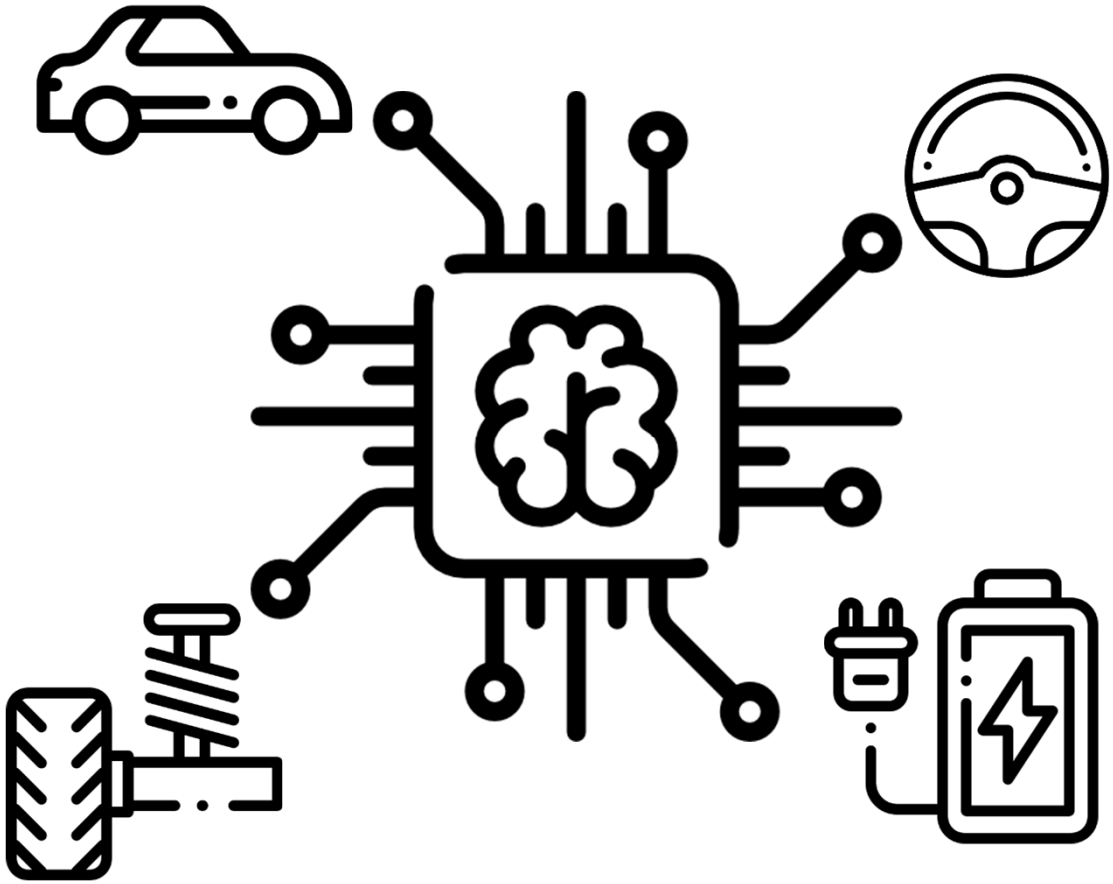
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp /Prof. Dr.-Ing. Boris Lohmann

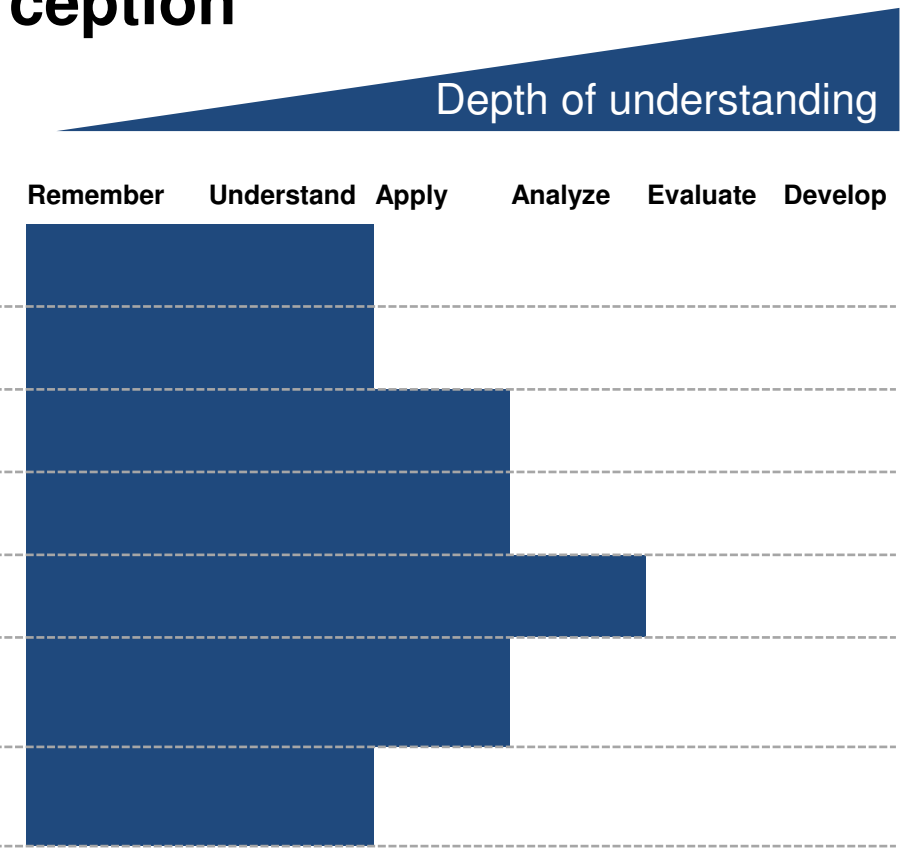


Lecture Overview

Lecture 16:15 – 17:45	Practice 17:45 – 18:30
1 Introduction: Artificial Intelligence 17.10.2019 – Johannes Betz	**Practice 1** 17.10.2019 – Johannes Betz
2 Perception 24.10.2019 – Johannes Betz	**Practice 2** 24.10.2019 – Johannes Betz
3 Supervised Learning: Regression 31.10.2019 – Alexander Wischnewski	**Practice 3** 31.10.2019 – Alexander Wischnewski
4 Supervised Learning: Classification 7.11.2019 – Jan Cedric Mertens	**Practice 4** 7.11.2019 – Jan Cedric Mertens
5 Unsupervised Learning: Clustering 14.11.2019 – Jan Cedric Mertens	**Practice 5** 14.11.2019 – Jan Cedric Mertens
6 Pathfinding: From British Museum to A* 21.11.2019 – Lennart Adenaw	**Practice 6** 21.11.2019 – Lennart Adenaw
7 Introduction: Artificial Neural Networks 28.11.2019 – Lennart Adenaw	**Practice 7** 28.11.2019 – Lennart Adenaw
8 Deep Neural Networks 5.12.2019 – Jean-Michael Georg	**Practice 8** 5.12.2019 – Jean-Michael Georg
9 Convolutional Neural Networks 12.12.2019 – Jean-Michael Georg	**Practice 9** 12.12.2019 – Jean-Michael Georg
10 Recurrent Neural Networks 19.12.2019 – Christian Dengler	**Practice 10** 19.12.2019 – Christian Dengler
11 Reinforcement Learning 09.01.2020 – Christian Dengler	**Practice 11** 09.01.2020 – Christian Dengler
12 AI-Development 16.01.2020 – Johannes Betz	**Practice 12** 16.01.2020 – Johannes Betz
13 Guest Lecture: VW Data:Lab 23.01.2020 –	

Objectives for Lecture 2: Perception

After the lecture you are able to…



…understand the problems behind computer vision and be able to apply them to autonomous driving

…. understand the individual steps of the computer vision pipeline.

… connect any camera to the computer and extract an image from that camera.

… to remember and apply all variants of image processing.

…. analyze different images regarding their features.

…. remember variants of the feature extraction and be able to apply them.

…. know and understand variants of feature analysis

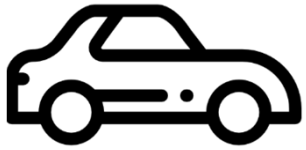
Perception

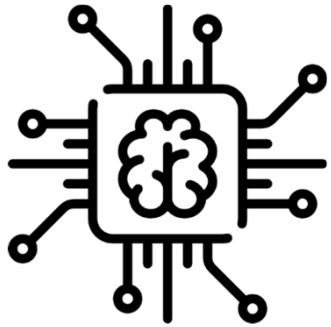
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

Agenda

1. **Chapter: Computer Vision**
2. Chapter: Machine Vision
3. Chapter: Image Processing
4. Chapter: Features
5. Chapter: Feature Analysis
6. Chapter: Information Analysis
7. Chapter: Summary



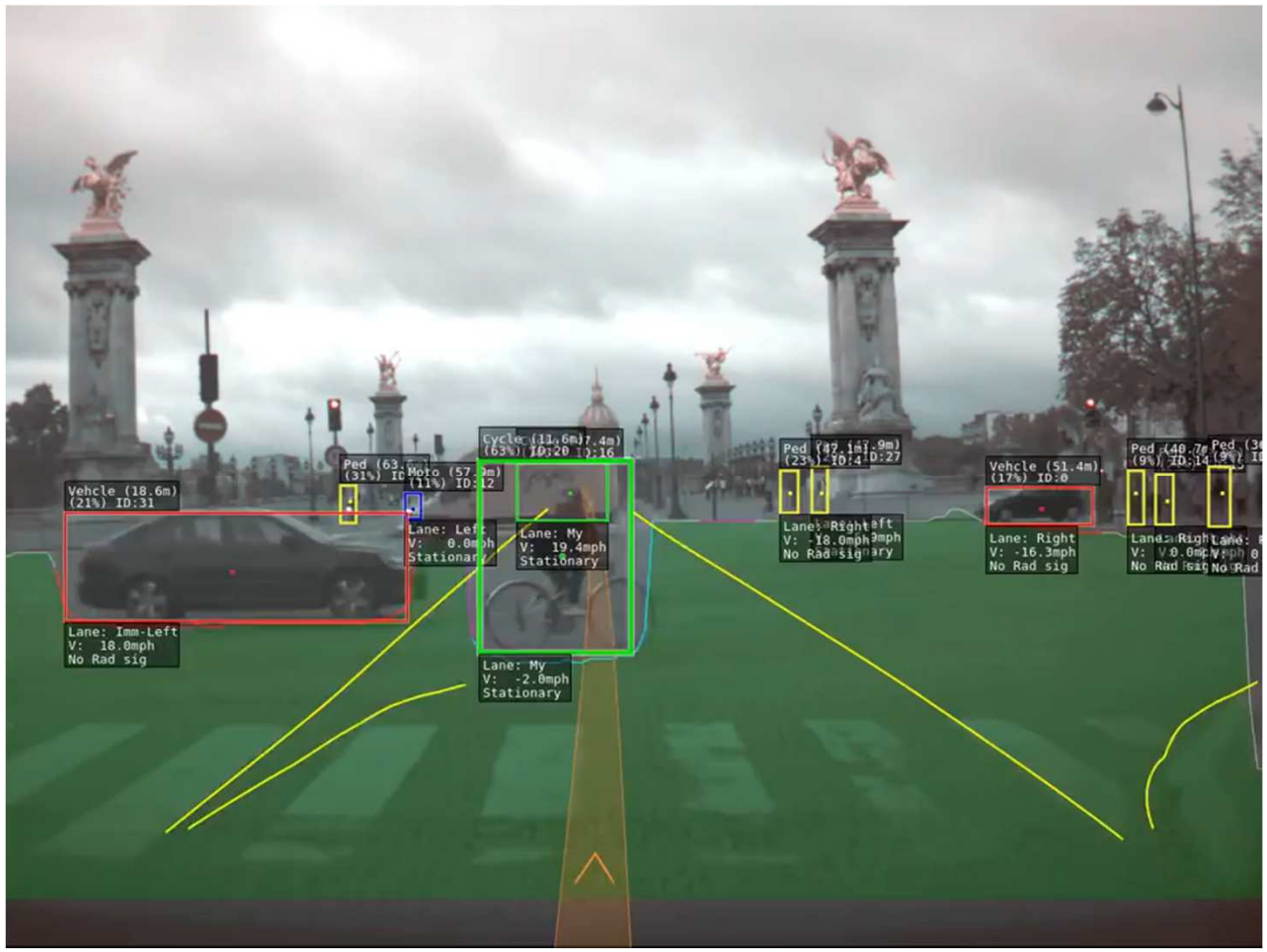


What is Artificial Intelligence? - Perception

The world is a complex and dynamic place

Source: Bryn Balcome, „What is level-5 Motorsport?“, GTC 2017 conference

What is Artificial Intelligence? - Perception



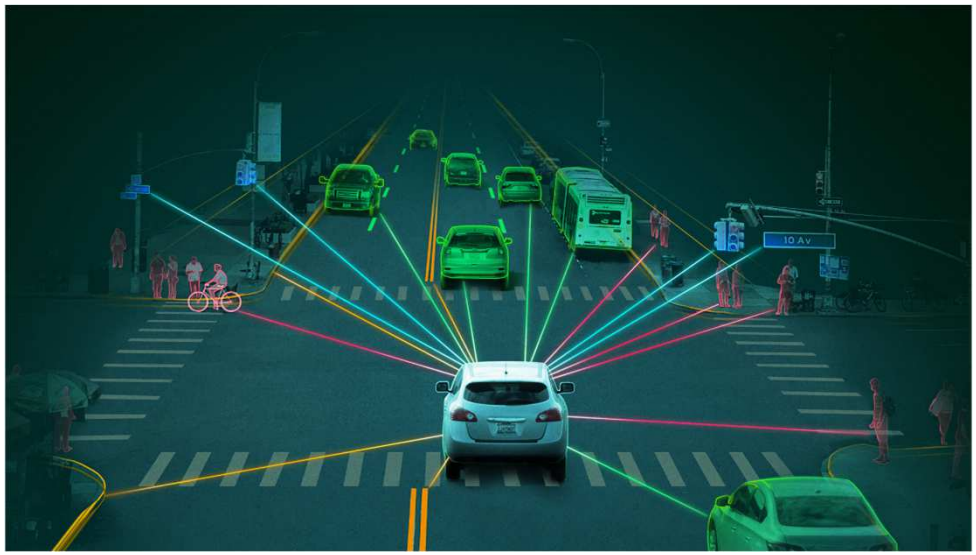
The world is a complex and dynamic place

Source: Paris Streets in the eyes of Tesla Autopilot: https://www.youtube.com/watch?time_continue=1&v=_1MHGUC_BzQ

What is Artificial Intelligence?

Breaking down the general problem of creating AI into 9 sub-problems:

1. **Reasoning & Problem Solving**
2. **Knowledge Representation**
3. **Planning**
4. **Learning**
5. **Natural Language Processing (NLP)**
6. **Perception**
7. **Motion and Manipulation**
8. **Social Intelligence**



Source: Artificial Intelligence – A modern Approach / https://itrade.gov.il/uk/2018/10/19/vayavisions-autonomous-vehicle-perception-system-raised-8m/

Computer Vision - The Perception Problem

Problem Description:

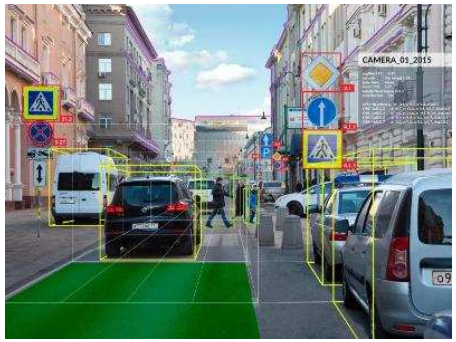
- **A computer is acquiring the ability to perceive the environment**
- Sensors are used as input: Camera, Lidar, Ultrasonic, Radar, microphones, …
- **Machine Perception:** Capability to interpret data related to the environment
- **Computer Vision (CV):** The camera input **(images/videos)** is analyzed and information is extracted

Focus in this lecture: Computer Vision (CV) and Camera

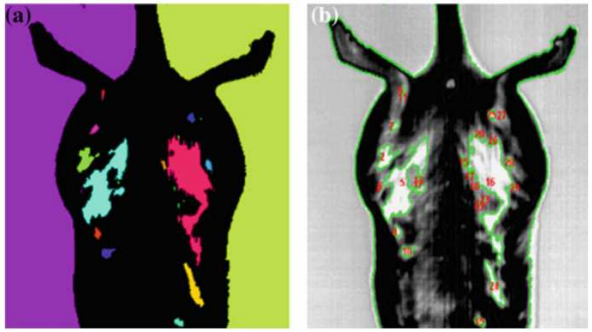
- Camera is the cheapest sensor of the car
- Camera is similar to „Human Eyes“
- Camera imagery includes almost every information about the environment
- Camera images can be process by classical CV-Algorithms and Deep Learning-Algorithms

Source: https://www.bhphotovideo.com/c/product/1308820-REG/nikon_d5600_dslr_camera_with.html

Computer Vision - Why it matters

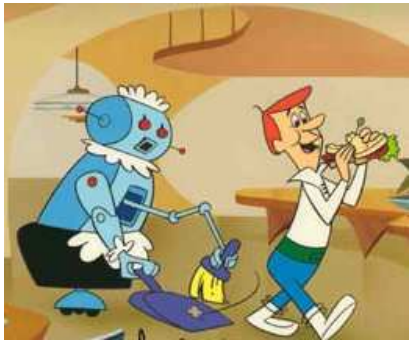


Autonomous Driving



Health

Security



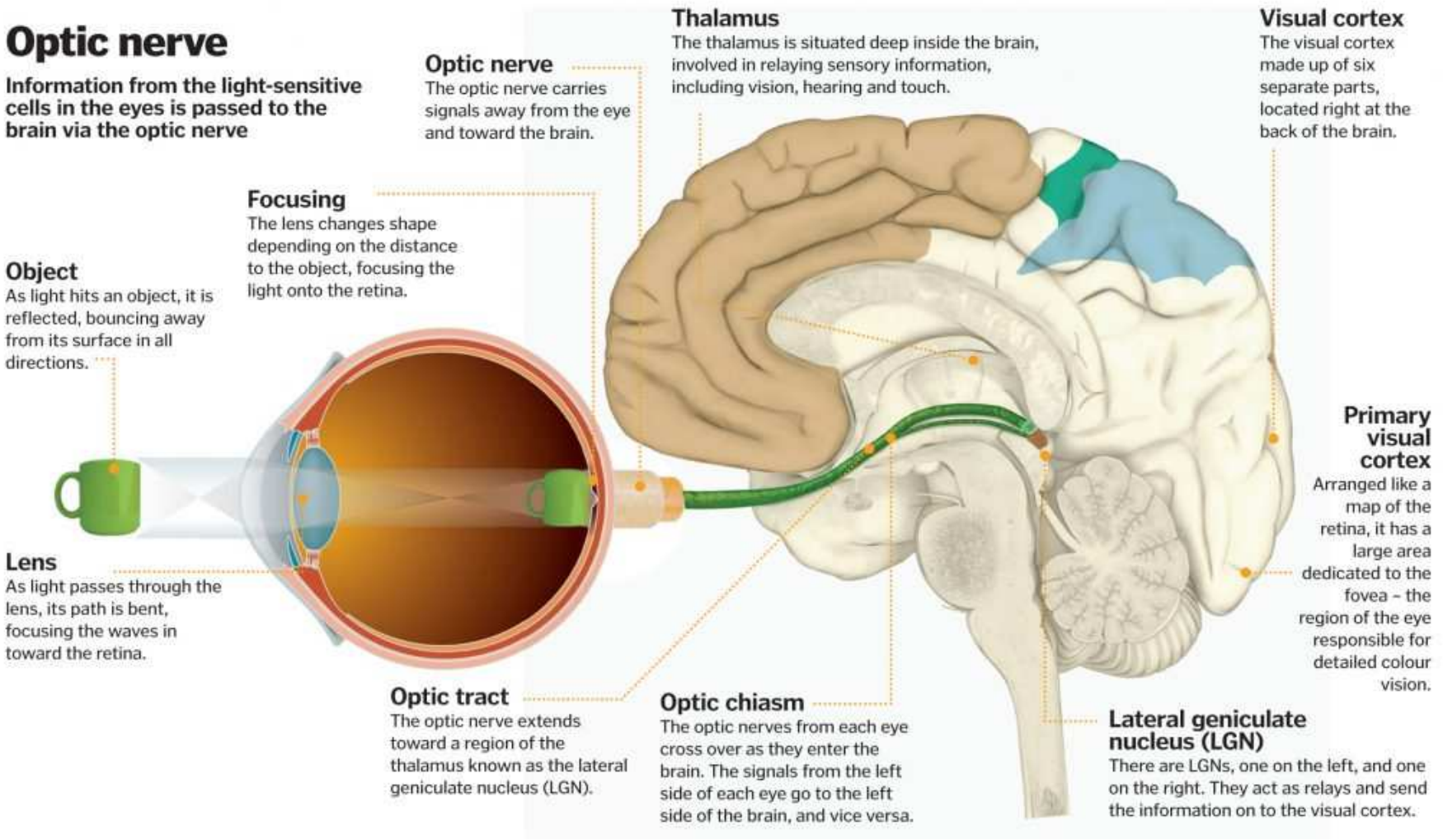
Comfort

Fun & Games

Access

Source: http://www.magazine.unimore.it/site/home/international-news/articolo820043703.html /https://medium.com/health-ai/heat-screening-for-breast-cancer-computer-vision-for-dieting-brain-wave-passwords-c5294b5e73b / http://osxdaily.com/2017/11/10/can-use-iphone-x-without-face-id/ http://gamechanger.co.ke/product/xbox-360-kinect-sensor/ https://www.express.de/duesseldorf/duesseldorf-archiv/fahndungsfoto-hier-ueberfallen-zwei-raeuber-eine-tankstelle-in-duesseldorf--24421170

