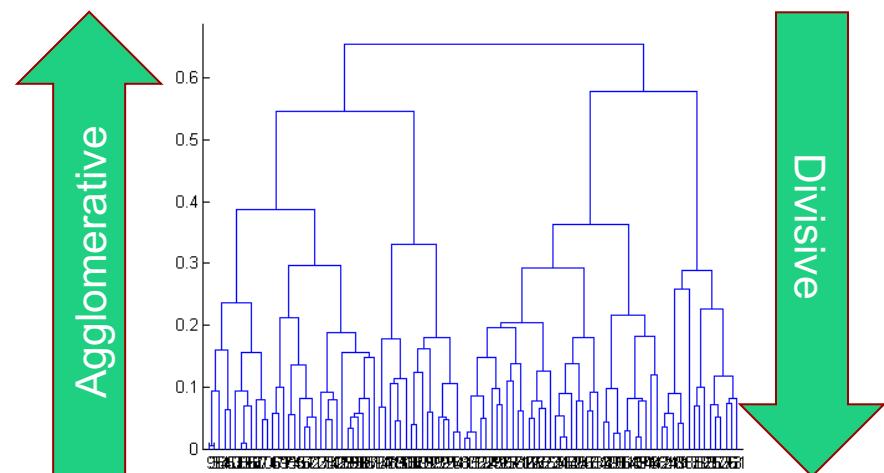
Cons

- Some points (noise points) are not assigned to any cluster (is this necessarily bad?)
- Need to define ϵ and $MinPts$
- Problems with point clouds that contain regions of varying densities

Builds a **hierarchy of clusters**, i.e., *clusters that consist of other smaller clusters*.

The 2 types of hierarchical clustering:

- **Agglomerative**: This is a "bottom-up" approach. *Each point starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.*
- **Divisive**: This is a "top-down" approach. *All points start in one cluster, and splits are performed recursively as one moves down the hierarchy.*



Agglomerative / Bottom-up hierarchical clustering

is based only on distance (similarity) between the points.

Algorithm steps:

1. Start by assigning each point to its own cluster, obtaining as many clusters as points.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster.

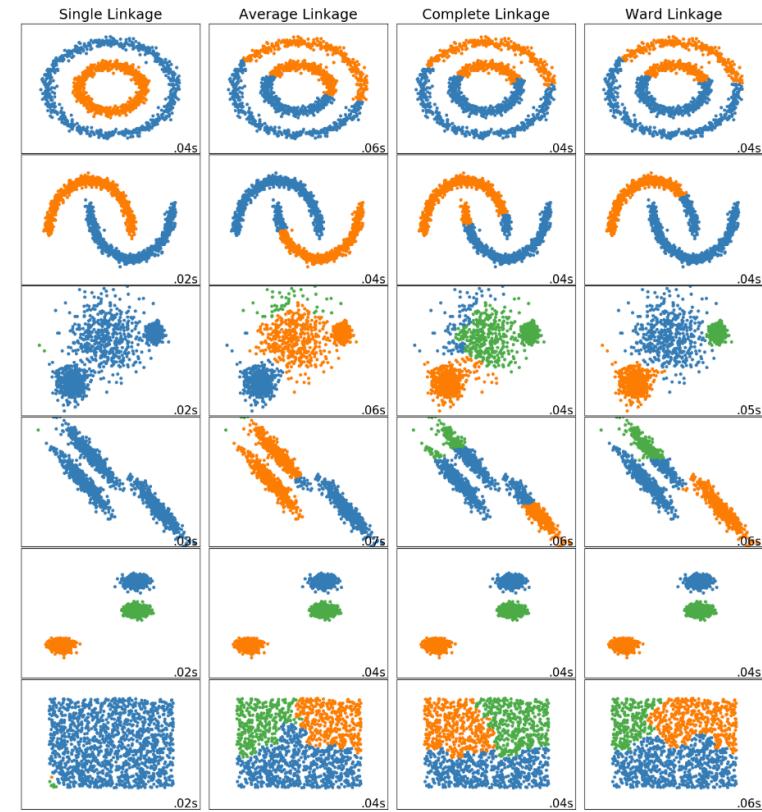
Agglomerative clustering is much more popular than Divisive hierarchical clustering.

Clustering – Hierarchical Clustering

How is distance (similarity) between clusters assessed?

Linkage methods:

- **Single-link:** the distance between two clusters is equal to the minimum distance from any member of one cluster to any member of the other cluster.
- **Complete-link:** the distance between two clusters is equal to the maximum distance from any member of one cluster to any member of the other cluster.
- **Average link:** the distance between two clusters is equal to the average distance from any member of one cluster to any member of the other cluster.
- **Ward-link:** the distance between two clusters is equal to the sum of squared differences within any member of one cluster to any member of the other cluster.



Hierarchical clustering representation: The output of the hierarchical clustering is a dendrogram.

A dendrogram:

- Consists of many Π-shaped lines that connect data points in a hierarchical tree.
- The height of each Π represents the distance between the two data points being connected.

Characteristics of Hierarchical Clustering:

- Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level
- They may correspond to meaningful taxonomies

- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering
- Summary

Summary

- Machine Learning techniques can provide useful tools for:
 - finding the underlying structure or
 - fitting modelsto point clouds.
- Regression can fit models of lines, planes, non-linear surfaces
- Clustering is a very powerful tool.
 - No need for labeled data, as it belongs to the unsupervised learning algorithms
 - Many different methods for clustering with different pros and cons.
- 3D point clouds might need to be expanded in higher dimensional spaces to consider additional information (e.g. color, gradients, texture, ...)
 - Machine Learning techniques still applicable in these higher dimensional spaces!

Lazaros Nalpantidis

3D Point Cloud Processing - Clustering & Regression

Perception for Autonomous Systems

Lecture 4 - 3D Point Cloud Processing (Clustering & Regression) (01/03/2021)

Outline/Content:

- A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
 - k-means
 - Mean Shift
 - DBSCAN
 - Hierarchical Clustering

Reading Material:

- Book A, Sections 6.1 and 6.2
- [Paper 1 - ICP] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, Feb. 1992.
- [Paper 2 - Spin Images] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 433-449, May 1999.
- [Paper 3 - PFH] R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.
- [Paper 4 - FPFH] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning Point Cloud Views using Persistent Feature Histograms," in Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

A taxonomy of ML algorithms

Supervised Learning: The target values are known:

- Regression: The target value is numeric
- Classification: The target value is nominal

Unsupervised Learning: The target values are unknown

- Clustering: Group together similar instances

Reinforcement learning: Interacting with a dynamic environment the system must perform a certain goal.

Why is ML important for Perception of Autonomous Systems? [Article](#)

Machine learning can be employed as a replacement for traditional computer-vision algorithms, making it useful in autonomous vehicles for **object detection, classification, segmentation, tracking, and prediction**.

Doing this will impact the system's level of determinism, safety, and security.

**The Benefits of Using ML for Object Detection and Classification*

ML algorithms

- achieve greater degrees of accuracy than vision-based systems (when trained sufficiently)
- are more adaptable and scalable than vision systems
- can easily handle large volumes of data
- can evolve without human input
- reliance on determinist behavior

For greater elaboration on those bullet points, see the linked article.

The Limitations of Machine Learning for Object Detection and Classification

ML algorithms

- rely heavily on the data quality on which the ML model is trained on. High data quality crucial for good performance in the real world. Otherwise, it could lead to undesirable and possibly dangerous outcomes.
- take long to train/validate and require huge amounts of computing processing resources.
- are limited when it comes to teaching a system how to respond to something that humans have innately. For instance, the "sixth sense" that a car may be about to pull in front of you, or a truck may suddenly slam on the brakes.

For greater elaboration on those bullet points, see the linked article.

What are Clustering and Regression?

Regression

Regression tries to assign correct significance (weights) to input variables and formulate a weighted linear combination of them that can predict the output variables.

- The number of input and output variables can vary depending on the specific problem
- The final output of the algorithm is a regression line
- The goal is to find an equation $f()$ that models the relationship between input x and output y such as

Regression Models:

- Linear Regression: Assume that function f is linear.
- Locally Weighted Regression: Creates multiple linear models on small neighbor data.

Other types of Regression models exist, but are not relevant for the continuing part of the course [Link](#):

- Logistic Regression
- Lasso Regression
- Ridge Regression
- Elastic Net Regression
- Stepwise Regression

The performance for the linear regression method is evaluated by an error function. The most used one is the Mean Squared Error (MSE):

Equation:

$$MSE = \frac{1}{N} \sum_{n=1}^N (t - y)^2$$

where t is the true value of the learned output and y is the predicted value.

Linear Regression Medium

Linear Regression assumes that $f(x)$ is a linear combination between the inputs x and a set of regression coefficients b:

Equation:

$$y = f(b, x) = b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n + c = b^T x + c$$

The goal of linear regression is to find regression coefficients b that minimize the squared error between the true values of the output and the predicted values.

How do we minimize the squared error?

This can be done by using the Gradient Descent algorithm. The main goal of Gradient descent is to minimize the cost value of a function.

Gradient descent has an analogy in which we have to imagine ourselves at the top of a mountain valley and left stranded and blindfolded, our objective is to reach the bottom of the hill. Feeling the slope of the terrain around you is what everyone would do. Well, this action is analogous to calculating the gradient descent, and taking a step is analogous to one iteration of the update to the parameters.

Polynomial Regression Medium Article

Linear regression only works with linear data, while polynomial regression, which is a special case of linear regression, is used to deal with non-linear data points. In this case, a curvilinear relationship is established between input x and output y.

This is done by adding dimensions to the input data, where n is the degree of polynomial.

Polynomial regression is seen as a special case of linear regression since it is still linear in the regression coefficients.

Equation:

$$Y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots + \theta_n x^n$$

Where:

- Y is the target
- x is the predictor
- Omega_0 is the bias
- Omega_1 - Omega_n are the weights in the equation of the polynomial regression
- n is the degree of the polynomial

The degree of polynomial can be found by cross-validation

Pros	Cons
Provides the best approximation of the relationship between the dependent and independent variables	The presence of one or two outliers in the data can seriously affect the result of the nonlinear analysis
A broad range of functions can be fit under it	Too sensitive to outliers
Polynomial basically fits a wide range of curvature	Presence of fewer model validation tools for the detection of outliers in nonlinear regression

Use cases for Regression on 3D point clouds:

- Line fitting
- Plane fitting
- Grid point fitting [Paper](#)
- Fitting other (non-linear) surfaces

If you were wondering about whether 3D point clouds and 3D meshes are the same:

- A point cloud is a collection of points to represent an object in the given coordinate system. For example, in a 3 dimensional (X, Y, Z) coordinate system, a point cloud represents a 3D object. Point clouds are used to create 3D meshes and other models used in 3D modeling
- A 3D mesh is the structural build of a 3D model consisting of polygons. 3D meshes use reference points in X, Y and Z axes to define shapes with height, width and depth.

Clustering (or "Cluster Analysis")

Find previously unknown groups ("clusters") in a data set, such that:

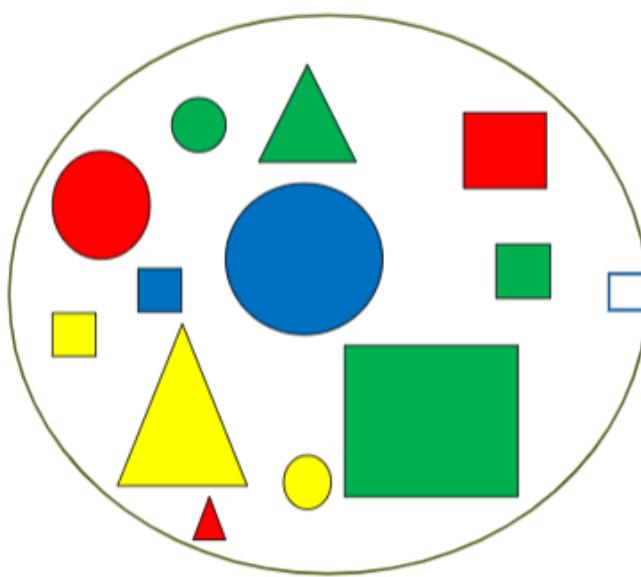
- data objects within one cluster are **similar to each other**
- data objects of different clusters are **dissimilar to each other**

Some applications of clustering:

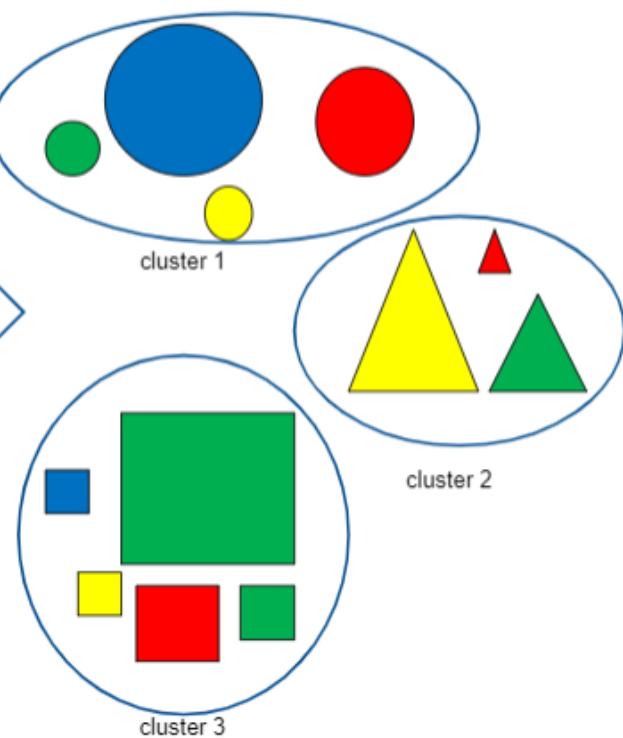
- find different types of customers
 - e.g. an online shop could find groups of customers that are likely to spend a lot of money and customers that have not spent money for a long time
- detect communities in social networks
- group similar pixels in images (image processing)
- group similar documents (search engines, text mining)
- find similar recordings from technical systems (automotive, automation)

- etc.

Data set:



Clustering result
(groups found by the similarity of objects)



The image could be clustered in different ways, for example shape, sizes or colors. Clustering is known as unsupervised learning because no information about classes is available for the instances.

The most common types of clustering methods are:

- partitioning methods
- hierarchical methods
- density-based methods
- grid-based methods

Partitioning method (k-means)

- find k clusters in the data set (k has to be pre-defined !)
- each cluster must contain ≥ 1 instances
- each instance must belong to exactly one cluster
- usually distance-based

One distance based approach is to minimize the sum of squared euclidean distances (distortion)

$$J = \sum_k \sum_{n \in k} d(x_n, \mu_k)^2$$

- where x is a vector representing the n^{th} data point assigned to cluster k and μ_k is the centroid of the cluster k . The function $d()$ is the Euclidean distance between x and μ_k

Steps:

1. Creation of initial partitioning (e.g. randomly)
2. Iterative improvement of the partitioning by moving objects from one cluster to another. This is done by optimizing some criterion of what a "good" partitioning should look like.
3. Stop, if the partitioning quality criterion is satisfied.

Algorithm k-Means visualised

Algorithm: k-means. The k -means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- k : the number of clusters,
- D : a data set containing n objects.

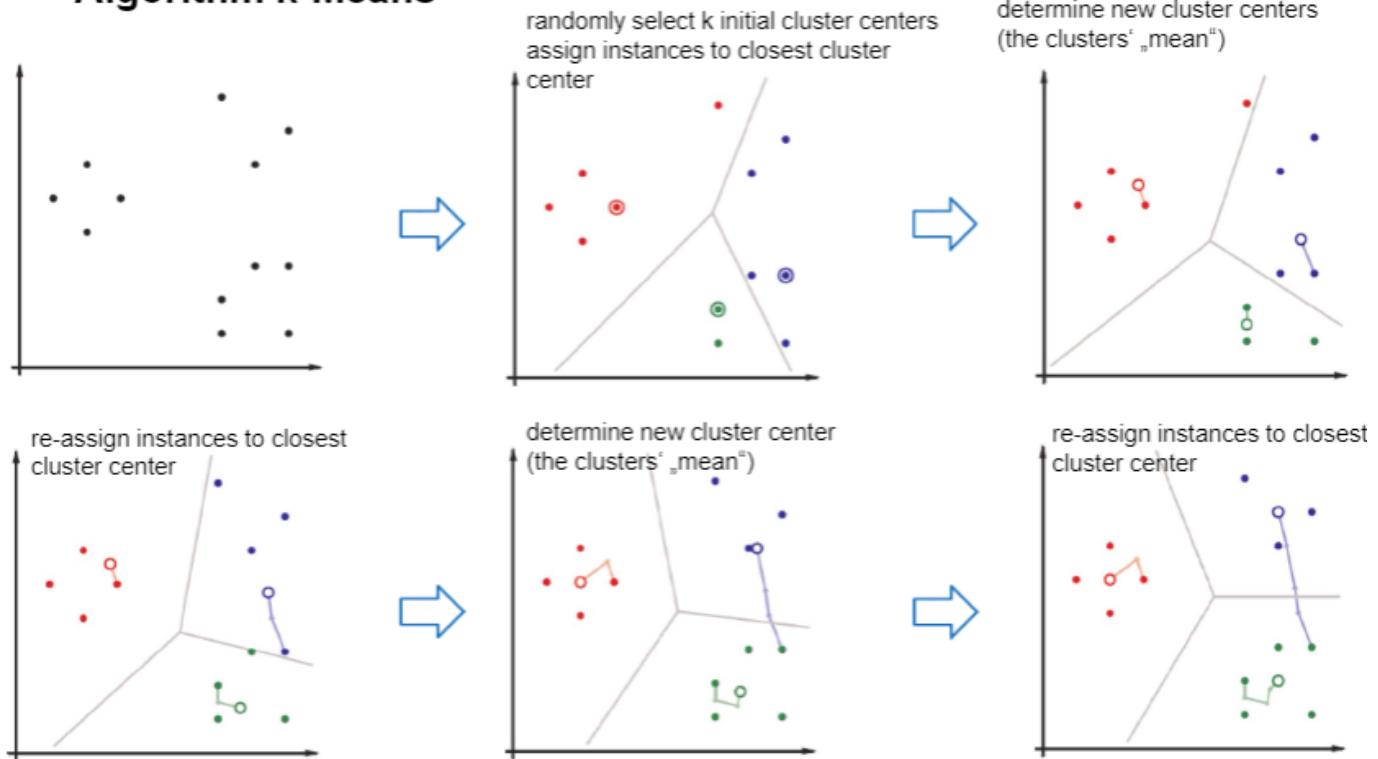
Output: A set of k clusters.

Method:

- (1) arbitrarily choose k objects from D as the initial cluster centers;
- (2) **repeat**
- (3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
- (4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
- (5) **until** no change;

Algorithm k-Means visualised

Algorithm k-Means



How to define k?

There are multiple ways to find the best k value for a given data set. However, two approaches are the most popular, namely:

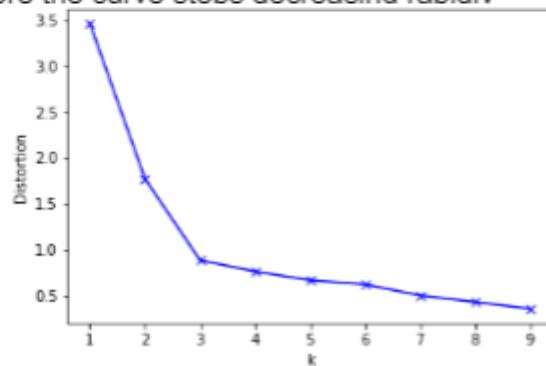
- Elbow method
- Silhouette Analysis

The following illustrations elaborate the key points of those approaches, nicely.

How to define k ?

– Elbow method

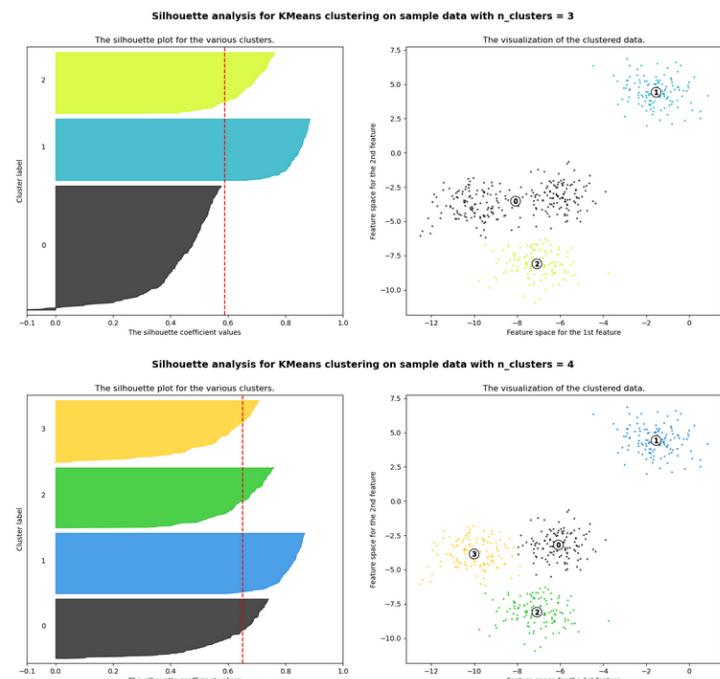
- Run k-means for several k and calculate distortion for each k
 - » Distortion: sum of squared distances of each point to the center of the closest cluster
- Plot distortion vs k
- Look for k where the curve stops decreasing rapidly



How to define k ?

– Silhouette Analysis

- Calculate Silhouettes for various k
 - Silhouette coefficient shows how far each sample is from other cluster centers
 - » +1: far from others
 - » 0: at the boundary
 - » -1: sample is on average closer to the points of another cluster
- Thickness shows cluster size (number of points assigned to cluster)
- Avoid k that leads to:
 - Scores for clusters < average
 - Individual scores < 0
 - Big differences in cluster sizes (thickness)



Limitations of k-means:

- K needs to be specifically defined

- Final cluster assignments depend on initialization
 - Cluster assignments may be different on different runs
 - K-means may not achieve the global optimum
- Can describe only convex clusters geometries
- Computationally demanding as the number of points dimensions increases

3D Point Cloud Segmentation by k-means [Point Cloud Expert: Florent Poux Medium](#)

My take on that is, that in terms of dimensionality it should be 3, for k means and mean shift respectively, since clustering methods usually act in a manner of dimensionality reduction. However, there might be exceptions of that rule? Or maybe, I'm wrong with my assumption altogether.

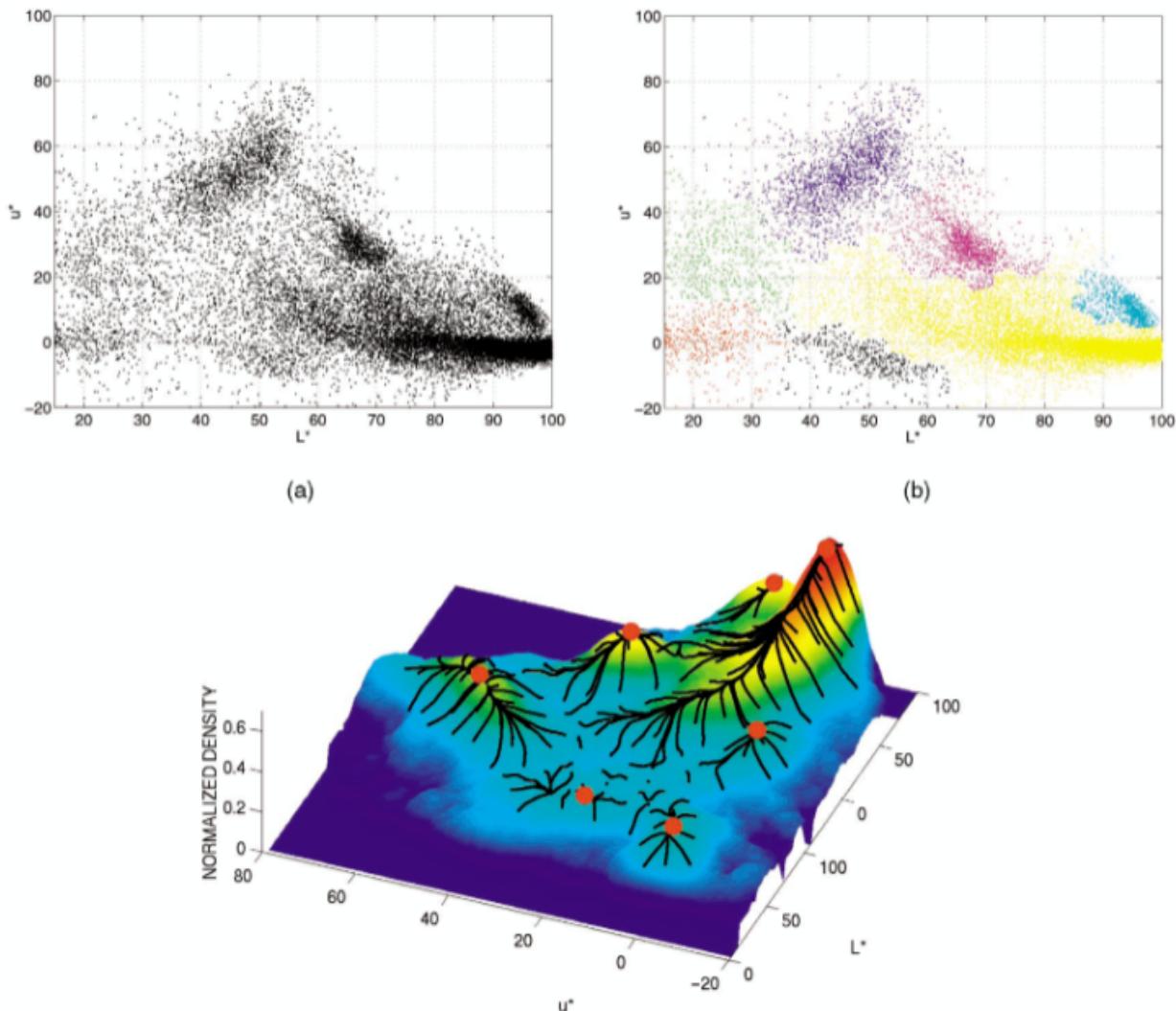
Clustering - Mean shift

Definition: Mean Shift is an algorithm that finds the maxima - the modes - of a density function given discrete data sampled from that function. Thus, it is a density hill climbing algorithm

- Every point gives rise to a cluster.
- Each cluster is defined by the radius, a.k.a "bandwidth", h of its region, and a kernel function K that is used to calculate the contribution of the points included within this radius.
- In the end, only unique clusters are considered.
- So, we do not need to set the number of clusters!
 - However, we need to define the bandwidth (and the kernel function).

Further definitions:

- Attraction basin: the region for which all trajectories lead to the same mode
- Cluster: all data points in the attraction basin of a mode

**Pros**

General, application-independent tool

Model-free, does not assume any prior shape (spherical, elliptical, etc.) on data clusters

Just a single parameter (window size h)

Finds variable number of modes

Robust to outliers

Cons

Output depends on window size

Window size (bandwidth) selection is not trivial

Computationally (relatively) expensive (~2s/image)

Does not scale well with dimension of feature space

Kernel density estimation

Kernel density estimation function

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$$

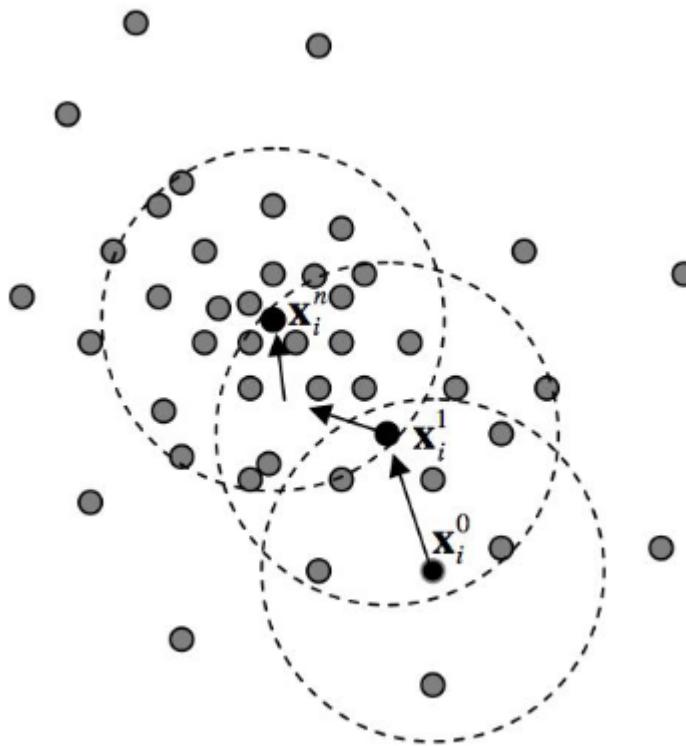
Gaussian kernel (typically used)

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}.$$

Mean shift clustering

The mean shift algorithm seeks modes of the given set of points

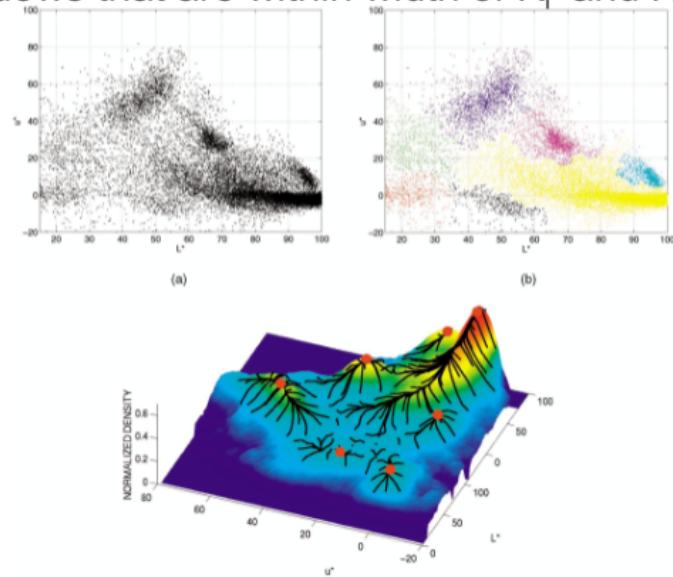
1. Choose kernel and bandwidth
2. For each point:
 1. Center a window on that point
 2. Compute the mean of the data in the search window
 3. Center the search window at the new mean location
 4. Repeat (2,3) until convergence
3. Assign points that lead to nearby modes to the same cluster



Mean shift procedure. Starting at data point x_i , run the mean shift procedure to find the stationary point of the density function. Superscripts denote the successive window centres, respectively, and the dotted circles denote the density estimation windows.

2D image Segmentation by Mean Shift

- Compute features for each pixel (color, gradients, texture, etc)
- Set kernel size for features K_f and position K_s
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- Merge windows that are within width of K_f and K_s



3D Point Cloud Segmentation by Mean Shift

Point Cloud Expert: Florent Poux Medium

My take on that is, that in terms of dimensionality it should be 3, for k means and mean shift respectively, since clustering methods usually act in a manner of dimensionality reduction. However, there might be exceptions of that rule? Or maybe, I'm wrong with my assumption altogether.

Clustering - DBSCAN Article

Definition: DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise and is a unsupervised density-based clustering algorithm.

DBSCAN requires only two parameters: $\$\\epsilon$ and minPoints. $\$\\epsilon$ is the radius of the circle to be created around each data point to check the density and minPoints is the minimum number of data points required inside that circle for that data point to be classified as a Core point.

In density-based clustering:

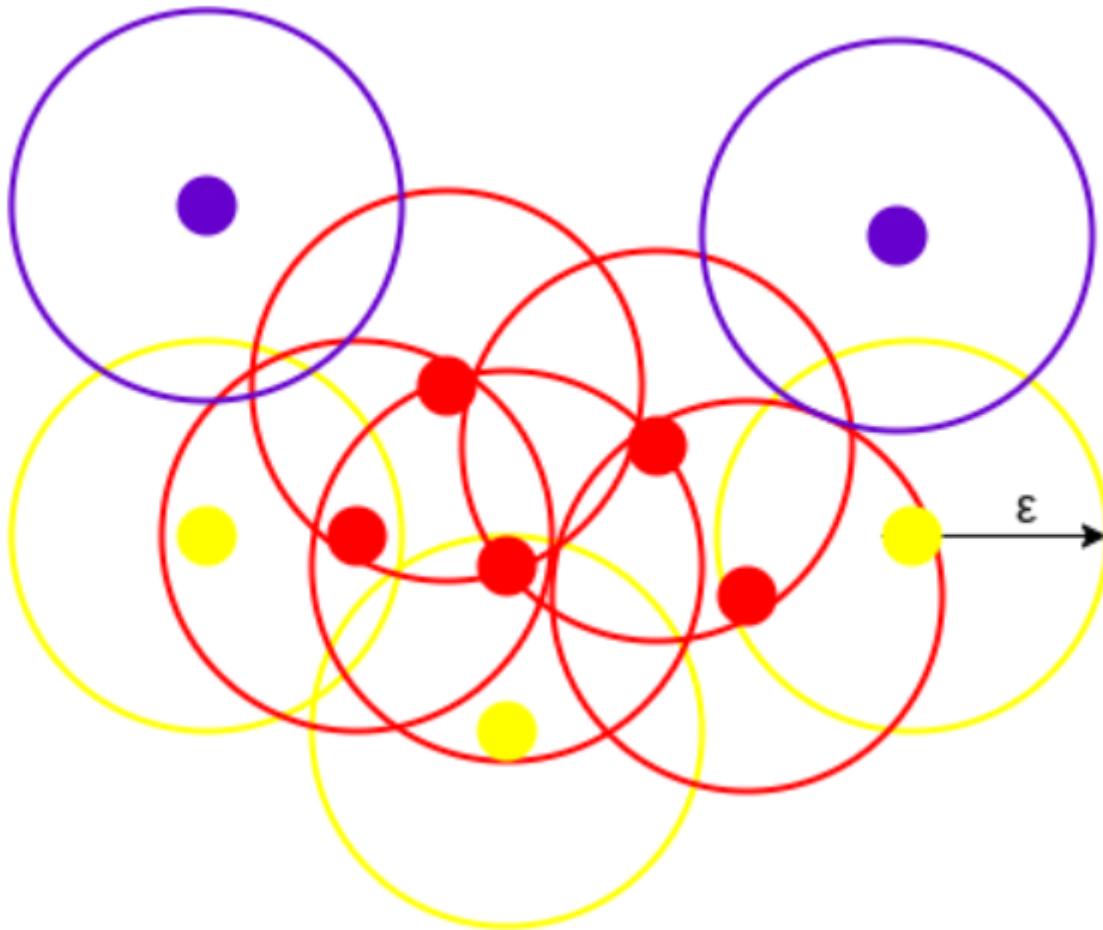
- we partition points into dense regions separated by not-so-dense regions.
- a cluster is defined as a maximal set of density-connected points
- we can discover clusters of arbitrary shape

Density definition:

- Density at point p is defined as the number of points within a circle/sphere of radius $\$\\epsilon$
- A region is dense if the circle/sphere of radius $\$\\epsilon$ contains at least MinPts points.

Types of points:

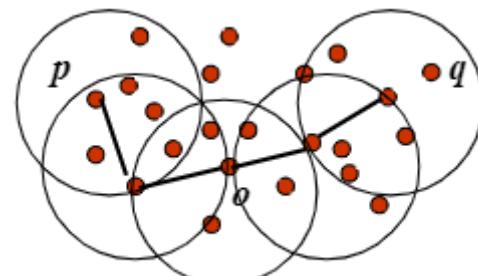
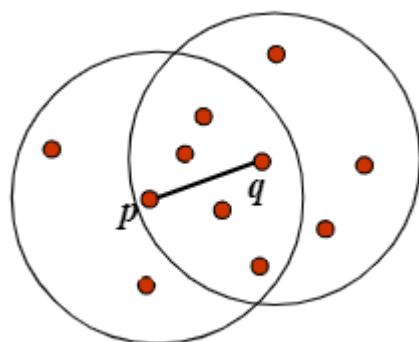
- A core point has more than a specified number of points (MinPts) within $\$\\epsilon$
- A border point has fewer than MinPts within $\$\\epsilon$, but is the neighborhood of a core point.
- A noise point is any point that is not a core point or a border point.

Example

All the data points with at least 3 points ($\text{MinPnts} = 2$) in the circle including itself are considered as Core points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as Border points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as Noise represented by the purple color.

Reachability and Connectivity

Reachability states if a data point can be accessed from another data point directly or indirectly, whereas Connectivity states whether two data points belong to the same cluster or not.

Density edge**Density-connected**

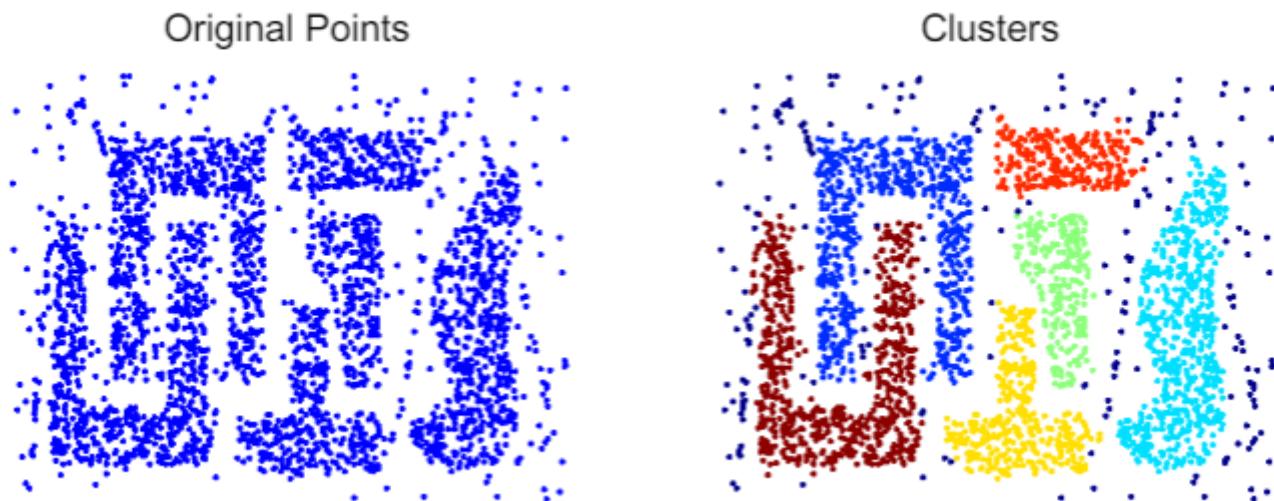
Density edge	Density-connected
We place an edge between two core points q and p if they are within distance ϵ	A point p is density-connected to a point q if there is a path of edges from p to q
There are other accessibility and connectivity metrics: e.g. direct density-reachable and density-reachable. See the linked article for further explanation.	

DBSCAN algorithm

1. Label points as core, border and noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster
 - o Create a new cluster with the point p and all the points that are density-connected to p.
4. Assign border point to the cluster of the closest core point.

For locating data points in space, DBSCAN uses Euclidean distance, although other methods can also be used (like great circle distance for geographical data). It also needs to scan through the entire dataset once, whereas in other algorithms we have to do it multiple times.

Visualization



Pros	Cons
Can create clusters of arbitrary shapes	Some points (noise points) are not assigned to any cluster (is this necessarily bad?)
No need to define the number of clusters	Need to define ϵ and MinPts
Resistant to noise	Problems with point clouds that contain regions of varying densities

Hierarchical Clustering

Hierarchical methods create a hierarchy of splits/merges of a dataset:

- **agglomerative methods (bottom-up):**

1. start with each data object being one cluster
2. iteratively merge the clusters
3. stop when all clusters are merged or a stopping condition is met

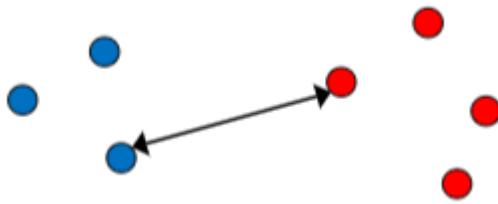
- **divisive methods (top-down):**

1. start with all data objects being one cluster
2. iteratively split each cluster into smaller ones
3. stop when each object forms its own cluster or a stopping condition is met

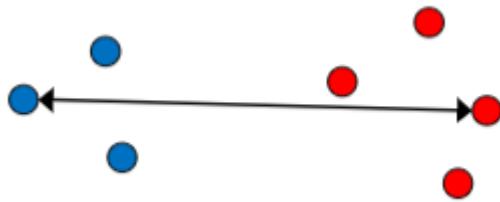
- merging or splitting is done based on dissimilarities (= distances, so-called "linkages") or on densities
- a merging or splitting step can not be undone

Linkage variants

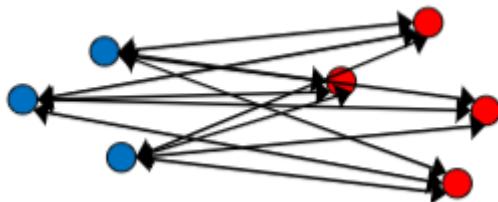
Single-link: the distance between two clusters is equal to the minimum distance (dissimilarity) between the two most similar data objects of the two clusters.



Complete-link: the distance between two clusters is equal to the maximum distance (dissimilarity) between the two most similar data objects of the two clusters.



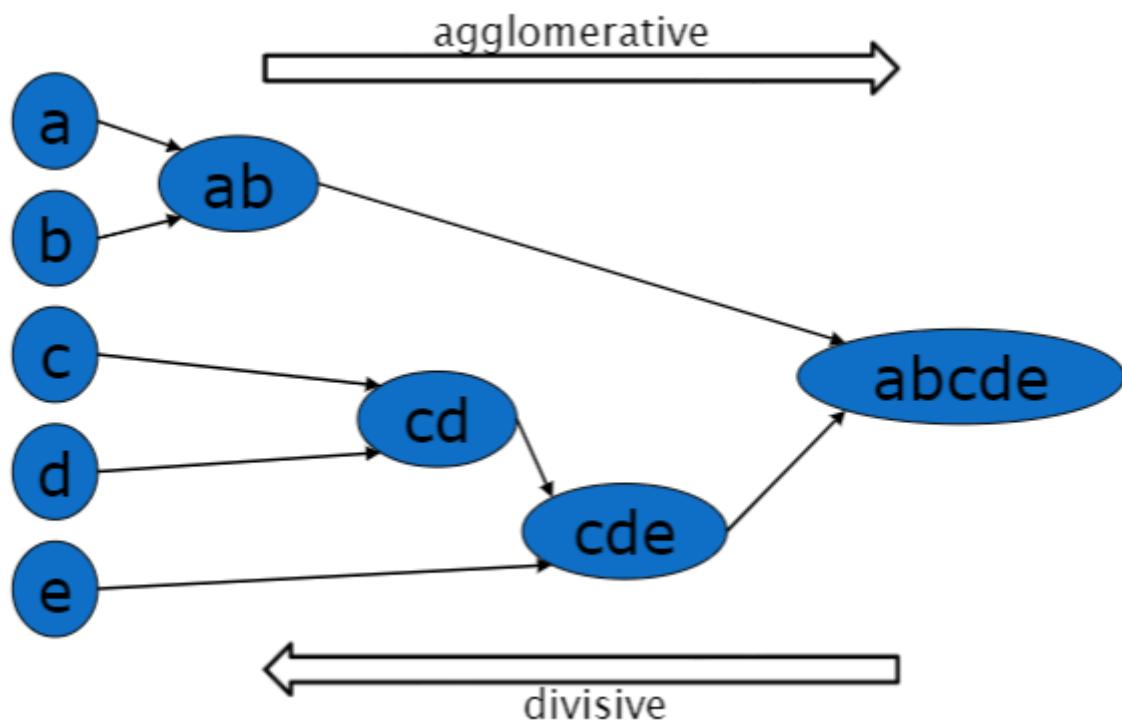
Average-link: the distance between two clusters is equal to the average distance from any member of one cluster to any member of the other cluster.



Centroid: the distance between two clusters is equal to the dissimilarity between the centroids, e.g. the mean value vectors of those two clusters.



The following figure illustrates the process of the hierarchical methods:



Hierarchical clustering representation: The output of the hierarchical clustering is a dendrogram.

A dendrogram:

- Consists of many Π-shaped lines that connect data points in a hierarchical tree like structure.
- The height of each Π represents the distance between the two data points being connected.

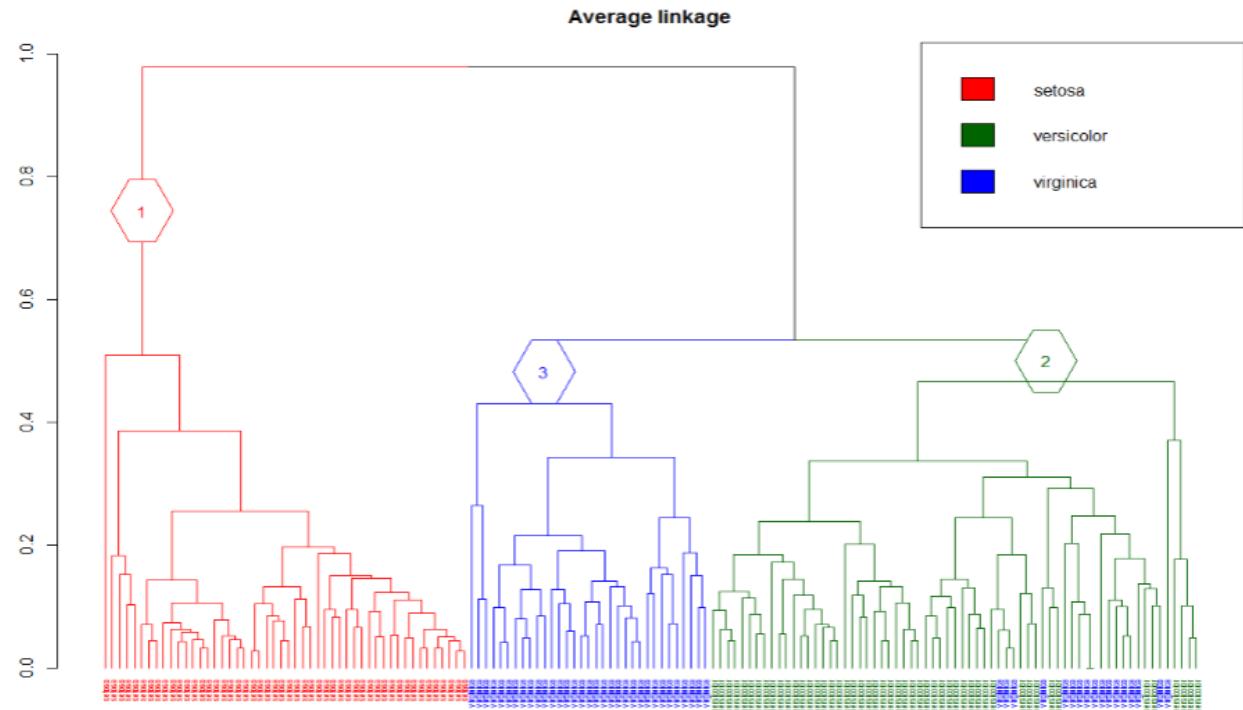
Characteristics of Hierarchical Clustering:

- Any desired number of clusters can be obtained by 'cutting' the dendrogram at the proper level
- They may correspond to meaningful taxonomies

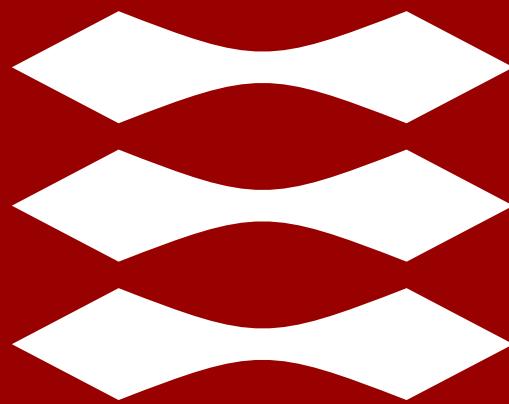
Illustration of a dendrogram

dendrogram of iris dataset

(with colours corresponding to cluster assignments, for 3 clusters)



DTU



Perception for Autonomous Systems 31392:

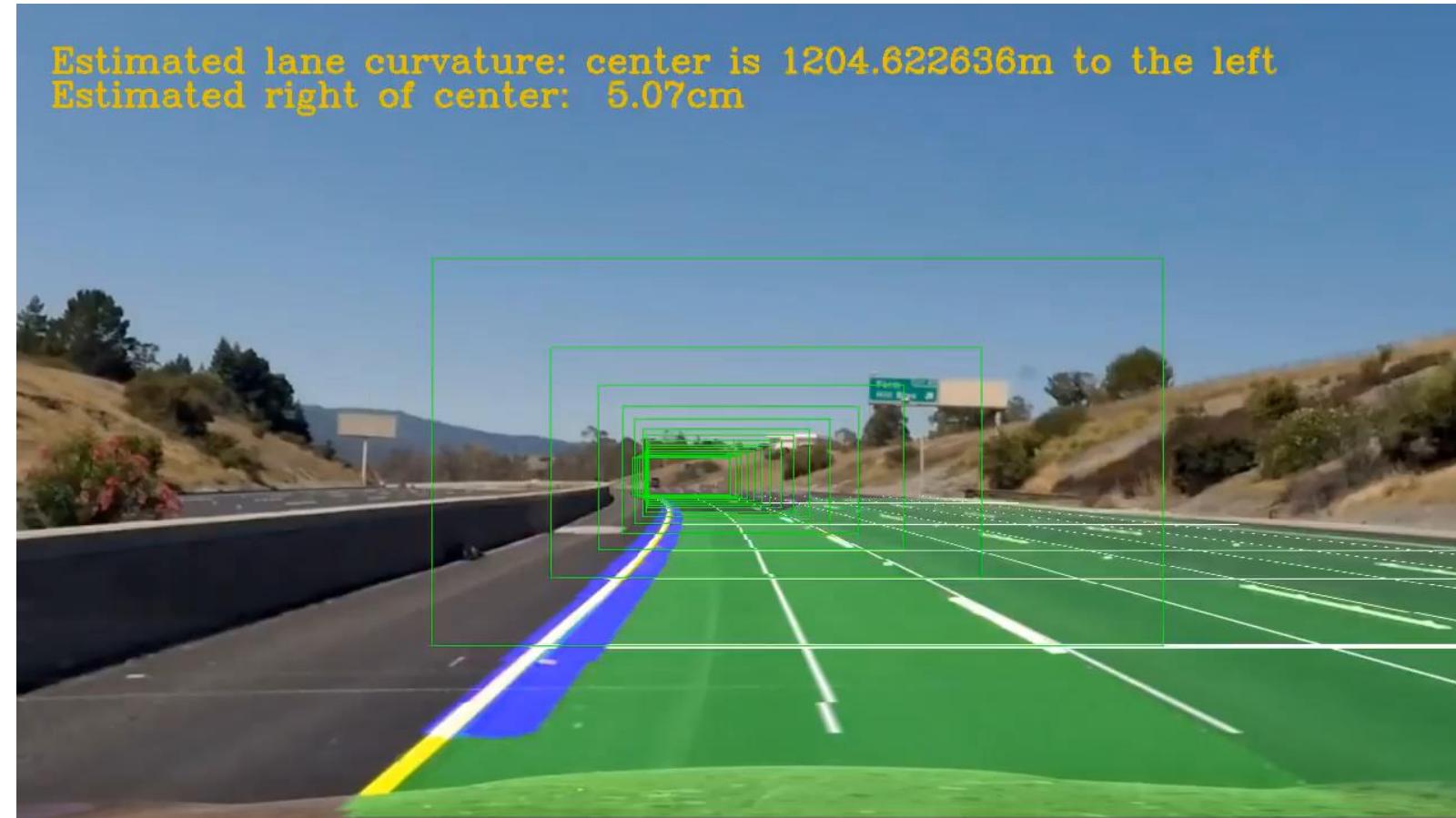
State Estimation - Histogram Filter

Lecturer: Evangelos Boukas—PhD

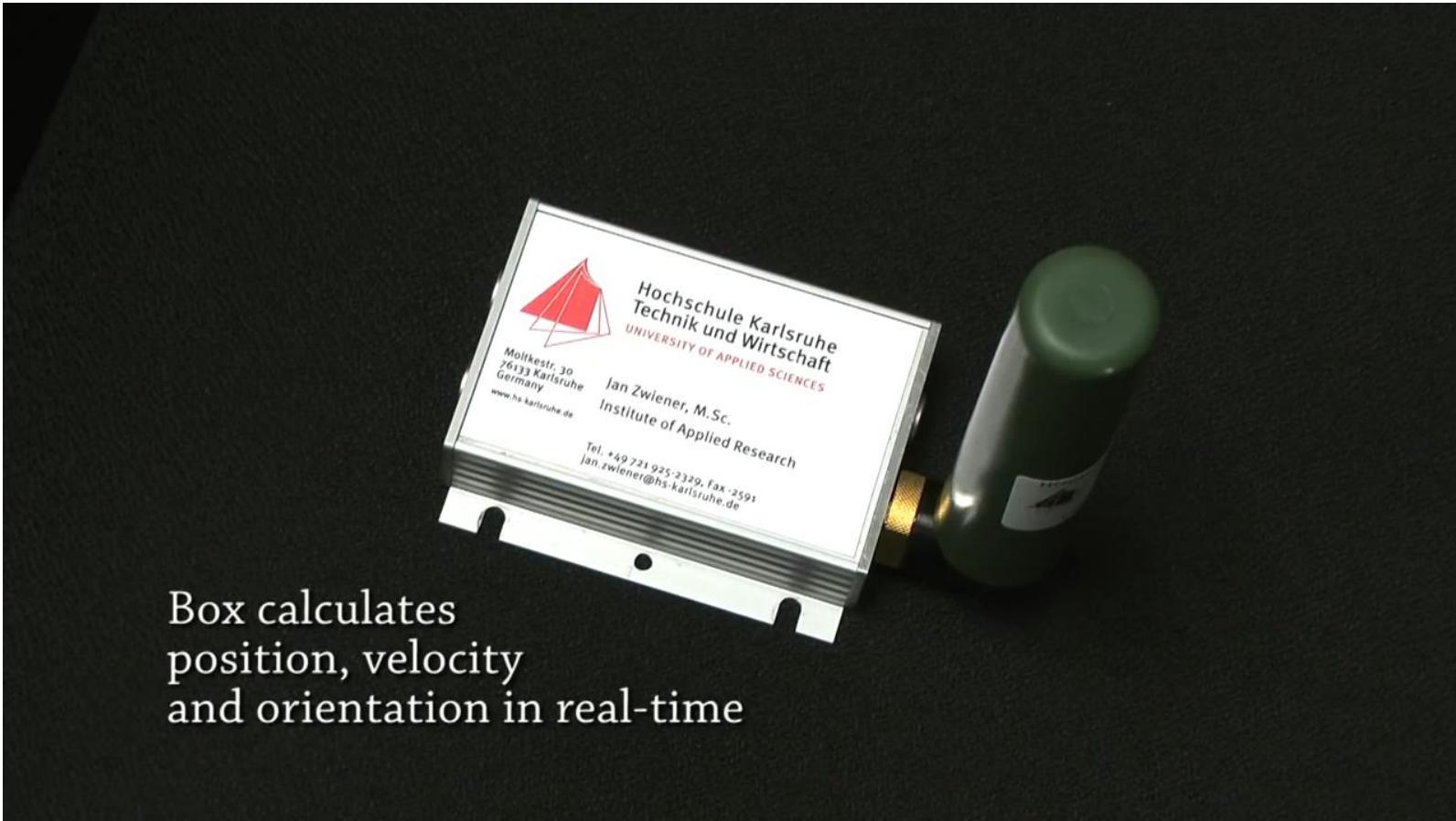
Cases of State Estimation



Cases of State Estimation



Cases of State Estimation



What is State Estimation

Goal:

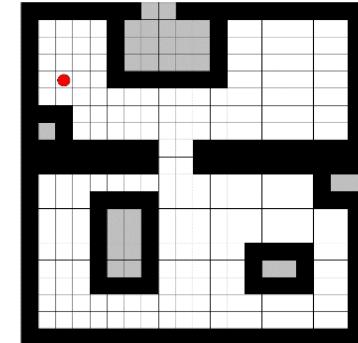
- Given a State Vector of a system
- Estimate over time the state using input of external sensors

Useful for:

- Localization
- Tracking
- Prediction
- Sensor Fusion
- ...

