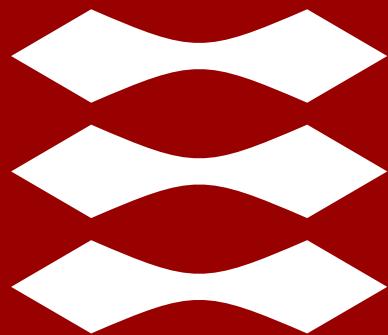


**DTU**



Lazaros Nalpantidis

# Introduction to 31392 Perception for Autonomous Systems

# Outline

- Overview
- Lecturers
- “Perception for Autonomous Systems”: *What and Why?*
- General Objectives
- Learning Objectives
- Literature
- Course Terminology
- Course Timeline
- Important Dates
- Exams / Evaluation
- Summary

# Overview

- 10 ECTS
- Technological Specialization course for the MSc. in Autonomous Systems
- Elective for other DTU MSc programmes
  
- All course lectures and activities:
  - Mondays 13:00-17:00 and Thursdays 08:30-12:00
  
- Lectures and Guest Lectures will be held:
  - physically: Building 341 / Auditorium 21
  - and also, live-streamed electronically on Zoom, Link:  
<https://dtudk.zoom.us/j/69140074865?pwd=SmhuYWI0RHJ2eEhTL21BQmgzMmRKZz09>
  
- Exercises, Weekly Projects and Final Project work will be supported:
  - purely electronically through Discord, Link: <https://discord.gg/dZhSUYPjDq>
  
- You are very welcome to use our room (Building 341 / Auditorium 21) both on Mondays and Thursdays for all these activities with your groups.
  
- Communication and Material Sharing will take place through DTU Learn

# Lecturers

- **Lecturers:**

- Lazaros Nalpantidis, Associate Professor, build. 326, room 018.
- Evangelos Boukas, Associate Professor, build. 326, room 020.



- **Teaching Assistants:**

- Jonathan Binner Becktor, PhD Student
- Junru Ren, PhD Student



# What is “Perception”?

# What is an “Autonomous System”?

# Why Autonomous Systems need Perception?

# General Objectives

- Theory and practical applications of perception for autonomous systems.
- Transform sensory input from a variety of imaging and 3D sensors into more abstract description.
- Allow autonomous systems to perceive their environment and act within it or interact with it.
- Both the mathematical descriptions and programming tools to implement such perception techniques.
  - Open source tools will be used as much as possible, such as Python, OpenCV and Open3D.
- Enable you to use the taught concepts and tools to further develop either embodied (e.g. robotic) or intangible (e.g. software agent) Autonomous Systems.

# Learning Objectives

A student who has met the objectives of the course will be able to:

- Describe the steps that lead to 3D reconstruction using multiple views.
- Define commonly used image feature extraction and matching techniques.,
- Discuss characteristics of various ranging sensors and techniques.
- Apply software tools to process 3D point clouds.
- Combine visual and 3D sensory input with state estimation techniques.
- Describe the differences between classical and learning-based object/scene classification techniques.
- Describe the different steps in visual odometry and explain the operation of the related algorithms.
- Combine the taught material to propose and describe possible implementations of further perception applications.

# Literature

In this course we will be using 2 books as our main sources. However, topic-specific research papers and other additional material might be distributed in certain classes.

- **Book A:**

- Richard Szeliski. 2010. Computer Vision: Algorithms and Applications (1st. ed.). Springer-Verlag, Berlin, Heidelberg.

- **Book B:**

- David A. Forsyth and Jean Ponce. 2002. Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference.

# Course Terminology (1/2)

- **Lectures & Exercises:**

- As a general rule, Mondays will start with Lectures of new theoretical topics and finish with Exercises applying the topics of the day.
  - » Exercises will be not be accompanied by solutions and it is up to the students to solve them, using the provided material. No hand-in is required for Exercises.

- **Weekly Projects:**

- Thursdays will be mainly devoted to Weekly Projects. They are small, self-contained assignments where students need to exhibit self-driven behavior, use the taught material but also possibly go beyond that in order to solve them. No report is required for Weekly projects.
  - » No report is required for Weekly projects. In most cases, there are more than one correct ways of dealing with the Weekly Projects. Due to their open nature and to avoid limiting exploratory learning of students, no exact solutions to the Weekly projects will be provided. Instead, an indicative list of abstract steps in the form of hints, will be provided.

- **Mini-Quizzes:**

- Throughout the course we will be having some Mini-Quizzes. They are short multiple-choice quizzes that will prepare the students for the style of questions expected in the final exam.

# Course Terminology (2/2)

- **Guest Lectures:**
  - Towards the end of the course, a number of Guest Lectures are invited, where externals will link the course topics to real applications and industrial needs.
- **Final Project:**
  - The last period of the course is devoted to a Final Project,
  - Students are expected to work in groups of 5 people and solve a bigger assignment that will be given to them.
  - The outcome of the Final Project is a report of  $10\pm2$  pages that includes a link to a video demonstrating the group's main achievements.
  - A positive evaluation (pass/fail) of the report is mandatory for the group members to participate in the final exam.

# Course Timeline

Calendar Week		F2A (Monday 13:00-17:00)		F2B (Thursday 08:00-12:00)
5	Mon 31. Jan	<ul style="list-style-type: none"> <li>• Introduction and Plan</li> <li>• Lecture: Image Processing</li> <li>• Python Environment Setup</li> </ul>	Thu 3. Feb	<ul style="list-style-type: none"> <li>• Group Formation</li> <li>• Weekly Project: Image Processing</li> </ul>
6	Mon 07. Feb	<ul style="list-style-type: none"> <li>• Lecture: Image Feature Description and Matching</li> <li>• Exercises</li> </ul>	Thu 10. Feb	<ul style="list-style-type: none"> <li>• Mini-Quiz</li> <li>• Weekly Project: Image Features</li> </ul>
7	Mon 14. Feb	<ul style="list-style-type: none"> <li>• Lecture: Multiple View Geometry 1</li> <li>• Exercises</li> </ul>	Thu 17. Feb	<ul style="list-style-type: none"> <li>• Weekly Project: Multiple View Geometry</li> </ul>
8	Mon 21. Feb	<ul style="list-style-type: none"> <li>• Lecture: Multiple View Geometry 2 / Ranging</li> <li>• Exercises</li> </ul>	Thu 24. Feb	<ul style="list-style-type: none"> <li>• Mini-Quiz</li> <li>• Weekly Project: Multiple View Geometry</li> </ul>
9	Mon 28. Feb	<ul style="list-style-type: none"> <li>• Lecture: 3D Point Cloud Processing 1</li> <li>• Exercises</li> </ul>	Thu 03. Mar	<ul style="list-style-type: none"> <li>• Weekly Project: 3D Point Cloud Processing</li> </ul>
10	Mon 07. Mar	<ul style="list-style-type: none"> <li>• Lecture: 3D Point Cloud Processing 2</li> <li>• Exercises</li> </ul>	Thu 10. Mar	<ul style="list-style-type: none"> <li>• Mini-Quiz</li> <li>• Weekly Project: 3D Point Cloud Processing</li> </ul>
11	Mon 14. Mar	<ul style="list-style-type: none"> <li>• Lecture: State Estimation</li> <li>• Exercises</li> </ul>	Thu 17. Mar	<ul style="list-style-type: none"> <li>• Weekly Project: State Estimation</li> </ul>
12	Mon 21. Mar	<ul style="list-style-type: none"> <li>• Lecture: Classification</li> <li>• Exercises</li> </ul>	Thu 24. Mar	<ul style="list-style-type: none"> <li>• Weekly Project: Classification</li> </ul>
13	Mon 28. Mar	<ul style="list-style-type: none"> <li>• Lecture: Visual Odometry</li> <li>• Final Project Description</li> <li>• Exercises</li> </ul>	Thu 31. Mar	<ul style="list-style-type: none"> <li>• Mini-Quiz</li> <li>• Weekly Project: Visual Odometry</li> </ul>
14	Mon 4. Apr	<ul style="list-style-type: none"> <li>• Lecture: SLAM</li> <li>• Final Project</li> </ul>	Thu 7. Apr	<ul style="list-style-type: none"> <li>• Final Project</li> </ul>
15	-----	Easter Holiday	-----	Easter Holiday
16	-----	Easter Holiday	Thu 21. Apr	<ul style="list-style-type: none"> <li>• Guest Lecture</li> <li>• Final Project</li> </ul>
17	Mon 25. Apr	<ul style="list-style-type: none"> <li>• Guest Lecture</li> <li>• Final Project</li> </ul>	Thu 28. Apr	<ul style="list-style-type: none"> <li>• Guest Lecture</li> <li>• Final Project</li> </ul>
18	Mon 2. May	<ul style="list-style-type: none"> <li>• Final Project</li> </ul>	Thu 5. May	<ul style="list-style-type: none"> <li>• Final Project</li> </ul>
19	Mon 9. May	<ul style="list-style-type: none"> <li>• Final Project</li> <li>• Report Hand-in</li> </ul>	-----	

# Important Dates

<b>3. February 2021</b>	Deadline for group formation. Students form groups of 5 people to work together on Exercises, Weekly Projects and Final Project.
<b>28. March 2021</b>	Announcement from the teachers of the Final Project description.
<b>9. May 2021</b>	Group report hand-in.
<b>11. May 2021</b>	Information about non-qualification for participating in the exam by email to the members of groups with inadequate reports.
<b>16. May 2021</b>	Exam

# Exams / Evaluation

The assessment for this course will take place by evaluating together the group report and the personal final exam score:

- **Report:**

- Groups of 5 people will submit reports about their Final Project by the end of the course. Submission will take place by uploading on DTU Learn.
- The report needs to be  $10\pm2$  pages and include a link to a video demonstrating the group's main outcomes.
- Approval of report (based on a pass/fail evaluation) is mandatory for the group members to participate in the exam.
- The report will account for 10% of the final grade.

- **Exam**

- The examination type is multiple-choice test with questions and problems on the taught material.
- The duration of the exam will be 4 hours.
- The exam score will account for 90% of the final grade.

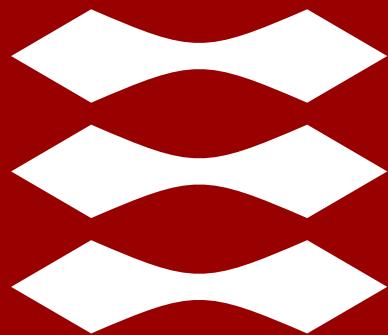
# Summary

- Our goal is to learn a lot and connect theory to real problems.
- We need your feedback during the semester!
- We are looking forward to an exciting course!

Lazaros Nalpantidis

# Introduction to 31392 Perception for Autonomous Systems

**DTU**



Lazaros Nalpantidis

# Image Processing

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

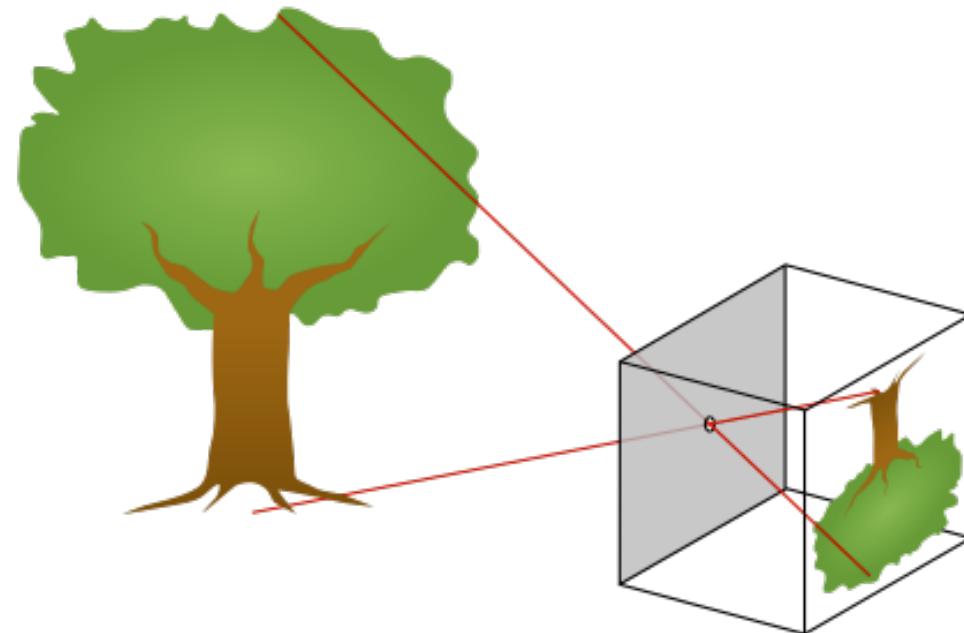
- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

# What is Image Processing?

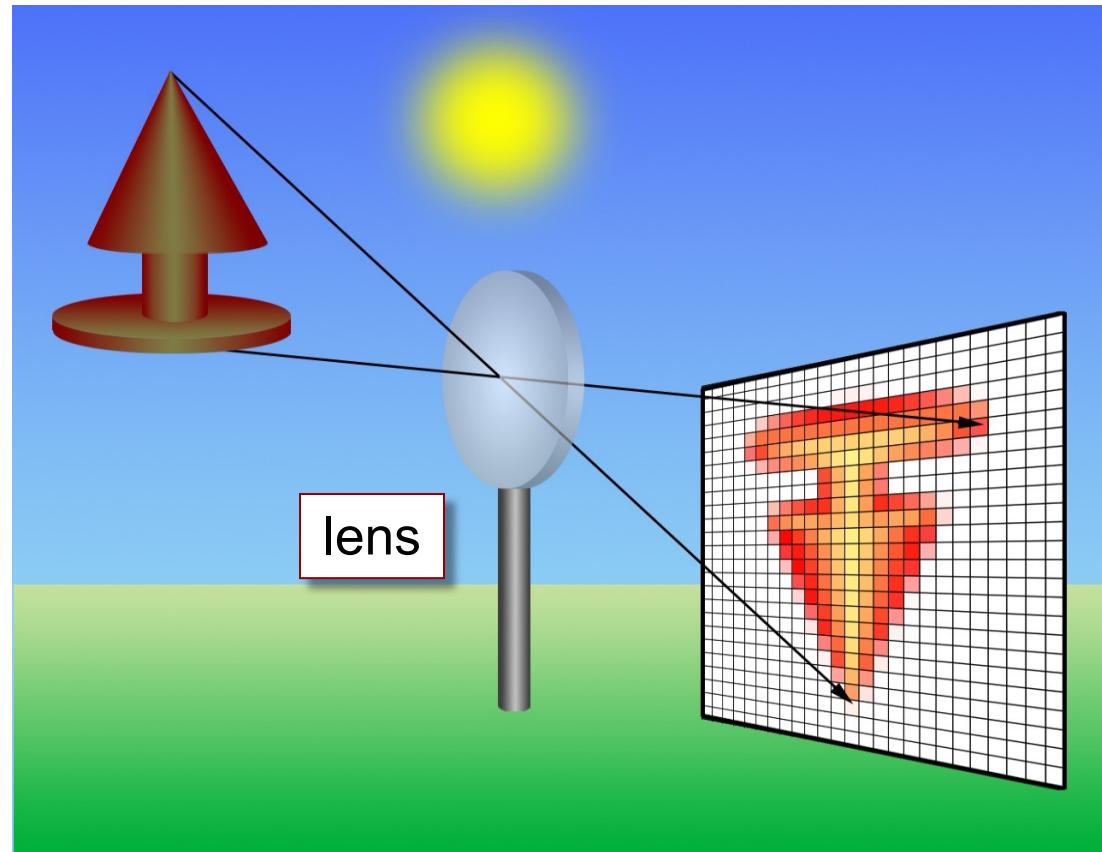
- Image processing is:
  - the operations we perform on an image to change or enhance it and make it suitable for further analysis,
  - so that useful information can be highlighted or get extracted from it.

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

- Pinhole model

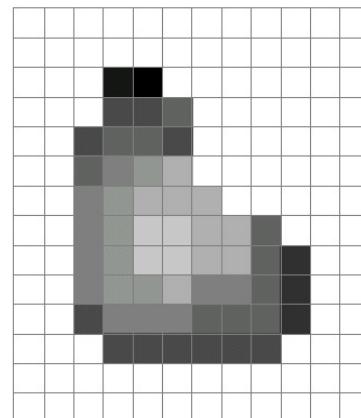
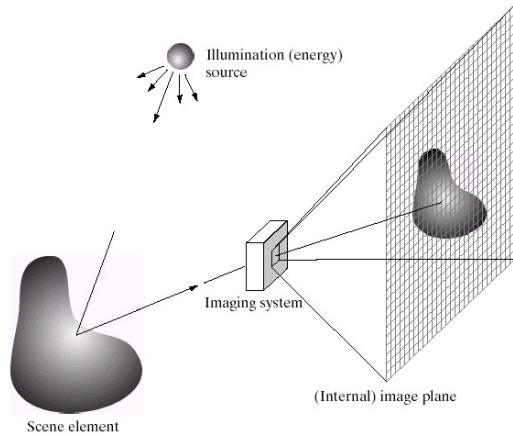


- Image Formation



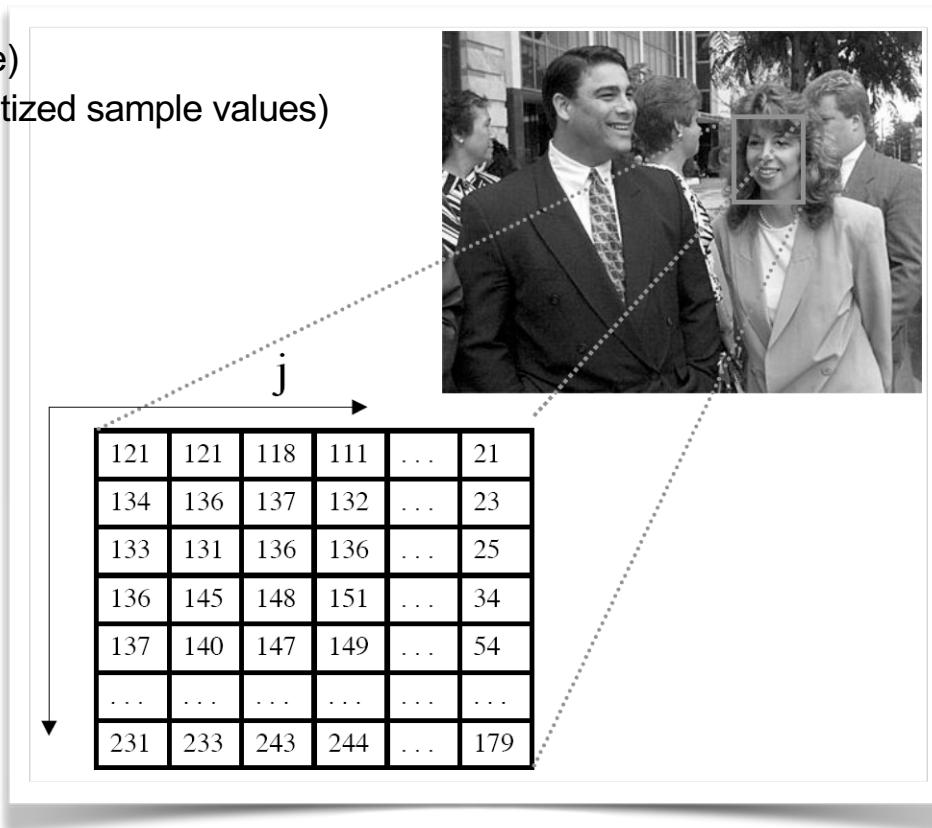
*creditis: Richard Alan Peters II*

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

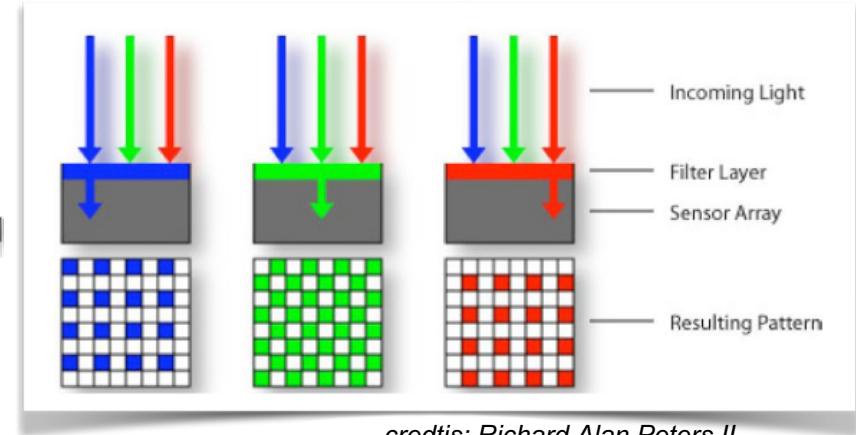
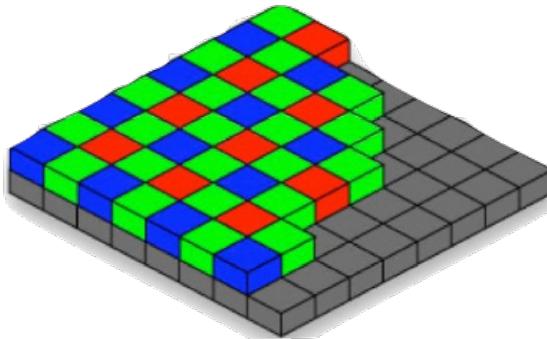
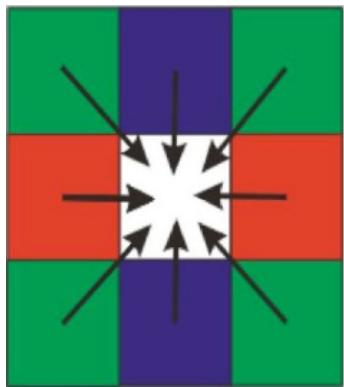
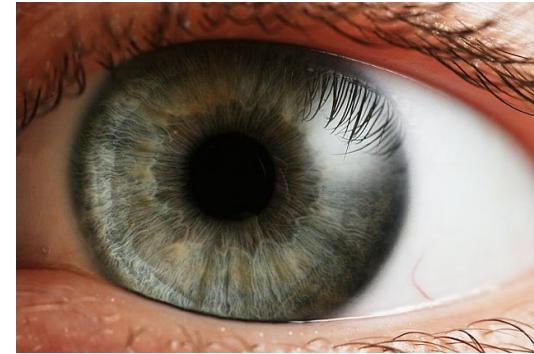


255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255	255	255
255	255	96	127	145	175	255	255	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	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)

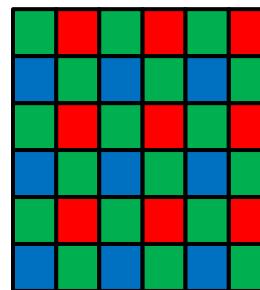


- Color Images
  - Bayer Color Filter Array

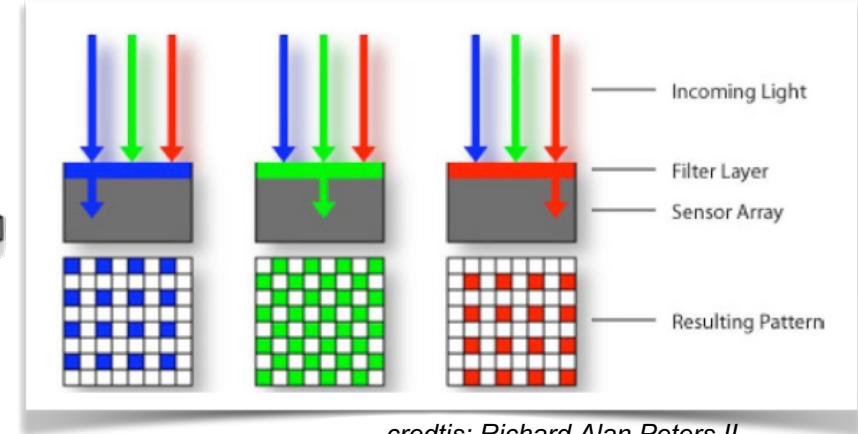
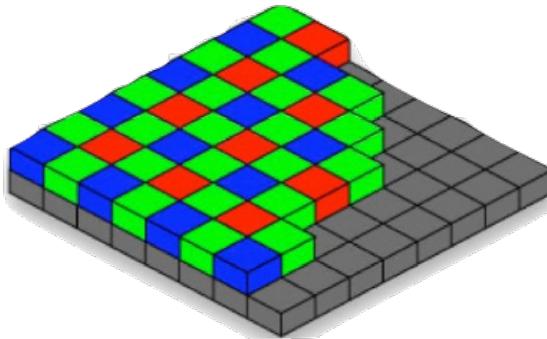
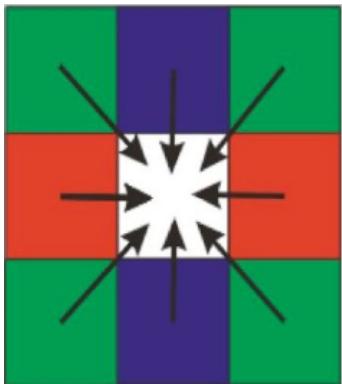
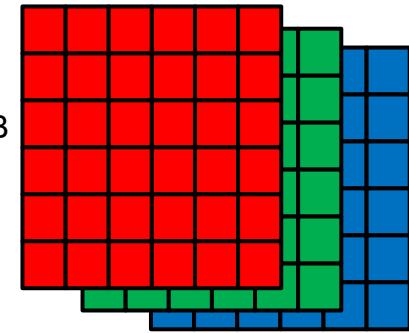


credit: Richard Alan Peters II

- Color Images
  - Bayer Color Filter Array



Reproduce full RGB

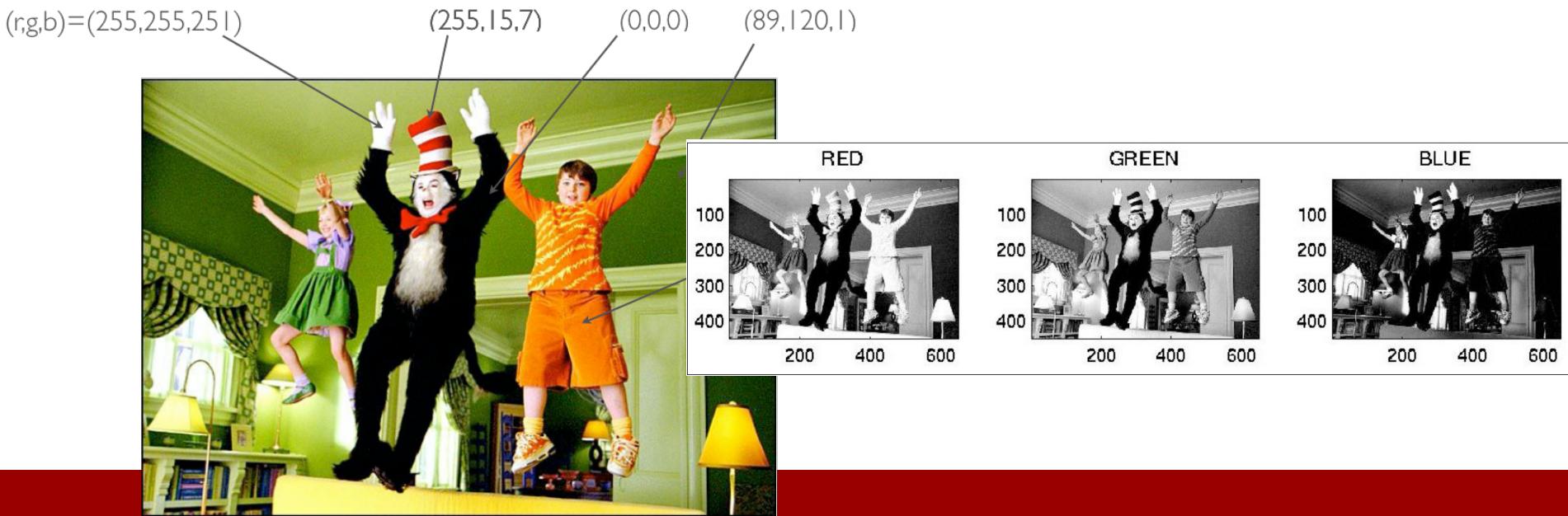


credit: Richard Alan Peters II

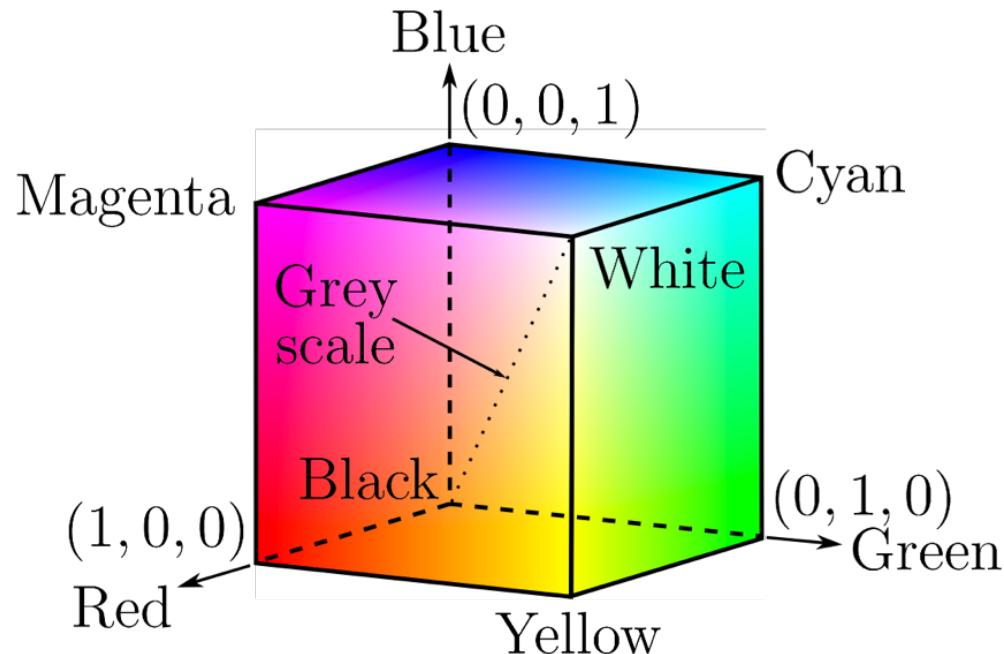
- Color Images
  - are composed of 3 2D images: R(x,y), G(x,y), B(x,y)
  - Each pixel (x,y) of these 3 images consists of values between 0 and 255
  - The values of a specific pixel  $(x_1, y_1)$  in the 3 images R, G, B describe the red-ness, green-ness and blue-ness of that particular pixel.



- 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

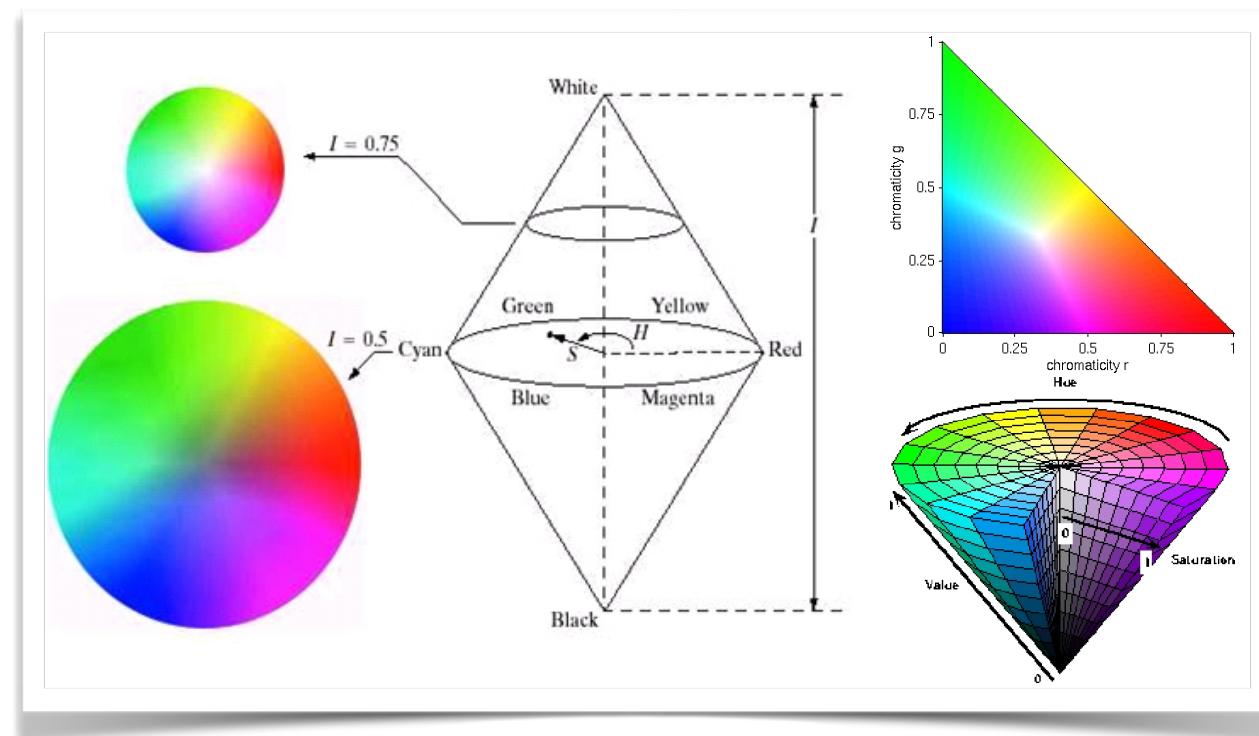
• **Hue:** color, chromatic information



• **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	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	50	0	20	10	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	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

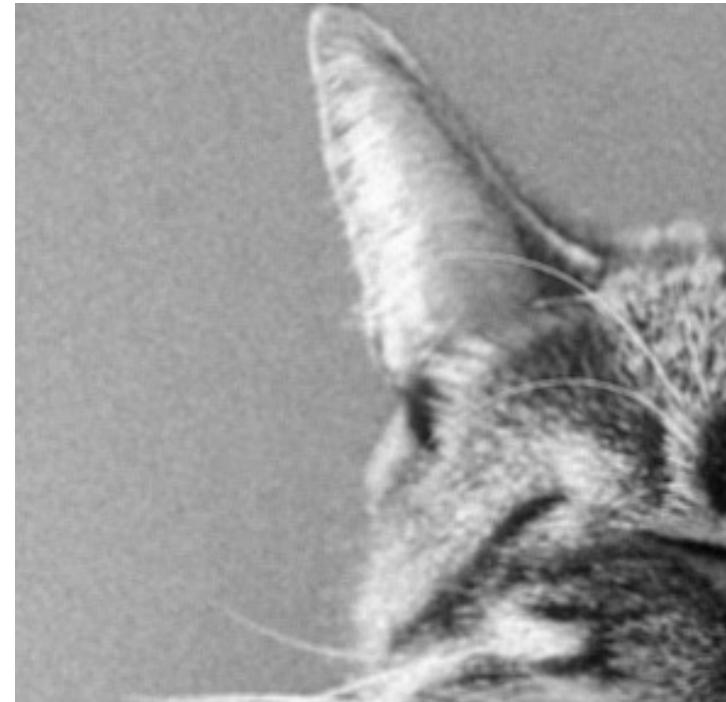
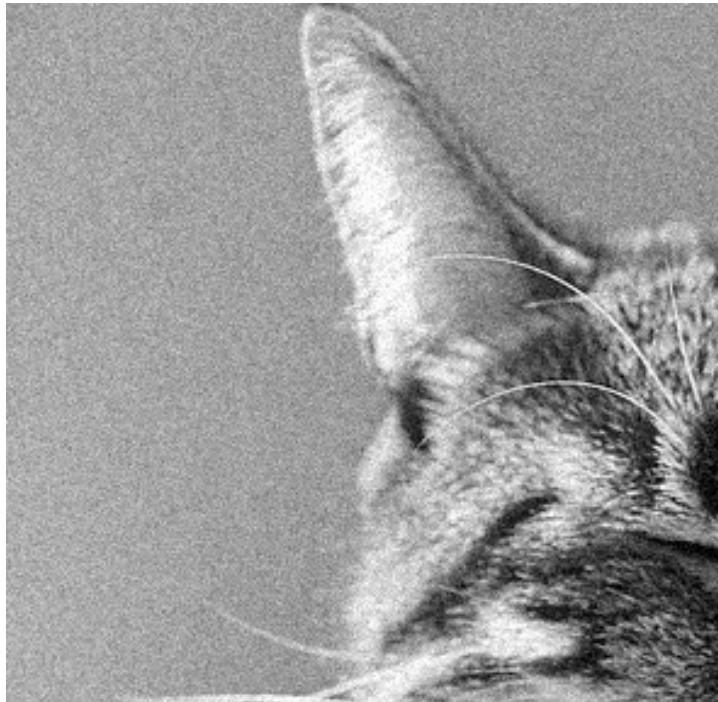
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = \\ 7.77$$

# 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

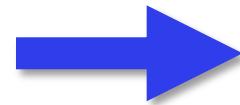
Kernel / filter

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

# 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

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

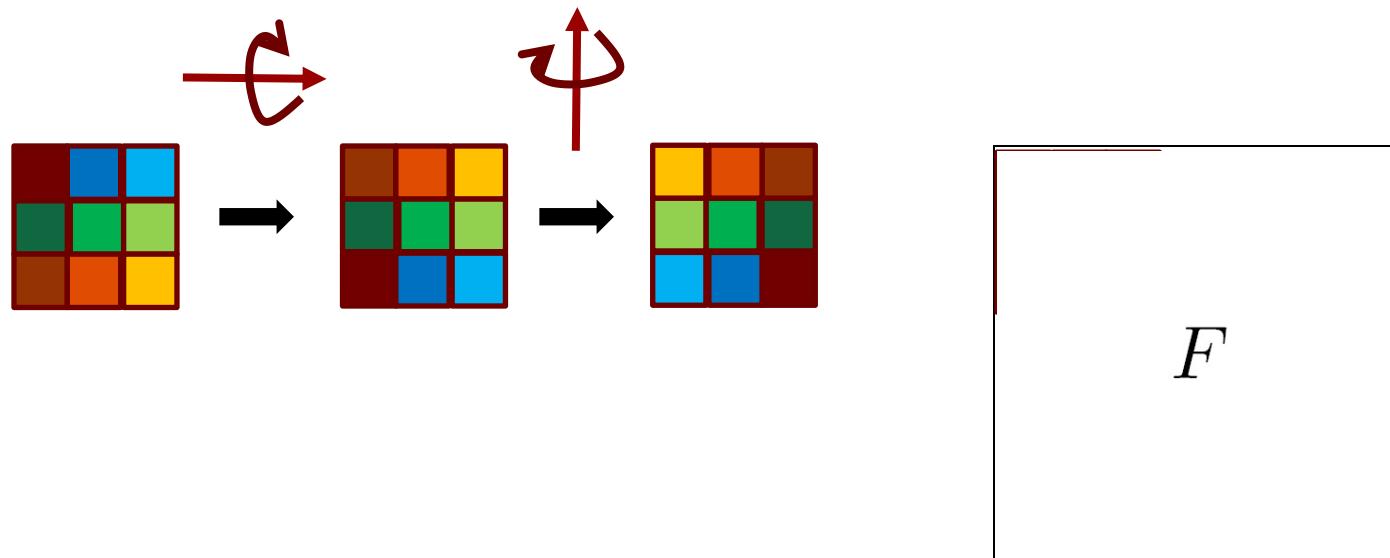
1	2	3
4	5	6
7	8	9

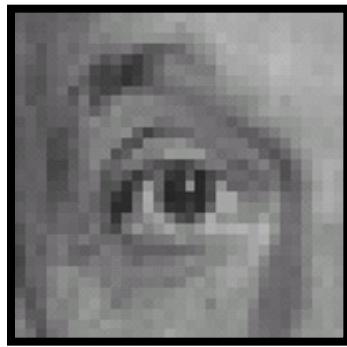
W

1	2	3
4	5	6
7	8	9

f

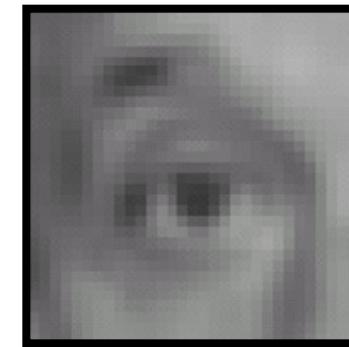
$$1*9 + 2*8 + 3*7 + 4*6 + 5*5 + 6*4 + 7*3 + 8*2 + 9*1$$



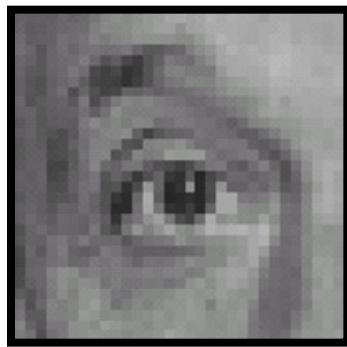


Original (f)

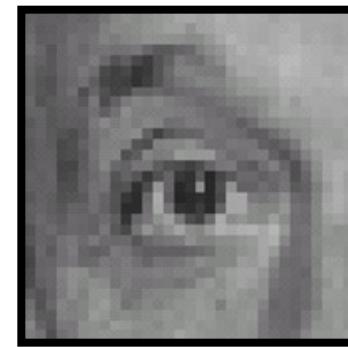
$$\ast \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \text{Kernel (k)}$$

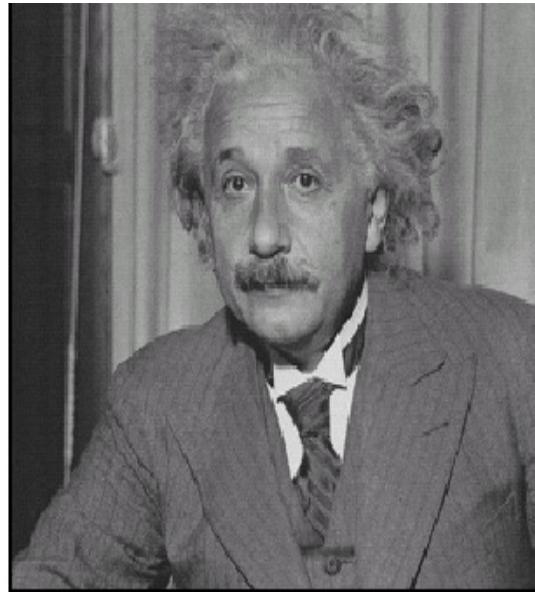


Blur (with a mean filter) (g)

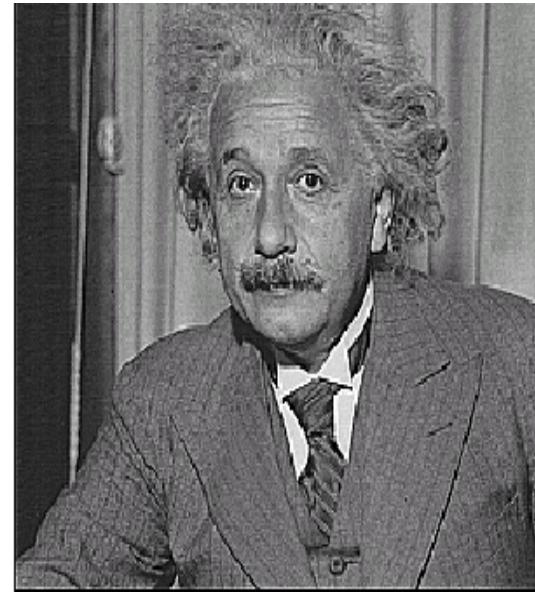
Original ( $f$ ) $*$ 

0	0	0
0	1	0
0	0	0

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



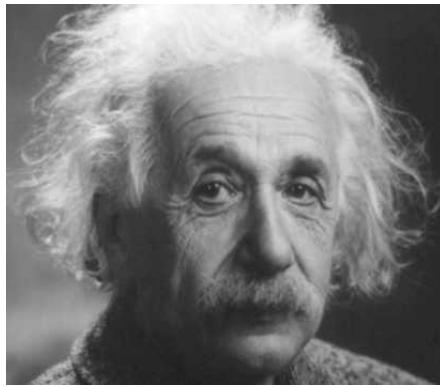
**before**



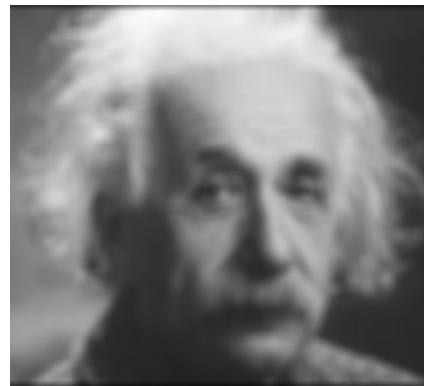
**after**

# Sharpening

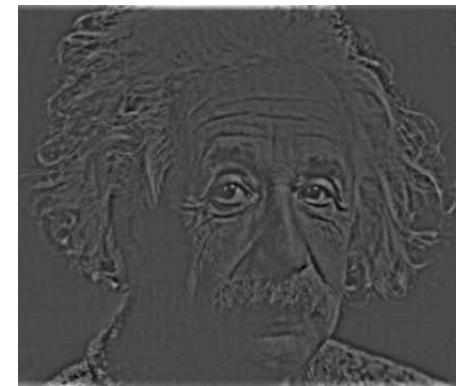
- What does blurring take away?



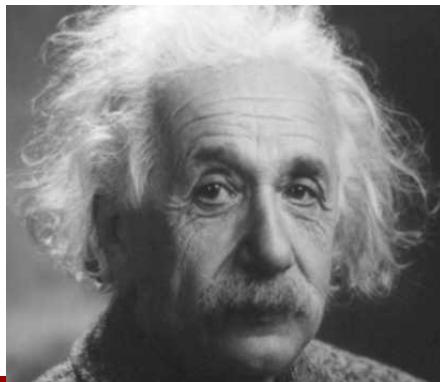
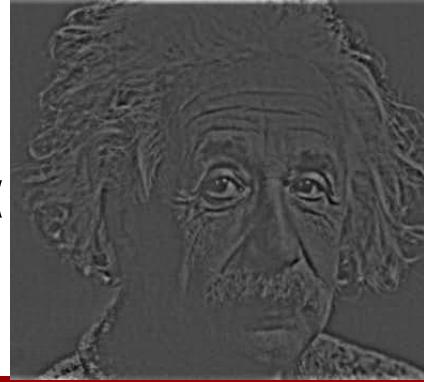
-



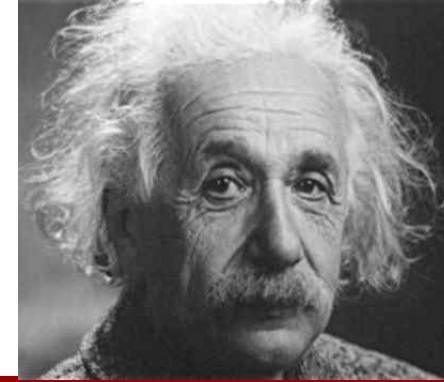
=

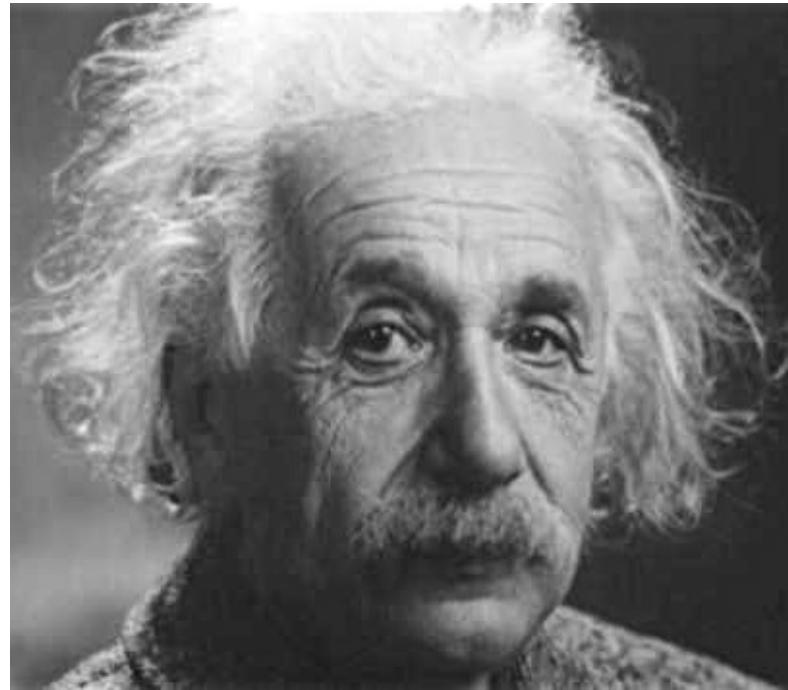


Let's add it back:

 $+ \alpha$ 

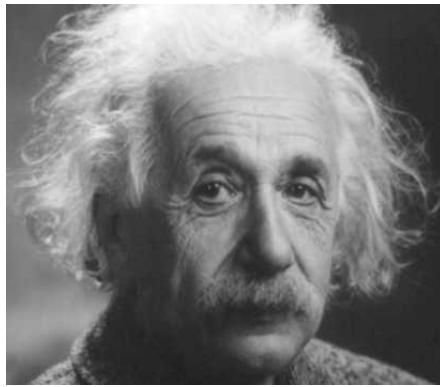
=



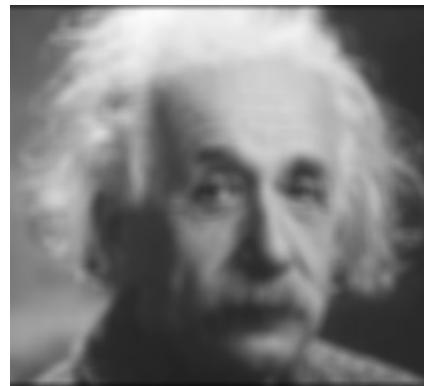


# Sharpening

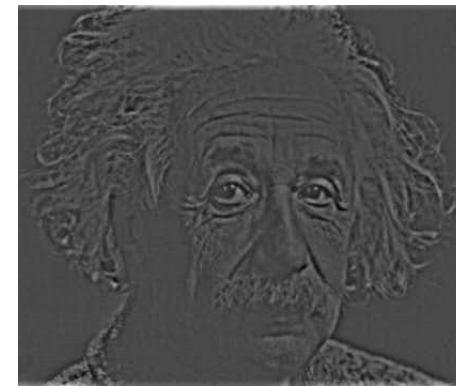
- What does blurring take away?



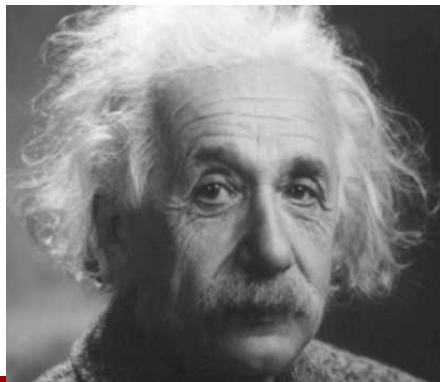
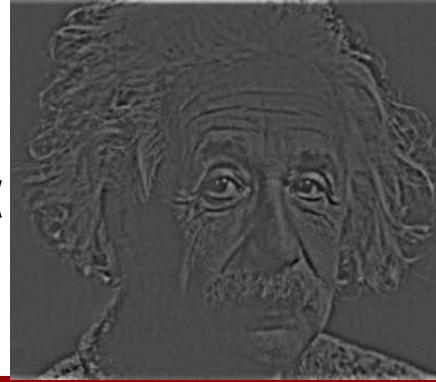
-



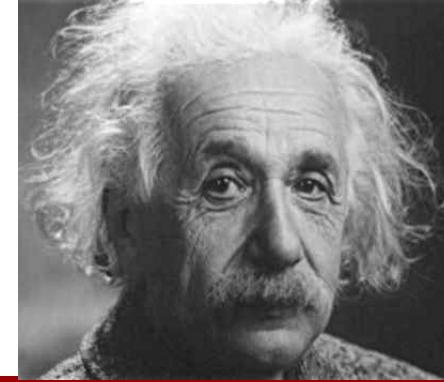
=



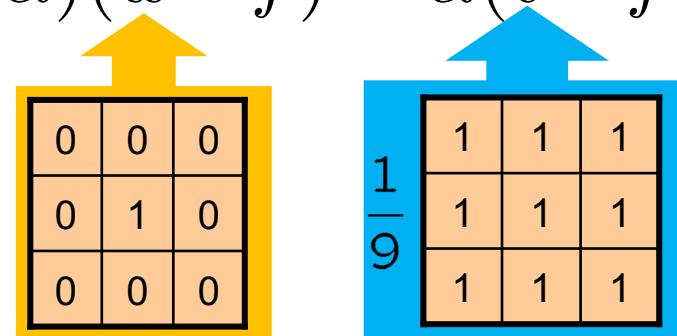
Let's add it back:

 $+ \alpha$ 

=

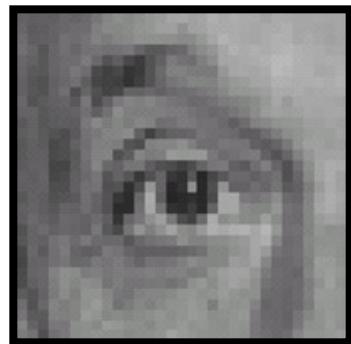


$$\begin{aligned}f_{sharp} &= f + \alpha(f - f_{blur}) \\&= (1 + \alpha)f - \alpha f_{blur} \\&= (1 + \alpha)(w * f) - \alpha(v * f)\end{aligned}$$



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

# 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

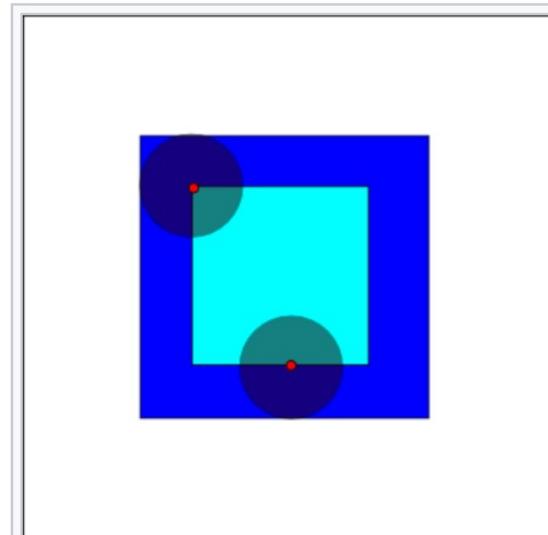


- What is Image Processing?
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connected Components Analysis
- Summary

# Morphology

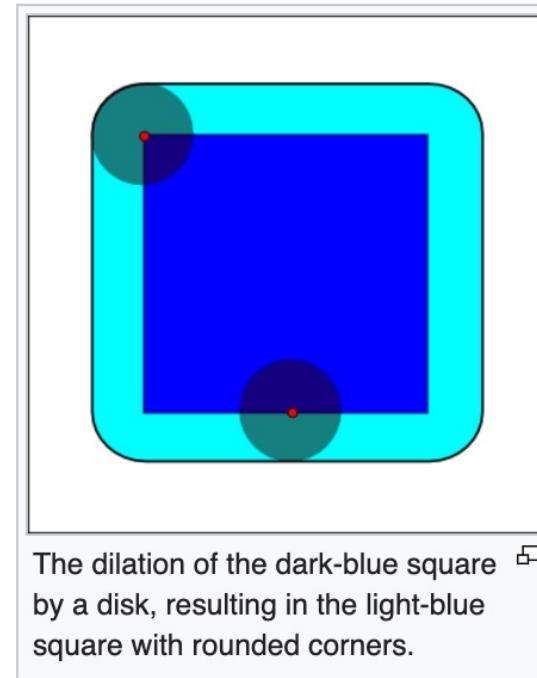
- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
  - Erosion
  - Dilation
  - Opening
  - Closing
    - All of them rely on convolution with a Structuring Element
      - » The Structuring Element can be a disk, rectangle, or of any other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

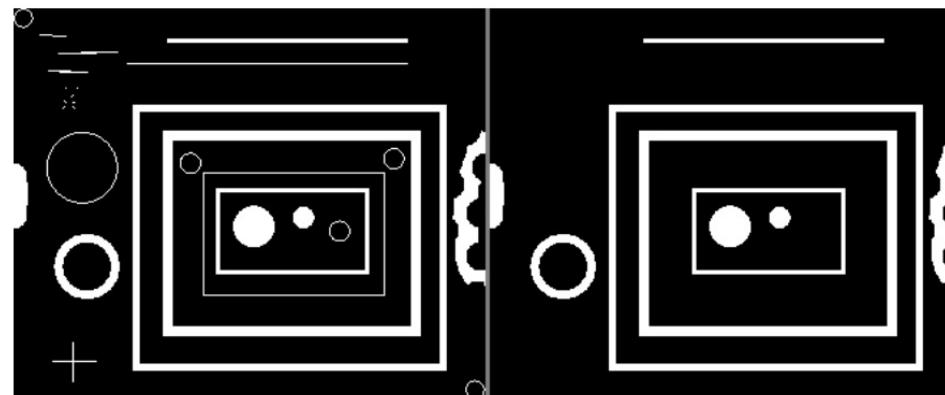


The erosion of the dark-blue square by a disk, resulting in the light-blue square.

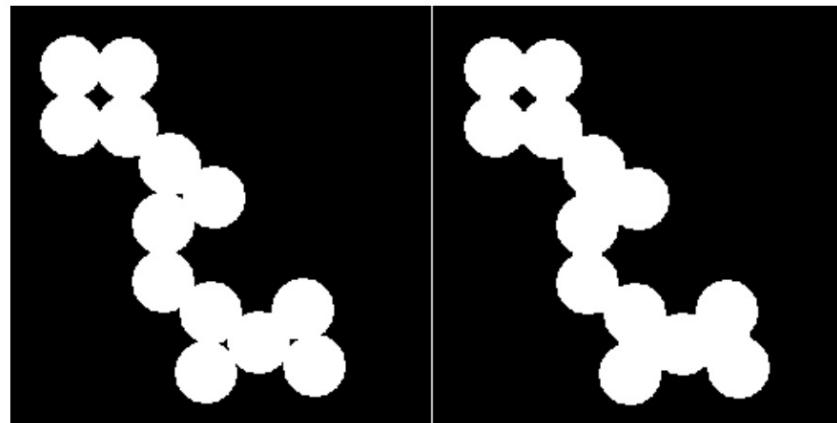
- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
  - Dilation



- 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.



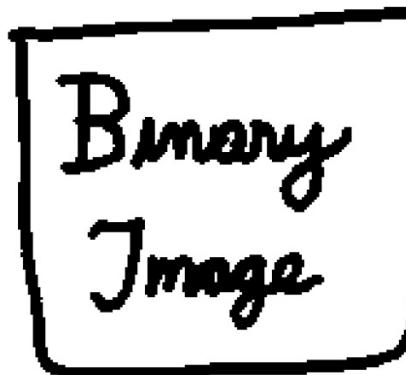
- The most common binary image operations are called Morphological Operations
- Basic Morphological Operations:
  - Closing = Dilation and then Erosion
    - » Morphological closing is useful for filling small holes from an image while preserving the shape and size of the objects in the image.



- 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)

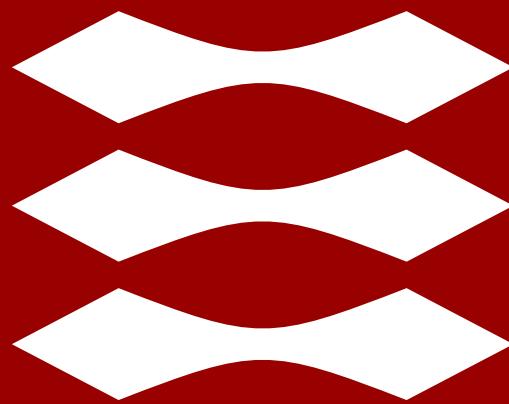
# Summary

- 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

**DTU**



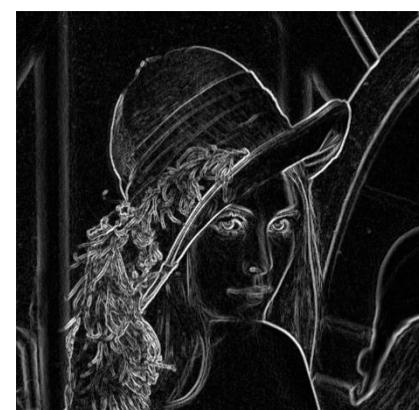
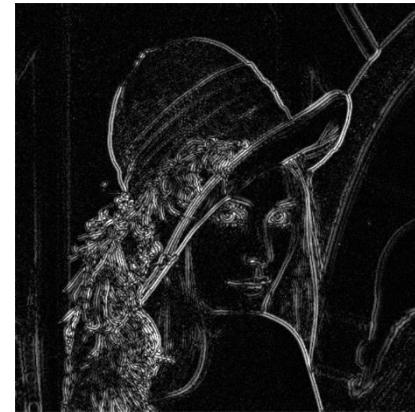
Perception for Autonomous Systems 31392:

# Edge Detection

Lecturer: Evangelos Boukas—PhD

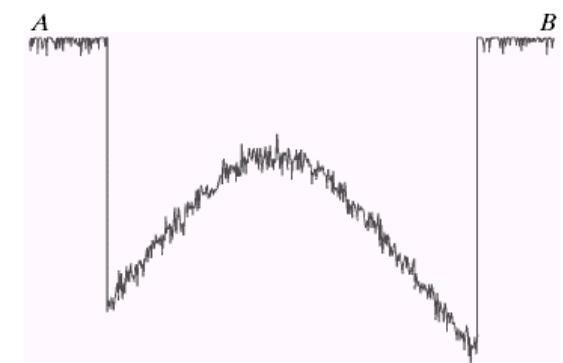
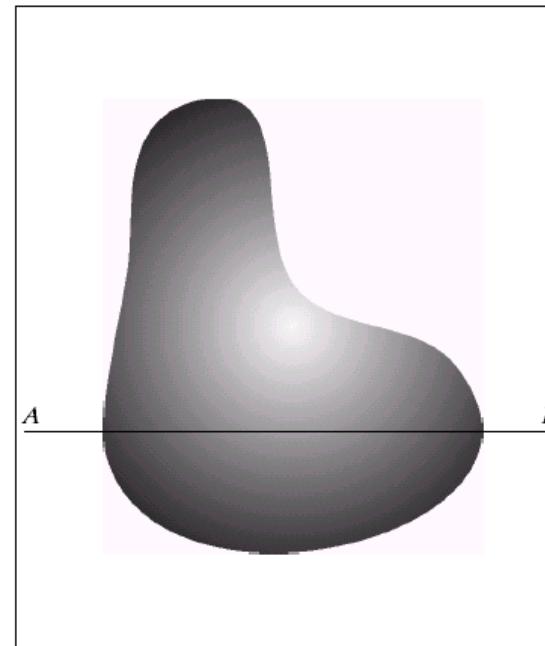
# Edge Detection

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



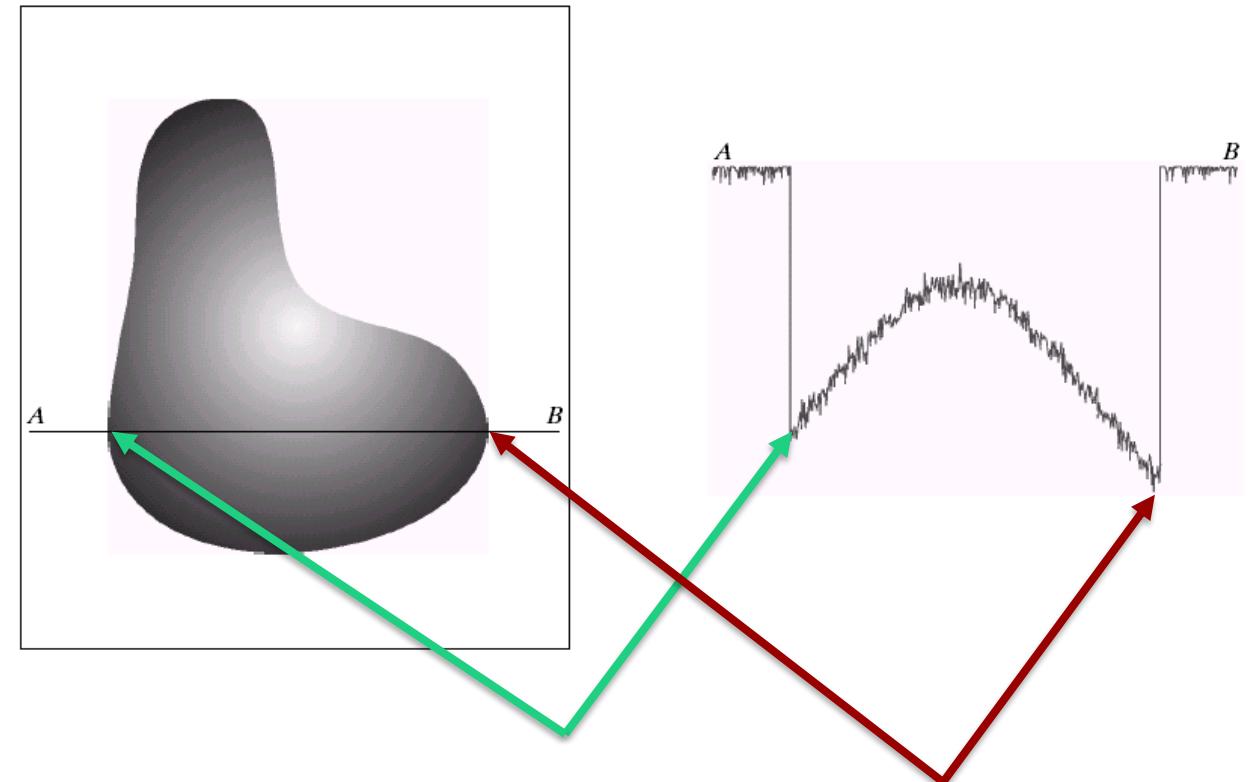
# Edge Detection

- What is and edge?



# Edge Detection

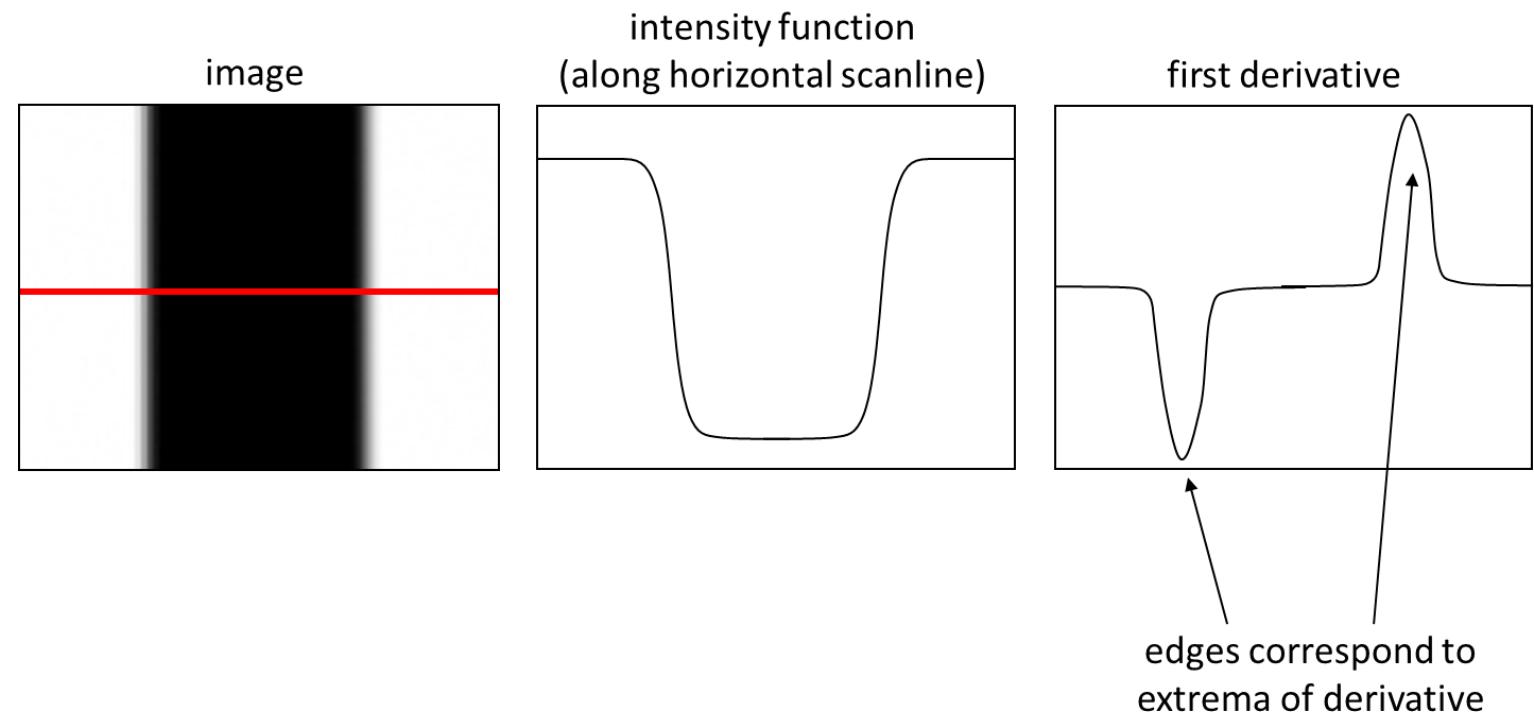
- What is and edge?



# Edge Detection

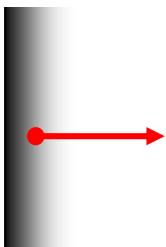
- What is and edge?
- Derivative of an Image:

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

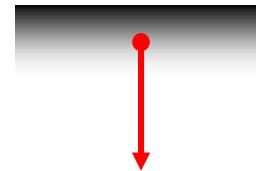


- The gradient is a vector which points in the direction of most rapid change in intensity:

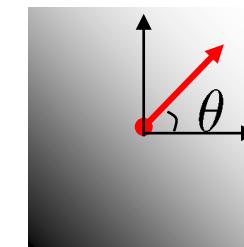
$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$



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

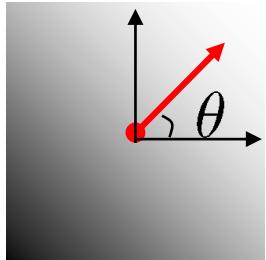


$$\nabla f = \left[ 0, \frac{\partial f}{\partial y} \right]$$



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

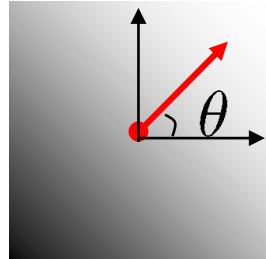
# Gradient Simplified



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

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

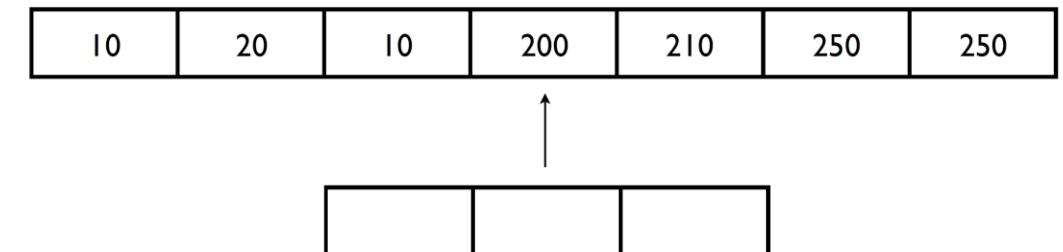
# Gradient Simplified



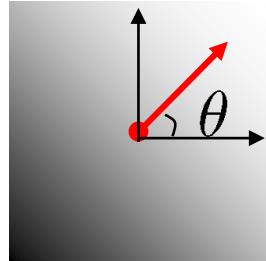
$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \longrightarrow \frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

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

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



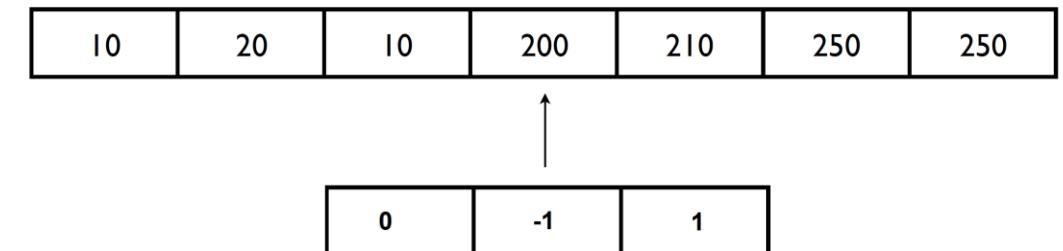
# Gradient Simplified



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

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

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



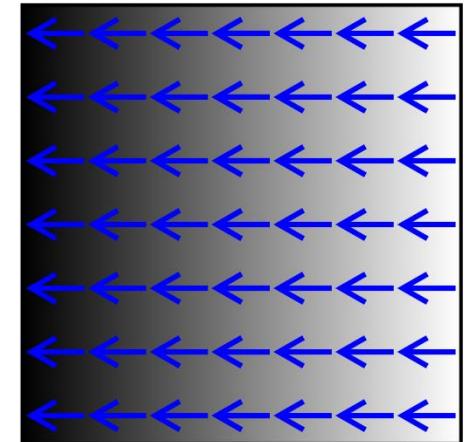
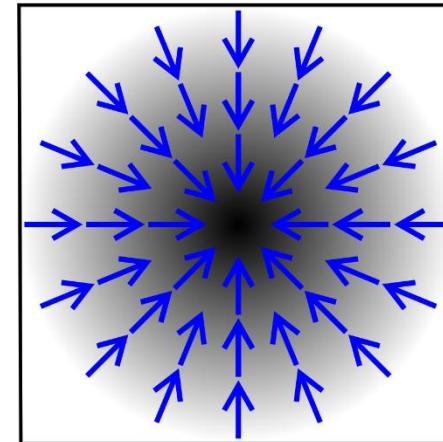
# Gradient Simplified

- The gradient is defined by its orientation:

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

- and magnitude:

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



- Practically:

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

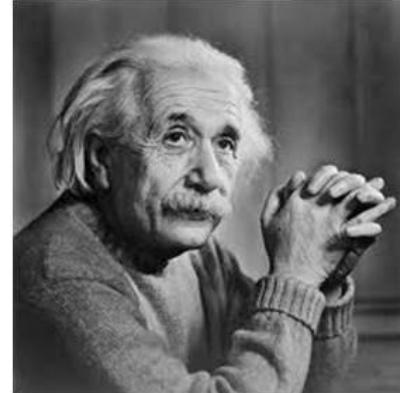
$$g_y = \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

- Magnitude:

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

- Orientation:

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



# Derivative Filters

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

Scharr

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

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

Roberts

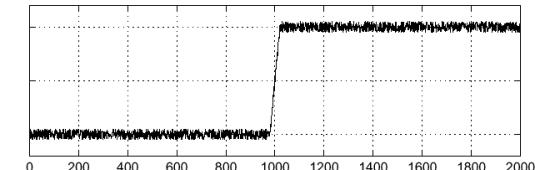
0	1
-1	0

1	0
0	-1

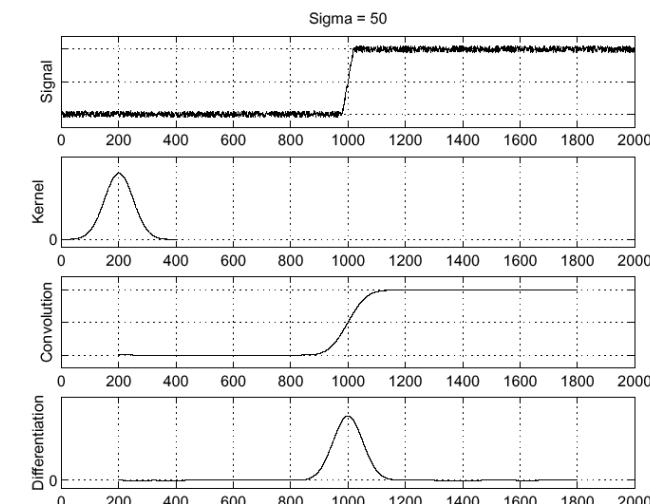
# Preprocessing to Edge Detection

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

$$f(x)$$



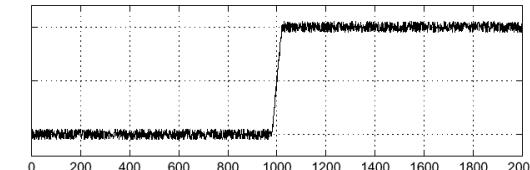
- To overcome this issue we can smooth the signal beforehand



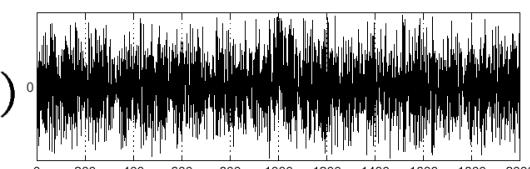
# Preprocessing to Edge Detection

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

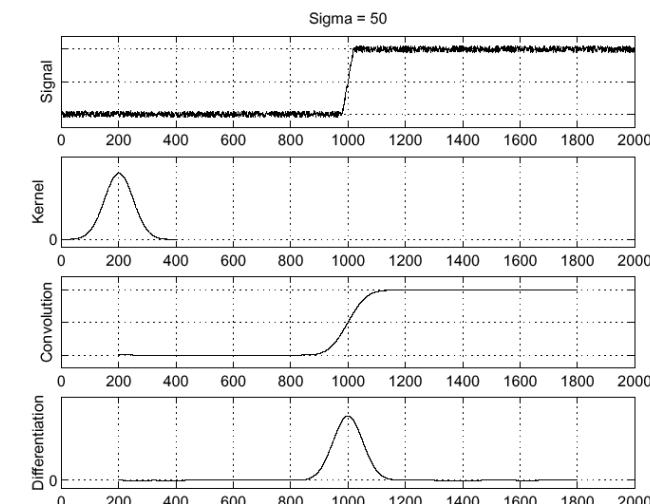
$$f(x)$$



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

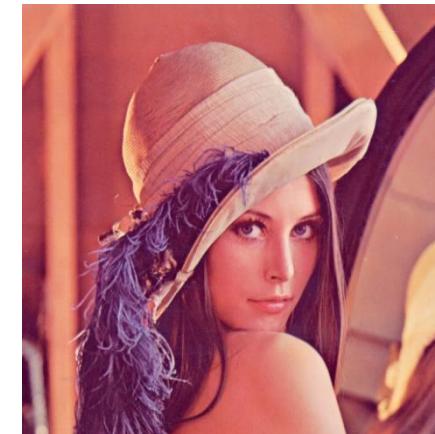
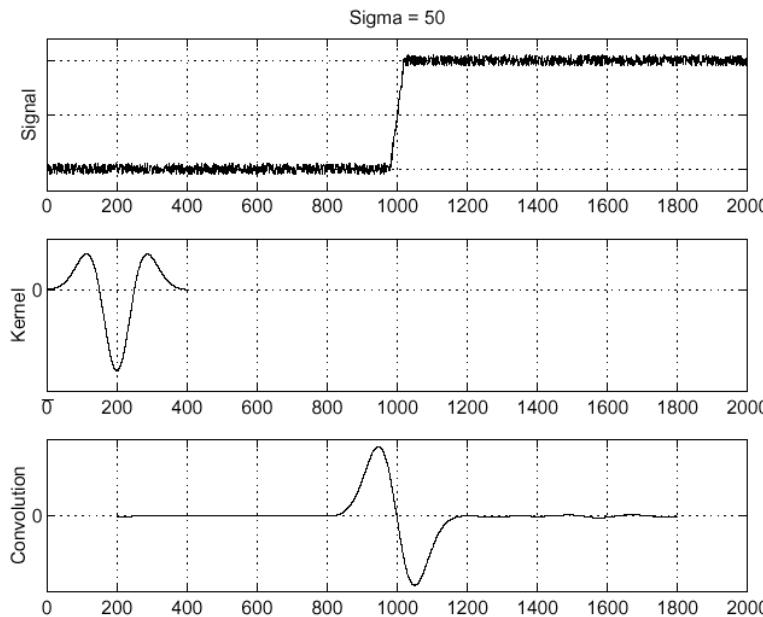


- To overcome this issue we can smooth the signal beforehand



# Laplacian of Gaussian

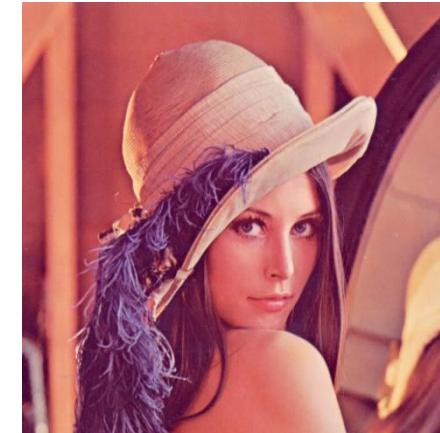
- $\frac{\partial^2}{\partial x^2}(h \star f)$



# Canny Edge Detection

Example of a Complex system:

- Noise reduction
- Gradient calculation
- Non-maximum suppression
- Double threshold
- Edge Tracking by Hysteresis.