Computer Vision - Human Vision



How Humans see

https://www.howitworksdaily.com/science-of-vision-how-do-our-eyes-enable-us-to-see/

Computer Vision – Problem for computers

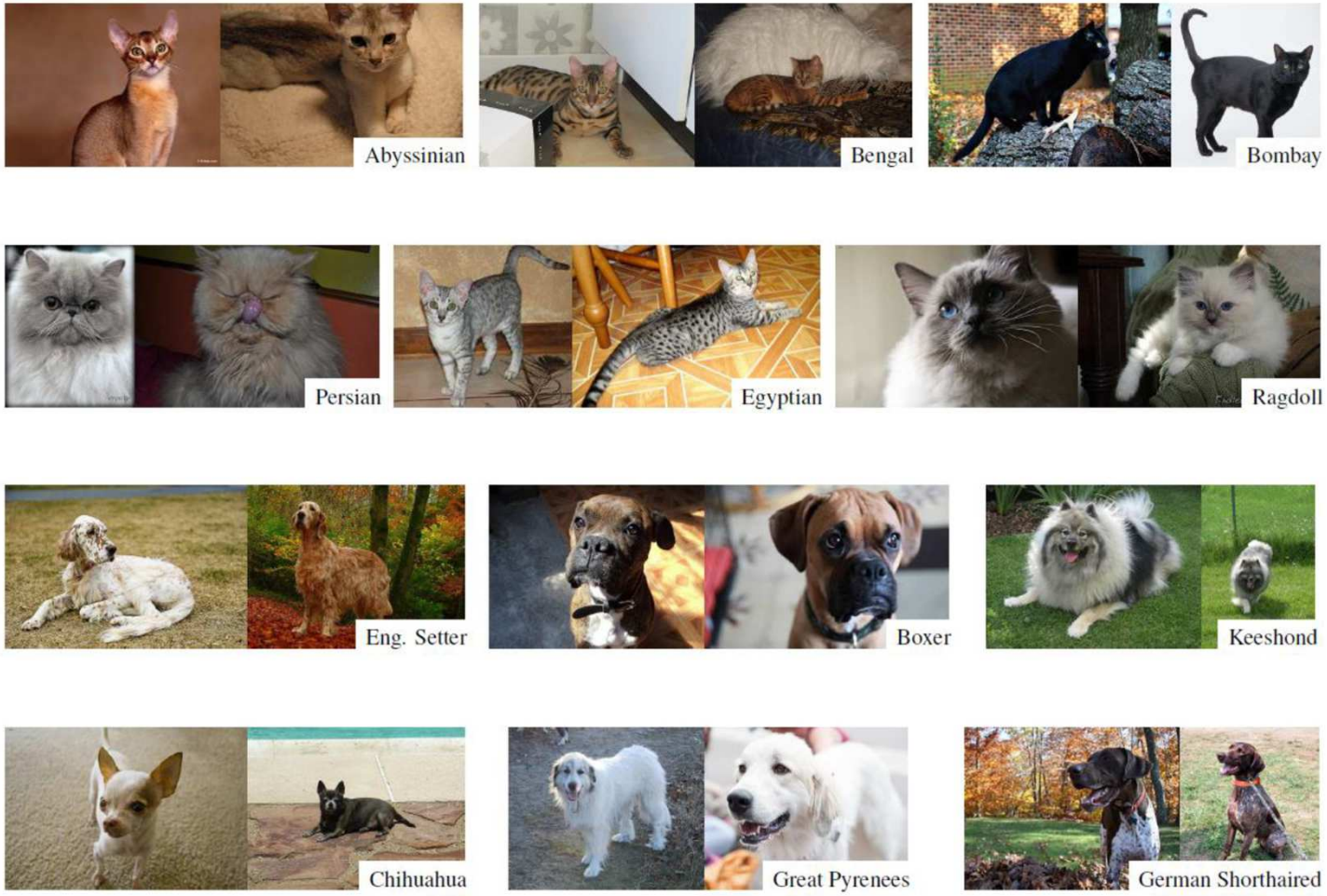
- Objects can be highly variable in shape – e.g., trees, cars, animals, …
- Loss of information in sensing process – 3D objects projected onto 2D images
- Missing data: Occlusions and hidden surfaces
- Shadows and noise obscure signal

Illumination Variability

Occlusion

Source: https://slideplayer.com/slide/7773132/ https://www.robots.ox.ac.uk/~vgg/publications/2011/Parkhi11/parkhi11.pdf

Computer Vision – Problem for computers



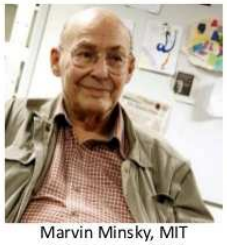
Intra Class Variability

Source: http://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf

Computer Vision – History

- 1966: Minsky assigns computer vision as an undergrad summer project
- 1970's: Some progress on interpreting selected images
- 1980's: ANNs come and go; shift toward geometry and increased mathematical rigor
- 1990's: Face recognition; statistical analysis in vogue → **Rule Based**
- 2000's: Broader recognition; large annotated datasets available; video processing starts
- 2010's: Deep learning with ConvNets

  → **Classification**

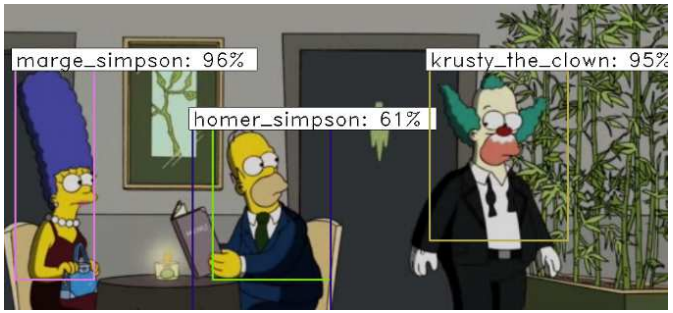


Marvin Minsky, MIT
Turing award, 1969

"In 1966, Minsky hired a first-year undergraduate student and assigned him a problem to solve over the summer:

connect a camera to a computer and get the machine to describe what it sees."

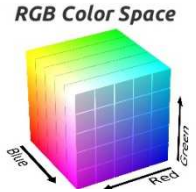
Crevier 1993, pg. 88

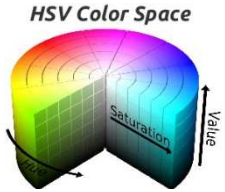


Source: https://www.slideshare.net/tw_dsconf/computer-vision-crash-course-56934508 /
https://ibug.doc.ic.ac.uk/research/detection-static-geometric-facial-features/
https://computervisionblog.wordpress.com/2018/03/25/deep-learning-approaches-for-object-detection/

Computer Vision – The CV-Pipeline

OpenCV



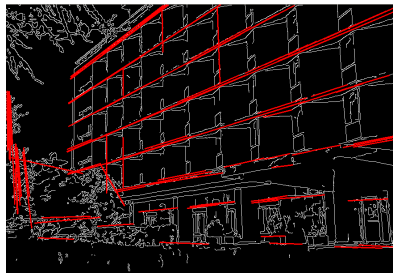


1. Choosing Hardware: Camera, Lens,…

2. Acquire an image/video stream

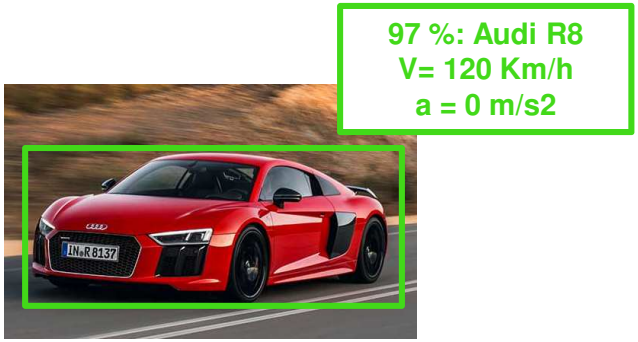
3. Integrate the image in your software pipeline

4. Preprocess the image for further evaluation



5. Detect/Extract features in the image

6. Extract information from the features



7. Setup your CV-Application

Source: https://picclick.com/Point-Grey-Flea2-FL2-08S2M-IEEE-1394-Mono-Digital-Camera-252748955187.html / https://www.pexels.com/photo/man-holding-black-silver-bridge-camera-taking-photo-during-daytime-167571/ / https://de.wikipedia.org/wiki/Datei:OpenCV_Logo_with_text.png / ttps://www.researchgate.net/publication/323952018_Practices_and_pitfalls_in_inferring_neural_representations/figures?lo=1&utm_source=google&utm_medium=organic / https://stackoverflow.com/questions/35299878/probabilistic-hough-transform-line-segment-connection / https://www.autoscout24.de/auto/audi/audi-r8/

Additional Slide

Interactive Lecture Code: Access the camera

1. Choose the right hardware (camera and lens): For choosing the right camera, there is **no** general procedure. You have to choose regarding the application you want to achieve, the software you are using and on your budget. In the automotive sector the company *FLIR (https://eu.ptgrey.com/)* provides a good portfolio of different camera types.

In this lecture we are using the ZED Stereo Camera (https://www.stereolabs.com/zed/) which provides a dual 4MP Camera with high frame rates, an USB 3.0 connection and a 110°field of view. Stereolabs provides different SDK´s for accessing the camera via C++ and Python Code

2. Choose the right software: In this lecture we are working with **OpenCV in Python3**. All examples we are display are written in Python with the ZED SDK. If you want to use the code with other images, the only thing you have to change **is the way of acquiring an image.**

Lecture Code which is displayed now: **01_Camera_input.py**

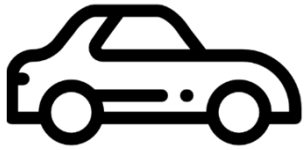
Perception

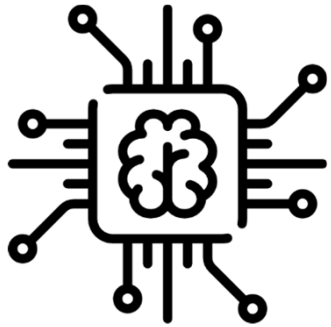
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

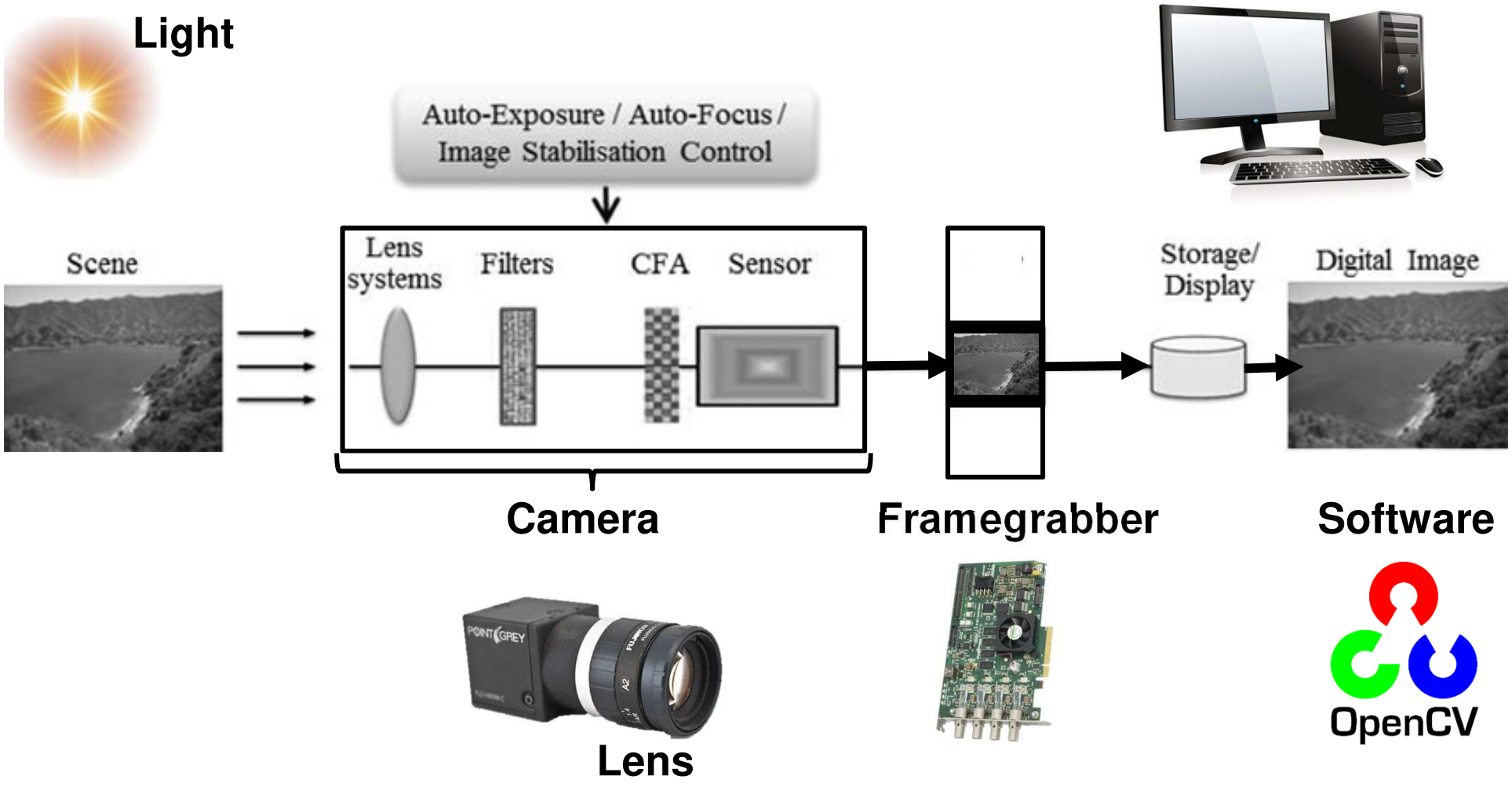
Agenda

1. Chapter: Computer Vision
2. **Chapter: Machine Vision**
3. Chapter: Image Processing
4. Chapter: Features
5. Chapter: Feature Analysis
6. Chapter: Information Analysis
7. Chapter: Summary





Machine Vision – Image Acquisition



Source: https://www.researchgate.net/publication/281476188_Smartphone_image_acquisition_forensics_using_sensor_fingerprint/figures?lo=1 / https://picclick.com/Point-Grey-Flea2-FL2-08S2M-IEEE-1394-Mono-Digital-Camera-252748955187.html / https://de.wikipedia.org/wiki/Datei:OpenCV_Logo_with_text.png / https://www.activesilicon.com/de/product-category/frame-grabbers-de/ http://home.bt.com/tech-gadgets/computing/give-your-pc-a-new-lease-of-life-5-upgrade-options-you-should-be-considering-11363940257353

Machine Vision – Image Acquisition

- **Lights:** The human eye can see well over a wide range of lighting conditions, but a machine vision system is **not as capable.** You must therefore carefully illuminate the scene under observation so that the machine vision system can **“see” it clearly.**

→ **Problem for Autonomous Driving:** Light changes during the day, weather etc.

- **Optics and lenses:** The lens gathers the light reflected (or transmitted) from objects in the camera’s field-of-view, and forms an image in the camera sensor. The proper lens allows you to adjust the **field-of-view (FOV)** and to place the camera at a convenient working distance from the scene. To pick the proper lens you will first need to know the **FOV** and the working distance. The **FOV** is the size of the area you want to capture.

Source: https://www.teledynedalsa.com/en/learn/knowledge-center/machine-vision-101-an-introduction/

Machine Vision – Image Acquisition

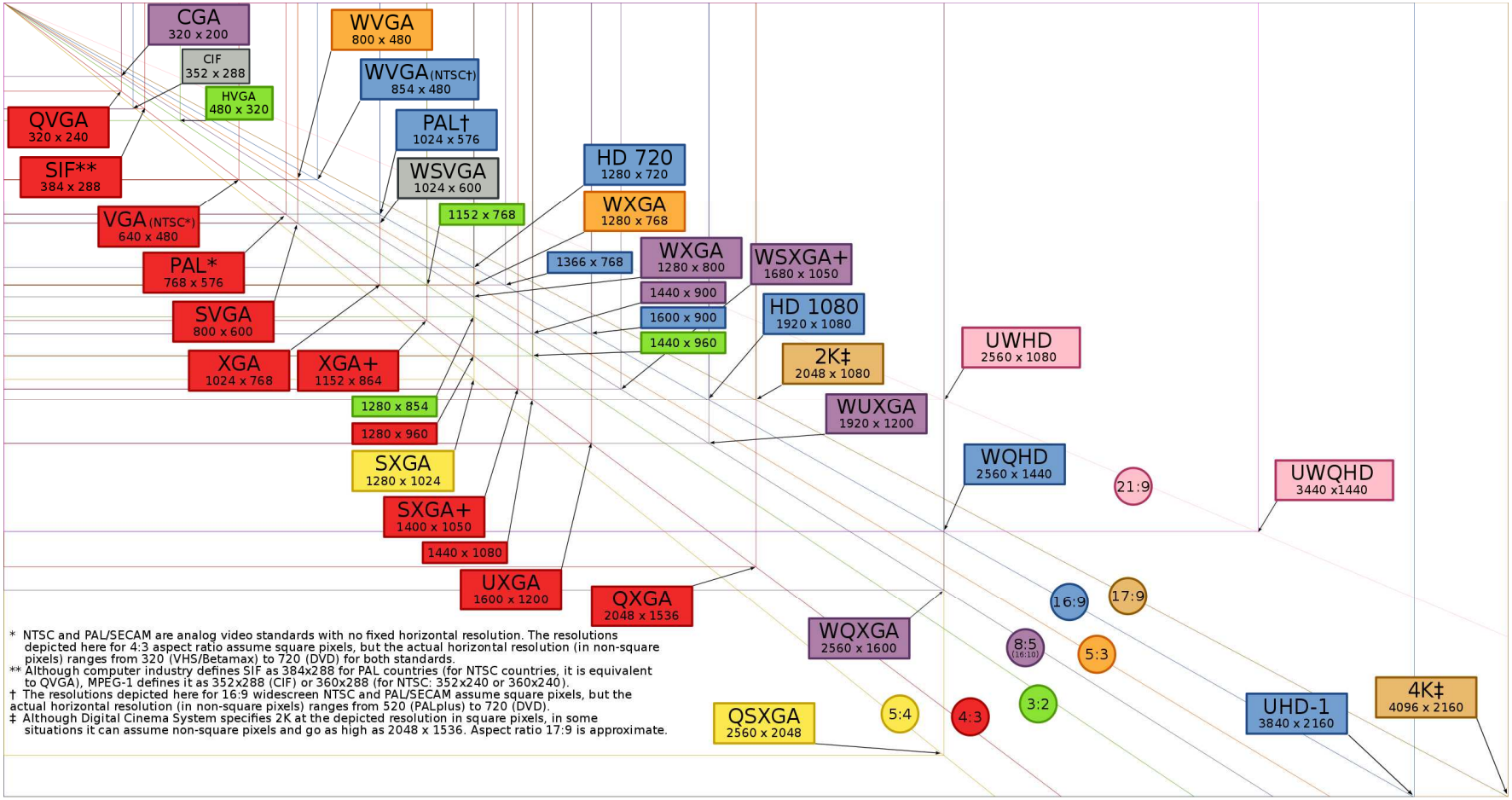
- **Camera:** The camera contains a **sensor** that converts light from the lens into electrical signals. These signals are digitized into an array of values called **pixels.** The resolution (precision) of the inspection depends upon the working distance, the field-of-view (FOV), and the number of physical pixels in the camera's sensor. A standard VGA camera has 640 x 480 physical pixels (= width x height), and each physical pixel is about 7.4 microns square

- **Framegrabber and Software:** Normally a digital Framegrabber sends the digitialzed image over a **cable/Bussystem** (FireWire, USB 2.0/3.0, Ethernet) into the computer. The grabbing can be done asynchronous or synchronous. After a frame was grabbed the image is put in the computer memory and can be processed by specific software e.g. OpenCV

Source: https://www.teledynedalsa.com/en/learn/knowledge-center/machine-vision-101-an-introduction/

Additional Slide

Image Resolution



Source: https://en.wikipedia.org/wiki/Display_resolution#/media/File:Vector_Video_Standards8.svg

Additional Slide

Interactive Lecture Code: Capture an Image

1. First of all we have to access the camera again.

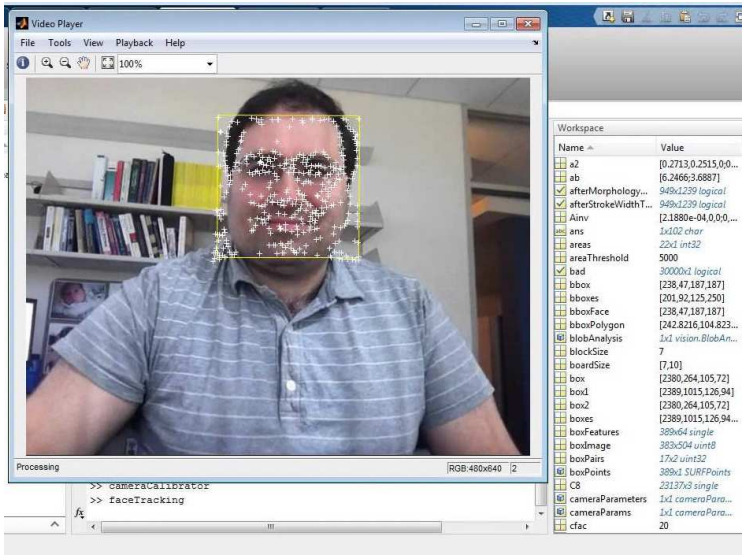
2. In the second step, we have to parametrize the camera:
   - 2.1 Resolution of the Image: VGA, HD720, HD1080,HD2k
   - 2.2 Framerate per second: 15,30,60,100
     → Not every framerate works with every resolution
   - 2.3 Initialize the camera.