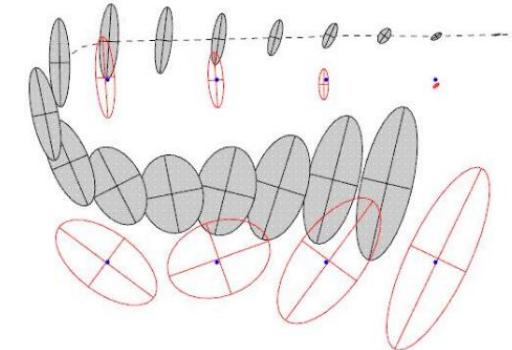
Continuous vs Discrete State Unimodal vs Multimodal Distribution

- State:

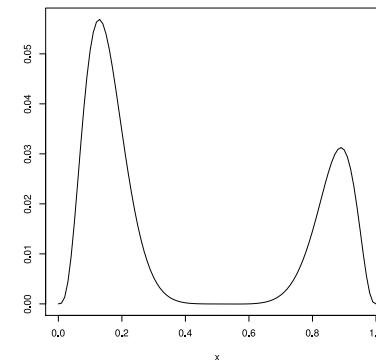


Discrete

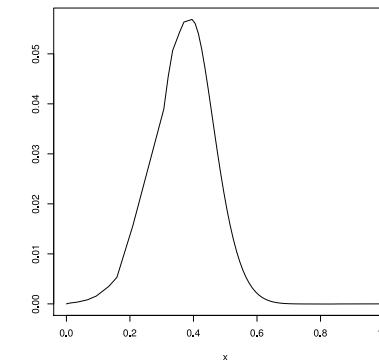
- Distribution



Continuous



Multimodal



Unimodal

Simple State Estimation Example

- Assume a robot in an area like this one:

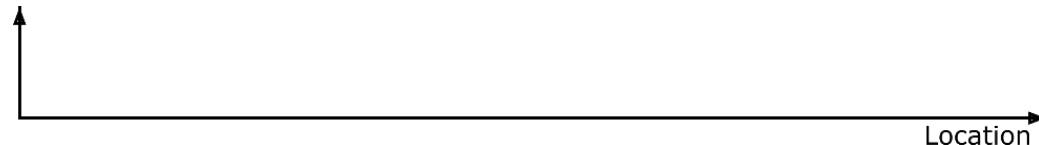


Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



Simple State Estimation Example

- Assume a robot in an area like this one:



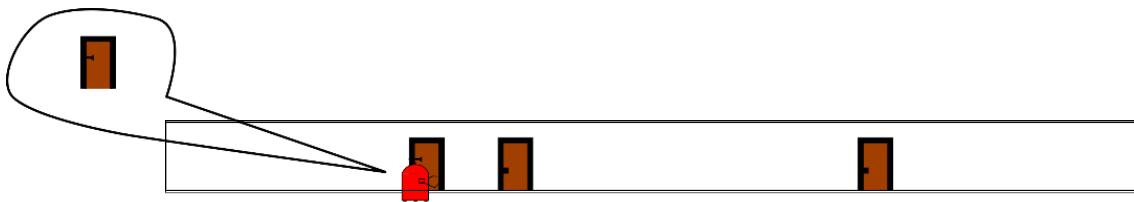
- We model the position of the robot as follows:



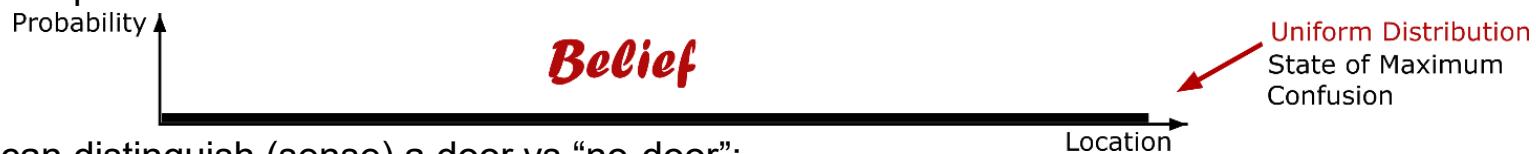
- The robot can distinguish (sense) a door vs “no-door”:

Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:

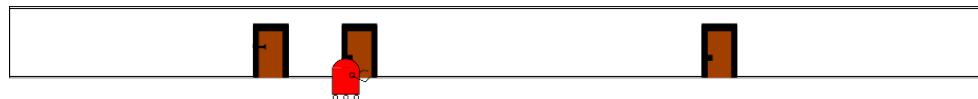


- The robot can distinguish (sense) a door vs "no-door":



Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



- The robot can distinguish (sense) a door vs “no-door”:

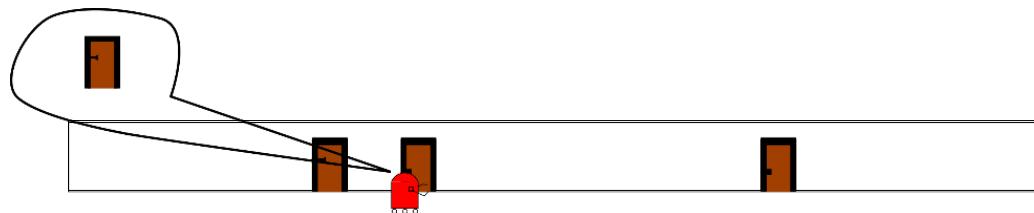


- The robot moves to the right:

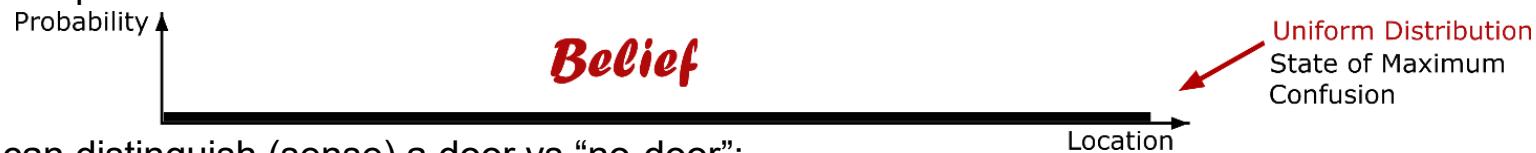


Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



- The robot can distinguish (sense) a door vs “no-door”:



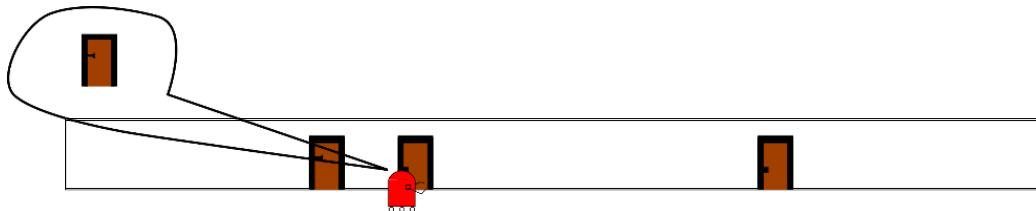
- The robot moves to the right:



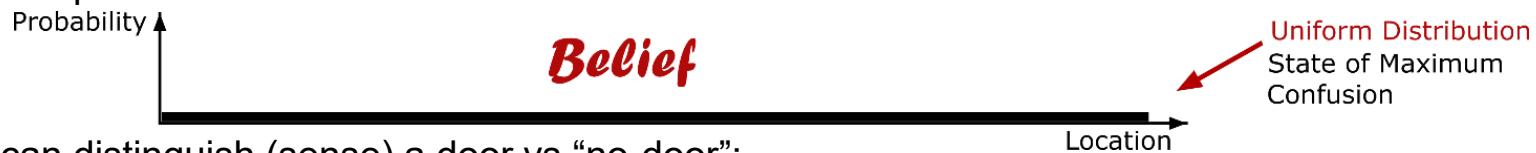
- The robot senses again:

Simple State Estimation Example

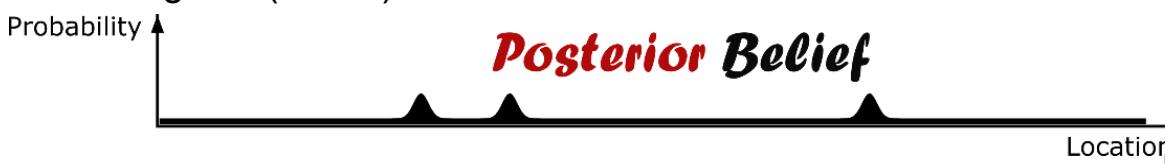
- Assume a robot in an area like this one:



- We model the position of the robot as follows:



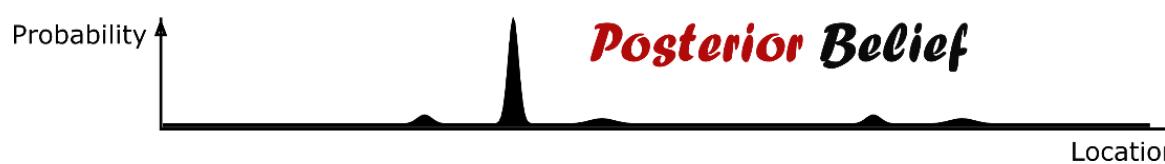
- The robot can distinguish (sense) a door vs “no-door”:



- The robot moves to the right:

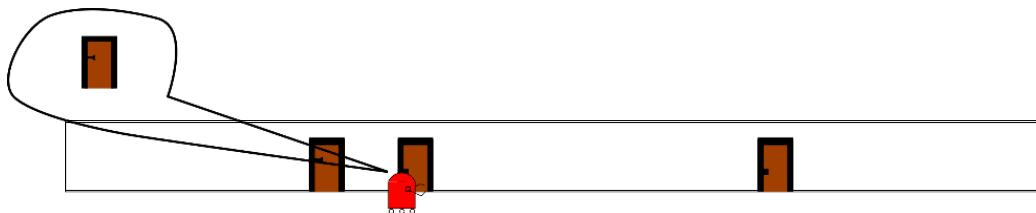


- The robot senses again:

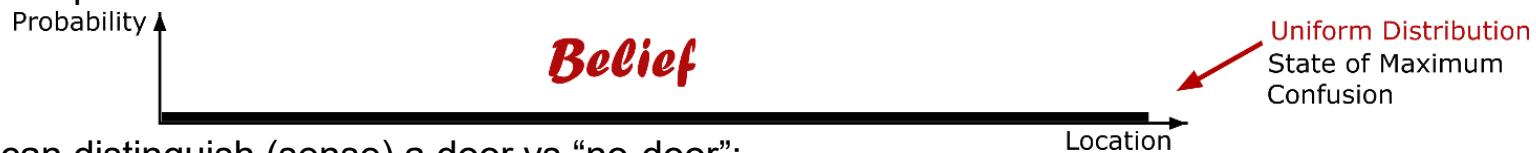


Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



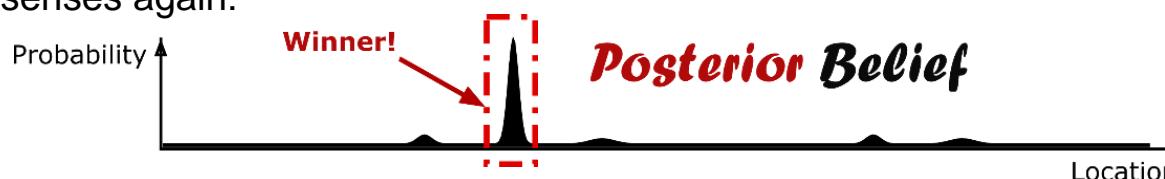
- The robot can distinguish (sense) a door vs “no-door”:



- The robot moves to the right:

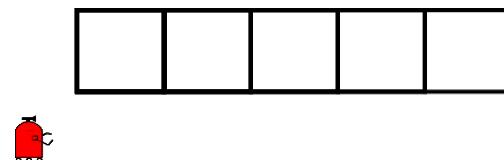


- The robot senses again:



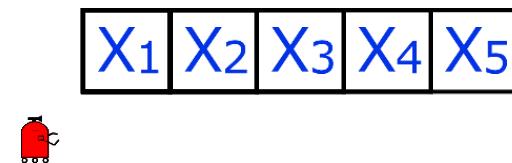
Sense, Let's build it ourselves!

- This state estimation problem is called localization
- Assume a robot which can be in one of five blocks:



Sense, Let's build it ourselves!

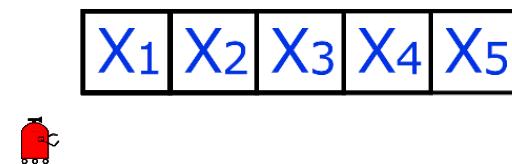
- This state estimation problem is called localization
- Assume a robot which can be in one of five blocks:



- Without any info, What is the probability of the robot being in each cell?

Sense, Let's build it ourselves!

- This state estimation problem is called localization
- Assume a robot which can be in one of five blocks:



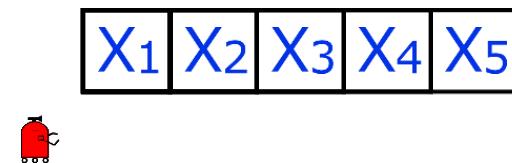
- Without any info, What is the probability of the robot being in each cell?

$$p(X_i) = \underline{\hspace{2cm}}$$

for i in (1 ... 5)

Sense, Let's build it ourselves!

- This state estimation problem is called localization
- Assume a robot which can be in one of five blocks:

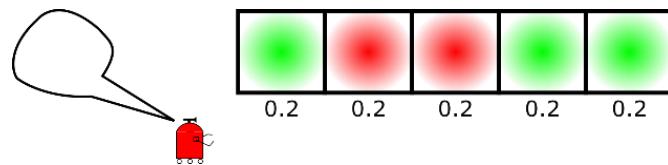


- Without any info, What is the probability of the robot being in each cell?

$$p(X_i) = \frac{0.2}{\text{for } i \text{ in } (1 \dots 5)}$$

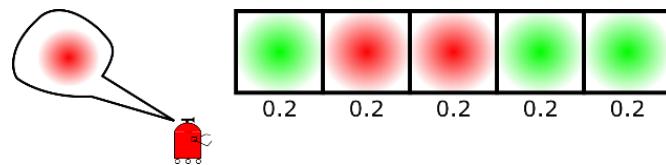
Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



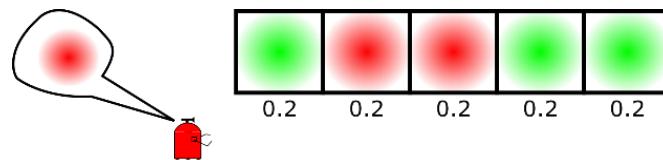
Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



Sense, Let's build it ourselves!

- Now our robot is allowed to sense:

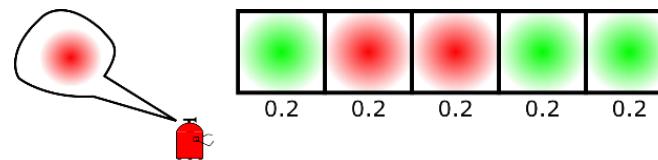


- What does that mean for our probability?



Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



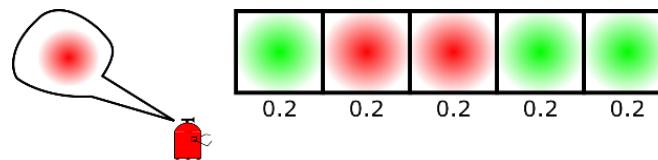
- What does that mean for our probability?



- Let's start by arbitrarily multiplying any correct "sensing" with **0.6** & any incorrect with **0.2**

Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



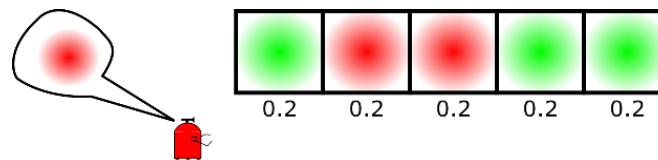
- What does that mean for our probability?



- Let's start by arbitrarily multiplying any correct "sensing" with **0.6** & any incorrect with **0.2**
- This is close to the belief. However, it is wrong, why?

Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



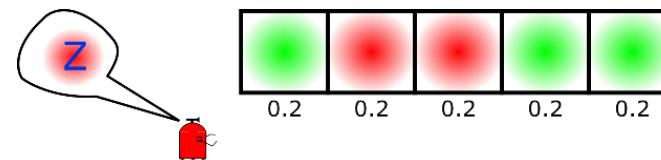
- What does that mean for our probability?

0.04	0.12	0.12	0.04	0.04
------	------	------	------	------

- Let's start by arbitrarily multiplying any correct "sensing" with **0.6** & any incorrect with **0.2**
- This is close to the belief. However, it is wrong, why?

Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



- What does that mean for our probability?

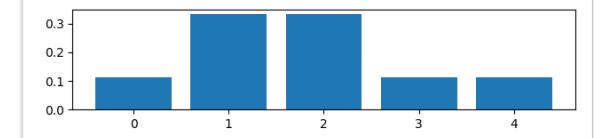
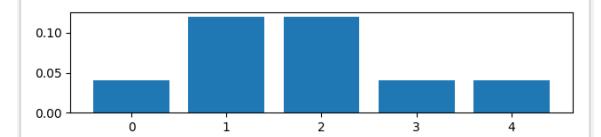
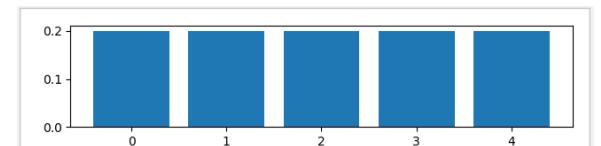
0.11	0.33	0.33	0.11	0.11
------	------	------	------	------

- Let's start by arbitrarily multiplying any correct "sensing" with **0.6** & any incorrect with **0.2**
- This is close to the belief. However, it is wrong, why?
- It does not add up to one. How to do it?

once standardized -> posterior

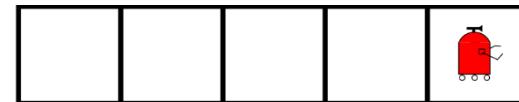
$$p(X_i | Z)$$

Posterior Distribution



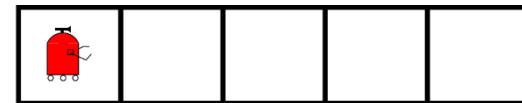
Motion, Let's build it ourselves!

- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



Motion, Let's build it ourselves!

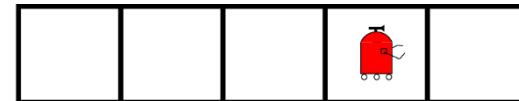
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



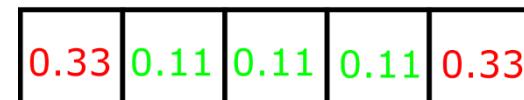
0.11	0.33	0.33	0.11	0.11
------	------	------	------	------

Motion, Let's build it ourselves!

- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?

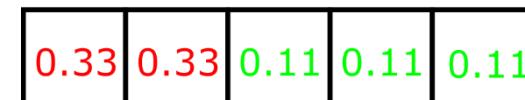


Motion, Let's build it ourselves!

- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



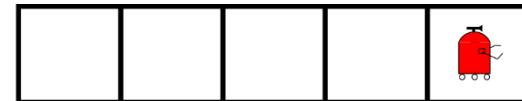
- What will happen with the posterior probability?



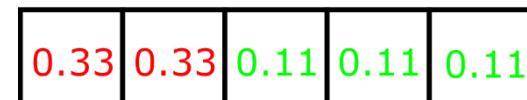
- However, this doesn't happen. Usually the distribution becomes more uncertain:

Motion, Let's build it ourselves!

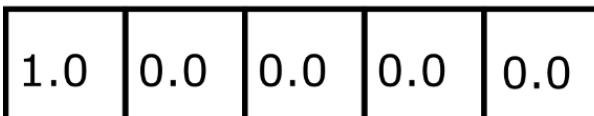
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



- However, this doesn't happen. Usually the distribution becomes more uncertain:



Motion, Let's build it ourselves!

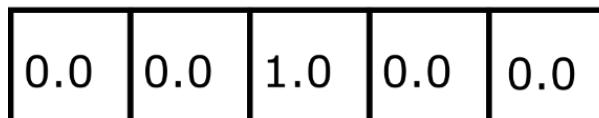
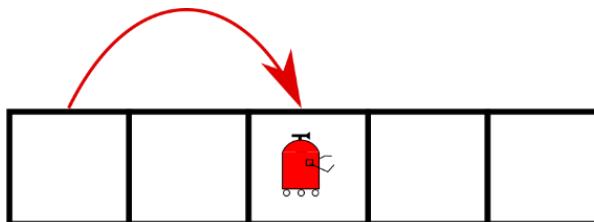
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



- However, this doesn't happen. Usually the distribution becomes more uncertain:



Motion, Let's build it ourselves!

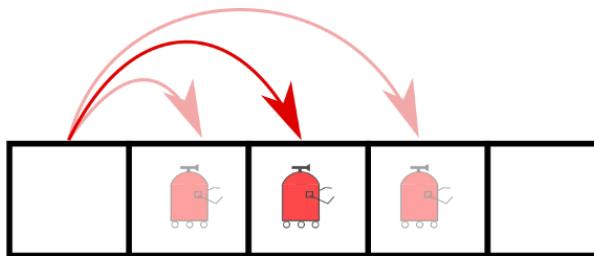
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



- However, this doesn't happen. Usually the distribution becomes more uncertain:

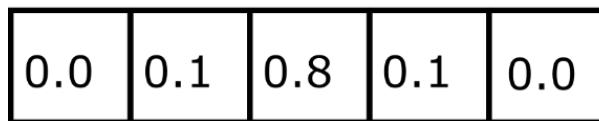


Uncertain
Motion

$$p(X_{i+U-1}) = 0.1$$

$$p(X_{i+U}) = 0.8$$

$$p(X_{i+U+1}) = 0.1$$



Motion, Let's build it ourselves!

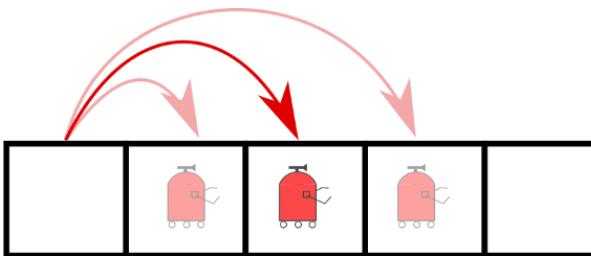
- Assume that the space is circular (i.e when moving right in the last bin, you end up in the first bin)



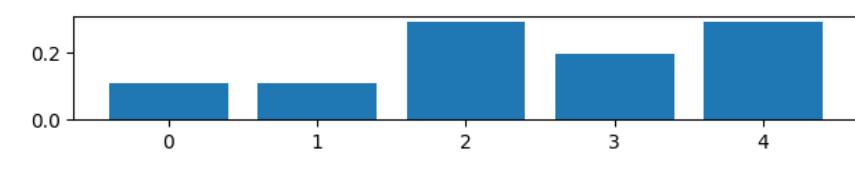
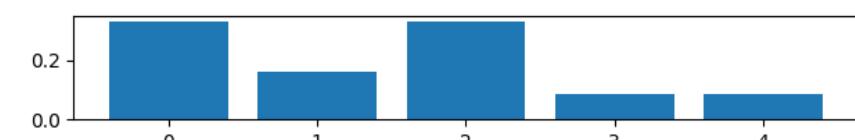
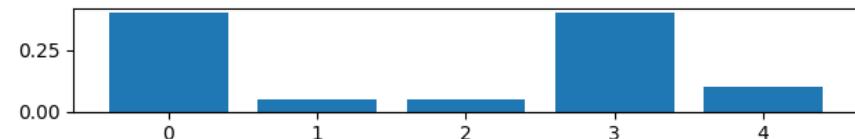
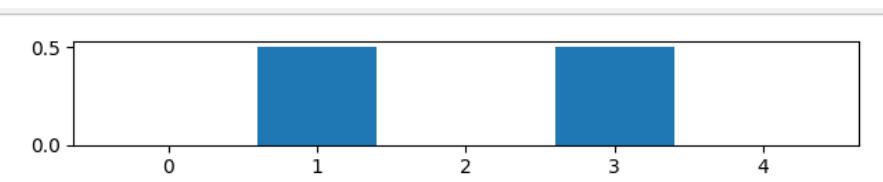
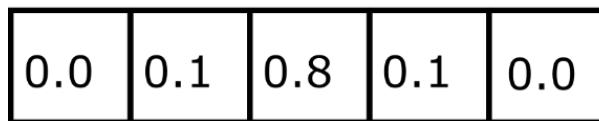
- What will happen with the posterior probability?



- However, this doesn't happen. Usually the distribution becomes



Uncertain Motion



$$p(X_{i+U-1}) = 0.1$$

$$p(X_{i+U}) = 0.8$$

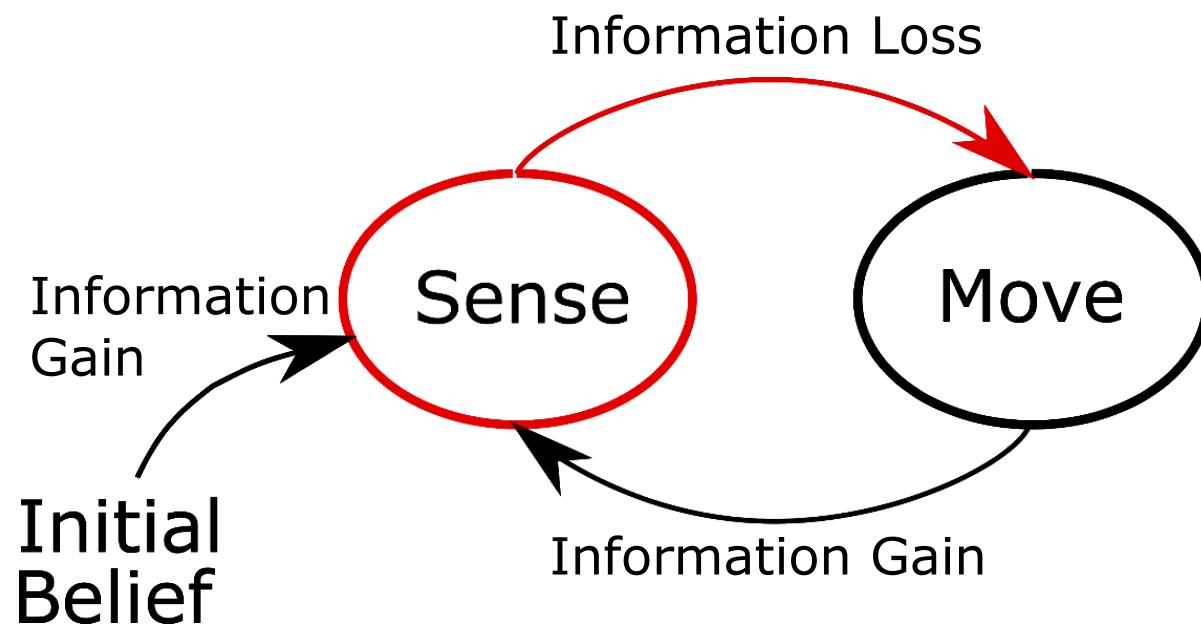
$$p(X_{i+U+1}) = 0.1$$

That's very good!!!!

- Localization is just a sense/move cycle:

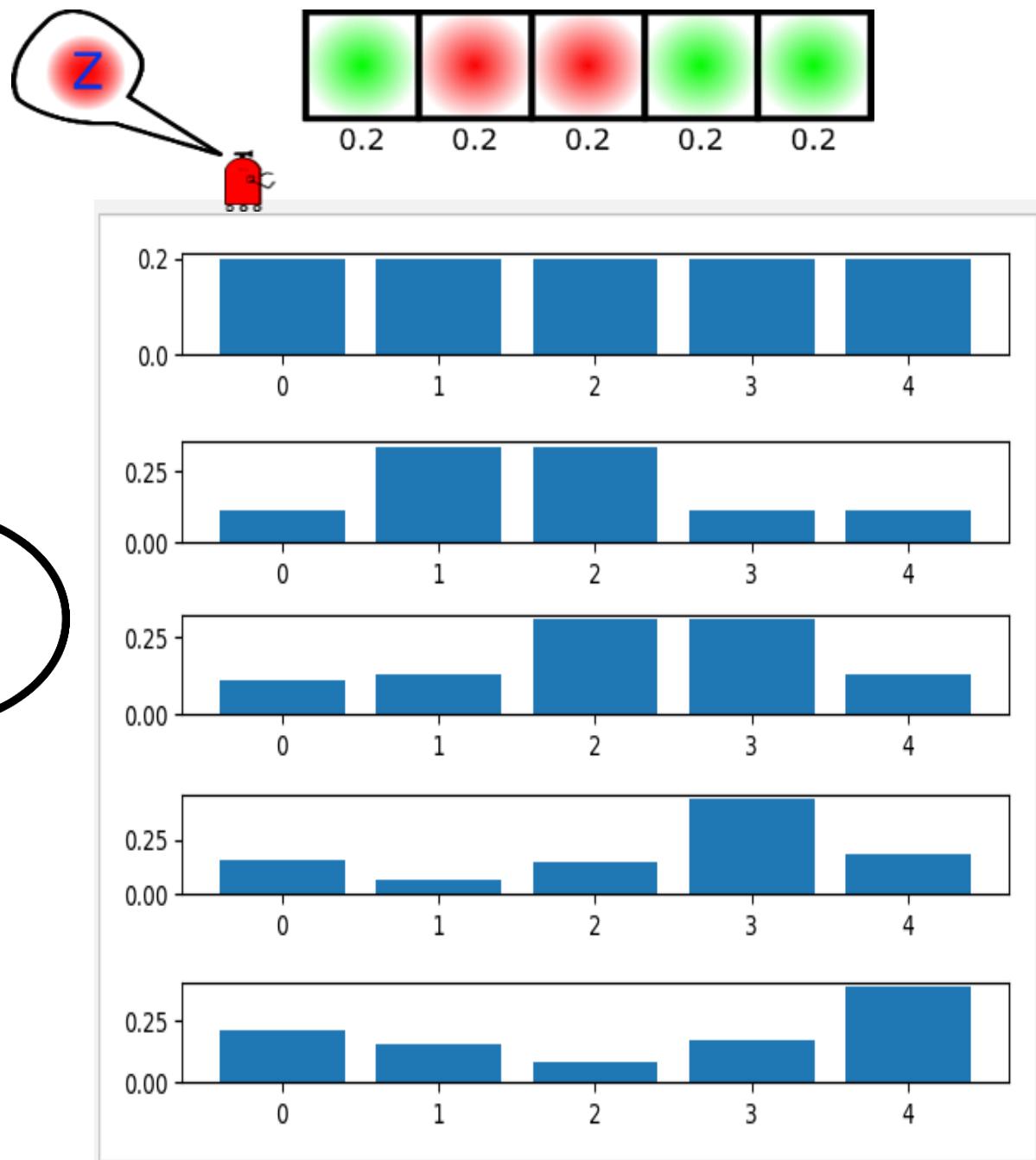
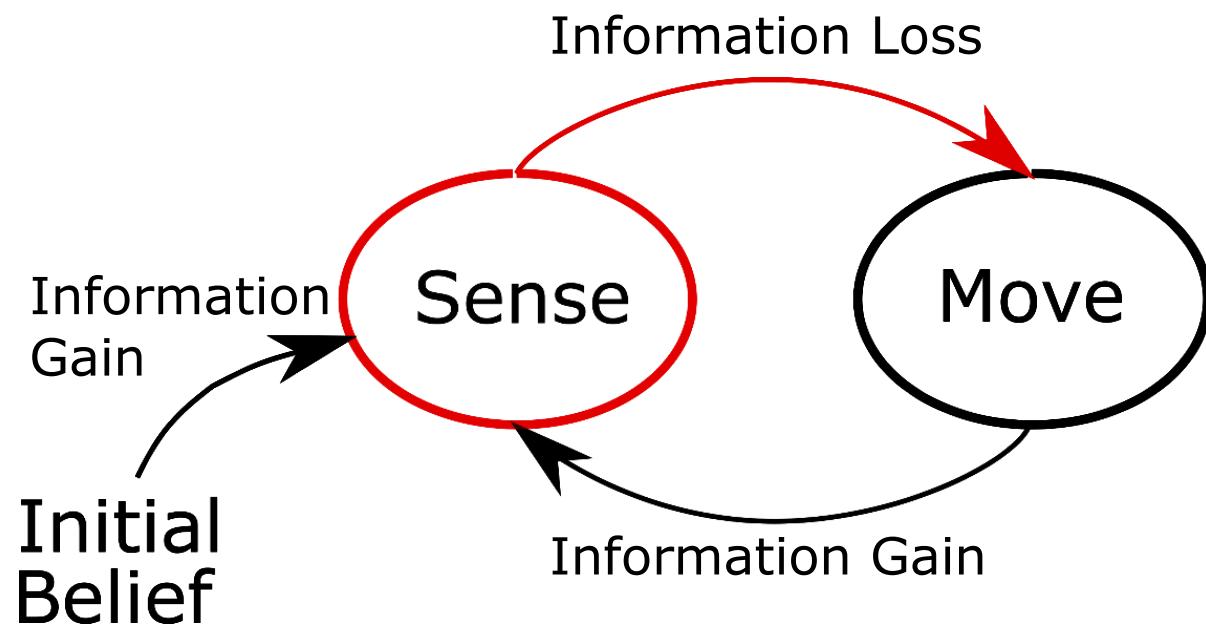
That's very good!!!!

- Localization is just a sense/move cycle:



That's very good!!!!

- Localization is just a sense/move cycle:



Sum Up Global Localization

- Belief → Probability
- Measurements → Multiplication followed by Normalization
- Moving → Convolution

Belief – Formal Definitions

- Probability:

$$0 \leq p(X) \leq 1$$

Belief – Formal Definitions

- Probability:

$$0 \leq p(X) \leq 1$$

- Assuming 2 states:

$$p(X_1) = 0.2$$

$$p(X_2) = \underline{\hspace{2cm}}$$

Belief – Formal Definitions

- Probability:

$$0 \leq p(X) \leq 1$$

- Assuming 2 states:

$$p(X_1) = 0.2$$

$$p(X_2) = \underline{0.8}$$

Belief – Formal Definitions

- Probability:

$$0 \leq p(X) \leq 1$$

- Assuming 2 states:

$$p(X_1) = 0.2$$

$$p(X_2) = \underline{0.8}$$

- Assuming 5 states:

0.1	0.1	0.1	0.1	
-----	-----	-----	-----	--

Belief – Formal Definitions

- Probability:

$$0 \leq p(X) \leq 1$$

- Assuming 2 states:

$$p(X_1) = 0.2$$

$$p(X_2) = \underline{0.8}$$

- Assuming 5 states:

0.1	0.1	0.1	0.1	0.6
-----	-----	-----	-----	-----

Measurement – Formal Definitions

- Bayes Rule
- Assuming a grid cell and the measurements:

X grid cell Z measurement

- The belief of the location given a measurement:

$$p(X_i | Z) = \text{_____}$$

Measurement – Formal Definitions

- Bayes Rule
- Assuming a grid cell and the measurements:

X grid cell Z measurement

- The belief of the location given a measurement:

$$p(X_i|Z) = \frac{p(Z|X_i)p(X_i)}{p(Z)}$$

Measurement Probability *Prior*

- A product of the prior with the measurement probability
- The “probability of seeing a measurement independently of location” (normalizer...)

Movement – Formal Definitions

- This is a somewhat complicated formula:
- Notice:
 - Grid Location
 - Time
- Here are the components:
 - Prior
 - Movement

$$p(X_i^t) = \sum_j p(X_j^{t-1}) p(X_i | X_j)$$

Time

Grid Location

Prior Probability

Movement Probability

Movement – Formal Definitions

- This is a somewhat complicated formula:
- Notice:
 - Grid Location
 - Time
- Here are the components:
 - Prior
 - Movement
- This is what is called Total Probability

$$p(X_i^t) = \sum_j p(X_j^{t-1}) p(X_i | X_j)$$

Time
Grid Location

Prior Probability *Movement Probability*

$$\Pr(A) = \sum_n \Pr(A | B_n) \Pr(B_n),$$

Histogram-based State Estimation

Main histogram-based global localization problem (Markov Localization):

- Memory scaling is exponential
- So, it is unfeasible in large real world problems.

Sum-UP so far

- State Estimation
- Markov Localization
- Probability
- Bayes
- Total Probability

Coming UP:

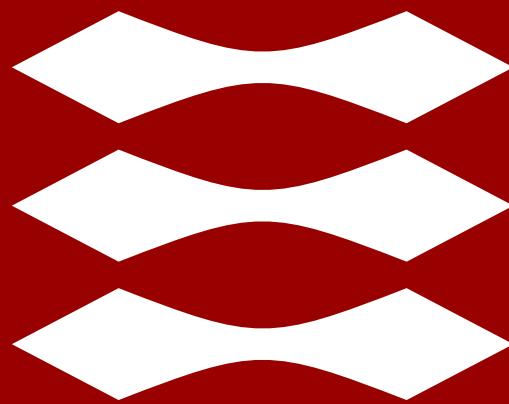
- Kalman Filter

Perception for Autonomous Systems 31392:

State Estimation - Histogram Filter

Lecturer: Evangelos Boukas—PhD

DTU



Perception for Autonomous Systems 31392:

State Estimation - Kalman Filter

Lecturer: Evangelos Boukas—PhD

Sum-UP so far

- State Estimation
- Markov Localization
- Probability
- Bayes
- Total Probability

Coming UP:

- Kalman Filter

What is State Estimation

Goal:

- Given a State Vector of a system
- Estimate over time the state using input of external sensors

Useful for:

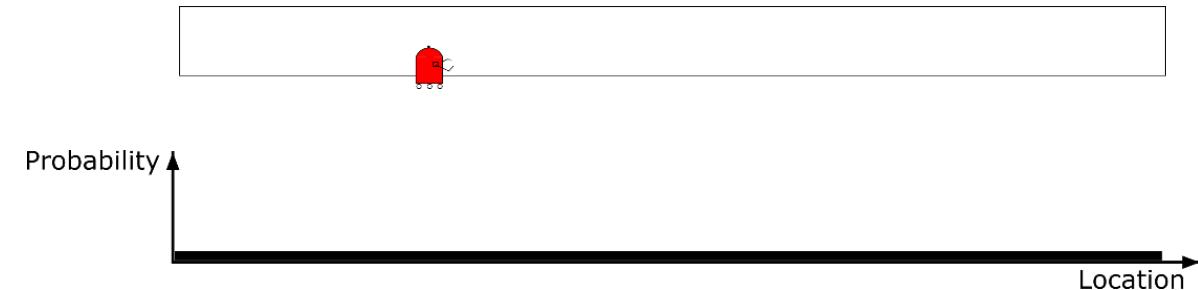
- Localization
- Tracking
- Prediction
- Sensor Fusion
- ...

Catching up

- Last part, we did state estimation specifically Localization

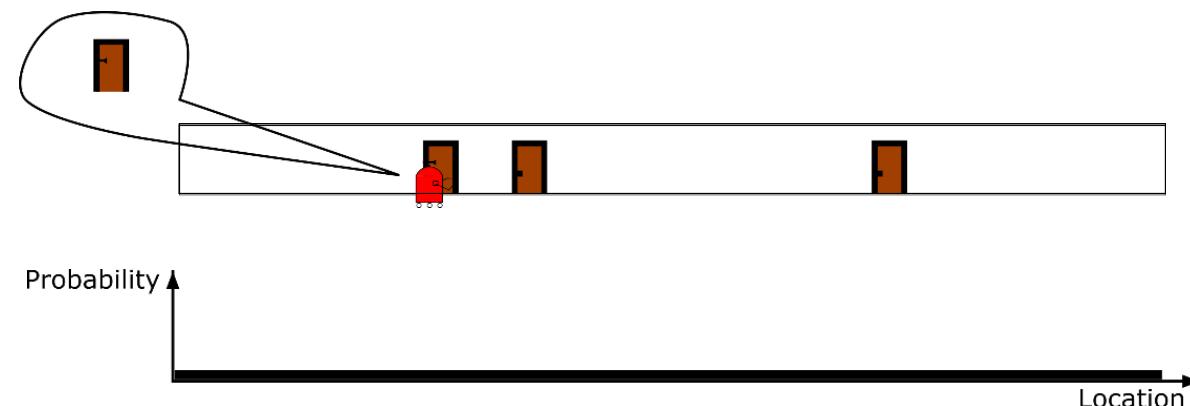
Catching up

- Last part, we did state estimation specifically Localization



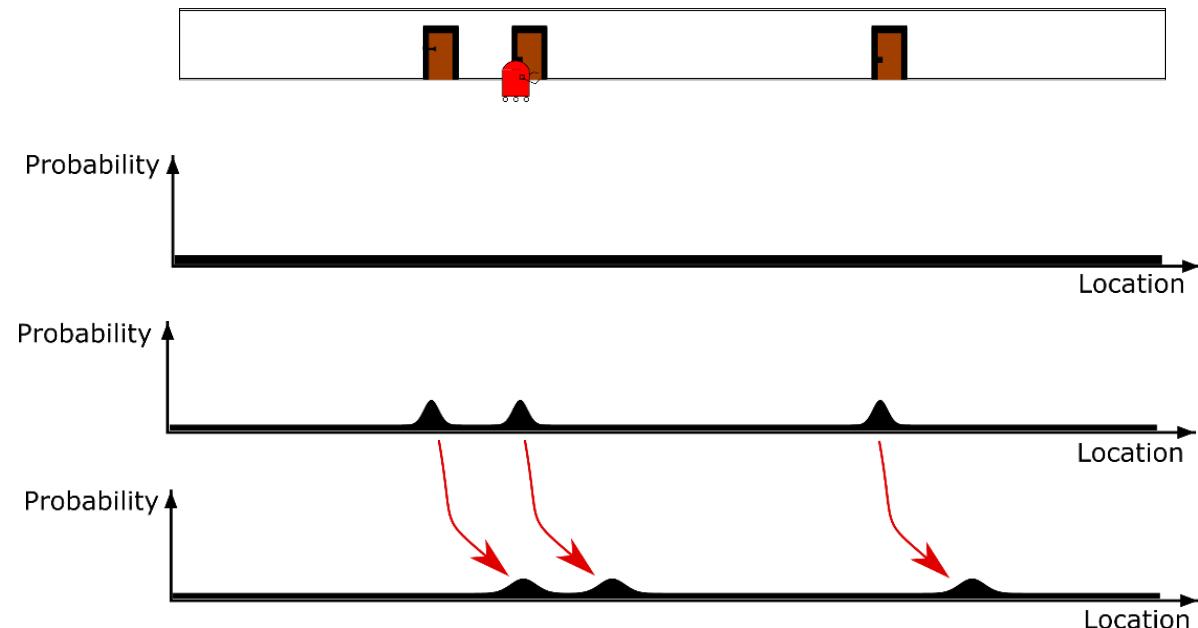
Catching up

- Last part, we did state estimation specifically Localization



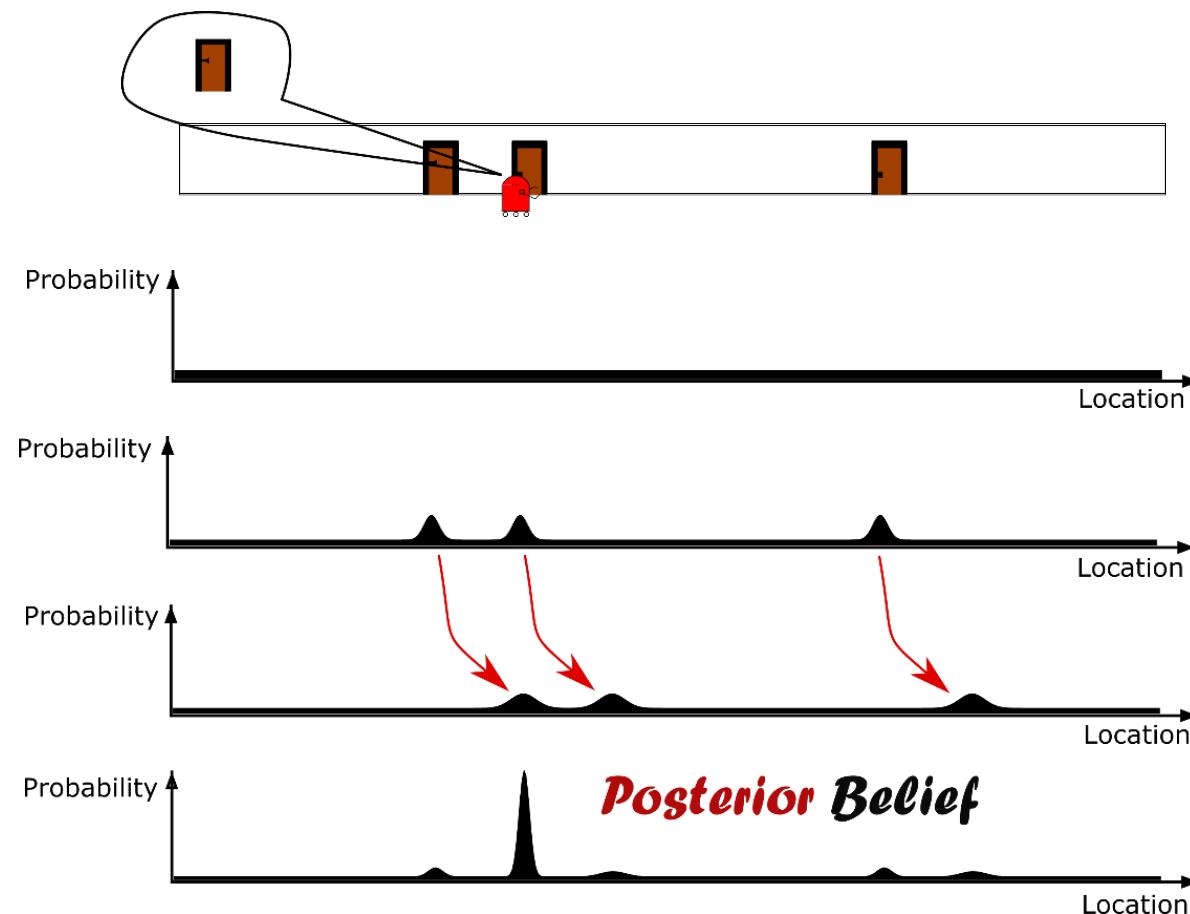
Catching up

- Last part, we did state estimation specifically Localization



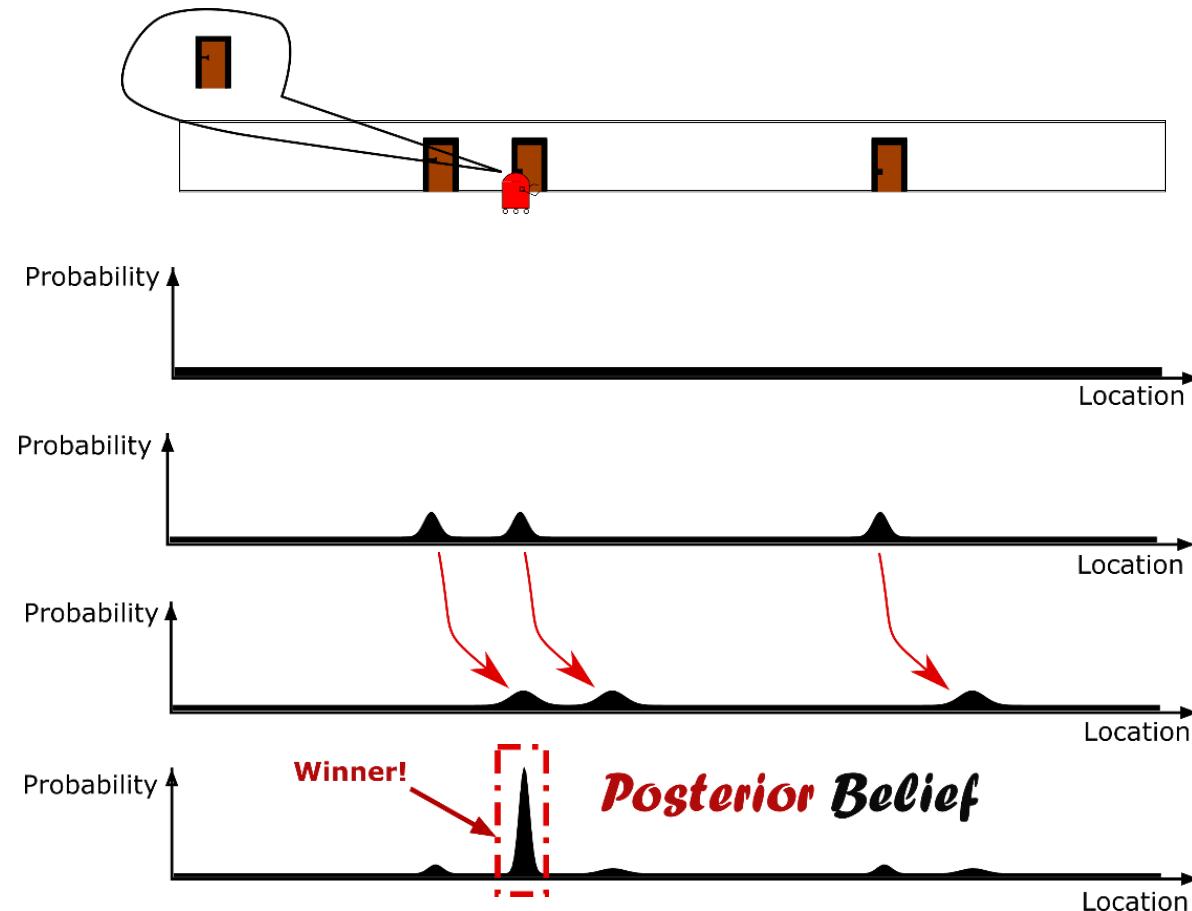
Catching up

- Last part, we did state estimation specifically Localization



Catching up

- Last part, we did state estimation specifically Localization



Catching up

- Last part, we did state estimation specifically Localization
- Maximum confusion to Location estimation

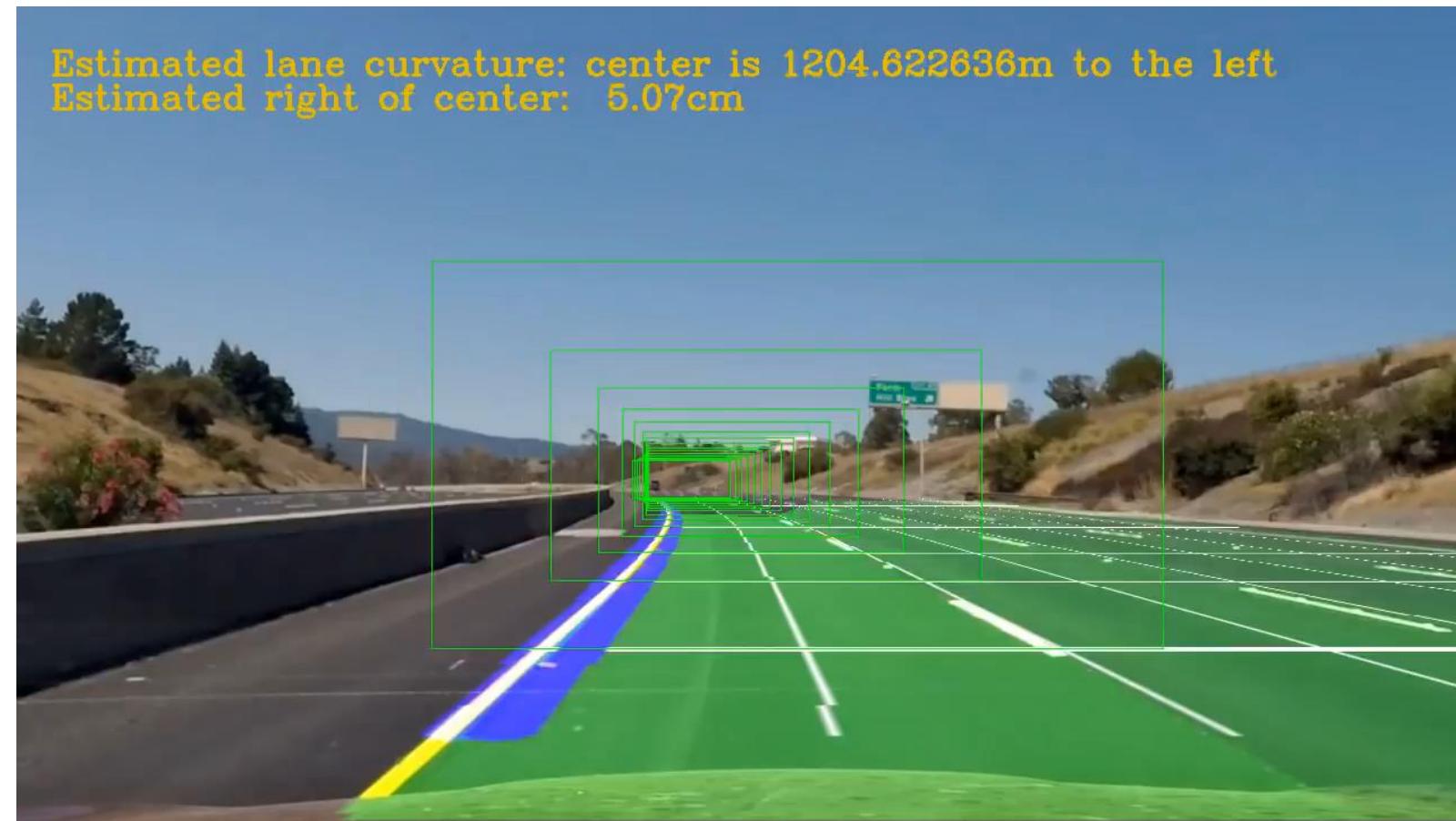
Catching up

- Last part, we did state estimation specifically Localization
- Maximum confusion to Location estimation
- In other words:
“Used sensor information to manipulate an original belief (a uniform distribution) into a high confidence probability density function centered on our correct location”

In this part..

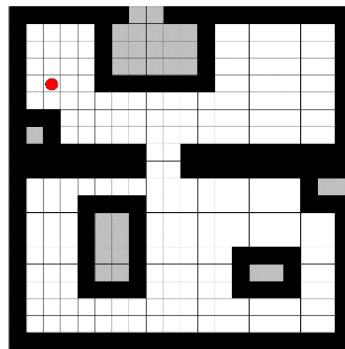
- We will see how we can track objects.
 - Not only their location as in the previous localization case,
 - But also infer their speed.
-
- In the case of autonomous driving it is quite important to track and predict the movement of objects.
 - Why?
 - Which other examples can we find?

Cases of Tracking

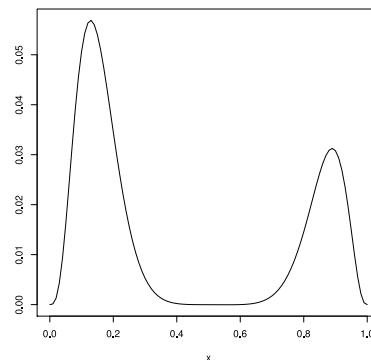


Differences between state estimation filters

Histogram filter

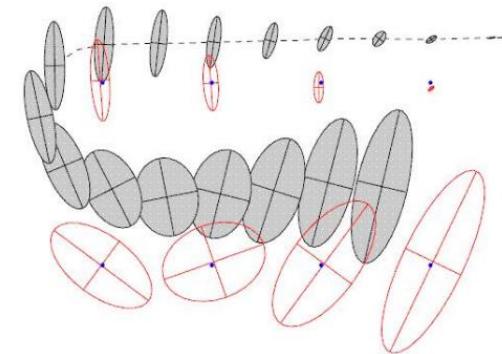


Discrete

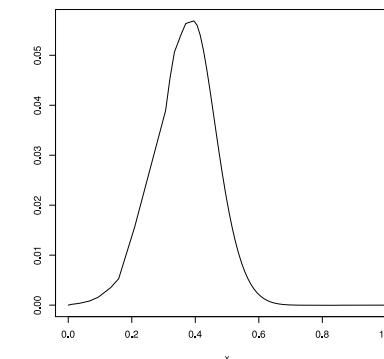


Multimodal

Kalman Filter



Continuous



Unimodal

Let's see tracking as an example

- Assuming there is a point in space like this:
- The object moves like this:



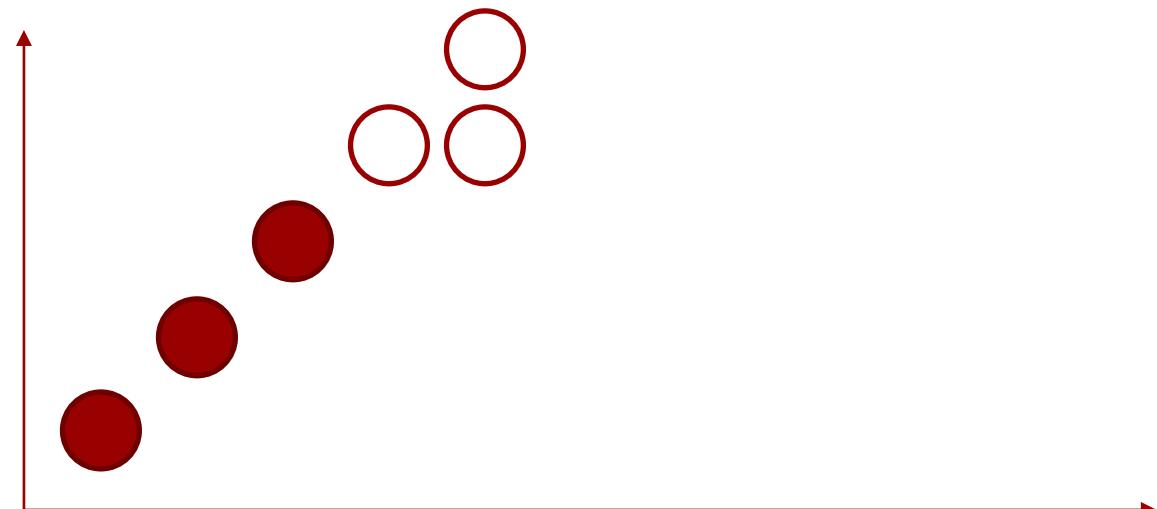
Let's see tracking as an example

- Assuming there is a point in space like this:
- The object moves like this:
- What is the next point?



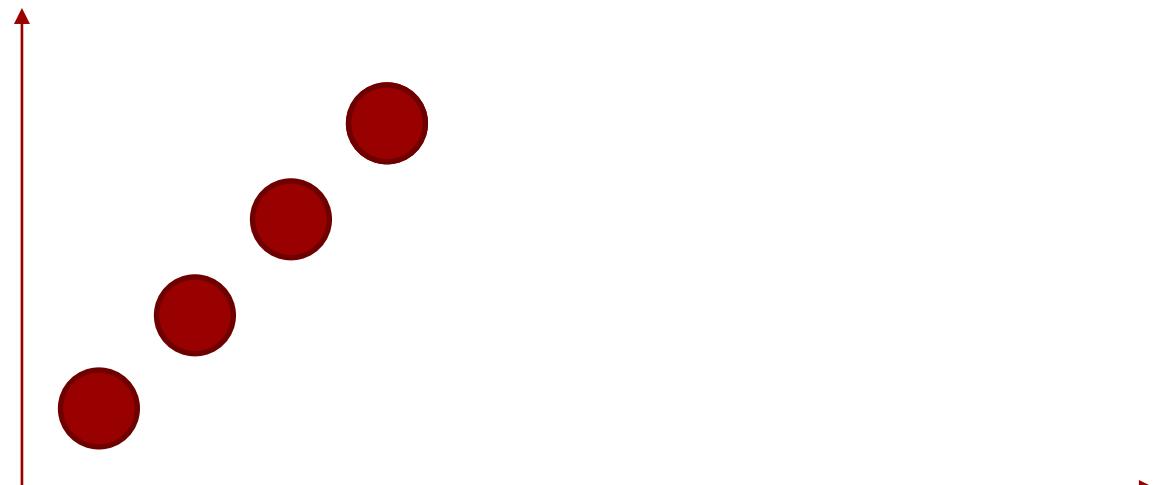
Let's see tracking as an example

- Assuming there is a point in space like this:
- The object moves like this:
- What is the next point?



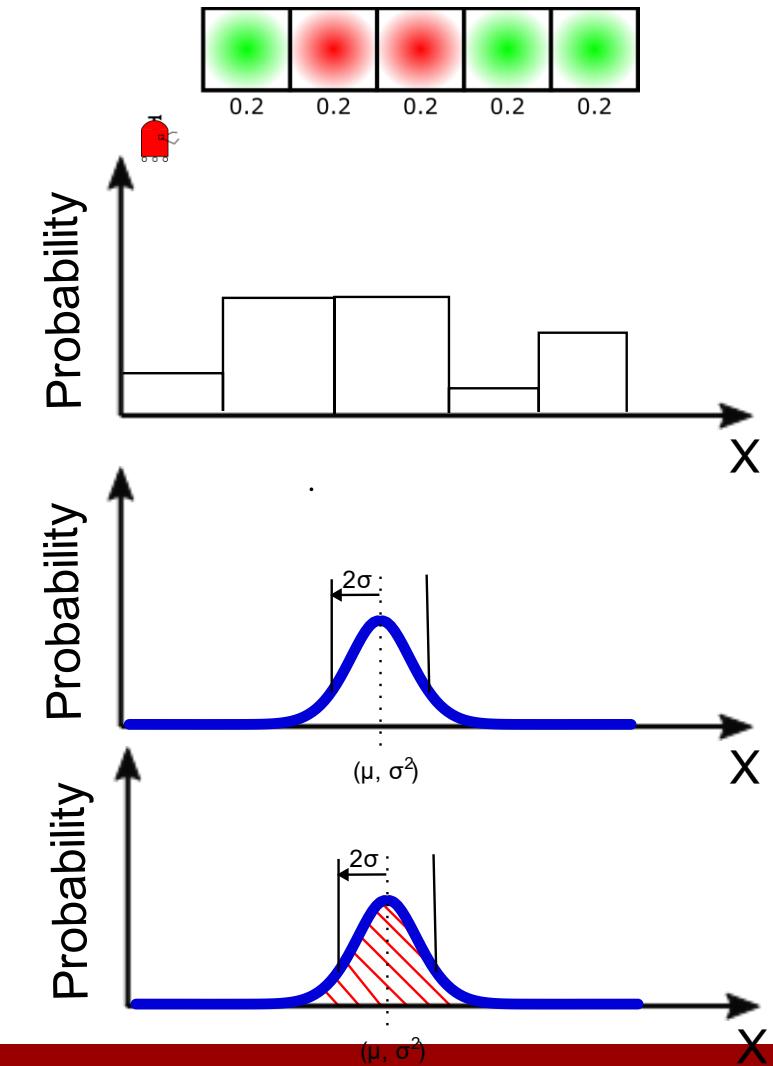
Let's see tracking as an example

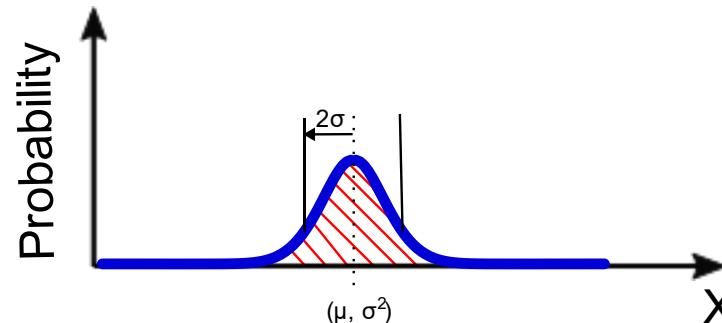
- Assuming there is a point in space like this:
 - The object moves like this:
 - What is the next point?
-
- For you it is easy! How about a machine?



Kalman has a thing for Gaussians

- In our Markov model the world was divided into discrete grids and each grid had a probability
- This is called a histogram:
- In Kalman Filter we describe the distribution as a Gaussian:
 - It is a continuous function and
 - The area under the curve is: 1

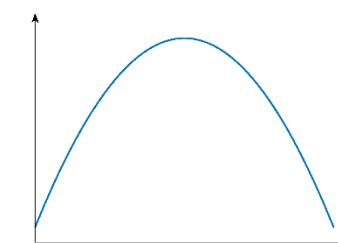
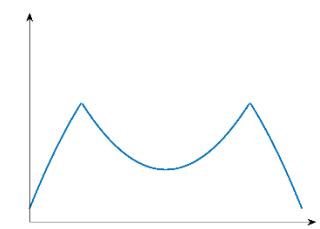
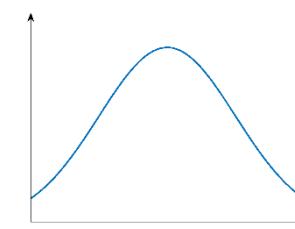
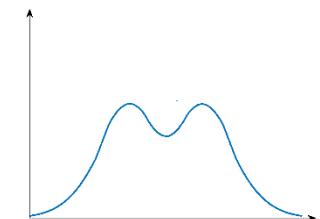
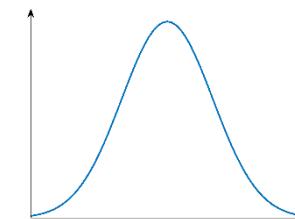




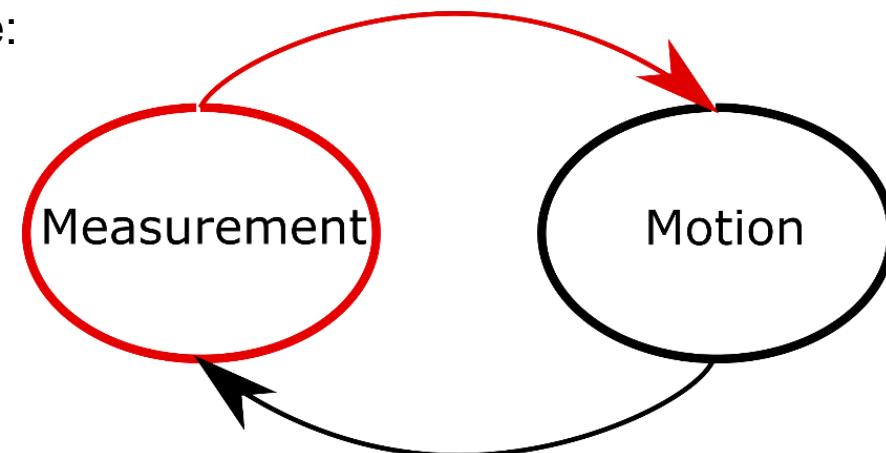
- 1D Gaussian is described by the pair: (μ, σ^2)

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

- Which of the following are Gaussians?
- Which of the following have small, medium, larger (co)variance?
- When doing state estimation which one do we prefer?