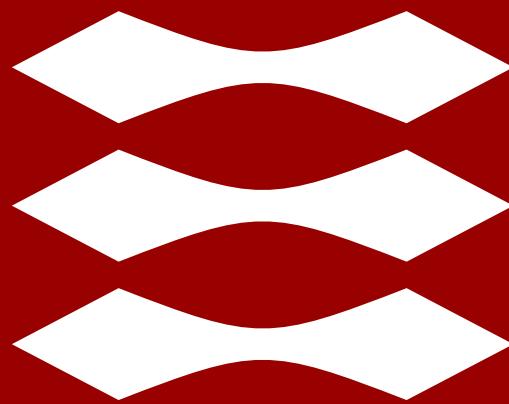


Perception for Autonomous Systems 31392:

# 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

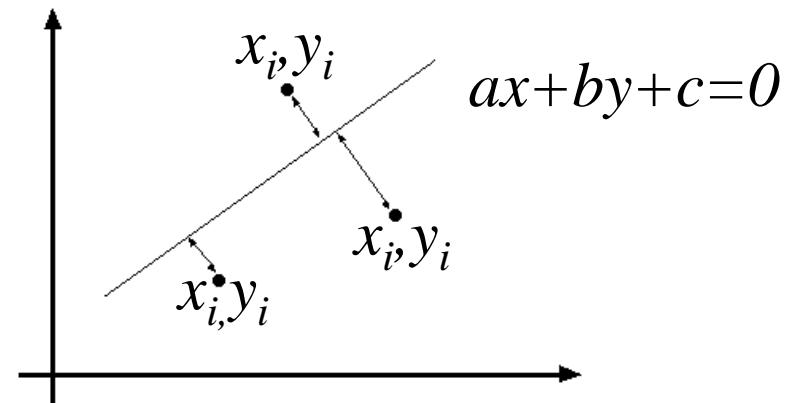
- Let the line depicted here be described by:  
 $ax+by+c = 0$
- Then the distance of a point  $x_i, y_i$  is defined as:

$$|ax_i + by_i + c|$$

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

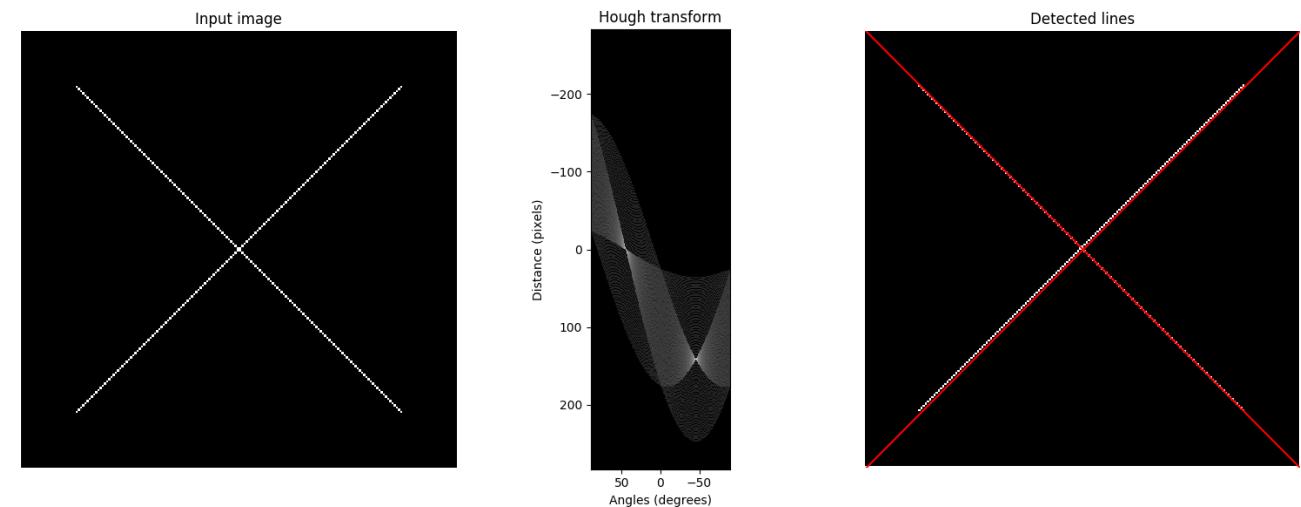
$$E = \sum_{i=1}^n (ax_i + by_i + c)^2$$

- 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:

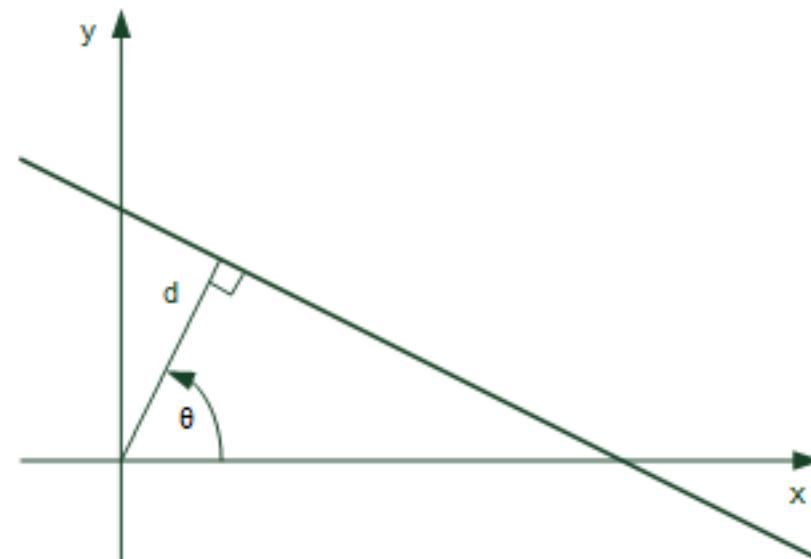
$$Y = aX + b$$

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

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

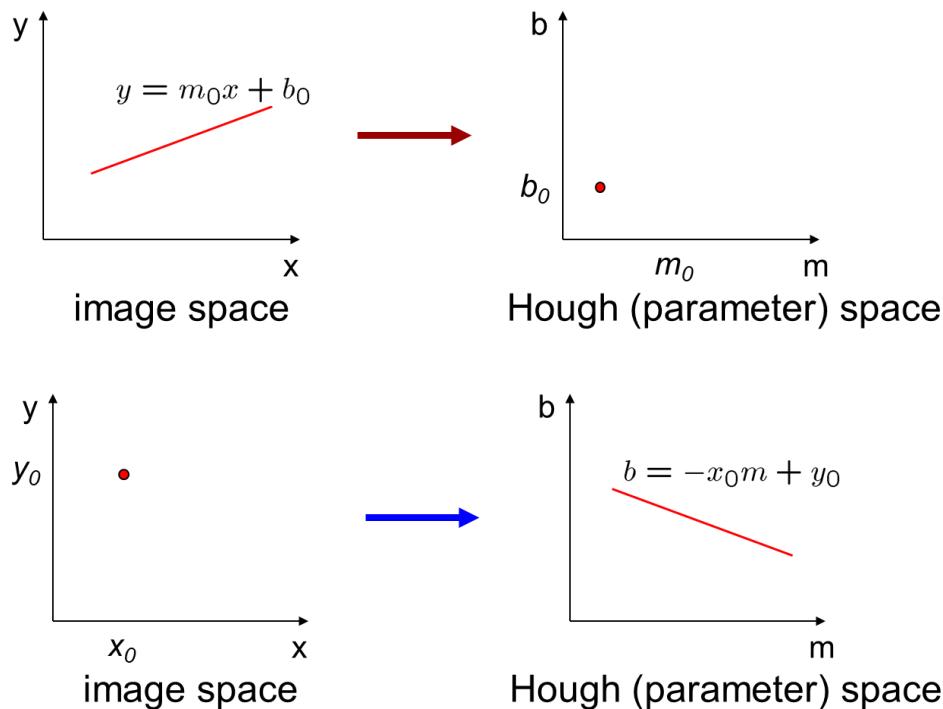
$d$  : perpendicular distance from  
line to origin

$\theta$  : angle the perpendicular  
makes with the x-axis

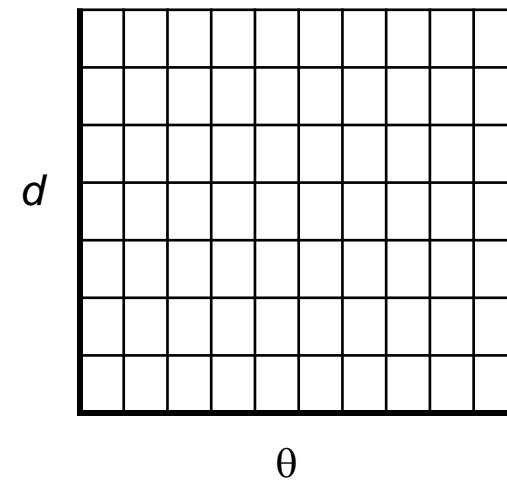


# Hough Transform (2/4)

- Next step:
  - Initialize the grid using  $(d, \theta)$

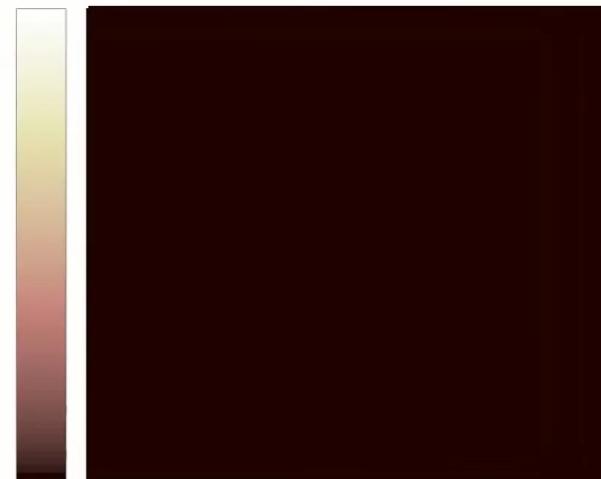


$H$ : accumulator array (votes)

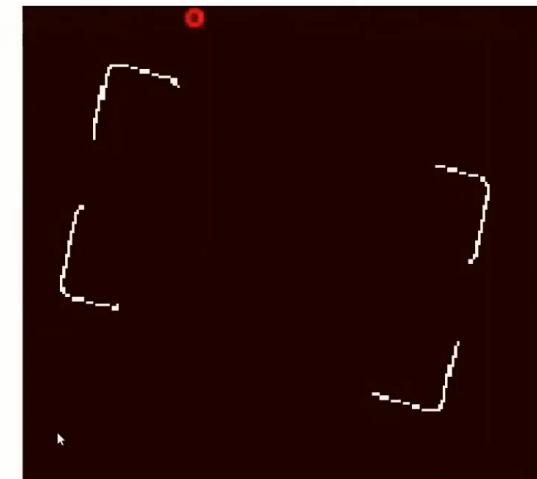


# Hough Transform (3/4)

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

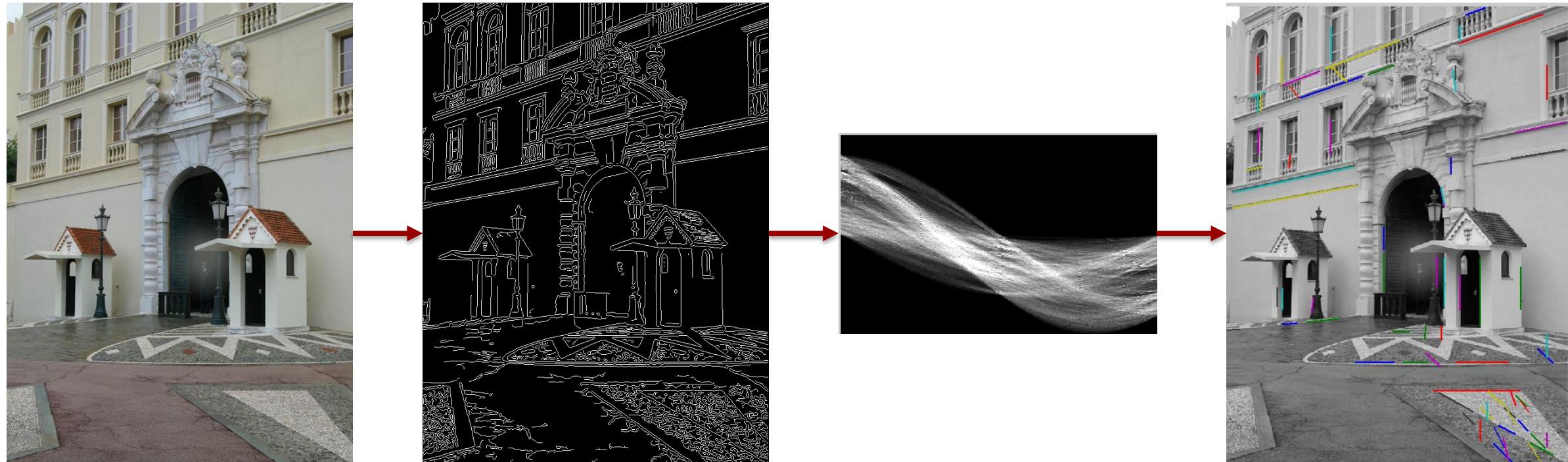


input image



# Hough Transform (4/4)

- Identify maxima and track lines back to image:



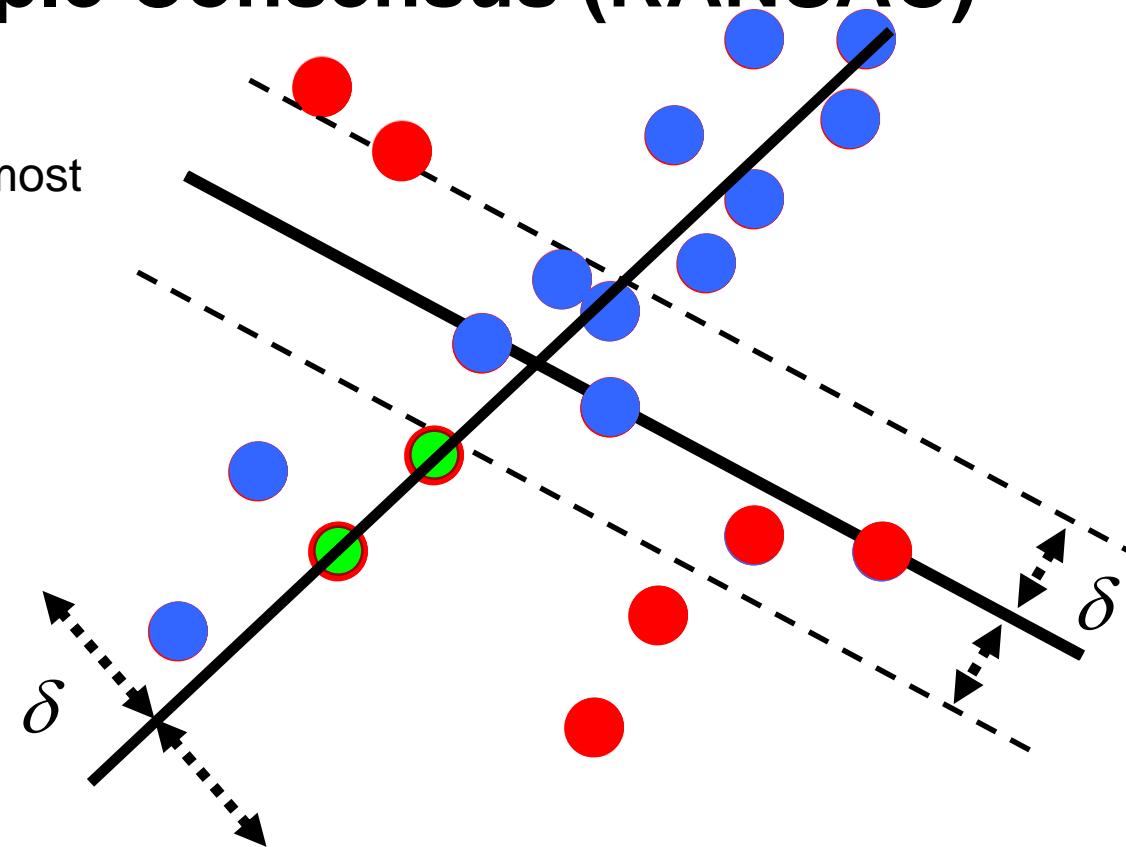
# RANdom SAmple Consensus (RANSAC)

## 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
  - $\delta$ - 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.

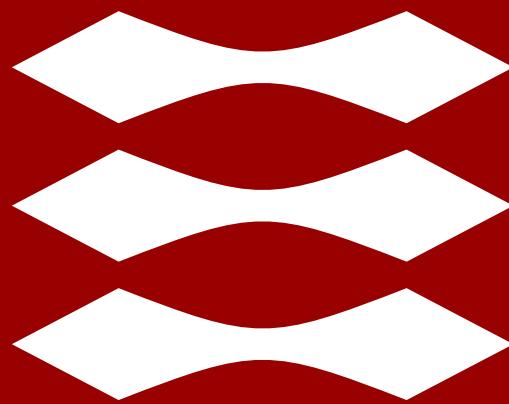


Perception for Autonomous Systems 31392:

# Fitting

Lecturer: Evangelos Boukas—PhD

**DTU**



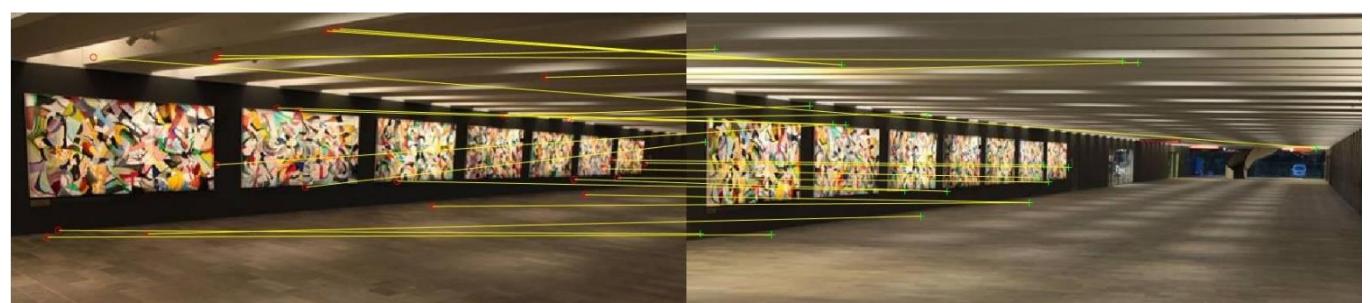
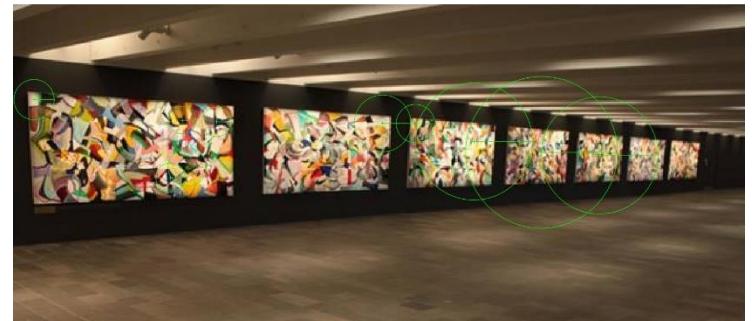
Perception for Autonomous Systems 31392:

# Image Feature Detection and Description

Lecturer: Evangelos Boukas—PhD

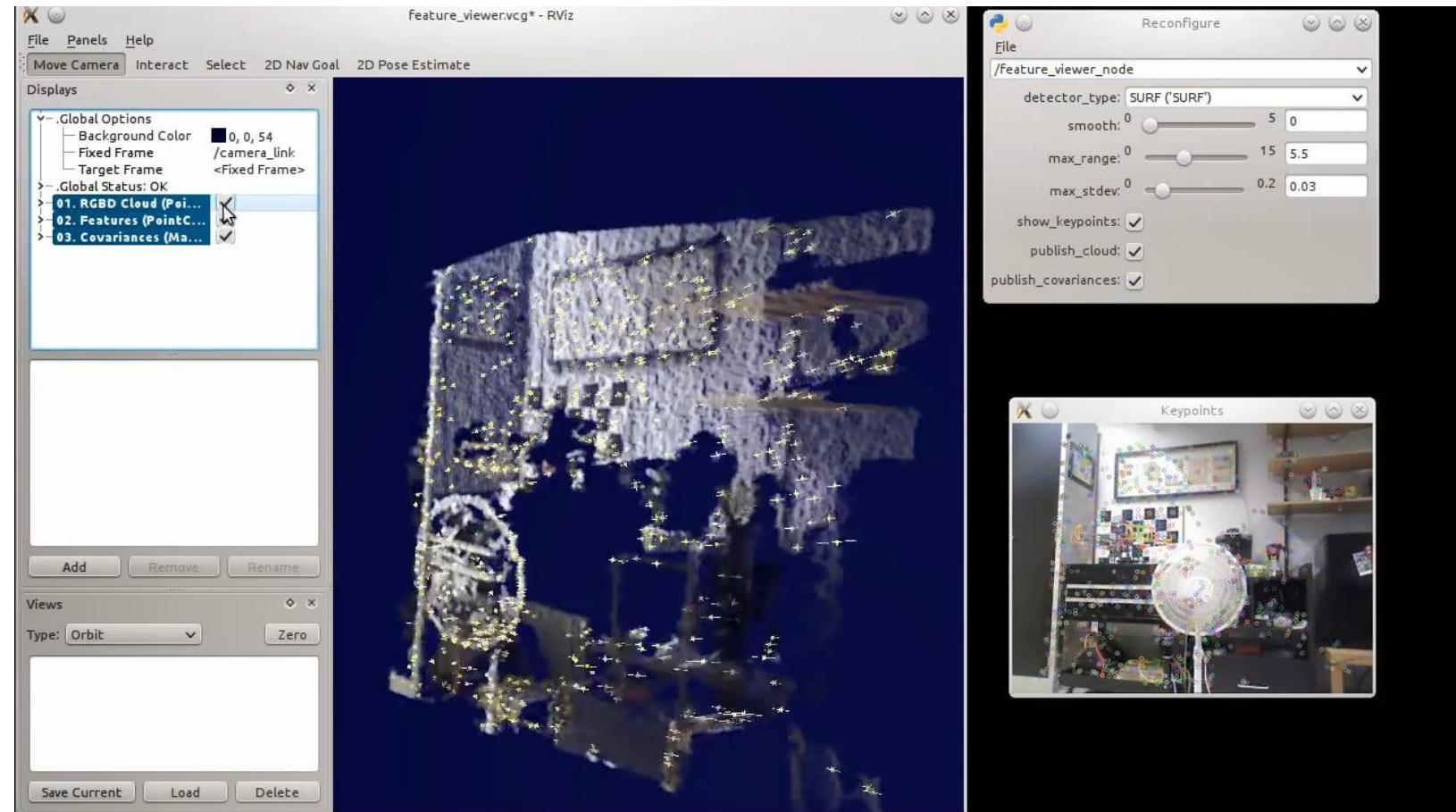
# Image Features

- 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



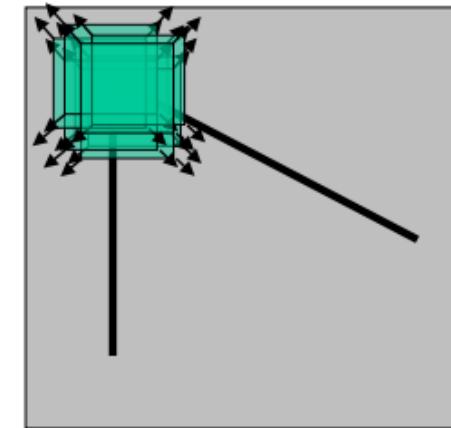
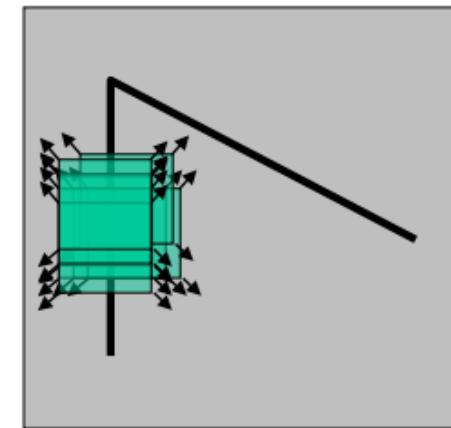
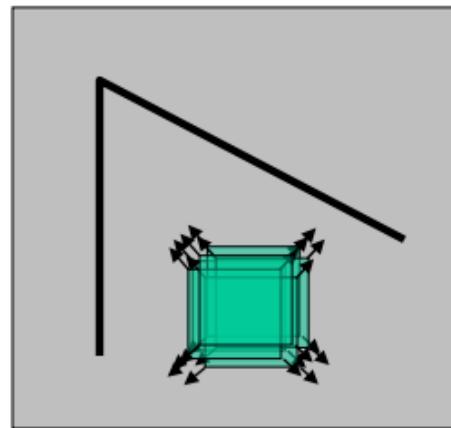
# Image Features

- A quick view



# Feature Detection: Harris corner detector

- Corners are great for features



- Lines not so, why

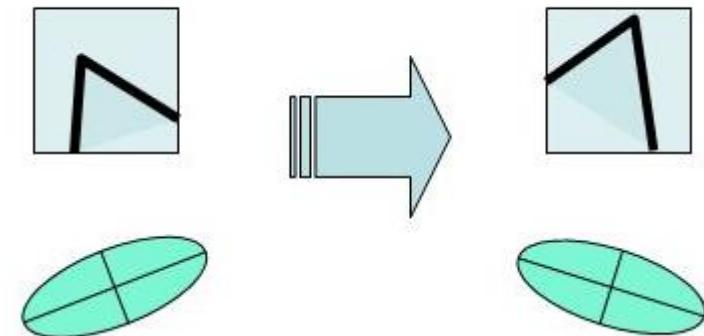
“flat” region:  
no change in  
all directions

“edge”:  
no change along  
the edge direction

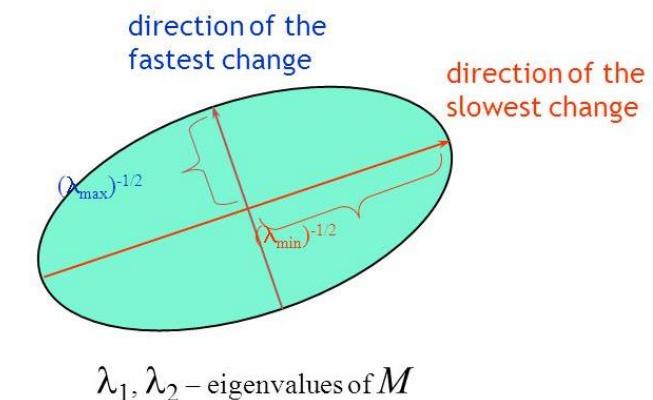
“corner”:  
significant change  
in all directions

# Feature Detection: Harris corner detector

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

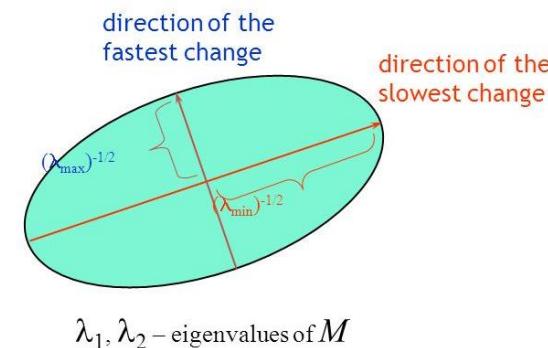


- Notice how by rotating it does not change
- Linear algebra



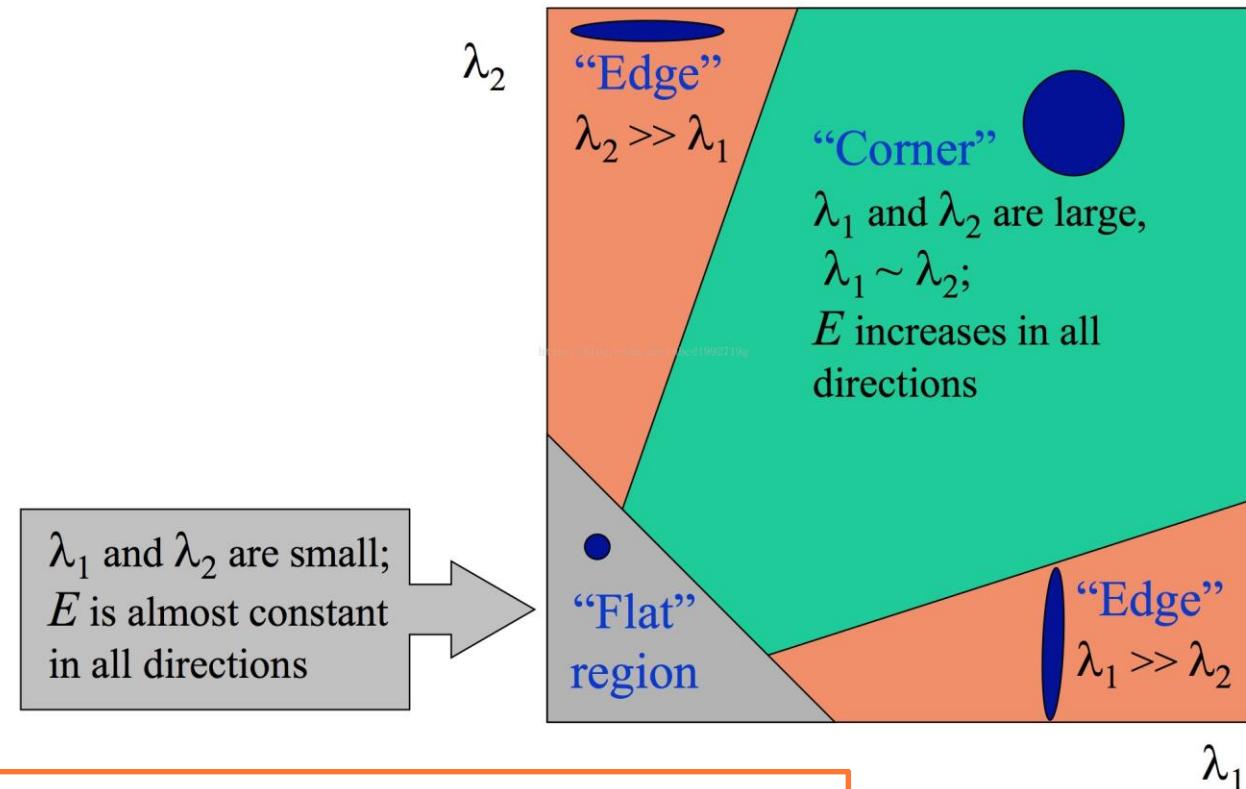
# Feature Detection: Harris corner detector

- So how are these eigenvalues useful?



# Feature Detection: Harris corner detector

- So how are these eigenvalues useful?

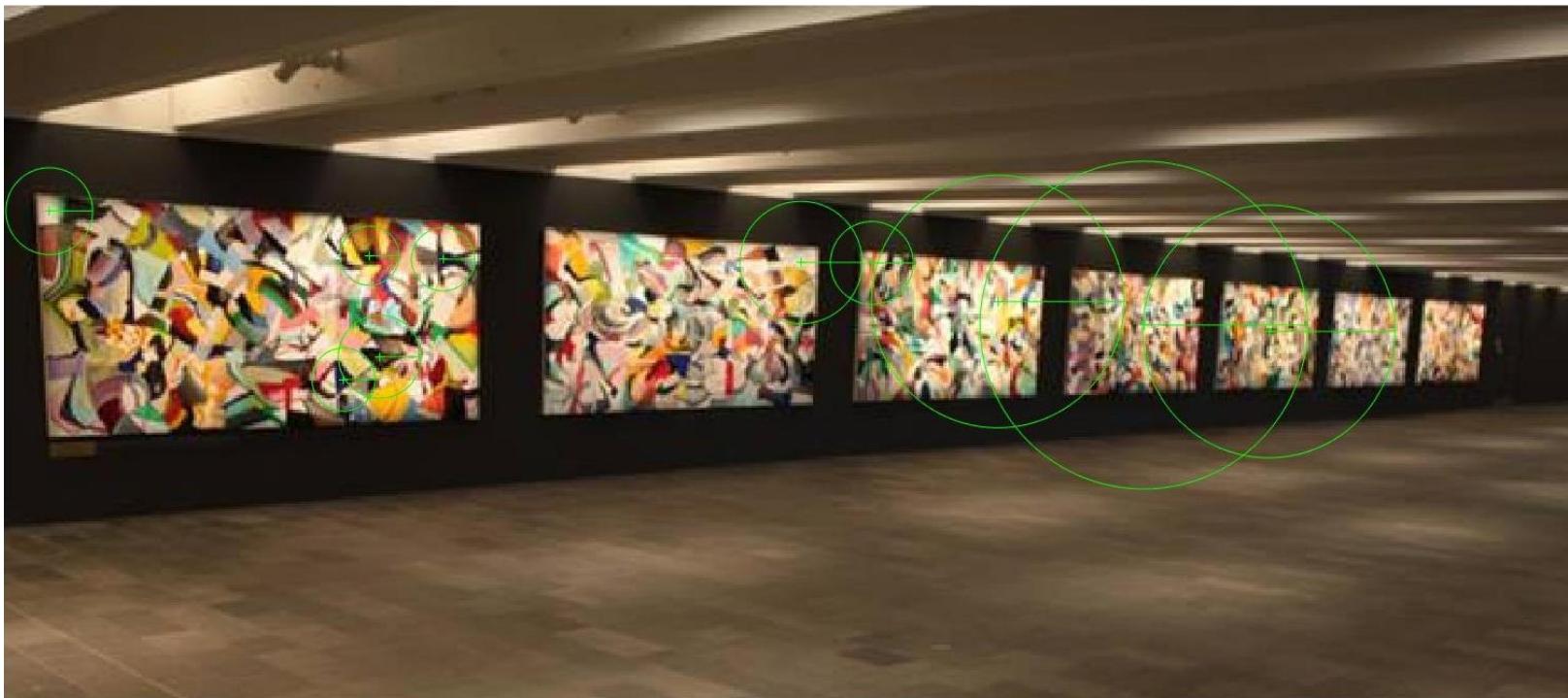


Mikolajczyk, K., and Schmid, C., "A performance evaluation of local descriptors".

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?

 $\lambda_1$ 

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

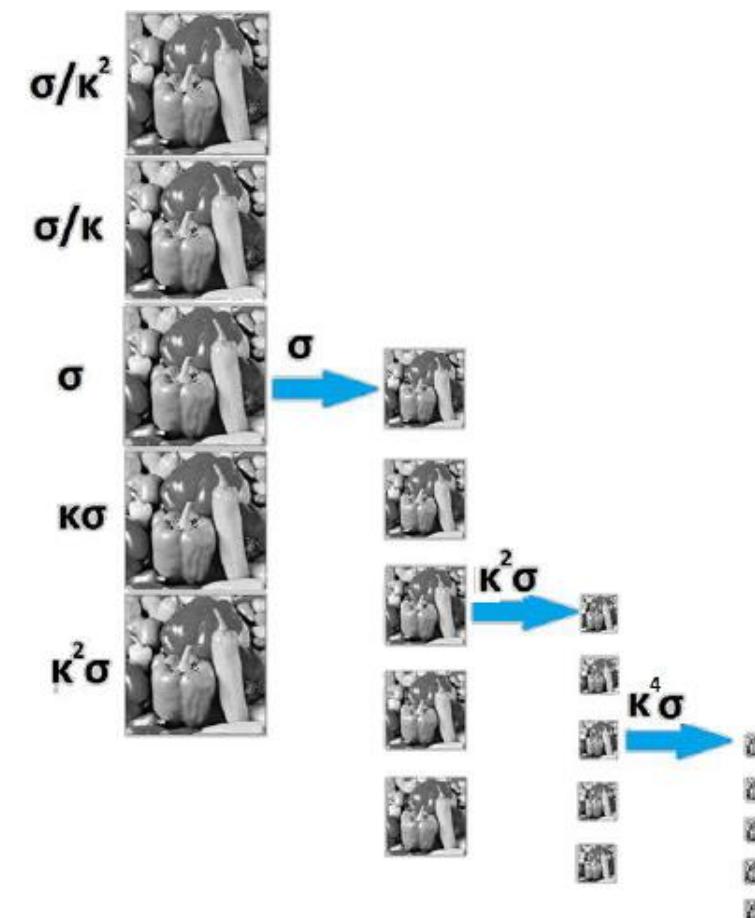
- So we can find corners.
- But how descriptive are these corners?
  - Not really,
  - Think that the roof of a building has corners
  - And your desk has corners...
- Finally a revelation:

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.

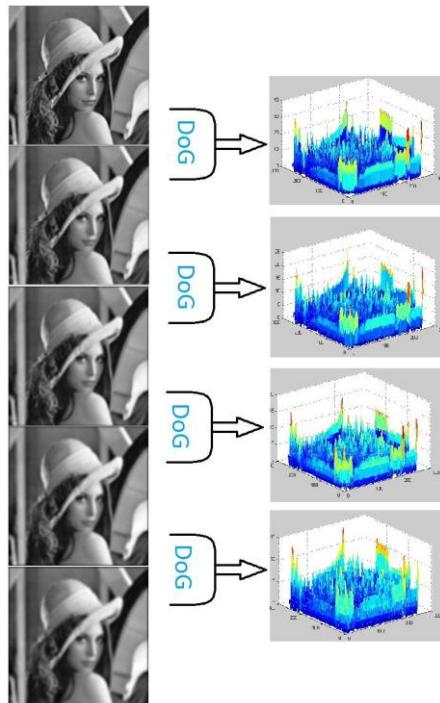
# Feature Detection: Scale Space Theory

- 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:



# 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

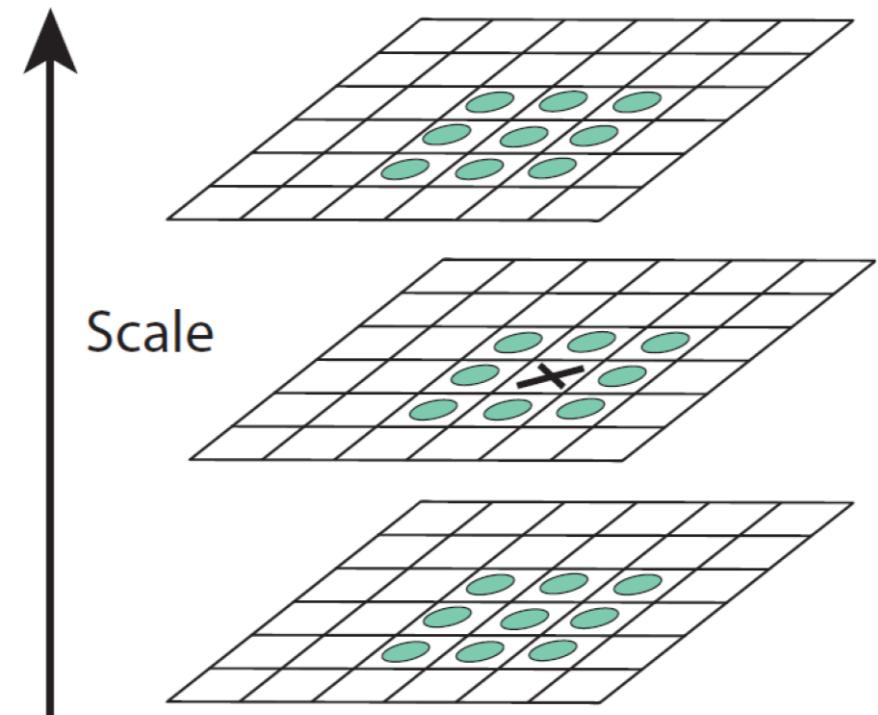
Main Feature is that it is robust in:

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

# Scale Invariant Feature Transform - SIFT

## Detection

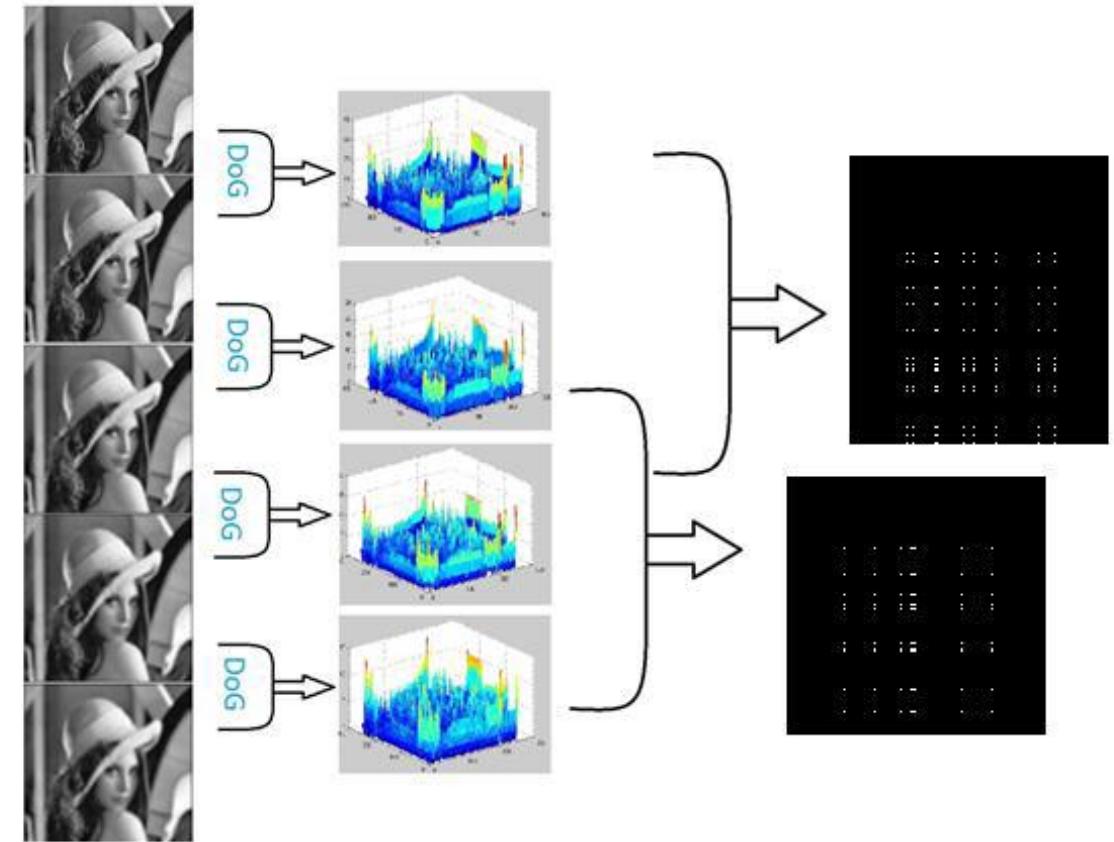
- Use DoG on Scale Space
  - Only the maximum or minimum in a neighborhood are considered
  - All octaves are investigated



# 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

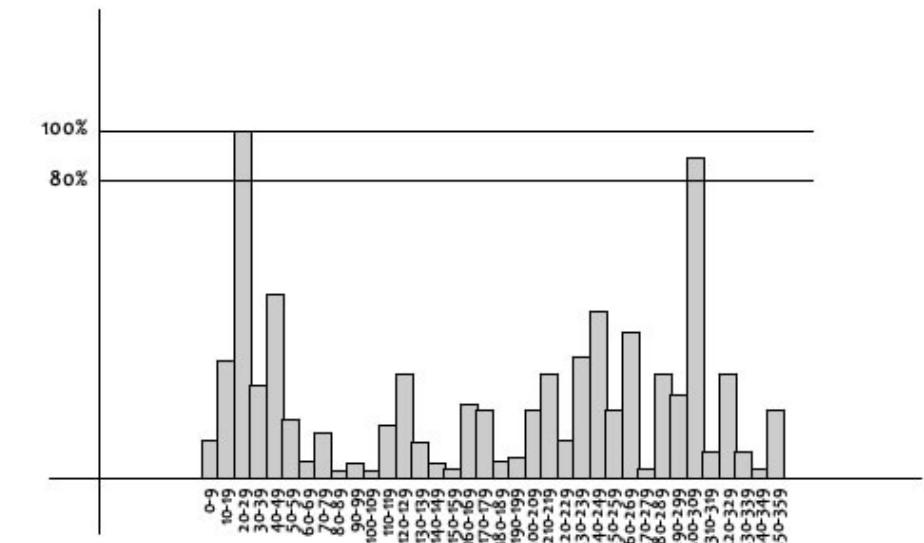
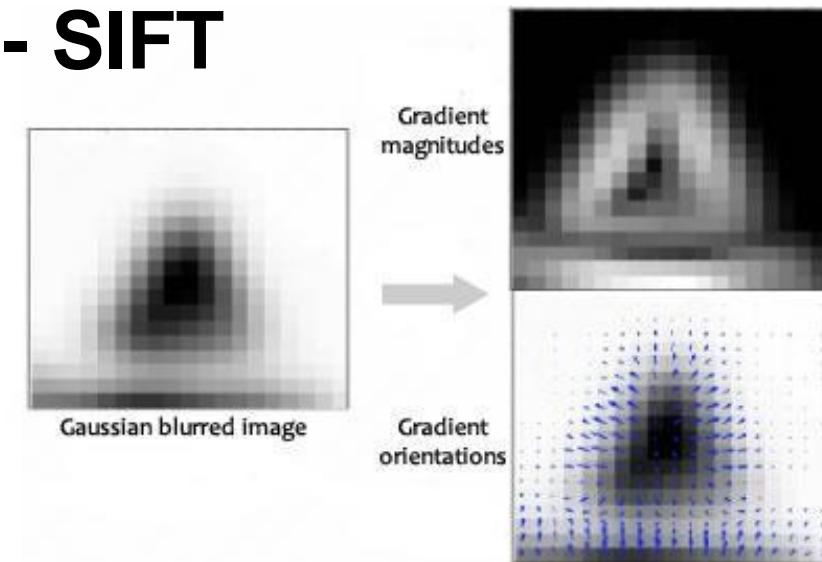
## Description

- Orientation:
  - Based on the gradient

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

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

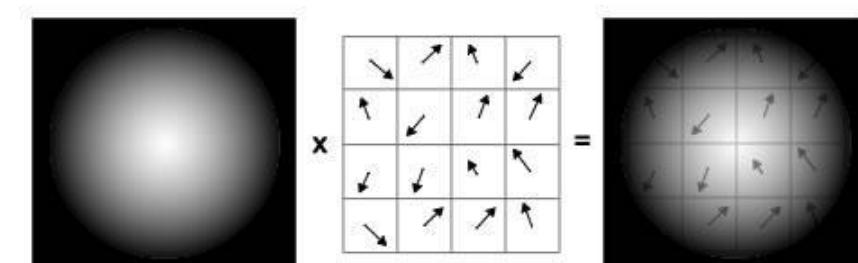
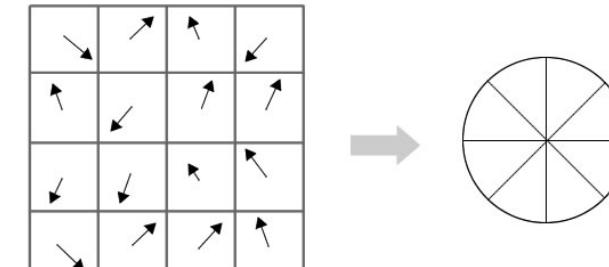
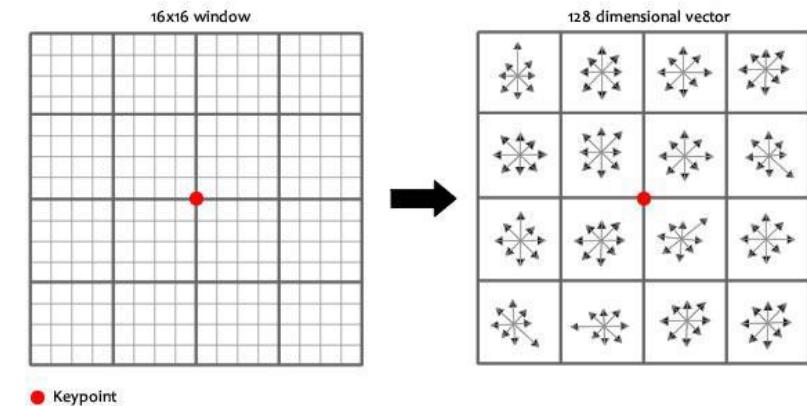
- Create a histogram of orientations  
Consisting of 36 bins (every 10 degrees)
- Fingerprint



# Scale Invariant Feature Transform - SIFT

## Description

- Orientation
- Fingerprint:
  - Assume a  $16 \times 16$  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:

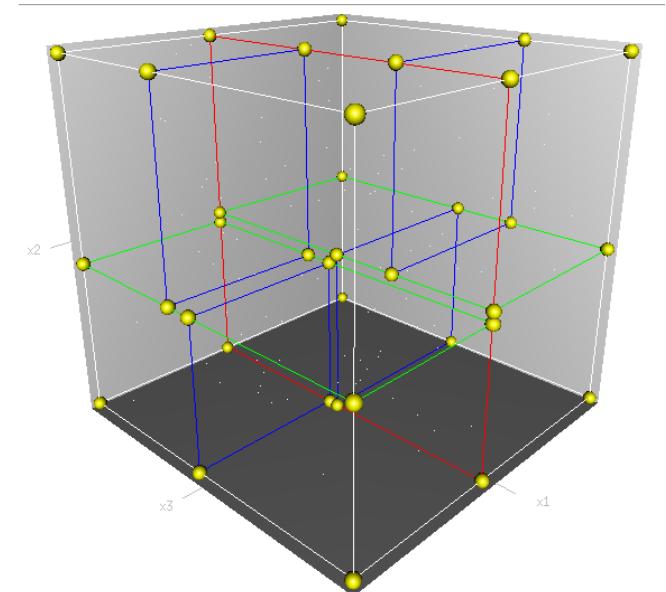
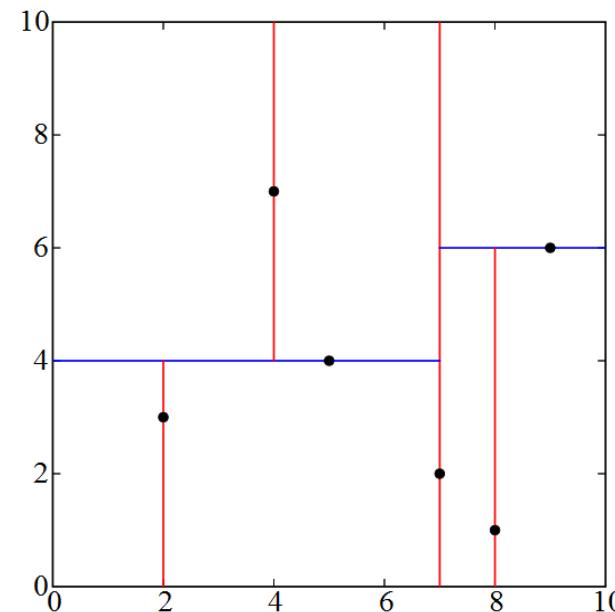
$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2}$$

- 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



# Feature points and areas

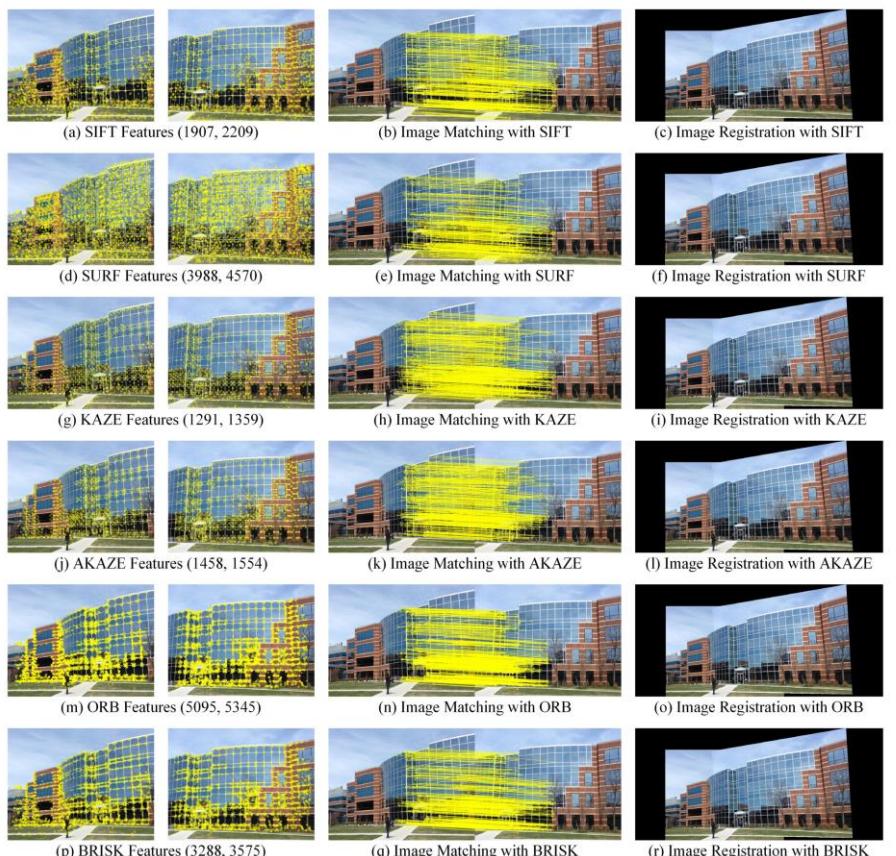
- A multitude of features have been proposed (specific for each application):
  - SIFT
  - SURF
  - BRISK
  - FREAK
  - MSER
  - ORB
  - ...

# 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

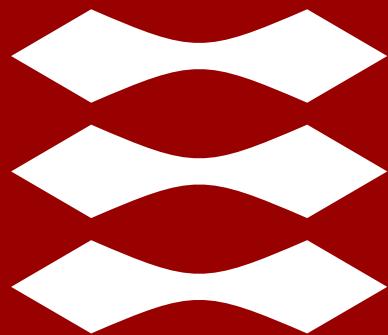
- What did we learn?
  - Is that a good feature?
  - How to get scale and rotational invariance in the features we get?
  - How to detect points of interest?
  - How to describe the feature of points so,
  - We can match them across multiple images.
  - How can we use features?

Perception for Autonomous Systems 31392:

# Image Feature Detection and Description

Lecturer: Evangelos Boukas—PhD

**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?

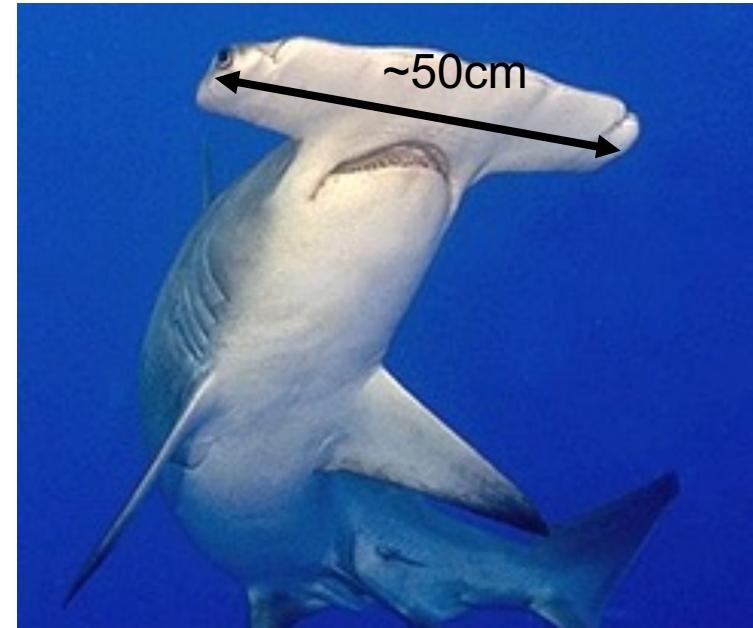
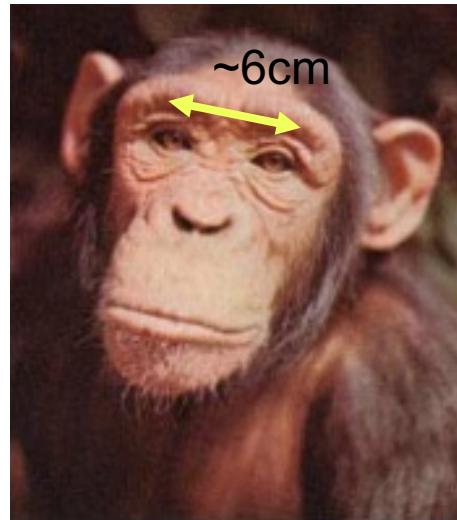


# What is Stereo Vision?

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



# What is Stereo Vision?

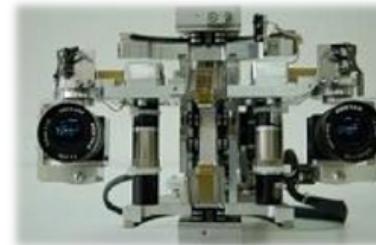
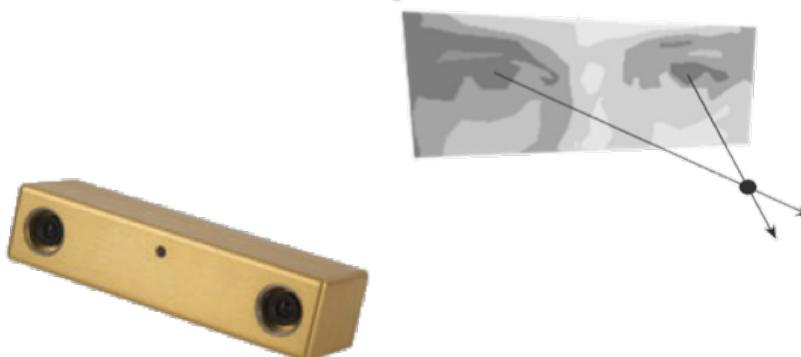


# What is Stereo Vision?

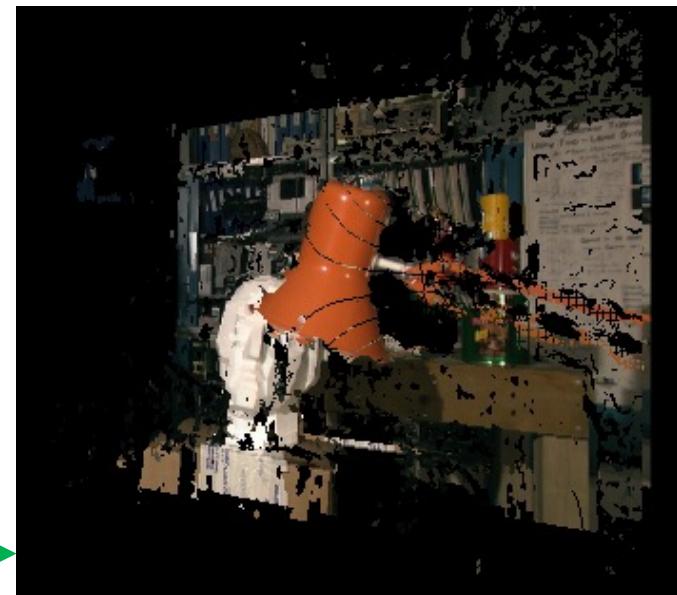
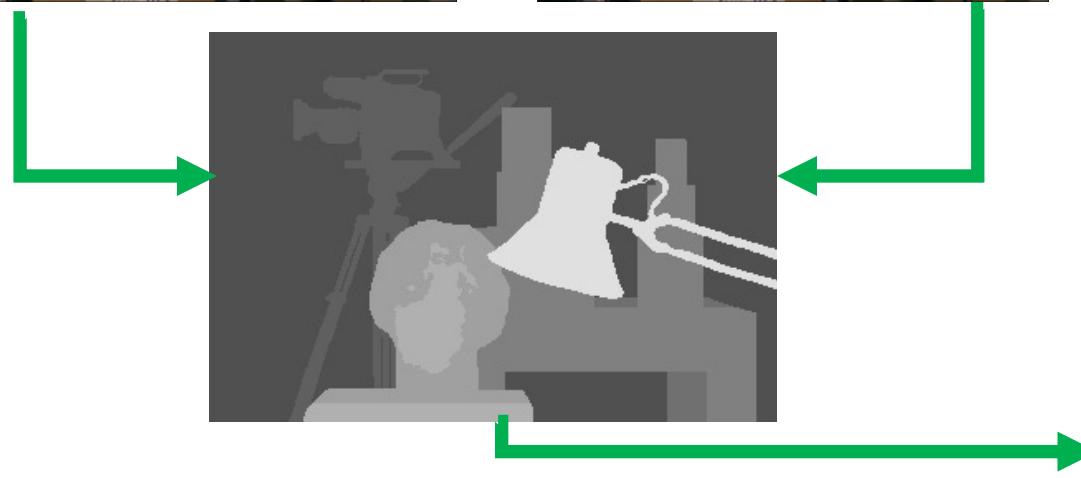


”... the mind perceives an object of three-dimensions by means of the two dissimilar pictures projected by it on the two retinae...”

*Sir Charles Wheatstone, 1838*

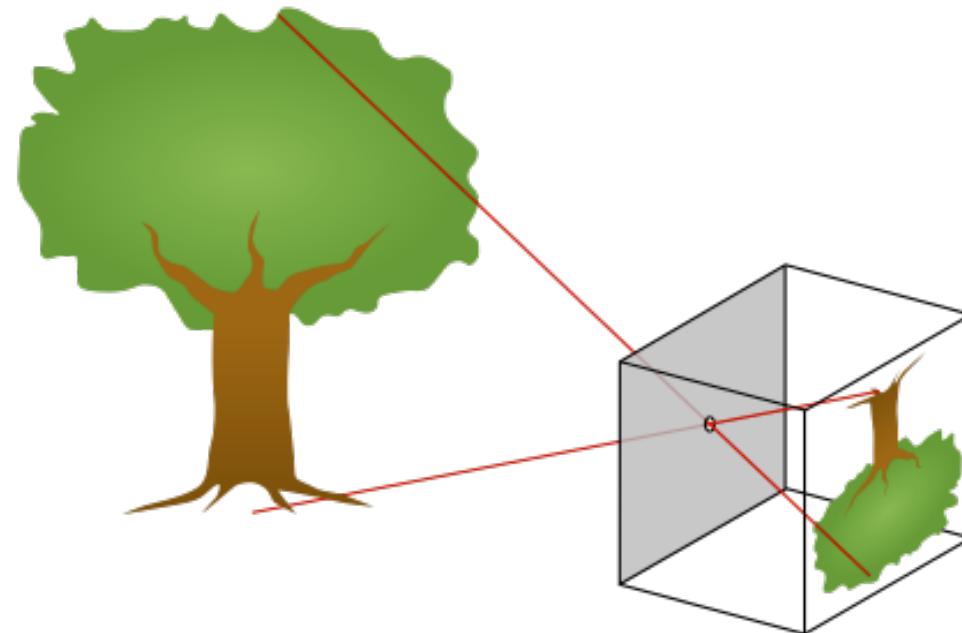


# Stereo Vision Computation



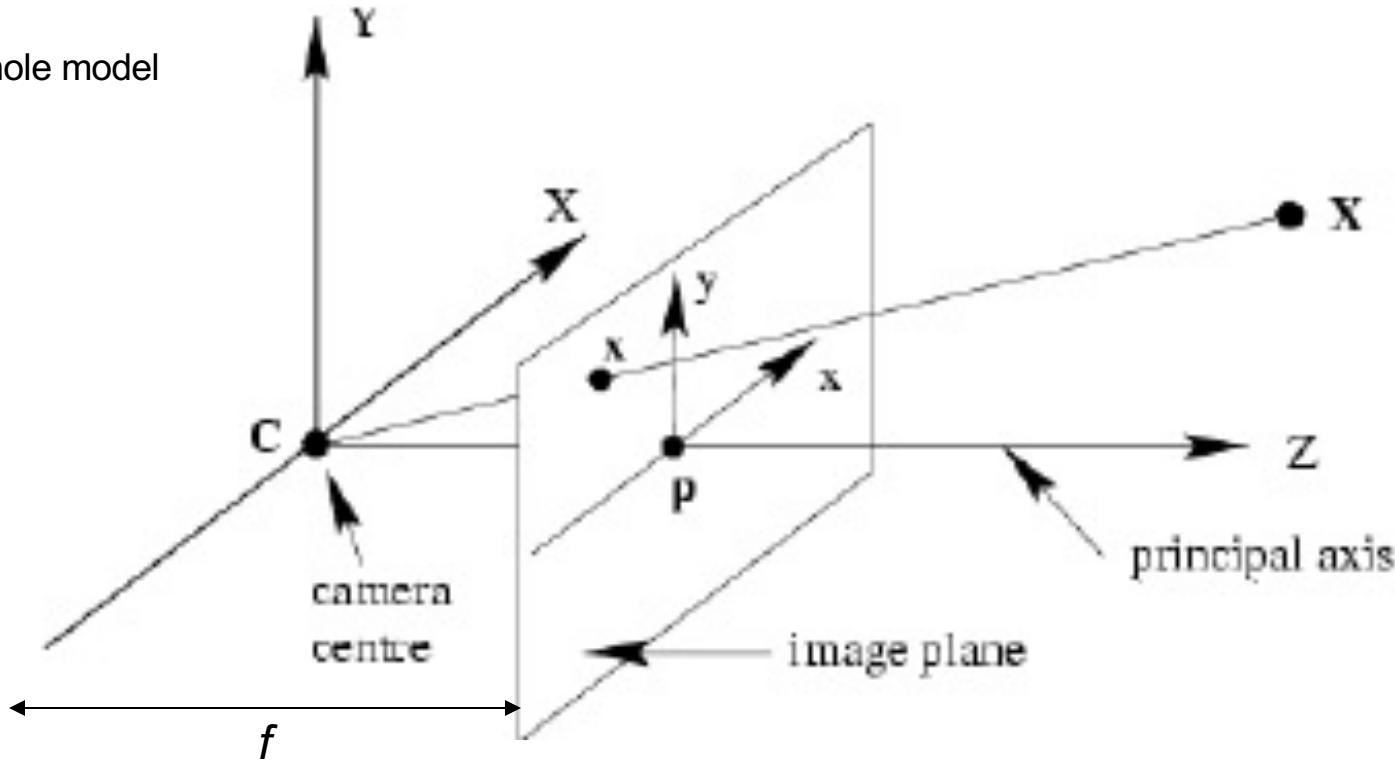
- What is Stereo Vision?
- Stereo/Epipolar Geometry
- Rectified Stereo Case
- Depth from Stereo Matches
- Correspondence Problem
  - **Dense** vs Sparse Correspondence
  - **Local** vs Global Correspondence
  - (Dis)-Similarity Measures
- Summary

- Pinhole model

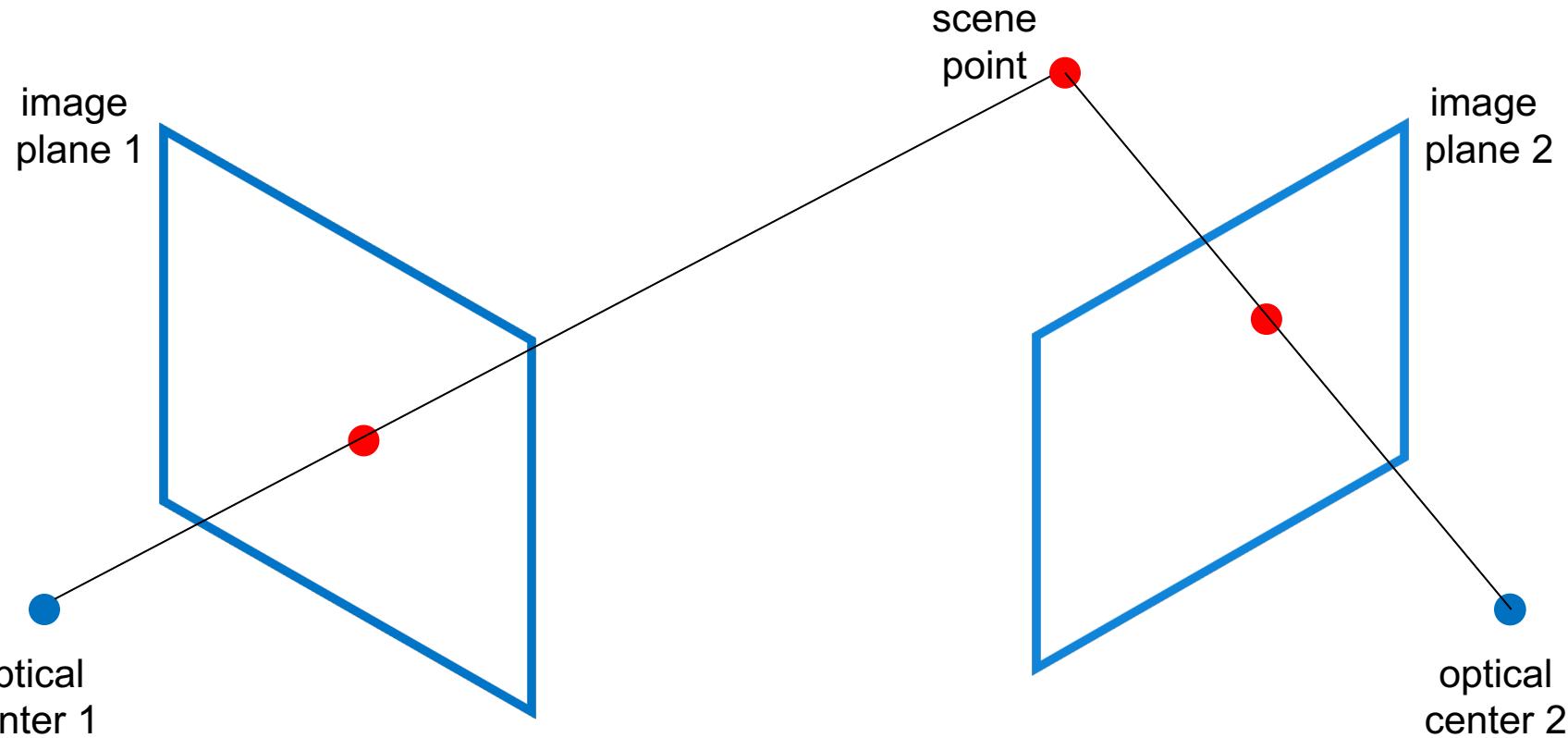


# Camera Model

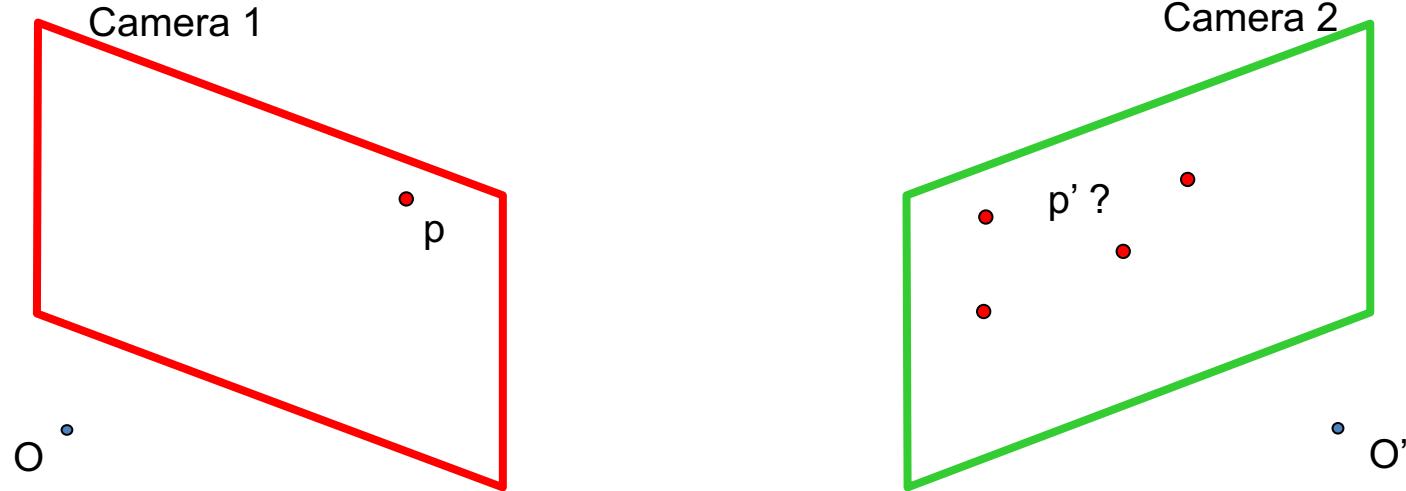
- Pinhole model



# Epipolar Geometry

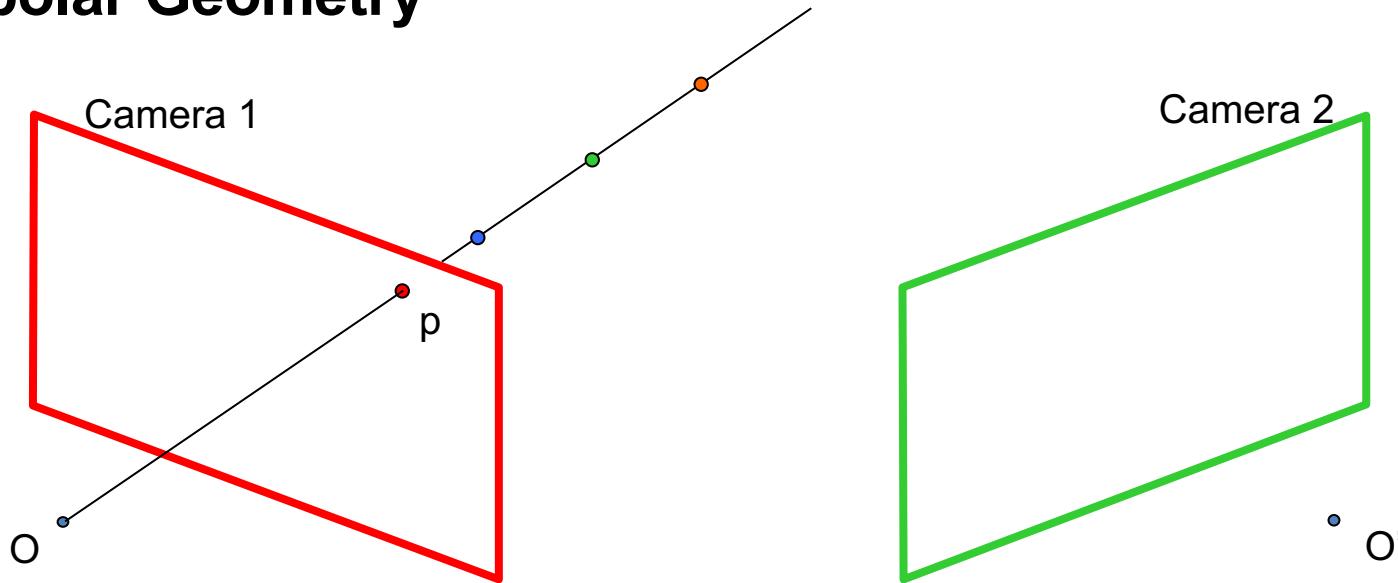


# Epipolar Geometry



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

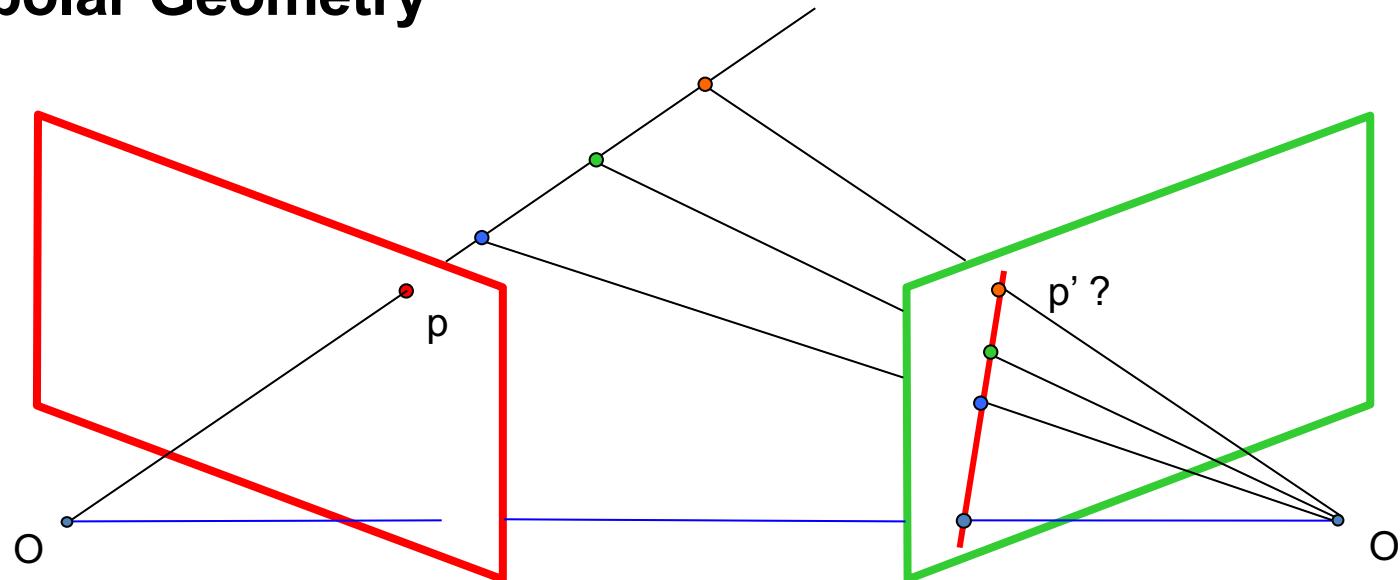
# Epipolar Geometry



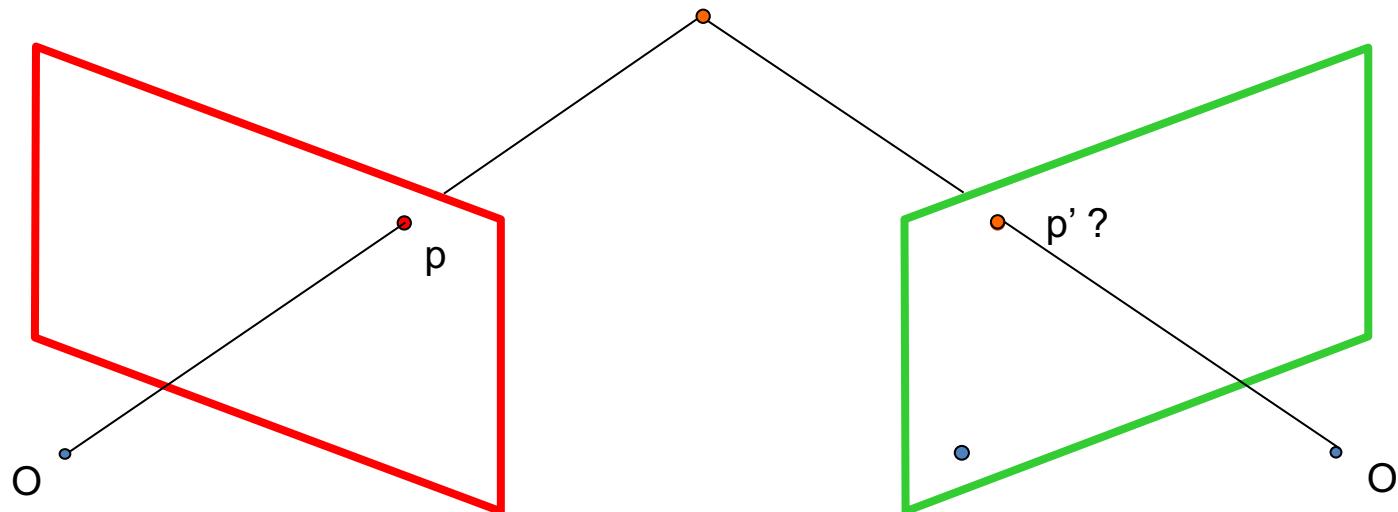
If we see a point in camera 1, are there any constraints on where we will find it on camera 2?

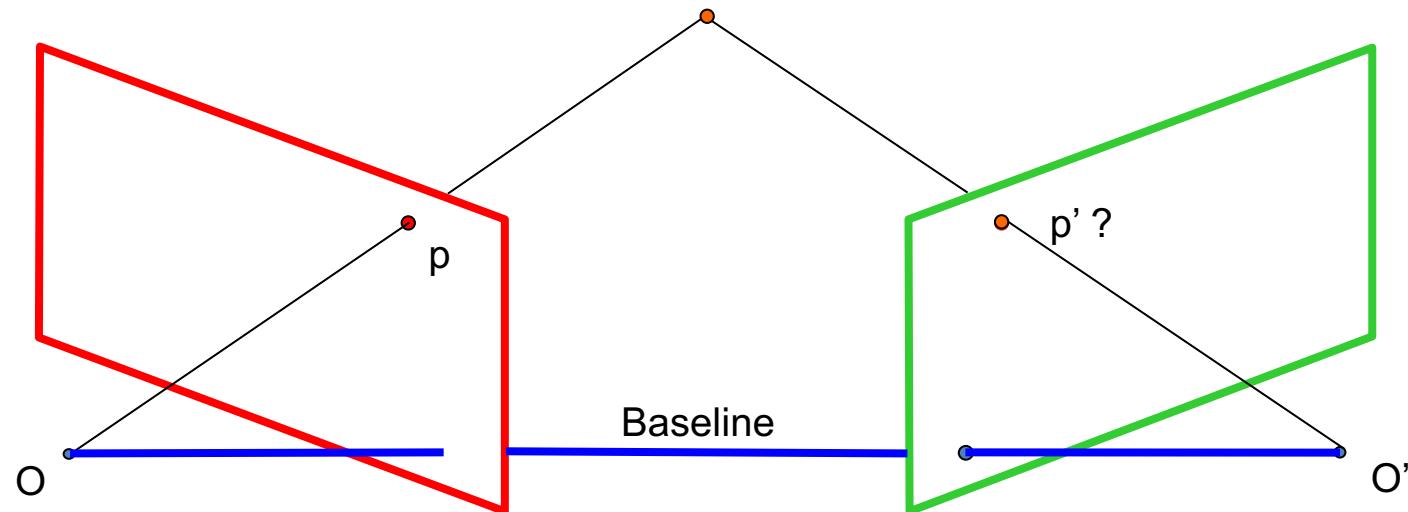


# Epipolar Geometry



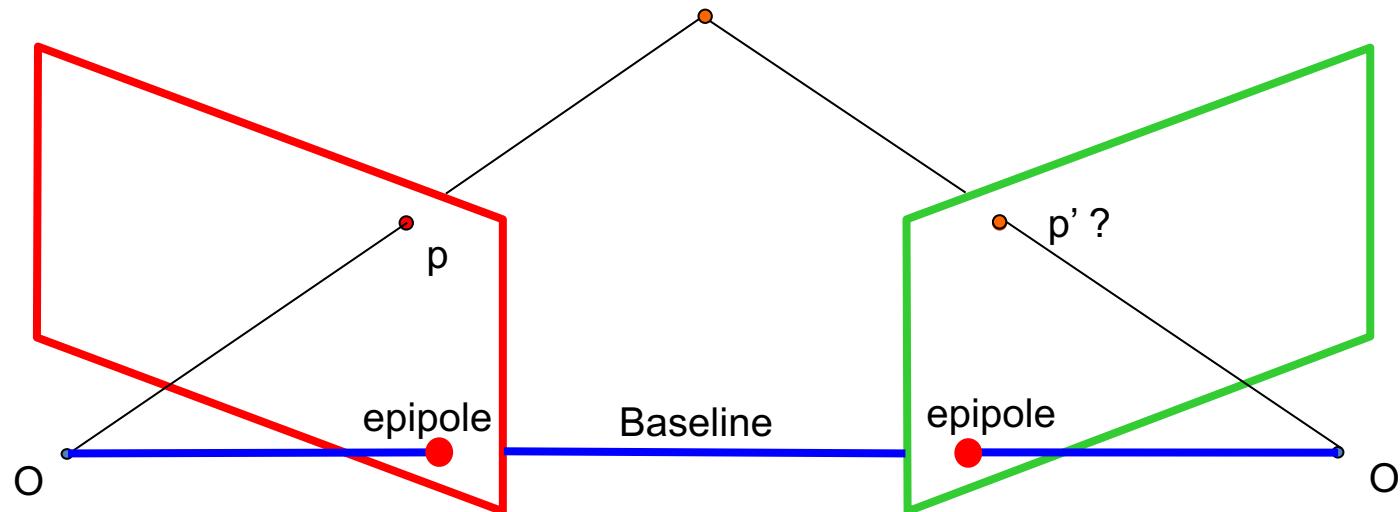
# Epipolar Geometry





**Baseline:** the line connecting the two camera centers

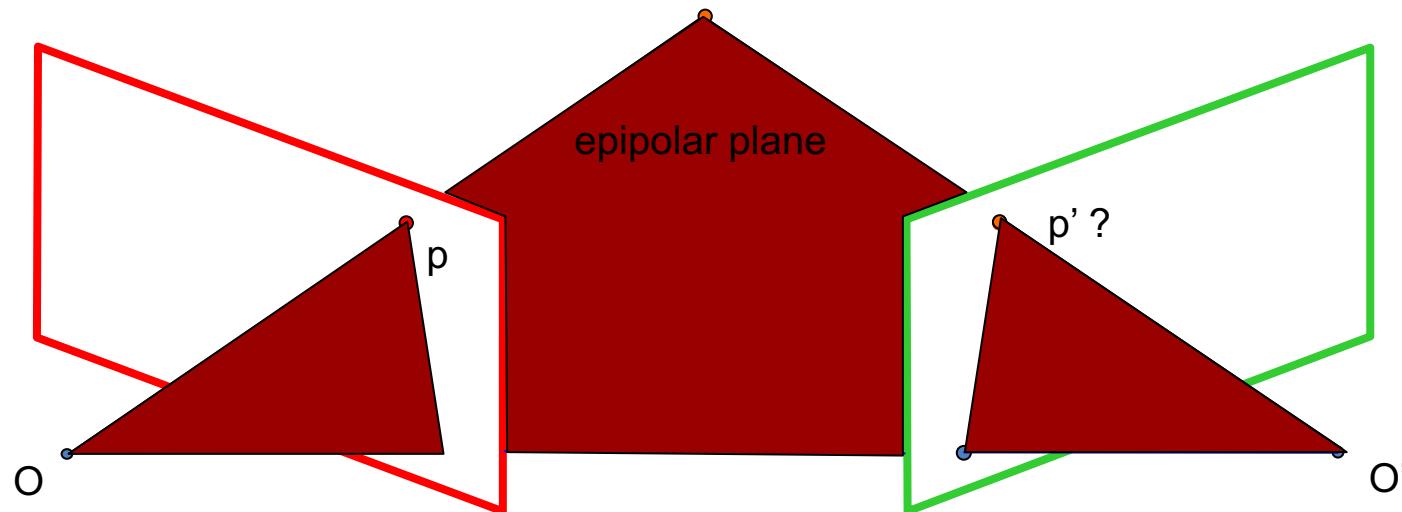
# Epipolar Geometry



**Baseline:** the line connecting the two camera centers

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

# Epipolar Geometry

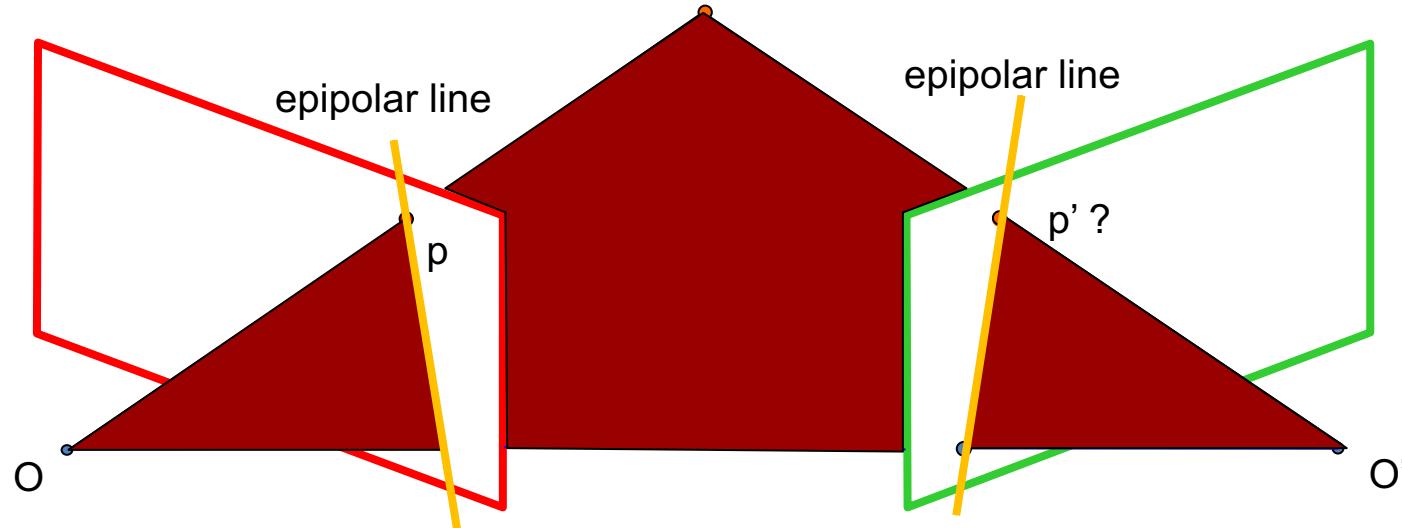


**Baseline:** the line connecting the two camera centers

**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



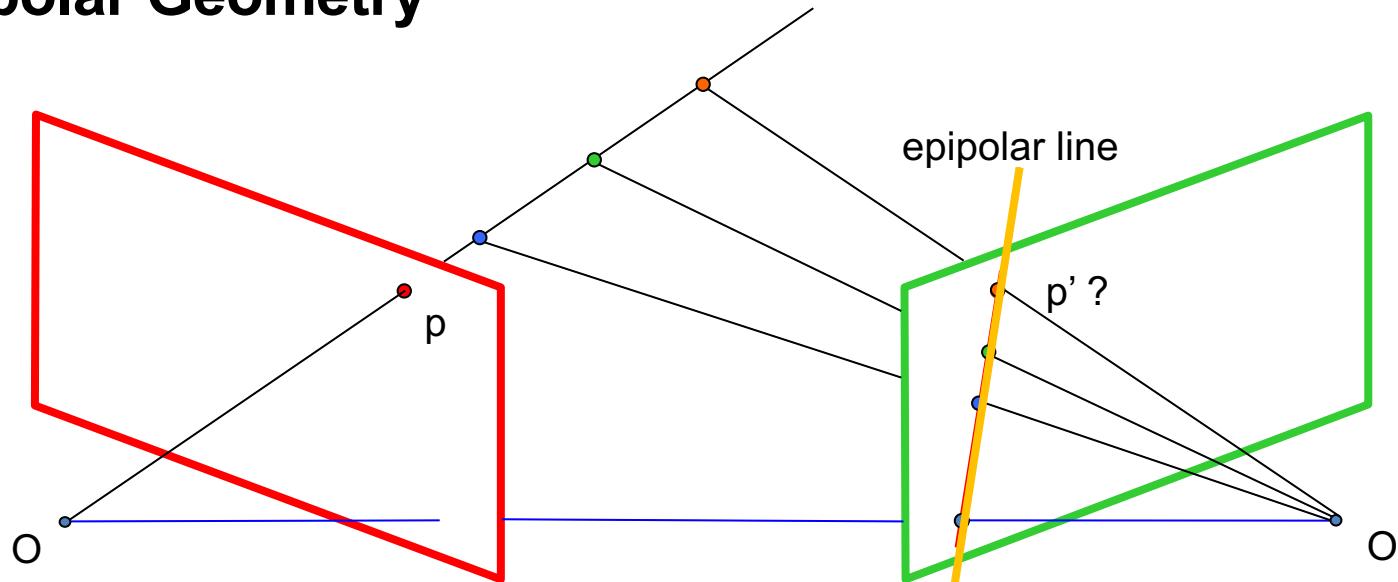
**Baseline:** the line connecting the two camera centers

**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

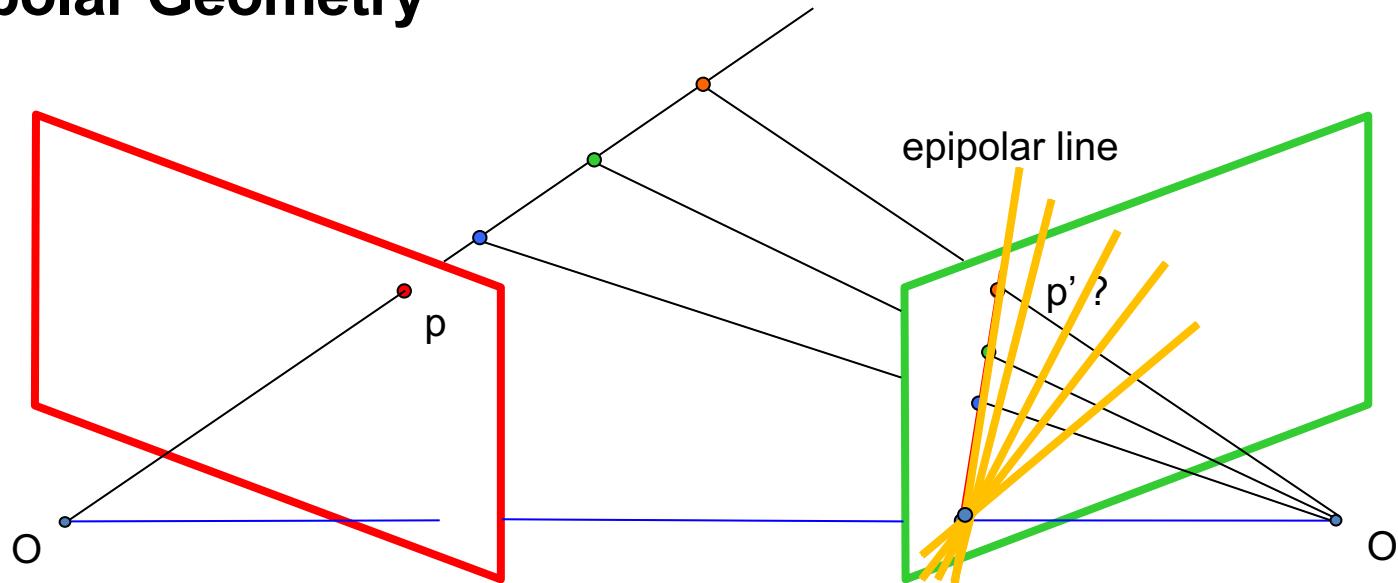
**Epipolar line:** intersection of the *epipolar plane* with each image plane

# Epipolar Geometry

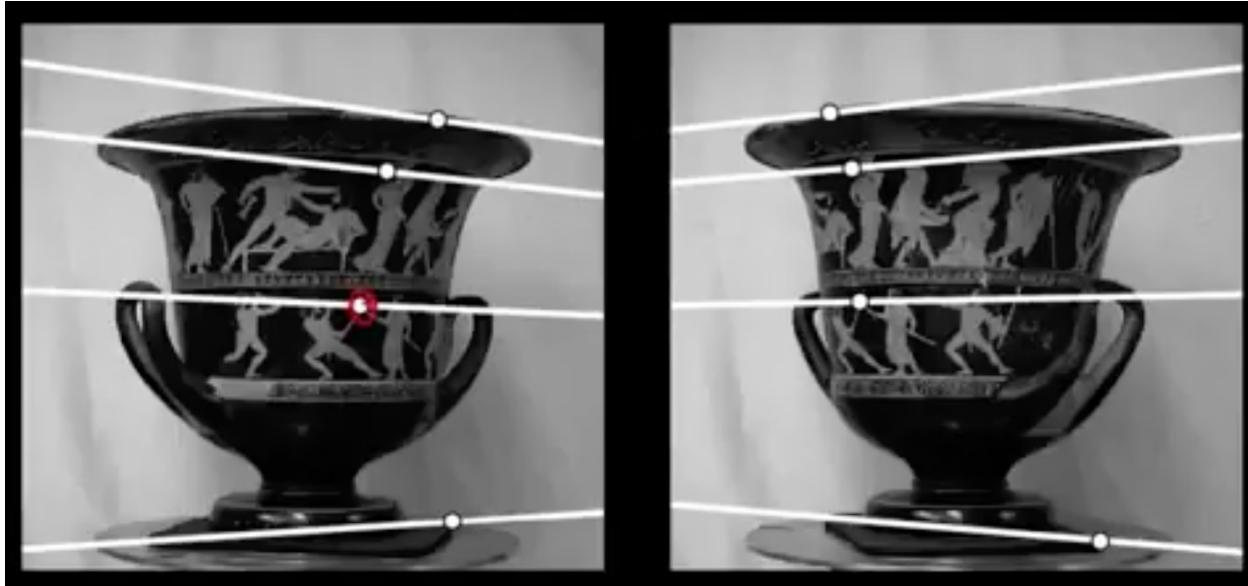


- We can search for matches across epipolar lines.
  - Search space for correspondences reduces to a 1D problem!