3. Chose how many Images should be taken.
4. Save the images to a folder.

Lecture Code which is displayed now: **02_Image_capture.py**

Machine Vision – Computer Vision Software

- Commercial Software (Free for **students)**
- Computer Vision System Toolbox
- **Language:** Matlab, Simulink
- **Operating System:** Linux, macOS, Windows
- **Includes:**
  - Input/Output graphics, CV-basics, Camera Calibration, Feature Detection/Extraction, Deep Learning
- **Pros:** Easy to use, good documentation, GPU boost
- **Cons:** Closed Environment, Performance

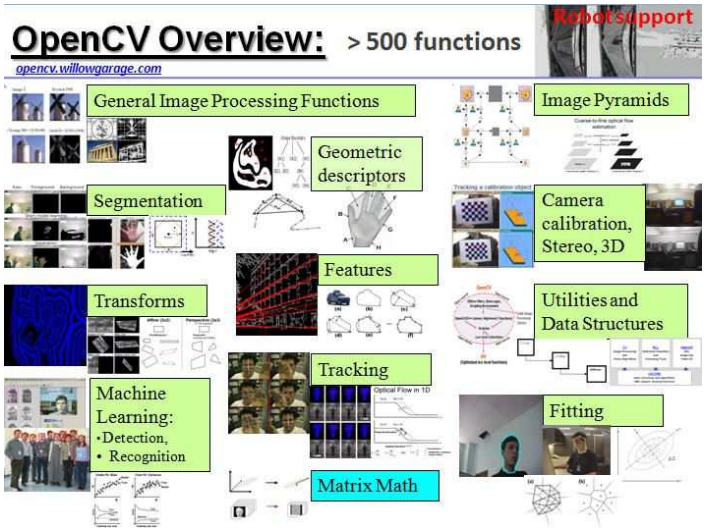


Source: https://www.youtube.com/watch?v=yipHjyLUzA4

Machine Vision – Computer Vision Software

- Free software for **everyone**
- OpenCV (Open Source Computer Vision)
- **Language:** C++, Python
- **Operating System:** Linux, macOS, Windows
- **Includes:**
  - Input/Output Graphics, CV Basics, Camera Calibration, Feature Detection/Extraction, Deep Learning, Egomotion, Motion Tracking
- **Pros:** Everything you need, Good Documentation, powerful, GPU boost
- **Cons:** Installation (especially GPU)

OpenCV



Source: https://www.pinterest.de/pin/174092341823170778/?lp=true

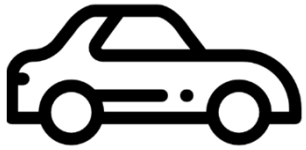
Perception

Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

Agenda

1. Chapter: History, Pipeline, Definitions
2. Chapter: Machine Vision
3. **Chapter: Image Processing**
4. Chapter: Features
5. Chapter: Feature Analysis
6. Chapter: Information Analysis
7. Chapter: Summary



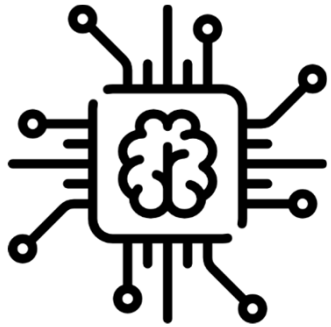
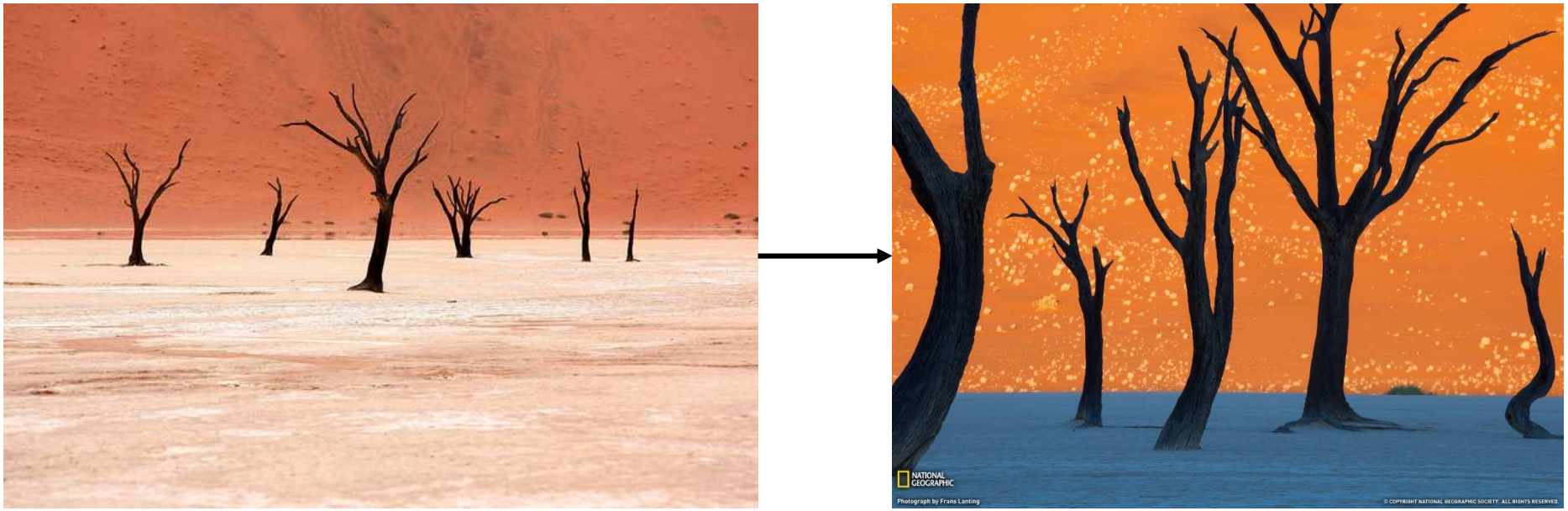


Image Processing

Image processing is a method to perform some operations on an image in order to get an **enhanced image** or to **extract** useful information from it.

In Image processing different **computer vision algorithms**/ **mathematical operations** are used



Source: Georgia Tech – CS 6476 Computer Vision – Fall 2018Lecture 6

Image Processing – The Pipeline

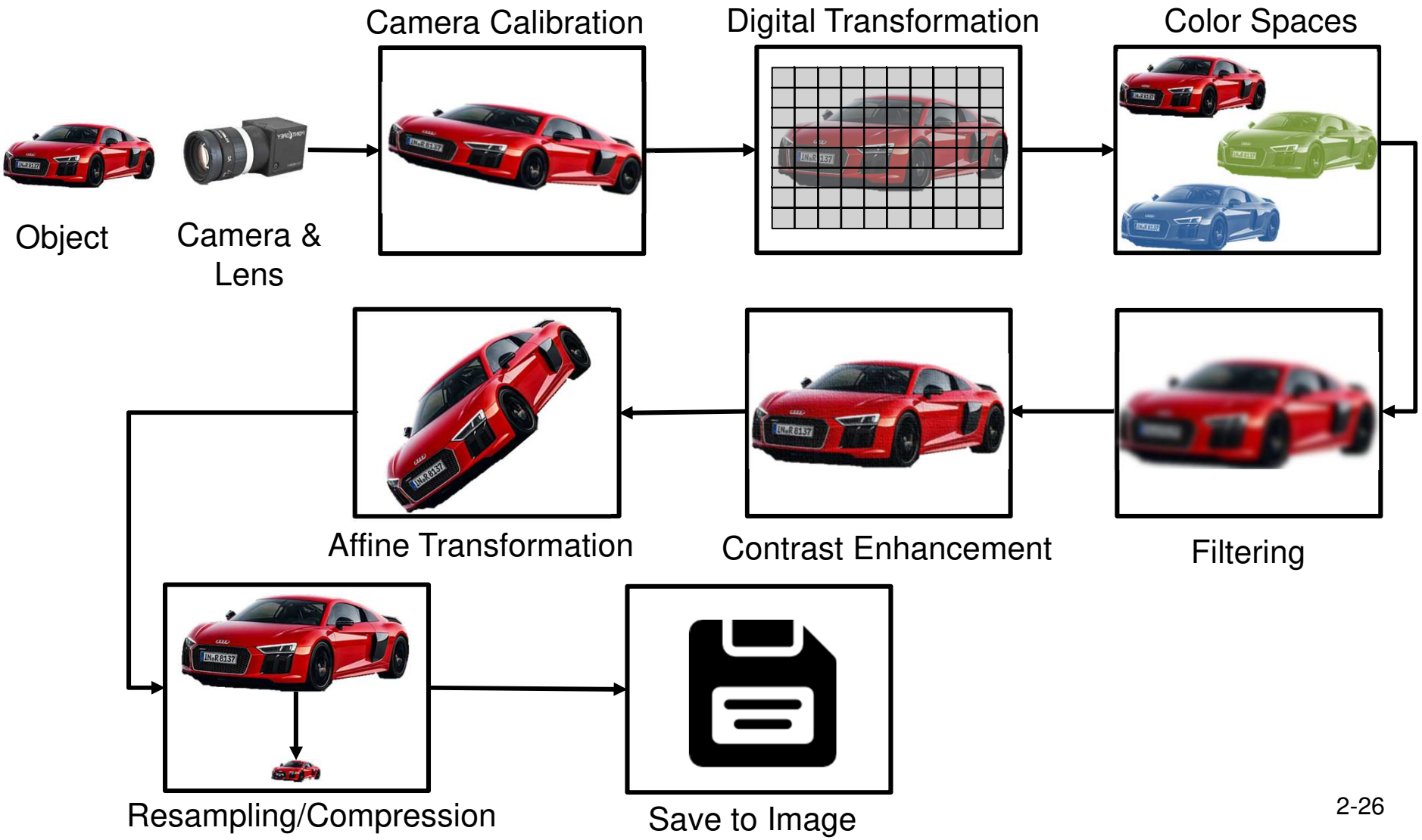
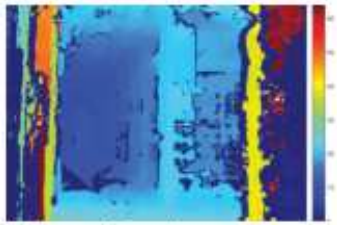


Image Processing – Camera Calibration

Remove Lens Distortion


Estimate 3-D Structure from Camera Motion


Estimate Depth
Using a Stereo Camera


Measure Planar Objects

- A camera projects 3D world-points onto a 2D image plane
- Todays cameras are cheap and produce distortion (radial, tangential)
- **Camera Calibration:** Finding the internal quantities of the camera and lens that affect the imaging processing:
  - Extrinsic camera parameters: Position of camera center, camera heading..
  - Intrinsic camera parameters: Focal length, image sensor format,…
  - Lens distortion parameters

Source: https://de.mathworks.com/help/vision/ug/camera-calibration.html

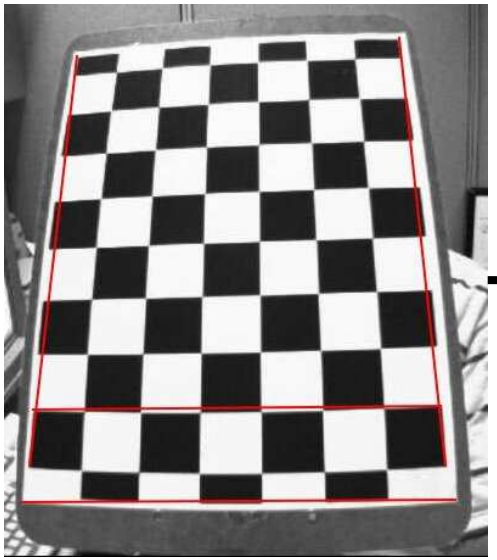
Additional Slide

Types of Distortion

Real cameras use curved lenses to form an image, and light rays often bend a little too much or too little at the edges of these lenses. This creates an effect that distorts the edges of images, so that lines or objects appear more or less curved than they actually are. This is called **radial distortion**, and it's the most common type of distortion.

Another type of distortion, is **tangential distortion**. This occurs when a camera's lens is not aligned perfectly parallel to the imaging plane, where the camera film or sensor is. This makes an image look tilted so that some objects appear farther away or closer than they actually are.

Image Processing – Camera Calibration Process

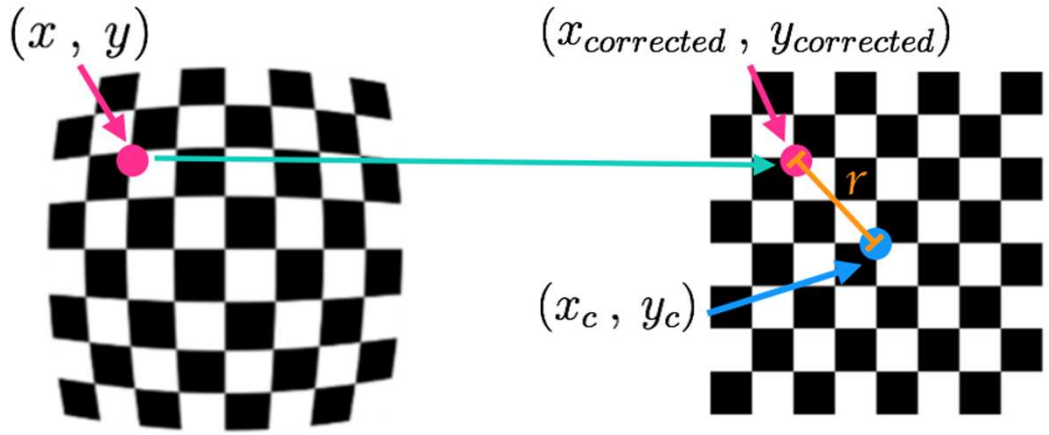


1. Distorted Picture: Lines or objects appear more or less curved than they actually are.

2. Find and draw corners in the picture → Find the distortion coefficients and correction

3. Undistortion: Compute the distortion matrix and apply it to the picture

Additional Slide



Distorted **Undistorted**

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Additional Slide

Interactive Lecture Code: Calibrate the Camera

1. Now we want to use our camera for further applications. To get undistorted images, we have to calibrate the camera. We can do this with different ways.

2. First of all we can use an internal calibration software. The ZED SDK is providing such an calibration software which makes it easy and fast to compute the distortion matrix. But be aware, not all camera SDK´s provide such a software.

3. We can do the calibration on our own. To do this, we have to write specific code which finds the parameters for distortions of the camera, intrinsic and extrinsic parameters of camera etc. We are using the OpenCV Python Tutorial which can be found here: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

    3.1 Print out a checkerboard : https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf

    3.2 Take at least 10 different pictures with the checkerboard and save them.

    3.2 Calculate the corners and distances in each picture.

    3.3 Calculate the camera matrix, distortion coefficients, rotation and translation vectors.

    3.4 Undistort any image with these parameters.

Lecture Code which is displayed now: **03_Camera_calibration.py**

Image Processing – What is an image?

- A **digital image** is a numeric representation, normally binary, of a two-dimensional image.

→ Standard: A digital image is just a **matrix of values** based on color.

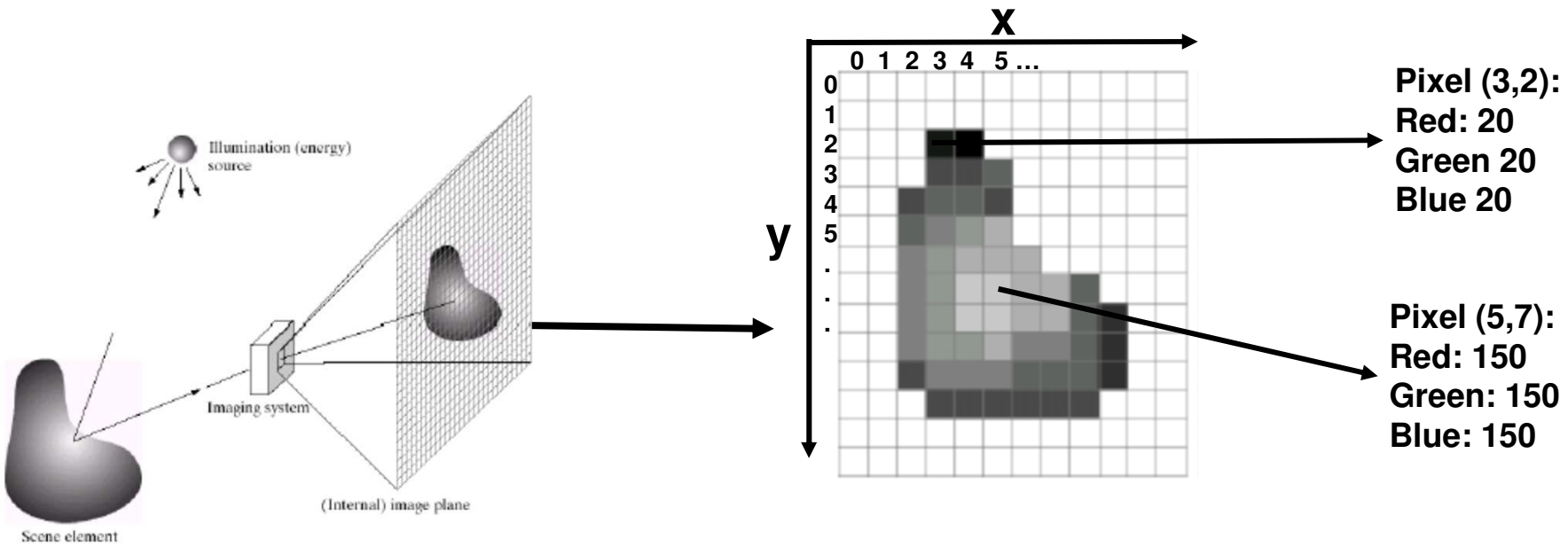
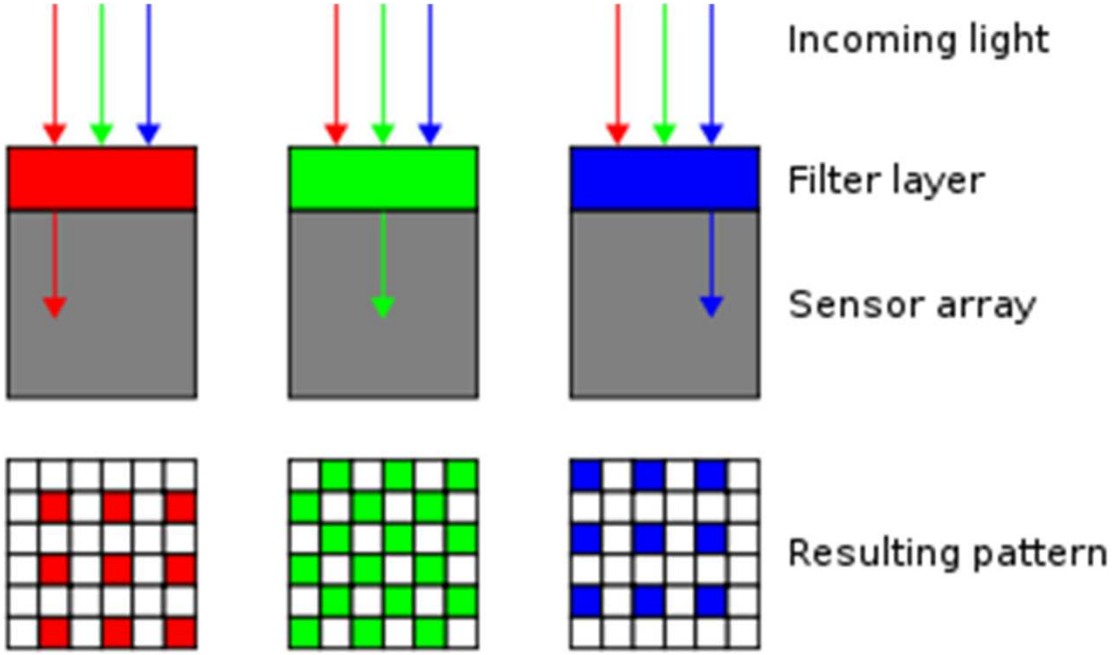


Image Processing – What is an image?