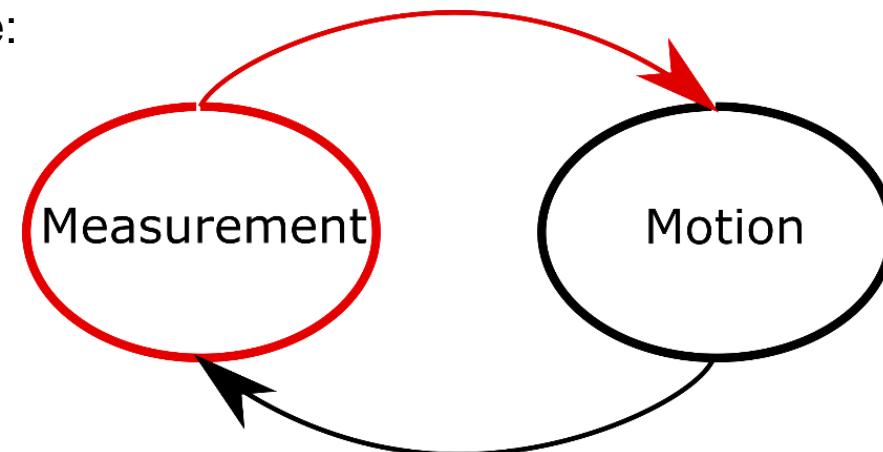


- Kalman, as with the histogram filter involves the measurement \Leftrightarrow motion cycle:



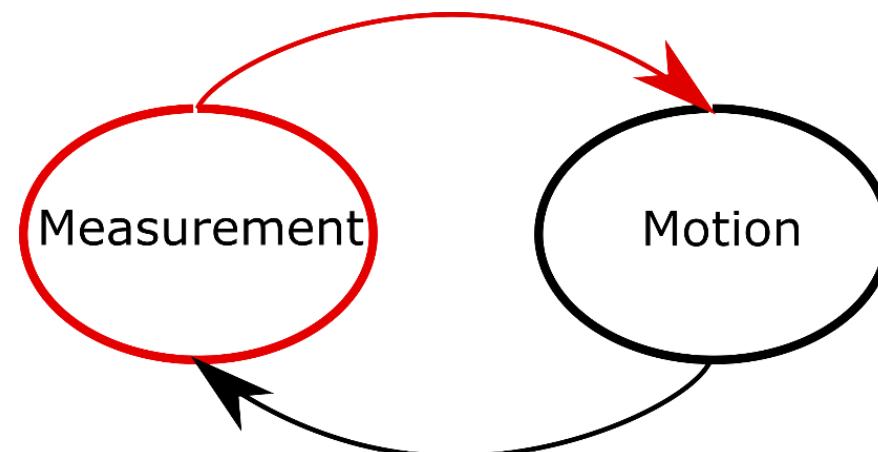
- Which one requires **convolution** and which one a **product**?

- Kalman, as with the histogram filter involves the measurement \Leftrightarrow motion cycle:



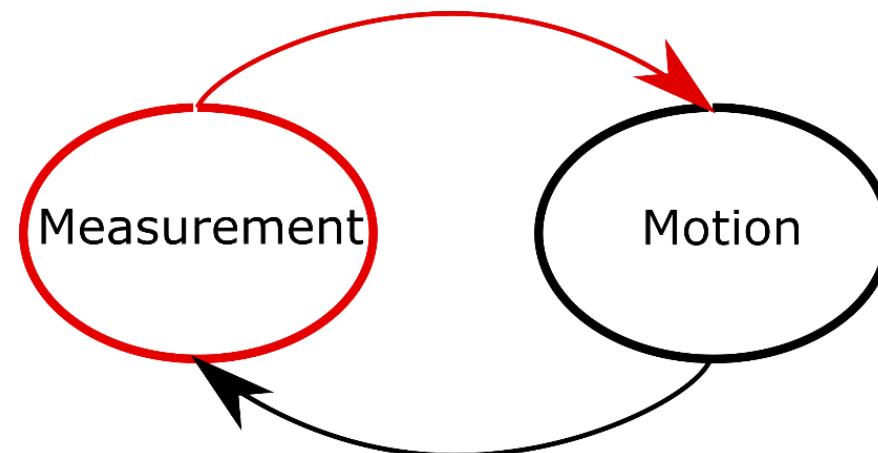
- Which one requires **convolution** and which one a **product**?
 - Measurement \Leftrightarrow Product
 - Motion \Leftrightarrow Convolution

- Kalman, as with the histogram filter involves the measurement \Leftrightarrow motion cycle:

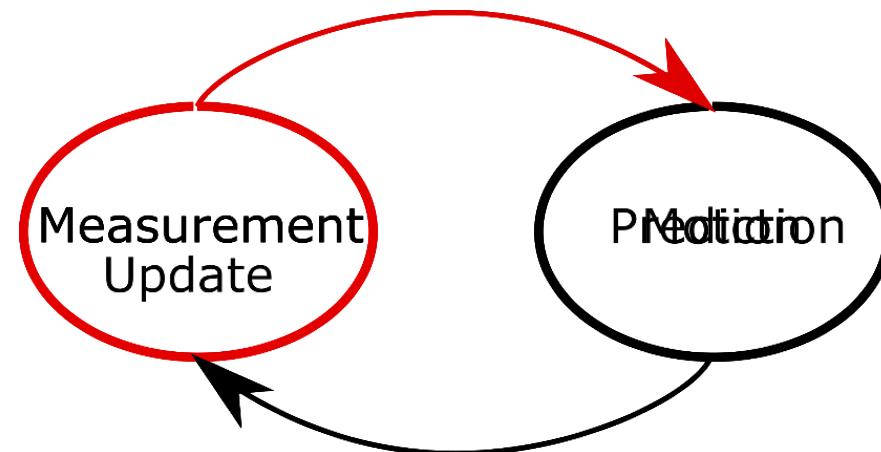


- Which one applies **Bayes Rule** and which one **Total Probability**?

- Kalman, as with the histogram filter involves the measurement \Leftrightarrow motion cycle:



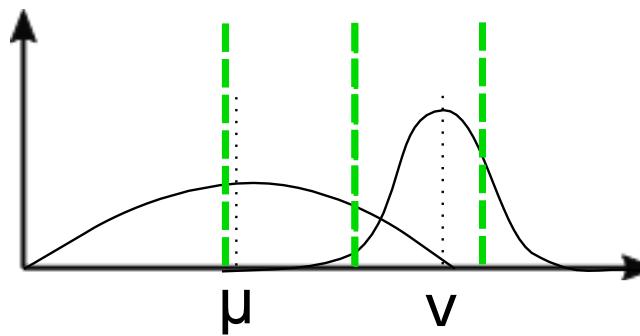
- Which one applies **Bayes Rule** and which one **Total Probability**?
 - Measurement \Leftrightarrow Bayes Rule
 - Motion \Leftrightarrow Total Probability



- In Kalman we call them “Measurement Update” and “Prediction”
- Both of these involve the Gaussians

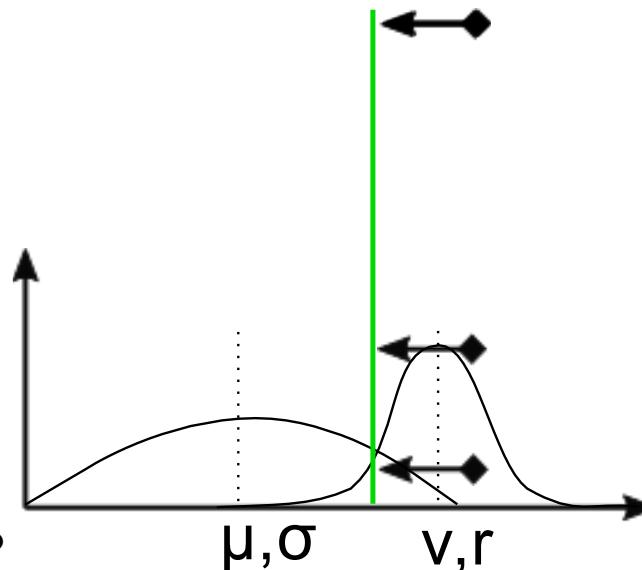
Kalman Measurement Update

- Assume we are localizing another robot with a prior as follows:



- Then we have a measurement which inform us that we have this location:
- Where will the new mean be?

Kalman Measurement Update

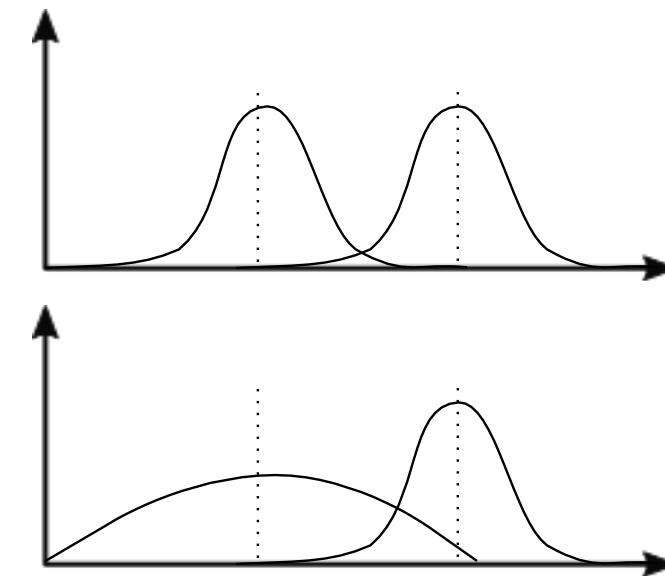
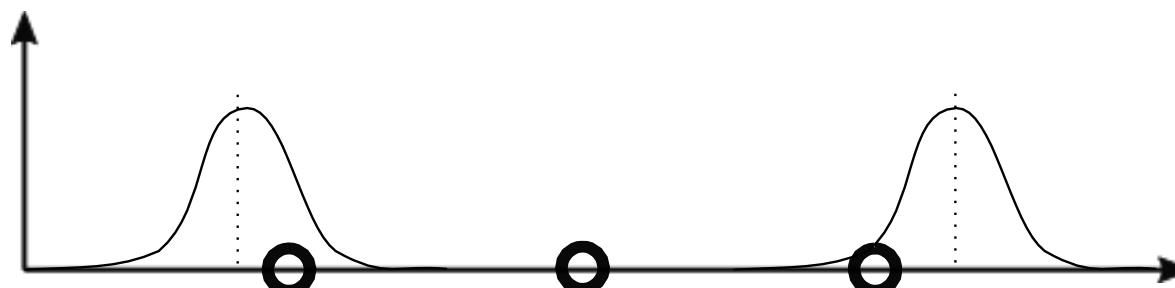


- Where will the new peak be?
- The higher one -> as we gain information
- Let's prove it

$$\mu' = \frac{r^2 \mu + \sigma^2 v}{r^2 + \sigma^2}$$

$$\sigma'^2 = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}}$$

- Assuming these gaussians:
 - $\mu = 10$, $\sigma^2 = 4$
 - $\nu = 12$, $r^2 = 4$
- Assuming these gaussians:
 - $\mu = 10$, $\sigma^2 = 8$
 - $\nu = 13$, $r^2 = 2$
- Assuming these gaussians:



Motion Update

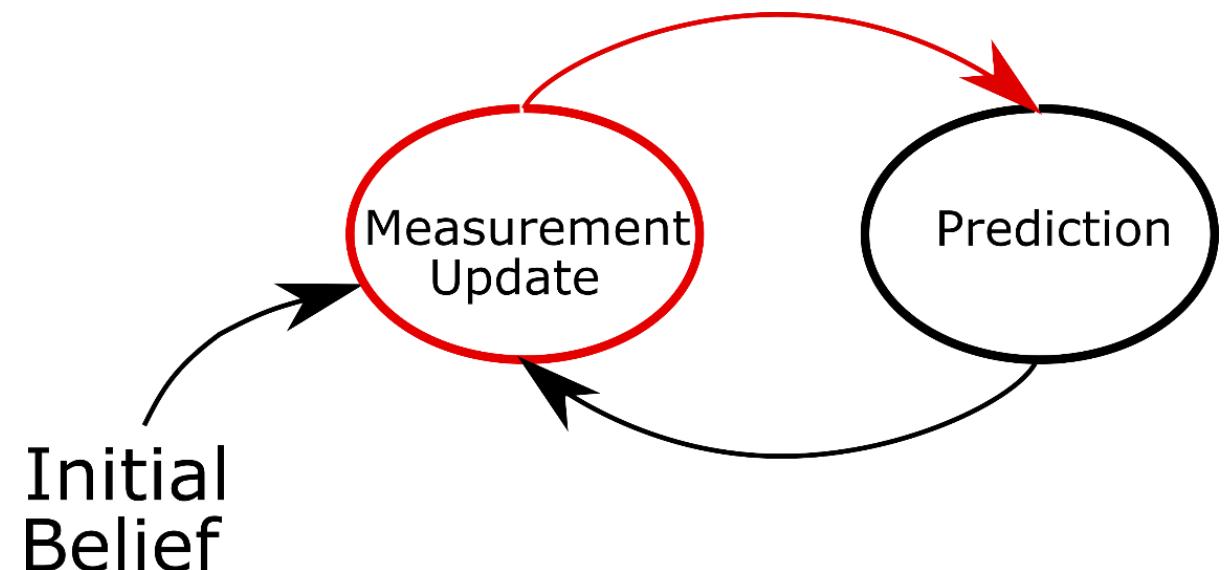
- Called also prediction:
- As we move we lose some information:



- Assuming a gaussian before the prediction:
 - $\mu = 8$, $\sigma^2 = 4$
- And a movement gaussian
 - $v = 10$, $r^2 = 6$
- What's the Gaussian after the update?

Let's code the 1D Kalman

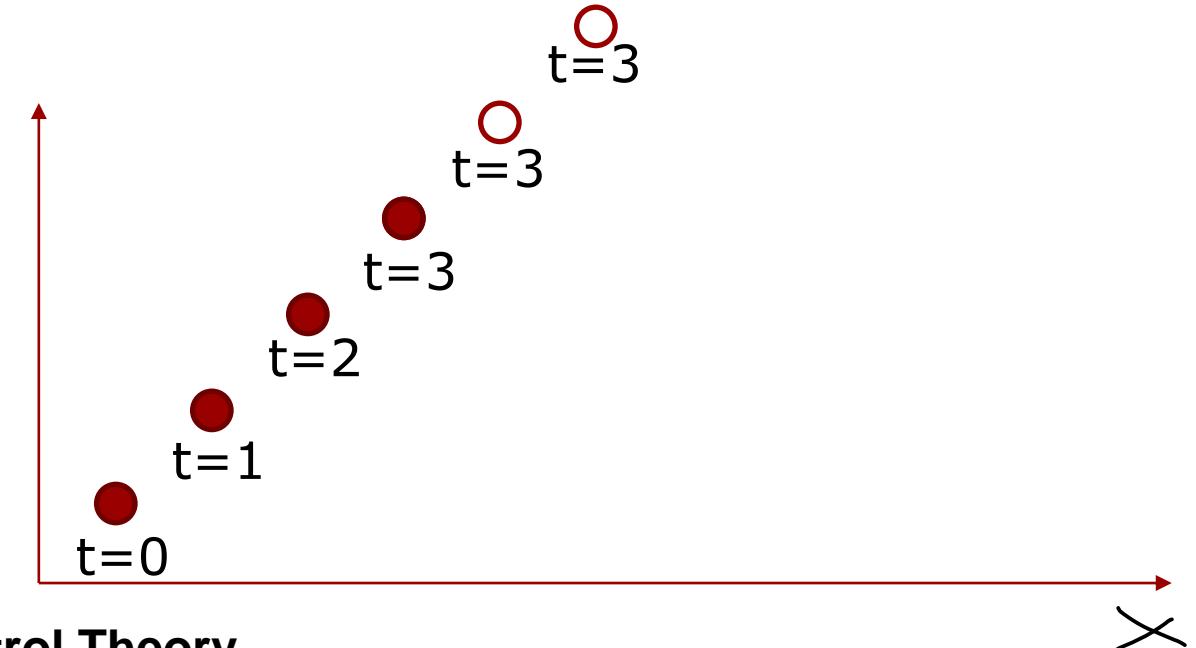
- Start with a initial belief of:
 - $u=0$,
 - $\sigma^2=10000$
- Motion:
 - $[1, 1, 2, 1, 1]$
 - Uncertainty: 2
- Measurement:
 - $[5, 6, 7, 9, 10]$
 - Uncertainty: 4



From 1D to Many D's

- We just implemented a Full 1D Kalman filter.
- However the Kalman Filter shines in Many D's
- Let's see an example:
 - A camera
 - Or a pedestrian
in front of a car
 - Where should it be at
 $t=3$?
- That is the power of Kalman!!!

→ AI and Control Theory



Multivariate Gaussians

$$\frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}}$$

As promised we have married the Gaussians today

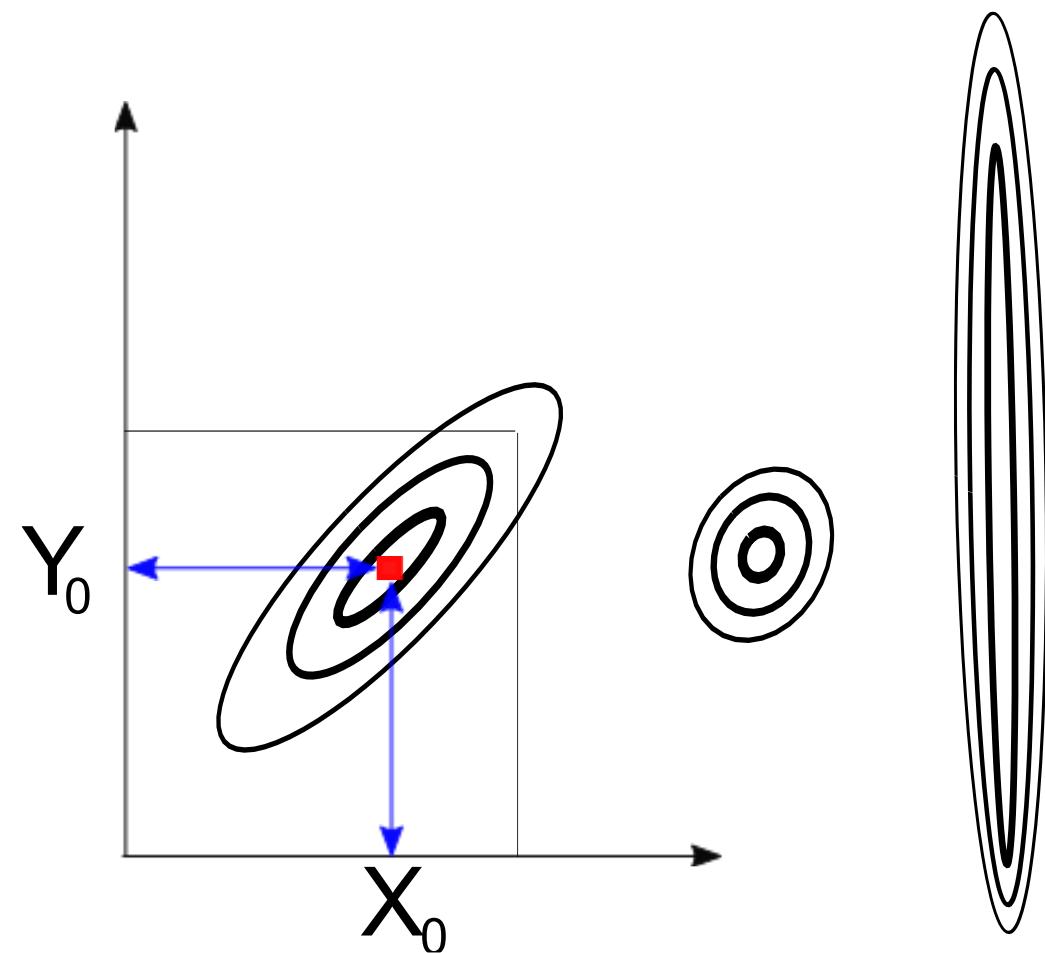
Multi-Dimensional Gaussians

- The mean is a vector:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \vdots \\ \boldsymbol{\mu}_D \end{pmatrix}$$

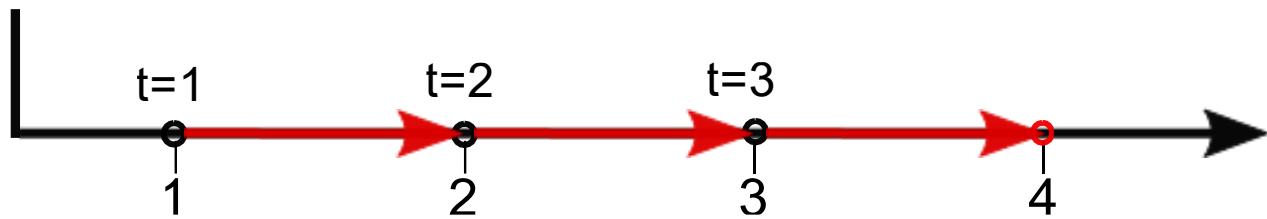
- The variance now is called covariance and is a matrix:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \ddiamond & \ddiamond & \ddiamond & \ddiamond & \ddiamond \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

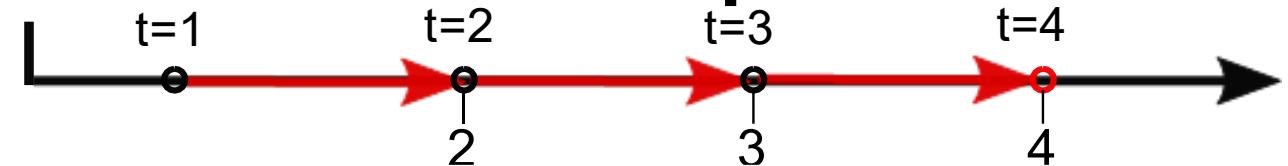


Multivariate Gaussians

- Let's start with an one dimensional motion example:



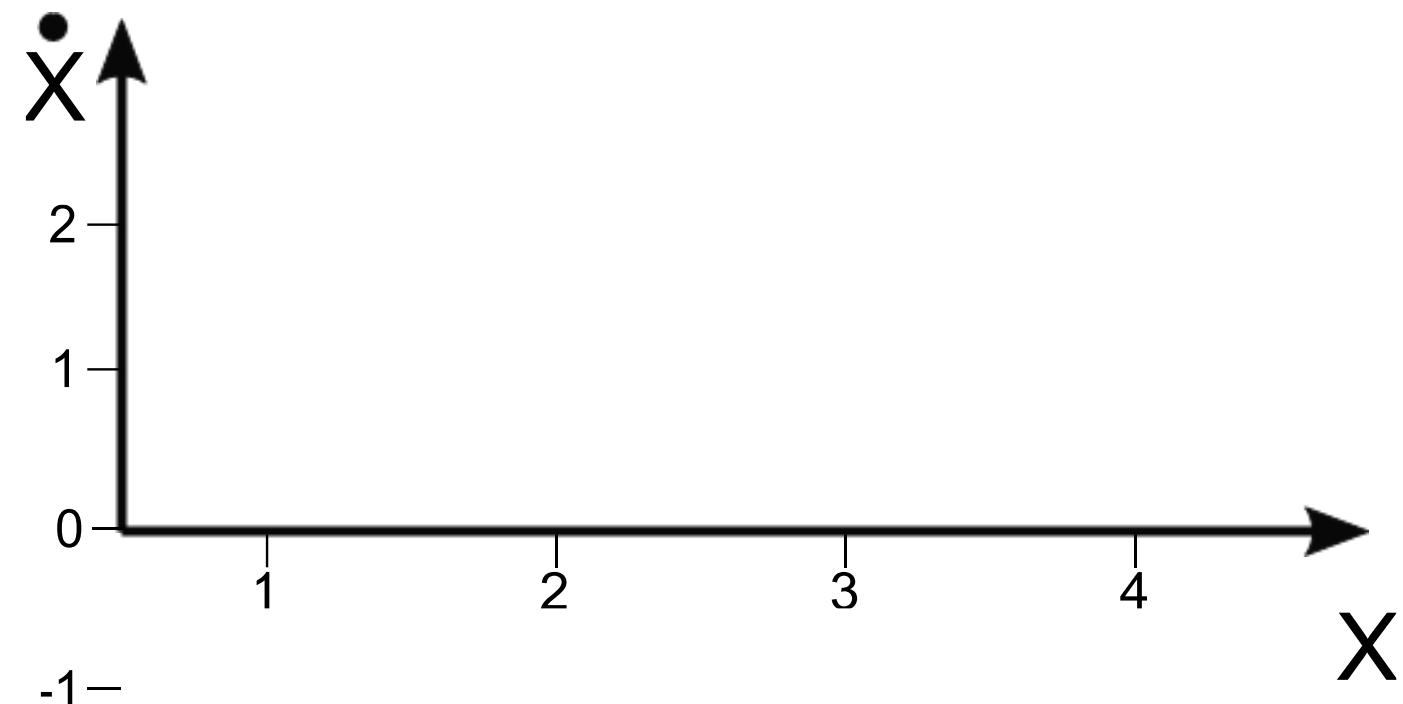
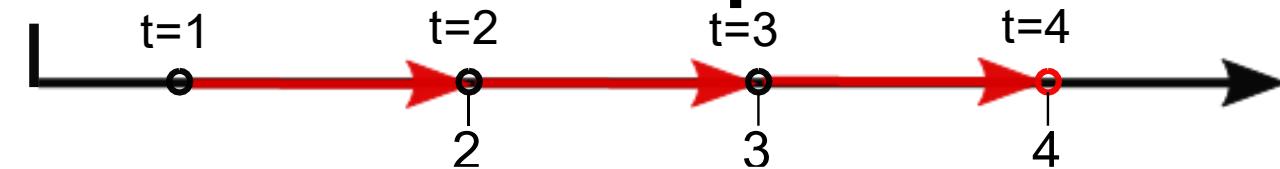
Multivariate Gaussians - Kalman State Space



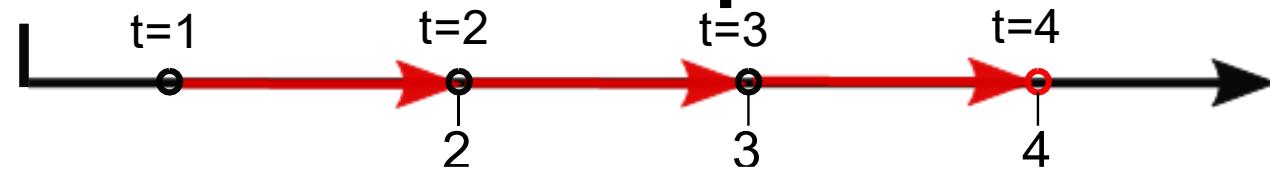
- Let's go at the Kalman state space:

Multivariate Gaussians - Kalman State Space

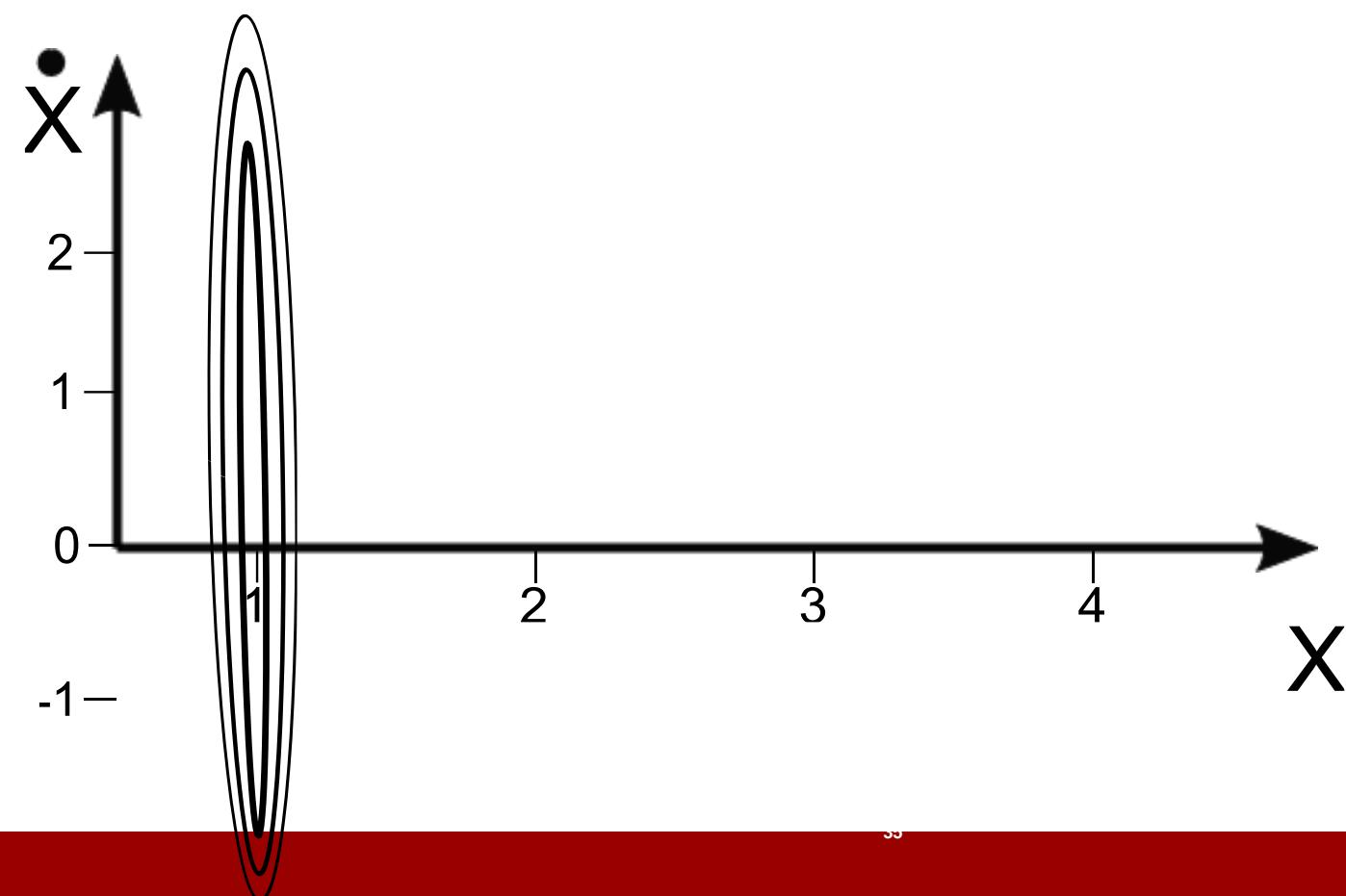
- Let's go at the Kalman state space:



Multivariate Gaussians - Kalman State Space

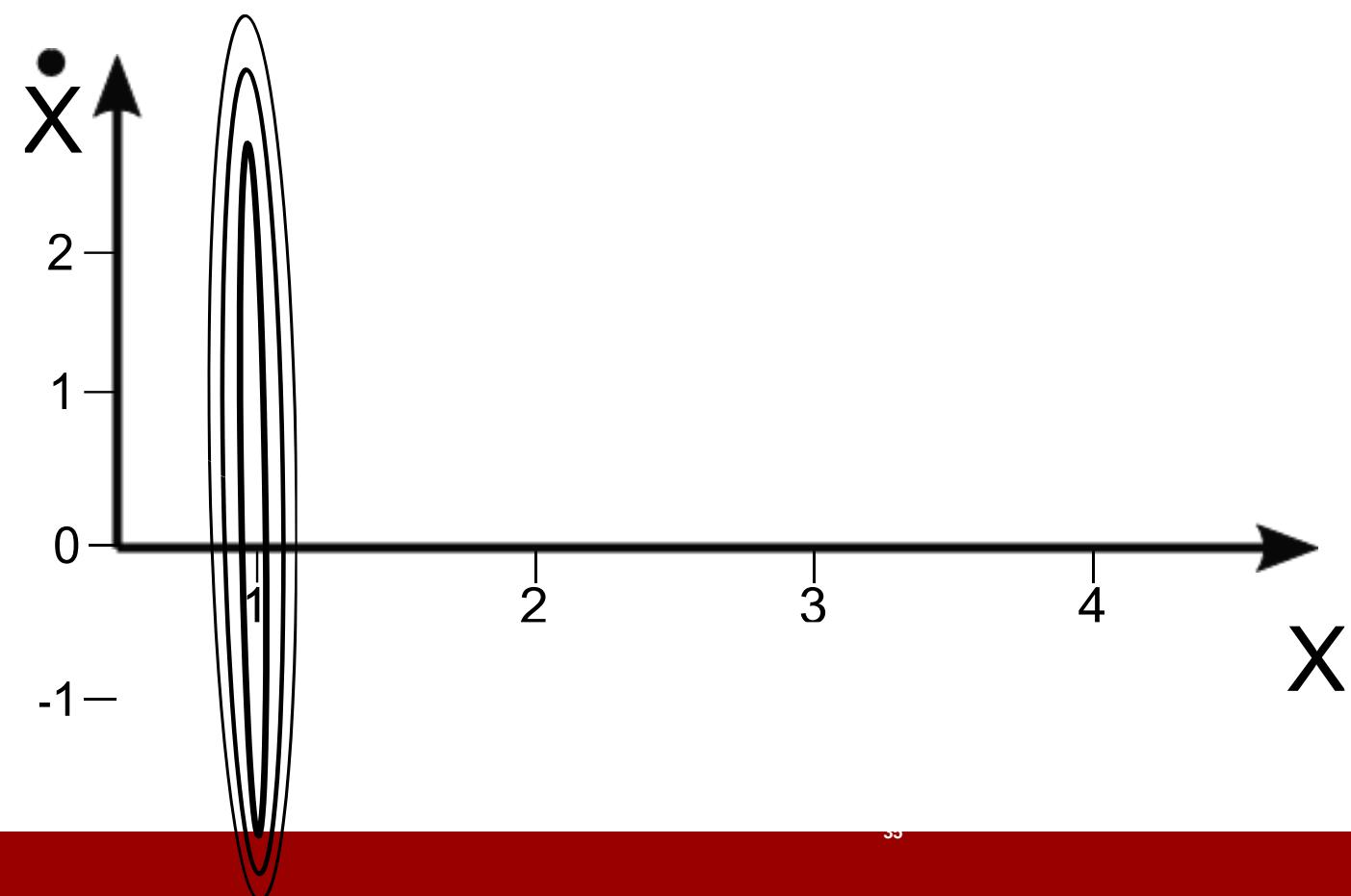
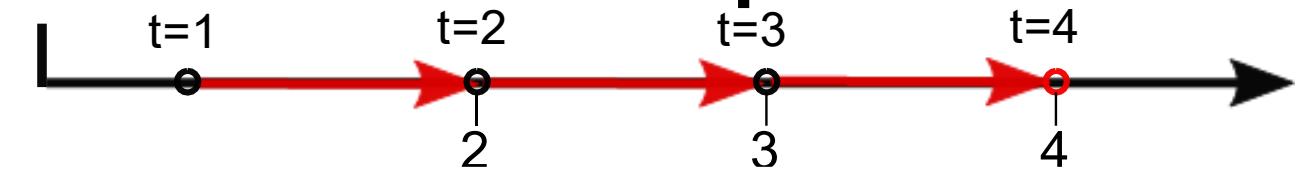


- Let's go at the Kalman state space:
- Prior:
 - Location: 1



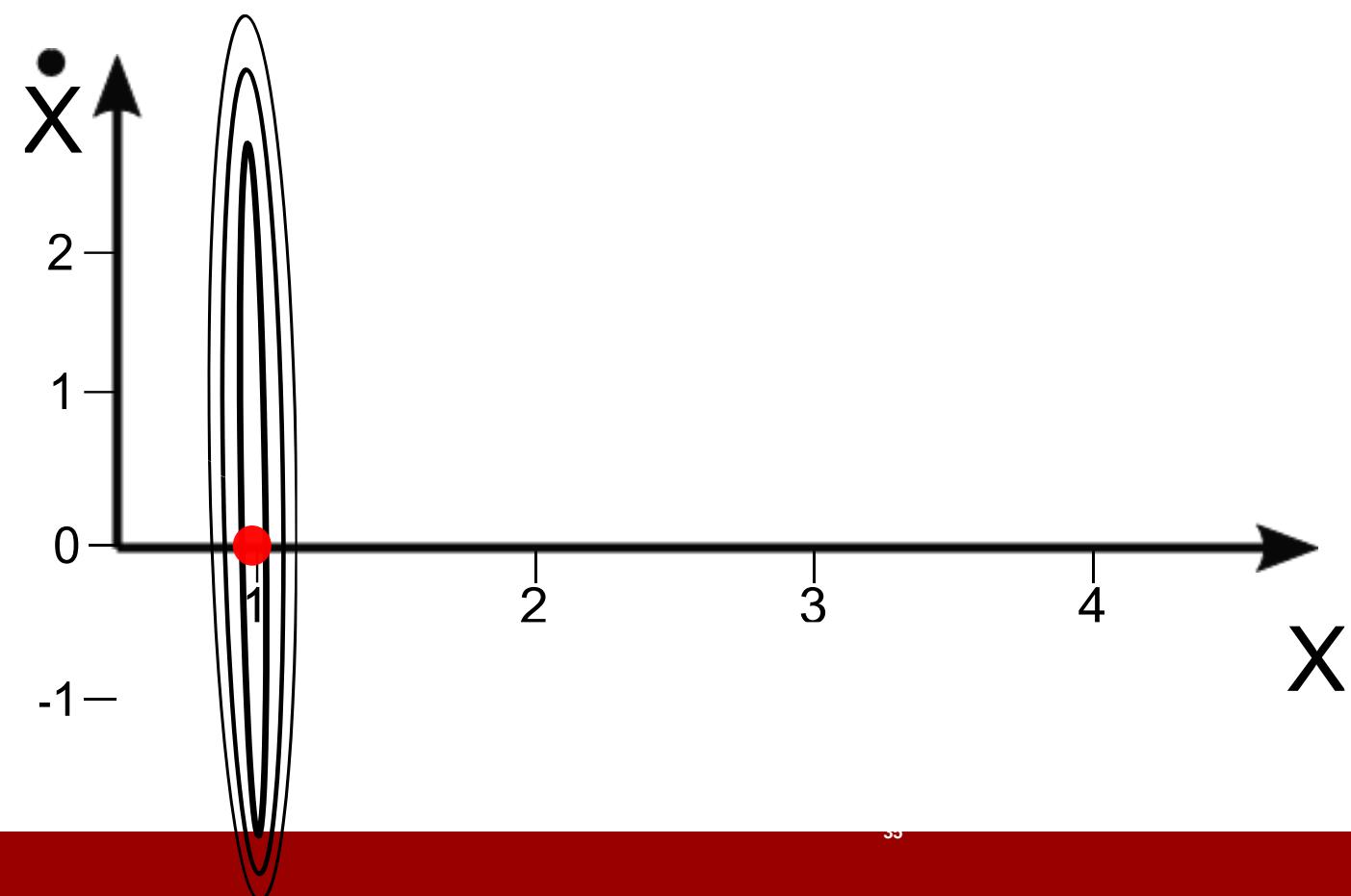
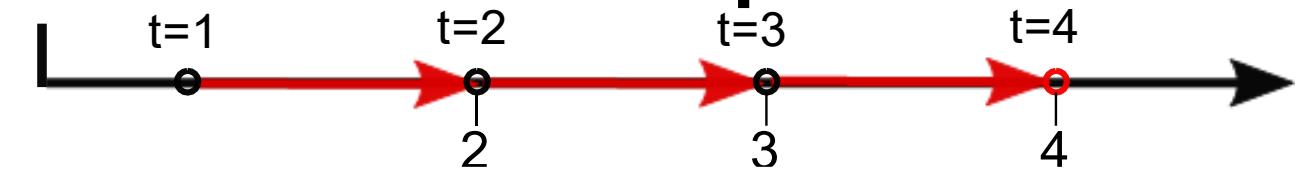
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0



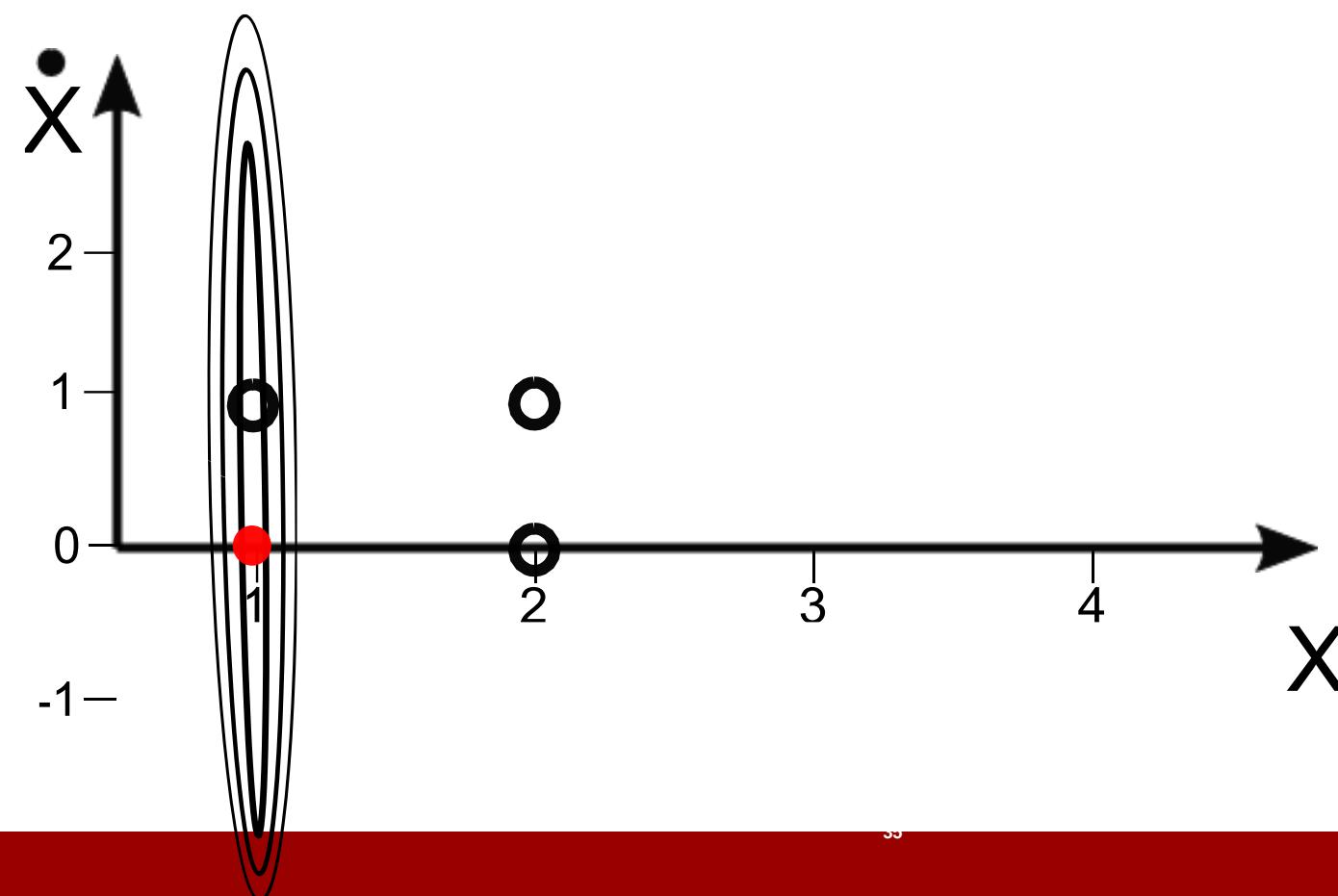
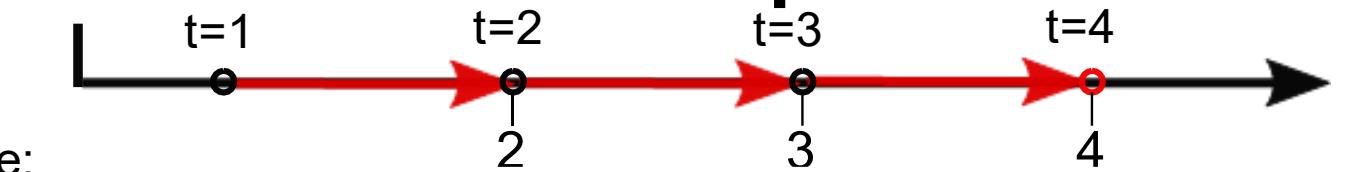
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0



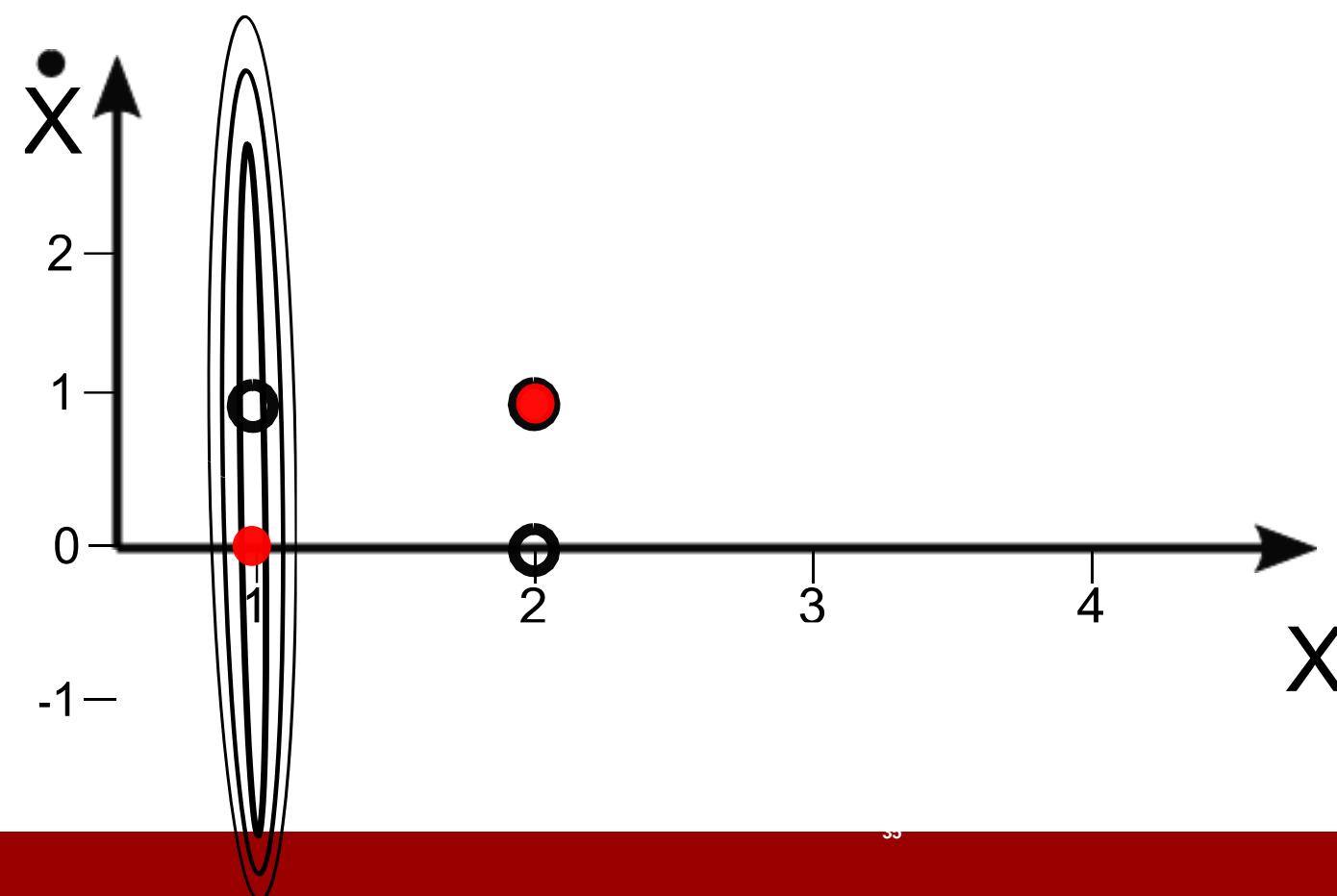
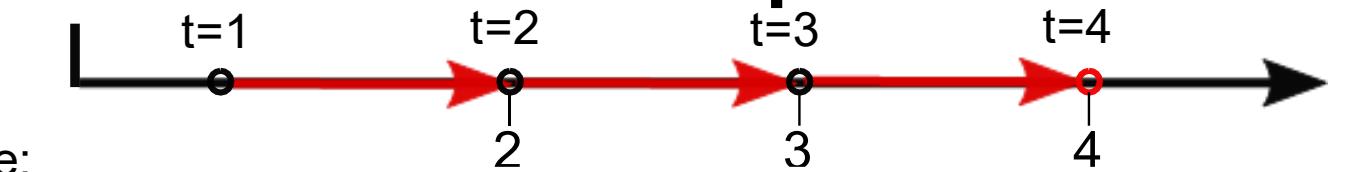
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1



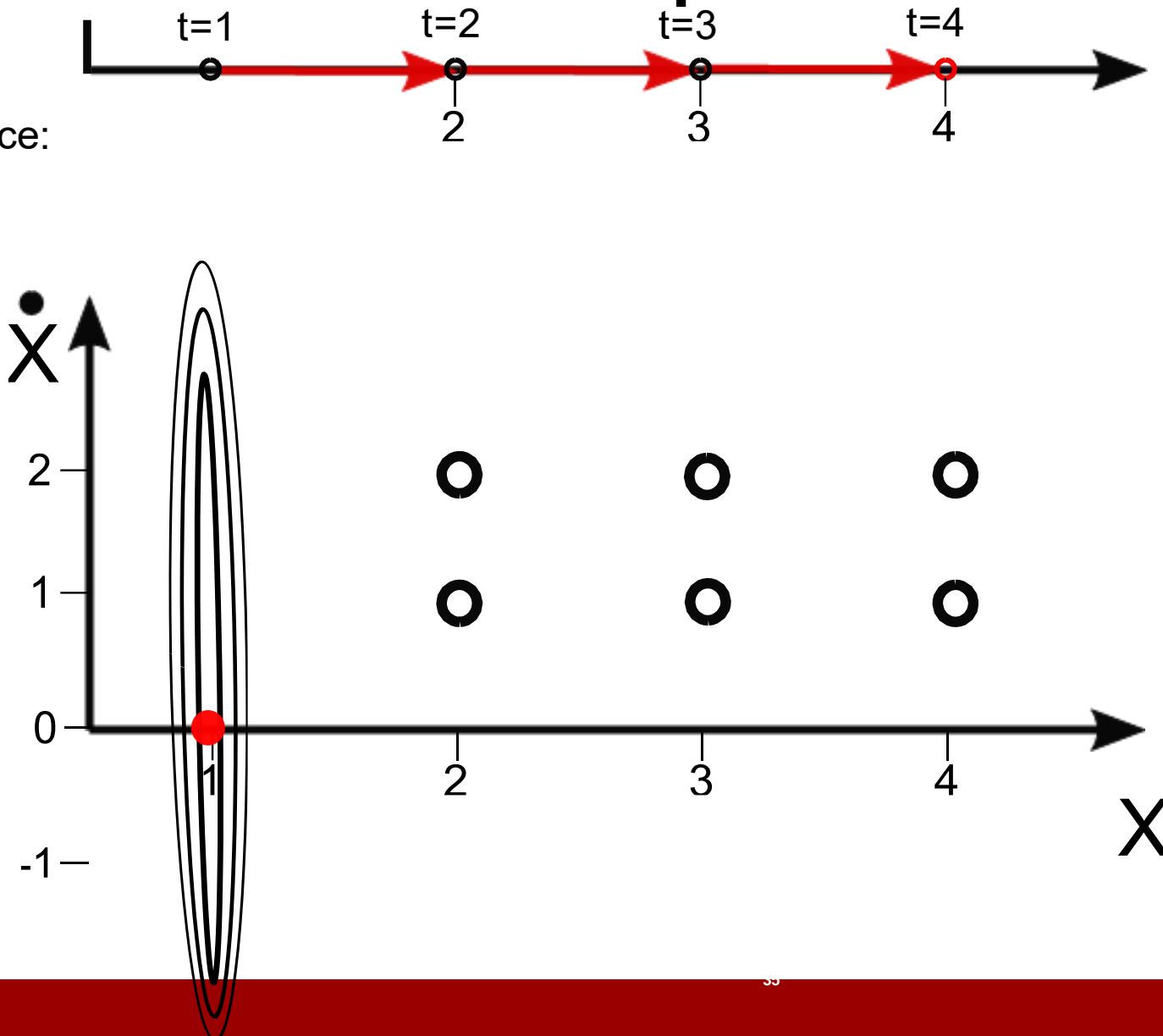
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1



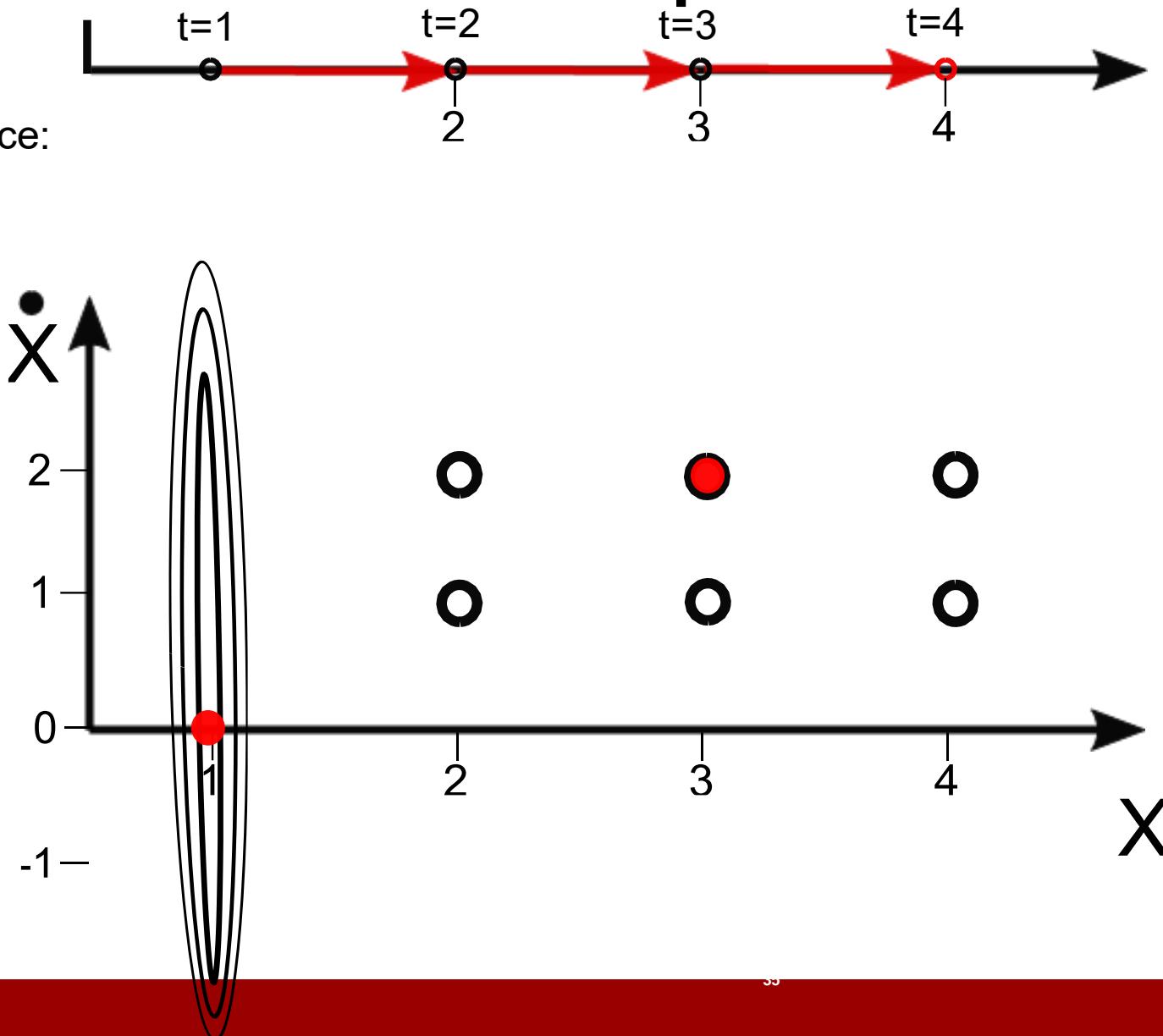
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2



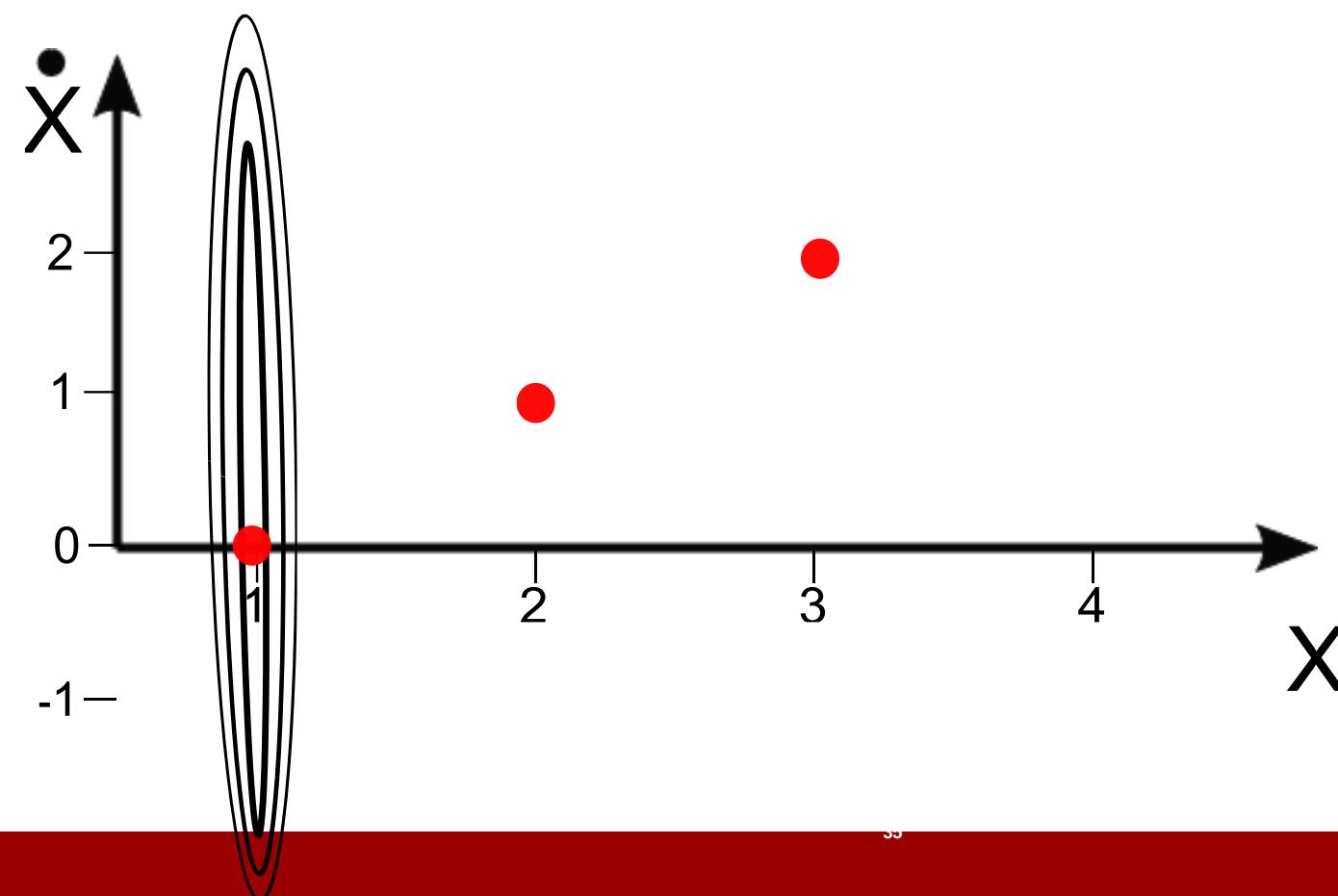
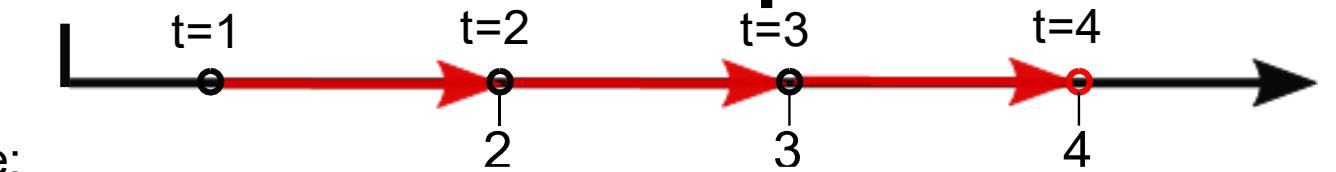
Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2



Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2



Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:

- Prior:

- Location: 1

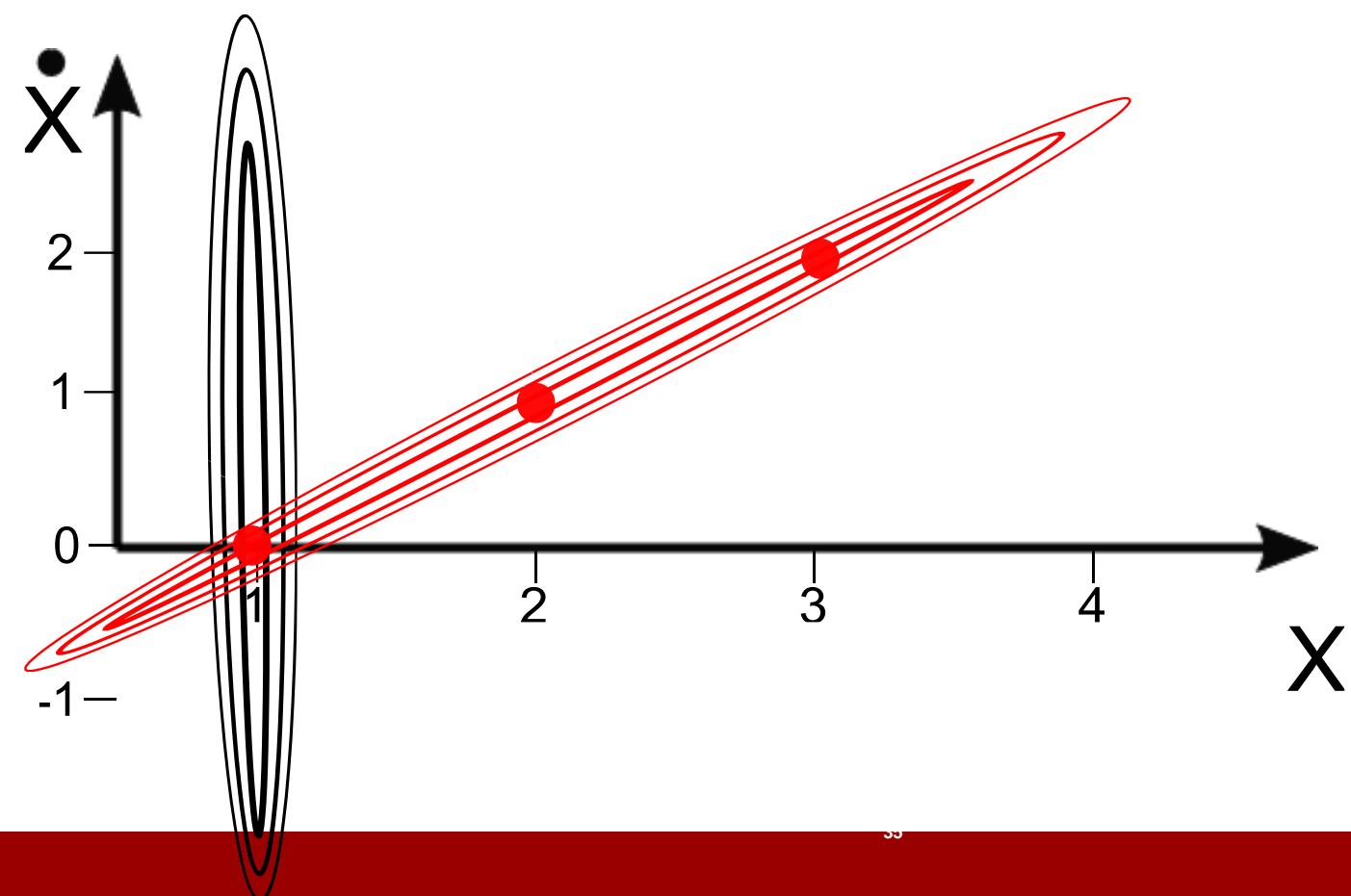
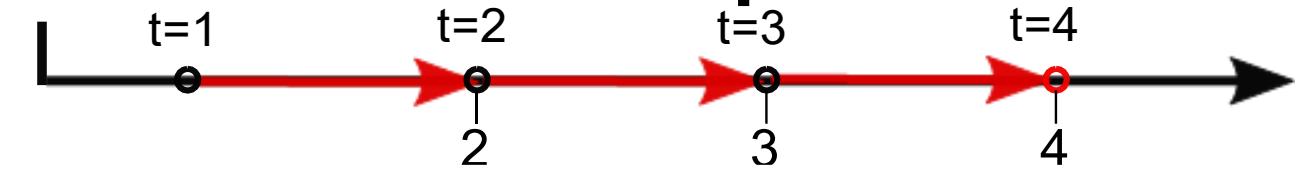
- Prediction:

- Velocity: 0

- Velocity: 1

- Velocity: 2

Consider this Gaussian



Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:

- Prior:

- Location: 1

- Prediction:

- Velocity: 0

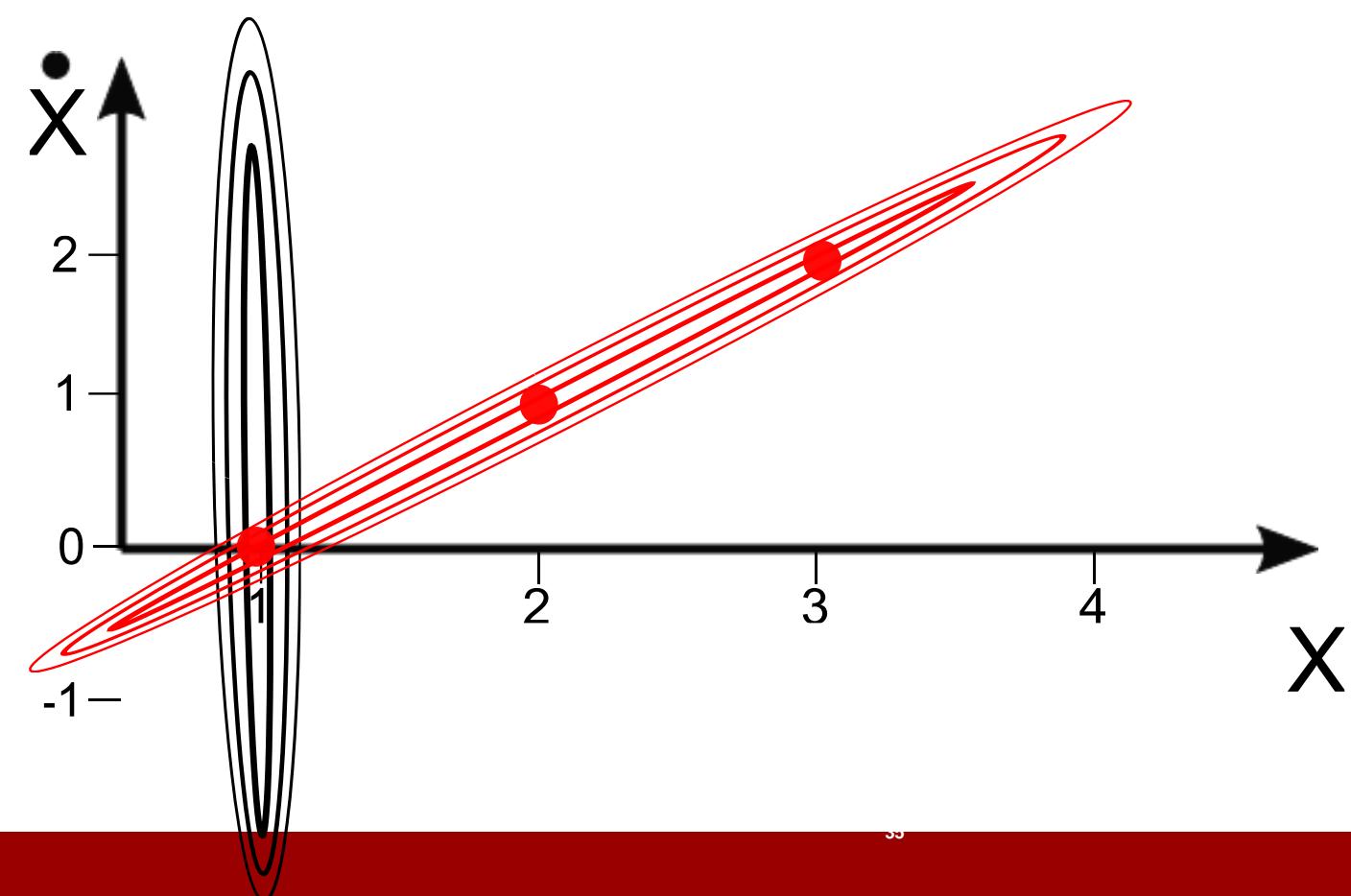
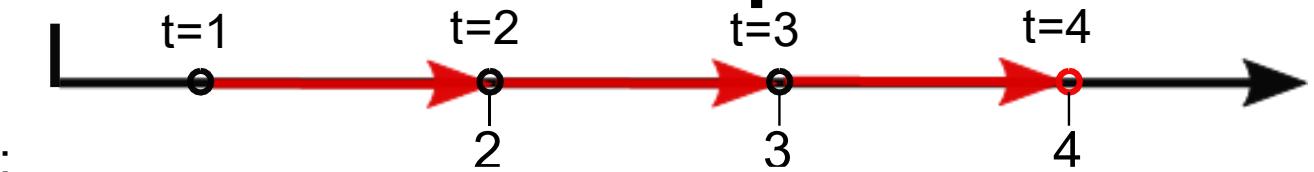
- Velocity: 1

- Velocity: 2

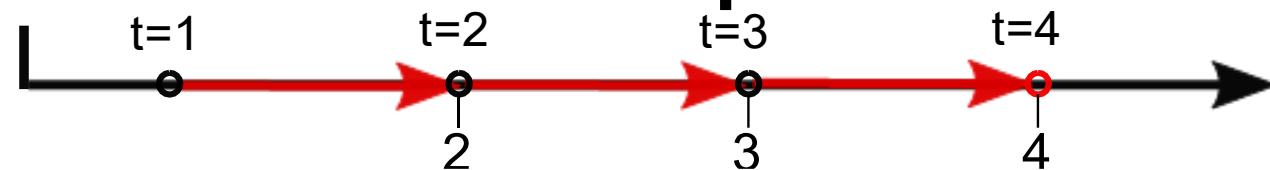
Consider this Gaussian

- Measurement:

- $Z = 2$



Multivariate Gaussians - Kalman State Space

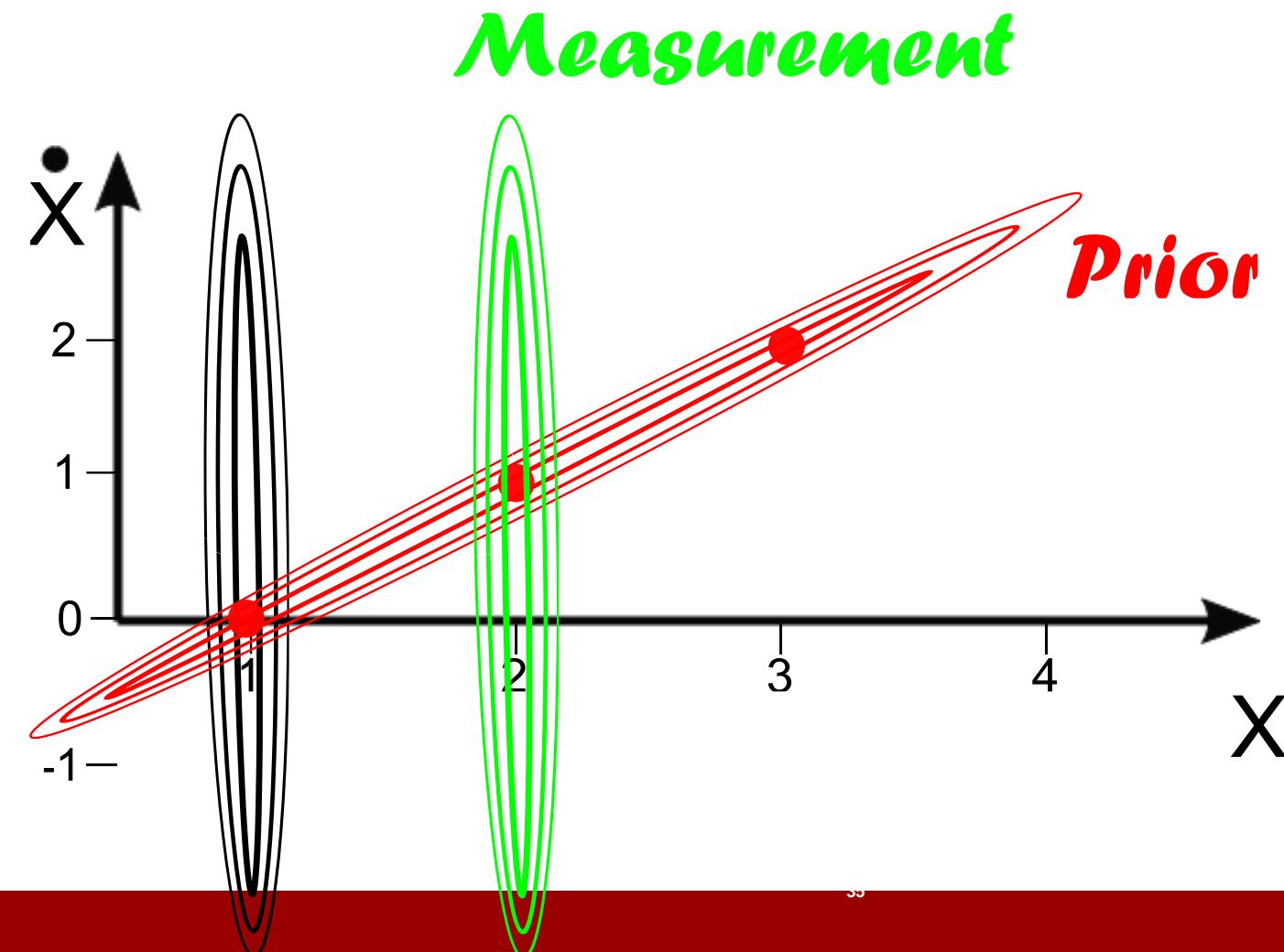


- Let's go at the Kalman state space:

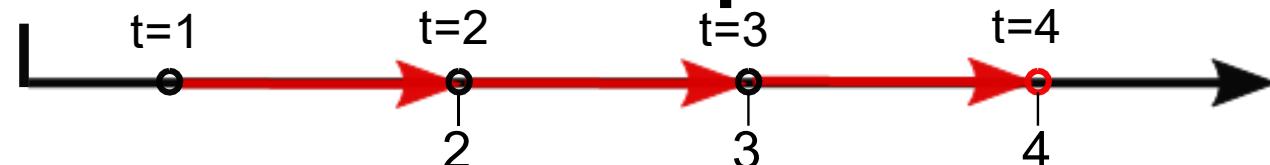
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2

Consider this Gaussian

- Measurement:
 - $Z = 2$



Multivariate Gaussians - Kalman State Space

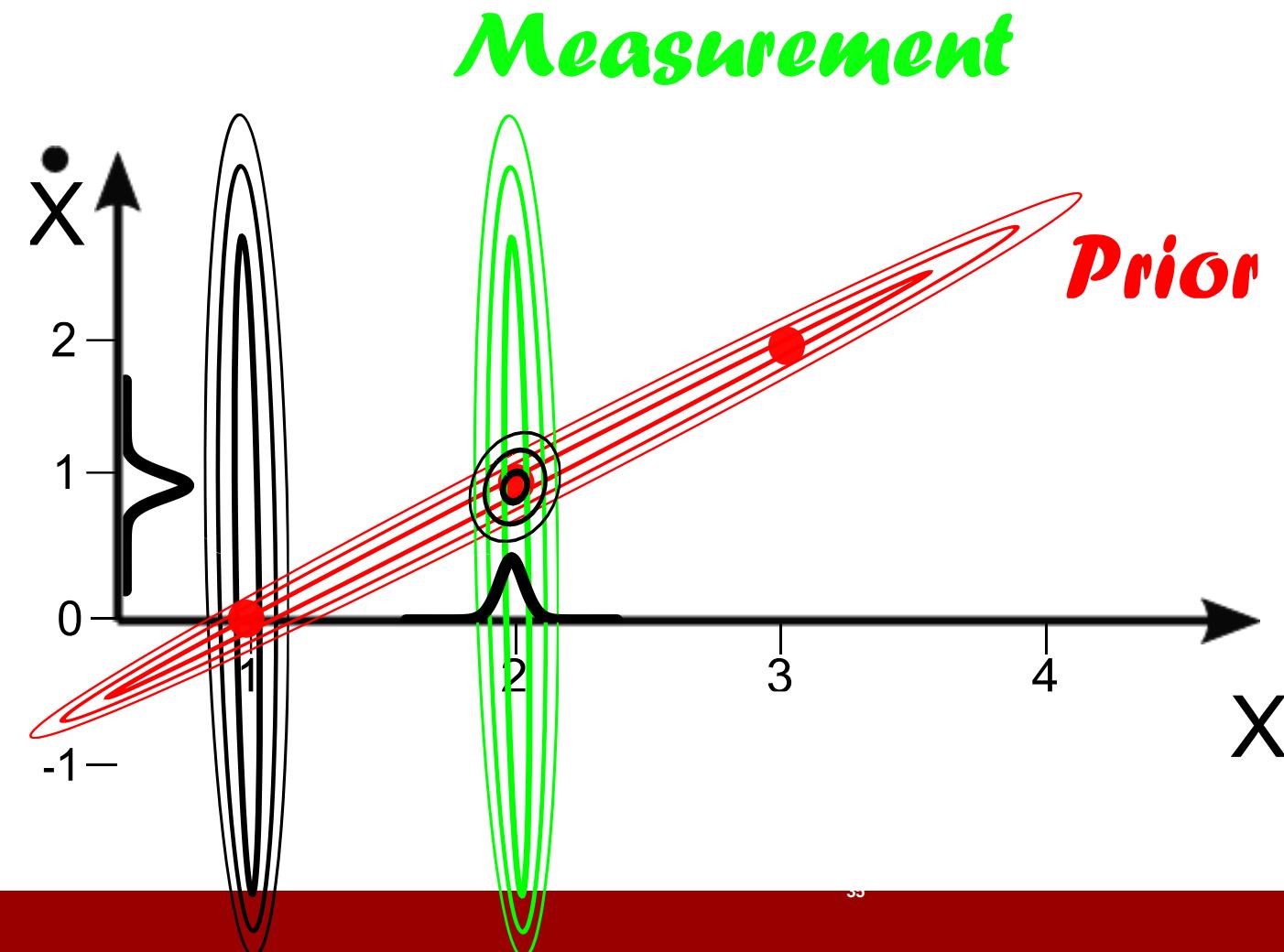


- Let's go at the Kalman state space:

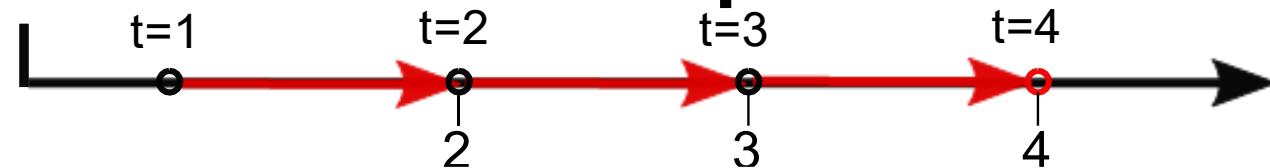
- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2

Consider this Gaussian

- Measurement:
 - $Z = 2$



Multivariate Gaussians - Kalman State Space

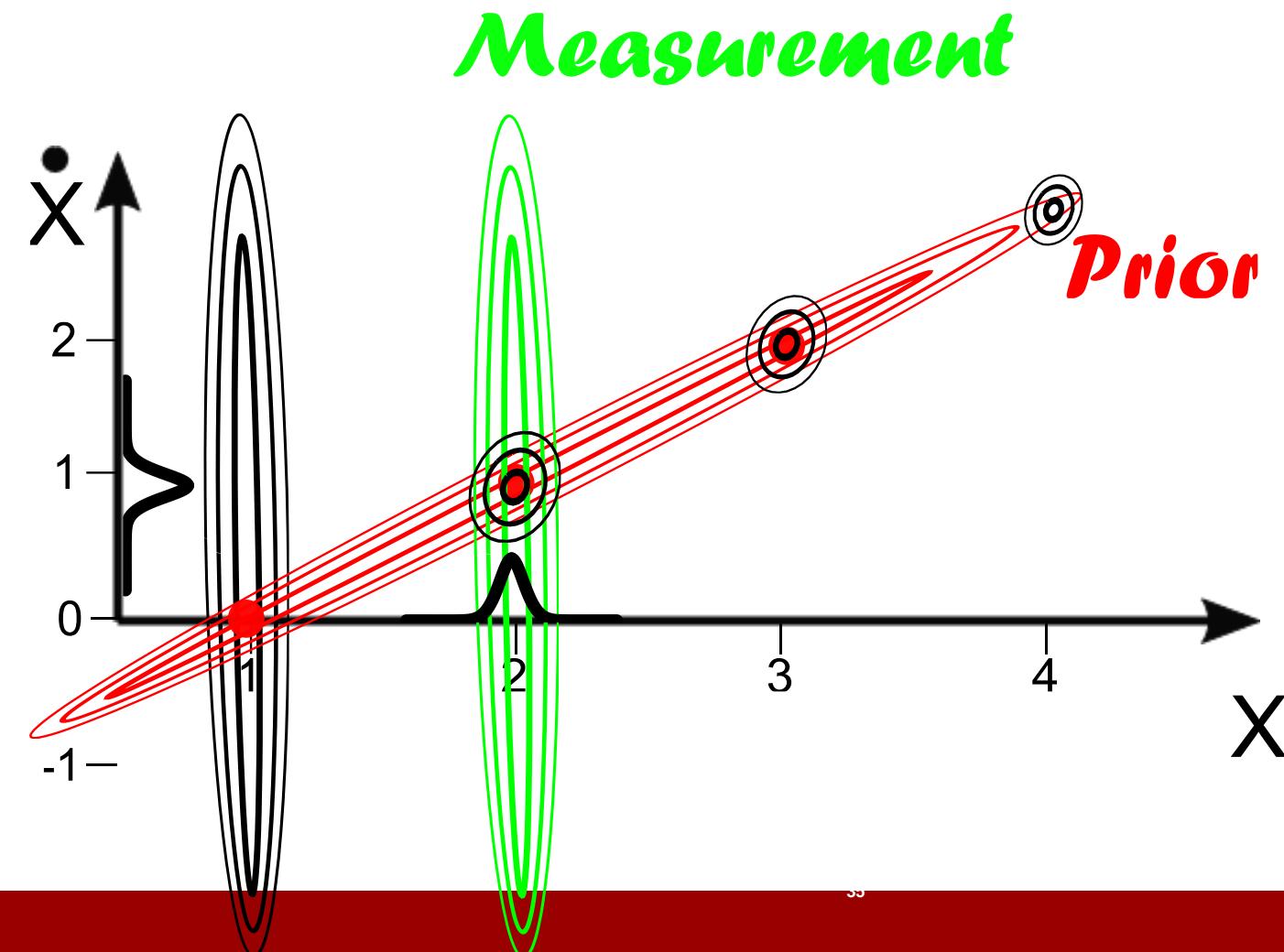


- Let's go at the Kalman state space:

- Prior:
 - Location: 1
- Prediction:
 - Velocity: 0
 - Velocity: 1
 - Velocity: 2

Consider this Gaussian

- Measurement:
 - $Z = 2$



Design of a Kalman Filter

- Two types of States variables:
 - Observables
 - Hidden
- State Transition Function:
 - Matrix
- Measurement Function:
 - Vector (Usually be Matrix)

The diagram illustrates the state transition function and measurement function for a Kalman filter.

State Transition Function:

$$\begin{pmatrix} \dot{x} \\ x' \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

Measurement Function:

$$(z') \leftarrow \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

A coordinate system is shown with the horizontal axis labeled x and the vertical axis labeled \dot{x} . A point x' is plotted on the x -axis, and a point \dot{x}' is plotted on the \dot{x} -axis. The equations $x' = x + x'$ and $\dot{x}' = \dot{x}$ are written near the axes.

Linear Algebra Formulation For Kalman Filter

(No need to memorize these:)

Prediction (Ingredients):

- X: State Vector (Including our Prior Info)
- P: Uncertainty Covariance (Incl. Prior Info)
- F: State Transition Matrix (we just discussed it)
- u: External Motion (E.g. Deceleration from car)

Measurement Update (Ingredients): :

- Z: Measurement
- H: Measurement Matrix
- R: Measurement Noise
- y: Error
- K: Gain
- I : Identity Matrix

Recipe

$$\dot{X} = F \cdot X + u$$

$$\dot{P} = F \cdot P \cdot F^T$$

$$y = Z - H \cdot X$$

$$S = H \cdot P \cdot H^T + R$$

$$K = P \cdot H^T \cdot S^{-1}$$

$$\dot{X} = X + K \cdot Y$$

$$\dot{P} = (I - K \cdot H) \cdot P$$

Conclusion

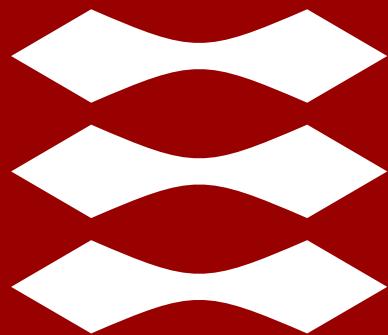
- We've acquired an amazing skill!!!
- We now understand what state estimation is
- We understand what a covariance represents
- We know how to track objects in space
 - (We can handle even occlusions)
- We know how to estimate our position (attitude) over time
- We'll come back on this (project) to do an end-to-end example!

Perception for Autonomous Systems 31392:

State Estimation - Kalman Filter

Lecturer: Evangelos Boukas—PhD

DTU



Lazaros Nalpantidis

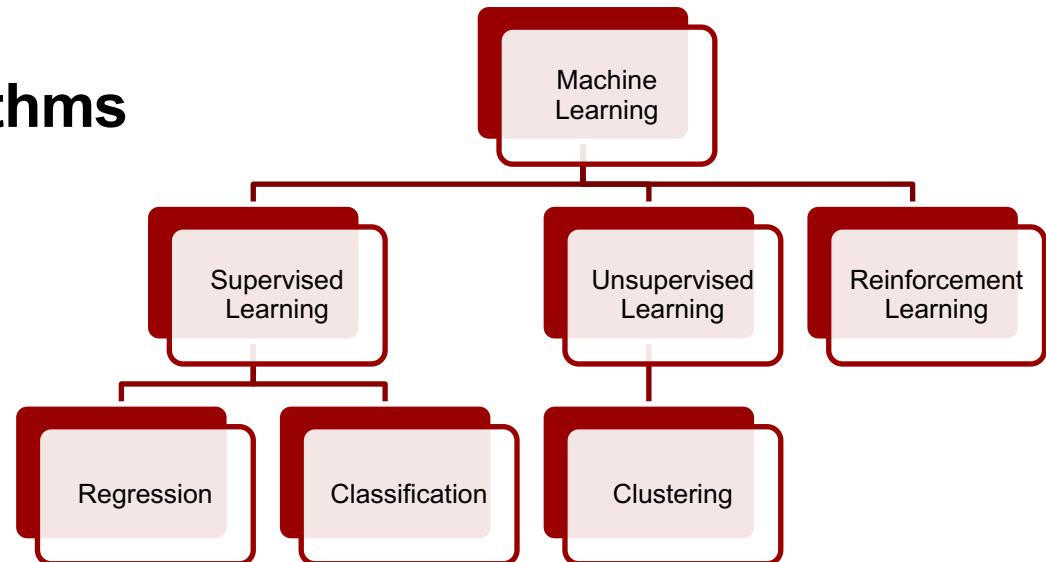
Classification

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

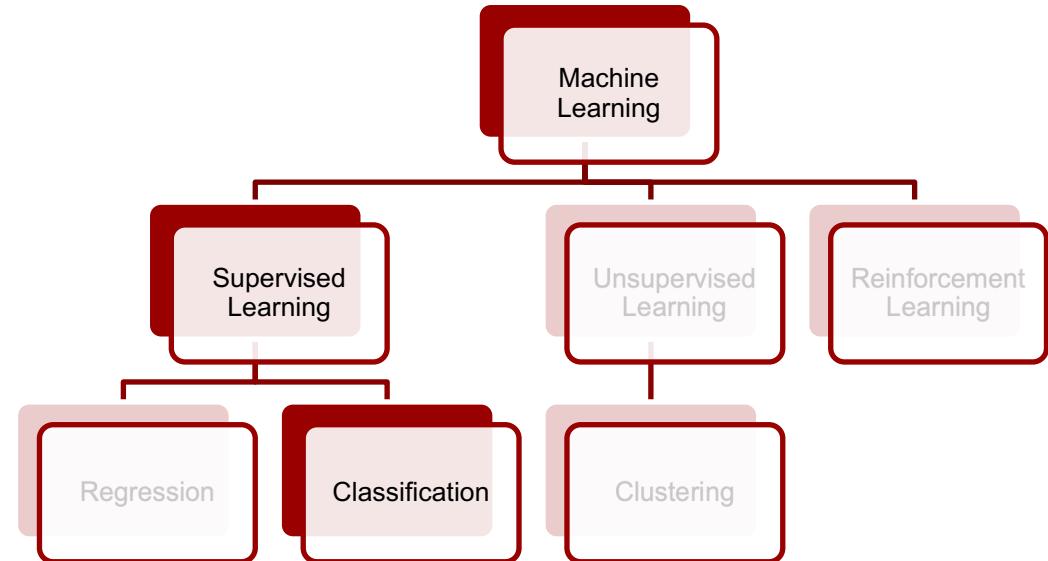
What is Classification?

A taxonomy of ML algorithms



- **Supervised Learning** – The target values are known
 - Regression – The target value is numeric
 - Classification – The target value is nominal
- **Unsupervised Learning** – The target values are unknown
 - Clustering – Group together similar instances
- **Reinforcement Learning** – Interacting with a dynamic environment the system must perform a certain goal.

Classification



- **Supervised Learning** – The target values are known
 - Classification – The target value is nominal

Classification

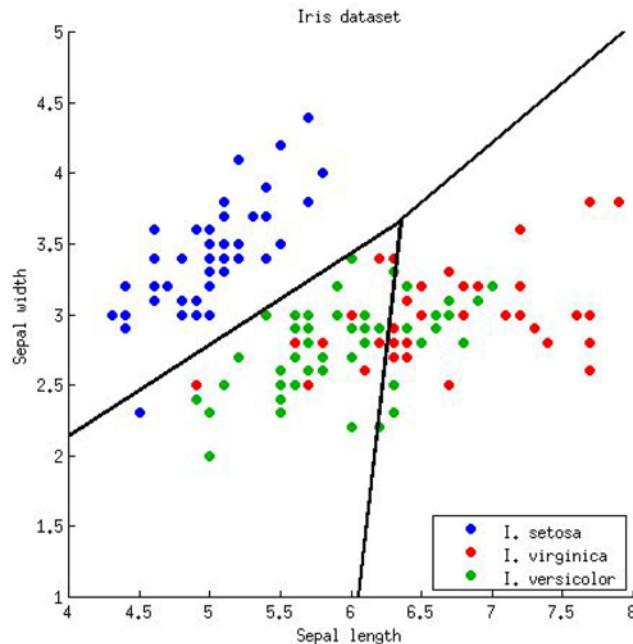
- Some terminology:
 - **Concept:** The description of the dataset. What can be learned by the examined dataset (e.g. the labels)
 - **Instances:** Individual and independent examples of the concept (e.g. an image)
 - **Attributes:** The features/dimensions that describe each instance.

	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Type
1	5.1	3.5	1.4	0.2	Iris setosa
2	4.9	3.0	1.4	0.2	Iris setosa
3	4.7	3.2	1.3	0.2	Iris setosa
4	4.6	3.1	1.5	0.2	Iris setosa
5	5.0	3.6	1.4	0.2	Iris setosa
...					
51	7.0	3.2	4.7	1.4	Iris versicolor
52	6.4	3.2	4.5	1.5	Iris versicolor
53	6.9	3.1	4.9	1.5	Iris versicolor
54	5.5	2.3	4.0	1.3	Iris versicolor
55	6.5	2.8	4.6	1.5	Iris versicolor
...					
101	6.3	3.3	6.0	2.5	Iris virginica
102	5.8	2.7	5.1	1.9	Iris virginica
103	7.1	3.0	5.9	2.1	Iris virginica
104	6.3	2.9	5.6	1.8	Iris virginica
105	6.5	3.0	5.8	2.2	Iris virginica
...					



What is Classification?

- The final output of a classification algorithm is usually decision boundaries that separate the classes.



- So, the examined concept is classified according to which boundary side it is found on.

Image classification pipeline

- Our input is a training dataset that consists of N images, each labeled with one of K different classes.
- Then, we use this training set to train a classifier to learn what every one of the classes looks like.
- In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it's never seen before. We'll then compare the true labels of these images to the ones predicted by the classifier.

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

Dimensionality Reduction

- When each concept is described by a large number of attributes, we say that the concept is described by a vector with large dimensionality.
- Generally, large dimensional vectors are well suited for machine learning.
- However,
 - some of the attributes may be redundant, i.e, attributes with small variance.
 - having many dimensions, might lead to the curse of dimensionality (data sparsity).
- Solution :
 - Reduce the dimensionality of the problem to boost speed and performance.

Dimensionality Reduction

- Principal Component Analysis (PCA)

Dimensionality Reduction

- Principal Component Analysis (PCA)
 - A solution for dealing with/reducing high dimensionality.
- Characteristics
 - Creates **new features** that are linear combinations of the original features
 - New features are orthogonal to each other
 - Keep the new features that account for a large amount of the variance in the original dataset
 - Re-base the dataset's coordinate system in a new space defined by its lines of greatest variance

Dimensionality Reduction

- Principal Component Analysis (PCA)

1. Center the input data

2. Calculate Covariance Matrix

$$C = \frac{1}{N-1} X^T X$$

3. Compute eigenvectors & eigenvalues of the Covariance Matrix

- The first principal component is the eigenvector of the covariance matrix that has the largest eigenvalue

- » This vector points towards the direction of the largest variance of the data

- » The corresponding eigenvalue defines the magnitude of this vector

- The second largest eigenvector is orthogonal to the largest eigenvector, and points into the direction of the second largest spread of the data.

4. Sort the eigenvectors according to their eigenvalues

5. Calculate the variance score (significance).

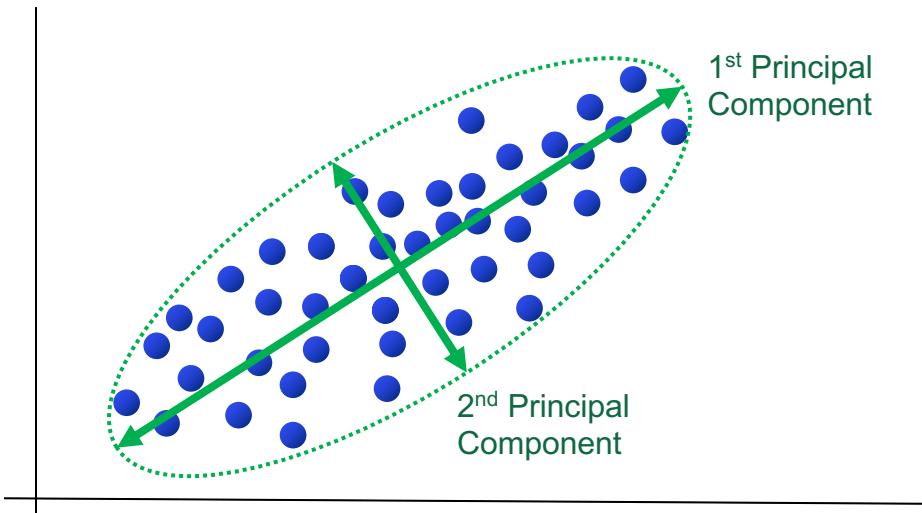
6. Keep eigenvectors that explain most (e.g. 95%) variance / remove the rest! (\leftarrow dimensionality reduction)

7. Project instances to eigenvectors y.

$$y = W^t x$$

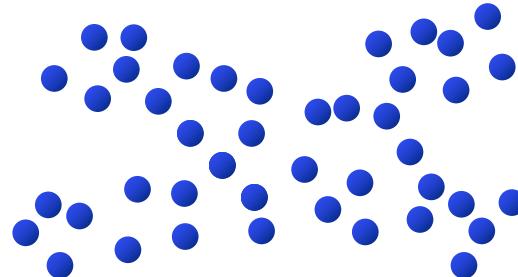
Dimensionality Reduction

- Principal Component Analysis (PCA)
 - A solution for dealing with/reducing high dimensionality.
- **Eigenvector:** points to the direction of variance.
- **Eigenvalues:** shows how much of the total variance is explained by the corresponding eigenvector.
- Keep the eigenvectors that explain most (e.g. 95%) of the variance.



Dimensionality Reduction

- Principal Component Analysis (PCA)
 - A solution for dealing with/reducing high dimensionality.
- **Eigenvector:** points to the direction of variance.
- **Eigenvalues:** shows how much of the total variance is explained by the corresponding eigenvector.
- Keep the eigenvectors that explain most (e.g. 95%) of the variance.



← What are the Principal Components here?

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

Eigenfaces

- Face Recognition
- Consider many images of faces (each image with the same dimensions).
- Consider each image, not as a 2D table, but just as a very long vector!
 - How many dimensions does such a vector have?
- We want to construct a low-dimensional linear subspace that best explains the variation in our set of face images.
- Most face images lie on a low-dimensional subspace determined by the first some(!) directions of maximum variance.
- Use PCA to find the vectors(eigenfaces) that determine this subspace
- Then, all face images can be expressed as linear combinations of the eigenfaces

M. Turk and A. Pentland, 1991. Eigenfaces for Recognition, Journal of Cognitive Neuroscience, 3(1), 71-86.

Large dataset of face images

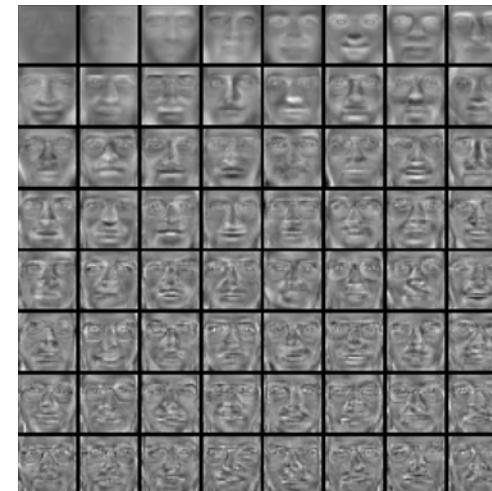


Mean Image



PCA

The first 64 eigenvectors (eigenfaces)



- Then an image of our dataset becomes:

$$\text{Image} = \text{Mean Image} + c_1 * (\text{1st Eigenface}) + c_2 * (\text{2nd Eigenface}) + \dots$$

- So, these coefficients c_1, c_2, \dots, c_{64} represent the face image!!
- We can perform Face Recognition
 - For a new face image
 - We can calculate its 64 coefficients
 - Find the closest training face (most similar coefficients) in the 64-dimensional space (the most similar face of my training dataset)
 - **Classify** the new face as this most similar training face.

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

k-Nearest Neighbors (k-NN)

k-NN is a geometric method for classification.

- It is based on the assumption that neighbor instances belong to the same class.

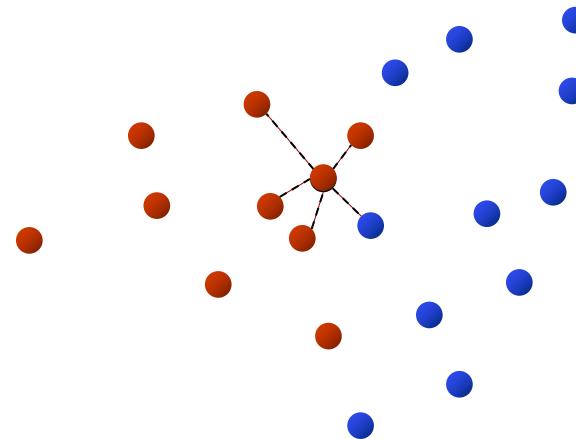
For classifying a new instance x_* :

- we calculate its distance from all the other instances.
- Then we select the k-nearest instances.
- The probability of the class is given by:

$$p(C_j|x_*) = \frac{k_j}{K}$$

where k_j is the number of nearest neighbors that belong to class j , and K is the total number of nearest neighbors.

k-Nearest Neighbors (k-NN)



e.g. $k=5$

What is the class of the black dot?

The black dot belongs to the “red” class with probability 4/5

k-Nearest Neighbors (k-NN)

The parameters of k-NN are:

- the number of nearest neighbors k
- the distance metric that would be used.
 - Most common distance metric is Euclidean -- other possibilities are: Mahalanobis, Manhattan etc.

k-Nearest Neighbors (k-NN)

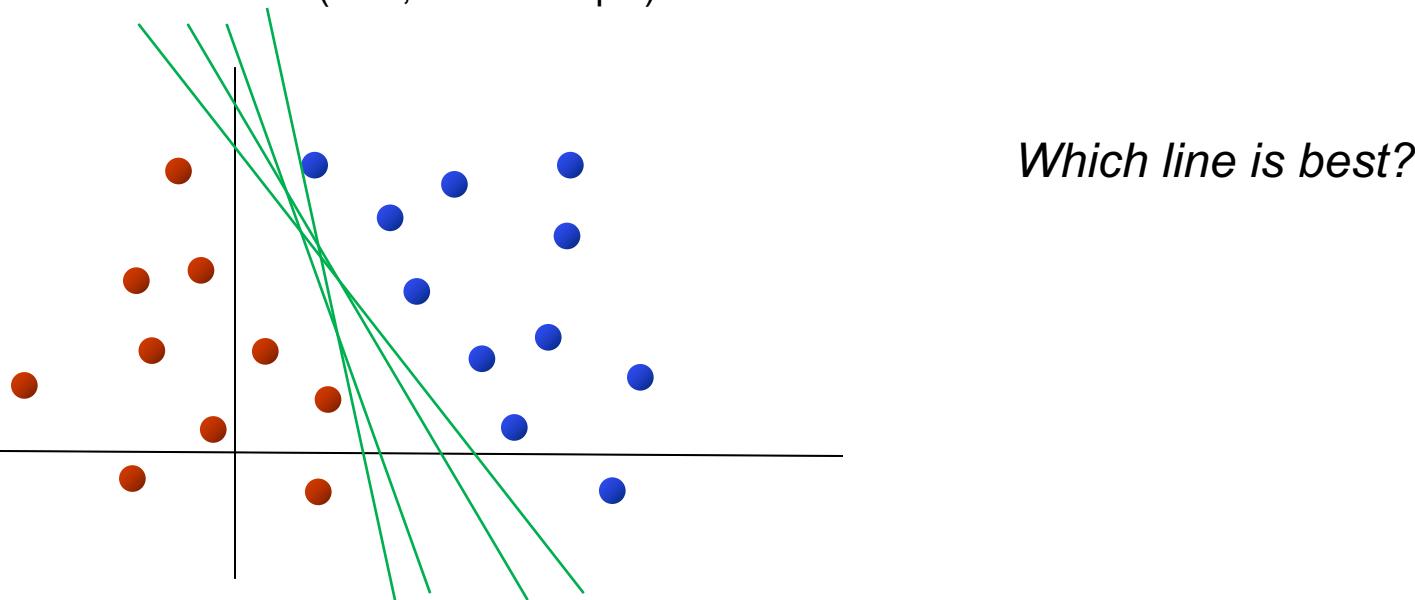
Characteristics of k-NN:

- Very small numbers of k do not perform well on noisy data.
- k-NN is a non-linear classifier.
- Despite its simplicity is powerful.
- Can be very slow for large datasets.

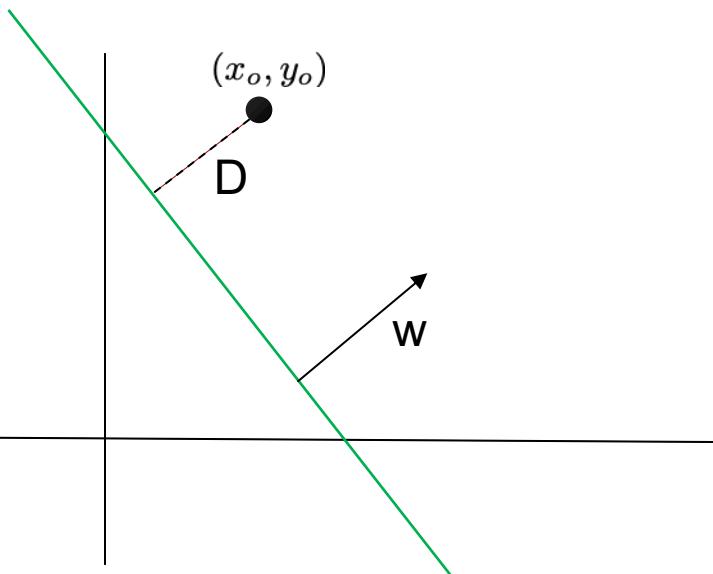
- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

Support Vector Machines (SVMs)

- SVMs are geometric methods for classification
 - Find a line/surface in feature space that separates the classes
- Linear Classifier: (here, a 2D example)



- Lines in 2D



$$\mathbf{w} = \begin{bmatrix} p \\ q \end{bmatrix}$$
$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$px + qy + b = 0$$

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0$$

$$D = \frac{|px_0 + qy_0 + b|}{\sqrt{p^2 + q^2}} = \frac{\mathbf{w}^T \cdot \mathbf{x} + b}{|\mathbf{w}|}$$

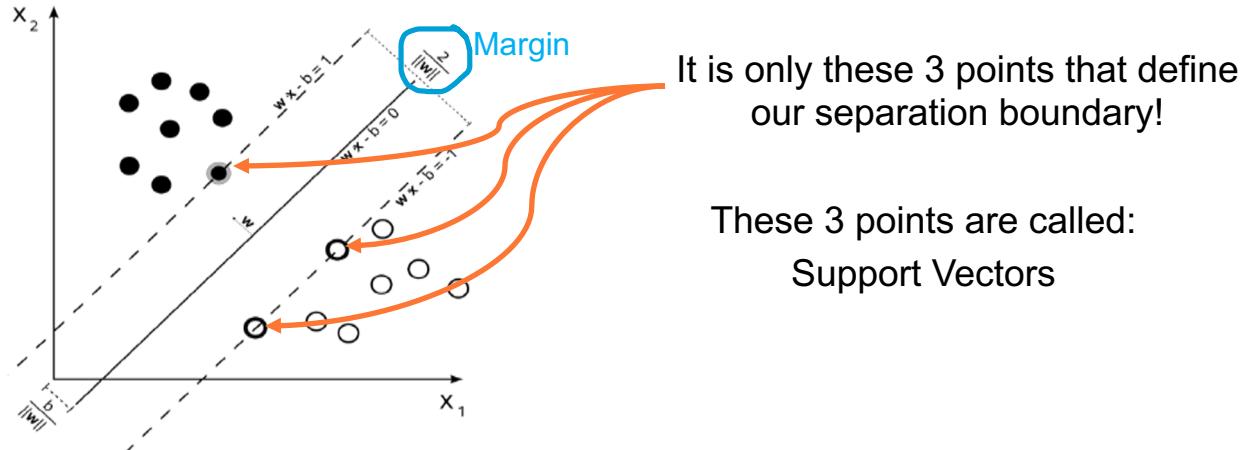
Support Vector Machines (SVMs)

- SVMs are geometric methods for classification
- They derive a linear decision boundary which satisfies two criteria:
 1. The distance between the instances and the boundary is as large as possible (max margin).
 2. The instances are classified as good as possible.
- Thus, the decision boundary is a hyperplane $h(x)$ defined as:

$$h(x) = w^T x + b$$

...the goal is to find the w that satisfy the above criteria.

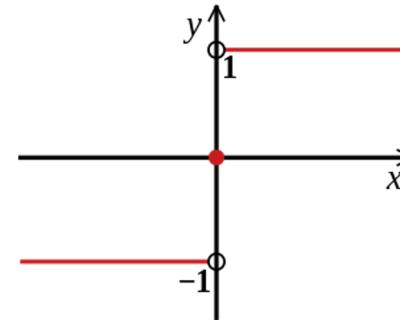
- SVMs are geometric methods for classification
- They derive a linear decision boundary which satisfies two criteria:
 1. The distance between the instances and the boundary is as large as possible (max margin).
 2. The instances are classified as good as possible.



Support Vector Machines (SVMs)

- Maximize the margin $\frac{2}{\|\mathbf{w}\|}$ while correctly classifying all training data points
- The goal is to find the \mathbf{w} that satisfies the criteria of: $h(x) = \mathbf{w}^T x + b$
- The classification rule is expressed as: $f(x_*) = \text{sgn}(h(x_*))$
- The sign function is defined as:

$$\text{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$



- Thus, a new instance is labeled with 1 if it is located above the boundary and with -1 otherwise.

Support Vector Machines (SVMs)

- Extending SVMs!
 - Moving beyond 2D
 - There might be more than 2 classes
 - Data might not be linearly separable

Support Vector Machines (SVMs)

- Extending SVMs!
 - **Moving beyond 2D**
 - There might be more than 2 classes
 - Data might not be linearly separable
- Our math work just fine for more than 2 dimensions!
 - Instead of lines, we get planes/hyperplanes...

Support Vector Machines (SVMs)

- Extending SVMs!
 - Moving beyond 2D
 - **There might be more than 2 classes**
 - Data might not be linearly separable
 - Multi-class SVMs
 - Possible solutions:
 - 1-vs-all
 - » Combine multiple binary classifiers.
 - » Choose the one with the largest decision value
 - 1-vs-1
 - » Train one classifier for each pair of classes
 - » Choose the class that most classifiers vote for

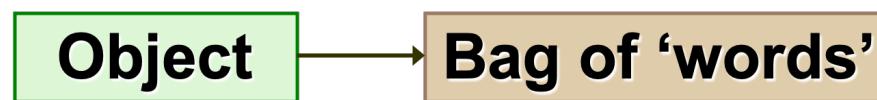
Support Vector Machines (SVMs)

- Extending SVMs!
 - Moving beyond 2D
 - There might be more than 2 classes
 - **Data might not be linearly separable**
 - The kernel trick
 - We can use kernels (functions)
 - We are moving our problem in a higher- (or even infinite-) dimensional space.
 - In this new space our problem is linearly separable!
 - Typical kernels:
 - Linear
 - Polynomial
 - Gaussian (Radial Basis Function – RBF)

- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

Class/Category Recognition

- Bag of Words



Class/Category Recognition

- Bag of Words

Analogy to documents

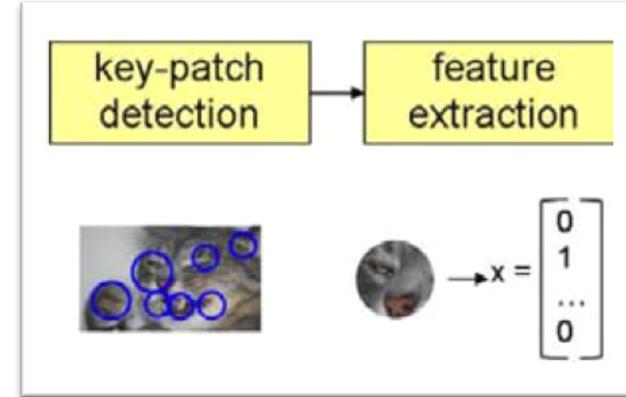
Of all the sensory impressions proceeding to the brain, the visual experiences are the dominant ones. Our perception of the world around us is based essentially on the messages that reach our brain via our eyes. For a long time it was believed that the retinal image was processed by the visual centers in the cerebral cortex. This was a movie screen theory. In 1960, two researchers discovered that the eye, cell, optical nerve, image and Hubel and Wiesel.

Hubel and Wiesel have demonstrated that the message about the image falling on the retina undergoes a top-down analysis in a system of nerve cells stored in columns. In this system each column has its specific function and is responsible for a specific detail in the pattern of the retinal image.

China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% increase in exports to \$750bn, compared with \$660bn. The Chinese government, annoyed that the US is pressuring China's central bank to allow the yuan to appreciate, has agreed to let the yuan rise against the dollar. The Chinese government also needs to encourage foreign demand so that the Chinese economy can grow. The country. China has been allowed to trade within a narrow range of the dollar, but the US wants the yuan to be allowed to trade freely. However, Beijing has made it clear that it will take its time and tread carefully before allowing the yuan to rise further in value.

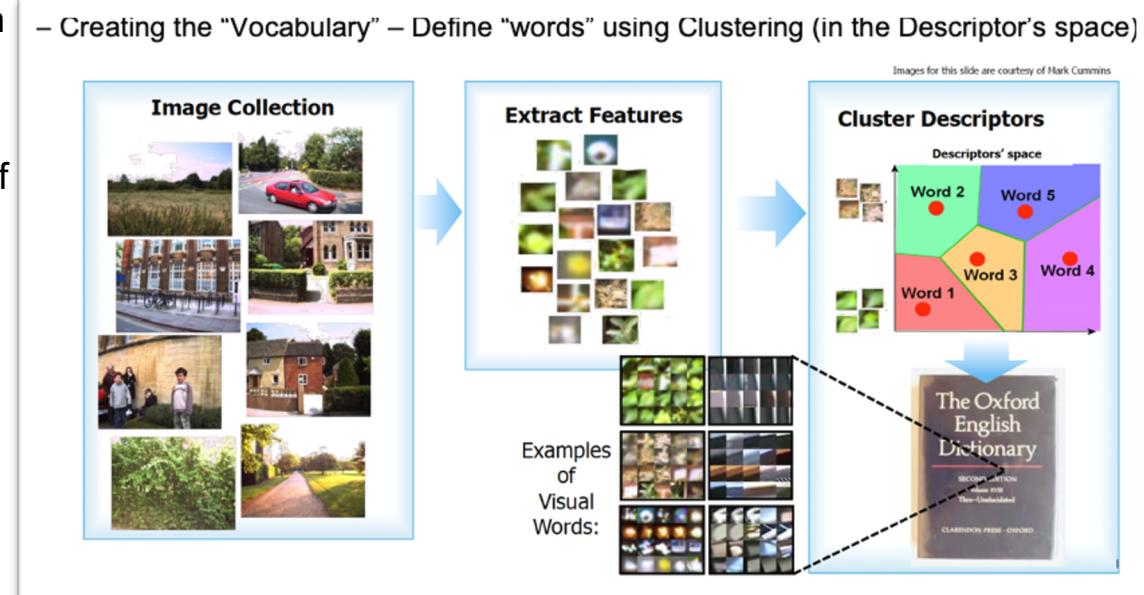
Class/Category Recognition

- **Bag of Words**
 - Feature Detection
 - Feature Extraction/Description (e.g. SIFT, ...)
 - Codebook/Vocabulary Generation
 - Image description:
 - Histogram Computation
 - a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.
 - Classification of new image in the descriptor's space



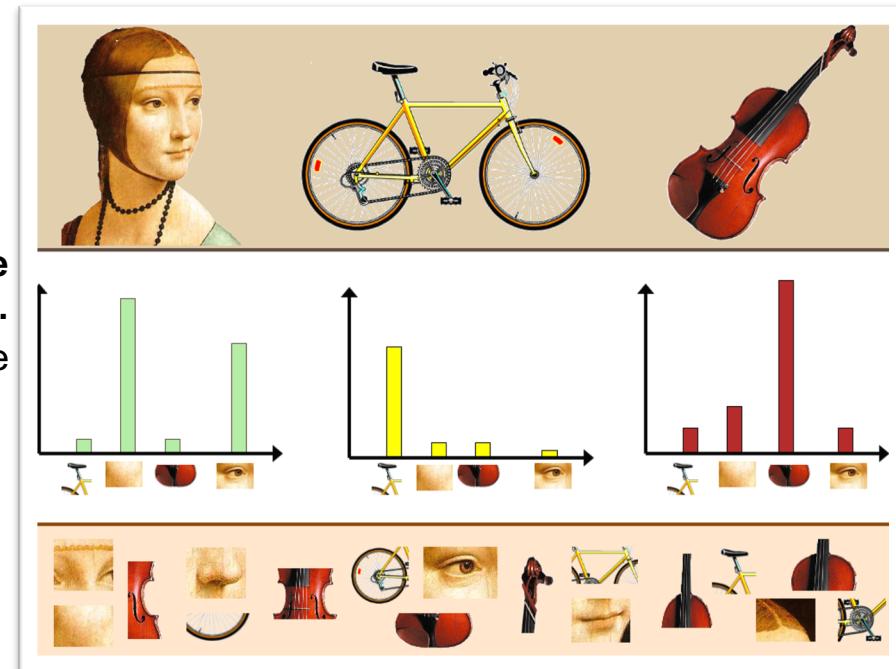
Class/Category Recognition

- **Bag of Words**
 - Feature Detection
 - Feature Extraction/Description (e.g. SIFT, ...)
- **Codebook/Vocabulary Generation**
- Image description:
 - Histogram Computation
 - a bag of visual words is a vector of



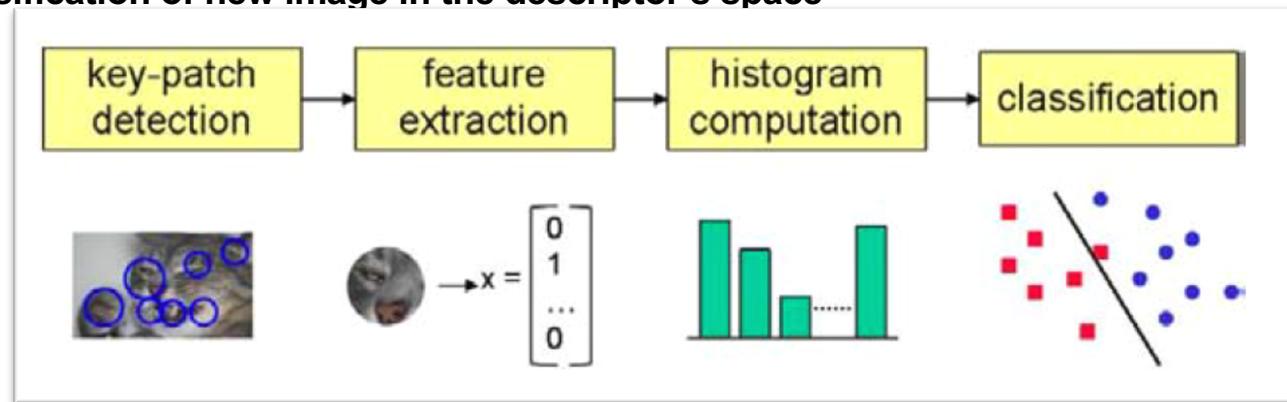
Class/Category Recognition

- **Bag of Words**
 - Feature Detection
 - Feature Extraction/Description (e.g. SIFT, ...)
 - Codebook/Vocabulary Generation
- **Image description:**
 - Histogram Computation
 - **a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.**
- Classification of new image in the descriptor's space



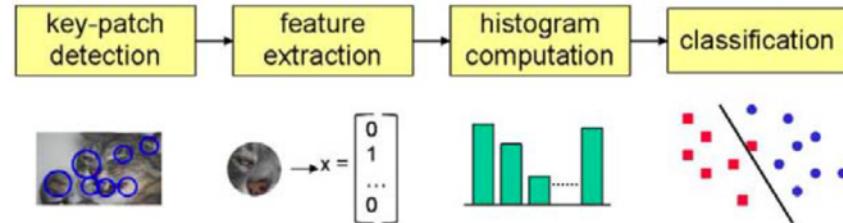
Class/Category Recognition

- **Bag of Words**
 - Feature Detection
 - Feature Extraction/Description (e.g. SIFT, ...)
 - Codebook/Vocabulary Generation
 - Image description:
 - Histogram Computation
 - a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.
- **Classification of new image in the descriptor's space**



- What is Classification?
- Dimensionality Reduction
 - PCA
 - Face detection
- Classifiers
 - k-Nearest Neighbors
 - Support Vector Machines
- Class/Category Recognition
- Summary

- Machine Learning techniques can find the best label for a new instance, based on known labeled training instances.
 - We might
 - know what we are looking for → **object detection**
 - have a specific rigid object we are trying to recognize → **instance recognition**
 - want to recognize instances of extremely varied classes (e.g. animals or furniture) → **category/class recognition**



"Woven into all of these techniques is the topic of learning, since hand-crafting specific object recognizers seems like a futile approach given the complexity of the problem."

R. Szelinski, "Computer Vision: Algorithms and Applications", 2010.

- Latest methods rely on deep neural networks.