# Epipolar Geometry



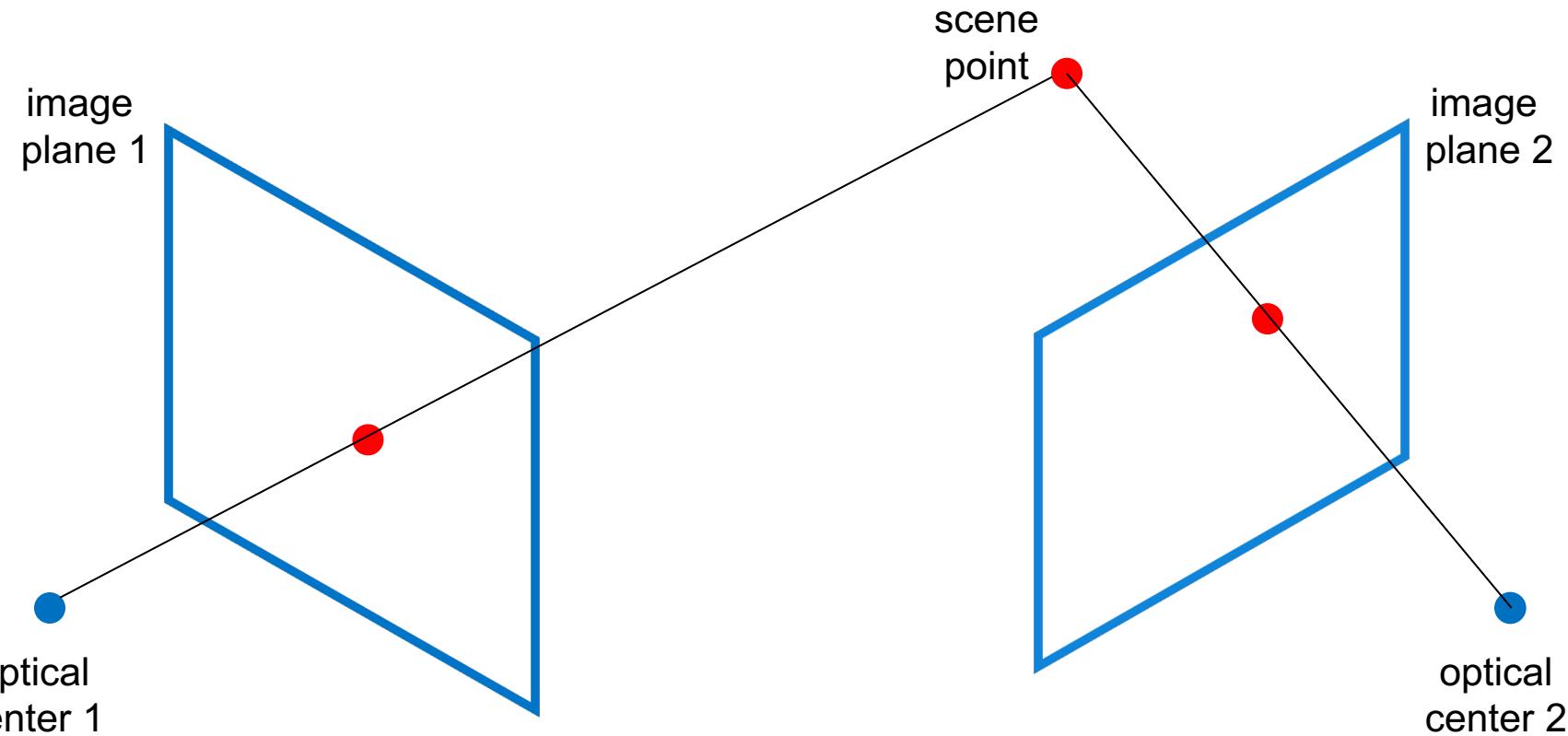
- We can search for matches across epipolar lines
  - Search space for correspondences reduces to a 1D problem!
- All epipolar lines intersect at the epipoles



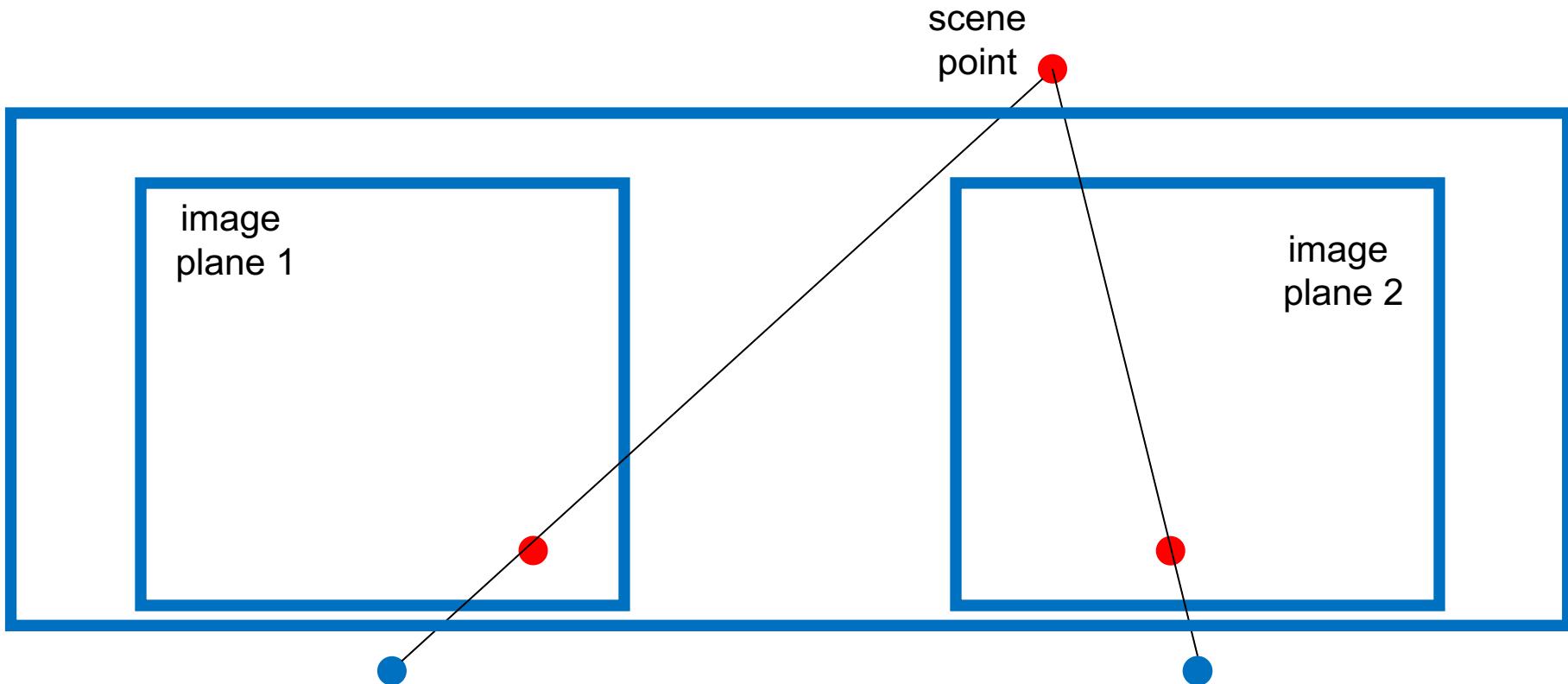
- We can search for matches across epipolar lines
  - Search space for correspondences reduces to a 1D problem!
- All epipolar lines intersect at the epipoles
  - **Where are the epipoles in this case?**

- 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

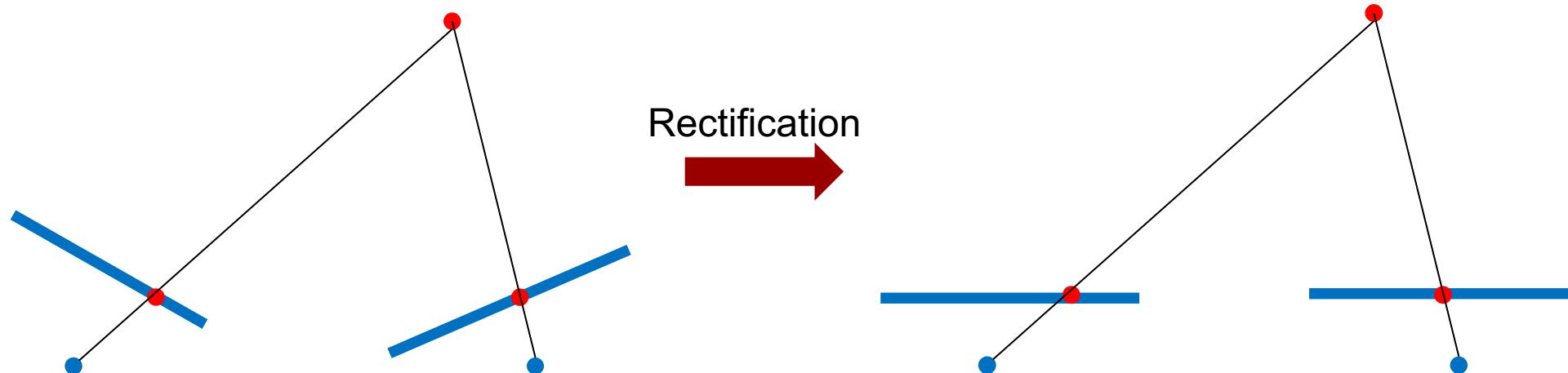


# Rectified Stereo – a simpler case

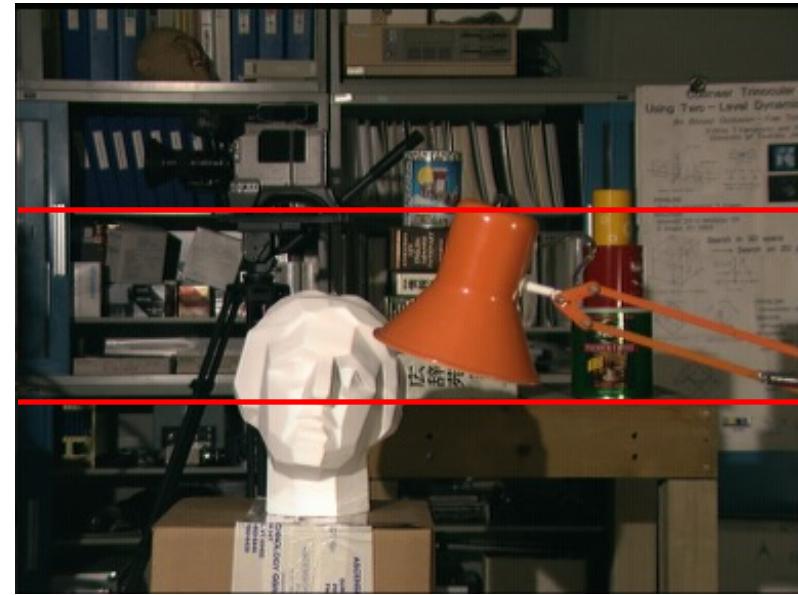
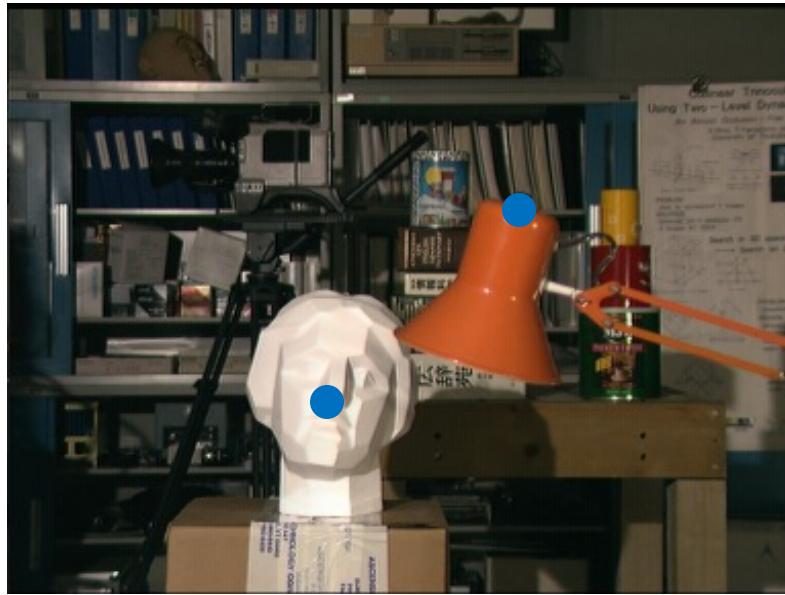


# Rectified Stereo – a simpler case

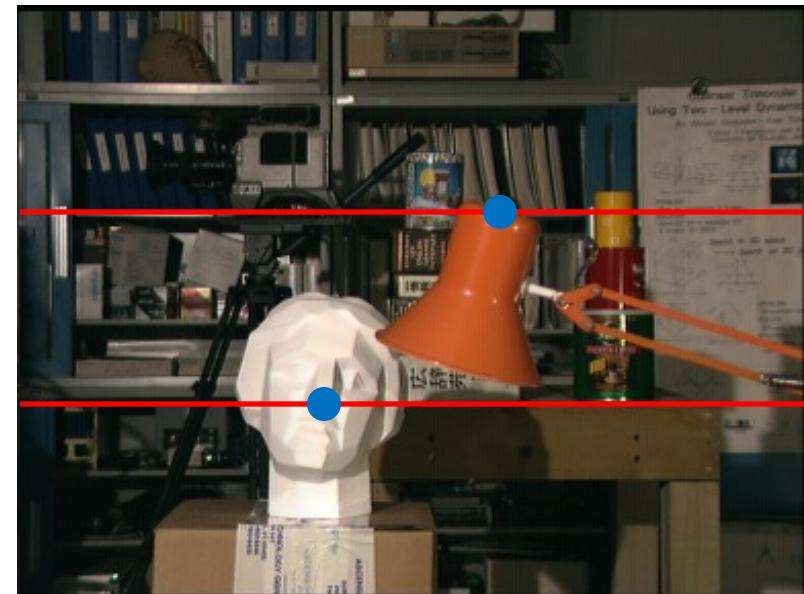
- Rectification:
  - The initial images are reprojected on a common plane that is parallel to the baseline B joining the optical centers of the initial images.
  - Epipolar lines become parallel (and under certain conditions they become also horizontal)



# Rectified Stereo



# Rectified Stereo

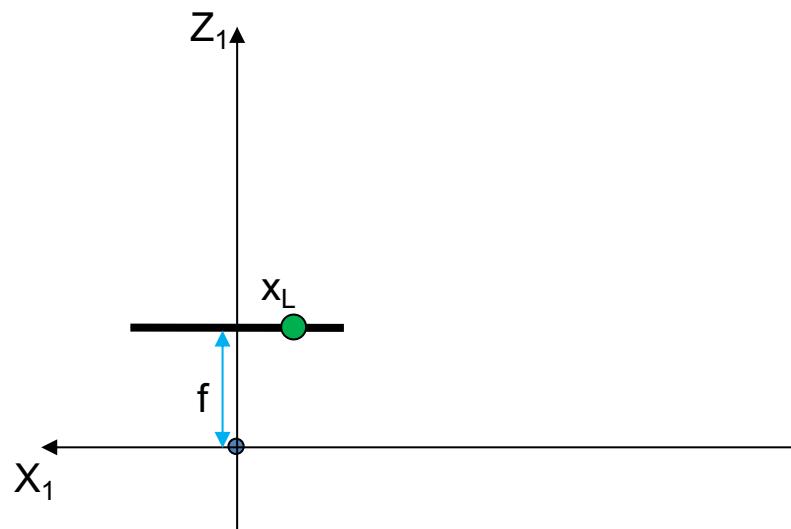


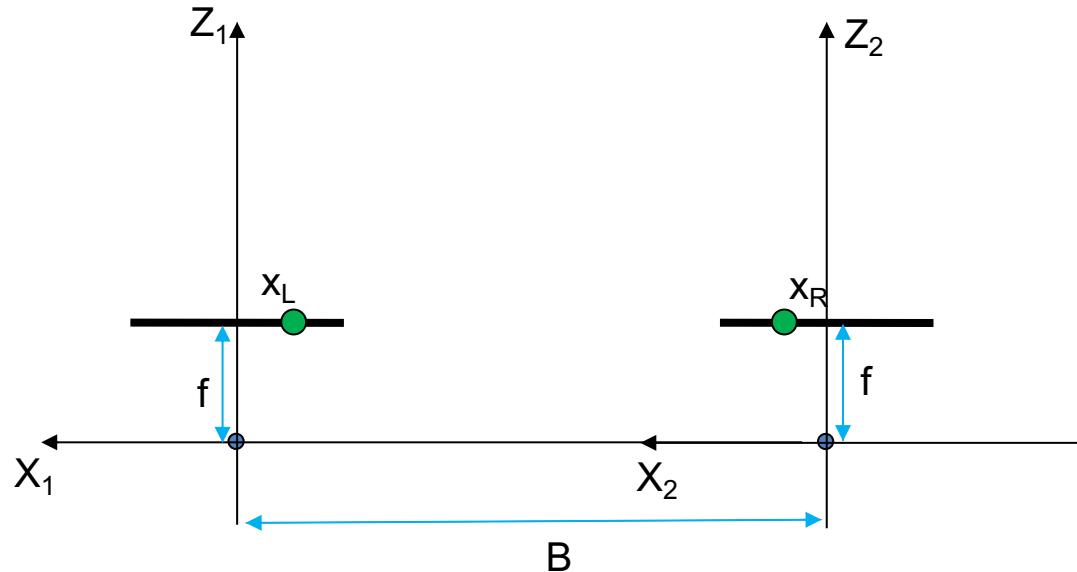
- “All epipolar lines intersect at the epipoles”
  - **Where are the epipoles in this case?**

- 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

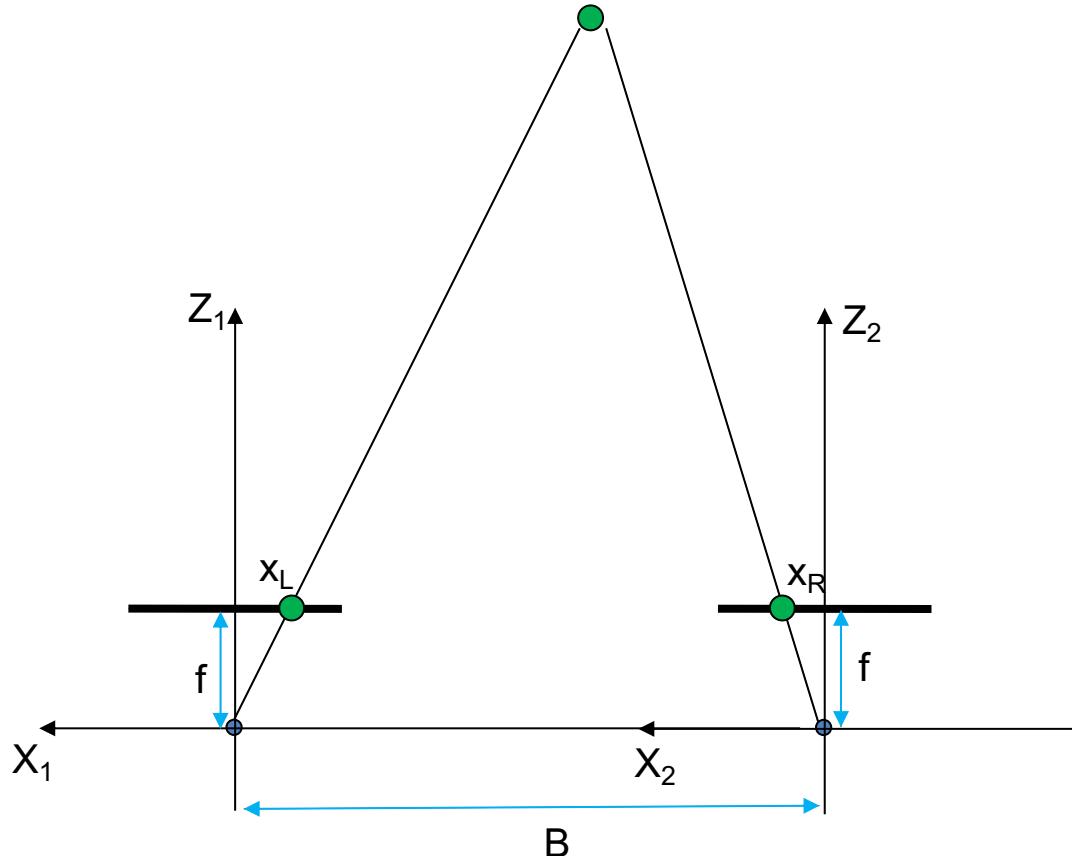
# Depth from Stereo Matches

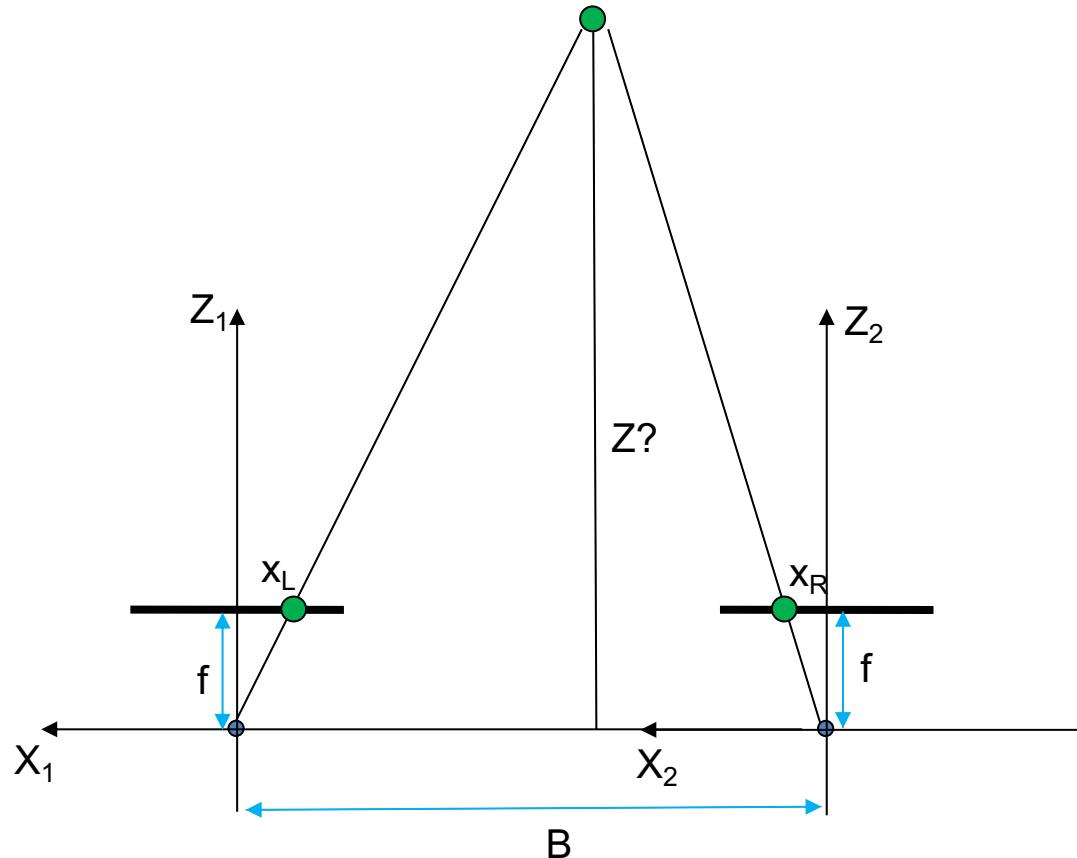
- Let us assume (for now!!) that:
  - we can check the points along the epipolar line and
  - we can find the point (on the right image) that is most similar to our reference point (in the left image), i.e. we can solve the correspondence problem!



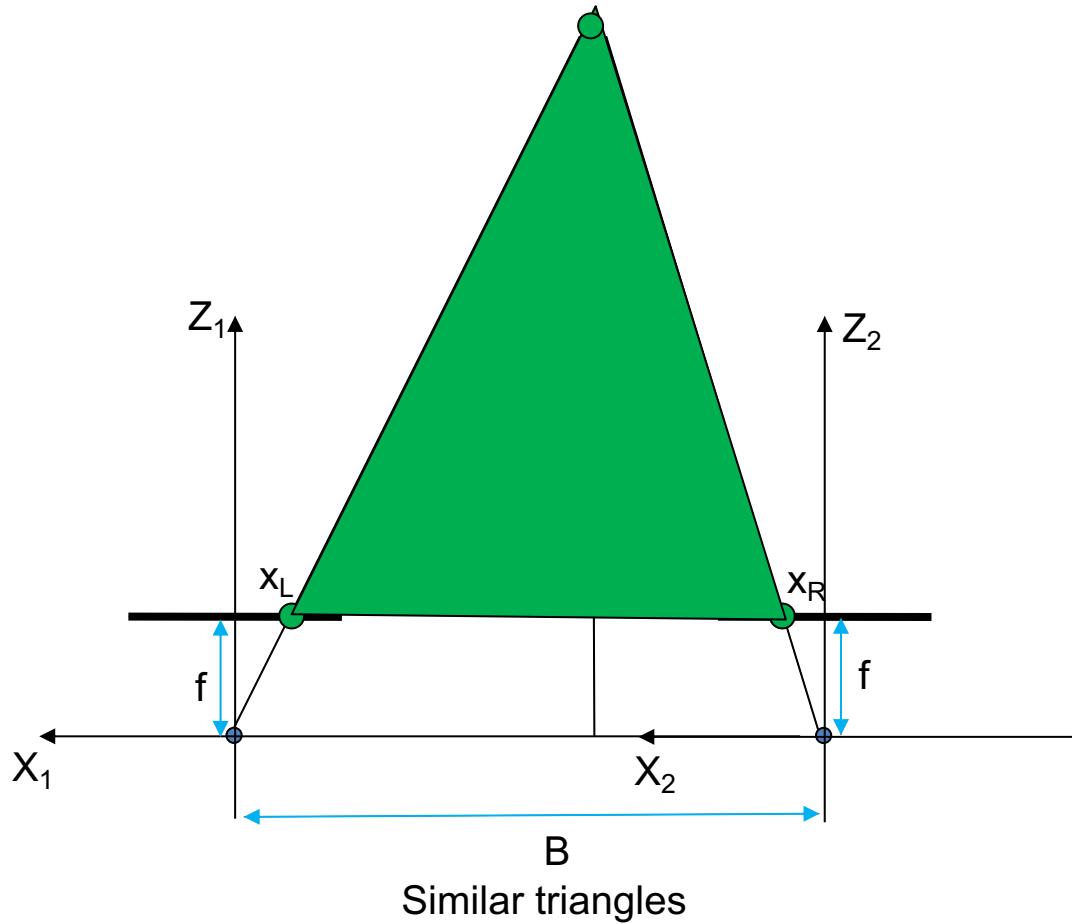


# Depth from Stereo Matches

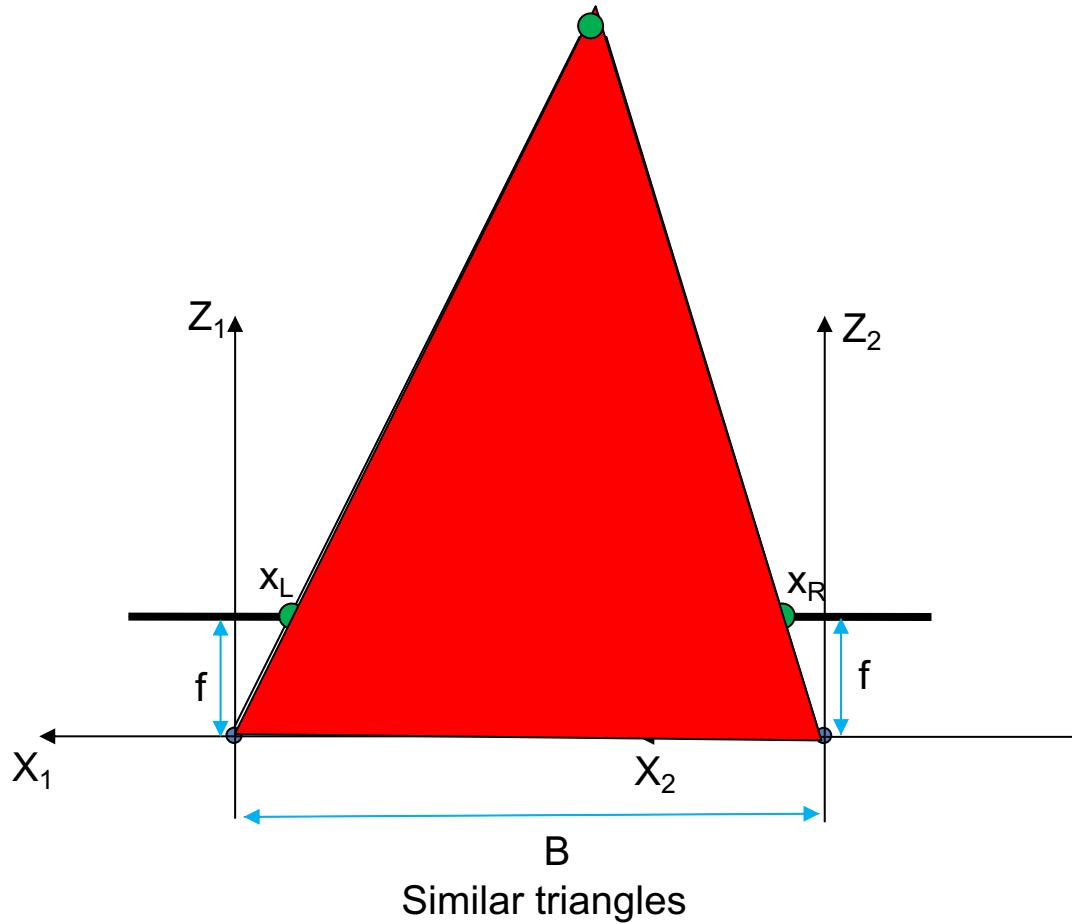




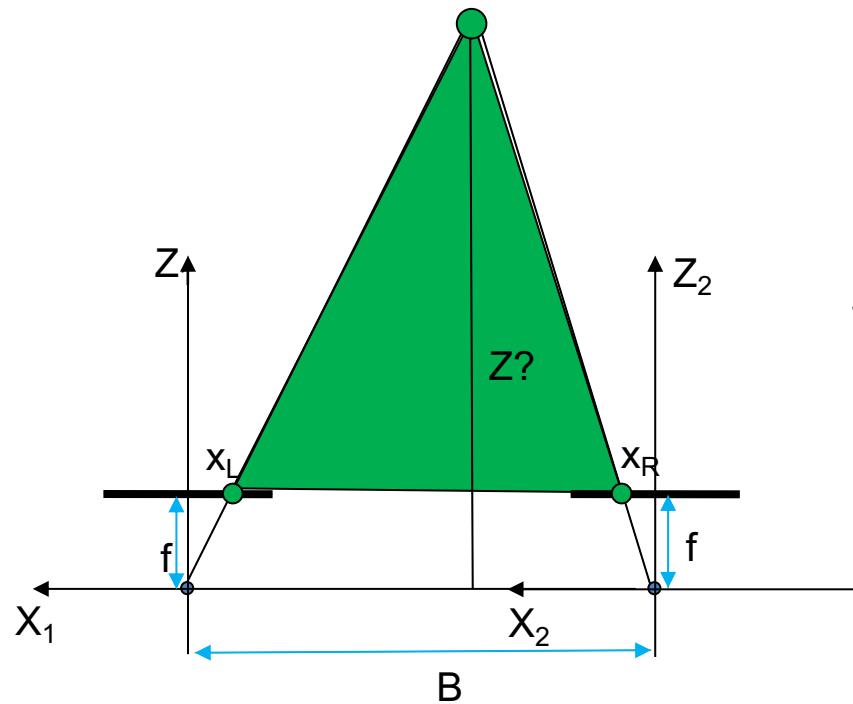
# Depth from Stereo Matches



# Depth from Stereo Matches



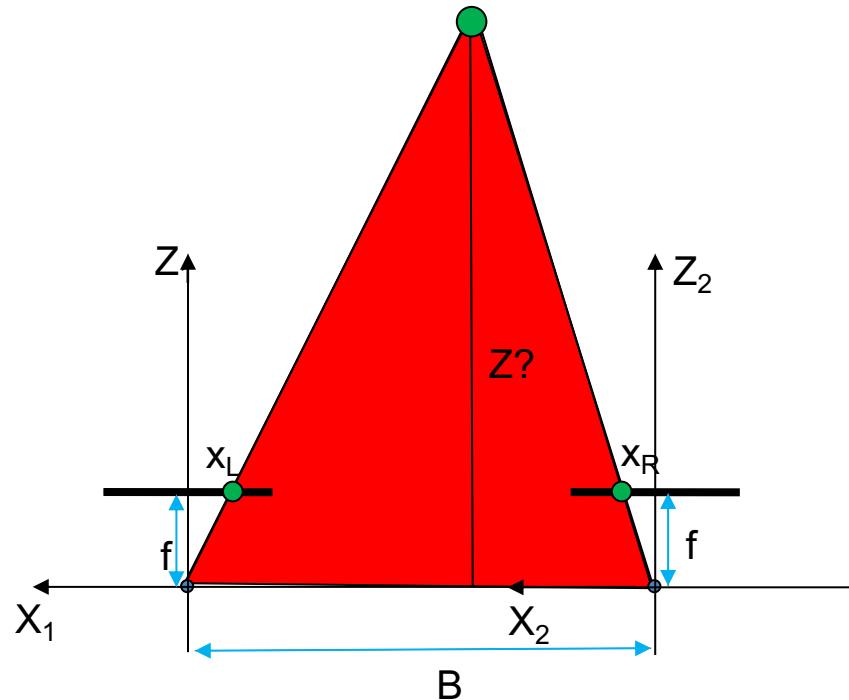
# Depth from Stereo Matches



Similar triangles:

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

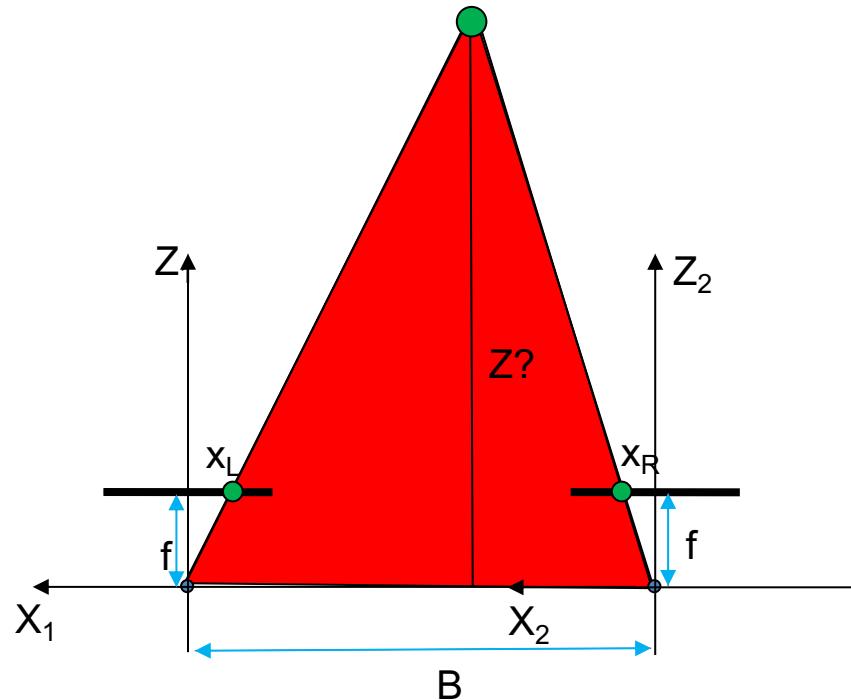
# Depth from Stereo Matches



Similar triangles:

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

# Depth from Stereo Matches



Similar triangles:

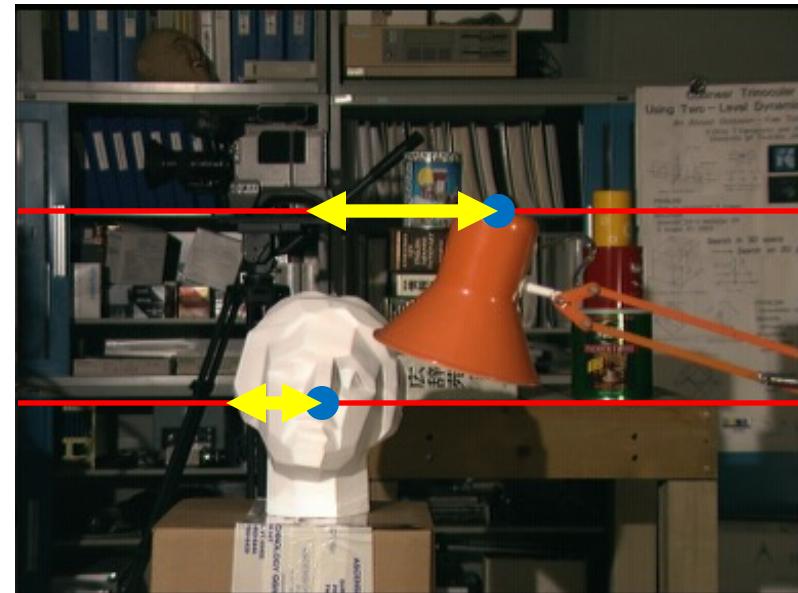
$$\frac{B - x_L + x_R}{Z - f} = \frac{B}{Z}$$

Solving for  $Z$ :

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

Disparity

# Depth from Stereo Matches



# Depth from Stereo Matches

Reference Image



Disparity Map



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

Disparity

A diagram illustrating the geometric relationship between disparity and depth. A red elliptical outline represents a point in the scene. A horizontal line segment connects the center of the ellipse to a point on its left edge. The distance between the center and the point on the edge is labeled  $X_L - X_R$ . The distance from the center to the ellipse is labeled  $B$ . A green arrow points from the text "Disparity" to the horizontal line segment.

- 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

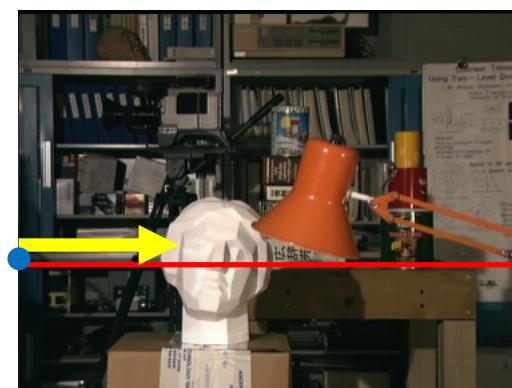
- Local Methods (Area-based)
- Global Methods (Energy-based)
- Other Methods

# Sparse Stereo Correspondence

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

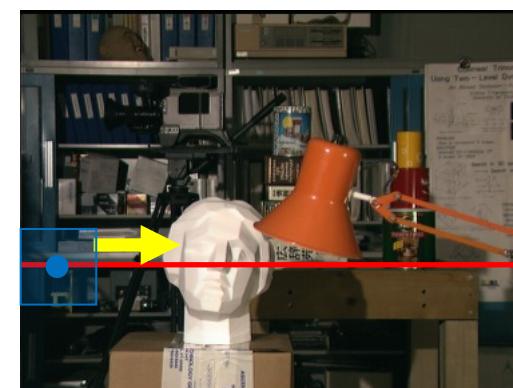
# Dense Stereo Correspondence : Local Methods

- 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)

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

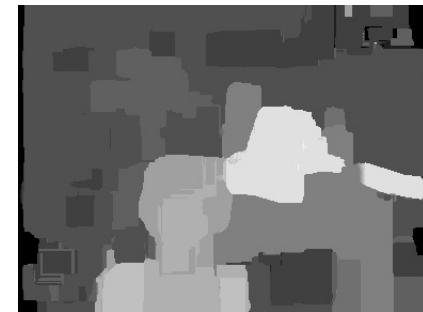
- Sum of Squared Differences (SSD)

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

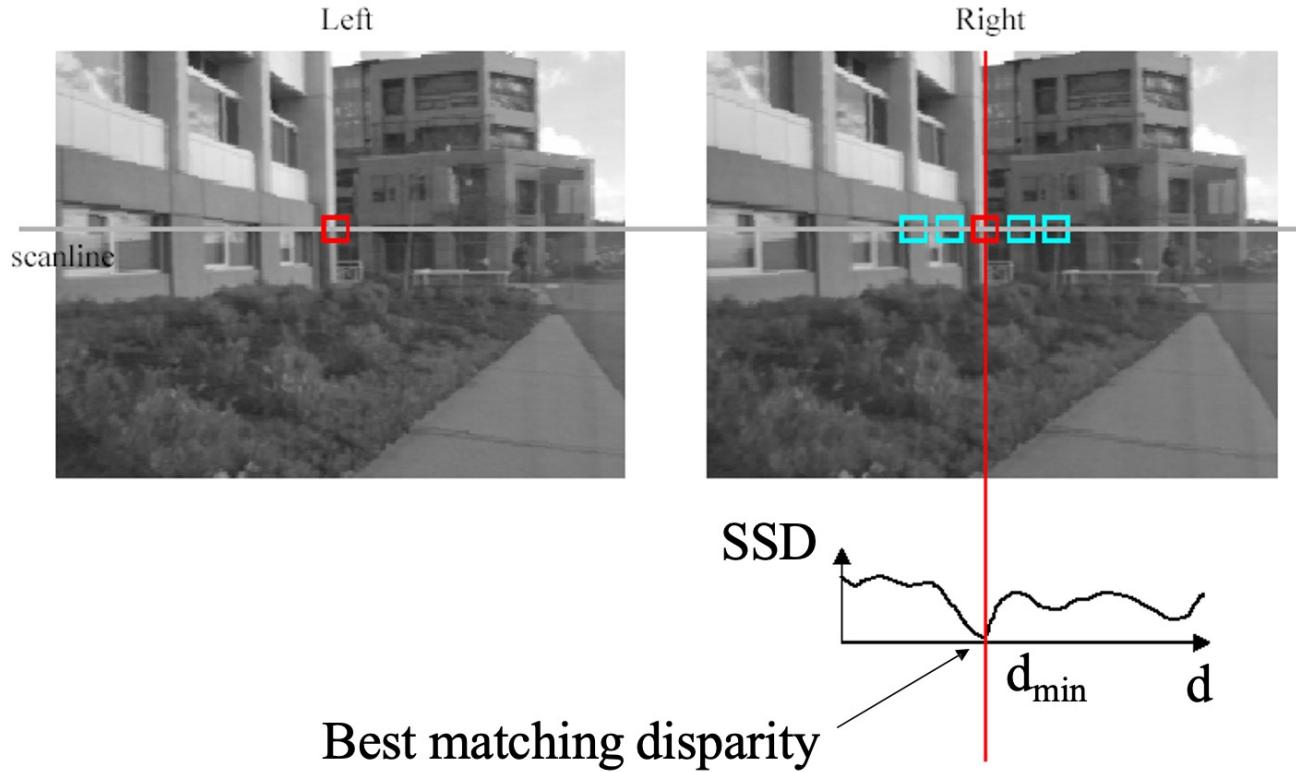
- Normalized Cross-Correlation

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

- ...many many more!!!



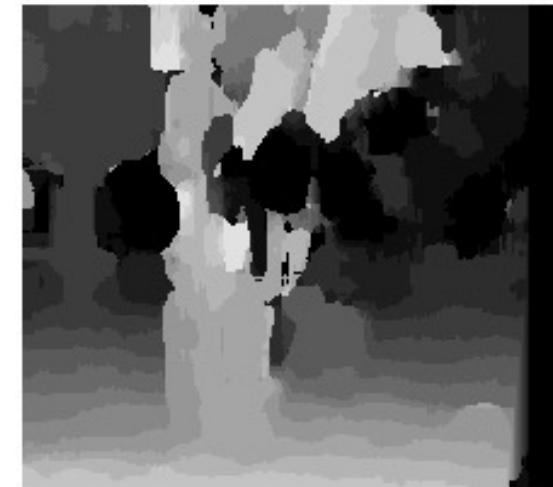
# Stereo Correspondence Metrics: SSD



# Stereo Correspondence Metrics: SSD on various windows



$W = 3$



$W = 20$

- Small vs Big windows
- What are their Pros and Cons?

# Stereo Correspondence Metrics: Good/Bad areas



- In this stereo image pair:
  - what would be good areas to match?
  - where would you expect to face problems and why?

# 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)

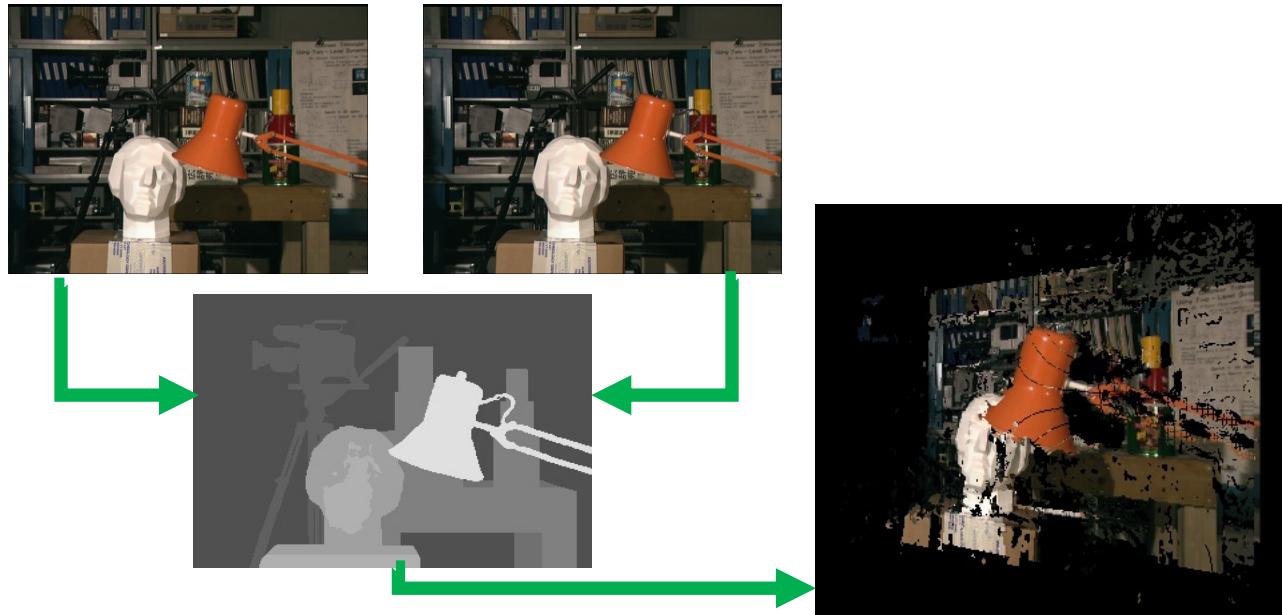


# Global Stereo Correspondence

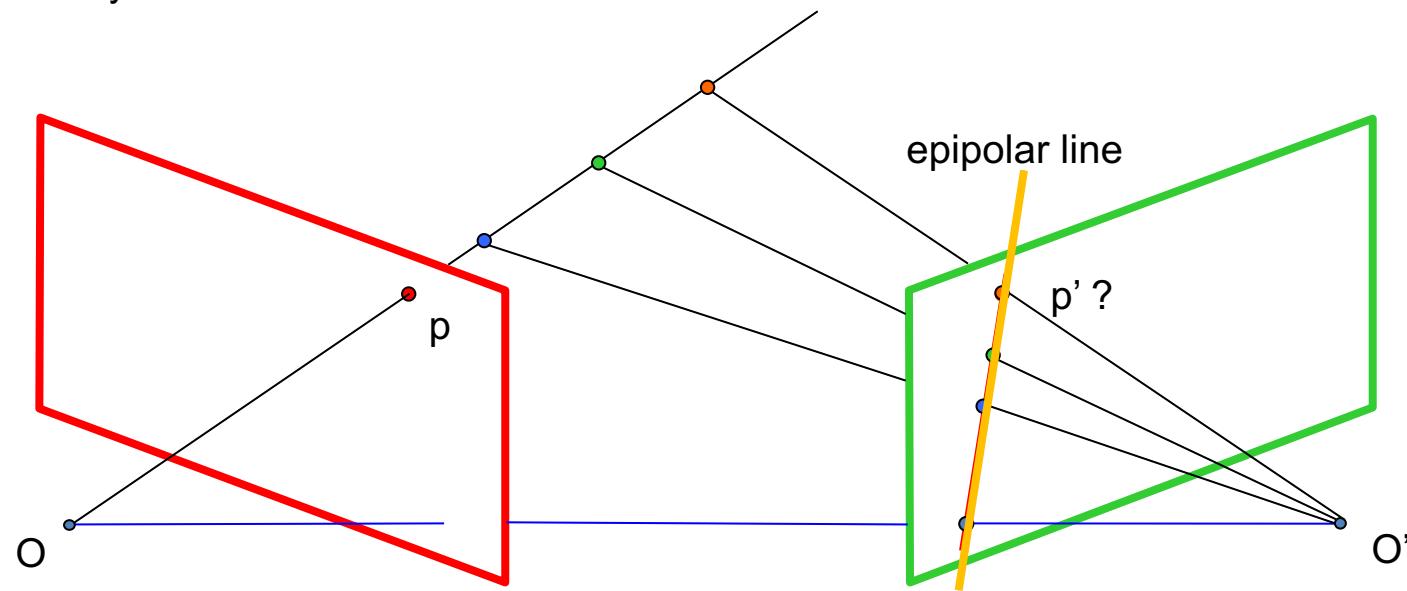
- Up to this point, the disparity of each pixel was determined only by the information of the pixel itself and its neighborhood.
  - Thus, those methods are called "local" or "area-based" methods.
- Global methods find better solutions in expense of more computations
  - Optimize jointly the disparity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

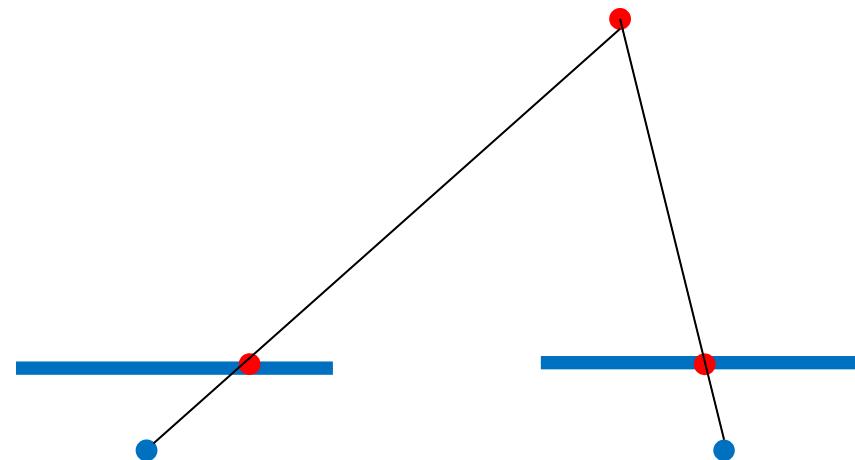
- We discussed about what Stereo Vision is.



- 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

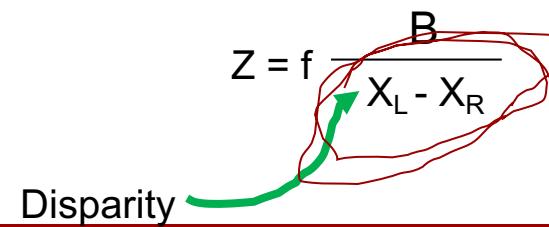


- We discussed about what Stereo Vision is.
- We learned about :
  - Stereo/Epipolar Geometry
  - Rectified Stereo Case
  - Depth from Stereo Matches

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

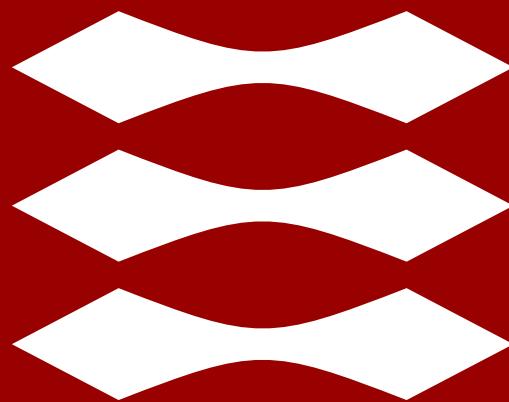
Dense output

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

Lazaros Nalpantidis

# Stereo Vision

**DTU**



Perception for Autonomous Systems 31392:

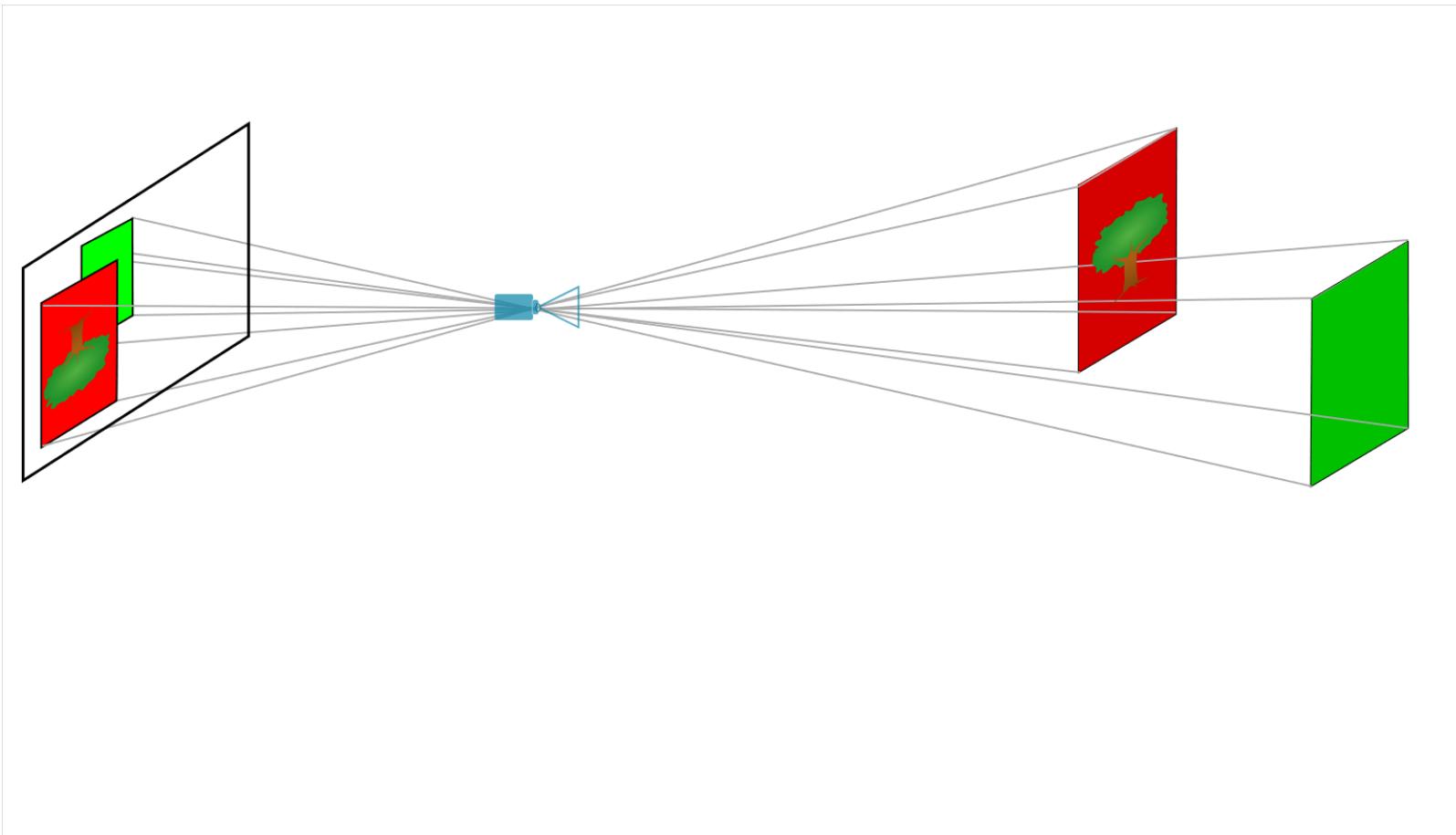
# **Camera Matrix and Camera Calibration**

Lecturer: Evangelos Boukas—PhD

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

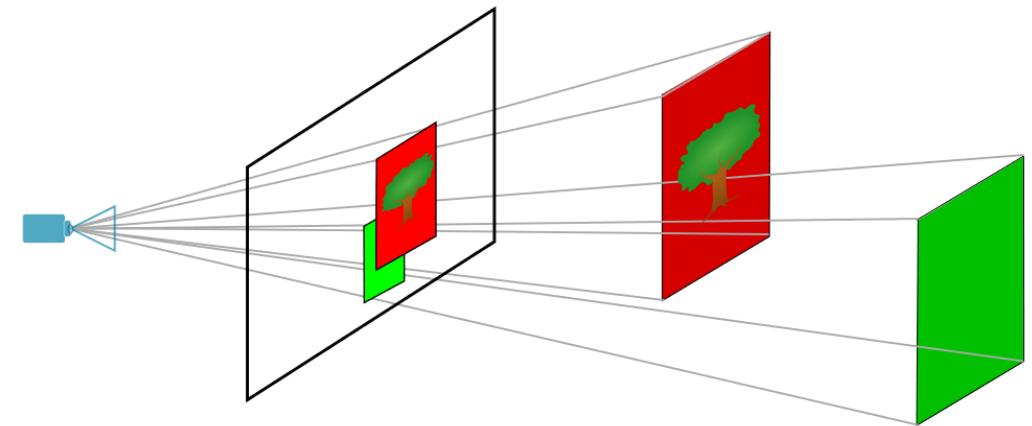
# Image Formation

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



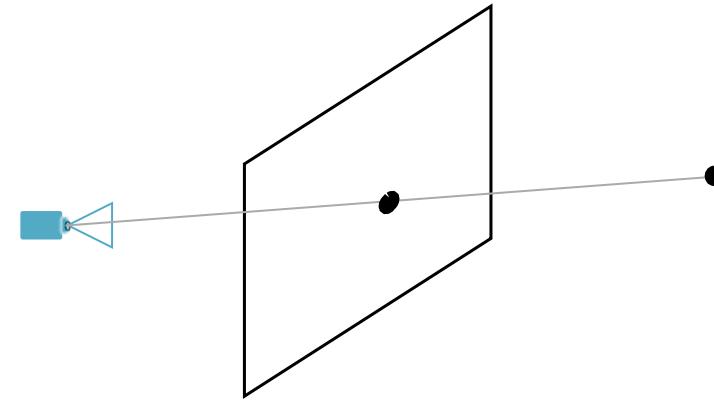
# Image Formation

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



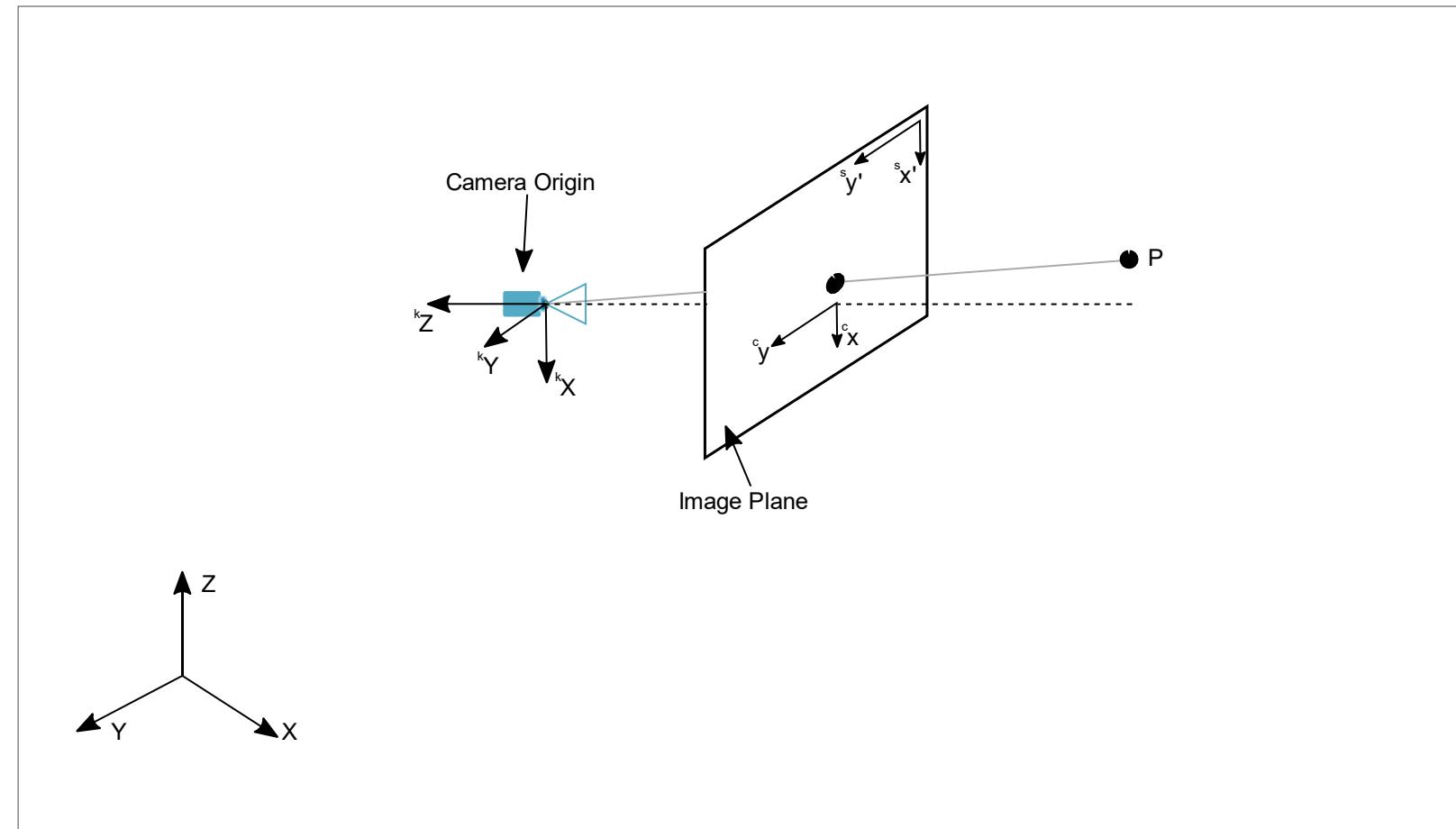
# Image Formation

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



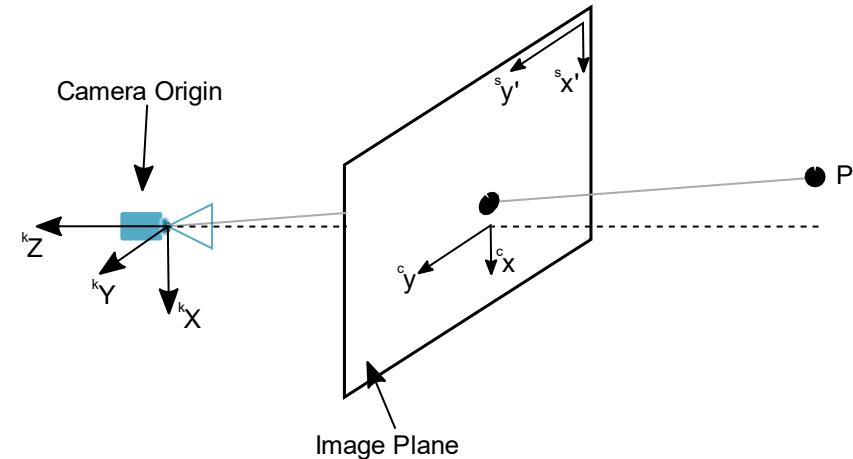
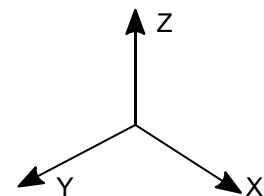
# Image Formation

- 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

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

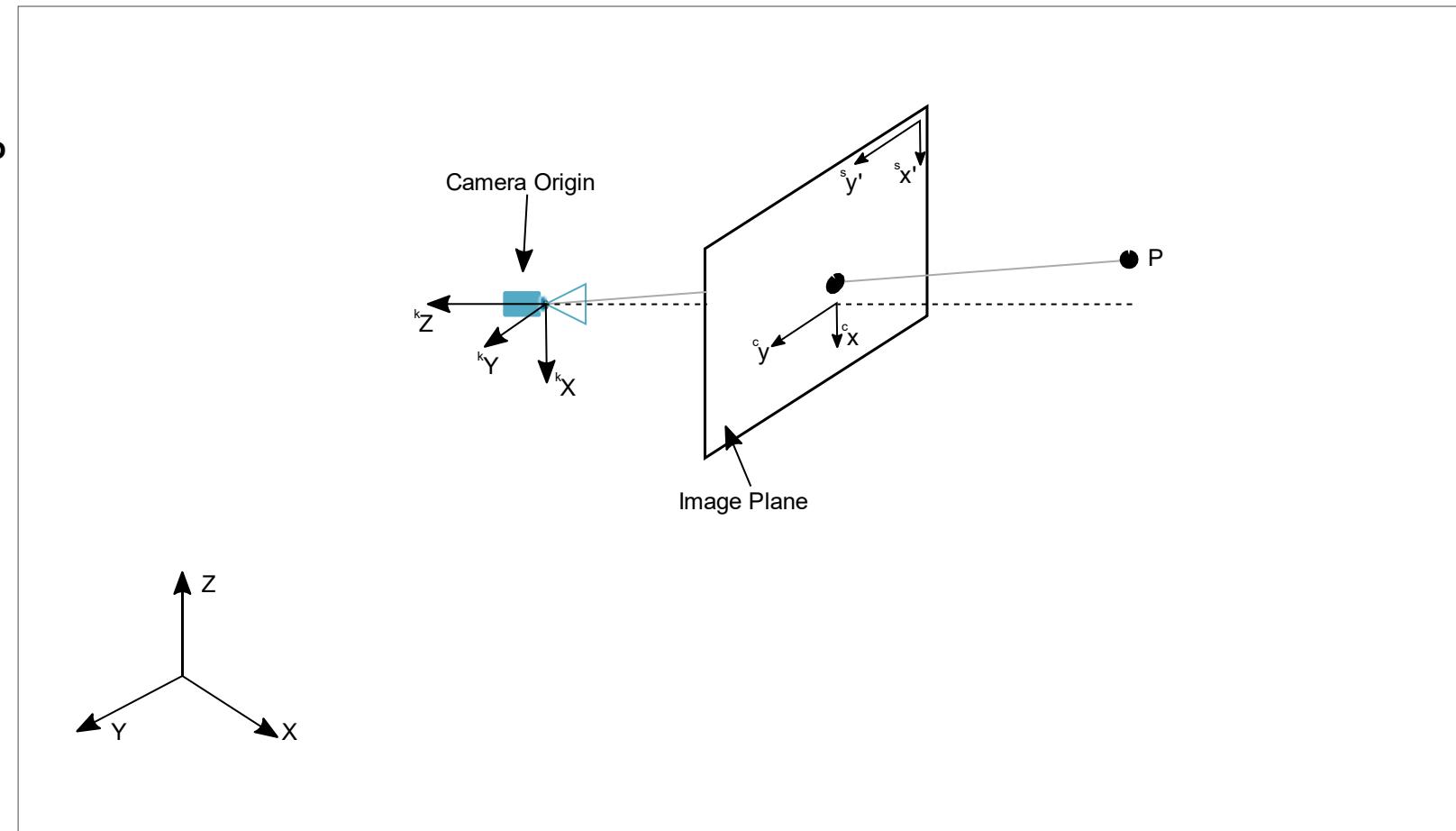
- To do so we need to use  
 the camera Origin

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

- Then:

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

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

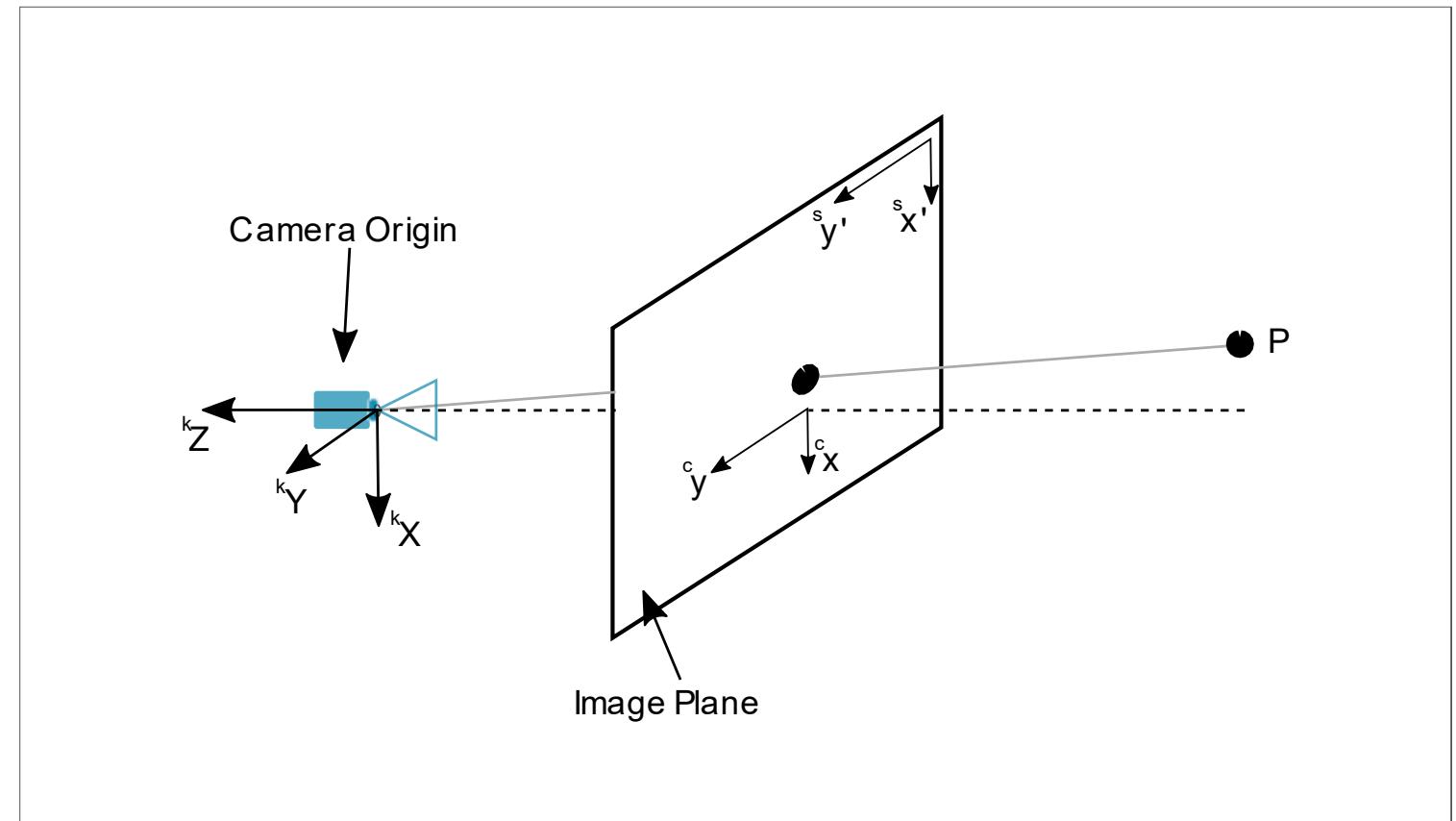


# Camera Extrinsics

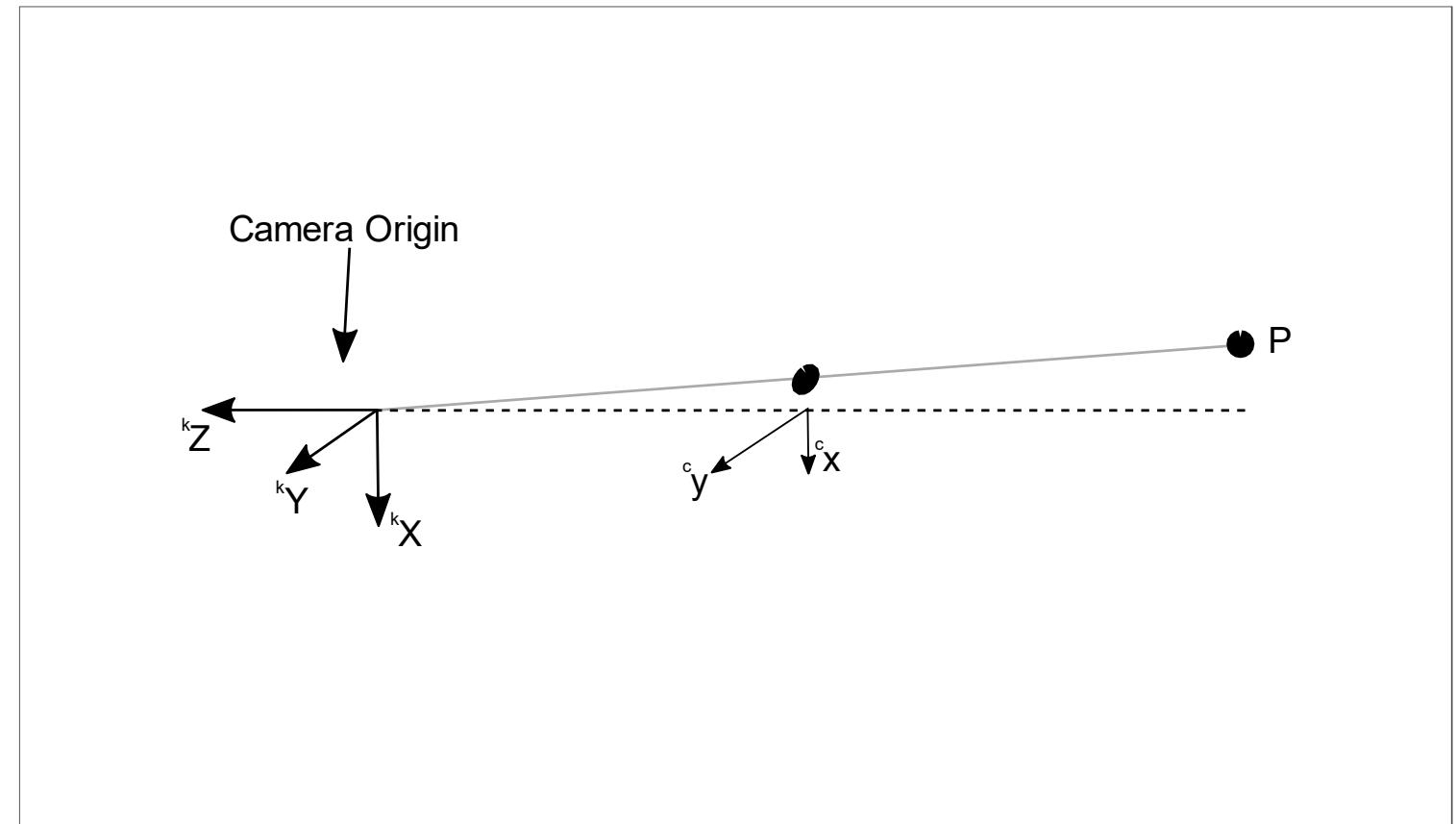
- So now we have a description for the 3D point in camera frame

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

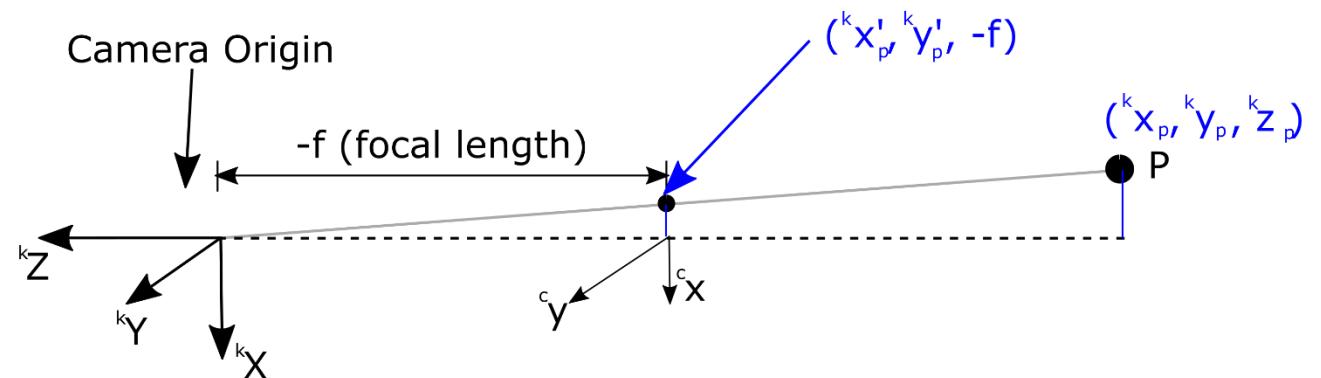
- Let's get rid of things we don't need



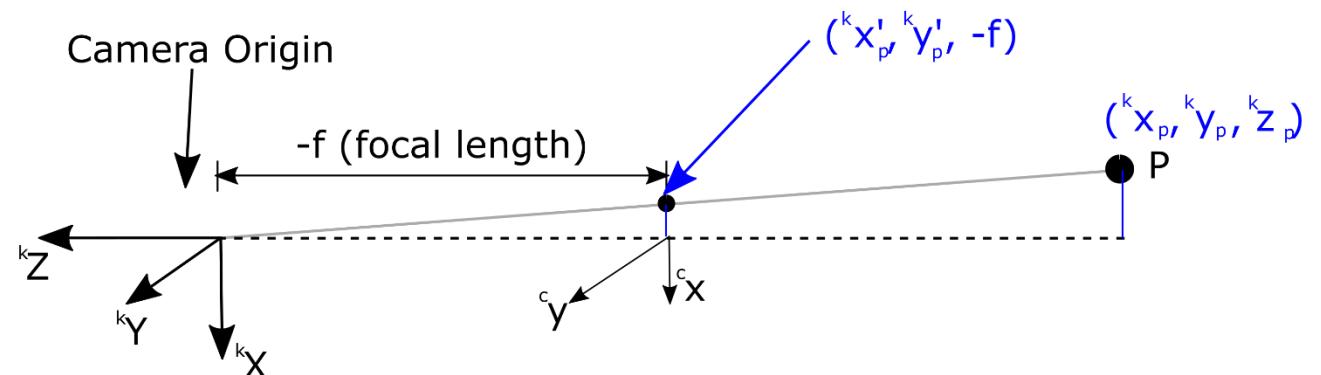
- Let's get rid of things we don't need



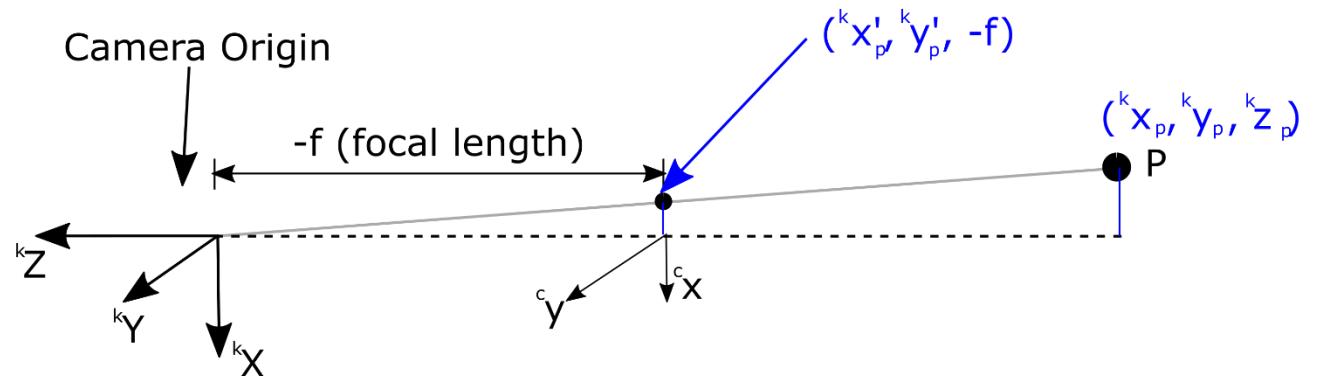
- Let's get rid of things we don't need



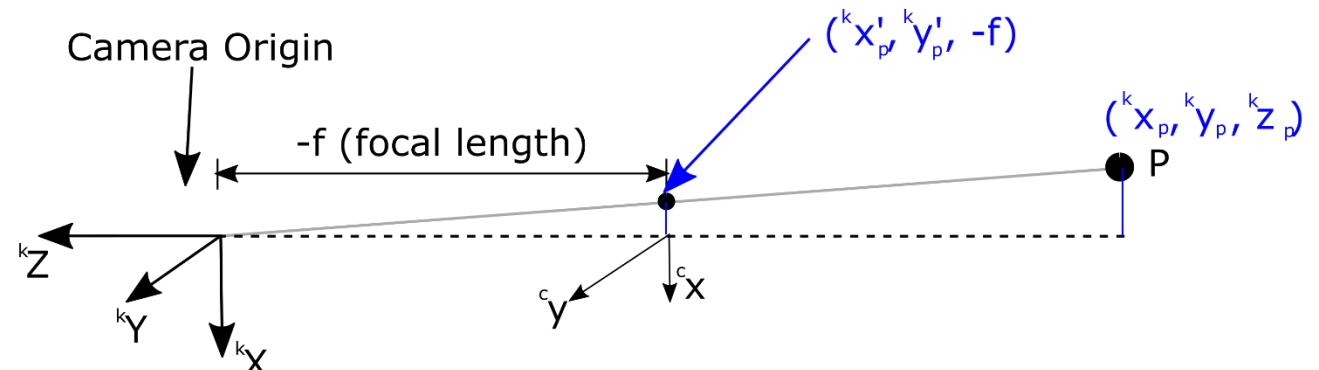
- Let's get rid of things we don't need
- So it is clear that the position of the projection on the image frame is dependent on the ratio  $x/z$



- 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

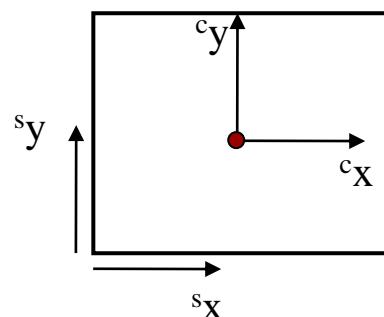
- From the previous

- 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,  $\lambda$  is used for scale and therefore does not provide a 3<sup>rd</sup> 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*)

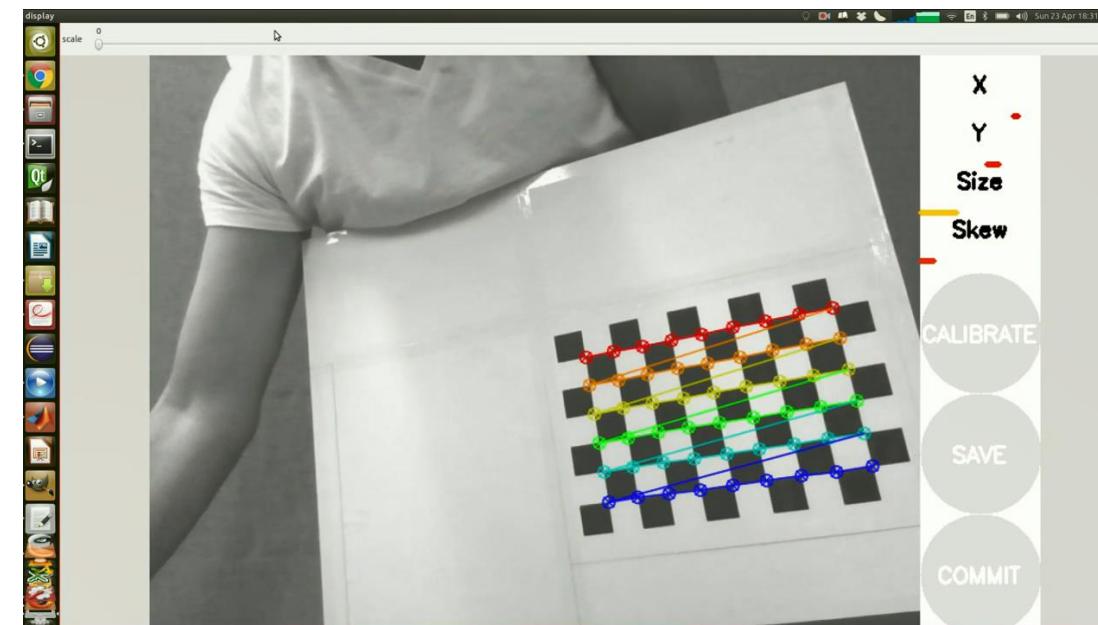
-Reshape into the P matrix

# 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

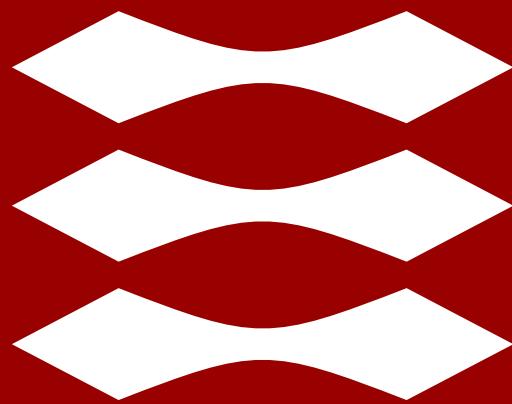
- What did we learn?