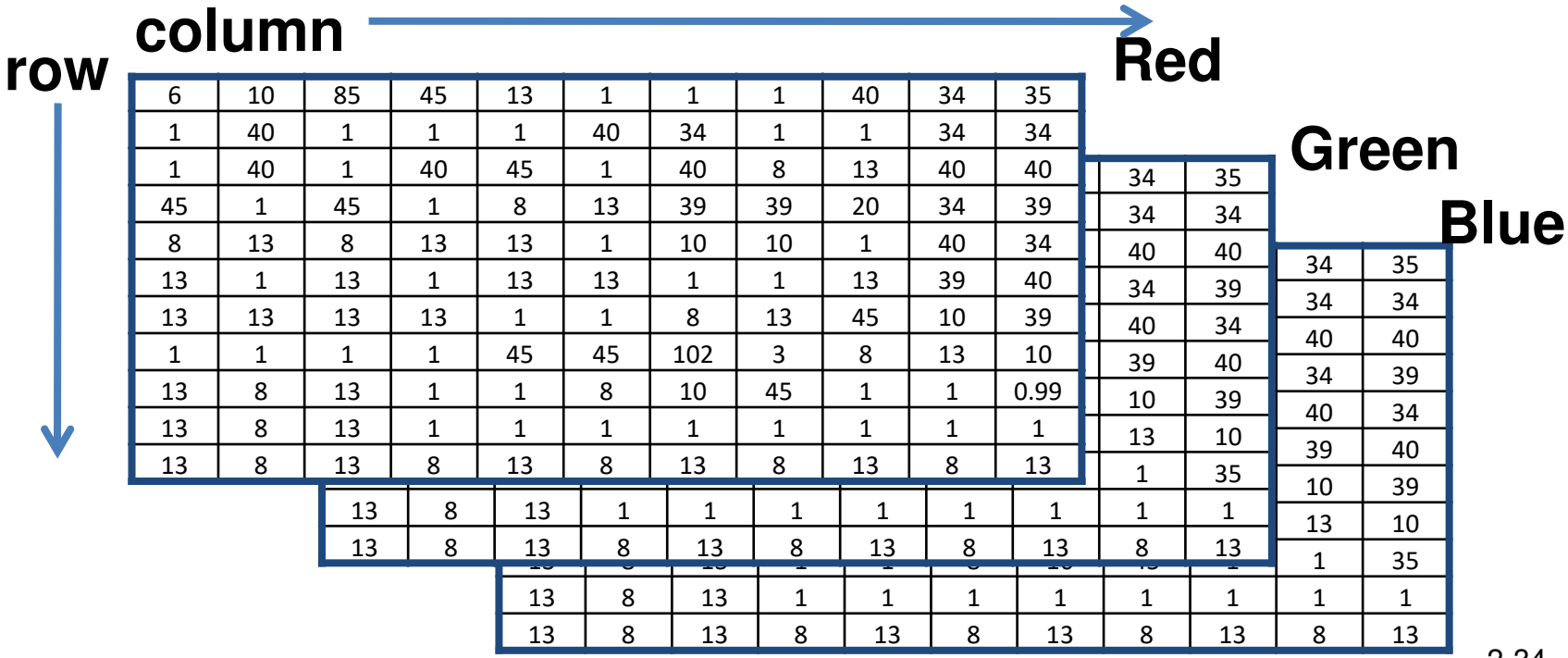
- A digital image consists of a rectangular grid of **evenly spaced pixels**.
- Each pixel can be thought of as a measurement or sample of the light from a subject.
- Commonly, the original samples are obtained using a scanner or digital camera by **averaging the amount of red, green and blue light** that falls on the sensitive area of each of its CCD sensing elements.



Source: https://en.wikipedia.org/wiki/Bayer_filter

Image Processing – Digital Transformation & Color Space

- Images represented as a matrix with a matrix size based on the camera resolution
- Suppose we have a **NxM** RGB image with a variable called **im**
  - –im(1,1,1) = top-left pixel value in R-channel
  - –im(y, x, b) = y pixels down, x pixels to right in the bth channel
  - –im(N, M, 3) = bottom-right pixel in B-channel



6	10	85	45	13	1	1	1	40	34	35
1	40	1	1	1	40	34	1	1	34	34
1	40	1	40	45	1	40	8	13	40	40
45	1	45	1	8	13	39	39	20	34	39
8	13	8	13	13	1	10	10	1	40	34
13	1	13	1	13	13	1	1	13	39	40
13	13	13	13	1	1	8	13	45	10	39
1	1	1	1	45	45	102	3	8	13	10
13	8	13	1	1	8	10	45	1	1	0.99
13	8	13	1	1	1	1	1	1	1	1
13	8	13	8	13	8	13	8	13	8	13

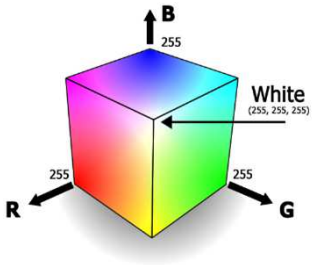
34	35									
34	34									
40	40									
34	39									
40	34									
39	40									
10	39									
13	10									
1	35									
1	1									
13	8	13	1	1	1	1	1	1	1	1
13	8	13	8	13	8	13	8	13	8	13

34	35									
34	34									
40	40									
34	39									
40	34									
39	40									
10	39									
13	10									
1	35									
1	1									
13	8	13	1	1	1	1	1	1	1	1
13	8	13	8	13	8	13	8	13	8	13

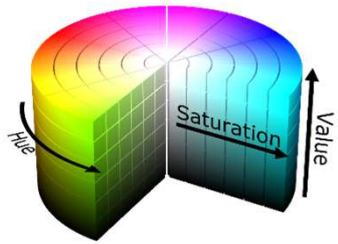
Image Processing – Color Spaces

- Reproducing human color perception/vision
- **Different color spaces for different problems** (monitor, printer, …)
- Different color spaces for more exact color extraction
- Most common color spaces are **RGB** and **HSV**

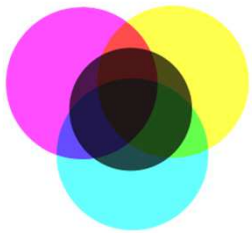
RGB



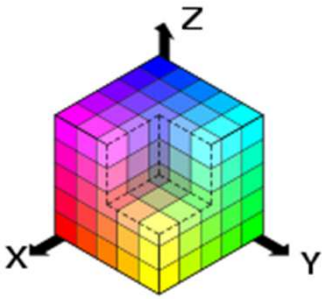
HSV



CMYK



CIE

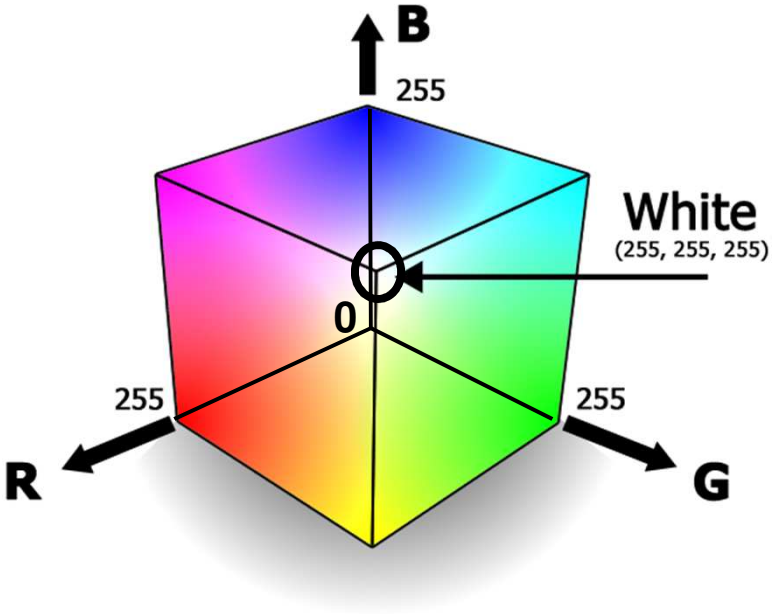


Source:
https://old.medialooks.com/mformats/docs/CK%20Advanced.html
https://cs.wikipedia.org/wiki/Soubor:HSV_color_solid_cylinder.png
https://design.tutsplus.com/articles/advanced-color-theory-what-is-color-management--cms-26307
https://www.codeproject.com/articles/243610/the-known-colors-palette-tool-final-revision-hopef

Image Processing – Color Spaces

RGB – **R**ed **G**reen **B**lue

- Values between 0 – 255 (8bit)
- 0,0,0: Black
- 255,255,255: White



R = 255
(G=0,B=0)

G = 255
(R=0,B=0)

B = 255
(R=0,G=0)

Source: https://old.medialooks.com/mformats/docs/CK%20Advanced.html

Additional Slide

RGB

The most common is the RGB color space with 8 bits per channel, corresponding to $(2^8)^3$ = 16,777,216 (approx. 16.8 million) theoretically possible colors.

With 16 bits (per channel) this results in 281,474,976,710,656 (281 trillion) color options

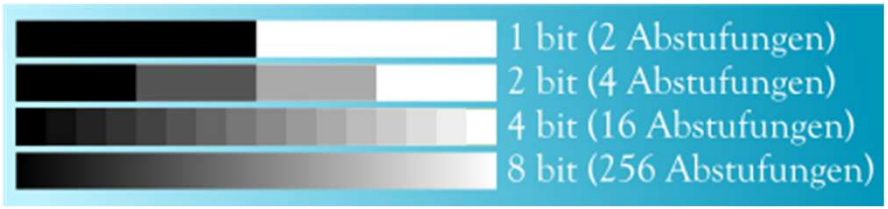
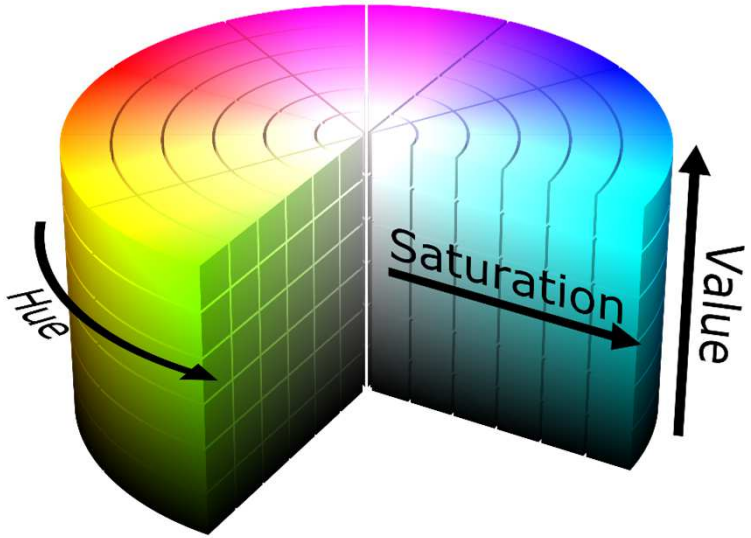
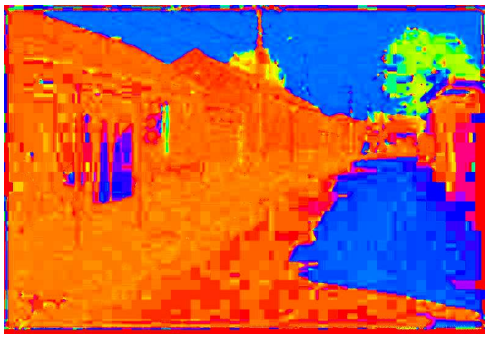


Image Processing – Color Spaces

HSV – **H**ue **S**aturation **V**alue

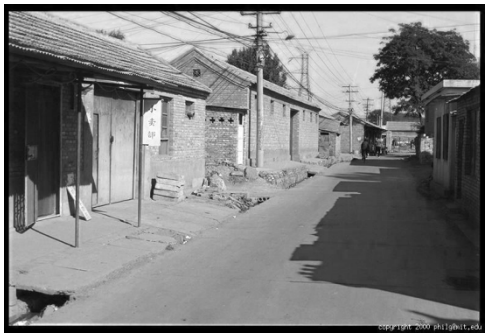
- Hue (0° Red, 120° Green, 240° Blue)
- Saturation (0% Gray – 100% saturated Color)
- Value (0% dark – 100% light)





H
(S=1,V=1)

S
(H=1,V=1)



V
(H=1,S=0)

Source: https://cs.wikipedia.org/wiki/Soubor:HSV_color_solid_cylinder.png

Additional Slide

Interactive Lecture Code: Live Color Change

1. Now we are showing how to extract different colors or color spaces from an image.

2. First we are displaying a live stream from the camera. The live stream is just displaying one frame/image after another. Each frame/image can be seen as a matrix of values.

3. We are changing the current live stream to a greyscale image.

4. We are splitting the live stream into three images. Each image is showing either the red, blue or green colors of the live stream.

5. At the end, we are changing the color space from RGB to HSV. In addition, we are splitting the HSV live stream into three images. Each image is showing the Hue, Saturation and Value of the live stream.

Lecture Code which is displayed now: **04_Change_color_spaces.py**

Image Processing – Filtering

- Image filters in **spatial domain:** Filter is a mathematical operation of a grid of numbers → Methods: Smoothing, sharpening, measuring texture

- Image filters in the **frequency domain:** Filtering is a way to modify the frequencies of images → Methods: Denoising, sampling, image compression

- Templates and Image Pyramids: Filtering is a way to match a template to the image → Methods: Detection, coarse-to-fine registration

Image Processing – Filtering

- Preprocessing image to **emphasize information** for next steps
- Mathematic multiplication with quadratic or odd kernel, convolution matrix or mask
- Convolution kernel depending on purpose/effect

Image Processing – Filtering Examples

Mean

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Blur
Gauss 3x3

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Blur
Gauss 5x5

$$\frac{1}{256}\begin{bmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{bmatrix}$$

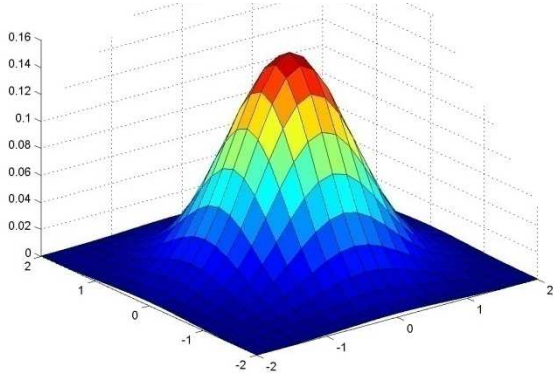
Sharpen

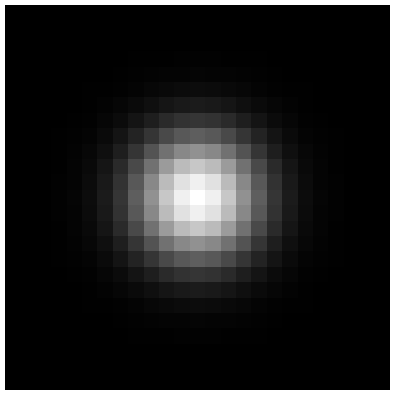
$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Additional Slide

Gaussian Filter

- Remove “high-frequency” components from the image (low-pass filter)
  Images become more smooth
- Convolution with itself is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Additional Slide

Interactive Lecture Code: Image Filter

1. Now we are showing how to apply a filter to a live image.

2. First we are displaying a live stream from the camera. The live stream is just displaying one frame/image after another. Each frame/image can be seen as a matrix of values.

3. We are changing the current live stream to a greyscale image.

4. In the next step we are applying a 2D Convolutional Filter to the live stream.

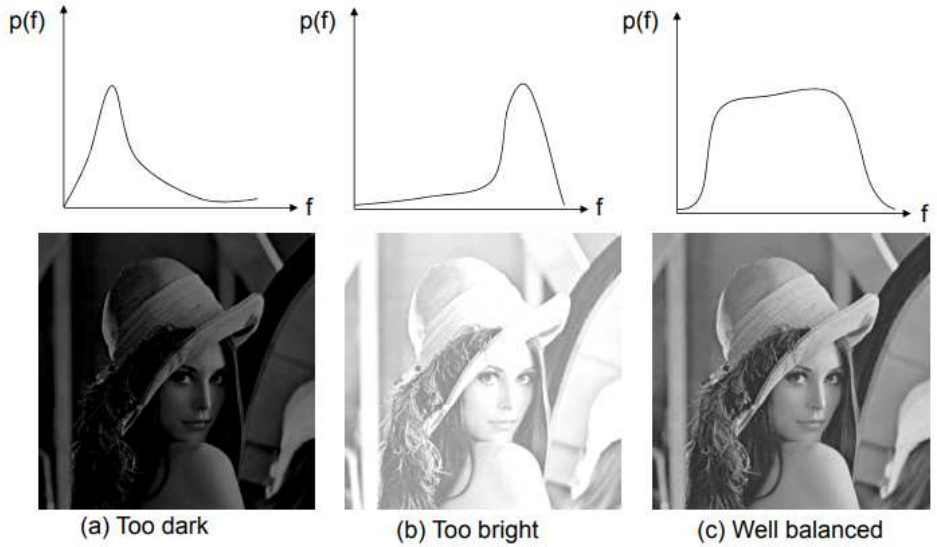
5. After that we are applying a Blurring/smoothing filter (Gaussian Blur) Filter to the live stream.

Lecture Code which is displayed now: **05_Image_filtering.py**

Image Processing – Contrast Enhancement

Contrast: Contrast is the range of difference between different tones in a photograph. In black and white photography, contrast describes the difference between the darkest and lightest tones, but it also defines the grayscale. In color photography, contrast applies to how sharply colors stand out from one another.

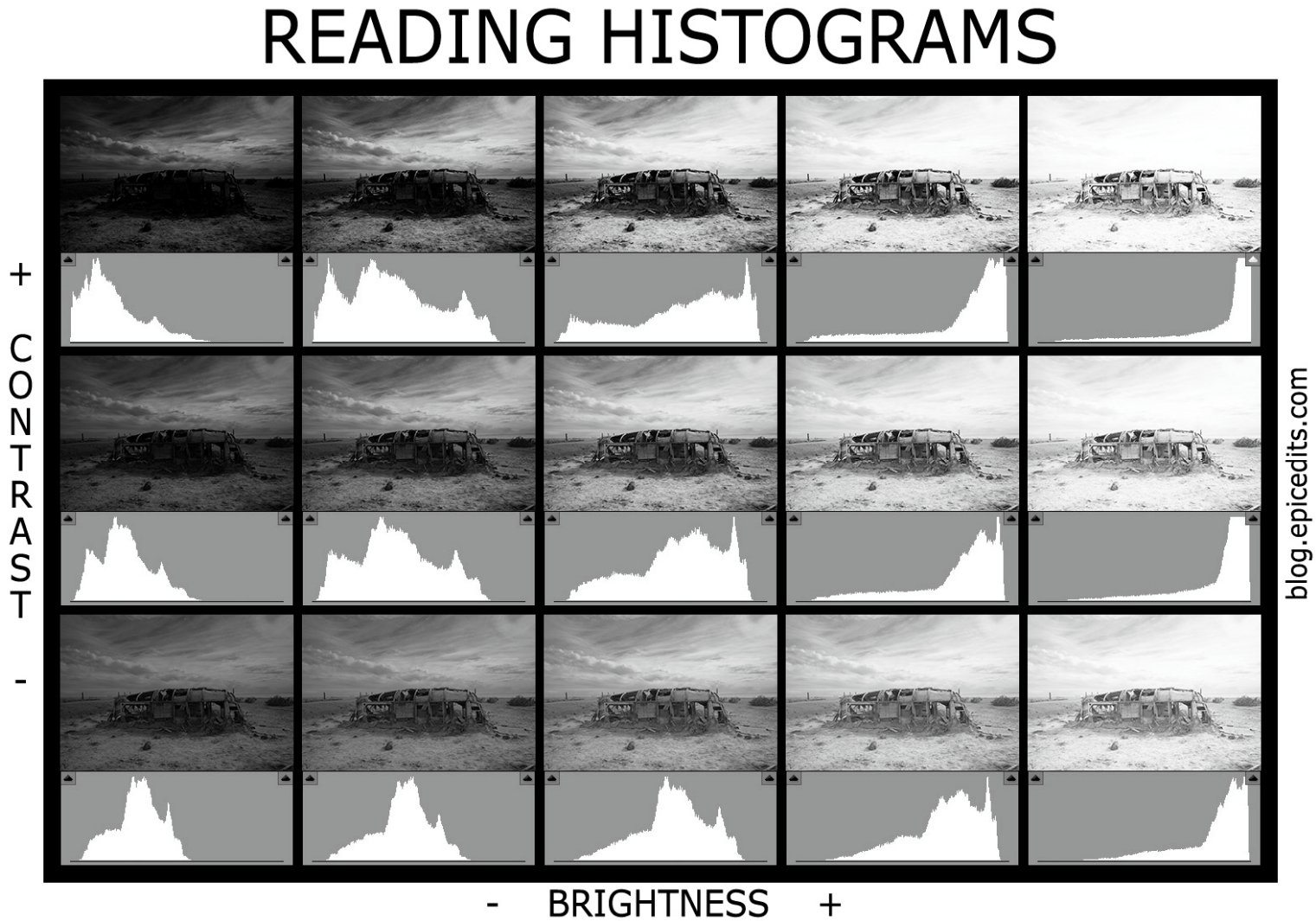
- **Low contrast** → Image values concentrated near a narrow range (mostly dark, or bright or medium values).
- Contrast of an image can be revealed by its histogram: Representation of the tonal distribution in an digital image.
- **Contrast enhancement** → Change the image value distribution to cover a wide range.