- **Object Detection in images:**

- YOLO

- » J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 779-788.
 - » <https://arxiv.org/abs/1506.02640>

- Faster R-CNN

- » S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.
 - » <https://arxiv.org/abs/1506.01497>

- **Classification in 3D point clouds**

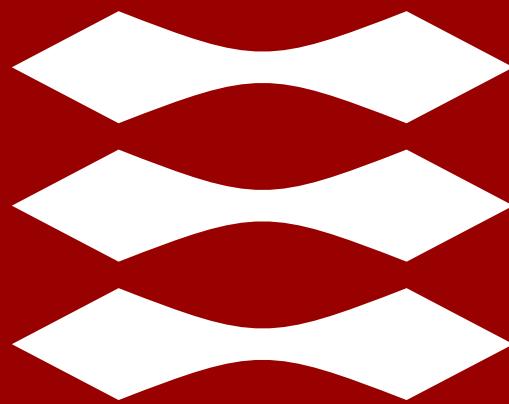
- PointNet

- » R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 77-85.
 - » <https://arxiv.org/abs/1612.00593>

Lazaros Nalpantidis

Classification

DTU



Perception for Autonomous Systems 31392:

Visual Odometry

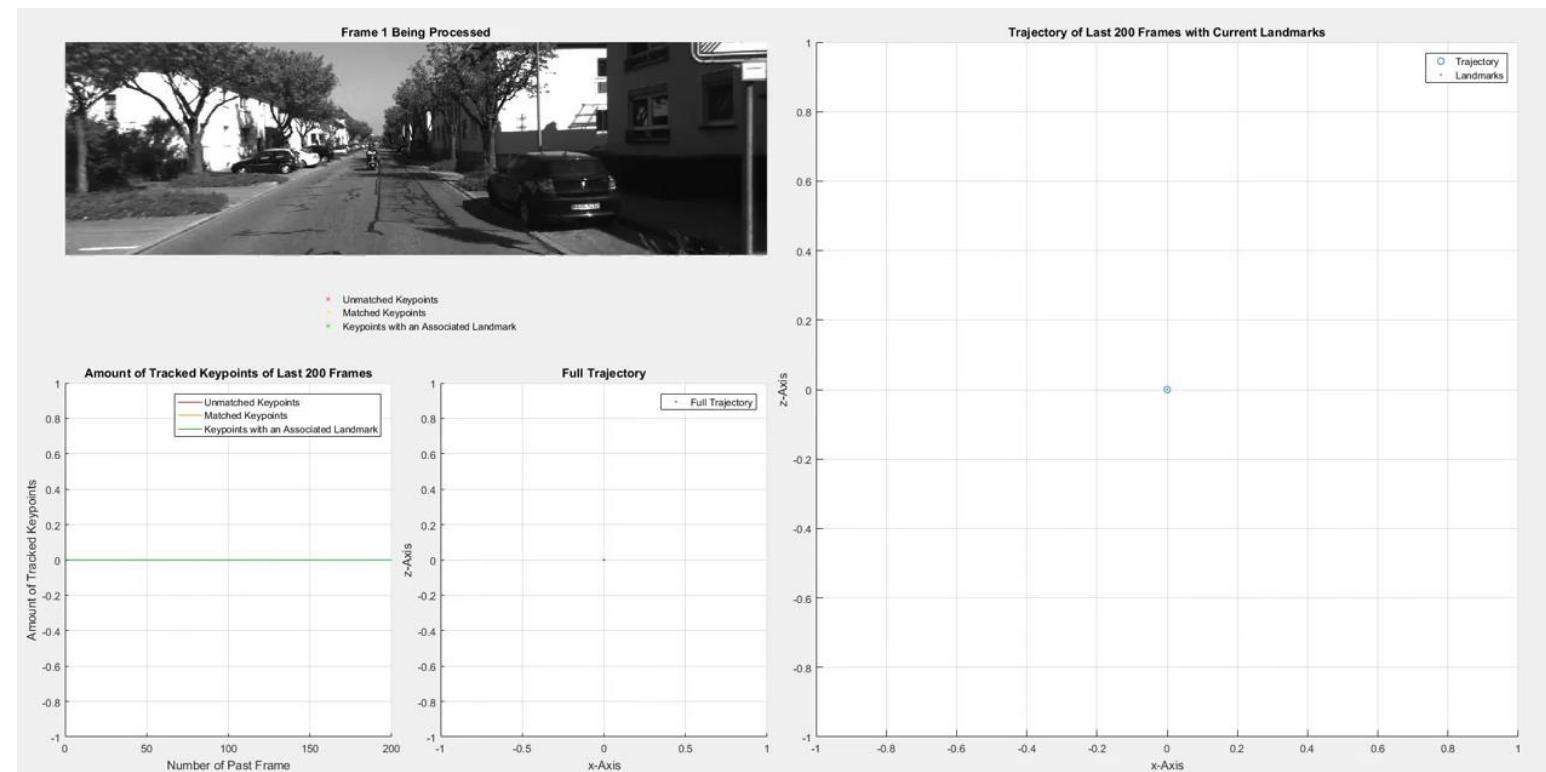
Lecturer: Evangelos Boukas—PhD

Outline

- Orientation (Attitude) Representations
- Relative Pose Estimation
 - 3D registration
 - PnP
 - Least Squares - SVD
- Visual Odometry
 - 3D-3D
 - 3D-2D
 - 2D-2D
- Local Bundle Adjustment
- Visual Inertial Odometry -VIO
 - Loosely Coupled EKF
 - Tightly Coupled EKF

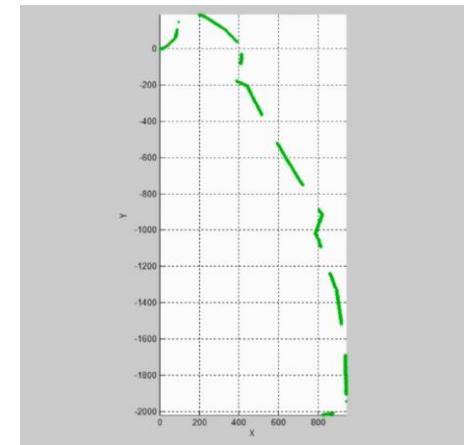
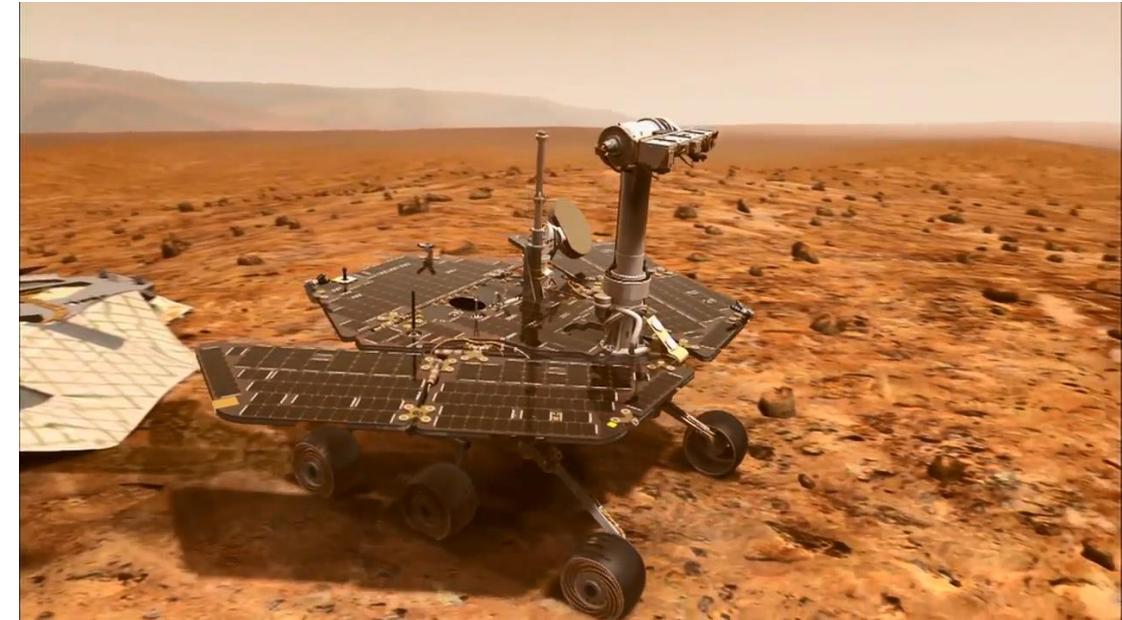
What is Visual Odometry and what it is not

- Visual Odometry (VO) concerns the use of cameras to estimate the Pose (position and orientation) of a mobile system, by observing the apparent motion of the “static” world.
- VO assumes a static world where the only moving object is the mobile system
- VO does not provide a map of the environment neither it uses previous states of the world to improve its accuracy (SLAM)



Video by prof. Scaramuzza University of Zurich

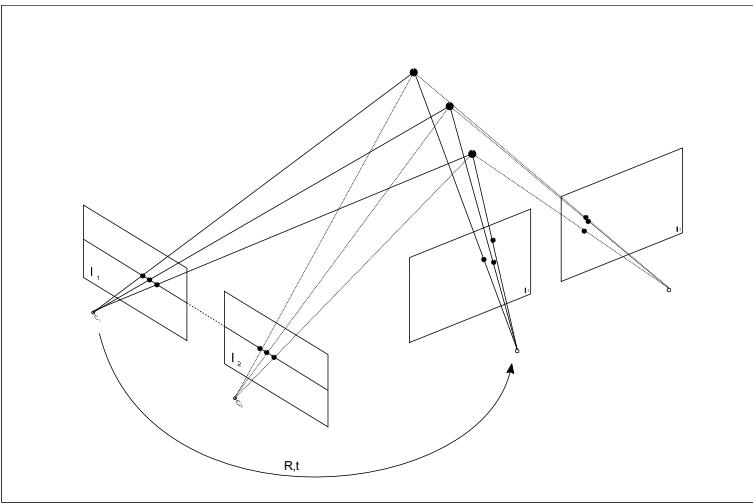
- It has been proposed as an alternative to Wheel odometry
- Its accuracy is –assuming reasonable trajectories- ~1%
- VO estimations are usually combined with other sensors
 - GPS, IMU, Laser, Wheel odometry
 - VINS, VIO stands for Visual Inertial Odometry
- VO has had some extraordinary usages



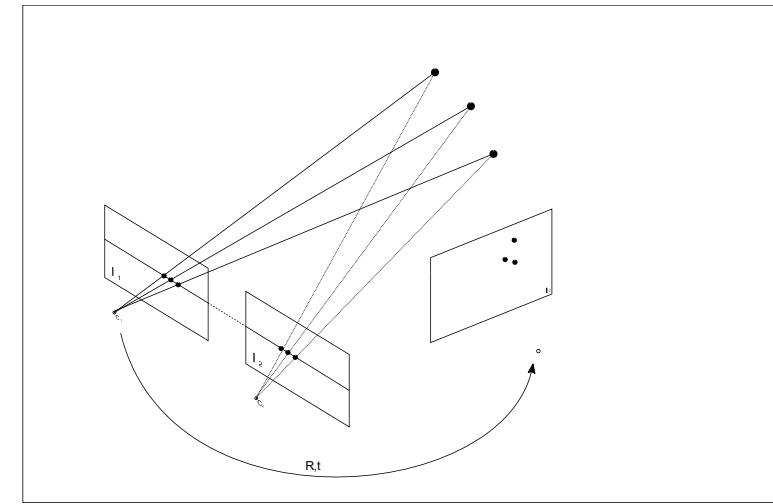
VO is neither SFM nor VSLAM

- Structure from motion tries to solve also for the feature points as well:
 - “Given Calibrated point projections of $p=1 \dots N$ points in camera (or frame) $f=1 \dots F$ (x_p^f, y_p^f)
 - Find the rigid transformation $R^T t$ and the point’s 3D position $F X_p = (X_p, Y_p, Z_p)$ which satisfies the projection equations
- Visual SLAM uses state estimation to exploit additional constraints and re-observations of the same areas(Loop-closures) to optimize the localization

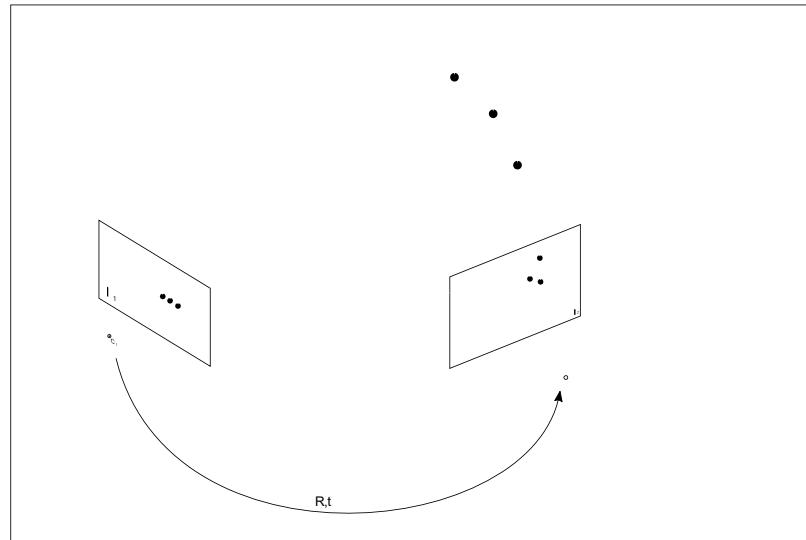
The 3 main variants of VO



3D-3D



3D-2D



2D-2D

- Rotation Matrix
 - Positives (The king of Orientation)
 - Unique, no gimbal lock
 - Negatives:
 - No perturbation, interpolation, unintuitive
- Euler anglers
 - Positives
 - Minimal representation, intuitive
 - Negatives
 - Gimbal Lock, non commutative
- Axis Angle:
 - Positives
 - No gimbal lock, minimal representation, nice for perturbation, linear mapping to rotation matrix
 - Negative
 - Not linear “scaling” wrt magnitude
 - Exponential coordinates
- Quaternions
 - Positives
 - all the axis angle ones, smooth trajectory
 - Negatives
 - No direct geometric representation

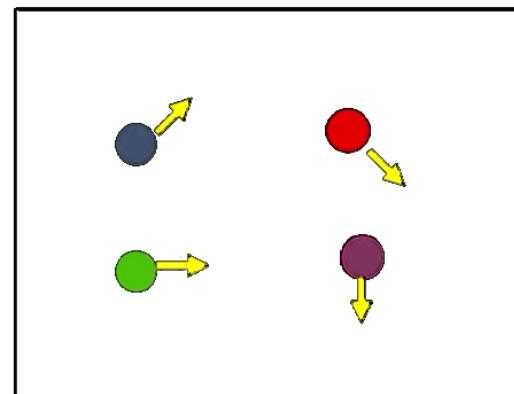
Shuster, M.D. (1993). "[A Survey of Attitude Representations](#)". *Journal of the Astronautical Sciences* **41** (4): 439–517. [Bibcode: 1993JAnSc..41..439S](#)

Motion tracking

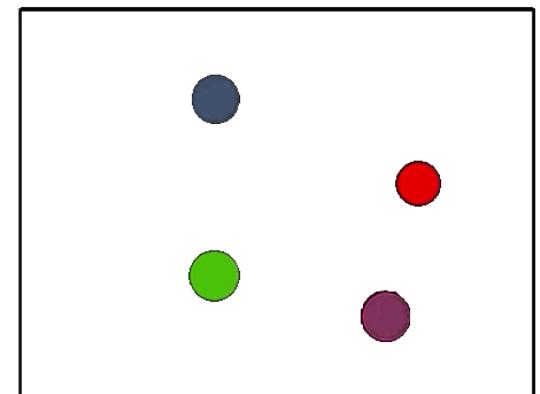
- Two main approaches
 - Feature based
 - Optic Flow
- Feature based:
 - Calculate feature on both images
 - Match among the features
- or
- Calculate features
- Do block Matching around our initial point (for small motion)

Motion tracking – Optical Flow

- Estimate the apparent motion
- Given a pixel in location $I(x,y,t)$,
find the “nearby pixels with the same
color”
 - Same intensity (in the local window)
 - Limited displacement



$I(x, y, t)$

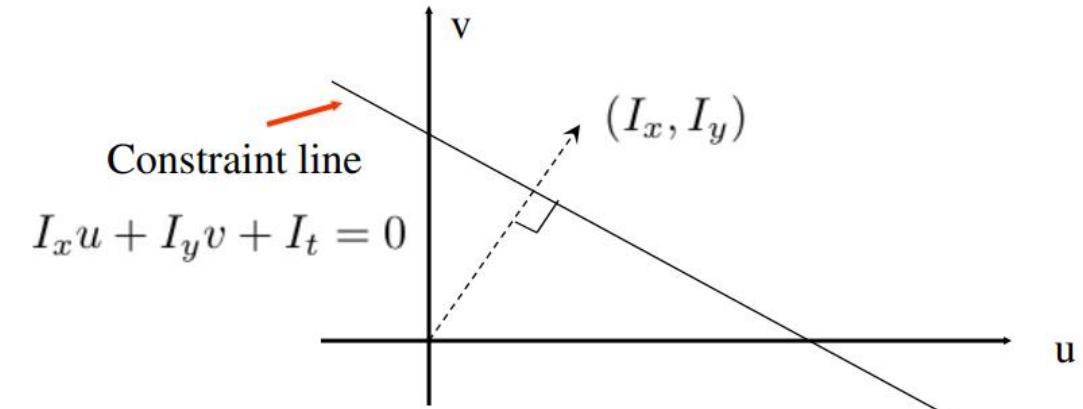


$I(x, y, t + 1)$

- $I(x + y, t) = I(x + u, y + v, t + 1)$
- $I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \rightarrow I_t + \nabla I \cdot (u, v) = 0 \rightarrow I_x u + I_y v + I_t = 0$
- This problem is the problem of “global Optical Flow” and is hard to solve due to *under definition*

Motion tracking – Optical Flow – Lukas Kanade

- The previous function is a line in the u, v space
- $I_x u + I_y v + I_t = 0$



- We can try to impose more constraints so the line becomes a point
- By assuming that for an image neighborhood we have constant “velocity”: We want to minimize:

$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

Motion tracking – Optical Flow – Lukas Kanade

- If we use a 5x5 window,
that gives us 25 equations per pixel
- This does not work
 - For edges
 - For large areas
- How to solve it:
The iterative approach
 - Estimate Motion
 - Warp Image
 - Repeat until no change

$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

A
 25×2

d
 2×1

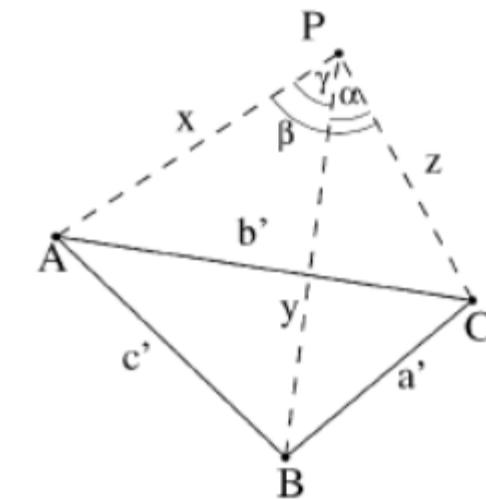
b
 25×1

Relative Pose estimation – 2 corresponding 3D Point Clouds

- Depends on the available data:
 - 3D or 2D
(Eg: Do we work on a monocular camera or a stereo one?)
 - 3D registration – case where f_k and f_{k-1} are in specified in 3D points
 - PCA
 - SVD, RANSAC
 - ICP, combination of above
- What if we have correspondences?
- Rigid Transformation using RANSAC (3 points)

Relative Pose estimation – 3D “Point clouds” and 2D Image Points

- The problem where f_{k-1} is specified in 3D points and f_k in 2D image coordinates - This problem is known as perspective from n points (PnP)
- A popular implementation is the P3P (perspective from 3 points)
 - Let P be the Center of Perspective
 - A, B, C, the “control points”, the 3D correspondence
 - Applying the cosine law (e.g.: $x^2+z^2-2*x*z*\cos\beta = b'$), we get 2 quadratic equations with 2 unknowns, resulting to 4 possible solutions for R,t
 - Using in Ransac to find the correct, or employ a 4th point, check orientation consistency
- Many other implementations:
 - One of the most prominent is EPnP ($n \geq 4$)
 - Reformulate the problem with virtual “control points”



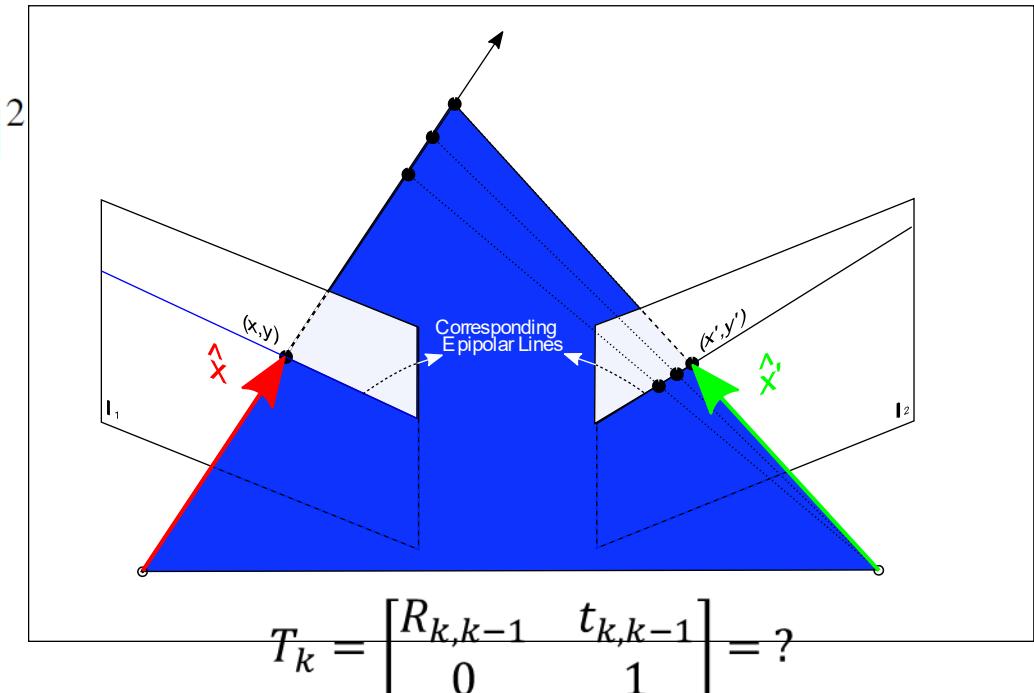
Relative Pose estimation - 2D Image Points

- The problem where f_{k-1} and f_k are specified in 2D image coordinates
- The minimal-case solution involves 5-point correspondences
- The solution is found by determining the transformation that minimizes the reprojection error of the corresponding points,

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

where p_k^i is the points on image k and $g(X^i, C_k)$ the reprojection of the corresponding $k-1$ point on the camera k

Wait but WHY?



Relative Pose estimation - 2D Image Points

The Essential Matrix can be computed directly from the image coordinates (using SVD).

At least 5 points needed! The more points, the better!

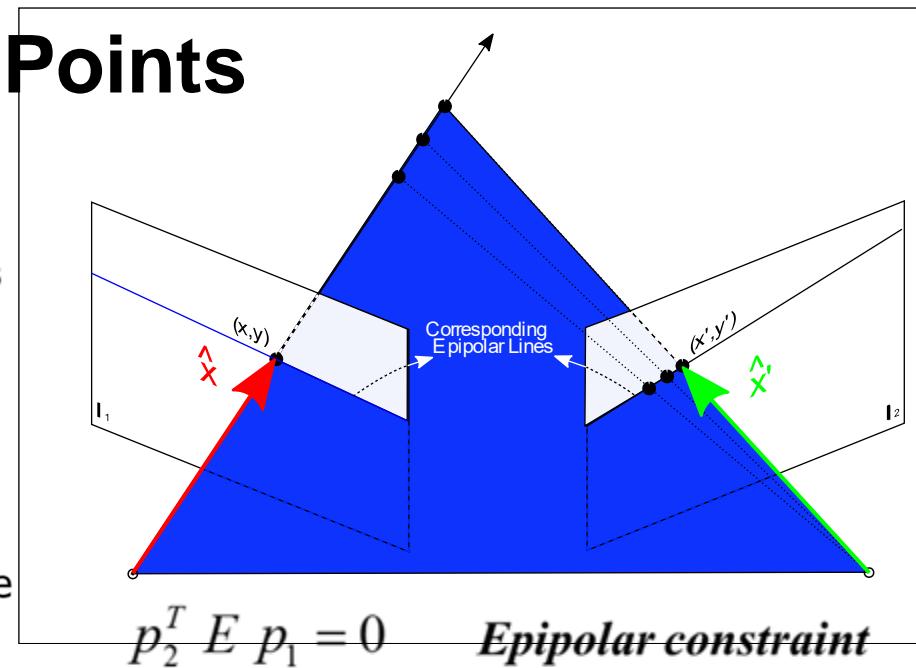
The Essential Matrix can be decomposed into R and t (again using SVD)

Let $p_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$, $p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$ be the coordinates one feature correspondence

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} e_{11} \\ \vdots \\ e_{33} \end{bmatrix}$$

$$p_2^T E p_1 = 0 \Rightarrow [x_1 x_2 \ y_1 x_2 \ z_1 x_2 \ x_1 y_2 \ y_1 y_2 \ z_1 y_2 \ x_1 z_2 \ y_1 z_2 \ z_1 z_2] E = 0$$

which can be solved with SVD



$$E = [t]_x R \quad \text{essential matrix}$$

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

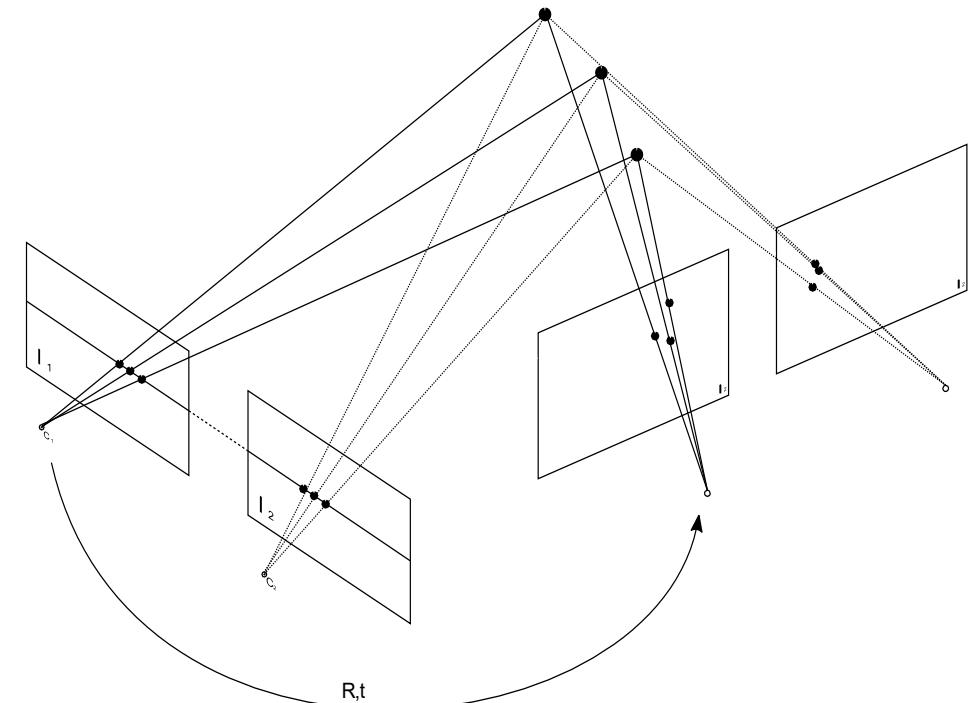
Relative Pose estimation - 2D Image Points

- The scale problem:
 - The essential matrix is calculated up to scale, That means that the t of Rt is also up to scale:
 - How can we recover it?
We have to figure out a relative accurate movement between frames f_k and f_{k-1}
Any ideas?

Visual Odometry 3D – to 3D

Algorithm 2. VO from 3-D-to-3-D correspondences.

- 1) Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$
- 2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
- 3) Triangulate matched features for each stereo pair
- 4) Compute T_k from 3-D features X_{k-1} and X_k
- 5) Concatenate transformation by computing
 $C_k = C_{k-1} T_k$
- 6) Repeat from 1).

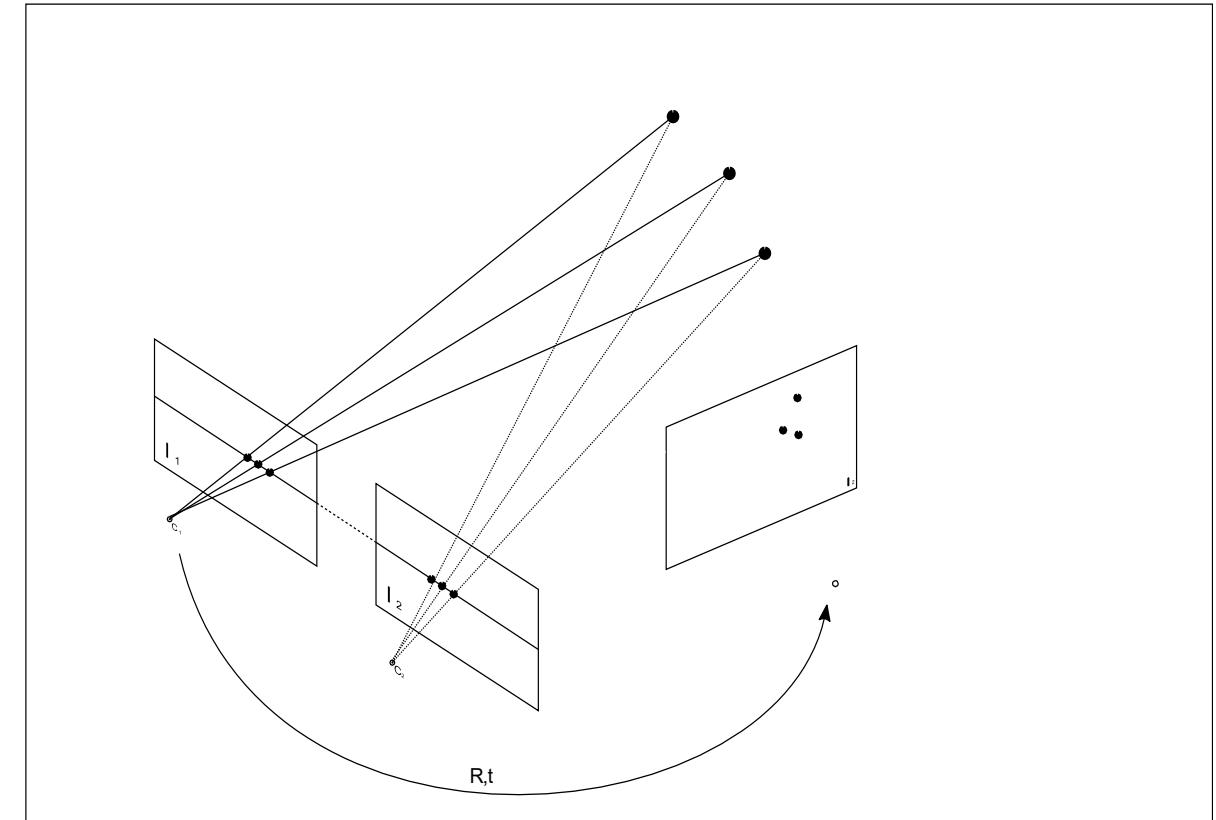


3D-3D

Visual Odometry 3D – to 2D

Algorithm 3. VO from 3-D-to-2-D Correspondences.

- 1) Do only once:
 - 1.1) Capture two frames I_{k-2}, I_{k-1}
 - 1.2) Extract and match features between them
 - 1.3) Triangulate features from I_{k-2}, I_{k-1}
- 2) Do at each iteration:
 - 2.1) Capture new frame I_k
 - 2.2) Extract features and match with previous frame I_{k-1}
 - 2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
 - 2.4) Triangulate all new feature matches between I_k and I_{k-1}
 - 2.5) Iterate from 2.1).

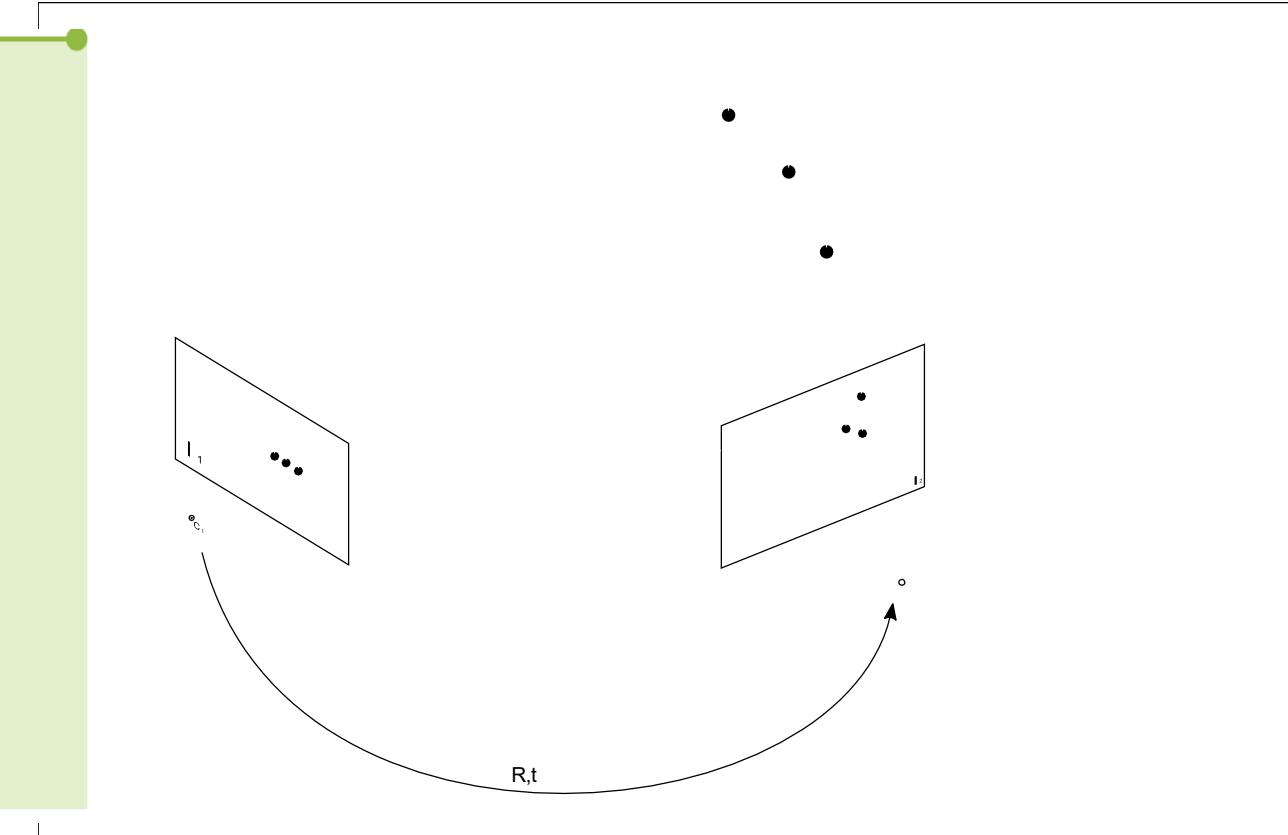


3D-2D

Visual Odometry 2D – to 2D

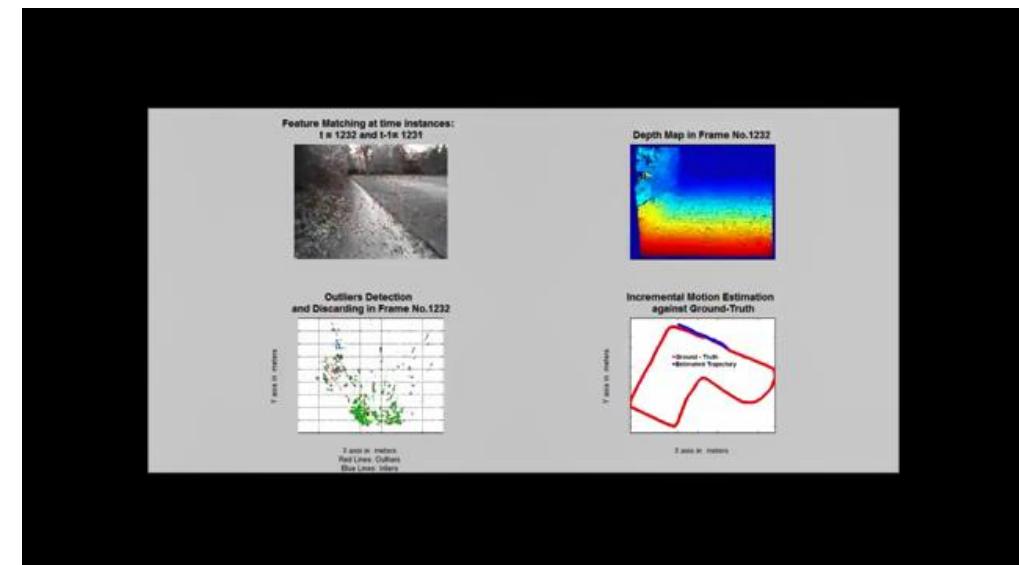
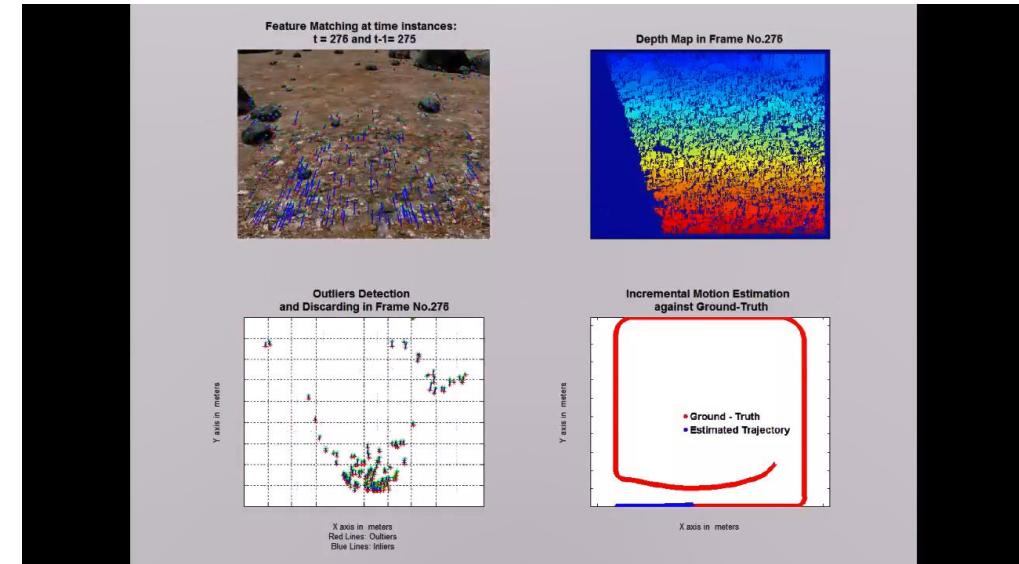
Algorithm 1. VO from 2-D-to-2-D correspondences.

- 1) Capture new frame I_k
- 2) Extract and match features between I_{k-1} and I_k
- 3) Compute essential matrix for image pair I_{k-1}, I_k
- 4) Decompose essential matrix into R_k and t_k , and form T_k
- 5) Compute relative scale and rescale t_k accordingly
- 6) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 7) Repeat from 1).



Some Notes

- 2D-2D and 3D-3D are better than 3D-3D. Why?
- Stereo VO even with 2D-2D is better. Why?
- Any ideas on improving VO?



Some Notes

- 2D-2D and 3D-32 are better than 3D-3D. Why?
- Stereo VO even with 2D-2D is better. Why?
- Any ideas on improving VO?

CNN-SVO:

Improving the Mapping in Semi-Direct Visual Odometry
Using Single-Image Depth Prediction

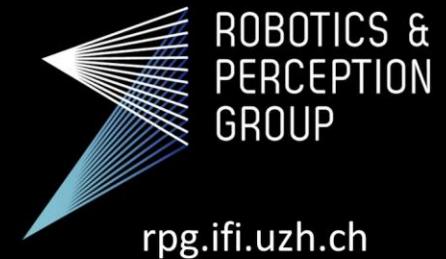
Shing Yan Loo, Ali Jahani Amiri,

Sai Hong Tang, Syamsiah Mashohor, Hong Zhang



SVO 2.0: Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems

Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, Davide Scaramuzza



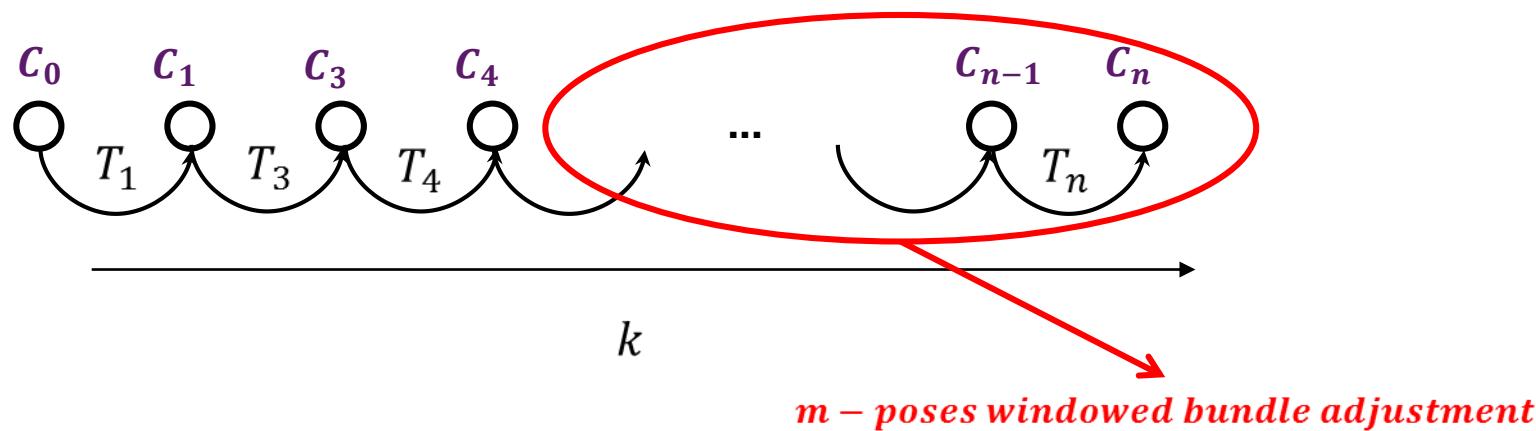
University of
Zurich^{UZH}

Department of Informatics

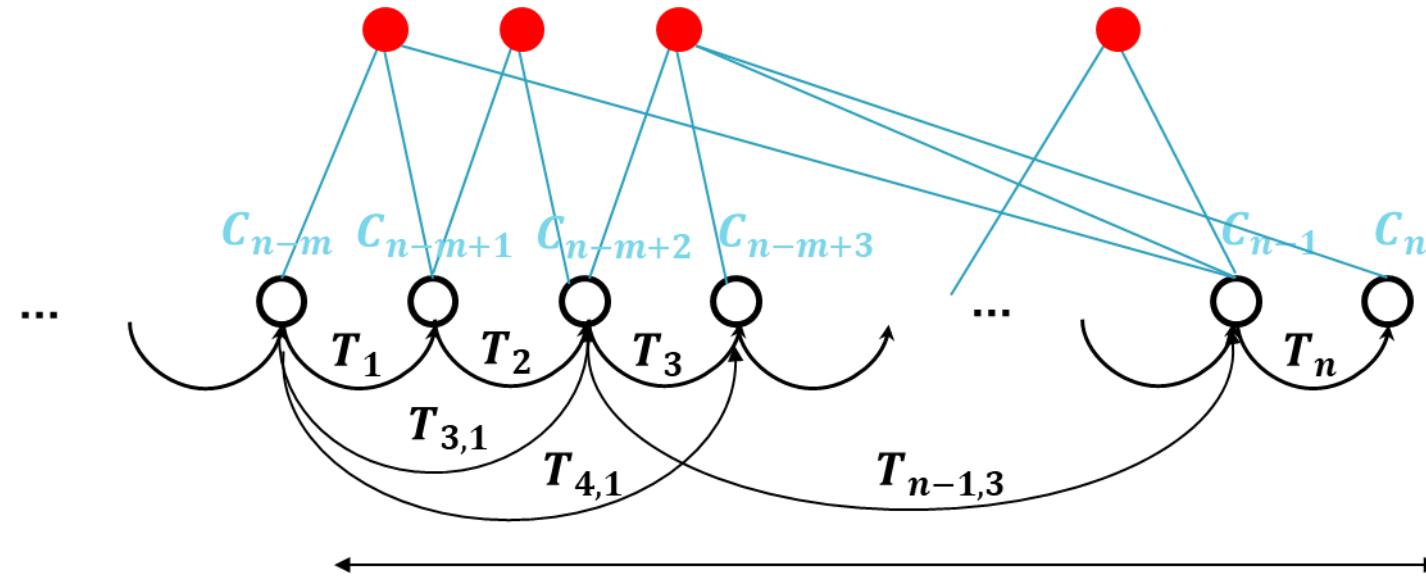


What happens over time?

- VO accumulates the transformations T_k from frame f_{k-1} to frame f_k over time providing the full trajectory $C_{0:n}$.
- What is the problem with that?
- How can we solve it?
- We can optimize over multiple frames in a procedure which is called **bundle adjustment**



Windowed Bundle Adjustment (BA)



- Similar to pose-optimization but it also optimizes 3D points m
- In order to not get stuck in local minima, the initialization should be close the minimum
- Levenberg-Marquadt can be used

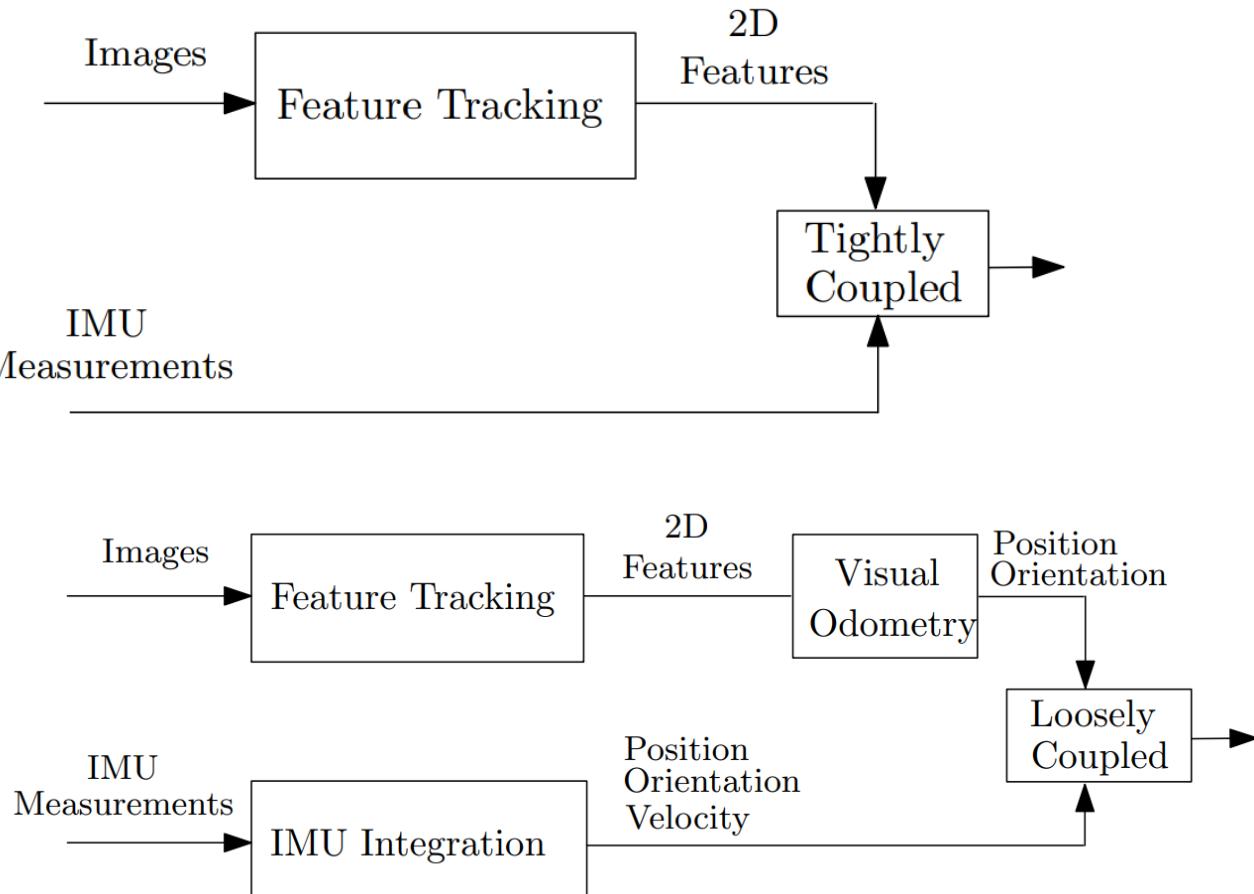
Improving the Accuracy of VO

- Other sensors can be used such as
 - IMU (called inertial VO)
 - Compass
 - GPS
 - Laser
- An IMU combined with a single camera allows the estimation of the absolute scale. Why?

Visual Inertial Odometry VIO

“Visual-Inertial odometry (VIO) is the process of estimating the state (pose and velocity) of an agent (e.g., an aerial robot) by using only the input of one or more cameras plus one or more Inertial Measurement Units (IMUs) attached to it”

- Cameras are slow and information rich
 - IMUs are fast but high noise
 - Two main paradigms of Filtering for State Estimation:
 - Loosely coupled
 - Tightly Coupled
- (Depending on the integration of the visual info)



Scaramuzza, D. and Zhang, Z., 2019. Visual-Inertial Odometry of Aerial Robots. *arXiv preprint arXiv:1906.03289*.

Visual Inertial Odometry VIO

- Kalman state:

$$\mathbf{X}_i = [\mathbf{T}_{WI}^i, \mathbf{v}_{WI}^i, \mathbf{b}_a^i, \mathbf{b}_g^i], \quad i = 1, 2, 3, \dots, N$$

- , where T_{w1}^i is the 6-DoF pose of the IMU, v_{w1}^i is the velocity of the IMU, b_a^i and B_g^i are the biases of the accelerometer and gyroscope respectively.

- a single moving camera allows us to measure the geometry of the 3D scene and the camera motion up to an unknown metric scale:
 - the projection function satisfies project(p) = project(s· p) for an arbitrary scalar s and an arbitrary point p;
 - a single IMU, instead, renders metric scale and gravity observable (due to the presence of gravity)

VIO, notable examples

- MSCKF, Multi-State Constraint Kalman Filter (MSCKF) –tightly
- OKVIS (Leutenegger et al 2013, 2015) - Open Keyframe-based Visual-Inertial SLAM (OKVIS) –tightly
- Robust Visual Inertial Odometry (ROVIO) is a visual-inertial state estimator based on an extended – tightly Kalman Filter (EKF)
- VINS-Mono –loosely
- SVO+MSF –loosely

- Visual Odometry
 - 3D-3D
 - 3D-2D
 - 2D-2D
- Local Bundle Adjustment
- Visual Inertial Odometry -VIO
 - Loosely Coupled EKF
 - Tightly Coupled EKF

Perception for Autonomous Systems 31392:

Visual Odometry

Lecturer: Evangelos Boukas—PhD

Perception for Autonomous Systems

Lecture 5 - Visual Odometry (12/04/2021)

Outline/Content:

- Orientation (Attitude) Representations
- Relative Pose Estimation
 - 3D registration
 - PnP
 - Least Squares - SVD (Singular Value Decomposition)
- Visual Odometry
 - 3D-3D
 - 3D-2D
 - 2D-2D
- Local Bundle Adjustment
- Visual Inertial Odometry (VIO)
 - Loosely Coupled EKF (Extended Kalman Filter)
 - Tightly Coupled EKF

Reading Material:

- Scaramuzza, D. and Fraundorfer, F., 2011. Visual odometry [tutorial]. IEEE robotics & automation magazine, 18(4), pp.80-92. Fraundorfer, F. and Scaramuzza, D., 2012. Visual odometry: Part ii: Matching, robustness, optimization, and applications. IEEE Robotics & Automation Magazine, 19(2), pp.78-90. Solution of P3P Problem, by Tomas Werner on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

What is Visual Odometry and what it is not? [Lecture Slides - University of Toronto](#)

Definition 1: Visual Odometry (VO) concerns the use of cameras to estimate the Pose (position and orientation) of a mobile system, by observing the apparent motion of the "static" world.

Definition 2: Visual Odometry is the process of estimating the motion of a camera in real-time using successive images.

Definition 3: VO is the process of incrementally estimating the pose (position and orientation) of the vehicle by examining the changes that motion induces on the images of its onboard cameras.

VO Facts/Assumptions

Facts:

- VO does not provide a map of the environment neither it uses previous states of the world to improve its accuracy (SLAM).
- It has been proposed as an alternative to wheel odometry
- VO estimations are usually combined with other sensors
 - GPS, IMU, Laser, Wheel odometry

- VINS (Visual Search for Mobile Interface Design), VIO stands for Visual Inertial Odometry
- VO has had some extraordinary usages

Assumptions:

- Sufficient illumination is in the environment
- Enough texture to allow apparent motion to be extracted
- Sufficient scene overlap between consecutive frames
- VO assumes a static world where the only moving object is the mobile system
- Its accuracy is - assuming reasonable trajectories ~1%

Illustration of the VO Pipeline

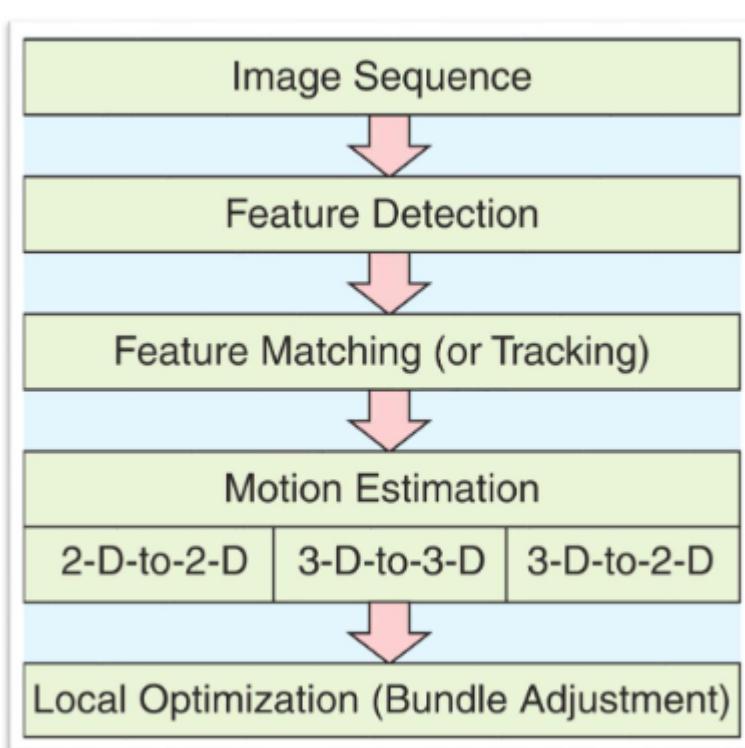
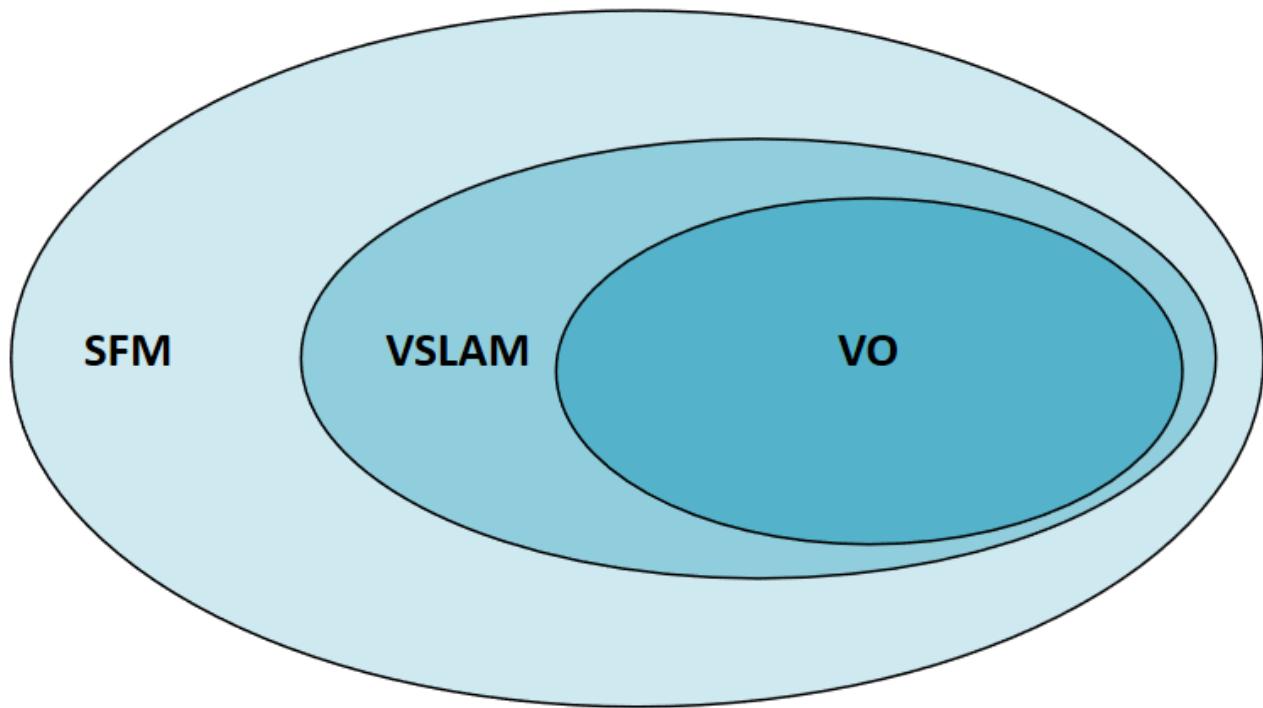


Image from Scaramuzza and Fraundorfer, 2011

**VO is neither SFM **(Structure from Motion) nor VSLAM (Visual SLAM) [Lecture Slides - University of Zurich](#)



SFM (Structure from Motion)

SFM is more general than VO and tackles the problem of 3D reconstruction and 6DOF pose estimation from unordered image sets

Structure from motion tries to solve also for the feature point as well:

- "Given Calibrated point projections of $p=1\dots N$ points in camera(of frame) $f=1\dots F$ x_p^f, y_p^f
- Find the rigid transformation $R^T t$ and the point's 3D position $F X_p = (X_p, Y_p, Z_p)$ which satisfies the projection equations

VO vs SFM

1. VO is a particular case of SFM
2. VO focuses on estimating the 3D motion of the camera sequentially (as a new frame arrives) and in real time
3. Terminology: sometimes SFM is used as a synonym of VO

V-Slam (Visual Slam)

Visual SLAM uses state estimation to exploit additional constraints and re-observations of the same areas(Loop-closures) to optimize the localization

VO vs Visual Slam

1. VO only aims to the local consistency of the trajectory
2. SLAM aims to the global consistency of the trajectory and of the map

3. VO can be used as a building block of SLAM
4. VO is SLAM before closing the loop
5. The choice between VO and V-SLAM depends on the tradeoff between performance and consistency, and simplicity in implementation
6. VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera.

Applications of VO:

- Motion estimation for vehicles
- Driver assistance
- Autonomy
- Point-Cloud Mapping
- VO to estimate the motion of a lidar during the collection of a single scan
- Reduce rolling shutter effect

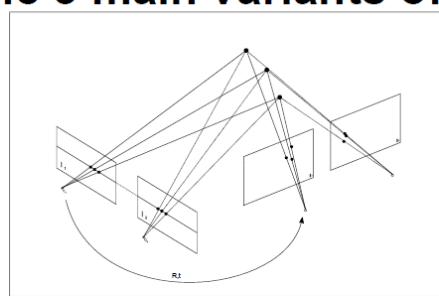
Challenges of VO:

- Robustness to lightning conditions
- Lack of features / non-overlapping images
- Without loop closure the estimate still drifts

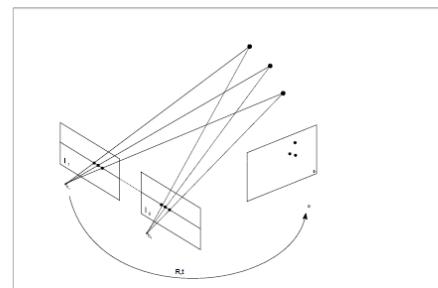
The 3 main variants of VO



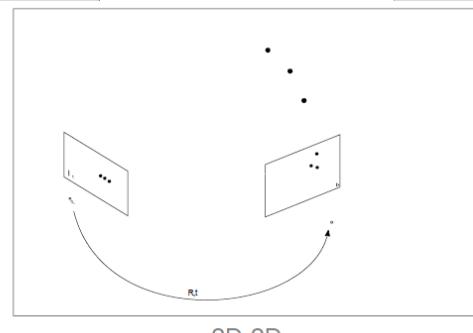
The 3 main variants of VO



3D-3D



3D-2D



2D-2D

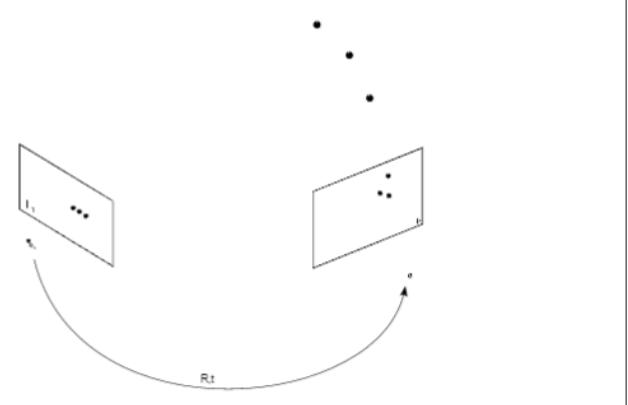
Visual Odometry 2D - to 2D



Visual Odometry 2D – to 2D

Algorithm 1. VO from 2-D-to-2-D correspondences.

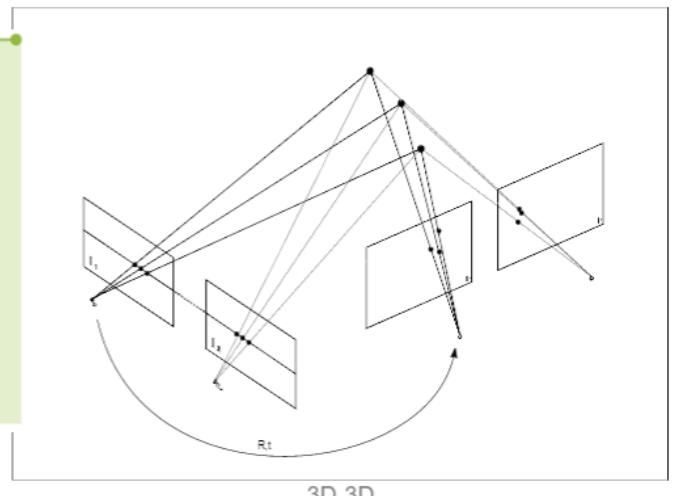
- 1) Capture new frame I_k
- 2) Extract and match features between I_{k-1} and I_k
- 3) Compute essential matrix for image pair I_{k-1}, I_k
- 4) Decompose essential matrix into R_k and t_k , and form T_k
- 5) Compute relative scale and rescale t_k accordingly
- 6) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 7) Repeat from 1).