Perception for Autonomous Systems 31392:

# **Camera Matrix and Camera Calibration**

Lecturer: Evangelos Boukas—PhD

**DTU**



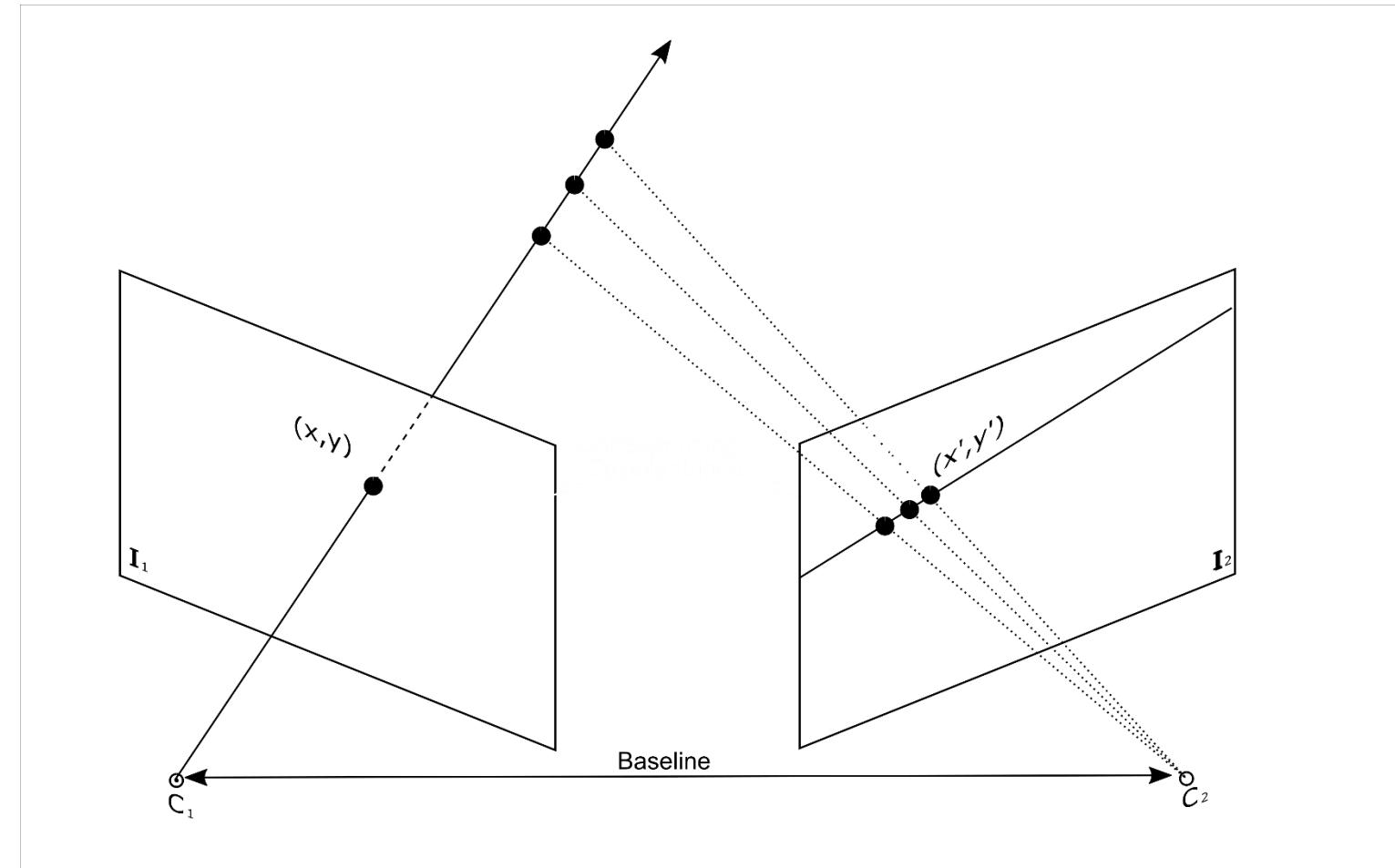
Perception for Autonomous Systems 31392:

# **Epipolar Geometry and the Fundamental Matrix**

Lecturer: Evangelos Boukas—PhD

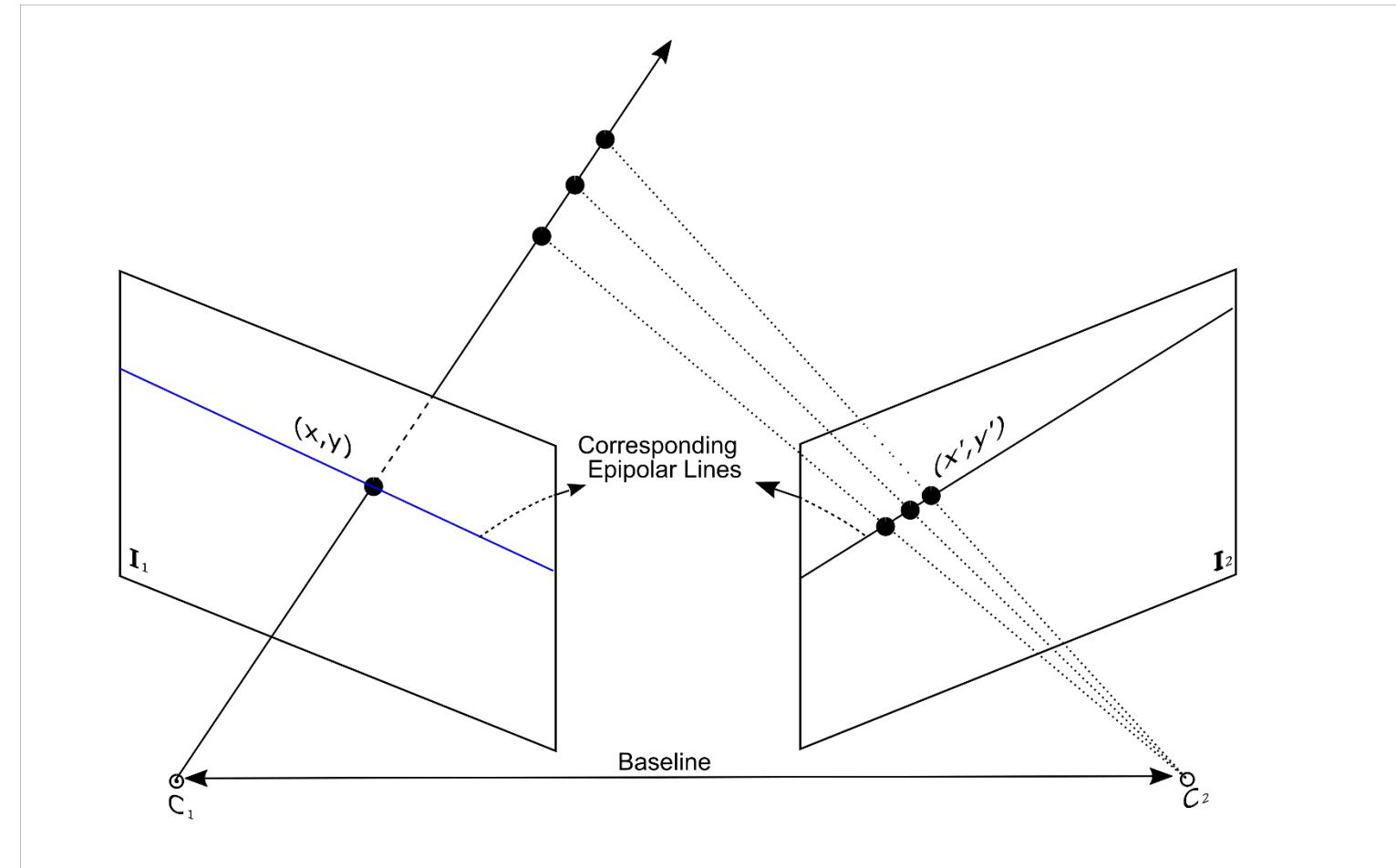
# Epipolar Geometry: General Case

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



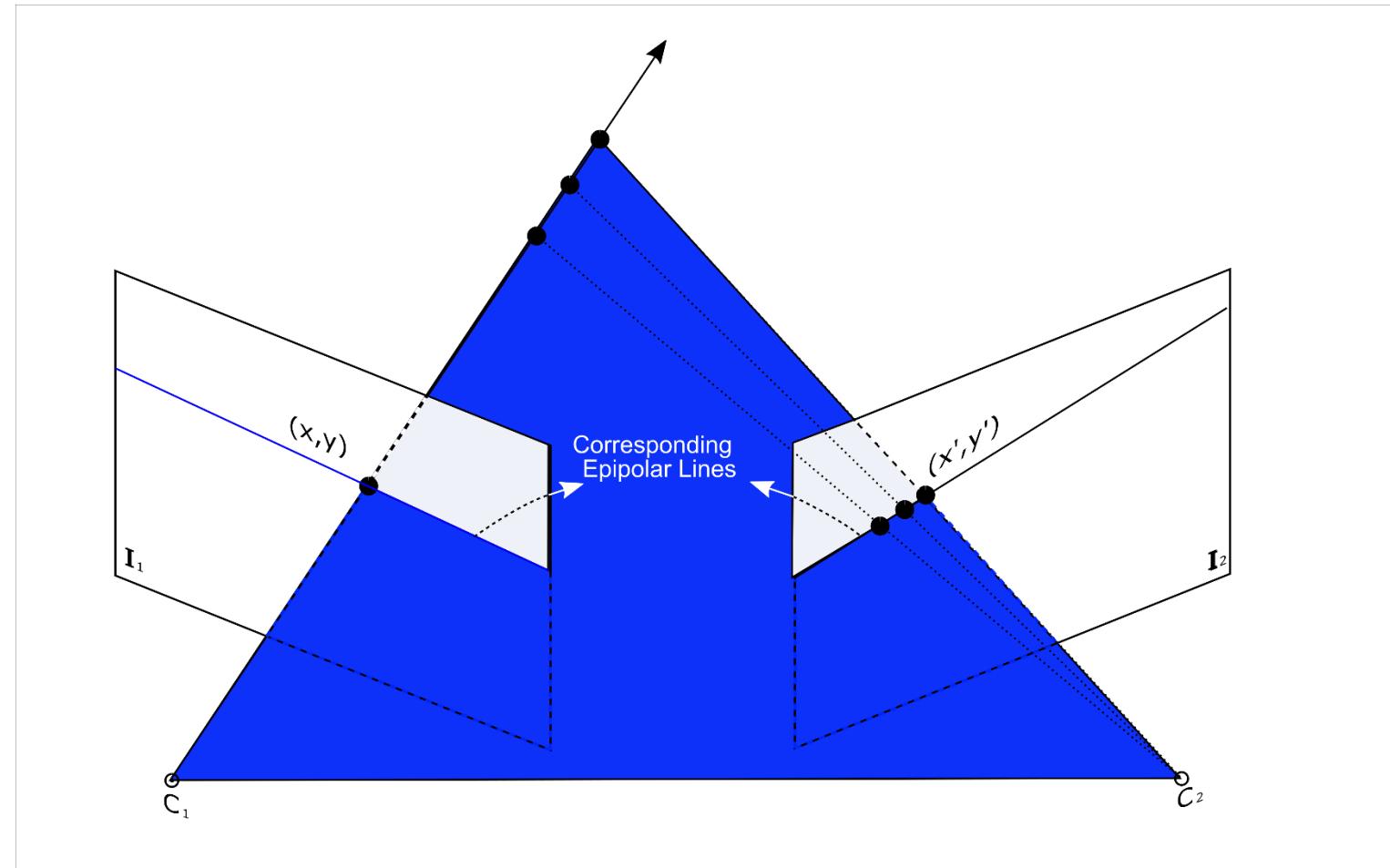
# Epipolar Geometry: General Case

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



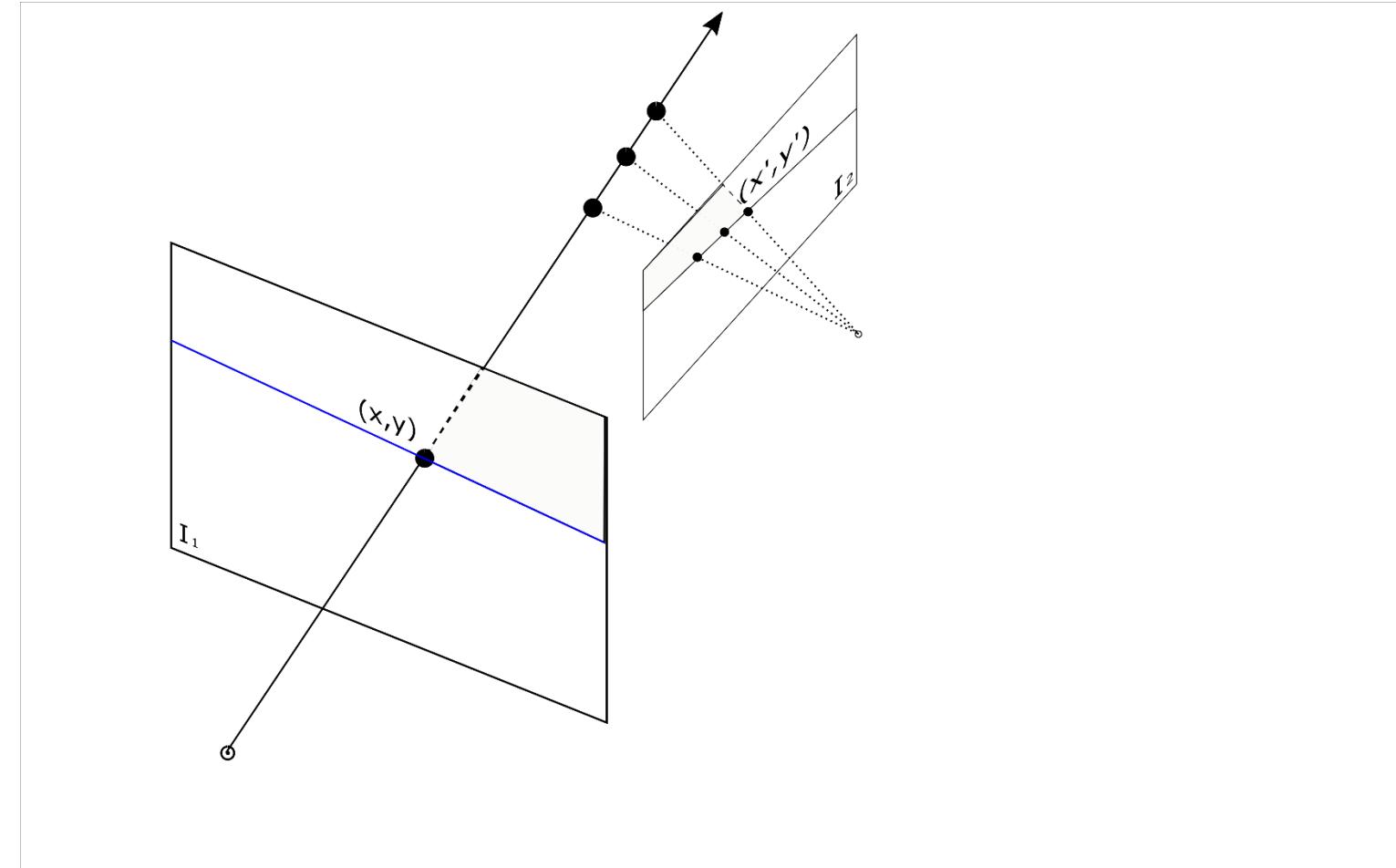
# Epipolar Geometry: General Case

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



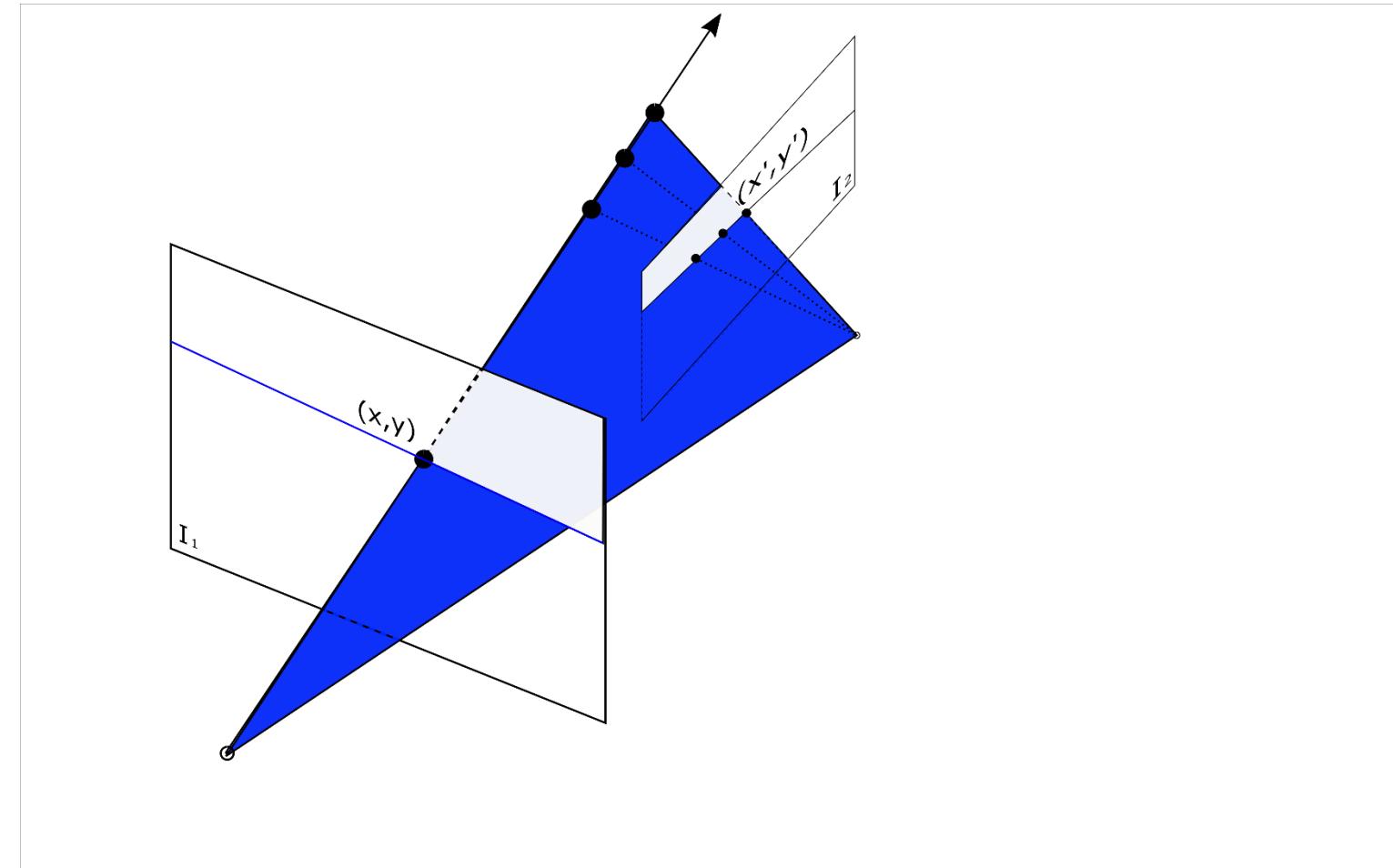
# Epipolar Geometry

- What about this case?



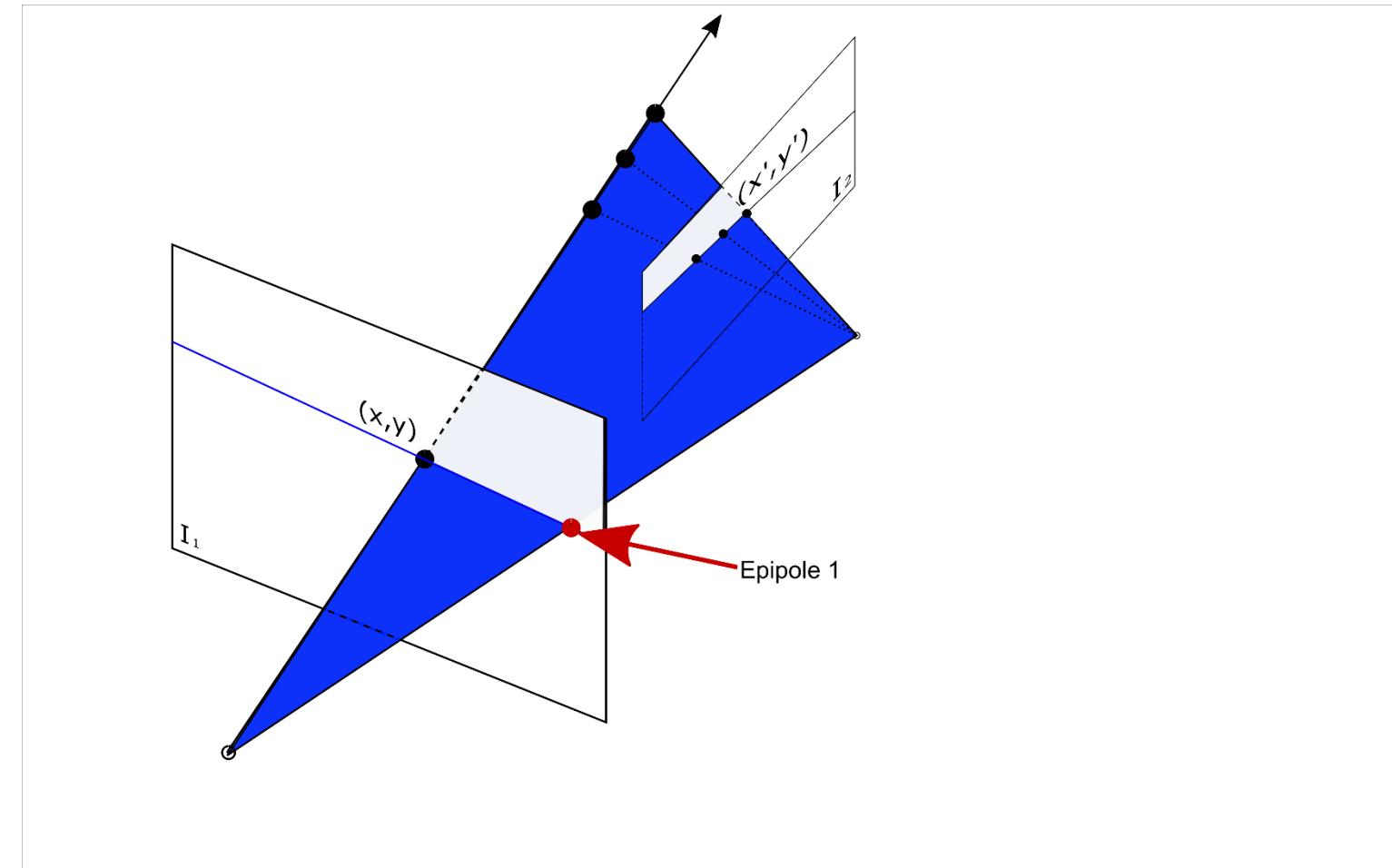
# Epipolar Geometry

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



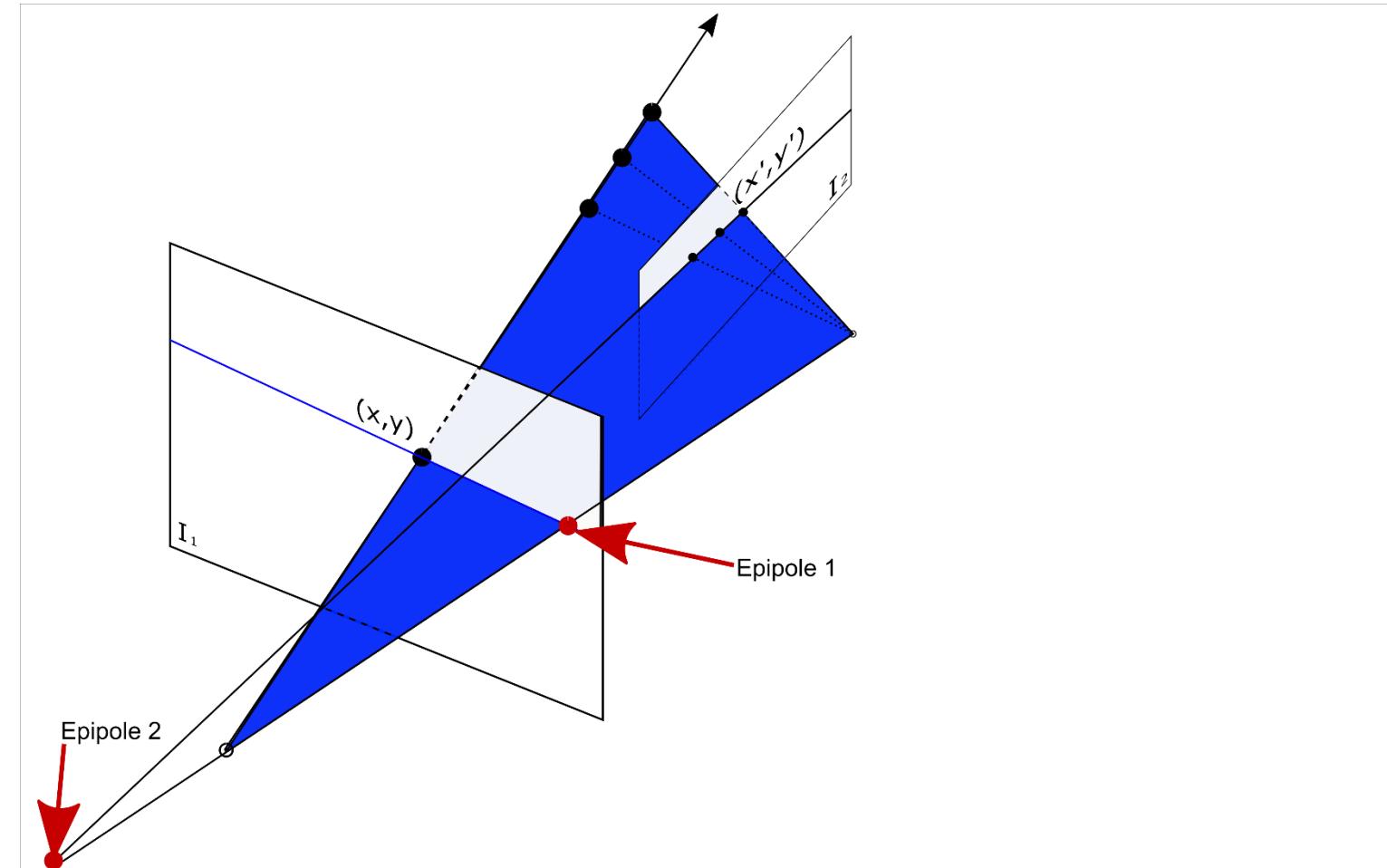
# Epipolar Geometry

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



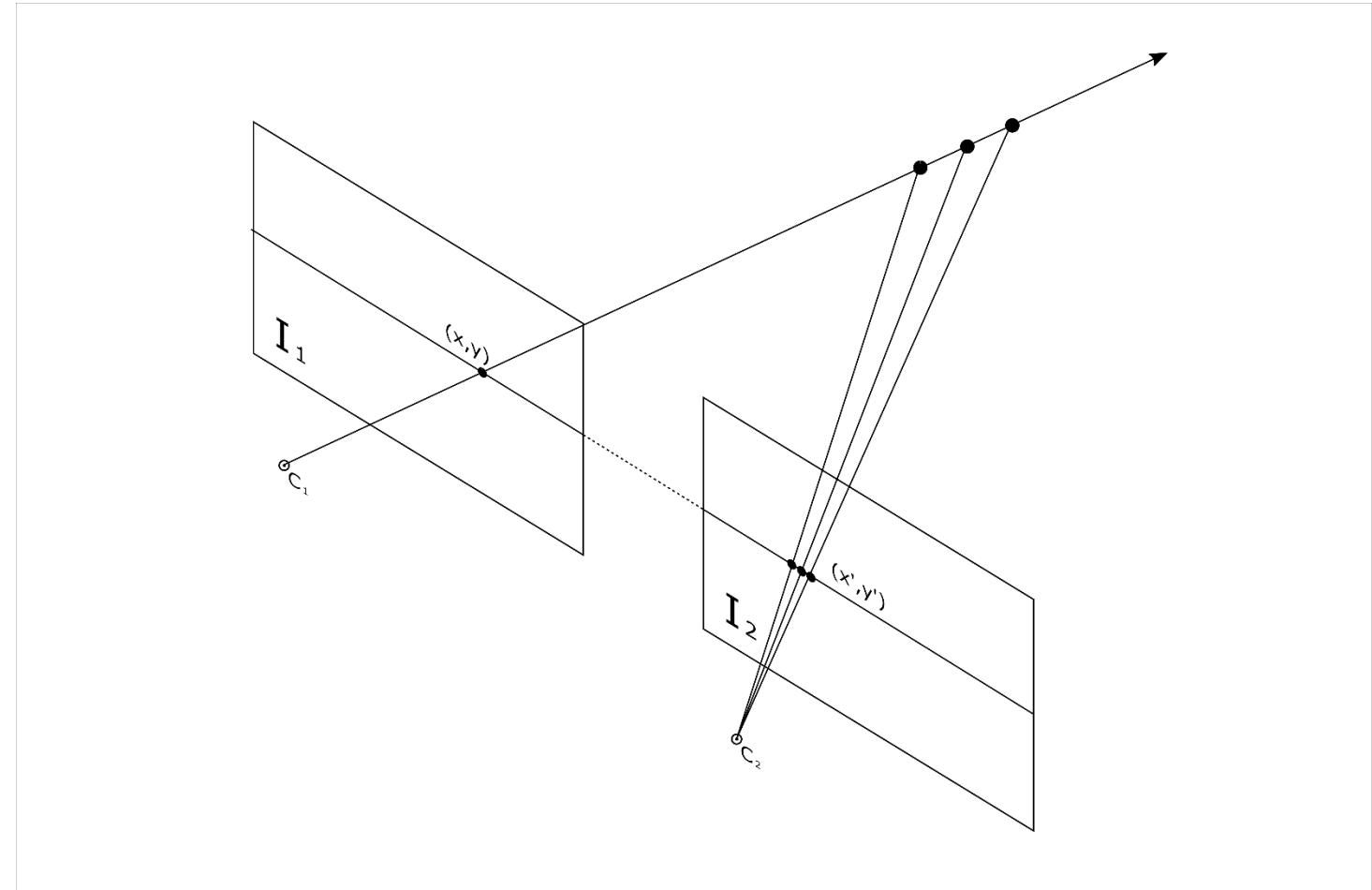
# Epipolar Geometry

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



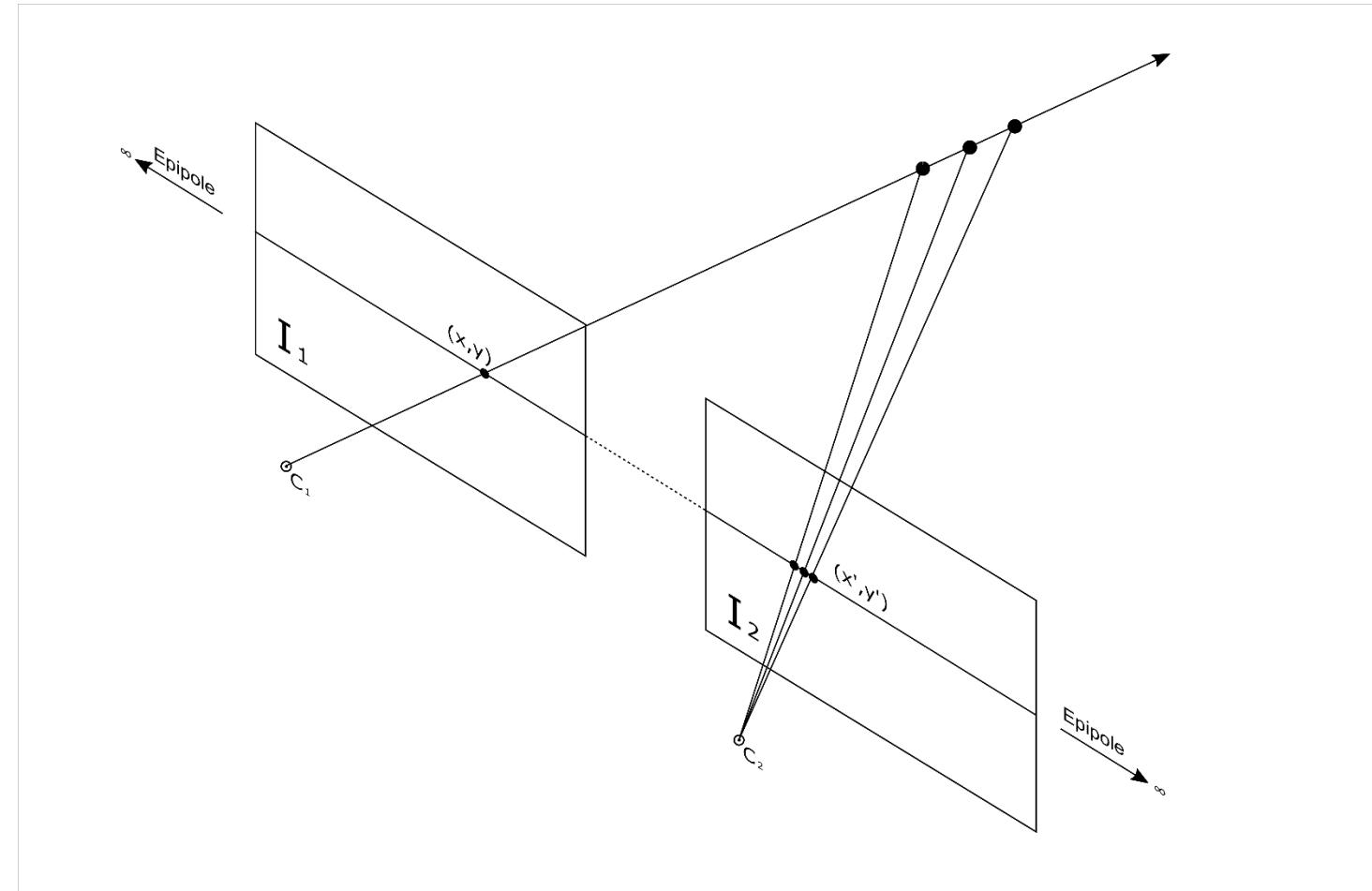
# Epipolar Geometry:

- How about this case?



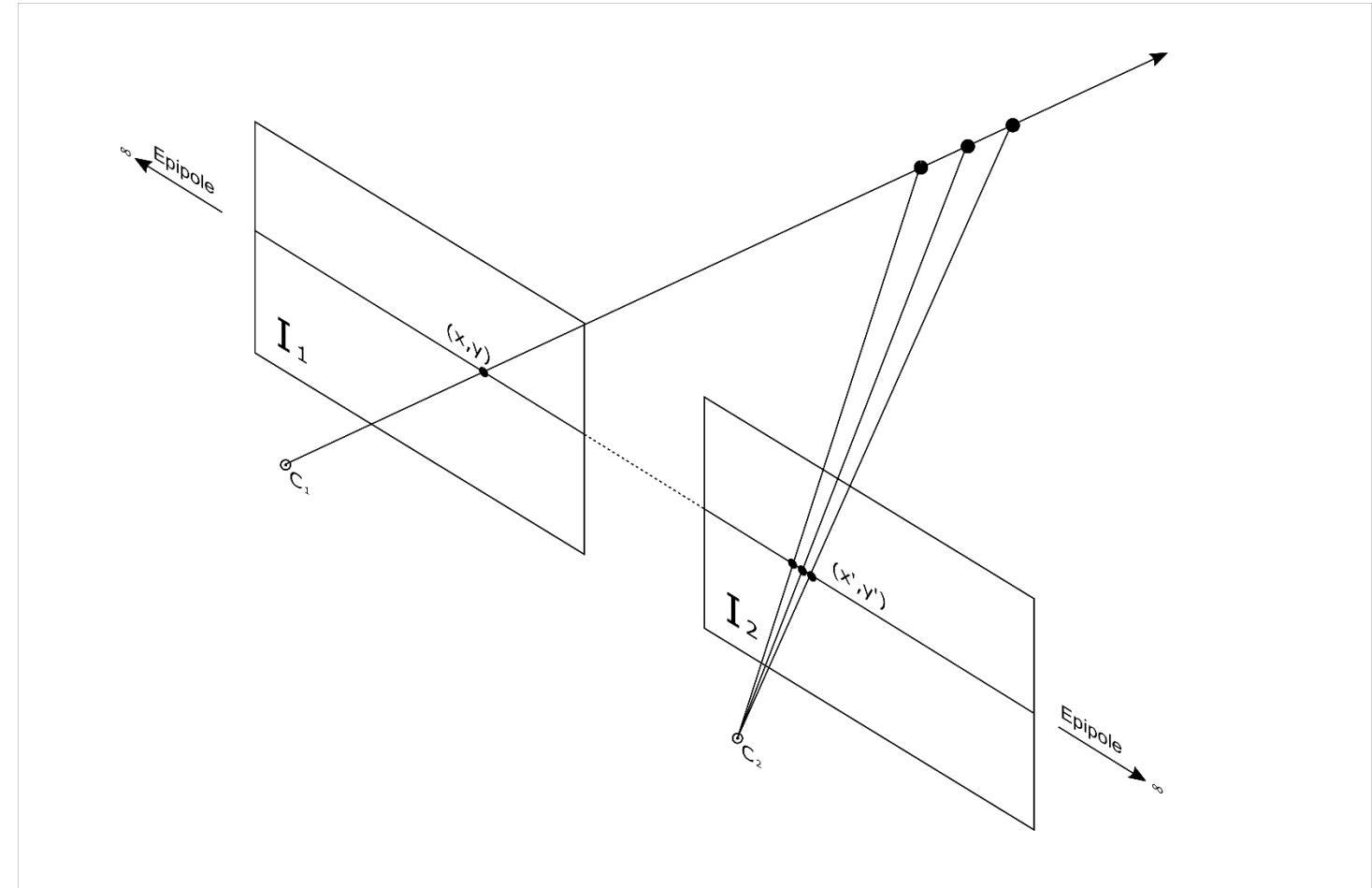
# Epipolar Geometry:

- 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”

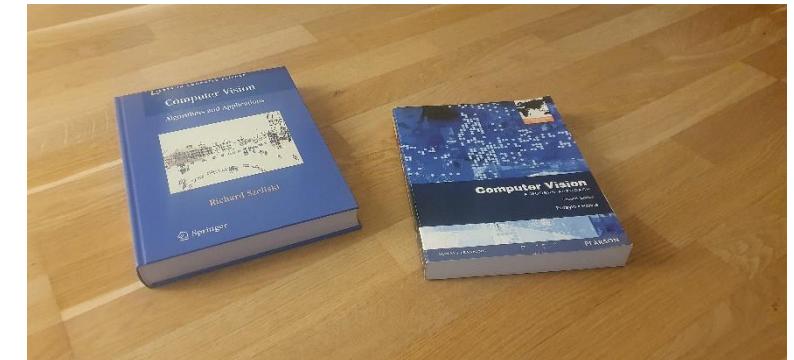
- Let's think of this example:



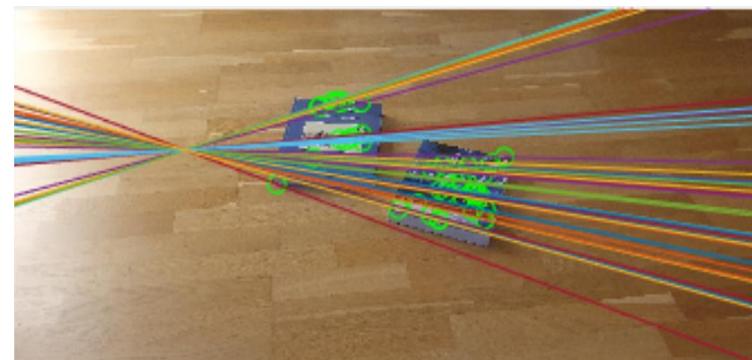
- Where should the epipole be?

# 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”

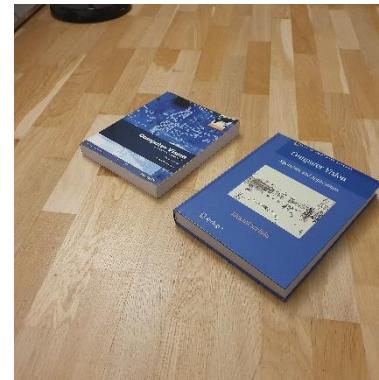
- What about this:



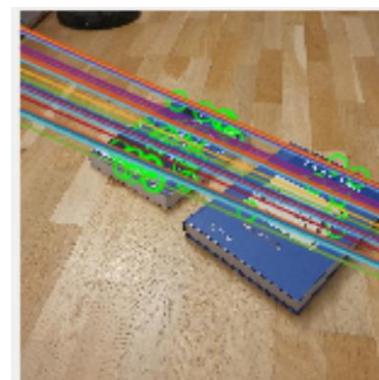
- How should the epipolar Lines look like?

# Example of Epipolar Lines “in the wild”

- What about this:



- How should the epipolar Lines look like?

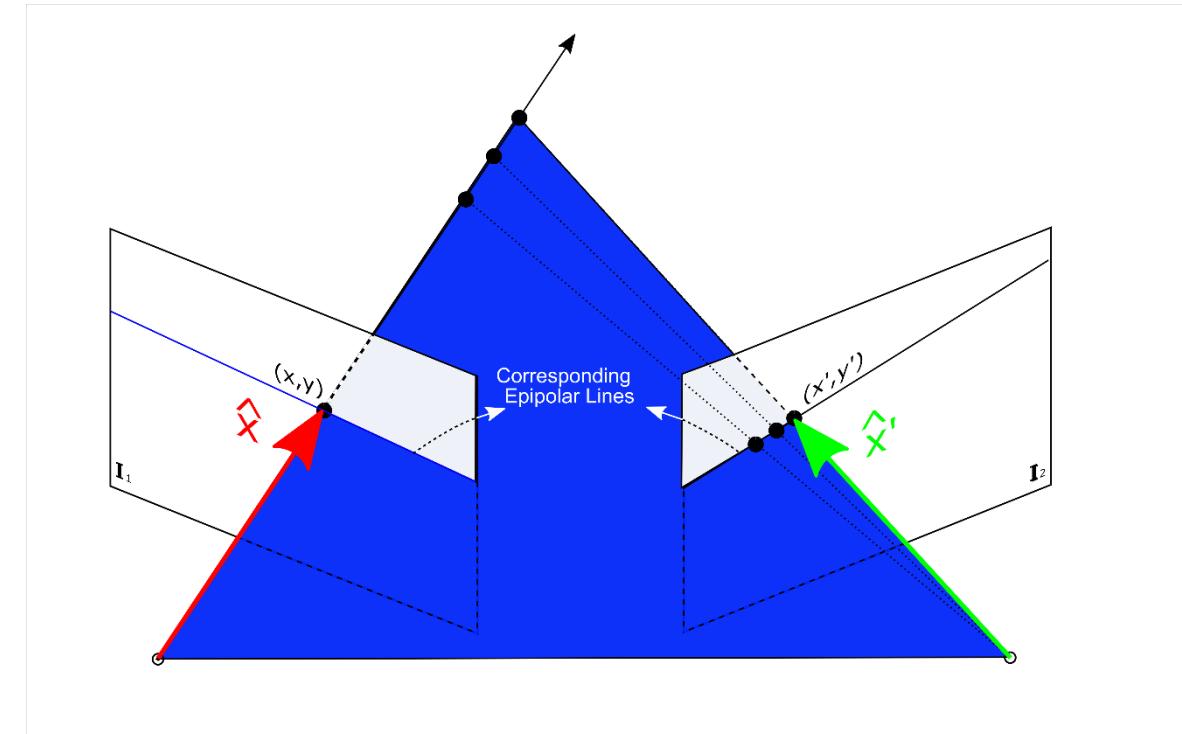


# Essential Matrix

- 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

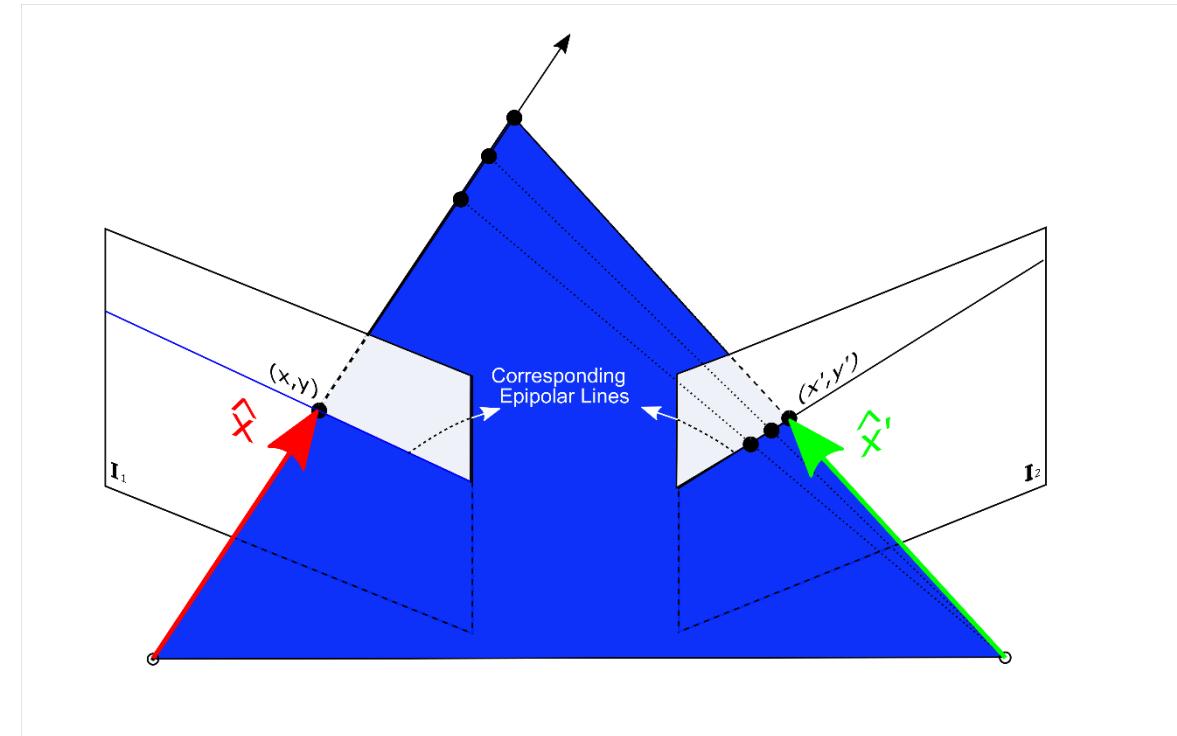
- We can express the left homogeneous point to the right one:
- $\hat{x} = R * \hat{x}' + T$ , where R is the Rotation Matrix and T the translation vector
- Then we can prove that there is a Matrix connecting the two points  $\hat{x}, \hat{x}'$
- Trying to eliminate the left side by Applying cross product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

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

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

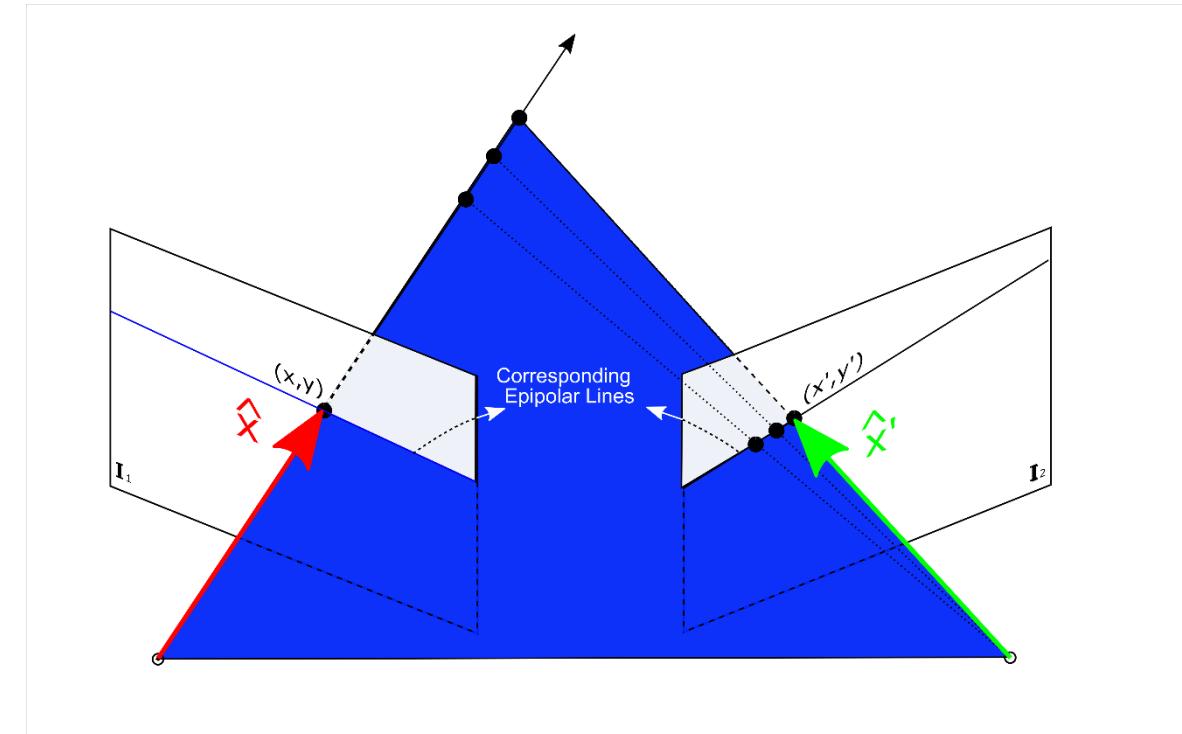
$$0 = \hat{x} \cdot T \mathbf{x} R * \hat{x}'$$

- 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  $3 \times 3$  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



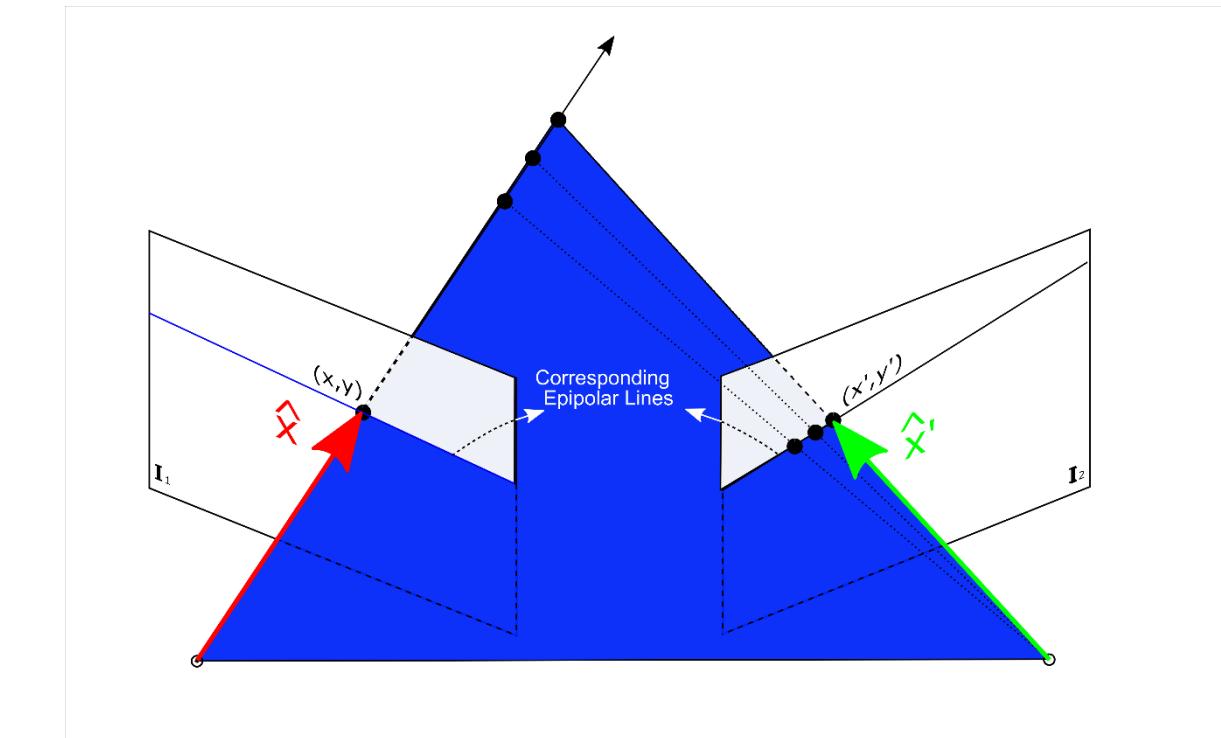
# Fundamental Matrix

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

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

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

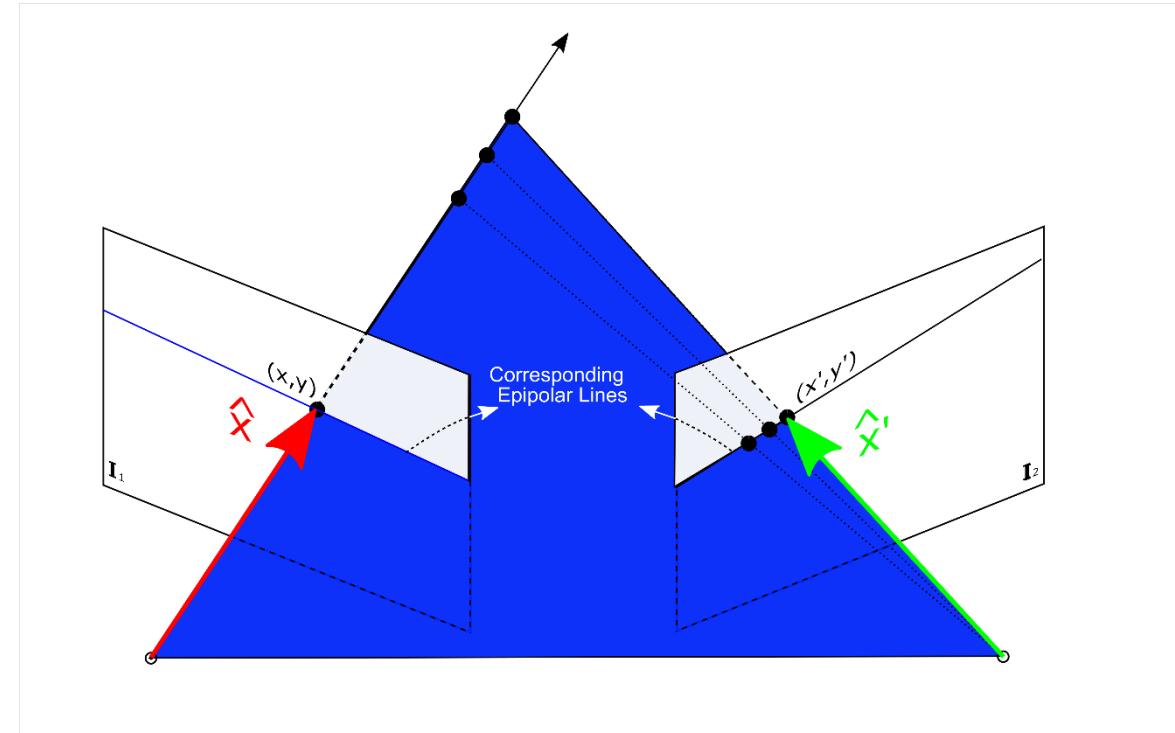
$$\hat{x}' = K'^{-1}x'$$



Which is the fundamental matrix

# Fundamental Matrix

- The fundamental matrix  $F = K^{-T} E K'^{-1}$  is a  $3 \times 3$  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?**

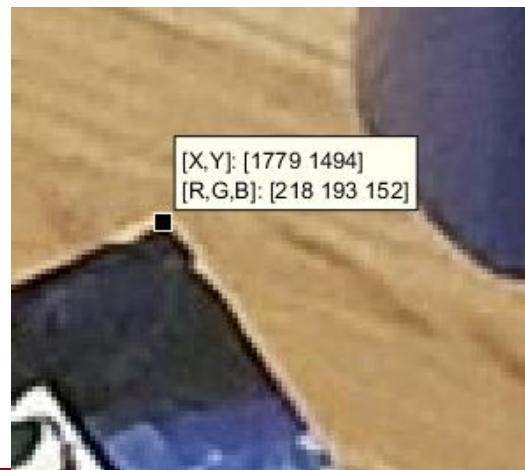
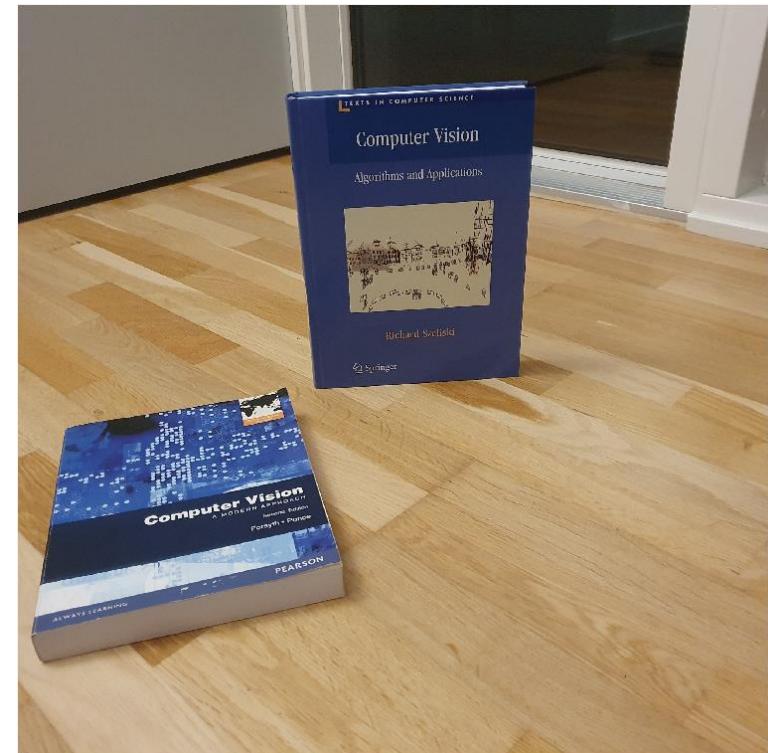


# Rectified Epipolar Geometry



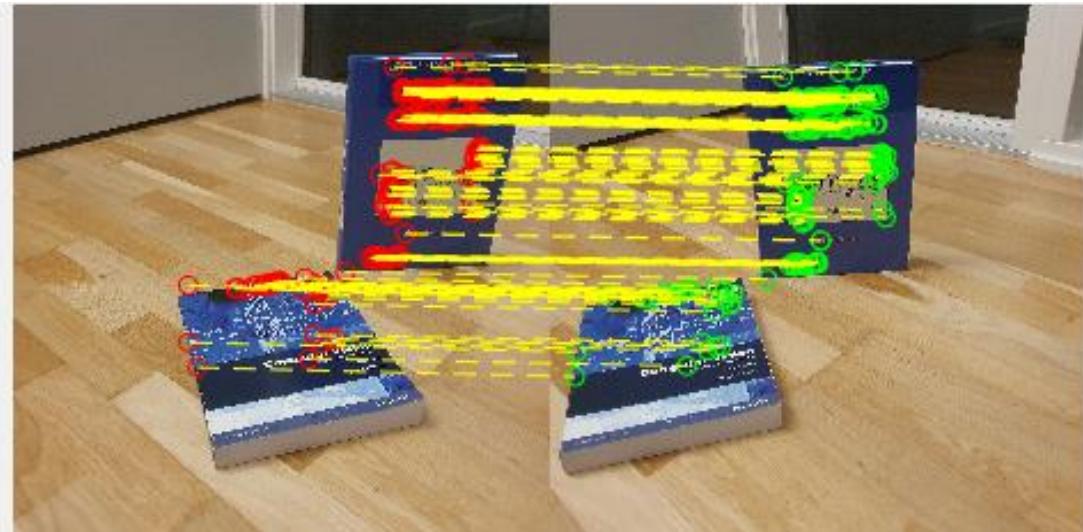
# An Example

- Assume these two images

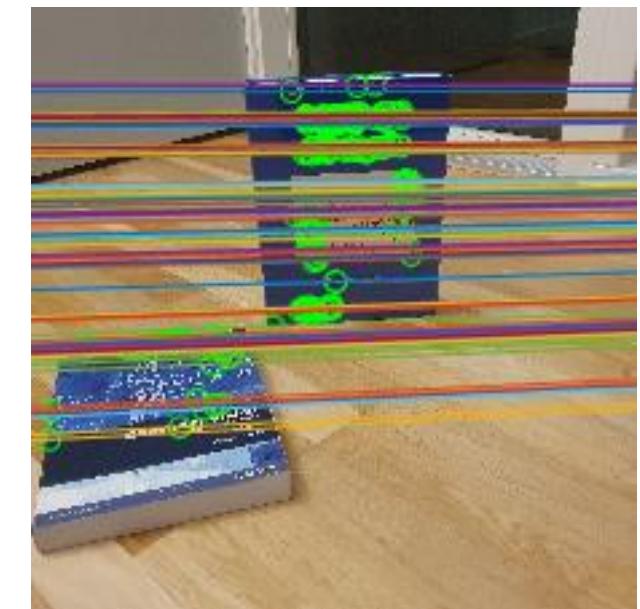
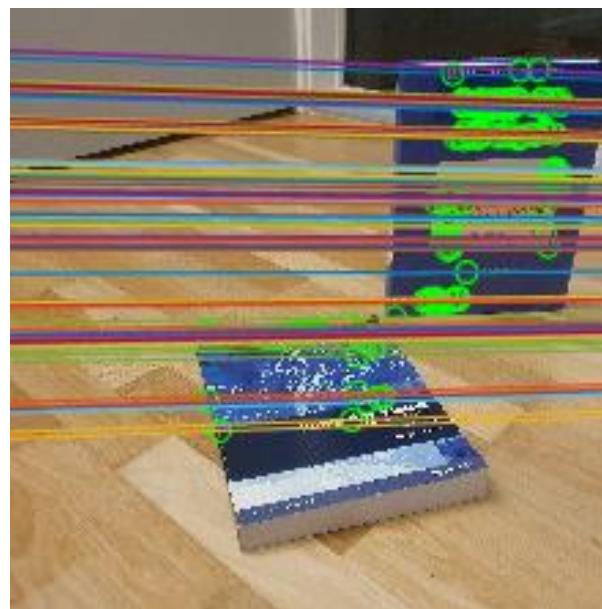


# An Example

- First step is to match some points

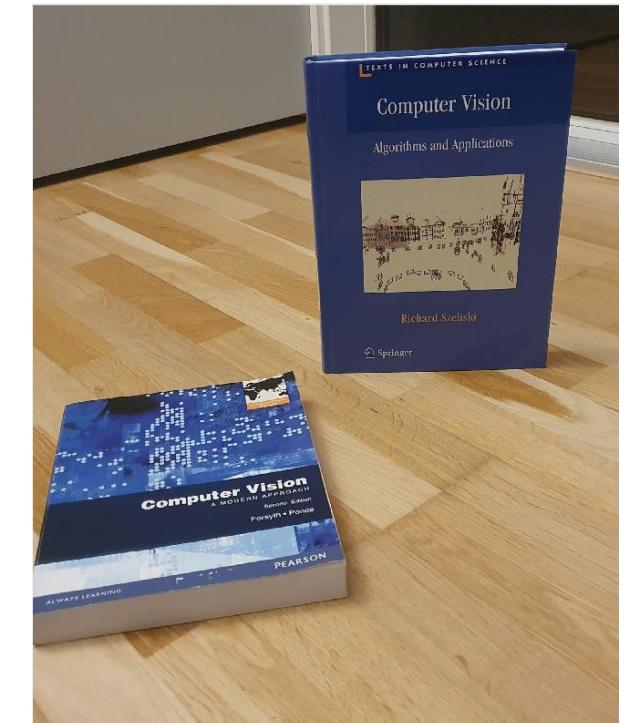
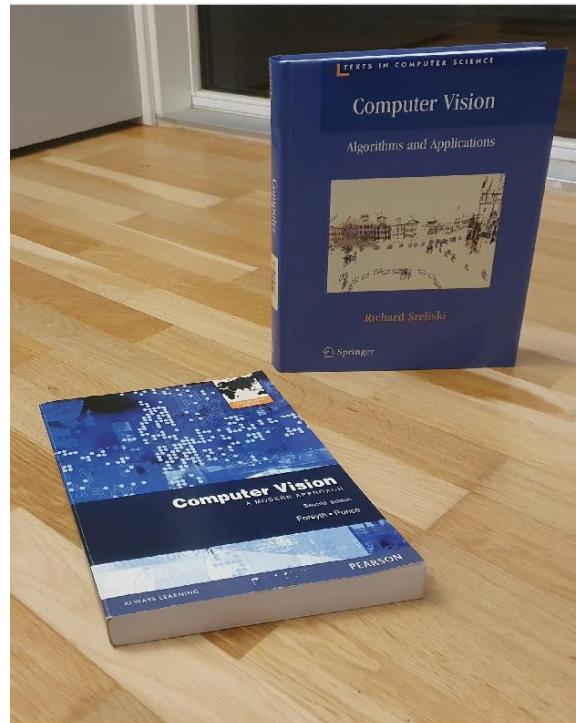


- Next, calculate the fundamental matrix



# An Example

- Finally calculate the rectified images



- Notice how the images are now scan line

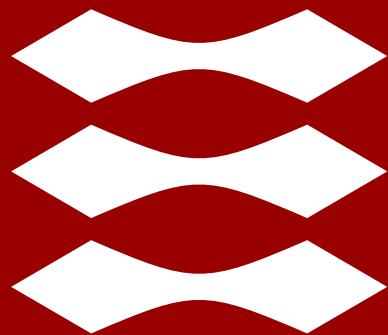


Perception for Autonomous Systems 31392:

# **Epipolar Geometry and the Fundamental Matrix**

Lecturer: Evangelos Boukas—PhD

**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?

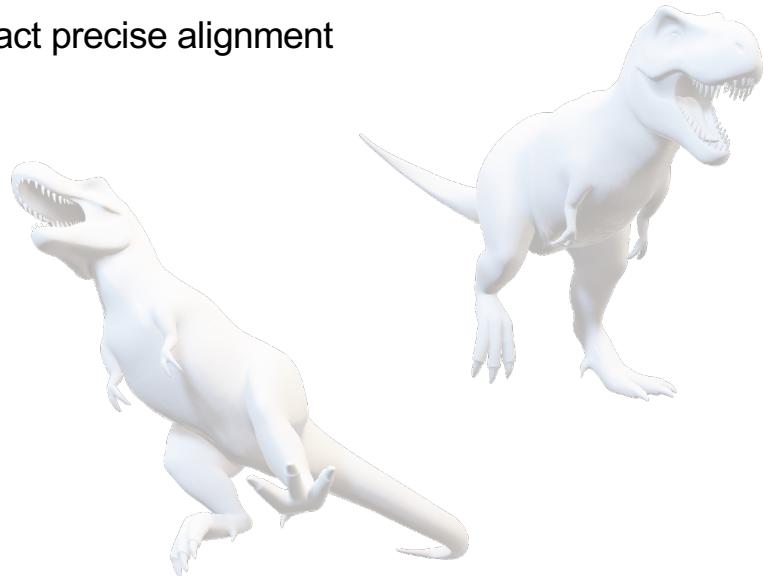
# Why is 3D Pose Estimation Important?

- 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

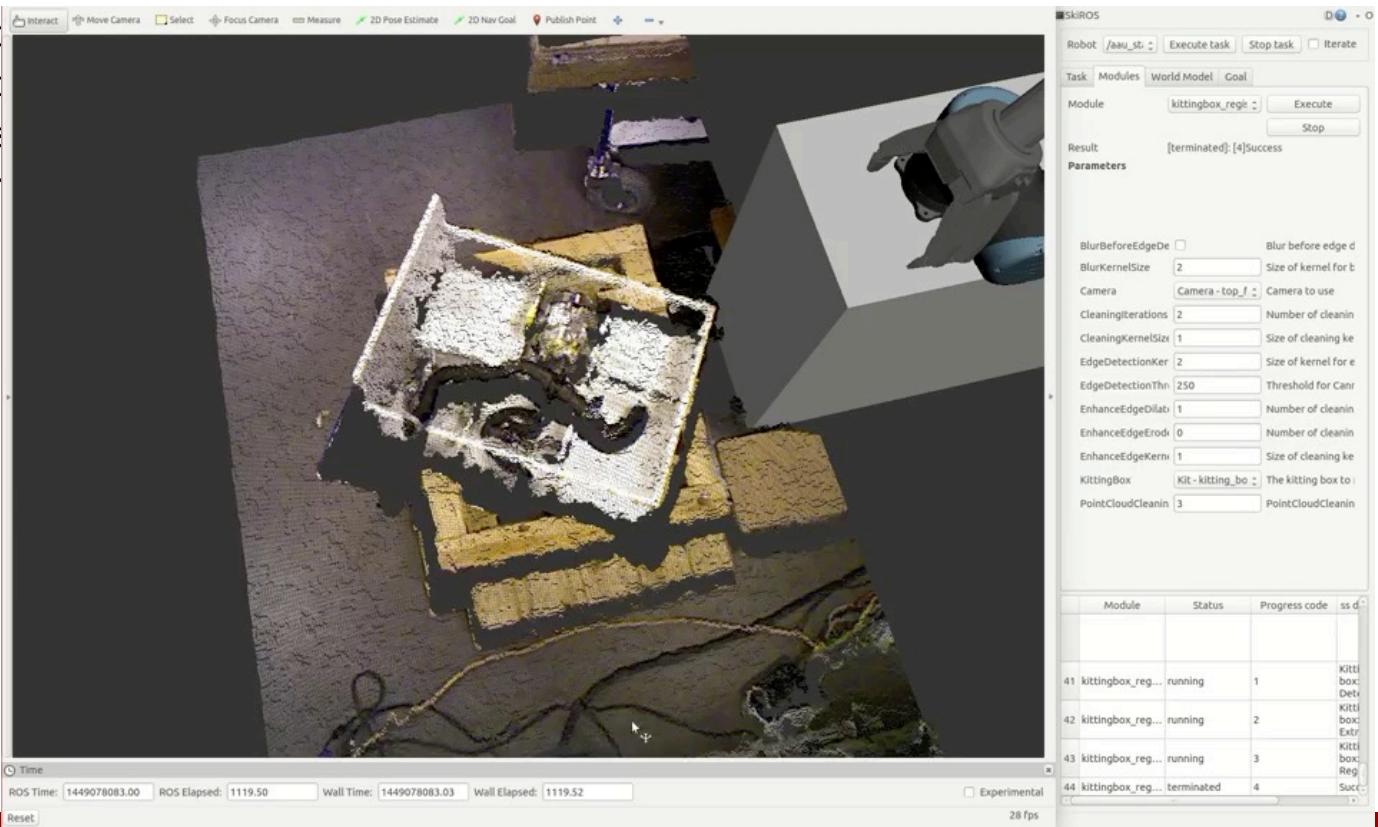
- Iterative Closest Point (ICP)
  1. For each object point  $p$  in model 1, find the nearest point  $q$  in model 2
  2. Use all pairs  $(p,q)$  to estimate the transformation from model 1 to model 2
  3. Apply the transformation to the points of Model 1
  4. Repeat steps 1-3 until convergence/stop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

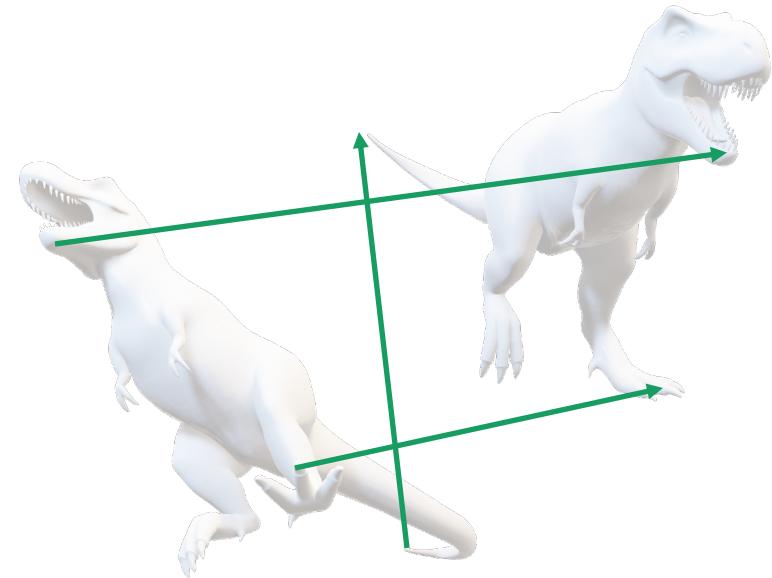
# Global Alignment

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



# Global Alignment

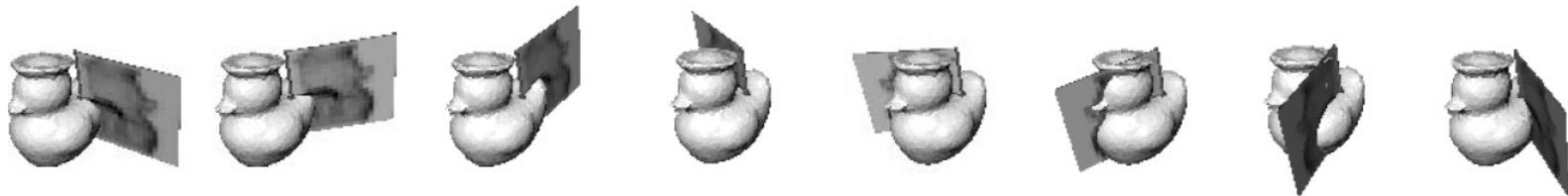
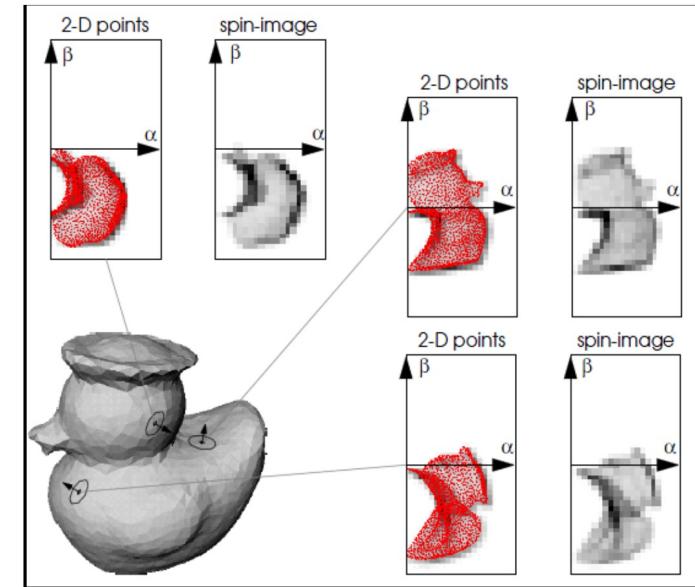
- Consider 2 models (point clouds) of the same object (fully or partially)
  - How can we “roughly” align them?
    - We cannot use ICP, if the initial poses are too different.
    - Can we use some kind of (3D) “features” and match them?



# 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.

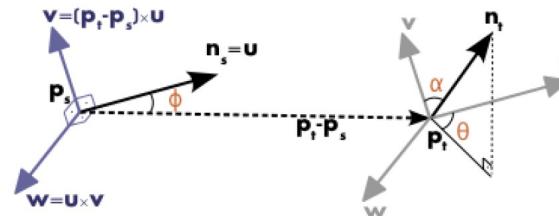
# Global Alignment – 3D Feature Descriptors

- PFH (Point Feature Histograms)

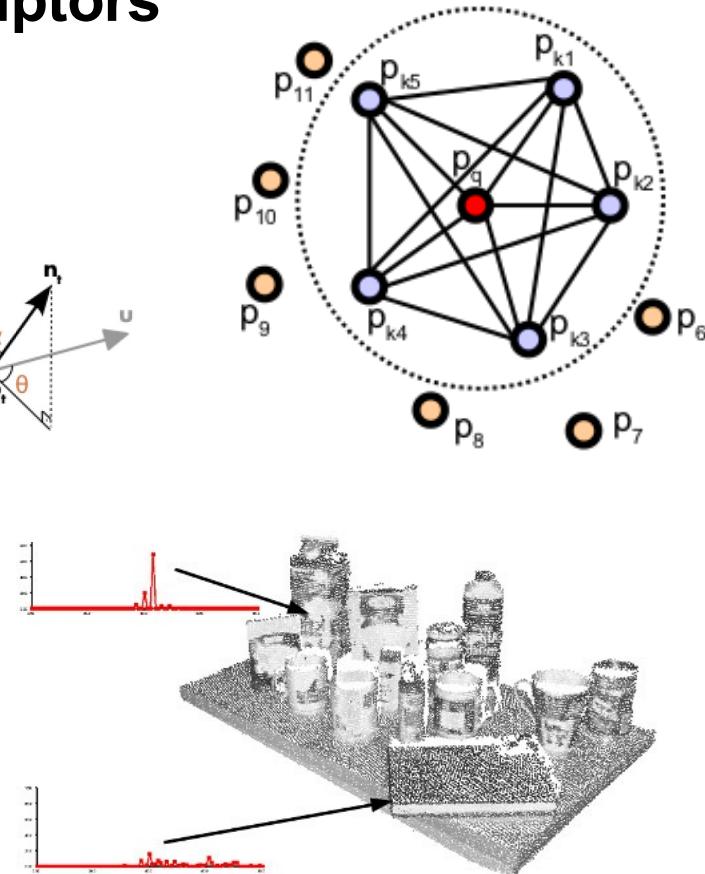
$$\alpha = \arccos(v \cdot n_t)$$

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

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



- 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

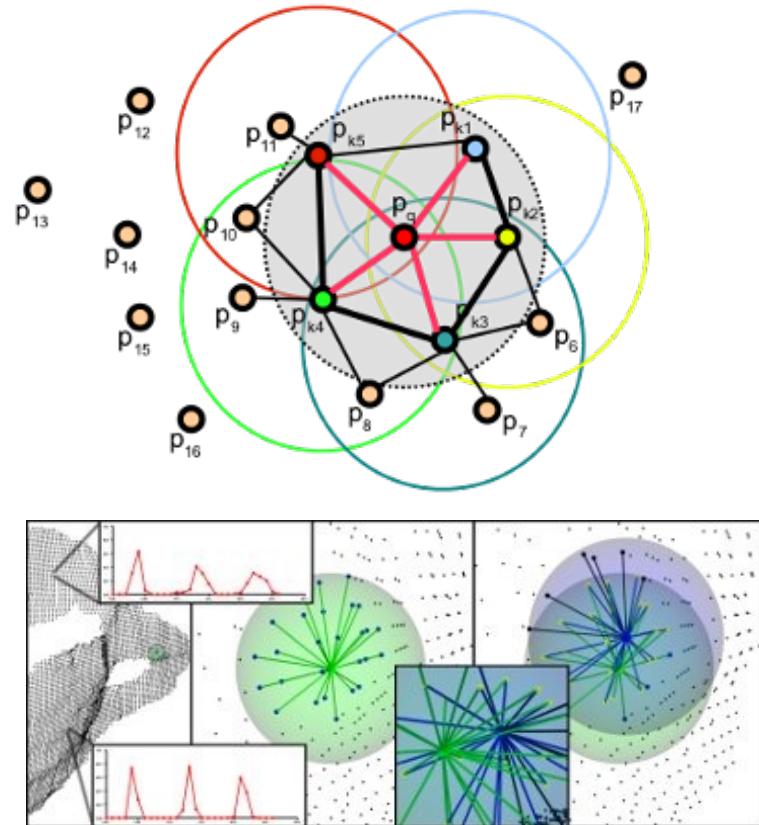


R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.