P(f) = Number of Pixel f = tonal distribution

Source: http://eeweb.poly.edu/~yao/EL5123/lecture3_contrast_enhancement.pdf

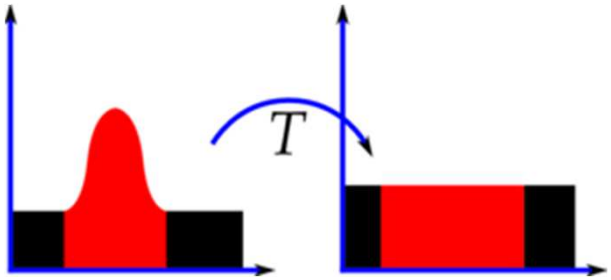
Image Processing – Contrast Enhancement



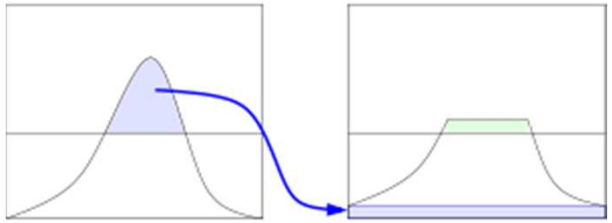
Source: http://eeweb.poly.edu/~yao/EL5123/lecture3_contrast_enhancement.pdf

Image Processing – Contrast Enhancement

Methods



Histogram Equalization



Adaptive Histogram Equalization:
CLAHE (Contrast Limited Adaptive Histogram Equalization)

Source: https://research.ijcaonline.org/volume64/number17/pxc3885679.pdf

Additional Slide

Interactive Lecture Code: Contrast Enhancement

1. Now we are showing how to do a contrast enhancement for an image
2. First we are taking an image with the camera
3. Then we are plotting a histogram, which gives us an overview over the pixel density
4. At the end we are applying a histogram equalization with the function cv2.equalizeHist

Lecture Code which is displayed now: **06_Contrast_Enhancement.py**

Image Processing – Affine Transformations

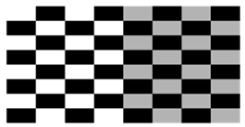
- In geometry, an **affine transformation**, **affine map** or an **affinity** (from the Latin, *affinis*, "connected with") is a function between affine spaces which preserves points, straight lines and planes.
- Sets of parallel lines remain parallel after an affine transformation.
- An affine transformation does not necessarily preserve angles between lines or distances between points, though it does preserve ratios of distances between points lying on a straight line.

Rotate $\begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Reflect $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Shear $\begin{pmatrix} 1 & cx=0.5 & 0 \\ cy=0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Scale $\begin{pmatrix} cx=2 & 0 & 0 \\ 0 & cy=1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

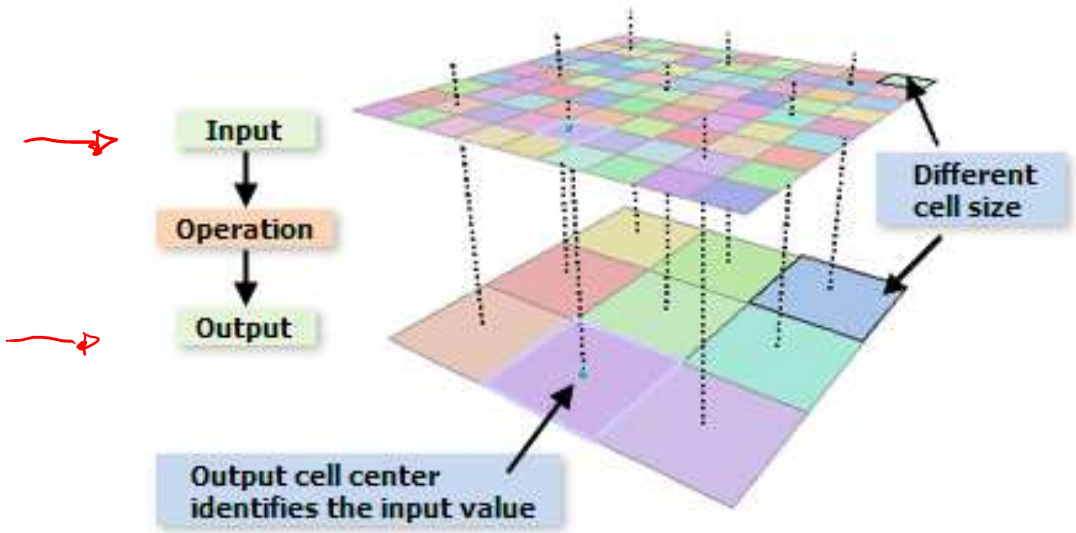


Source: https://en.wikipedia.org/wiki/Affine_transformation

Image Processing – Re-Sampling

- Resampling is the mathematical technique used to create a **new version** of the image with a different width and/or height in pixels. Increasing the size of an image is called **upsampling;** reducing its size is called **downsampling.**

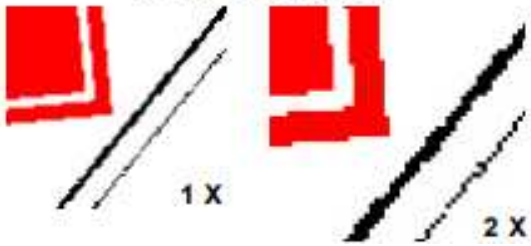
- A value for each cell in the new raster object must be computed by sampling or interpolating over some neighborhood of cells in the corresponding position in the original raster object.



Source: https://desktop.arcgis.com/de/arcmap/latest/extensions/spatial-analyst/performing-analysis/cell-size-and-resampling-in-analysis.htm

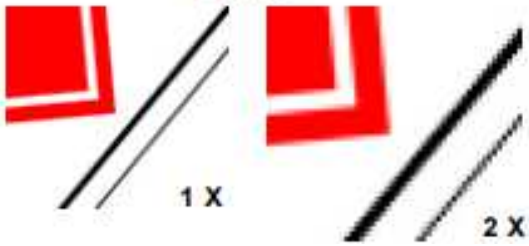
Image Processing – Re-Sampling

Nearest Neighbor Method



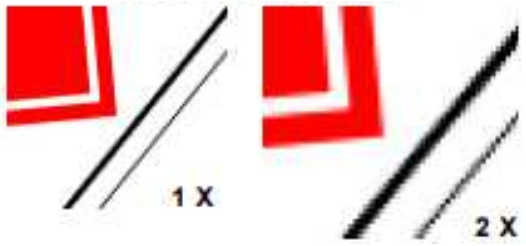
Before After

Bicubic Method



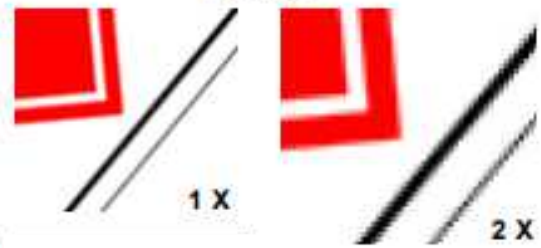
Before After

Bicubic Smoother Method



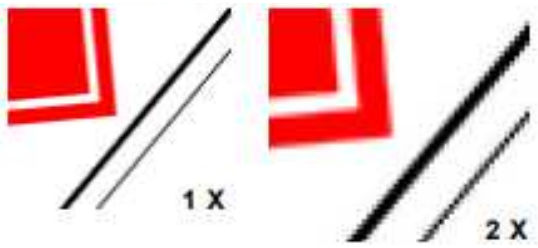
Before After

Bilinear Method



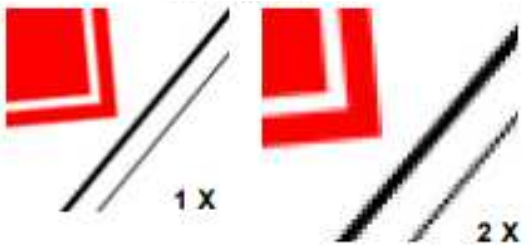
Before After

Bicubic Sharper Method



Before After

Lanczos 4x4 Method



Before After

Source: http://www.dl-c.com/Temp/downloads/Whitepapers/Resampling.pdf

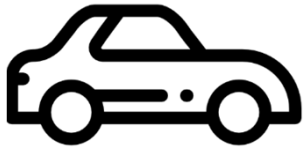
Perception

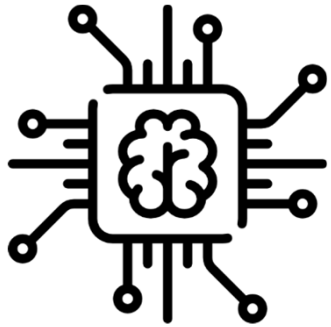
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

Agenda

1. Chapter: History, Pipeline, Definitions
2. Chapter: Machine Vision
3. Chapter: Image Processing
4. **Chapter: Features**
5. Chapter: Feature Analysis
6. Chapter: Information Analysis
7. Chapter: Summary





Feature Extraction

Which of the following features could be useful in the identification of lane lines on the road?

Source: Udacity Course „Selfdriving Nanodegre“ Lecture 1

Feature Extraction

- A **Feature** is a piece of information which is relevant for solving the computational task related to a certain application
- **Feature Detection** includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not.
  - The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions.
  - Examples: Edges, Corners, Region of Interest, Ridges

- **Feature Extraction:** Once features have been detected, a local image patch around the feature can be extracted. For Computer Vision this means the isolation of various desired portions or shapes (features) of a digitized image or video stream

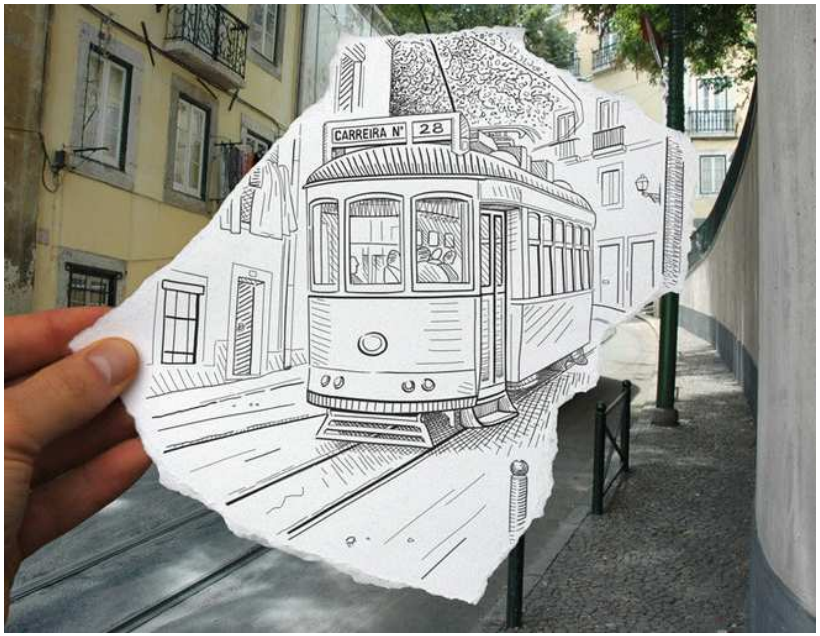
Feature Extraction

- Feature points are used for:
  - Image alignment
  - 3D reconstruction
  - Motion tracking
  - Robot navigation
  - Indexing and database retrieval
  - Object recognition

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 05

Feature Extraction – Edge Detection

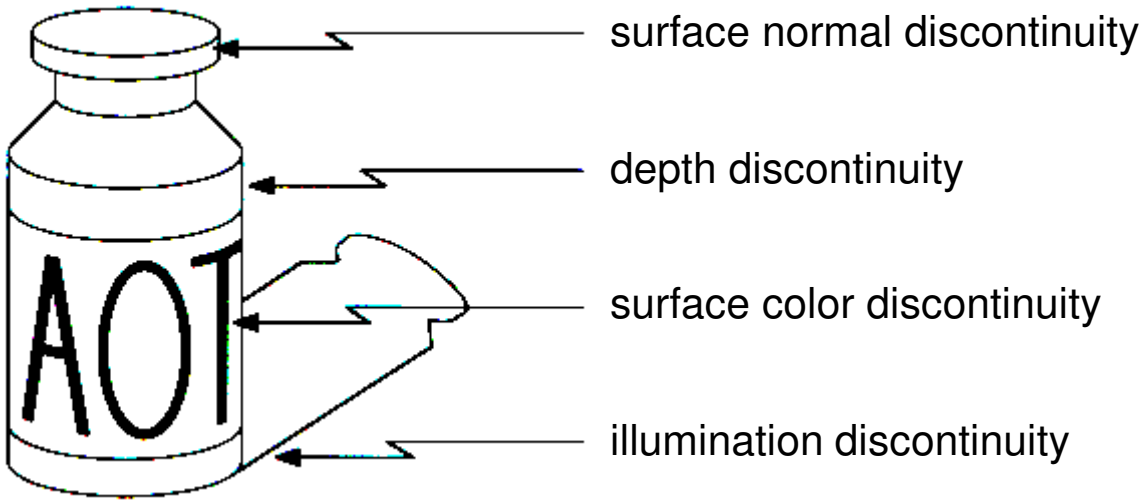
- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels
- **Ideal:** Artist's line drawing (but artist is also using object-level knowledge)



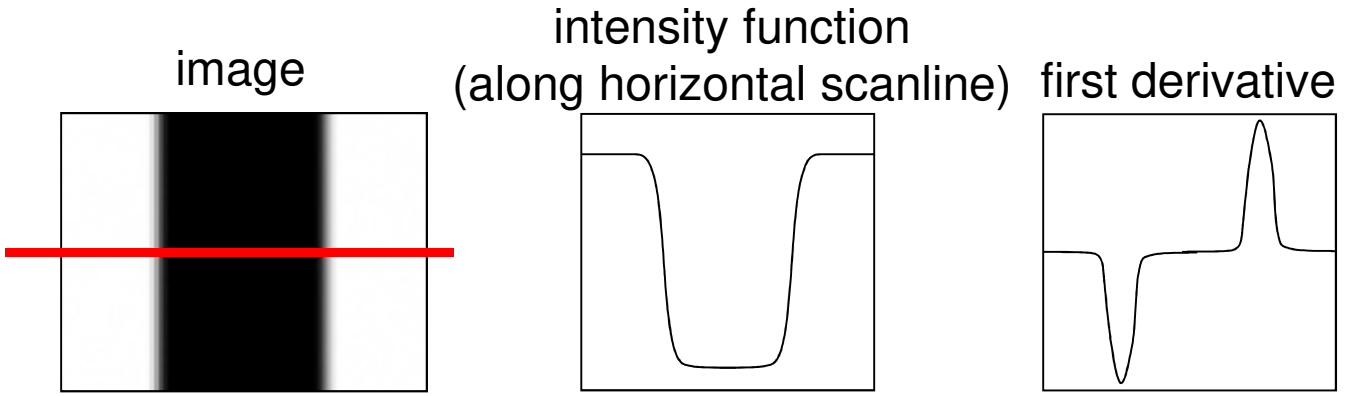
Source: https://curiator.com/art/ben-heine/1

Feature Extraction – Edge Detection

- Edges are caused by a variety of factors

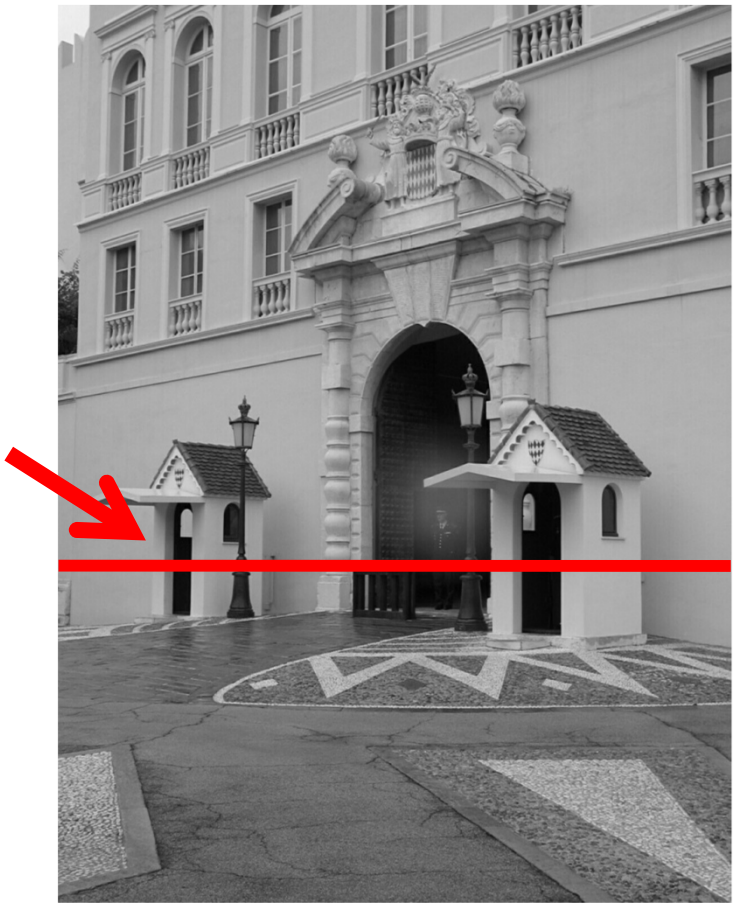


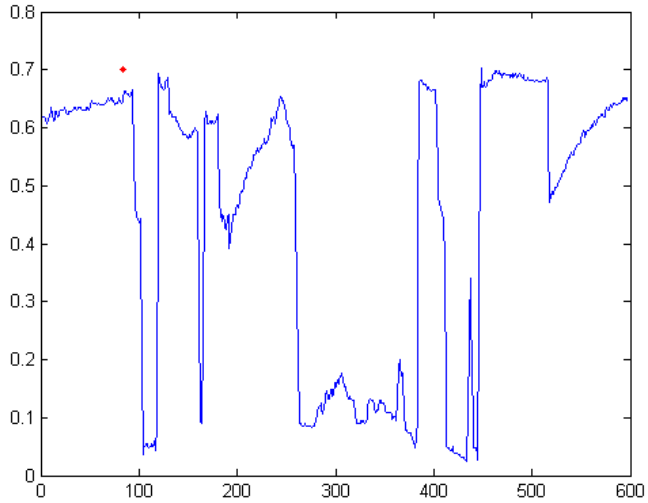
- An edge is a place of rapid change in the image **intensity function**



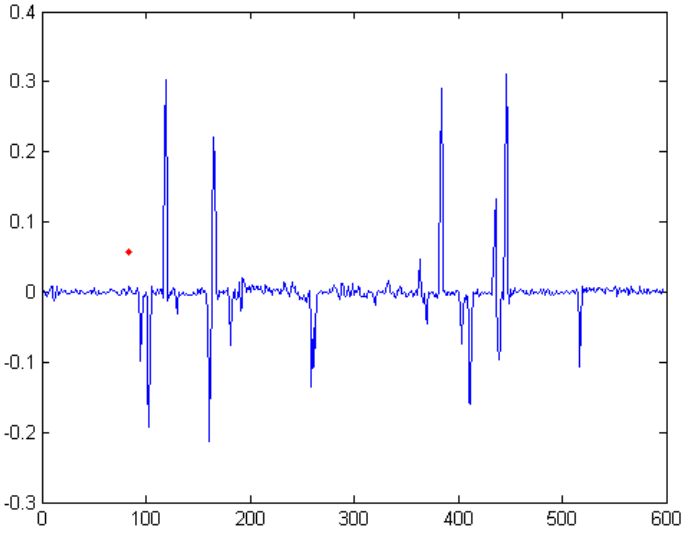
Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 07

Feature Extraction – Edge Detection





intensity function (along horizontal scanline)



first derivative

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 07

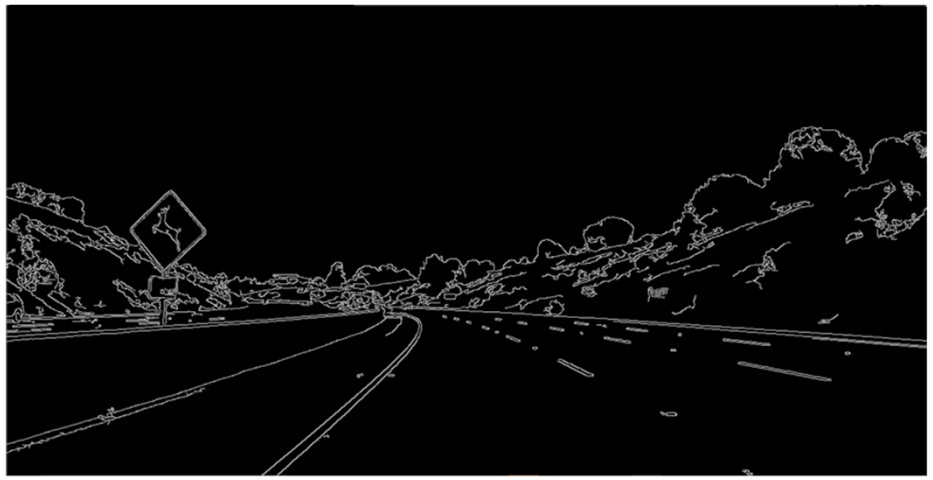
Feature Extraction – Canny Edge Detection

- Don't do Edge Detection by yourself: **Use the Canny Edge Detection Algorithm**
- This is probably the most widely used edge detector in Computer Vision
- Theoretical model: step-edges corrupted by **additive Gaussian** noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

Paper:
J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

27,000 citations!