Visual Odometry 3D - to 3D

Algorithm 2. VO from 3-D-to-3-D correspondences.

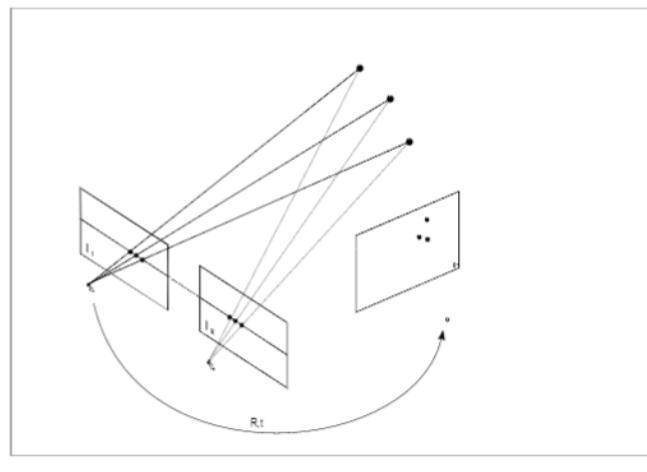
- 1) Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$
- 2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
- 3) Triangulate matched features for each stereo pair
- 4) Compute T_k from 3-D features X_{k-1} and X_k
- 5) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 6) Repeat from 1).



Visual Odometry 3D - to 2D

Algorithm 3. VO from 3-D-to-2-D Correspondences.

- 1) Do only once:
 - 1.1) Capture two frames I_{k-2}, I_{k-1}
 - 1.2) Extract and match features between them
 - 1.3) Triangulate features from I_{k-2}, I_{k-1}
- 2) Do at each iteration:
 - 2.1) Capture new frame I_k
 - 2.2) Extract features and match with previous frame I_{k-1}
 - 2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
 - 2.4) Triangulate all new feature matches between I_k and I_{k-1}
 - 2.5) Iterate from 2.1).



Type of correspondences	Monocular	Stereo
2D-2D	X	X
3D-3D		X
3D-2D	X	X

	Structure (scene geometry)	Motion (camera geometry)	Measurements
Pose Estimation	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

3D-3D is inferior to the other approaches due to the fact that the ICP algorithm is being used, which doesn't guarantee to find optimal solutions.

Orientation

Rotation Matrix:

- Positives (The king of Orientation)
 - Unique, no gimbal lock
- Negatives
 - No perturbation, interpolation, unintuitive

Euler angles:

- Positives
 - Minimal representation, intuitive

- Negatives
 - Gimbal lock, non commutative

Axis Angles:

- Positives
 - No gimbal lock, minimal representation, nice for perturbation, linear mapping to rotation matrix
- Negatives
 - Non linear "scaling" w.r.t. magnitude
 - exponential coordinates

Quaternions

- Positives
 - all the axis angle ones, smooth trajectory
- Negatives
 - No direct geometric representation

Motion tracking

- Two main approaches
 - Feature based
 - Optical flow
 - Feature based:
 - Calculate feature on both images
 - Match among the features or
 - Calculate features
 - Do block Matching around our initial point (for small motion)

Feature-based methods

1. Extract & match features (+RANSAC)
2. Minimize Reprojection error minimization

- ✓ Large frame-to-frame motions
- ✓ Accuracy: Efficient optimization of structure and motion (Bundle Adjustment)
- ✗ Slow due to costly feature extraction and matching
- ✗ Matching Outliers (RANSAC)

$$T_{k,k-1} = \arg \min_T \sum_i \| \mathbf{u}'_i - \pi(\mathbf{p}_i) \|_{\Sigma}^2$$

Direct methods

1. Minimize photometric error

$$T_{k,k-1} = \arg \min_T \sum_i \| I_k(\mathbf{u}'_i) - I_{k-1}(\mathbf{u}_i) \|_{\sigma}^2$$

where $\mathbf{u}'_i = \pi(T \cdot (\pi^{-1}(\mathbf{u}_i) \cdot d))$

[Jin,Favaro,Soatto'03] [Silveira, Malis, Rives, TRO'08], [Newcombe, Engel et al., ECCV'14], [Forster et al., ICRA'14]

- ✓ All information in the image can be exploited (precision, robustness)
- ✓ Increasing camera frame-rate reduces computational cost per frame
- ✗ Limited frame-to-frame motion
- ✗ Joint optimization of dense structure and motion too expensive

17

Motion tracking - Optical Flow [Youtube Lecture Slides - Carnegie Mellon University](#)

Definition: The most general (and challenging) version of motion estimation is to compute an independent estimate of motion at each pixel, which is generally known as optical (or optic) flow. This generally involves minimizing the brightness or color difference between corresponding pixels summed over the image.

Aim: Simply put, the goal of optical flow is basically to find a motion vector for each pixel within an image so that you then know in which direction each pixel is moving.

Different surfaces can move in different directions

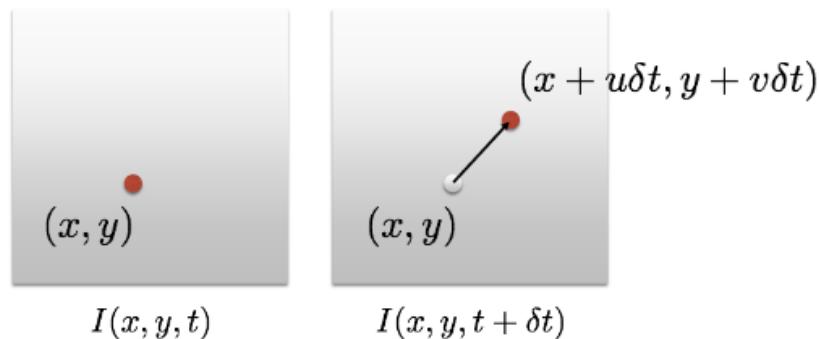
Assumption that need to be made:

- color/brightness constancy
 - otherwise, the optic flow algorithms would assume that the change of lightning is due to motion within the image.
- Small Motion, we analyze the motions for only very short changes in time (e.g. only 2 frames)
- enough features are needed (a glass sphere would not work (textureless, reflects a lot))

Application in which optical flow is used for:

- Image stabilization
- To see what global motion is happening

Small motion



Optical flow (velocities): (u, v) Displacement: $(\delta x, \delta y) = (u\delta t, v\delta t)$

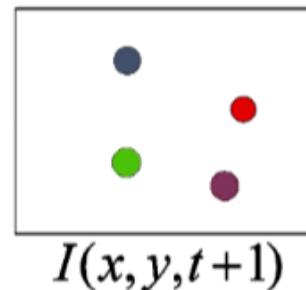
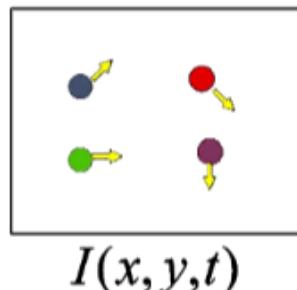
For a really small space-time step...

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

... the brightness between two consecutive image frames
is the same

Terminology: $I(x,y,t)$ is the image irradiance at time t at the image point (x,y) . $u(x,y)$ and $v(x,y)$ are the optical flow functions.

- Estimate the apparent motion
- Given a pixel in location $I(x,y,t)$, find the "nearby pixels with the same color"
 - Same intensity (in the local window)
 - Limited displacement

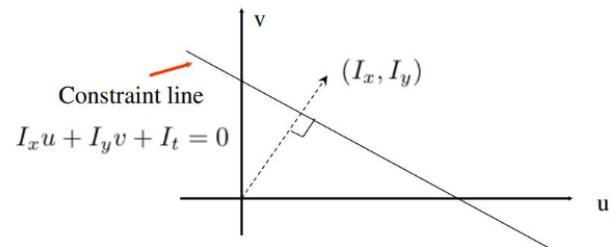


- $I(x + u, y + v, t + 1) = I(x, y, t)$
- $I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \rightarrow I_t + \nabla I \cdot (u, v) = 0 \rightarrow I_x u + I_y v + I_t = 0$
- This problem is the problem of "global Optical Flow" and is hard to solve due to *under definition*



Motion tracking – Optical Flow – Lukas Kanade

- The previous function is a line in the u,v space
- $I_x u + I_y v + I_t = 0$



- We can try to impose more constraints so the line becomes a point
- By assuming that for an image neighborhood we have constant "velocity": We want to minimize:

$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

What do the term of the brightness constancy equation represent?

flow velocities

$$I_x u + I_y v + I_t = 0$$

↑
Image gradients
(at a point p)

↑
temporal gradient

How do you compute these terms?

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference

Sobel filter

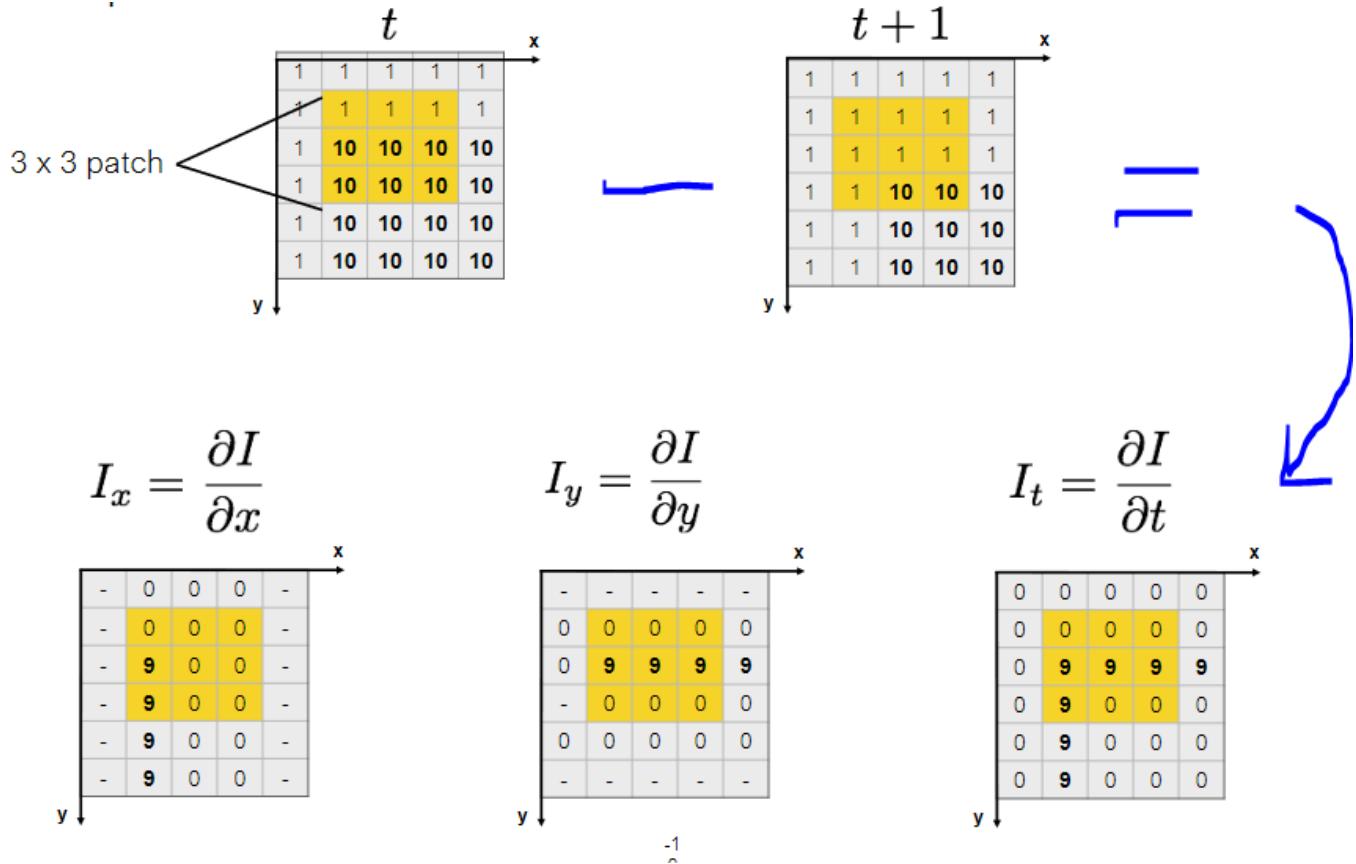
Derivative-of-Gaussian filter

We need to solve for this!

(this is the unknown in the optical flow problem)

frame differencing

...



Relative pose estimation [Lecture Slides - University of Toronto](#)

- Depends on the available data
 - 3D or 2D
 - (Eg: Do we work on a monocular camera or a stereo one?)
 - 3D registration - case where f_k and f_{k-1} are specified in 3D points
 - PCA
 - SVD, RANSAC

- ICP, combination of above
- What if we have correspondences?
- Rigid Transformation using RANSAC

3D-to-2D

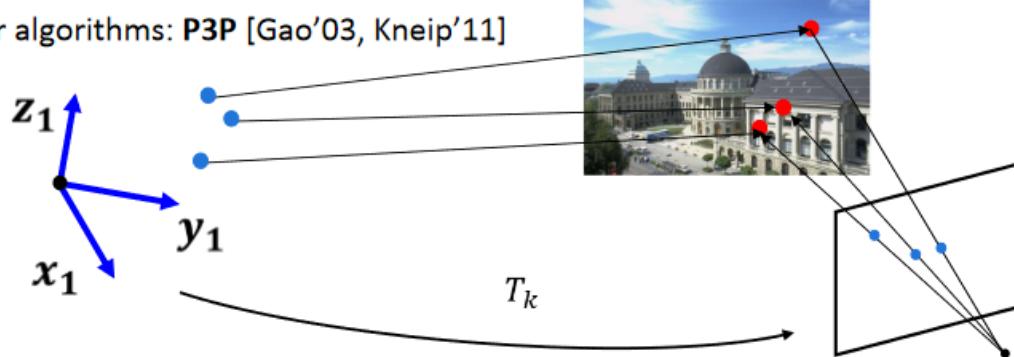
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from 3D Structure and Image Correspondences

- f_{k-1} is specified in 3D and f_k in 2D
- This problem is known as *camera resection* or PnP (perspective from n points)
- The minimal-case solution involves **3 correspondences** (+1 for disambiguating the 4 solutions)
- The solution is found by minimizing the reprojection error:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{T_k} \sum_i \|p_k^i - \hat{p}_{k-1}^i\|^2$$

- Popular algorithms: **P3P** [Gao'03, Kneip'11]



2D-to-2D

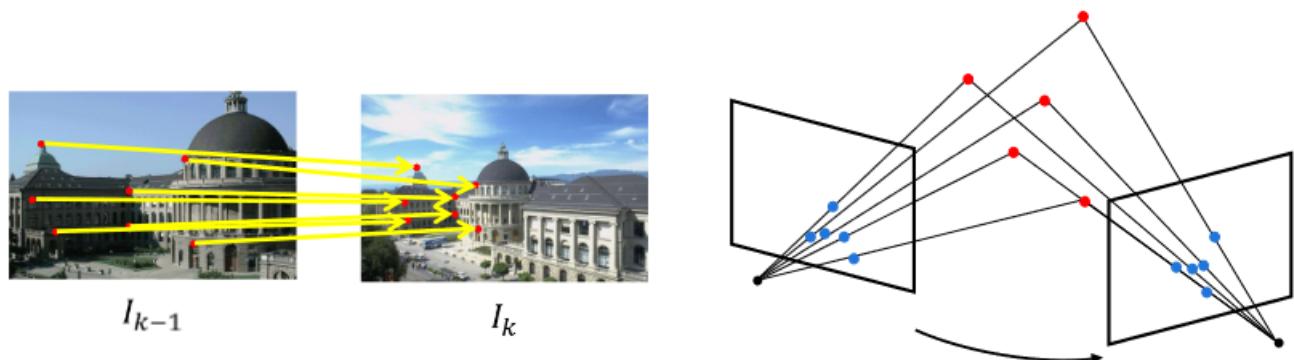
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from Image Feature Correspondences

- Both feature points f_{k-1} and f_k are specified in 2D
- The minimal-case solution involves 5-point correspondences
- The solution is found by minimizing the reprojection error:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- Popular algorithms: 8- and 5-point algorithms [Hartley'97, Nister'06]



Relative Pose estimation - 2D Image Points

The Essential Matrix can be computed directly from the image coordinates (using SVD).

At least 5 points needed! The more points, the better!

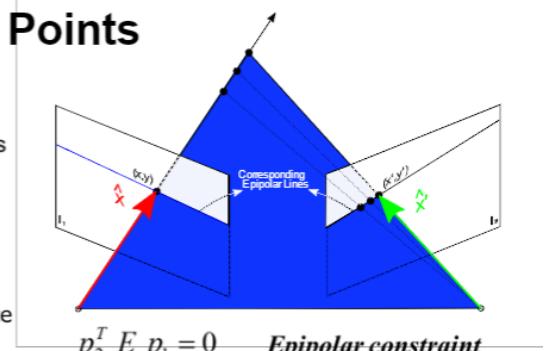
The Essential Matrix can be decomposed into R and t (again using SVD)

Let $p_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$, $p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$ be the coordinates one feature correspondence

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} e_{11} \\ \vdots \\ e_{33} \end{bmatrix}$$

$$p_2^T E p_1 = 0 \Rightarrow [x_1 x_2 \ y_1 x_2 \ z_1 x_2 \ x_1 y_2 \ y_1 y_2 \ z_1 y_2 \ x_1 z_2 \ y_1 z_2 \ z_1 z_2] E = 0$$

which can be solved with SVD



$$E = [t]_x R \quad \text{essential matrix}$$

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

3D-to-3D

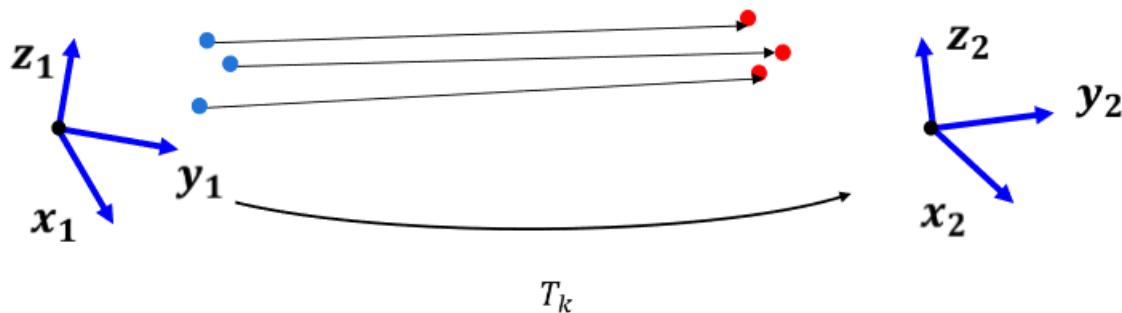
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from 3D-3D Point Correspondences (point cloud registration)

- Both f_{k-1} and f_k are specified in 3D. To do this, it is necessary to triangulate 3D points (e.g. use a stereo camera)
- The minimal-case solution involves 3 non-collinear correspondences
- The solution is found by minimizing the 3D-3D Euclidean distance:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- Popular algorithm: [Arun'87] for global registration, ICP for local refinement or Bundle Adjustment (BA)



What happens over time? [Lecture Slides - University of Toronto](#)

VO Working Principle

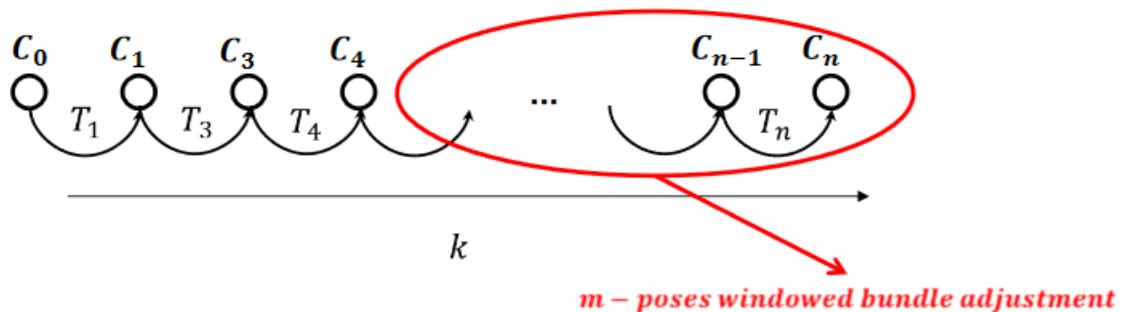
1. Compute the relative motion T_k from images I_{k-1} to image I_k

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

2. Concatenate them to recover the full trajectory

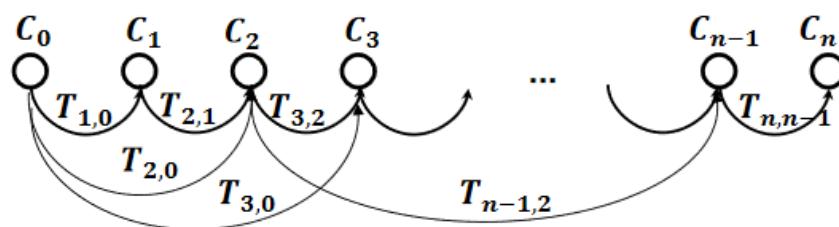
$$C_n = C_{n-1} T_n$$

3. An optimization over the last m poses can be done to refine locally the trajectory (Pose-Graph or Bundle Adjustment)



Pose-Graph Optimization

- So far we assumed that the transformations are between consecutive frames

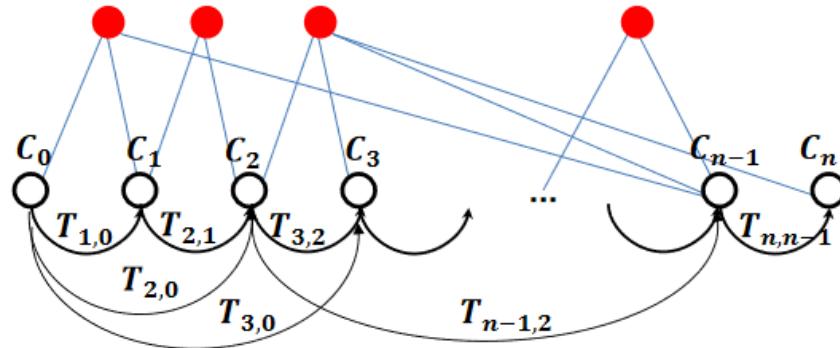


- Transformations can be computed also between non-adjacent frames T_{ij} (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve cameras poses by minimizing the following:

$$\sum_i \sum_j \|C_i - T_{ij} C_j\|^2$$

- For efficiency, only the last m keyframes are used
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: g2o, GTSAM, Google Ceres

Bundle Adjustment (BA)



- Similar to pose-graph optimization but it also optimizes 3D points

$$\arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- In order to not get stuck in local minima, the initialization should be close to the minimum
- Gauss-Newton or Levenberg-Marquadt can be used. For large graphs, efficient open-source software exists: GTSAM, g2o, Google Ceres can be used.

Bundle Adjustment vs Pose-graph Optimization

- BA is **more precise** than pose-graph optimization because it adds additional constraints (*landmark constraints*)
- But **more costly**: $O((qM + lN)^3)$ with M and N being the number of points and camera poses and q and l the number of parameters for points and camera poses. Workarounds:
 - A **small window size** limits the number of parameters for the optimization and thus makes real-time bundle adjustment possible.
 - It is possible to reduce the computational complexity by just optimizing over the camera parameters and keeping the 3-D landmarks fixed, e.g., (**motion-only BA**)

Improving the Accuracy of VO

- IMU
- Compass
- GPS
- Laser

Perception for Autonomous Systems

Lecture 5 - Visual Odometry (12/04/2021)

Outline/Content:

- Orientation (Attitude) Representations
- Relative Pose Estimation
 - 3D registration
 - PnP
 - Least Squares - SVD (Singular Value Decomposition)
- Visual Odometry
 - 3D-3D
 - 3D-2D
 - 2D-2D
- Local Bundle Adjustment
- Visual Inertial Odometry (VIO)
 - Loosely Coupled EKF (Extended Kalman Filter)
 - Tightly Coupled EKF

Reading Material:

- Scaramuzza, D. and Fraundorfer, F., 2011. Visual odometry [tutorial]. IEEE robotics & automation magazine, 18(4), pp.80-92. Fraundorfer, F. and Scaramuzza, D., 2012. Visual odometry: Part ii: Matching, robustness, optimization, and applications. IEEE Robotics & Automation Magazine, 19(2), pp.78-90. Solution of P3P Problem, by Tomas Werner on Intelligent Robots and Systems (IROS), Nice, France, September 22-26, 2008.

What is Visual Odometry and what it is not? [Lecture Slides - University of Toronto](#)

Definition 1: Visual Odometry (VO) concerns the use of cameras to estimate the Pose (position and orientation) of a mobile system, by observing the apparent motion of the "static" world.

Definition 2: Visual Odometry is the process of estimating the motion of a camera in real-time using successive images.

Definition 3: VO is the process of incrementally estimating the pose (position and orientation) of the vehicle by examining the changes that motion induces on the images of its onboard cameras.

VO Facts/Assumptions

Facts:

- VO does not provide a map of the environment neither it uses previous states of the world to improve its accuracy (SLAM).
- It has been proposed as an alternative to wheel odometry
- VO estimations are usually combined with other sensors
 - GPS, IMU, Laser, Wheel odometry

- VINS (Visual Search for Mobile Interface Design), VIO stands for Visual Inertial Odometry
- VO has had some extraordinary usages

Assumptions:

- Sufficient illumination is in the environment
- Enough texture to allow apparent motion to be extracted
- Sufficient scene overlap between consecutive frames
- VO assumes a static world where the only moving object is the mobile system
- Its accuracy is - assuming reasonable trajectories ~1%

Illustration of the VO Pipeline

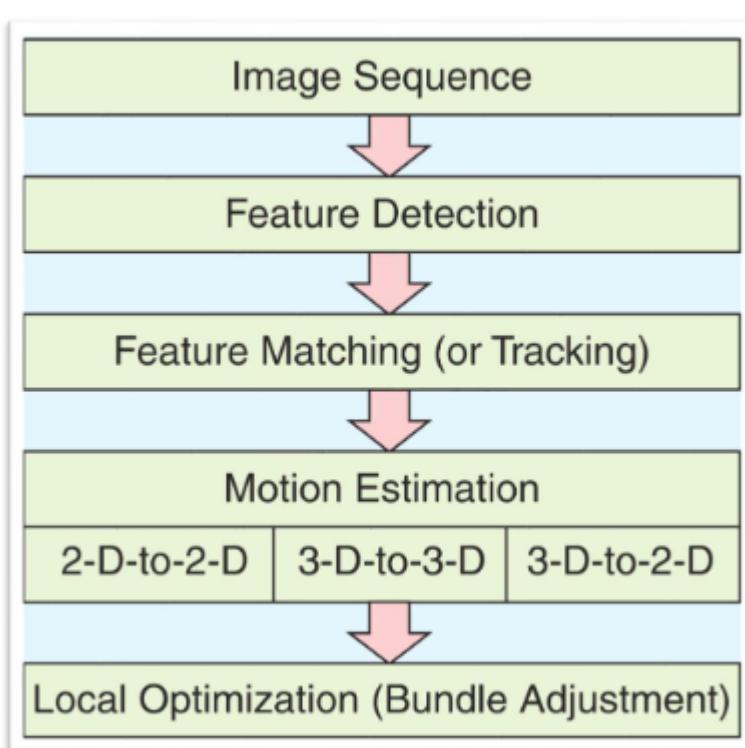
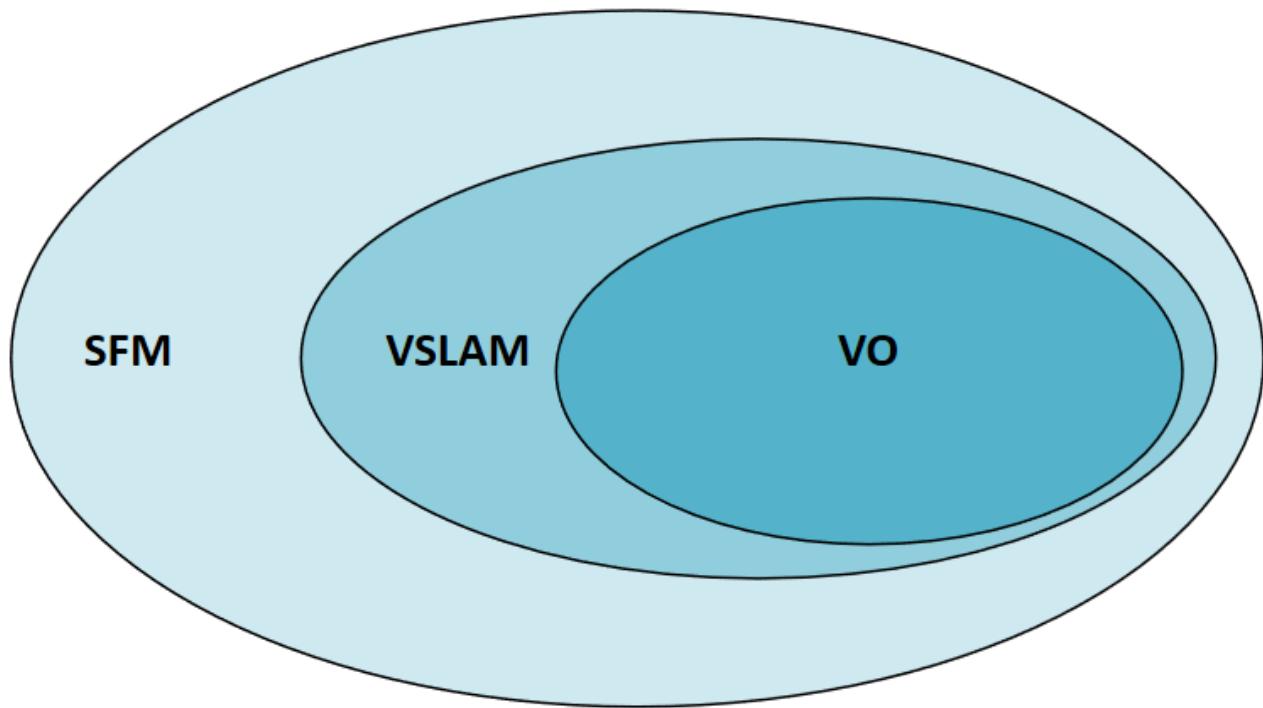


Image from Scaramuzza and Fraundorfer, 2011

**VO is neither SFM **(Structure from Motion) nor VSLAM (Visual SLAM) [Lecture Slides - University of Zurich](#)



SFM (Structure from Motion)

SFM is more general than VO and tackles the problem of 3D reconstruction and 6DOF pose estimation from unordered image sets

Structure from motion tries to solve also for the feature point as well:

- "Given Calibrated point projections of $p=1\dots N$ points in camera(of frame) $f=1\dots F$ x_p^f, y_p^f
- Find the rigid transformation $R^T t$ and the point's 3D position $F X_p = (X_p, Y_p, Z_p)$ which satisfies the projection equations

VO vs SFM

1. VO is a particular case of SFM
2. VO focuses on estimating the 3D motion of the camera sequentially (as a new frame arrives) and in real time
3. Terminology: sometimes SFM is used as a synonym of VO

V-Slam (Visual Slam)

Visual SLAM uses state estimation to exploit additional constraints and re-observations of the same areas(Loop-closures) to optimize the localization

VO vs Visual Slam

1. VO only aims to the local consistency of the trajectory
2. SLAM aims to the global consistency of the trajectory and of the map

3. VO can be used as a building block of SLAM
4. VO is SLAM before closing the loop
5. The choice between VO and V-SLAM depends on the tradeoff between performance and consistency, and simplicity in implementation
6. VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera.

Applications of VO:

- Motion estimation for vehicles
- Driver assistance
- Autonomy
- Point-Cloud Mapping
- VO to estimate the motion of a lidar during the collection of a single scan
- Reduce rolling shutter effect

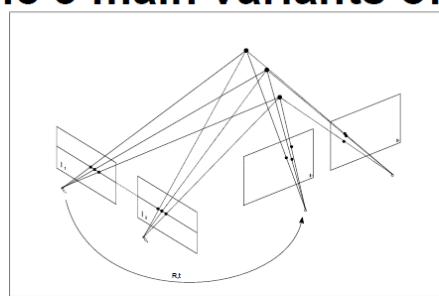
Challenges of VO:

- Robustness to lightning conditions
- Lack of features / non-overlapping images
- Without loop closure the estimate still drifts

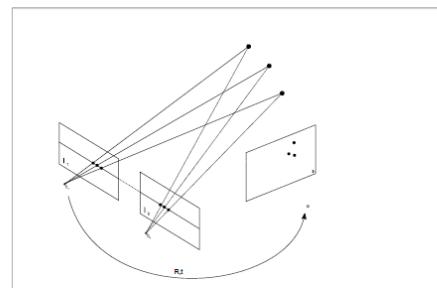
The 3 main variants of VO



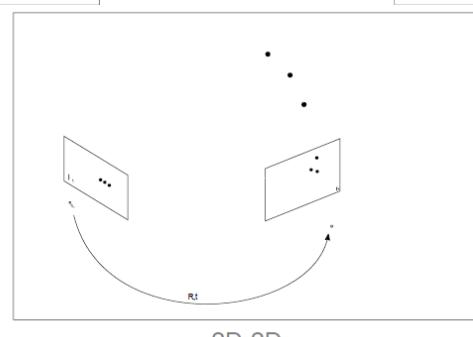
The 3 main variants of VO



3D-3D



3D-2D



2D-2D

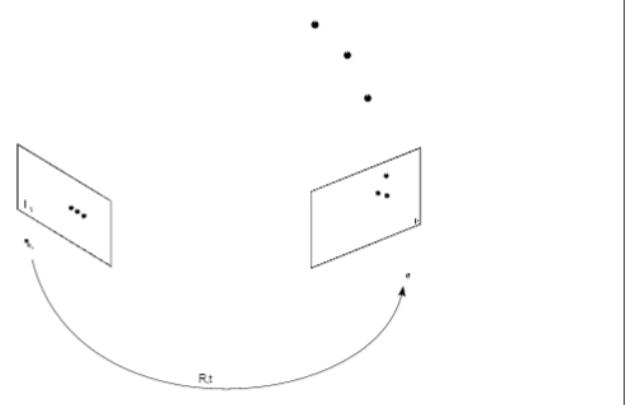
Visual Odometry 2D - to 2D



Visual Odometry 2D – to 2D

Algorithm 1. VO from 2-D-to-2-D correspondences.

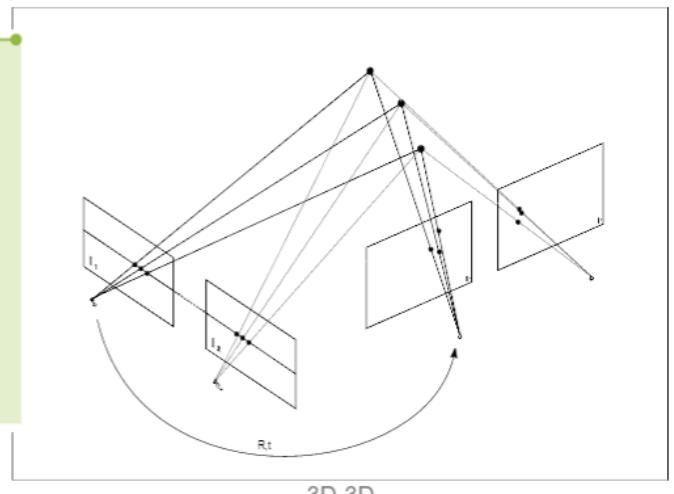
- 1) Capture new frame I_k
- 2) Extract and match features between I_{k-1} and I_k
- 3) Compute essential matrix for image pair I_{k-1}, I_k
- 4) Decompose essential matrix into R_k and t_k , and form T_k
- 5) Compute relative scale and rescale t_k accordingly
- 6) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 7) Repeat from 1).



Visual Odometry 3D - to 3D

Algorithm 2. VO from 3-D-to-3-D correspondences.

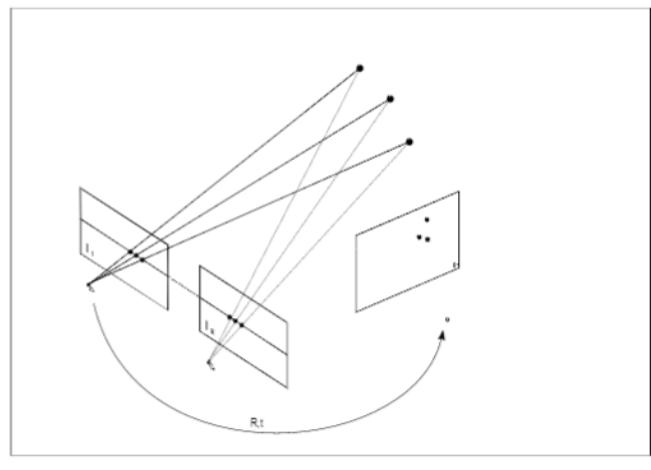
- 1) Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$
- 2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
- 3) Triangulate matched features for each stereo pair
- 4) Compute T_k from 3-D features X_{k-1} and X_k
- 5) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 6) Repeat from 1).



Visual Odometry 3D - to 2D

Algorithm 3. VO from 3-D-to-2-D Correspondences.

- 1) Do only once:
 - 1.1) Capture two frames I_{k-2}, I_{k-1}
 - 1.2) Extract and match features between them
 - 1.3) Triangulate features from I_{k-2}, I_{k-1}
- 2) Do at each iteration:
 - 2.1) Capture new frame I_k
 - 2.2) Extract features and match with previous frame I_{k-1}
 - 2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
 - 2.4) Triangulate all new feature matches between I_k and I_{k-1}
 - 2.5) Iterate from 2.1).



Type of correspondences	Monocular	Stereo
2D-2D	X	X
3D-3D		X
3D-2D	X	X

	Structure (scene geometry)	Motion (camera geometry)	Measurements
Pose Estimation	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

3D-3D is inferior to the other approaches due to the fact that the ICP algorithm is being used, which doesn't guarantee to find optimal solutions.

Orientation

Rotation Matrix:

- Positives (The king of Orientation)
 - Unique, no gimbal lock
- Negatives
 - No perturbation, interpolation, unintuitive

Euler angles:

- Positives
 - Minimal representation, intuitive

- Negatives
 - Gimbal lock, non commutative

Axis Angles:

- Positives
 - No gimbal lock, minimal representation, nice for perturbation, linear mapping to rotation matrix
- Negatives
 - Non linear "scaling" w.r.t. magnitude
 - exponential coordinates

Quaternions

- Positives
 - all the axis angle ones, smooth trajectory
- Negatives
 - No direct geometric representation

Motion tracking

- Two main approaches
 - Feature based
 - Optical flow
 - Feature based:
 - Calculate feature on both images
 - Match among the features or
 - Calculate features
 - Do block Matching around our initial point (for small motion)

Feature-based methods

1. Extract & match features (+RANSAC)
2. Minimize Reprojection error minimization

- ✓ Large frame-to-frame motions
- ✓ Accuracy: Efficient optimization of structure and motion (Bundle Adjustment)
- ✗ Slow due to costly feature extraction and matching
- ✗ Matching Outliers (RANSAC)

$$T_{k,k-1} = \arg \min_T \sum_i \| \mathbf{u}'_i - \pi(\mathbf{p}_i) \|_{\Sigma}^2$$

Direct methods

1. Minimize photometric error

$$T_{k,k-1} = \arg \min_T \sum_i \| I_k(\mathbf{u}'_i) - I_{k-1}(\mathbf{u}_i) \|_{\sigma}^2$$

where $\mathbf{u}'_i = \pi(T \cdot (\pi^{-1}(\mathbf{u}_i) \cdot d))$

[Jin,Favaro,Soatto'03] [Silveira, Malis, Rives, TRO'08], [Newcombe, Engel et al., ECCV'14], [Forster et al., ICRA'14]

- ✓ All information in the image can be exploited (precision, robustness)
- ✓ Increasing camera frame-rate reduces computational cost per frame
- ✗ Limited frame-to-frame motion
- ✗ Joint optimization of dense structure and motion too expensive

17

Motion tracking - Optical Flow [Youtube Lecture Slides - Carnegie Mellon University](#)

Definition: The most general (and challenging) version of motion estimation is to compute an independent estimate of motion at each pixel, which is generally known as optical (or optic) flow. This generally involves minimizing the brightness or color difference between corresponding pixels summed over the image.

Aim: Simply put, the goal of optical flow is basically to find a motion vector for each pixel within an image so that you then know in which direction each pixel is moving.

Different surfaces can move in different directions

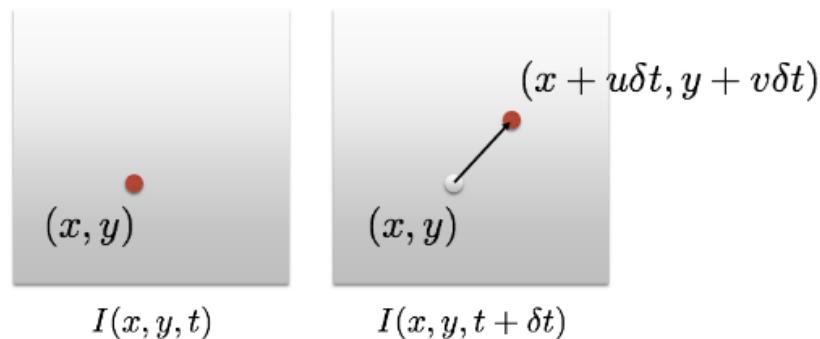
Assumption that need to be made:

- color/brightness constancy
 - otherwise, the optic flow algorithms would assume that the change of lightning is due to motion within the image.
- Small Motion, we analyze the motions for only very short changes in time (e.g. only 2 frames)
- enough features are needed (a glass sphere would not work (textureless, reflects a lot))

Application in which optical flow is used for:

- Image stabilization
- To see what global motion is happening

Small motion



Optical flow (velocities): (u, v) Displacement: $(\delta x, \delta y) = (u\delta t, v\delta t)$

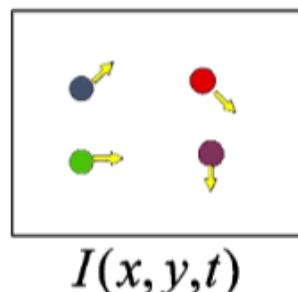
For a really small space-time step...

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

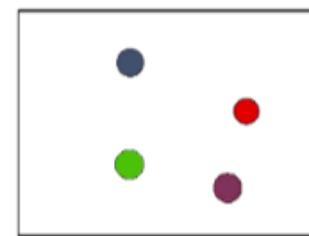
... the brightness between two consecutive image frames
is the same

Terminology: $I(x,y,t)$ is the image irradiance at time t at the image point (x,y) . $u(x,y)$ and $v(x,y)$ are the optical flow functions.

- Estimate the apparent motion
- Given a pixel in location $I(x,y,t)$, find the "nearby pixels with the same color"
 - Same intensity (in the local window)
 - Limited displacement



$I(x, y, t)$



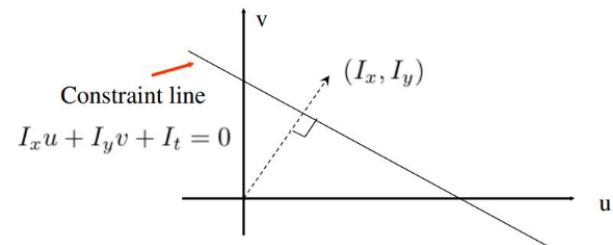
$I(x, y, t+1)$

- $I(x + y, t) = I(x + u, y + v, t + 1)$
- $I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \rightarrow I_t + \nabla I \cdot (u, v) = 0 \rightarrow I_x u + I_y v + I_t = 0$
- This problem is the problem of "global Optical Flow" and is hard to solve due to *under definition*



Motion tracking – Optical Flow – Lukas Kanade

- The previous function is a line in the u,v space
- $I_x u + I_y v + I_t = 0$



- We can try to impose more constraints so the line becomes a point
- By assuming that for an image neighborhood we have constant “velocity”: We want to minimize:

$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

What do the term of the brightness constancy equation represent?

flow velocities

$$I_x u + I_y v + I_t = 0$$

↑
Image gradients
(at a point p)

↑
temporal gradient

How do you compute these terms?

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

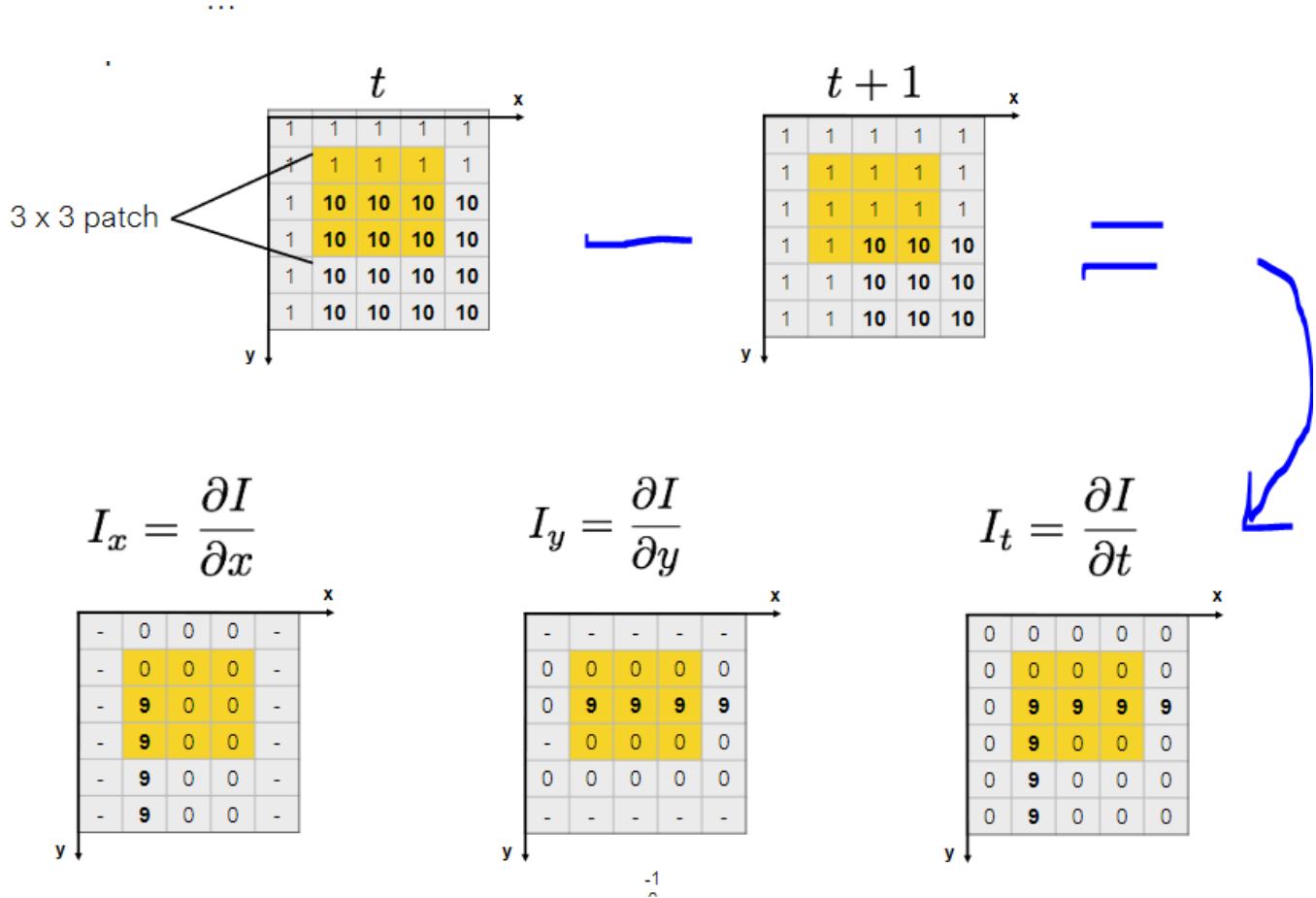
$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference
Sobel filter
Derivative-of-Gaussian filter

We need to solve for this!
(this is the unknown in the optical flow problem)

frame differencing



Relative pose estimation [Lecture Slides - University of Toronto](#)

- Depends on the available data
 - 3D or 2D
 - (Eg: Do we work on a monocular camera or a stereo one?)
 - 3D registration - case where \$f_k\$ and \$f_{k-1}\$ are specified in 3D points
 - PCA
 - SVD, RANSAC

- ICP, combination of above
- What if we have correspondences?
- Rigid Transformation using RANSAC

3D-to-2D

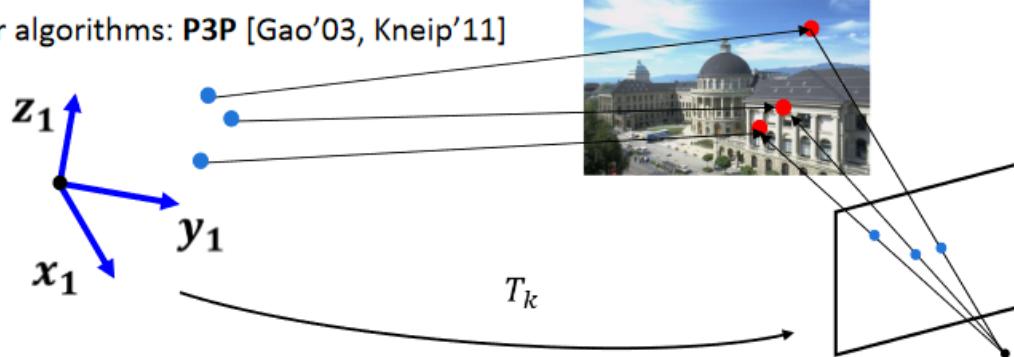
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from 3D Structure and Image Correspondences

- f_{k-1} is specified in 3D and f_k in 2D
- This problem is known as *camera resection* or PnP (perspective from n points)
- The minimal-case solution involves **3 correspondences** (+1 for disambiguating the 4 solutions)
- The solution is found by minimizing the reprojection error:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{T_k} \sum_i \|p_k^i - \hat{p}_{k-1}^i\|^2$$

- Popular algorithms: **P3P** [Gao'03, Kneip'11]



2D-to-2D

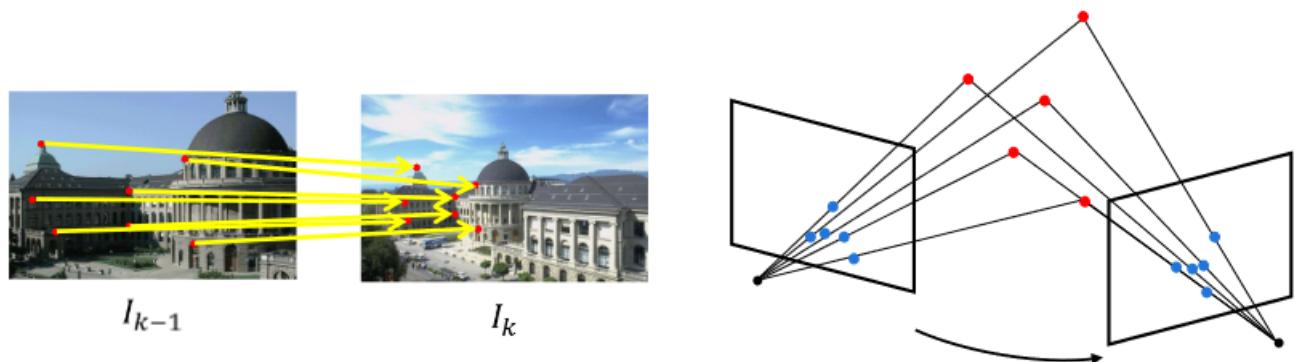
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from Image Feature Correspondences

- Both feature points f_{k-1} and f_k are specified in 2D
- The minimal-case solution involves 5-point correspondences
- The solution is found by minimizing the reprojection error:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- Popular algorithms: 8- and 5-point algorithms [Hartley'97, Nister'06]



Relative Pose estimation - 2D Image Points

The Essential Matrix can be computed directly from the image coordinates (using SVD).

At least 5 points needed! The more points, the better!

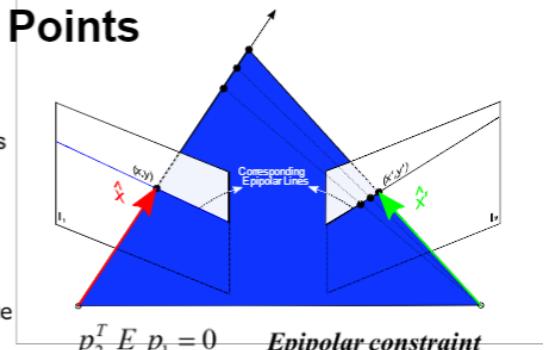
The Essential Matrix can be decomposed into R and t (again using SVD)

Let $p_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$, $p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$ be the coordinates one feature correspondence

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} e_{11} \\ \vdots \\ e_{33} \end{bmatrix}$$

$$p_2^T E p_1 = 0 \Rightarrow [x_1 x_2 \ y_1 x_2 \ z_1 x_2 \ x_1 y_2 \ y_1 y_2 \ z_1 y_2 \ x_1 z_2 \ y_1 z_2 \ z_1 z_2] E = 0$$

which can be solved with SVD



$$E = [t]_x R \quad \text{essential matrix}$$

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

3D-to-3D

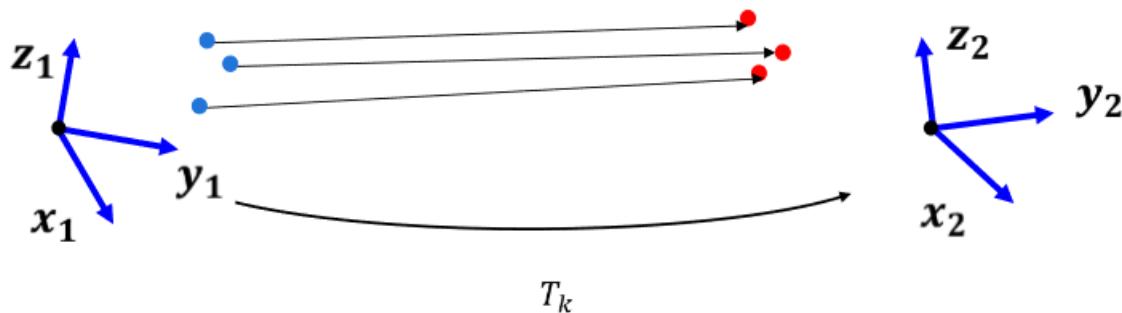
Motion estimation		
2D-2D	3D-2D	3D-3D

Motion from 3D-3D Point Correspondences (point cloud registration)

- Both f_{k-1} and f_k are specified in 3D. To do this, it is necessary to triangulate 3D points (e.g. use a stereo camera)
- The minimal-case solution involves 3 non-collinear correspondences
- The solution is found by minimizing the 3D-3D Euclidean distance:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- Popular algorithm: [Arun'87] for global registration, ICP for local refinement or Bundle Adjustment (BA)



What happens over time? [Lecture Slides - University of Toronto](#)

VO Working Principle

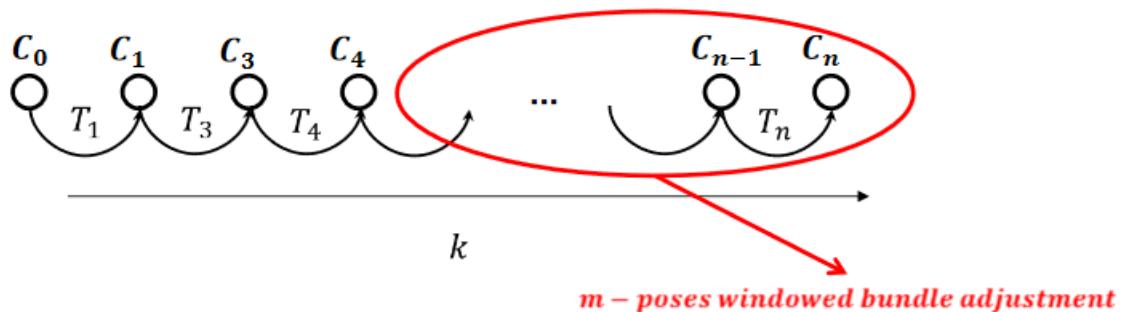
1. Compute the relative motion T_k from images I_{k-1} to image I_k

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

2. Concatenate them to recover the full trajectory

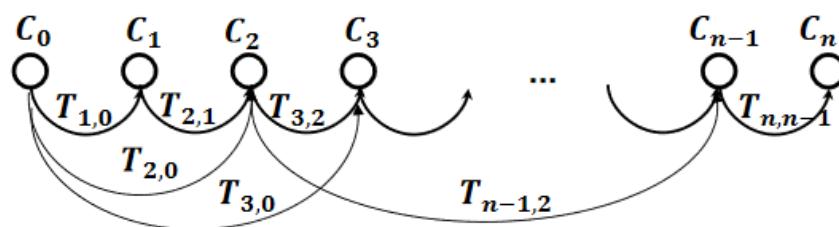
$$C_n = C_{n-1} T_n$$

3. An optimization over the last m poses can be done to refine locally the trajectory (Pose-Graph or Bundle Adjustment)



Pose-Graph Optimization

- So far we assumed that the transformations are between consecutive frames

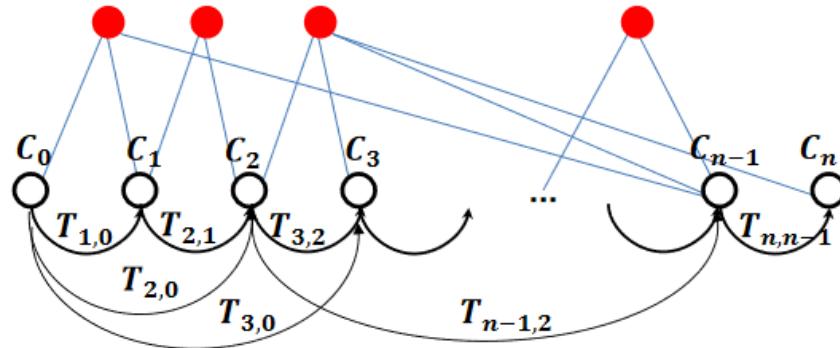


- Transformations can be computed also between non-adjacent frames T_{ij} (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve cameras poses by minimizing the following:

$$\sum_i \sum_j \|C_i - T_{ij} C_j\|^2$$

- For efficiency, only the last m keyframes are used
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: g2o, GTSAM, Google Ceres

Bundle Adjustment (BA)



- Similar to pose-graph optimization but it also optimizes 3D points

$$\arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2$$

- In order to not get stuck in local minima, the initialization should be close to the minimum
- Gauss-Newton or Levenberg-Marquadt can be used. For large graphs, efficient open-source software exists: GTSAM, g2o, Google Ceres can be used.