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

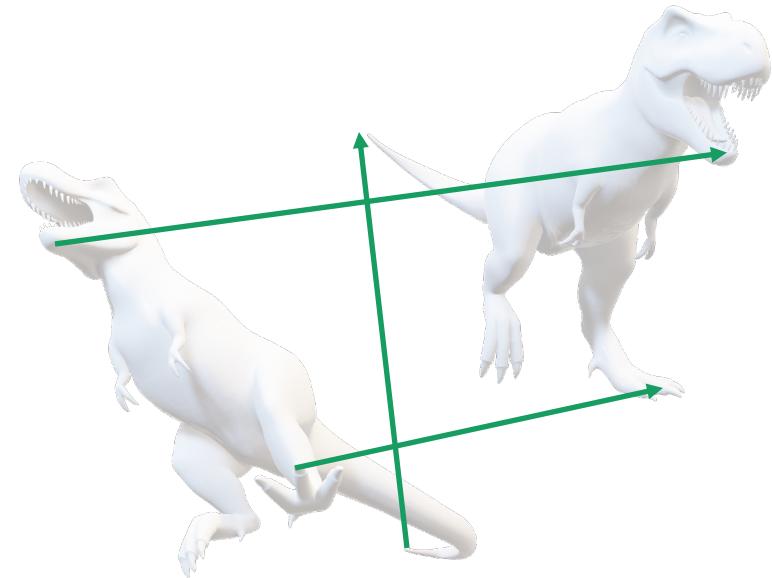


R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 3212-3217.

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.

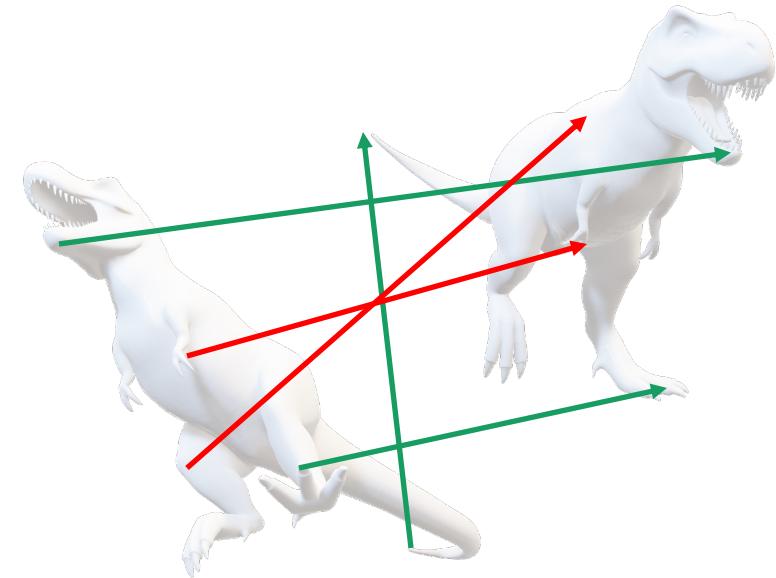
# Global Alignment

- 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?



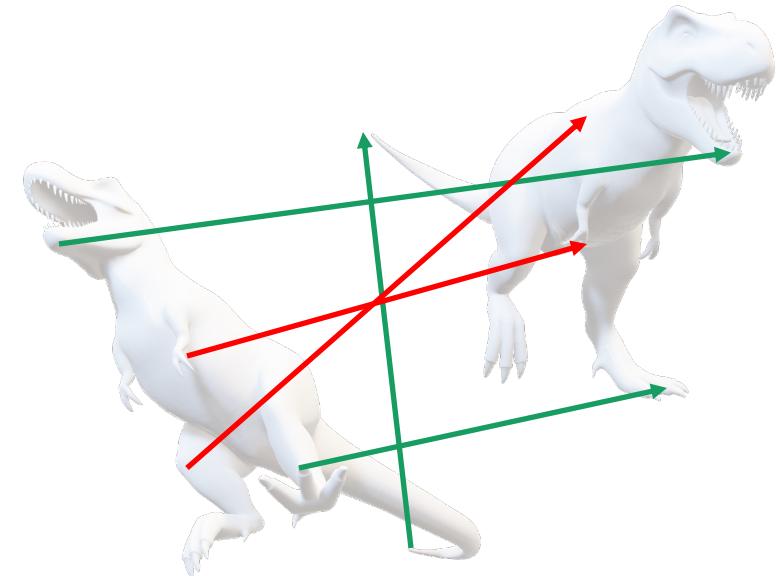
# Global Alignment

- 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.



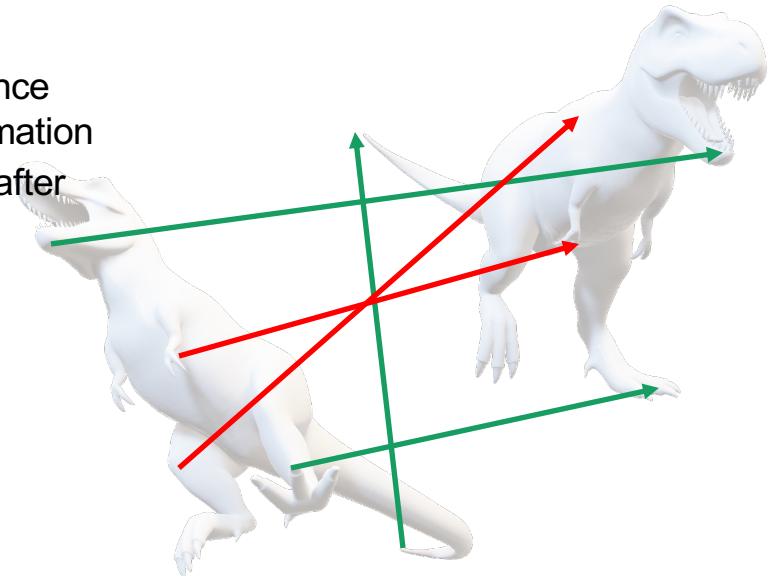
# Global Alignment – RANSAC in 3D

- 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

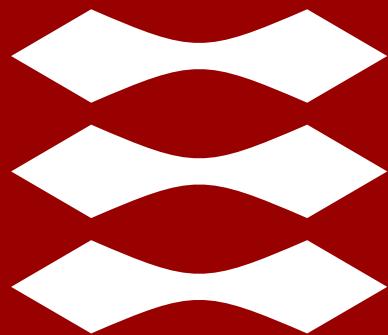
# Summary

- 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

**DTU**



Lazaros Nalpantidis

# 3D Point Cloud Processing - Clustering & Regression

# Outline

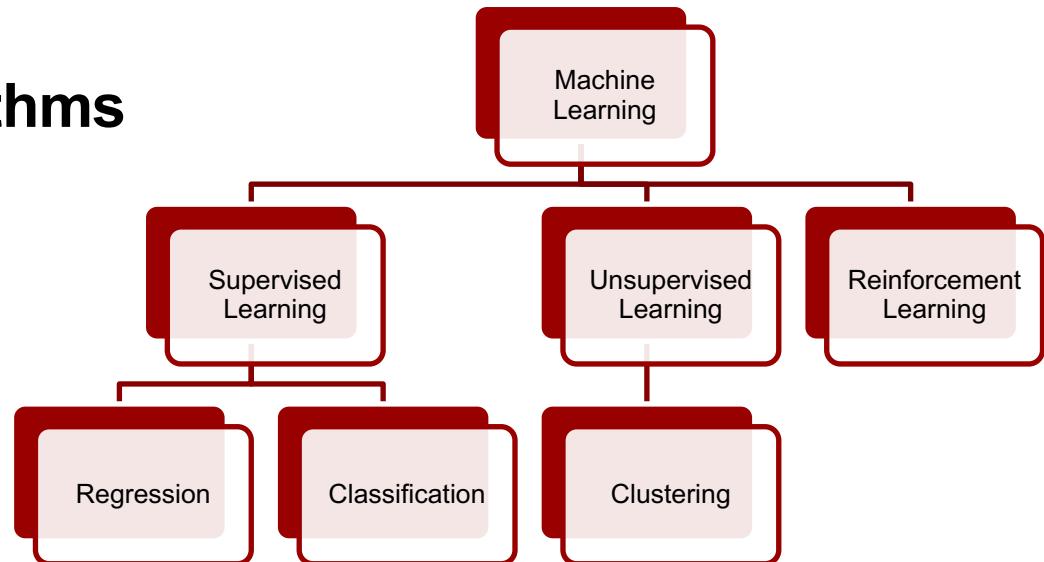
- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
  - k-means
  - Mean Shift
  - DBSCAN
  - Hierarchical Clustering
- Summary

# 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

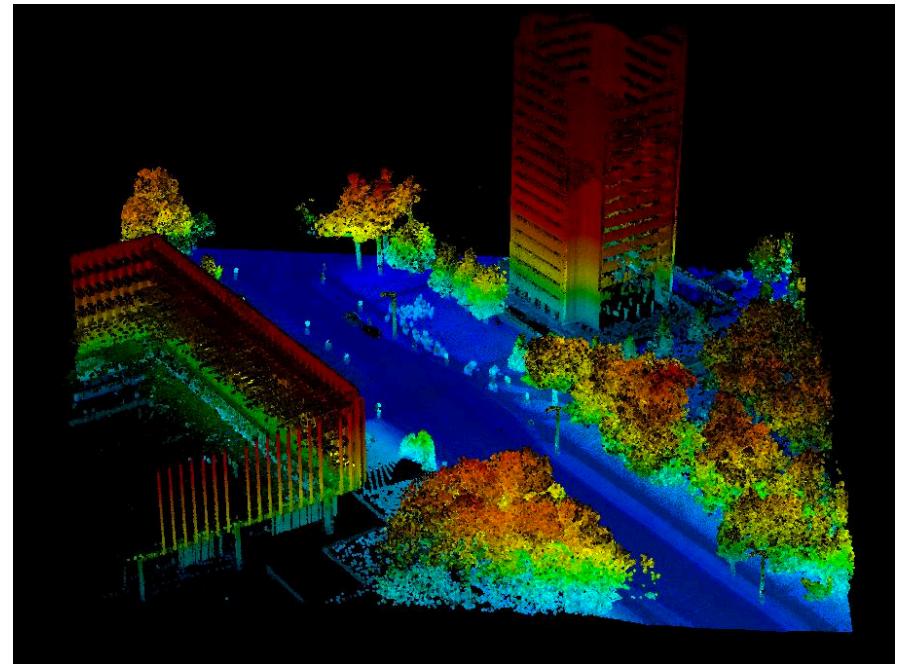


- **Supervised Learning** – The target values are known
  - Regression – The target value is numeric
  - Classification – The target value is nominal
- **Unsupervised Learning** – The target values are unknown
  - Clustering – Group together similar instances
- **Reinforcement Learning** – Interacting 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?

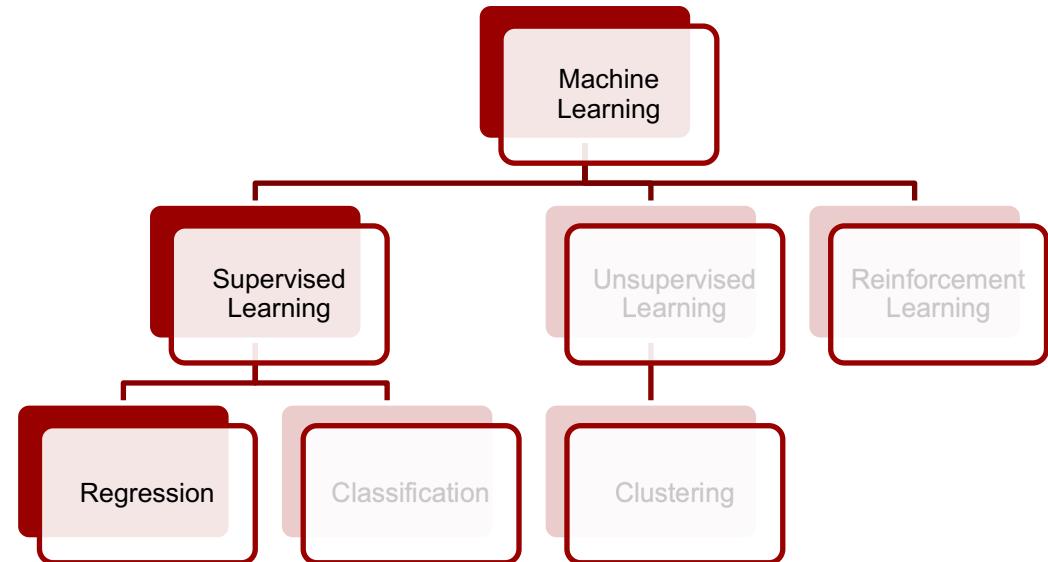
- Create initial object hypotheses
  - input to pose estimation?
- Abstract representation of scene
- ...what else?



# 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

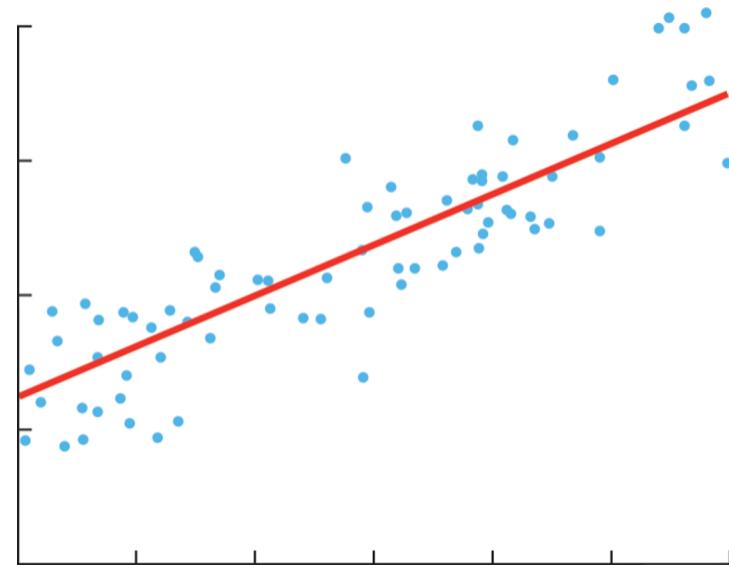
# Regression



- **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.

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

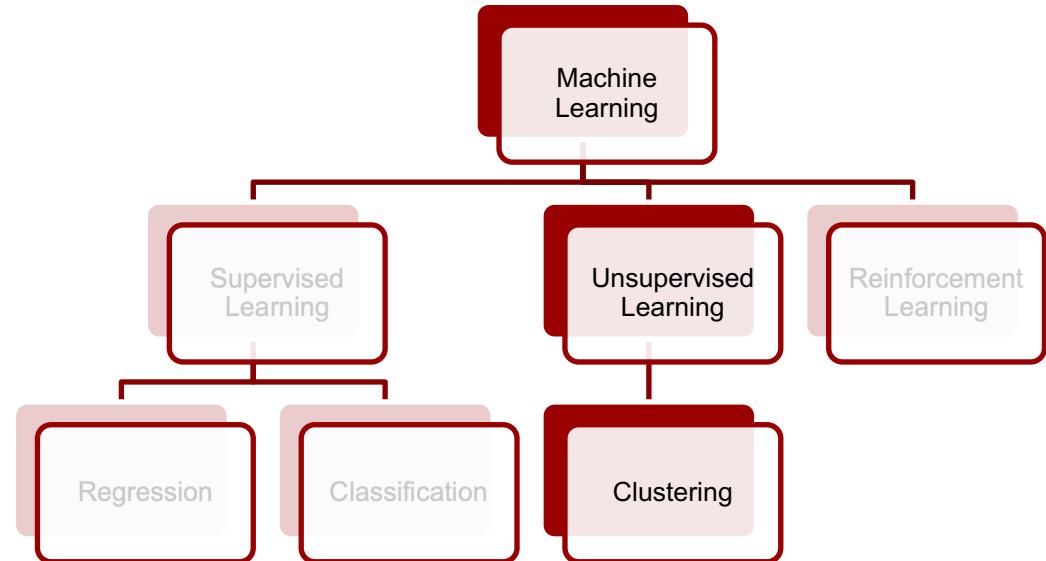
- The degree of polynomial can be found by cross-validation.

# 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?

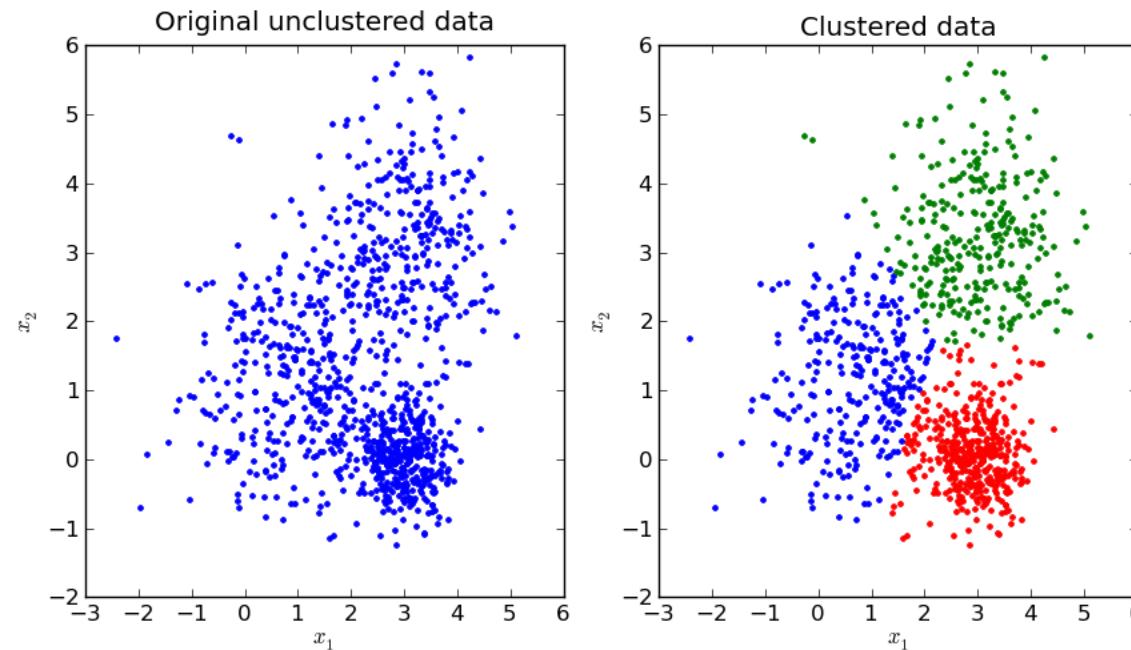
- What are Clustering and Regression? A taxonomy of ML algorithms
- Why is ML important for Perception of Autonomous Systems?
- Regression
- Clustering
  - k-means
  - Mean Shift
  - DBSCAN
  - Hierarchical Clustering
- Summary

# 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.



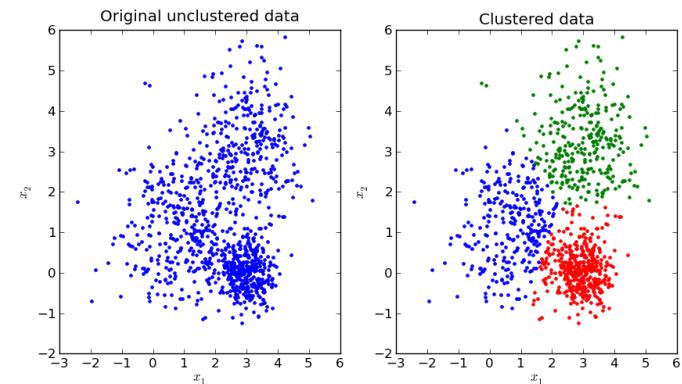
# Clustering – k-means

- 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  $n^{\text{th}}$  data point assigned to cluster  $k$  and  $\mu_k$  is the centroid the cluster  $k$ .  
The function  $d()$  is the Euclidean distance between  $x$  and  $\mu$ .

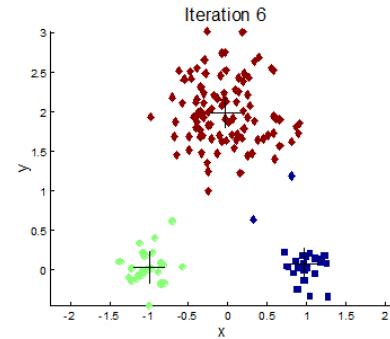
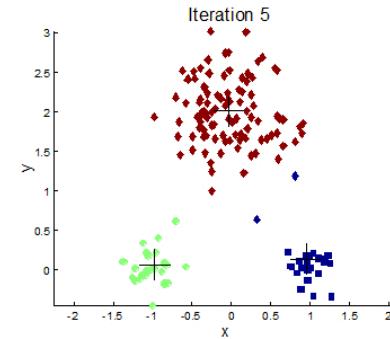
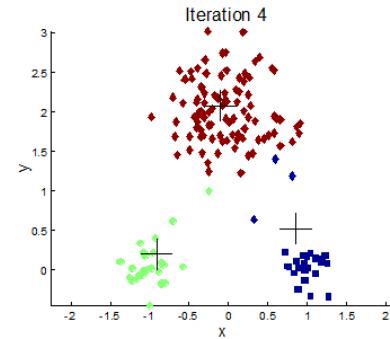
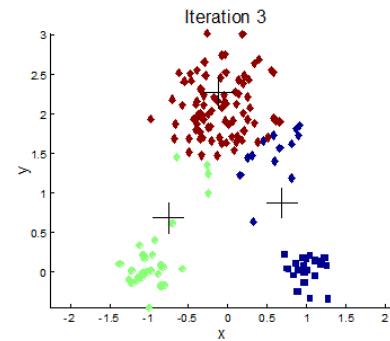
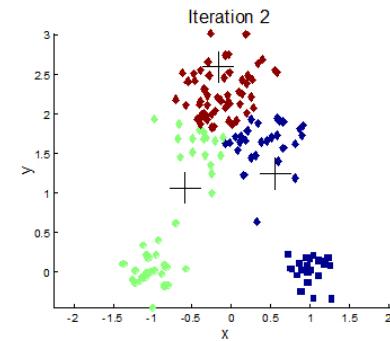
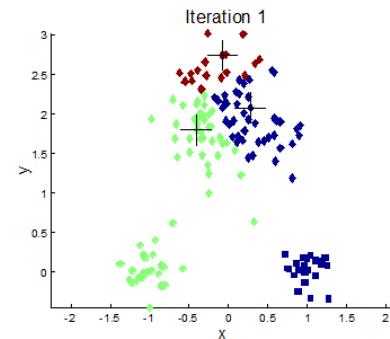
- **Important:** K-means requires known number of clusters  $k$



# Clustering – k-means

It is an iterative process:

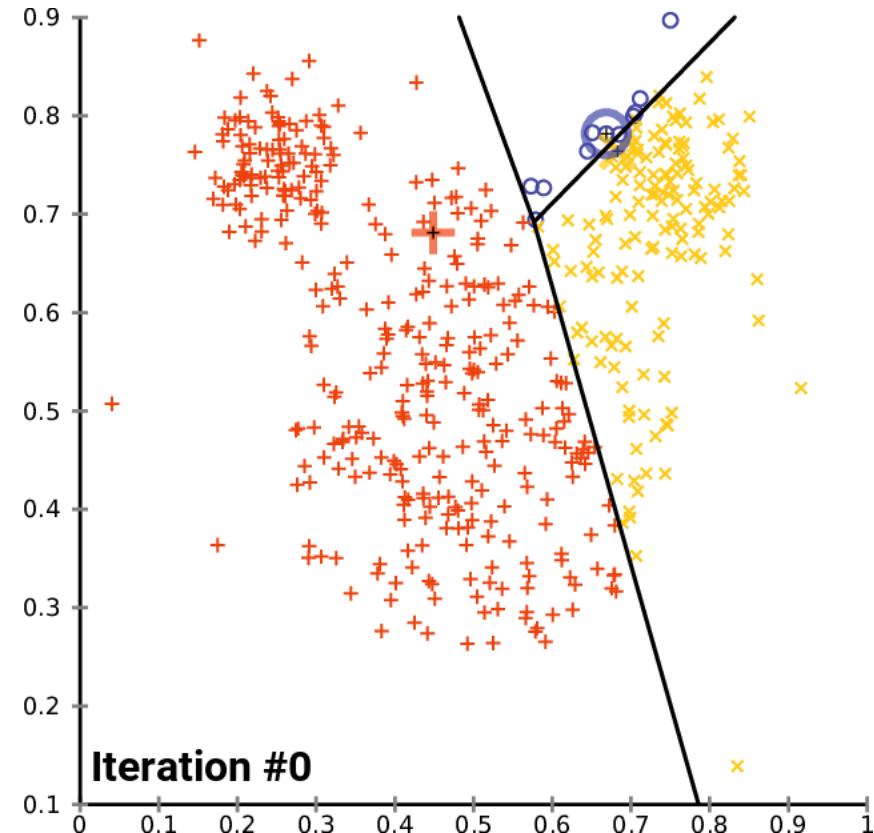
- Step 1:
  - Randomly initialize k cluster centers
- Step 2:
  - Assign each to its nearest cluster center
- Step 3:
  - Re-compute each cluster-center as the centroid of all points assigned to each cluster
- Step 4:
  - Check convergence/stop criterion, otherwise repeat Steps 2-4



# 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

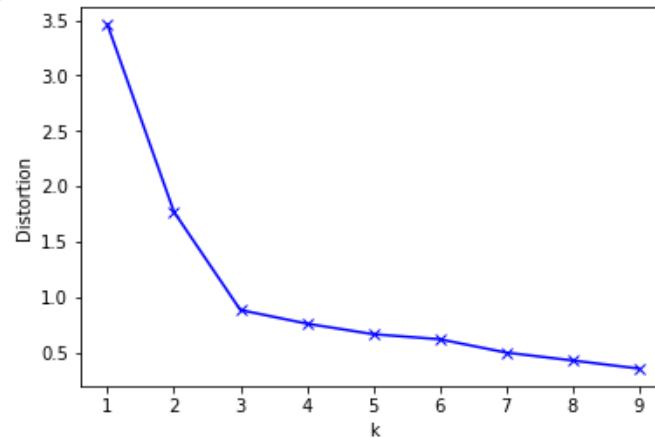


# Clustering – k-means

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

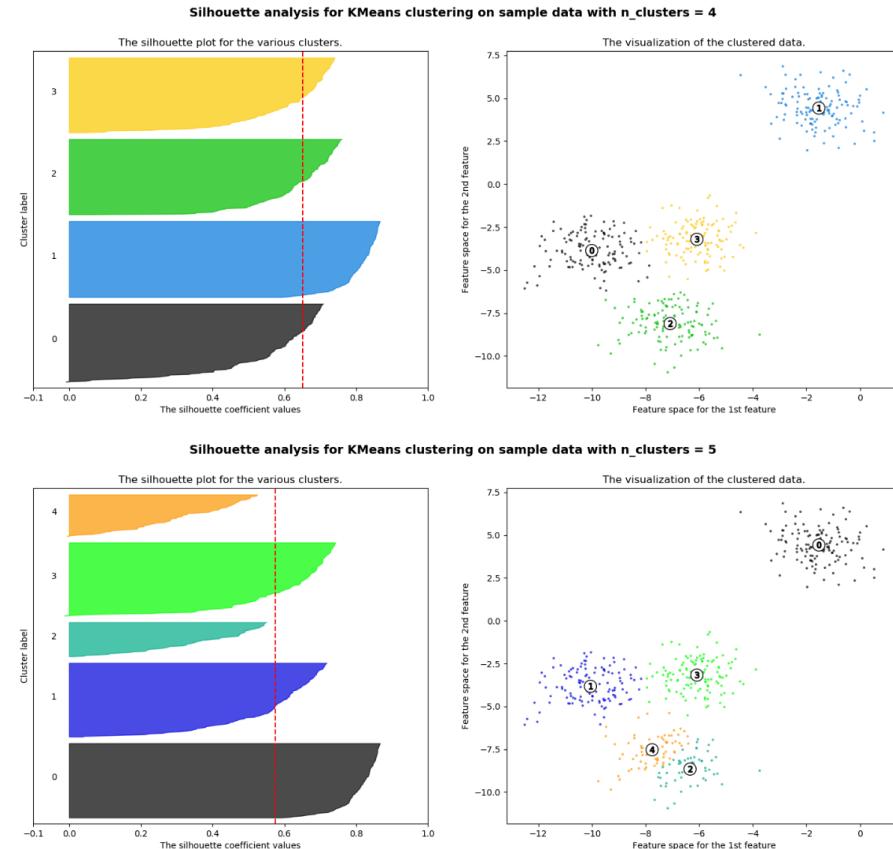


# Clustering – k-means

How to define  $k$  ?

## – Silhouette Analysis

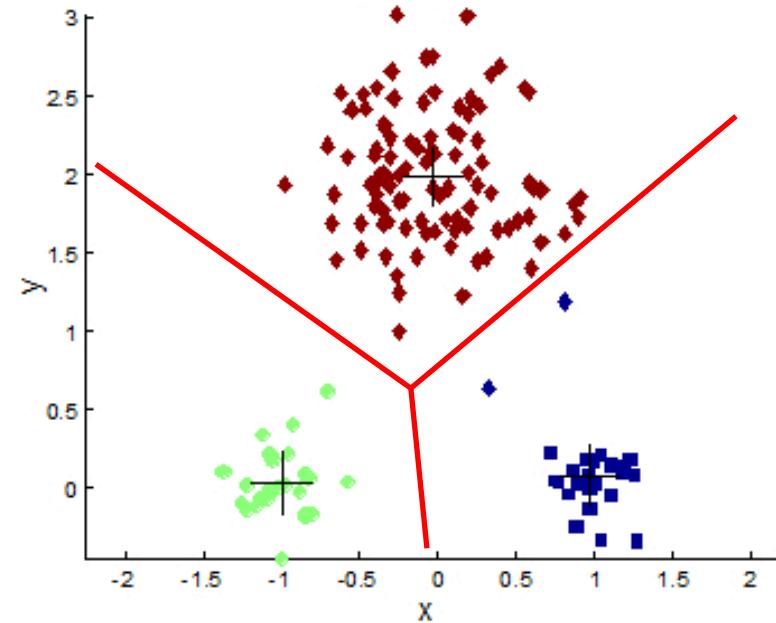
- Calculate Silhouettes for various  $k$ 
  - Silhouette coefficient shows how far each sample is from other cluster centers
    - » +1: far from others
    - » 0: at the boundary
    - » -1: sample is on average closer to the points of another cluster
- Thickness shows cluster size (number of points 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

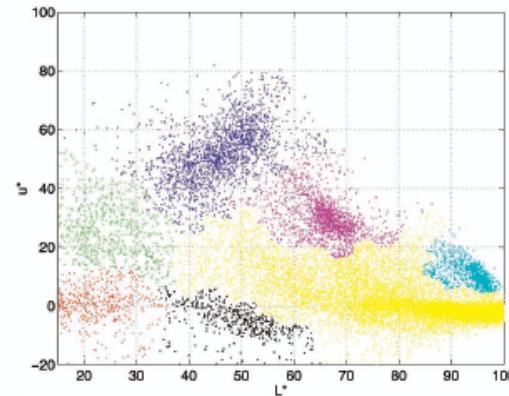
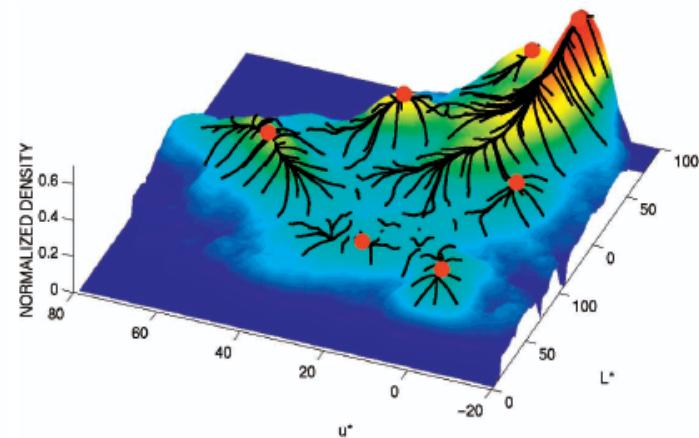
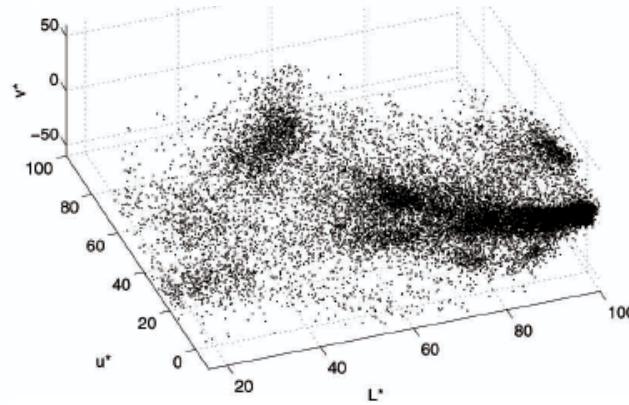
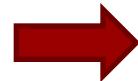
- **Mean Shift** is an algorithm that finds the maxima—the modes—of a density function given discrete data sampled from that function. Thus, it is a density hill climbing algorithm.
  - Every point gives rise to a cluster.
  - Each cluster is defined by the radius, a.k.a. “bandwidth”,  $h$  of its region, 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



# Kernel density estimation

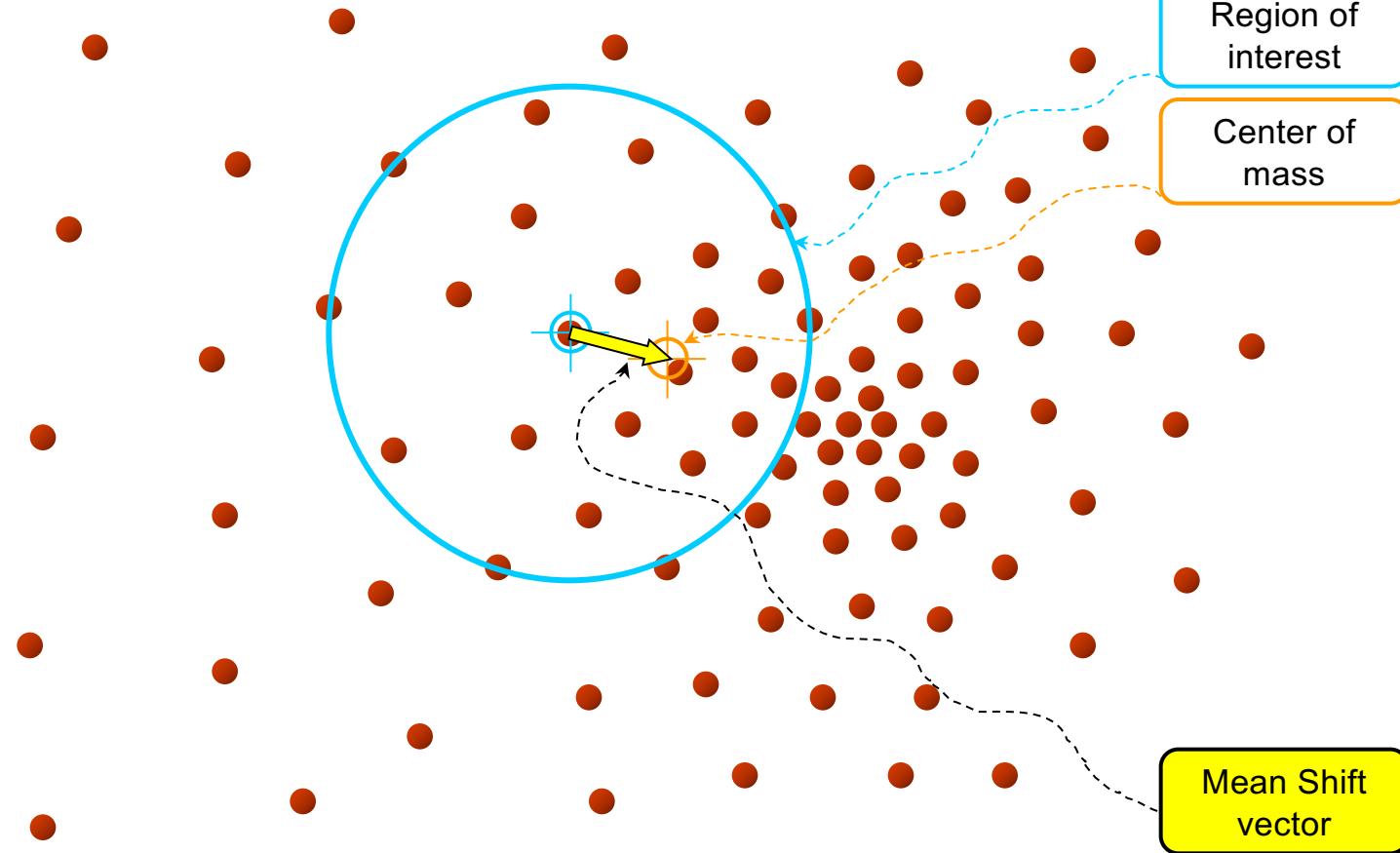
Kernel density estimation function

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

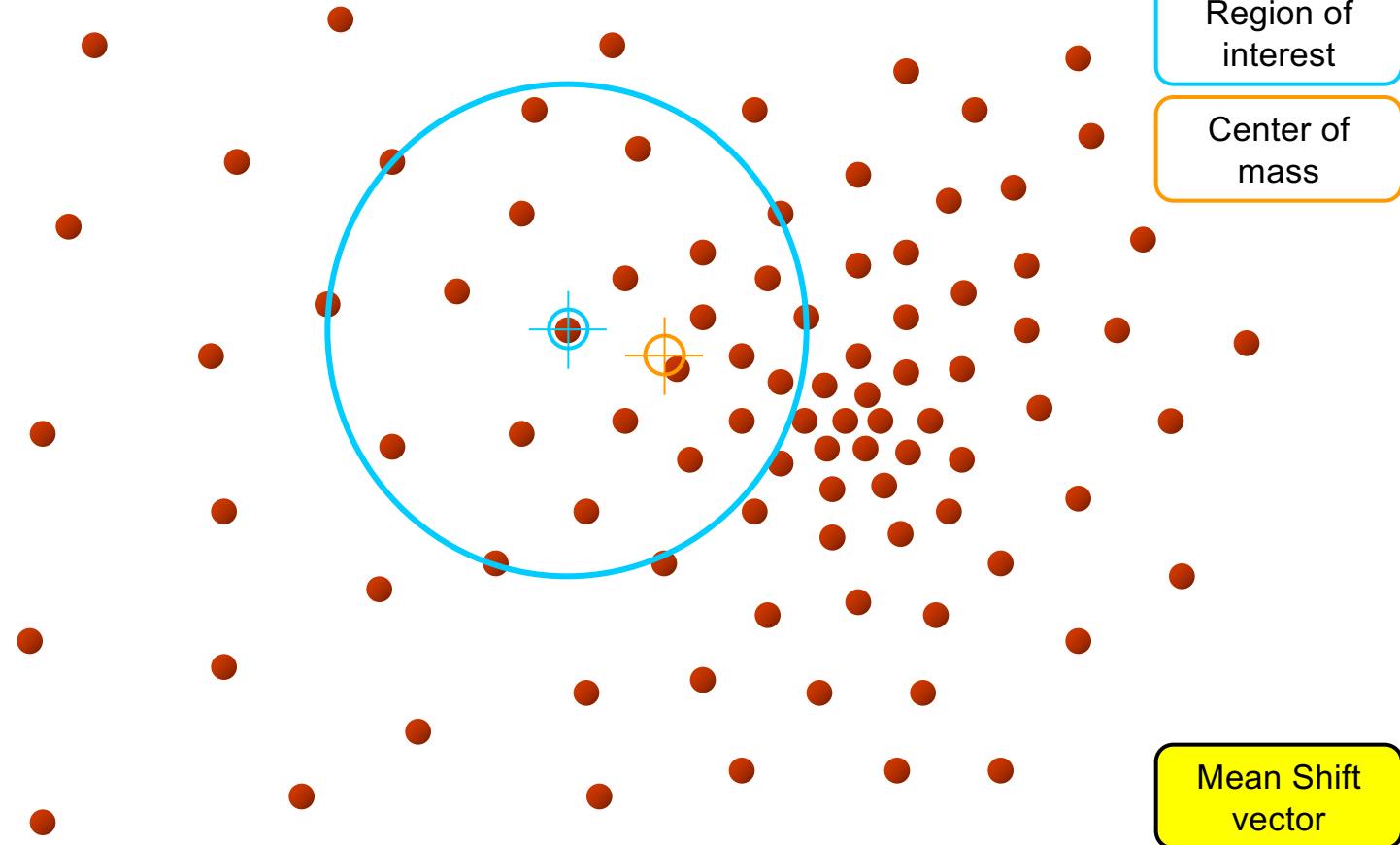
Gaussian kernel (typically used)

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

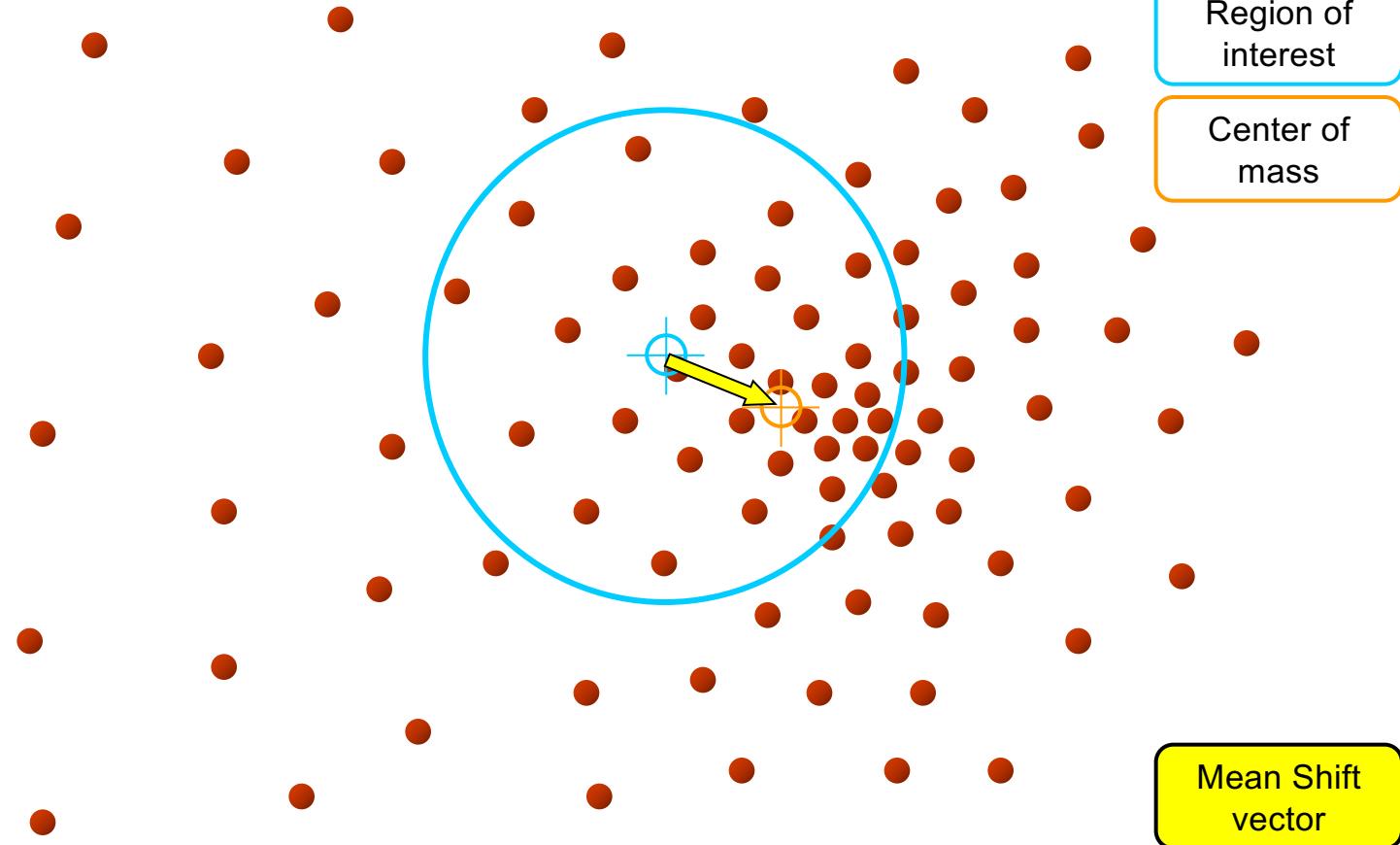
# Mean shift



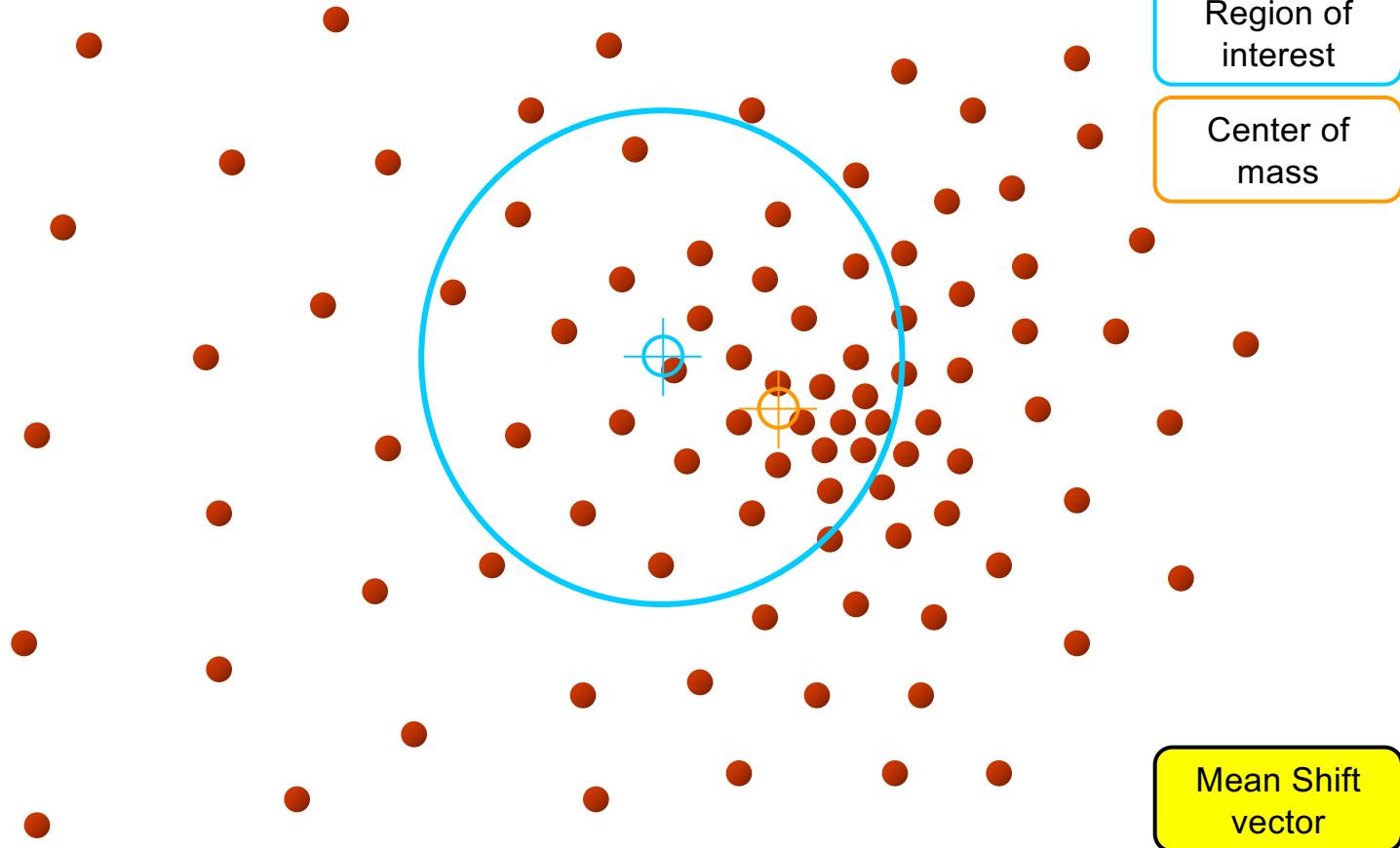
# Mean shift



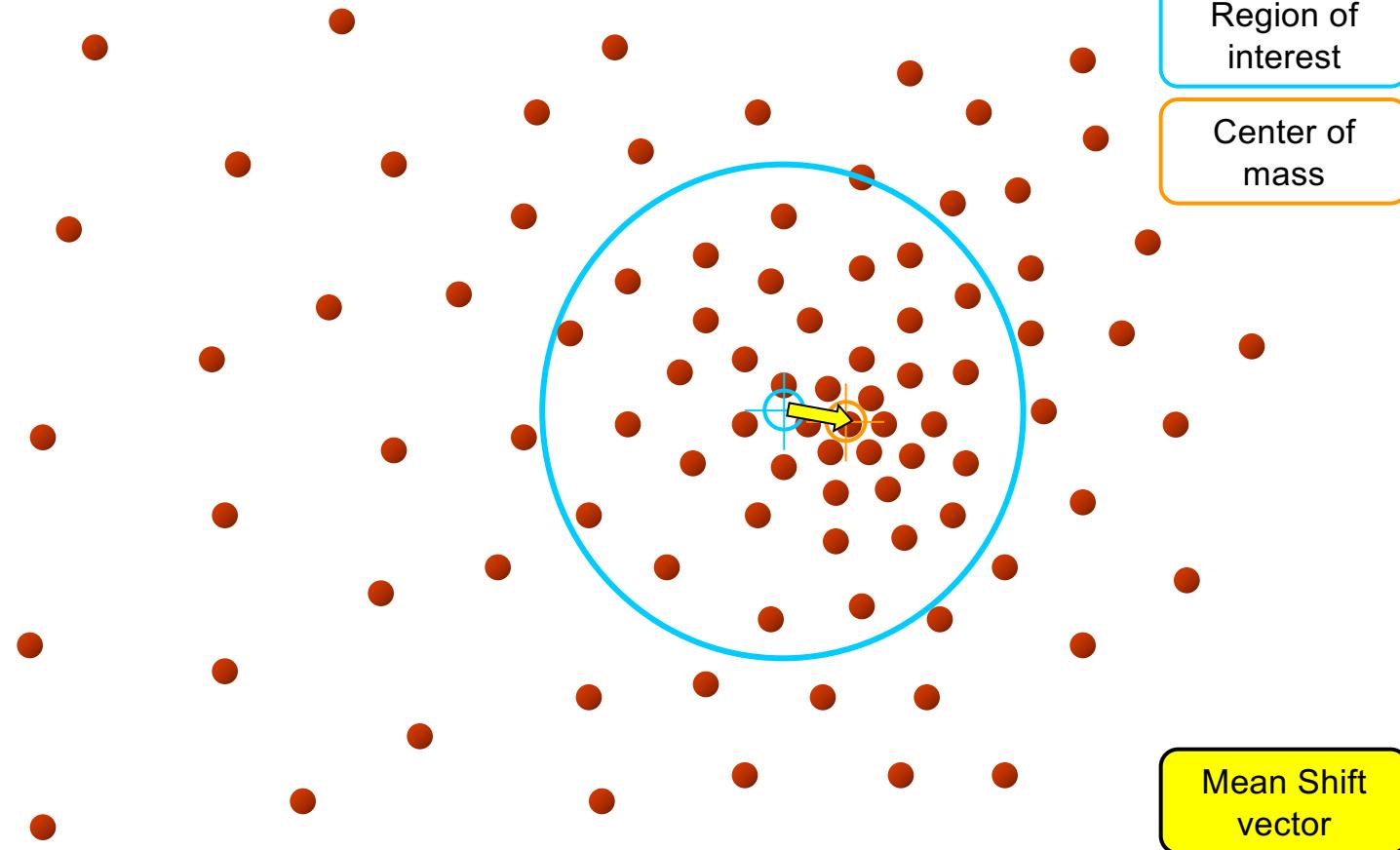
# Mean shift



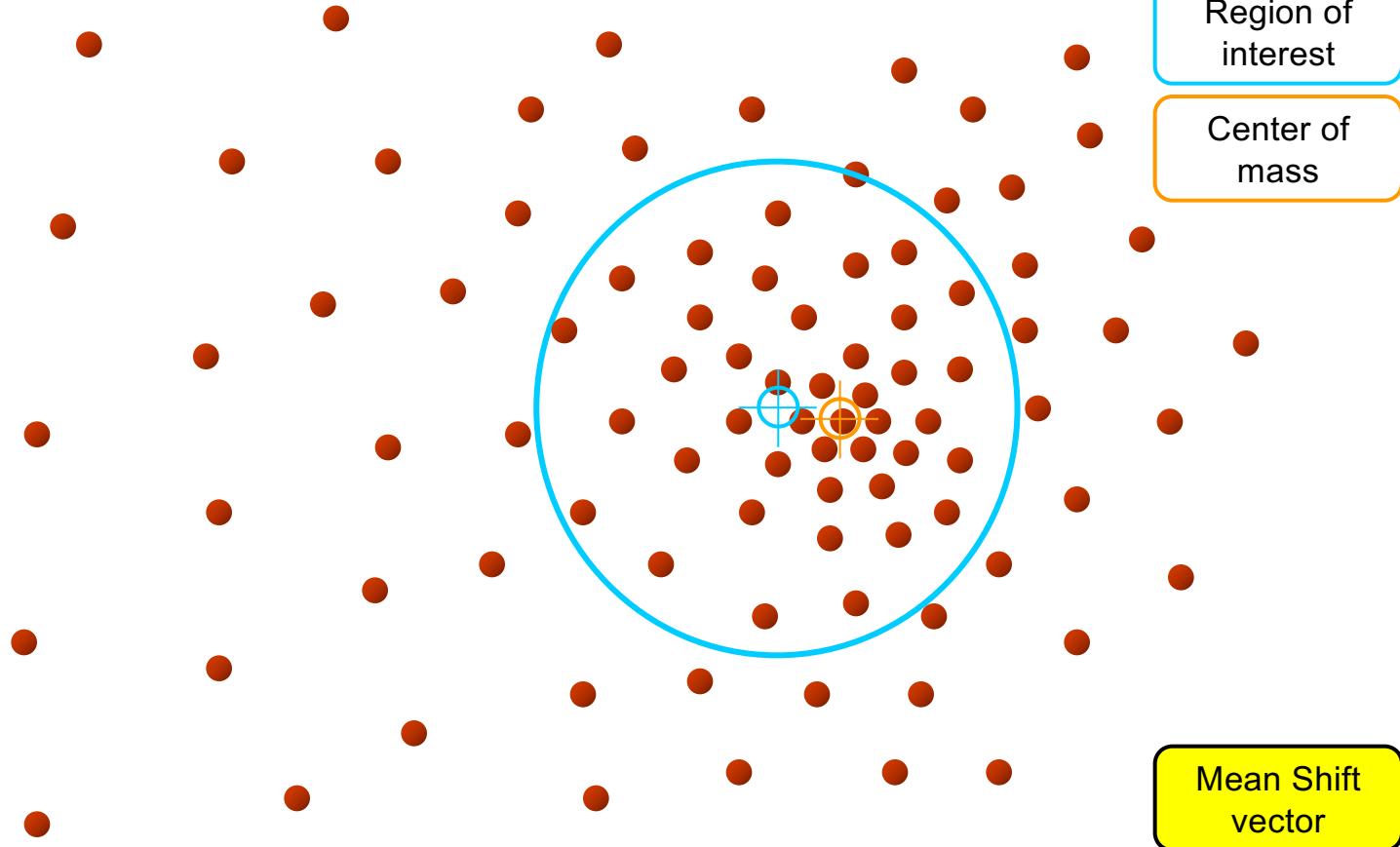
# Mean shift



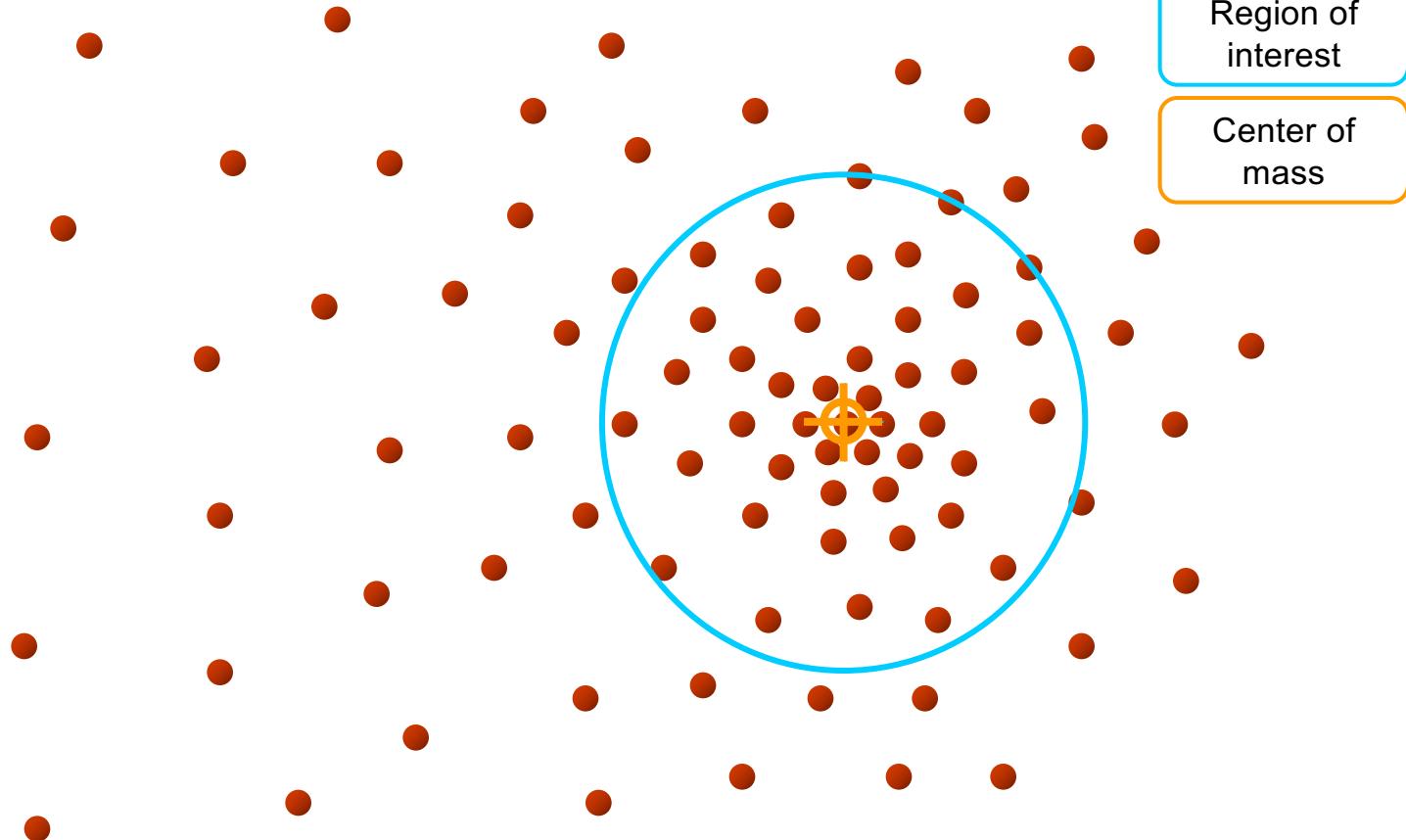
# Mean shift



# Mean shift

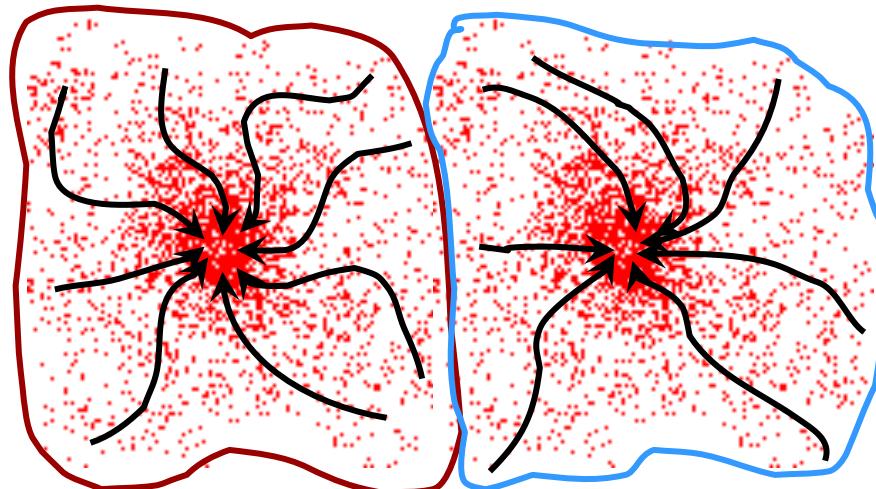


# Mean shift

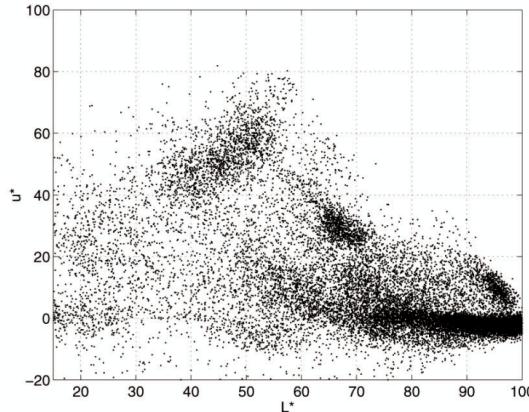


# Attraction basin

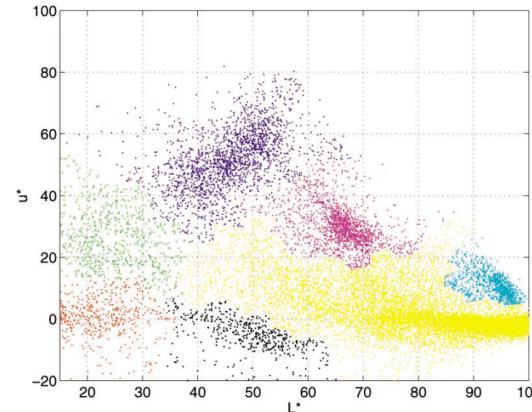
- Attraction basin: the region for which all trajectories lead to the same mode
- Cluster: all data points in the attraction basin of a mode



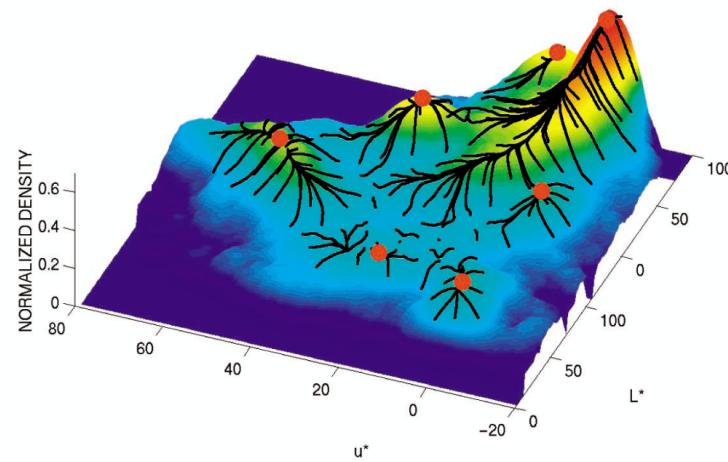
# Attraction basin



(a)



(b)



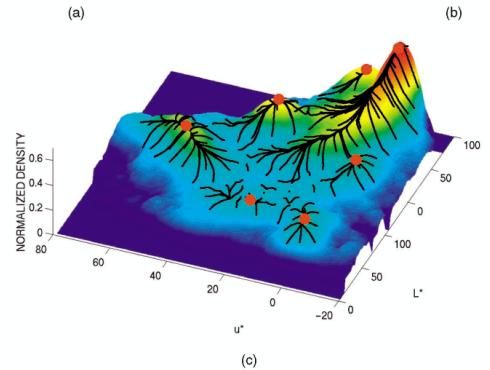
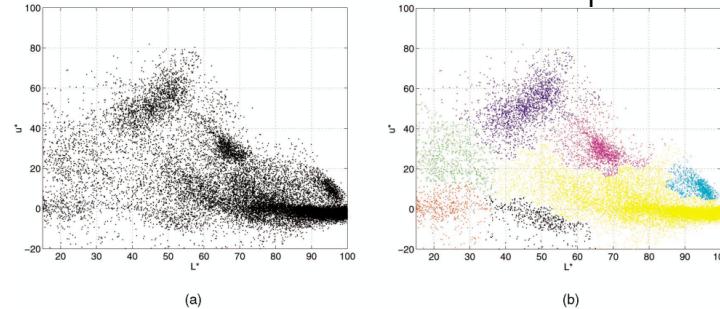
# Mean shift clustering

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  $\varepsilon$
  - A region is dense if the circle/sphere of radius  $\varepsilon$  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:

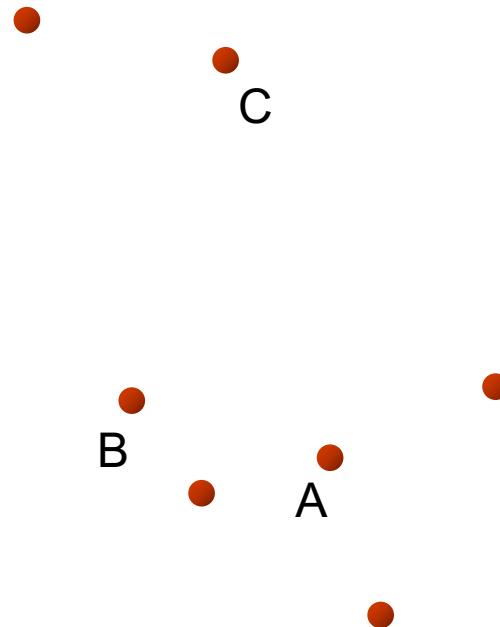
- A **core point** has more than a specified number of points ( $MinPts$ ) within  $\varepsilon$
- A **border point** has fewer than  $MinPts$  within  $\varepsilon$ , but is in the neighborhood of a core point.
- A **noise point** is any point that is not a core point or a border point.

# Clustering - DBSCAN

Assume:

$$\varepsilon=1$$

$$MinPts=4$$



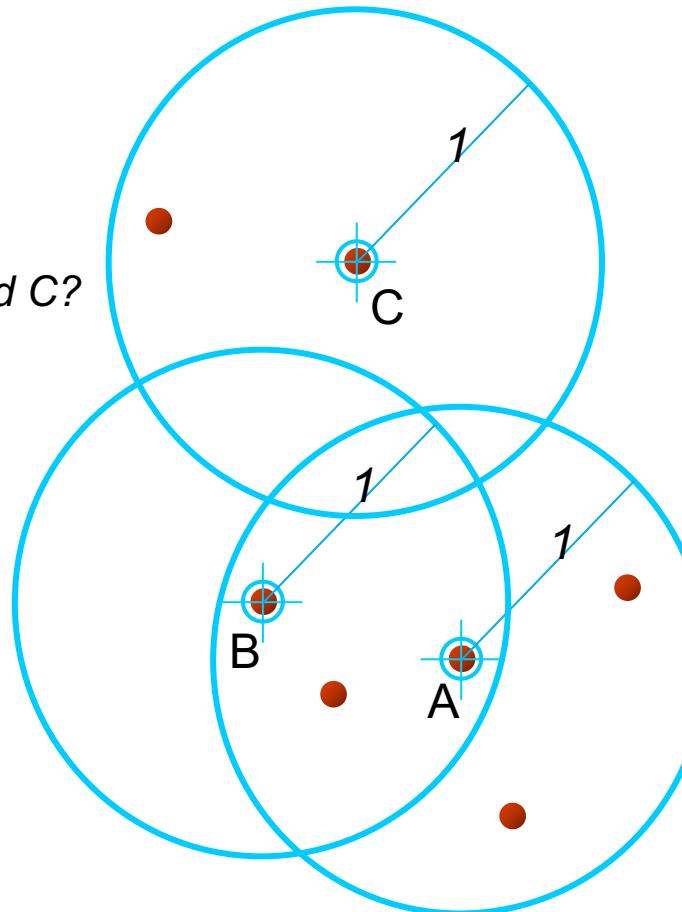
# Clustering - DBSCAN

Assume:

$$\varepsilon=1$$

$$\text{MinPts}=4$$

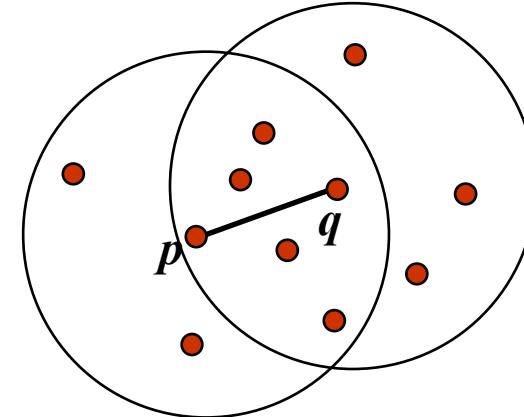
What is the type of points A, B and C?



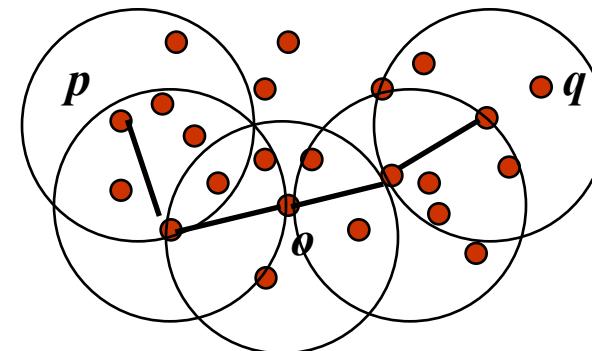
# Clustering - DBSCAN

Some more definitions:

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



- **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:

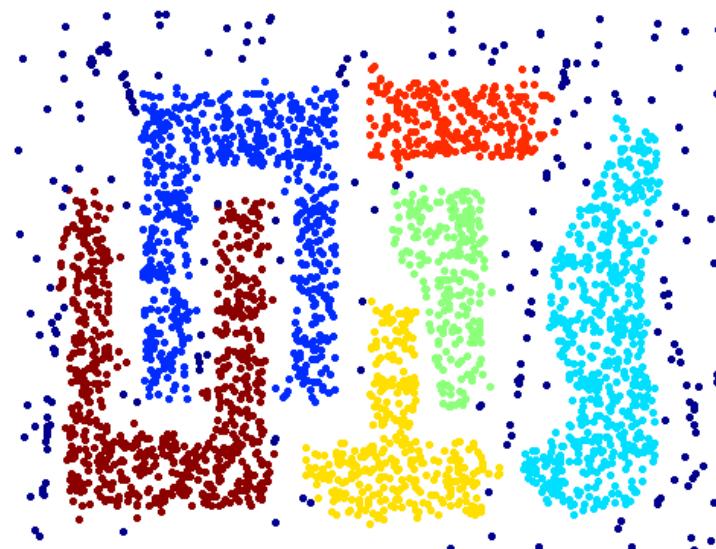
- all core points that can be reached by following a sequence of density-connected core points
- border points that are closest to one of the above-clustered core points

# 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

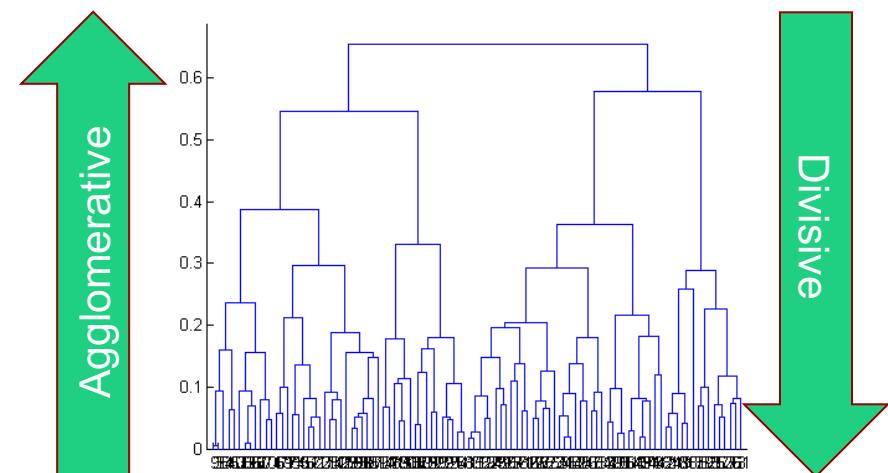
## Cons

- Some points (noise points) are not assigned to any cluster (is this necessarily bad?)
- Need to define  $\epsilon$  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:

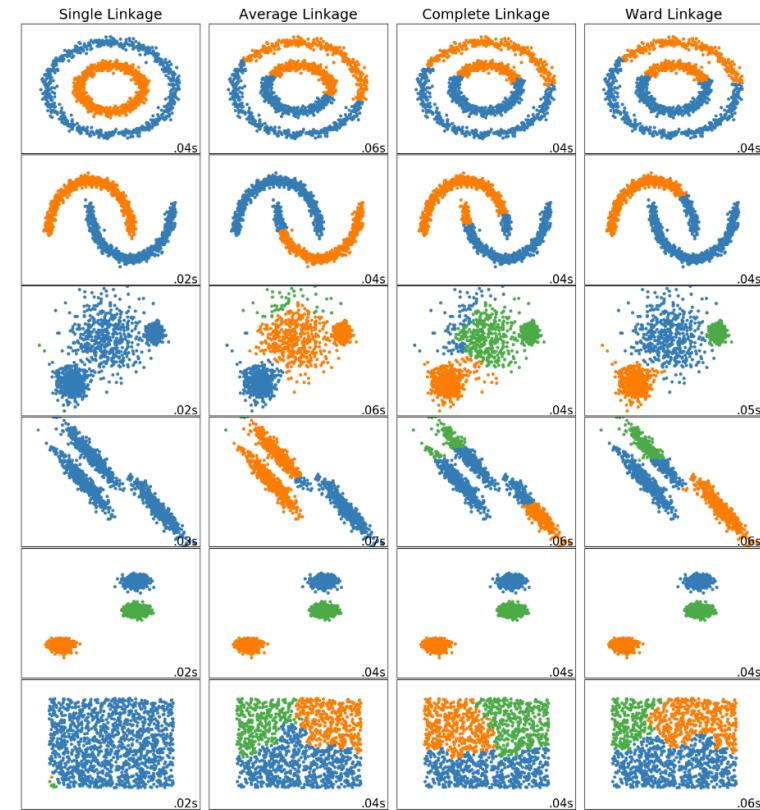
1. Start by assigning each point to its own cluster, obtaining as many clusters as points.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until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

Agglomerative clustering is much more popular than Divisive hierarchical clustering.

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

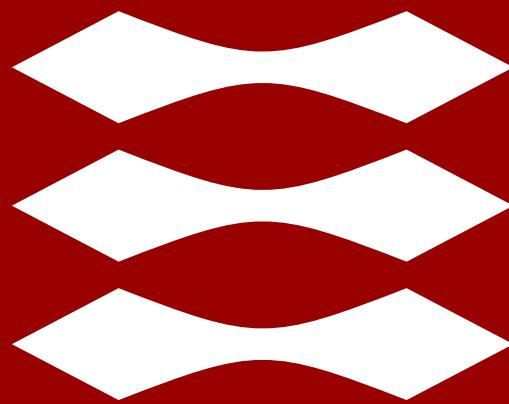
# Summary

- 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

**DTU**



Perception for Autonomous Systems 31392:

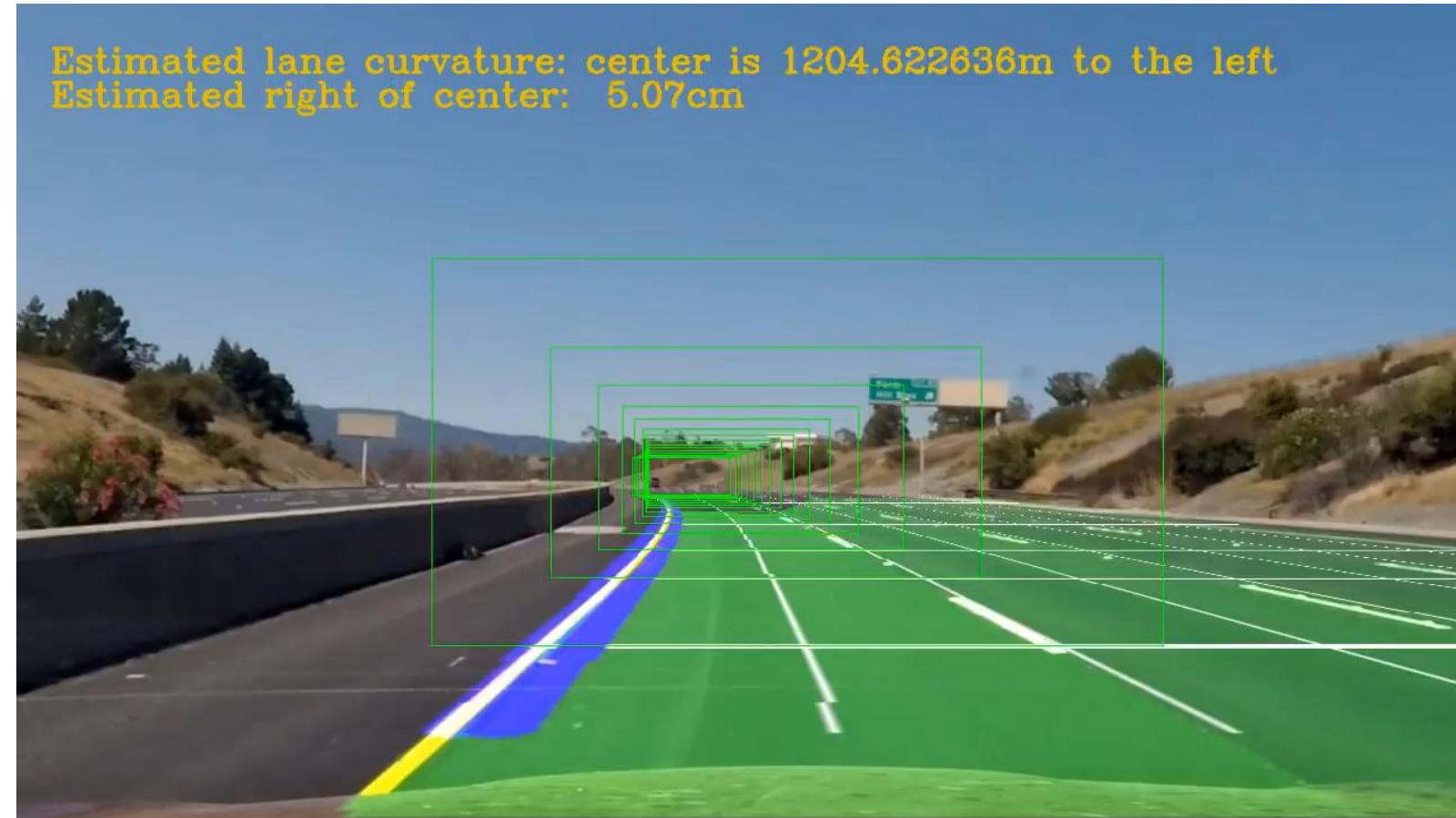
# **State Estimation - Histogram Filter**

Lecturer: Evangelos Boukas—PhD

# Cases of State Estimation



# Cases of State Estimation



# Cases of State Estimation



# What is State Estimation

Goal:

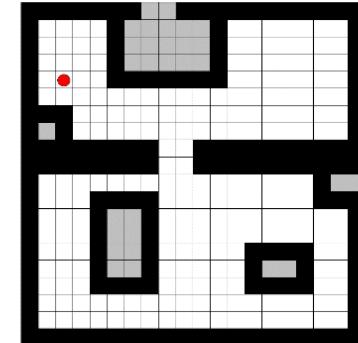
- Given a State Vector of a system
- Estimate over time the state using input of external sensors

Useful for:

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

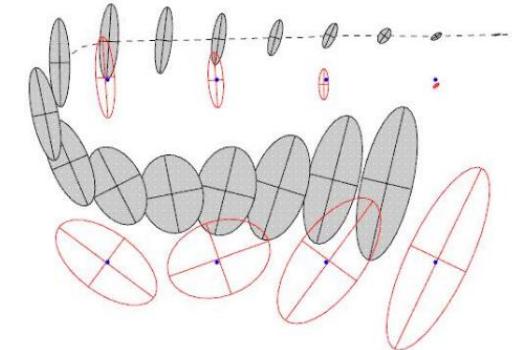
# Continuous vs Discrete State Unimodal vs Multimodal Distribution

- State:

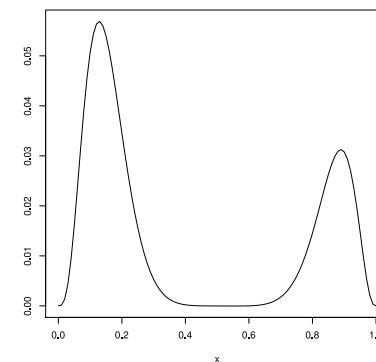


Discrete

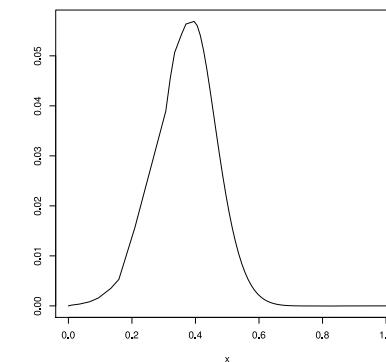
- Distribution



Continuous



Multimodal



Unimodal

# Simple State Estimation Example

- Assume a robot in an area like this one:

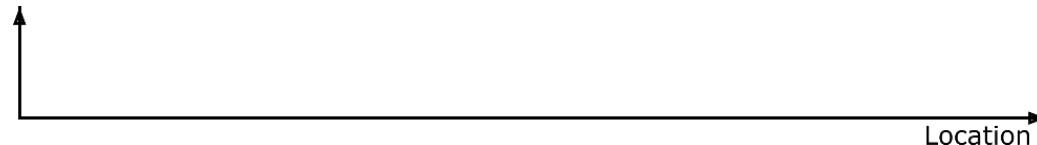


# Simple State Estimation Example

- 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:



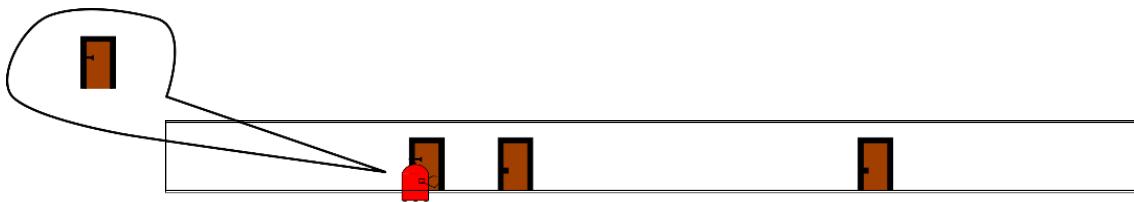
- We model the position of the robot as follows:



- 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:

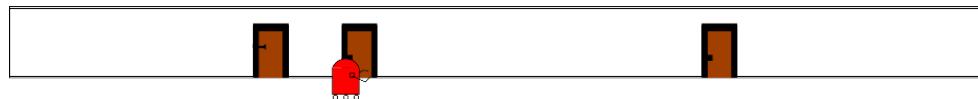


- The robot can distinguish (sense) a door vs "no-door":



# 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":

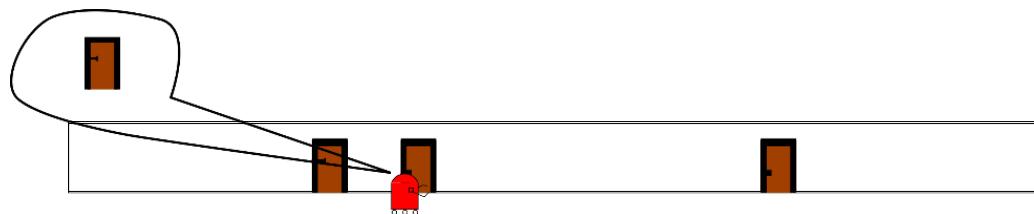


- The robot moves to the right:



# 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”:



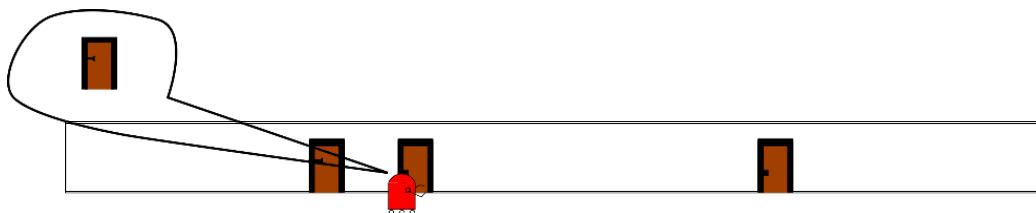
- The robot moves to the right:



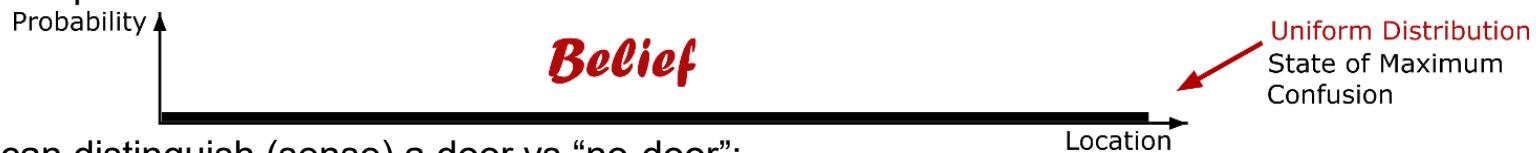
- The robot senses again:

# Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



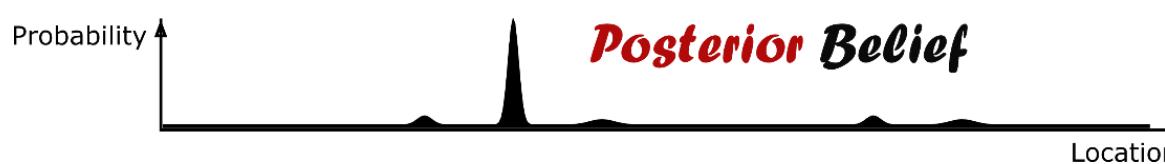
- The robot can distinguish (sense) a door vs “no-door”:



- The robot moves to the right:

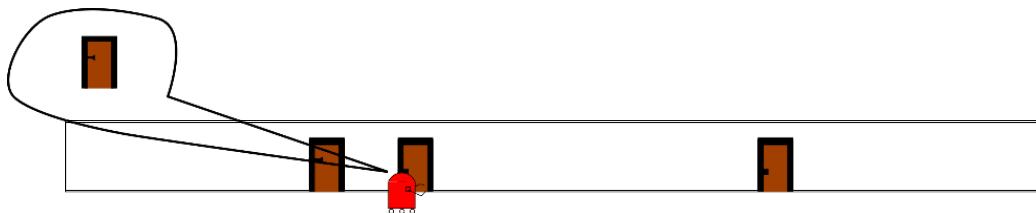


- The robot senses again:

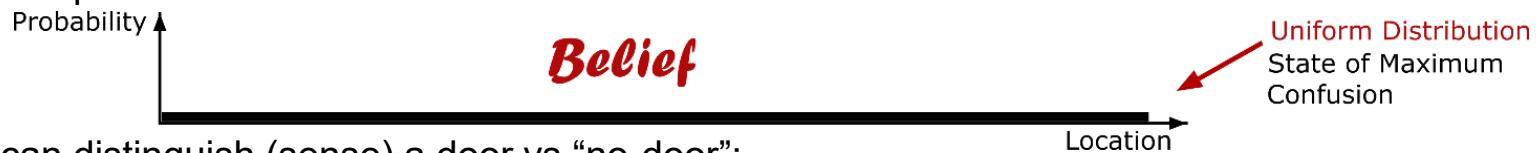


# Simple State Estimation Example

- Assume a robot in an area like this one:



- We model the position of the robot as follows:



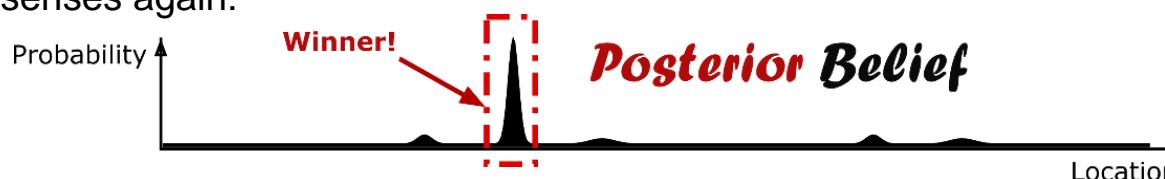
- The robot can distinguish (sense) a door vs “no-door”:



- The robot moves to the right:

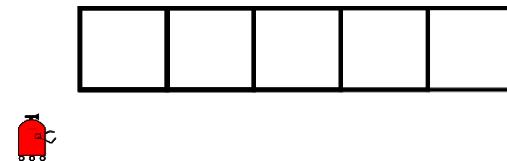


- The robot senses again:



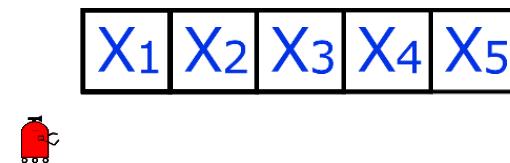
# Sense, Let's build it ourselves!