Additional Slide



1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression: Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

- Adjust Thresholds: Threshold at low/high levels to get weak/strong edge pixels
- Adjust Kernel-Size: Large kernel detects large scale, low kernel fine features

Additional Slide

Interactive Lecture Code: Canny Edge Detection

1. Now we are showing how to apply a canny edge detection to a live image.

2. First we are displaying a live stream from the camera. The live stream is just displaying one frame/image after another. Each frame/image can be seen as a matrix of values.

3. We are changing the current live stream to a greyscale image.

4. After that we are applying a Blurring/smoothing filter (Gaussian Blur) filter to the live stream.

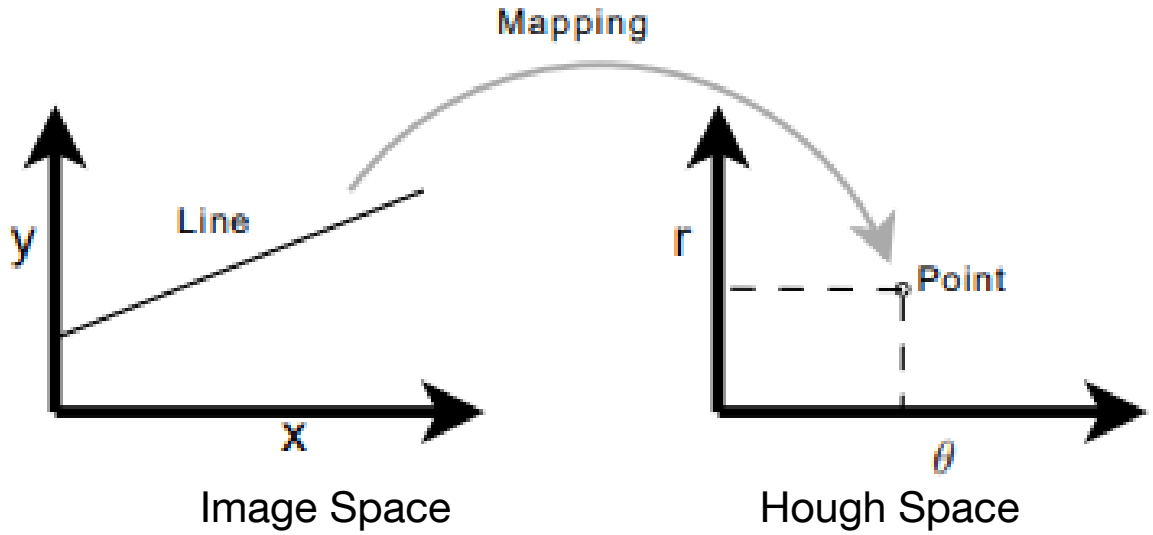
5. At the end, we are applying a Canny Edge Detection to the livestream.
https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

Lecture Code which is displayed now: **07_Canny_Edge_Detection.py**

The Problem

- Edge detection makes it possible to reduce the amount of data in an image considerably. However the output from an edge detector is **still an image described by it's pixels.**
- If lines, ellipses and so forth could be defined by their characteristic equations, the amount of data would be reduced even more.
- We can use the **Hough Line Algorithm**, which was originally developed to recognize lines and has later been generalized to cover arbitrary shapes e. g. circles

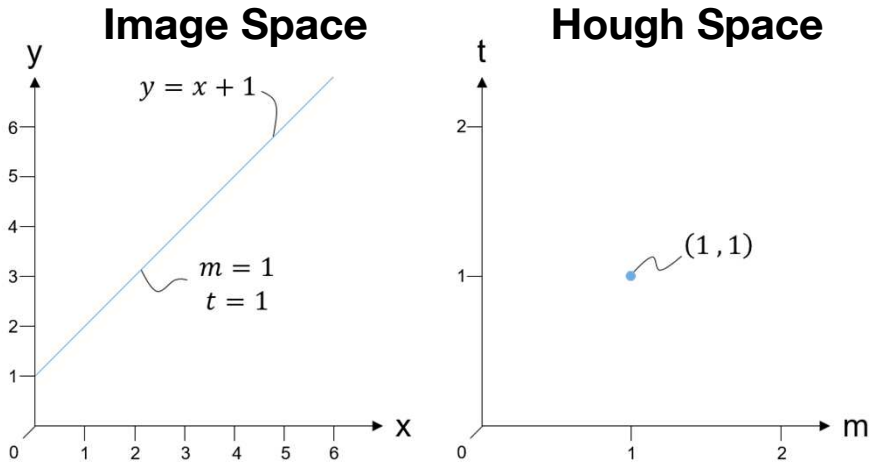
Feature Extraction – Hough Line



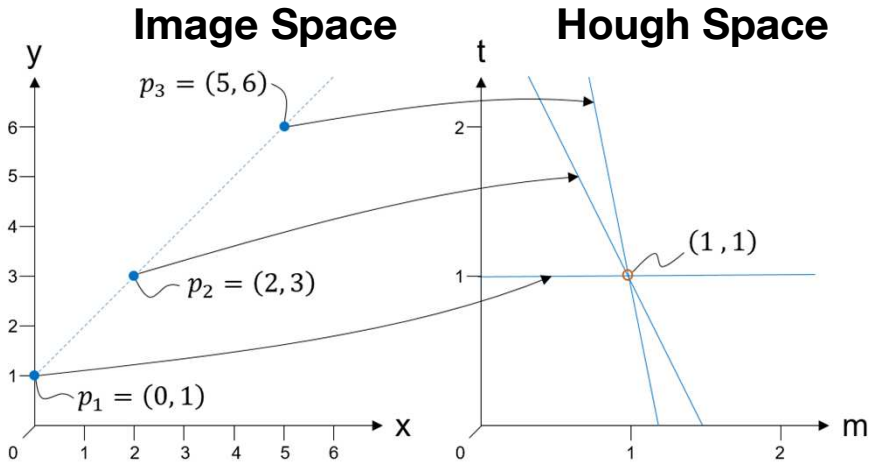
The algorithm for detecting straight lines can be divided into the following steps:

1. **Edge detection**, e.g. using the Canny edge detector
2. **Mapping** of edge points to the **Hough space** and storage in an accumulator.
3. **Interpretation** of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and possibly other constraints.
4. **Conversion** of infinite lines to finite lines.

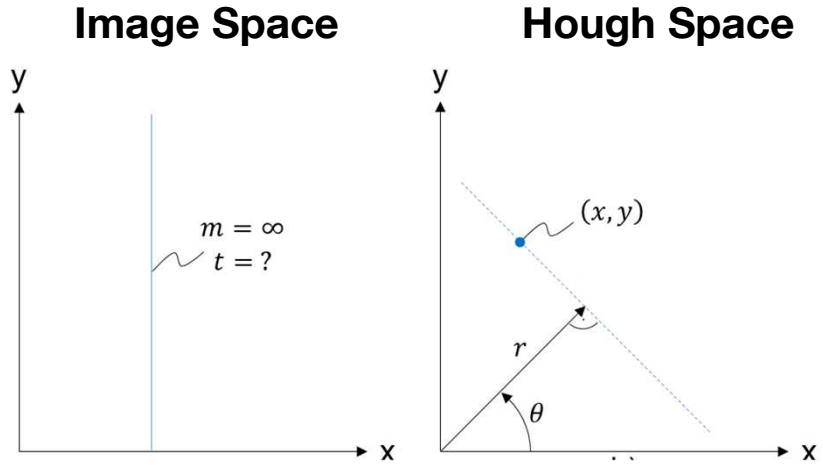
Additional Slide



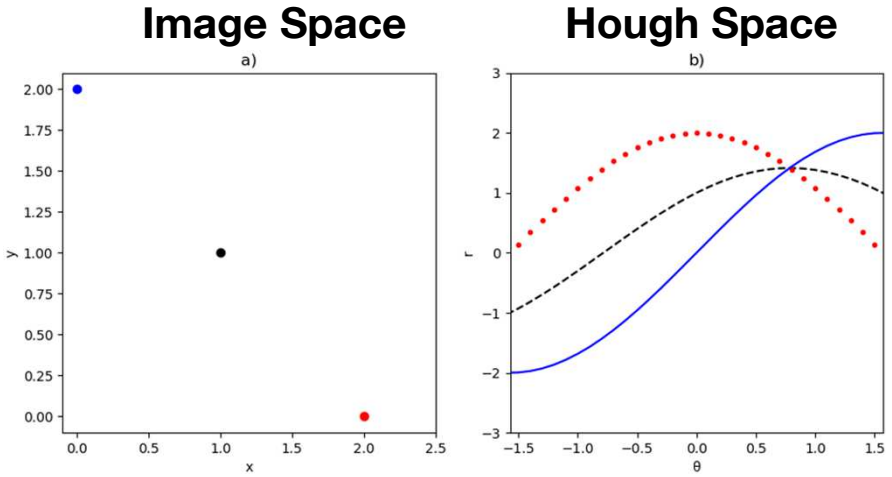
Line to Point



Point to Line

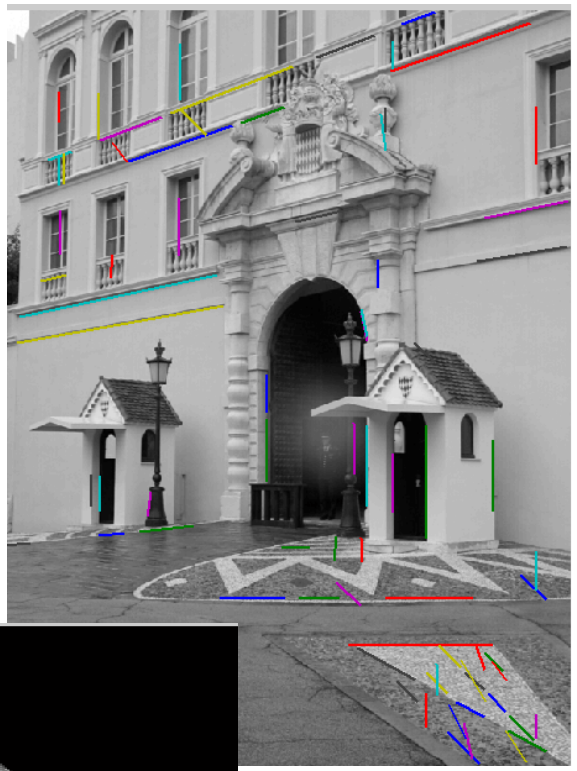


Line to Point



Point to Line

Feature Extraction – Hough Line Example



Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 07

Additional Slide

Interactive Lecture Code: Hough Line Detection

1. Now we are showing how to apply a canny edge detection to a live image

2. First we are displaying a live stream from the camera. The live stream is just displaying one frame after another. Each frame can be seen as a matrix of values

3. We are changing the current live stream to a greyscale image

4. After that we are applying a Blurring/smoothing filter (Gaussian Blur) Filter to the live stream

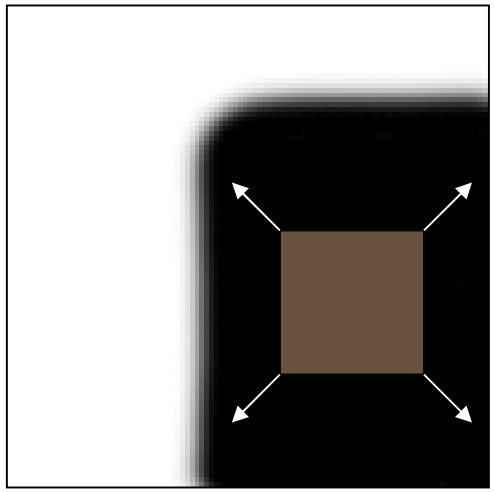
5. After that we are applying a Canny Edge Detection to the livestream.

6. At the end we are applying a Hough line detection with the function cv2. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
   The function needs different parameters for input

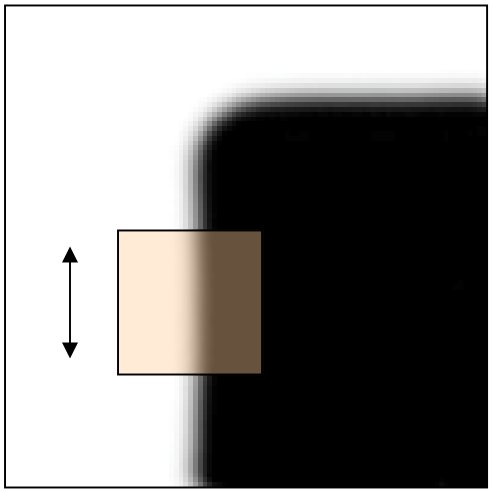
Lecture Code which is displayed now: **08_Hough_Line_Detection.py**

Feature Extraction – Corner Detection

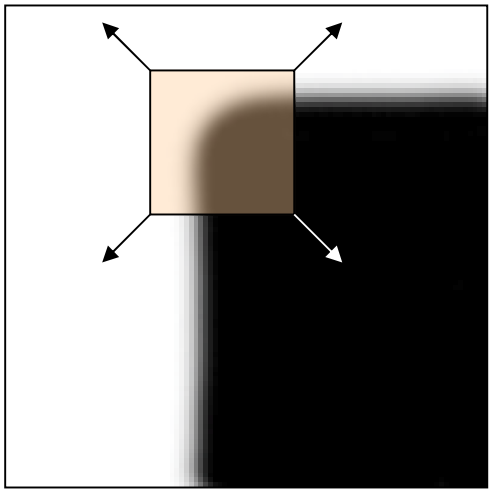
- We should easily recognize the point by looking through a small window
- Shifting a window in **any direction** should give **a large change** in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction

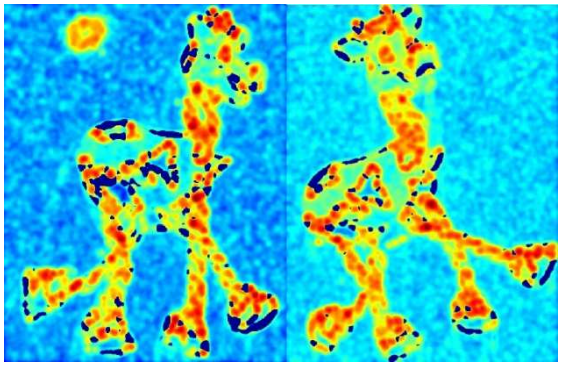


"corner":
significant change in all directions

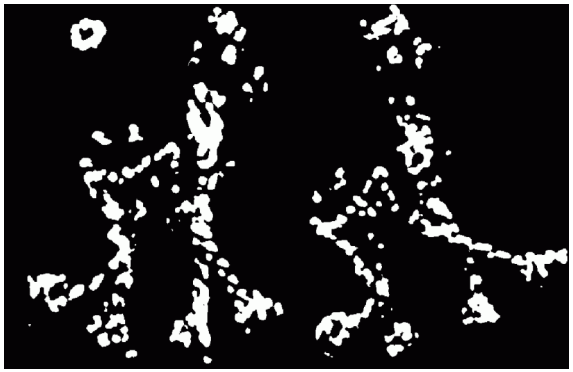
Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 08

Additional Slide

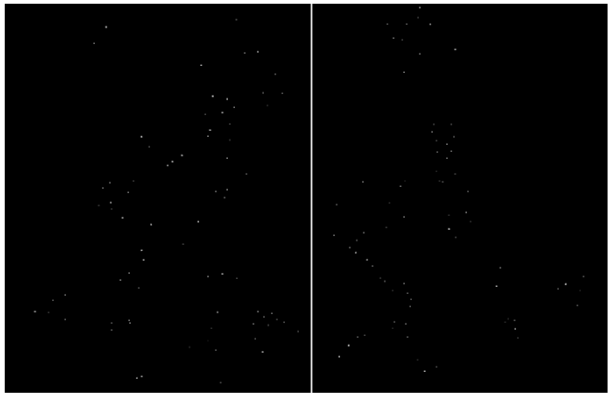
1. Original Image



2. Compute Corner Response R



3. Find Points with larger corner Response

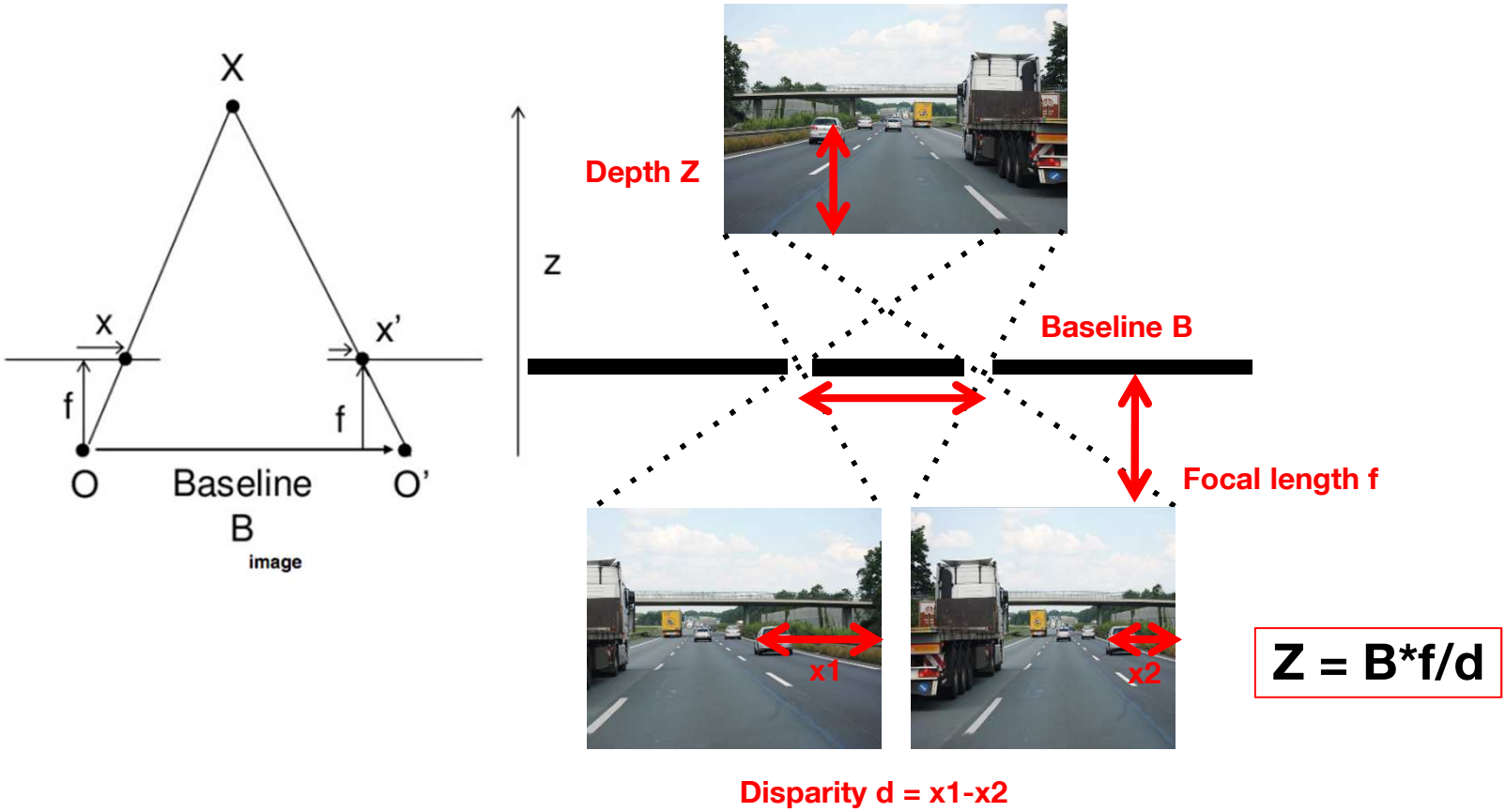


4. Points of local maxima of R

5. Show points in image

Feature Extraction – Depth Detection

- Disparity between 2 images → disparity map



Feature Extraction – Depth map

Additional Slide

Feature Extraction – Dilation, Erosion, Opening, Closing

- Morphology-based operations to connect points and edges and build regions/areas
- A set of operations that process images based on shapes. Morphological operations apply a **structuring element** to an input image and generate an output image.
- Dilation and Erosion are **minkowski addition and subtraction**
- Opening and Closing are **combined** Dilation and Erosion

Additional Slide

Feature Extraction – Dilation, Erosion, Opening, Closing

Dilation:

- The operations consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle.
- Kernel B has a defined anchor point, usually being the center of the kernel.
- As kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name dilation). Take as an example the image above. Applying dilation we can get:

Additional Slide

Feature Extraction – Dilation, Erosion, Opening, Closing

Erosion:

This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel.

As kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.

Analogously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see in the result below that the bright areas of the image (the background, apparently), get thinner, whereas the dark zones (the "writing") gets bigger.