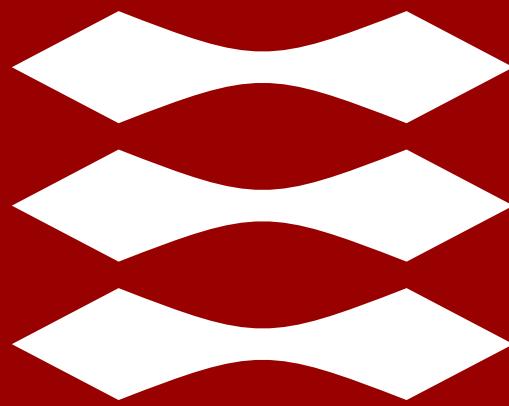
Bundle Adjustment vs Pose-graph Optimization

- BA is **more precise** than pose-graph optimization because it adds additional constraints (*landmark constraints*)
- But **more costly**: $O((qM + lN)^3)$ with M and N being the number of points and camera poses and q and l the number of parameters for points and camera poses. Workarounds:
 - A **small window size** limits the number of parameters for the optimization and thus makes real-time bundle adjustment possible.
 - It is possible to reduce the computational complexity by just optimizing over the camera parameters and keeping the 3-D landmarks fixed, e.g., (**motion-only BA**)

Improving the Accuracy of VO

- IMU
- Compass
- GPS
- Laser

DTU



Perception for Autonomous Systems 31392:

Visual SLAM

Simultaneous Localization & Mapping

Lecturer: Evangelos Boukas—PhD

Outline

- Sum up Localization from last time
- Some terminology
- Pose-Landmark Graph Slam
- Example of Linear 1D SLAM
- Non-Linear Optimization approaches
- Bundle Adjustment
- Visual Slam System architecture
- ORBSLAM

Visual Odometry is great

- Lets Sum Up What we did last time
- We performed 3D-to-2D



Visual Odometry is great

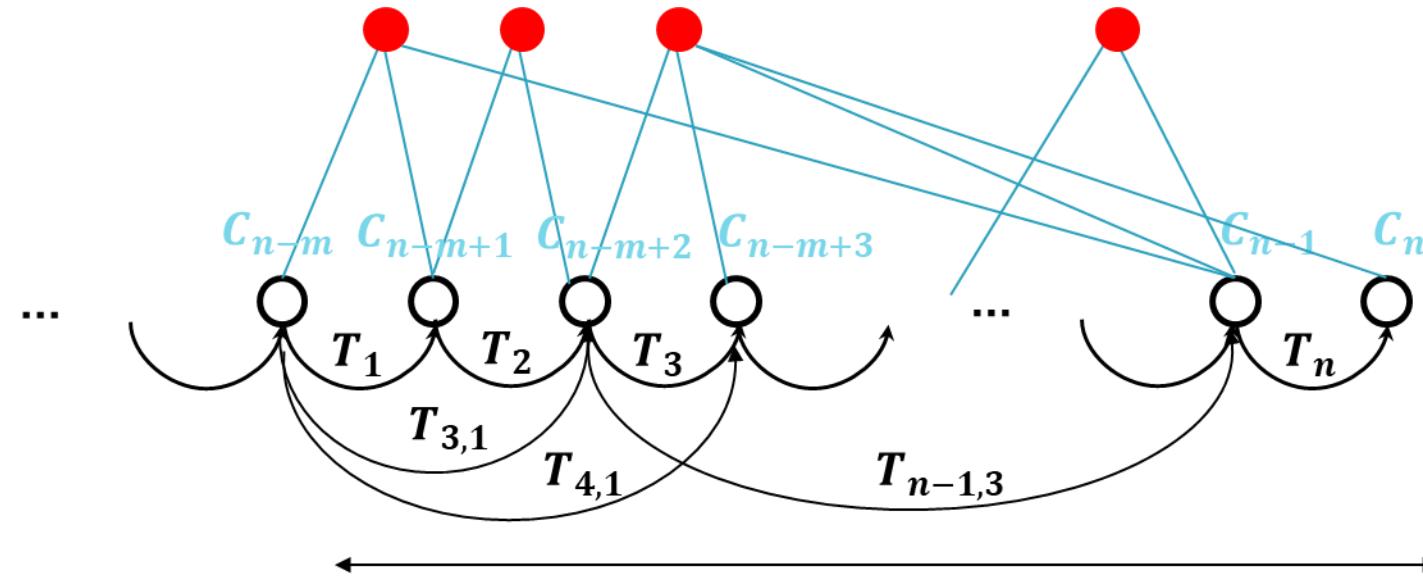
- Lets Sum Up What we did last time
- We performed 3D-to-2D



Visual Odometry is great

- Anything more we mentioned?
 - Windowed Bundle Adjustment (BA)

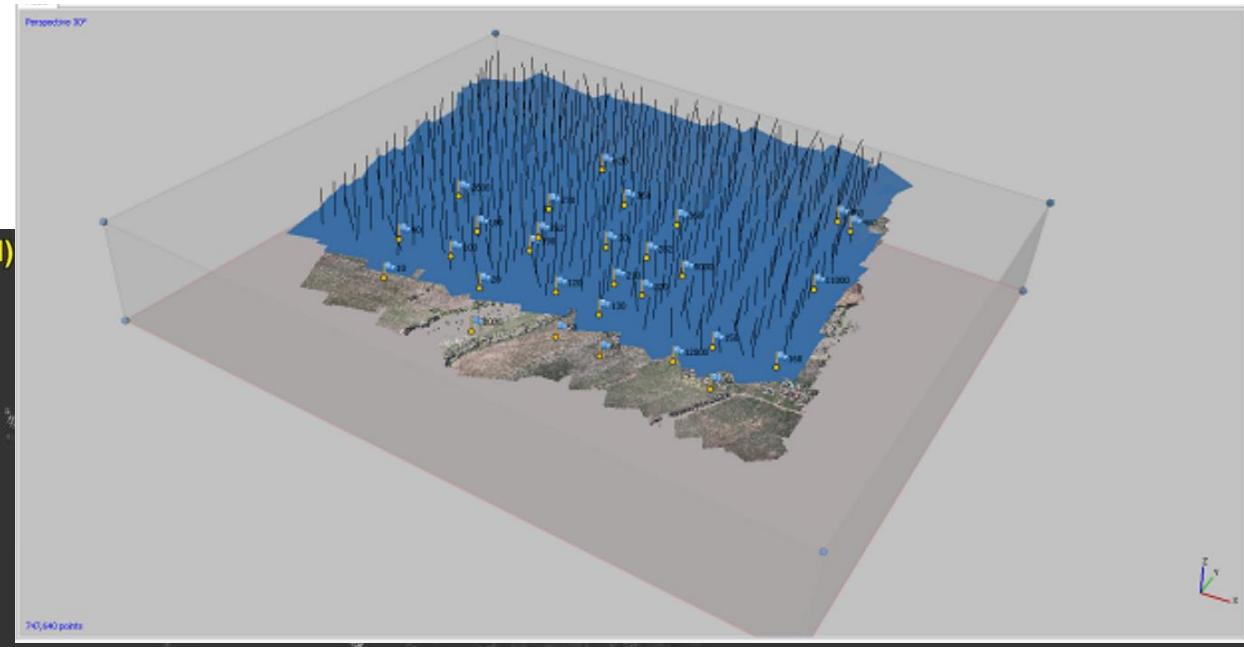
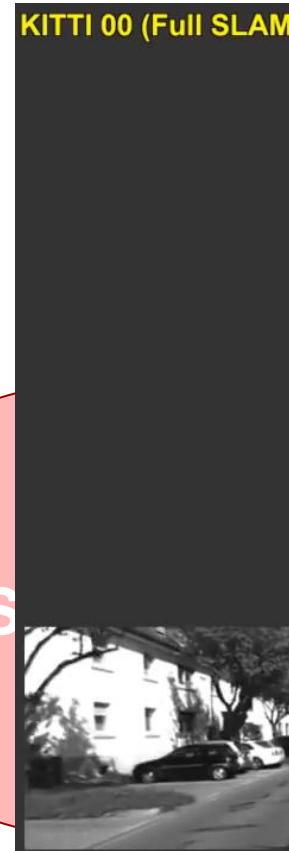
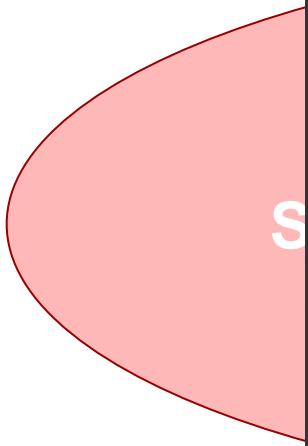
Windowed Bundle Adjustment (BA)



- Similar to pose-optimization but it also optimizes 3D points m
- In order to not get stuck in local minima, the initialization should be close the minimum
- Levenberg-Marquadt can be used

Formal definitions

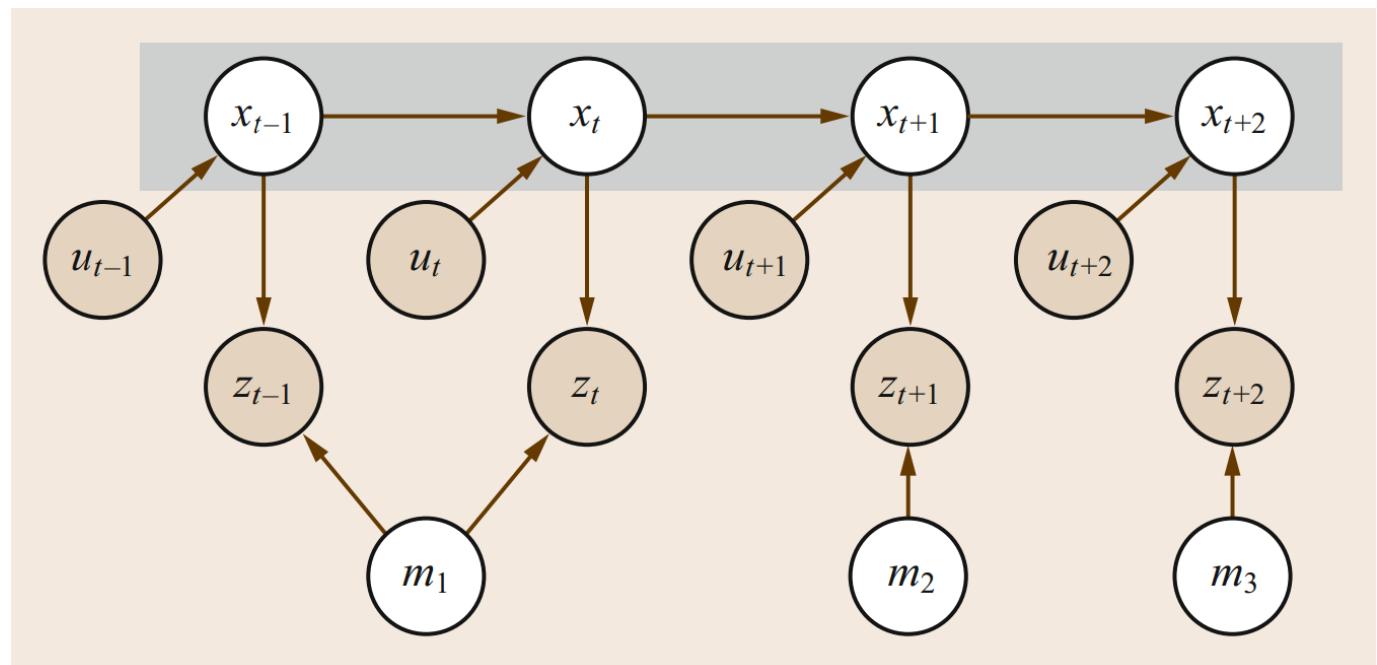
- Visual Odometry
- Structure from Motion (SfM)
- Bundle Adjustment
- Visual SLAM



- Sum up Localization from last time
- Some terminology
- Pose-Landmark Graph Slam
- **Example of Linear 1D SLAM**
- Non-Linear Optimization approaches
- Bundle Adjustment
- Visual Slam System architecture
- ORBSLAM

Pose-Landmark Graph-Slam

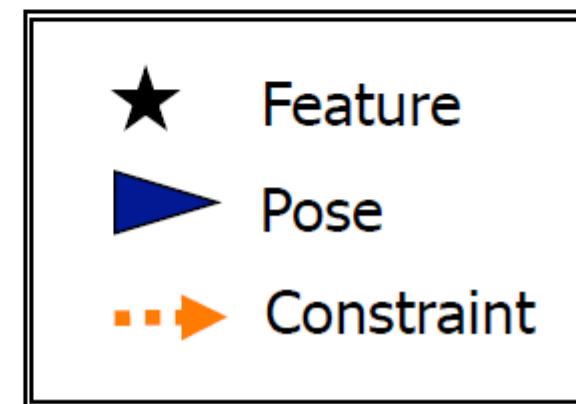
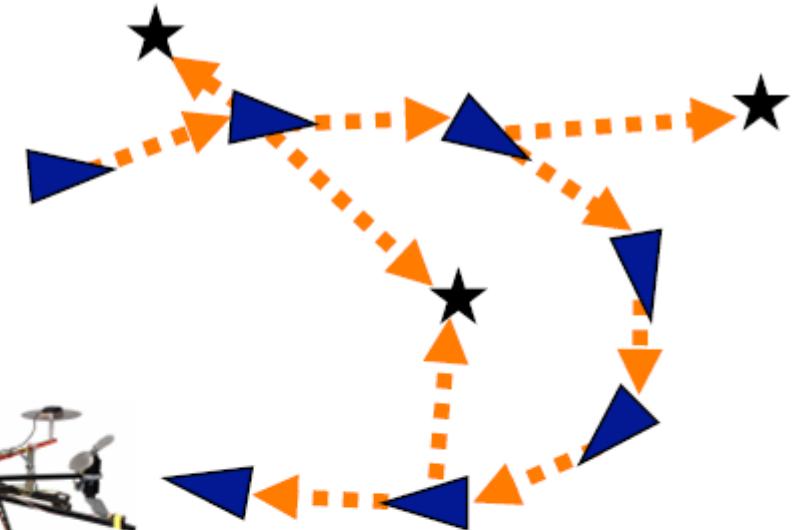
- SLAM problem depicted as Bayes network graph
- At each **location** x_t
- Observes a nearby feature in the **map** $m = \{m_1; m_2; m_3\}$
- Movement u_t
- An arrow defines causal relationship



Graph-Based SLAM

Definition

- Use a graph to represent the problem
- Nodes represent:
 - poses or
 - locations
- Edges Represent:
 - Landmark observations
 - Odometry Measurements
- The minimization optimizes the landmark locations and robot poses



Graph-Based SLAM: Build the graph and find a node configuration that minimize the error introduced by the constraints

Graph-Based SLAM (intuition of optimization)

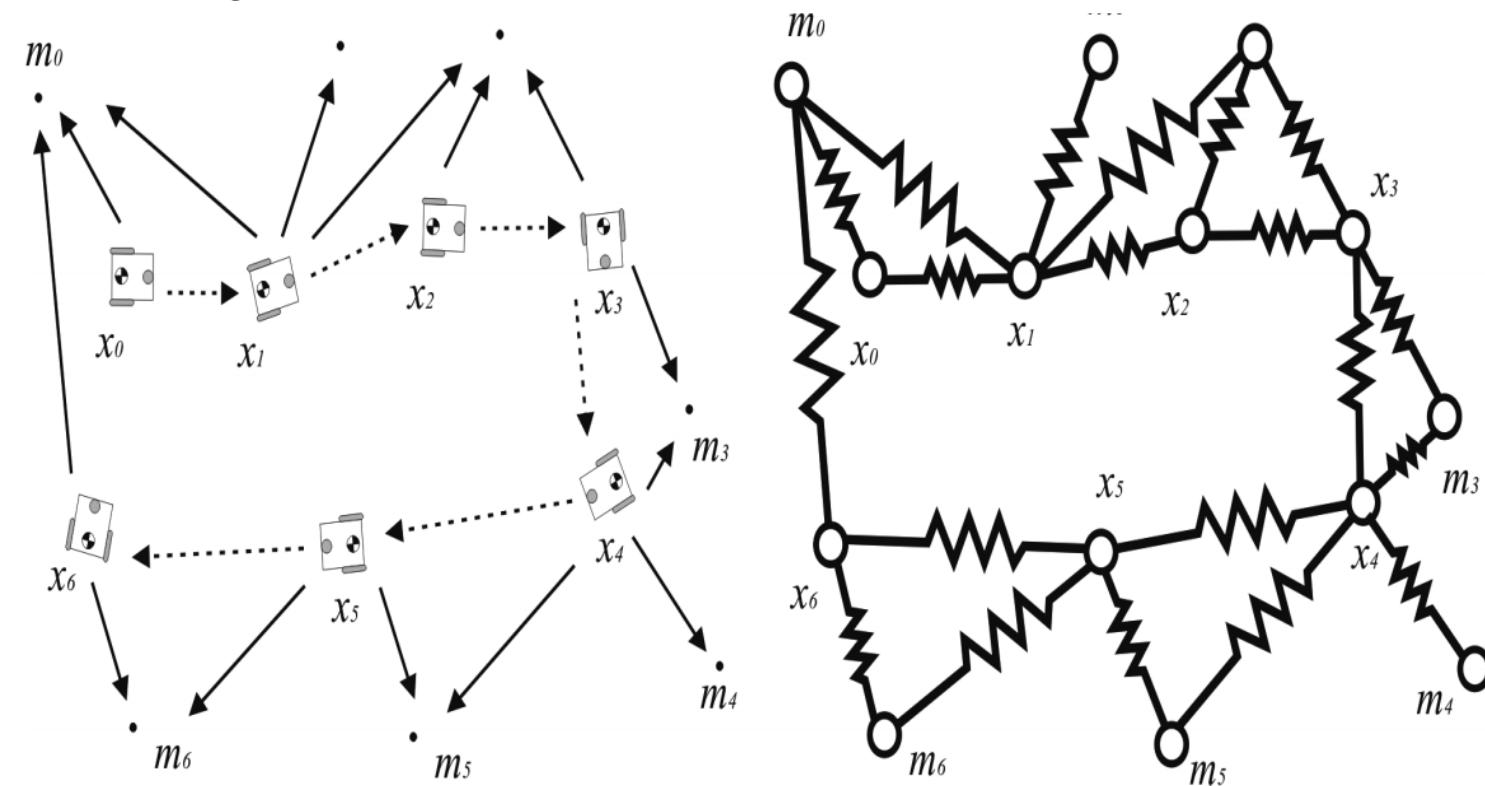
- Observing previously seen areas generates constraints between non-successive poses
- Treat constraints (motion and measurement) as “soft” elastic springs
- Want to minimize the total energy in the springs

We can define the error as follows

- Expected observation (2D sensor)

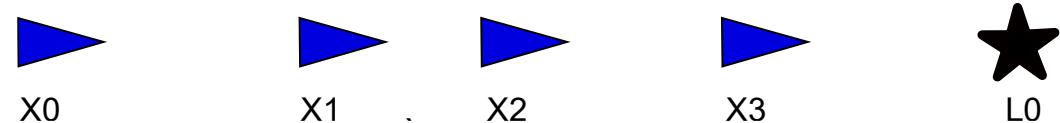
– With the error: $e_{ij}(x_i, x_j) = \hat{z}$

$$= I$$



1D Linear SLAM

- In the linear case we can solve as follows:



- First construct all constrains

- Absolute Constrain:

$$X(0) = Q - \text{starting position}$$

- Movement Constrains:

$$X(t) = X(t-1) + D_x(t)$$

- Measurement constrains:

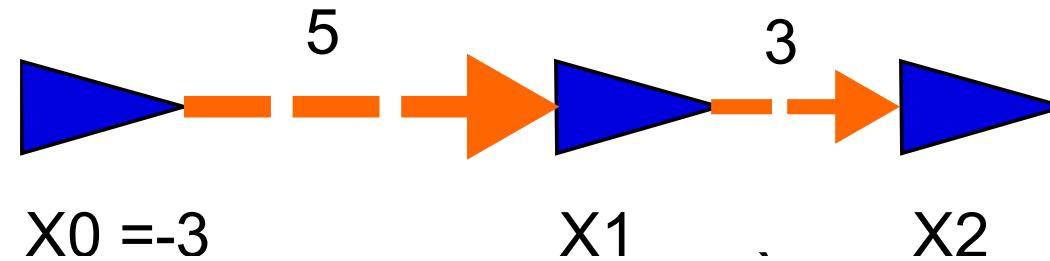
$$L(k) = X(t) + N$$

- *Then, solve linear equations*

1D Linear SLAM – case 1

- Case 1 - Exact solution exists:

$$\begin{bmatrix} & \\ & \end{bmatrix} \cdot \begin{bmatrix} & \\ & \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$



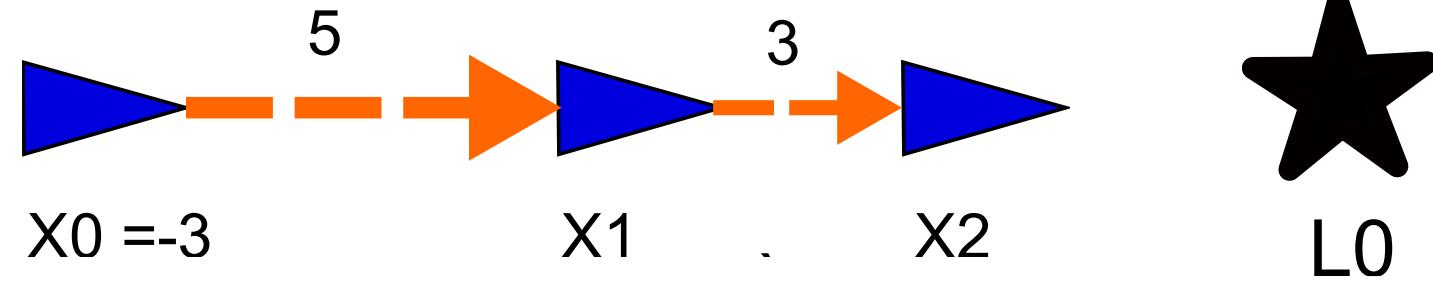
$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \end{bmatrix}$$

$$A^*X=B \quad X=A^{-1}*B \quad A^{-1}=\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$x = [-3 \ 2 \ 5]$$

1D Linear SLAM – case 2

- Case 2 – Overdefined problem:
 - X_0 sees L_0 at distance 10
 - X_1 sees L_0 at distance 5
 - X_2 sees L_0 at distance 2



$$\left[\quad \quad \right] \cdot \left[\quad \quad \right] = \left[\quad \quad \right]$$

$$A^*X = B \quad X = A^{-1} * B \quad A^{-1} = [?]$$

$$X = (A^T * A)^{-1} * A^T * B$$

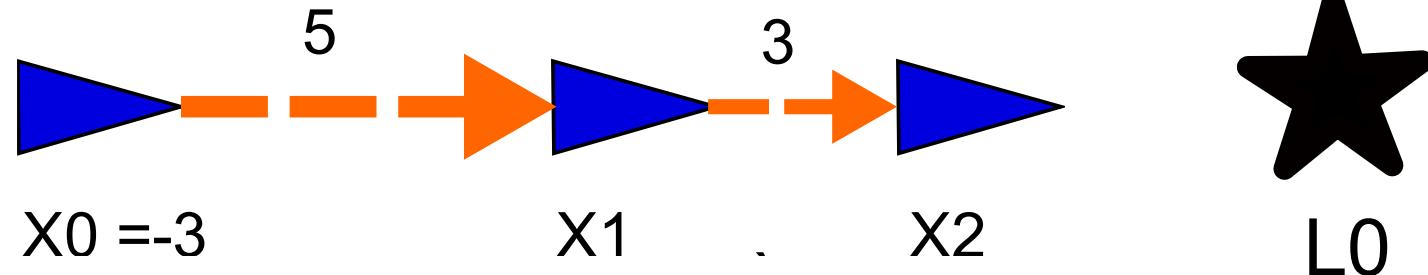
$$x = [-3 \ 2 \ 5 \ 7]$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ L_0 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \\ -10 \\ -5 \\ -2 \end{bmatrix}$$

We infer a consistent landmark position

1D Linear SLAM – case 3

- Case 3 – Inconsistent Measurements :
 - X_0 sees L_0 at distance 10
 - X_1 sees L_0 at distance 5
 - X_2 sees L_0 at distance 1 (**Wrong**)



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ L_0 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \\ -10 \\ -5 \\ -1 \end{bmatrix}$$

$$X = (A^T * A)^{-1} * A^T * B$$

$$x = [-3 \ 2.125 \ 5.5 \ 6.875]$$

We handled inaccurate measurements

1D Linear SLAM – case 4

Case 4 – Inconsistent Measurements
with Confidence Matrix:

- Linear Least Squares allows us to include a weighting of each linear constraint.
- We can include weights in the computation
- We weight each constraint by a diagonal matrix where the weights are 1/variance for each constraint.
- Let's say X_2 variance is 5

$$X = (A^T * W * A)^{-1} * A^T * W * B$$

$$x = \begin{bmatrix} -3 & 2.18 & 5.71 & 6.82 \end{bmatrix}$$

$$\begin{array}{ccccc} X_0 = -3 & & X_1 & & X_2 \\ \left[\begin{array}{rrrr} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{array} \right] & \cdot & \left[\begin{array}{c} X_0 \\ X_1 \\ X_2 \\ L_0 \end{array} \right] & = & \left[\begin{array}{c} -3 \\ 5 \\ 3 \\ -10 \\ -5 \\ -1 \end{array} \right] \\ W = & & \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{array} \right] & & \end{array}$$

Why did the estimation just become worse??

What about non-Linear Least Squares?

- Large number of geometric problems in computer vision are non-linear least-squares problems.

$$\mathbf{x} = \mathbf{h}(\theta)$$

where $\mathbf{h} : \mathbf{R}^n \rightarrow R^m$.

- \mathbf{x} is the measurement vector, θ is the parameter vector.
- Write $\mathbf{f}(\theta) = \mathbf{h}(\theta) - \mathbf{x}$.
- We desire to minimize

$$\|\mathbf{f}(\theta)\|^2$$

over all choices of parameter θ .

Gauss Newton Solution

1. Start from an initial value θ_0 .
2. At step i assume a linear approximation for the function at θ_i

$$\mathbf{f}(\theta_i + \Delta) = \mathbf{f}(\theta_i) + \mathbf{f}_\theta \Delta \text{ where } \mathbf{f}_\theta = \partial \mathbf{f} / \partial \theta = \mathbf{J}$$

3. Solve

$$\mathbf{f}(\theta_i + \Delta) = \mathbf{f}(\theta_i) + \mathbf{J}\Delta = 0$$

or

$$\mathbf{J}\Delta = -\mathbf{f}(\theta_i)$$

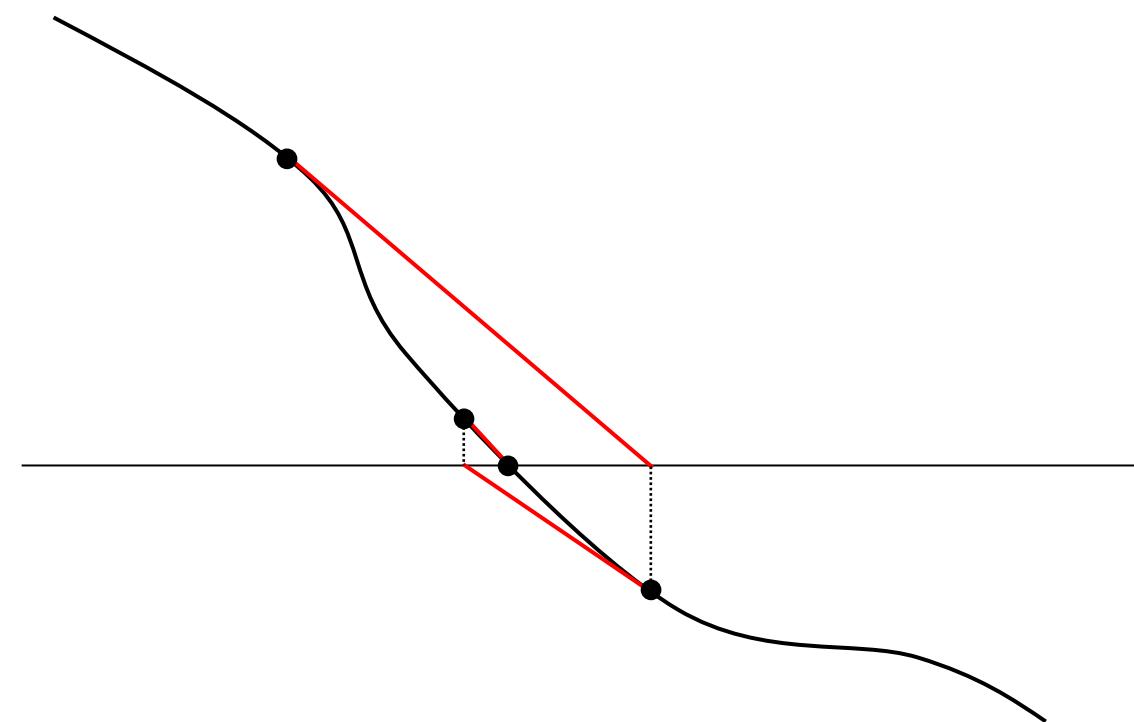
4. This is a linear least-squares problem (solve for Δ):

$$\mathbf{J}^\top \mathbf{J}\Delta = \mathbf{J}^\top \mathbf{f}(\theta_i)$$

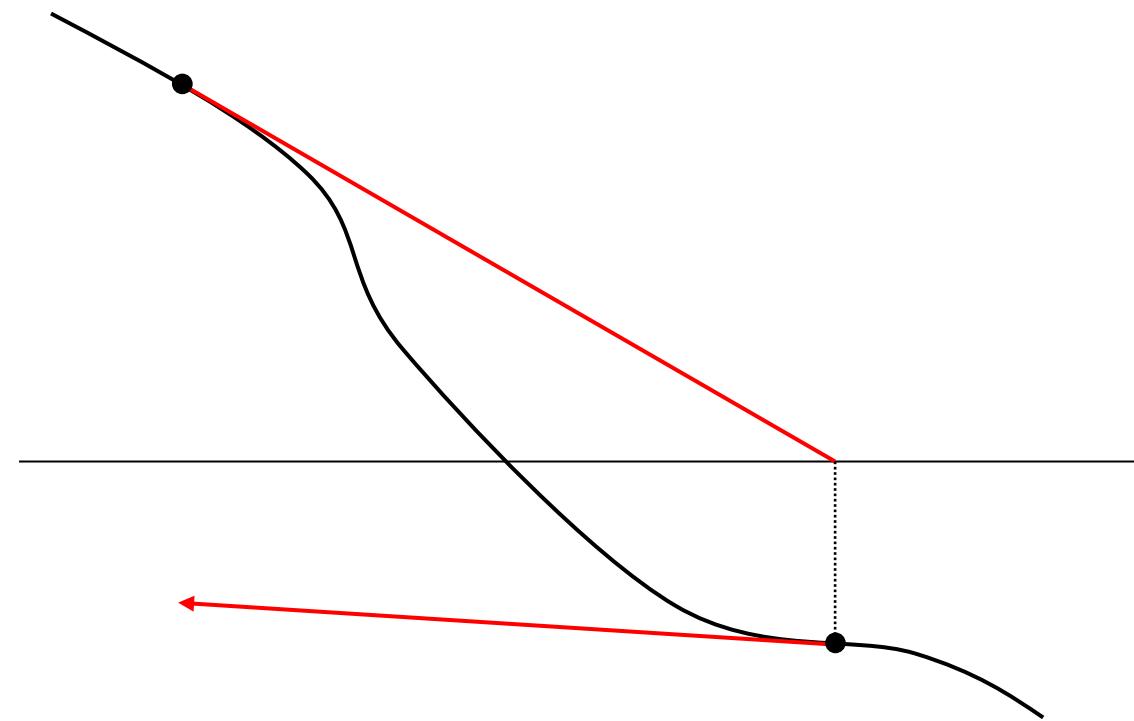
5. Then set $\theta_{i+1} = \theta_i + \Delta$.

Gauss-Newton update equation

$$\mathbf{J}^\top \mathbf{J}\Delta = -\mathbf{J}^\top \mathbf{f}$$



1D Gauss-Newton (Newton)
iteration.



1D Gauss-Newton (Newton)
iteration (failure)

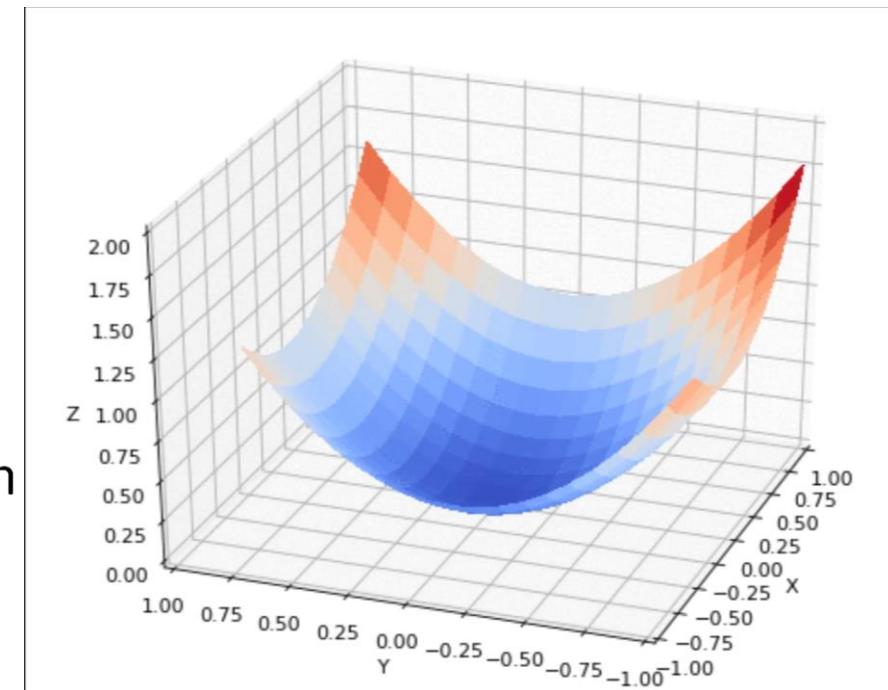
Gradient Descent

Search direction is the direction of fastest descent of the function g .

Gradient descent update equation

$$\lambda \Delta = -g_\theta = -\mathbf{J}^\top \mathbf{f}$$

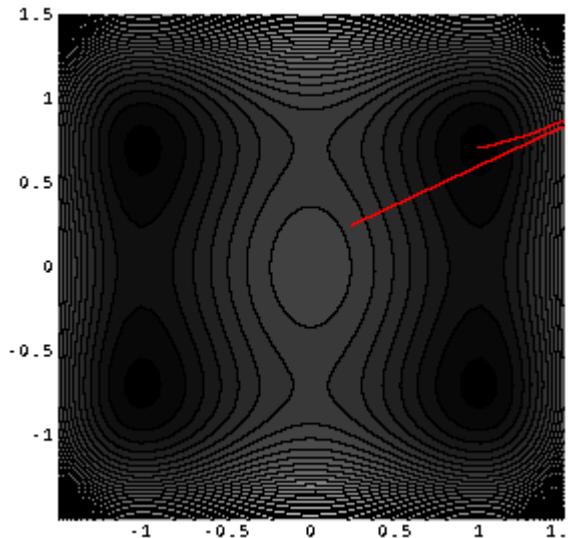
Requires a 1D line search in λ to find the optimum direction.



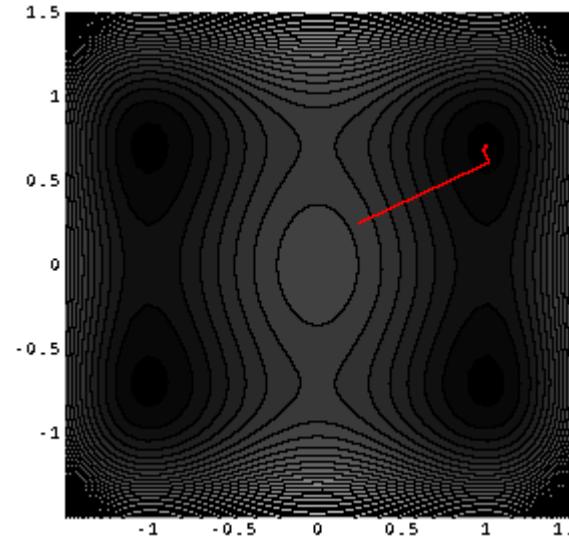
- Mixture of Gauss-Newton and Gradient descent.
- Acts like Gauss-Newton when close to the minimum (quadratic region)
- Gradient descent when improvement is difficult.
- Depends on a parameter λ which
 1. Controls the mixture of Gauss-Newton and Gradient Descent
 2. Controls the step-length.

What about non-Linear Least Squares?

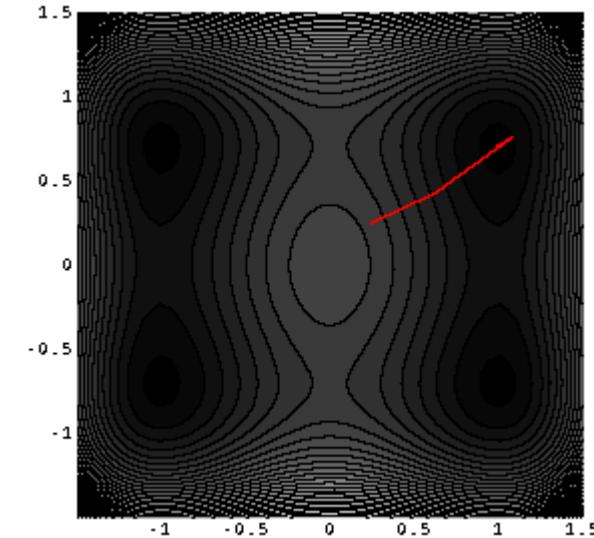
- Lets See some examples 1:



Gauss-Newton



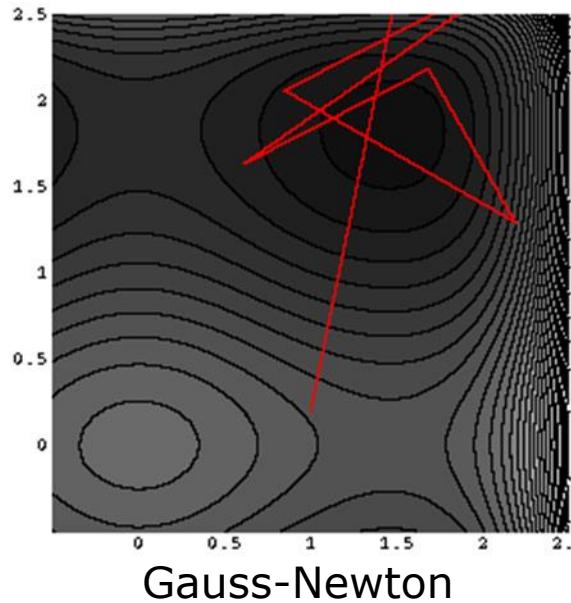
Gradient descent



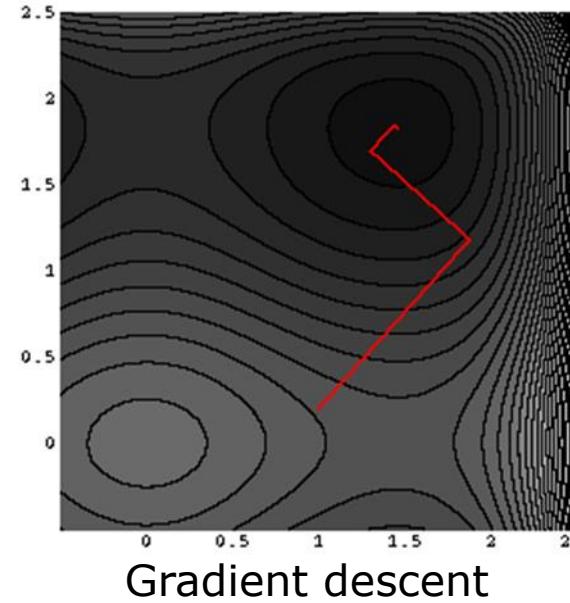
Levenberg

What about non-Linear Least Squares?

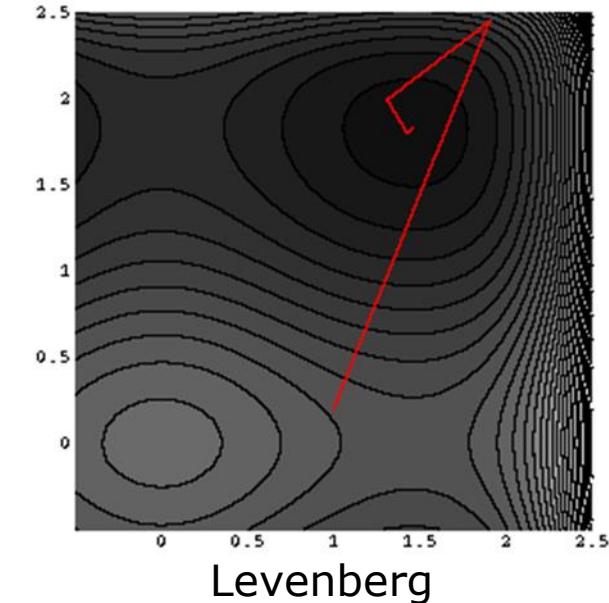
- Lets See some examples 2:



Gauss-Newton



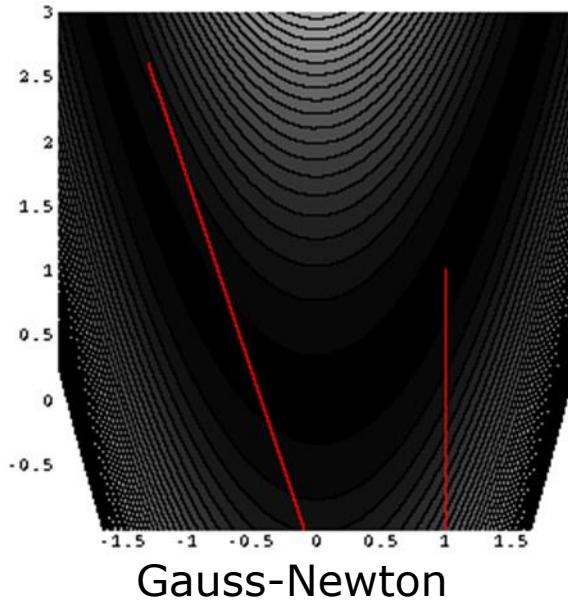
Gradient descent



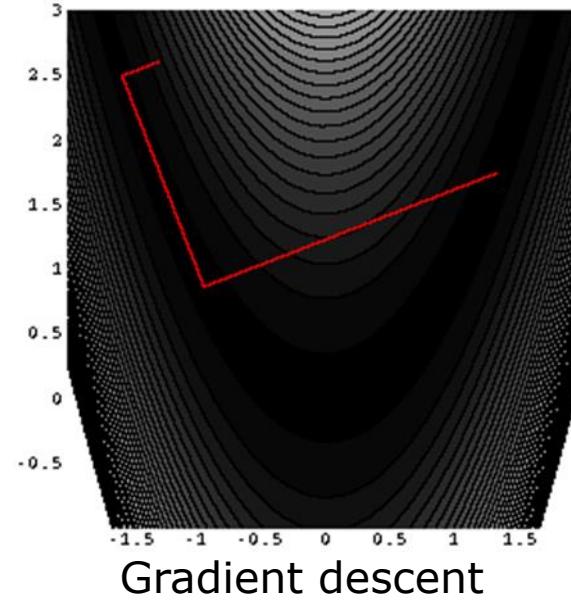
Levenberg

What about non-Linear Least Squares?

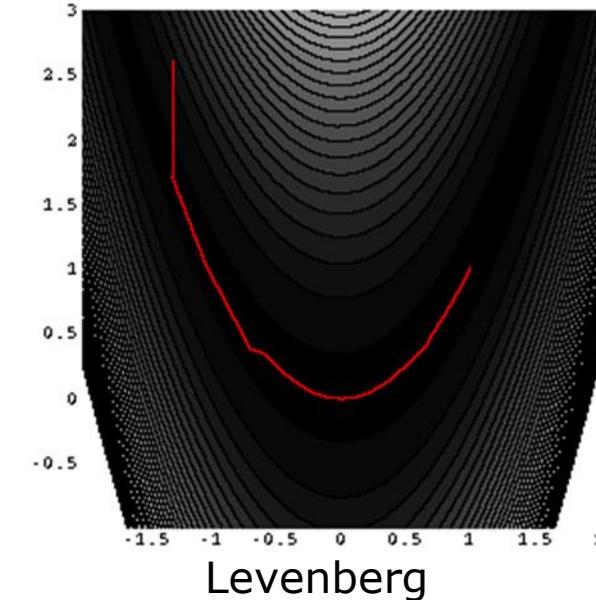
- Lets See some examples 3:



Gauss-Newton



Gradient descent



Levenberg

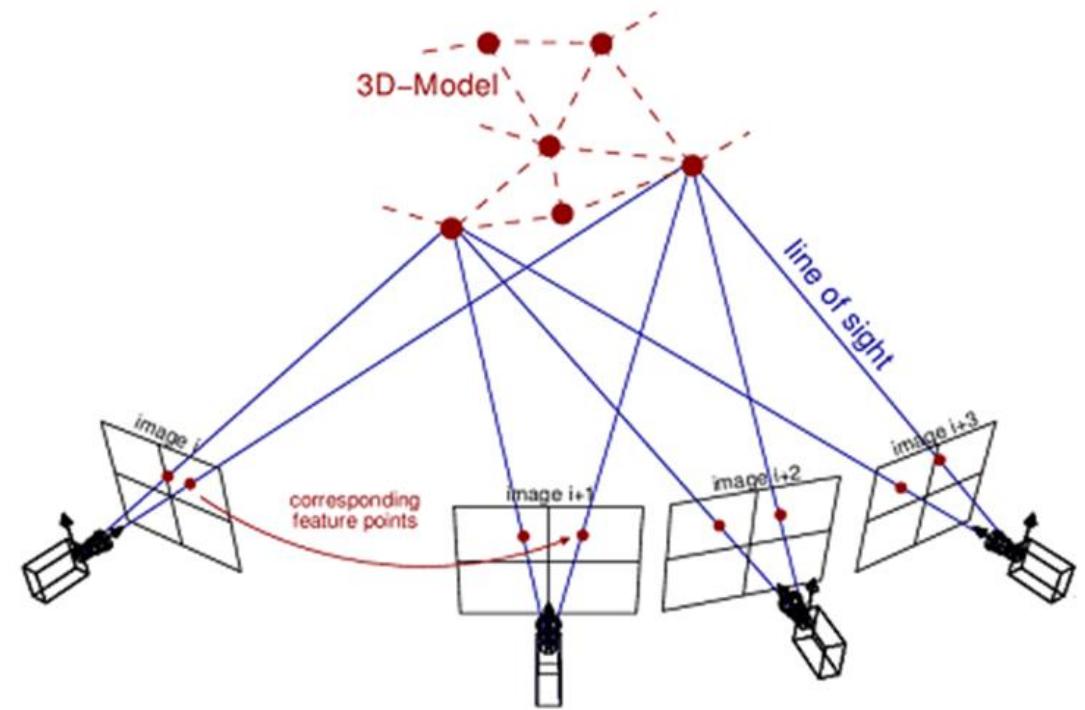
- It is obvious that Levenberg Marquadt displays robustness

Bundle Adjustment

- Bundle Adjustment is the employment of nonlinear optimization in the problem of the minimization of the re-projection error, by finding the optimal Poses (extrinsics) of the cameras and the locations of the 3D points.

$$\arg \min_{\mathbf{w}, \boldsymbol{\theta}} \sum_{f=1}^F \sum_{n=1}^N \|\mathbf{x}_n^f - \pi(\mathbf{w}_n; \boldsymbol{\theta}^f)\|_2^2$$

\mathbf{x} 2D projection \mathbf{w} 3D point
 $\boldsymbol{\theta}$ extrinsics N no. of points
 π projection function
 F no. of frames



Bundle Adjustment – Linearization

$$\pi(\mathbf{w}_n + \Delta\mathbf{w}_n; \boldsymbol{\theta}_f \circ \Delta\boldsymbol{\theta}_f) \approx \pi(\mathbf{w}_n; \boldsymbol{\theta}_f) + \mathbf{J}_n^f \begin{bmatrix} \Delta\boldsymbol{\theta}_f \\ \Delta\mathbf{w}_n \end{bmatrix}$$



$$\arg \min_{\Delta\boldsymbol{\theta}, \Delta\mathbf{w}} \sum_{f=1}^F \sum_{n=1}^N \rho_n^f \| \mathbf{x}_n^f - \pi(\mathbf{w}_n; \boldsymbol{\theta}_f) - \mathbf{J}_n^f \begin{bmatrix} \Delta\boldsymbol{\theta}_f \\ \Delta\mathbf{w}_n \end{bmatrix} \|_2^2$$

\mathbf{x} 2D projection $\mathbf{w} \leftarrow$ 3D point

$\boldsymbol{\theta}$ extrinsics N no. of points

π projection function

F no. of frames $\rho \rightarrow$ visibility $\in [0, 1]$

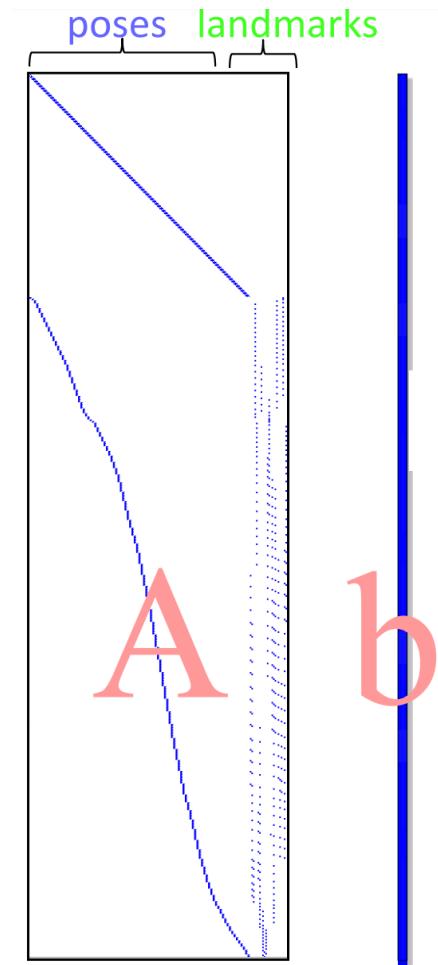
Bundle Adjustment – Linearization

- The Linearization of the minimization happens by calculating the Jacobian of the projection matrix
 - Assuming the following projection matrix:
- $$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} f_x & s_k & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} [R \quad T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$
- We first define the orientation as the rotation matrix associated with the axis angle w_x, w_y, w_z using the Rodrigues equation
 - The Jacobian **FUNCTION** can be calculated as:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial u}{\partial w_x} & \frac{\partial u}{\partial w_y} & \frac{\partial u}{\partial w_z} & \frac{\partial u}{\partial f} & \frac{\partial u}{\partial u_0} & \frac{\partial u}{\partial v_0} & \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial w_x} & \frac{\partial v}{\partial w_y} & \frac{\partial v}{\partial w_z} & \frac{\partial v}{\partial f} & \frac{\partial v}{\partial u_0} & \frac{\partial v}{\partial v_0} & \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix}$$

Bundle Adjustment – Comments

- Bundle adjustment (and graph optimization) is the backbone of all SLAM algorithms
- Keep in mind that:
 - We need to provide the Jacobian of the projection
 - We usually provide a covariance matrix (See the linear case for uncertainty)
 - It is solved using Levenberg-Marquadt
 - There are a lot of computational issues which we can overcome by exploiting the sparsity of the function $AX=b$ (see least squares)
Look at the following A and b Matrices
This solution is called **Sparse Bundle Adjustment!**

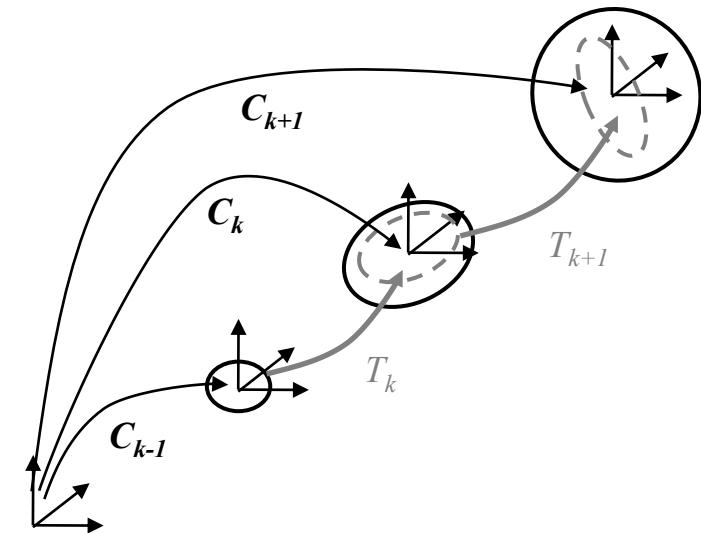


Bundle Adjustment – Covariance

- The uncertainty of the camera pose C_k is a combination of the uncertainty at C_{k-1} (black-solid ellipse) and the uncertainty of the transformation T_k (gray dashed ellipse)

- $C_k = f(C_{k-1}, T_k)$
- The combined covariance Σ_k is

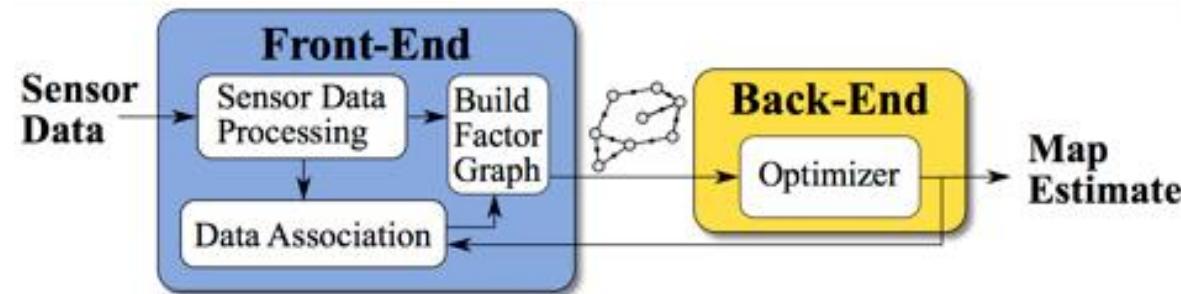
$$\begin{aligned}\Sigma_k &= J \begin{bmatrix} \Sigma_{k-1} & 0 \\ 0 & \Sigma_{k,k-1} \end{bmatrix} J^\top \\ &= J_{\vec{C}_{k-1}} \Sigma_{k-1} {J_{\vec{C}_{k-1}}}^\top + J_{\vec{T}_{k,k-1}} \Sigma_{k,k-1} {J_{\vec{T}_{k,k-1}}}^\top\end{aligned}$$



- The camera-pose uncertainty is always increasing when concatenating transformations. Thus, it is important to keep the uncertainties of the individual transformations small

Recent Visual Slam Solutions - Intro

- That was too much info, let's see now some recent solutions to the Slam Problem:
- Most recent visual SLAM methods are split in two parts:
- The **Frontend**: where the raw data are converted into pose graphs and Loop constraints and the **Backend** where, given a graph with constrains, the new pose of the robot is calculated as well as the surrounding map points.



Recent Visual Slam Solutions – Common Architecture

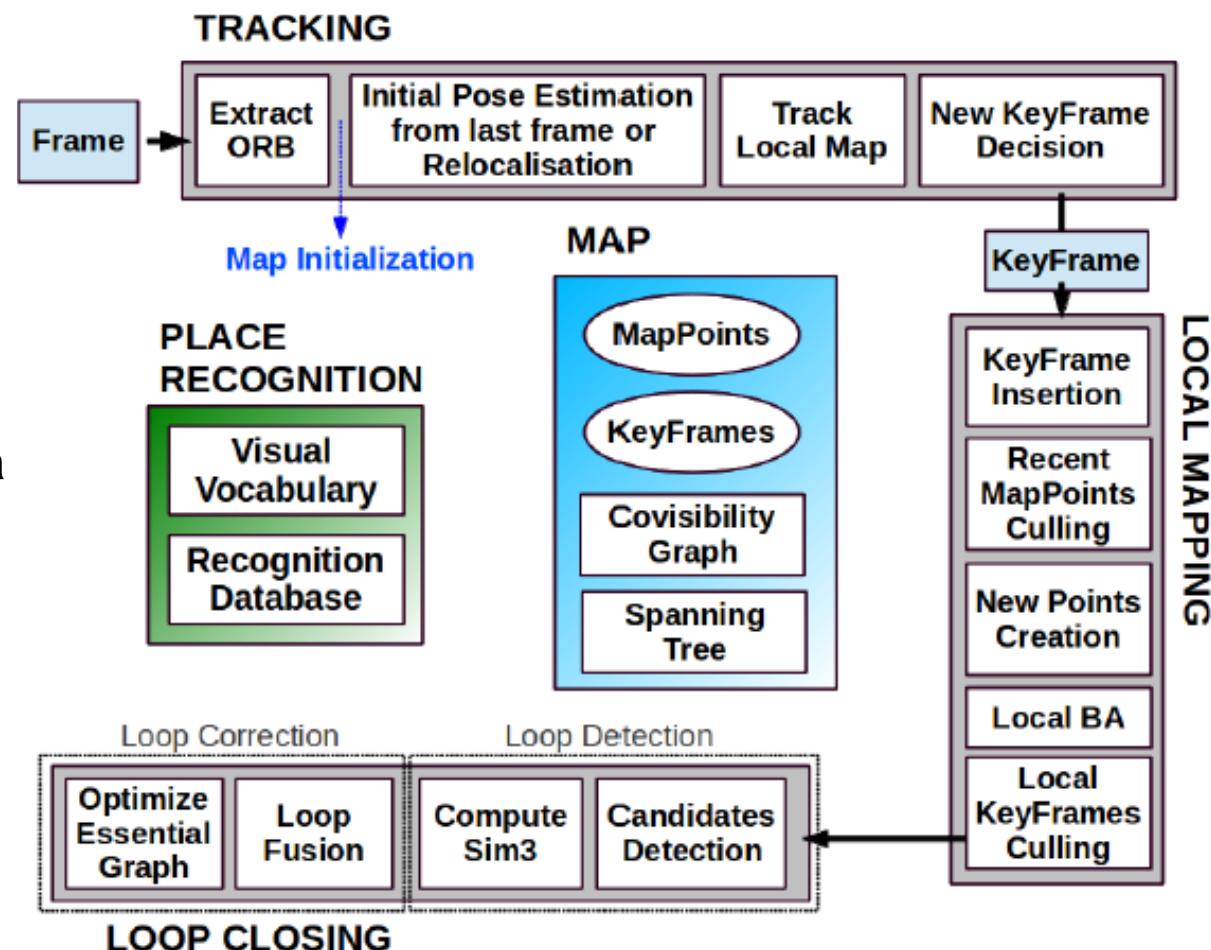
- Front End
 - Data Association
 - Frame to Frame
 - Multi-frame
 - Loop Closure Detection
 - Geometric Initialization
 - Pose Estimation
 - Landmark Triangulation
 - System Formation
 - Observation Matrix
 - Covariance Matrix
 - Graph Generation and Update
- Back End
 - Filter-Based State Estimation
 - Extended Kalman Filter
 - Particle Filters
 - Least squares optimization
 - Bundle Adjustment
 - Graph Optimization
 - Key Frame

Recent Visual Slam Solutions – recent advances

Towards Realtime operation?:

- The computational cost of bundle adjustment has lead to the idea of **keyframing**:
i.e.: identifying and describing some of the frames to be used for graph optimization.
- Bags of words for robust loop closure.
 - What can you tell me about that?
- Co-visibility Graph

- The ORBSLAM algorithm is one of the most well performing opensource implementations of visual slam.
- Three parallel threads:
 - tracking,
 - localizing the camera with every frame and deciding when to insert a new keyframe
 - local mapping
 - Processes new keyframes and performs local BA to achieve an optimal
 - reconstruction in the surroundings of the camera
 - loop closing
 - The loop closing searches for loops with every new keyframe
 - Essential Graph



ORB SLAM on Kitti

ORB-SLAM

Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós

{raulmur, josemari, tardos} @unizar.es



Instituto Universitario de Investigación
en Ingeniería de Aragón
Universidad Zaragoza



Universidad
Zaragoza

Sum up

- Sum up Localization from last time
- Some terminology
- Pose-Landmark Graph Slam
- Example of Linear 1D SLAM
- Non-Linear Optimization approaches
- Bundle Adjustment
- Visual Slam System architecture
- ORBSLAM

Perception for Autonomous Systems 31392:

Visual SLAM

Simultaneous Localization & Mapping

Lecturer: Evangelos Boukas—PhD

Final Exam Prep

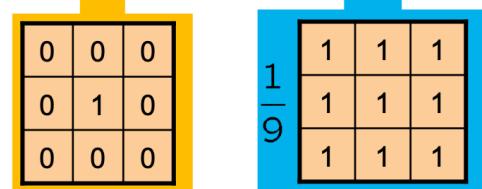
- ✓ Before Friday: modules 1-3
- ✓ Friday: modules 4-6
- ✓ Saturday: modules 7-10
- ✓ Sunday: quizzes

Week 1:

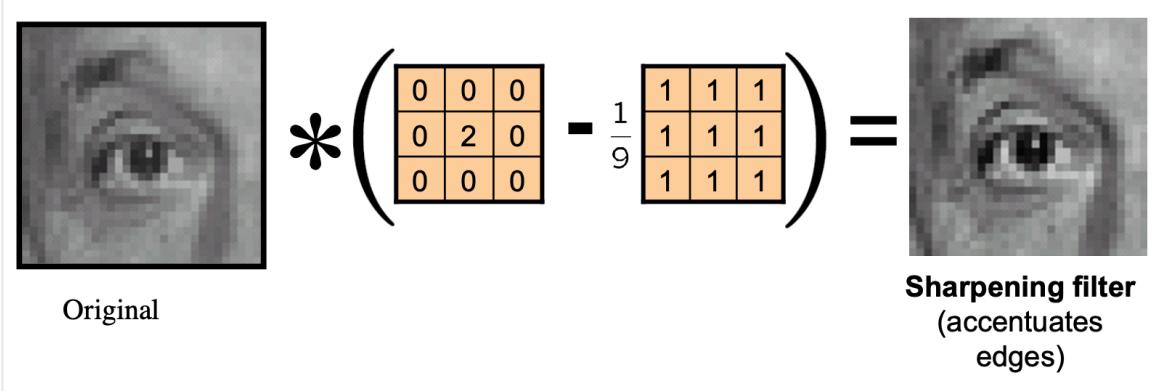
- Colors: RGB, HSI/HSV/HSB (hue-saturation-intensity/value/brightness), etc
- **Image Filtering cv2 functions**
- Linear Filtering: used for noise deduction by averaging neighboring pixels since nearby pixels are likely to belong to same object and therefore have similar color (mean filtering)
- Another option is kernel filter with **cross correlation, convolution**
 - Kernel examples: all 1's with a factor scaler, like 3x3 1's * 1/9 gives blur; 3x3 kernel with zeros and 1 in middle gives identical images.