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



# Sense, Let's build it ourselves!

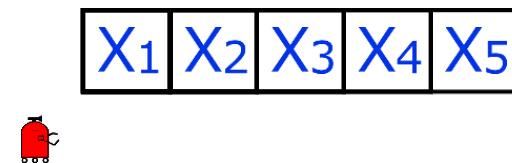
- This state estimation problem is called localization
- Assume a robot which can be in one of five blocks:



- 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:



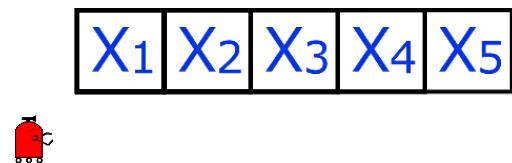
- Without any info, What is the probability of the robot being in each cell?

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

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:

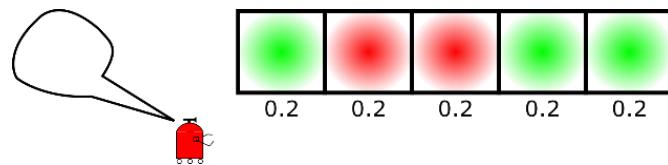


- Without any info, What is the probability of the robot being in each cell?

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

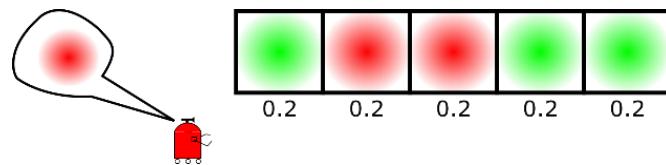
# Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



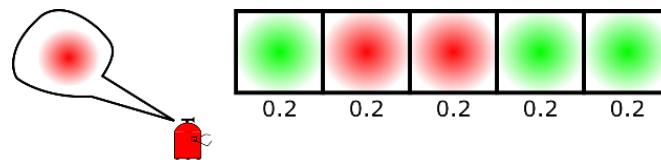
# Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



# Sense, Let's build it ourselves!

- Now our robot is allowed to sense:

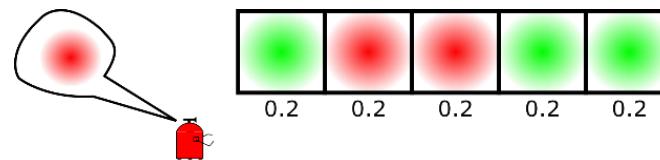


- What does that mean for our probability?



# Sense, Let's build it ourselves!

- Now our robot is allowed to sense:



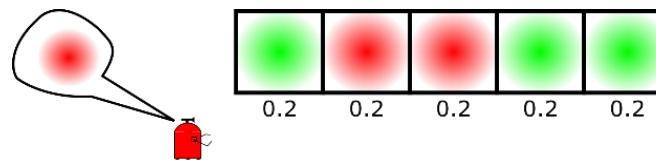
- What does that mean for our probability?



- 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:



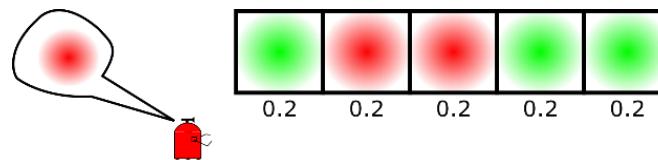
- What does that mean for our probability?



- 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:



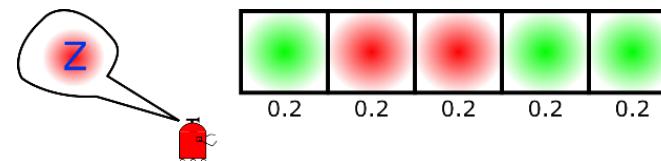
- What does that mean for our probability?

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:



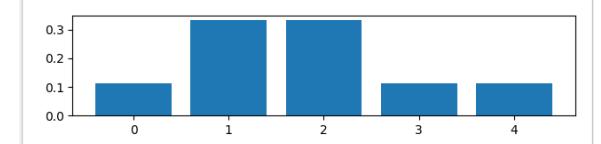
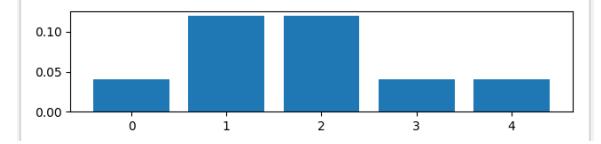
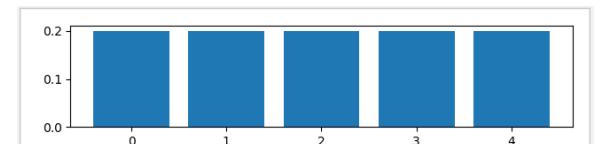
- What does that mean for our probability?

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?

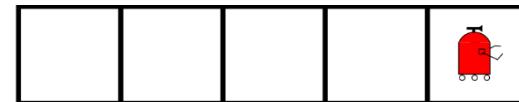
$$p(X_i | Z)$$

*Posterior Distribution*



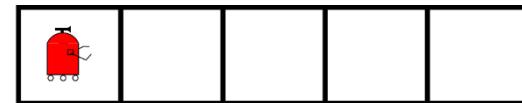
# Motion, Let's build it ourselves!

- 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):



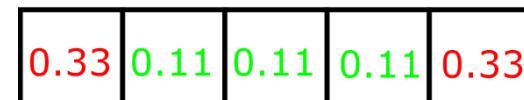
0.11	0.33	0.33	0.11	0.11
------	------	------	------	------

# 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):



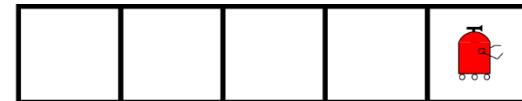
- What will happen with the posterior probability?



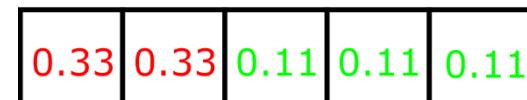
- However, this doesn't happen. Usually the distribution becomes more uncertain:

# Motion, Let's build it ourselves!

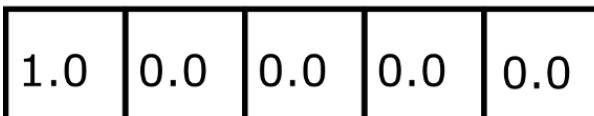
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



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



# Motion, Let's build it ourselves!

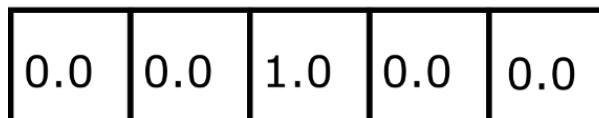
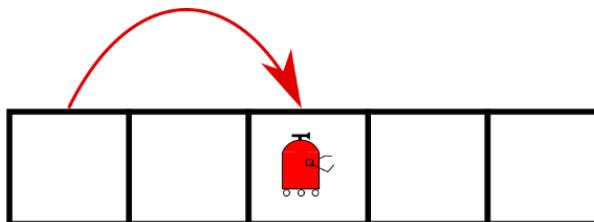
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



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

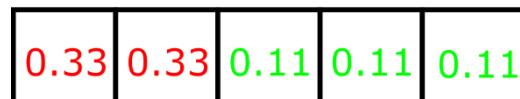


# Motion, Let's build it ourselves!

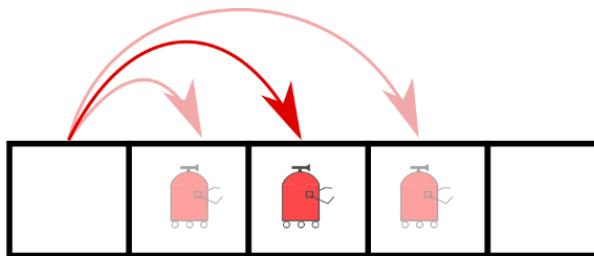
- Assume that the space is circular (i.e when moving right in the last cell you go to the first):



- What will happen with the posterior probability?



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

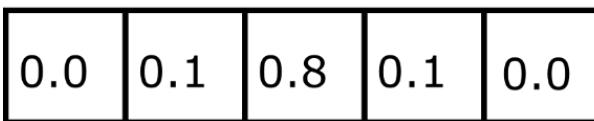


Uncertain  
Motion

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

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

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



# Motion, Let's build it ourselves!

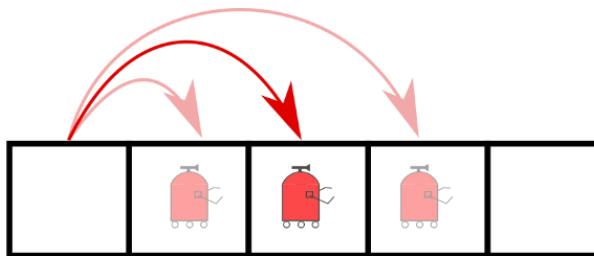
- Assume that the space is circular (i.e when moving right in the last bin, you end up in the first bin)



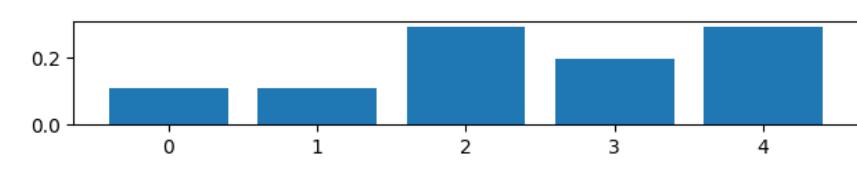
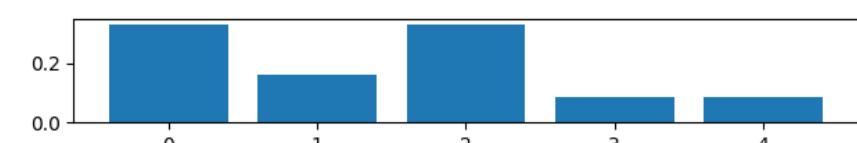
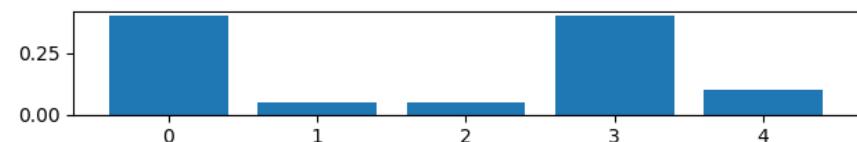
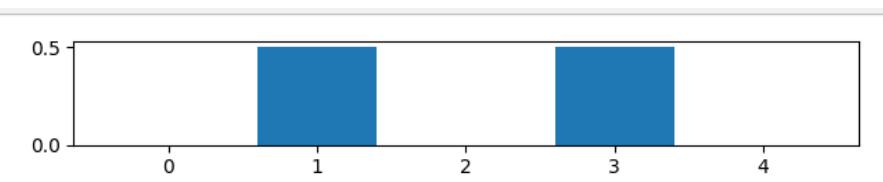
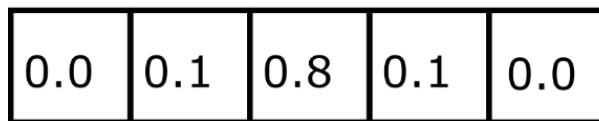
- What will happen with the posterior probability?



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



*Uncertain Motion*



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

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

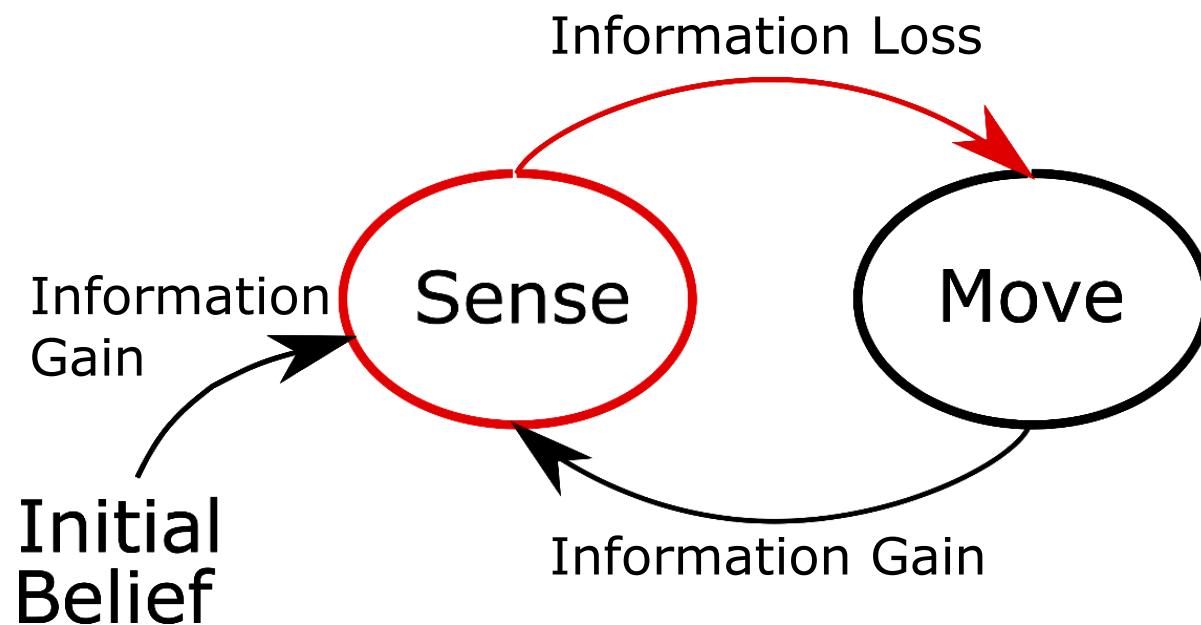
$$p(X_{i+U+1}) = 0.1$$

# That's very good!!!!

- Localization is just a sense/move cycle:

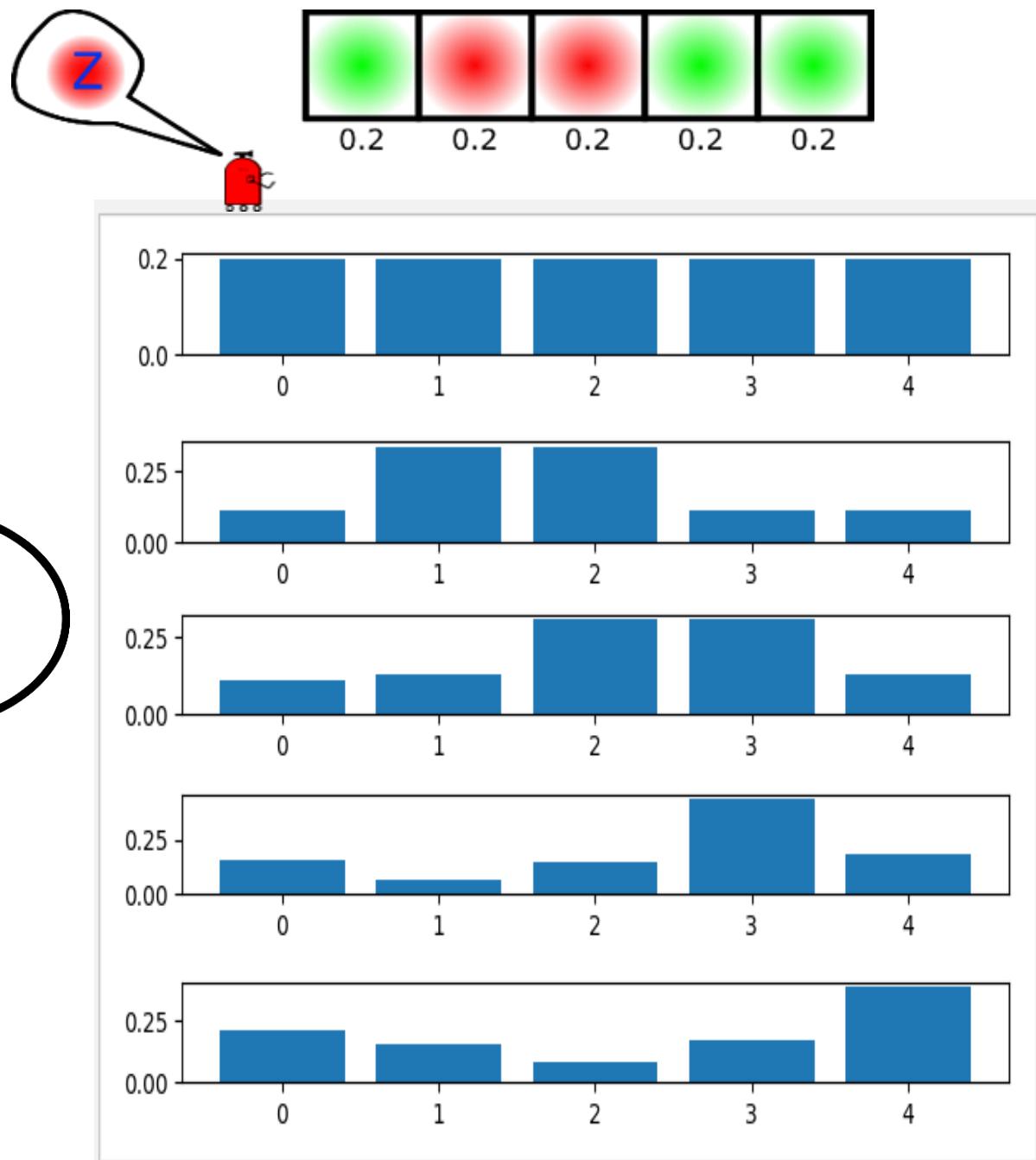
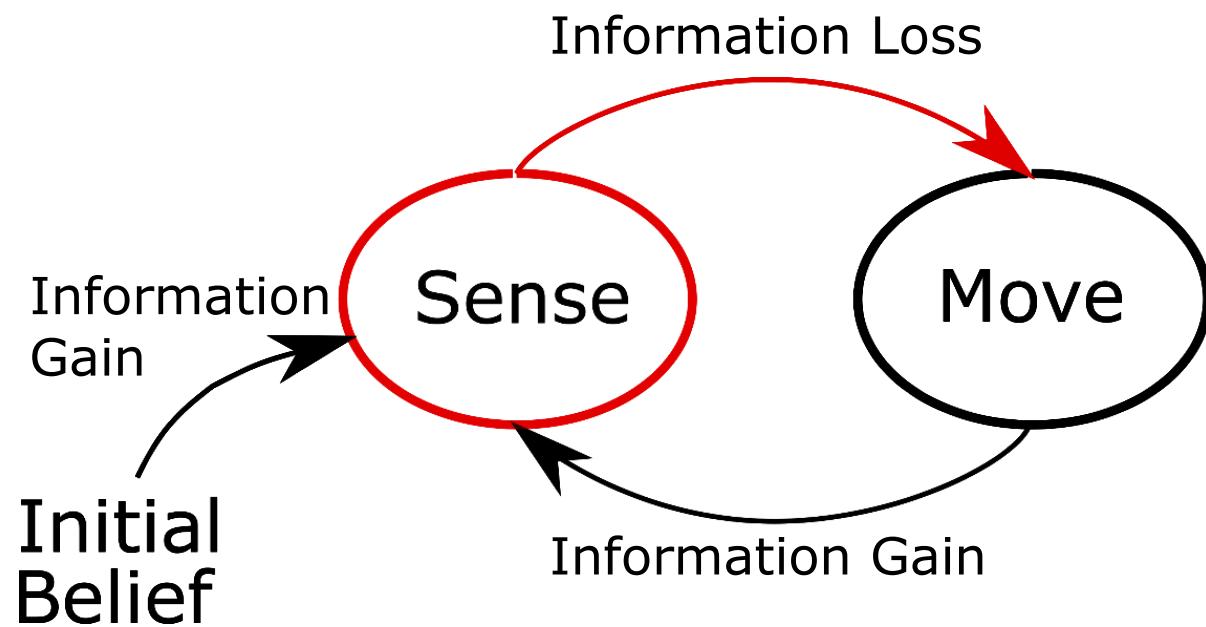
# That's very good!!!!

- 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*

The diagram illustrates the formula for movement probability. It shows the formula  $p(X_i^t) = \sum_j p(X_j^{t-1}) p(X_i | X_j)$ . A curly brace on the left side of the summation symbol groups the term  $p(X_j^{t-1})$  and is labeled 'Prior Probability' in red. Another curly brace on the right side of the summation symbol groups the term  $p(X_i | X_j)$  and is labeled 'Movement Probability' in red. Arrows point from the labels to their respective terms in the formula.

# 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:

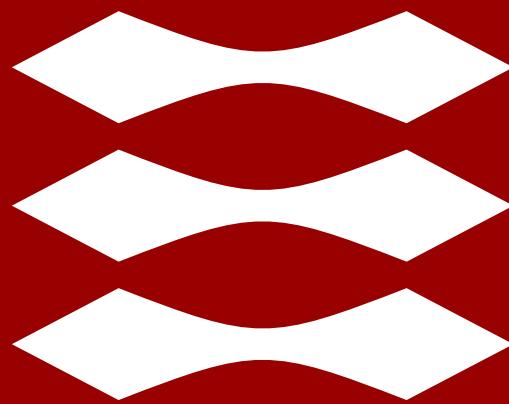
- Kalman Filter

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:

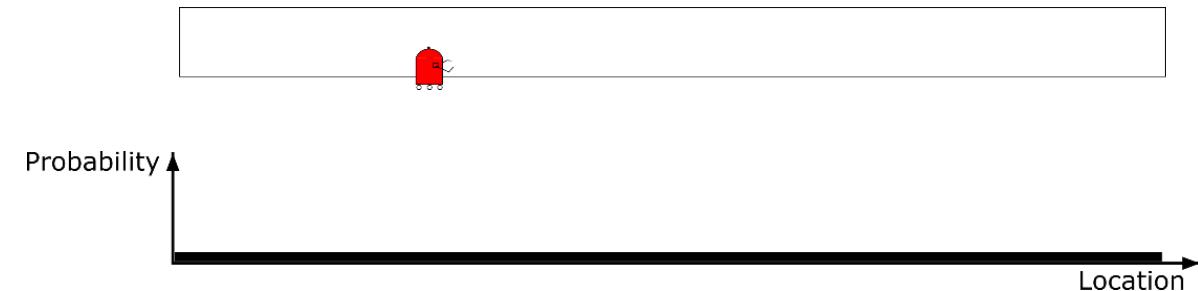
- Localization
- Tracking
- Prediction
- Sensor Fusion
- ...

# Catching up

- Last part, we did state estimation specifically Localization

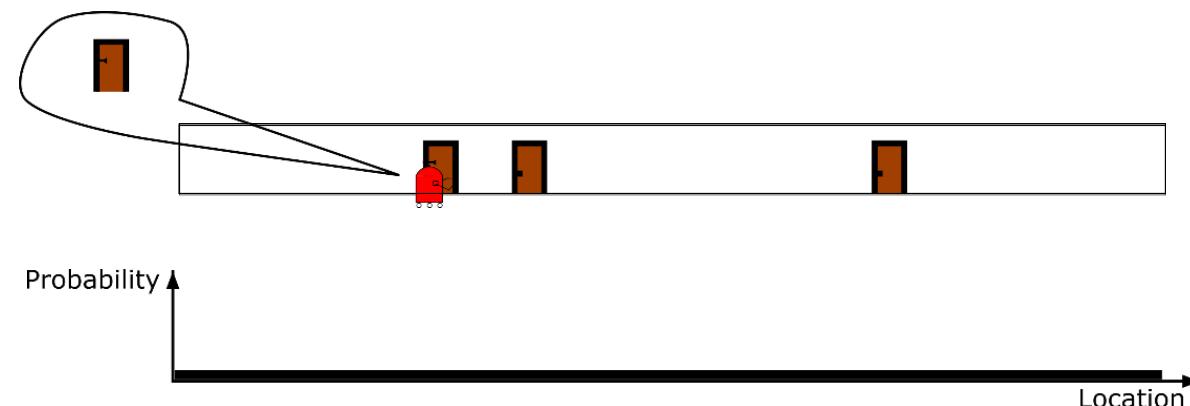
# Catching up

- Last part, we did state estimation specifically Localization



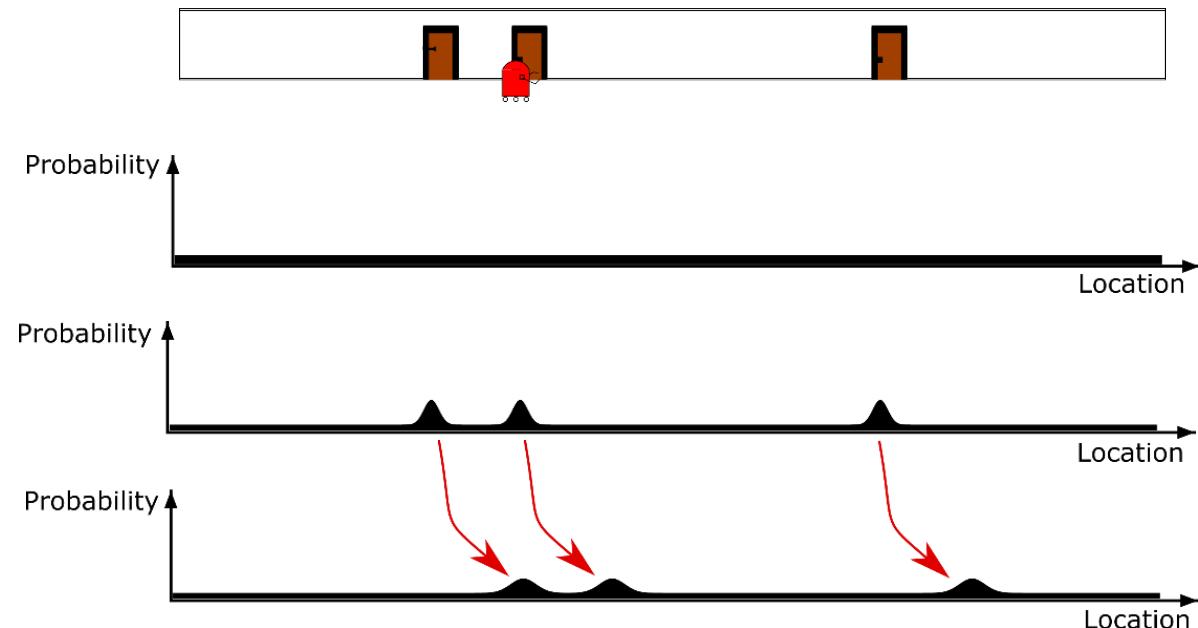
# Catching up

- Last part, we did state estimation specifically Localization



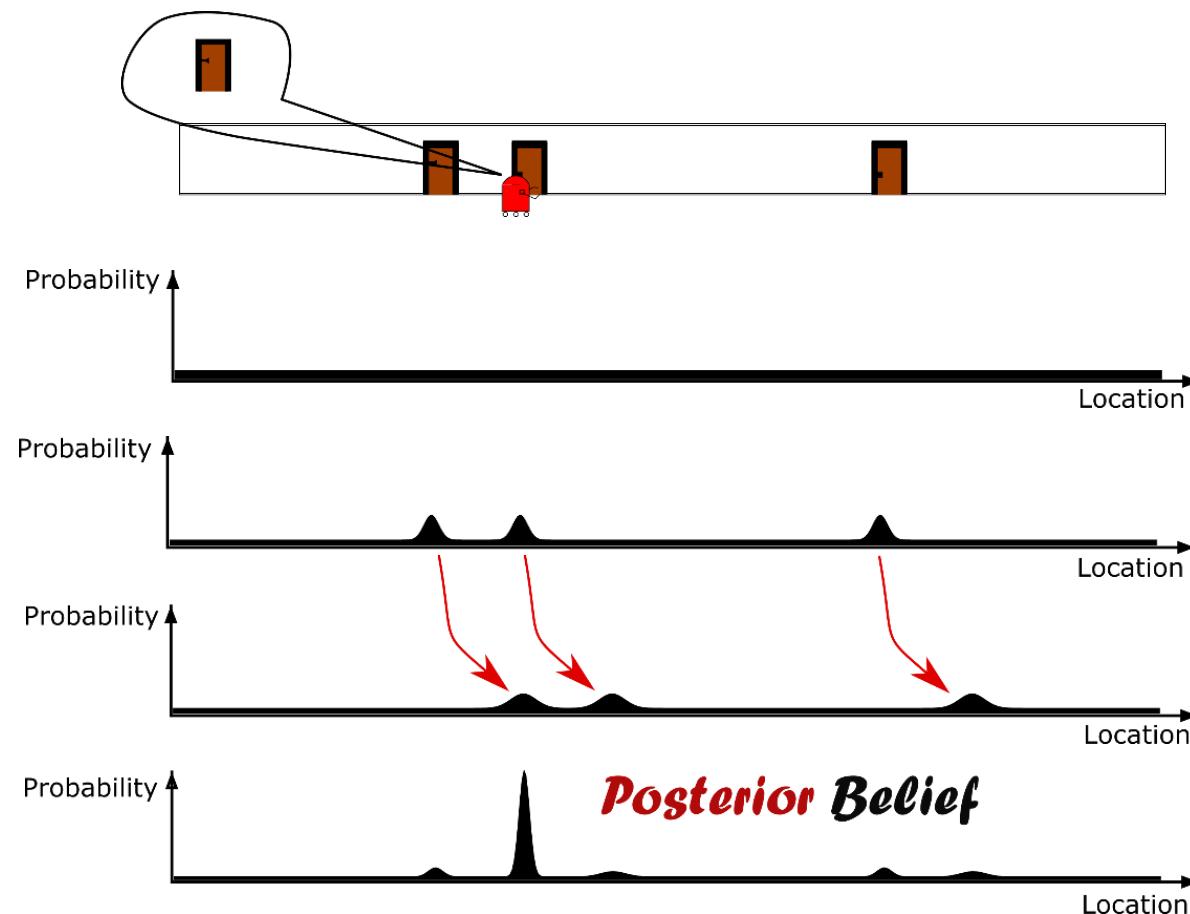
# Catching up

- Last part, we did state estimation specifically Localization



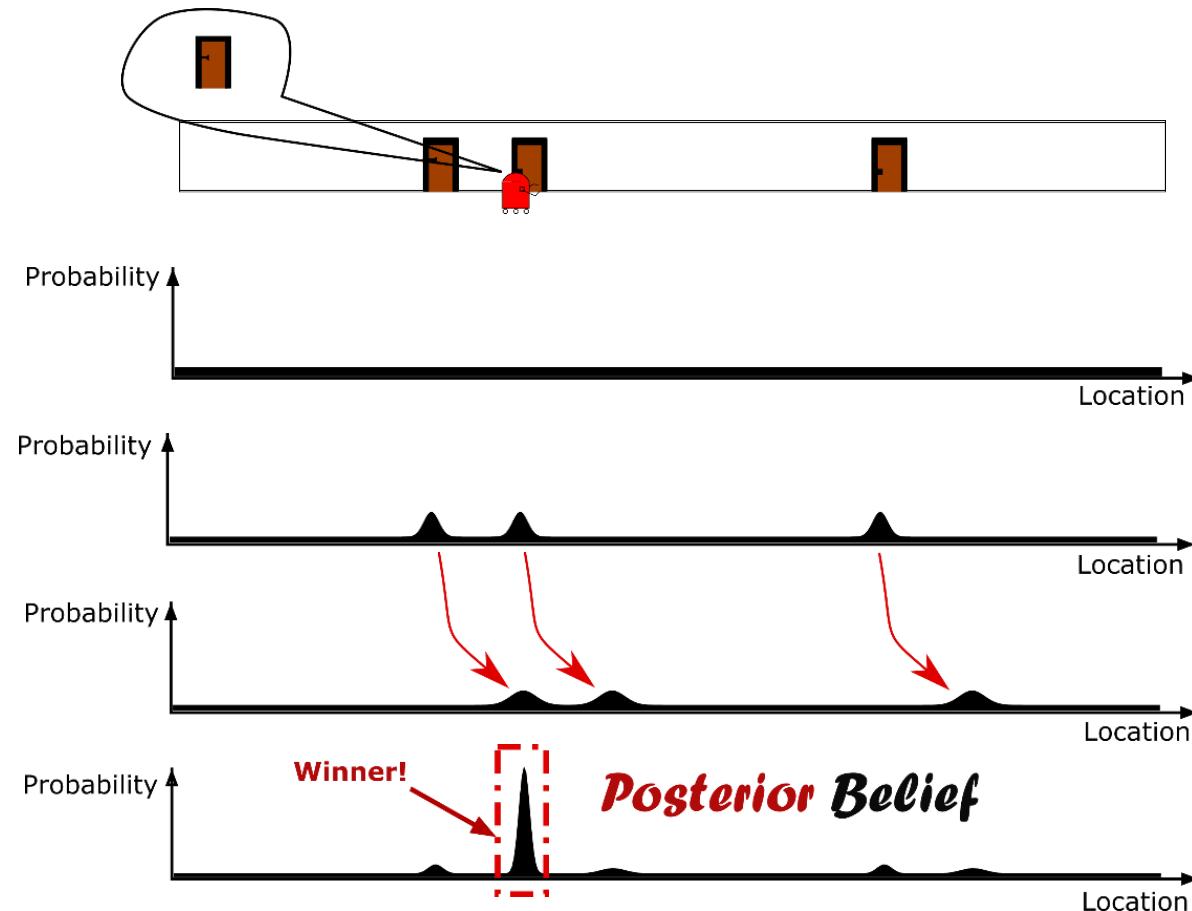
# Catching up

- 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

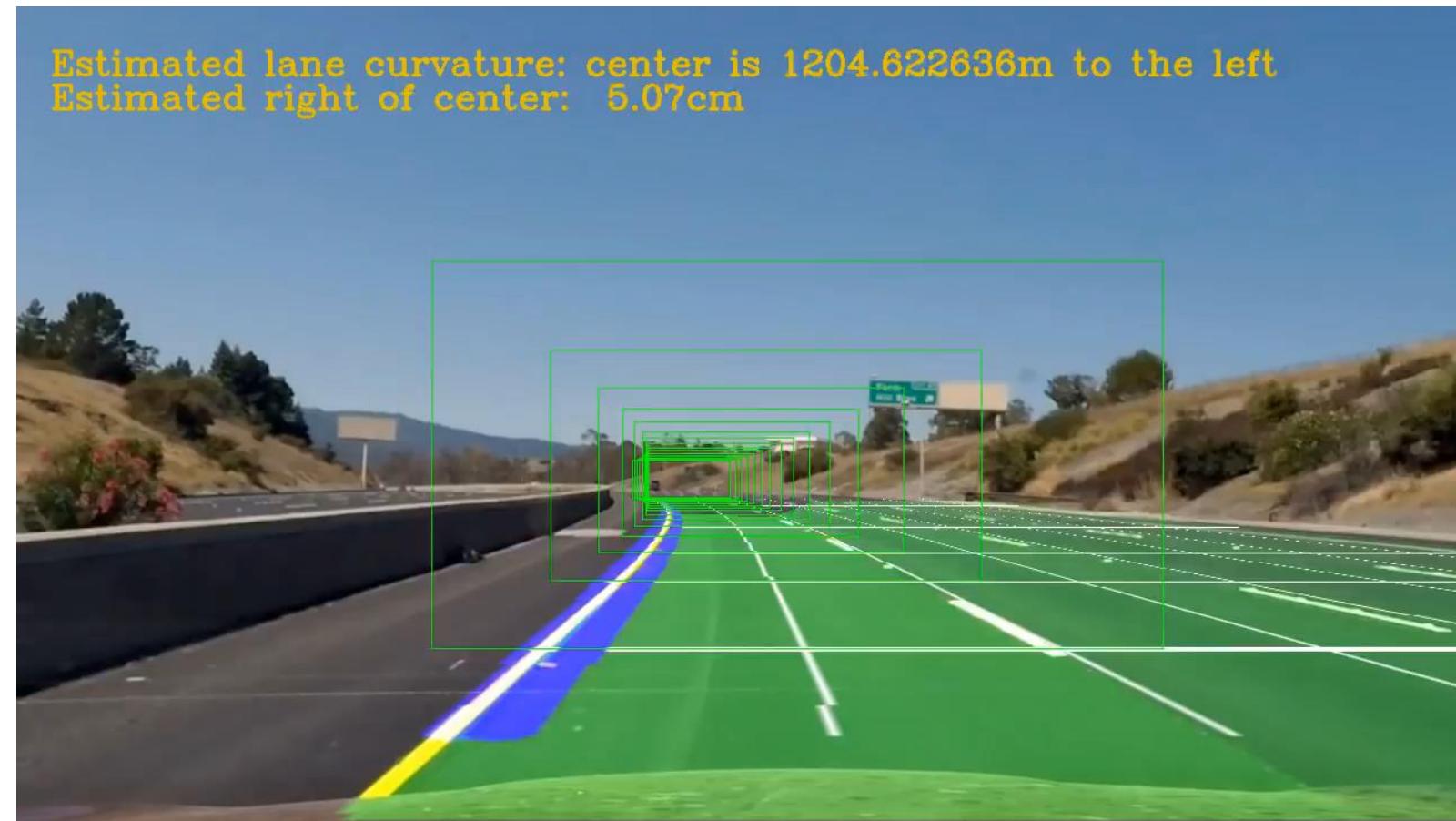
# Catching up

- 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..

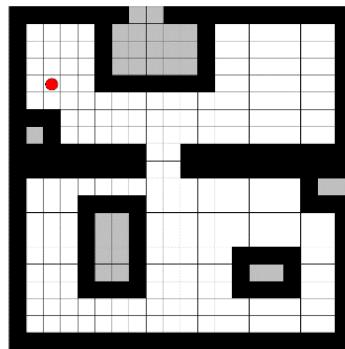
- We will see how we can track objects.
  - Not only their location as in the previous localization case,
  - But also infer their speed.
- 
- In the case of autonomous driving it is quite important to track and predict the movement of objects.
    - Why?
    - Which other examples can we find?

# Cases of Tracking

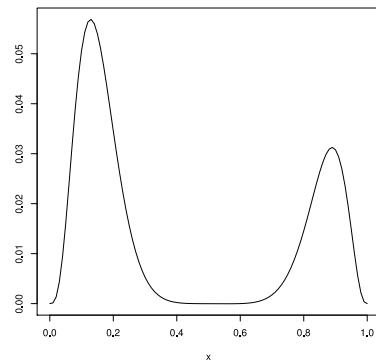


# Differences between state estimation filters

Histogram filter

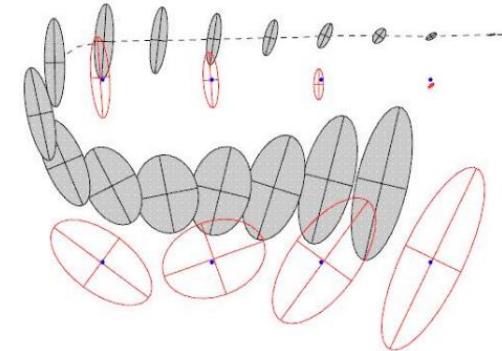


Discrete

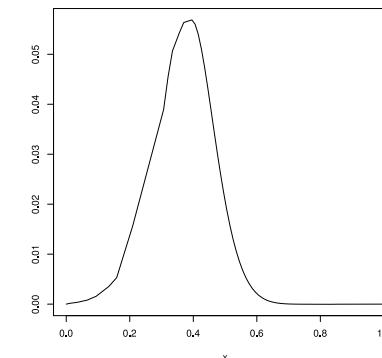


Multimodal

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:



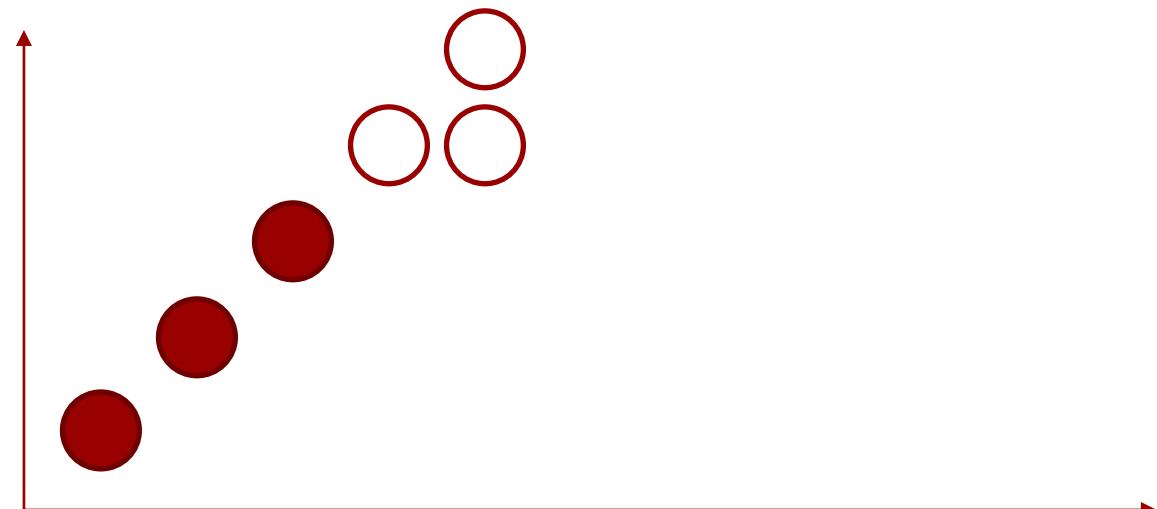
# Let's see tracking as an example

- Assuming there is a point in space like this:
- The object moves like this:
- What is the next point?



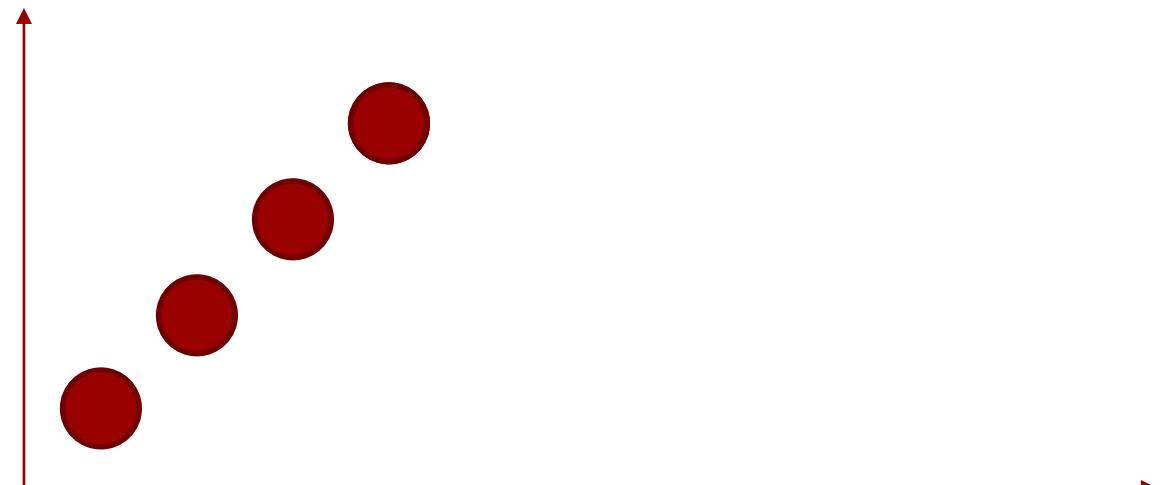
# Let's see tracking as an example

- Assuming there is a point in space like this:
- The object moves like this:
- What is the next point?



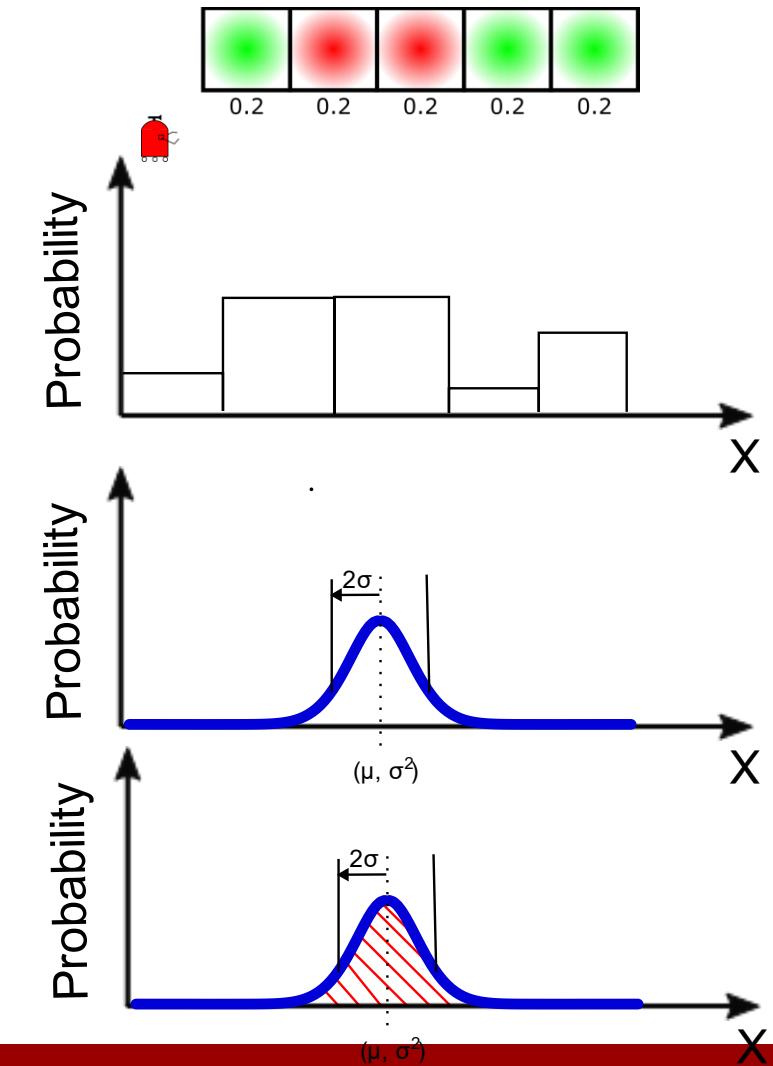
# Let's see tracking as an example

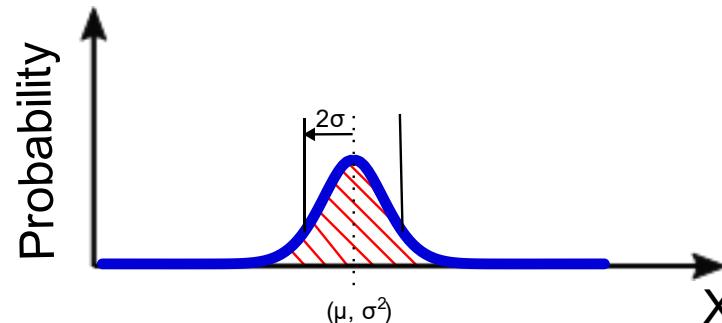
- Assuming there is a point in space like this:
  - The object moves like this:
  - What is the next point?
- 
- For you it is easy! How about a machine?



# 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

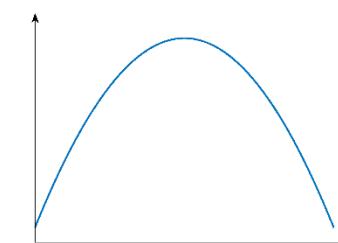
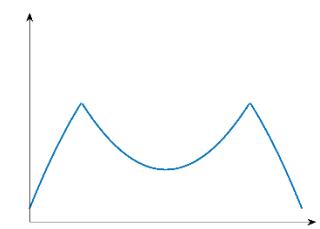
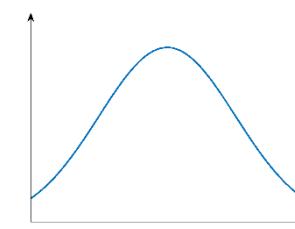
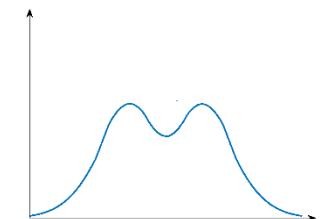
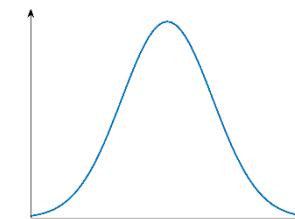




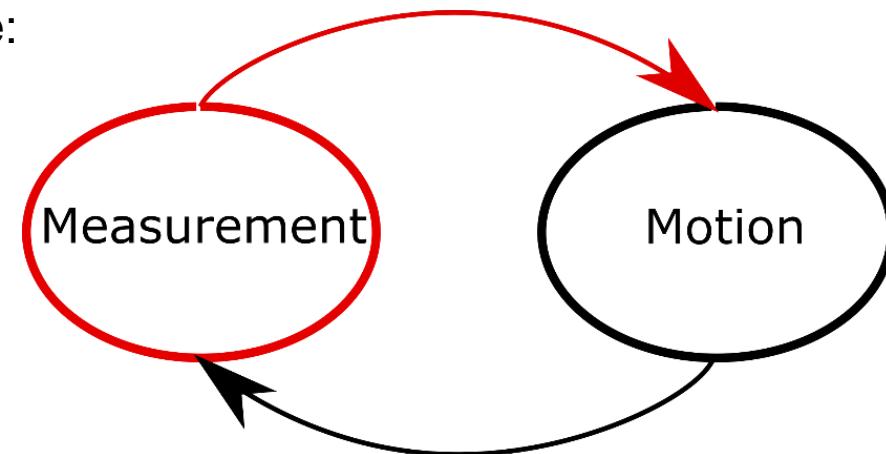
- 1D Gaussian is described by the pair:  $(\mu, \sigma^2)$

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

- 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?

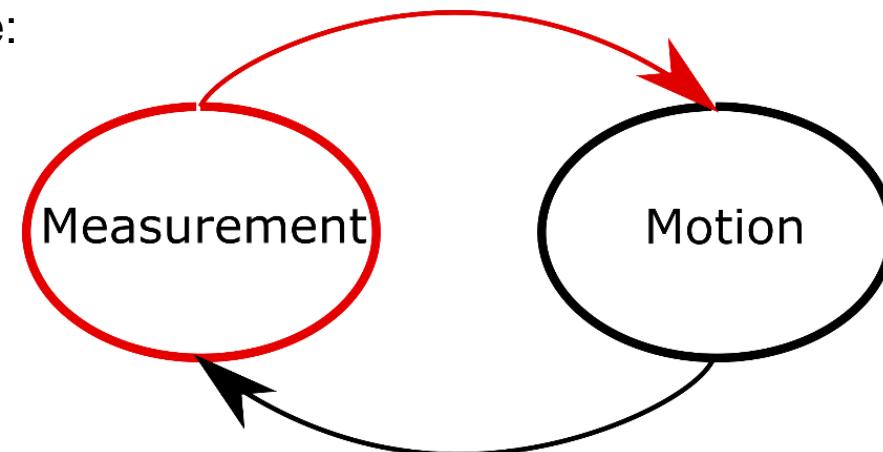


- Kalman, as with the histogram filter involves the measurement  $\Leftrightarrow$  motion cycle:



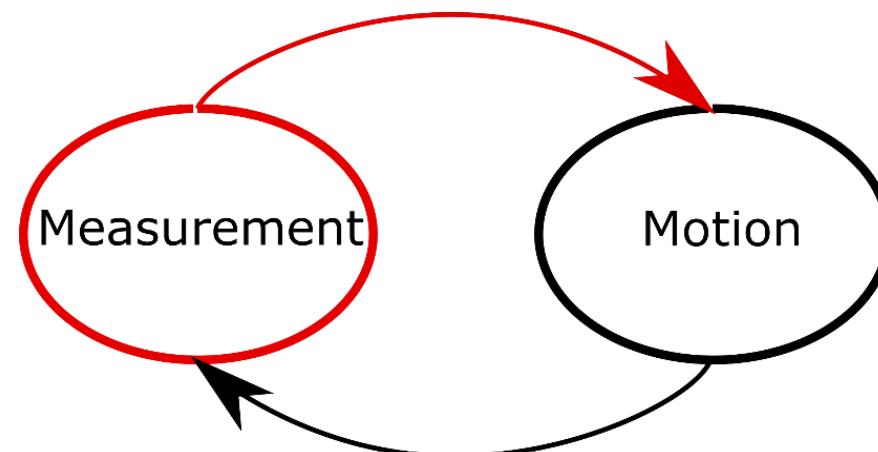
- Which one requires **convolution** and which one a **product**?

- Kalman, as with the histogram filter involves the measurement  $\Leftrightarrow$  motion cycle:



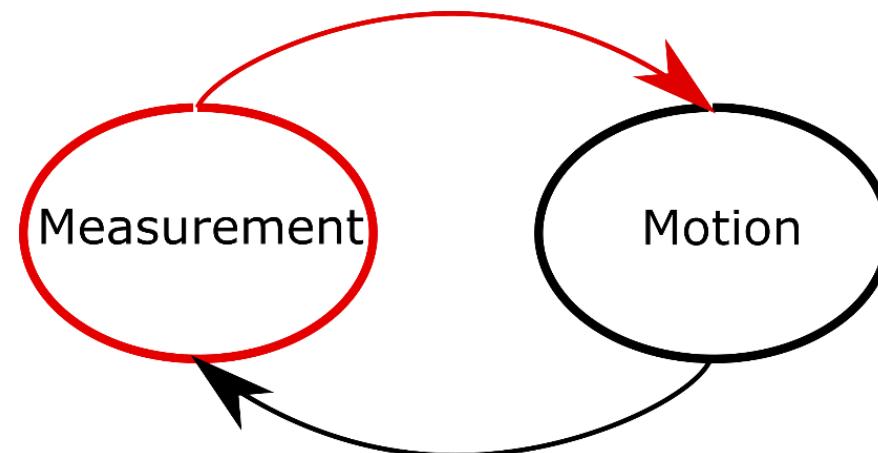
- Which one requires **convolution** and which one a **product**?
  - Measurement  $\Leftrightarrow$  Product
  - Motion  $\Leftrightarrow$  Convolution

- Kalman, as with the histogram filter involves the measurement  $\Leftrightarrow$  motion cycle:

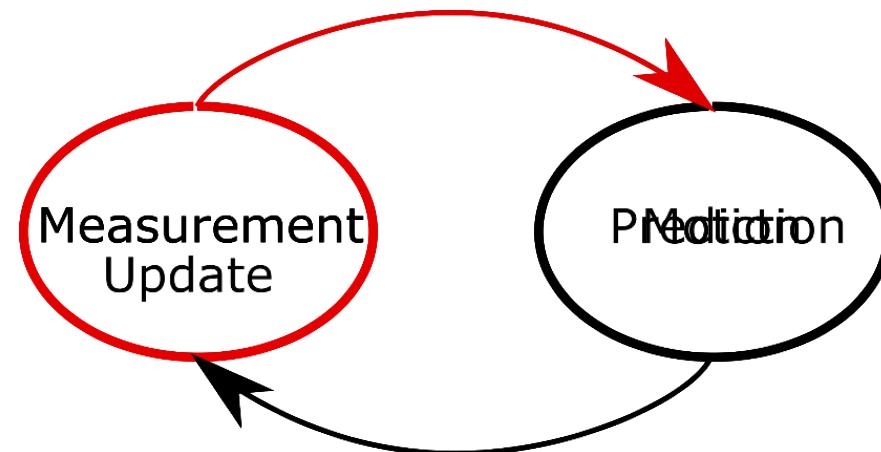


- Which one applies **Bayes Rule** and which one **Total Probability**?

- Kalman, as with the histogram filter involves the measurement  $\Leftrightarrow$  motion cycle:



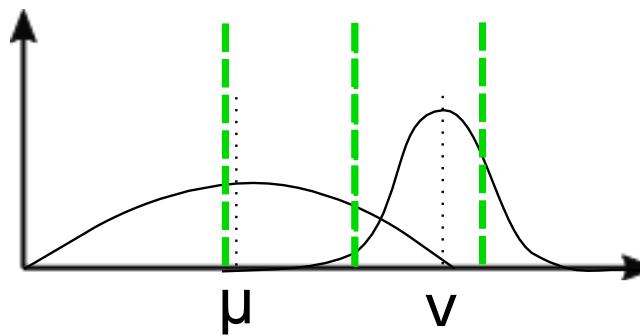
- Which one applies **Bayes Rule** and which one **Total Probability**?
  - Measurement  $\Leftrightarrow$  Bayes Rule
  - Motion  $\Leftrightarrow$  Total Probability



- In Kalman we call them “Measurement Update” and “Prediction”
- Both of these involve the Gaussians

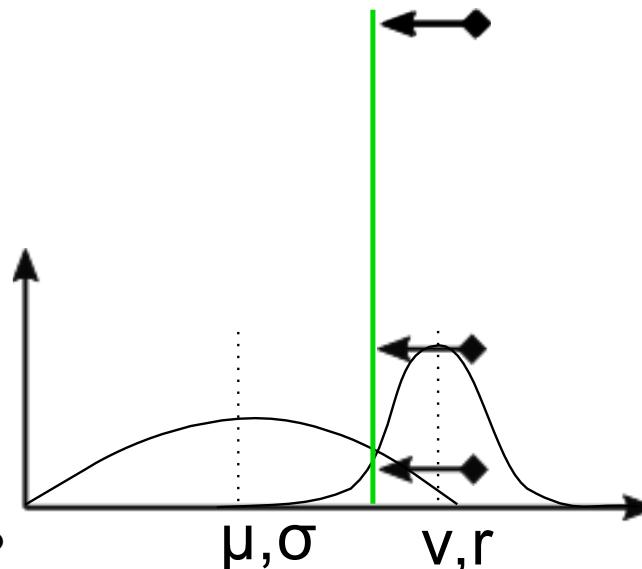
# Kalman Measurement Update

- 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

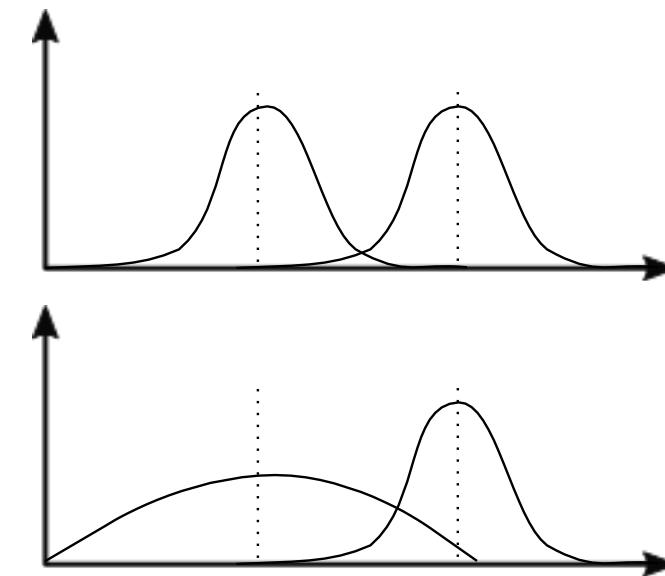
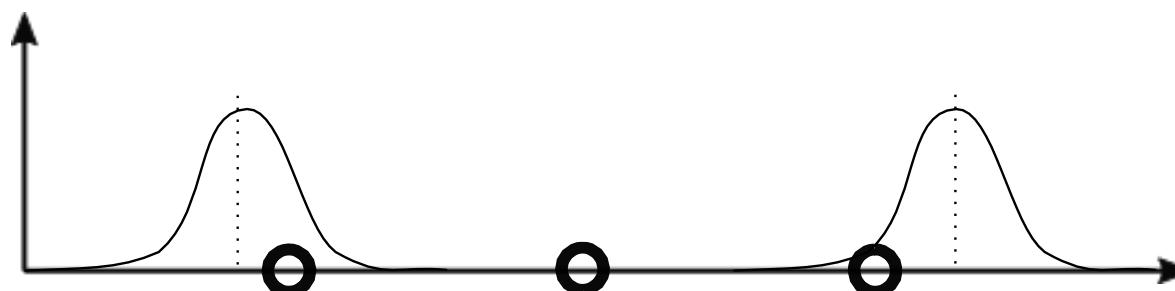


- Where will the new peak be?
- The higher one -> as we gain information
- Let's prove it

$$\mu' = \frac{r^2 \mu + \sigma^2 v}{r^2 + \sigma^2}$$

$$\sigma'^2 = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}}$$

- 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

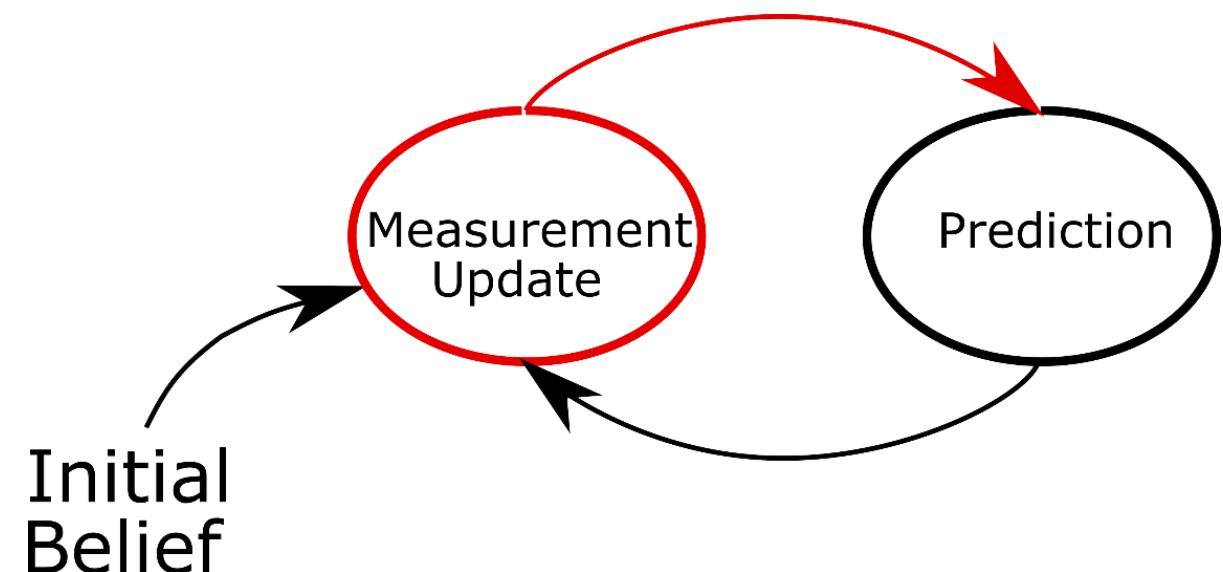
- Called also prediction:
- As we move we lose some information:



- 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

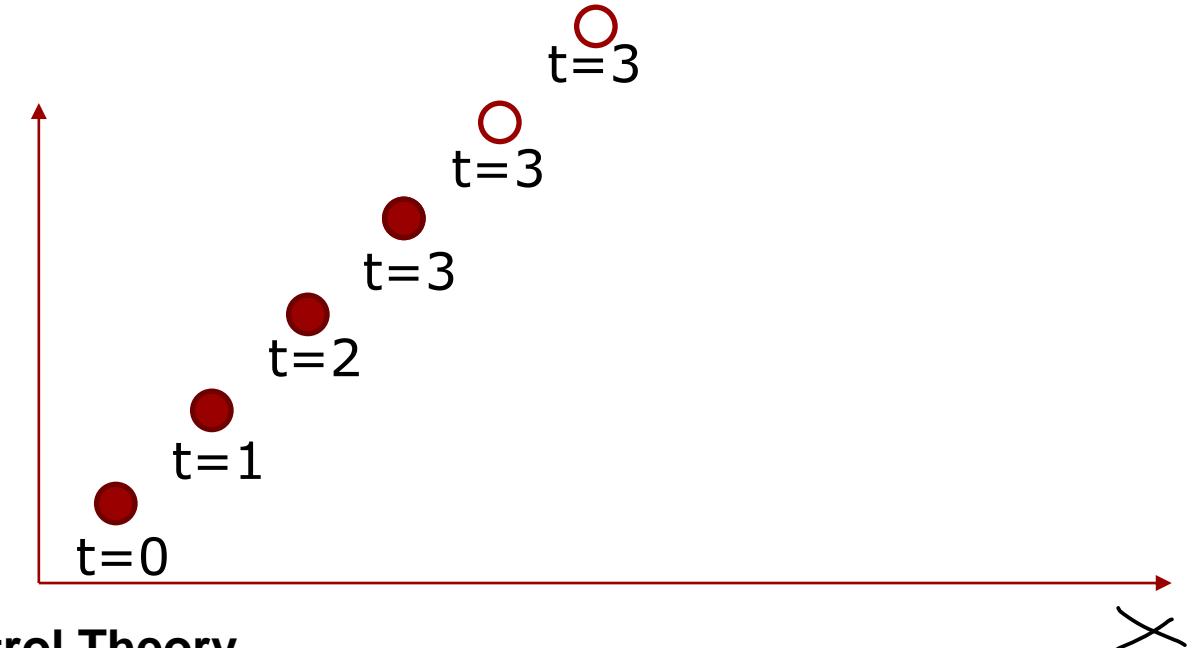
- Start with a initial belief of:
  - $u=0$ ,
  - $\sigma^2=10000$
- Motion:
  - $[1, 1, 2, 1, 1]$
  - Uncertainty: 2
- Measurement:
  - $[5, 6, 7, 9, 10]$
  - Uncertainty: 4



# 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

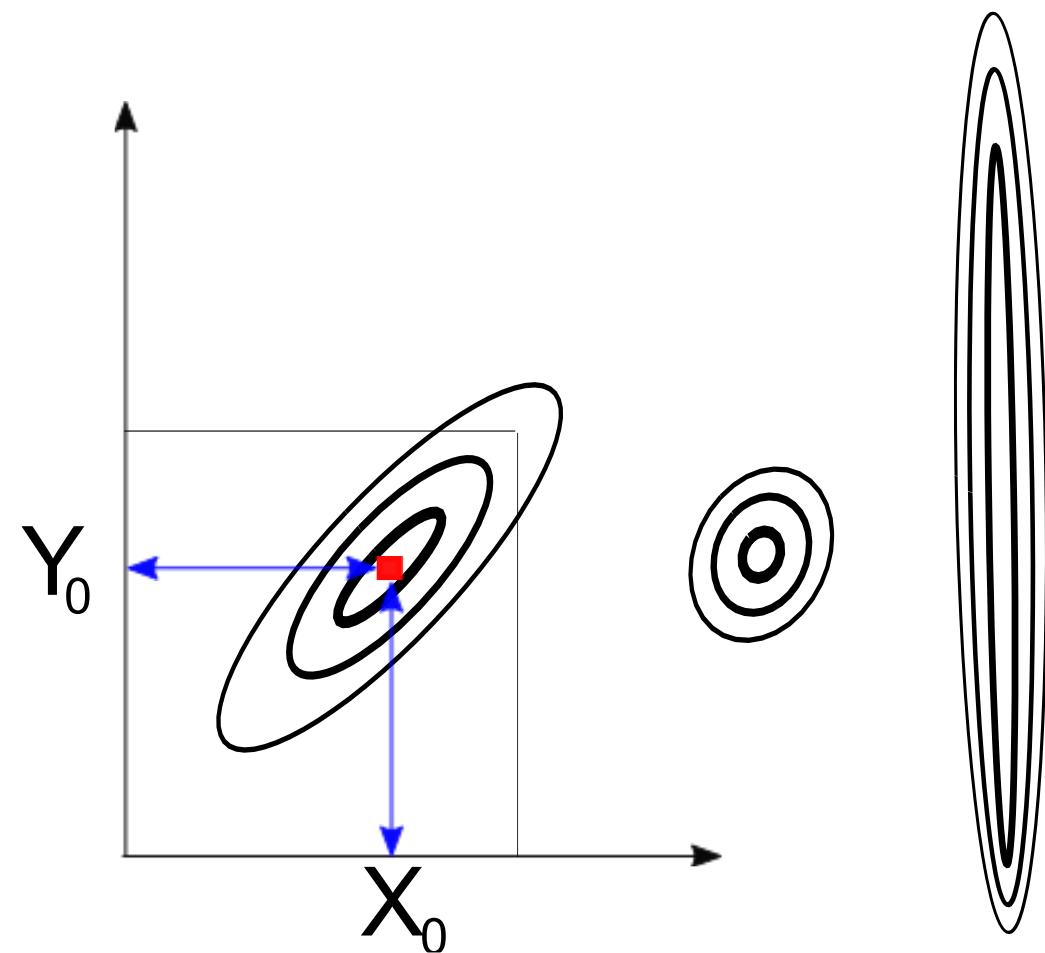
Multi-Dimensional Gaussians

- The mean is a vector:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \vdots \\ \boldsymbol{\mu}_D \end{pmatrix}$$

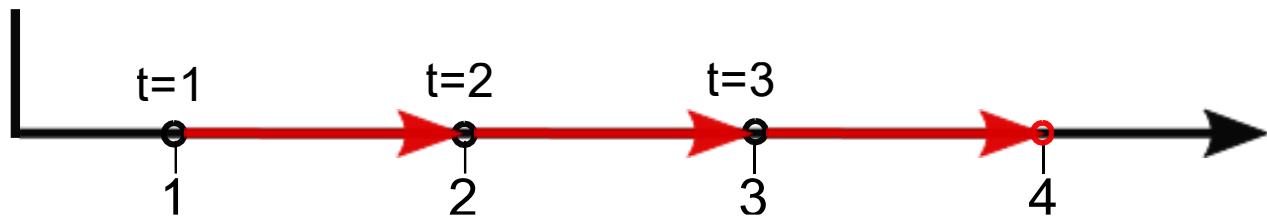
- The variance now is called covariance and is a matrix:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \ddiamond & \ddiamond & \ddiamond & \ddiamond & \ddiamond \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

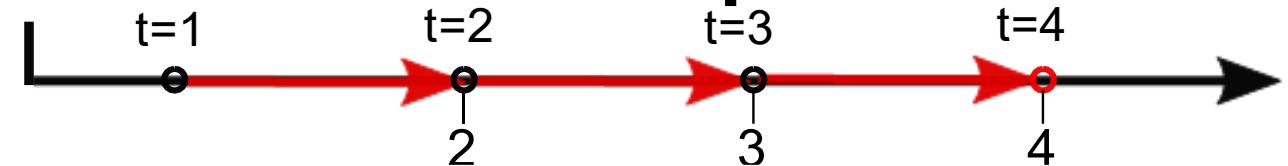


# Multivariate Gaussians

- Let's start with an one dimensional motion example:



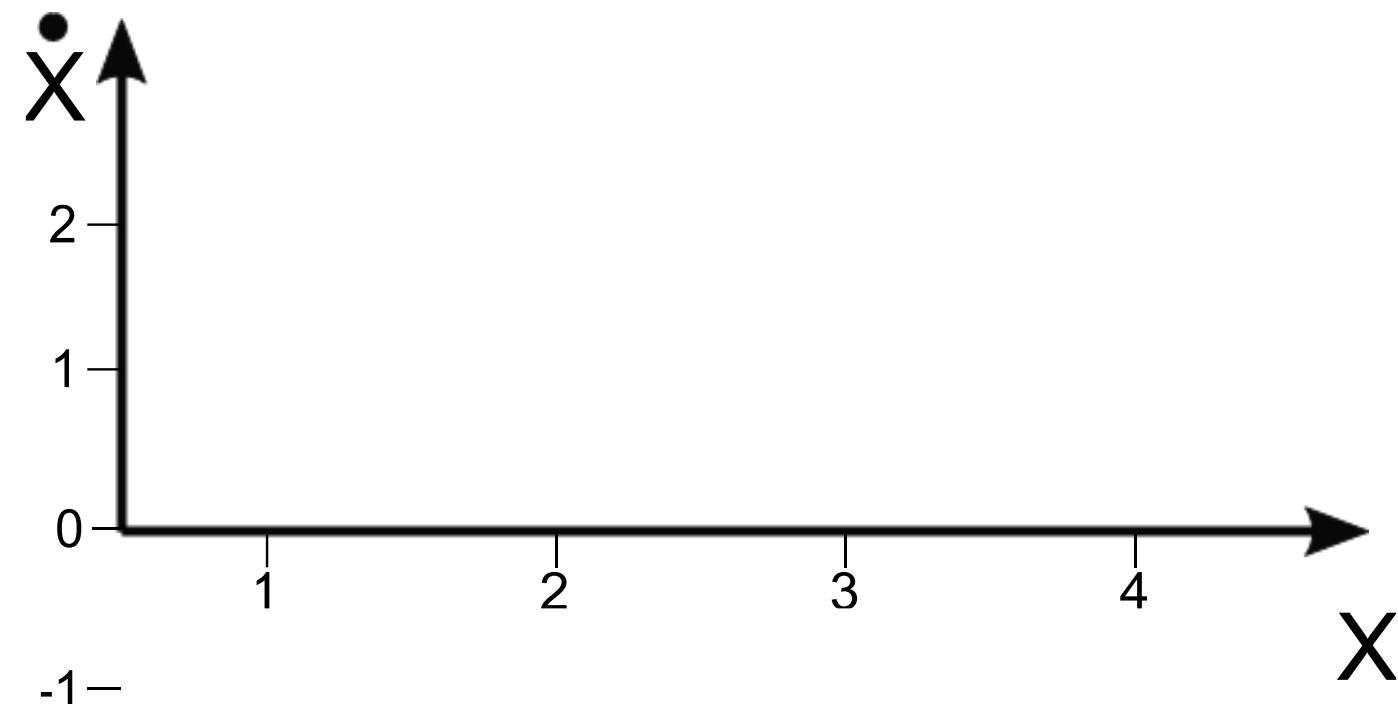
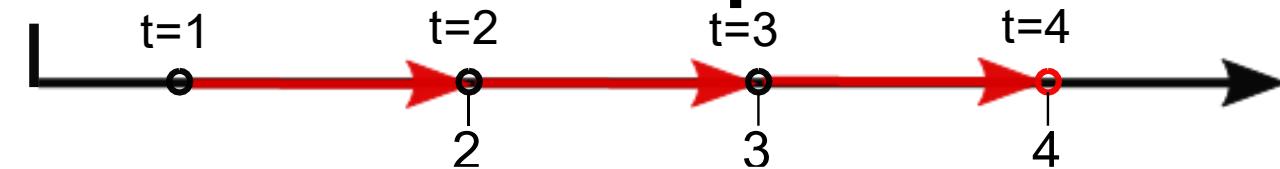
# Multivariate Gaussians - Kalman State Space



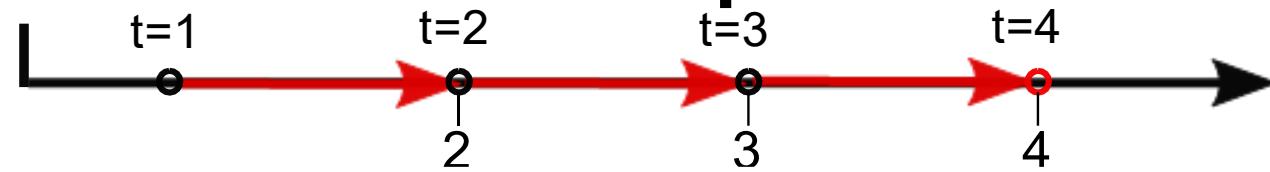
- Let's go at the Kalman state space:

# Multivariate Gaussians - Kalman State Space

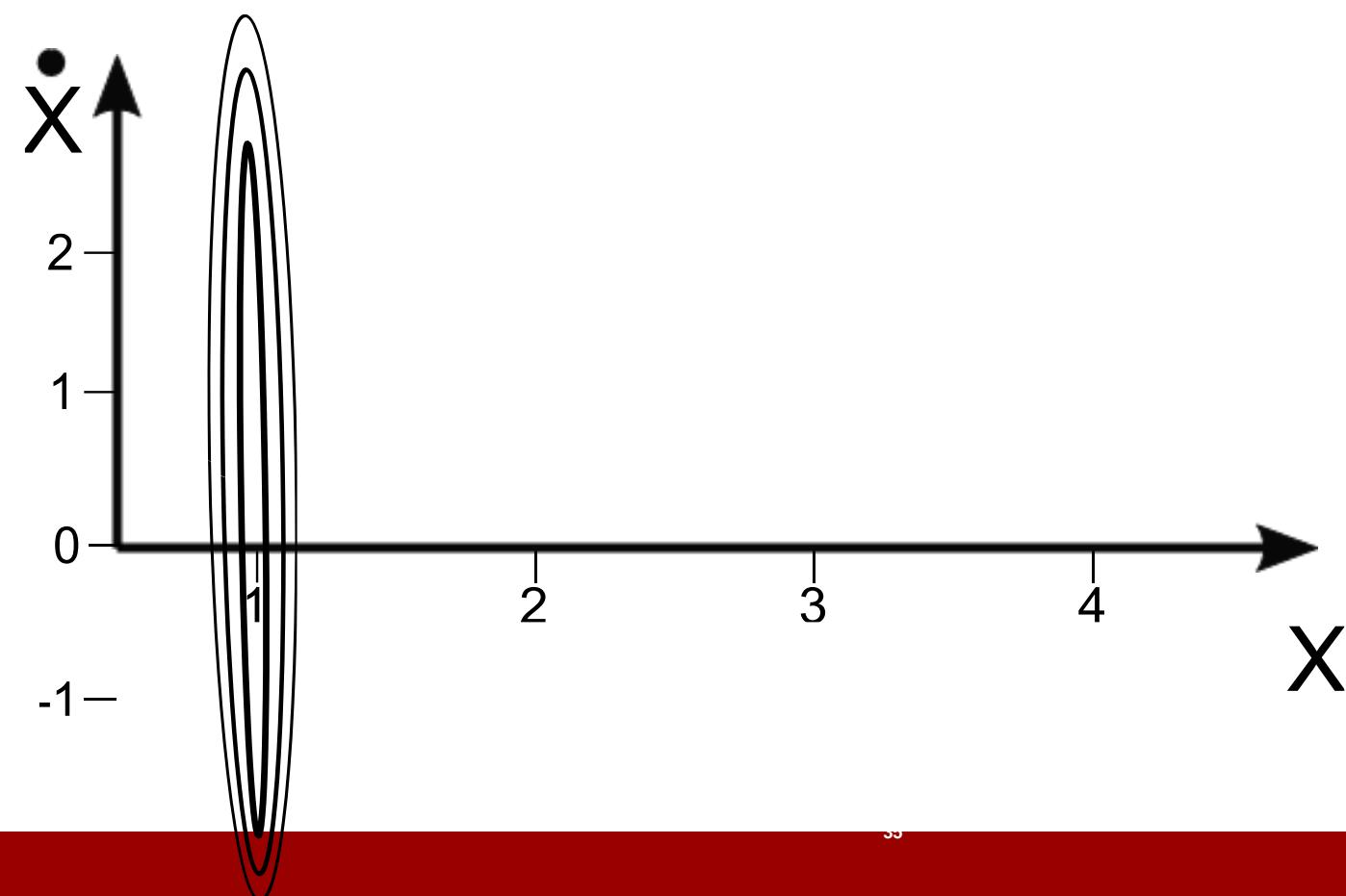
- Let's go at the Kalman state space:



# Multivariate Gaussians - Kalman State Space

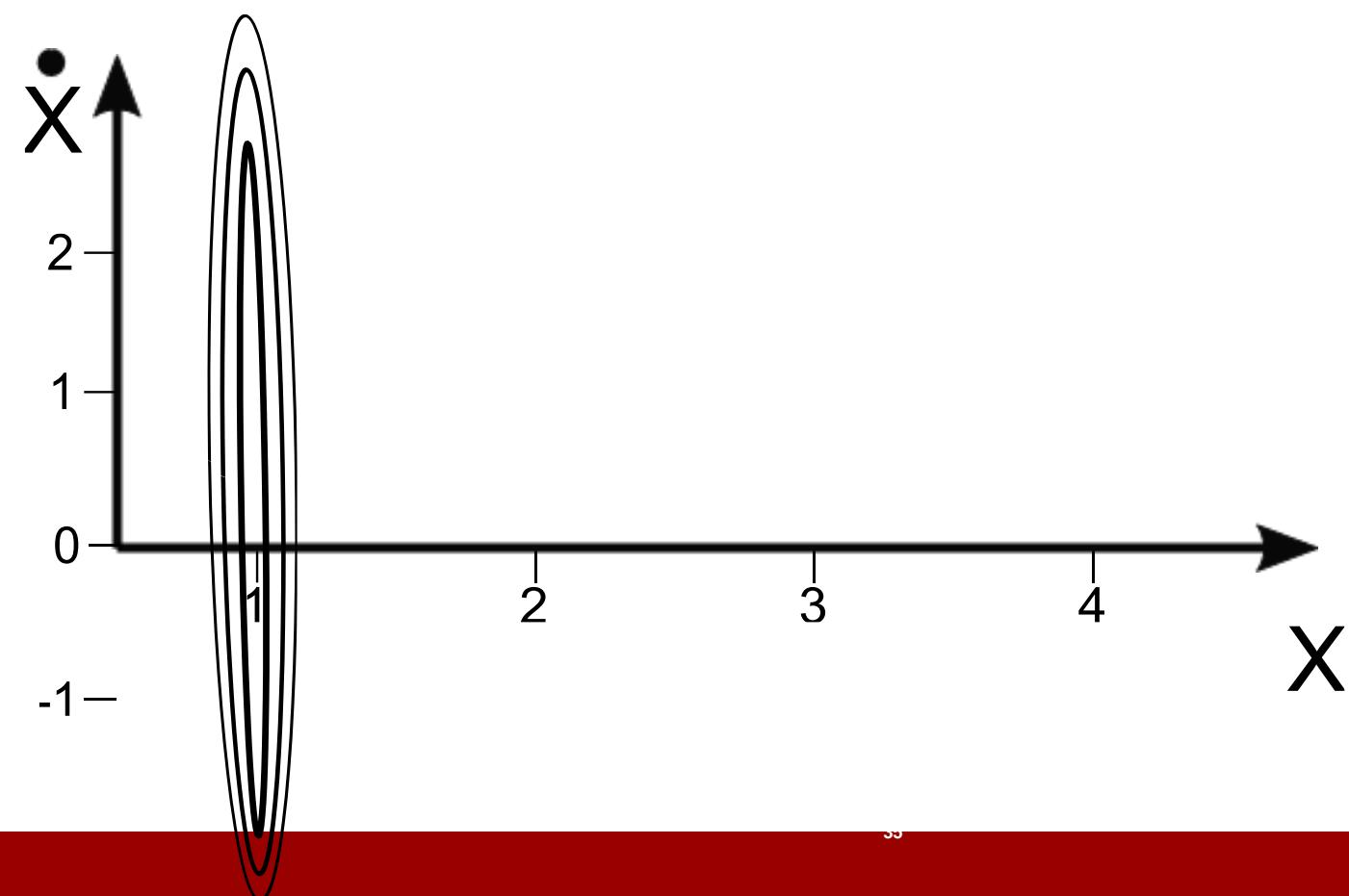
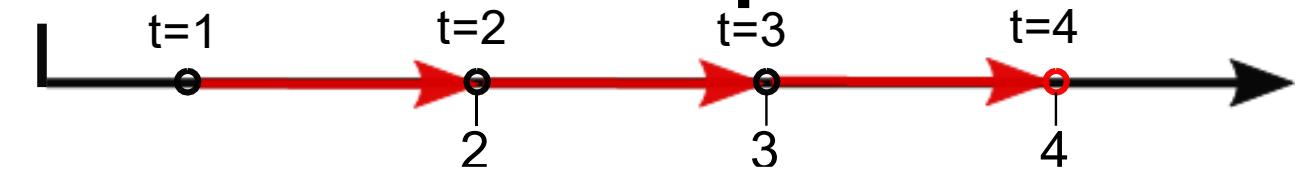