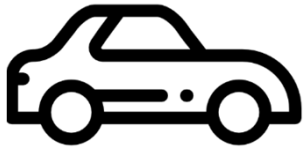
Perception

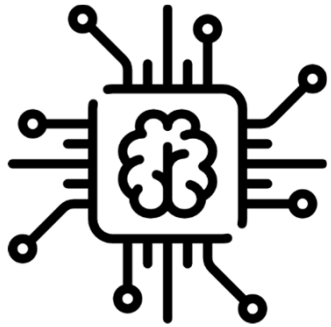
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

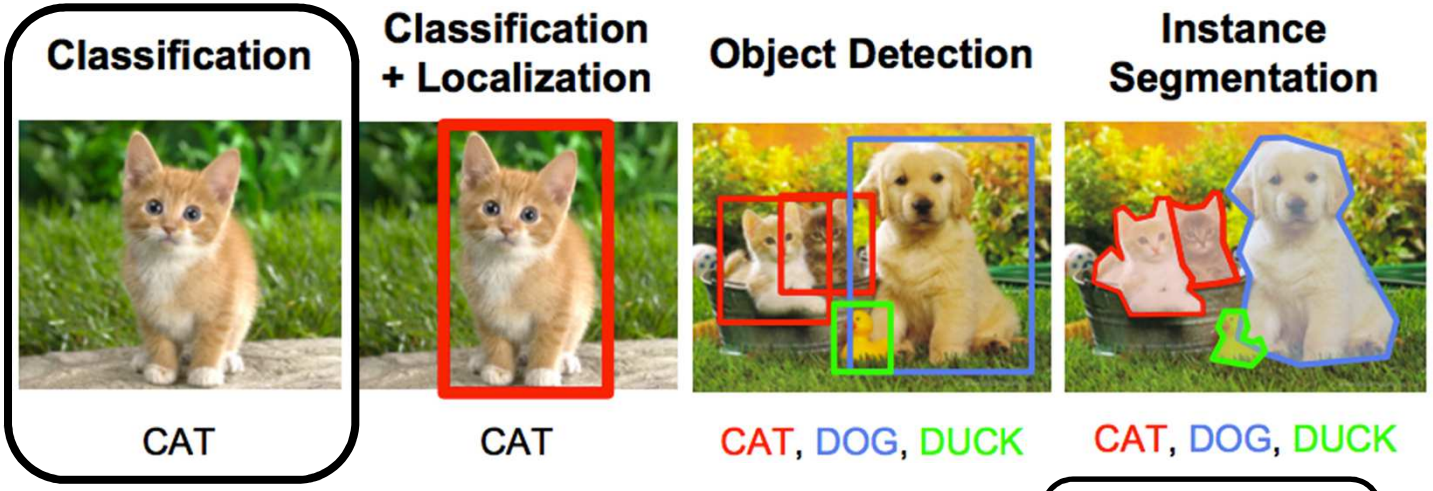
Agenda

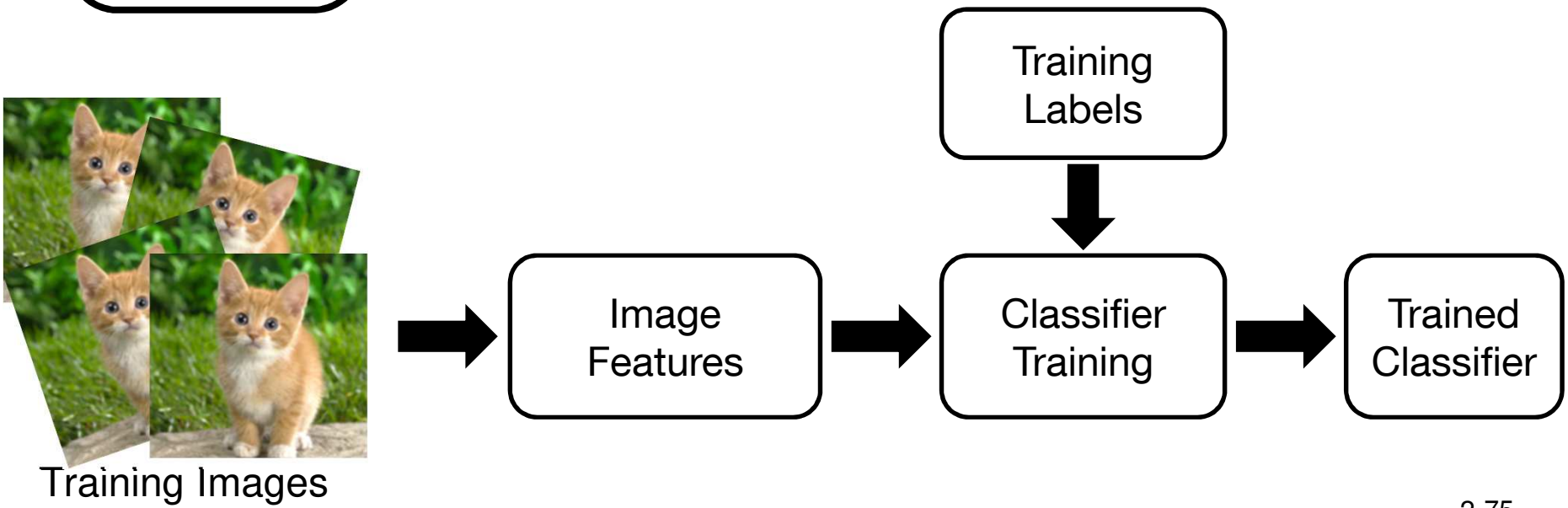
1. Chapter: History, Pipeline, Definitions
2. Chapter: Machine Vision
3. Chapter: Image Processing
4. Chapter: Features
5. **Chapter: Feature Analysis**
6. Chapter: Information Analysis
7. Chapter: Summary





Feature Analysis – Image Classification



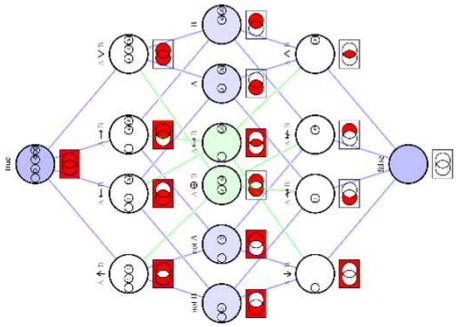


Source: https://www.datacamp.com/community/tutorials/object-detection-guide

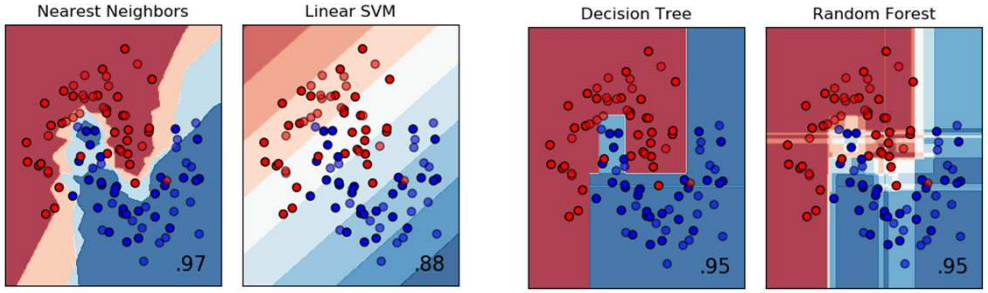
Feature Analysis – Classification

Methods for Classification:

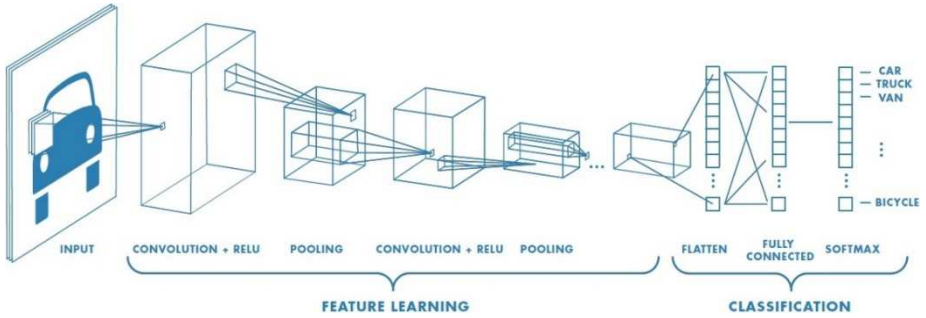
1. **Rule based, logic, datebase:**



2. **Machine Learning:**



3. **Deep Learning:**



Source: https://de.mathworks.com/solutions/deep-learning/convolutional-neural-network.html?requestedDomain /https://commons.wikimedia.org/wiki/Logic_diagram

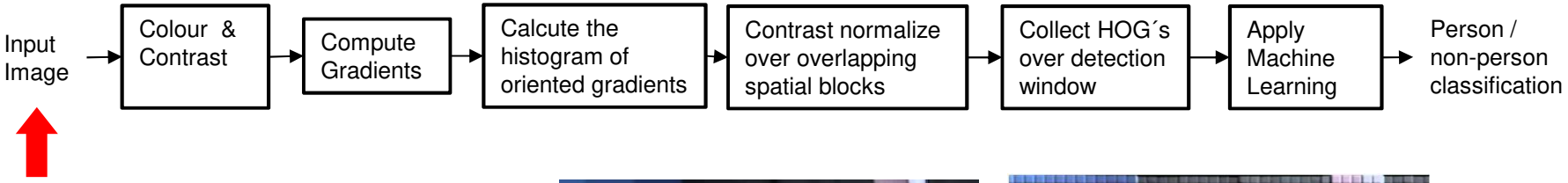
Feature Analysis – Classification with Manual Features

Object

Bag of features: Gradients, Edges, Pixels

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

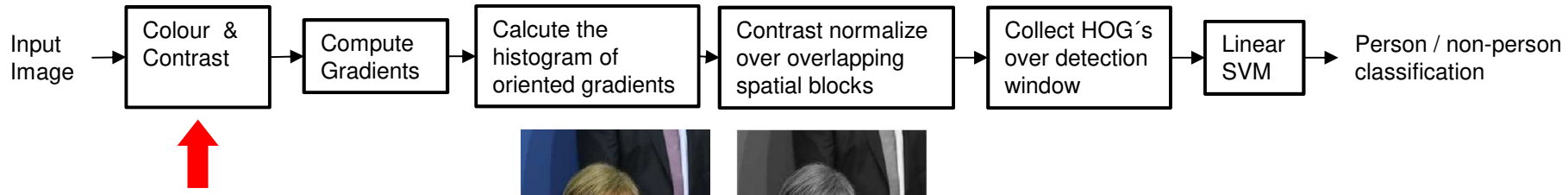
Feature Analysis – Classification with Manual Features

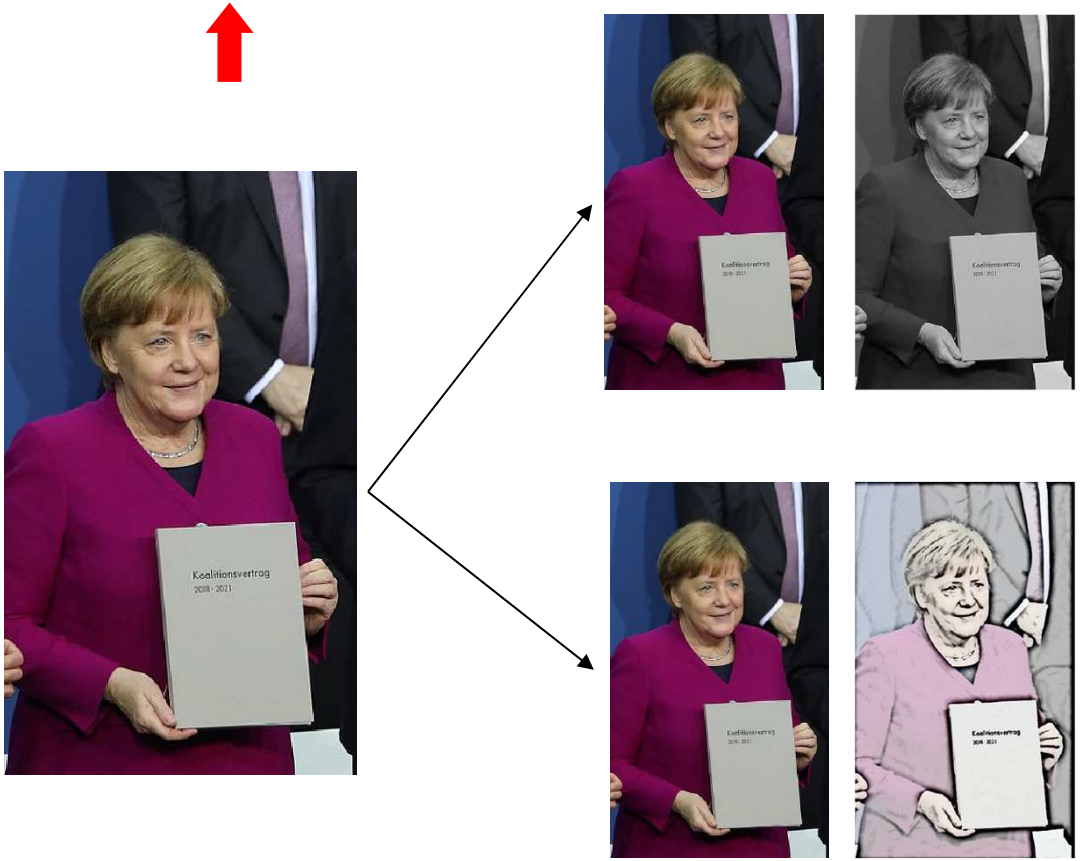


1. Input image
2. Get information about image size

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

Feature Analysis – Classification with Manual Features



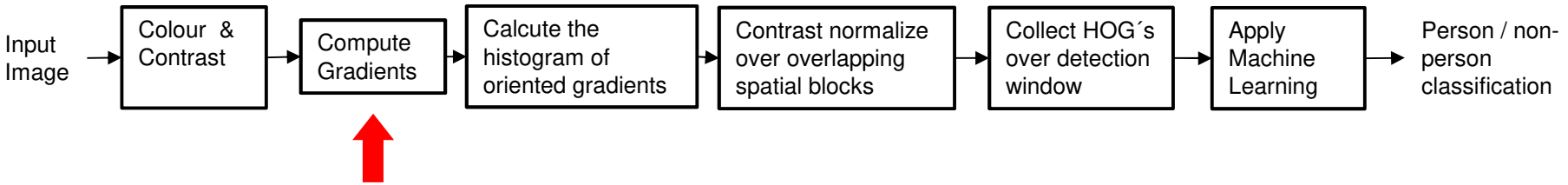


- Change color space
  → RGB offer more features because of different color seperation

- Contrast Enhancement
  → Offers more features because of better edge detection

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

Feature Analysis – Classification with Manual Features



Compute Gradients by selecting the best kernel:

0	1
-1	0

diagonal

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

Sobel

-1	0	1

centered

-1	1

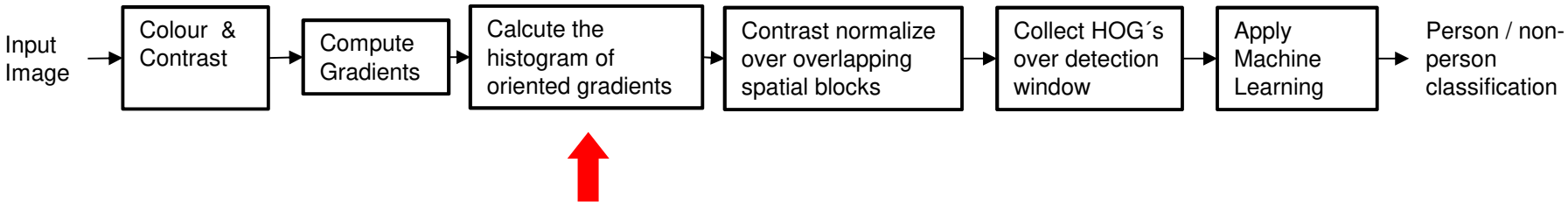
uncentered

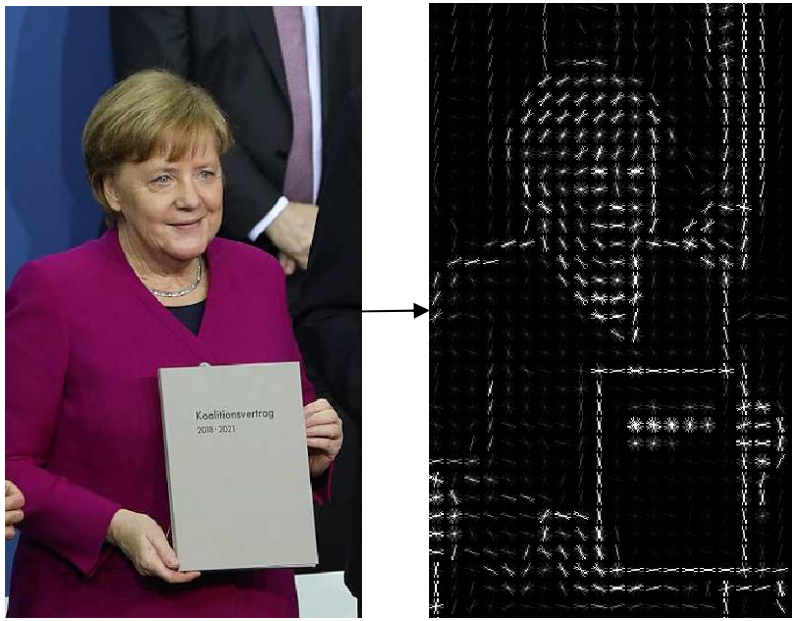
1	-8	0	8	-1

cubic-corrected

Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

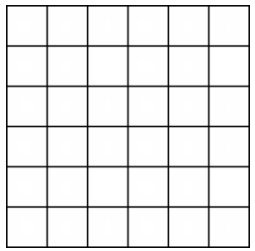
Feature Analysis – Classification with Manual Features



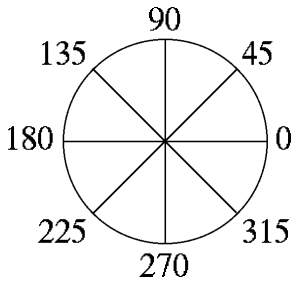


- Calculate the Histogram of gradient orientations
  - Votes weighted by magnitude
  - Bilinear interpolation between cells

Histograms in k*k pixel cells



Orientation: 9 bins (for unsigned angles 0 -180)



Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

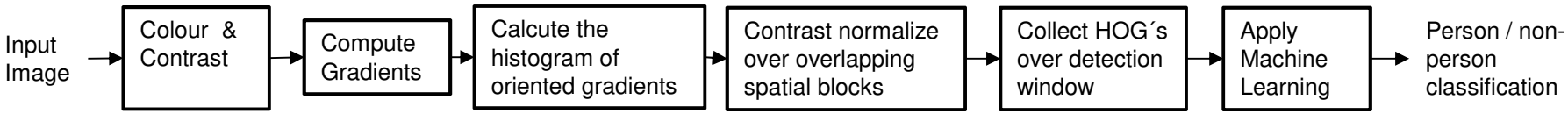
Additional Slide

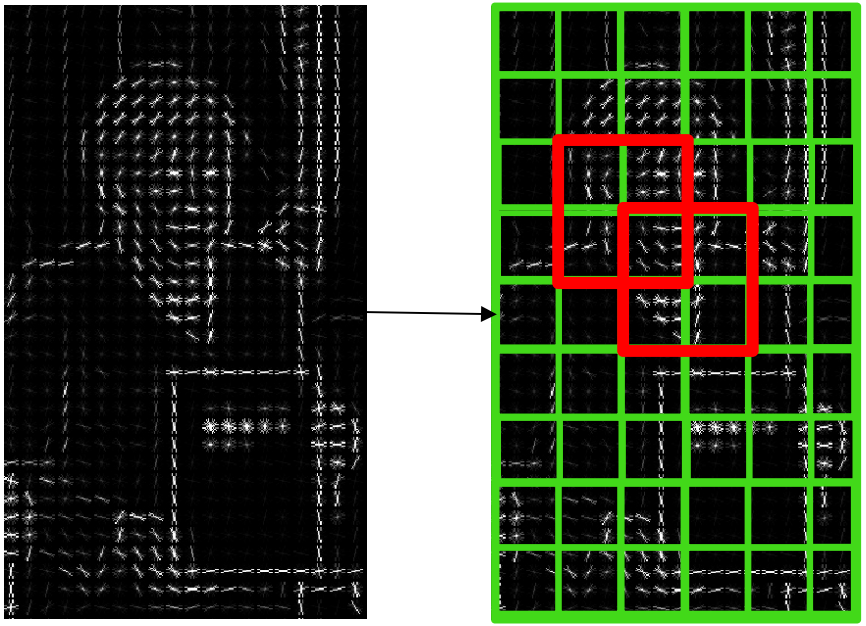
Histogram of oriented gradients

The **histogram of oriented gradients (HOG)** is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

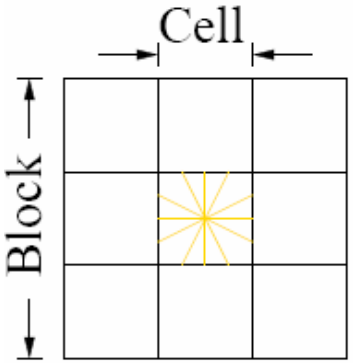
Robert K. McConnell of Wayland Research Inc. first described the concepts behind HOG without using the term HOG in a patent application in 1986.[1] In 1994 the concepts were used by Mitsubishi Electric Research Laboratories.[2] However, usage only became widespread in 2005 when Navneet Dalal and Bill Triggs, researchers for the French National Institute for Research in Computer Science and Automation (INRIA), presented their supplementary work on HOG descriptors at the Conference on Computer Vision and Pattern Recognition (CVPR). In this work they focused on pedestrian detection in static images, although since then they expanded their tests to include human detection in videos, as well as to a variety of common animals and vehicles in static imagery.