Sharpening

$$\begin{aligned}f_{sharp} &= f + \alpha(f - f_{blur}) \\&= (1 + \alpha)f - \alpha f_{blur} \\&= (1 + \alpha)(w * f) - \alpha(v * f)\end{aligned}$$



$$= ((1 + \alpha)w - \alpha v) * f$$



- Non-Linear filters: **thresholding**, rectification, median of neighbors
- **Morphology**: most common binary (or greyscale) image operations
 - Structuring element: **cv2.getStructuringElement()**
 - **Erosion**, **dilation**, opening, closing (all rely on convolution with a structuring element like a disk, rectangle, or any other shape)
 - Opening is just another name of **erosion followed by dilation** ex `cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`.
 - Closing is reverse of Opening, **Dilation followed by Erosion** ex `cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`.
 - Morphological Gradient: the difference between dilation and erosion of an image (result looks like the outline of the object) ex `cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)`.
- Connected Components Analysis
 - Connectivity if two neighboring pixels share the same or similar intensity/color value
 - Works for binary, grayscale, color

Extra: **contours**

Qs:

*Exercises_Wk_1, exercise 2: how to remove red or green ball? (Look at weekly project ex)

Week 2:

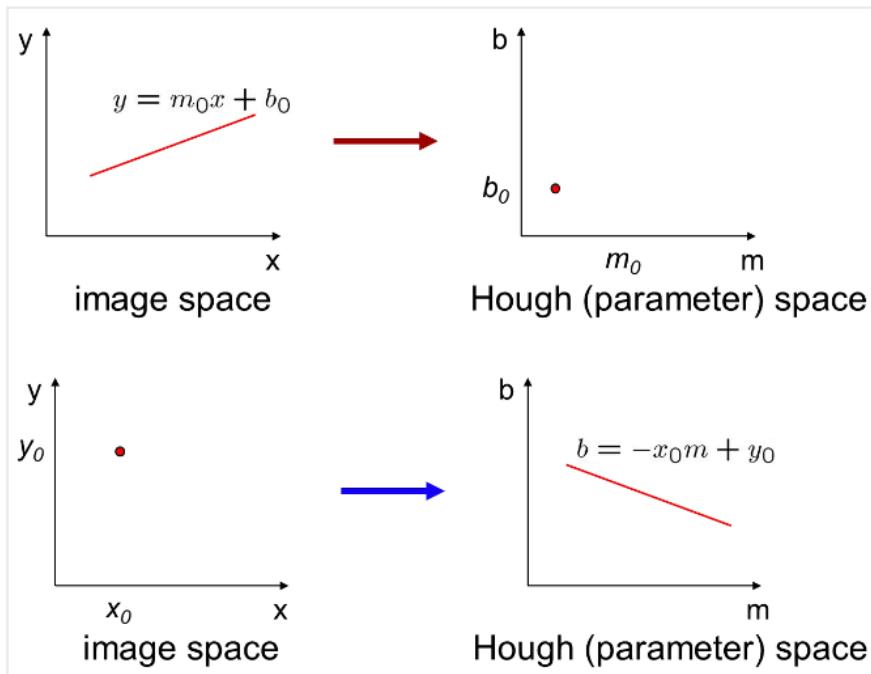
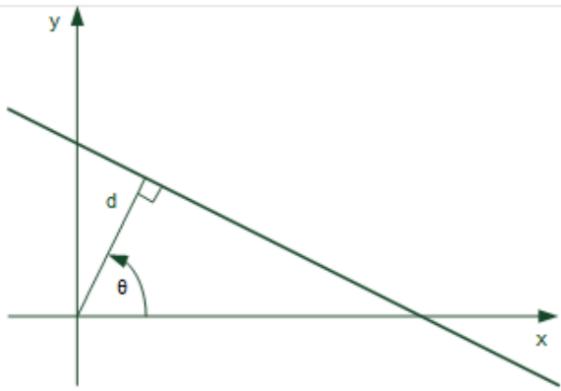
- Image Features
- Feature Detection: find points/areas in an image likely to be detected in other images too
 - Ex. Harris Corner Detector (can distinguish bet. flat, edge, or corner) - rotational invariance but corners are not very unique (desk and building have corners, but desk is not similar to a building)
 - Ex. Shi-Tomasi feature detection (**GoodFeaturesToTrack**)
 - Ex. **Difference of Gaussians** (DoG)
- Feature Description: create a unique descriptor fingerprint for each feature point
- Feature Matching: find correspondences among diff images. Allows us

- to match multiple images and “piece together” a larger image.
- Scale-Invariant Feature Transform (SIFT) contains all: detection, description, matching
 - Robust in: change of translation, change in scale, change in rotation, change in 3D view point, change in illumination
 - Uses DoG on scale space for blob detection (only the max or min in a neighborhood are considered, all octaves (variances/stdevs) are investigated)
 - description: Creates a histogram of gradient orientations (every 10 deg -> 36 bins)
 - matching: all-all Euclidean distance can give us 1000 features for a 640x640 img (with 128 values each feature!) Instead, use Kd-Trees.
- SIFT, SURF (Speeded-Up Robust Features), BRISK, FREAK, MSER, ORB, Patches
- Applications: motion estimation, localization, mapping, photogrammetry, image retrieval, object detection & recognition, autonomous driving, etc.
- Edge Detection
 - Gradient is a vector that points in the direction of most rapid change in intensity.
 - Like all vectors, defined by its orientation and magnitude: $\theta = \arctan(df/dy / df/dx)$, $\|\nabla f\| = \sqrt{(df/dx)^2 + (df/dy)^2}$
 - Positive gradients go from black towards white.
 - Derivative Filters:
 - Sobel, Scharr, Prewitt, Roberts
 - Use kernel and convolution to smooth signals.
 - Canny Edge Detection: noise reduction, gradient calculation, non-maximum suppression, double threshold, edge tracking by **hysteresis** (history)
- Fitting (handling outliers)
 - Hough Transform: $ax+by+c=0$ is unbounded, so instead we use polar coordinates, initialize the grid using (θ, d) , populate the grid by passing through the image and adding votes (see vid)
 - Identify maxima and track lines back to img
 - For non-linear features (ex circles), same approach is used but complexity grows exponentially (for circles, 3 params x, y, r are required. Not advised to go beyond 4.)

$$x \cos \theta - y \sin \theta = d$$

d : perpendicular distance from line to origin

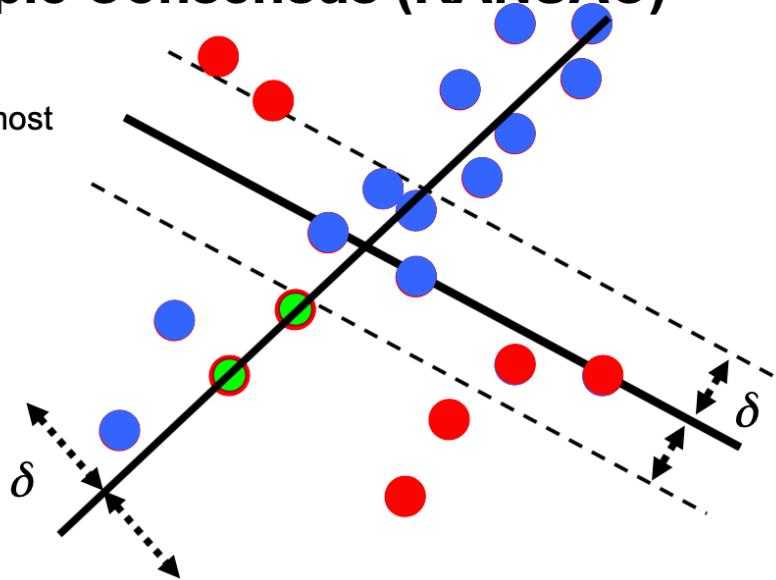
θ : angle the perpendicular makes with the x-axis



- RANdom Sample Consensus (RANSAC) algorithm:
 - 1. Samples (random) # points required to fit model
 - 2. Solve for model params using samples
 - 3. Score by the fraction of inliers within a preset threshold of model
 - Repeat 1-3 until best model is found (high confidence)
 - (Like SVM's, but used for fitting, not classification like SVMs)

RANdom SAmple Consensus (RANSAC)

- Select the models with most inliers to create lines



Week 3:

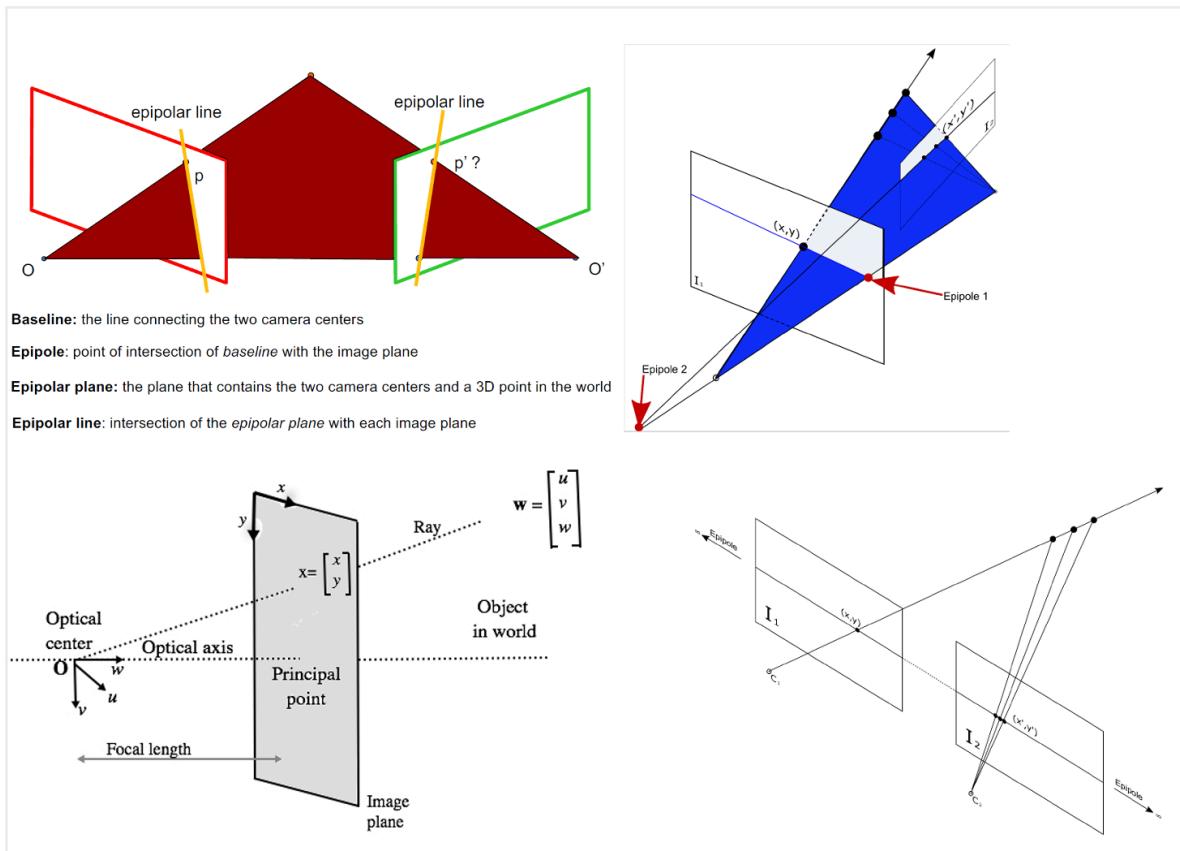
Stereo matching - the process of taking 2 or more images and estimating a 3d model of the scene by finding matching pixels in the images and converting their 2d positions into 3d depths.

Epipolar Geometry

- **Baseline:** the line connecting the 2 camera centers
- **Epipole:** point of intersection of *baseline* with the image plane
- **Epipolar Plane:** the plane that contains the two camera centers and a 3D point in the world
- **Epipolar Line:** intersection of the *epipolar plane* with each image plane

Searching for feature matches along epipolar lines becomes therefore a 1D problem!

- All epipolar lines intersect at the epipoles.

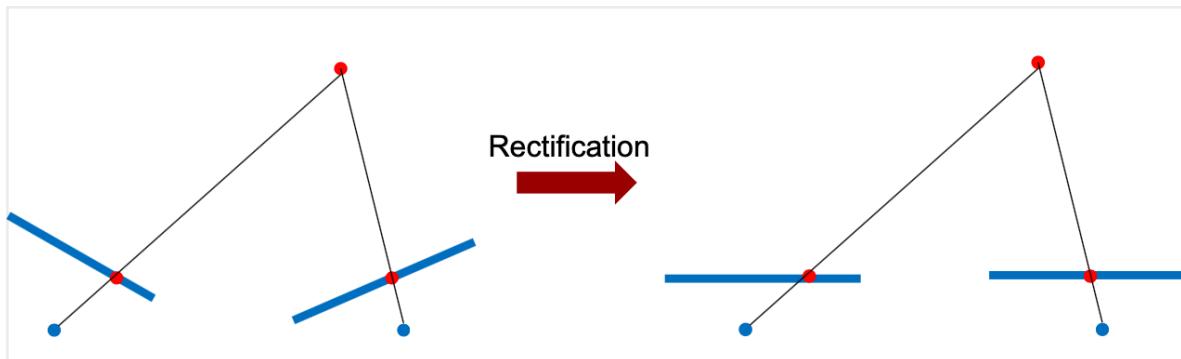


Consider a stereo vision system.
Choose all the statements below that are true.

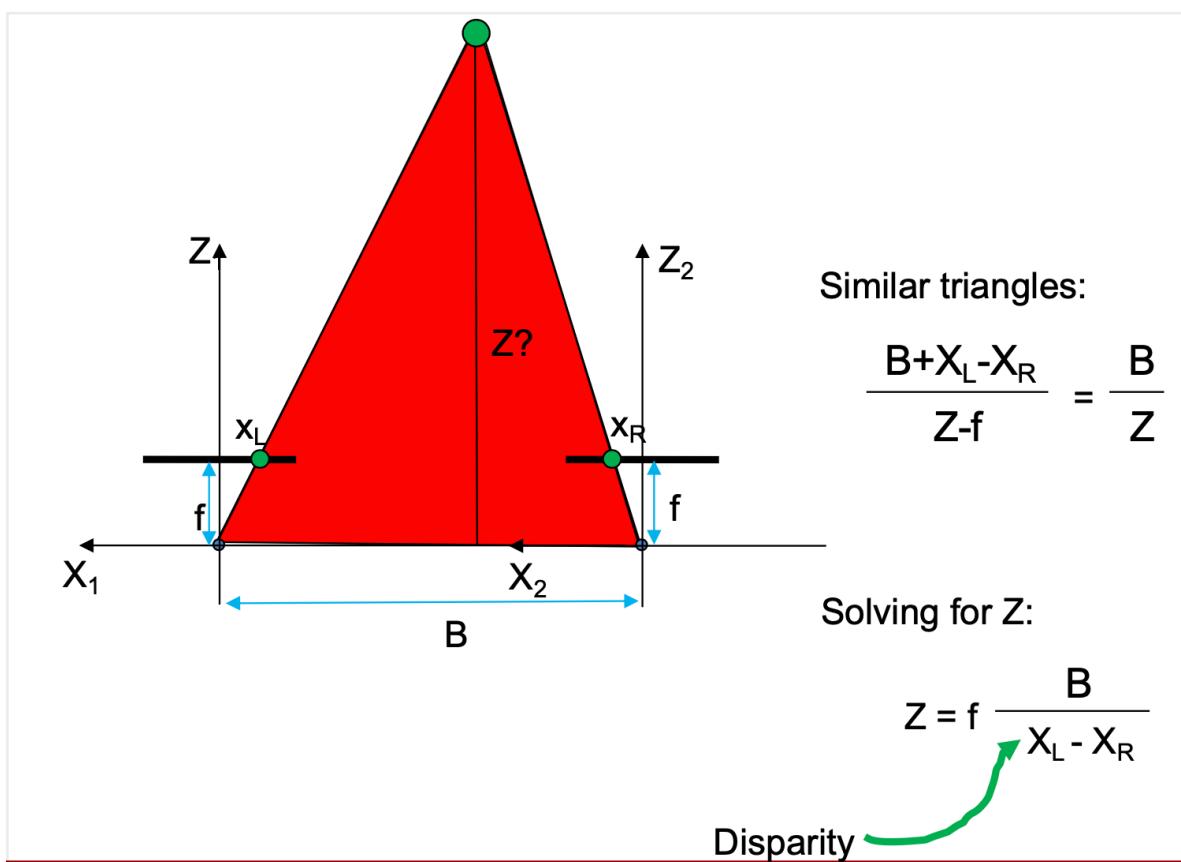
- All epipolar lines are parallel to the optical axis
 - In certain cases, there can exist just one epipole
 - ⇒ The epipoles can be outside the images
 - The baseline intersects the epipolar plane at the epipoles
 - All epipolar lines meet at the optical center
 - ⇒ The epipoles lie on the baseline-containing line
 - ⇒ All epipolar lines intersect at the epipoles
1. False. The optical axis is the line going from O to the point.
 2. False. There will always be two epipoles for a stereocamera.
 3. True. If the cameras are rectified.
 4. False. The epipoles are where the baseline intersects the image plane.
 5. False. The optical center has nothing to do with epipolar lines.
 6. True. The epipoles are where the baseline intersects the image plane.
 7. True. They all cross the epipoles.

Rectification

- Initial images reprojected on common plane parallel to baseline of initial images



Disparity to find depth: large disparity means objects are close to the camera; small disparity means objects are far away (inverse relation)



Correspondence Problem

- There are also "soft" constraints beyond epipolar geometry that help identify corresponding points:
 - Similarity, uniqueness, ordering, disparity gradient is limited
- Spare stereo matching:
 - Extract features (using e.g. SIFT, SURF, Harris, etc)
- Dense stereo matching:
 - Local methods (area-based)
 - Disparity of each pixel determined only by the information of the pixel itself and its neighborhood (hence "local")

- For each epipolar line, compare each pixel in left img with every pixel on same epipolar line in right img and choose pixel that maximizes a similarity metric (or minimizes a dissimilarity metric, like minimizing a difference in e.g. SAD or SSD)
- Improvement: match windows instead of pixels
- Metric ex's: SAD, SSD, NCC (see pic below)
- Global methods (energy-function minimization problem)
 - Find better solutions in exchange for more computations
 - Optimize jointly the disparity values of all pixels of each scanline (e.g. Dynamic Programming) and of the image (e.g. graph cuts)
- Other methods

Stereo Correspondence Metrics

- Sum of Absolute Differences (SAD)

$$SAD(x, y, d) = \sum_{x,y \in W} |I_l(x, y) - I_r(x, y - d)|$$

- Sum of Squared Differences (SSD)

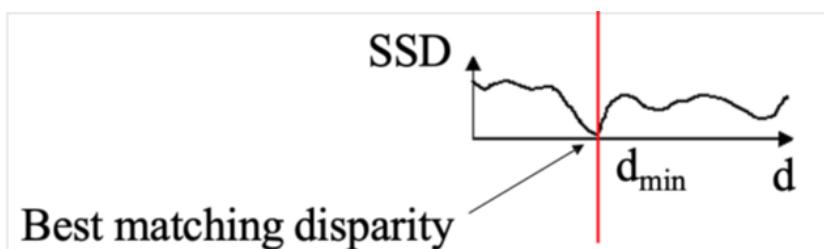
$$SSD(x, y, d) = \sum_{x,y \in W} (I_l(x, y) - I_r(x, y - d))^2$$



- Normalized Cross-Correlation

$$NCC(x, y, d) = \frac{\sum_{x,y \in W} I_l(x, y) \cdot I_r(x, y - d)}{\sqrt{\sum_{x,y \in W} I_l^2(x, y) \cdot \sum_{x,y \in W} I_r^2(x, y - d)}}$$

- ...many many more!!!



Q: exercise 2 last part - can't get to work or export

Q: project - which file correct? and did we ever get last part working?

Takes too long..

Week 4:

In quaternion, w (the 4th dim) is a scalar that stores the rotation around the vector.

Projecting 3D points on the sensor:

- Camera **Extrinsics** - orientation of camera wrt world coordinates
- Camera **Intrinsics** - rest of parameters

Calculating relative orientation between 2 cameras ([link](#)):

Calibrated camera

- angle preserving mapping
- Free parameters: orientation and position (3 for translation, 3 for rotation for each camera, so 12 total params)
- We can estimate 5 of the params:
 - Rotation R of 2nd camera wrt first (3 params), direction B of line connecting 2 centers (2 params)
- We cannot estimate scale or translation or rotation of first camera
- Gives us photogrammetric model

Uncalibrated camera

- straight-line preserving mapping
- Free parameters 2 cameras (each with 5 additional params) so 22 params needed in order to describe mapping
- We can estimate 7 parameters given two images

Relative Orientation Summary

Cameras	#params /img	#params /img pair	#params for RO	#params for AO	min #P
calibrated	6	12	5	7	3
not calibrated	11	22	7	15	5

RO = relative orientation

AO = absolute orientation

min #P = min. number of control points

The cameras project 3d points into our image: $x' = P'x$ for camera #1, $x'' = P''x$ for camera #2

Calibration steps:

- 1. Get min 6 3d-2d correspondences

- 2. Form matrix A ($AP=0$)
 - 3. Calculate SVD
 - 4. Get last column of V which corresponds to the values of P (the projection matrix)
 - In the real world, we use predefined setup that allows us to know relative position of 3d points (the chessboard setup)

So now we have a description for the 3D point in camera frame

- We get to the following

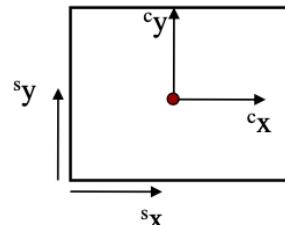
$$[{}^s\mathbf{x}, {}^s\mathbf{y}, \mathbf{w}]^T =$$

Camera Matrix $\begin{bmatrix} -f s_x & 0 & x'_c \\ 0 & -f s_y & y'_c \\ 0 & 0 & 1 \end{bmatrix}$	extrinsics $[\mathbf{R}_{3x3} \quad \mathbf{t}_{3x1}]$	Homogeneous Point in World Coordinates $[x_p \quad y_p \quad z_p \quad 1]^T$
---	--	--

- Where W is the scale
 - **This is the analytic version of the projection matrix**

- We can define the Projection Matrix as follows

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



- **This is great news for Calibration!**
 - It means that our problem is a system of linear equations
 - How many unknowns?
 - 11: (5 + 6) Intrinsic+Extrinsic

- So for every 3D to 2D correspondence we get 2 equations:

$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})x = P_{11}X + P_{12}Y + P_{13}Z + P_{14}$$

$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})y = P_{21}X + P_{22}Y + P_{23}Z + P_{24}$$

- How many points do we need?
 - 6 Points x 2 equations

- Calculate the Singular Value decomposition (SVD)

$$\text{SVD} : \mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

- Get the last column of \mathbf{V}

$\mathbf{P} = \mathbf{V}_{\text{smallest}} \text{ (column of } \mathbf{V} \text{ corr. to smallest singular value)}$

- Reshape into the \mathbf{P} matrix

Calibration using Homography (\mathbf{H}):

- Set $z=0$ so projection matrix used to project points from one plane to another loses a DOF
- 1. using a static camera, move pattern and get images
- 2. Calculate correspondence using pattern
- 3. Solve using normalized DLT (direct linear transformation)
- However, DLT known to be prone to outliers..
- Other options: Nonlinear Least Squares, RANSAC
 - 1. Choose # samples N
 - 2. Choose 4 random potential matches
 - 3. Compute \mathbf{H} using normalized DLT
 - 4. Project points from \mathbf{x} to \mathbf{x}' for each potentially matching pair
 - 5. Count points with projected distance $< t$
 - 6. Repeat steps 2-5 N times
 - Choose \mathbf{H} with most inliers

Lens Distortion

- Lenses introduce distortion to the image.
- Barrel, pincushion, fisheye
- Use quartic (biquadratic) polynomial or higher order
- 1. First do DLT for projection matrix
- 2. Then form non-linear least square prob including both linear projection and non-linear distortion using Levenberg Marquardt algorithm.

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

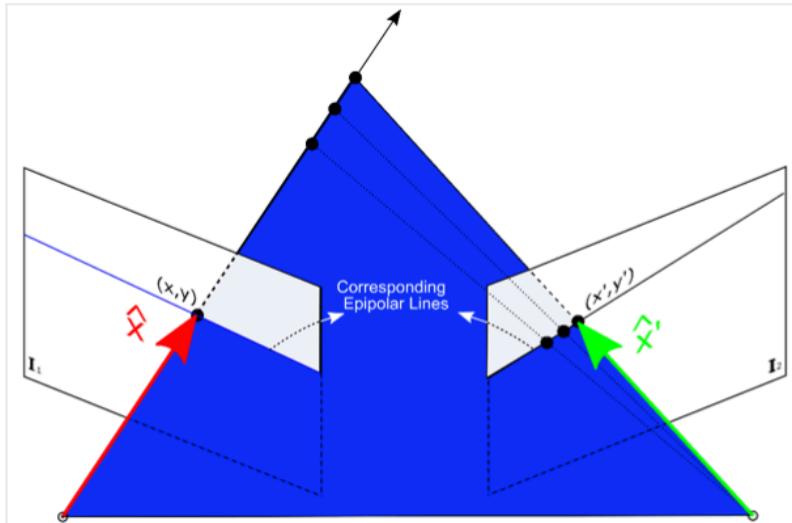


More on epipolar geometry:

- Essential Matrix:
- Assuming 2 calibrated stereo pairs, we can express x, y from the img plane to homogeneous coordinates using the inverse of the camera matrix
- The essential matrix includes the pose of the cameras WRT each other.

$$\hat{x} = K^{-1}x = X$$

$$\hat{x}' = K'^{-1}x' = X'$$



The essential matrix $E = [t] \times R$ is a 3×3 matrix, for which:

- $E x'$ is the epipolar line associated with x' ($l = E x'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $E e' = 0$ and $E^T e = 0$
- E is singular (rank two)
- E has five degrees of freedom

- Fundamental Matrix:

- We know how to get from a homogeneous point in one camera to another but how do we get directly from one image to another?
- See slides for how to compute fundamental matrix.
- Homography (no translation) vs fundamental matrix (translation)
- The fundamental matrix projects a point in the right image frame to a point in the left image.

$$\hat{x}^T E \hat{x}' = 0$$

$$\hat{x} = K^{-1}x \quad \rightarrow \quad x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1}$$

$$\hat{x}' = K'^{-1}x'$$

Which is the fundamental matrix



The fundamental matrix $F = K^{-T} E K'^{-1}$ is a 3×3 matrix, for which:

- $F x'$ is the epipolar line associated with x'
- $F^T x$ is the epipolar line associated with x
- $F e' = 0$ and $F^T e = 0$
- F is singular (rank two): $\det(F)=0$
- F has seven degrees of freedom:

To rectify an image:

1. Match some points/features
2. Calculate fundamental matrix (using coordinates of matched points)
3. Calculate the rectified image

Thus both the **Essential** and **Fundamental matrices** completely describe the geometric relationship between corresponding points of a stereo pair of cameras. The only difference between the two is that the former deals with calibrated cameras, while the latter deals with uncalibrated cameras.

Week 5:

3D Point Clouds

- **Pose:** the transformation (translation + rotation) needed to map one point cloud (model) to another point cloud (model) of the same (fully or partially) object or scene. i.e. "determining a camera's position relative to a known 3D object or scene" (Szelinski)

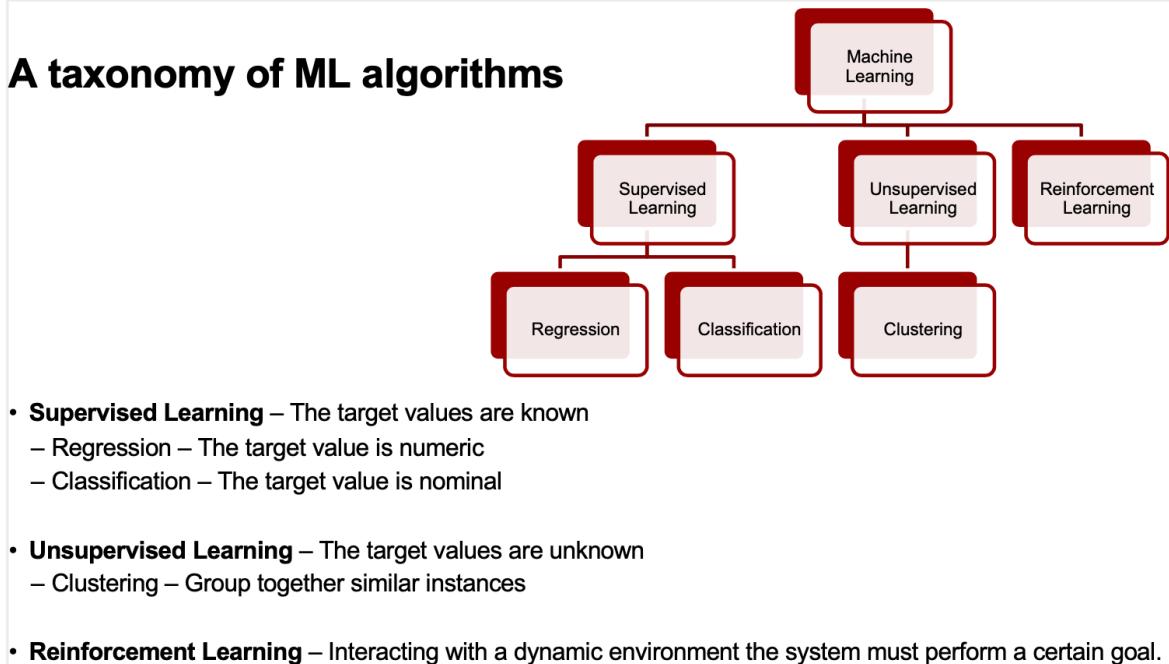
Point Cloud Registration

- Global alignment - the 2 models are roughly aligned
- Local alignment - starting from a rough initial alignment, find the exact precise alignment
- **Local alignment:**
 - Generally 2 steps are needed to register 2 given point clouds of the same object (fully or partially)

- Iterative Closest Point (**ICP**) algo:
 - Consider 2 models (point clouds) of the same object (fully or partially) that are almost aligned. What is the difference of their pose? i.e. What is the transformation that can fully align them?
 - 1. For each object point p in model 1, find the nearest point q in model 2.
 - If model 1 has X points, model 2 has Y points, then calculate XY distances.
 - Use k-d trees (k -dimensional) to speed up search.
 - 2. Use all pairs (p, q) to estimate the transformation from model 1 to model 2.
 - Kabsch algo / Procrustes Analysis: translate centroids of both models to origin, compute centroids (c_p, c_q) , subtract from each point coords the coords of its corresponding centroid ($p' = p - c_p$, same for q), compute com matrix ($C_{pq} = \text{sum}(p'^* \text{trans}(q'))$), compute optimal rotation (calc SVD of C_{pq} and rot matrix $R = U^* \text{trans}(V)$), compute optimal translation $T = c_q - R^* c_p$.
 - 3. Apply the transformation to the points of model 1.
 - 4. Repeat steps 1-3 until convergence or stop criterion is met.
- **Global Alignment:**
 - Cannot use ICP if initial poses are too different.
 - Instead, find 3D feature descriptors and match them.
 - **Spin Images:** "spin" a discretized 2D grid around the surface normal of a point, accumulate neighboring pixels in the grid bin while spinning, the descriptor is the 2D grid (img) where each elem (pixel) contains the # accumulated points.
 - **Point Feature Histograms (PFH):** calculate 3 angle values for all pairs of points within a radius r from considered point (maybe also distance) and bin the set of all triplets/quadruplets in a histogram.
 - **Fast Point Feature Histograms (FPFH):** same as PFH algo but within k-d tree (so reduces computational complexity)
 - Matching 3D features generally results in many false-matches.
 - Sol'n: RANSAC!
 - Robust to outliers.
 - 1. Randomly choose 3 pairs matched points
 - 2. Estimate the relative pose (kabsch / procrustes)
 - 3. Apply the transformation and assess its "validity"
 - 4. Keep transformation with most inliers.
 - 5. Repeat random sampling.

Week 6:

Machine Learning for 3D point clouds



Regression:

- Supervise learning
- Target value is numeric
- Tries to assign correct significance (weights) to input variables and make a weighted linear combination of them to predict output variables.
- Finds an eqn f that models the linear relationship bet input and output:
 $y = f + e$
- Performance evaluated by error function
 - Most used it Mean Squared Error (**MSE**) = $(1/N) \sum(t-y)^2$ where t is true value of learned output and y is predicted value.
- Goal is to minimize the MSE.
- Polynomial regression for when mapping is not linear.

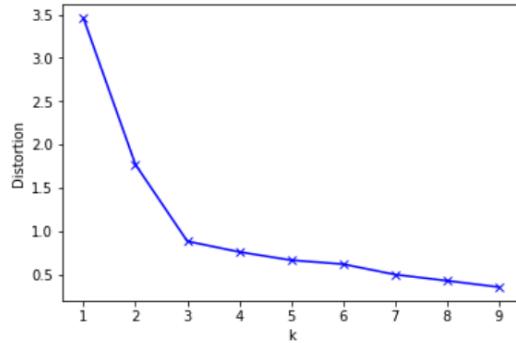
Clustering:

- Unsupervised learning
- Target values are unknown
- Group together similar instances
- Finds underlying structure/partitions of points, ie clusters
- K-means
 - Minimizes sum of squared euclidean distances (distortion) between points and their randomly assigned cluster centers, then updates centers
 - Required to know number of clusters k (use elbow method to determine best k , or silhouette analysis)
 - Iterative process

How to define k ?

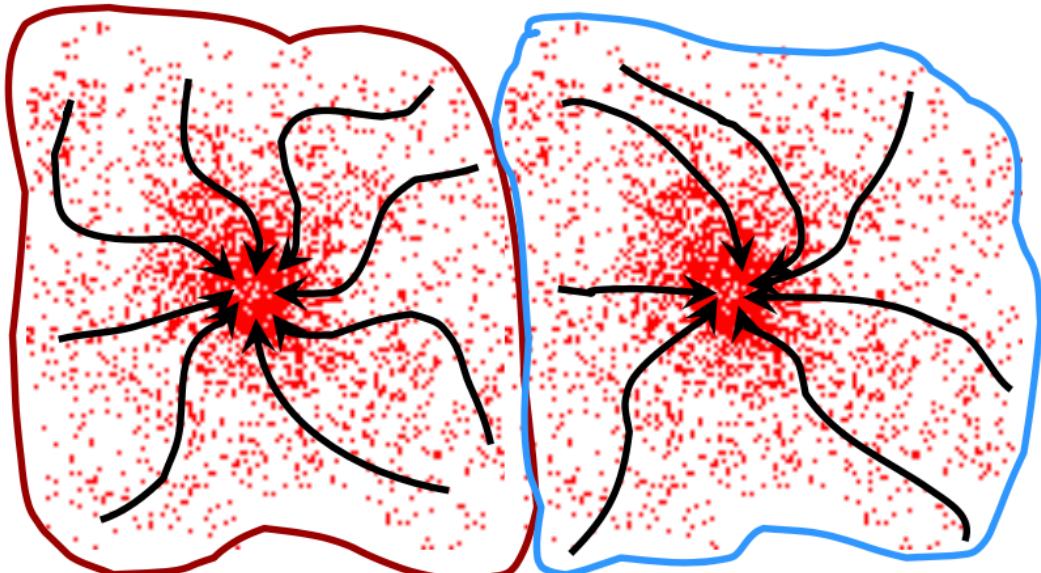
- Elbow method

- Run k-means for several k and calculate distortion for each k
 - » Distortion: sum of squared distances of each point to the center of the closest cluster
- Plot distortion vs k
- Look for k where the curve stops decreasing rapidly



- Mean Shift

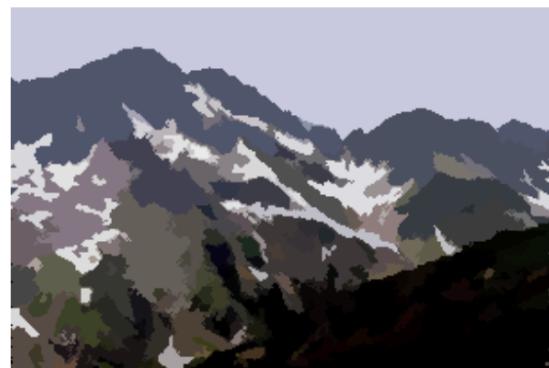
- Finds the maxima (modes) of a density function given discrete data sampled from that function.
- Each cluster defined by the radius ("bandwidth" h) of its region and kernel function K .
- Only unique clusters considered (so we don't need to set number of clusters).
- **Attraction basin:** region for which all trajectories lead to same mode
- **Cluster:** all data points in attraction basin of a mode



The mean shift algorithm seeks *modes* of the given set of points

1. Choose kernel and bandwidth
2. For each point:
 - a) Center a window on that point
 - b) Compute the mean of the data in the search window
 - c) Center the search window at the new mean location
 - d) Repeat (b,c) until convergence
3. Assign points that lead to nearby modes to the same cluster

Mean shift segmentation results

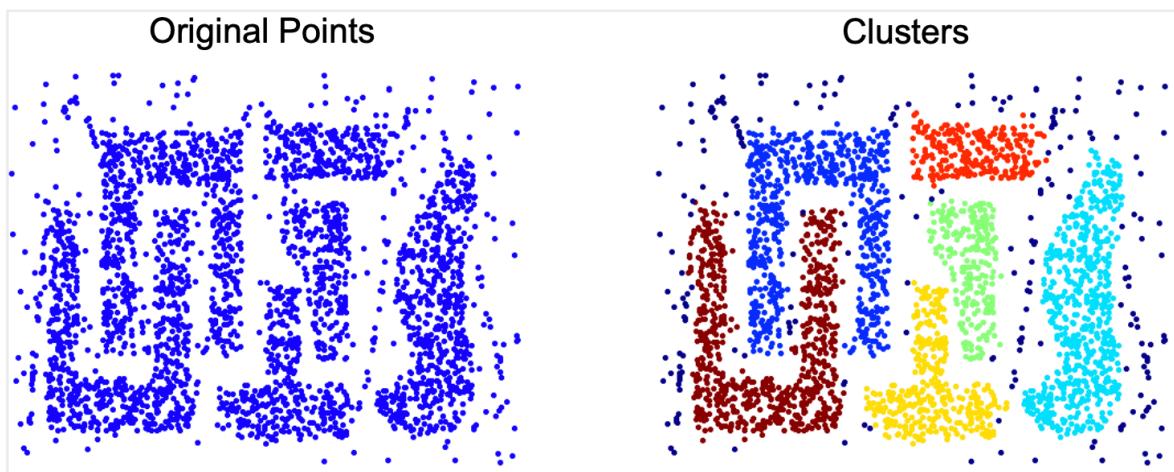


- Density-based spatial clustering of applications with noise (**DBSCAN**)
 - Partition points into dense regions separated by not-so-dense regions.
 - A cluster is defined as a maximal set of density-connected points (of arbitrary shape)
 - Density at point p is the #points within a circle/sphere of radius e (dense if contains at least MinPts points)
 - **Core point** has more than MinPts within e
 - **Border point** has fewer than MinPts within e but is in the neighborhood of a core point
 - **Noise point** otherwise
 - **Density edge** is a line between 2 core points if they are within distance e
 - A point p is **density-connected** to a point q if there's a path of

- edges from p to q.
- A cluster contains all points that can be reached by following a sequence of density-connected core points and border points that are closest to one of the above-clustered core points.

DBSCAN algorithm

1. Label points as core, border and noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster
 - Create a new cluster with the point p and all the points that are density-connected to p.
4. Assign border points to the cluster of the closest core point.



- Hierarchical Clustering
 - Builds a hierarchy of clusters (clusters that consist of other smaller clusters)
 - **Agglomerative** = bottom-up approach (each point starts in its own cluster and pairs of clusters are merged as one moves up the hierarchy)
 - Based only on distance (similarity) between points

Agglomerative / Bottom-up hierarchical clustering

is based only on distance (similarity) between the points.

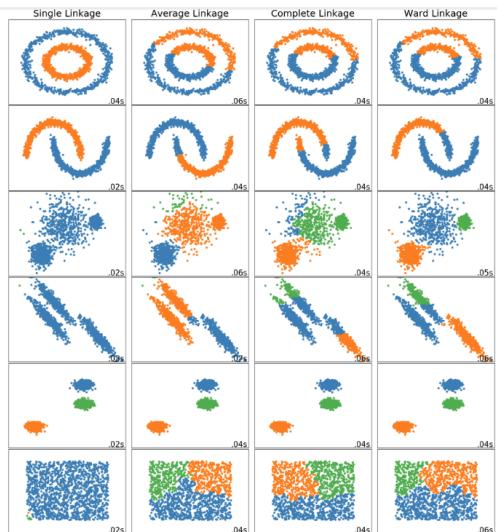
Algorithm steps:

1. Start by assigning each point to its own cluster, obtaining as many clusters as points.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster.

How is distance (similarity) between clusters assessed?

Linkage methods:

- **Single-link:** the distance between two clusters is equal to the minimum distance from any member of one cluster to any member of the other cluster.
- **Complete-link:** the distance between two clusters is equal to the maximum distance from any member of one cluster to any member of the other cluster.
- **Average link:** the distance between two clusters is equal to the average distance from any member of one cluster to any member of the other cluster.
- **Ward-link:** the distance between two clusters is equal to the sum of squared differences within any member of one cluster to any member of the other cluster.



- **Divisive** = top-down approach (all points start in one cluster and split recursively as one moves down the hierarchy)

Week 7:

Histogram Filter

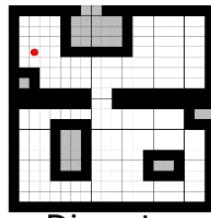
Kalman Filter

State Estimation

- Given a state vector of a system, estimate over time the state using input of external sensors

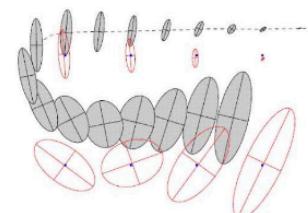
Continuous vs Discrete State Unimodal vs Multimodal Distribution

- State:

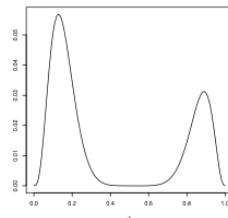


Discrete

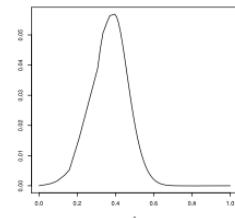
- Distribution



Continuous



Multimodal



Unimodal

- Localization is a sense/move cycle
- Global localization:
 - Belief \rightarrow Probability

- Measurements -> Multiplication followed by normalization
- Moving -> Convolution
- Bayes Rule

$$p(X_i | Z) = \frac{p(Z | X_i) p(X_i)}{p(Z)}$$

Measurement Probability *Prior*

- Total Probability

$$\Pr(A) = \sum_n \Pr(A | B_n) \Pr(B_n)$$

Kalman uses gaussian probabilities and variance is now a covariance matrix.

Prediction (Ingredients):

- X: State Vector (Including our Prior Info)
- P: Uncertainty Covariance (Incl. Prior Info)
- F: State Transition Matrix (we just discussed it)
- u: External Motion (E.g. Deceleration from car)

Recipe

$$\begin{aligned} X' &= F X + u \\ P' &= F \cdot P \cdot F^T \\ y &= Z - H \cdot X \\ S &= H \cdot P \cdot H^T + R \\ K &= P \cdot H^T \cdot S^{-1} \\ X' &= X + K \cdot Y \\ P' &= (I - K \cdot H) \cdot P \end{aligned}$$

Measurement Update (Ingredients):

- Z: Measurement
- H: Measurement Matrix
- R: Measurement Noise
- y: Error
- K: Gain
- I : Identity Matrix

Week 8:

Classification

- Supervised learning
- Output is usually decision boundaries that separate the classes

Dimensionality Reduction

- Principal Component Analysis (**PCA**)
 - Reduces high dimensionality.
 - Creates new features that are linear combinations of original features.
 - New features are orthogonal to each other.
 - Keep new features that account for large amount of variance in original dataset.
 - Re-base the dataset's coordinate system in a new space defined by its lines of greatest variance.

- Ex: detecting faces (facial recognition)

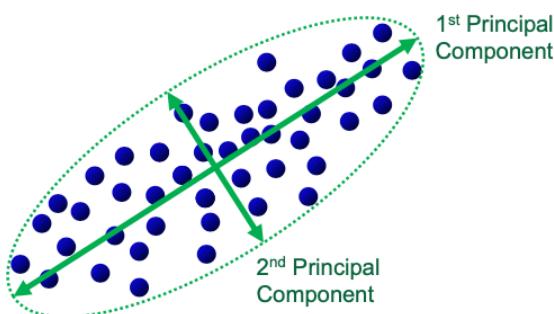
- **Principal Component Analysis (PCA)**

1. Center the input data
2. Calculate Covariance Matrix

$$C = \frac{1}{N-1} X^T X$$
3. Compute eigenvectors & eigenvalues of the Covariance Matrix
 - The first principal component is the eigenvector of the covariance matrix that has the largest eigenvalue
 - » This vector points towards the direction of the largest variance of the data
 - » The corresponding eigenvalue defines the magnitude of this vector
 - The second largest eigenvector is orthogonal to the largest eigenvector, and points into the direction of the second largest spread of the data.
4. Sort the eigenvectors according to their eigenvalues
5. Calculate the variance score (significance).
6. Keep eigenvectors that explain most (e.g. 95%) variance / remove the rest! (\leftarrow dimensionality reduction)
7. Project instances to eigenvectors y .

$$y = W^T x$$

- **Eigenvector:** points to the direction of variance.
- **Eigenvalues:** shows how much of the total variance is explained by the corresponding eigenvector.
- Keep the eigenvectors that explain most (e.g. 95%) of the variance.



Eigenfaces

- Then an image of our dataset becomes:

$$\text{Image} = \text{Mean Image} + c_1 * (\text{1st Eigenface}) + c_2 * (\text{2nd Eigenface}) + \dots$$

- So, these coefficients c_1, c_2, \dots, c_{64} represent the face image!!
- We can perform Face Recognition
 - For a new face image
 - We can calculate its 64 coefficients
 - Find the closest training face (most similar coefficients) in the 64-dimensional space (the most similar face of my training dataset)
 - **Classify** the new face as this most similar training face.

Classifiers

- K-Nearest Neighbors (**KNN**)
 - A geometric method for classification based on the assumption

that neighbor instances belong to the same class.

For classifying a new instance x_* :

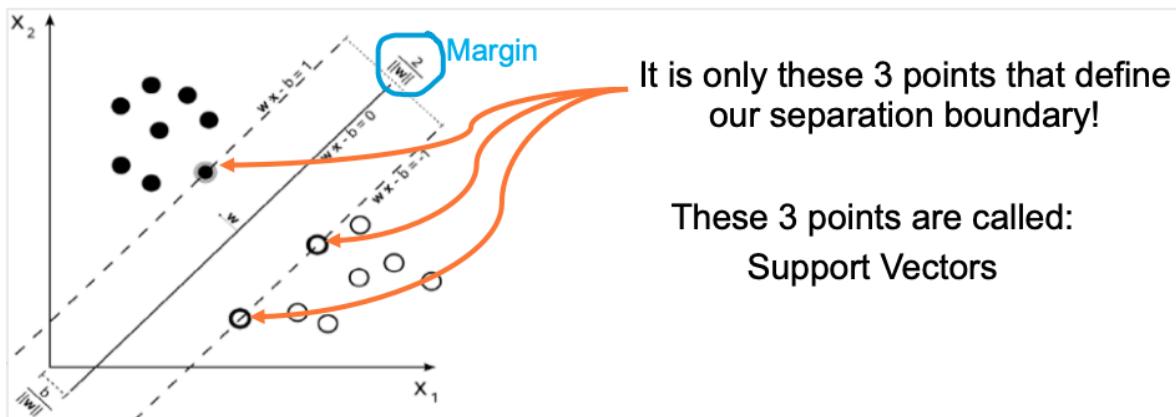
- we calculate its distance from all the other instances.
- Then we select the k-nearest instances.
- The probability of the class is given by:

$$p(C_j|x_*) = \frac{k_j}{K}$$

where k_j is the number of nearest neighbors that belong to class j , and K is the total number of nearest neighbors.

- Support Vector Machines (SVM)

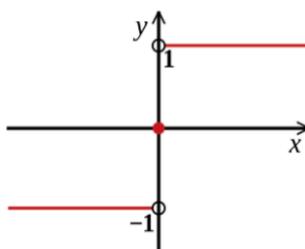
- A geometric method for classification that finds a line/surface in feature space that separates the classes
- The linear decision boundary satisfies 2 criteria:
 - 1. The distance between the instances and the boundary is as large as possible (max margin)
 - 2. The instances are classified as good as possible
- Thus, the decision boundary is a hyperplane $h(x)$ defined as $h(x) = \text{trans}(w) * x + b$ and the goal is to find the w that satisfy 1 and 2.
- Maximize the margin $2/\|w\|$ while correctly classifying all training data points.



The classification rule is expressed as: $f(x_*) = \text{sgn}(h(x_*))$

The sign function is defined as:

$$\text{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

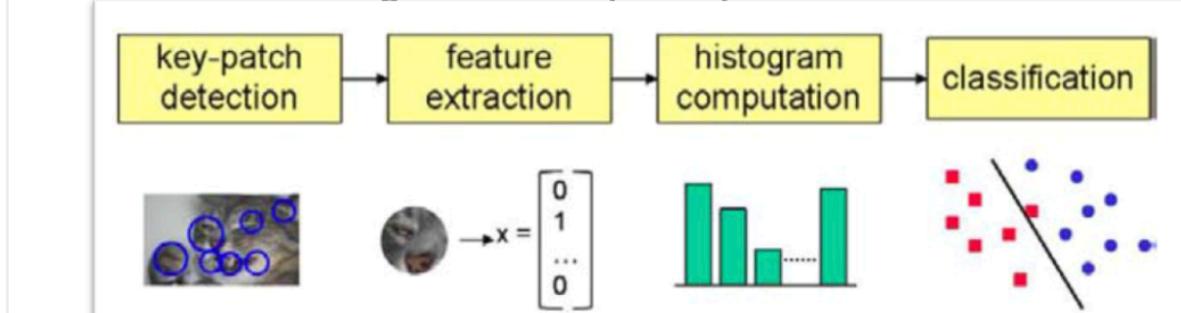


Thus, a new instance is labeled with 1 if it is located above the boundary and with -1 otherwise.

- If data is above 2D or not linearly separable, we can use Multi-class SVMs and kernels (linear, polynomial, gaussian like radial)

- basis function RBF)
 Bag of Words
 – A vector of occurrence counts of a vocabulary of local image features.

Classification of new image in the descriptor's space



Object Detection:

- when we know what we are looking for

Instance Recognition:

- when we have a specific rigid object we are trying to recognize

Category/Class Recognition:

- when we want to recognize instances of extremely varied classes (e.g. animals or furniture)

Latest methods rely on deep neural networks

- e.g. **YOLO** and **Faster R-CNN** for object detection
- e.g. **PointNet** for classification in 3D point clouds

Week 9:

Visual Odometry (VO)

- Has been proposed as an alternative to wheel odometry.
- Concerns the use of cameras to estimate the Pose (position and orientation) of a mobile system by observing the apparent motion of the "static" world.
- Assumes a static world where the only moving object is the mobile system.
- Does not provide a map of the environment.
- It uses previous states of the world to improve its accuracy (SLAM simultaneous localization and mapping).
- Estimates are usually combined with other sensors (GPS, IMU, Laser, Wheel odometry; VINS, VIO visual inertial odometry)
- 3 main variants: 3D-3D, 2D-2D, 3D-2D
 - 2D-2D and 3D-2D are better than 3D-3D
- Accumulates transformations T_{k-1} from frame f_{k-1} to frame f_k over time, providing full trajectory $C_{0:n}$
 - Can optimize over multiple frames using **bundle adjustment**
 - Similar to pose-optimization but also optimizes 3D points
 - Use levenberg-marquadt

- "Given a set of images depicting a number of 3D points from different viewpoints, bundle adjustment can be defined as the problem of simultaneously refining the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera(s)." ([Wikipedia](#))
- Overall: uses consecutive camera frames (frame-to-frame approach) to calculate the relative pose (position and orientation) of the cameras.

Levenberg-Marquadt

- Mixture of Gauss-Newton and Gradient descent.
- Acts like Gauss-Newton when close to the minimum (quadratic region)
- Gradient descent when improvement is difficult.
- Depends on a parameter λ which
 1. Controls the mixture of Gauss-Newton and Gradient Descent
 2. Controls the step-length.

Visual SLAM

- Uses state estimation to exploit additional constraints and re-observations of the same areas (loop-closures) to optimize the localization.

Motion Tracking

- Two main approaches:
 - 1. Feature based
 - Calculate feature on both images, match among features or do block matching around initial point (for small motion)
 - 2. Optic Flow
 - estimate apparent motion
 - Lukas Kanade

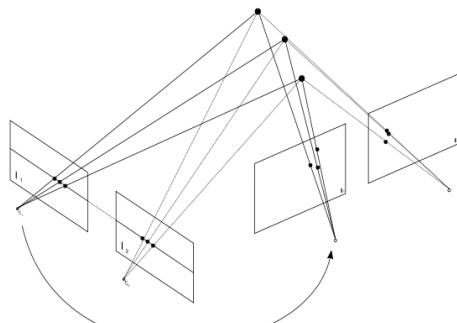
Orientation

- Rotation Matrix
 - Positives (The king of Orientation)
 - Unique, no gimbal lock
 - Negatives:
 - No perturbation, interpolation, unintuitive
- Euler angles
 - Positives
 - Minimal representation, intuitive
 - Negatives
 - Gimbal Lock, non commutative
- Axis Angle:
 - Positives
 - No gimbal lock, minimal representation, nice for perturbation, linear mapping to rotation matrix
 - Negative
 - Not linear “scaling” wrt magnitude
- Quaternions
 - Positives
 - all the axis angle ones, smooth trajectory
 - Negatives
 - No direct geometric representation

☒ Visual Odometry 3D – to 3D

Algorithm 2. VO from 3-D-to-3-D correspondences.

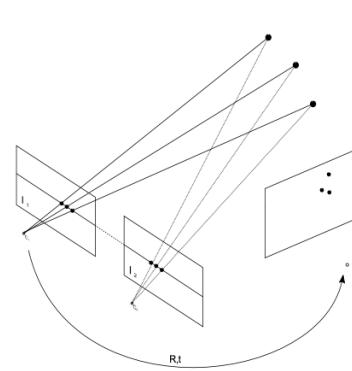
- 1) Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$
- 2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
- 3) Triangulate matched features for each stereo pair
- 4) Compute T_k from 3-D features X_{k-1} and X_k
- 5) Concatenate transformation by computing
 $C_k = C_{k-1} T_k$
- 6) Repeat from 1).



☒ Visual Odometry 3D – to 2D

Algorithm 3. VO from 3-D-to-2-D Correspondences.

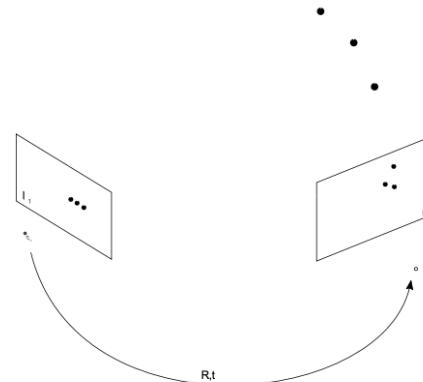
- 1) Do only once:
 - 1.1) Capture two frames I_{k-2}, I_{k-1}
 - 1.2) Extract and match features between them
 - 1.3) Triangulate features from I_{k-2}, I_{k-1}
- 2) Do at each iteration:
 - 2.1) Capture new frame I_k
 - 2.2) Extract features and match with previous frame I_{k-1}
 - 2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
 - 2.4) Triangulate all new feature matches between I_k and I_{k-1}
 - 2.5) Iterate from 2.1).



→ Visual Odometry 2D – to 2D

Algorithm 1. VO from 2-D-to-2-D correspondences.

- 1) Capture new frame I_k
- 2) Extract and match features between I_{k-1} and I_k
- 3) Compute essential matrix for image pair I_{k-1}, I_k
- 4) Decompose essential matrix into R_k and t_k , and form T_k
- 5) Compute relative scale and rescale t_k accordingly
- 6) Concatenate transformation by computing $C_k = C_{k-1} T_k$
- 7) Repeat from 1).



Week 10:

Visual SLAM (simultaneous localization & mapping)

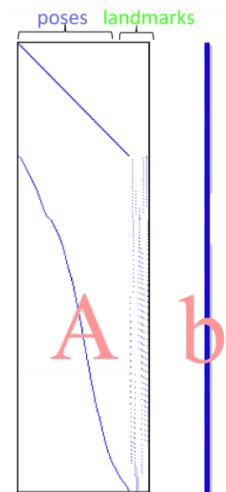
- Pose-landmark graph-slam
- Graph-based slam

Bundle Adjustment

- "Minimizing the reproduction error between the image locations of observed and predicted image points, which is expressed as the sum of squares of a large number of nonlinear, real-valued functions."
- High computational cost

Bundle Adjustment – Comments

- Bundle adjustment (and graph optimization) is the backbone of all SLAM algorithms
- Keep in mind that:
 - We need to provide the Jacobian of the projection
 - We usually provide a covariance matrix (See the linear case for uncertainty)
 - It is solved using Levenberg-Marquadt
 - There are a lot of computational issues which are overcome exploiting the sparsity of the function $AX=b$ (see least squares)
Look at the following A and b Matrices
This solution is called **Sparse Bundle Adjustment!**



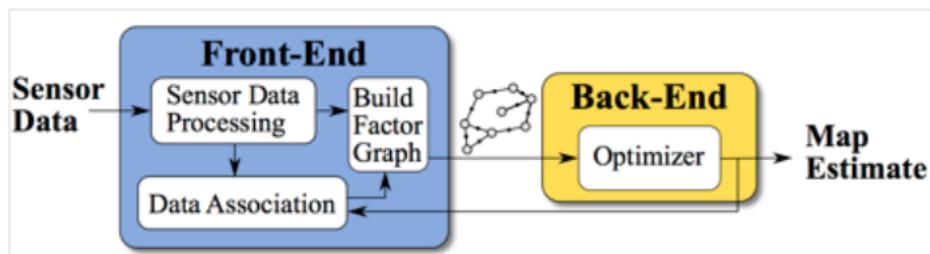
point i by a vector \mathbf{b}_i . Bundle adjustment minimizes the total reprojection error with respect to all 3D point and camera parameters, specifically

$$\min_{\mathbf{a}_j, \mathbf{b}_i} \sum_{i=1}^n \sum_{j=1}^m v_{ij} d(\mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i), \mathbf{x}_{ij})^2,$$

where $\mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i)$ is the predicted [projection](#) of point i on image j and $d(\mathbf{x}, \mathbf{y})$ denotes the Euclidean distance between the image points represented by vectors \mathbf{x} and \mathbf{y} . Because the minimum is computed over

Most recent visual SLAM methods are split into 2 parts:

- **Frontend:** where the raw data are converted into pose graphs and loop constraints.
- **Backend:** where, given a graph with constraints, the new pose of the robot is calculated as well as the surrounding map points.



Recent Visual Slam Solutions – Common Architecture

- Front End
 - Data Association
 - Frame to Frame
 - Multi-frame
 - Loop Closure Detection
 - Geometric Initialization
 - Pose Estimation
 - Landmark Triangulation
 - System Formation
 - Observation Matrix
 - Covariance Matrix
 - Graph Generation and Update
- Back End
 - Filter-Based State Estimation
 - Extended Kalman Filter
 - Particle Filters
 - Least squares optimization
 - Bundle Adjustment
 - Graph Optimization
 - Key Frame

Keyframing:

- The idea of identifying and describing some of the frames to be used for graph optimization (moving towards realtime operation)

Bag-of-words for robust loop closure.

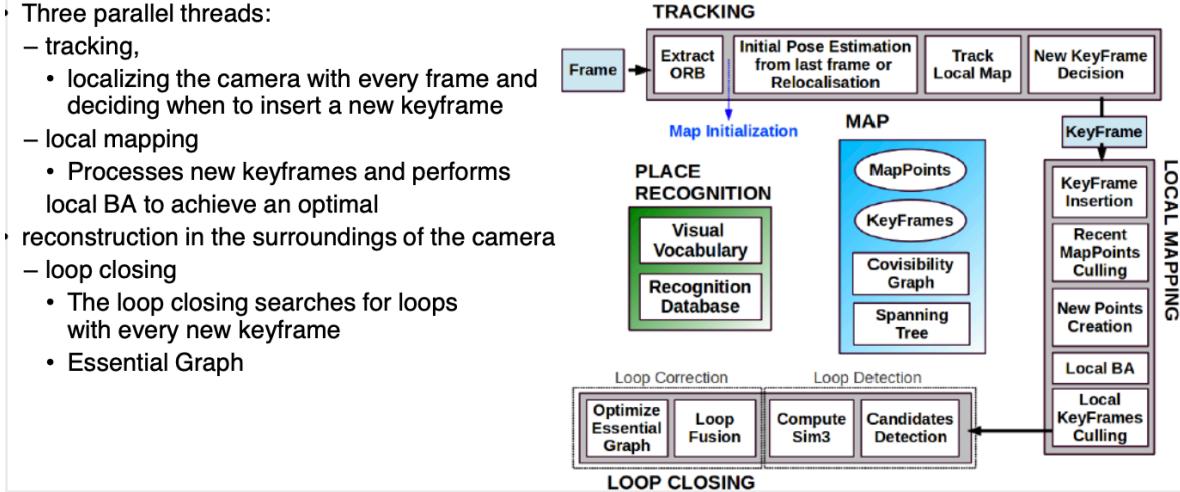
ORBSLAM algorithm

- One of the most well performing opensource implementations of visual slam.
- 3 parallel threads:
 - Tracking
 - Local mapping

- Loop closing

ORB SLAM

- The ORB-SLAM algorithm is one of the most well performing open-source implementations of visual SLAM.
- Three parallel threads:
 - tracking,
 - localizing the camera with every frame and deciding when to insert a new keyframe
 - local mapping
 - Processes new keyframes and performs local BA to achieve an optimal reconstruction in the surroundings of the camera
 - loop closing
 - The loop closing searches for loops with every new keyframe
 - Essential Graph



OTHER:

ICP: provides a rigid transformation between 2 point clouds

In mathematics, a **rigid transformation** (also called Euclidean transformation or Euclidean isometry) is a geometric transformation of a Euclidean space that preserves the Euclidean distance between every pair of points. The rigid transformations include rotations, translations, reflections, or their combination.

An **affine transformation** is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).