- Let's go at the Kalman state space:
- Prior:
  - Location: 1



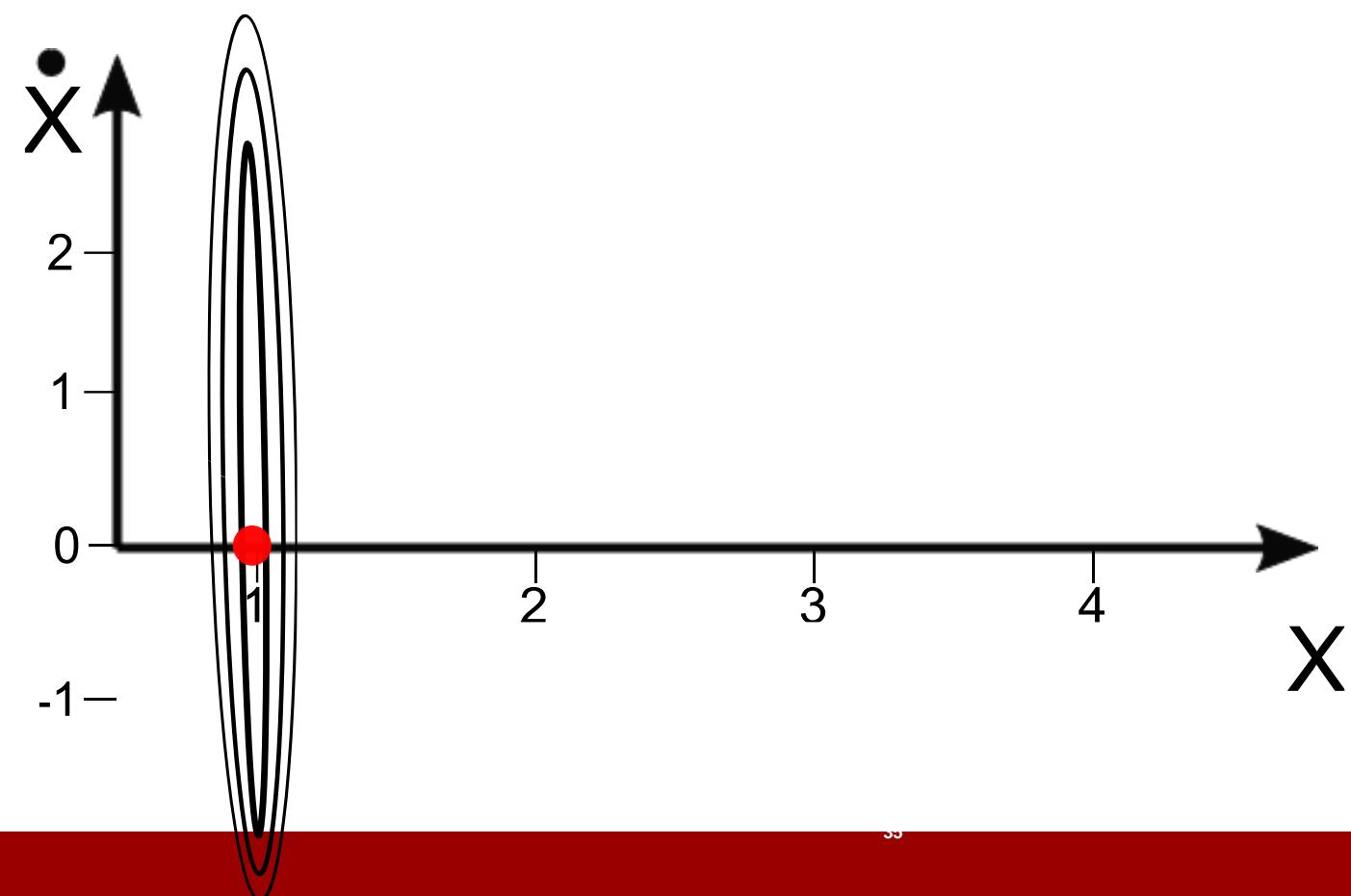
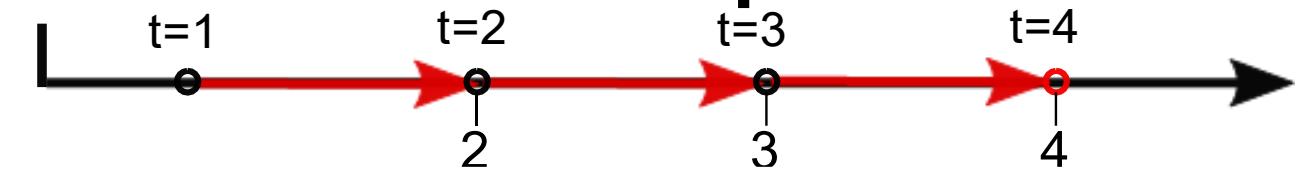
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0



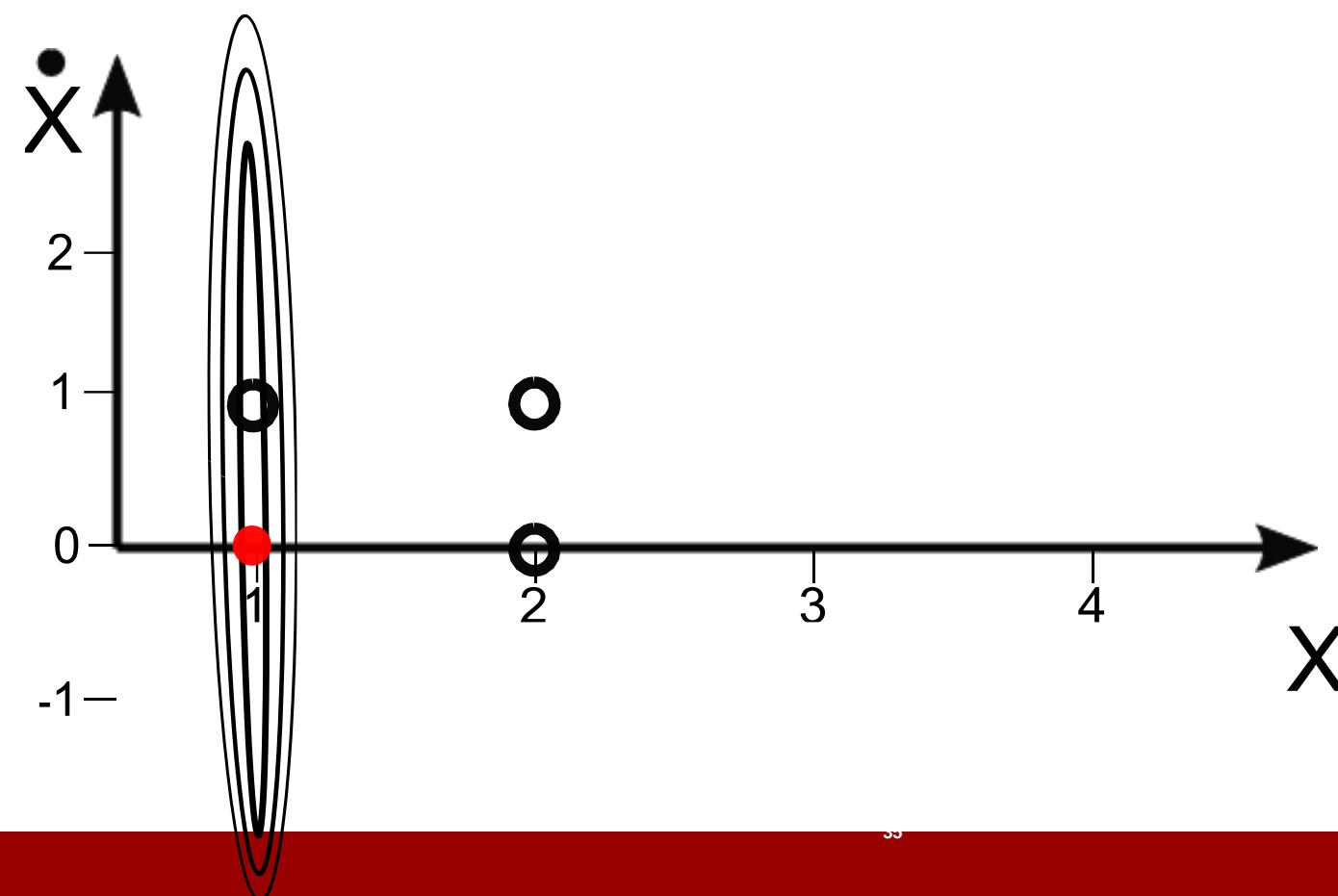
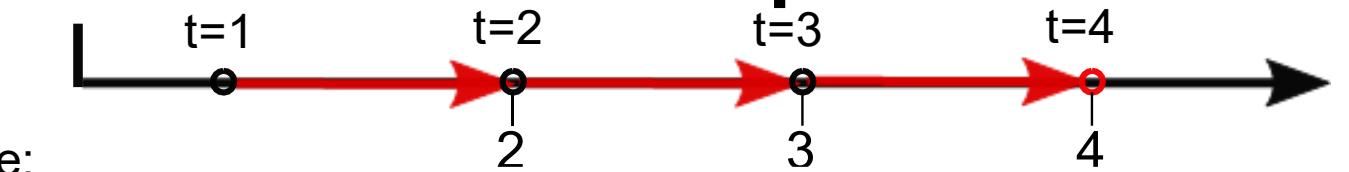
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0



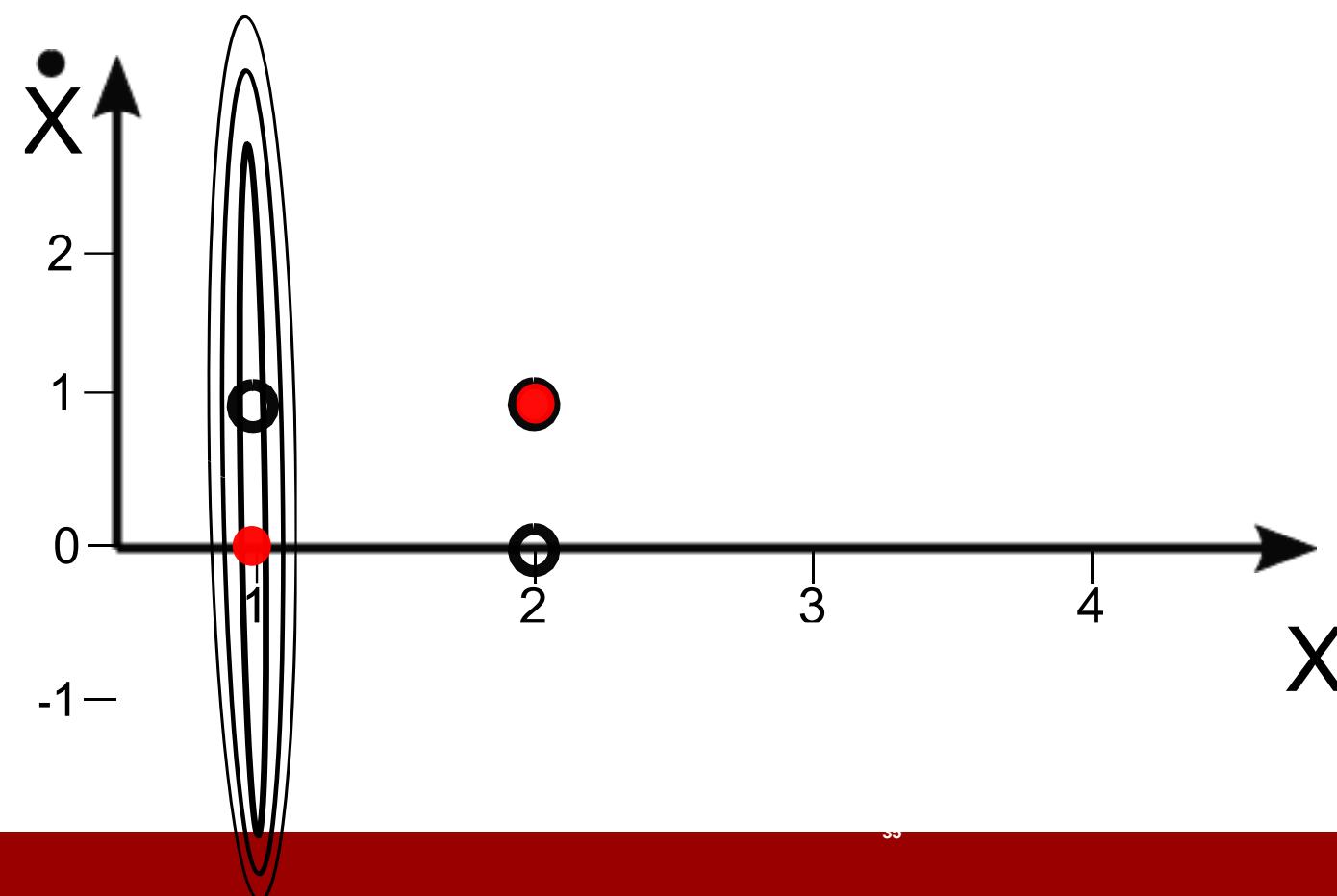
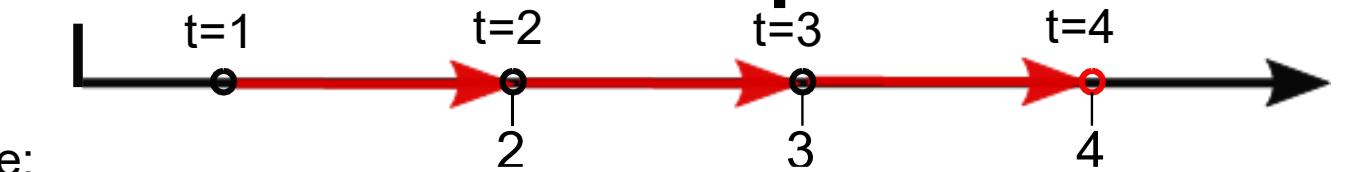
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1



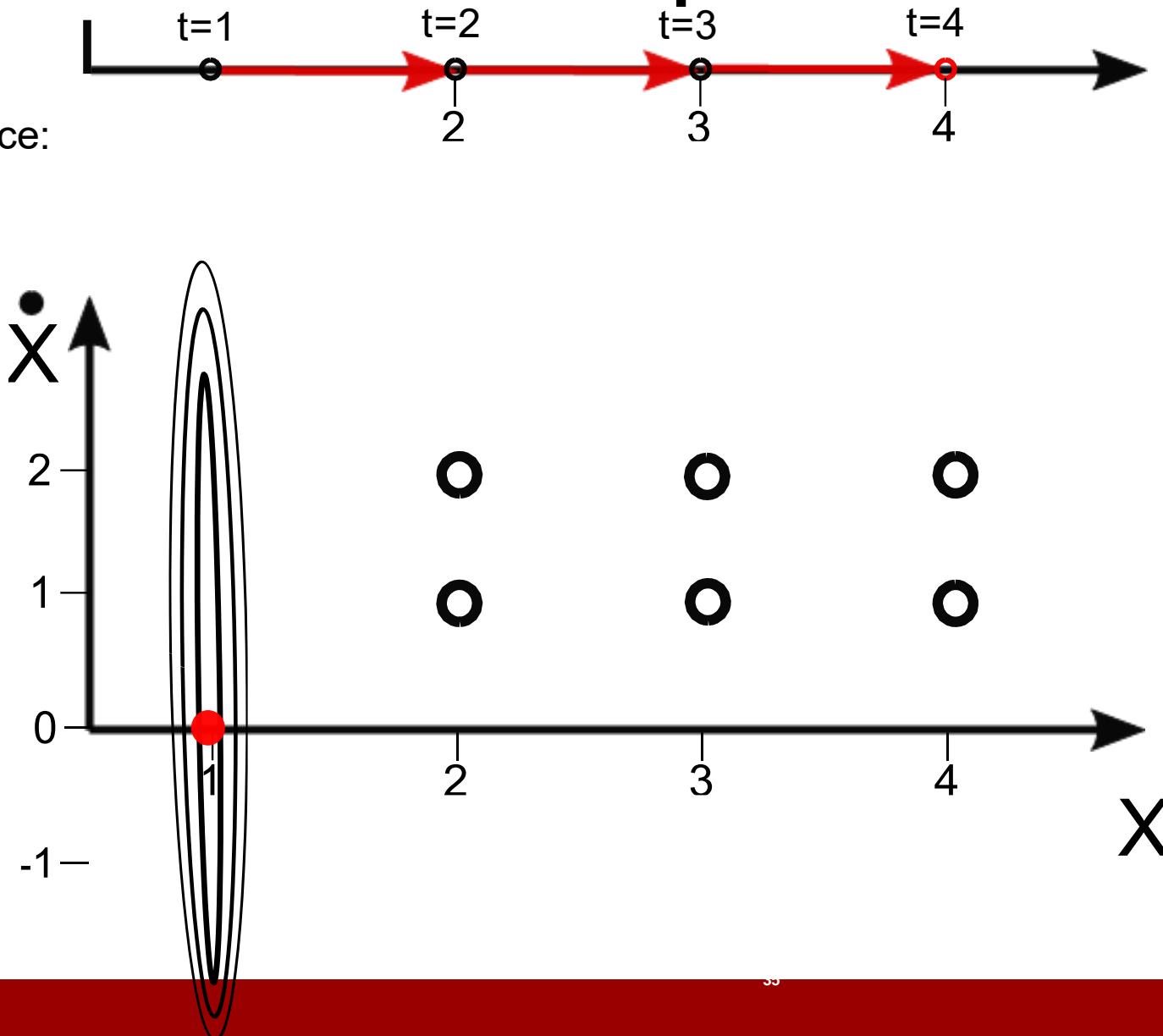
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1



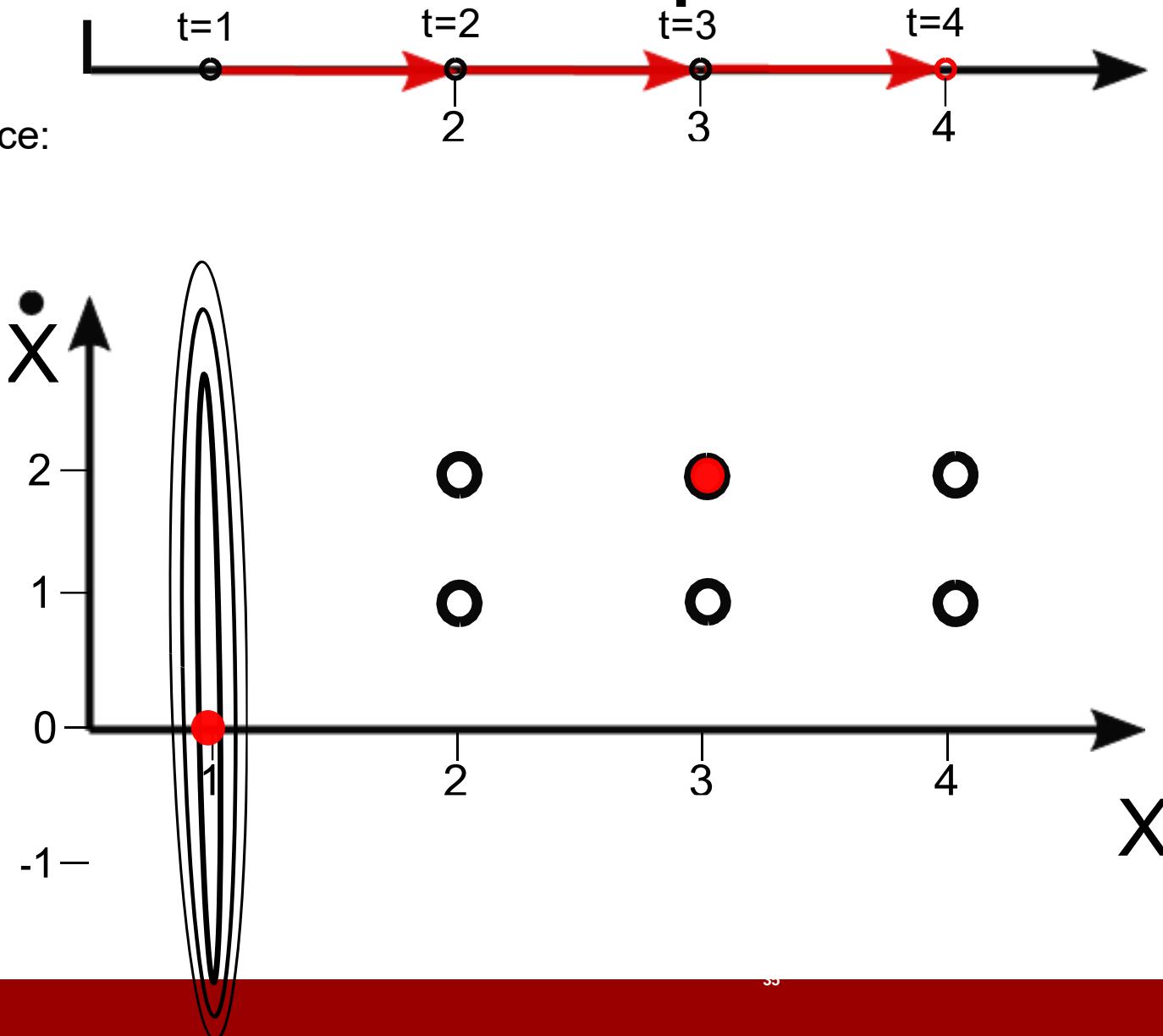
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1
  - Velocity: 2



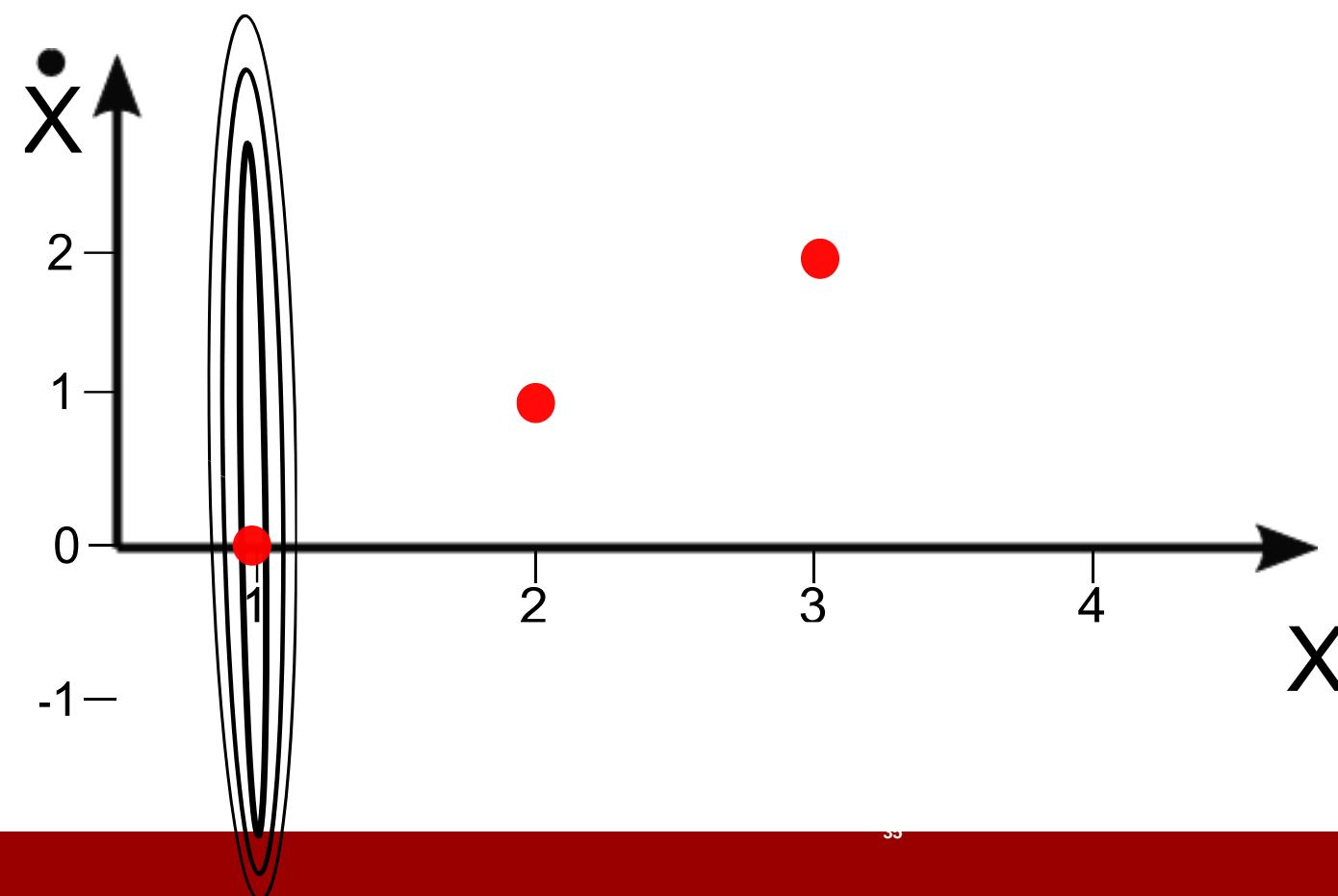
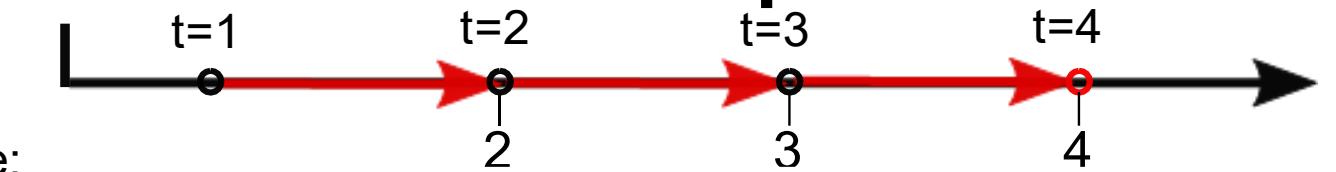
# Multivariate Gaussians - Kalman State Space

- 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

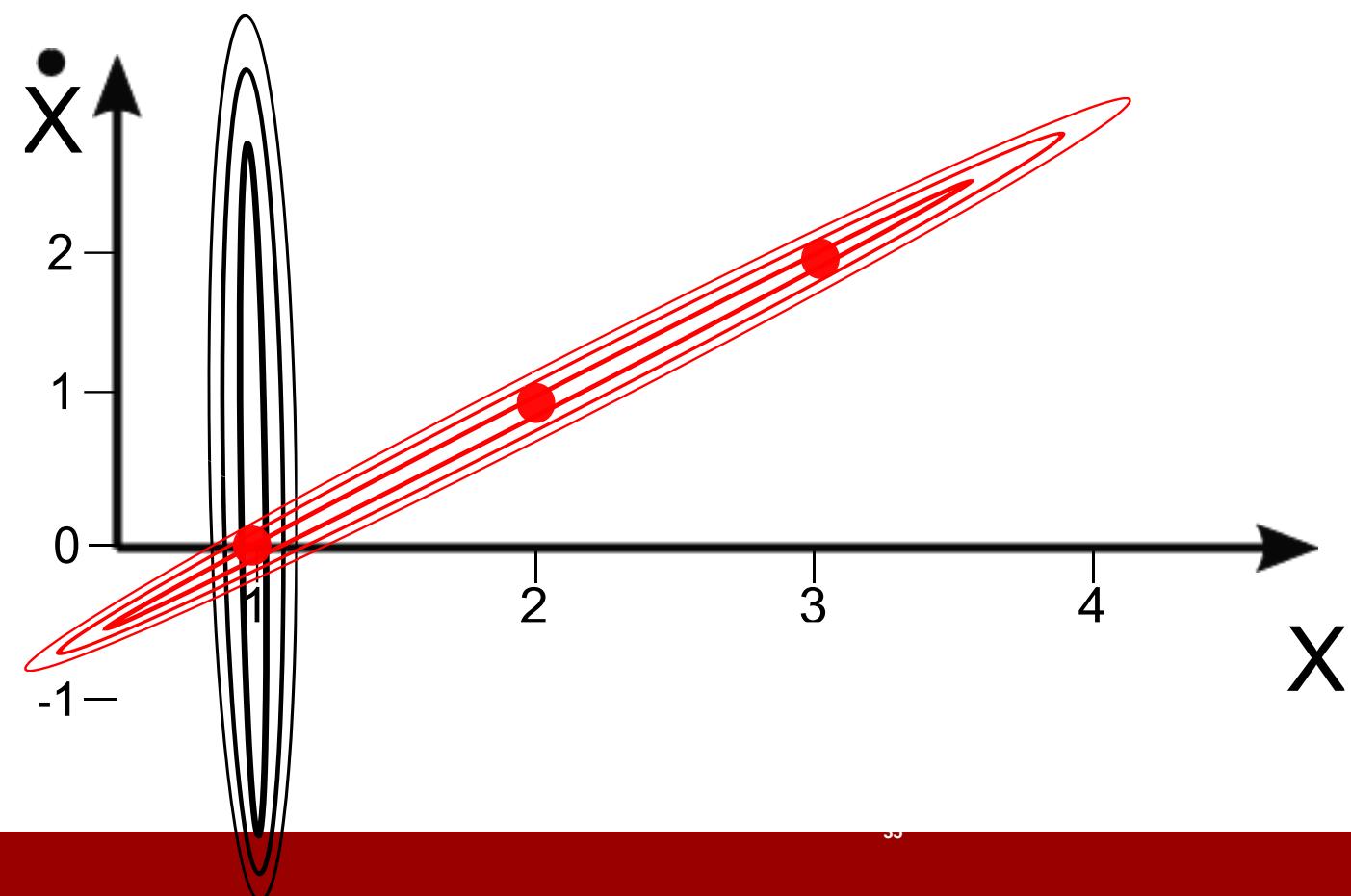
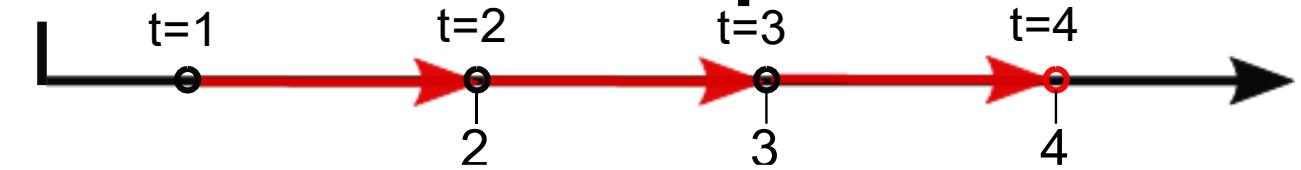
- Prediction:

- Velocity: 0

- Velocity: 1

- Velocity: 2

Consider this Gaussian



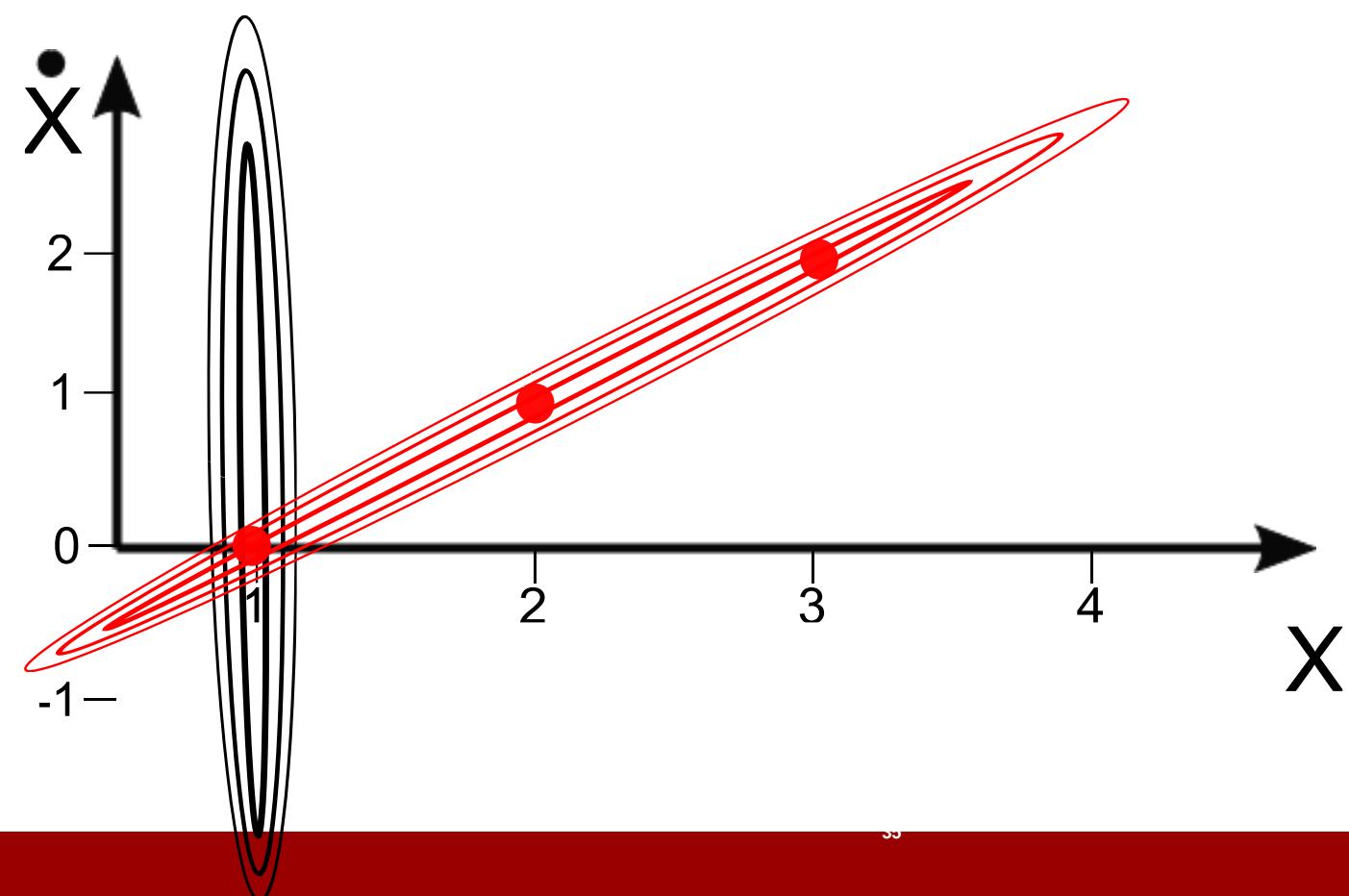
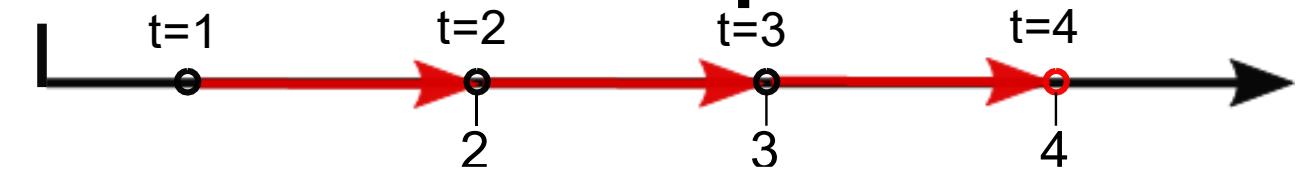
# Multivariate Gaussians - Kalman State Space

- Let's go at the Kalman state space:

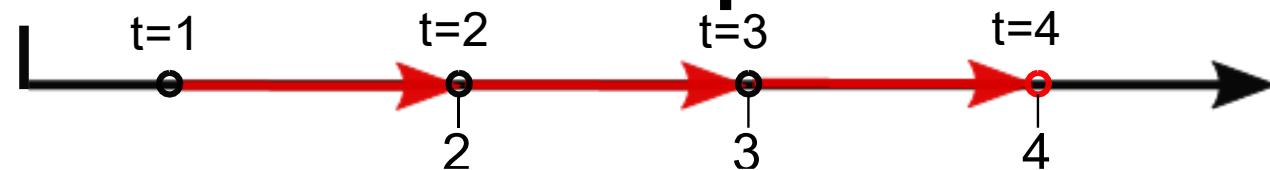
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1
  - Velocity: 2

Consider this Gaussian

- Measurement:
  - $Z = 2$



# Multivariate Gaussians - Kalman State Space

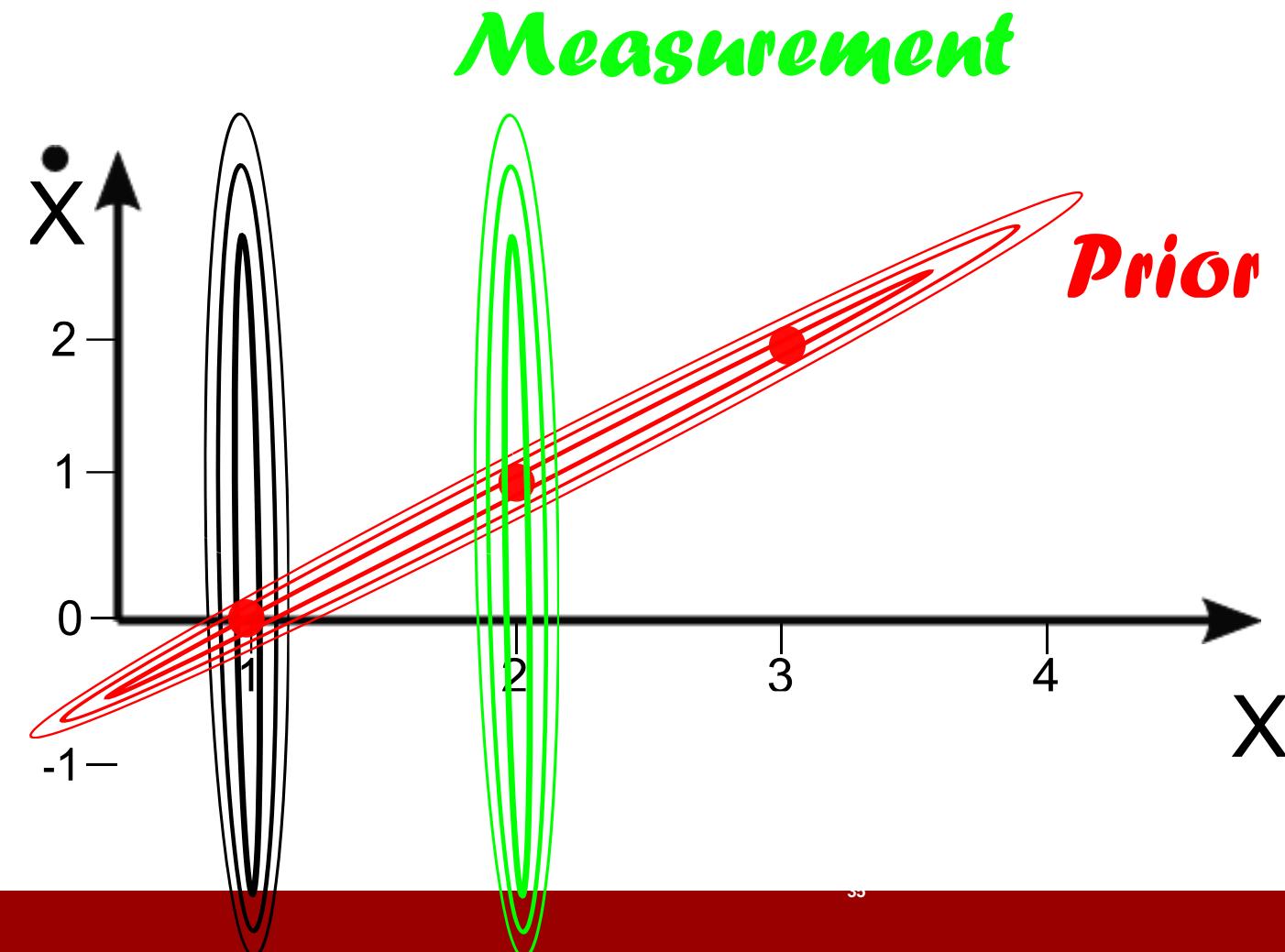


- Let's go at the Kalman state space:

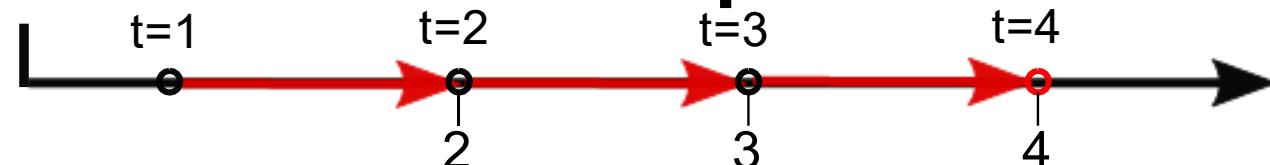
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1
  - Velocity: 2

Consider this Gaussian

- Measurement:
  - $Z = 2$



# Multivariate Gaussians - Kalman State Space

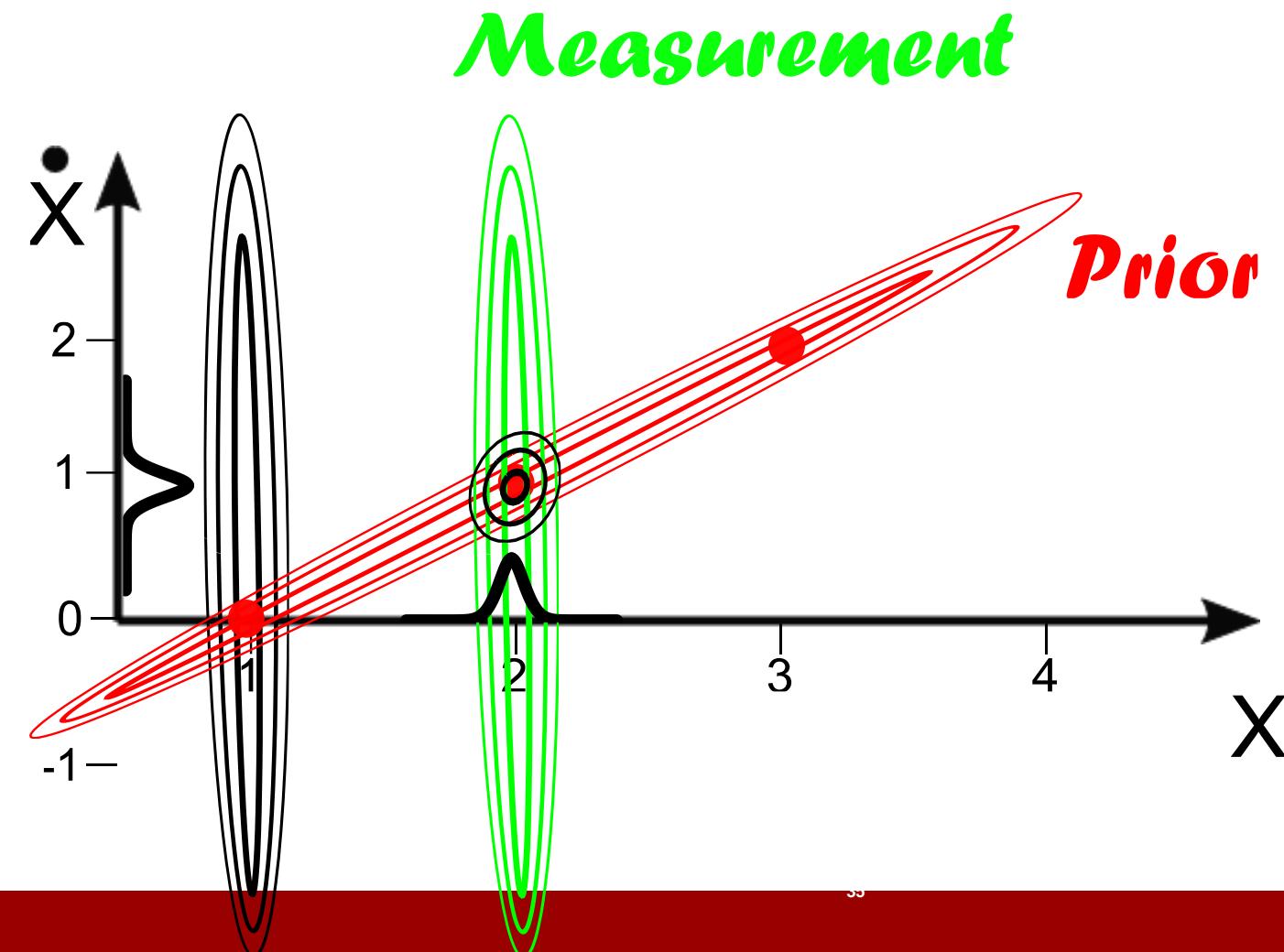


- Let's go at the Kalman state space:

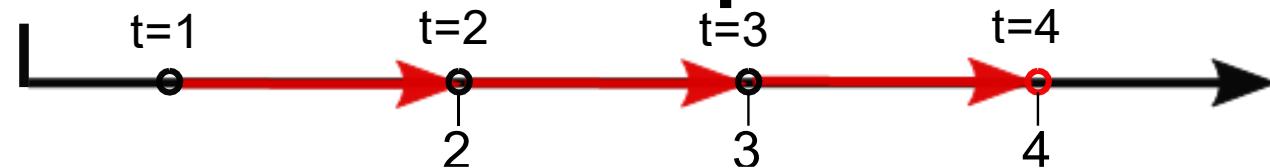
- Prior:
  - Location: 1
- Prediction:
  - Velocity: 0
  - Velocity: 1
  - Velocity: 2

Consider this Gaussian

- Measurement:
  - $Z = 2$



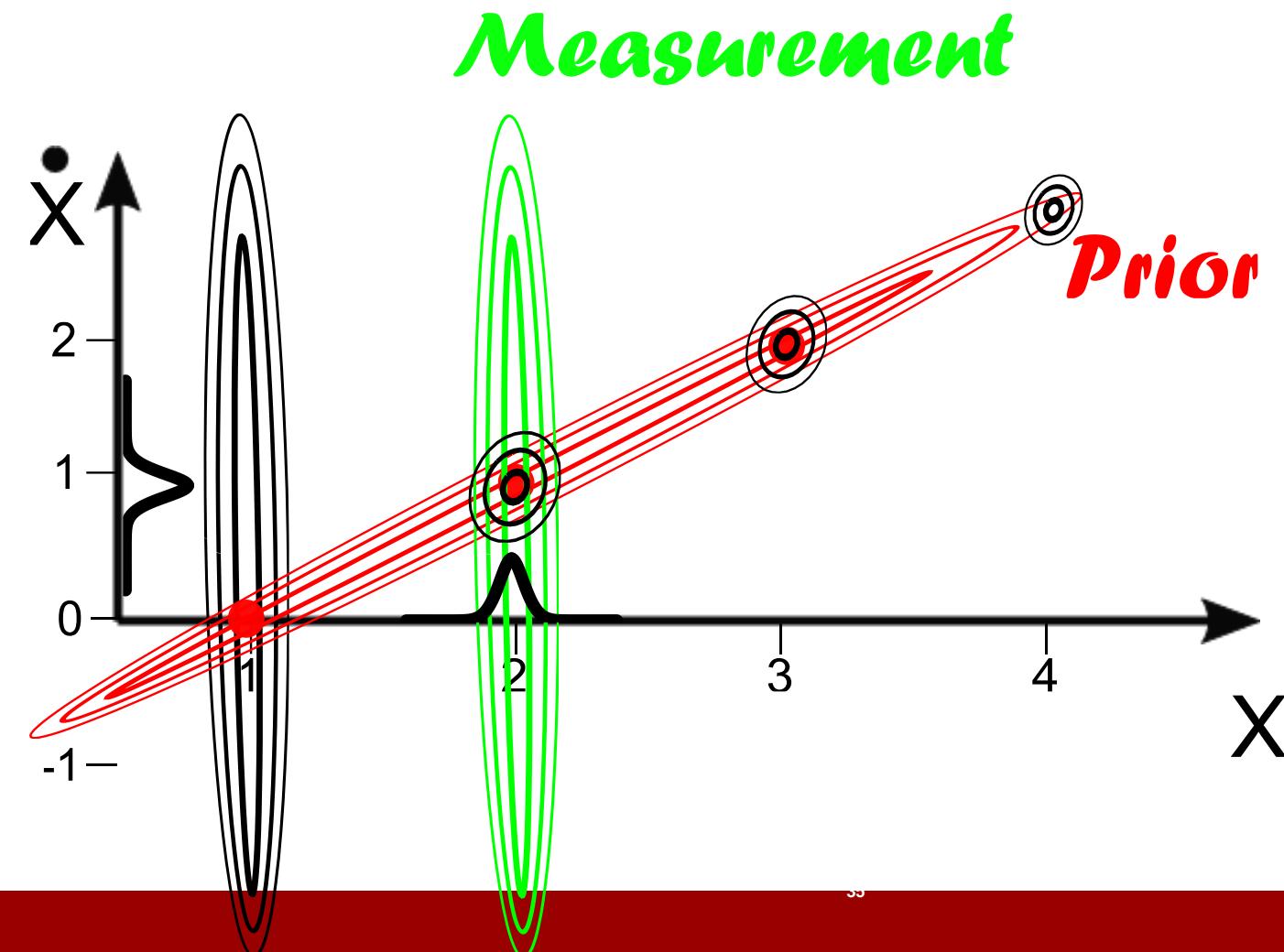
# Multivariate Gaussians - Kalman State Space



- 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

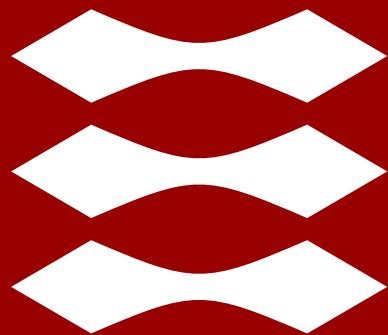
- We've acquired an amazing skill!!!
- We now understand what state estimation is
- We understand what a covariance represents
- We know how to track objects in space
  - (We can handle even occlusions)
- We know how to estimate our position (attitude) over time
- We'll come back on this (project) to do an end-to-end example!

Perception for Autonomous Systems 31392:

# State Estimation - Kalman Filter

Lecturer: Evangelos Boukas—PhD

**DTU**



Lazaros Nalpantidis

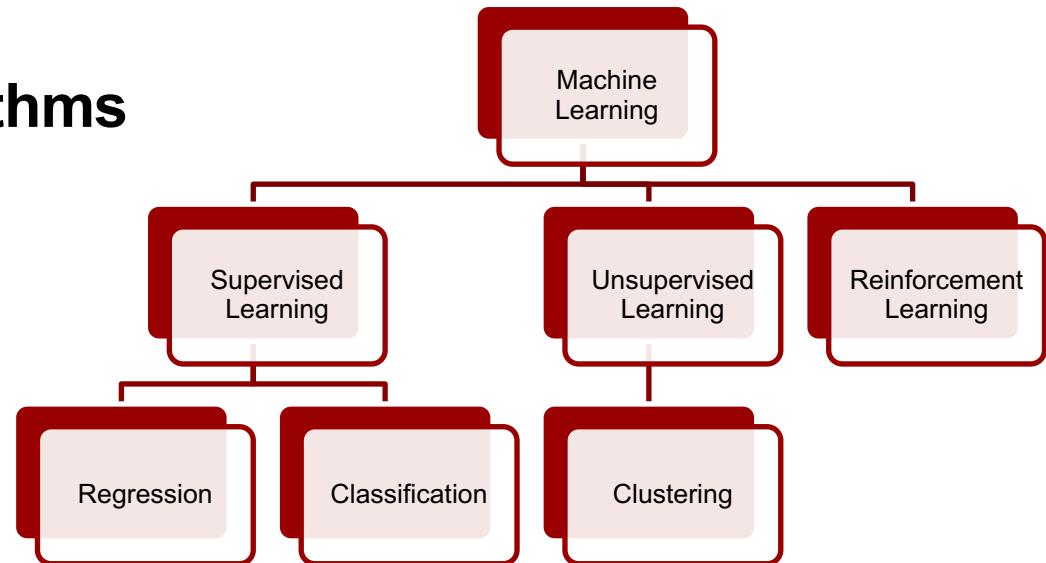
# Classification

- 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

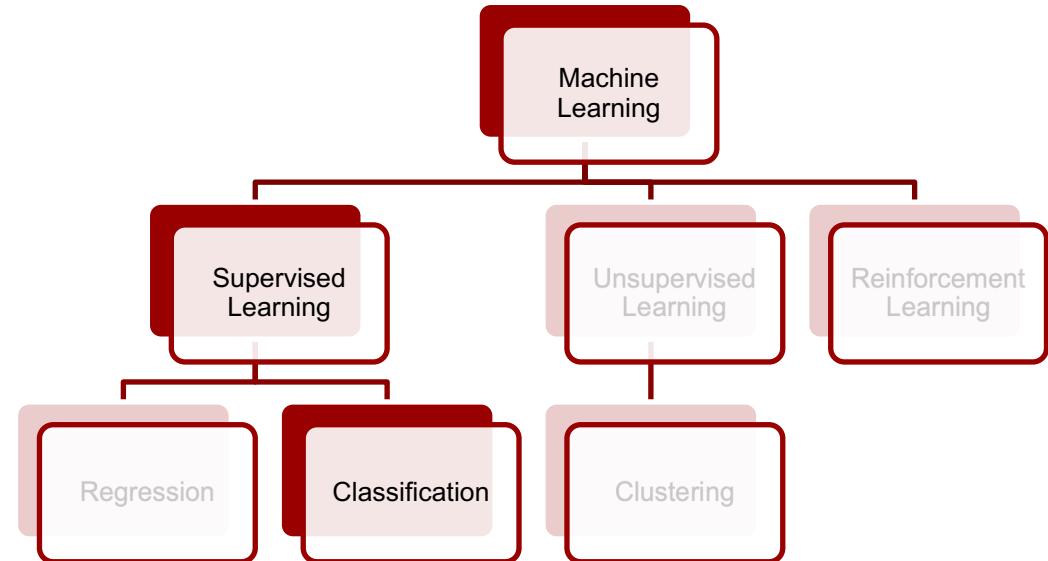
# What is Classification?

# 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

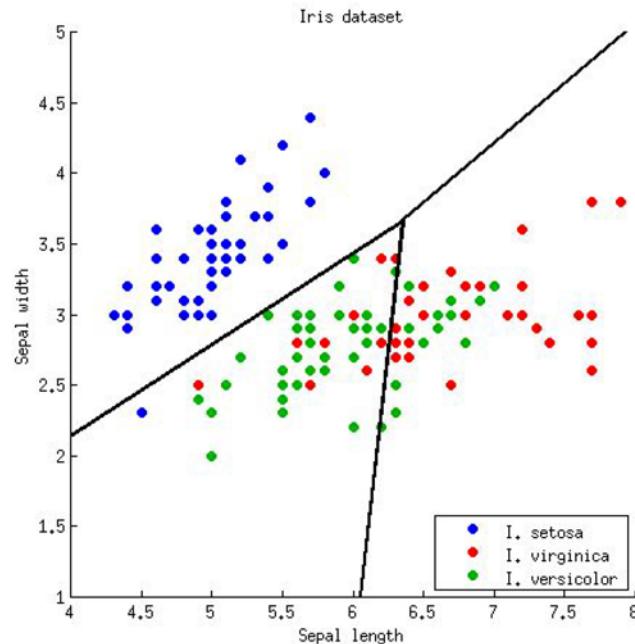
- Some terminology:
  - **Concept:** The description of the dataset. What can be learned by the examined dataset (e.g. the labels)
  - **Instances:** Individual and independent examples of the concept (e.g. an image)
  - **Attributes:** The features/dimensions that describe each instance.

	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).

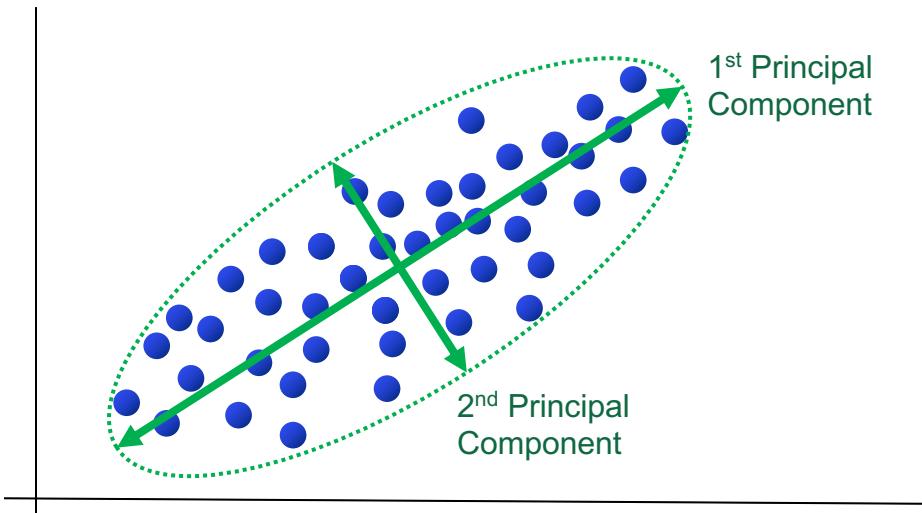
6. Keep eigenvectors that explain most (e.g. 95%) variance / remove the rest! ( $\leftarrow$  dimensionality reduction)

7. Project instances to eigenvectors y.

$$y = W^t x$$

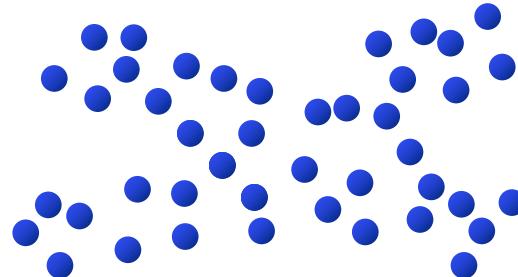
# Dimensionality Reduction

- 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.*

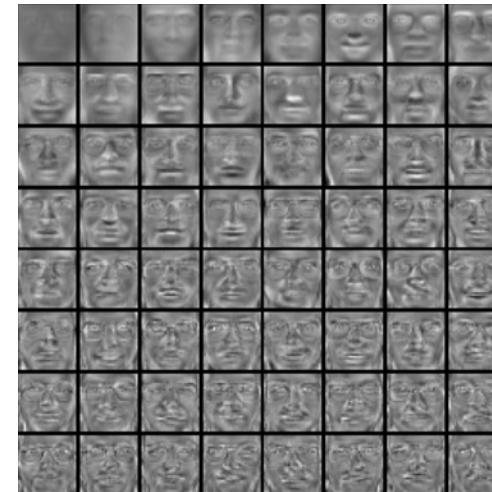
Large dataset of face images



Mean Image



The first 64 eigenvectors (eigenfaces)



PCA

- 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.

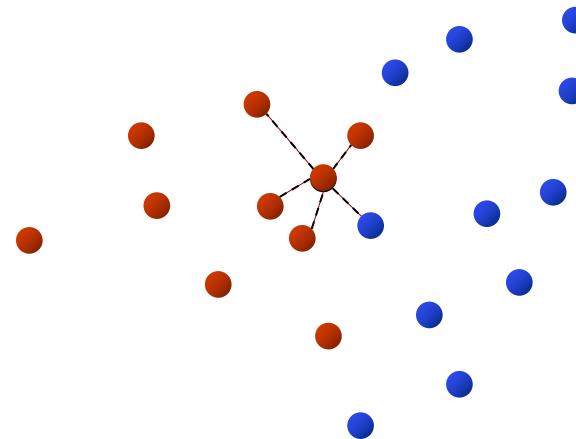
For classifying a new instance  $x_*$  :

- 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)

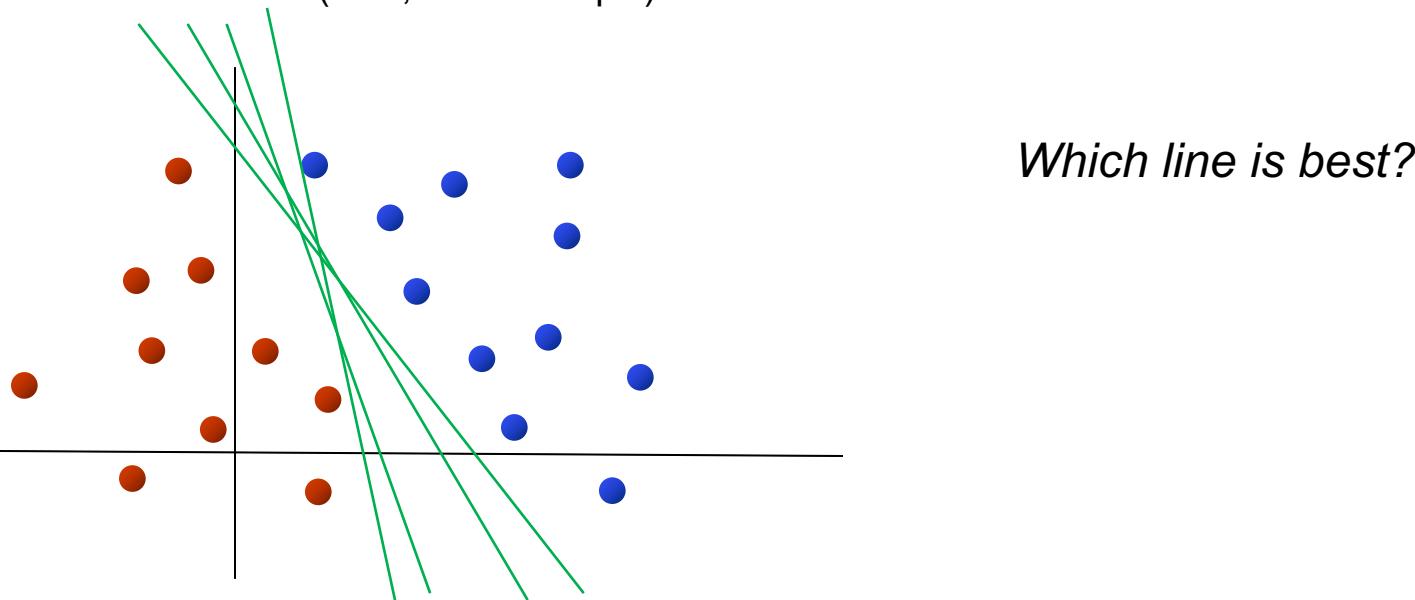
Characteristics of k-NN:

- 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.

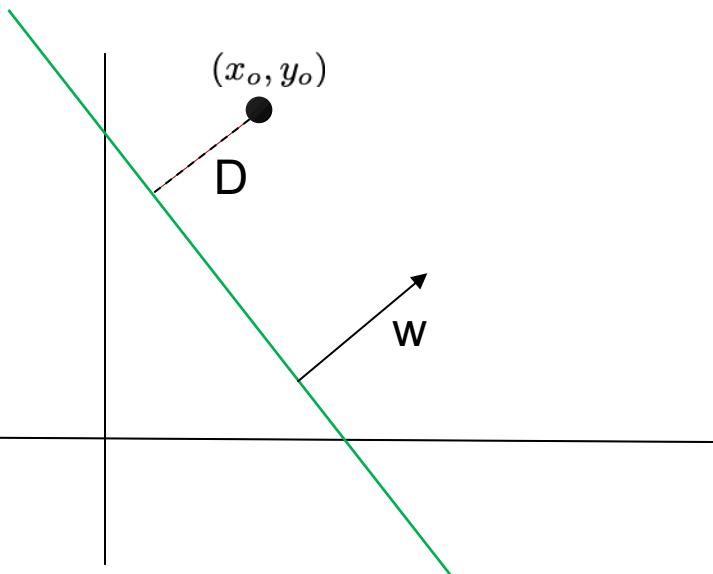
- What is Classification?
- Dimensionality Reduction
  - PCA
  - Face detection
- Classifiers
  - k-Nearest Neighbors
  - Support Vector Machines
- Class/Category Recognition
- Summary

# 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



$$\mathbf{w} = \begin{bmatrix} p \\ q \end{bmatrix}$$
$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$px + qy + b = 0$$

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0$$

$$D = \frac{|px_0 + qy_0 + b|}{\sqrt{p^2 + q^2}} = \frac{\mathbf{w}^T \cdot \mathbf{x} + b}{|\mathbf{w}|}$$

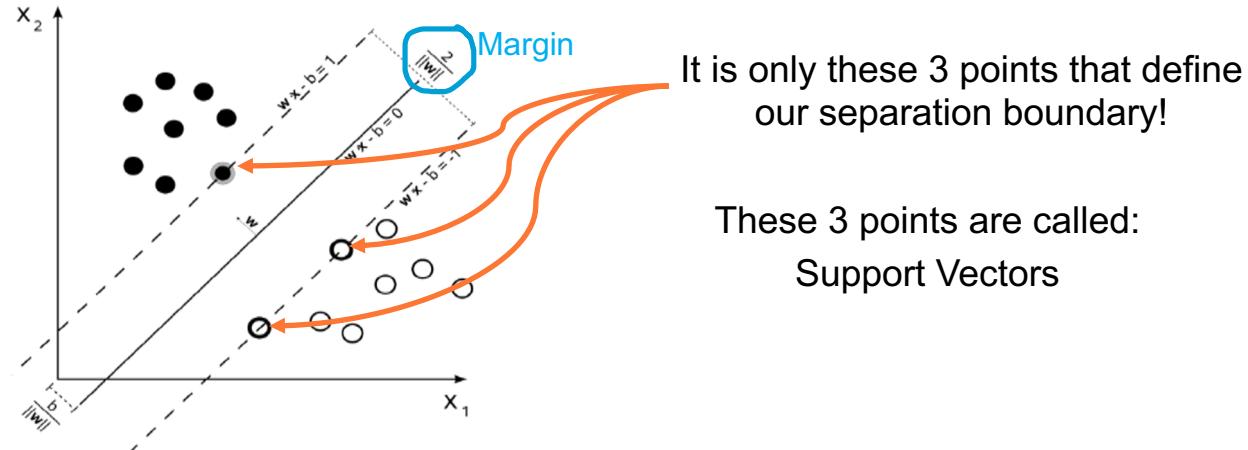
# Support Vector Machines (SVMs)

- 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.

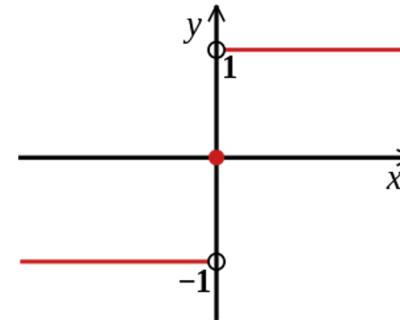
- SVMs are geometric methods for classification
- They derive a linear decision boundary which satisfies two criteria:
  1. The distance between the instances and the boundary is as large as possible (max margin).
  2. The instances are classified as good as possible.



# 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 demonstrated that the message about the image falling on the retina undergoes 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.

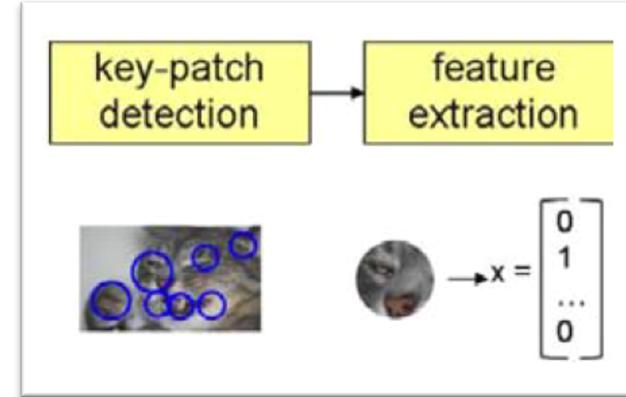


China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% increase in exports to \$750bn, compared with \$660bn. The Chinese government, annoyed that China's trade surplus, commerce, exports, imports, US, yuan, bank, domestic, foreign, increase, trade, value



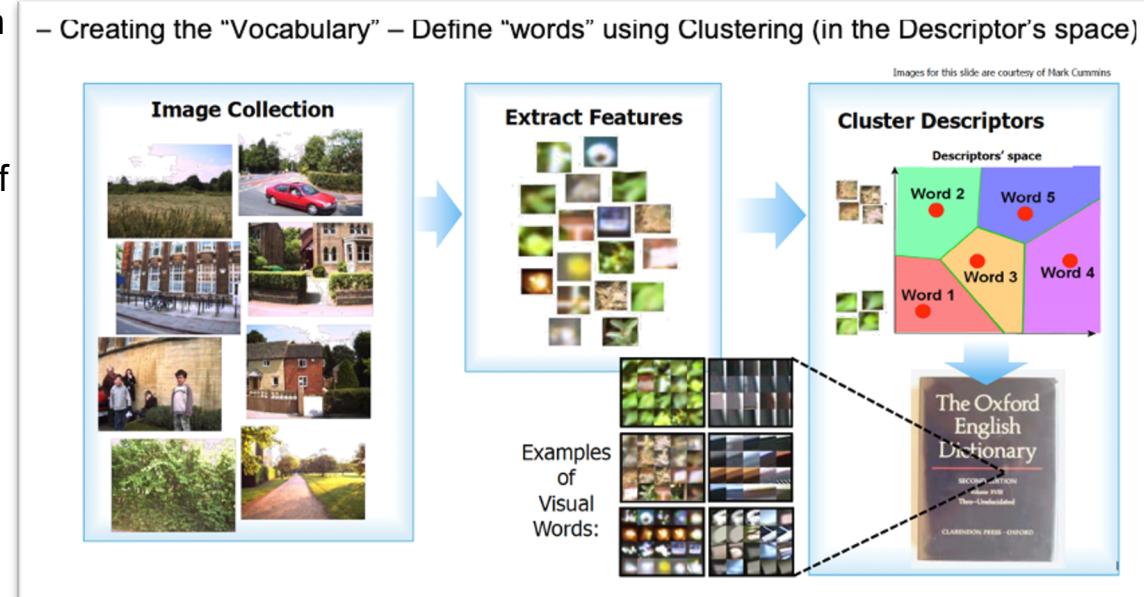
# Class/Category Recognition

- **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



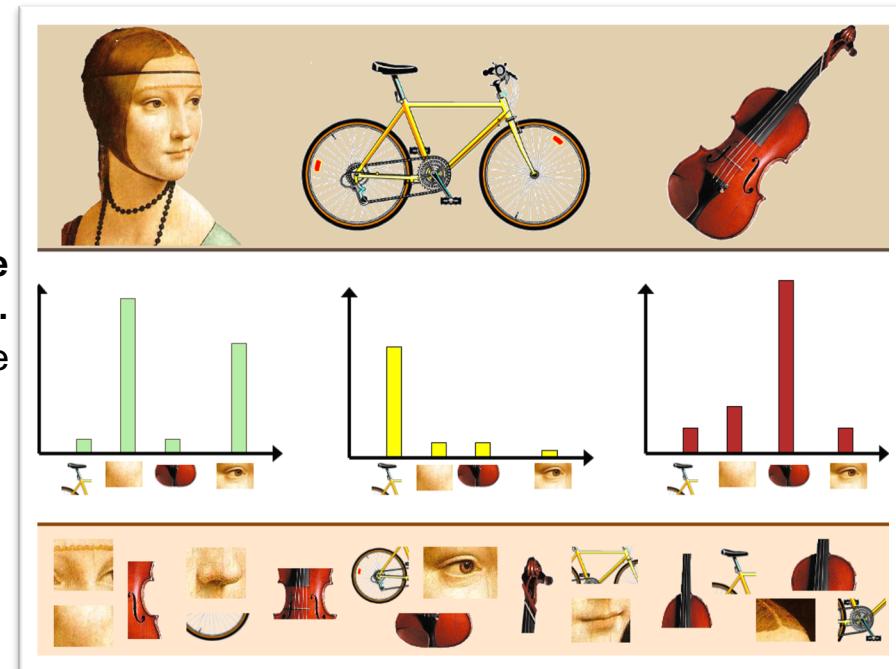
# Class/Category Recognition

- **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



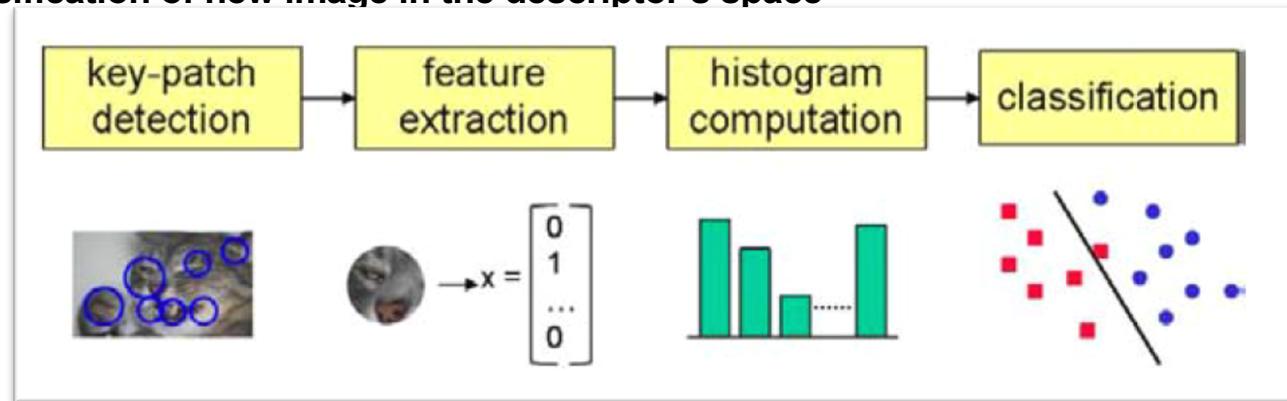
# Class/Category Recognition

- **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



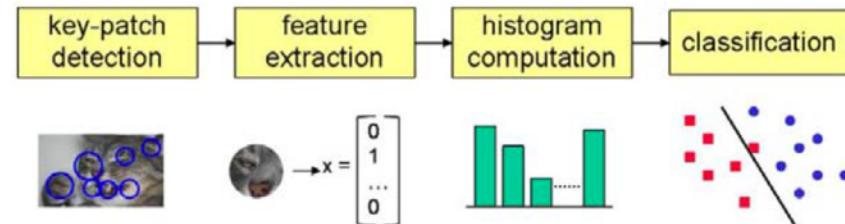
# Class/Category Recognition

- **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**

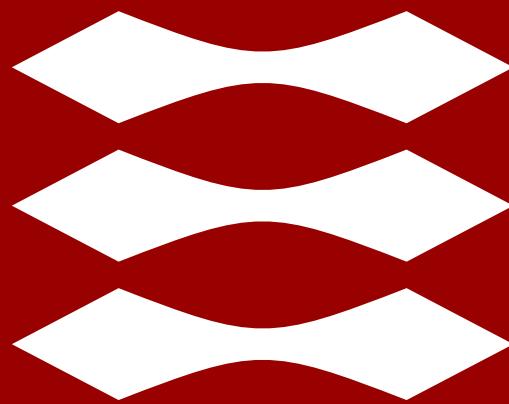
- PointNet

- » 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

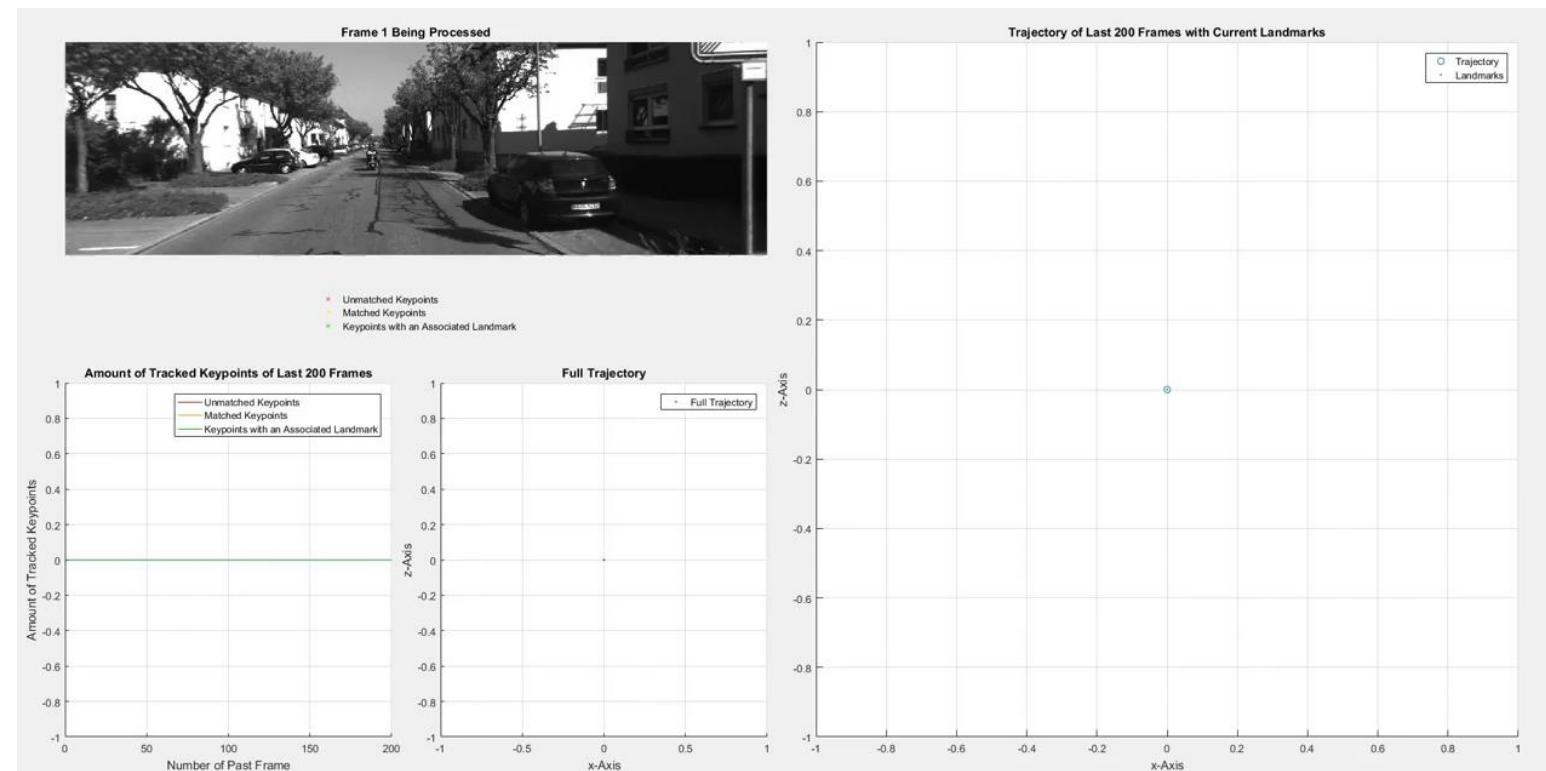
Lecturer: Evangelos Boukas—PhD

# 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

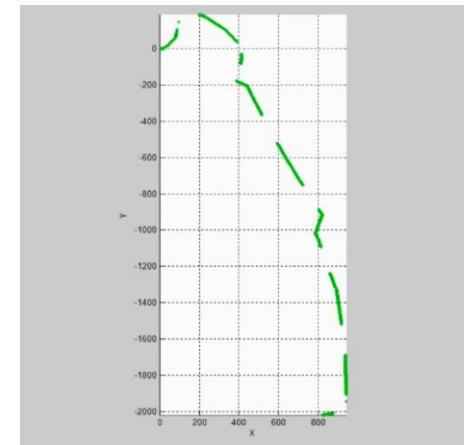
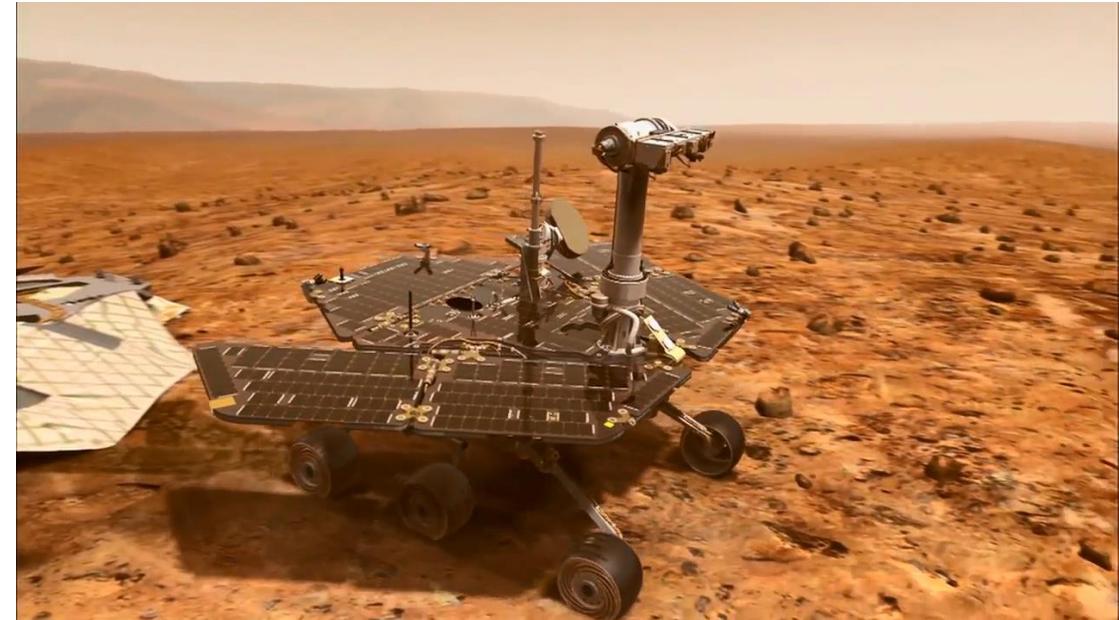
# What is Visual Odometry and what it is not

- 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*

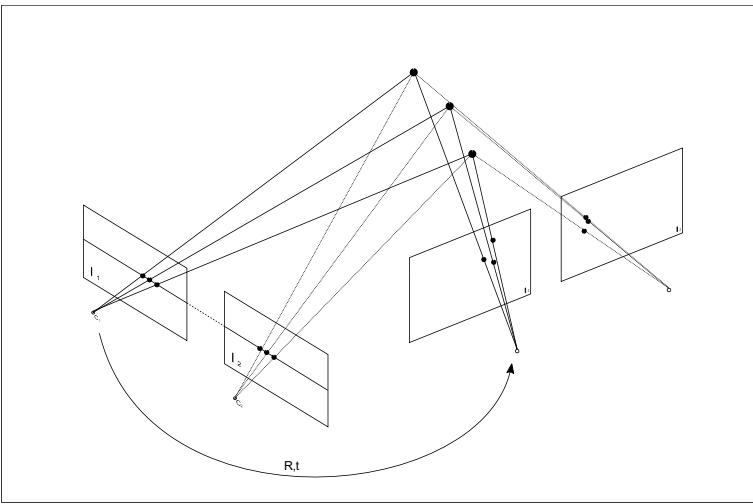
- It has been proposed as an alternative to Wheel odometry
- Its accuracy is –assuming reasonable trajectories- ~1%
- VO estimations are usually combined with other sensors
  - GPS, IMU, Laser, Wheel odometry
  - VINS, VIO stands for Visual Inertial Odometry
- VO has had some extraordinary usages



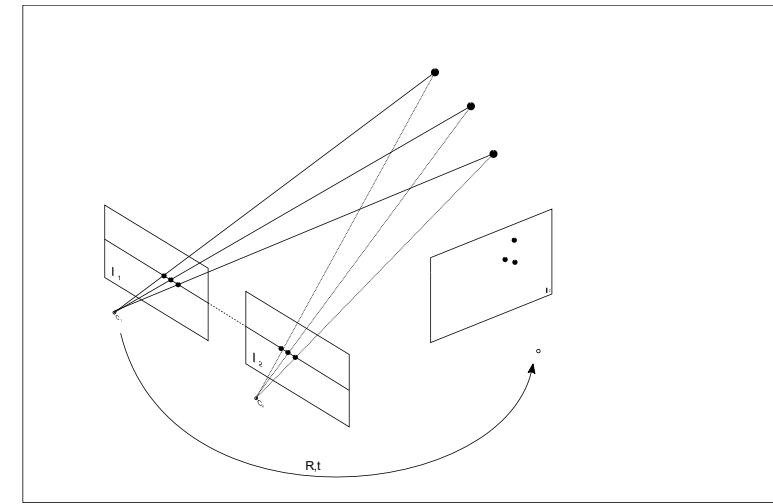
# VO is neither SFM nor VSLAM

- 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

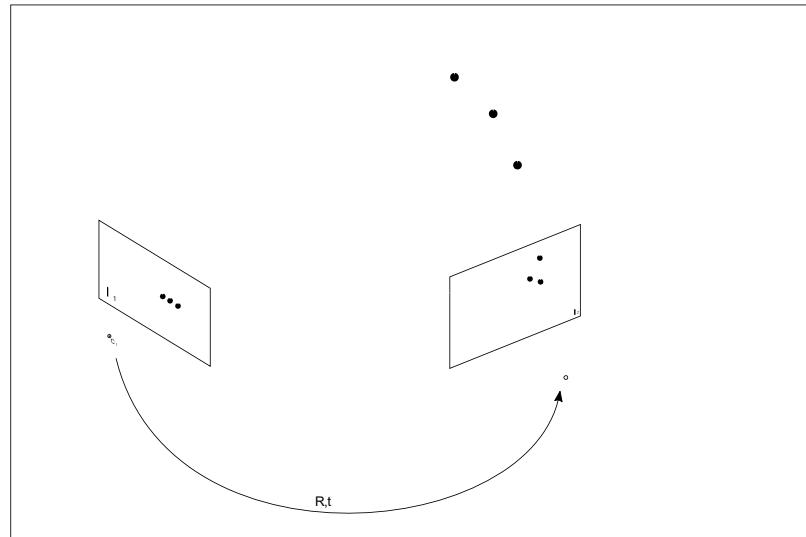
# The 3 main variants of VO



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

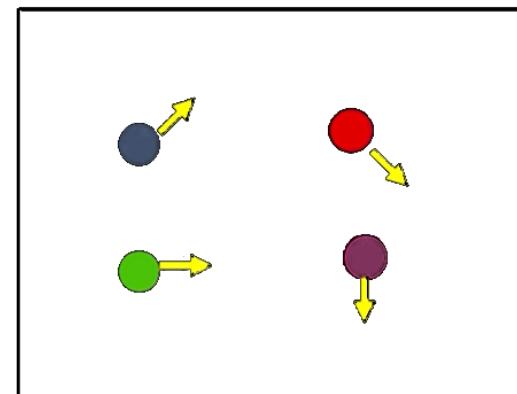
Shuster, M.D. (1993). "[A Survey of Attitude Representations](#)". *Journal of the Astronautical Sciences* **41** (4): 439–517. [Bibcode: 1993JAnSc..41..439S](#)

# Motion tracking

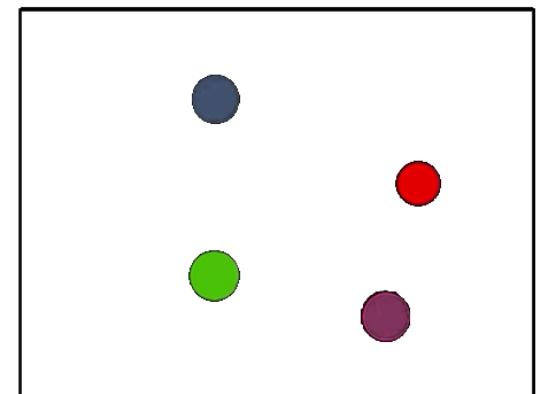
- Two main approaches
  - Feature based
  - Optic Flow
- Feature based:
  - Calculate feature on both images
  - Match among the features
- or
- Calculate features
- Do block Matching around our initial point (for small motion)

# 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



$I(x, y, t)$

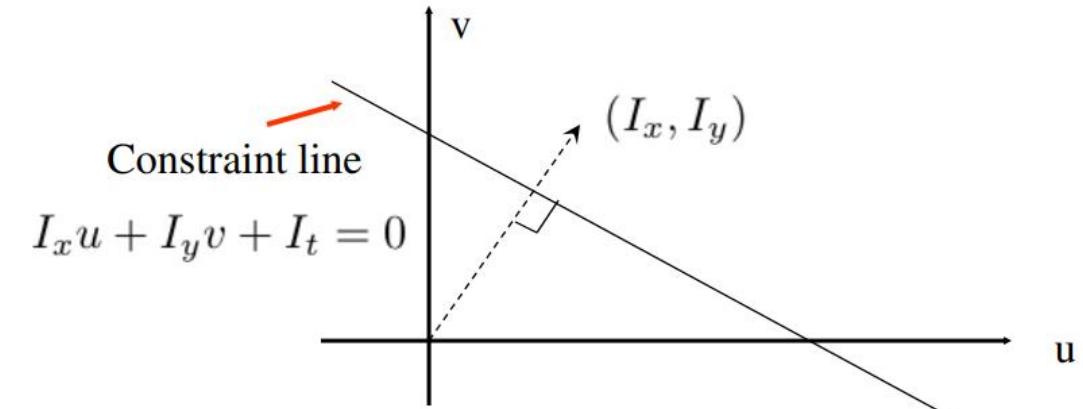


$I(x, y, t + 1)$

- $I(x + u, y + v, t + 1) \approx I(x, y, t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$
- $I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \rightarrow I_t + \nabla I \cdot (u, v) = 0 \rightarrow I_x u + I_y v + I_t = 0$
- 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

$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$A$   
 $25 \times 2$

$d$   
 $2 \times 1$

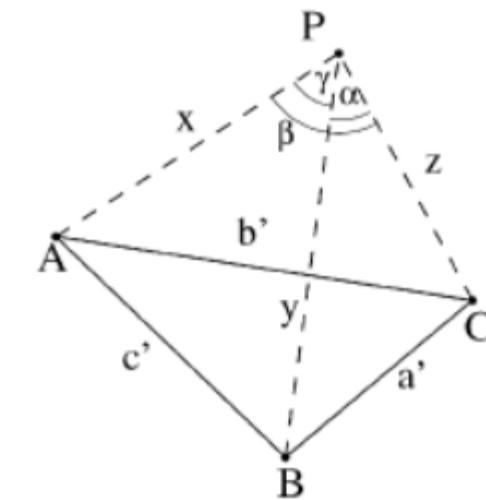
$b$   
 $25 \times 1$

# 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 4<sup>th</sup> point, check orientation consistency
- Many other implementations:
  - One of the most prominent is EPnP ( $n \geq 4$ )
    - Reformulate the problem with virtual “control points”



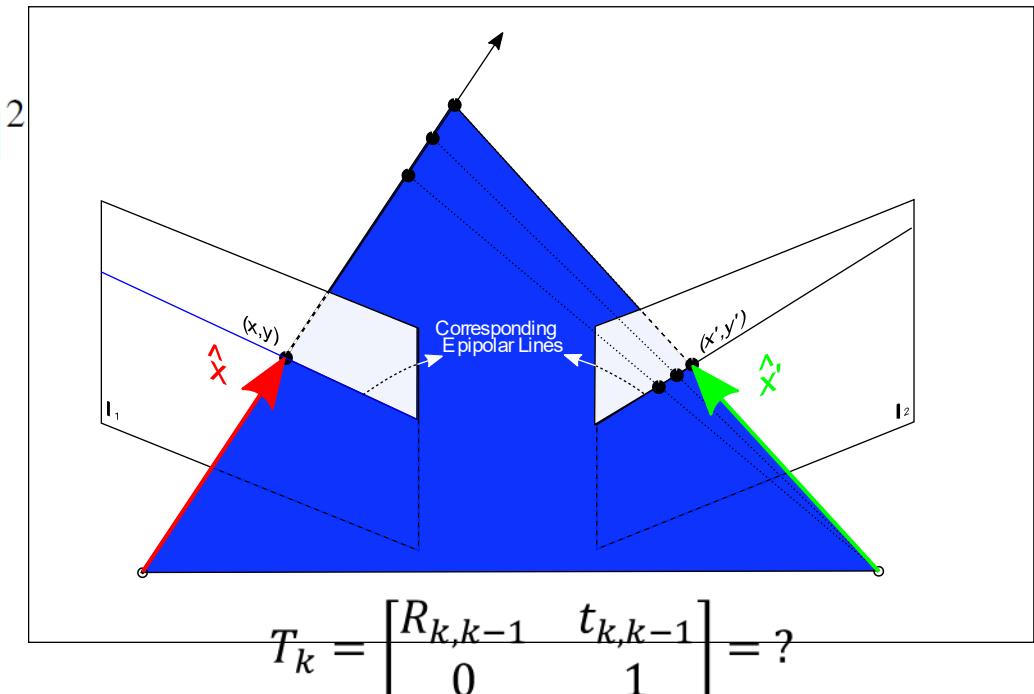
# Relative Pose estimation - 2D Image Points

- 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!