Feature Analysis – Classification with Manual Features



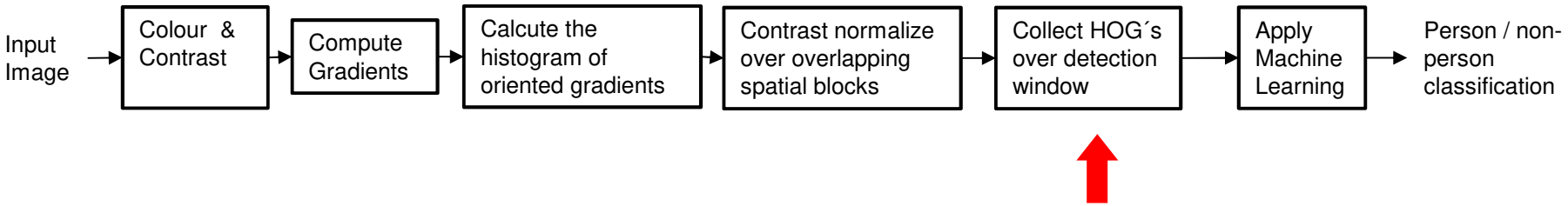


- Normalization of the blocks by super-imposition
- Normalization of cells is independent of blocks
- HoG for overlapping 2x2 blocks of cells are normalized and collected to form the final descriptor

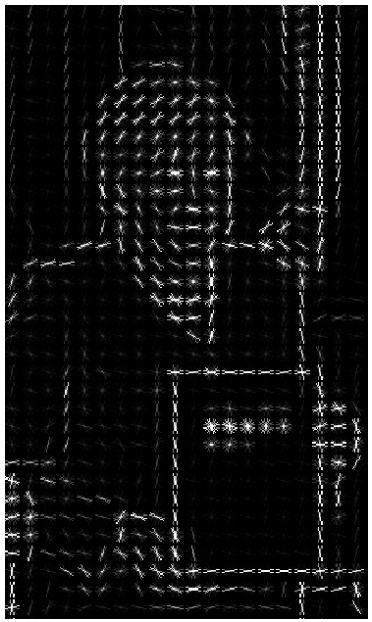


Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

Feature Analysis – Classification with Manual Features



X=



- HOGs of all blocks are written to feature output vector
- Vector is used in window classifier
- Feature vector f = [..., ..., ...]

orientations

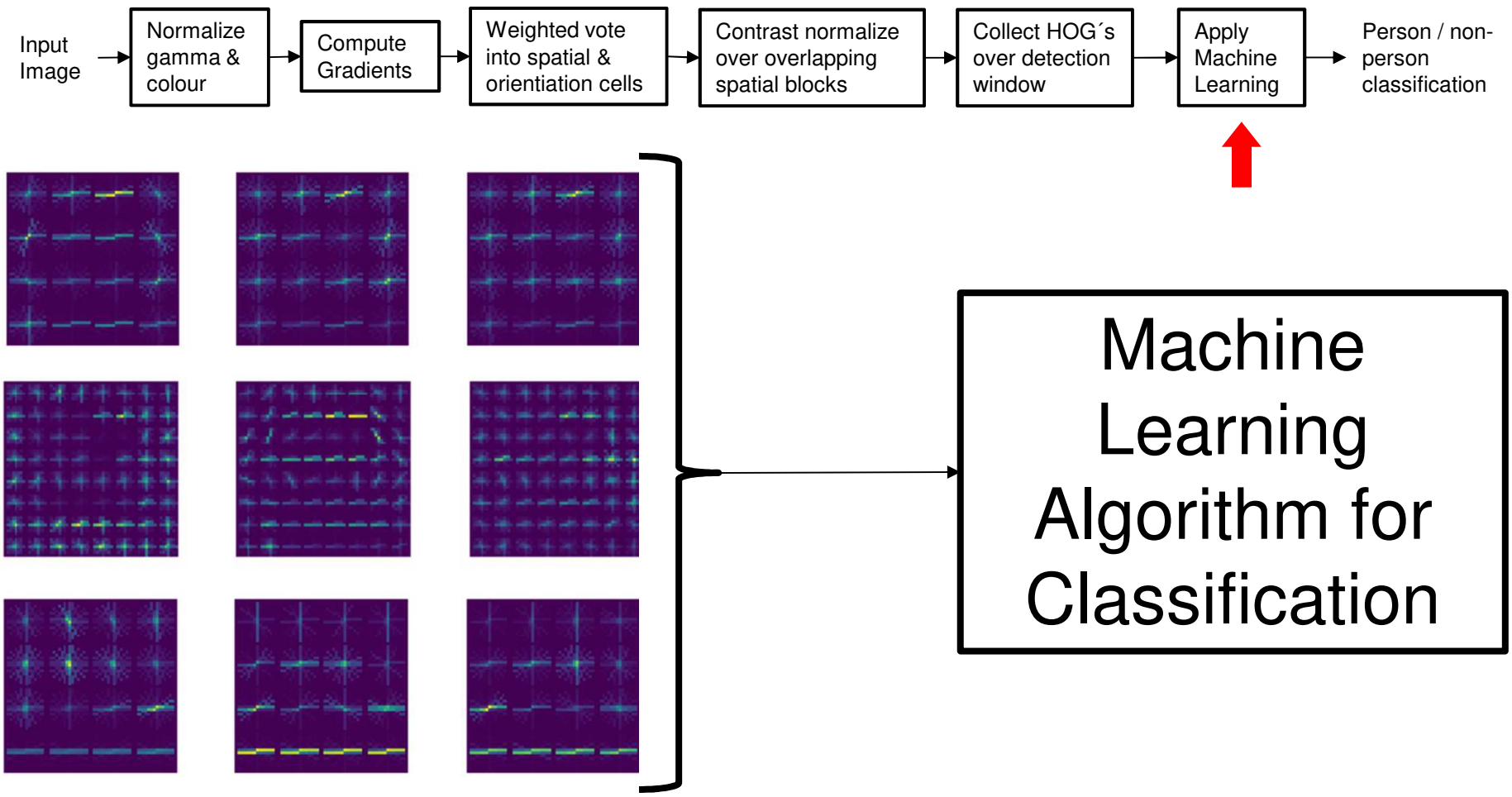
features = 15 x 7 x 9 x 4 = 3780

cells

normalizations by neighboring cells

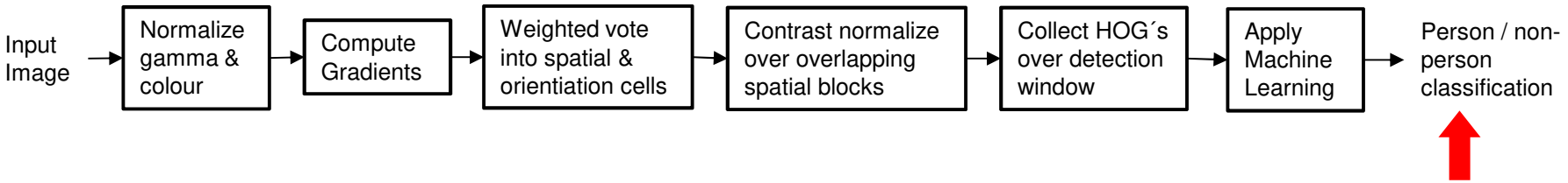
Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

Feature Analysis – Classification with Manual Features



Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

Feature Analysis – Classification with Manual Features



Source: Georgia Tech – Lecture CS 6476 Computer Vision – Lecture 18

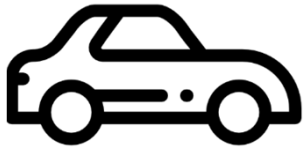
Perception

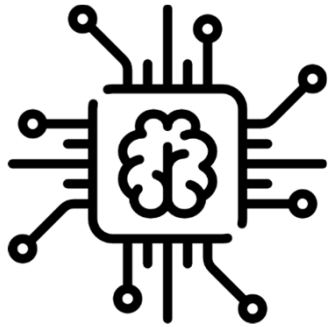
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

Agenda

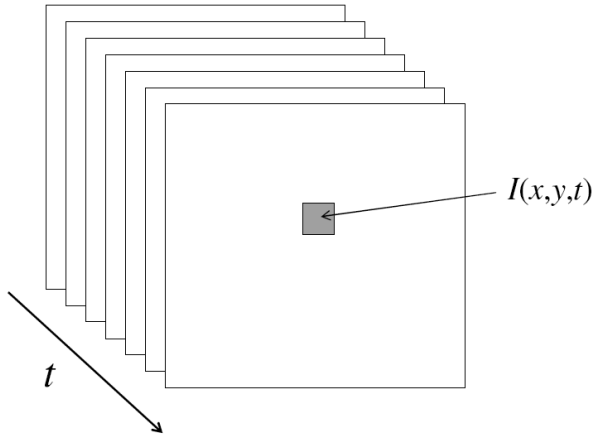
1. Chapter: History, Pipeline, Definitions
2. Chapter: Machine Vision
3. Chapter: Image Processing
4. Chapter: Features
5. Chapter: Feature Analysis
6. **Chapter: Information Analysis**
7. Chapter: Summary



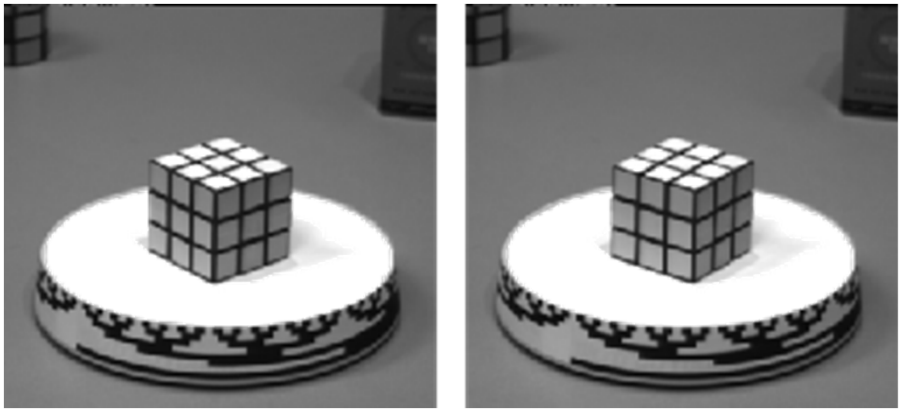


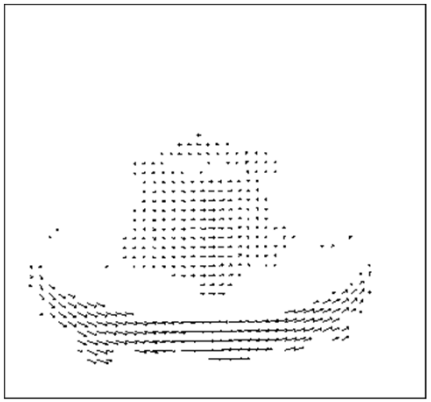
Information Analysis – Optical Flow

- A **video** is a sequence of frames captured over time
- Now our image data is a function of space (x, y) and **time (t)**



- **Optical flow** or optic flow is the **pattern of apparent motion** of objects, surfaces, and edges in a visual scene caused by the **relative motion between an observer and a scene**



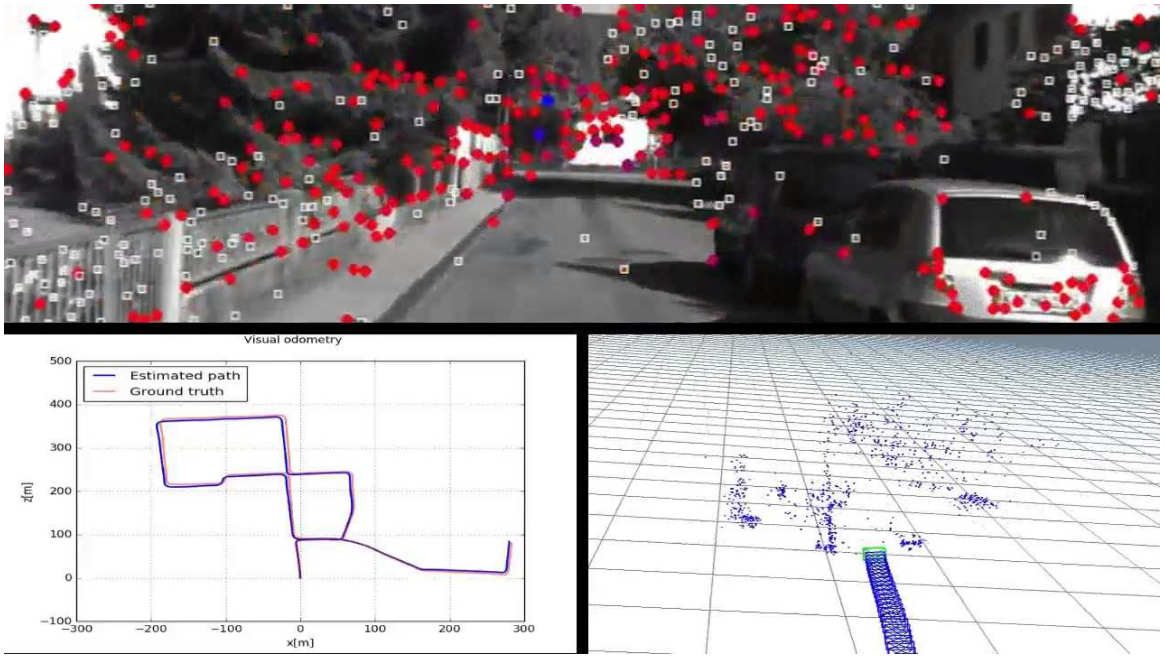


Information Analysis – Optical Flow

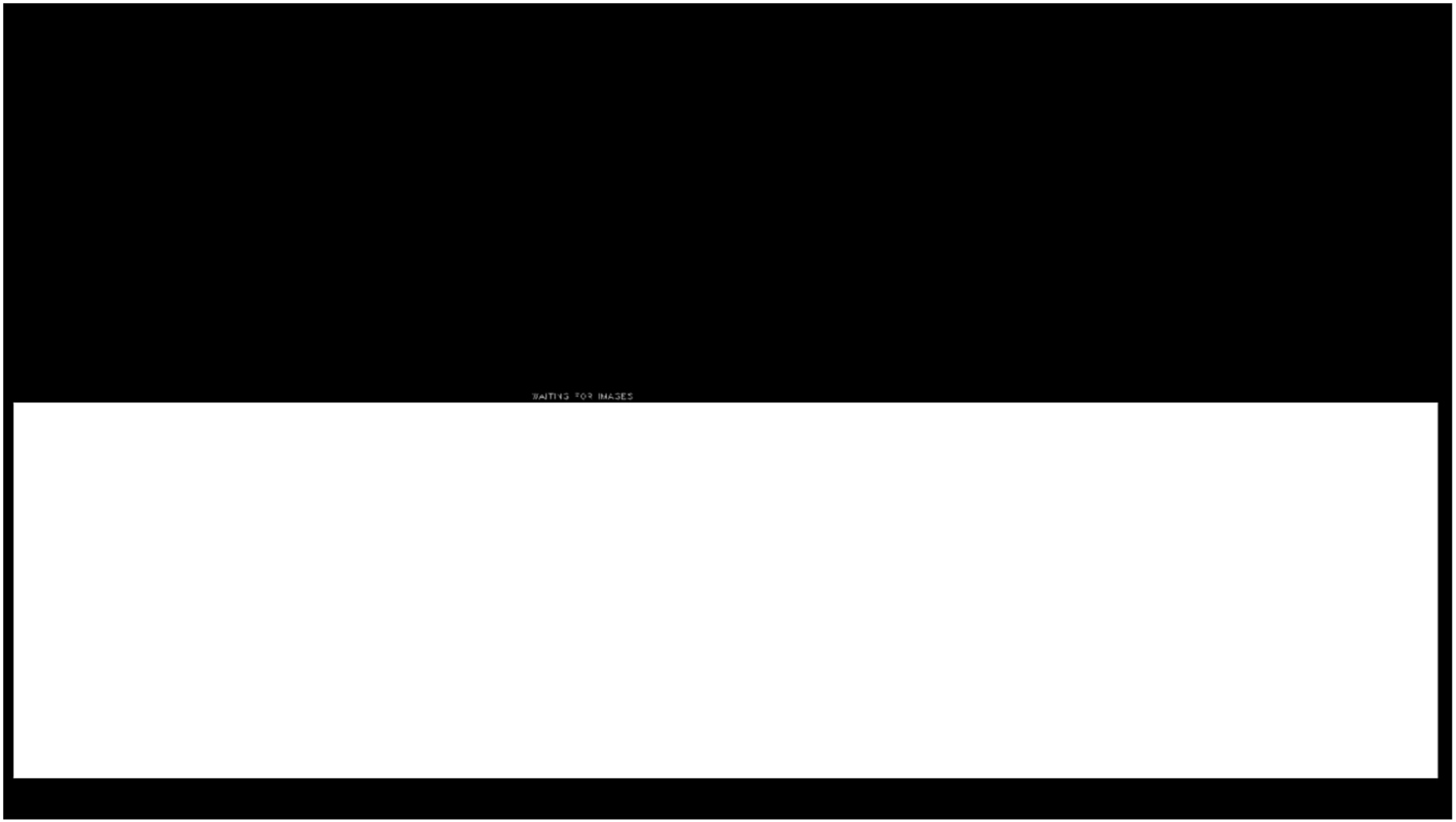
Source: https://www.youtube.com/watch?v=LNLPNK0pc2U

Information Analysis – Egomotion Estimation and SLAM

- In robotics and computer vision, **visual odometry** is the process of determining the position and orientation of a robot by analyzing the associated camera images.
- **SLAM** = Simultaneous Localization and Mapping



Information Analysis – Egomotion Estimation and SLAM



Source: https://www.youtube.com/watch?v=8DISRmsO2YQ

Additional Slide

Pipeline for Egomotion Estimation

1. Acquire input images: using camera (Mono, Stereo,…)
2. Image correction: apply image processing techniques
3. Feature detection: define interest operators, and match features across frames and construct optical flow field.
    1. Use correlation to establish correspondence of two images, and no long term feature tracking.
    2. Feature extraction and correlation.
    3. Construct optical flow field (Lucas–Kanade method).
4. Check flow field vectors for potential tracking errors and remove outliers.
5. Estimation of the camera motion from the optical flow.
    1. Choice 1: Kalman filter for state estimate distribution maintenance.
    2. Choice 2: find the geometric and 3D properties of the features that minimize a cost function based on the re-projection error between two adjacent images. This can be done by mathematical minimization or random sampling.
6. Periodic repopulation of trackpoints to maintain coverage across the image.

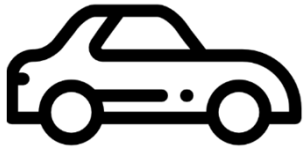
Perception

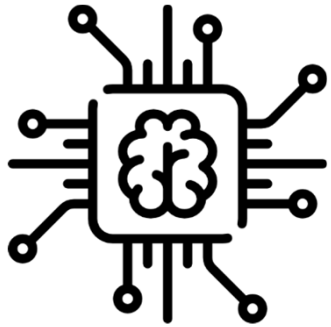
Johannes Betz / Prof. Dr. Markus Lienkamp /
Prof. Dr. Boris Lohmann

(Johannes Betz, M. Sc.)

Agenda

1. Chapter: History, Pipeline, Definitions
2. Chapter: Machine Vision
3. Chapter: Image Processing
4. Chapter: Features
5. Chapter: Feature Analysis
6. Chapter: Information Analysis
7. **Chapter: Summary**





Summary

What did we learn today:

- **Perception** is an Artificial Intelligence problem.
- We can divide perception into **Machine Vision** and **Computer Vision**
- We focus on Machine Vision and **Computer Vision** → The ability to retrieve information out of a **camera.**
- Reasons for using a camera in a vehicle: Cameras are cheap, easy to access and provide a lot of information about the current environment → Similarity to human eye.
- There are important differences between Machine Vison setups: We need to know what **camera, lenses, computer setup and environment we are operating in** when using machine vision.

Summary

What did we learn today:

- We defined a **CV pipeline**: The pipeline consists of different steps we need to do for setting up a CV application
- **Matlab/Simulik** and the **OpenCV** Library are the first software tools to choose for processing digital images
- First of all we have **to process** the image for getting better information (features) out of one picture: **Denoise (Filtering), Contrast Enhancement, Color Space change…**
- Then we have to **extract the features** like **Edges, Corners**,…
- We can then use **the features** for getting information out of the picture: Detect and recognize objects, classify objects,…
- We can use **machine learning algorithms** for doing the classification based on the features we defined and the images we **processed earlier**