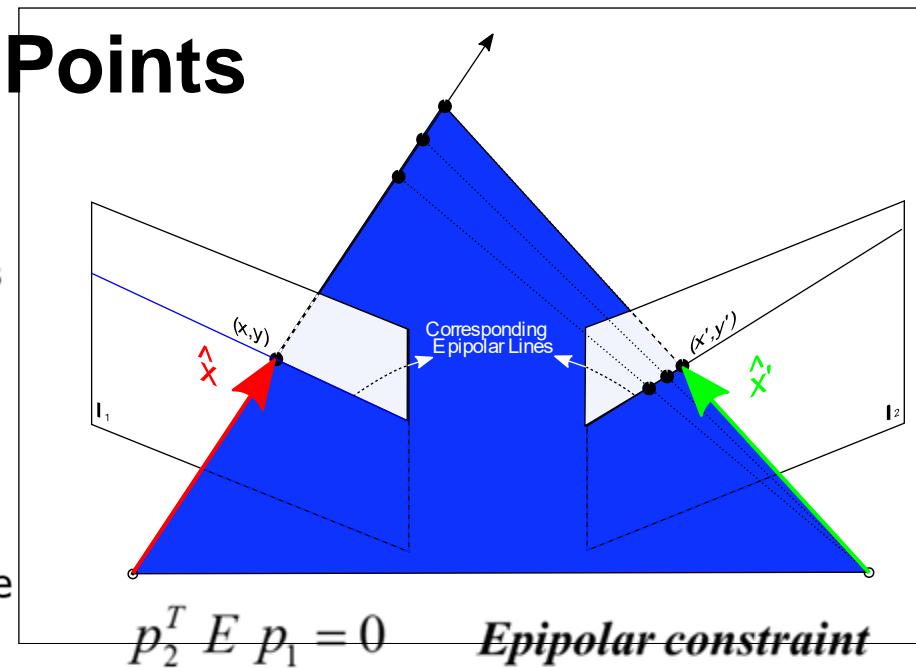
The Essential Matrix can be decomposed into  $R$  and  $t$  (again using SVD)

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



$$E = [t]_x R \quad \text{essential matrix}$$

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

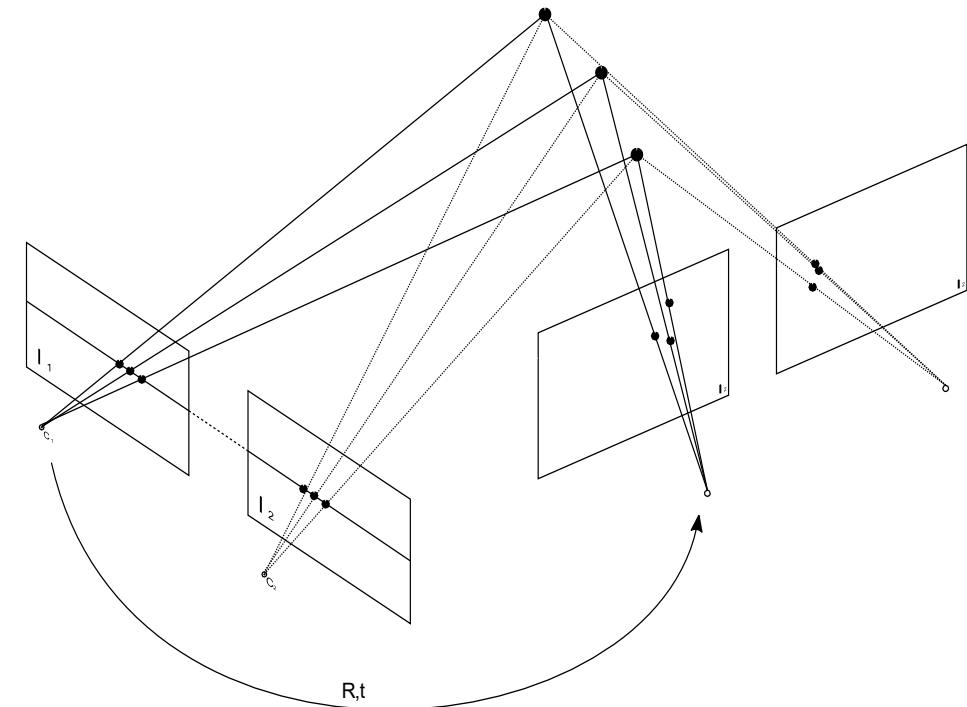
# Relative Pose estimation - 2D Image Points

- 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).

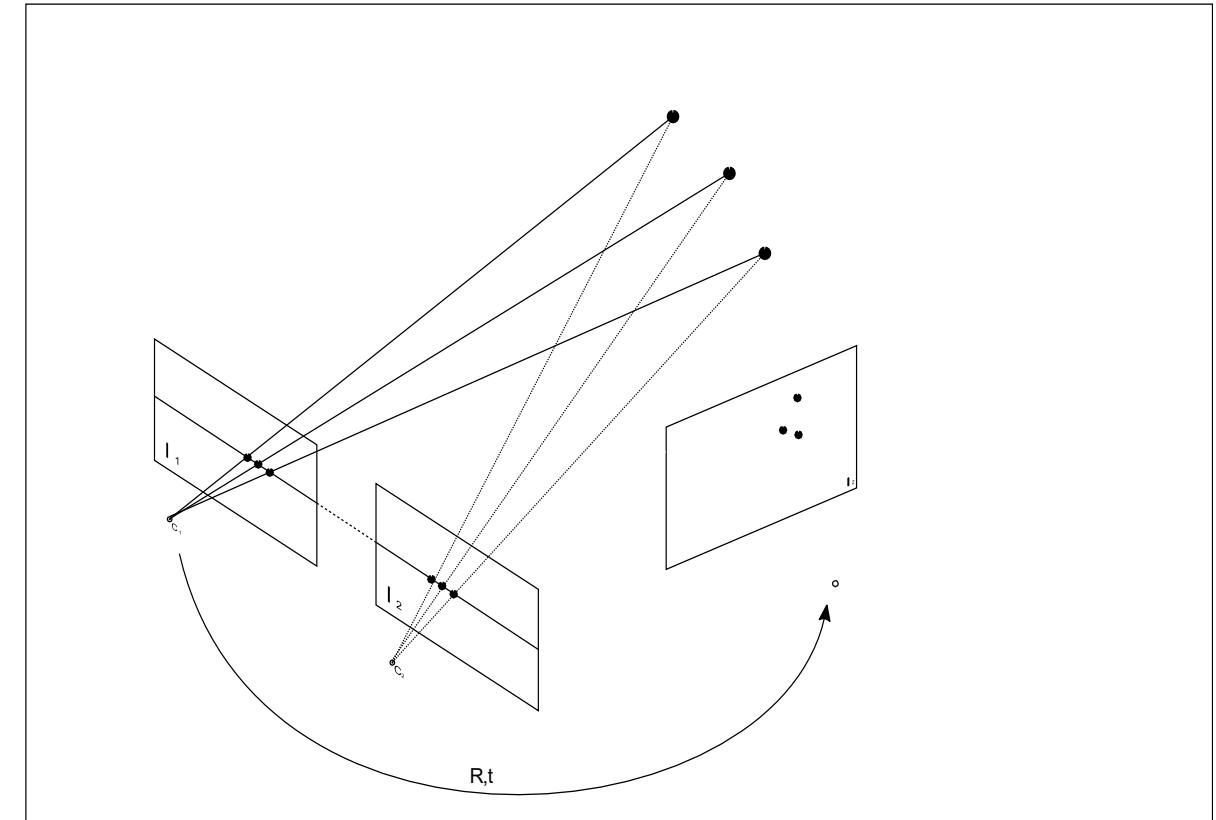


3D-3D

# 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).

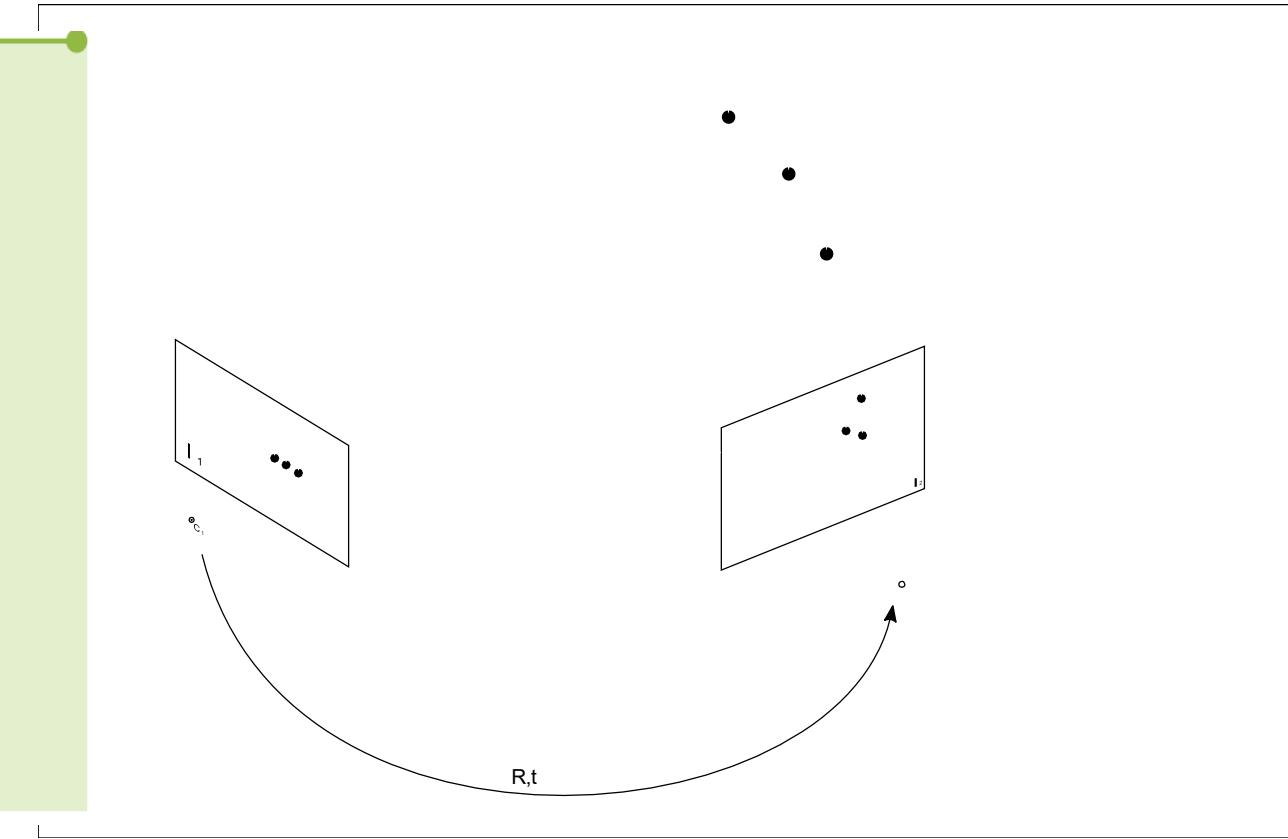


3D-2D

# Visual Odometry 2D – to 2D

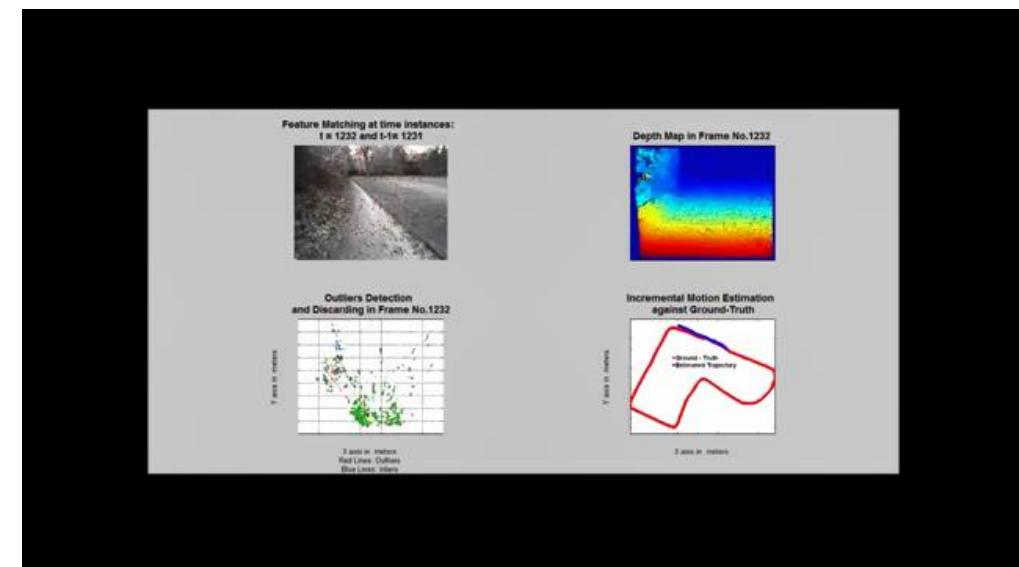
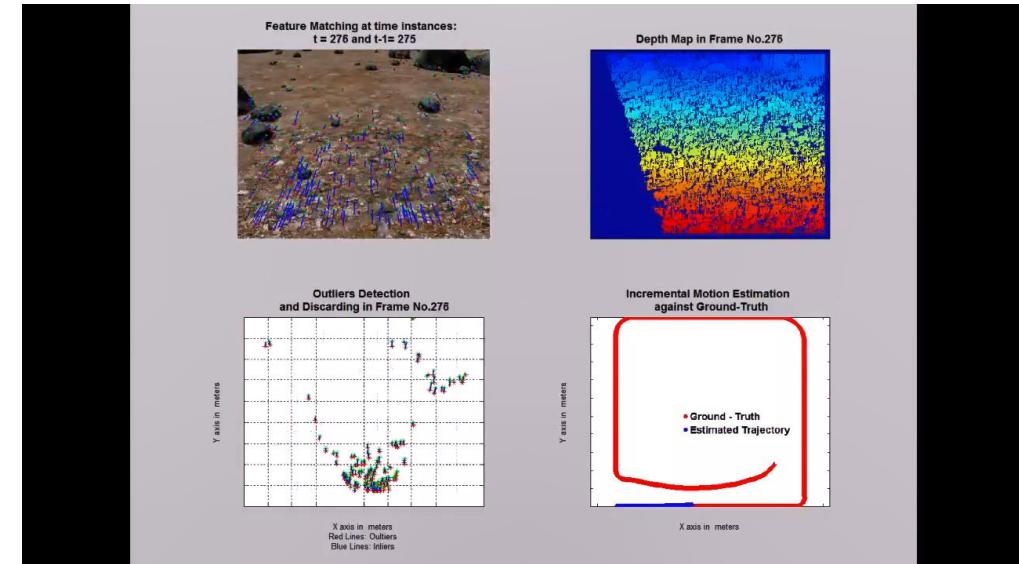
## Algorithm 1. VO from 2-D-to-2-D correspondences.

- 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

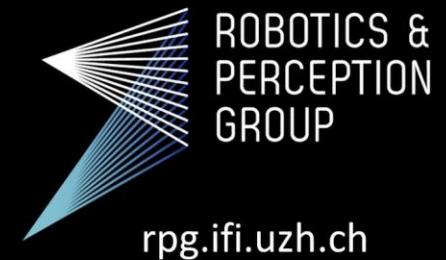
Shing Yan Loo, Ali Jahani Amiri,

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



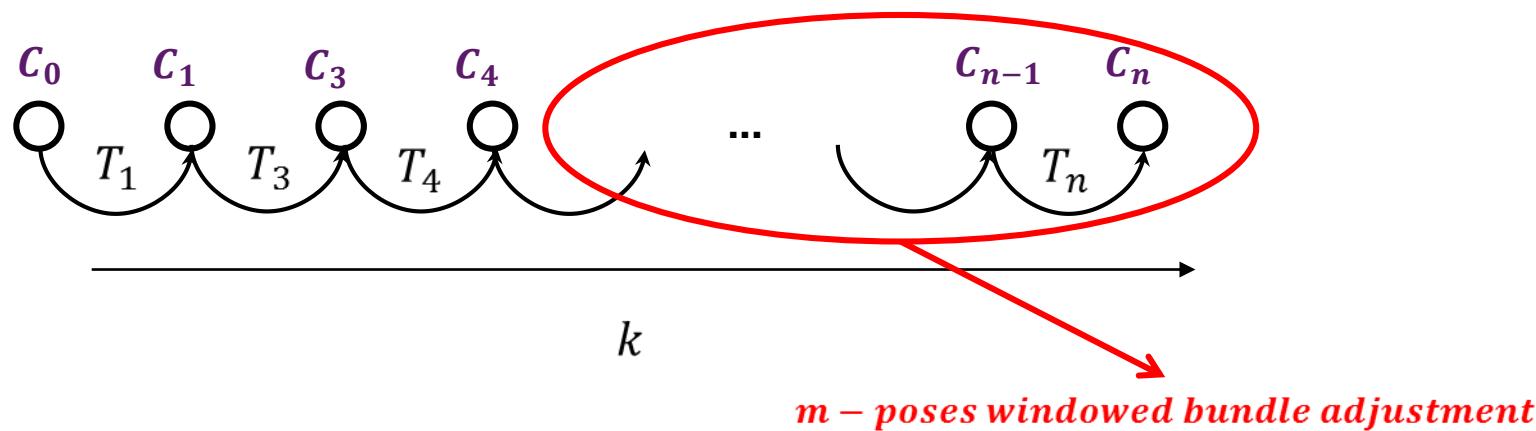
University of  
Zurich<sup>UZH</sup>

Department of Informatics

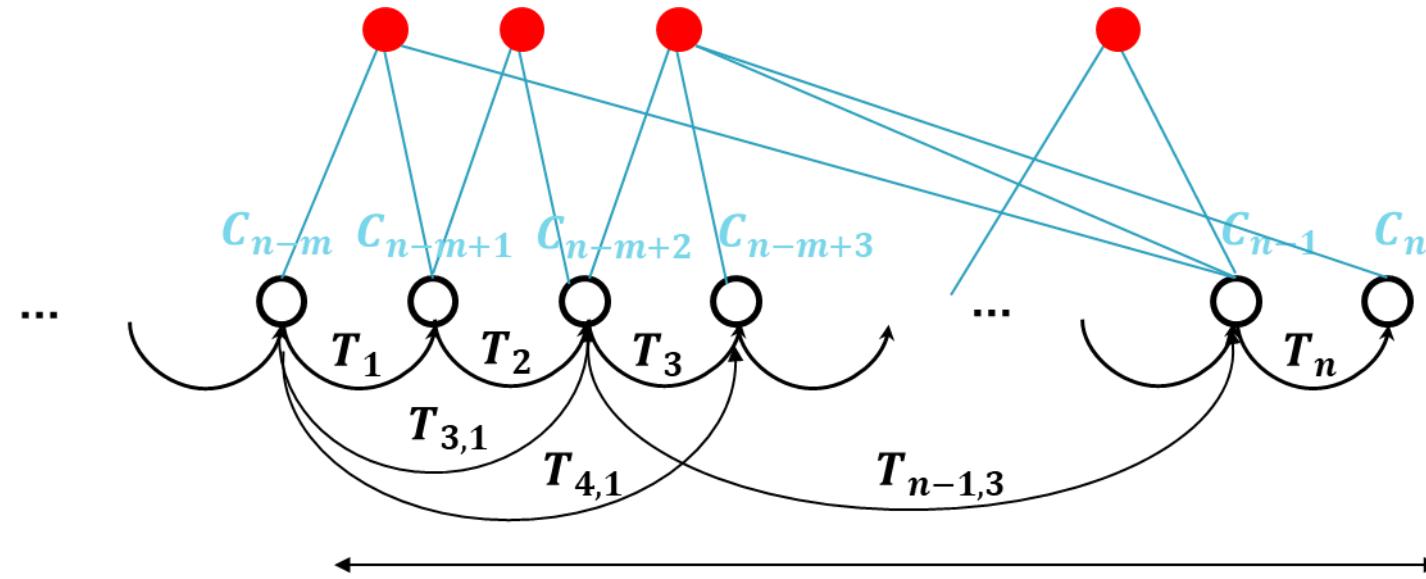


# What happens over time?

- 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

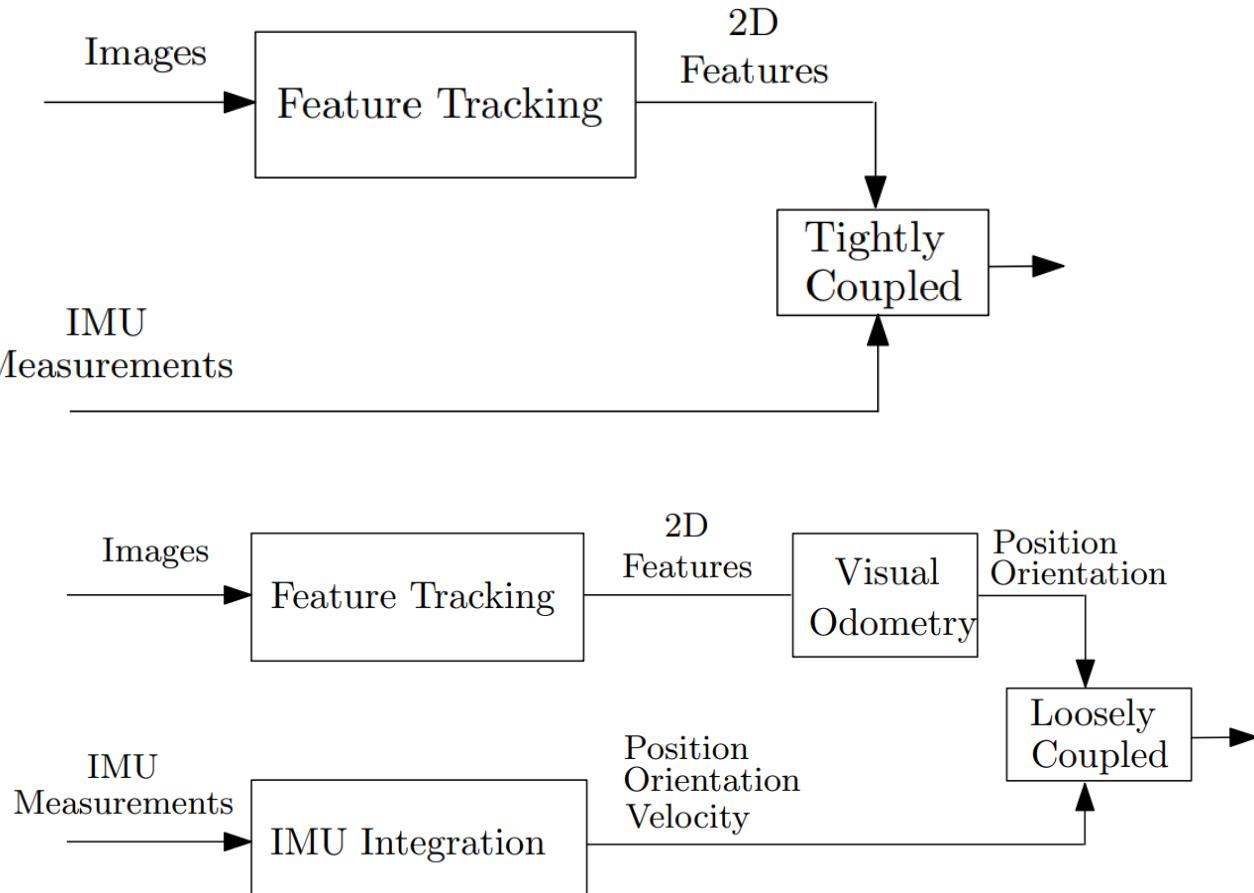
# Improving the Accuracy of VO

- 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

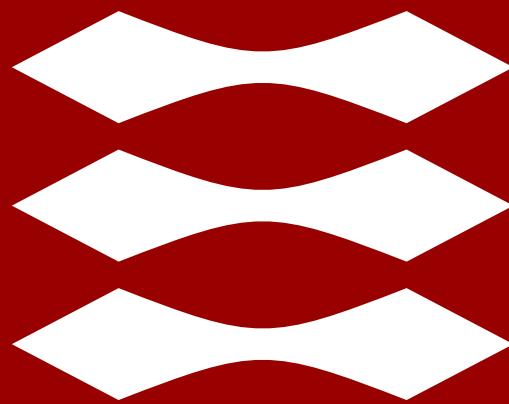
- Visual Odometry
  - 3D-3D
  - 3D-2D
  - 2D-2D
- Local Bundle Adjustment
- Visual Inertial Odometry -VIO
  - Loosely Coupled EKF
  - Tightly Coupled EKF

Perception for Autonomous Systems 31392:

# Visual Odometry

Lecturer: Evangelos Boukas—PhD

**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

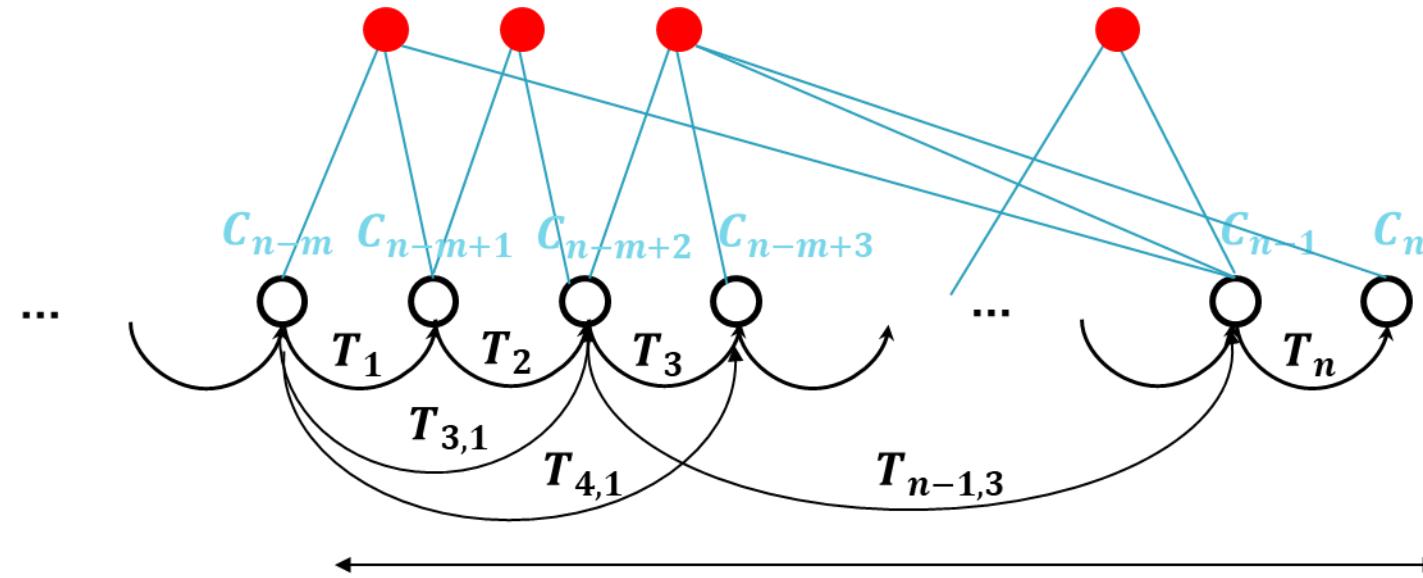
- Lets Sum Up What we did last time
- We performed 3D-to-2D



# Visual Odometry is great

- Anything more we mentioned?
  - Windowed Bundle Adjustment (BA)

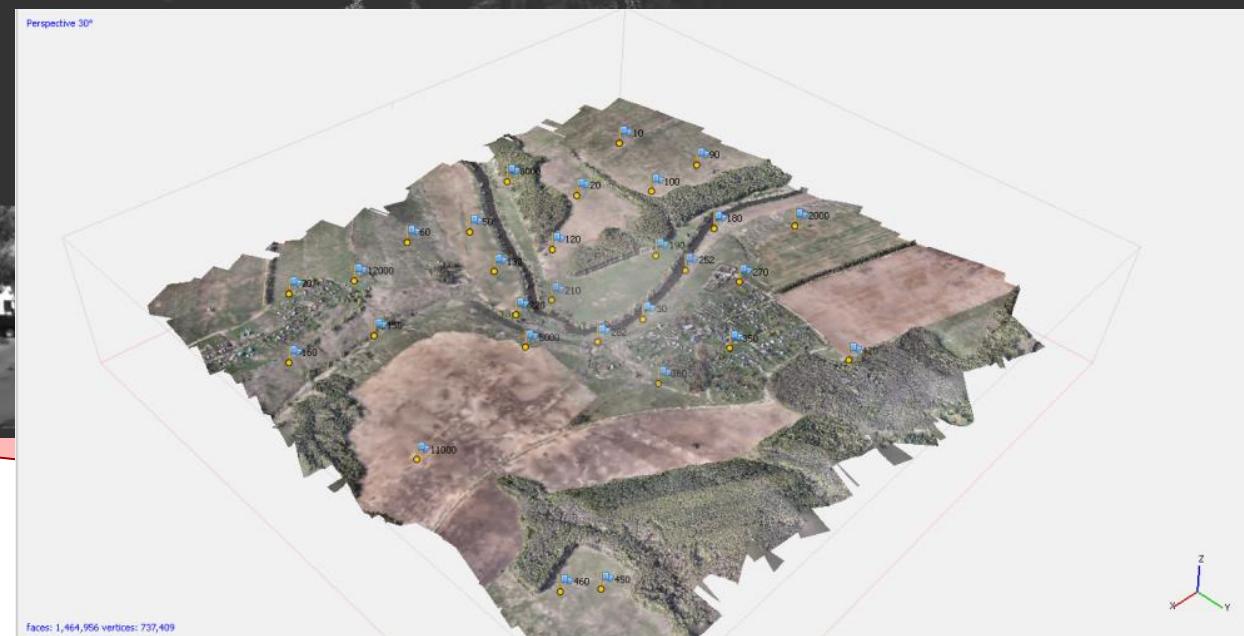
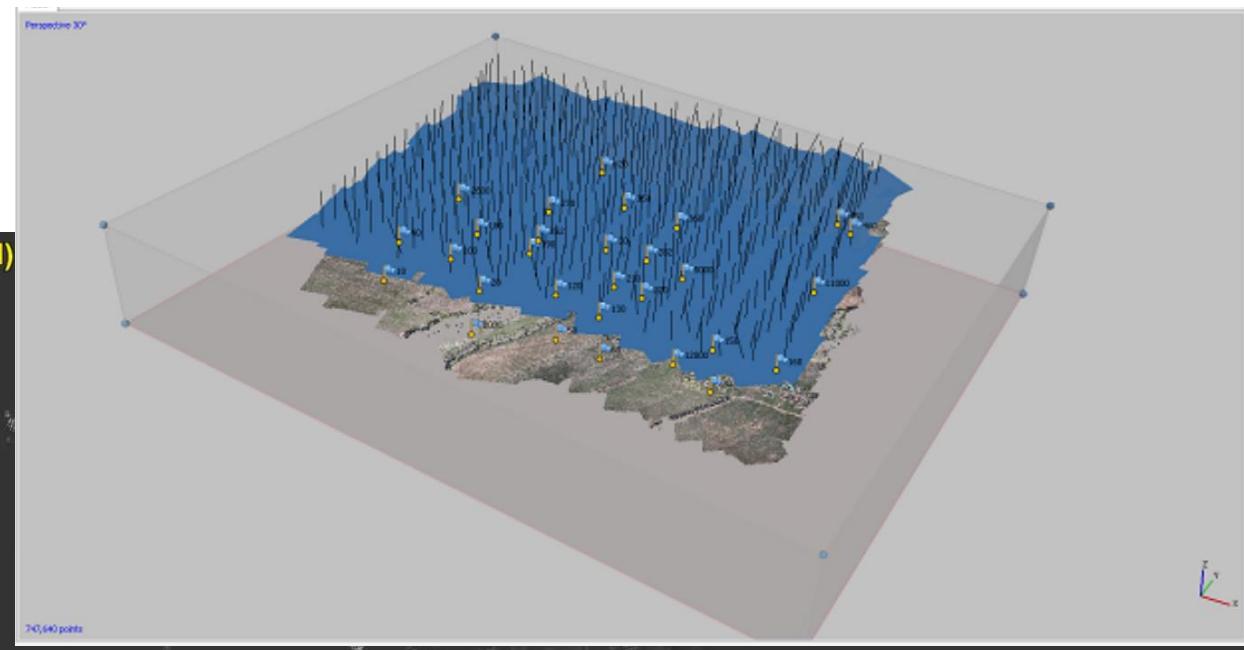
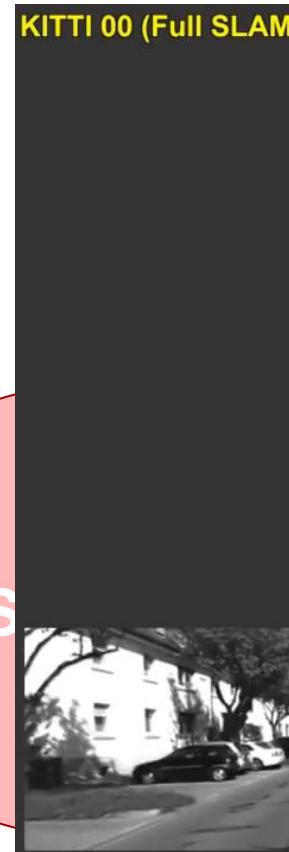
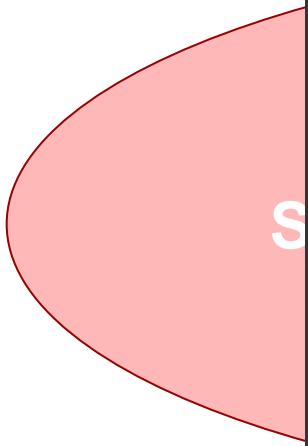
# Windowed Bundle Adjustment (BA)



- 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

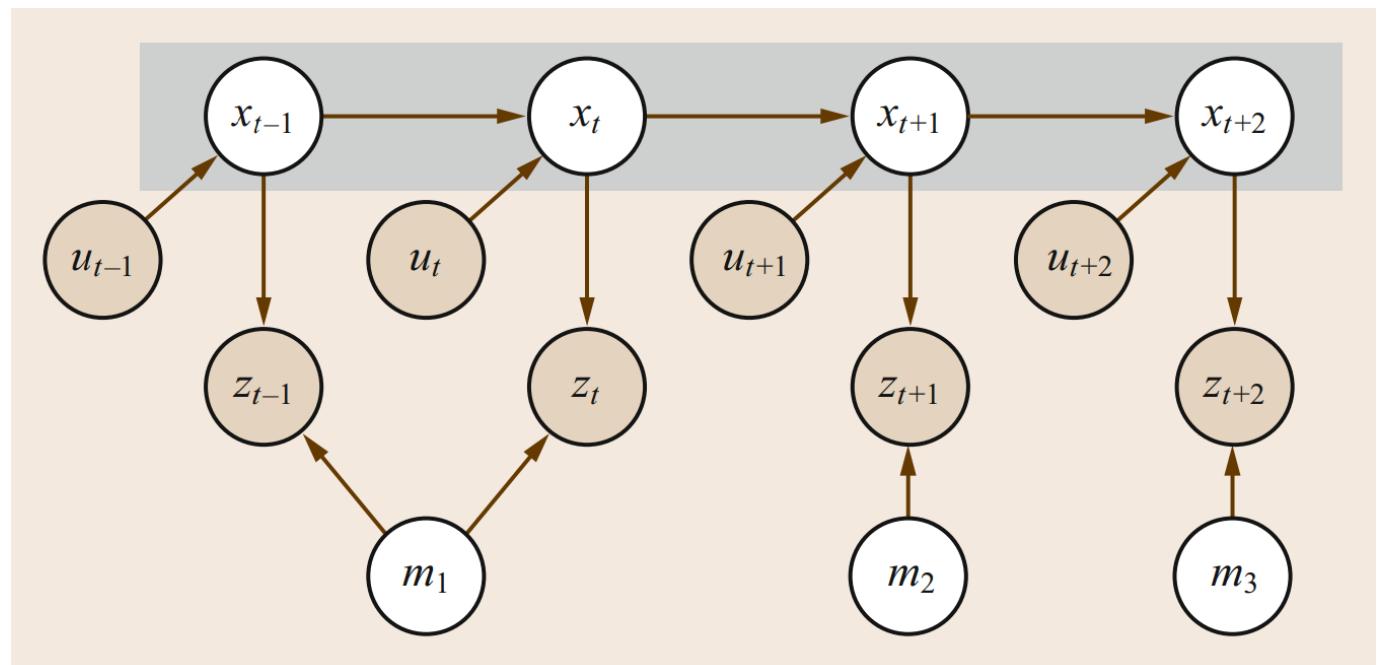
- Visual Odometry
- Structure from Motion (SfM)
- Bundle Adjustment
- Visual SLAM



- 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

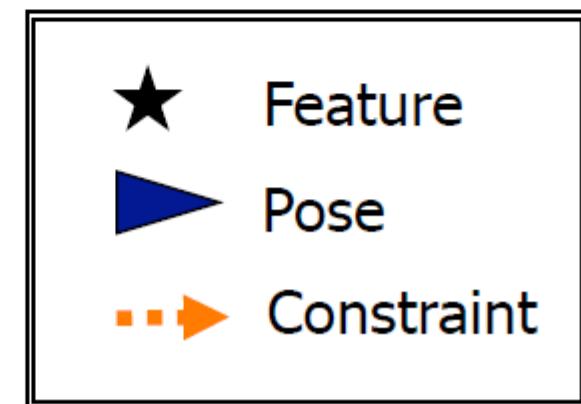
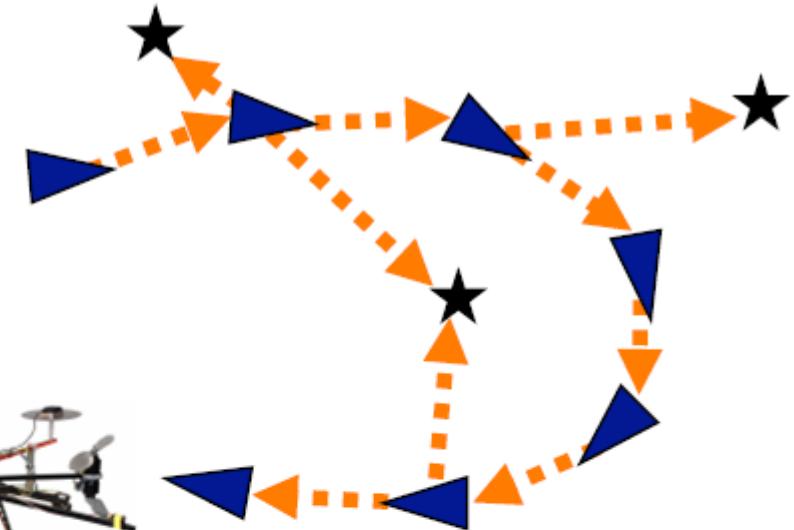
- SLAM problem depicted as Bayes network graph
- At each **location**  $x_t$
- Observes a nearby feature in the **map**  $m = \{m_1; m_2; m_3\}$
- Movement  $u_t$
- An arrow defines causal relationship



# 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)

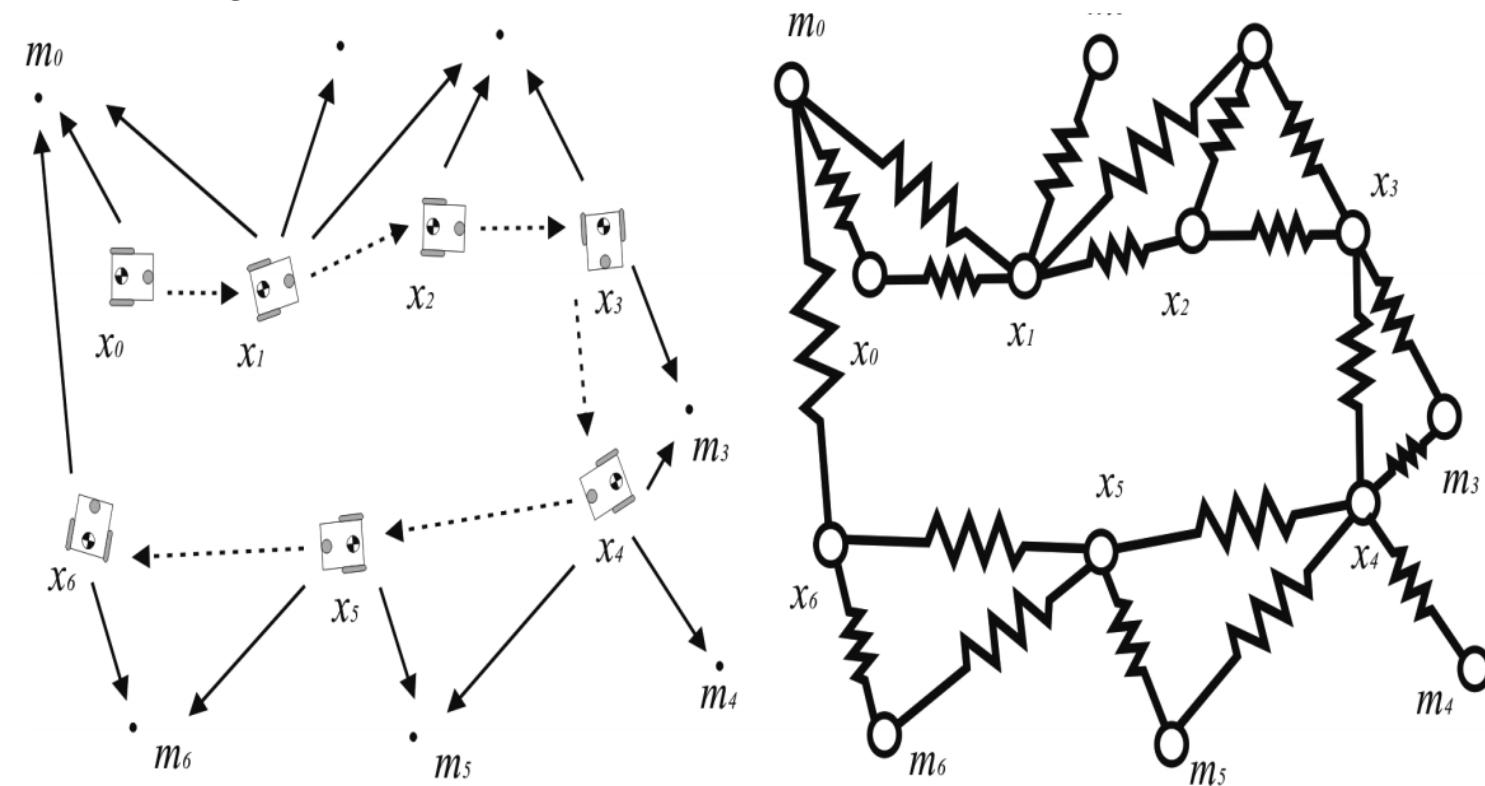
- Observing previously seen areas generates constraints between non-successive poses
- Treat constraints (motion and measurement) as “soft” elastic springs
- Want to minimize the total energy in the springs

We can define the error as follows

- Expected observation (2D sensor)

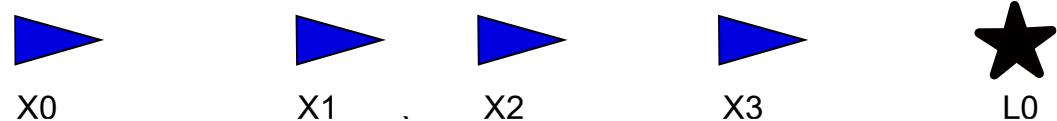
– With the error:  $e_{ij}(x_i, x_j) = \hat{z}$

$$= I$$



# 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:

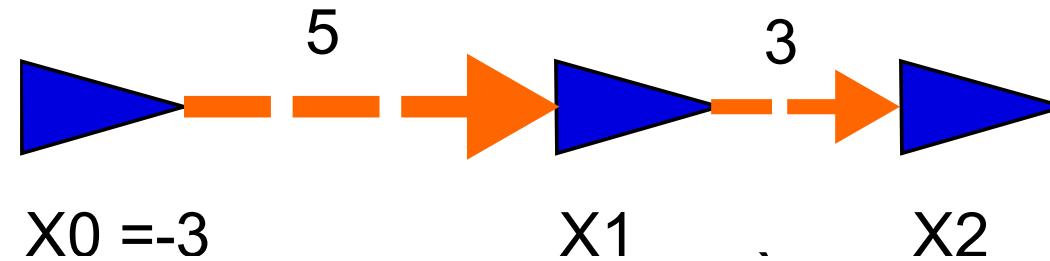
$$L(k) = X(t) + N$$

- *Then, solve linear equations*

# 1D Linear SLAM – case 1

- Case 1 - Exact solution exists:

$$\begin{bmatrix} & \\ & \end{bmatrix} \cdot \begin{bmatrix} & \\ & \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$



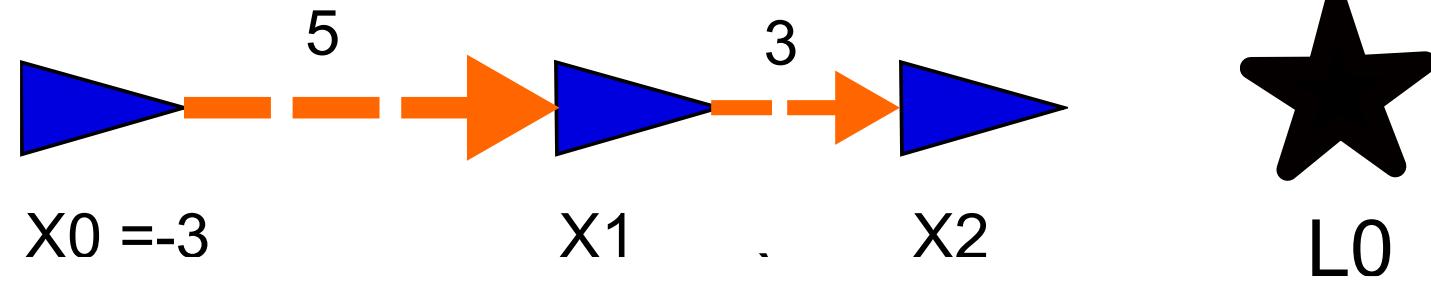
$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \end{bmatrix}$$

$$A^*X = B \quad X = A^{-1} * B \quad A^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$x = [-3 \ 2 \ 5]$$

# 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



$$\left[ \quad \quad \right] \cdot \left[ \quad \quad \right] = \left[ \quad \quad \right]$$

$$A^*X = B \quad X = A^{-1} * B \quad A^{-1} = [?]$$

$$X = (A^T * A)^{-1} * A^T * B$$

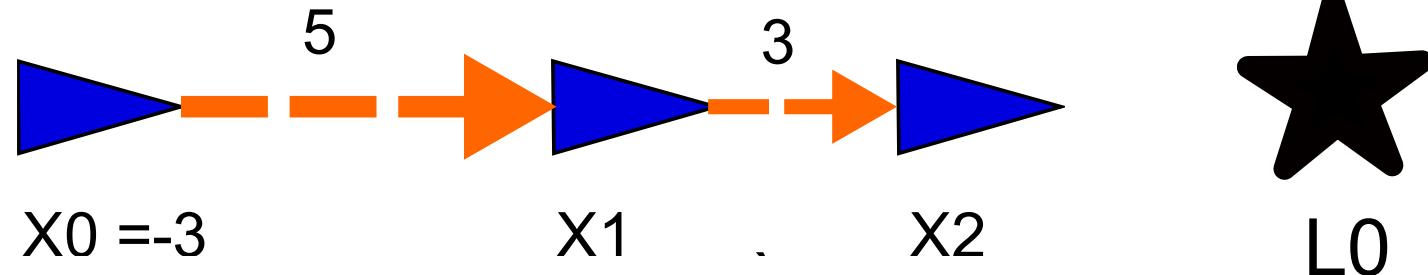
$$x = [-3 \ 2 \ 5 \ 7]$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ L_0 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \\ -10 \\ -5 \\ -2 \end{bmatrix}$$

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

$$X = (A^T * W * A)^{-1} * A^T * W * B$$

$$x = \begin{bmatrix} -3 & 2.18 & 5.71 & 6.82 \end{bmatrix}$$

$$\begin{array}{ccccc} & & 5 & & \\ & \text{X0} & \longrightarrow & \text{X1} & \longrightarrow & \text{X2} & \longrightarrow & \text{L0} \\ & & & & & & & \\ & & & & & & & \star \end{array}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ L_0 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \\ 3 \\ -10 \\ -5 \\ -1 \end{bmatrix}$$

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

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,  $\theta$  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  $\theta$ .

# Gauss Newton Solution

1. Start from an initial value  $\theta_0$ .
2. At step  $i$  assume a linear approximation for the function at  
 $\theta_i$

$$\mathbf{f}(\theta_i + \Delta) = \mathbf{f}(\theta_i) + \mathbf{f}_\theta \Delta \text{ where } \mathbf{f}_\theta = \partial \mathbf{f} / \partial \theta = \mathbf{J}$$

3. Solve

$$\mathbf{f}(\theta_i + \Delta) = \mathbf{f}(\theta_i) + \mathbf{J}\Delta = 0$$

or

$$\mathbf{J}\Delta = -\mathbf{f}(\theta_i)$$

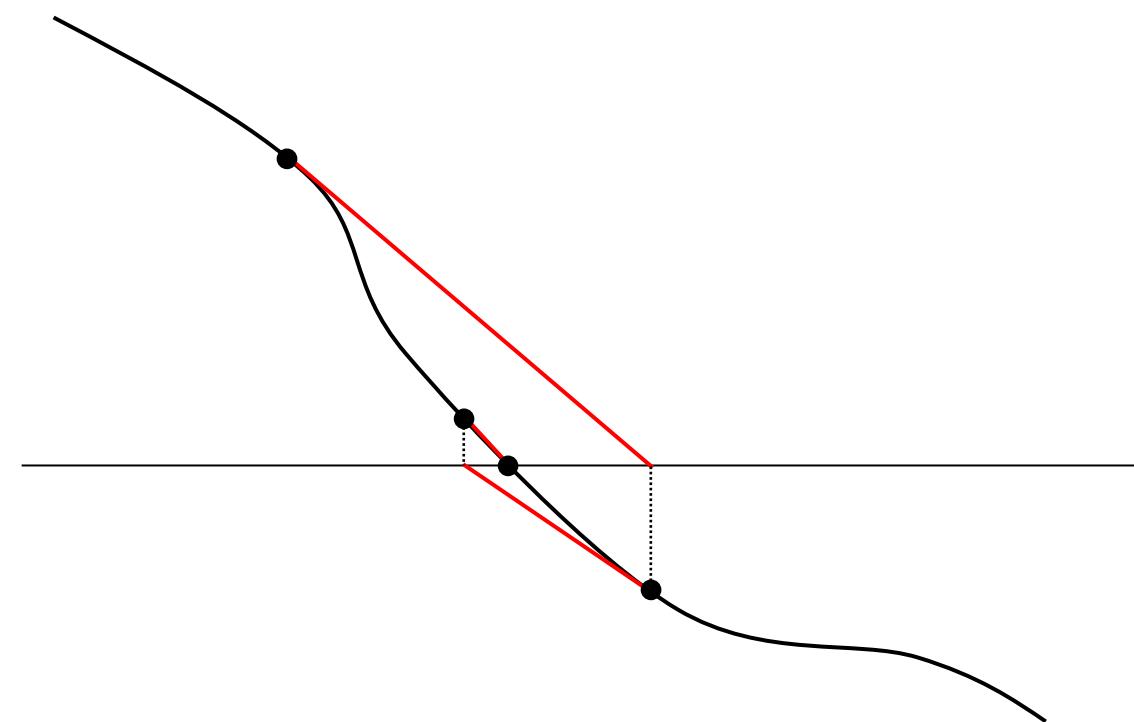
4. This is a linear least-squares problem (solve for  $\Delta$ ):

$$\mathbf{J}^\top \mathbf{J}\Delta = \mathbf{J}^\top \mathbf{f}(\theta_i)$$

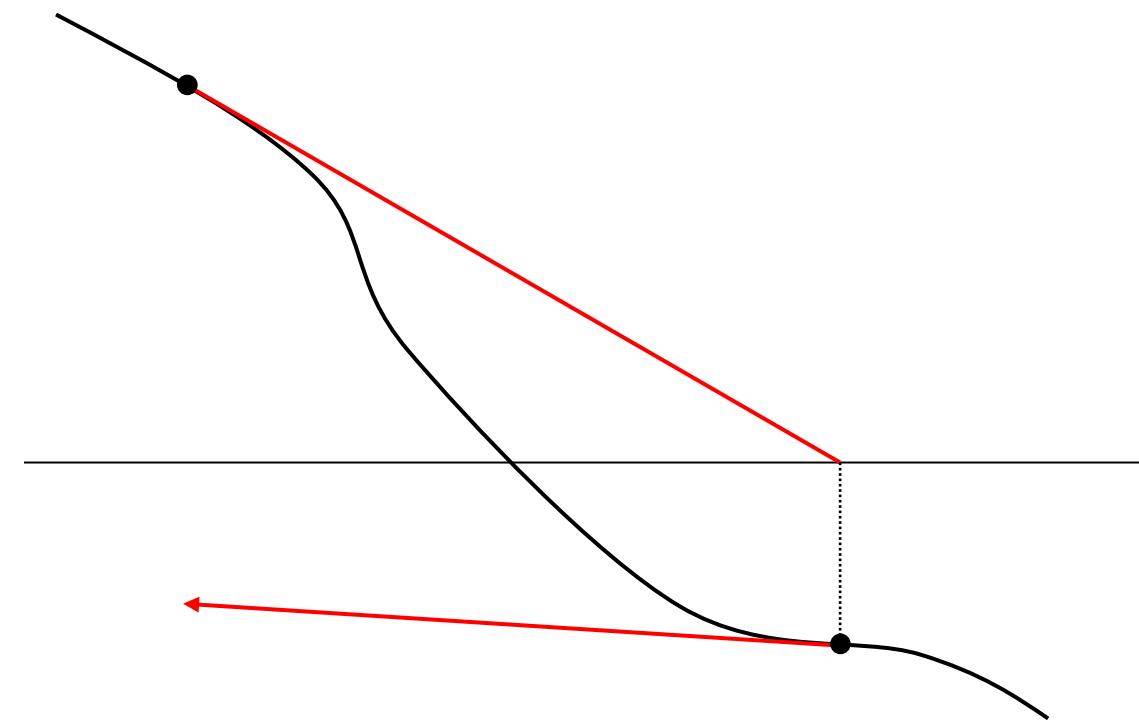
5. Then set  $\theta_{i+1} = \theta_i + \Delta$ .

## 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)

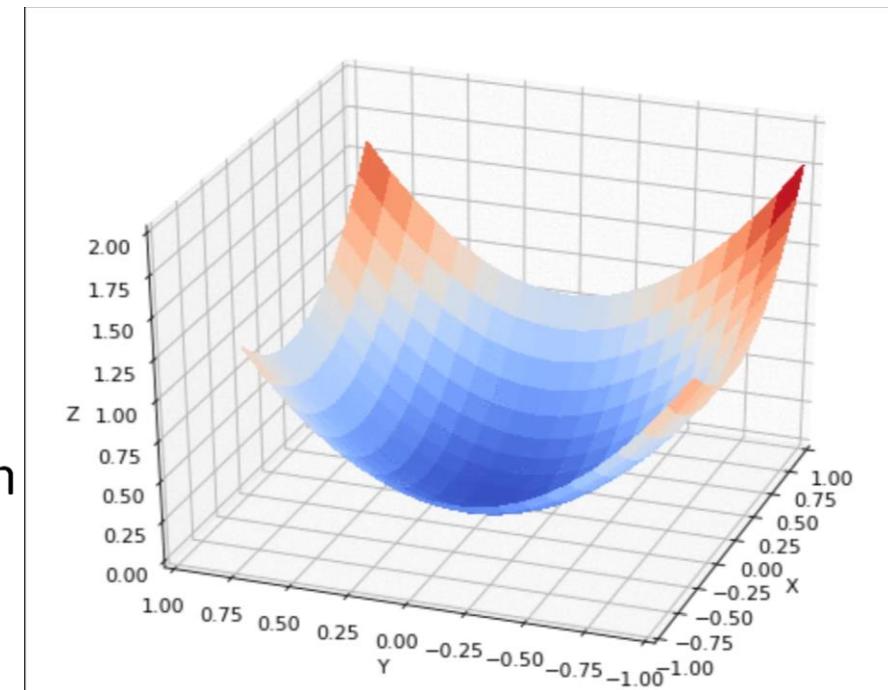
# Gradient Descent

Search direction is the direction of fastest descent of the function  $g$ .

## Gradient descent update equation

$$\lambda \Delta = -g_\theta = -\mathbf{J}^\top \mathbf{f}$$

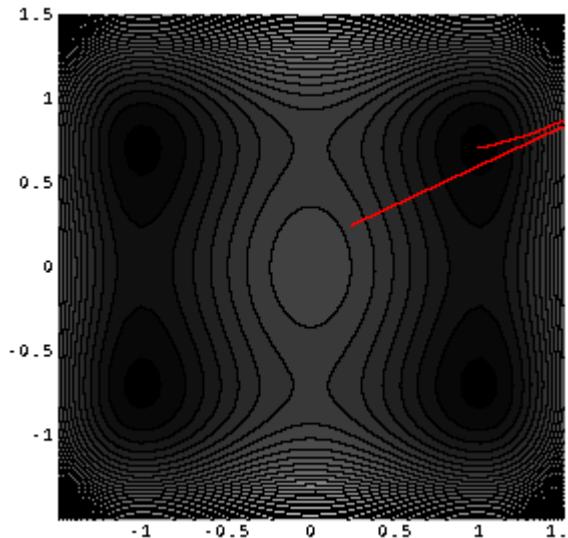
Requires a 1D line search in  $\lambda$  to find the optimum direction.



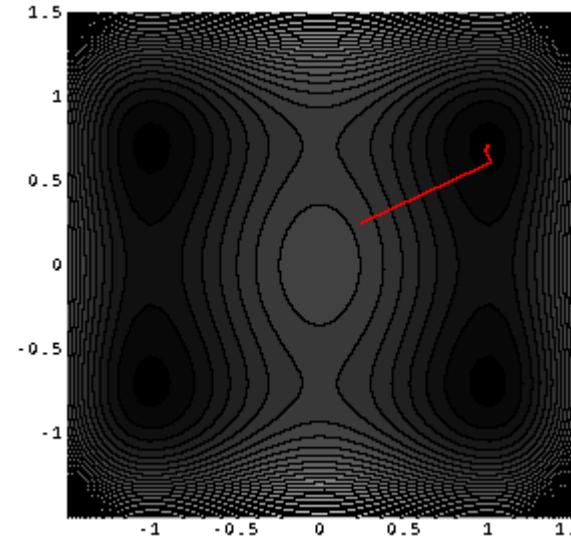
- Mixture of Gauss-Newton and Gradient descent.
- Acts like Gauss-Newton when close to the minimum (quadratic region)
- Gradient descent when improvement is difficult.
- Depends on a parameter  $\lambda$  which
  1. Controls the mixture of Gauss-Newton and Gradient Descent
  2. Controls the step-length.

# What about non-Linear Least Squares?

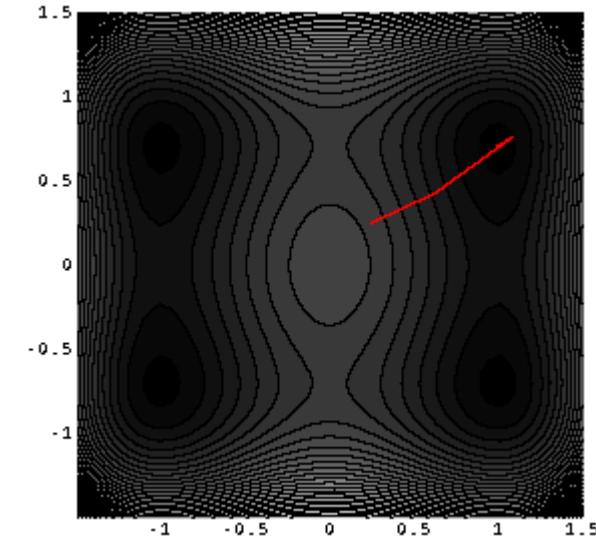
- Lets See some examples 1:



Gauss-Newton



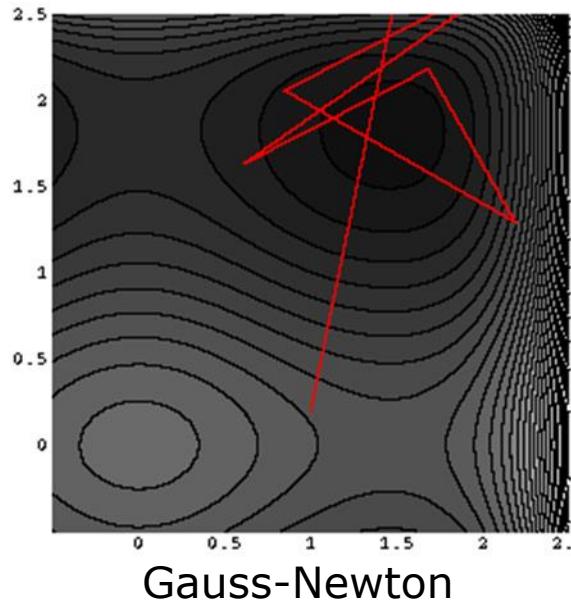
Gradient descent



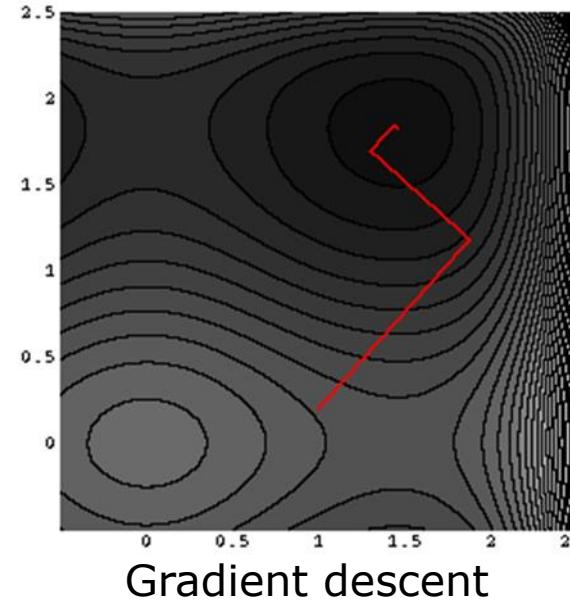
Levenberg

# What about non-Linear Least Squares?

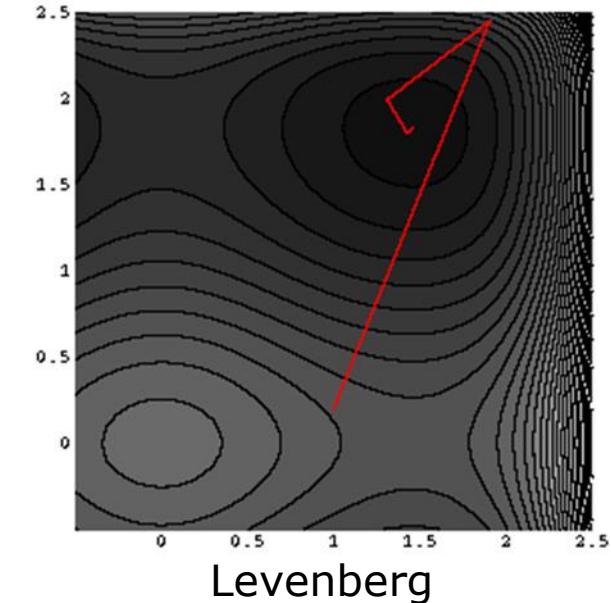
- Lets See some examples 2:



Gauss-Newton



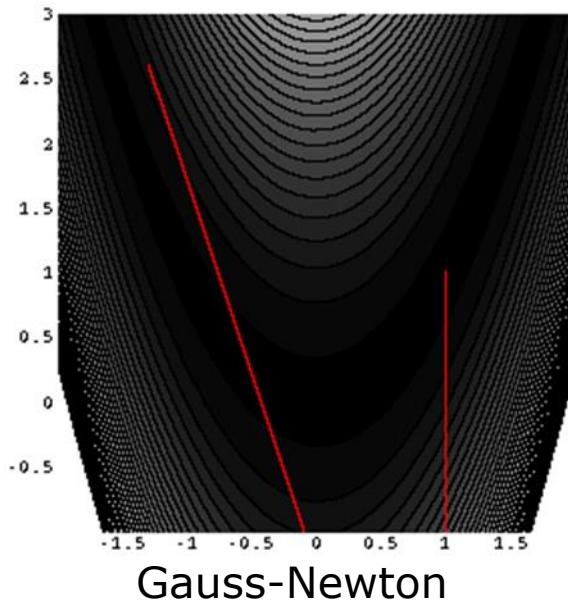
Gradient descent



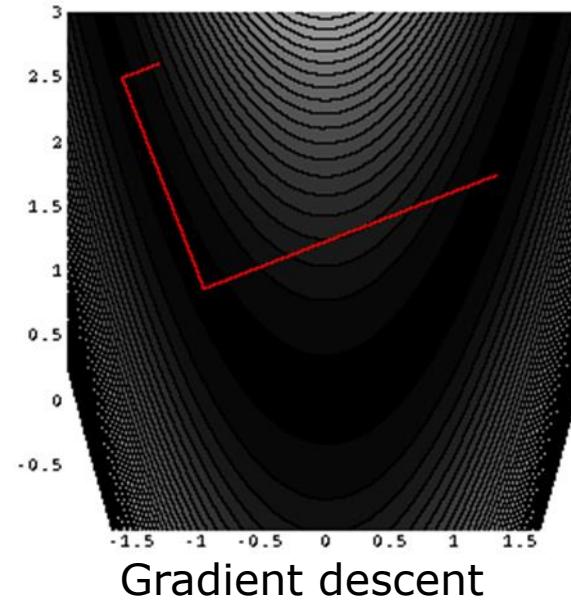
Levenberg

# What about non-Linear Least Squares?

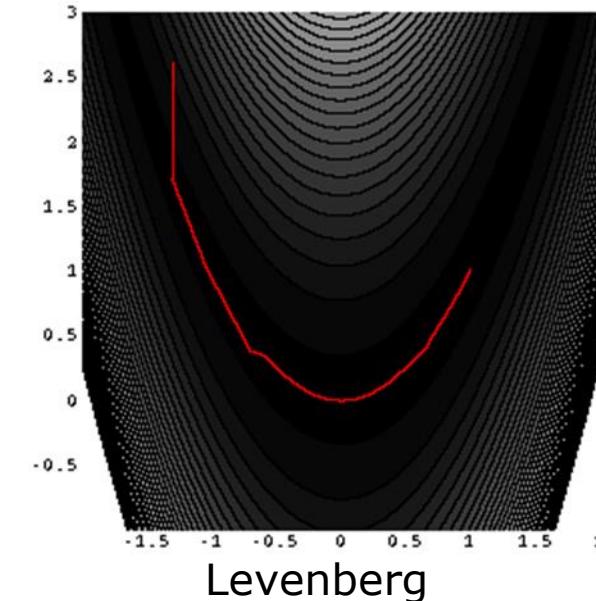
- Lets See some examples 3:



Gauss-Newton



Gradient descent



Levenberg

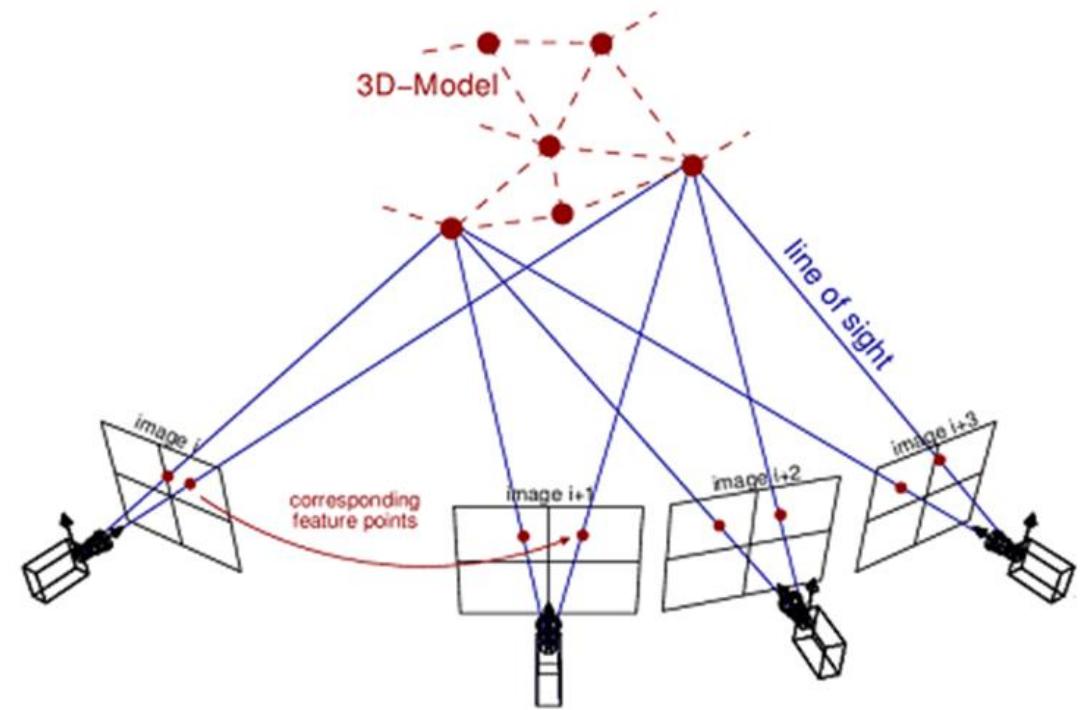
- It is obvious that Levenberg Marquadt displays robustness

# 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  
 $\pi$  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

$\pi$  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:

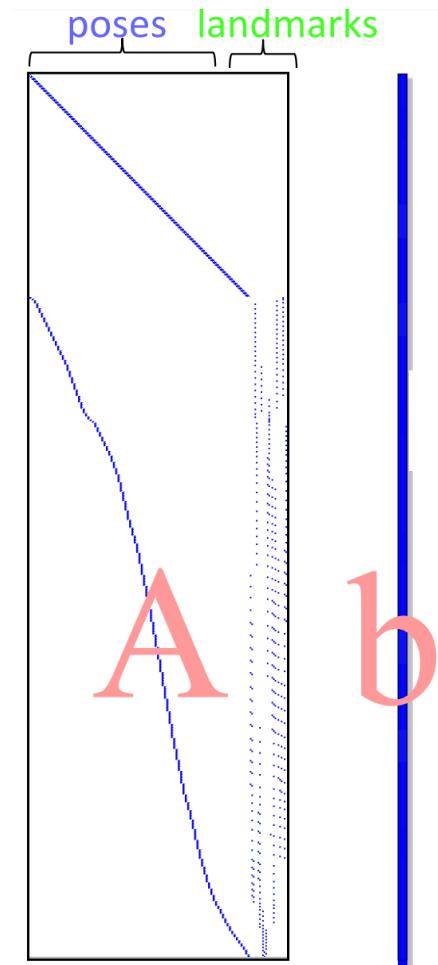
$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} f_x & s_k & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} [R \quad T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- 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:

$$\mathbf{J} = \left[ \begin{array}{ccccccccc} \frac{\partial u}{\partial w_x} & \frac{\partial u}{\partial w_y} & \frac{\partial u}{\partial w_z} & \frac{\partial u}{\partial f} & \frac{\partial u}{\partial u_0} & \frac{\partial u}{\partial v_0} & \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial w_x} & \frac{\partial v}{\partial w_y} & \frac{\partial v}{\partial w_z} & \frac{\partial v}{\partial f} & \frac{\partial v}{\partial u_0} & \frac{\partial v}{\partial v_0} & \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{array} \right]$$

# 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!**



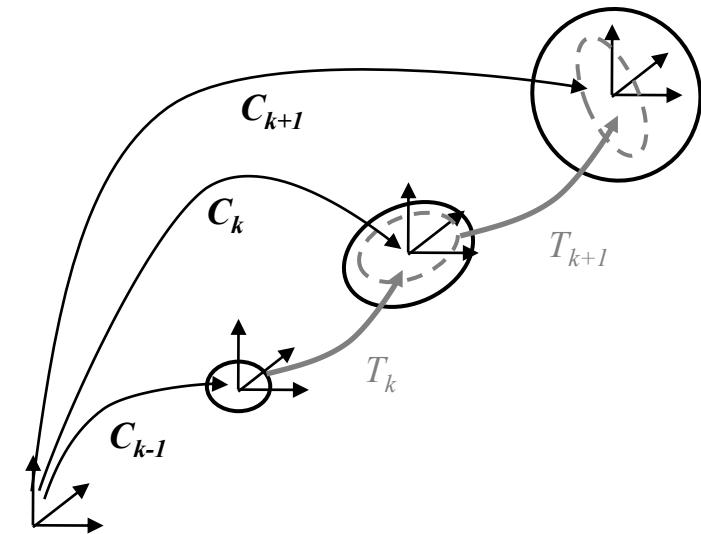
# Bundle Adjustment – Covariance

- 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  $\Sigma_k$  is

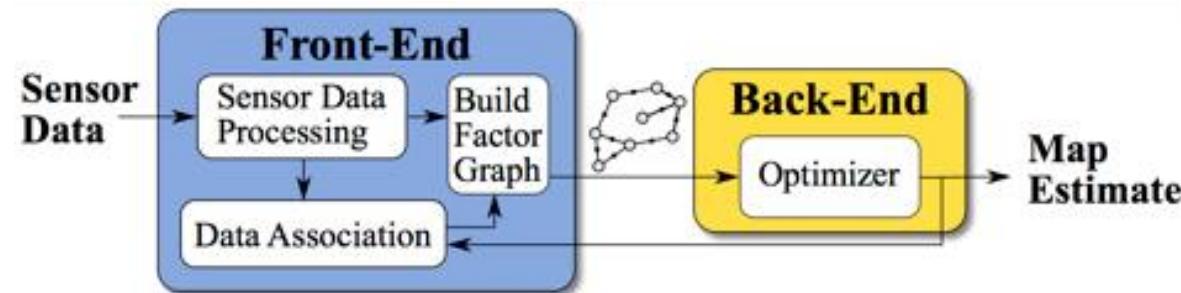
$$\begin{aligned}\Sigma_k &= J \begin{bmatrix} \Sigma_{k-1} & 0 \\ 0 & \Sigma_{k,k-1} \end{bmatrix} J^\top \\ &= J_{\vec{C}_{k-1}} \Sigma_{k-1} {J_{\vec{C}_{k-1}}}^\top + J_{\vec{T}_{k,k-1}} \Sigma_{k,k-1} {J_{\vec{T}_{k,k-1}}}^\top\end{aligned}$$



- 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

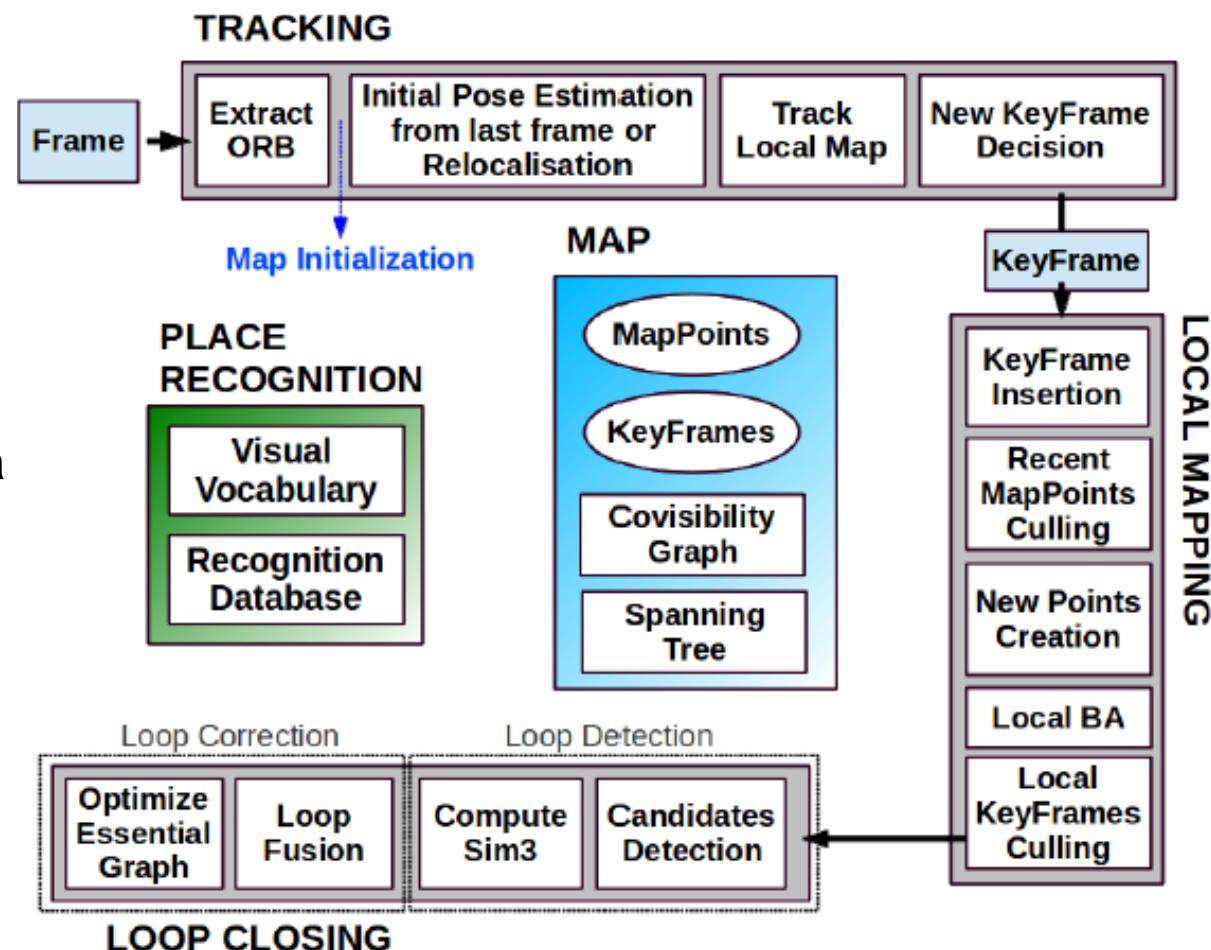
- Front End
  - Data Association
    - Frame to Frame
    - Multi-frame
    - Loop Closure Detection
  - Geometric Initialization
    - Pose Estimation
    - Landmark Triangulation
  - System Formation
    - Observation Matrix
    - Covariance Matrix
    - Graph Generation and Update
- Back End
  - Filter-Based 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